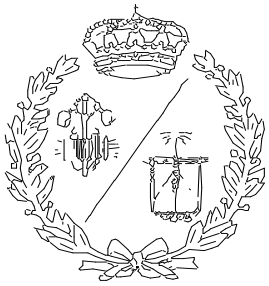
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Proyecto Fin de Grado

DISEÑO DE UN ESCÁNER 3D DE BAJO COSTE

(Designing a low cost 3D scanner)

Para acceder al Título de

GRADUADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: Francisco de Borja Lanza Ortega

Septiembre - 16

Índice general:

DOCUMENTO 1

MEMORIA

ÌNDICE

1. PLANTEAMIENTO

1.1. INTRODUCCIÓN

Este trabajo está encaminado a servir como proyecto de fin de carrera para la obtención del título de Grado en Ingeniería Industrial en Electrónica y Automática que se expide en la Universidad de Cantabria.

El objetivo del proyecto es diseñar un escáner 3D que sea asequible económicamente, fácil de montar y que permita a los usuarios poder obtener una imagen 3D del objeto a escanear pudiendo configurar ciertas características que permitan realizar el proceso de una forma más eficiente. A diferencia de otros escáneres este será de tamaño reducido y portátil para que de esta forma cualquier usuario pueda tenerlo en su casa.

El escáner funcionará con un láser que se irá desplazando por unos carriles, a su vez tendrá una plataforma giratoria para que de esta forma sea posible reducir el número de oclusiones donde el escáner es incapaz de reconstruir tridimensionalmente los puntos del objeto a escanear.

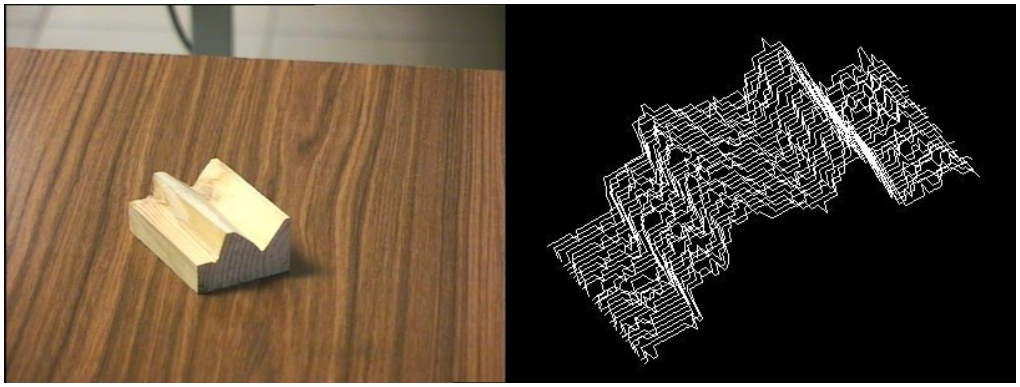
El escáner constará de una placa Raspberry Pi, un láser, una cámara y varios motores paso a paso; ya que se requiere saber en todo momento la posición del láser y la rotación de la plataforma.

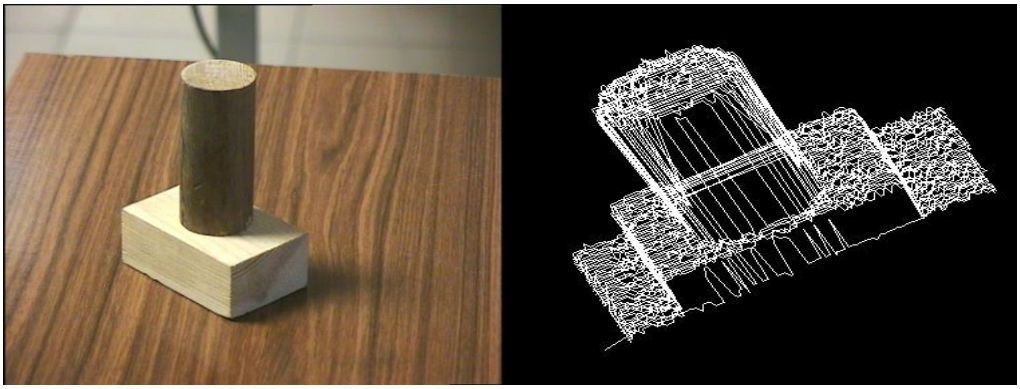
Antes de proceder a temas más complejos es necesario previamente poseer una serie de conocimientos básicos y técnicos sobre los métodos reconstrucción 3D que pueda facilitarle al lector un mejor entendimiento sobre el trabajo realizado.

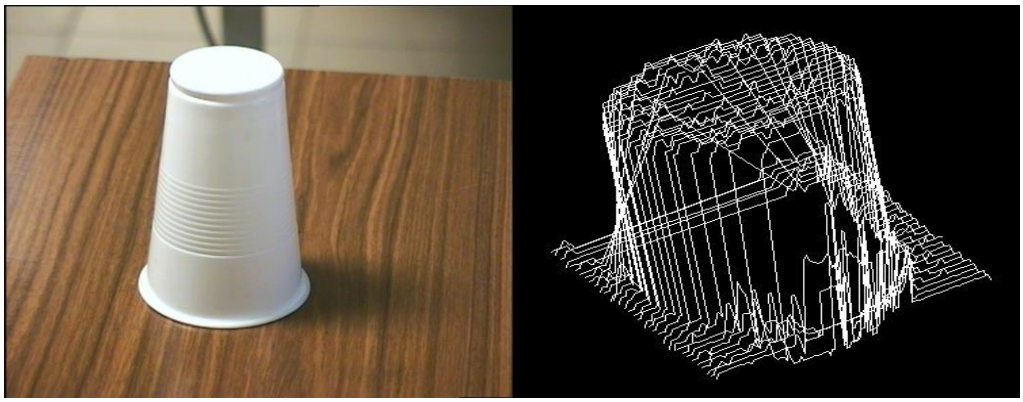
Según la Universidad Politécnica de Madrid la reconstrucción 3D se define como: “el proceso mediante el cual, objetos reales, son reproducidos en la memoria de una computadora, manteniendo sus características físicas (dimensiones, volumen y forma). Existen dentro de la visión artificial, multitud de técnicas de reconstrucción y métodos de mallado 3D, cuyo objetivo principal es obtener un algoritmo que sea capaz de realizar la conexión del conjunto de puntos representativos del objeto en forma de elementos de superficie, ya sean triángulos, cuadrados o cualquier otra forma geométrica.” [1]

Todas las reconstrucciones darán como resultado una nube de puntos, la cual estará formada por un conjunto de puntos en el que cada uno constará de sus correspondientes coordenadas [x, y, z] y al unirlos se creará una malla con la forma del objeto a reconstruir.

A continuación se muestran varios ejemplos de reconstrucciones 3D partiendo de objetos reales y la obtención de sus nubes de puntos.







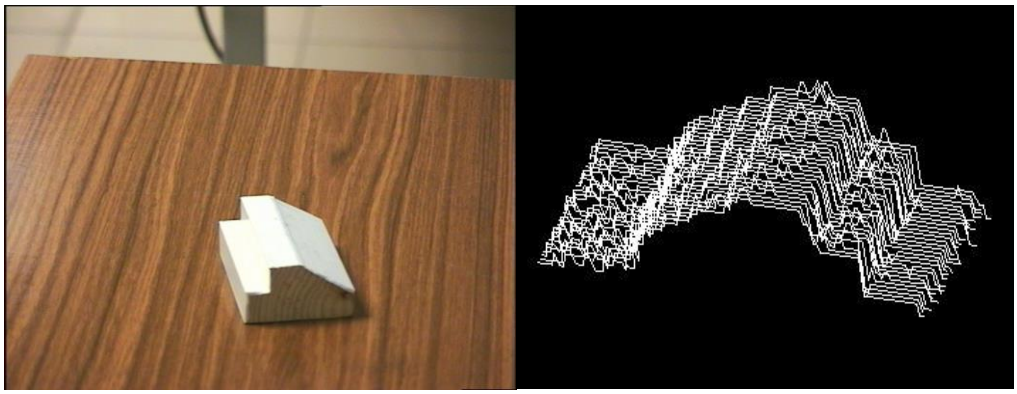


Figura 1.1; Ejemplos [1]

Las nubes de puntos mostradas en la figura 1.1 presentan ciertos errores que pueden ser debidos tanto a la técnica aplicada como al material del objeto reconstruido.

En la utilización de los métodos de reconstrucción 3D dependiendo de la estructura utilizada para la obtención del contorno del objeto que se quiere reconstruir, pueden ser clasificados de la siguiente forma:

<table>
<tr><td colspan="3">Métodos de reconstrucción 3D</td></tr>
<tr><td>Contacto</td><td colspan="2">Sin Contacto</td></tr>
<tr><td rowspan="2">Guías lineales</td><td>Activo</td><td>Pasivo</td></tr>
<tr><td>Tiempo de vuelo</td><td>Sistemas estereoscópicos</td></tr>
<tr><td rowspan="2">Brazo articulado</td><td>Triangulación</td><td>Sistemas fotométricos</td></tr>
<tr><td>Holografía conoscópica</td><td rowspan="3">Técnicas de detección de siluetas</td></tr>
<tr><td rowspan="2">Mixto</td><td>Escáner láser de mano</td></tr>
<tr><td>Luz estructurada</td></tr>
</table>

Tabla 1. Clasificación de los distintos tipos de métodos de reconstrucción 3D.

- Contacto:

  Este tipo de método de reconstrucción requiere del contacto físico para captar las coordenadas 3D, para ello el objeto a reconstruir debe estar sobre una plataforma plana de precisión, cuando el objeto no es plano y tiene una superficie irregular por la cual no es posible que se mantenga de forma estable en la plataforma de escaneado, este será aguantado por un soporte transparente.

  Este método podrá tener tres diferentes configuraciones:

  - Un sistema de guías con brazos rígidos que se mantienen en una relación perpendicular y cada eje se desplaza a lo largo de unas guías.
  - Un brazo articulado con elementos rígidos y sensores angulares de alta precisión. La localización del punto final de la articulación requiere el cálculo por medio de procesos matemáticos.
  - Se puede usar una combinación de ambos métodos, los cuales pueden ser desde un brazo articulado hasta un vagón deslizante, para mapear grandes superficies con cavidades interiores o superficies superpuestas.

  Un ejemplo de este tipo de máquinas es la CMM (coordinate measuring machine) usada principalmente en procesos que requieren una gran precisión. La desventaja de este tipo de máquinas y la de todos los métodos de reconstrucción 3D de contacto es que al requerir el contacto con los objetos y al ejercer presión sobre estos para ser escaneados se puede llegar a producir daños o deformaciones en dicho objeto. Otra desventaja de este tipo de máquinas es su reducida velocidad; ya que mientras la velocidad de este tipo de máquinas es de unos cientos de hercios, los sistemas ópticos pueden llegar a funcionar hasta 500 veces más rápido.

- Sin contacto – Activo
  - Tiempo de vuelo:

    Es un sistema activo que precisa de un láser y un buscador de rango para reconstruir un objeto. Este buscador encuentra la distancia a la superficie del objeto, midiendo el tiempo de retorno al receptor. Ya que la velocidad de la luz es conocida, el tiempo de vuelo nos dará la distancia de la superficie. De esta forma si "*T*" es el tiempo de vuelo y "*C*" la velocidad de la luz, así pues la distancia hasta el objeto será de C*T/2. Por tanto la precisión de la reconstrucción dependerá de cuan preciso podamos medir el tiempo de vuelo.

Debido a que el buscador de rango láser solo detecta la distancia de un punto en la dirección en la que esté enfocado el sensor, este método de reconstrucción no registra todo su campo visual a la vez, sino que lo hace punto a punto y con el tiempo se cambiará la dirección de dicho sensor. Esta dirección puede ser cambiada, ya sea rotando el mismo sensor o mediante un sistema de espejos.

- Triangulación:

Algunos de los métodos de reconstrucción 3D están basados en la triangulación activa; estos proyectan un láser sobre un objeto y utilizan una cámara para buscar la ubicación del punto láser. En función de la distancia en la que el láser alcance la superficie, el punto aparecerá en diferentes lugares del campo de visión de la cámara.

Esta técnica es denominada triangulación debido a que dicho punto, la cámara y el emisor forman un triángulo. De este triángulo se conoce longitud del lado correspondiente a la cámara y al láser, el ángulo del láser y el ángulo de la cámara. Con estos tres datos se puede determinar todas las dimensiones del triángulo, y con ello la posición del punto deseado en el espacio tridimensional.

La desventaja de este sistema es que el tamaño del objeto a reconstruir viene limitado por el ángulo del vértice opuesto al escáner, ya que a medida que este ángulo se aparte de 90º menor será la precisión de la reconstrucción. Ya que este ángulo viene relacionado con la distancia del emisor y la cámara, aumentar la distancia entre ellos supone un incremento en las dimensiones del equipo de medida.

- Holografía conoscópica:

Según la definición de la página web Escáner e Impresión 3D: “Es una técnica interferométrica que consiste en hacer pasar un rayo reflejado en una superficie a través de un cristal birrefringente, esto es un cristal con dos índices de refracción, uno fijo y otro dependiente del ángulo de incidencia.

El resultado son dos rayos paralelos que se hacen interferir con una lente cilíndrica, esta interferencia es capturada por un sensor CCD, la frecuencia de esta interferencia determina la posición del objeto en el que se proyectó el rayo láser. Esta técnica permite la medición de orificios en su configuración colineal, alcanzando precisiones mejores que una micra. La ventaja de esta técnica es que puede utilizar luz no coherente, esto quiere decir que la fuente de iluminación no tiene porqué ser un láser, la única condición es que sea monocromática." [20]

- Escáner láser de mano:

Los escáneres láser de mano crean una imagen en 3D a través del método de triangulación activa descrito anteriormente. Este sistema proyecta un punto o una línea sobre el objeto a reconstruir desde un dispositivo portátil que cuenta con un sensor que mide la distancia a la superficie. Esta distancia es tomada con relación al sistema de coordenadas internas del propio aparato portátil y por ello una vez obtenida se debe pasar esta distancia a coordenadas globales por lo que es necesario poder determinar la posición del escáner. Esta posición puede ser determinada ya sea mediante una serie de marcas colocadas en la superficie de escaneo que servirán como referencia o mediante el uso de un método de seguimiento externo, el cual puede estar formado por un sistema de seguimiento láser (para proporcionar la posición del sensor) con una cámara integrada (para determinar la orientación del escáner) o se podrían aplicar otras soluciones como puede ser una fotogramétrica en la cual mediante el uso de tres o más cámaras se pueden determinar todos los grados de libertad necesarios para situar y orientar el escáner. Estas técnicas utilizan por norma diodos emisores de luz infrarroja conectados al escáner los cuales son captados por una o varias cámaras a través de filtros que proporcionan una capacidad de resistencia a la luz ambiental.

Los datos se recogen por un ordenador y se registran como puntos de un espacio tridimensional, estos datos más tarde serán unidos entre ellos formando una malla que tendrá la forma del cuerpo 3D.

- Luz estructurada:

Los métodos de reconstrucción 3D de luz estructurada consisten en la proyección de un patrón de luz determinado sobre el objeto a reconstruir y la detección de las deformaciones producidas en dicho patrón. En este método el patrón se proyecta utilizando un proyector LCD o bien otra fuente de luz estable para que una cámara,

situada a poca distancia del proyector, pueda captar la forma del patrón, calculando la distancia de cada punto en el campo visual.

Una de las mayores ventajas de este proceso es la velocidad y la precisión. Mientras otros métodos realizan una reconstrucción punto a punto, este método de reconstrucción escanea múltiples puntos o todo el campo visual a la vez. Esta reconstrucción es realizada en una fracción de un segundo para con ello lograr reducir sino eliminar la distorsión debida al movimiento. Gracias a esta elevada velocidad algunos de estos escáneres son capaces de reconstruir objetos en tiempo real.

Un análisis en tiempo real es aquel en el cual se recurre a la proyección de franjas digitales y a la técnica de cambio de fase para la capturara, reconstrucción, y renderización de objetos dinámicos deformables (tales como expresiones faciales) que pueden ir a una velocidad desde 40 hasta 120 fotogramas por segundo. También es posible aislar superficies para ser reconstruidas, por ejemplo, una expresión facial, gracias al desenfoque binario, este método de reconstrucción 3D puede llegar a cientos de miles de fotogramas por segundo.

- Sin contacto – Pasivo

Los métodos de reconstrucción 3D pasivos son aquellos que no emiten ningún tipo de radiación, sino que recurren a la detección de la radiación ambiental reflejada en los objetos a reconstruir. La mayoría de los métodos de este tipo detectan la luz visible debido a que es una radiación fácilmente disponible. También podría utilizarse otro tipo de radiación, como la luz infrarroja. La principal ventaja de los métodos pasivos es su reducido precio, porque en la mayoría de los casos no es necesario un gran desembolso en hardware, sino que con simples cámaras digitales es posible realizar la reconstrucción.

Dentro de este método de reconstrucción 3D podemos encontrar tres tipos:

- Sistemas estereoscópicos: este tipo de sistemas se sirven de dos cámaras de vídeo situadas una al lado de la otra con muy poca distancia entre ellas mirando al mismo objetivo. Este sistema se basa en el principio utilizado por el sistema de visión del ser humano el cual analiza las diferencias entre ambas imágenes para poder determinar la distancia de cada punto.

- Sistemas fotométricos: este sistema solo requiere como mínimo de una sola cámara, la cual tomará diferentes imágenes de la misma escena en diferentes condiciones de iluminación. Esta técnica se sirven del modelo de formación de la imagen para obtener la orientación de la superficie de cada pixel
- Técnicas de reconocimiento de siluetas: utilizan contornos creados a partir de una secuencia de fotografías en torno a un objeto tridimensional sobre un fondo con un buen contraste para a continuación ser extruidas y cruzadas para obtener una aproximación visual del objeto

1.2. ANTECEDENTES Y OBJETIVO DEL PROYECTO

El objetivo de este proyecto es la creación de un escáner 3D de fácil montaje y de un precio económico. La idea de este proyecto surgió a partir de las impresoras 3D, las cuales montadas de fábrica, como la que se muestra a continuación, cuestan alrededor de 600 euros mientras que un paquete de montaje de su impresora equivalente cuesta alrededor de 300 euros**.**

Figura 1.2; Ejemplos de Impresoras 3D

Como podemos observar en la figura 2 la diferencia de precio es clara siendo casi un 50% del precio total del producto. Sin embargo cuando se buscan escáneres 3D predominan 2 tipos de modelos: uno con plataforma giratoria y cámara con desplazamiento vertical, y otro solo con plataforma giratoria.

Figura 1.3; Ejemplos de escáneres 3D

El primer escáner de la figura 1.3 es más económico y será capaz de realizar reconstrucciones 3D de los objetos que deseemos pero la cámara no se eleva y por lo tanto tendrá ciertos puntos muertos donde el escáner no llega a escanear, mientras que el segundo no tiene este problema pero su precio son casi 400€ superior al modelo anterior lo cual es un incremento importante para lograr el efecto deseado.

Por ello se ha propuesto como objetivo de este proyecto crear un prototipo de un escáner 3D de fácil montaje, que tenga la capacidad de elevar la cámara y cuyo precio sea más asequible.

1.3. ELECCIÓN DE LA TECNOLOGÍA

La tecnología seleccionada para la realización de este escáner 3D se ha basado en el bajo coste, tamaño y sencillez de sus componentes.

Para la realización de este escáner 3D portátil y de bajo coste, se utilizará una Raspberry Pi para el procesado de las imágenes y varios motores para mover el objeto, un láser para delimitar la zona a reconstruir, así como una cámara para capturar la imagen. Todos estos elementos formarán parte de un sistema de visión artificial para el cual se desarrollará el software adecuado.

Se puede definir la visión artificial como el proceso de obtención de datos del mundo real a partir de imágenes, gracias a un ordenador. Desde un punto de vista práctico la visión artificial es un sistema diseñado para imitar las funciones del sistema de visión humano. Este sistema de visión es capaz desde la reconstrucción de objetos sencillos hasta complicadas reconstrucciones tridimensionales.

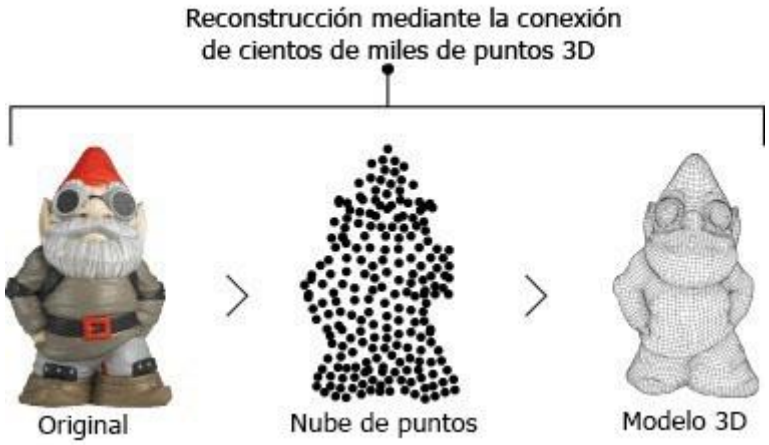


Figura 1.4; Resultado de un método de reconstrucción 3D [12]

Un sistema de visión artificial está formado por los siguientes elementos:

- Subsistema de iluminación: Consiste en un conjunto de elementos fotónicos que emiten sobre el objeto a reconstruir. En nuestro caso el escáner 3D contará con un láser.
- Subsistemas de captación: Compuesto por los transductores que transforman la radiación reflectada en una señal digital. Esto incluye todas las cámaras con sus distintos rangos de frecuencia. Aquí nos serviremos de una webcam.
- Subsistemas de adquisición: Existe una fuerte tendencia a la utilización de sistemas digitales, sin embargo aún persisten los de carácter analógico, los cuales para poder ser usados tienen que ser muestreados y cuantificados, para lo cual se utilizan tarjetas de adquisición de datos (“frame grabbers”).
- Subsistemas de procesamiento: Formados por un ordenador o clúster de ordenadores, los cuales se encargan del tratamiento de las imágenes así como de la obtención y manipulación de los datos. En nuestro caso será la Raspberry Pi.
- Subsistemas de periféricos: Son los receptores de la información de alto nivel, por ejemplo un monitor.

