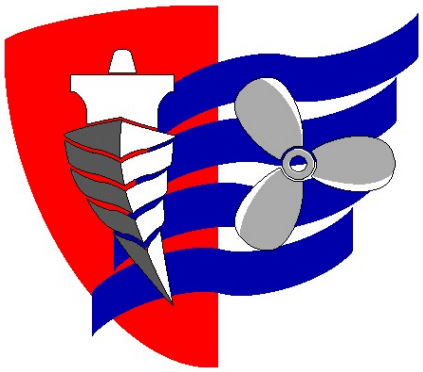
ESCUELA TÉCNICA SUPERIOR DE NÁUTICA

UNIVERSIDAD DE CANTABRIA



Trabajo Fin de Grado

DISEÑO Y DESARROLLO SISTEMA DE ESCANEO LASER 3D DE BAJO COSTE

DESIGN AND DEVELOP OF A LOW COST 3D SCANNING SYSTEM

Para acceder al Título de Grado en

INGENIERÍA MARÍTIMA

Autor: Abel Báscones

Director: Miguel Mateo

Julio - 2017

ESCUELA TÉCNICA SUPERIOR DE NÁUTICA

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

DISEÑO Y DESARROLLO DE UN SISTEMA DE ESCANEO LASER 3D DE BAJO COSTE

DESIGN AND DEVELOP OF A LOW COST 3D SCANNING SYSTEM

Para acceder al Título de Grado en

INGENIERÍA MARÍTIMA

Julio - 2017

1 - INDICE GENERAL:

2. RESUMEN:

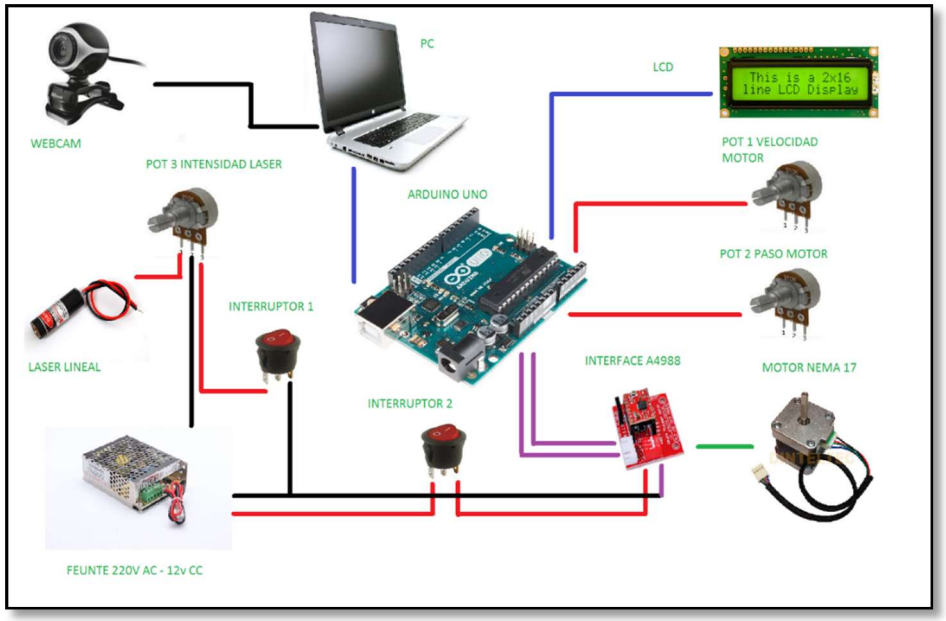


***Figura 01**- Sistema de escaneo laser 3D de bajo coste.*
***Fuente:** Elaboración propia.*

2.1. Resumen (Versión Española)

Este trabajo trata de mostrar cómo construir un sistema de adquisición de datos económico, sencillo y rápido de elaborar, integrando elementos fácilmente asequibles y desarrollando un código simplificado. Se utilizarán sistemas tipo Open software, gratuitos y sencillos de manipular.

El sistema de captura de datos LASER permitirá la obtención de perfiles y contornos de objetos fruto del reflejo de un haz lineal laser mediante un sistema rotatorio que realizará barridos completos de 360 grados.

La configuración del equipo está compuesto por un conjunto formado por un motor paso-paso controlable por programa, conectado a un brazo que da soporte a un LASER lineal de bajo costo y una Webcam que será la encargada de realizar la captura de las imágenes.

El sistema de control del motor, monitorización y ajuste de parámetros está gestionado por un microcontrolador del tipo ARDUINO UNO, donde se ha desarrollado un código específico para controlar el paso, velocidad, barridos y ángulo máximo de abertura del sector de barrido.

El proceso de captura de imágenes y posterior procesado de las mismas se ha realizado mediante dos códigos, ambos desarrollados en PROCESSING. Un primer programa controla la gestión de las imágenes: captura, procesado y esqueletización del haz laser. El segundo genera un archivo de texto externo con la nube de puntos en el espacio en coordenadas XYX + ángulo, tras someter las imágenes procesadas en la primera etapa a el proceso de triangulación.

2.2. Summary (English Version)

This paper tries to show how to desing and build an economic data acquisition system, simple and fast to elaborate, integrating elements easily accessible and developing a simplified code. Open software systems will be used, free and simple to manipulate.

The LASER data capture system, will allow the obtention of profiles and contours of objects resulting from the reflection of a linear laser beam by means of a rotating system that will perform full sweeps of 360 degrees.

The configuration of the equipment consists of a set of a stepper motor programmable, connected to an arm that supports a low-cost linear LASER and a Webcam that will be in charge of the images capture.

The motor control system, monitoring and adjusted by a different parameters is managed by a microcontroller of type ARDUINO UNO, where a specific code has been developed to control the pitch, speed, sweeps and maximum opening angle of the sweep sector.

The process of image capture and subsequent processing of the same, has been made by two codes, both developed in PROCESSING. A first program controls the management of images: capture, processing and skeletonization of the laser beam. The second, generates an external text file with the point cloud in the space in XYX + angle coordinates, after subjecting the processed images in the first stage to the triangulation process

2.3. Palabras clave del proyecto:

LASER, TRIANGULACION, DIGITALIZACION, ESCANER, ADQUISICION DE DATOS, BARRIDO, DIGITALIZACION, 3D, LASER 3D, ESCANER LASER 3D.

3. INTRODUCCION Y MOTIVACION

Siempre he sentido una fascinación especial a cerca de los diferentes métodos de digitalización de superficies. Guardar de manera atemporal datos no solo de calidad fotográfica en dos dimensiones, si no como objetos obtenidos de la realidad espacial en 3 dimensiones, supone disponer de datos para poder reproducir el mismo objeto con diferencias morfológicas mínimas.

He seguido la evolución de los diferentes sistemas de captura de datos o nube de puntos casi desde sus orígenes. Considero que esta tecnología está ya en fase de maduración, con resultados muy interesantes, tanto en los procesos de captura de datos como en la gestión de los mismos a posteriori, gracias a los potentes equipos informáticos de los que hoy en día se disponen, sin embargo esta tecnología resulta todavía cara e inaccesible cuando es necesario el procesado de grandes áreas que conlleven nubes de puntos y datos en grandes cantidades.

Fue durante la última etapa de mi formación académica, concretamente durante mi periodo de prácticas, cuando comenzó a gestarse la idea de desarrollar el instrumento que en breves expondré en este trabajo.

He tenido la fortuna de poder desempeñar esta formación práctica en un astillero de reparaciones y transformación de buques, donde he estado inmerso en varios proyectos pioneros de conversión de buques, focalizados en la modernización de los sistemas de escape de varios de los mismos.

Estos proyectos, en la fase de estudio, se idearon con la única ayuda de sistemas CAD para elaborar planos e isométricas para los sistemas de canalización y tubería a modificar.

La incidencia en obra se mostró insuficiente a lo largo de la ejecución de la misma, aunque el diseño fue realizado empleando todos los recursos tanto técnicos como humanos en desarrollarlos, se mostraron imprecisos, debido a que en muchas ocasiones el entorno real era diferente al simulado, directamente de planos “as built” creados años atrás en el astillero de construcción original.

Como consecuencia problemas de previsión de costes, retrasos inesperados por redefinición de trazado de tubería y estructuras aparecieron debido a las colisiones entre los sistemas existentes y los nuevos a añadir en obra. Esto supuso pérdidas de

tiempo en búsqueda de soluciones alternativas, y como no, incrementos en el presupuesto inicial.

Debido a esta experiencia se consideró el diseño de un sistema de adquisición de datos tridimensional, para escanear entornos y digitalizarlos posteriormente, para que en una etapa posterior de comparación entre el entorno real obtenido en 3D y el nuevo modelado, se pudiera obtener un filtro predictivo de coincidencia de sistemas, colisiones y demás que provocara tomar decisiones previas tanto técnicas como económicas sobre la viabilidad del diseño antes de comenzar la ejecución real de la obra.

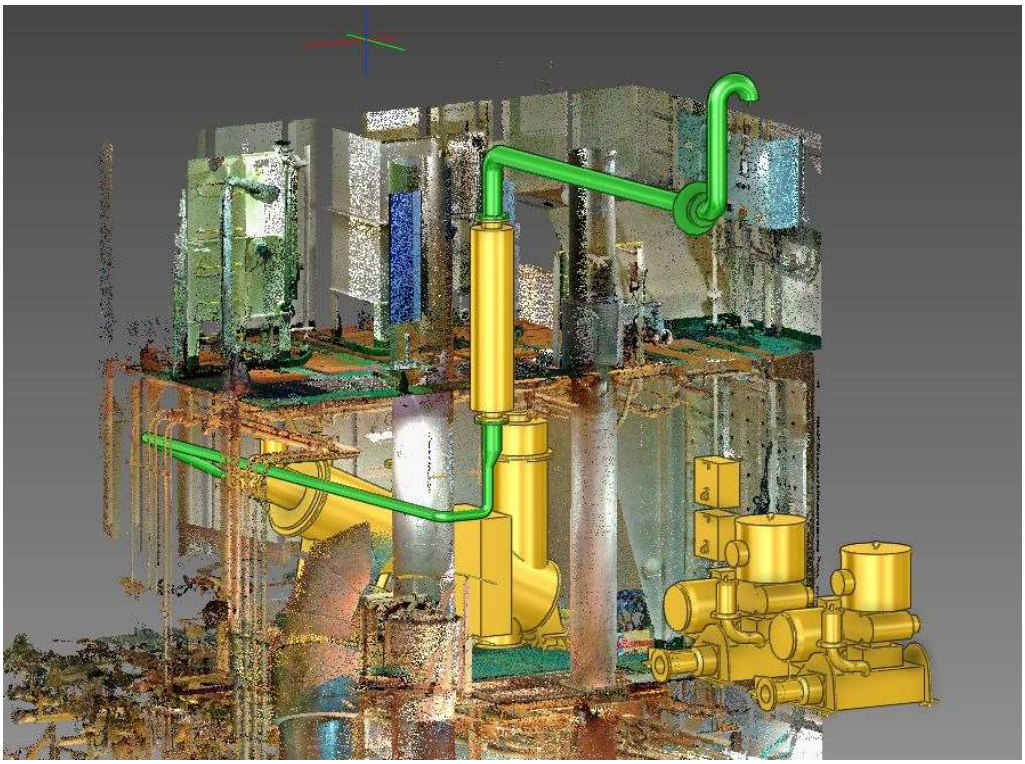


***Figura 02**- Colision tuberías drenaje con sistema de ventilación.*

***Fuente**: STX_France ®*

4. OBJETIVOS GENERALES:

Básicamente este trabajo versará en el diseño y desarrollo de un software que gestione la adquisición de imágenes mediante los brillos del láser reflejados sobre los objetos a escanear mediante procesado posterior de dichas imágenes. A su vez se diseñará y se desarrollará un software de control que gestione el giro de un motor paso paso que soportara un brazo con los componentes de emisión de luz y captura de datos.

Se buscará también, integrar todos los elementos físicos que hagan posible cumplir lo descrito anteriormente en la etapa del software.

Para ello se diseñara un equipo físico con los requisitos estructurales eléctricos y electrónicos que hagan cumplir los parámetros indicados anteriormente.

El objetivo final será la adquisición de una serie de nube de puntos en el espacio que mantenga las características físicas de los objetos reales[1].

[1] Este trabajo no abordará (sería objeto de uno nuevo que lo complemente o una 2º parte que lo complete) el estudio y procesado de imágenes, que pueda convertir esa nube de puntos en superficies poligonales y texturizadas. También el uso de la fotogrametría para facilitar una mejor asimilación de las características físicas externas de los objetos escaneados entraría dentro del alcance de esta segunda parte.

5. POSIBLES USOS

Un escáner 3d es una máquina que sirve para capturar la geometría de un objeto y su color para después formar un modelo 3d del mismo. Ha de ser capaz de transformar un objeto real en uno virtual que podemos manipular con nuestros ordenadores y programas especializados, a modo de rotar en el espacio, voltear o cambiar de posición o modificar por software su apariencia física.

Su uso es muy variado y actualmente se utiliza en muchos sectores:

- ***Patrimonio:*** Conservación de obras de arte, reproducción de esculturas…
- ***Ingeniería:*** Control dimensional, control de calidad, ingeniería inversa de piezas…
- ***Medicina:*** Diseño de prótesis y estudios odontológicos, escaneo y posterior reproducción de férulas a medida con baja densidad y adaptables a la morfología del cuerpo…
- ***Arquitectura:*** Medición y planificación de edificios, levantamiento de planos…
- ***Arqueología:*** Estudio de yacimientos arqueológicos, reconstrucción de piezas…
- ***Topografía:*** Estudio de terrenos, minería, trazados viales, digitalización de cavidades…

6. CONSTRUCCION DEL HARDWARE:

6.1. OBJETIVOS:

La implementación de todos los dispositivos que se explican a continuación buscan el poder obtener un movimiento secuencial y controlado para coordinar la toma de imágenes sucesivas conforme al avance de la rotación, para así poder procesarlas y almacenarlas codificadas.

El movimiento de rotación esta generado por un motor paso a paso controlado por programa y generado en ARDUINO. El eje del motor está unido a un brazo de dimensiones conocidas. En los extremos del mismo se han colocado una cámara web rotada permanentemente 40º y un haz laser lineal perpendicular.

6.2. ELEMENTOS QUE CONFORMAN EL HARDWARE:

Todos los elementos hardware necesarios para construir la máquina, están mostrados en el diagrama de bloques general que se explica a continuación:

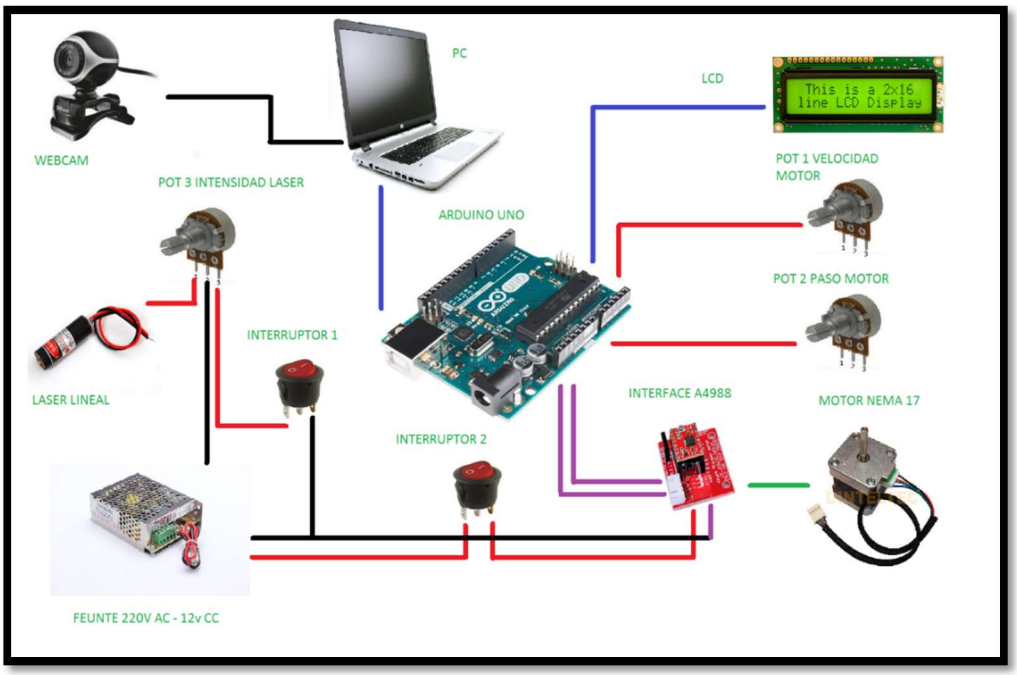


Figura 03 *- Sistema de escaneo laser 3D de bajo coste.*

Fuente: *Elaboración propia.*

Los componentes utilizados se detallan a continuación:

6.2.1. ARDUINO UNO[2]:

Arduino es una plataforma de prototipos electrónica de código abierto (open-source) basada en hardware y software flexibles y fáciles de usar. Está pensado para artistas, diseñadores, como hobby y para cualquiera interesado en crear objetos o entornos interactivos.

Arduino puede sentir el entorno mediante la recepción de entradas desde una variedad de sensores y puede afectar a su alrededor mediante el control de luces, motores y otros artefactos. El microcontrolador de la placa se programa usando el *Arduino Programming Language* (basado en Wiring) y el *Arduino Development Environment* (basado en Processing). Los proyectos de Arduino pueden ser autonomos o se pueden comunicar con software en ejecución en un ordenador (por ejemplo con *Flash, Processing, MaxMSP*, etc.).

Las placas se pueden ensamblar a mano o encargarlas pre ensambladas; el software se puede descargar gratuitamente. Los diseños de referencia del hardware (archivos CAD) están disponibles bajo licencia open-source, por lo que eres libre de adaptarlas a tus necesidades.

Arduino simplifica el proceso de trabajo con microcontroladores, pero ofrece algunas ventajas:

1. **Barato:** Las placas Arduino son relativamente baratas comparadas con otras plataformas microcontroladoras y puede ser ensamblada a mano.
2. **Multiplataforma:** El software de Arduino se ejecuta en sistemas operativos Windows, Macintosh OSX y GNU/Linux.
3. **Entorno de programación simple y claro:** El entorno de programación de Arduino es fácil de usar para principiantes, pero esa además flexible para que usuarios avanzados puedan aprovecharlo también.

[2] http://arduino.cl/que-es-arduino/

4. **Código abierto y software extensible:** El software Arduino está publicado como herramientas de código abierto, disponible para extensión por programadores experimentados.
5. **Código abierto y hardware extensible:** El Arduino está basado en microcontroladores ATMEGA8 y ATMEGA168 de Atmel. Los planos para los módulos están publicados bajo licencia Creative Commons, por lo que diseñadores experimentados de circuitos pueden hacer su propia versión del módulo, extendiéndolo y mejorándolo. Incluso usuarios relativamente inexpertos pueden construir la versión de la placa del módulo para entender como funciona y ahorrar dinero.

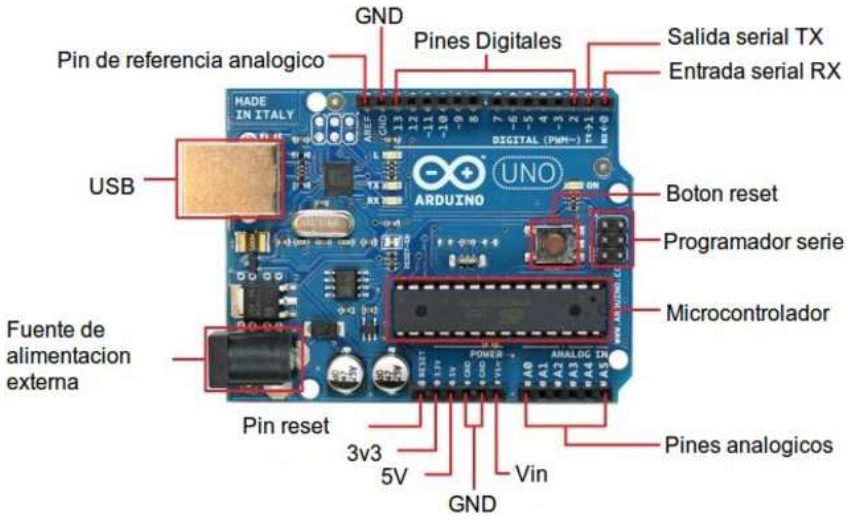


Figura 04 – Placa ARDUINO UNO ®

Fuente: www.arduino.com

6.2.2 MOTOR PASO PASO TIPO NEMA 17:

El motor paso a paso NEMA 17 seleccionado para este proyecto es un motor bipolar, con un ángulo de paso de 1.8º (200 pasos por vuelta) y cada bobinado es de 1.2 A a 4 V, capaz de cargar con 3.2 kg/cm.

Es un motor muy robusto ampliamente utilizando en impresoras 3D caseras.

Características:

- Tamaño: 42.3×48mm, sin incluir el eje (NEMA 17)
- Peso: 350 gramos (13 oz.)
- Diámetro del eje: 5 mm "D"
- Longitud del eje: 25 mm
- Pasos por vuelta: 200 (1,8º/paso)
- Corriente: 1.2 Amperios por bobinado
- Tensión: 4 V
- Resistencia: 3.3 Ohm por bobina
- Torque: 3.2 kg/cm (44 oz-in)
- Inductancia: 2.8 mH por bobina

Figura 05 *– Motor paso paso + sistema de bobinado*

Fuente: *http://tienda.bricogeek.com/motores-paso-a-paso*

6.2.3 TARJETA DE CONTROL A4988

La tarjeta seleccionada incorpora el chip de Allegro A4988. Es usado como controlador de motores paso a paso de hasta 2 A de corriente por bobina.

Características

- Cinco resoluciones diferentes: paso completo, medio paso, un cuarto de paso, un octavo de paso, y un dieciseisavo de paso.
- Control de corriente ajustable que permite ajustar la salida de corriente máxima con un potenciómetro, que le permite utilizar tensiones superiores a la tensión nominal del motor paso a paso para lograr mayores tasas de paso.
- Protección por sobrecalentamiento térmico, cierre por baja tensión, y protección por sobre pico de corriente.

Figura 06 - *Controladora A4988*

Fuente: *http://tienda.bricogeek.com/impresion-3d/553-pololu-a4988-stepstick-prusa-reprap.html*

6.2.4 WEBCAM:

Se ha utilizado una económica cámara web con resolución de hardware de 640 x 480 ppp, estando indicada para usar Vídeo USB 2.0 con una imagen de vídeo nítida y sin interrupciones

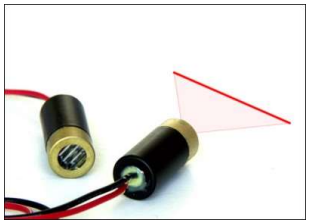
No requiere la instalación de un controlador.

Figura 07 *– Cámara web a emplear.*

Fuente: http://www.trust.com/products/product.aspx?artnr=17003

6.2.5 LÁSER LINEAL:

El modulo láser seleccionado proyecta una línea recta (deseada para la adquisición de datos), con un ángulo de 60°, disponible en 650 nm, y con una potencia de salida de 1, 3 o 5 mW. Cumple con las normas europeas CE.



Especificaciones:

Tipo de láser :	II o IIIa
Largo de onda :	650nm
Potencia :	1, 3, 5mW
Lente :	Plástica
Voltaje :	3V DC
Amperaje :	35mA
Temperatura de utilización :	-10°C a +40°C
Proyección :	línea (ángulo : 60°)
Divergencia :	<2.0 mrd
Capa exterior :	Metálica
Largo :	22 mm (+/-0.5)
Diámetro :	10 mm (+/- 0.1)
Largo de los cables :	100mm

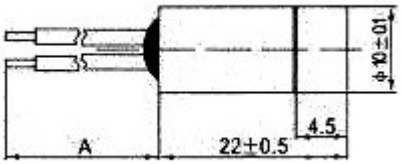


Figura 08 - *Láser lineal utilizado*

Fuente: *http://www.electan.com/emisor-laser-rojo-linea*

6.2.6 PANTALLA LCD

Utilizaremos un display LCD compatible con el **módulo HD44780 de Hitachi**. Soporta 132 caracteres alfanuméricos y 32 de control. Las líneas de control que posee se encuentran en los pines 4, 5 y 6. Cuando la línea Enable Signal pasa de 1 a 0, el controlador del LCD leerá el resto de líneas, ya sean de datos o de control. Cuando R/W está a 0, se podrá escribir sobre el LCD y, cuando está a 1, se podrá leer del LCD. Si RS está a nivel bajo, es decir, a 0 voltios, el dato es tratado como un comando o una orden sobre el LCD. Sin embargo, si está a nivel alto, el dato enviado es el texto a mostrar en el display LCD.

Características:

No.	Symbol	Function
1	VSS	Ground (0V)
2	VDD	Supply Voltage for Logic (+5V or +3.3V)
3	VO	Contrast Adjustment
4	RS	Data/Instruction Select
5	R/W	Read/Write Select
6	E	Enable Signal
7	DB0	Data Bus
8	DB1	Data Bus
9	DB2	Data Bus
10	DB3	Data Bus
11	DB4	Data Bus
12	DB5	Data Bus
13	DB6	Data Bus
14	DB7	Data Bus
15	LED_A	LED Power Supply + (5V)
16	LED_K	LED Power Supply - (5V)

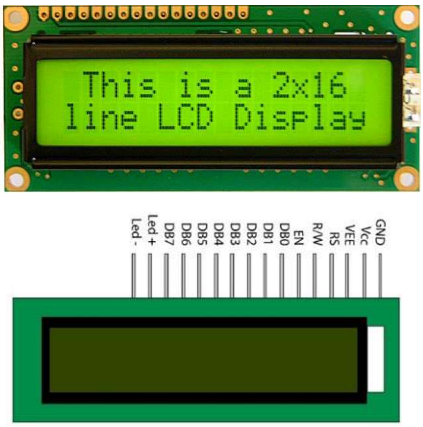


Figura 09 **-** módulo HD44780 de Hitachi**.**

Fuente: *http://panamahitek.com/uso-de-pantalla-lcd-con-arduino/*.

6.2.7 Otros componentes electrónicos:

Se utilizarán otros componentes electrónicos para configurar el funcionamiento y control del sistema diseñado.

Estos son un conjunto de potenciómetros regulables (intensidad de brillo de la pantalla LCD, regulación intensidad de emisión LASER, regulador de velocidad motor paso paso y ángulo máximo de barrido), fusibles protectores (motor y Laser a 12V) e interruptores de inicialización. Además contaremos con una fuente de alimentación de 12 V de CC, que alimentara al sistema láser y la electrónica de control, incluyendo el motor paso a paso.

6.3. EXQUEMA ELECTRICO:

El conexionado de los elementos es sencillo. Se conectará a la placa Arduino a un controlador para el motor paso a paso, una pantalla LCD y un par de potenciómetros.

Por otra parte el láser se integrara en el conjunto de manera independiente, teniendo solo un circuito de protección y activación inicial[3].

Lo mismo ocurre con la Webcam, que irá conectada directamente el puerto USB del ordenador y se integrará en el conjunto más adelante, una vez que se construya la estructura de la máquina.

6.3.1 Conexión de la Pantalla LCD.

Se usará una pantalla LCD 16x2 caracteres. Conectamos a +5V/GND los pines 1 y 2 que alimentan el componente. Esta tensión se puenteará directamente de las salidas estables de la tarjeta ARDUINO (5V y GND). El PIN 3 irá conectado a la salida regulada del Potenciómetro, que regula el contraste de cada celda de caracteres.

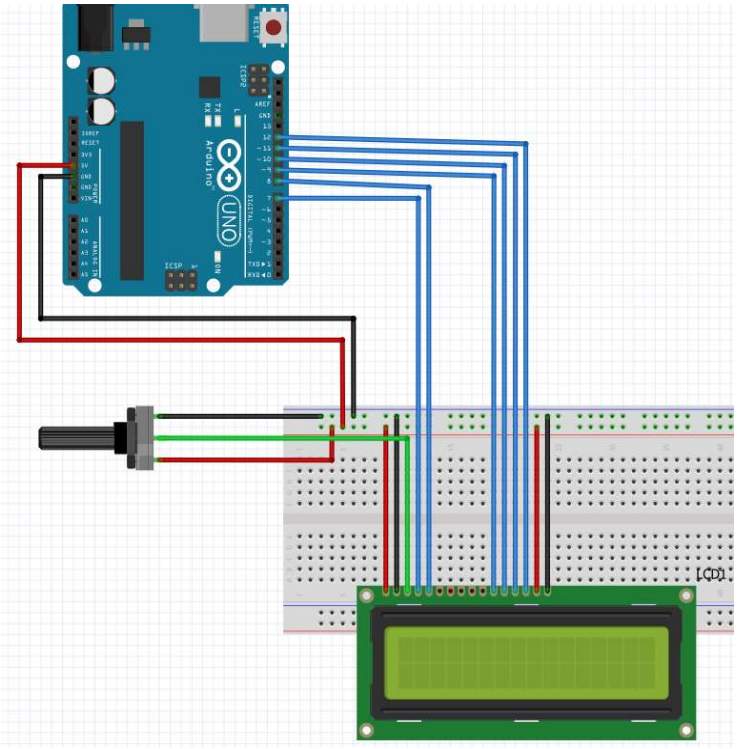
Los pines 5 y 6 (Selección de datos y configuración de lectura / escritura en el LCD) irán conectados a los pines 6 y 7 en modo salida digital de la tarjeta ARDUINO. Se configuraran en modo alto o bajo por programa del mismo.

Se ha seleccionado como BUS de datos de 1 byte (0000) las líneas 11, 12, 13 y 14

[3] Considerando este proyecto como versión BETA, se procederá a mejorar la interacción del mismo en el futuro de un modo más completo.

del LCD, que irán conectadas a las salidas digitales del ARDUINO 9, 10, 11 y 12.

Finalmente, la retroiluminación de fondo, se configurará en el LCD como +5V el pin 15 y a GND el pin 16.



No.	Symbol	Function
1	VSS	Ground (0V)
2	VDD	Supply Voltage for Logic (+5V or +3.3V)
3	VO	Contrast Adjustment
4	RS	Data/Instruction Select
5	R/W	Read/Write Select
6	E	Enable Signal
7	DB0	Data Bus
8	DB1	Data Bus
9	DB2	Data Bus
10	DB3	Data Bus
11	DB4	Data Bus
12	DB5	Data Bus
13	DB6	Data Bus
14	DB7	Data Bus
15	LED_A	LED Power Supply + (5V)
16	LED_K	LED Power Supply - (5V)

Figura 09 *- Conexionado LCD + Arduino +Potenciometro.*

Fuente: *elaboración propia.*

6.3.2 Conexión del Controlador del Motor NEMA 17:

En el controlador de motor paso a paso A4988, los pines ***VDD*** y ***GND*** está conectado a 5V y tierra del Arduino respectivamente. Los pines ***VMOT*** y ***GND*** están conectados a una fuente de alimentación externa de 12 V. Esta conexión está protegida con un condensador polar ***100uF*** en paralelo para amortiguar los picos de voltaje desde el motor **NEMA17**.

El ***RESET*** Y ***SLEEP*** se puentea.

Los comandos que definen el PASO y el sentido de giro del eje motor ***DIR,*** se conectaran a las salidas del ARDUINO Digitales ***10*** y ***9***.

El **NEMA 17** es un motor bipolar con dos bobinas que irán conectadas a las tomas del sensor ***1A, 2A*** y ***1B, 2B*** respectivamente[4].

Conectar un par motor de alambres para **1A** y **1B** y el otro par a **2A** y **2B**.

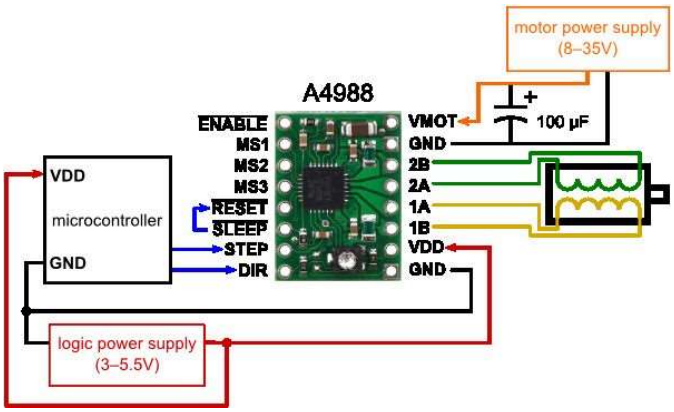


Figura 10 *- Conexionado Motor NEMA 17 – Pololu A4988*

Fuente: *A4988 datashet*

6.3.3 Inicio, protecciones y configuración del motor nema 17:

Las conexiones a 12V (LED e interface A4988) se conmutan a voluntad por medio de un interruptor de paso. Además tienen una protección adicional por medio de fusible que evita daños al circuito por sobretensiones.

Las salidas de las bobinas del motor nema 17, están protegidas por 4 diodos para evitar retornos de tensiones, fruto de movimientos involuntarios del eje motor que genere en sentido contrario tensiones fuera de control.

Los datos analógicos variables (ángulo máximo de barrido del motor PIN A0 y velocidad de giro PIN A1), están gestionados por sendos potenciómetros de 10K que configuran dicho valor entre dos valores que limitan el mínimo y el máximo de su resistividad.

6.3.4 Conexión LASER y Webcam:

La conexión LASER se realiza directamente a la fuente conmutada de +/-12 V. Se iniciará su funcionamiento mediante un interruptor y tendrá como protección un

[4] https://www.staticboards.es/blog/motores-paso-paso/

fusible que evite los sobre picos de la fuente de alimentación.

La intensidad es clave, pues la captura de datos se ve condicionada por el grosor e intensidad del haz emitido. Para ejercer control sobre el mismo un potenciómetro de 10 K regulara a voluntad la intensidad del mismo.

La webcam no ira implementada en este circuito y se conectara directamente a un puerto USB liberado del ordenador.

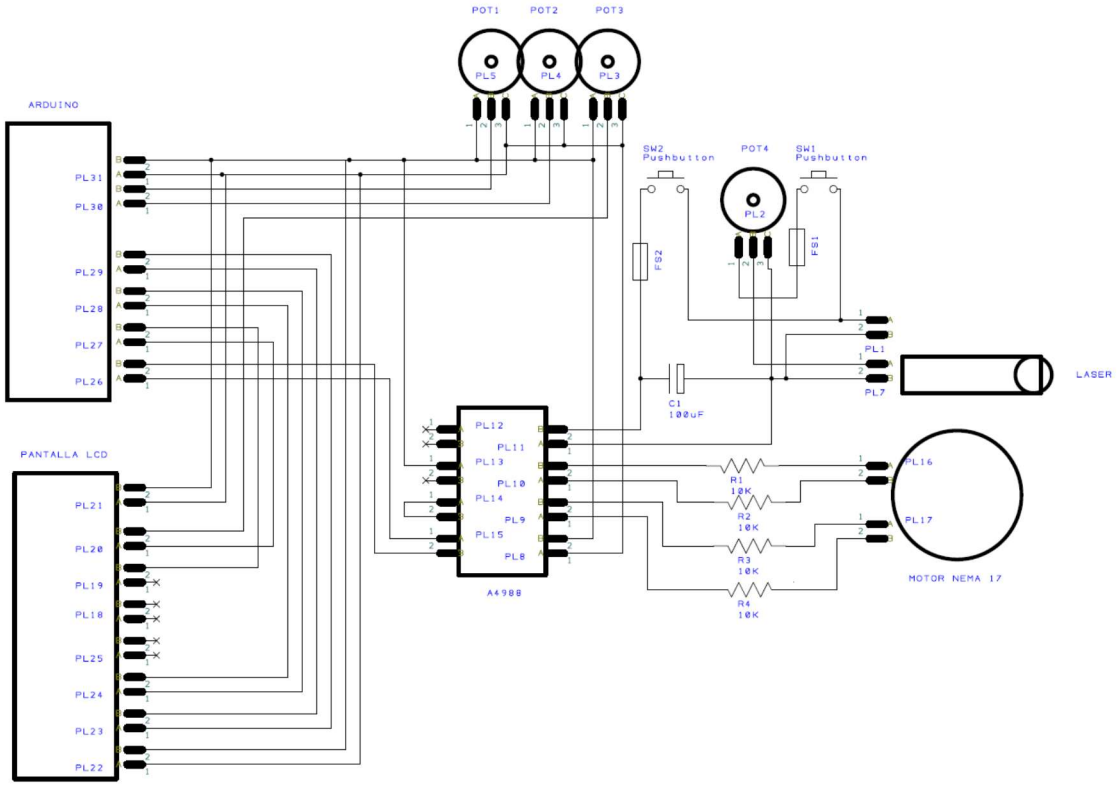


Figura 11 - *Exquema general del cableado – Sistema filar.*

Fuente: *elaboración propia.*

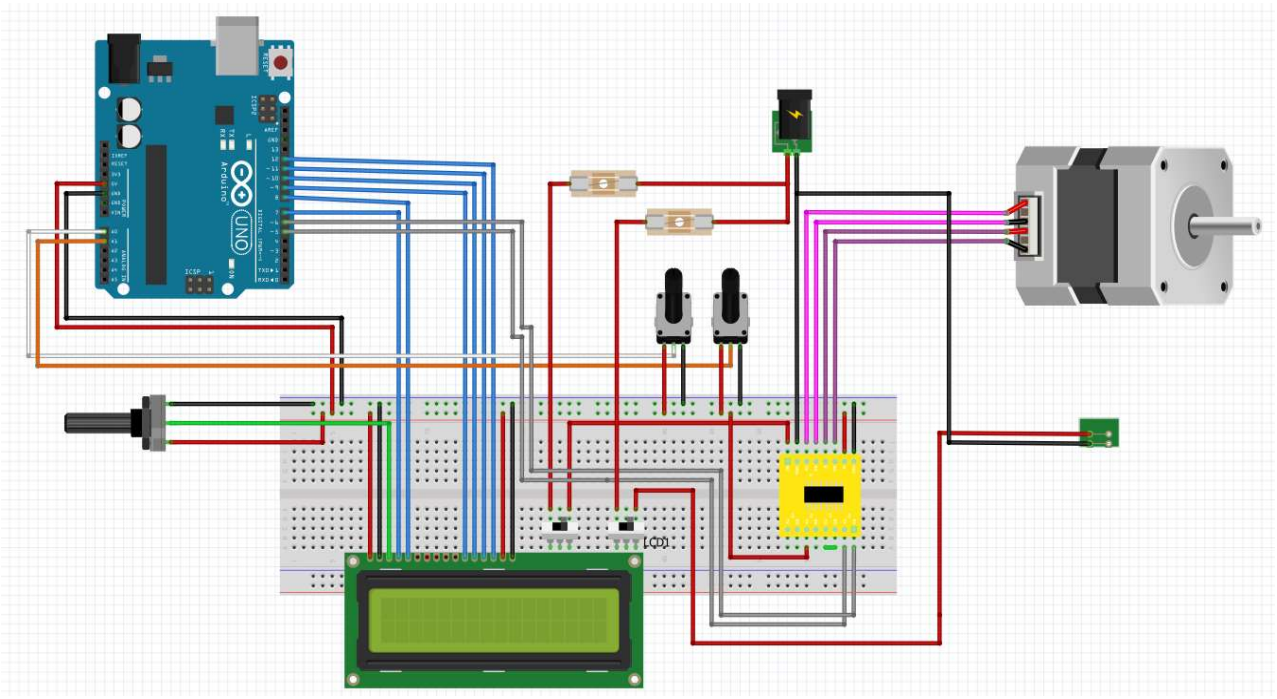


Figura 12 - *Exquema general del cableado – Cableado protoboard.*

Fuente*: elaboración propia.*

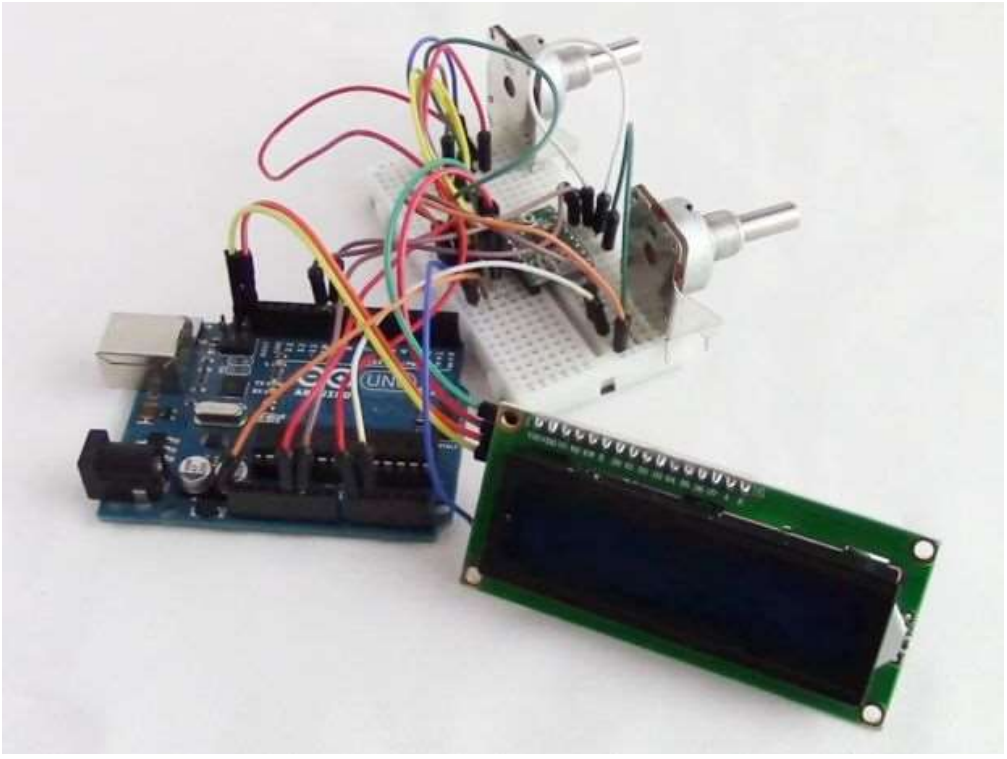


Figura 13 - *Exquema general del cableado – Montaje real.*

Fuente: *3D-Environment-Laser-Scanner-From-Scratch.*

6.4. DESARROLLO DE LA ESTRUCTURA EXTERNA (Configuración propuesta):

La implementación física de todos los componentes descritos anteriormente se ha fundamentado en el diseño y nueva construcción de una estructura que de soporte a todo el conjunto y que se presenta a modo de propuesta, parte de este proyecta versa no solo en la solución teórica, sino también en la solución práctica, de tal modo que lo expuesto pueda ser demostrado mediante la interacción física y real de los componentes presentados. Es por ello que la maquina descrita se construirá, y es en esta etapa del desarrollo del proyecto donde se presentará la propuesta constructiva con una versión ajustada a tal efecto.

Se ha intentado proporcionar un acceso directo y fácil a todos los elementos de seguimiento y control para una fácil interactuación y modificación externa de parámetros (pantalla LCD, interruptores y potenciómetros de configuración.

Como los elementos empotrados en el frontal de la maquina mecanizado impide un correcto cableado directo, se ha fabricado una tarjeta electrónica que hace de interface entre la conexión de los periféricos y el final conexionado con las tarjetas y módulos de control.

Los planos constructivos se presentan con más detalle en los anexos finales del documento.

6.4.1. Brazo giratorio:

Se construirá en las medidas que se indican a continuación.

- **Distancia cámara – Laser**: 350 mm
- **Distancia centro eje motor – Cámara**: 160 mm
- **Fijación Laser al brazo pivotante**: 90º
- **Fijación Cámara al brazo pivotante**: Oscilante entre 10º para enfoque objetos cercanos a escanear – 40º máximo para enfoque objetos a gran distancia. Debe ser regulable.

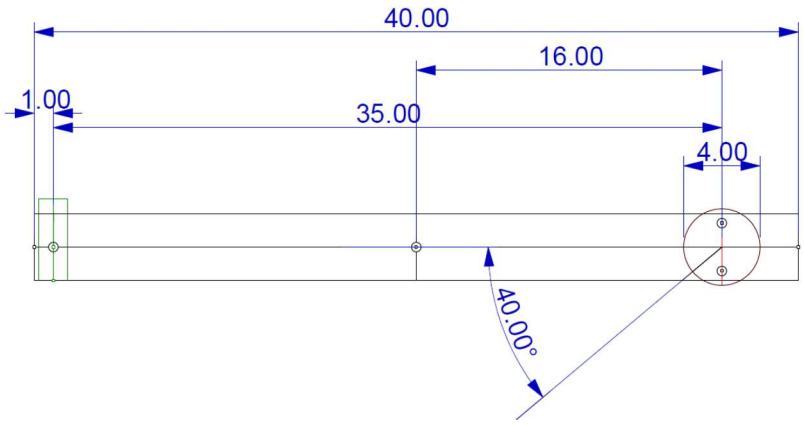


Figura 14 - Disposición general brazo giratorio

Fuente: elaboración propia

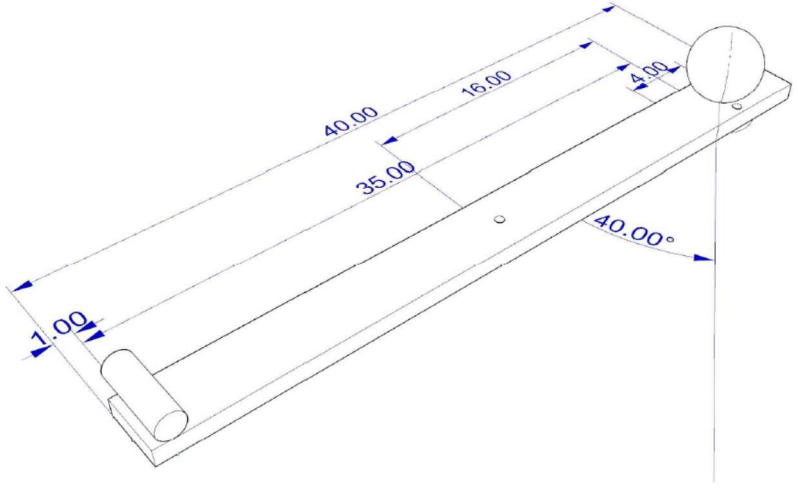


Figura 15 - Disposición en perspectiva del brazo giratorio

Fuente: elaboración propia

Los materiales deben ser ligeros y que ofrezcan un buen sistema de fijación al eje del motor. Madera de baja densidad o mecanizado de Aluminio podría ser una opción interesante. Como nota, dejar posibilidad de ajuste a la orientación de la Webcam según lo que se ha indicado anteriormente.