1.4. ETAPAS DE UN SISTEMA DE ESCANEADO 3D

El proceso de escaneado láser consta básicamente de dos etapas:

1. Calibración de la cámara y del plano láser.
2. Proceso de escaneado.

Por ello a continuación se representan las distintas etapas de ambos procesos:

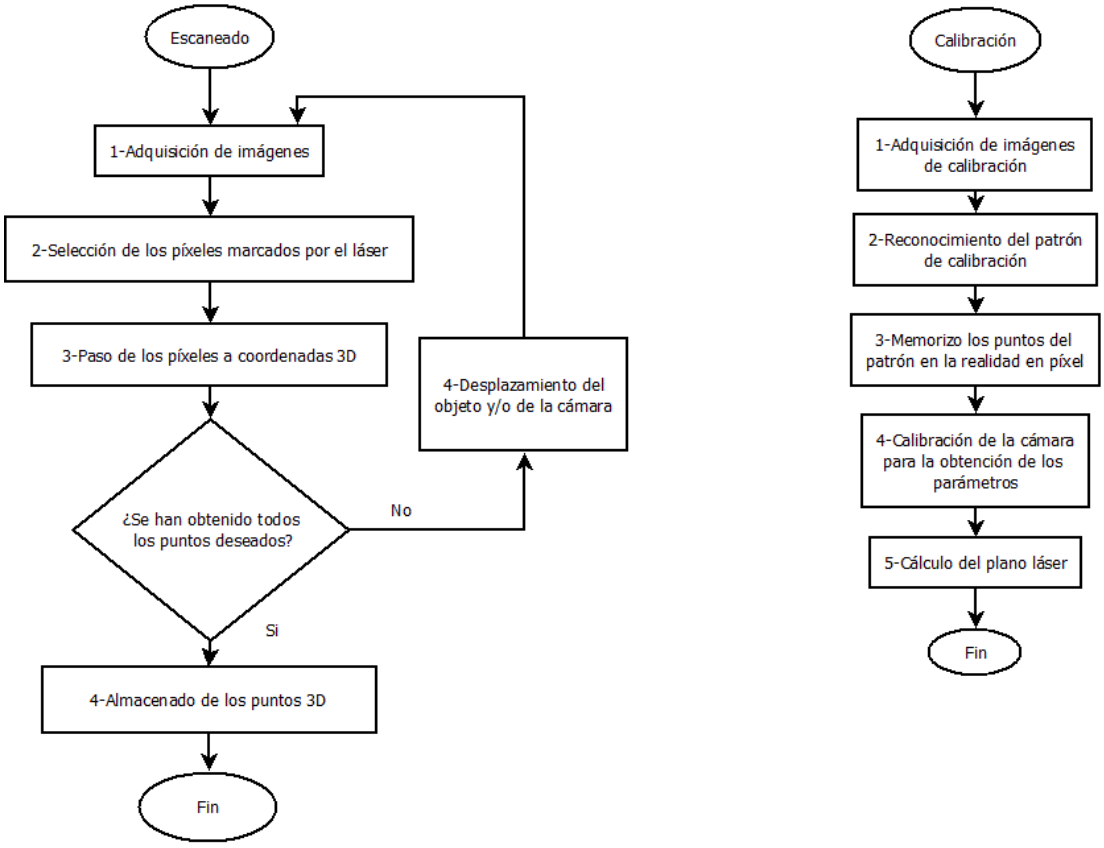


Figura 1.5; Flujograma del funcionamiento de un escáner 3D.

1.5. FORMATOS DE LOS ESCÁNERES 3D

Existen diferentes formatos capaces de almacenar las coordenadas [x y z] de una gran cantidad de puntos. Estos formatos se utilizan para representar objetos 3D a partir de lo que se denomina una nube de puntos.

Entre los diferentes formatos los más frecuentes son:

Formatos:
LAS
XYB
FLS
FWS
XYZ
TXT
ASC
PTG
PTS
PTX
CLR
CL3

Tabla 2; Formatos frecuentes de archivos de un escáner 3D.

Y entre todos ellos, los más conocidos y utilizados son sin lugar a dudas los formatos ".xyz", por ello se procederá a continuación a explicar la estructura interna de dicho formato.

Este formato es un archivo ASCII o un archivo binario que contiene un conjunto de vértices.

En un archivo ".xyz" aparecen conjuntos de tres números en punto flotante que representan las coordenadas X, Y, Z de un vértice.

Hay dos tipos o categorías de archivos ".xyz" llamados Mesh y Scan:

XYZ Tipo 1: Mesh
• Los vértices de la malla están dispuestos en una matriz de M por N. • M corresponde al número de columnas del archivo. • N corresponde al número de filas del archivo y cada una debe contener M puntos de muestra. • El tiempo de muestreo entre dos puntos de la malla es variable, dependiendo este del diseño del escáner. • La elevación del objeto escaneado puede ser positiva o negativa. • Los vértices de la malla se encuentran distribuidos en un espacio 3D, estando relacionados entre ellos por los valores asignados a las variables M y N. • Las coordenadas de cada punto de la malla se pueden encontrar en ASCII con un espacio de separación entre cada coordenada o bien los usuarios pueden establecer delimitadores específicos y cualquier número de terminaciones de línea. Estos datos se pueden almacenar por fila o por columna.

Ejemplo de formato ".xyz" tipo Mesh:

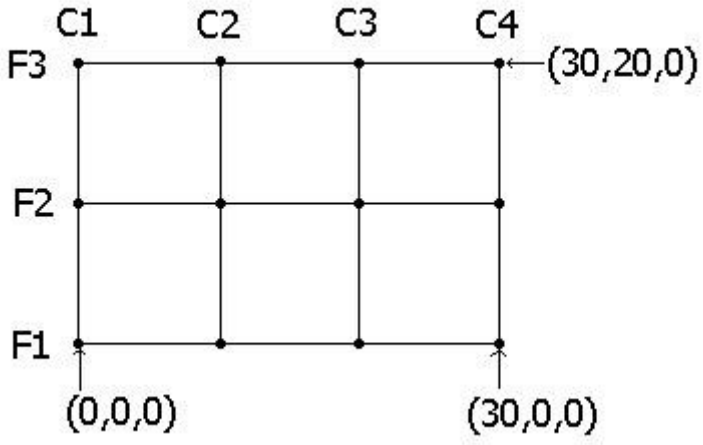


Figura 1.6; Imagen guía

Ahora simularemos un fichero ".xyz" escrito de forma manual que contiene los puntos de la imagen anterior. Para ello podemos observar que esta imagen tiene 3 filas, 4 columnas y 12 vértices. Estos vértices pasarán a ser los puntos representados con las coordenadas [x, y, z] y que podrán ser guardados con una configuración de filas o de columnas de la siguiente forma:

Por Fila:			Por Columna:		
0	0	0	0	0	0
10	0	0	0	10	0
20	0	0	0	20	0
30	0	0	10	0	0
0	10	0	10	10	0
10	10	0	10	20	0
20	10	0	20	0	0
30	10	0	20	10	0
0	20	0	20	20	0
10	20	0	30	0	0
20	20	0	30	10	0
30	20	0	30	20	0

Tabla 3; Ejemplo de puntos 3D guardados de diferente forma

XYZ Tipo 2: Probe Scan
Un archivo de sonda ("Scan") ".xyz" se genera bajo ciertas condiciones: • Este tipo de escáner se coloca por encima del objeto a reconstruir. • El escáner se desplazara en un plano paralelo al XY que se mantiene a una distancia continua (Z) por encima del objeto. • Para cada uno de los pases de escaneo se hace que la sonda viaje en paralelo al eje X (todas las muestras tendrán una Y de valor constante y sólo las coordenadas X y Z serán variables).

• Al final de cada pase de escaneado, la sonda se moverá a una nueva posición Y y el proceso de exploración se repetirá.
• El número de muestras tomadas así como la distancia entre ellas y el tiempo de escaneado es variable y está determinado por las condiciones de diseño de la sonda.

• La dirección del muestreo puede ser unidireccional o bidireccional.

Además de esto también existe una variación de este tipo de ficheros en el cual aparte de las tres coordenadas, se añade el color de ese punto en formato RGB y para ello se añadirán 3 números representando el color.

1.6. SISTEMAS DE ACCIONAMIENTO DEL PROTOTIPO

En este apartado se describen las características de los motores que han de mover tanto la plataforma como el sistema de poleas de elevación de la cámara y el láser. Asimismo se definirán y se seleccionarán los demás dispositivos que se encargarán de controlar dicho motores.

1.6.1. Selección de los motores

Para la creación de un escáner 3D se requiere un motor con el que se tenga un control completo de la posición del rotor, así como de la diferencia de grados entre cada posición y la capacidad de quedar enclavado en una posición determinada. En base a esto se ha elegido para este proyecto un motor paso a paso, puesto que este tipo de motores poseen las siguientes características:

- Se desplazan un paso por cada pulso que se les aplica y este paso puede variar de 1.8º a 90º por lo que los de 90º necesitarán 4 pasos y los de 1.8º 200 pasos para completar un giro completo de 360º.
- Tienen la capacidad de poder quedar enclavados en una posición o bien totalmente libres.

Este enclavamiento se produce si alguna de las bobinas del motor esta activada y en caso contrario quedará libre.

Dependiendo de la construcción interna de los motores paso a paso se clasifican en:

1. Motores de reluctancia variable.
2. Motores de imán permanente.
3. Motores híbridos.

Según la información obtenida en la página web de la Universidad del País Vasco, cada uno de estos motores se caracteriza:

Tipo de motor:	Detalles:
De reluctancia variable	Su rotor está fabricado por un cilindro de hierro dentado y el estator está formado por bobinas. Este tipo de motor trabaja a mayor velocidad que los de imán permanente.
De imán permanente	Su rotor es un imán que posee una ranura en toda su longitud y el estátor está formado por una serie de bobinas enrolladas alrededor de un núcleo o polo. Este proyecto se centrará en este tipo de motores puesto que son los más sencillos y utilizados.
Híbridos	Son una combinación de los anteriores, de ellos se obtiene un alto rendimiento a una alta velocidad.

Tabla 4; Clasificación de los diferentes tipos de motores paso a paso. [19]

Una vez hecha esta distinción pasamos a centrarnos en los motores paso a paso de imán permanente; ya que estos, aparte de ser los más fáciles de usar, también son los más extendidos y por ello serán los empleados en el prototipo.

En esta clase de motor paso a paso se pueden distinguir a su vez dos tipos:

- Unipolares: Constituidos por un polo.
- Bipolares: Constituidos por dos polos.

Es muy fácil distinguirlos debido a su número de cables; los unipolares tienen 6 (Figura 1.7, Derecha) y los bipolares tienen 4 (Figura 1.7, Izquierda).

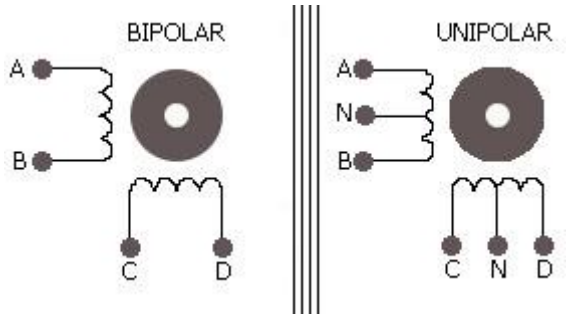


Figura 1.7; Disposición interna de los motores paso a paso

Para este proyecto en concreto se ha escogido el motor paso a paso bipolar de imán permanente, Nema 17 modelo: 17HD34008-22B marca: Busheng.

1.6.2. Descripción y especificaciones del motor seleccionado

Descripción:

- Cable de 1m, uno se conecta a la máquina eléctrica, en el otro extremo tiene una variedad de puertos opcionales; con efecto de tubería de encogimiento para prevenir que el alambre se doble.
- Adecuado para el motor a pasos tipo mezcla 17HD34008-22B.
- Modo de conducción: Conducción de corriente constante de onda en horquillas.
- Modo de salida: Dos fases 4 alambres.
- Corriente nominal (una fase): 1.5A DC.
- Voltaje nominal: 3.45V.
- Ángulo a pasos: 1.8°.

Especificaciones:

- Condiciones Operativas: Temperatura Ambiental: -20~50?; RH: 90%MAX; Posición de Montura: Instalación en Eje horizontal o vertical.
- Resistencia al viento de corriente directa (25): 2.3±10%.
- Inductancia de bobinado: 3mH±20%.
- Torsión del engrane: 12mN.m REF.
- Torsión de soporte: 300mN.m I=1.5ª.
- Frecuencia de inicio máxima sin carga: 1500pps.
- Frecuencia de funcionamiento máxima sin carga: 8000pps.

- Elevación de Temperatura: <80K.
- Precisión del ángulo a pasos: 1.8°± 5%.
- Inercia de giro: 38g.cm2.
- Peso del Motor: 0.23Kg/PC REF.
- Resistencia al aislamiento: La resistencia al aislamiento en frío debería ser más de 100m (entre el núcleo del estator del motor y el terminal).
- Fuerza dieléctrica: El espacio entre el núcleo del estator del motor y el terminal debería ser capaz de soportar 600V/1s de CA sin averiarse.
- La corriente de fuga es menor a 1mA.

1.6.3. Control del motor

Para controlar este motor se va a necesitar un puente en H por cada bobina, es decir, dos por cada motor.

Sin embargo, a pesar de que existen muchos diseños de puentes en H, todos se basan en el mismo proceso. El objetivo de este puente es cambiar el sentido de circulación de la corriente eléctrica que pasa por la bobina del motor.

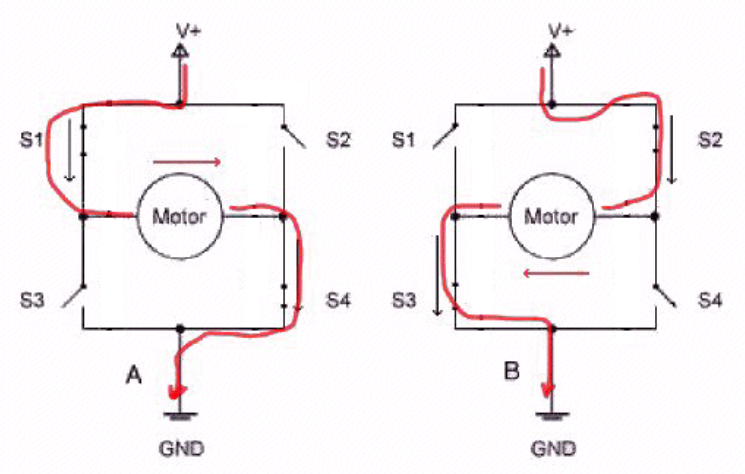


Figura 1.8; Sentido de la corriente en un motor paso a paso [18]

Así en la figura 1.8 se observa que activando S1 y S4 se logra hacer circular una corriente que atraviesa el motor de izquierda a derecha, mientras que activando S2 y S3 se consigue hacerla circular en sentido inverso.

De entre todos los puentes en H, se ha escogido para nuestro prototipo la utilización del chip "l293" que contiene en su interior dos puentes en H (Figura 1.9).

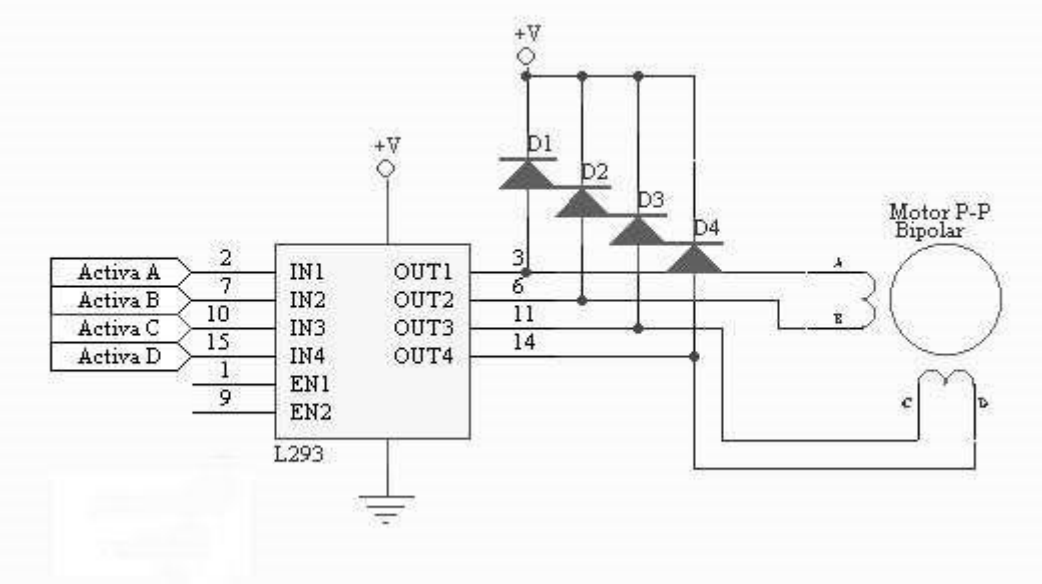


Figura 1.9; Circuito de control de un M. Bipolar [17]

Especificaciones del circuito:

- +Vcc puede ser de cualquier valor entre 3 y 15V DC dependiendo del voltaje del motor usado.
- No conectar el terminal "Activa C" y "Activa D" a +Vcc al mismo tiempo, provocará un cortocircuito.
- No conectar el terminal "Activa A" y "Activa B" a +Vcc al mismo tiempo, provocará un cortocircuito.

Este circuito ha sido escogido ya que tanto la potencia como la intensidad requerida por el circuito pueden ser proporcionadas por una Raspberry Pi.

En la figura 1.9, se muestra la forma de utilización del chip "l293" para el control de un motor paso a paso bipolar, siendo "Activa A"," Activa B"," Activa C" y "Activa D" las entradas del circuito que se controlarán y por las cuales se posiciona el motor.

1.6.4. Secuencias para el manejo de motores paso a paso

Una vez expuestas las características de los motores paso a paso, sus tipos y el equipo necesario para su funcionamiento, se explicará cómo usar dicho equipo ya que dependiendo del tipo y el sentido de rotación, su secuencia de activación será diferente.

Secuencias para el manejo de motores paso a paso Bipolares

Estos motores utilizan el cambio de sentido de la corriente en su bobinado con una determinada secuencia para conseguir su correcto funcionamiento. Cada cambio de sentido de la corriente origina el desplazamiento del eje en un paso, cuyo sentido de giro vendrá dado por la secuencia realizada. La tabla con la secuencia para el control de motores paso a paso bipolares es la siguiente:

PASO	TERMINALES			
	A	B	C	D
1	+V	-V	+V	-V
2	+V	-V	-V	+V
3	-V	+V	-V	+V
4	-V	+V	+V	-V

Tabla 5. Secuencia de control de un motor bipolar

Secuencias para el manejo de motores paso a paso Unipolares

Estos motores funcionan mediante la utilización de tres tipos diferentes de secuencias, todas ellas comienzan en el paso 1 y al llegar al último paso vuelven al paso inicial. Para cambiar el sentido de giro, se realizan las secuencias de forma inversa.

- **Secuencia Normal:** Esta es la más utilizada y la que recomienda la mayoría de los fabricantes. Esta secuencia se caracteriza por la utilización en todos sus pasos de dos bobinas, lo cual producirá un alto torque de paso y de retención.
- **Secuencia del tipo wave drive:** Esta secuencia se caracteriza por la utilización de una sola bobina por cada paso, esto proporciona un movimiento más suave del motor pero con el inconveniente de obtener un torque de paso y de retención menor que la secuencia normal.
- **Secuencia del tipo medio paso:** Esta secuencia cuenta con 8 pasos en vez de los 4 de las otras secuencias. Este tipo de secuencia se realiza activándose primero dos bobinas y luego una y así sucesivamente.

Todos los datasheet de los elementos usados por el prototipo aparecen en el anexo.

1.7. MODELO DE CÁMARA Y CALIBRACIÓN

Lo primero que tenemos que tener claro es que el modelo lineal de una cámara está definido por dos matrices; la matriz de parámetros intrínsecos y la matriz de parámetros extrínsecos.

Matriz de parámetros intrínsecos

$$\begin{bmatrix} f_c(1) & alpha\_c * f_c(1) & cc(1) \\ 0 & f_c(2) & cc(2) \\ 0 & 0 & 1 \end{bmatrix}$$

Matriz de parámetros extrínsecos

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}$$

Estas dos matrices se obtienen a través de la calibración de la cámara, utilizando para ello un conjunto de imágenes que contengan un patrón de calibración. La matriz de parámetros intrínsecos será la misma para todas las imágenes ya que representa las características físicas de la cámara; mientras que la segunda representa la transformación euclídea entre el sistema de coordenadas de cámara y el sistema de coordenadas de referencia.

La matriz de parámetros intrínsecos consta de:

- $f_c(1)\ y\ f_c(2)$:
  Representan la localización del punto focal y teóricamente tendrían que ser iguales. Sin embargo debido a diferentes razones serán distintos:
  - Defectos en el sensor de la cámara digital.
  - La imagen ha sido escalada de manera no uniforme en el post-procesado.
  - La lente de la cámara introduce una distorsión indeseada.
  - La cámara utiliza un formato anamórfico, donde la lente comprime un tamaño de pantalla a una de tamaño estándar.
  - Errores en la calibración de la cámara.

  En todos estos casos la imagen tiene pixeles no cuadrados.

- cc(1) y cc(2):
  Estas dos coordenadas vendrán definidas por la distancia entre el punto de intersección del eje óptico (recta perpendicular al plano que pasa por el punto focal) con el plano de imagen y el origen del sistema de coordenadas de imagen.
  Estas coordenadas "cc(1) y cc(2)" serán medidas en dicho sistema de coordenadas.

- coeficiente de inclinación:
  El coeficiente de inclinación que define el ángulo entre los ejes X e Y del píxel; este se almacena en la variable alpha_c.