6.4.2. Frontal instrumentos:

El panel frontal estará taladrado para dejar espacio a los elementos de intercambio de información. Así pues encontraremos el interruptor de encendido/apagado del sistema, y los potenciómetros de control de ángulo máximo y velocidad de rotación.

Material elegido: Lamina de policarbonato en medidas de 3 mm.

La zona de estos elementos queda dividida en una secundaria donde se colocaran los elementos de potencia de control del LÁSER. Aquí se encontrara el interruptor de encendido y apagado y también el potenciómetro que regulara la intensidad del mismo.

El remate final se realizara mediante una cobertura adhesiva con los códigos de color, escalas graduadas y anotaciones a leer antes de inicializar la máquina[5].

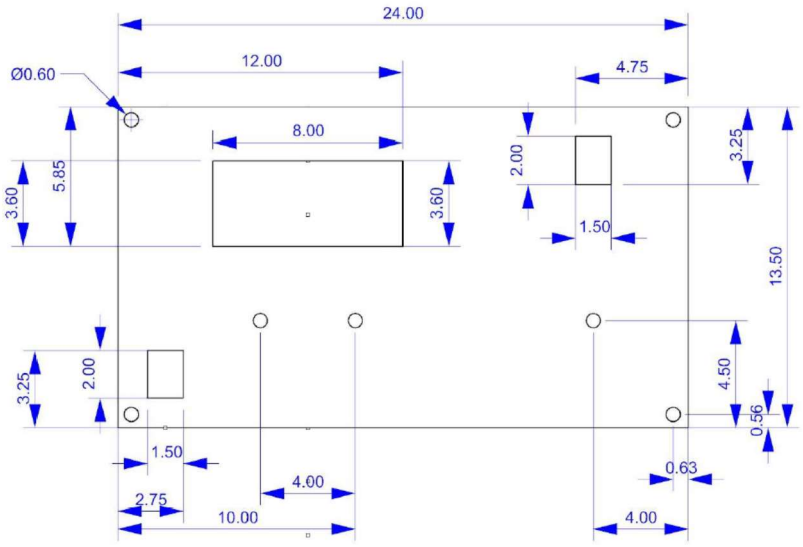


Figura 16 - Disposición frontal del panel de instrumentos

Fuente: elaboración propia

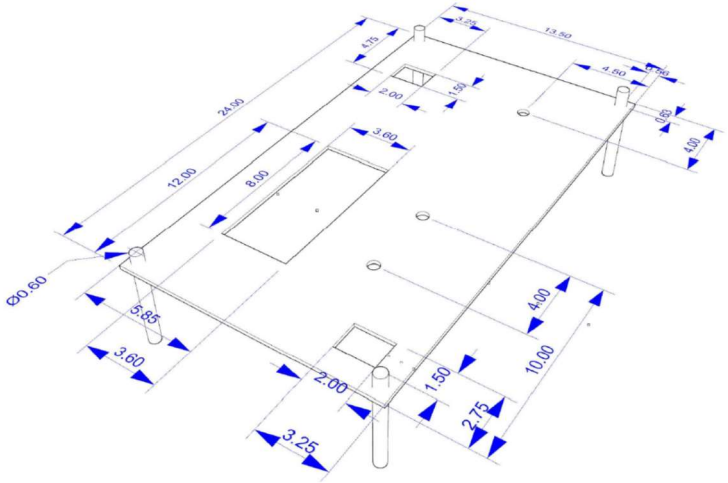


Figura 17 - Disposición en perspectiva del panel de instrumentos

Fuente: elaboración propia

[5] Para más detalle, consultar los planos constructivos en los anexos finales.

6.4.2.1. Disposición de instrumentos:

En los espacios taladrados, se colocaran los instrumentos que harán de interface a la hora de control y programación de los parámetros de la máquina.

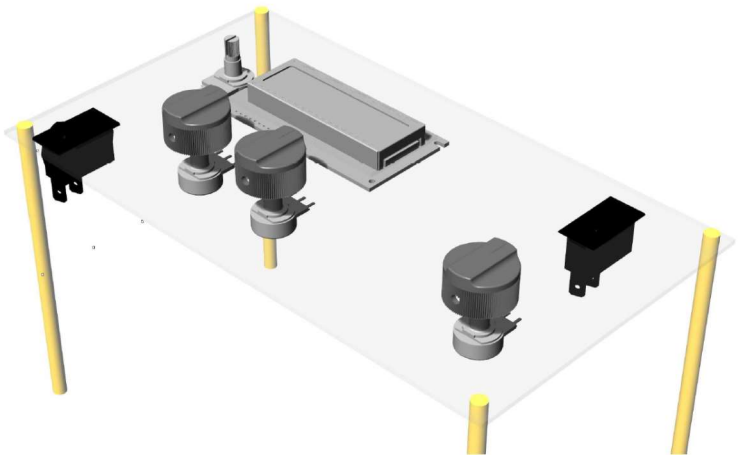


Figura 18 - Disposición en perspectiva renderizada del panel de instrumentos

Fuente: elaboración propia

6.4.3. Parte trasera:

El panel trasero de la caja de instrumentos será confeccionado del mismo material que el panel frontal (policarbonato transparente de 3 mm en medidas similares).

Debido a la concentración de instrumentación, y aunque los consumos están controlados, se ha añadido una fuente extra de ventilación forzada mediante un ventilador de flujo continuo de 12 V.

La idea es la refrigeración de:

- Fuente de alimentación de 220/12V
- Parte inferior motor NEMA 17.
- Chip Pololu A4988.

El ventilador ira empotrado en la parte central de dicho panel.

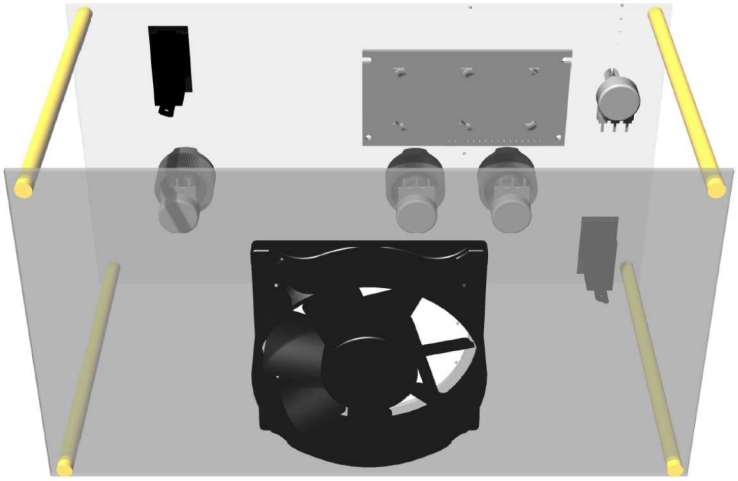


Figura 19 – Disposición trasera en perspectiva renderizada.

Fuente: elaboración propia.

6.4.4. Estructura general y distribución electrónica.

Se ha dotado de una estructura ligera a la vez que armoniosa que de consistencia y estabilidad a todo el conjunto asi como a los dispositivos que han de ser asegurados en el interior de la misma.

Para ello, se ha elegido un sistema de pilares y traviesas construidos en madera de balsa[6] .

Estas estructuras reforzarán las plataformas y los tirantes darán soporte a los diferentes elementos electrónicos mencionados.

Tanto el panel frontal como el trasero irán ajustados mediante 4 varillas roscadas, que facilitaran su des-ensamblaje cuando sea necesario acceder al interior del cableado para ajustar o modificar algún modulo en cuestión.

[6] ver medidas y dimensiones en los anexos finales.

6.4.4.1. Elementos estructurales.

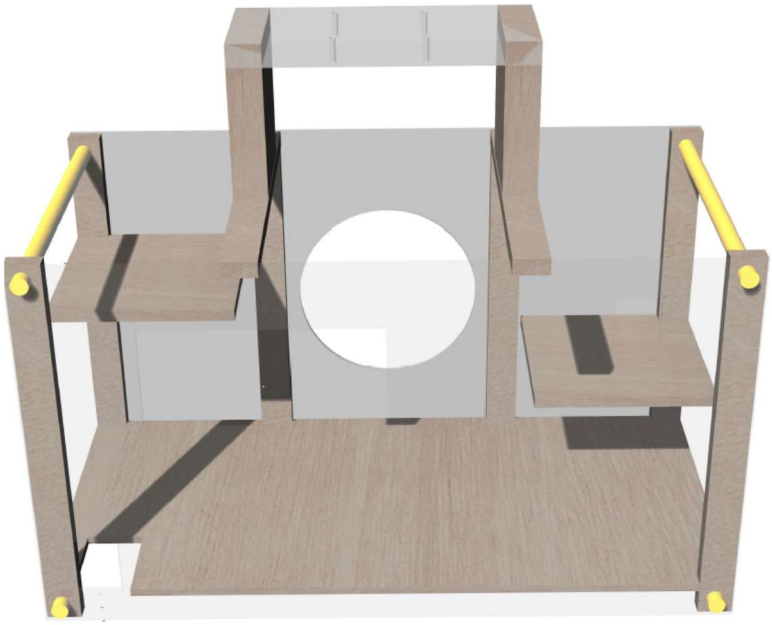


Figura 20 – Render estructura.

Fuente: elaboración propia.

6.4.4.2. Soportados:

6.4.4.2.1. Soportado Fuente de alimentación

Aprovecharemos que la fuente de alimentación es el elemento de más volumen y peso para colocarlo en la posición que más estabilidad confiera al sistema, teniendo en cuenta las inercias del brazo giratorio y el peso del motor, que ha de ser colocado en posición elevada.

El soportado será la propia base de la estructura de la caja de control, donde atornillaremos la fuente de alimentación a ella utilizando los orificios horadados de fábrica.

Las medidas y dimensiones tanto de la plancha de madera como de los taladros y tornillos a emplear se especifican en los anexos finales.

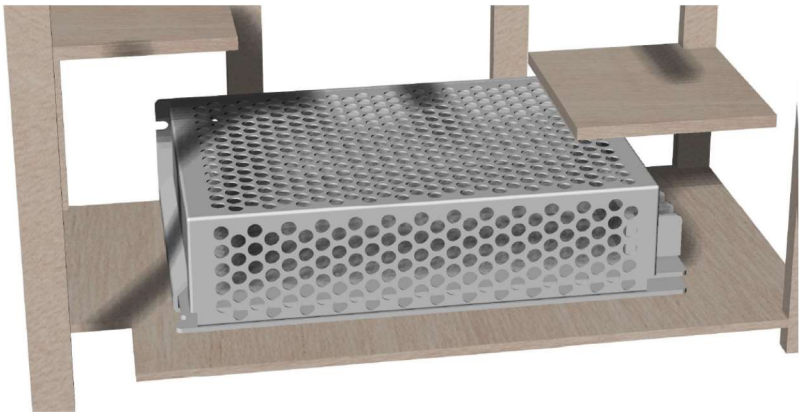


Figura 21 – Render soportado Fuente Alimentación.

Fuente: elaboración propia.

6.4.4.2.2. Soportado tarjeta arduino:

El soporte del microcontrolador ARDUINO será colocado en la parte derecha de la caja de instrumentos. El microcontrolador está provisto de su propio encapsulado (aunque no se aprecie en el renderizado) e irá atornillada o sujeto con bridas plásticas en la posición definida en una plataforma auxiliar por encima de la fuente de alimentación.

Nótese que la posición no es banal, y la conexión USB será dirigida hacia la parte derecha del encapsulado. Igual que la conexión a la red eléctrica de la fuente de alimentación.

Esto permite un flujo más limpio al cableado de los componentes empotrados en los paneles y los cables dirigidos a la tarjeta electrónica de control adicional que se ha añadido al conjunto.

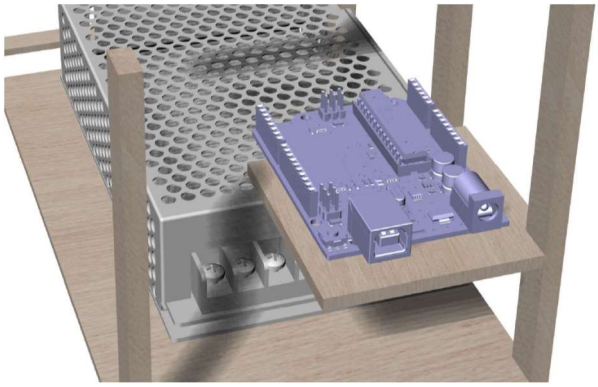


Figura 22 – Render soportado microcontrolador

Fuente: elaboración propia.

6.4.4.2.3. Soportado electrónica de interface

Debido a la deslocalización de los componentes, se ha decidido implementar una nueva electrónica que haga de puente entre el microcontrolador, alimentación y los periféricos.

Con ello podemos hacer más accesible y funcional la interconexión de todos los elementos que hacen funcionar el circuito.

La electrónica está basado en el circuito de la figura 23, y se ha configurado en una PCB con el ruteo siguiente:

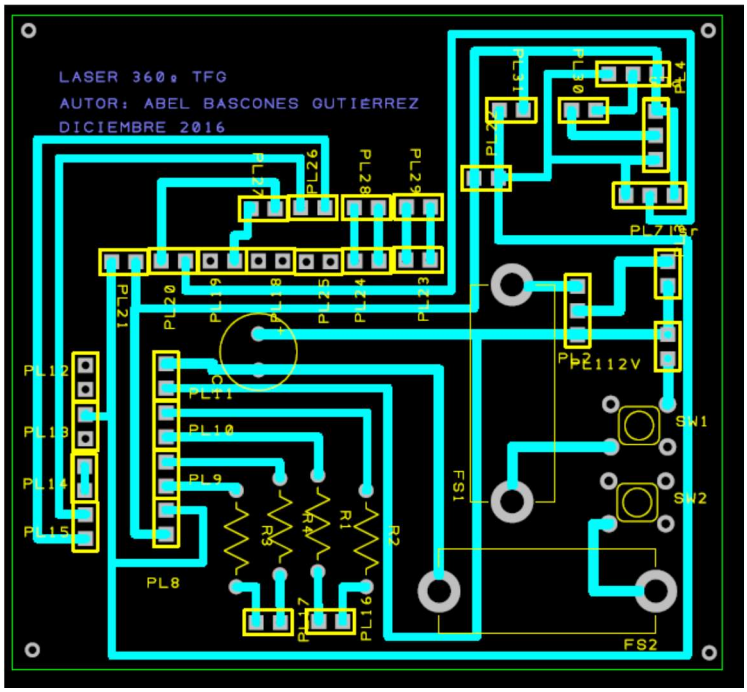


Figura 23 – Diseño PCB

Fuente: elaboración propia.

Las características físicas de los elementos se pueden ver a continuación:

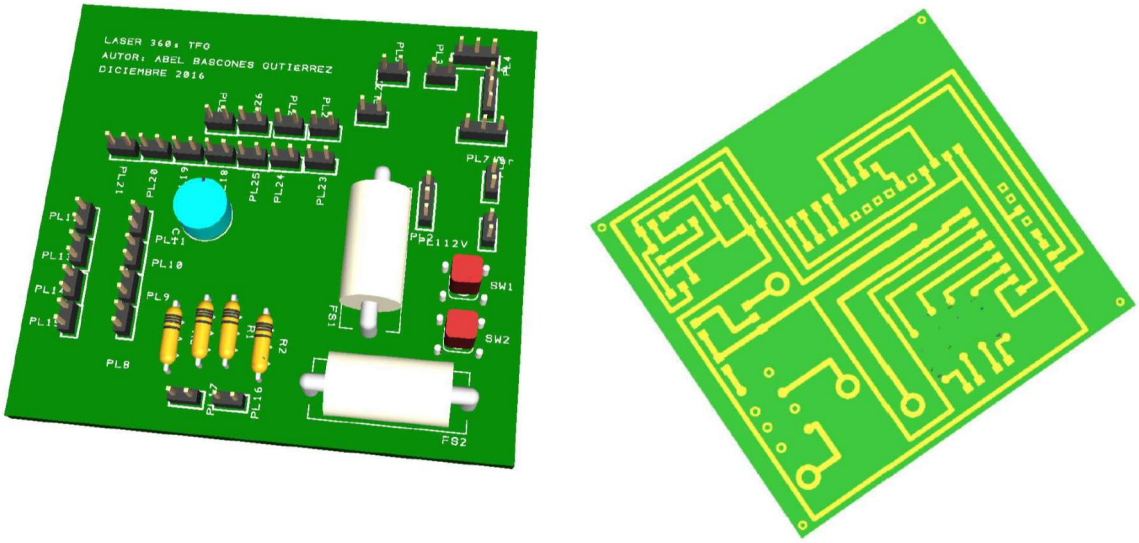


Figura 24 – Diseño PCB imagen 3D

Fuente: elaboración propia

Dentro del sistema, se ha reservado una plataforma de similares características a la utilizada en el Arduino para posicionar dicha PCB orientada adecuadamente de tal modo que se pueda acceder fácilmente a los pines de conexión para poder chequear con el polímetro cualquier fallo o malfuncionamiento, haciendo de los métodos predictivos y de mantenimiento un asunto muy a tener en cuenta en este proyecto.

Su posición en el montaje se detalla a continuación:

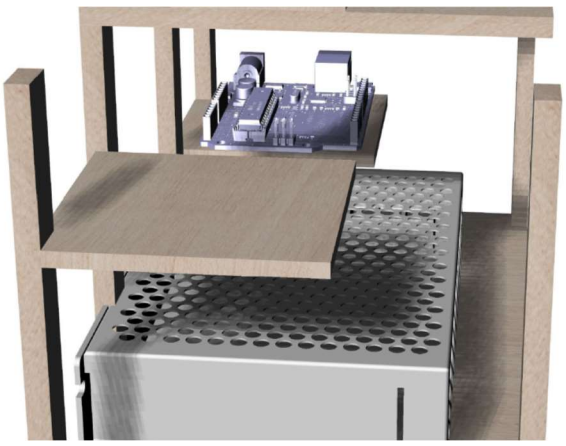


Figura 25 – Render soportado PCB

Fuente: elaboración propia

6.4.4.2.4. Soportado motor + brazo giratorio.

Debido al libre giro del brazo que da soporte a la cámara y al laser, se ha decidido colocar el conjunto motor + brazo libre de obstáculos

Para ello se ha integrado el motor atornillado por la parte superior en una plancha de policarbonato unido a dos soportes de madera, que mantendrán suspendido y firme el motor paso paso sobre el resto de los elementos de la máquina de escaneo laser.

Con esto, aparte de proporcionar giro libre al brazo, mantendremos refrigerada la parte inferior del motor gracias a la redirección del chorro de aire proporcionado por el ventilador colocado en la parte trasera de la caja, más propensa a calentarse tras largos periodos de funcionamiento.

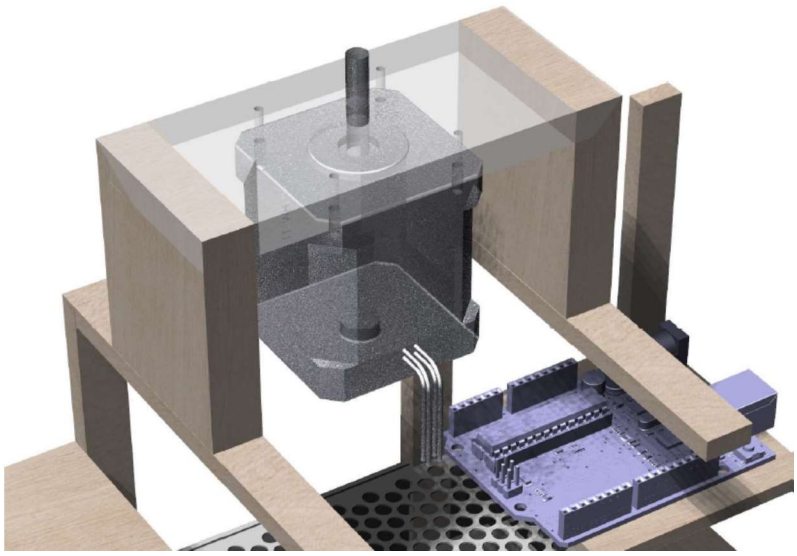


Figura 26 – Render soportado motor NEMA 17

Fuente: elaboración propia

Aspecto de la maquina con todos los complementos modelados según la opción ofrecida y en ausencia del cableado:

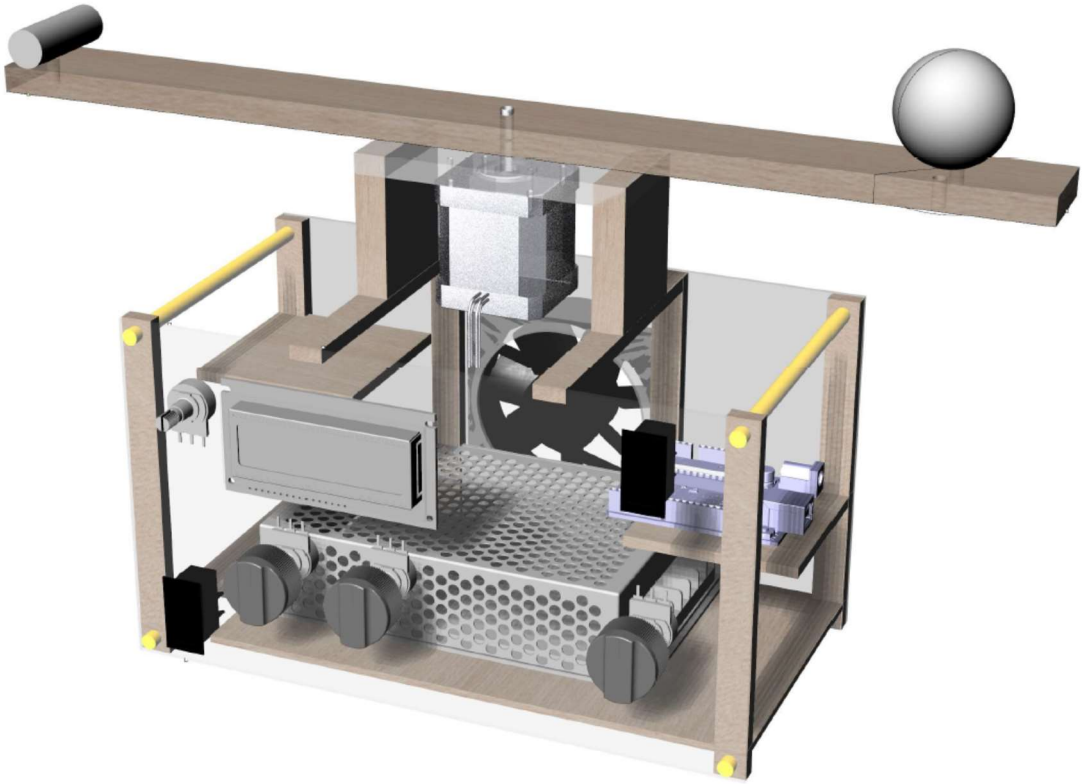


Figura 27 – Render conjunto completo de la máquina

Fuente: elaboración propia

7. ELABORACION SOFTWARE:

Los códigos han sido generados con el siguiente software libre:

- PROGRAMA CONTROL ARDUINO: **IDE 1.6.5. R5 para windows.**
- PROGRAMAS PROCESADO DE IMAGENES Y TRIANGULACION DE PUNTOS: **Processing**

El método está basado en la triangulación[7]. Esto significa que debe formarse un triángulo, teniendo la cámara, el dispositivo láser y un punto desconocido como vértices.

Utilizando la ley de los senos, se puede calcular la distancia desde el punto desconocido, resolviendo así el problema geométrico 3D:

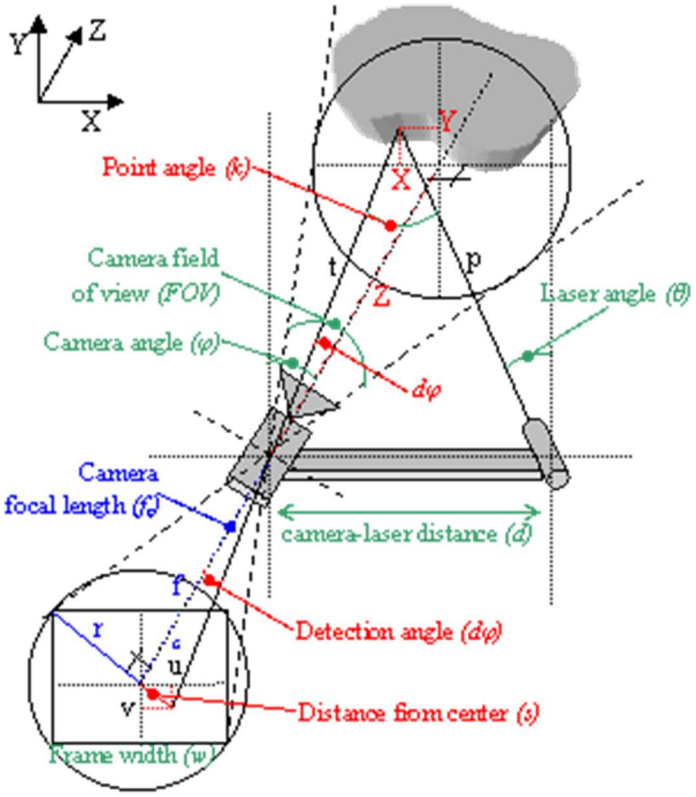


Figura 28 – Proceso físico de triangulación

Fuente: http://georgepavlides.info/research/LaserScanningAndTriangulation.php

[7] http://georgepavlides.info/research/LaserScanningAndTriangulation.php

La distancia de referencia entre la cámara y el diodo láser se denota por **d,** mientras que **ϕ** es el ángulo de la cámara y **θ** el ángulo del láser, ambos con el eje vertical a la línea que los conecta. En total, las variables que son los parámetros conocidos de esta disposición son (para un modelo de cámara estenopeica):

- El campo de visión de la cámara (ángulo**, FOV**)
- La resolución del fotograma de la cámara, es decir, la anchura **w** del marco y la altura **h** del bastidor.
- La topología relativa de la disposición, es decir, los ángulos de cámara y láser **ϕ** y **θ** y la distancia entre ellos (**d**).
- El valor clave de la distancia focal de la cámara puede deducirse de los parámetros conocidos:

$$\left.\begin{array}{l}\tan\left(\frac{FOV}{2}\right)=\frac{r}{f_c}\\ (2r)^2=w^2+h^2\end{array}\right\}\Rightarrow fc=\frac{\frac{\sqrt{w^2+h^2}}{2}}{\tan\left(\frac{FOV}{2}\right)}$$

La triangulación se basa en la ley de los senos, que establece que si los lados de un triángulo arbitrario son a, byc los ángulos opuestos a esos lados son A, B y C

$$\frac{a}{\sin A}=\frac{b}{\sin B}=\frac{c}{\sin C}=2R$$

Donde **R** es el radio de la circunferencia del triángulo.

En el caso descrito, (2) se convierte en:

$$\frac{d}{\sin k}=\frac{p}{\sin\left(\frac{\pi}{2}-\varphi+d\varphi\right)}=\frac{t}{\sin\left(\frac{\pi}{2}-\vartheta\right)}$$

En esta topología el problema es un típico problema geométrico que se puede resolver fácilmente, ya que:

$$k+\left(\frac{\pi}{2}-\varphi+d\varphi\right)+\left(\frac{\pi}{2}-\vartheta\right)=2\pi\Rightarrow$$
$$k=2\pi-\left(\frac{\pi}{2}-\varphi+d\varphi\right)-\left(\frac{\pi}{2}-\vartheta\right)=\pi+\varphi+\vartheta-d\varphi$$

Y, a partir de esto, la distancia desconocida de la cámara es:

$$\frac{d}{\sin k} = \frac{t}{\sin\left(\frac{\pi}{2} - \vartheta\right)} \Rightarrow t = d\frac{\sin\left(\frac{\pi}{2} - \vartheta\right)}{\sin(\pi + \varphi + \vartheta - d\varphi)}$$

La única variable desconocida aquí es el ángulo **dϕ**, que puede ser fácilmente estimado trigonométricamente:

$$\tan(d\varphi) = \frac{s}{f_c} \Rightarrow d\varphi = \arctan\left(\frac{s}{f_c}\right) \Rightarrow$$
$$d\varphi = \arctan\left(2\frac{\sqrt{u^2 + v^2}}{\sqrt{w^2 + h^2}}\tan\left(\frac{FOV}{2}\right)\right)$$

En la práctica, es más conveniente estimar las coordenadas X, Y, Z del punto detectado en lugar de su distancia desde la cámara **t**. Esto puede lograrse si en lugar de trabajar con la distancia s estimamos los ángulos en los ejes **X** e **Y** por separado. La noción se representa gráficamente en la Fig. 28, donde el ángulo de detección **dϕ** está representado por dos ángulos que son relativos a un eje, es decir, **dϕX** para X y dϕY para Y.

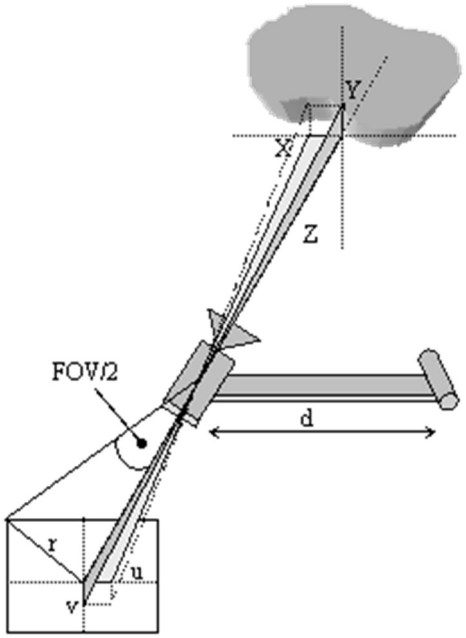


Figura 29 – Estimación de las coordenadas de un punto detectado

Fuente: http://georgepavlides.info/research/LaserScanningAndTriangulation.php

La estimación de estos ángulos es equivalente a la estimación de dϕ en (6). Las ecuaciones finales son:

$$d\varphi_x = \arctan\left(\frac{u}{r}\tan\left(\frac{FOV}{2}\right)\right)$$

$$d\varphi_y = \arctan\left(\frac{v}{r}\tan\left(\frac{FOV}{2}\right)\right)$$

$$L = 2\tan\left(\frac{FOV}{2}\right)\Big/\sqrt{w^2+h^2}$$

$$Z = d\frac{\cos(\arctan(uL))\sin(\pi-\vartheta)}{\sin((\pi-\vartheta)+\varphi+\arctan(uL))}$$

$$Y = ZvL$$

$$X = ZuL$$

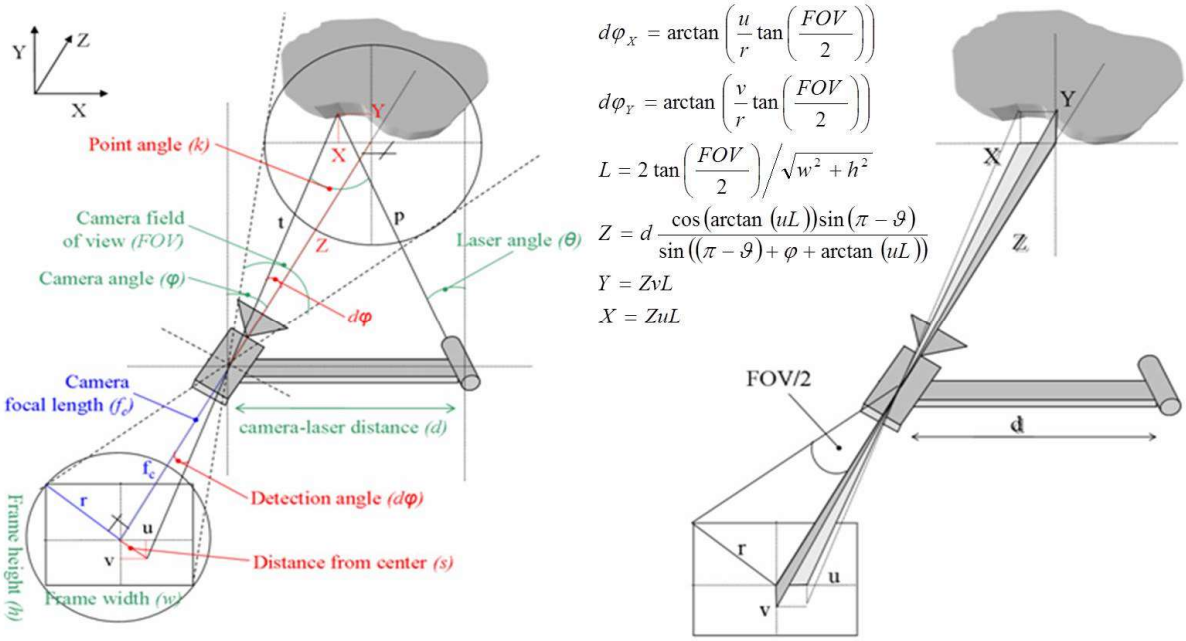


Figura 30 – Estimación de las coordenadas de un punto detectado

Fuente: http://georgepavlides.info/research/LaserScanningAndTriangulation.php

7.1. CONTROL ANGULO MOTOR PASO A PASO

7.1.1. Objetivo del programa:

Programa desarrollado en ARDUINO ®, con el objetivo de realizar el control de un motor paso a paso modelo NEMA 17, cuya señal de control estará gestionada desde el microcontrolador ARDUINO, a través de un interface de control modelo A4988 conectado a los pines de entrada y de salida de datos.

El ángulo de barrido, también se podrá modificar mediante dos potenciómetros que gestionan el control de la velocidad de barrido y el control de abertura máxima de dicho barrido.

Todos estos parámetros variables podrán ser controlados y seguidos a través de una pantalla LCD de dos líneas para monitorizarlos en tiempo real.

El programa, además, controla la gestión de entrada y salida de datos mediante el puerto serie (creando un puerto COM virtual para evitar interferencias del flujo de datos en caso que otros dispositivos usaran u ocuparan el resto de los puertos series físicos del equipo). Esta información también podrá ser gestionada y utilizada por el software creado en PROCESSING®. La posición del conjunto cámara – laser montados en el mismo brazo generada por ARDUINO será leída en PROCESSING ® para procesar las imágenes obtenidas y calcular la triangulación del reflejo del haz laser fotograma a fotograma.

7.1.2. Explicación detallada del programa:

7.1.2.1. *Definiciones iniciales del programa*:

Comenzamos importando las librerías en ***ARDUINO*** necesarias para poder acceder a los periféricos con los que interactuaremos a la hora de leer los datos generados.

Si no tenemos dichas librerías, deberemos proceder a descargarlas e instalarlas en el apartado del IDE dedicado a tal efecto[8].

Se procede a precargar la librería **<Wire.h>** que gestiona el control en el puerto serie y su flujo de datos, del ARDUINO al PC y viceversa.

[8] Ver ayuda en el IDE de programación o en la web www.arduino.com . Se suponen conocimientos mínimos de programación y manejo del editor de código.

A continuación, se procede a precargar la librería **<LiquidCrystal.h>**, para efectuar el control sobre la pantalla LCD. Procedemos a la inicialización de los pines de control del LCD, en este caso los pines 7, 8, 9, 10, 11 y 12. Más adelante se explicará en el programa el uso dado a cada pin.

Declaramos como enteros:

- **PIN 6 Arduino = dir,** conectado al interface A4988 y que controlara el giro del motor (0V sentido horario, 5V sentido anti horario).
- **PIN 5 Arduino = stp**, conectado al interface A4988 y que controlará el paso del motor.
- **PIN 0 Arduino = pot1**, conectado al potenciómetro de control 1, que controlara la velocidad de giro del motor.
- **PIN 1 Arduino = pot2**, conectado al potenciómetro de control 2, que controlara el ángulo máximo que alcanzará el motor.

```
27 // CODIGO:.....................................................
28
29
30 #include <Wire.h>
31 #include <LiquidCrystal.h>
32
33 LiquidCrystal lcd(7,8,9,10,11,12);
34
35 int dir = 6;
36 int stp = 5;
37 int pot1 = 0;
38 int pot2 = 1;
```

Figura 31 – Extracto de código ARDUINO líneas 27-38.

Fuente: elaboración propia.

7.1.2.2. *Desarrollo del programa*:

Una vez realizadas las definiciones iniciales, procedemos a la inicialización de la parte general del código invocando a la función vacía **void setup ().**

Inicializamos la comunicación serie. Usaremos una velocidad de datos de 9600 mps, que es la velocidad de transmisión por defecto del ARDUINO modelo UNO

Proseguimos configurando los pines del ARDUINO definidos anteriormente:

- PIN 6 (**dir**) SALIDA.

- PIN 5 (**stp**) SALIDA.
- PIN 0 (**pot1**) ENTRADA.
- PIN 1 (**pot2**) ENTRADA.

Inicializamos la pantalla LCD, con el comando **lcd.begin (16, 2);** definiendo el tipo de LCD que estamos utilizando (16 caracteres, 2 filas).

```
39 //---------------------------------
40 void setup(){
41
42   Serial.begin(9600);
43
44   pinMode(dir, OUTPUT);
45   pinMode(stp, OUTPUT);
46   pinMode(pot1, INPUT);
47   pinMode(pot2, INPUT);
48
49 lcd.begin (16, 2);
50
51 }
52 //---------------------------------
```

Figura 32 – Extracto de código ARDUINO líneas 39-52.

Fuente: elaboración propia.

Definimos como entero el valor "***del***", que indica el retardo del ciclo del programa, es decir, determina la velocidad del motor mediante el valor analógico leído en el potenciómetro de control 1.

Se define como valor flotante (0.000 hasta cuatro decimales) el ángulo de abertura inicial y desde donde comienza a tomar referencias cada captura de imágenes. Por defecto se inicializa en 0º.

Se define como valor entero "***Angle end",*** que determina el ángulo máximo de abertura que alcanzara el motor. Este valor es configurable mediante el potenciómetro de control nº2.

Se define como valor entero ***"spd***", que hace referencia a la velocidad. Este dato es fijo, porque los pulsos en voltaje e intensidad gestionados son constantes. Solo se tiene en cuenta a efectos de retardos en la ejecución del código y solo tangible en la velocidad de monitorización de datos a través del monitor LCD.

Se define como valor entero "***pass",*** que sirve para denominar el número de barridos o pasadas a realizar en cada dirección. En este caso se inicializaran en cero.

Se define como valor entero ***"repeat",*** es el número efectivo de barridos a realizar. Este valor es configurable pero solamente por programa[9]. A un número mayor de pasadas, más definición en el escaneo, pero mayor es el tiempo de espera.

```
52 //---------------------------------
53 int del;
54 float angle = 0;
55 int angleEnd;
56 int spd;
57 int pass = 0;
58 int repeat = 360;
59 //---------------------------------
```

Figura 33 – Extracto de código ARDUINO líneas 52-59.

Fuente: elaboración propia.

En el siguiente bloque se describe el bucle principal **"void loop () {",** que se ejecutará cíclicamente hasta que se alcancen los valores programados previamente (ángulo máximo alcanzado y numero de pasadas).

Se procede a la lectura analógica de los valores obtenidos desde el potenciómetro 1 que servirá para configurar dos valores que serán básicos en el desarrollo del código en una etapa posterior: el retardo (**del**) y la velocidad de rotación del motor paso paso (**spd**). El código ejecuta un mapeo de los valores de las bobinas de los potenciómetros a base de una comparación al alta.

Lo mismo ocurre con la lectura del potenciómetro 2, que define el ángulo de abertura máximo que alcanzara el motor paso a paso.

```
61
62 //---------------------------------
63
64 void loop(){
65
66 //1°.............................
67
68   del = map(analogRead(pot1), 0, 1023, 500, 50);
69   spd = map(analogRead(pot1), 0, 1023, 1, 100);
70   angleEnd = map(analogRead(pot2), 0, 1023, 0, 360);
71
72 //2°.............................
73
```

Figura 34 – Extracto de código ARDUINO líneas 61-73.

Fuente: elaboración propia.

[9] En una versión posterior se generara mediante un interface externo para precargarlas antes de ejecutar el código.

Dentro del bucle, y una vez se hayan leído los valores anteriores, se alamcena en el búffer de memoria para proceder con las rutinas principales que se basarán en comparaciones de los nuevos datos leídos con los valores de referencia.

En primer lugar, se realiza una comparación (secuencia if – else) donde, si el número de pasadas realizadas (referencia configurable en el código del programa) es mayor al número de barridos (valor leído del contador) se procederá con la sub secuencia deonde se realiza una segunda comparación if-else, del valor ***"pass",*** o el barrido del motor, inicializado a 0 y que se añade +1 cada vez que completa un y lo compara con un valor alto (1) o bajo (0), con esto detecta el sentido de giro del motor (0 sentido horario, 1 sentido anti horario). Nótese que cada avance supone el incremento o decremento de un cuarto del paso total, que para este modelo de motor (NEMA 17) es de 1,8º.

El programa seguirá ejecutando las rutinas dentro del bucle hasta que llegue al final. Esto se configura por programa, estableciendo un límite de pasadas de 500.

Cuando esto ocurre, esta información a modo de finalización de programa saca por la ventana de comandos a través del puerto serie el mensaje: "***500.0***"

Si se cumple que ***pass < repeat, y pass%2 == 0***, damos paso al avance de paso, que tras un retardo de 300 milisegundos se para, enviando una señal alta y luego una señal baja al interface A4988 a través del PIN 5 stp. 300 milisegundos, es el tiempo necesario para que el motor paso paso avance ¼ del paso.

```
71
72 //2°.............................
73
74   if (pass < repeat){
75
76     if(pass%2 == 0){
77       digitalWrite(dir, LOW);
78       angle = angle + 1.8/4;
79     }
80     else if(pass%2 == 1){
81       digitalWrite(dir, HIGH);
82       angle = angle - 1.8/4;
83
84       Serial.println(500.0);
85     }
86
87     digitalWrite(stp, HIGH);
88     delayMicroseconds(300);
89     digitalWrite(stp, LOW);
90   }
91
92 //---------------------------------
```

Figura 35 – Extracto de código ARDUINO líneas 71-92.

Fuente: elaboración propia.

Finalmente, se produce la orden que actualiza el contador del paso en cada barrido. Para ello se produce una comparación entre el ángulo actual del eje del motor y el ángulo final (configurado a través del potenciometro2) en donde si el ángulo actual es mayor o igual que el ángulo configurado externamente y si el ángulo actual es mayor o menor que 0º, y si además (condición forzada AND) el paso actual es menor que repeat (360º por programa) el paso avanza (pass ++, contador).

Esto es así porque puede ocurrir que manipulemos el potenciómetro de ángulo durante la ejecución del programa, variando los datos del mismo durante la propia ejecución. Así pues si en un ciclo previo el valor de referencia es 90º y estamos en una magnitud angular de 40º la condición se cumple (quedan 50º), pero si manipulamos el potenciómetro a la baja y la referencia de 90º la variamos a 45º como ángulo máximo a alcanzar, el programa se para ya que el valor actual angular creciente ha superado el valor máximo recién leído.

El programa se parará hasta volver a configurar un ángulo máximo a alcanzar igual o mayor al actual.

```
91
92 //---------------------------------
93
94   if ((angle >= angleEnd || angle <=0) && pass < repeat){
95     pass++;
96   }
97
98 //---------------------------------
99
```

Figura 36 – Extracto de código ARDUINO líneas 91-99.

Fuente: elaboración propia.

7.1.2.3. *Salida de datos por pantalla*:

En este último bloque del código, se tratan todas las salidas por los periféricos usados a modos de seguimiento y control de la ejecución del mismo, tanto por la pantalla LCD externa como por el puerto serie a través de la pantalla del PC.

Comenzamos con la instrucción Serial.println (angle); con esta imprimimos el valor actual del ángulo del eje del motor NEMA por el puerto serie.

A continuación, procedemos a la impresión de los caracteres a través de la pantalla LCD.

El procedimiento es repetitivo y se realiza de manera similar para cada

carácter. Primeramente se posiciona el puntero en la posición definida por dos caracteres, que son las coordenadas de los espacios permitidos para escribir. Usamos un LCD de dos filas con 16 caracteres cada una, es decir una matriz de 2 X 16 caracteres.

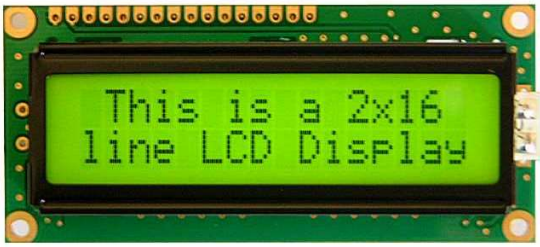


Figura 37 – Imagen pantalla LCD 2x16 caracteres.

Fuente: elaboración propia.

Una vez posicionado el cursor ***"lcd.setCursor (0,0);"*** en la fila (0-1) y el carácter (0-16), con el comando ***"lcd.print ("Angul");",*** procedemos a imprimir los caracteres "a, n, g, u, l". Estos son fijos y no variaran. El valor variable viene a continuación.

Posicionamos de nuevo el cursor:***" lcd.setCursor (6,0);"*** en la fila 1, carácter 6 y con el comando: ***"lcd.print (" ");"*** reservamos el espacio para el carácter variable que será buscado en la variable entera donde está guardado, ***"lcd.print (int (angle))".***

El proceso se repite con todos los caracteres fijos y variables que se deseen mostrar a través de la pantalla de cristal líquido.

Tras el proceso de volcado de datos y escritura, forzamos un retardo y este estará condicionado por el valor leído del potenciómetro 1.

```
98  //---------------------------------
99
100   Serial.println(angle);
101
102   lcd.setCursor(0,0);
103   lcd.print("Angul");
104   lcd.setCursor(6,0);
105   lcd.print("   ");
106   lcd.setCursor(6,0);
107   lcd.print(int(angle));
```

Figura 38 – Extracto de código ARDUINO líneas 98-100.