Explicación del paso 3D-2D:
A continuación se exponen dos imágenes en diferentes perspectivas de los sistemas de referencia:

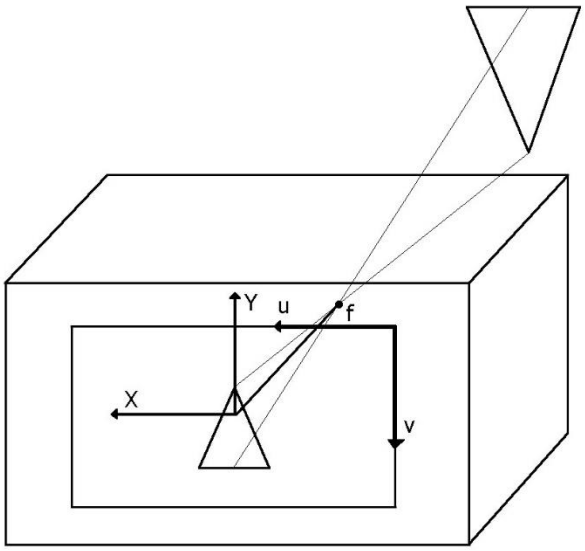


Figura 1.10

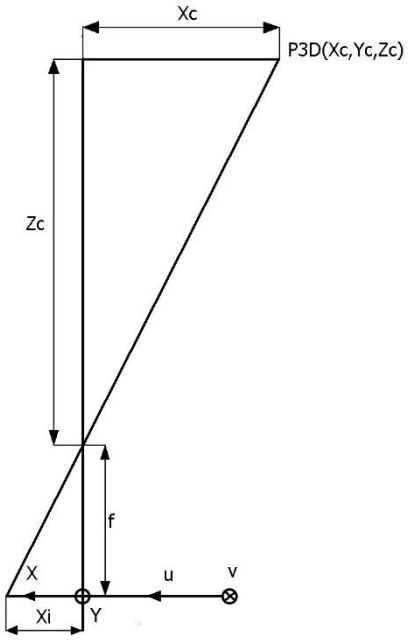


Figura 1.11

Para pasar de 3D a 2D lo primero de todo es coger un punto 3D expresado en coordenadas de cámara y obtener las coordenadas Xi e Yi de su proyección en el plano de imagen.
Como se observa en la Figura 1.11, se conoce el valor de la distancia focal "f" y las coordenadas del punto 3D. Teniendo en cuenta que los ángulos correspondientes entre ambos triángulos son iguales, podemos establecer una relación entre las longitudes de sus lados. Por ello:

$$Xi = f_c(1) * \frac{x_c}{z_c}$$
$$Yi = f_c(2) * \frac{y_c}{z_c}$$

Estos "Xi y Yi" serán ya las coordenadas de la. A continuación, se expresa dicho punto en el sistema de coordenadas de imagen (añadiendo cc(1) cc(2) a su correspondiente coordenada):

$$Xi = f_c(1) * \frac{x_c}{z_c} + cc(1)$$
$$Yi = f_c(1) * \frac{y_c}{z_c} + cc(2)$$

Esto sería equivalente a la siguiente operación matricial:

$$\begin{bmatrix} \alpha * x \\ \alpha * y \\ \alpha \end{bmatrix} = \begin{bmatrix} f_c(1) & alpha\_c * f_c(1) & cc(1) \\ 0 & f_c(2) & cc(2) \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$$

También tomamos "$alpha\_c * f_c(1)$" como "0". Esto nos dará como resultado un punto en coordenadas homogéneas al que se dividirá cada coordenada resultante por α.

A continuación tendremos que aplicar una modificación en dichas coordenadas ya que la óptica producirá una serie de deformaciones en la imagen. Usando OpenCV y su comando "cv2.calibrateCamera()" uno de los valores que nos devuelve será un vector con los coeficientes de distorsión. Estos estarán distribuidos de la siguiente forma en OpenCV:

$$Coeficientes\ de\ distorsión = (k_1\ k_2\ p_1\ p_2\ k_3)$$

Los elementos 1, 2 y 5 del vector ("kc") de coeficientes de distorsión, se aplican en la siguiente fórmula:

$$x_d = \begin{bmatrix} x_d(1) \\ x_d(2) \end{bmatrix} = (1 + k_c(1)r^2 + k_c(2)r^4 + k_c(5)r^6)x_n + d_x$$

Siendo dx el valor de la deformación tangencial. Para ello se aplican los otros dos coeficientes que faltan mediante la siguiente formula:

$$dx = \begin{bmatrix} 2k_c(3) * x * y + \ k_c(4)(r^2 + 2x^2) \\ k_c(3)(r^2 + 2y^2) + 2k_c(4) * x * y \end{bmatrix}$$

Por lo tanto, el vector kc contiene tanto la distorsión radial y los coeficientes de distorsión tangencial (observar que el coeficiente del término de sexto orden de la distorsión radial es la quinta entrada del vector kc).

Cabe destacar que este modelo de distorsión fue introducido por primera vez por Brown en 1966 y llamado modelo "plomada" (polinomio radial + "prisma delgado"). La distorsión tangencial se debe al "descentramiento", o del centrado imperfecto de los componentes de las lentes y otros defectos de fabricación en una lente compuesta.

Una vez que se aplica la distorsión, el píxel final de las coordenadas x_pixel = [Xi; Yi] de la proyección de P sobre el plano de la imagen es:

$$\begin{cases} Xi = f_c(1)(x_d(1) + alpha_c * x_d(2)) + cc(1) \\ Yi = f_c(2) * x_d(2) + cc(2) \end{cases}$$

Por lo tanto el vector de coordenadas píxel y el vector de coordenadas normalizadas (distorsionadas) xd están relacionados entre sí a través de la ecuación lineal:

$$\begin{bmatrix} Xi \\ Yi \\ 1 \end{bmatrix} = KK * \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix}$$

Donde KK, es la matriz de cámara y está definida como:

$$KK = \begin{bmatrix} f_c(1) & alpha\_c * f_c(1) & cc(1) \\ 0 & f_c(2) & cc(2) \\ 0 & 0 & 1 \end{bmatrix}$$

Una vez realizados todos los procedimientos anteriores se aplica una última transformación para pasar de este sistema de coordenadas al utilizado como referencia.

Para ello aplico la matriz de parámetros extrínsecos:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}$$

Esta matriz compuesta de una primera matriz 3x3 (“r”) que es una matriz de rotación al que se le añade una columna (“t”) que es la matriz de translación.
Por todo ello todas las operaciones de paso de 3D-2D se podrían resumir de la siguiente manera:

$$\begin{bmatrix} \alpha * x \\ \alpha * y \\ \alpha \end{bmatrix} = \begin{bmatrix} f_c(1) & alpha\_c * f_c(1) & cc(1) \\ 0 & f_c(2) & cc(2) \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} r_{11}\ r_{12}\ r_{13}\ t_x \\ r_{21}\ r_{22}\ r_{23}\ t_y \\ r_{31}\ r_{32}\ r_{33}\ t_z \end{bmatrix} * \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix}$$

$$\alpha * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = [M] * \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix}$$

1.8. RECONSTRUCCIÓN TRIDIMENSIONAL

Una vez introducido el modelo matemático por el cual se pasa de 3D a 2D procedemos a explicar los métodos que permiten reconstruir la posición tridimensional (es decir, pasar de 2D a 3D), los cuales son principalmente dos: sistemas de visión estéreo y sistemas de triangulación activa.

- Sistema estéreo:
  Este sistema se basa en dos cámaras que enfocan al objeto del que desean obtener las coordenadas 3D.
  Partiendo de las ecuaciones del modelo 3Dpodemos decir que de cada una de las cámaras:

$$\begin{bmatrix} \alpha * u_i \\ \alpha * v_i \\ \alpha_i \end{bmatrix} = [M_i] * [P_{3d}]$$

$$\begin{bmatrix} \alpha * u_d \\ \alpha * v_d \\ \alpha_d \end{bmatrix} = [M_d] * [P_{3d}]$$

Despejando estas ecuaciones podemos sacar las siguientes ecuaciones para ambas cámaras:

$$\alpha * u = \boldsymbol{m}_1^T * [P_{3d}]$$
$$\alpha * v = \boldsymbol{m}_2^T * [P_{3d}]$$
$$\alpha = \boldsymbol{m}_3^T * [P_{3d}]$$

(Los subíndices indican la fila de la matriz que representa)

Una vez obtenidos estas 3 ecuaciones podemos simplificarlas aún más sustituyendo la tercera ecuación en las dos anteriores:

$$(u * \boldsymbol{m}_3^T - \boldsymbol{m}_1^T) * [P_{3d}] = 0$$
$$(v * \boldsymbol{m}_3^T - \boldsymbol{m}_2^T) * [P_{3d}] = 0$$

Con estas ecuaciones aplicadas en ambas cámaras terminaremos obteniendo:

$$\begin{bmatrix} u_i * \boldsymbol{m}_3^T - \boldsymbol{m}_1^T \\ v_i * \boldsymbol{m}_3^T - \boldsymbol{m}_2^T \\ u_d * \boldsymbol{m}_3^T - \boldsymbol{m}_1^T \\ v_d * \boldsymbol{m}_3^T - \boldsymbol{m}_2^T \end{bmatrix} * [P_{3d}] = A * [P_{3d}] = 0$$

Siendo P_{3d}:

$$P_{3d} = \alpha \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Con esta última ecuación seríamos capaces de obtener un punto 3D a partir de las coordenadas pixel de las dos cámaras.
Sin embargo para realizar este proceso necesitamos saber qué pixeles son los que coinciden para el punto 3D que nosotros deseamos saber.

Partimos de una distribución de las dos cámaras como la siguiente:

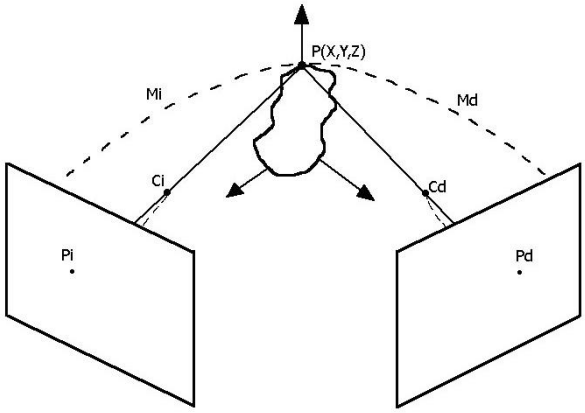


Figura 1.12

A partir de estas dos distribuciones de la cámara lo que hacemos es crear un nuevo plano en el cual nosotros colocaremos las dos imágenes creadas por la cámara. Tal y como se muestra en la siguiente imagen:

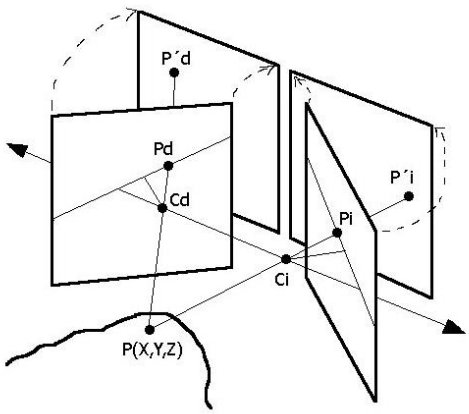


Figura 1.13

Y con ello obteniendo lo siguiente:

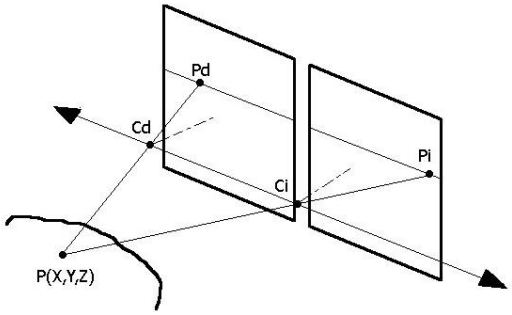


Figura 1.14

Como observamos en esta imagen se crea un plano formado por tres puntos (el punto 3D y los puntos focales de cada cámara); este plano al cortar los dos planos de la cámara formará una línea en la cual se encontraran los dos píxeles.
Nosotros partiremos para seleccionar el pixel una línea paralela a la parte inferior y después nosotros buscaremos en la misma línea de cada cámara dos pixeles uno en cada cámara que compartan características similares. Una vez aislados dichos pixeles procedemos a las ecuaciones anteriormente descritas para hallar las coordenadas del punto 3D.

- Sistema de triangulación activa :

En este sistema nos basaremos en un láser junto con la cámara para poder pasar de 2D-3D. Para ello partiremos de la ecuación general previa:

$$\alpha * \begin{bmatrix} u_d \\ v_d \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

A partir de esto lo que haremos será añadir a las ecuaciones ya existentes la ecuación del plano:

A*x+B*y+C*z+D=0

Dando como resultado:

$$\alpha * \begin{bmatrix} u_d \\ v_d \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ A & B & C & D \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Despejando conseguiremos:

$$\alpha' * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ A & B & C & D \end{bmatrix}^{-1} * \begin{bmatrix} u_d \\ v_d \\ 1 \\ 0 \end{bmatrix}$$

De esta forma ya hemos logrado conseguir una ecuación para pasar de 2D a 3D.

1.9. CALIBRACIÓN DEL PLANO LÁSER Y OBTENCIÓN DE LOS PÍXELES ILUMINADOS

Por último para terminar la explicación matemática sobre este tipo de escáner, procederemos a explicar las ecuaciones para obtener la ecuación del plano láser.

Existen diferentes métodos para obtener la ecuación del plano como por ejemplo el método de mínimos cuadrados. En este proyecto se utilizará la descomposición en valor singular debido a su gran precisión y a que ya existe en Python una función que realiza este procedimiento.

Partiendo de los puntos 3D que forman el plano láser que se encuentran almacenados en la matriz A, el procedimiento para obtener la descomposición en valores singulares es la siguiente:

$$A = \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 \\ X_2 & Y_2 & Z_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ X_n & Y_n & Z_n & 1 \end{bmatrix}_{nx4}$$

$$A \times A^t = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \\ r_{41} & r_{42} & r_{43} & r_{44} \end{bmatrix}_{4x4}$$

Se aplica la siguiente fórmula para la obtención de los valores propios de A:

$$|A \times A^t - \lambda I| = 0 \rightarrow \begin{matrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{matrix}$$

Siendo los valores singulares:

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

A partir de los valores propios se obtienen los vectores singulares por la izquierda.

$$(A \times A^t - \lambda_1) = 0 \rightarrow u_1 = [p_1]_{4x1} = \frac{1}{\sqrt{d_1}}[j_1]_{4x1}$$

$$(A \times A^t - \lambda_2) = 0 \rightarrow u_2 = [p_2]_{4x1} = \frac{1}{\sqrt{d_2}}[j_2]_{4x1}$$

$$(A \times A^t - \lambda_3) = 0 \rightarrow u_3 = [p_3]_{4x1} = \frac{1}{\sqrt{d_3}}[j_3]_{4x1}$$

$$(A \times A^t - \lambda_4) = 0 \rightarrow u_4 = [p_4]_{4x1} = \frac{1}{\sqrt{d_4}}[j_4]_{4x1}$$

Estos vectores tienen que ser ortonormales por ello $d_i = \sqrt{p_{i1}^2 + p_{i2}^2 + p_{i3}^2 + p_{i4}^2}$

Una vez obtenidos los vectores singulares por la izquierda se procede a obtener los vectores singulares por la derecha. Para ello se aplica la siguiente formula:

$$v_i = \frac{1}{\sigma_i} A^t u_i$$

Con ello se obtiene:

$$v_1 = \frac{1}{\sigma_1} A^t u_1 = \frac{1}{\sqrt{\lambda_1}} A^t \frac{1}{\sqrt{d_1}} [j_1]_{4x1}$$
$$v_2 = \frac{1}{\sigma_2} A^t u_2 = \frac{1}{\sqrt{\lambda_2}} A^t \frac{1}{\sqrt{d_2}} [j_2]_{4x1}$$
$$v_3 = \frac{1}{\sigma_3} A^t u_3 = \frac{1}{\sqrt{\lambda_3}} A^t \frac{1}{\sqrt{d_3}} [j_3]_{4x1}$$
$$v_4 = \frac{1}{\sigma_4} A^t u_4 = \frac{1}{\sqrt{\lambda_4}} A^t \frac{1}{\sqrt{d_4}} [j_4]_{4x1}$$

Con todo esto se puede escribir la descomposición en valor singular:

$$U = [u_1\ u_2\ u_3\ u_4]_{4x4}$$

$$S = \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 \\ 0 & 0 & 0 & \sigma_4 \end{bmatrix}_{4x4}$$

$$V = [v_1\ v_2\ v_3\ v_4]_{4x4}$$

La última fila de la matriz V será la ecuación del plano que contenga todos los puntos de A. Por lo tanto con esto ya es posible obtener la ecuación del plano y de realizar una triangulación activa para pasar de 2D a 3D.

Lo único que nos queda es elegir el proceso por el cual escogemos los puntos que forman parte del plano láser. Para ello lo que hacemos es coger la imagen y separarla en filas, tomando estas en un vector y en cada posición del vector teniendo un valor diferente equivalente a su componente de color rojo. Una vez hecho esto, se podrían aplicar cualquiera de las siguientes fórmulas para obtener el pico óptimo de intensidad de la banda láser:

Estimación	Fórmula
BR	$\hat{\delta} = \begin{cases} \frac{g(i)}{g(i)-g(i+1)} & f(i+1) > f(i-1) \\ \frac{g(i-1)}{g(i-1)-g(i)} & f(i+1) < f(i-1) \end{cases}$
GA	$\hat{\delta} = \frac{1}{2} * \left(\frac{\ln(a)-\ln(c)}{\ln(a)+\ln(c)-2*\ln(b)} \right)$
CM	$\hat{\delta} = \frac{c-a}{a+b+c}$
LA	$\hat{X} = \begin{cases} x - \frac{a-c}{2*(b-a)} & c > a \\ x - \frac{a-c}{2*(b-a)} & c =< a \end{cases}$
PA	$\hat{\delta} = \frac{1}{2} * \frac{a-b}{c-2*b+a}$

Tabla 10. Métodos de estimación.[23]

Explicación de la tabla:

- $\hat{\delta}$ representa el desplazamiento de la posición del píxel.
- a, b y c representan los 3 pixeles consecutivos del pico, donde b es el pixel de mayor intensidad.

De las fórmulas de la tabla 10 se ha elegido la aproximación parabólica:

$$\hat{\delta} = \frac{1}{2} * \frac{a-b}{c-2*b+a}$$

Esta ha sido elegida por los pocos recursos que consume de la Raspberry Pi. Ya que el uso de una aproximación mucho más precisa elevaría mucho la carga y el tiempo de procesado de imágenes.

2. CONSTRUCCIÓN DEL PROTOTIPO

2.1. ELEMENTOS DEL PROTOTIPO

A continuación se procede a explicar los elementos que forman parte del prototipo así como su funcionamiento. Este escáner 3D, aparte de la base, cuenta con cuatro elementos principales:

- Soporte de la cámara y del láser.
- Plataforma de escaneado.
- Raspberry Pi y PCB.
- Cámara y láser.

El soporte de la cámara y del láser puede ser fijo o móvil. Este prototipo contará con un soporte móvil.

2.1.1. Soporte de la cámara y del láser

La plataforma de la cámara y del láser de este prototipo será una plataforma móvil, elevada por un motor, mediante una correa dentada que pasa por un sistema de poleas constituido por tres poleas de 22 mm de diámetro. (Descripción en plano general).

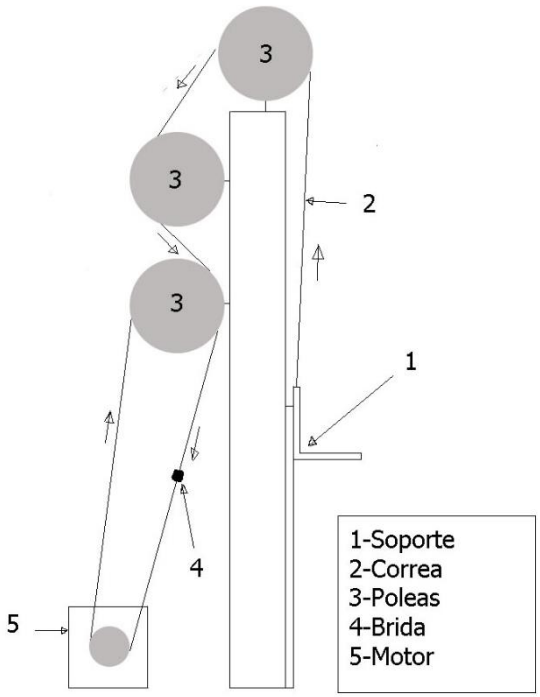


Figura 2.1; Sistema de elevación de plataforma mediante poleas.

En la figura 2.1 se puede observar el sistema de poleas previamente mencionado. En la imagen se muestra mediante flechas el movimiento que seguirá la correa para realizar la acción de elevación de la plataforma.

Hay que hacer especial hincapié en que la correa una vez que pasa por la polea inferior y por el motor, vuelve a dicha polea que al salir será fijada a si misma mediante una abrazadera, para que se mantenga siempre en tensión impidiendo que se escape de los piñones del motor.

Una vez construido queda de la siguiente forma:

Figura 2.2; Sistema de poleas del prototipo.

Figura 2.3; Correa dentada y sujeción.

El motor utilizado para la elevación de la plataforma es un motor paso a paso bipolar, de una potencia de 4W. Este motor además de controlar con precisión la posición de la plataforma tiene la potencia suficiente para elevarla.

2.1.2. Plataforma de escaneado

Esta plataforma será sobre la que se coloquen los elementos a escanear. El esquema de la plataforma de escaneado del prototipo es la siguiente:

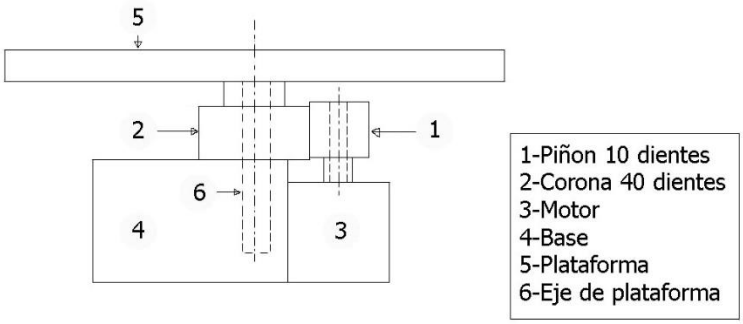


Figura 2.4; Esquema de la plataforma de escaneado. (Descripción en plano general)

Hay diferentes puntos que hay que destacar a la hora de diseñar esta pieza:

- El engranaje (2) sobre el que va la plataforma va unido a la estructura por medio de un eje (6).
- La plataforma (5) debe asentarse de una forma segura al eje (6) ya que cualquier tipo de holgura puede llevar a perturbaciones en la lectura.
- Los motores paso a paso (3) tienen un paso mínimo de 1.8º, esto es demasiado grande para un escáner 3D, por ello los engranajes deberán tener una relación de cómo mínimo “1:4” para que de esta forma el paso del motor pase de 1.8º a 0.45º.
- Se debe prestar especial atención a que los engranajes (1 y 2) queden perfectamente acoplados el uno con el otro ya que de no ser así la plataforma (5) adquiriría una holgura que produciría perturbaciones en el escaneado.