Fuente: elaboración propia.

```
108
109   lcd.setCursor(10,0);
110   lcd.print("Paso");
111   lcd.setCursor(15,0);
112   lcd.print(pass);
113
114   lcd.setCursor(0,1);
115   lcd.print("Fin");
116   lcd.setCursor(6,1);
117   lcd.print("    ");
118   lcd.setCursor(6,1);
119   lcd.print(angleEnd);
120
121   lcd.setCursor(10,1);
122   lcd.print("Vel");
123   lcd.setCursor(13,1);
124   lcd.print("    ");
125   lcd.setCursor(13,1);
126   lcd.print(spd);
127
128   delay(del);
129 }
```

Figura 38 – Extracto de código ARDUINO líneas 98-129.

Fuente: elaboración propia.

7.1.2.4. *Comentarios adicionales al programa:*

Para finalizar se adjuntan algunos de los comentarios que están incluidos en el programa a modo de información adicional para entender y clarificar algunos de los aspectos surgidos durante el proceso de programación:

1. El número de barridos se ha de configurar por programa. En versiones mejoradas se intentara mediante panel de control con teclado o mando a distancia.
2. Se ha utilizado un motor paso paso modelo NMEA17. Cada paso del motor son 1.8º. Ver anexo especificación técnica.
3. Comunicación por el puerto serie al subprograma PROCESSING parar tras el primer barrido completo del Escáner.
4. Para que PROCESSING detecte el final de cada barrido definimos 500.0 como bandera marcadora.
5. Si se mueve el potenciómetro que define el ángulo final a alcanzar a una posición menor o igual que el ángulo actual del motor, se interrumpirá el

programa congelándolo hasta que el potenciómetro alcance un valor de, al menos, 1º por encima del valor del ángulo alcanzado por el motor en ese momento.

6. Información que PROCESSING leerá por el puerto serie y usara para la triangulación del haz laser.

7.2. PROCESSING – PROGRAMA CAPTURA Y PROCESADO DE IMÁGENES.

7.2.1. Objetivo del programa:

Programa desarrollado en ***Processing®***[10] debido a las prestaciones que ofrece en la comunicación por medio del puerto serie y por consiguiente con ARDUINO®.

EL programa realiza una captura de imagen de la webcam para cada posición angular del motor y el conjunto cámara y Laser.

La imagen obtenida se procesará pixel a pixel, procediendo en una etapa previa a la esqueletización de la línea laser a un solo pixel de grosor (que sería el pixel central y por lo tanto, el de mayor intensidad).

La siguiente etapa será la de almacenar en una cadena de datos dentro de un archivo ***.txt*** la siguiente secuencia (***ángulo, coordenada X, coordenada Y, coordenada Z***). Este archivo será leído e interpretado posteriormente con el 2º programa en Processing®.

7.2.2 Explicación detallada del programa:

7.2.2.1 *Definiciones iniciales del programa*:

Comenzamos importando las librerías en ***Processing*** necesarias con las subrutinas que ejecutaran los módulos dedicados a sacar rendimiento a la webcam y la conexión serie con nuestro ***ARDUINO*** vía USB. El carácter "*" define el final de la línea de importación. Se nota que en ***Processing***, como en muchos de los lenguajes

[10] Ver mas info en la web :https://processing.org/

de programación, la secuencia de comandos a interpretar se finalizan con ";".

```
34
35 import processing.video.*;
36 import processing.serial.*;
37
```

Figura 39 – Extracto de código PROCESSING_1 líneas 35-36.

Fuente: elaboración propia.

La librería Serie "***Serial***", lee y escribe datos a y desde elementos externos bit a bit al ordenador. El puerto serie tiene 9 pines I/O virtuales que se emulan a través del USB con una velocidad de transmisión de datos de 9600 baudios (valores que se configuran en el IDE del ARDUINO, ver capitulo anterior).

Definimos como entero de 10 caracteres la cadena de datos que se grabará a través del puerto serie, en este caso para definir el ángulo del motor.

```
38 Serial port;
39 int linefeed = 10;
```

Figura 40 – Extracto de código PROCESSING_1 líneas 38-39.

Fuente: elaboración propia.

A continuación se definen los siguientes objetos:

1- Objeto que permite la escritura de cadenas de caracteres.
2- Tipo de dato para almacenar y manipular capturas de video desde la webcam configurada.
3- Tipo de dato usado para almacenar imágenes con casi todas las extensiones de imagen (*.gif, *.jpg, *.png). Pueden renderizarse en 2D y 3D. Se ha de configurar previamente anchura y altura además de las unidades mínimas de imagen o pixeles. Se crean los datos ***Img1***, donde se almacena la imagen inicial capturada con el haz laser en pixeles rojos y los datos ***Img2***, donde se almacena la imagen procesada con el haz laser en pixeles blancos y ***"esqueletizados***".

```
41 PrintWriter output;
42
43 Capture video;
44
45 PImage img1;
46 PImage img2;
47
```

Figura 41 – Extracto de código PROCESSING_1 líneas 41-46.

Fuente: elaboración propia.

Inicializamos el programa con la función "***Void setup ()"*** y definimos la resolución en pixeles con la que se va a trabajar (resolución de la webcam utilizada).

La función video nos permite que cada nueva captura sea igual a la resolución definida, lo cual condiciona al tamaño de la imagen generada y renderizada posteriormente. Inicializamos la función de video.

Inicializamos el puerto serie (***COM4*** es el puerto virtual que el ***IDE*** de ***ARDUINO*** ha configurado y puede variar según equipo), la velocidad de transmisión de datos es de 9600 baudios. Inicializamos a cero para evitar cualquier dato.

Una vez leídos los datos desde los periféricos (ARDUINO + webcam), se procesaran y guardaran a través de una cadena de caracteres en el archivo de texto nombrado como "***test.txt***" (puede ser variable).

```
48 void setup(){
49
50   size(640,960);
51   video = new Capture(this, 640, 480);
52   video.start();
53   port = new Serial(this, "COM4", 9600);
54   port.clear();
55   output = createWriter("test.txt");
56
57 }
```

Figura 42 – Extracto de código PROCESSING_1 líneas 48-57.

Fuente: elaboración propia.

7.2.3 Desarrollo del programa:

Definimos secuencia de caracteres con el comando ***String***, esto lo que hace es examinar diferentes cadenas de caracteres, analizando y comparando cada uno. Estos se almacenan en ***myString*** y corresponden a los datos obtenidos del puerto

Serie con ***ARDUINO*** y que identifican la magnitud angular del motor para cada momento.

Inicializamos a 0 el valor ***motangle***, que es de tipo ***float***, es decir de coma flotante (decimal). Nos aseguramos que el valor inicial es 0 y no 0,00001

```
62
63 String myString;
64 float motangle = 0;
```

Figura 43 – Extracto de código PROCESSING_1 líneas 62-64.

Fuente: elaboración propia.

A continuación se llama a la función ***"draw",*** donde se ejecutan las líneas de código contenidas dentro del bloque hasta que el programa finaliza o se produce una interrupción. La función "***call"*** se llama por código y nunca se ejecuta de manera explícita. ***"Processing"*** siempre realiza el refresco de pantalla al acabar las rutinas contenidas en "***draw"*** y nunca antes. El contenido del código que define la función "***draw"*** se explica a continuación.

En primer lugar crearemos el objeto **Img1**, que como se ha definido antes es la imagen real que captura la webcam respetando los pixeles rojos del haz laser y discriminando el resto, convirtiéndolos a negros. La resolución será la misma que la de la webcam (640X480) y se deberá configurar en función de la cámara que utilicemos.

7.2.3.1 Definiciones previas:

Definimos el objeto **Img2**, que es la imagen donde volcaremos los datos procesados, es decir, pixeles convertidos a blanco y procesados tras el proceso de esqueletización, el fondo e mantendrá con pixeles negros. La definición de la resolución es igual a la de **Img1** y claramente, a la de la cámara web.

Los siguientes comandos sirven para proceder con la impresión del carácter **";"** para indicar una nueva captura, y por lo tanto una nueva serie de caracteres o datos.

La línea 73 del código, nos da el ángulo del motor en ese momento. Tener en cuenta que es un dato tipo ***"float"*** y será un decimal del tipo X.XXX almacenado en ***"motangle"***

```
65
66 void draw(){
67
68   img1 = createImage(640, 480, RGB);
69   img2 = createImage(640, 480, RGB);
70   background(0);
71
72   output.print(";");
73   output.print(motangle);
74
```

Figura 44 – Extracto de código PROCESSING_1 líneas 65-74.

Fuente: elaboración propia.

7.2.3.2 Lectura y escritura de datos:

7.2.3.2.1 Lectura Webcam.

Procedemos con la captura de imágenes en la webcam a través de un bucle "if", que permite al programa decidir que código ejecutar. Si tras el "if" cumple, se ejecutan las líneas. Si es falso no se ejecutan las líneas del programa.

En este caso, si hay disponibilidad en la webcam (capturas anteriores finalizadas) se procederá a leer desde la misma.

```
79
80
81   if(video.available()){
82     video.read();
83   }
84
```

Figura 45 – Extracto de código PROCESSING_1 líneas 79-84.

Fuente: elaboración propia.

7.2.3.2.2 Lectura Puerto Serie.

Para la lectura del puerto serie se procede de manera similar, si el puerto Serie **COM*** está disponible y puede enviar una cadena de caracteres tipo float de al menos 4 caracteres, entonces se procederá al volcado de caracteres en ***myString.***

```
86
87   if (port.available() >= 4){
88     myString = port.readStringUntil(linefeed);
89     if(myString !=null){
90       motangle = float(myString);
91     }
92   }
93
```

Figura 45 – Extracto de código PROCESSING_1 líneas 86-93.

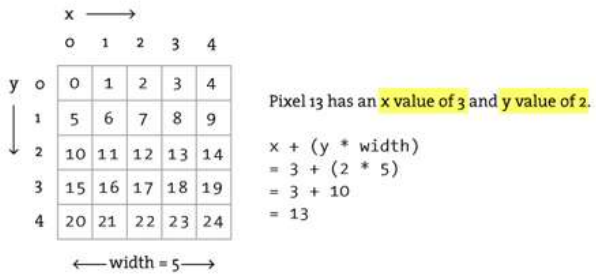
Fuente: elaboración propia.

7.2.3.3 Procesado de imagen capturada:

En esta parte del programa se irá estudiando secuencialmente pixel por pixel. A través de un bucle **"for",** usando un contador ***i,*** se realiza el salto de pixel a pixel, teniendo en cuenta la relación en la posición del mismo mediante la relación ***"(int i = 0; i < width*height/2; i++)***", que está definida en función de la resolución de la webcam utilizada.

Para identificar la posición de cada pixel se sigue la siguiente secuencia[11]:

1. Suponemos una ventana o imagen con una determinada WIDTH y HEIGHT.
2. Sabemos que la matriz de píxeles tiene un número total de elementos igual a WIDTH * HEIGHT.
3. Para cualquier punto X (contador i++), Y (contador k++) dado en la ventana, la ubicación en nuestra matriz de píxeles es: LOCATION = X + Y * WIDTH



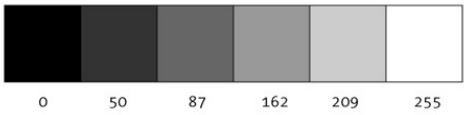
Las variables de sistema almacenan la altura y anchura de la ventana de visualización. Este valor se establece mediante el primer y segundo parámetro de la función ***size ().*** Por ejemplo, el tamaño de llamada de función (640, 480) establece la variable de altura en el valor 640 y la anchura en 480. El valor de altura o anchura por defecto es 100 si ***size ()*** no se utiliza en un programa.

Mientras esta relación de se cumpla, el contador añadirá una posición de tal modo que el salto de pixel a pixel se asegura. Cuando esto se cumple, el código añade una nueva línea tipo ***"if"*** que se cumplirá siempre que sea una condición verdadera. El código que se ejecutará si se cumple lo anterior consiste en una comparar y filtrar el pixel que se está analizando en la posición ***"i".*** El color del pixel se comparará con un color base (código 50 – tono oscuro), si el tono del color del pixel es más claro que

[11] https://processing.org/tutorials/pixels/

el establecido por el patrón, este se guardará como pixel de color blanco (código 255 – blanco) en el nuevo objeto Img1, donde se guardará la imagen procesada.

Los patrones de color que se manejan en ***"Processing"*** se codifican siguiendo el modelo siguiente[12]:



El bucle **if** se cierra cuando se han analizado todos los pixeles 640 x 480 = 307.200 pixeles con esta resolución.

```
94
95   for (int i = 0; i < width*height/2; i++){
96     if(red(video.pixels[i]) > 50){
97       img1.pixels[i] = color(255);
98     }
99   }
100
```

Figura 46 – Extracto de código PROCESSING_1 líneas 94-100.

Fuente: elaboración propia.

7.2.3.3.1 Proceso de esqueletización del haz laser.

El siguiente paso a realizar es el de transformar el perfil binario a uno de esqueleto binario, que tendrá la misma forma del perfil binario, pero con ancho de un pixel, exhibiendo el centro del perfil original, cuyos valores de distancia horizontal y vertical con respecto a un referencia dada serán utilizados para su posterior análisis.

Comenzaremos por la estructura de código explicada antes. Volvemos al contador de pixeles "i", que se inicializa a cero y va saltando de posición siguiendo el proceso ya mencionado.

A continuación inicializamos la variable entera k a cero. Este contador nos posicionara a través de la fila de pixeles blancos ya procesados.

Se ejecutará un ciclo while. Este ejecuta continuamente en forma de bucle cerrado el código contenido hasta que la condición inicial deje de cumplirse (los ciclos "for" solo ejecutan el código una vez solamente mientras que se cumpla la condición

[12] http://py.processing.org/tutorials/color/

inicial). Este bucle será ejecutado siempre y cuando se detecte un pixel blanco en Img1, no se salga del marco establecido por la sucesión de pixeles blancos y si corresponde a una secuencia máxima de 5 pixeles blancos. Se utiliza el comando ***&&*** para realizar esta triple comparación. Si cualquiera de los tres valores a categorizar no se cumple, se sale del bucle.

Primeramente se procede a analizar el valor del brillo del pixel con el que se está trabajando. El comando ***brightness*** extrae el valor del brillo del color blanco (255) del pixel guardado en Img1 con posición i. Se comprueba que es blanco (***brightness (img1.pixels[i]) == 255***). Se utiliza == porque compara cadenas de caracteres almacenadas en la misma ubicación de memoria.

A continuación nos aseguramos que no se ejecutara fuera del marco de la imagen (marco de 640 x480 pixeles). El contador siempre quedara un pixel por debajo de la anchura máxima de la imagen procesada **(i < (width*height/2)-1).**

Finalmente, se analiza el tercer grado comparativo, que exige cumplir este análisis en una concatenación máxima de 5 pixeles blancos **(i/width) %5 == 0).**

Si esto se cumple, los contadores **i** (contador de pixeles) y **k** (contador de pixeles blancos en una cadena máxima de 5 pixeles blancos) se incrementaran.

Para finalizar este proceso se realiza una comparación global del resto de los pixeles para detectar ruidos o falsos reflejos en forma de pixeles blancos aislados. Estos se eliminaran de la **img1**. Con las cadenas de pixeles blancos (máximo 5) se procederá a seleccionar el pixel intermedio, discriminando los adyacentes. Con esto termina el proceso de esqueletización devolviendo valores procesados y afinando la línea reflejada del haz laser a un solo pixel de grosor. Estos caracteres se procederán a guardar en el archivo de imagen procesada o **Img2**.

```
100
101
102  for (int i = 0; i < width*height/2; i++){
103    int k = 0;
104
105    while(brightness(img1.pixels[i]) == 255 && i < (width*height/2)-1 && (i/width)%5 == 0){
106
107
108
109
110      k++;
111      i++;
112    }
113
114    if(k > 0 && !(i-(k/2) == 195201)){
115      img2.pixels[i-(k/2)] = color(255);
116
```

Figura 47 – Extracto de código PROCESSING_1 líneas 101-115.

Fuente: elaboración propia.

A modo de resumen, el proceso grafico seguido ha sufrido la siguiente transformación:

1. ***La imagen A*** corresponde al pixelado inicial fruto del reflejo de la haz laser y captura realizada por la webcam. Esta imagen muestra diferentes tonos en el rojo de cada pixel, fruto del comportamiento del haz incidente sobre la superficie más externa del objeto debido al material y su comportamiento con la luz.
2. ***La imagen B*** muestra el primer proceso de sintetizado, donde se discriminan los colores rojos y negro del fondo para proceder a su conversión a pixeles blancos, manteniendo su posición inicial.
3. ***La imagen C*** muestra el trabajo realizado en el conteo de cadenas de caracteres de 5 pixeles y la ubicación del pixel central de cada una (proceso de esqueletización).
4. ***La imagen D*** es el resultado del proceso mencionado anteriormente.

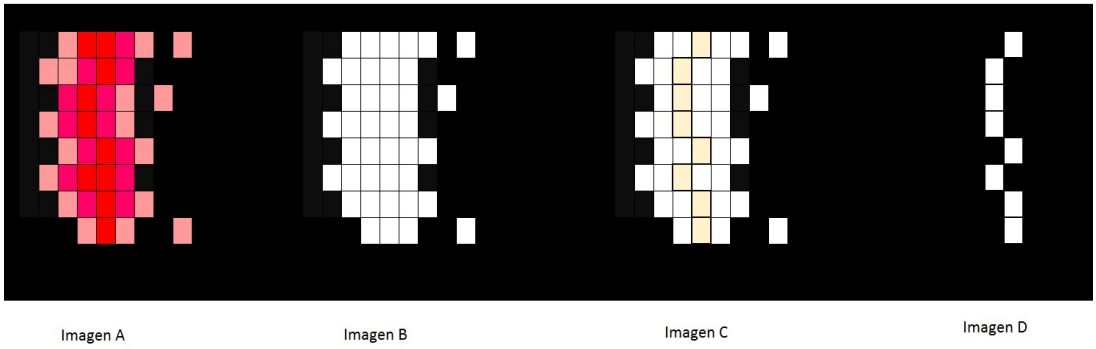


Figura 48 – Proceso de esqueletización del reflejo Láser

Fuente: elaboración propia.

7.2.3.4 Adquisición de datos y grabado del texto:

Tras el procesado de imágenes toca el turno de la gestión y archivo de datos generados. A continuación y en esta parte del programa en particular, se procederá a crear las rutinas de salida y almacenamiento de la información obtenida.

Se ha visto anteriormente que cada nueva secuencia de datos se almacenara en el archivo *.txt comenzando por los caracteres ";" (ver línea de código 72), a continuación se escribirá el ángulo del motor obtenido a través de la secuencias de

datos generada por el ARDUINO en el Puerto Serie (ver línea de código 73).

Esta fase del programa completa la información incluyendo el separador "," a continuación del ángulo de giro del motor anteriormente escrito.

Seguidamente procederemos a anotar la posición del pixel blanco obtenido tras el proceso anterior, la COLUMNA del nuevo píxel blanco (coordenada X) ira a continuación. Tras un separador adicional en forma de ",", se incluirá la FILA del nuevo píxel blanco (coordenada Y).

El proceso se repite para cada pixel blanco detectado.

```
119
120         output.print(",");
121         output.print(i%640);
122         output.print(",");
123         output.print(i/640);
124       }
125     }
126
```

Figura 49 – Extracto de código PROCESSING_1 líneas 119-126.

Fuente: elaboración propia.

7.2.3.5 Presentación en pantalla:

Este bloque del programa ejecuta los comandos relativos a la presentación que ofrece el mismo durante su ejecución. Para ofrecer un aspecto más interactivo, durante la ejecución se presentara por la pantalla del ordenador el siguiente aspecto:

1- Imagen real de la webcam a su resolución típica (para este caso la resolución de nuestra webcam)
2- Imagen virtual con la imagen ya procesada, y resaltando los pixeles blancos ya procesados.
3- En la parte inferior, se presentara la línea de comandos, donde se mostrara a tiempo real la velocidad de salida de los datos relativos al ángulo del motor paso paso que tiene en ese preciso instante. Se ha de mencionar que tanto la velocidad como el ángulo máximo alcanzado se visualizara siempre en función de las modificaciones realizadas durante su ejecución a través de los potenciómetros de la consola de control física.

La finalización del programa se ejecutara cuando el ángulo máximo programado del motor se alcance (En Arduino está configurado a 500 pasos o lo que es lo mismo

500 pasos por 1,18 el paso = 590 grados).

Cuando se alcanza este número, se detiene el proceso de escritura en el archivo ***.txt*** (**flush ()**) y a continuación se cierra dicho archivo ***(close ()).***

Finalmente con el comando ***exit (),*** terminamos la ejecución del código y salimos a la ventana de comandos.

```
128
129   image(video,0,0);
130   image(img2,0,480);
131
132   println(motangle);
133
134   if (motangle > 450.0){
135     output.flush();
136     output.close();
137     exit();
138   }
139 }
```

Figura 50 – Extracto de código PROCESSING_1 líneas 128-139.

Fuente: elaboración propia.

7.2.3.6 Interrupción forzada del programa:

Cabe mencionar que como opción de salida de emergencia usada para producir una interrupción del programa y parar su ejecución, se puede llamar en cualquier momento de la ejecución a la función **keyPressed (),** donde con la pulsación de una tecla hace que le proceso de escritura cese, cierre el archivo ***.txt** y salga del programa, antes de terminar de ejecutar el código completamente.

```
142
143 void keyPressed(){
144   output.flush();
145   output.close();
146   exit();
147 }
```

Figura 51 – Extracto de código PROCESSING_1 líneas 142-147.

Fuente: elaboración propia.

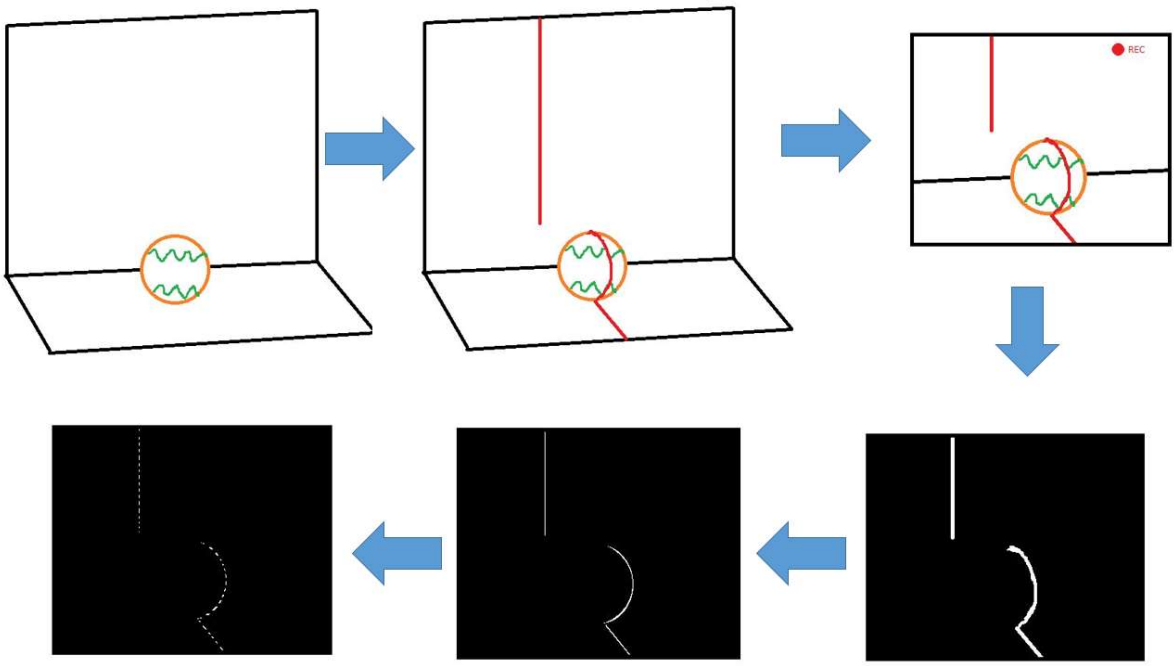


Figura 52 – Secuencia seguida para el escaneo por Processing.

Fuente: 3D-Environment-Laser-Scanner-From-Scratch.

7.3. PROCESSING – TRIANGULACION 3D

7.3.1. Objetivo del programa.

Una vez realizada la captura, procesado y almacenado de todos los pixeles relevantes obtenidos como válidos tras el proceso de escaneo se procederá a su implementación en el espacio virtual 3D. Este programa leerá la cadena de caracteres generados en la etapa anterior y almacenados en el archivo ***.txt**, para colocarlos en el espacio y representarles posteriormente.

Se virtualizará (ejes X, Y, Z – rojo, azul y verde) también el espacio 3D en el entorno virtual de Processing ®, donde se referenciaran dichos puntos.

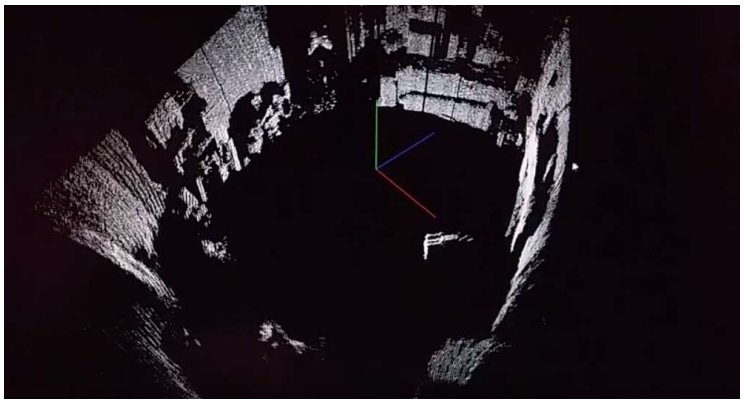


Figura 53 – Imagen Procesada tras escaneo.

Fuente: LASER escaner 3D.

7.3.2. Explicación detallada del programa:

7.3.2.1. Inicialización del programa

Inicializamos el programa con la función "***Void setup ()*** y definimos la resolución de la ventana de renderizado en 3D que crearemos (1000 x 1000). El comando ***P3D***, nos dice cómo usar el eje Z teórico para crear la ilusión de espacio tridimensional en la ventana de procesamiento[13].

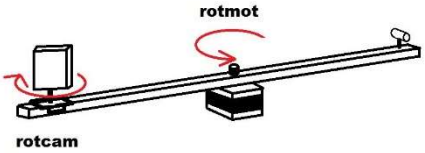
[13] https://processing.org/tutorials/p3d/

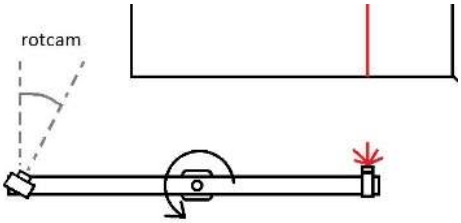
```
28 // CODIGO:
29 //
30 // 1_DEFINICIONES INICIALES DEL PROGRAMA
31 //   ---------------------------------
32
33 void setup(){
34   size(1000, 1000, P3D);
35 }
```

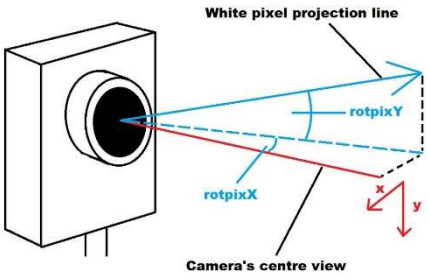
Figura 54 – Extracto de código PROCESSING_2 líneas 28-35.

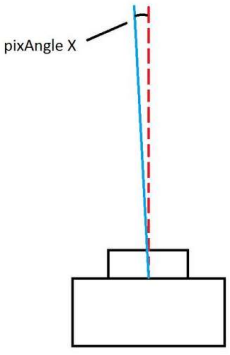
Fuente: elaboración propia.

Procedemos a declarar como **float**[14] los siguientes valores que se utilizaran en el programa:









- **"Rotmot"**: lugar donde guardará los valores del ángulo del motor (los tomara del archivo *.txt, generado en el programa anterior.
- **"Rotcam"**: almacenamiento del ángulo de la cámara, pero pasado a radianes.
- **"Rotlaser":** lugar donde almacenara el valor del ángulo del láser (es 0º ya que es perpendicular al brazo de giro).
- **"Rotpix X&Y"**: lugares donde se almacenarán los ángulos del píxel blanco actual desde la vista central de la cámara en la dirección X e Y.
- **"PixangleX&Y"**: son los lugares donde se almacenaran los ángulos entre el centro de la Web Cam y cada pixel blanco en la direccion X e Y.

[14] https://processing.org/reference/float.html

```
36
37 float rotmot;
38 float rotcam = 30*PI/180;
39 float rotlaser = 0;
40 float rotpixX;
41 float rotpixY;
42
43 float pixangleX = 50.0/640*PI/180;
44 float pixangleY = 50.0/640*PI/180;
```

Figura 55 – Extracto de código PROCESSING_2 líneas 36-44.

Fuente: elaboración propia.

Declaramos también como flotantes los siguientes valores que definirán:

- Parámetros para escribir la ecuación de una línea **[] A,B,AB;**
- Parámetros para escribir un plano **[] C,D,CD;**
- Constantes para definir la intersección plano-línea **n,t;**
- Coordenadas temporales para un punto en el espacio: **pointX&Y&Z;**

```
45
46 float[] A,B,AB;
47 float[] C,D,CD;
48 float n,t;
49 float pointX, pointY, pointZ;
```

Figura 56 – Extracto de código PROCESSING_2 líneas 45-49.

Fuente: elaboración propia.

Como valores flotantes también se declaran **[] drawpoint X&Y&Z,** lugares donde se almacenaran las coordenadas en el espacio (X, Y, Z) de todos los puntos generados.

```
50
51 float[] drawpointX;
52 float[] drawpointY;
53 float[] drawpointZ;
```

Figura 57 – Extracto de código PROCESSING_2 líneas 50-53.

Fuente: elaboración propia.

Finalizaremos inicializando **k = 0**, que actúa a modo de contador referenciando cada pixel blanco con el que se trabaja.

```
54
55 int k = 0;
56
```

Figura 58 – Extracto de código PROCESSING_2 líneas 54-56.

Fuente: elaboración propia.

7.3.2.2. Creación del espacio de Renderizado

Crearemos la ventana virtual de renderizado con la función **"void draw ()"** [15] y configuramos el fondo de la ventana render en color Negro (**background (0)**).

A continuación ajustamos la relación de perspectiva para ver toda la escena con la función ***"perspective ()"*[16]** , que establece una proyección de perspectiva, haciendo que los objetos distantes parezcan más pequeños que los más cercanos. Los parámetros definen un volumen de visualización con la forma de una pirámide truncada. Los objetos cercanos al frente del volumen aparecen a su tamaño real, mientras que otros objetos más lejanos parecen más pequeños. Esta proyección simula la perspectiva del entorno con mayor precisión que la proyección ortográfica.

```
58
59 // 2_CREACION DEL ESPACIO DE RENDERIZADO
60 //   ------------------------------------
61
62
63
64 void draw(){
65
66   background(0);
67   perspective(PI/3.0,(float)width/height,1,500);
```

Figura 59 – Extracto de código PROCESSING_2 líneas 58-67.

Fuente: elaboración propia.

Configuramos los ejes de referencia en el espacio, desde donde se proyectaran los puntos a representar. Diferenciaremos cada uno usando tres colores diferentes (rojo, azul y verde) con el comando **stroke.**

Las líneas de colores representaran un eje de coordenadas isométricas, cada línea tendrá una distancia de 50 pixeles en la orientación de cada coordenada.

[15] https://processing.org/reference/draw_.html
[16] https://processing.org/reference/perspective_.html

```
68  stroke(255, 0, 0);
69  line(0,0,0,50,0,0);
70  stroke(0, 255, 0);
71  line(0,0,0,0,50,0);
72  stroke(0, 0, 255);
73  line(0,0,0,0,0,50);
```

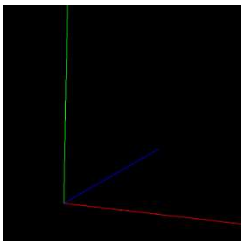


Figura 60 – Extracto de código PROCESSING_2 líneas 68-73 + ejes coordenados.

Fuente: elaboración propia.

Ahora vamos a definir el punto de vista de la imagen generada, que no será el mismo que desde donde se proyectan los puntos, en este caso el centro coordenado. Nos desplazaremos en distancia y en altura de tal modo que tengamos en la imagen el centro coordenado y los puntos proyectados desde el.

Establecemos la posición de la cámara **("camera ()")** mediante el ajuste de la posición del ojo, el centro de la escena y el eje que está orientado hacia arriba. Moviendo la posición del ojo y la dirección que está apuntando (el centro de la escena) permite que las imágenes sean vistas desde diferentes ángulos.

La variable del sistema **mouseX** siempre contiene la coordenada horizontal actual del ratón. Este proceso solo sólo será activo cuando el puntero del ratón este sobre la ventana de renderizado.

Conociendo la posición X, Y, Z del punto de vista del renderizado, definimos la posición X, Y, Z del punto de enfoque y rotar el espacio 3d donde se renderiza la nube de puntos en función del movimiento del puntero del ratón dentro de la ventana de renderizado.

```
74
75  camera(50*cos(mouseX/100.0)+10, 20, -50*sin(mouseX/100.0)-30,
76        10, 0, -30,
77        0.0, -1.0, 0.0);
78
```

Figura 61 – Extracto de código PROCESSING_2 líneas 74-78.

Fuente: elaboración propia

Finalmente, cuando se presione una tecla, se procederá a volcar la imagen renderizada, como se explicará a continuación.

```
78
79    if(keyPressed){
80
```

Figura 62 – Extracto de código PROCESSING_2 líneas 78-80.

Fuente: elaboración propia

7.3.2.3. Generación de la imagen Renderizada.

A continuación se describirá el proceso que se repetirá cíclicamente en bucle para cada punto a representar.

En primer lugar creamos los espacios virtuales de almacenamiento de las coordenadas de los puntos que vamos a representar en las coordenadas X, Y y Z. Serán cadenas de caracteres con parte decimal **(“drawpoint X&Y&Z”**). También el espacio definido para almacenar el ángulo del motor tiene este formato.

Este proceso está gestionado a través de un contado “**k**”, que será incrementado para cada lectura, procesado y representación de caracteres entre cada “**;**”(cada nueva captura).

```
81
82 // 3_GENERACION IMAGEN RENDERIZADA
83 //   ----------------------------
84
85
86
87     String[] myString = loadStrings("test.txt");
88     drawpointX = new float[splitTokens(myString[0], ",").length/2];
89     drawpointY = new float[splitTokens(myString[0], ",").length/2];
90     drawpointZ = new float[splitTokens(myString[0], ",").length/2];
91     myString = splitTokens(myString[0], ";");
92
93     k = 0;
94
```

Figura 63 – Extracto de código PROCESSING_2 líneas 81-94.

Fuente: elaboración propia

Repite el bucle para cada elemento en "myString" (número de capturas de la webcam registradas) y después divide cada "myString" en las coordenadas (X, Y) de cada pixel, separados por ",". Finalmente, guarda el ángulo del motor (primer dato) en el valor "Rotmot".

```
94
95      for(int i = 0; i < myString.length; i++){
96        String[] stringPart = splitTokens(myString[i], ",");
97        rotmot = float(stringPart[0]);
```

Figura 64 – Extracto de código PROCESSING_2 líneas 94-97.

Fuente: elaboración propia

Se crea una matriz con las coordenadas locales y se rota todo en el eje Y el ángulo del motor para cada captura.

```
98
99        pushMatrix();
100       rotateY(-rotmot*PI/180);
101
```

Figura 65 – Extracto de código PROCESSING_2 líneas 98-101.

Fuente: elaboración propia

Se crea una segunda matriz de coordenadas locales y se mueve todo a lo largo del brazo del escáner a la posición del láser en el espacio 3D. Finalmente rotamos todo en el eje Y por el ángulo del láser, que es 0º (recordar que es la posición del Laser física en el brazo giratorio y es perpendicular al mismo).

```
101
102       pushMatrix();
103       translate(-19, 2, 0);
104       rotateY(rotlaser);
```

Figura 66 – Extracto de código PROCESSING_2 líneas 101-104.

Fuente: elaboración propia

Esto que se ha realizado es simplemente una sucesión de traslaciones en el espacio de las coordenadas de los puntos X, Y a representar y por lo tanto de los propios puntos.

- ***1º paso***: representación plana de los puntos con coordenadas X, Y en (0, 0,0).
- ***2º paso:*** se rota en función del ángulo del motor en ese momento.
- ***3º paso:*** se traslada desde el centro del brazo hasta la posición del láser teniendo en cuenta la distancia del centro al mismo.

- ***4º paso***: Se rotan de nuevo las coordenadas en función del ángulo del láser sobre el brazo rotatorio.

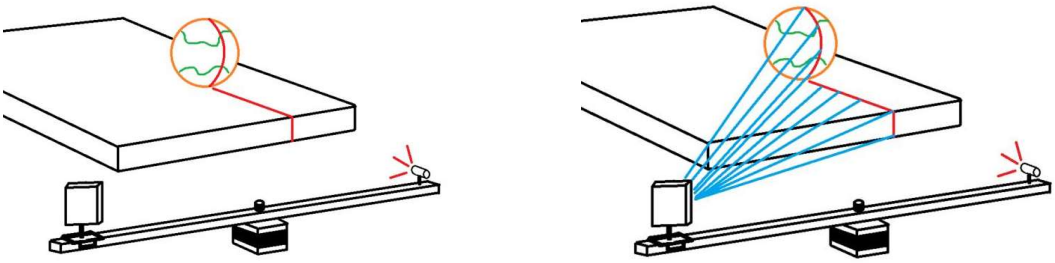


Figura 67 – Secuencia seguida en el programa.

Fuente: 3D-Environment-Laser-Scanner-From-Scratch

7.3.2.3.1. El Plano auxiliar.

A partir de aquí podemos crear la ecuación para el plano láser lineal. Cuando digo plano auxiliar del láser, estoy hablando de un plano donde el láser puede brillar en cualquier punto a lo largo del plano. Así que el plano pasa por el láser y por todo lo que brilla.

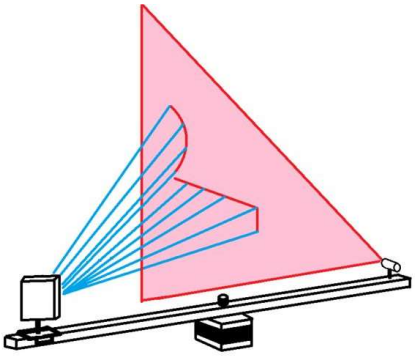


Figura 68 – Proceso a conseguir.

Fuente: 3D-Environment-Laser-Scanner-From-Scratch

Creamos un punto en el plano láser con coordenadas globales. PUNTO C. Trasladamos todo 10 unidades en la dirección X, se crea otro punto 10 unidades en dirección X fuera del plano LASER de referencia. PUNTO D, finalmente, Se crea una recta perpendicular al plano LASER usando los puntos C y D (NORMAL VECTOR), con esto queda definido nuestro plano con todos los puntos de una captura de imagen en particular.

```
105
106 //      ECUACION DEL PLANO
107 //      ------------------
108
109         C = new float[]{modelX(0,0,0), modelY(0,0,0), modelZ(0,0,0)};
110         translate(10, 0, 0);
111         D = new float[]{modelX(0,0,0), modelY(0,0,0), modelZ(0,0,0)};
112         CD = new float[]{D[0]-C[0], D[1]-C[1], D[2]-C[2]};
```

Figura 69 – Extracto de código PROCESSING_2 líneas 105-112.

Fuente: elaboración propia

A continuación, usamos popMatrix[17] para deshacer lo que hizo el pushMatrix. Solo lo usamos una vez para que volvamos al centro del motor pero todavía con su transformación de rotación.

En primer lugar cerramos la matriz con las coordenadas locales para devolver la posición actual al eje del motor con el ángulo del mismo. El bucle se repetirá para cada elemento de la cadena de caracteres **"stringPart"** (se repite para cada conjunto de coordenadas X-Y de los pixeles blancos en cada trama capturada). Tomamos la medida del ángulo de los pixeles blancos en la direccion X&Y desde el centro de la cámara **(rotpixX&Y).**

Así que ahora estamos trabajando con un solo píxel blanco que se convertirá en un único punto de datos. En este bucle primero necesitamos convertir las coordenadas XY blancas de píxeles en ángulos que actúan como los ángulos XY para la línea de proyección imaginaria desde la vista central de la cámara.

Ahora tenemos que usar **pushMatrix** de nuevo y trasladarlo a lo largo del eje del motor a lo largo del brazo al centro de la cámara. Luego giramos en Y por el ángulo de la cámara (entre 10º y 40º). Así que ahora nuestras coordenadas locales están en la cámara, apuntando hacia fuera a lo largo de la línea de proyección de un único pixel blanco y con dos magnitudes angulares medidas desde el centro de la cámara como coordenadas en vez de las anteriores X Y.

[17] https://processing.org/reference/popMatrix_.html

```
// CD[0]x + CD[1]y + CD[2]z = n

      popMatrix();

      for(int j = 1; j < stringPart.length; j = j+2){
        rotpixX = (float(stringPart[j])-320)*pixangleX;
        rotpixY = (240-float(stringPart[j+1]))*pixangleY;

        pushMatrix();
        translate(18, 4.5, 0);
        rotateY(rotcam);
        rotateX(rotpixY);
        rotateY(rotpixX);
```

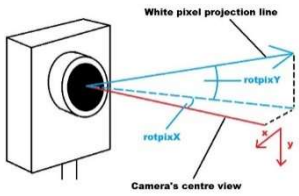


Figura 70 – Extracto de código PROCESSING_2 líneas 113-126.

Fuente: elaboración propia.

7.3.2.3.2. Línea auxiliar.

Ahora hacemos la línea de proyección de la misma manera que hicimos el vector normal para el plano.

Creamos un punto en el centro de la cámara A (0,0,0) y otro punto a alguna distancia a lo largo de la línea de proyección B, pero movido 10 unidades normales al centro de la cámara. Definimos la ecuación de la recta desde este punto A y el vector AB.

Aquí volvemos a pulsar la "**popMatrix ()**" para volver al eje del motor y deshacer la transformación anterior.

```
//      ECUACION DE LA LINEA
//      --------------------

      A = new float[]{modelX(0,0,0), modelY(0,0,0), modelZ(0,0,0)};
      translate(0,0,10);
      B = new float[]{modelX(0,0,0), modelY(0,0,0), modelZ(0,0,0)};
      AB = new float[]{B[0]-A[0], B[1]-A[1], B[2]-A[2]};

      popMatrix();
```

Figura 71 – Extracto de código PROCESSING_2 líneas 127-136.

Fuente: elaboración propia.

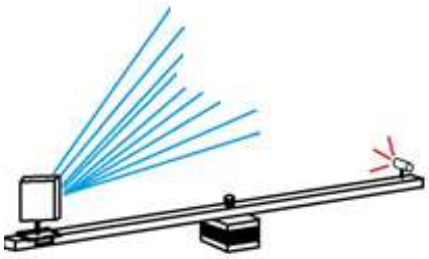
7.3.2.3.3. Intersección de la línea y el plano.

Ahora procederemos a recordar las clases del algebra de BUP, ya que tenemos un problema clásico de intersección de una recta con un plano.

La intersección del plano LASER con la ecuación de la recta insertada en ella se resolverá en un sistema de ecuaciones donde la variable independiente es "t", y determinar a partir de ella las coordenadas X, Y y Z.

Ahora tenemos nuestras coordenadas XYZ para un solo punto de datos y lo almacenamos en **“drawpoint”.**

Usaremos de nuevo “**popMatrix ()”** de nuevo para volver al inicio del sistema de coordenadas global sin transformaciones de rotación. Así que ahora tenemos 3 vectores enormes para los componentes X, Y y Z de cada punto de datos, pero en realidad no hemos dibujado ningún punto para la nube de puntos.



```
137
138 //      INTERSECCION DE LINEA Y PLANO
139 //      ----------------------------
140
141         t = (n - CD[0]*A[0] - CD[1]*A[1] - CD[2]*A[2])
142         /   (CD[0]*AB[0] + CD[1]*AB[1] + CD[2]*AB[2]);
143         pointX = A[0] + AB[0]*t;
144         pointY = A[1] + AB[1]*t;
145         pointZ = A[2] + AB[2]*t;
146
147         drawpointX[k] = (pointX);
148         drawpointY[k] = (pointY);
149         drawpointZ[k] = (pointZ);
150         k++;
151       }
152       popMatrix();
153     }
154   }
```

Figura 72 – Extracto de código PROCESSING_2 líneas 137-154.

Fuente: elaboración propia.

Así podemos hacer eso ahora. Utilizamos un bucle “for” para pasar por cada punto de datos, lo que lo hace siempre que **'drawpoint'**, y en el bucle, use **'point (drawpoint X [i], drawpointY [i], drawpointZ [i])**. Esto representa un punto en cada intersección entre la línea de proyección de un pixel blanco y el plano del láser.

```
155
156   if(k > 0){
157     for (int i = 0; i < drawpointX.length; i++){
158       stroke(255);
159       point(drawpointX[i], drawpointY[i], drawpointZ[i]);
160     }
161   }
162 }
```

Figura 73 – Extracto de código PROCESSING_2 líneas 155-162.

Fuente: elaboración propia.

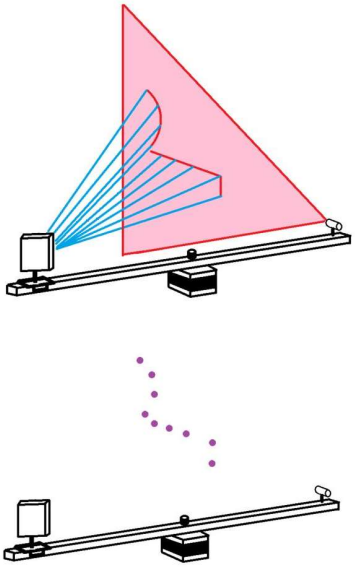


Figura 74 – Resumen visual del proceso seguido por PROCESSING_2.

Fuente: 3D-Environment-Laser-Scanner-From-Scratch.