Una vez que se tiene todo esto en cuenta, se procede al montaje. El resultado sería lo siguiente:

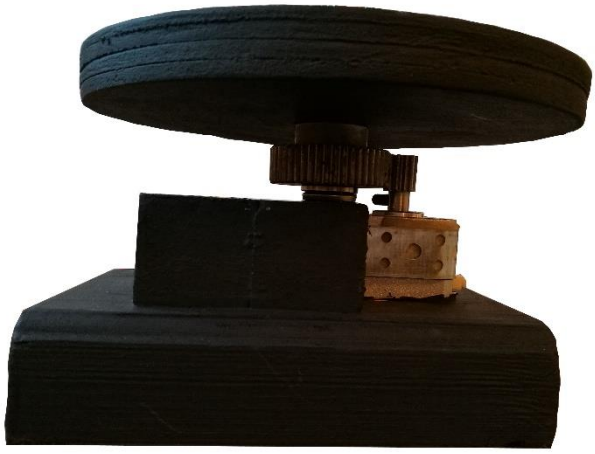


Figura 2.5; Plataforma de escaneado

El motor escogido para la rotación de la plataforma de escaneado es un motor paso a paso bipolar, con una potencia de 4W, idéntico al usado para la elevación. Este motor además de permitir controlar con absoluta seguridad la posición de la plataforma tiene potencia suficiente para rotar tanto la plataforma como a los objetos que se vayan a escanear. Aun así para reducir la carga del motor cuando se usen piezas de cierto peso se ha colocado un rodamiento axial entre el engranaje fijado en la plataforma y la base de esta.

2.1.3. Raspberry Pi y PCB

La Raspberry Pi será la encargada del procesamiento de imágenes y del control de los motores del prototipo mediante la conexión al PCB.

La Raspberry Pi ha sido escogida para este propósito debido a su capacidad de procesamiento de imágenes así como por su bajo coste que es muy inferior al del ordenador que se precisaría para utilizar otros modelos de escáner 3D. En concreto la Raspberry Pi utilizada en este prototipo es una Raspberry Pi 3 modelo B made in United Kingdom.

Especificaciones de la Raspberry Pi 3:

- Procesador:
  - Chipset Broadcom BCM2387.
  - 1,2 GHz de cuatro núcleos ARM Cortex-A53
- GPU
  - Dual Core VideoCore IV ® Multimedia Co-procesador. Proporciona Open GL ES 2.0, OpenVG acelerado por hardware, y 1080p30 H.264 de alto perfil de decodificación.
  - Capaz de 1 Gpixel / s, 1.5Gtexel / s o 24 GFLOPs con el filtrado de texturas y la infraestructura DMA
- RAM: 1GB LPDDR2.
- Conectividad
  - Ethernet socket Ethernet 10/100 BaseT
  - 802.11 b / g / n LAN inalámbrica y Bluetooth 4.1 (Classic Bluetooth y LE)
  - Salida de vídeo
    - HDMI rev 1.3 y 1.4
    - RCA compuesto (PAL y NTSC)
  - Salida de audio
    - jack de 3,5 mm de salida de audio, HDMI
    - USB 4 x Conector USB 2.0
  - Conector GPIO
    - 40-clavijas de 2,54 mm (100 milésimas de pulgada) de expansión: 2x20 tira
    - Proporcionar 27 pines GPIO, así como 3,3 V, +5 V y GND líneas de suministro
  - Conector de la cámara de 15 pines cámara MIPI interfaz en serie (CSI-2)
  - Pantalla de visualización Conector de la interfaz de serie (DSI) Conector de 15 vías plana flex cable con dos carriles de datos y un carril de reloj
  - Ranura de tarjeta de memoria Empuje / tire Micro SDIO

2.1.4. Cámara y láser

La cámara y el láser son los dos elementos de mayor importancia del escáner 3D y de ellos depende la precisión y la calidad de la reconstrucción. Ambos elementos han sido escogidos basándose fundamentalmente en la relación precio/calidad, sin olvidar que el objetivo de este proyecto es la obtención de un escáner 3D de bajo coste.

Cámara

Modelo de la cámara: Creative LIVE! Cam Chat HD.

Especificaciones:

- Sensor: sensor de imagen HD 720p (1.280 x 720).
- Resolución de vídeo: HD 720p (1.280 x 720 píxeles).
- Resolución de foto: 3,7 megapíxeles.
- Velocidad de fotogramas: hasta 30 fps de calidad HD 720p.
- Micrófono integrado con cancelación de ruido.
- Plug & chat en Windows, Mac OSX10.5 y Linux 2.6.
- Enfoque fijo.
- Base adaptable.
- Longitud del cable: 1,5 metros.
- USB 2.0 de alta velocidad.
- Longitud focal: 2.08mm.

Láser

Este elemento cuenta con un cabezal que nos permita emitir una franja láser y que funcione con el voltaje que proporciona la Raspberry Pi.

Modelo del láser: Genérico 650nm 5mW

Especificaciones:

- Salida: línea recta láser de color rojo (650nm).
- Tamaño: 12x35mm.
- Longitud del cable: 140mm.
- Voltage: 4.5-5V.
- Potencia: 5-20mW.
- Corriente: 45mA(MAX).

- Temperatura de operación: -10~40ºC.
- Ancho de línea: Ajustable.

2.2. DISEÑO DEL PCB

En este apartado se procederá a explicar el diseño y utilización del "PCB" usado en este prototipo. Como anteriormente se explicó en el apartado 1.6 este escáner 3D será controlado mediante dos motores paso a paso bipolares. Sin embargo estos motores no pueden ser controlados directamente si no que es necesario la utilización de puentes en H (dos por motor) para su correcto funcionamiento.

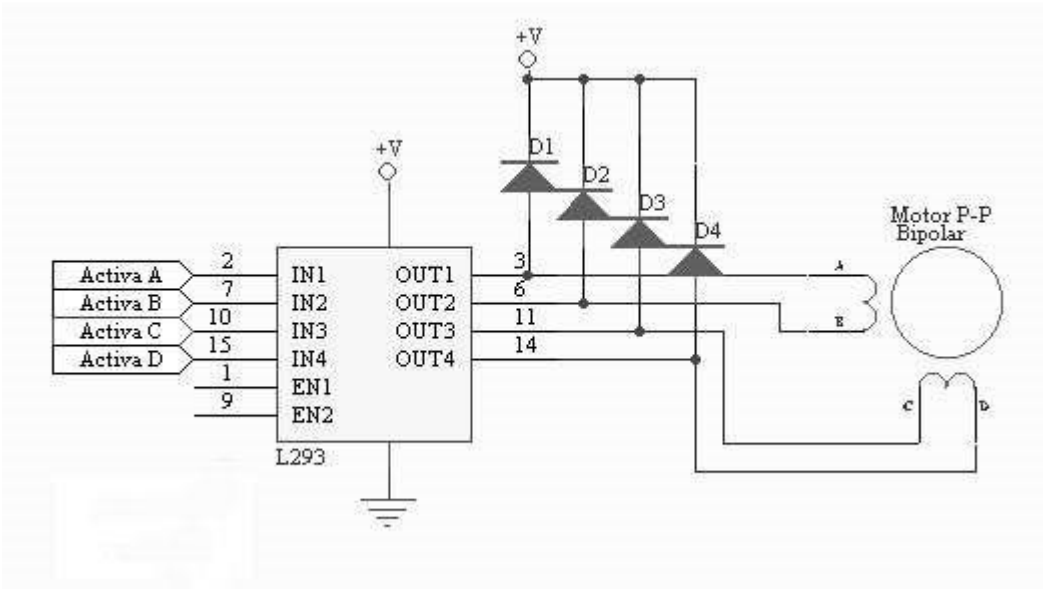


Figura 2.6; Esquema de un circuito de puentes en H para controlar un motor paso a paso bipolar.

Cada motor paso a paso bipolar deberá contar con el circuito representado anteriormente en la figura 2.6 para poder funcionar. En esta imagen "A, B, C y D" son las entradas correspondientes al motor y "Activa A, Activa B, Activa C y Activa D" corresponden a los pines de la Raspberry Pi en los cuales se introducirá la secuencia de activación de los motores paso a paso.

PASO	TERMINALES			
	A	**B**	**C**	**D**
1	+V	-V	+V	-V
2	+V	-V	-V	+V
3	-V	+V	-V	+V
4	-V	+V	+V	-V

Tabla 11. Secuencia de activación de los motores paso a paso bipolares.

Para la creación del PCB existen muchos programas; en este caso se utilizó el programa "Altium" al cual se le proporcionó el siguiente esquema eléctrico:

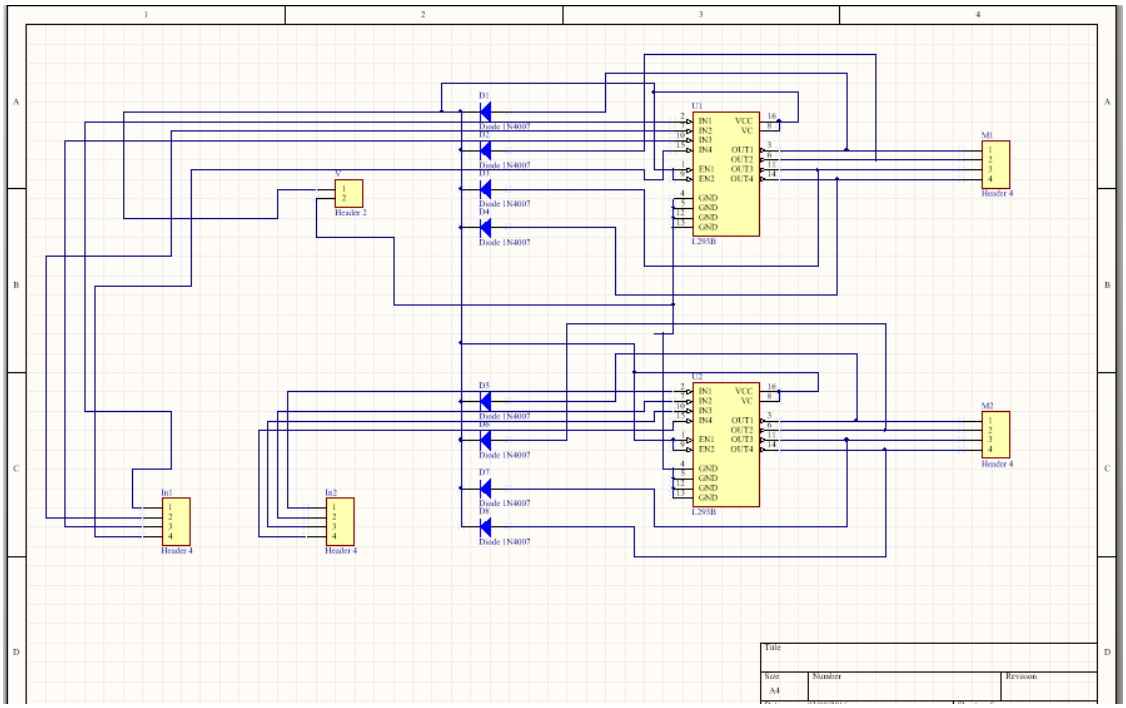


Figura 2.7; Esquema eléctrico del PCB del escáner 3D

Este esquema eléctrico cuenta con cuatro puentes en H, dos para cada motor y ocho diodos. Concretando más se trataría de dos chips “l293b” cada uno con dos puentes en H y ocho diodos “1N4007S”.

Los datasheet de estos elementos eléctricos se pueden ver en el anexo.

A partir de este diseño se obtuvo como resultado final un PCB como el siguiente:

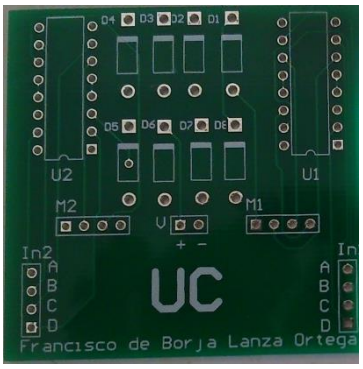


Figura 2.8; PCB obtenido con el diseño propuesto.

2.3. MONTAJE DEL PCB

A la placa PCB previamente diseñada se le soldarán tiras de pines machos y hembras, y los zócalos para el ensamblaje de los chips “L293B” así como todos los elementos que componen la placa. (Figura 2.9 y 2.10)

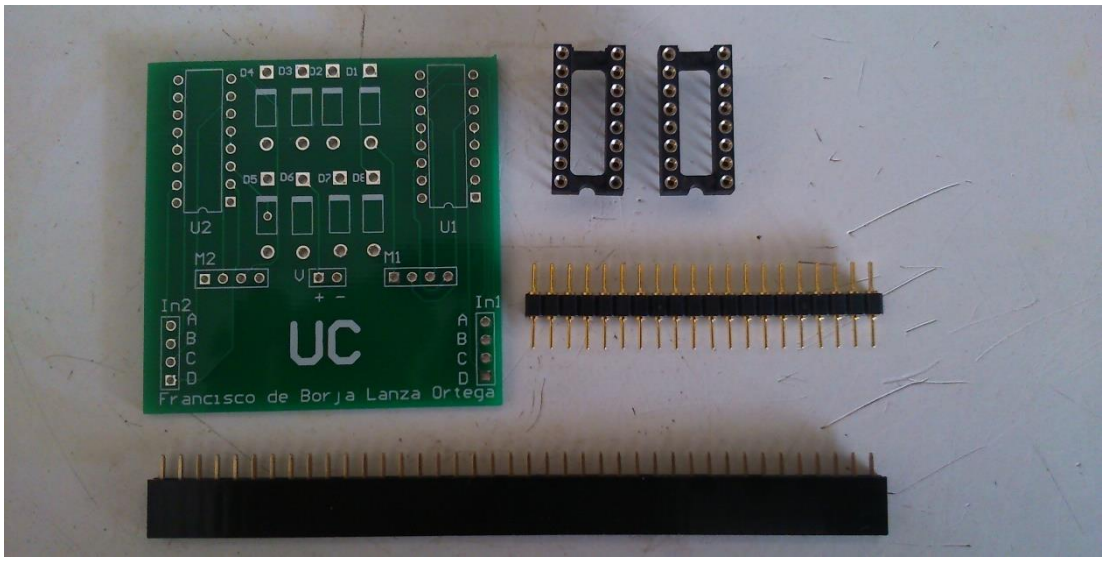


Figura 2.9; Componentes para el montaje del PCB

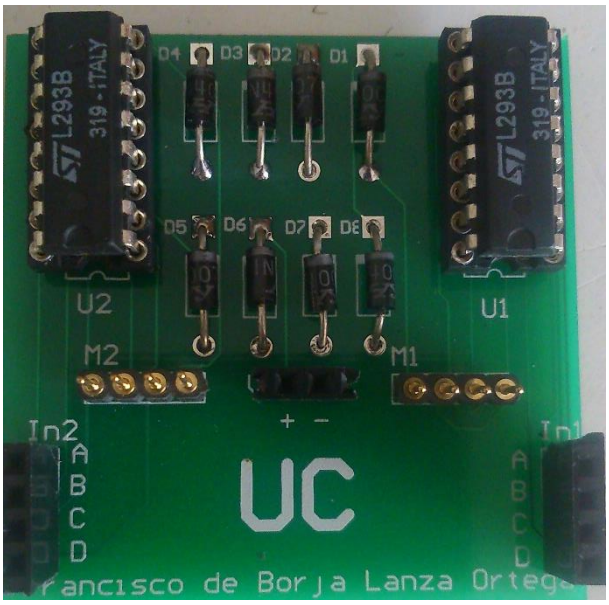


Figura 2.10; Aspecto final del “PCB”

Una vez completado el PCB se procede a conectar los motores y la Raspberry Pi a la placa; esta conexión se realizará según se representa en la figura 2.11.

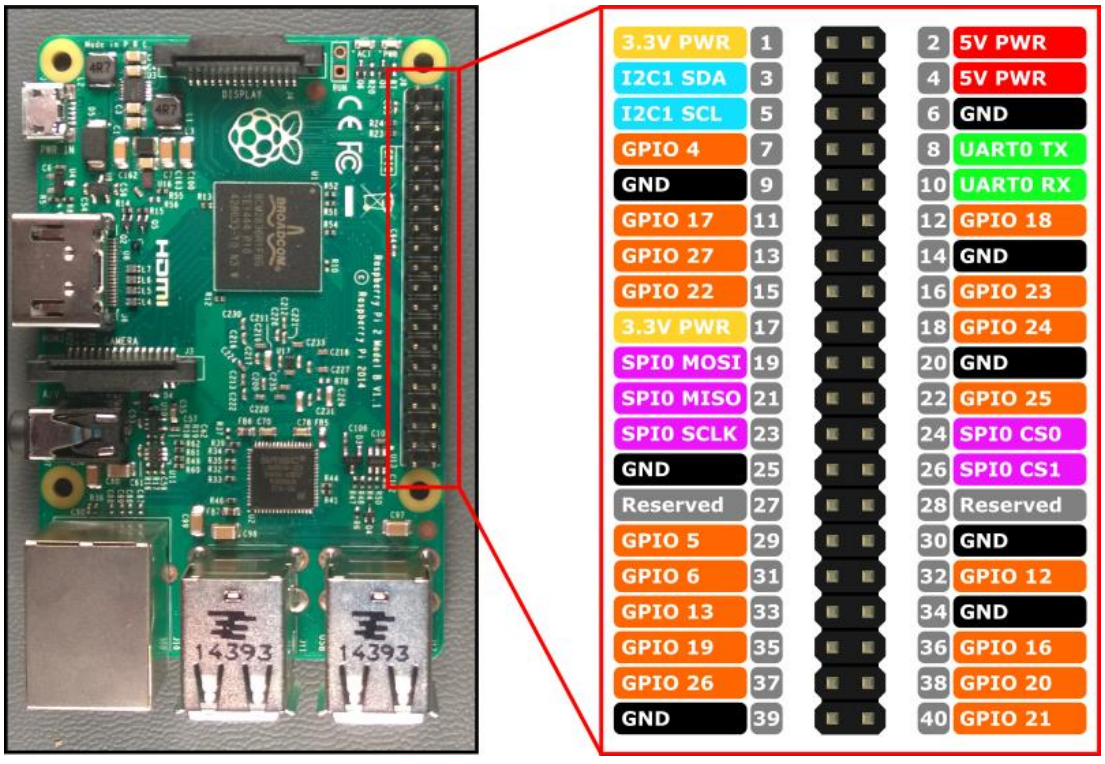


Figura 2.11; Distribución de los pines en una Raspberry Pi. [12]

La distribución de los pines como se observa en la figura 2.11, viene determinada de fábrica y no se puede cambiar. Los pines que se han utilizado para el control de los motores vienen determinados por el código del programa que se ha diseñado para este prototipo.

Los programas que controlarán los motores son: *pinelevacion.py* y *pinrotation.py*. Ambos programas aparecen en el anexo.

Estos programas se servirán de la librería "RPi.GPIO" para el control de los pines. En esta librería se deben declarar los pines que se desean usar e indicar si son pines de salida o de entrada de datos.

Una vez importada la librería "RPi.GPIO" como GPIO se procede a declarar los pines que necesitemos como pines de salida. A continuación se muestra un ejemplo del uso del código para declarar el pin nº 17 como pin de salida.

```
GPIO.setup(17, GPIO.OUT)
```

Una vez que se declaren todos los pines que se necesiten, al ser estos de tipo "GPIO" solo tendrán dos estados de salida "1" y "0". Estos estados serán asignados por el código a los pines de la siguiente manera:

```
GPIO.output(17, GPIO.LOW) →Pin 17 toma el valor "0"
GPIO.output(17, GPIO.HIGH) →Pin 17 toma el valor "1"
```

Una vez terminado todo el programa se liberarán los pines usados para poder ser utilizados en otras ocasiones, para ello se usará el comando:

```
GPIO.cleanup()
```

El método de instalación de la librería "RPi.GPIO" aparece en el anexo.

Por lo tanto las conexiones para el funcionamiento del prototipo con los programas: *pinelevacion.py* y *pinrotation.py* se harán de la siguiente manera:

Conexiones del prototipo:
• Conexión de los pines: 11, 12, 15 y 16, a las entradas: A, B, C y D respectivamente; conectando la salida de ese chip al motor encargado de rotar la plataforma. • Conexión los pines: 29, 31, 33 y 35, a las entradas: A, B, C y D respectivamente; conectando la salida de dicho chip al motor encargado de la elevación de la cámara. • Conexión de la placa PCB a la tensión de 5VCC y a tierra (GND).

2.4. INTERFAZ GRÁFICA

Para que el usuario use el escáner 3D de una forma más intuitiva y más fácil se ha desarrollado una interfaz gráfica.

Esta interfaz gráfica de por si no tiene una importancia relevante en el diseño del escáner 3D, ya que si nos fijamos en el código que viene más adelante, lo único que en verdad contribuye al uso del escáner es que dentro de esta interfaz se incluye una función llamada “cámara”, la cual nos permitirá tomar las fotos del patrón de calibración que más tarde será usado por otro programa para obtener la ecuación del plano láser.

La interfaz gráfica es una gran ayuda para todos los usuarios que no estén familiarizados con estos aparatos, así como de hacer el programa más manejable.

La interfaz permitirá acceder de forma rápida y sencilla a todas las funciones necesarias para hacer funcionar correctamente el escáner 3D.

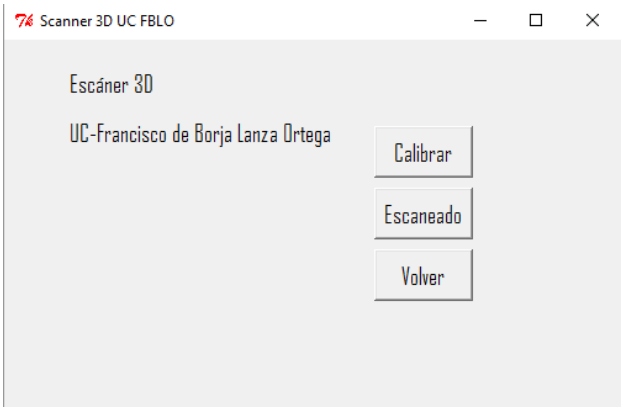


Figura 2.12; Ventana principal.

Como se muestra en la figura 2.12 al usuario se le ofrecerán dos opciones que serán las más utilizadas y necesarias en todo tipo de escáneres 3D. Estos son las funciones de escanear y calibrar el escáner 3D.

La opción de “Escaneado” se deberá realizar una vez que el escáner se encuentre calibrado. Esta opción hará que la interfaz gráfica llame a una función dentro de un programa aparte, llamado *escaneado.py*. Este realizará el escaneado del objeto y guardará todos los puntos calculados dentro de un archivo “.txt” llamado “Output.txt”.

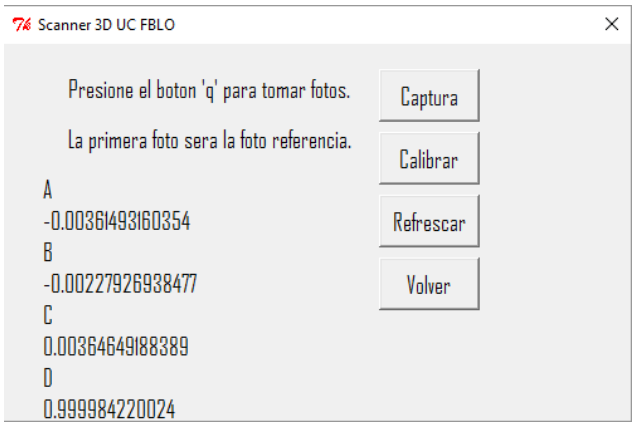


Figura 2.13; Subventana de calibrado.

Si es la primera vez que el usuario usa el escáner o ha pasado mucho tiempo desde que se utilizó por última vez, el usuario deberá realizar una calibración del escáner 3D para lo cual seleccionará la opción “Calibrar” de la ventana principal, ante lo cual el sistema le abrirá una subventana como la que aparece en la figura 2.13.

En esta subventana nos aparecerá la ecuación que actualmente está guardada en el escáner 3D y nos ofrecerá distintas opciones para la realización de la calibración.

Dentro de las cuales se encuentran:

- Captura: esta opción llamará a la función cámara definida dentro del código de la interfaz gráfica y nos permitirá tomar las fotos de calibración.
- Calibrar: una vez realizadas las fotos de calibración se procede a obtener la ecuación del plano láser. Para lo cual esta selección llama a una función dentro de un programa a parte llamado *USVoption.py*. Esta función nos permitirá seleccionar la banda láser en las imágenes y acto seguido nos proporcionará la ecuación del plano láser.
- Refrescar: esta opción refrescará la ventana, borrando la ecuación previa del plano láser y sustituyéndola por la que se acaba de obtener.
- Volver: esta opción cierra la subventana y devuelve al usuario a la ventana principal para proceder si el usuario lo desea a realizar el escaneado de un objeto 3D.

Una vez se ha calibrado el escáner y se pulsa el botón de escaneado aparece la siguiente ventana:

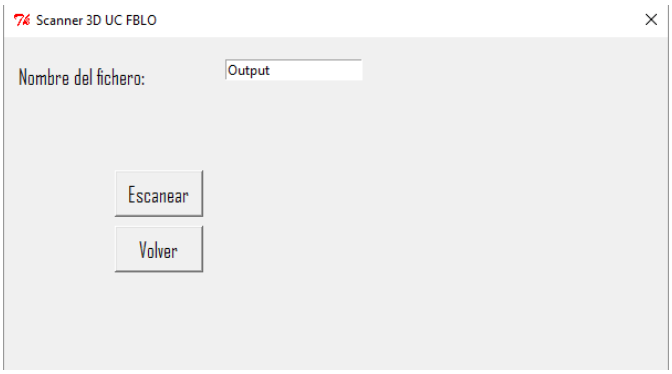


Figura 2.14; Subventana de escaneado.

En esta ventana nos permitirá escoger el nombre que tomará el archivo en el que se guarde el objeto escaneado.

El código de la interfaz gráfica es:

Código Python para la interfaz gráfica:

```
#-*-coding: utf-8-*-
# Programa en el cual se desarrolla toda la interfaz gráfica del programa.
# Se importan todas las librerías necesarias
from Tkinter import *
import cv2
import USVoption
import escaneado

# Se define la función "hija" la cual cuando se llame se abrirá una nueva ventana en la cual nos ofrecerá
#  diferentes opciones, entre ellas tomar fotos y obtener la ecuación del láser.
def hija():

    # Se define la función "cámara" la cual al llamarla tomará un número determinado de fotos y las guardará.
    def camara():

        cap = cv2.VideoCapture(0)
        i = 0
        while(True):
            # Capture frame-by-frame
```

```
ret, frame = cap.read()

# Display the resulting frame
cv2.imshow('frame', frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
    i += 1

    if i == 1:
        cv2.imwrite("Pattern.jpg", frame)

    if i == 2:
        cv2.imwrite("Calibration001.jpg", frame)
    if i == 3:
        cv2.imwrite("Calibration002.jpg", frame)
    if i == 4:
        cv2.imwrite("Calibration003.jpg", frame)
    if i == 5:
        cv2.imwrite("Calibration004.jpg", frame)
    if i == 6:
        cv2.imwrite("Calibration005.jpg", frame)
    if i == 7:
        cv2.imwrite("Calibration006.jpg", frame)
    if i == 8:
        cv2.imwrite("Calibration007.jpg", frame)
    if i == 9:
        cv2.imwrite("Calibration008.jpg", frame)
    if i == 10:
        cv2.imwrite("Calibration009.jpg", frame)
    if i == 11:
        cv2.imwrite("Calibration010.jpg", frame)
    if i == 12:
        cv2.imwrite("Calibration011.jpg", frame)
    if i == 13:
        cv2.imwrite("Calibration012.jpg", frame)
    if i == 14:
```

```
            break

    # Cuando la función se termine se libera la captura
        cap.release()
        cv2.destroyAllWindows()
    # Se define la función "refresh" la cual refrescará los valores del plano láser que
aparecen en la ventana
    def refresh():

        # Leemos los valores del plano láser del fichero ".txt"
        f = open("Prueba.txt")
        linea = f.readline()
        sep = linea.split()
        A = float(sep[0])

        linea = f.readline()
        sep = linea.split()
        B = float(sep[0])

        linea = f.readline()
        sep = linea.split()
        C = float(sep[0])

        linea = f.readline()
        sep = linea.split()
        D = float(sep[0])
        f.close()

        # Se borran los valores previos
        lblA = Label(ventana2, text='                                    ', font=("Agency FB",
14)).place(x=30, y=125)
        lblB = Label(ventana2, text='                                    ', font=("Agency FB",
14)).place(x=30, y=175)
        lblC = Label(ventana2, text='                                    ', font=("Agency FB",
14)).place(x=30, y=225)
```

```
        lblD = Label(ventana2, text='                                    ', font=("Agency FB",
14)).place(x=30, y=275)

        # Se muestran los nuevos valores
        lblA = Label(ventana2, text=A, font=("Agency FB", 14)).place(x=30, y=125)
        lblB = Label(ventana2, text=B, font=("Agency FB", 14)).place(x=30, y=175)
        lblC = Label(ventana2, text=C, font=("Agency FB", 14)).place(x=30, y=225)
        lblD = Label(ventana2, text=D, font=("Agency FB", 14)).place(x=30, y=275)
    # Crea la ventana hija.
    ventana2 = Toplevel(ventana)

    # Establece el tamaño para la ventana.
    ventana2.geometry("500x300+100+100")

    # Provoca que la ventana tome el focus
    ventana2.focus_set()

    # Deshabilita todas las otras ventanas hasta qué
    # esta ventana sea destruida.
    ventana2.grab_set()

    # Indica que la ventana es de tipo transient, lo que significa
    # que la ventana aparece al frente del padre.
    ventana2.transient(master=ventana)

    f = open("Prueba.txt")
    linea = f.readline()
    sep = linea.split()
    A = float(sep[0])

    linea = f.readline()
    sep = linea.split()
    B = float(sep[0])

    linea = f.readline()
```

```
    sep = linea.split()
    C = float(sep[0])

    linea = f.readline()
    sep = linea.split()
    D = float(sep[0])
    f.close()
    ventana2.title("Scanner 3D UC FBLO")

    # Se declaran todos los botones con las funciones que realiza cada botón.
    btnMover = Button(ventana2, text="Captura", command=camara, font=("Agency
FB", 14), width=10).place(x=300, y=20)
    btnCalibrar = Button(ventana2, text="Calibrar", command=USVoption.calibrate,
font=("Agency FB", 14), width=10).place(x=300, y=70)
    btnSalir = Button(ventana2, text="Refrescar", command=refresh, font=("Agency FB",
14), width=10).place(x=300, y=120)
    btnSalir = Button(ventana2, text="Volver", command=ventana2.destroy,
font=("Agency FB", 14), width=10).place(x=300, y=170)

    # Se escriben todos los textos.
    info = Label(ventana2, text="Presione el boton 'q' para tomar fotos.",
font=("Agency FB", 14)).place(x=50, y=20)
    info2 = Label(ventana2, text="La primera foto sera la foto referencia.",
font=("Agency FB", 14)).place(x=50, y=60)



    lblA = Label(ventana2, text="A", font=("Agency FB", 14)).place(x=30, y=100)
    lblB = Label(ventana2, text="B", font=("Agency FB", 14)).place(x=30, y=150)
    lblC = Label(ventana2, text="C", font=("Agency FB", 14)).place(x=30, y=200)
    lblD = Label(ventana2, text="D", font=("Agency FB", 14)).place(x=30, y=250)

    lblA = Label(ventana2, text=A, font=("Agency FB", 14)).place(x=30, y=125)
    lblB = Label(ventana2, text=B, font=("Agency FB", 14)).place(x=30, y=175)
    lblC = Label(ventana2, text=C, font=("Agency FB", 14)).place(x=30, y=225)
    lblD = Label(ventana2, text=D, font=("Agency FB", 14)).place(x=30, y=275)
```

```
    # Pausa el mainloop de la ventana de donde se hizo la invocación.
    ventana2.wait_window(ventana2)


def escanelect():

    # Crea la ventana hija.
    ventana2 = Toplevel(ventana)

    # Establece el tamano para la ventana.
    ventana2.geometry("600x300+100+100")

    # Provoca que la ventana tome el focus
    ventana2.focus_set()

    # Deshabilita todas las otras ventanas hasta que
    # esta ventana sea destruida.
    ventana2.grab_set()

    # Indica que la ventana es de tipo transient, lo que significa
    # que la ventana aparece al frente del padre.
    ventana2.transient(master=ventana)

    ventana2.title("Scanner 3D UC FBLO")
    global entradaU
    entradaU=StringVar()
    entradaU.set('Output')
    name = Entry(ventana2, textvariable=entradaU).place(x=200,y=20)
    print(str(entradaU.get()))
    info = Label(ventana2, text="Nombre del fichero:", font=("Agency FB",
14)).place(x=10, y=20)
    btnescan = Button(ventana2, text="Escanear", command=superscan, font=("Agency
FB", 14), width=10).place(x=100, y=120)
    btnSalir2 = Button(ventana2, text="Volver", command=ventana2.destroy,
font=("Agency FB", 14), width=10).place(x=100, y=170)
    # Pausa el mainloop de la ventana de donde se hizo la invocacion.
    ventana2.wait_window(ventana2)

def superscan():
    escaneado.inicio(entradaU.get())
# Declaramos las características de la ventana principal


# Declaramos las características de la ventana principal


ventana = Tk()

ventana.geometry("500x300+100+100")

ventana.title("Scanner 3D UC FBLO")
```

```
# Se establecen los botones y sus funciones
btnCalibrar = Button(ventana, text="Calibrar", command=hija, font=("Agency FB", 14), width=10).place(x=300, y=70)
btnEscaneado = Button(ventana, text="Escaneado", command=escaneado.inicio, font=("Agency FB", 14), width=10).place(x=300, y=120)
btnSalir = Button(ventana, text="Volver", command=ventana.destroy, font=("Agency FB", 14), width=10).place(x=300, y=170)

# Se muestra todo el texto deseado en la ventana
info = Label(ventana, text="Escáner 3D", font=("Agency FB", 14)).place(x=50, y=20)
info2 = Label(ventana, text="UC-Francisco de Borja Lanza Ortega", font=("Agency FB", 14)).place(x=50, y=60)


ventana.mainloop()
```

2.5. PROGRAMA DE CALIBRACIÓN

El sistema de reconstrucción 3D por triangulación activa necesita de la ecuación del plano láser ($A * x + B * y + C * z + D = 0$). La explicación matemática del proceso de obtención del plano láser se explica con detenimiento en el apartado “1.7 Modelo de cámara y calibración”. Hay que tener en cuenta que para ejecutar este programa debemos haber tomado previamente las fotos de calibración mediante la interfaz gráfica.

Una vez tomadas las fotos se comprobará mediante el comando “cv2.findChessboardCorners” que dentro de la imagen se encuentra el patrón de calibración, este patrón es un damero de 10x7 donde cada escaque tiene unas dimensiones de 15mm de lado, si lo hay se procede a detectar las esquinas de dicho patrón.

Una vez realizado este proceso con todas las imágenes, se procede a la calibración de la cámara mediante el comando “cv2.calibrateCamera()” la cual nos devolverá la matriz de parámetros intrínsecos, los coeficientes de distorsión, y los valores de rotación y traslación de cada imagen.

Una vez obtenidos todos estos valores se procede a guardarlos en un fichero ".txt" ya que estos valores serán necesarios a la hora de realizar la reconstrucción 3D. Este almacenamiento de variables se hace mediante el siguiente código, que es un ejemplo de cómo se guarda el valor de la variable "refmat" en el fichero "Refmat.txt".

```
outfile = open('Refmat.txt', 'w')
outfile.write(str(refmat))
outfile.close()
```

Una vez realizado esto el programa mostrará todas las imágenes almacenadas, en las cuales deberemos seleccionar la banda láser mediante el ratón. El programa cogerá esta banda y se la pasará en forma de vector línea a línea al programa "maxpixel" para que nos devuelva el píxel más significativo de cada línea. Una vez obtenidos todos los puntos el programa recalculará estas posiciones para corregir el error que ha producido la cámara al captar las imágenes. Para realizar esto el programa usará el comando "cv2.undistortPoints()".

Una vez obtenidos los puntos corregidos se procede a usar el comando "np.linalg.svd()", el cual nos devolverá tres valores "U, S y V". De estas tres variables la última columna de la matriz "V" corresponde a los valores de la ecuación del plano.

Estos valores de la ecuación del plano se guardan en el archivo "Prueba.txt", dejando un salto de línea entre ellos. Para realizar esto se utiliza el siguiente código:

```
outfile = open('Prueba.txt', 'w')
outfile.write(str(A))
outfile.write('\n')
outfile.write(str(B))
outfile.write('\n')
outfile.write(str(C))
outfile.write('\n')
outfile.write(str(D))
outfile.close()
```

Código Python para la calibración del escáner:

```
#-*-coding: utf-8-*-
# Programa en el cual se define una función la cual realizará una calibración del escáner 3D a partir de las imágenes
#  previamente guardadas


# Declaro todas las librerías necesarias:
import numpy as np
import cv2
import maxpixel
import glob

# Declaro la función:
def calibrate():
    # Termination criteria
    = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

    # Preparo los object points, como (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0) y los multiplico por el tamaño de los
    # cuadrados de la imagen de calibración; habrá uno de ellos por cada esquina interna.
    objp = np.zeros((5*8, 3), np.float32)
    objp[:, :2] = np.mgrid[0:8, 0:5].T.reshape(-1, 2)*30

    # Arrays para almacenar los object points y image points de todas las imágenes.
    objpoints = []  # 3d point en real world space
    imgpoints = []  # 2d points en image plane.
    n = 0  # Indicador de la matriz de referencia

    # Guardo en el vector "images" todas las imágenes que usare para calibrar.
    images = glob.glob('*.jpg')
    images2 = []

    # Realizo un bucle que se hará por cada imagen existente.
    for fname in images:
        # Selecciono una imagen y la paso a blanco y negro
        img = cv2.imread(fname)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        # Busco las esquinas del patrón de calibración.
        ret, corners = cv2.findChessboardCorners(gray, (8, 5), None)

        # Si se encuentran, añado object points, image points (después de ajustarlos)
        if ret:
            objpoints.append(objp)

            images2.append(fname)

            # Ajusto la localización de las esquinas
            corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
            imgpoints.append(corners2)
            # Dibujo y muestro las esquinas del patrón de calibración.
            img = cv2.drawChessboardCorners(img, (8, 5), corners2, ret)
```

```
            cv2.imshow('img', img)
            cv2.waitKey(2500)

            # Creo una condición para discernir cual de todas es la imagen de calibración
            if fname == 'Pattern.jpg':
                nref = n
            n += 1
    # Destruyo todas las imágenes
    cv2.destroyAllWindows()

    # Calibro la cámara con todos los puntos previamente obtenidos
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],
None, None)

    # Realizo las operaciones necesarias para obtener la matriz de transformación
mediante los datos de la matriz de
    #  referencia
    vectrot = np.array(rvecs[nref])
    rotmat, denhart = cv2.Rodrigues(vectrot)
    refmat = np.c_[rotmat, tvecs[nref]]
    rotref = rotmat
    transref = tvecs[nref]

    # Guardo las matrices que necesite posteriormente en diferentes archivos ".txt"

    outfile = open('Refmat.txt', 'w')
    outfile.write(str(refmat))
    outfile.close()

    outfile = open('CalibMat.txt', 'w')
    outfile.write(str(mtx))
    outfile.close()

    outfile = open('Dist.txt', 'w')
    outfile.write(str(dist))
    outfile.close()

    # Procedo a declarar diferentes variables que se usarán a continuación

    global ix, iy, fx, fy, selector
    pfinal = []
    n = -1

    drawing = False  # true si mouse es pulsado
    mode = True  # Si True, dibujo rectángulo
    ix, iy = -1, -1


    # Defino la mouse callback function, que será usada para seleccionar de las imágenes
la banda láser.
    def draw_circle(event, x, y, drawing, flags):
        global ix, iy, fx, fy, selector
```

```
        # Si se pulsa el botón izquierdo
        if event == cv2.EVENT_LBUTTONDOWN:
            drawing = True
            ix, iy = x, y

        # Si se ha pulsado el botón izquierdo y se ha soltado empieza a dibujar el rectángulo
        elif event == cv2.EVENT_MOUSEMOVE:
             if drawing:
                  cv2.rectangle(res2, (ix, iy), (x, y), (0, 255, 0), 1)

        # Cuando se suelta el botón izquierdo
        elif event == cv2.EVENT_LBUTTONUP:
                  drawing = False
                  cv2.rectangle(res2, (ix, iy), (x, y), (0, 255, 0), 1)
                  fx, fy = x, y
                  selector = 0
                  selector = 2


    # Defino una función que realice una suma de todos los elementos de un vector
    def sumarLista(lista):
          sumy = 0
          for i in range(0, len(lista)):
              sumy = sumy+lista[i]
          return sumy

    # Una vez definidas las funciones procedo a volver a ir imagen por imagen
seleccionando las bandas láser
    for fname in images2:
        img = cv2.imread(fname)
        z = np.shape(img)

        # Creo una máscara para seleccionar los valores que deseo que pasen el filtro
        b, g, r = cv2.split(img)

        # Defino el rango del color rojo en HSV
        lower_red = 100
        upper_red = 255

        # Establezco los límites de los valores del componente rojo en HSV
        mask = cv2.inRange(r, lower_red, upper_red)

        # Se aplica los límites en la imagen
        res = cv2.bitwise_and(r, r, mask=mask)
        res2 = cv2.bitwise_and(r, r, mask=mask)


        cv2.imshow('frame', img)
        cv2.imshow('mask', mask)
        cv2.imshow('res', res)


        cv2.namedWindow('image')
```

```
        cv2.setMouseCallback('image', draw_circle)


        selector = 1
        while selector == 1:
            cv2.imshow('image', res2)
            k = cv2.waitKey(1) & 0xFF

            if k == 140:
                break

        # Realizo dos condiciones para asegurarme que después de haber llamado a la
función de dibujar; los datos
        # que me devuelve están en la forma que deseo, es decir que sin depender de cómo
se dibuje el rectángulo; tener
        # bien definidas las coordenadas
        if ix > fx:
            change = ix
            ix = fx
            fx = change

        if iy > fy:
            change = iy
            = fy
            fy = change

        # A continuación creo una imagen del mismo tamaño que la original y que esté
completamente llena de ceros.
        #  Después substituyo en la nueva imagen el rectángulo que he dibujado en la
imagen original
        crop_img = res[iy:fy, ix:fx]
        img2 = np.zeros((z[0], z[1]), np.uint8)
        img2[iy:fy, ix:fx] = crop_img

        # Declaro diferentes variables que serán usadas posteriormente
        alfa2 = []
        alfa3 = []
        cimiento = []

        # Creo un bucle por el cual me recorrerá todas las filas de la imagen y si hay algún
numero distinto de cero me
        #  pasará dentro de la condición.
        for fila in range(0, z[0]):
            vector = img2[fila, :]
            cont = sumarLista(vector)

            if cont != 0:
                # Utilizo la función medpixel para obtener el píxel en el cual mediante
aproximación lineal se encuentra
                #  la línea láser
                punto = maxpixel.medpixel(vector)

                # Guardo las coordenadas en diferentes variables
```

```
            alfa2.append([fila])
            alfa3.append([punto])
        # Destruyo todas las ventanas
        cv2.destroyAllWindows()
        n += 1

        # A continuación realizo un serie de operaciones matemáticas por las cuales transformare la disposición de las
        # coordenadas de los puntos para posteriormente poder usar la función undistortPoints
        alfa4 = [alfa3, alfa2]

        tpoints = np.shape(alfa4)
        ambar = np.reshape(np.array(alfa4, dtype=np.float32), (max(tpoints), 1, 2))

        for z in range(0, max(tpoints)):
            ambar[z][0][0] = alfa4[0][z][0]
            ambar[z][0][1] = alfa4[1][z][0]

        corrected = cv2.undistortPoints(ambar, mtx, dist)

        # Una vez corregidos los puntos pixel se los pasare a una nueva variable pero añadiéndole dos coordenadas
        # nuevas; una de ellas siendo "z=0" y la otra "alfa=1".
        for x in range(0, max(tpoints)):
            cimiento.append([corrected[x][0][0], corrected[x][0][1], 0, 1])

        # Obtengo todos los parámetros y valores de la imagen en la que me encuentre
        vectrot = np.array(rvecs[n])
        rotmat, denhart = cv2.Rodrigues(vectrot)

        firstpart = np.c_[np.dot(np.transpose(rotref), rotmat), np.dot(np.transpose(rotref), (tvecs[n]-transref))]
        finalmatrix = np.array([firstpart[0], firstpart[1], firstpart[2], np.array([0, 0, 0, 1])])

        numpuntos = len(cimiento)

        # Transformo los puntos obtenidos y los guardo en una nueva variable
        for roe in range(0, numpuntos):
            pfinal.append(np.dot(finalmatrix, np.transpose(cimiento[roe])))


    # Realizo la función svd de la cual obtendré la ecuación del plano
    U, S, V = np.linalg.svd(pfinal, full_matrices=0, compute_uv=1)

    Plano = []
    ret = len(V)
    for roe in range(0, ret):
        Plano.append(V[roe][ret-1])

    A = Plano[0]
    B = Plano[1]
    C = Plano[2]
```

```
D = Plano[3]

# Guardo la ecuación del plano en un archivo ".txt"
outfile = open('Prueba.txt', 'w')
outfile.write(str(A))
outfile.write('\n')
outfile.write(str(B))
outfile.write('\n')
outfile.write(str(C))
outfile.write('\n')
outfile.write(str(D))
outfile.close()
```

2.6. PROGRAMA DE ESCANEADO

El programa *escaneado.py* es el programa principal del escáner 3D y será el encargado, mediante la ecuación del láser y con las matrices de parámetros intrínsecos y extrínsecos de obtener los puntos de la reconstrucción 3D del objeto real, guardando todos los puntos obtenidos en el archivo “Output.txt”.