8. INTEGRACION NUBE DE PUNTOS

Aunque esta parte está fuera del alcance de este trabajo, se procederá a comentar brevemente los pasos seguidos y los objetivos que se pretenden conseguir a la hora de trabajar con la nube de puntos y sus coordenadas que se han obtenido, fruto del proceso de adquisición de datos explicado anteriormente.

Siguiendo la política de trabajar con sistemas de código abierto, el tratamiento final de los datos escaneados se realizara en la mima línea y las herramientas a utilizar serán también de código abierto de acceso gratuito para todos los usuarios. Para ello necesitamos buscar y descargar el siguiente software:

- **MESHLAB** (http://www.meshlab.net/): Este software permite procesar y editar mallas de triángulos 3D. Meshlab es open source, con licencia GPL, portable, extensible, implementado para diversos sistemas operativos, escrito en C/C++, y desarrollado por el ISTI – CNR. Las extensiones de

Meshlab son basadas en plugins; pero es posible modificar el código fuente. Esta herramienta se basa en la librería VCG desarrollada por el Visual Computing Lab del ISTI – CNR.

Es ampliamente utilizada bajo el contexto de investigación, académico y desarrollo, en diversas áreas de las ciencias como la ingeniería (en todas las especialidades donde existan las mallas triangulares), microbiología, arte, paleontología, ortopedia, ortodoncia, entre otros. Básicamente, todas aquellas ramas donde la reconstrucción de mallas para el despliegue de modelos 3D sea parte de un proceso de desarrollo.

Meshlab permite el manejo de mallas de gran tamaño no-estructuradas y permite un conjunto de funcionalidades para su edición, optimización, inspección, rendering y conversión a otros formatos de mallas triangulares. Igualmente, es posible remover datos duplicados, así como vértices, caras y aristas no referenciadas en los datos de entrada, aplicar *remeshing*, filtros de suavizados, remoción de ruido, *registración* de mallas (basadas en ICP), *colorization/inspection*, *painting*, medidas sobre las estructuras triangulares, entre muchas otras funcionalidades.

- **BLENDER** (https://www.blender.org/download/): Blender es un programa informático multiplataforma, dedicado especialmente al modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales. También de composición digital utilizando la técnica procesal de nodos, edición de vídeo, escultura (incluye topología dinámica) y pintura digital. En Blender, además, se puede desarrollar vídeo juegos ya que posee un motor de juegos interno.

El resultado final a conseguir en este proyecto es una nube de puntos dispersa, o mejor todavía una nube de puntos densa, ya que esta contiene la información de la textura de nuestra escena escaneada.

Las nubes de puntos nos ofrecen una importante información dimensional y geométrica. Estas nubes de puntos las podemos retocar y analizar con herramientas como Meshlab, pero a priori, no tienen una especial valided para su procesado en herramientas de modelado 3D tipo Blender, para continuar con herramientas Open Source.

Blender, como cualquier otra herramienta de modelado 3D trabaja con objetos construidos por mayas, formadas a partir de la información de las nubes de puntos. Esta transformación de nube a maya es bastante simple de realizar y nos permitirá realizar tareas de virtualización como iluminaciones, movimientos de cámara etc, sobre nuestra escena.

8.1. MALLA DE POLIGONOS. Uso de Meshlab ®

En primer lugar, procederemos a guardar las nuevas coordenadas de cada punto escaneado. Recordemos que la adquisición de datos inicial se almaceno en un archivo del tipo *.txt, que tras su procesado para visualización en pantalla sufrió varias transformaciones. Para ello se debe completar el código en PROCESSING® que permita guardar las coordenadas x, y, z de cada punto en un nuevo archivo de texto, por ejemplo, “nube de puntos.txt”.

Meshlab permite la posibilidad de cargar una lista de datos (coordenadas de cada punto) tomándolas directamente de un archivo externo tipo txt, para ello habrá que configurar y programar un Script.

Un script es una secuencia de código escrita en lenguaje de alto nivel que realiza una subrutina determinada cuando se integra en el código del programa matriz.

Este script ha de conseguir en:

- Apertura del archivo *.txt.
- Inicializar puntero de datos en el primer carácter.
- Transmisión de información al programa matriz (Meshlab)
- Lectura ordenada y en bucle de cada cadena de caracteres.
- Tras el último punto, terminar y cerrar el programa.

Primero se calculan las normales, las cuales de forma simplificada son las que van a indicar la orientación de nuestra superficie, o lo que es lo mismo, determinar cuál es el derecho o el revés de nuestro modelo. Para ello vamos al menu:

Filters > Point Set > Compute normals for point sets

y en la ventana emergente dejamos el valor:

Number of neigbors 10

El siguiente paso es crear la maya con:

Filters > Point Set > Surface Reconstruction: Poisson

Y los valores

Octree Depth 10
Solver Divide 8
Samples per node 1
Surface offsetting 1

Una vez creada la maya si activamos la visión de sólido o maya (smoot o flat) en Meshlab nos encontraremos muchas veces con formas extrañas que para nada se corresponden a nuestro modelo. Esto es debido a cómo se computa la maya, que muchas veces posee una cierta tendencia a cerrar el modelo. Si esto nos sucede, simplemente con las herramientas de eliminar “faces” podemos ir borrando las partes de la maya que no nos son interesantes.

En este punto si revisamos las capas de nuestro proyecto nos encontramos con nuestra capa inicial o .ply y la maya.

El último paso es aplicar nuestra textura a la maya, de forma que tengamos un sólido con una textura. Para ello vamos al siguiente menú:

Filters > Sampling > Vertex Atribute Transfer

Y aplicamos los siguientes valores:

Source Mesh xxx.ply

Target Mesh Poisson mesh

Transfer Color ok

De forma que la capa de origen sea nuestra nube de puntos densa o .ply y la de destino nuestra malla de Poisson.

Con estos sencillos pasos ya podemos guardar nuestro modelo como .ply para luego importarlo dentro de Blender para comenzar los trabajos de virtualización.

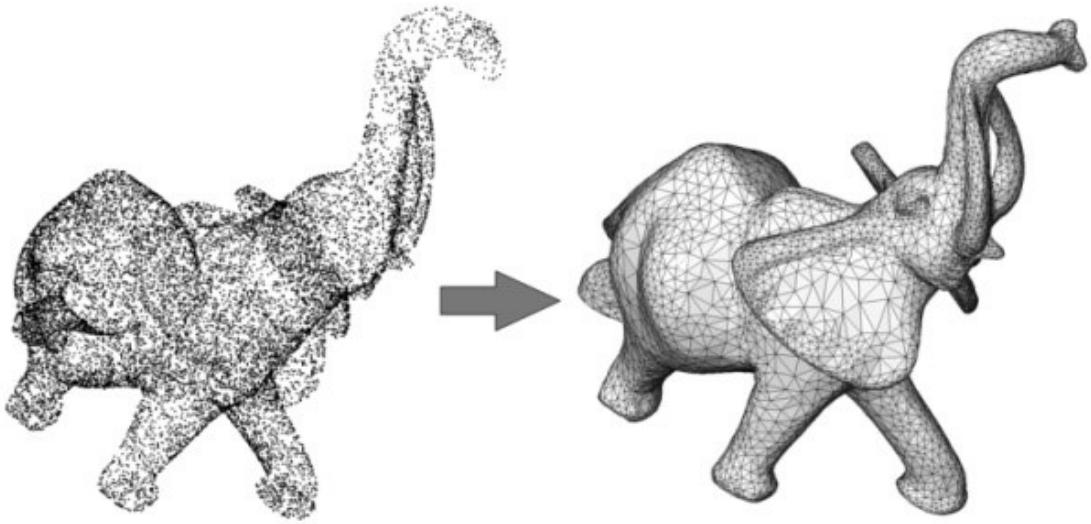


Figura 75 – Proceso de transformación de nube de puntos a malla poligonal.

Fuente: https://hangar.org/es/news/dijous-oberts-reconstruccio-de-malles-descanejat-3d-meshlab-blender/

Como conclusión se ha de considerar que una vez tengamos la nube de puntos en el espacio del entorno del programa, calcularemos las mallas por triangulación

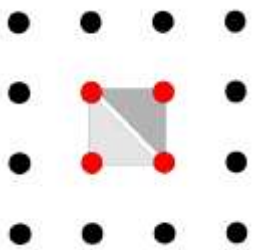


Figura 76 – Proceso de mallado de la nube de puntos

Fuente: https://hangar.org/es/news/dijous-oberts-reconstruccio-de-malles-descanejat-3d-meshlab-blender/

La **representación** del objeto con diferentes triángulos que forman una malla añade o elimina resolución.

La necesidad de puntos se reduce si la **superficie** es **uniforme**, si la superficie presenta **rugosidades** necesita mayor cantidad representación por triángulos. Esta información se guarda en **arrays** con información de cada punto.

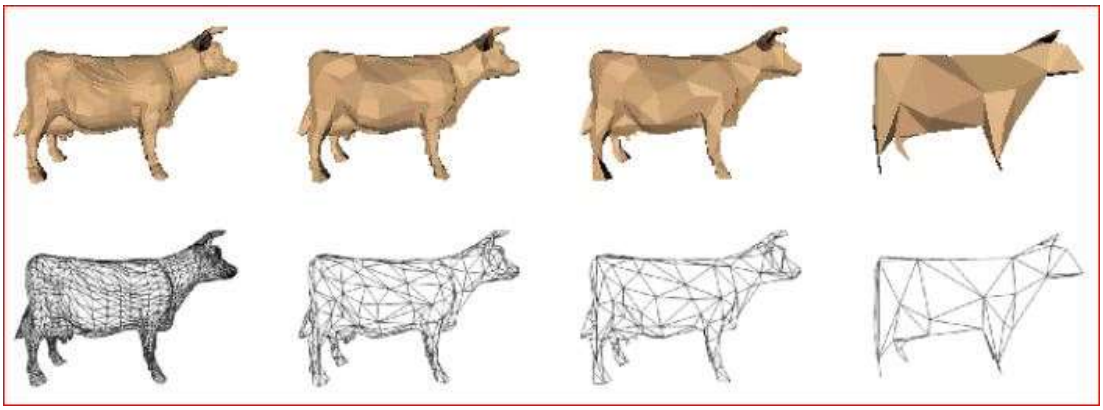


Figura 77 – Proceso de mallado de la nube de puntos y texturizado.

Fuente: https://hangar.org/es/news/dijous-oberts-reconstruccio-de-malles-descanejat-3d-meshlab-blender/

8.2. BLENDER:

Estaríamos en la fase final del digitalizado, en este punto se pretenderá:

Generar un mapa de textura es un método para agregar detalles a las superficies, proyectando imágenes y patrones sobre esas superficies. Las imágenes y patrones proyectados pueden ser configurados para afectar no solo el color, si no también la especularidad, la reflexión, la transparencia e incluso un falso relieve tridimensional. Es más común que las imágenes y patrones sean proyectados durante el procesamiento, pero los mapas de texturas son utilizados también para esculpir, pintar y deformar objetos.

En Blender, las Texturas pueden aplicarse a un Material, a un Pincel y al (Entorno). Además, las texturas también pueden usarse mediante varios Modificadores.

La configuración de materiales que hemos visto hasta ahora produce objetos lisos, *uniformes*, pero tales objetos no se ajustan particularmente a la realidad, donde

la uniformidad es poco común y está fuera de lugar. Para lidiar con esta uniformidad poco realista, Blender permite al usuario aplicar *texturas* con las que puede modificar la reflectividad, especularidad, rugosidad y otras cualidades de la superficie de cada material escaneado.

9. CONCLUSIONES

Como conclusiones finales con respecto a este proyecto he de destacar personalmente que he disfrutado en gran medida durante cada una de las etapas que ha conllevado la elaboración del mismo.

Sin embargo se puntualizan a continuación diferentes puntos donde se comentan las reflexiones realizadas tras la consecución de cada uno:

1. **Satisfacción en encontrar solución al problema planteado**. Tras los estímulos iniciales y valoración del alcance del proyecto para la problemática planteada, se disgregó el problema inicial en pequeñas fases a completar, lo cual hizo que el trabajo se simplificara y la interacción final de cada una (definición, diseño y construcción del hardware y definición, diseño e implementación del software) fuera más sencilla.
2. **Mayor conocimiento de la técnica de escaneo por triangulación.** Aunque no supone novedad este modo de adquisición de datos, el desarrollo de este proyecto y la consulta exhaustiva a la documentación disponible ha servido para tener en cuenta al máximo detalle el funcionamiento y teoría de este método. Esto ha llevado a comenzar el planteamiento de varias mejoras que se implementaran al sistema desarrollado.
3. **Involucración en herramientas de programación y diseño gráfico**. El desarrollo de las técnicas de control del motor y uso del elemento óptico de captura ha supuesto la iniciación en lenguajes de programación no conocidos. Ha supuesto el uso del lenguaje del IDE de ARDUINO (ya conocido y muy usado por mi parte) y el desarrollo de técnicas en PROCESSING (inmersión total dado a mi nulo conocimiento, pero con ideas previas debido al uso anteriormente de

C++ y lenguaje en BASIC).

4. **Demostración de encontrar una solución a un problema complejo de bajo coste**. Era la una de las premisas iniciales del proyecto y se ha conseguido realizar. Las modificaciones posteriores no aumentarían el presupuesto en términos brutos elevados.

10. MEJORAS FUTURAS:

La conclusión de la fase de diseño y pruebas de las diferentes etapas del proyecto se dio como válida el día 29 de enero de 2017. Con los datos generados y experiencia adquirida se ha confeccionado esta memoria y el resto de documentación anexa incluida.

No obstante, ya desde fechas anteriores a la conclusión del mismo, se ha realizado acopio de ideas de mejora surgidas a posteriori que serán incorporadas al proyecto en versiones posteriores que lo actualizaran y mejoraran en versatilidad.

Las siguientes mejoras que se implementaran se podrían enumerar a continuación:

1. **MEJORAS HARDWARE:**
    a. Renovación de la estructura en aluminio.
    b. Incorporación de soportes regulables en altura.
    c. Añadir nivel de burbuja para determinar estado de equilibrio. Se regularían desequilibrios con las patas mencionadas anteriormente.
    d. Leds indicadores de encendido y apagado.
    e. Opción de funcionamiento a baterías. Para asegurar la portabilidad.
    f. Implementación de un Shield Arduino con tarjeta SD de gran capacidad para el almacenamiento de la información de datos adquirida.
    g. Posibilidad de activación y manejo con control remoto

mediante mando a distancia.

h. Mejora en resolución de la webcam.
i. Mejora del sistema óptico LASER.
j. Implementación de pantalla de cristal líquido de 64 caracteres que permita la visualización de más datos.

2. MEJORAS SOFTWARE:

a. IDE ARDUINO: mejora de la programación añadiendo más información de operatividad a través del LCD.
b. IDE ARDUINO: implementación lectura y recepción de datos por infrarrojo para la adaptación de un mando a distancia de control remoto.
c. IDE ARDUINO: sensorización mediante nivel de mercurio que permita determinar grado de desnivel de la máquina.
d. IDE ARDUINO: sensorización del nivel de luminosidad e información previa por pantalla de calidad esperada en el escaneo.
e. PROCESSING: almacenamiento final de la información a cada pixel ya transformada en coordenadas X, Y y Z.
f. Entorno de utilización amigo mediante programa en JAVA a base de ventanas e hipervínculos.

En el tiempo transcurrido entre transcribir las ideas anotadas y la lectura del trabajo, seguramente esta lista habrá sido ampliada notablemente.

---ooo000ooo---

Santander, 19 Febrero 2017

11. ANEXOS

11.1. CRONOGRAMA_SECUENCIA TEMPORAL:

Nombre de tarea
1 - IDENTIFICACION DE OBJETIVOS
1 - A.1 - Tarea 1: Definición del problema
- Recopilación de Información
1 - A.2 - Tarea 2: Posibles soluciones
- viabilidad LASER luz estructurada
- viabilidad LASER Triangulación
2 - ELABORACION HARDWARE
2- A-CIRCUITERIA ELECTRONICA DE CONTROL
2 - A.1 - Diseño circuito en papel
2 - A.2 - Diseño circuito cad y simulación
2 - A.3 - Acopio materiales
2 - A.4 - Diseño circuito en protoboard
2 - A.5 - Integración componentes
2 - A.6 - Prueba física del circuito
2 - B-PCB INTERFASE
2 - B.1 - Diseño circuito en papel
2 - B.2 - Diseño circuito cad y simulación
2 - B.3 - Acopio materiales
2 - B.4 - Elaboración de la PCB
2 - B.5 - Soldadura componentes
2 - B.6 - Prueba física del circuito
2 - C.3-CARCASA Y ESTRUCTURA EXTERNA
2 - C.1 - Acopio materiales
2 - C.2 - Dimensionado y medidas
2 - C.3 - Corte y preparación
2 - C.4 - Ensamblaje
3 - ELABORACION SOFTWARE
3 - A-ARDUINO CONTROL MOTOR PP
3 - A.1 - Flujograma
3 - A.2 - Programación IDE ARDUINO
3 - A.3 - Grabación y prueba en Micro controlador
3 - B-PROCESING CAPTURA DE DATOS
3 - B.1 - Flujograma
3 - B.2 - Programación PROCESING
3 - B.3 - Prueba en PC
3 - C-PROCESING PROCESADO DE DATOS
3 - C.1 - Flujograma
3 - C.2 - Programación PROCESING
3 - C.3 - Prueba en PC
4 - MEMORIA TECNICA DEL PROYECTO
4 - A - MEMORIA DESCRIPTIVA
4 - B - ANEXOS
4 - B.1 - Cronograma
4 - B.2 - Diagramas Técnicos
4 - B.3 - Descripciones adicionales
4 - B.4 - Presupuesto Final
4 - C - REVISON
4 - C.1 - 1º revisión
4 - C.2 - 2º revisión

11.1.1. CRONOGRAMA_DIAGRAMA DE GANT:

11.1.1.1. Ver anexo A3.

11.2. HARDWARE

11.2.1. Características Arduino

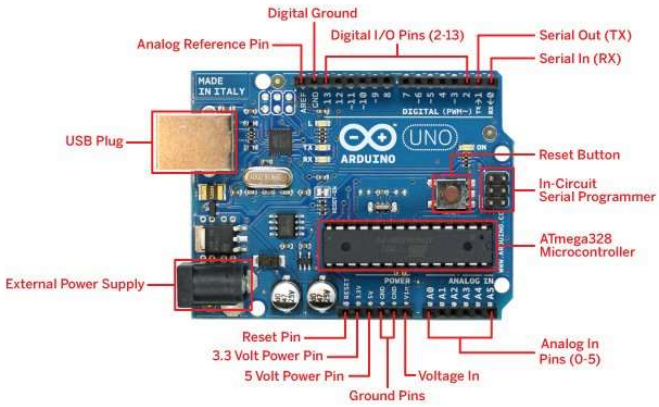
El Arduino Uno R3 utiliza el microcontrolador ATmega328. En adición a todas las características de las tarjetas anteriores, el Arduino Uno utiliza el ATmega16U2 para el manejo de USB en lugar del 8U2 (o del FTDI encontrado en generaciones previas). Esto permite ratios de transferencia más rápidos y más memoria. No se necesitan drivers para Linux o Mac (el archivo inf para Windows es necesario y está incluido en el IDE de Arduino).

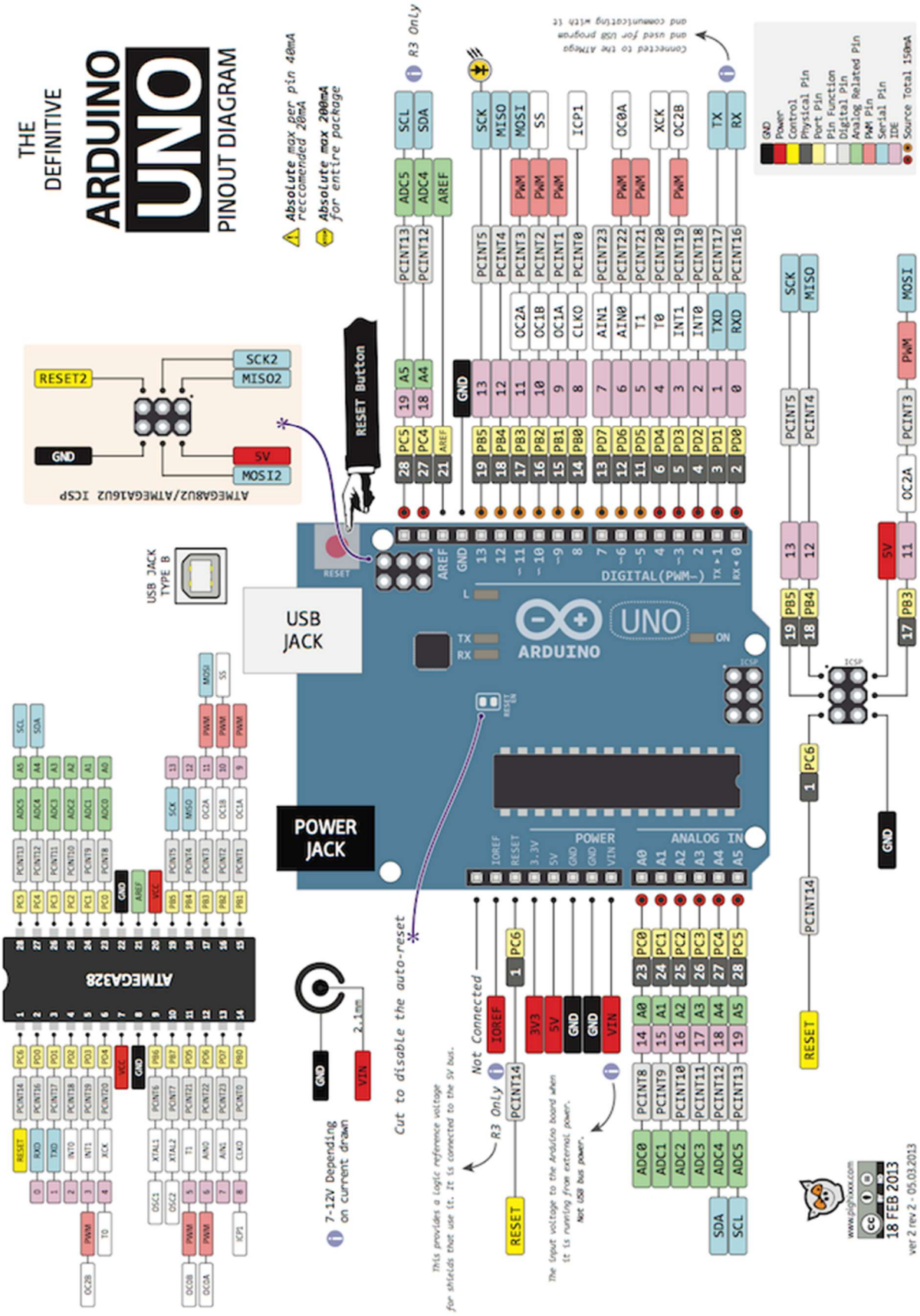
La tarjeta Arduino Uno R3 incluso añade pins SDA y SCL cercanos al AREF. Es más, hay dos nuevos pines cerca del pin RESET. Uno es el IOREF, que permite a los shields adaptarse al voltaje brindado por la tarjeta. El otro pin no se encuentra conectado y está reservado para propósitos futuros. La tarjeta trabaja con todos los shields existentes y podrá adaptarse con los nuevos shields utilizando esos pines adicionales.