Para hacerlo el programa establece al principio la relación entre el ángulo y los pasos del motor, la relación entre los pasos del motor y los milímetros que se eleva el escáner, el número de pasos que cuenta cada salto y el número de saltos o elevaciones que realizará el escáner en cada reconstrucción. Estas variables corresponden a las siguientes asignaciones:

```
relacionrot = -360./(4 * 200) # Relación del paso del motor con los ángulos de rotación. Siendo negativo para indicar que el sentido de giro es horario

relacionalt = 36/50 # Relación del paso del motor con los milímetros de la elevación de la plataforma

pasoalt = 10 # Números de pasos que contiene cada salto

numsaltos = 1 # Número de saltos que el escáner
```

Una vez establecidas estas variables se procede a leer de los correspondientes ficheros “.txt” todas las matrices y vectores que se necesitarán. Mediante la función del programa *leermatrix.py* se obtendrán tanto las matrices de parámetros intrínsecos (mtx) como la de parámetros extrínsecos (refmat), y el vector de los parámetros de distorsión(dist).

```
mtx = leermatrix.lectura(3, 'CalibMat.txt')
dist = leermatrix.lectura(1, 'Dist.txt')
refmat = leermatrix.lectura(3, 'Refmat.txt')
```

Ya por último se recogerá la ecuación del plano láser de su correspondiente archivo ".txt" de la siguiente forma:

```
f = open("Prueba.txt")
linea = f.readline()
sep = linea.split()
A = sep[0]

linea = f.readline()
sep = linea.split()
B = sep[0]

linea = f.readline()
sep = linea.split()
C  = sep[0]

linea = f.readline()
sep = linea.split()
D = sep[0]
f.close()
```

Una vez que se tienen todos los elementos necesarios para el escaneado 3D, se procede a añadir una fila a la matriz resultante del producto de las matrices de parámetros intrínsecos y extrínsecos. Sin embargo debido a que la función de corrección de los puntos nos devuelve los puntos normalizados no es necesario multiplicar por la matriz de parámetros intrínsecos, por ello será la matriz de parámetros extrínsecos a la cual se le añade la ecuación del plano láser.

$$\boldsymbol{I_{4\times4}} * \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \rightarrow \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ A & B & C & D \end{bmatrix}$$

Para la obtención de esta última matriz se utiliza el siguiente código Python:

```
transfmat = np.array([refmat[0], refmat[1], refmat[2], [A, B, C, D]])
invMFinal = cv2.invert(np.array(transfmat, dtype=np.float32))
```

Una vez iniciadas todas las variables se procede a entrar en dos bucles "for" los cuales servirán para regular, el primero la altura de la cámara y el láser y la segunda la rotación de la plataforma de escaneado.

Dentro de estos bucles lo primero que hace el programa es tomar una foto del objeto a reconstruir. Después se pasa por un filtro que toma solo al componente de color rojo de la imagen y les pone un valor mínimo a estos valores. Para esto se utiliza el siguiente código:

```
ret, frame = cap.read()
z = np.shape(frame)


b, g, r = cv2.split(frame)
lower_red = 180
upper_red = 255
mask = cv2.inRange(r, lower_red, upper_red)
res = cv2.bitwise_and(r, r, mask=mask)
```

Una vez obtenida la imagen filtrada se procede a obtener el pixel característico de cada fila de la imagen. Para ello se le pasara un vector que contiene una fila de la imagen filtrada, a la función "medpixel" que está definida en el programa *maxpixel.py* (aparecerá este programa en el Anexo). Esta función nos devolverá la posición del pixel característico.

Una vez obtenidos los píxeles característicos de todas las filas de la imagen se realiza una corrección de la posición de estos píxeles debido a las distorsiones producidas por la cámara. Para ello se utilizará el comando "cv2.undistortPoints()" el cual nos devolverá los puntos corregidos.

Ya obtenidos los pixeles corregidos se pasarán al plano de coordenadas reales mediante la siguiente operación:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ A & B & C & D \end{bmatrix}^{-1} * \alpha * \begin{bmatrix} u_d \\ v_d \\ 1 \\ 0 \end{bmatrix}$$

A estas coordenadas obtenidas se le aplicará una matriz de rotación alrededor del eje y. La matriz de rotación utilizada es:

$$R_y = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}$$

Una vez rotados los pixeles correspondientes, queda modificar el valor de la coordenada “y” dependiendo de la elevación de la cámara y una vez hecho guardar el valor en una variable. Todo este proceso descrito anteriormente se realiza en el programa mediante el siguiente código:

```
ptemp = np.dot(invMFinal[1], corrected2)
# Se divide toda la matriz entre alfa
ptemp2 = ptemp / ptemp[3]
# Se aplica la matriz de giro en sentido antihorario
ptemp[0] = math.cos(math.radians(relacionrot*paso)) * ptemp2[0] +
math.sin(math.radians(relacionrot*paso)) * ptemp2[2]
ptemp[2] = -math.sin(math.radians(relacionrot*paso)) * ptemp2[0] +
math.cos(math.radians(relacionrot*paso)) * ptemp2[2]
ptemp[1] = ptemp2[1] - relacionalt * pasoalt * salto
Pfinal.append([ptemp[0], -ptemp[2], -ptemp[1]])
```

Todo este proceso se repetirá con cada posición de la plataforma y por cada elevación de la cámara.

Finalmente se devolverá la cámara a la posición original y se guardarán todos los puntos obtenidos en el archivo “Output.txt”, para guardarlos se ejecuta el siguiente código:

```
x=np.shape(Pfinal)
outfile = open('Output.txt', 'w')
print(Pfinal[0][0])
print(x)
for pos in range(0,x[0]):
    print(x)
    outfile.write(str(Pfinal[pos]))
    outfile.write('\n')

outfile.close()
```

Código Python para la escaneado:

```
#-*-coding: utf-8-*-
# Programa en el cual se define una función la cual realizará un escaneado del objeto

# Declaro todas las librerías necesarias:
import cv2
import numpy as np
import sumatories
import maxpixel
import leermatrix
import math
import pinrotation
import pinelevacion

# Se define la función de escaneado a la cual se le proporciona el nombre que tomará el fichero en el que se almacenará
# la nube de puntos
def inicio(nombrefichero):
    # Se establecen las variables que se usarán a lo largo del programa
    relacionrot = -360./(4 * 200) # Relación del paso del motor con los ángulos de rotación. Siendo negativo para indicar que el sentido de giro es horario
    relacionalt = 36/50 # Relación del paso del motor con los milímetros de la elevación de la plataforma
    pasoalt = 10 # Números de pasos que contiene cada salto
    numsaltos = 1 # Número de saltos que el escáner
    Pfinal = []

    mtx = leermatrix.lectura(3, 'CalibMat.txt') # Se lee la matriz de parámetros intrínsecos
    dist = leermatrix.lectura(1, 'Dist.txt') # Se lee los coeficientes de distorsión
    refmat = leermatrix.lectura(3, 'Refmat.txt')# Se lee la matriz de parámetros extrínsecos

    # A continuación se lee cada uno de los coeficientes del plano láser
    f = open("Prueba.txt")
    linea = f.readline()
    sep = linea.split()
```

```
    A = sep[0]

    linea = f.readline()
    sep = linea.split()
    B = sep[0]

    linea = f.readline()
    sep = linea.split()
    C  = sep[0]

    linea = f.readline()
    sep = linea.split()
    D = sep[0]
    f.close()

    # Se añade una fila a la matriz de parámetros extrínsecos conteniendo la ecuación del plano láser
    transfmat = np.array([refmat[0], refmat[1], refmat[2], [A, B, C, D]])
    # Se invierte la matriz
    invMFinal = cv2.invert(np.array(transfmat, dtype=np.float32))

    # Se inicia la cámara
    cap = cv2.VideoCapture(0)

    # Se inician los pines de la Raspberry Pi correspondientes a la rotación y a la elevación
    pinrotation.initiation()
    pinelevacion.initiation()

    # Se procede a dos bucles "for"; el primero corresponde a la elevación y el segundo a la rotación
    for salto in range(0, numsaltos):
        pinelevacion.subida(pasoalt * salto)
        for paso in range(0, 801):
            pinrotation.movement(paso)
            pinrotation.movement(paso)
```

```
        # Se toma una foto con la cámara
        ret, frame = cap.read()

        z = np.shape(frame)
        # Se separa la imagen
        b, g, r = cv2.split(frame)

        # Se define el valor máximo y mínimo que tomará el componente rojo
        lower_red = 180
        upper_red = 255

        # Se crea una máscara con los valores previamente determinados
        mask = cv2.inRange(r, lower_red, upper_red)

        # Se aplica la máscara a la imagen
        res = cv2.bitwise_and(r, r, mask=mask)

        alfa2 = []
        alfa3 = []
        z = np.shape(res)

        # Una vez filtrada la imagen la separamos por filas y se obtiene el píxel característico
        for fila in range(0, z[0]):
            cal = sumatories.list(res[fila])
            if cal != 0:
                point = maxpixel.medpixel(res[fila])
                alfa2.append(fila)
                alfa3.append(point)

        # Se almacenan todas las coordenadas de los píxeles característicos en la variable coordsc
        coordsc = [alfa3, alfa2]

        tpoints = np.shape(coordsc)
```

```
        # Si hay mas de 5 pixeles característicos se continua dentro del if; en caso contrario se mueve la plataforma
        # y se vuelve a comenzar
        if max(tpoints) > 5:
            # Se utiliza la función reshape para modificar la forma de la variable coordsc
            coordsc2 = np.reshape(np.transpose(np.array(coordsc, dtype=np.float32)), (max(tpoints), 1, 2))
            # Se corrigen los puntos mediante los coeficientes de distorsión
            corrected = cv2.undistortPoints(coordsc2, np.array(mtx), np.array(dist[0]))

            for x in range(0, max(tpoints)):
                corrected2=np.array([corrected[x][0][0], corrected[x][0][1], 1, 0])
                ptemp = np.dot(invMFinal[1], corrected2)
                # Se divide toda la matriz entre alfa
                ptemp2 = ptemp / ptemp[3]
                # Se aplica la matriz de giro en sentido antihorario
                ptemp[0] = math.cos(math.radians(relacionrot*paso))*ptemp2[0]+math.sin(math.radians(relacionrot*paso))*ptemp2[2]
                ptemp[2] = -math.sin(math.radians(relacionrot*paso))*ptemp2[0]+math.cos(math.radians(relacionrot*paso))*ptemp2[2]
                ptemp[1] = ptemp2[1] - relacionalt * pasoalt * salto
                Pfinal.append([ptemp[0], -ptemp[2], -ptemp[1]])

    # Se devuelve la plataforma de la cámara a su posición inicial
    pinelevacion.bajada(pasoalt * salto * (numsaltos - 1))
    # Se liberan los pines de la Raspberry Pi y la cámara.
    pinelevacion.liberation()
    pinrotation.liberation()
    cap.release()

    # Se guardan los puntos calculados en el fichero txt con el nombre previamente proporcionado al programa
```

```
x = np.shape(Pfinal)
outfile = open(str(nombrefichero)+'.txt', 'w')
print(str(nombrefichero))
for pos in range(0, x[0]):
    outfile.write(str(Pfinal[pos][0]))
    outfile.write('    ')
    outfile.write(str(Pfinal[pos][1]))
    outfile.write('    ')
    outfile.write(str(Pfinal[pos][2]))
    outfile.write('\n')

outfile.close()
```

2.7. Resultados, Conclusiones y posibles mejoras

2.7.1. Resultados

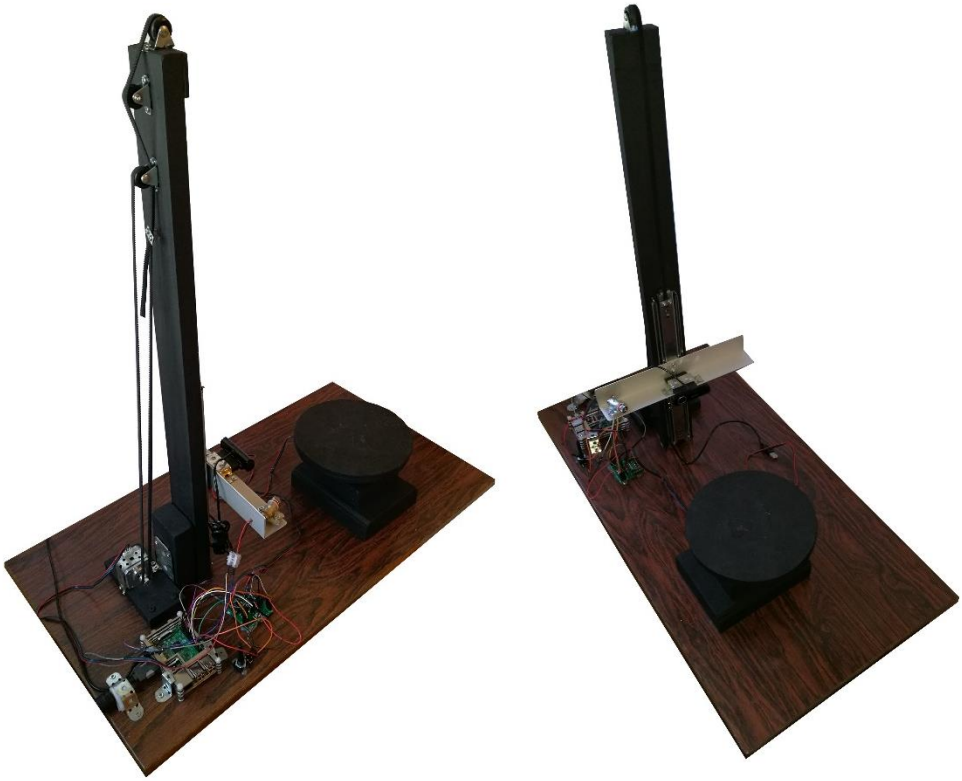
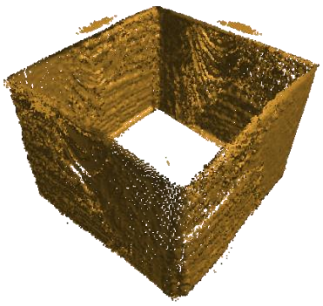
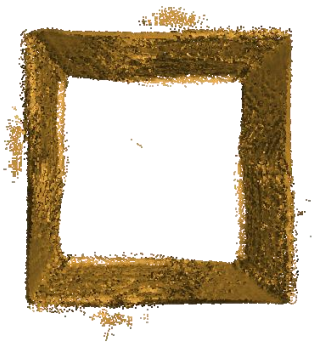
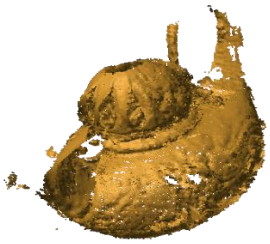


Figura 2.15; Prototipo de escáner 3D.

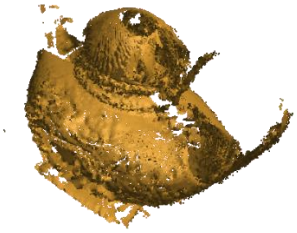
Se ha diseñado un escáner 3D capaz de realizar su propia calibración y la reconstrucción de un objeto real. Cuando se desee reconstruir un objeto se deberá calibrar la cámara y una vez calibrada se seleccionará la opción de escaneado del programa y este nos pedirá que se introduzca el nombre que se desee que tenga el archivo de la reconstrucción. Este archivo es un documento de texto ".txt" en formato "xyz".

El escáner ha sido probado con varios objetos de diferentes geometrías para comprobar el resultado de la reconstrucción. A continuación se presentan los objetos escaneados y sus correspondientes reconstrucciones:

Objeto a reconstruir:	Reconstrucción del objeto:
	 
 	 





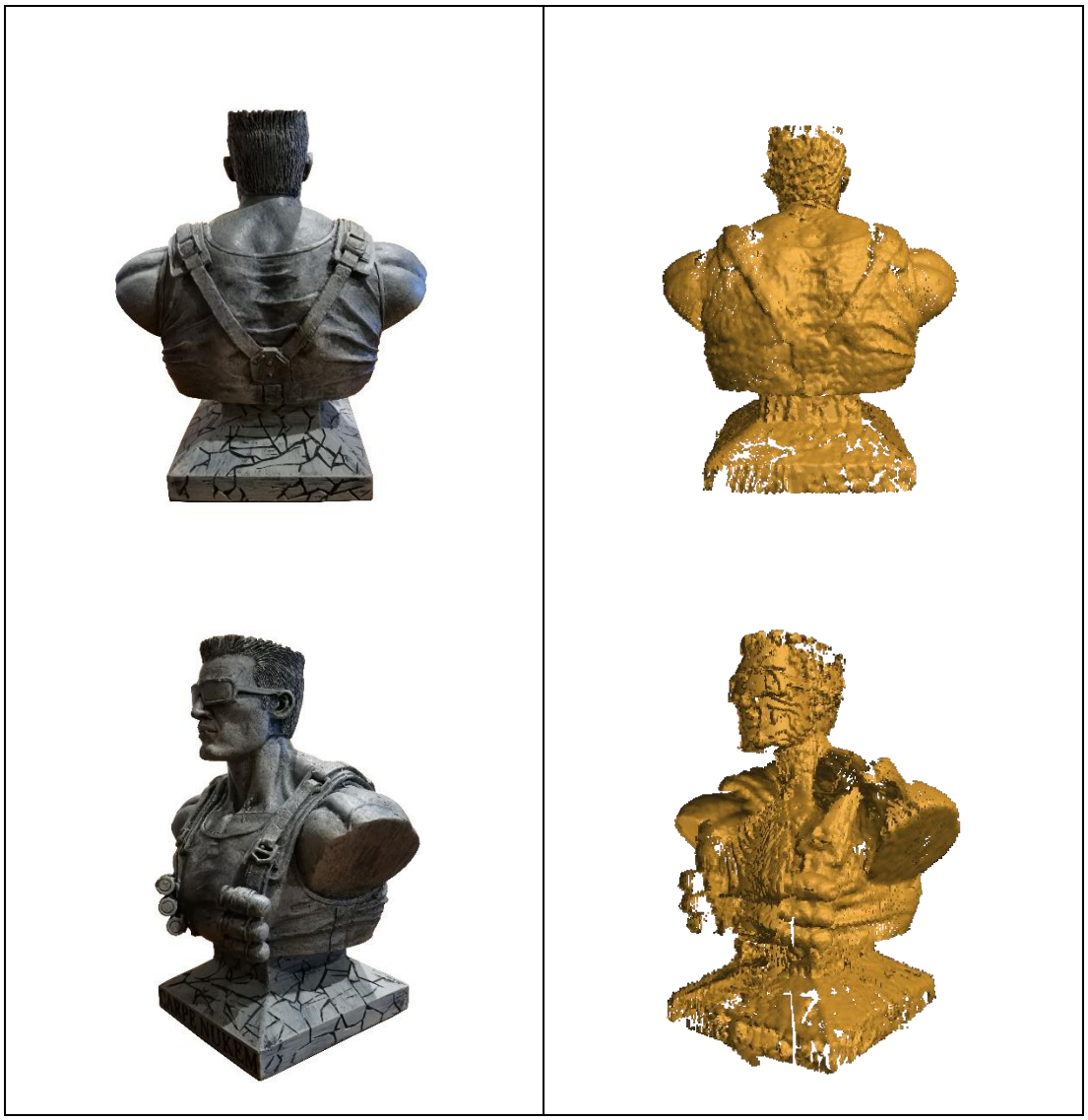


Tabla 16; Resultados de escaneado.

2.7.2. Conclusiones

Con los resultados obtenidos se puede llegar a la conclusión de que se ha logrado diseñar y construir un escáner 3D de bajo coste con una buena resolución. Sin embargo este escáner 3D no es comerciable debido a la duración del tiempo de escaneado. Un escáner 3D de los que están comercializados tarda alrededor de 3 minutos en realizar una reconstrucción, mientras que el escáner descrito en el proyecto tarda alrededor de 20 minutos. Esto es debido a que los escáneres comerciales utilizan un ordenador que cuenta con un procesador de gran potencia, mientras que el escáner diseñado utiliza una placa Raspberry Pi, la cual tiene una velocidad de procesado de 1.2 GHz.

Otra desventaja de este escáner es la falta de un software para unificar puntos redundantes de la reconstrucción, lo que da lugar a unas reconstrucciones más inexactas, por el contrario frente a estas desventajas el escáner diseñado presenta un coste más favorable que sus homólogos comerciales.

2.7.3. Posibles mejoras

Las posibles líneas de mejora de este escáner para mejorar su rendimiento y por tanto poderse comercializar son:

- Rediseño de la aplicación para que la Raspberry Pi se encargue del manejo de los motores y de un ordenador con una capacidad de procesamiento mayor para el procesado de las imágenes.
- Modificación del código para que el usuario tenga la opción de poder cambiar la configuración del escáner, de escaneado vertical a horizontal y viceversa.
- Debido a las características de la Raspberry Pi y las de los nuevos portátiles, que cuentan cada vez con más núcleos, es aconsejable modificar el código con una programación en paralelo, ya que de esta forma se logrará reducir una gran cantidad de tiempo en el procesado de las imágenes.
- Añadir software para la corrección de puntos 3D redundantes en la reconstrucción.

DOCUMENTO 2

ANEXO

ÍNDICE

1. DATASHEETS

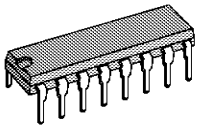
1.1. DATASHEET l293b

SGS-THOMSON MICROELECTRONICS

L293B
L293E

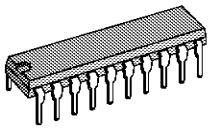
PUSH-PULLFOURCHANNELDRIVERS

- OUTPUTCURRENT1APERCHANNEL
- PEAKOUTPUTCURRENT2APERCHANNEL (nonrepetitive)
- INHIBITFACILITY
- HIGHNOISEIMMUNITY
- SEPARATELOGICSUPPLY
- OVERTEMPERATUREPROTECTION
-



DIP16

ORDERINGNUMBER: L293B



POWERDIP(16+2+2)

ORDERINGNUMBER: L293E

DESCRIPTION

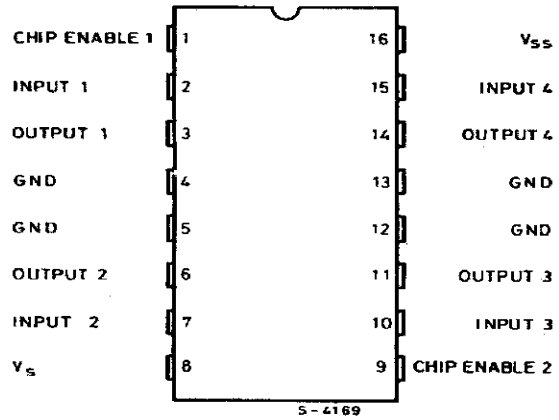
TheL293BandL293Earequadpush-pulldrivers capableofdeliveringoutputcurrentsto1Aperchannel.EachchanneliscontrolledbyaTTL-compatible logicinputandeachpairofdrivers(afullbridge)is equippedwithaninhibitinputwhichturnsoffallfour transistors.Aseparatesupplyinputisprovidedfor thelogicsothatitmayberunoffalowervoltageto reducedissipation.

Additionally,theL293Ehasexternalconnectionof sensingresistors,forswitchmodecontrol.

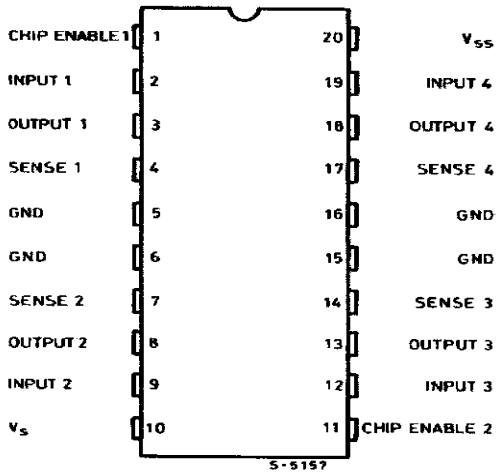
TheL293BandL293Earepackagein16and20-pin plasticDIPsrespectively;bothusethefourcenter pinstoconductheattotheprintedcircuitboard.

PINCONNECTIONS

DIP16-L293B



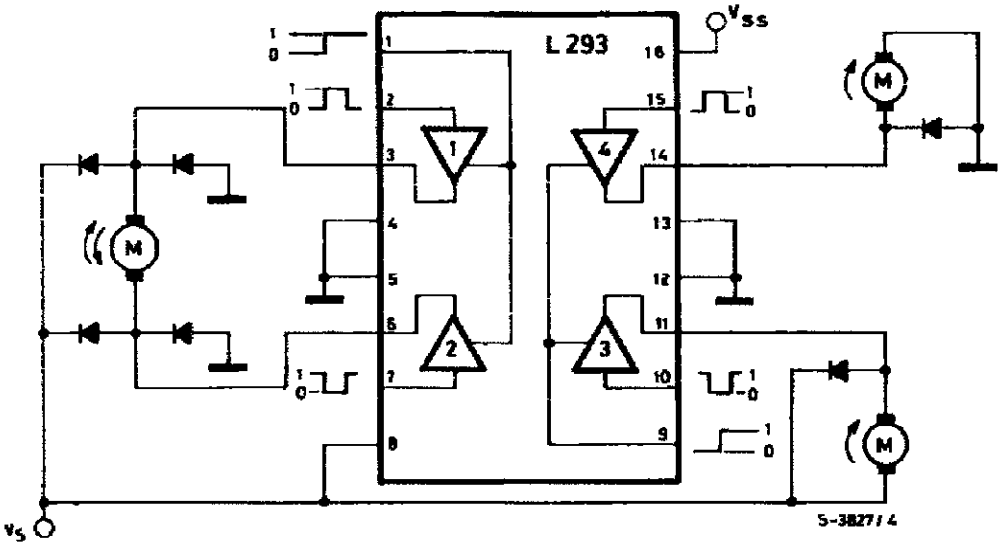
POWERDIP(16+2+2)-L293E



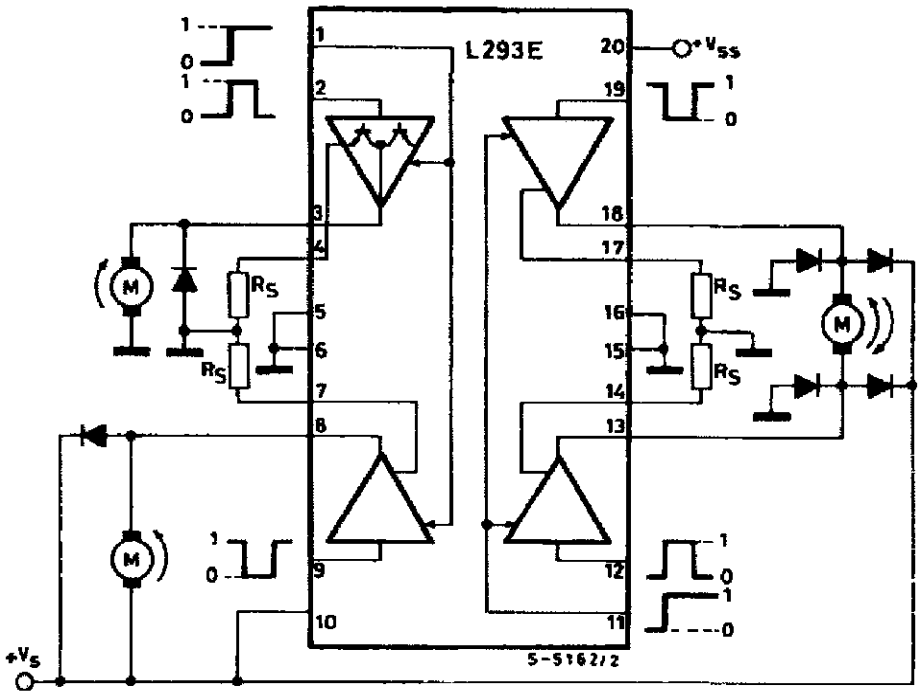
April1993

BLOCKDIAGRAMS

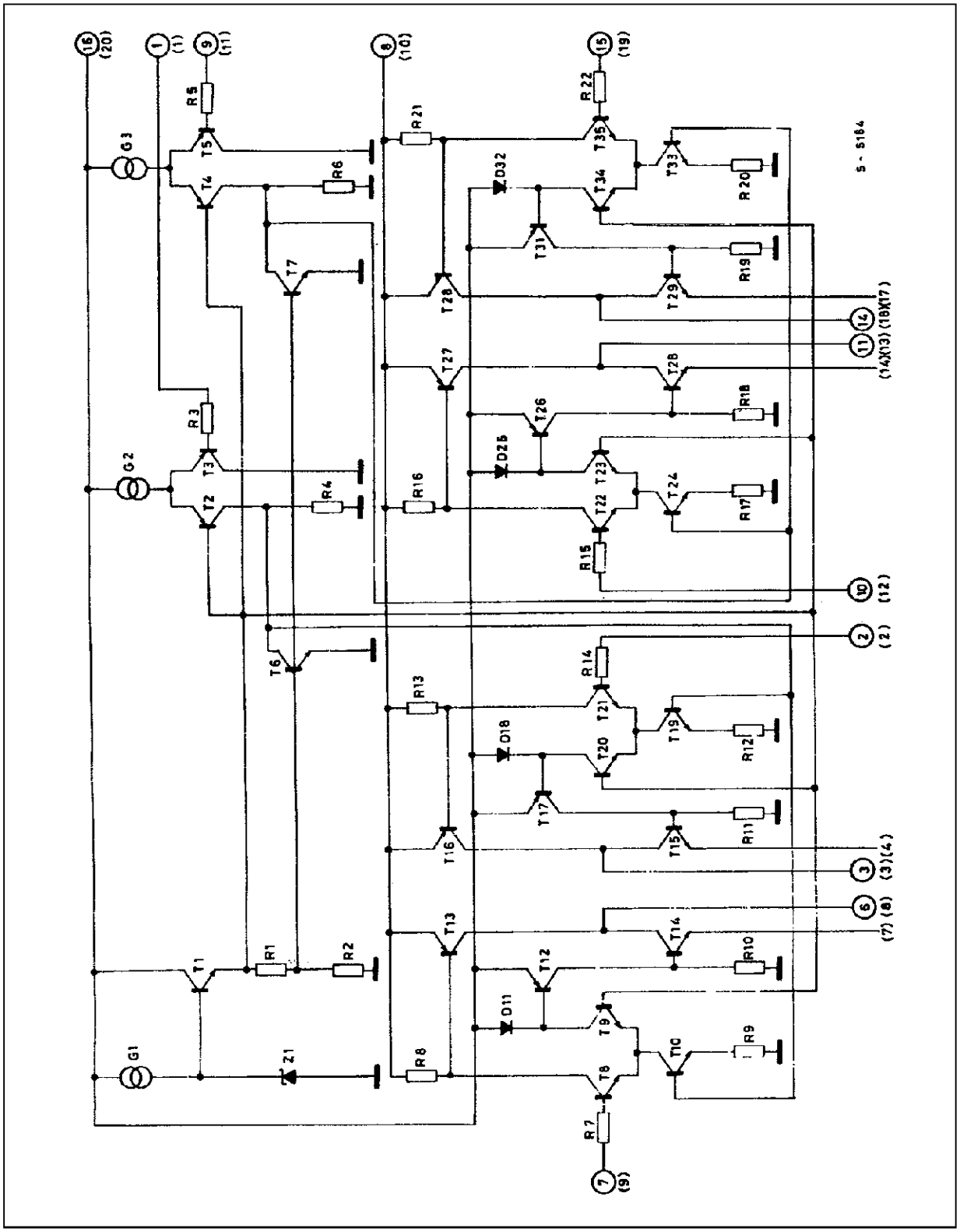
DIP16-L293B



POWERDIP(16+2+2)-L293E



SCHEMATICDIAGRAM



(*) IntheL293thesepointsarenotexternallyavailable.Theyareinternallyconnectedtotheground(substrate).
OPinsofL293 () PinsofL293E.

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_s	Supply Voltage	36	V
V_{ss}	Logic Supply Voltage	36	V
V_i	Input Voltage	7	V
V_{inh}	Inhibit Voltage	7	V
I_{out}	Peak Output Current (non repetitive t = 5ms)	2	A
P_{tot}	Total Power Dissipation at $T_{ground\text{-}pins}$ = 80°C	5	W
T_{stg}, T_j	Storage and Junction Temperature	–40 to +150	°C

THERMAL DATA

Symbol	Parameter		Value	Unit
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	Max.	14	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	80	°C/W

ELECTRICAL CHARACTERISTICS

For each channel, V_S = 24V, V_{SS} = 5V, T_{amb} = 25°C, unless otherwise specified

Symbol	Parameter	Test Conditions	Min.	TYp.	Max.	Unit
V_s	Supply Voltage		V_{ss}		36	V
V_{ss}	Logic Supply Voltage		4.5		36	V
I_s	Total Quiescent Supply Current	V_i = L I_o = 0 V_{inh} = H V_i = H I_o = 0 V_{inh} = H V_{inh} = L		2 16	6 24 4	mA
I_{ss}	Total Quiescent Logic Supply Current	V_i = L I_o = 0 V_{inh} = H V_i = H I_o = 0 V_{inh} = H V_{inh} = L		44 16 16	60 22 24	mA
V_{iL}	Input Low Voltage		-03.		1.5	V
V_{iH}	Input High Voltage	$V_{SS} \leq 7V$ $V_{ss} > 7V$	2.3 2.3		V_{ss} 7	V
I_{iL}	Low Voltage Input Current	V_{il} = 1.5V			-10	µA
I_{iH}	High Voltage Input Current	$2.3V \leq V_{IH} \leq V_{ss} - 0.6V$		30	100	µA
V_{inhL}	Inhibit Low Voltage		-0.3		1.5	V
V_{inhH}	Inhibit High Voltage	$V_{SS} \leq 7V$ $V_{ss} > 7V$	2.3 2.3		V_{ss} 7	V
I_{inhL}	Low Voltage Inhibit Current	V_{inhL} = 1.5V		-30	-100	µA
I_{inhH}	High Voltage Inhibit Current	$2.3V \leq V_{inhH} \leq V_{ss} - 0.6V$			±10	µA
V_{CEsatH}	Source Output Saturation Voltage	I_o = -1A		1.4	1.8	V
V_{CEsatL}	Sink Output Saturation Voltage	I_o = 1A		1.2	1.8	V
V_{SENS}	Sensing Voltage (pins 4, 7, 14, 17) (**)				2	V
t_r	Rise Time	0.1 to 0.9 V_o (*)		250		ns
t_f	Fall Time	0.9 to 0.1 V_o (*)		250		ns
t_{on}	Turn-on Delay	0.5 V_i to 0.5 V_o (*)		750		ns
t_{off}	Turn-off Delay	0.5 V_i to 0.5 V_o (*)		200		ns

* See figure 1
** Referred to L293E

TRUTH TABLE

V_i (each channel)	V_o	V_{inh} (∞)
H	H	H
L	L	H
H	X (°)	L
L	X (°)	L

(*) High output impedance
(**) Relative to the considerate channel

Figure1: SwitchingTimers

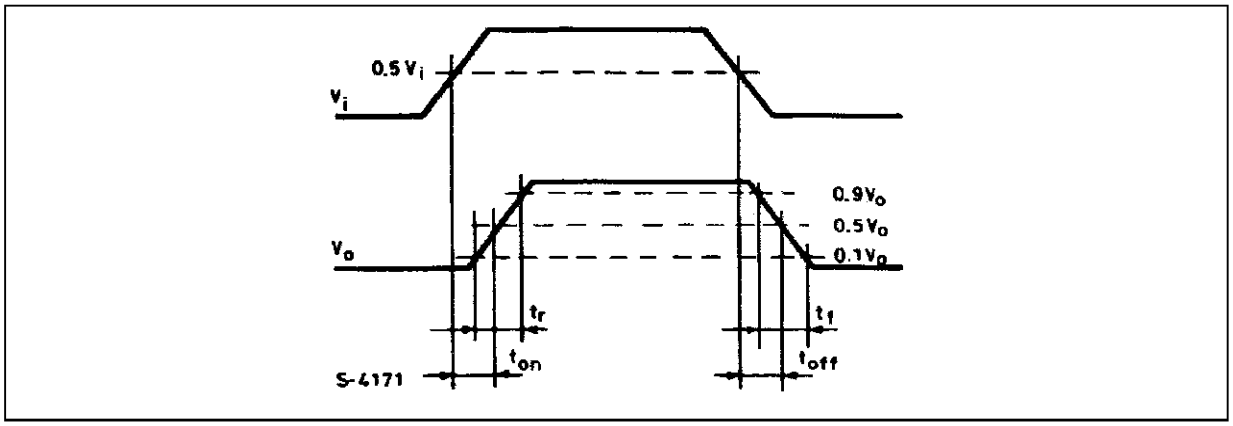


Figure2: SaturationvoltageversusOutput Current

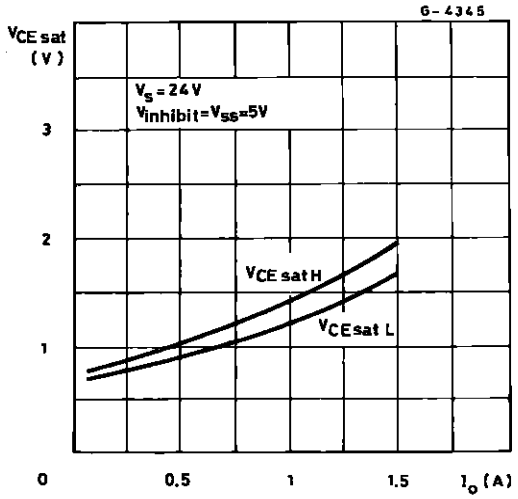


Figure3: SourceSaturationVoltageversus AmbientTemperature

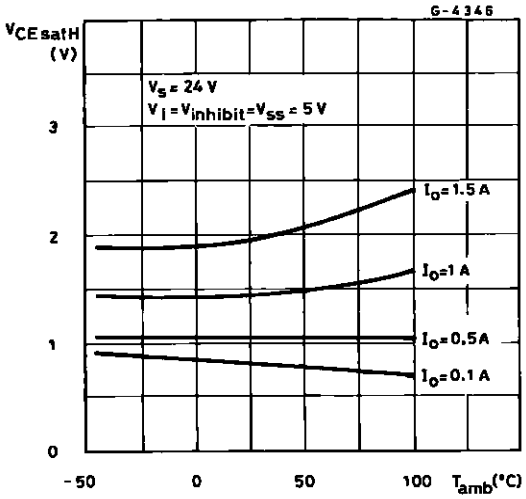


Figure4: SinkSaturationVoltageversus AmbientTemperature

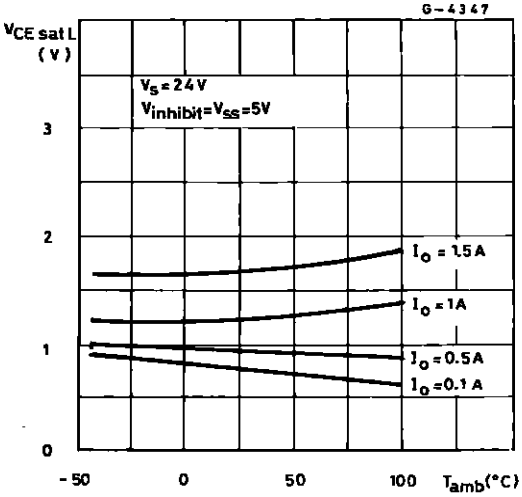


Figure5: QuiescentLogicSupplyCurrent versusLogicSupplyVoltage

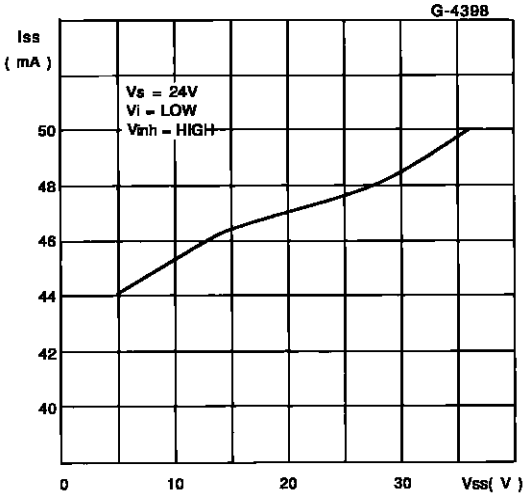


Figure6: OutputVoltageversus InputVoltage

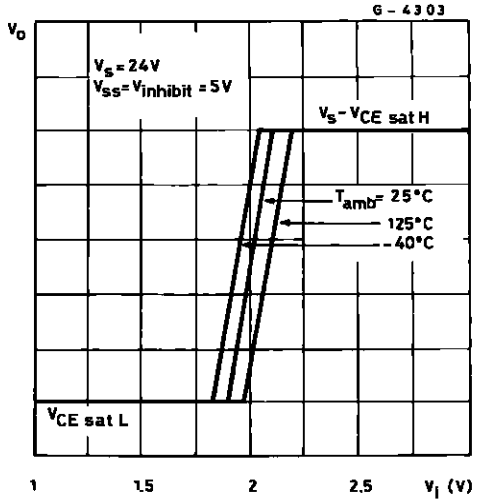
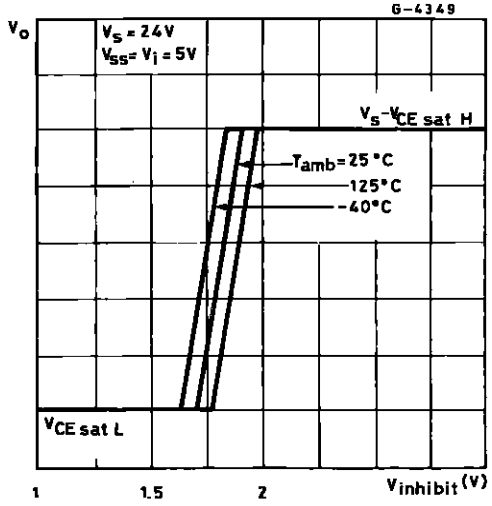
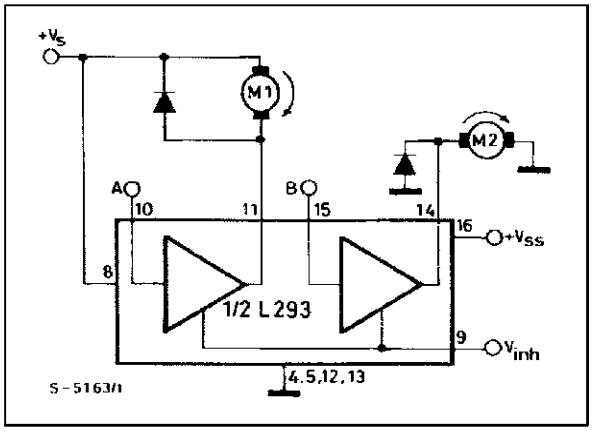


Figure7: OutputVoltageversus InhibitVoltage



APPLICATIONINFORMATION

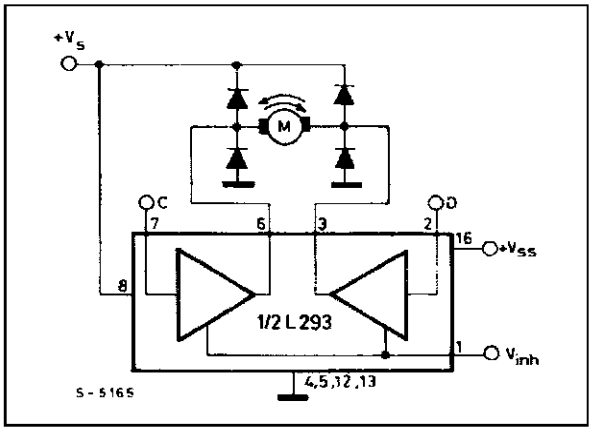
Figure8: DCMotorControls (withconnectiontogroundand tothesupplyvoltage)



V_{inh}	A	M1	B	M2
H	H	FastMotorStop	H	Run
H	L	Run	L	FastMotorStop
L	X	FreeRunning MotorStop	X	FreeRunning MotorStop

L=Low H=High X=Don'tCare

Figure9: BidirectionalDCMotorControl



Inputs	Function	
V_{inh} = H	C=H;D=L	TurnRight
	C=L;D=H	TurnLeft
	C=D	FastMotorStop
V_{inh} = L	C=X;D=X	FreeRunning MotorStop