El Arduino es una plataforma computacional física open-source basada en una simple tarjeta de I/O y un entorno de desarrollo que implementa el lenguaje Processing/Wiring. El Arduino Uno R3 puede ser utilizado para desarrollar objetos interactivos o puede ser conectado a software de tu computadora (por ejemplo, Flash, Processing, MaxMSP). El IDE open-source puede ser descargado gratuitamente (actualmente para Mac OS X, Windows y Linux).

Características:

- Microcontrolador ATmega328.
- Voltaje de entrada 7-12V.
- 14 pines digitales de I/O (6 salidas PWM).
- 6 entradas análogas.
- 32k de memoria Flash.
- Reloj de 16MHz de velocidad.





Fuente: *http://marcusjenkins.com/arduino-pinout-diagrams/*

11.2.2. Fuente de alimentación 12V

Transformador de AC a DC universal que nos facilita una salida de 12V DC a 6A. Capaz de aguantar picos de 75W. Uso indicado para cargas permanentes de hasta 60W.

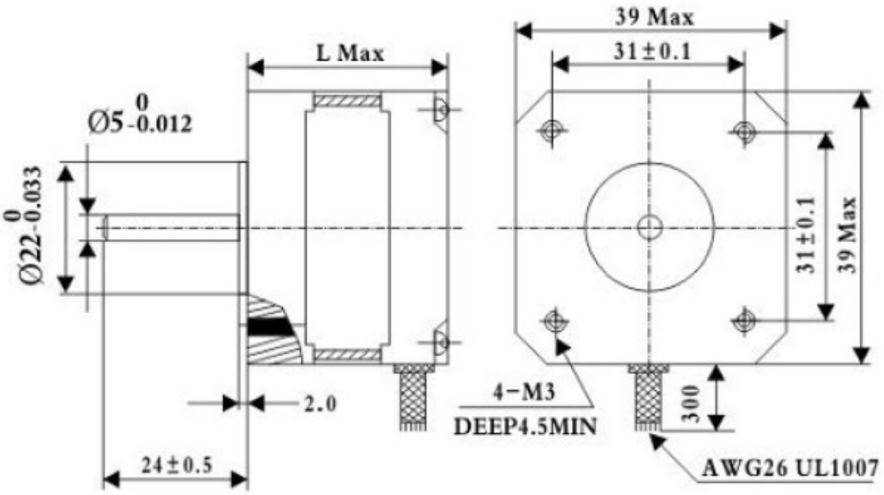
- Potencia: 12V DC - 6A - 75W
- Tensión: 220-240V AC
- Multitensión: 85-265V AC
- Dimensiones: 45x100x160 mm
- Material: Aluminio
- Peso: 309 g
- Certificados: CE & RoHS

Fuente: *https://www.amazon.es/CroLED-Fuente-Alimentaci%C3%B3n-Alimentador-Transformador/dp/B00FFS1Z56/ref=sr_1_1?ie=UTF8&qid=1497905518&sr=8-1&keywords=fuente+alimentaci%C3%B3n+12+v*

11.2.3. Motor NEMA 17

39HS34046:

Model	Step Angle	Rated Current	Phase Resistanc	Phase Inductanc	Holding Torque	Detent Torque	Rotor Inertia	Motor Length	Lead Wire
	(°)	(A)	(Ω)	(MH)	(N.cm)	(N.cm)	(g.cm	(mm)	(No.)
39HS20044	1.8	0.42	18	12	8	0.5	12	20	4
39HS26064	1.8	0.6	9	10	14	0.8	14	26	4
39HS34064	1.8	0.6	12	13	18	1	19	34	4
39HS34124	1.8	1.2	3.2	3	16	1	19	34	4
39HS34046	1.8	0.4	30	14	12	1	19	34	6
39HS40064	1.8	0.6	12	20	24	1.2	24	40	4
39HS40124	1.8	1.2	3.8	6.5	24	1.2	24	40	4
39HS40046	1.8	0.4	30	22	18	1.2	24	40	6



Fuente: http://tienda.bricogeek.com/motores-paso-a-paso/546-motor-paso-a-paso-nema-17-32kg-cm.html

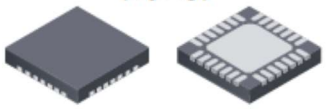
11.2.4. Controlador A4988

Features and Benefits

- Low $R_{DS(ON)}$ outputs
- Automatic current decay mode detection/selection
- Mixed and Slow current decay modes
- Synchronous rectification for low power dissipation
- Internal UVLO
- Crossover-current protection
- 3.3 and 5 V compatible logic supply
- Thermal shutdown circuitry
- Short-to-ground protection
- Shorted load protection
- Five selectable step modes: full, $^1/_2$, $^1/_4$, $^1/_8$, and $^1/_{16}$

Package:

28-contact QFN
with exposed thermal pad
5 mm × 5 mm × 0.90 mm
(ET package)



Approximate size

Description

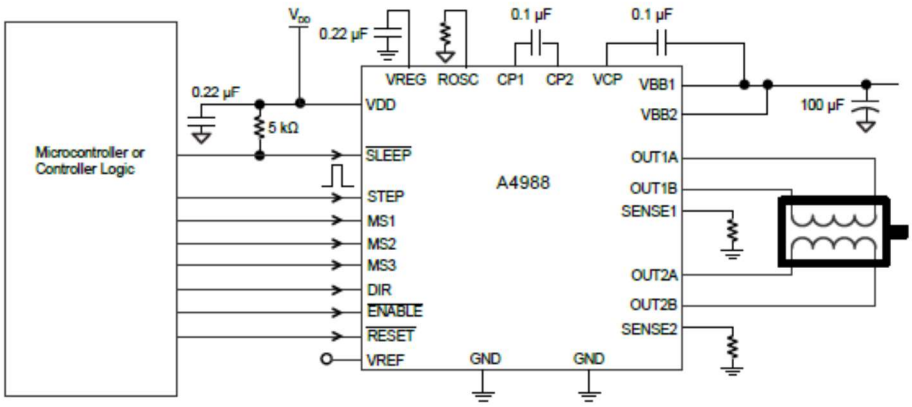
The A4988 is a complete microstepping motor driver with built-in translator for easy operation. It is designed to operate bipolar stepper motors in full-, half-, quarter-, eighth-, and sixteenth-step modes, with an output drive capacity of up to 35 V and ±2 A. The A4988 includes a fixed off-time current regulator which has the ability to operate in Slow or Mixed decay modes.

The translator is the key to the easy implementation of the A4988. Simply inputting one pulse on the STEP input drives the motor one microstep. There are no phase sequence tables, high frequency control lines, or complex interfaces to program. The A4988 interface is an ideal fit for applications where a complex microprocessor is unavailable or is overburdened.

During stepping operation, the chopping control in the A4988 automatically selects the current decay mode, Slow or Mixed. In Mixed decay mode, the device is set initially to a fast decay for a proportion of the fixed off-time, then to a slow decay for the remainder of the off-time. Mixed decay current control results in reduced audible motor noise, increased step accuracy, and reduced power dissipation.

Continued on the next page...

Typical Application Diagram



Time Duration	Symbol	Typ.	Unit
STEP minimum, HIGH pulse width	t_A	1	µs
STEP minimum, LOW pulse width	t_B	1	µs
Setup time, input change to STEP	t_C	200	ns
Hold time, input change to STEP	t_D	200	ns

Figure 1: Logic Interface Timing Diagram

Table 1: Microstepping Resolution Truth Table

MS1	MS2	MS3	Microstep Resolution	Excitation Mode
L	L	L	Full Step	2 Phase
H	L	L	Half Step	1-2 Phase
L	H	L	Quarter Step	W1-2 Phase
H	H	L	Eighth Step	2W1-2 Phase
H	H	H	Sixteenth Step	4W1-2 Phase

ELECTRICAL CHARACTERISTICS[1] at T_A = 25°C, V_{BB} = 35 V (unless otherwise noted)

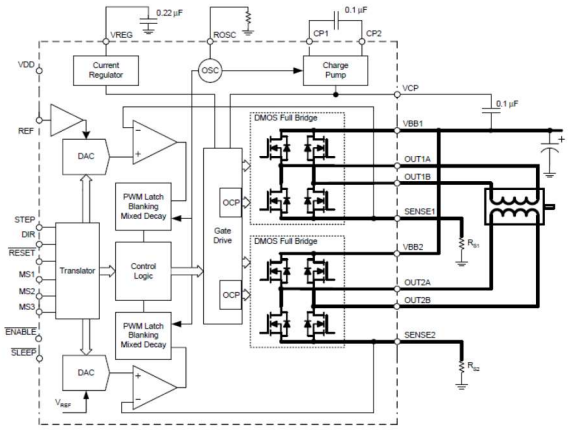
Characteristics	Symbol	Test Conditions	Min.	Typ.[2]	Max.	Units
Output Drivers						
Load Supply Voltage Range	V_{BB}	Operating	8	–	35	V
Logic Supply Voltage Range	V_{DD}	Operating	3.0	–	5.5	V
Output On Resistance	R_{DSON}	Source Driver, I_{OUT} = –1.5 A	–	320	430	mΩ
		Sink Driver, I_{OUT} = 1.5 A	–	320	430	mΩ
Body Diode Forward Voltage	V_F	Source Diode, I_F = –1.5 A	–	–	1.2	V
		Sink Diode, I_F = 1.5 A	–	–	1.2	V
Motor Supply Current	I_{BB}	f_{PWM} < 50 kHz	–	–	4	mA
		Operating, outputs disabled	–	–	2	mA
Logic Supply Current	I_{DD}	f_{PWM} < 50 kHz	–	–	8	mA
		Outputs off	–	–	5	mA
Control Logic						
Logic Input Voltage	$V_{IN(1)}$		V_{DD}×0.7	–	–	V
	$V_{IN(0)}$		–	–	V_{DD}×0.3	V
Logic Input Current	$I_{IN(1)}$	V_{IN} = V_{DD}×0.7	–20	<1.0	20	µA
	$I_{IN(0)}$	V_{IN} = V_{DD}×0.3	–20	<1.0	20	µA
Microstep Select	R_{MS1}	MS1 pin	–	100	–	kΩ
	R_{MS2}	MS2 pin	–	50	–	kΩ
	R_{MS3}	MS3 pin	–	100	–	kΩ
Logic Input Hysteresis	$V_{HYS(IN)}$	As a % of V_{DD}	5	11	19	%
Blank Time	t_{BLANK}		0.7	1	1.3	µs
Fixed Off-Time	t_{OFF}	OSC = VDD or GND	20	30	40	µs
		R_{OSC} = 25 kΩ	23	30	37	µs
Reference Input Voltage Range	V_{REF}		0	–	4	V
Reference Input Current	I_{REF}		–3	0	3	µA
Current Trip-Level Error[3]	err_I	V_{REF} = 2 V, %$I_{TripMAX}$ = 38.27%	–	–	±15	%
		V_{REF} = 2 V, %$I_{TripMAX}$ = 70.71%	–	–	±5	%
		V_{REF} = 2 V, %$I_{TripMAX}$ = 100.00%	–	–	±5	%
Crossover Dead Time	t_{DT}		100	475	800	ns
Protection						
Overcurrent Protection Threshold[4]	I_{OCPST}		2.1	–	–	A
Thermal Shutdown Temperature	T_{TSD}		–	165	–	°C
Thermal Shutdown Hysteresis	T_{TSDHYS}		–	15	–	°C
VDD Undervoltage Lockout	V_{DDUVLO}	V_{DD} rising	2.7	2.8	2.9	V
VDD Undervoltage Hysteresis	$V_{DDUVLOHYS}$		–	90	–	mV

[1]For input and output current specifications, negative current is defined as coming out of (sourcing) the specified device pin.

[2]Typical data are for initial design estimations only, and assume optimum manufacturing and application conditions. Performance may vary for individual units, within the specified maximum and minimum limits.

[3]V_{ERR} = [(V_{REF}/8) – V_{SENSE}] / (V_{REF}/8).

[4]Overcurrent protection (OCP) is tested at T_A = 25°C in a restricted range and guaranteed by characterization.



Fuente: http://tienda.bricogeek.com/impresion-3d/553-pololu-a4988-stepstick-prusa-reprap.html

11.3. PLANOS CONSTRUCTIVOS CAD

11.4. SOFTWARE

11.4.1. Programa ARDUINO

11.4.2. Programa Processing A

11.4.3. Programa Processing B

11.5. PRESUPUESTO – ANALISIS ECONOMICO DE VIABILIDAD

11.5.1. Material:

TFG - DESARROLLO SISTEMA DE ESCANEO LASER 3D DE BAJO COSTE

TITULO - Ingeniería Marítima Julio 2017

"Dirección":
"Ciudad":
"Provincia":
"Teléfono":

"Codigo Postal"
"Página Web"

Fecha Solicitud	
Comercial	
Número de Cliente	
Método de Pago	
Términos de Pedido	

Solicitado por:

comentarios: Valoración económica del material[18] utilizado

Producto	Cantidad	Descripción	Unidades	Precio / unidad	Precio
1	1	Arduino + contenedor	1	14,50 €	14,50 €
2	1	motor NEMA 17	1	15,00 €	15,00 €
3	1	interface Pololu A4889	1	3,00 €	3,00 €
4	1	Display LCD 16x2	1	7,00 €	7,00 €
5	5	potenciómetros	5	0,80 €	4,00 €
6	2	interruptores	2	0,50 €	1,00 €
7	1	Fuente Alimentación 12 V	1	20,00 €	20,00 €
8	1	LASER lineal	1	3,90 €	3,90 €
9	1	Webcam Trust	1	10,00 €	10,00 €
10	1	PCB fotosensible doble cara	1	5,00 €	5,00 €
11	1	Química para el atacado	1	2,00 €	2,00 €
12	1	Cableado	20	7,00 €	7,00 €
13	1	Zócalos soldables	20	5,00 €	5,00 €
14	1	Varilla roscada 6mm x 1 m	1	2,00 €	2,00 €
15	1	tornillería varia	1	3,00 €	3,00 €
16					
				Subtotal	102,40 €
				Total Venta	**102,40**

Si tiene alguna duda sobre este presupuesto no dude en comunicarse con nosotros

[18] Precios 2016 en Aliexpres (https://es.aliexpress.com)

11.5.2. Mano de Obra

Considerando los tiempos de cronograma teóricos, empleados para el desarrollo de este proyecto y estimando el coste de hora a 36.00 €, podemos estimar una cifra económica orientativa del coste de desarrollo del mismo.

TFG - DESARROLLO SISTEMA DE ESCANEO LASER 3D DE BAJO COSTE

TITULO - Ingeniería Marítima Julio 2017

"Dirección"
"Ciudad"
"Provincia"
"Teléfono"

"Código Postal"
"Página Web"

Fecha Solicitud	
Comercial	
Número de Cliente	
Método de Pago	
Términos de Pedido	

Solicitado por:

comentarios: Valoración Económica Mano de Obra estimada

Producto	Cantidad	Descripción		Precio / horas	Precio
1	84	1-Identifiacaion de Objetivos		36,00 €	1.008,00 €
2	636	2-Elaboracion Hardware	-40%	36,00 €	5.227,00 €
3	1.528	3-Elaboracion Software	-40%	36,00 €	22.000,00 €
4	283	4-Memoria Técnica		36,00 €	3.396,00 €
20					
				Subtotal	31.631,00 €
Si tiene alguna duda sobre este presupuesto no dude en comunicarse con nosotros			**0,00%**	IVA	0,00 €
				Costes de Envío	
			0,00%	**margen +**	0,00 €
				Total Venta	**31.631,00 €**

Tablas 1 y 2, listas de material y presupuesto de diseño
Fuente: Elaboracion propia

11.5.3 Amortizacion del presupuesto:

Se presenta a continuación dos opciones de presupuesto orientado a justificar la inversión en recursos materiales y humanos que se han utilizado para el desarrollo de este proyecto.

- **El primer caso**, define el presupuesto de ***amortización en el diseño y construcción de este prototipo para un determinado nº de unidades producidas para cada versión.***

  Para esta primera versión 1.0 antes de someterse a mejoras que se verán reflejadas en series posteriores, se han contemplado una producción de **100 unidades**, donde los costes iniciales de desarrollo iran incluidos proporcionalmente en cada unidad producida.

- **El segundo caso,** versa sobre la posibilidad de contemplar la ***amortización de los recursos*** explicados anteriormente *en el caso de* ***alquilar la maquina y el servicio de adquisición y procesado de datos***.

El primer caso, define el presupuesto de ***amortización en el diseño y construcción de este prototipo para un determinado nº de unidades producidas para cada versión.***

TFG - DESARROLLO SISTEMA DE ESCANEO LASER 3D DE BAJO COSTE

TITULO - Ingeniería Marítima Julio 2017

"Dirección"
"Ciudad"
"Provincia" "Codigo Postal"
"Telefóno" "Pagina Web"

Fecha Solicitud	
Comercial	
Número de Cliente	
Metodo de Pago	
Terminos de Pedido	

Solicitado por:

comentarios: Valoracion Económica Mano de Obra estimada / unidad de cada serie fabricada

Producto	Cantidad	Descripción	DED.- 40%	Precio / horas	Precio
1	84	1-Identifiacaion Obj.		36,00 €	1.008,00 €
2	636	2-Elaboracion Hardware		36,00 €	5.227,00 €
3	1.528	3-Elaboracion Software		36,00 €	22.000,00 €
4	283	4-Memoria Técnica		36,00 €	3.396,00 €
20	**10**	MATERIALES		138,24 €	1.382,40 €
				Subtotal	33.013,40 €
			21,00%	IVA	6.932,81 €
				Total Venta	**39.946,21 €**

NUMERO UNIDADES FABRICADAS SERIE	**10**	
COSTE UNITARIO POR CADA UNIDAD DE LA SERIE	**3.994,62 €**	**P.V.P 2017**

Tablas 3, Amortizacion venta
Fuente: Elaboracion propia

El segundo caso, versa sobre la posibilidad de contemplar la ***amortización de los recursos*** explicados anteriormente *en el caso de* ***alquilar la maquina y el servicio de adquisición y procesado de datos***.

TFG - DESARROLLO SISTEMA DE ESCANEO LASER 3D DE BAJO COSTE					
TITULO - Ingeniería Marítima Julio 2017					
Solicitado por:					
comentarios: Valoracion Económica Mano de Obra estimada / ALQUILER DE MAQUINA					
Producto	**Cantidad**	**Descripción**	**DED.- 40%**	**Precio / horas**	**Precio**
1	84	1-Identifiacaion de Objetivos		36,00 €	1.008,00 €
2	636	2-Elaboracion Hardware		36,00 €	5.227,00 €
3	1.528	3-Elaboracion Software		36,00 €	22.000,00 €
4	283	4-Memoria Técnica		36,00 €	3.396,00 €
20	**10**	MATERIALES		138,24 €	1.382,40 €
				Subtotal	33.013,40 €
			21,00%	IVA	6.932,81 €
				Total Venta	**39.946,21 €**
NUMERO UNIDADES FABRICADAS SERIE				**10**	
COSTE UNITARIO POR CADA UNIDAD DE LA SERIE				**3.994,62 €**	**P.V.P 2017**
A- SERVICIO ALQUILER MAQUINA				**250,00 €**	**P.V.P 2017**
B-TECNICO/HORA (MIN 6 H)				**216,00 €**	**P.V.P 2017**
			TOTAL	**466,00 €**	**SERVICIO *COMPLETO***
AMORTIZACION EN Nº DE SERVICIOS/UNIDAD SERIE				**16**	**ALQUILER**
AMORTIZACION EN Nº DE SERVICIOS COMPLETOS /UNIDAD SERIE				**9**	

Tablas 4, Amortizacion alquiler
Fuente: Elaboracion propia

11.5.3. CONCLUSIONES ANÁLISIS ECONÓMICO:

Se adjunta a continuación tabla orientativa del nivel de amortización de los costEs de producción para determinado número de unidades fabricadas por serie. Se ve claramente que para cada mayor número de unidades producidas los costes proporcionales a la misma caen en grado exponencial.

Como orientación a la hora de fijar un precio competitivo donde se de por cubiertos los costes de diseño y producción, se establece llegar a una fabricación de **150 unidades o más de cada serie**, fijando un precio de venta minimo de **422,00 € / unidad.**

OPCION A		
UNIDADES / SERIE	PRECIO	
1	38.440,00 €	
10	3.994,00 €	
20	2.081,00 €	
30	1.443,00 €	
40	1.124,00 €	
50	933,00 €	
100	550,00 €	
150	422,00 €	MINIMO
200	359,00 €	63,00 €
250	320,00 €	39,00 €
300	295,00 €	25,00 €
500	243,00 €	52,00 €

Tablas 5, viabilidad unidades venta
Fuente: Elaboracion propia

En el caso B, podemos observar la comparativa en caso de ofrecer los servicios de alquiler del dispositivo con la opción de un técnico si asi fuera requerido (facturación minima de 6 horas a 36,00 €/hora).

Se puede ver la evolución de cuando empieza a ser rentable tras amortizar los costes de producción en las diferentes unidades de la serie, tras el número de servicios facturados.

OPCION B			
		ALQUILER	ALQUILER y TECNICO
UNIDADES / SERIE	PRECIO	250,00 €	466,00 €
1	38.440,00 €	154	82
10	3.994,00 €	16	9
20	2.081,00 €	8	4
30	1.443,00 €	6	3
40	1.124,00 €	4	2
50	933,00 €	4	2
100	550,00 €	2	1
150	422,00 €	2	1
200	359,00 €	1	1
250	320,00 €	1	1
300	295,00 €	1	1
500	243,00 €	1	1

Tablas 6, Amortizacion alquiler
Fuente: Elaboracion propia

11.6. INDICE DE ILUSTRACIONES:

AVISO:

Este documento es el resultado final del Trabajo Fin de Grado de un alumno, siendo su autor responsable de su contenido.

Se trata por tanto de un trabajo académico que puede contener errores detectados po el tribunal y que pueden no haber sido corregidos por el autor en la presente edición.

Debido a dicha orientación académica no debe hacerse un uso profesional de su contenido.

Este tipo de trabajos, junto con su defensa, pueden haber obtenido una nota que oscila entre 5 y 10 puntos, por lo que la calidad y el número de errores que puedan contener difieren en gran medida entre unos trabajos y otros.

La Universidad de Cantabria, la Escuela Superior de Naútica, los miembros del Tribunal de Trabajos Fin de Grado asi como el profesor tutor/director no son responsables del contenido ultimo de este Trabajo.

- Puede variarse el ancho del lomo con sólo aumentar o disminuir el tamaño de letra. En la muestra el tamaño de letra es Univers 10.

TÍTULO **Título en Inglés**	**DESARROLLO DE UN SISTEMA DE ESCANO LASER 3D DE BAJO COSTE** *DEVELOP OF A LOW COST 3D SCANNING SYSTEM*			
AUTOR	**Abel Báscones Gutiérrez**			
DIRECTOR/A	**Miguel Ángel Mateo Larscorz**			
TITULACIÓN	***INGENIERÍA MARÍTIMA***	**FECHA**	18 Julio 2017	**TOMO I DEI**