L=Low H=High X=Don'tCare

Figure10: BipolarSteppingMotorControl

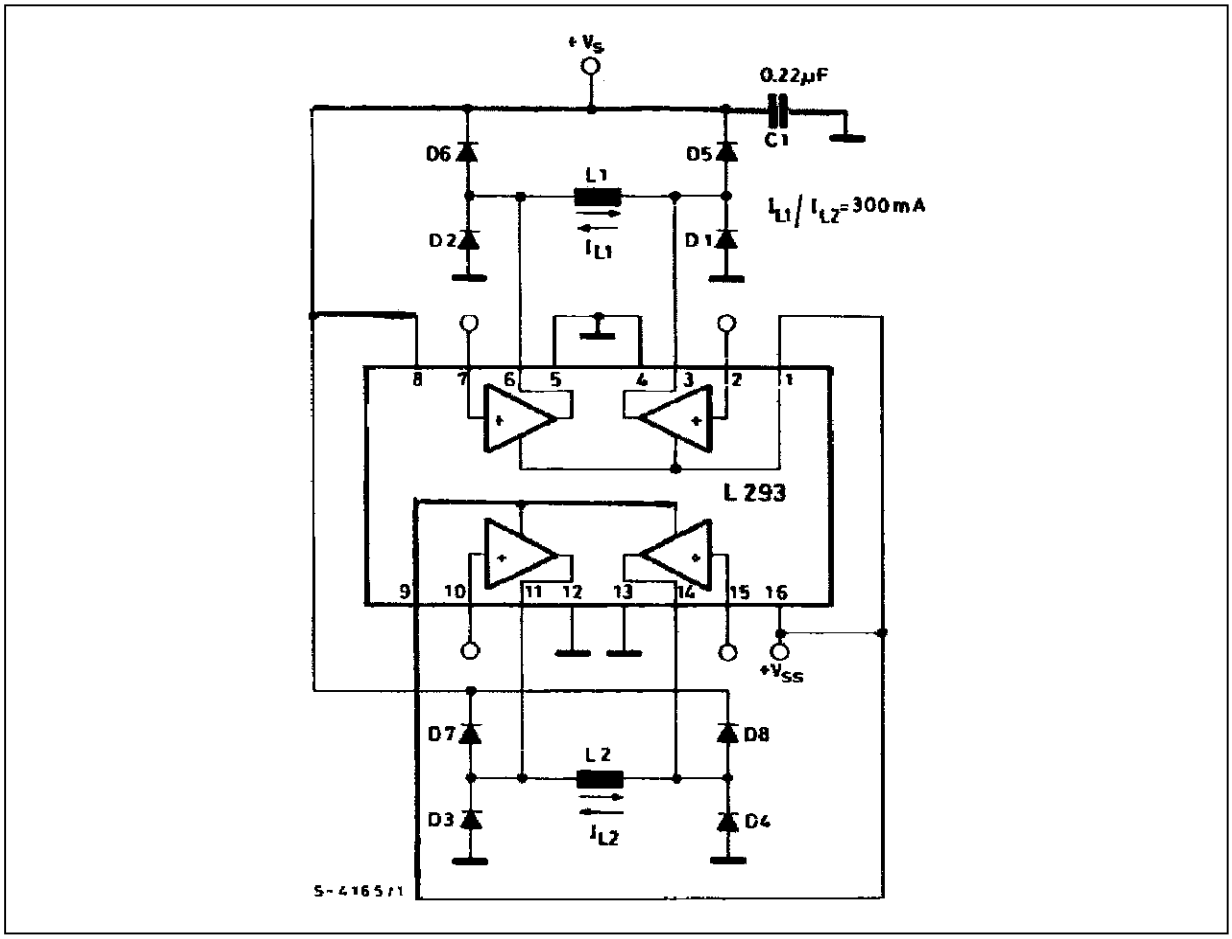
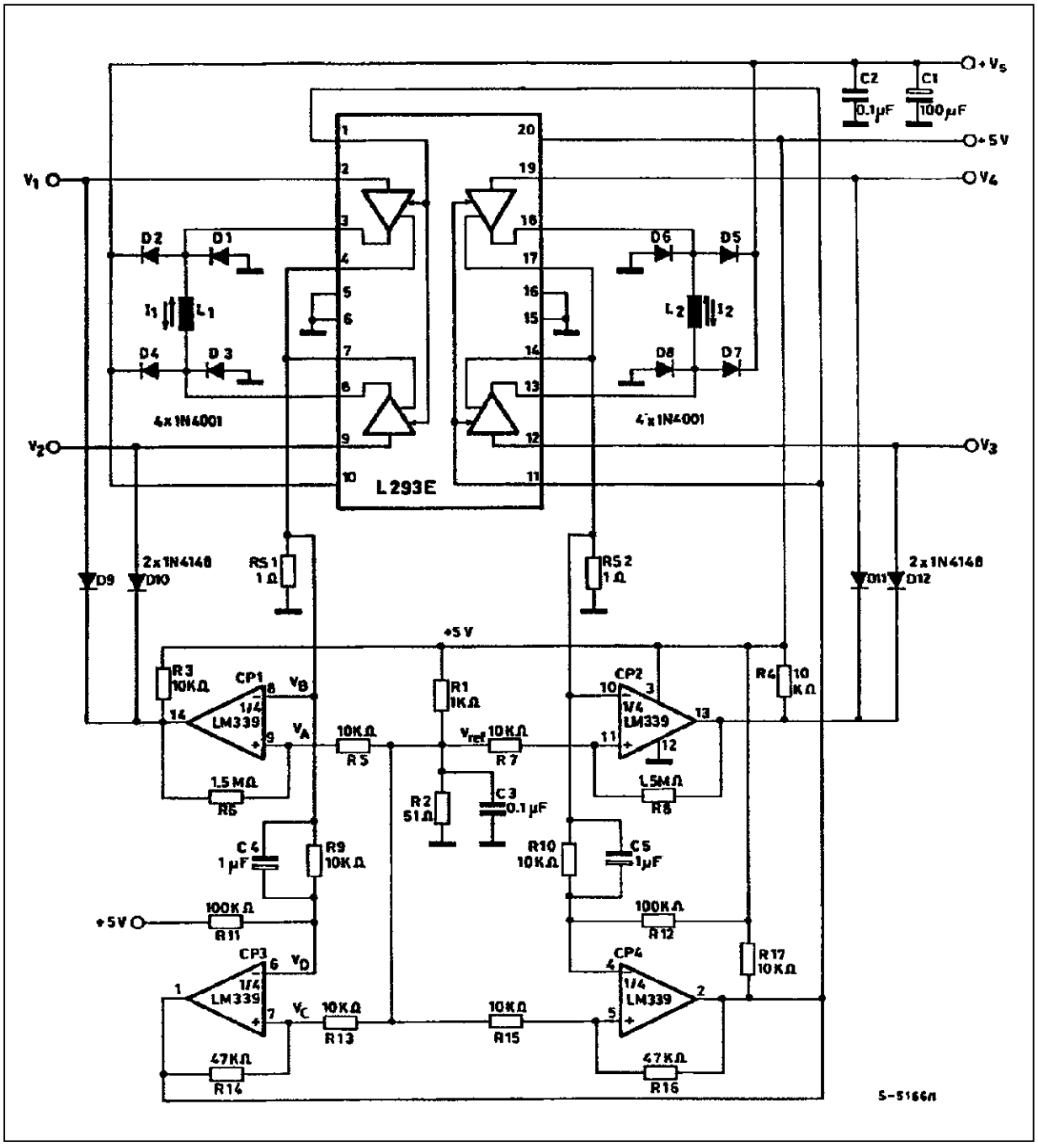


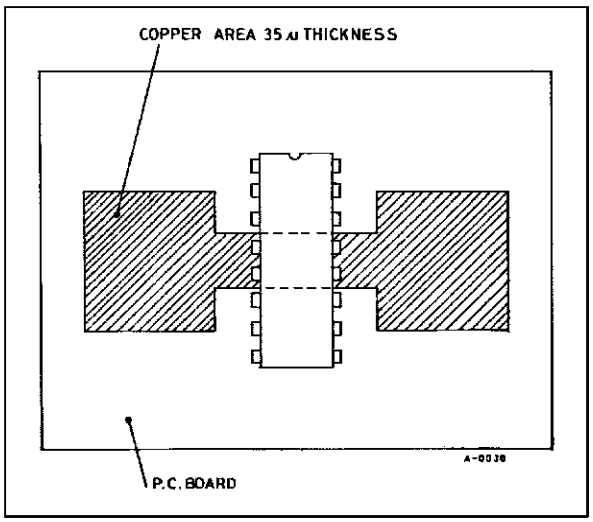
Figure11: SteppingMotorDriverwithPhaseCurrentControlandShortCircuitProtection



MOUNTINGINSTRUCTIONS

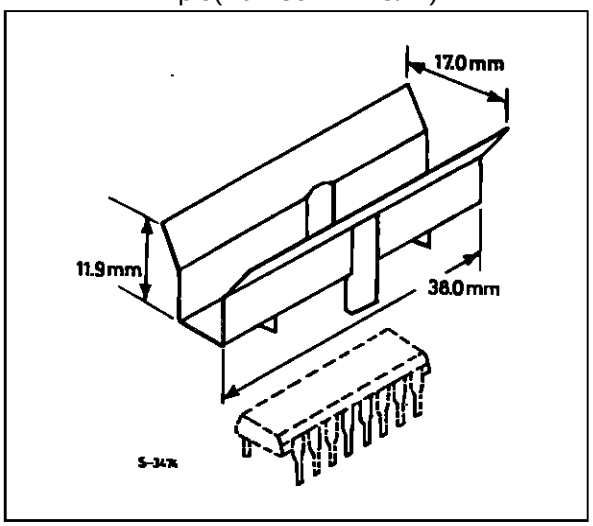
heR $_{hj\text{-}amb}$ ftheL293BandtheL293Ecanbere-ducedbysolderingtheGNDpinstoasuitablecop-erareaoftheprintedcircuitboardasshowninfig-re12ortoanexternalheatsink(figure13).

Figure12: xampleofP.C.BoardCopper AreawhichisUsedasHeatsink



Duringsolderingthepinstemperaturemustnotex-eed260 Candthesolderingtimemustnotbe ongerthan12seconds.
Theexternalheatsinkorprintedcircuitcopperarea mustbeconnectedtoelectricalground.

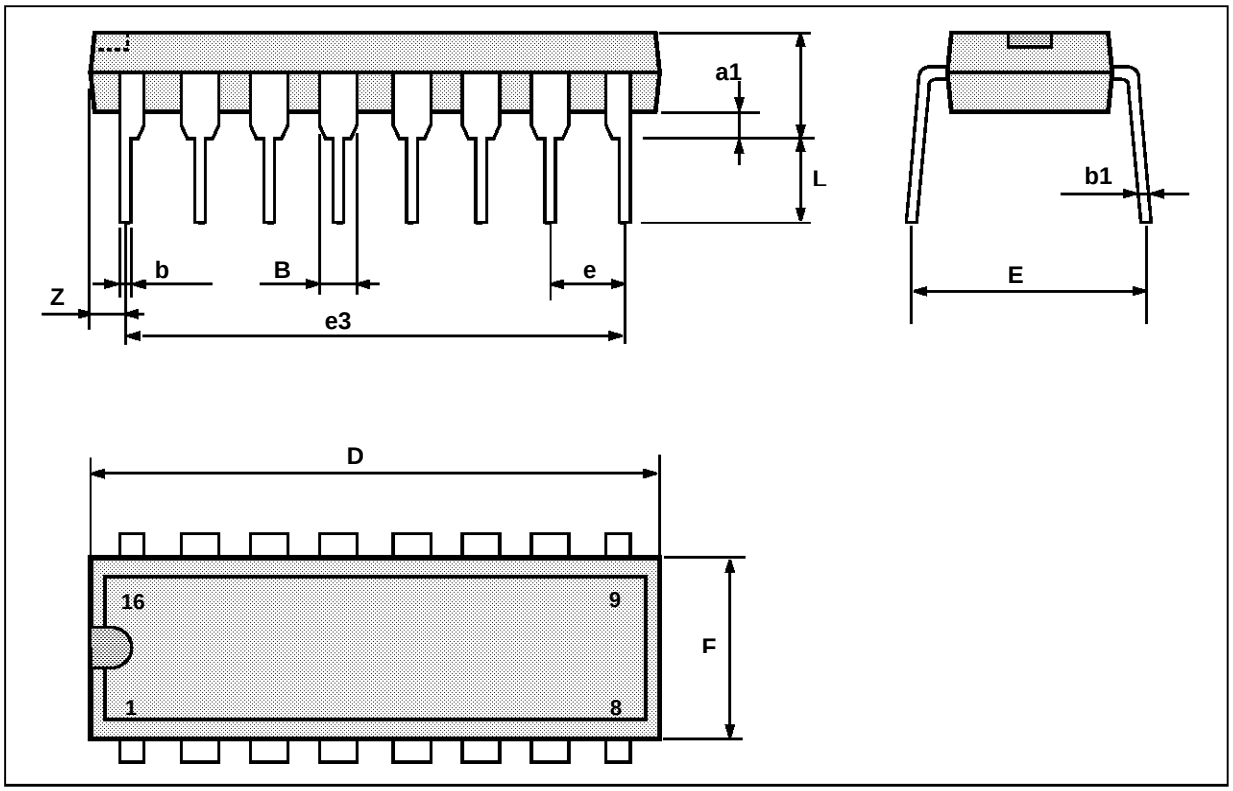
Figure13: ExternalHeatsinkMountingEx-mple(Rth=30 C/W)



DIP16PACKAGEMECHANICALDATA

Dimensions	Millimeters			Inches		
	Min.	Typ.	Max.	Min.	Typ.	Max.
a1	0.51			0.020		
B	0.77		1.65	0.030		0.065
b		0.5			0.020	
b1		0.25			0.010	
D			20			0.787
E		8.5			0.335	
e		2.54			0.100	
e3		17.78			0.700	
F			7.1			0.280
i			5.1			0.201
L		3.3			0.130	
Z			1.27			0.050

DIP16W.TBL

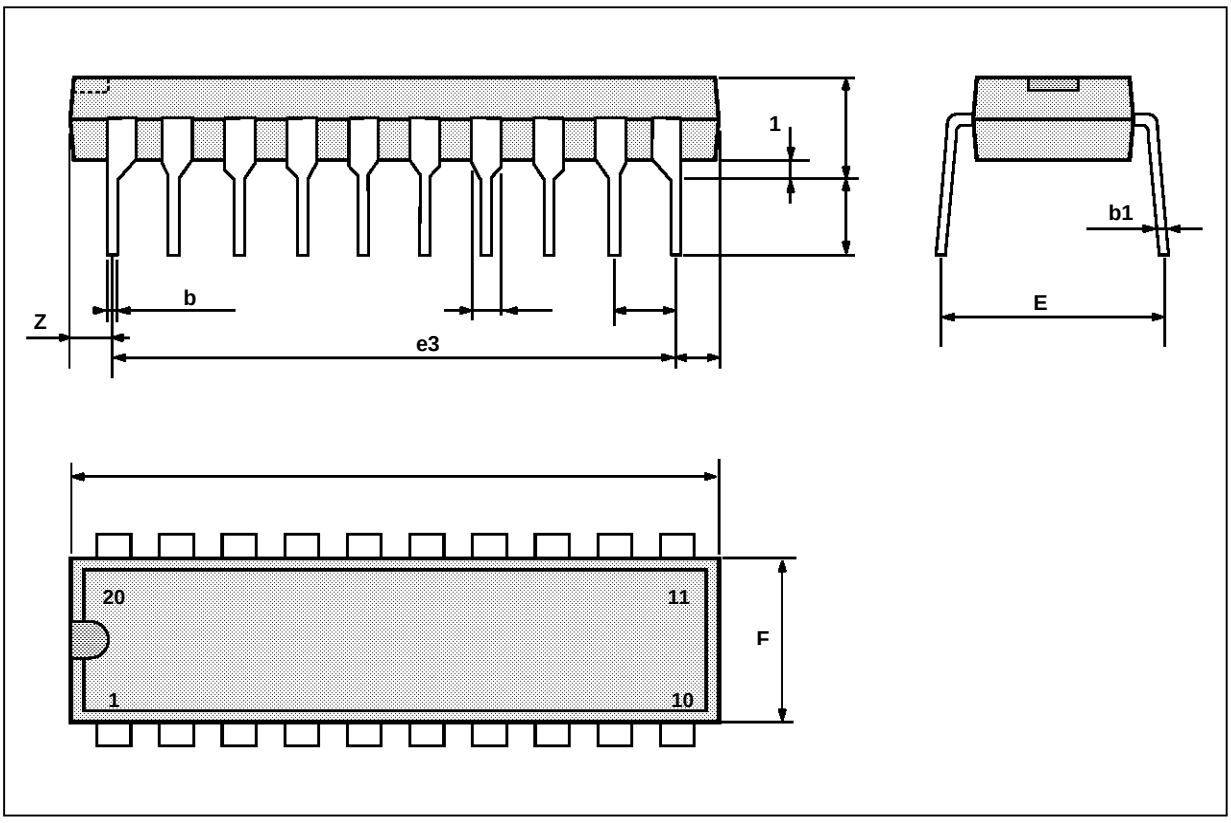


PMDIP16W.EPS

POWERDIP(16+2+2)PACKAGEMECHANICALDATA

Dimensions	Millimeters			Inches		
	Min.	Typ.	Max.	Min.	Typ.	Max.
a1	0.51			0.020		
B	0.85		1.4	0.033		0.055
b		0.5			0.020	
b1	0.38		0.5	0.015		0.020
D			24.8			0.976
E		8.8			0.346	
e		2.54			0.100	
e3		22.86			0.900	
F			7.1			0.280
i			5.1			0.201
L		3.3			0.130	
Z			1.27			0.050

DIP20W.TBL



PMDIP20WEP

1.2. DATASHEET 1N4007S

DEC

1N4001S THRU 1N4007S	CURRENT 1.0 Ampere VOLTAGE 50 to 1000 Volts

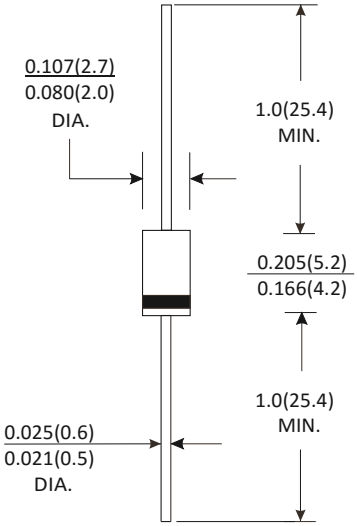
Features

A-405

· The plastic package carries Underwrites Laboratory Flammability Classification 94V-0
· Construction utilizes void-free molded plastic technique
· Low reverse leakage
· High forward surge current capability
· High reliability

Mechanical Data

· Case : A-405 molded plastic body · Terminals : Lead solderable per MIL-STD-750,
method 2026
· Polarity : Color band denotes cathode end
· Mounting Position : Any
· Weight : 0.008 ounce, 0.23 gram



Dimensions in inches and (millimeters)

Maximum Ratings And Electrical Characteristics

(Ratings at 25℃ ambient temperature unless otherwise specified, Single phase, half wave 60Hz, resistive or inductive load. For capacitive load, derate by 20%)

		Symbols	1N 4001 S	1N 4002 S	1N 4003 S	1N 4004 S	1N 4005 S	1N 4006 S	1N 4007 S	Units
Maximum recurrent peak reverse voltage		V_{RRM}	50	100	200	400	600	800	1000	Volts
Maximum RMS voltage		V_{RMS}	35	70	140	280	420	560	700	Volts
Maximum DC blocking voltage		V_{DC}	50	100	200	400	600	800	1000	Volts
Maximum average forward rectified current 0.375"(9.5mm) lead length T_A=25℃		$I_{(AV)}$	1.0							Amp
Peak forward surge current 8.3ms half sing wave superimposed on rated load (JEDEC method) at T_A=75℃		I_{FSM}	30.0							Amps
Maximum instantaneous forward voltage at 1.0A		V_F	1.1							Volts
Maximum reverse current at rated voltage	T_A=25℃	I_R	5.0							μA
	T_A=100℃		50.0							
Typical thermal resistance (Note 2)		$R\theta_{JA}$	50.0							℃/W
Typical junction capacitance (Note 1)		C_J	15.0							pF
Operating and Storage temperature Range		T_J T_{STG}	-50 to +175							℃

Notes:

(1) Measured at 1MHz and applied reverse voltage of 4.0V dc.

(2) Thermal resistance from junction to ambient and from junction to lead at 0.375"(9.5mm) lead length, p.c.b. mounted

FIG.1-FORWARD CURRENT DERATING CURVE

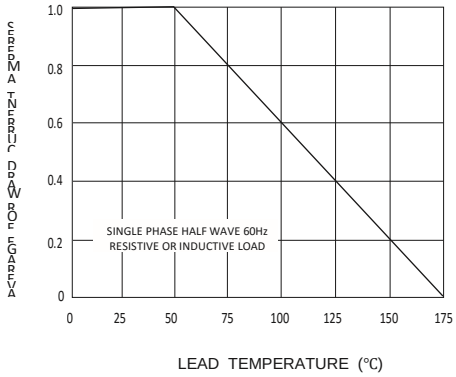
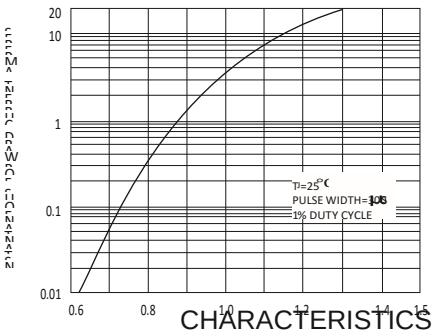


FIG.2-TYPICAL INSTANTANEOUS FORWARD CHARACTERISTICS



DEC

RATINGS AND CHARACTERISTIC CURVES 1N4001S THRU 1N4007S

FIG.3-MAXIMUM NON-REPETITIVE PEAK FORWARD SURGE CURRENT

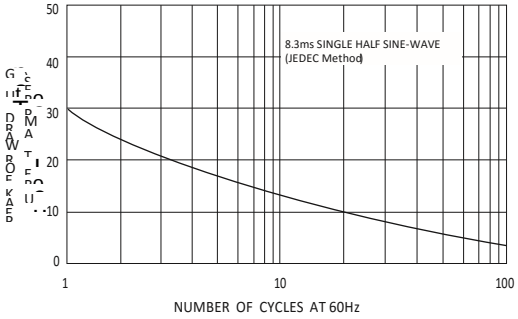


FIG.4-TYPICAL REVERSE CHARACTERISTICS

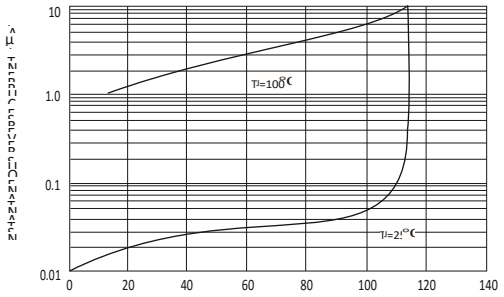
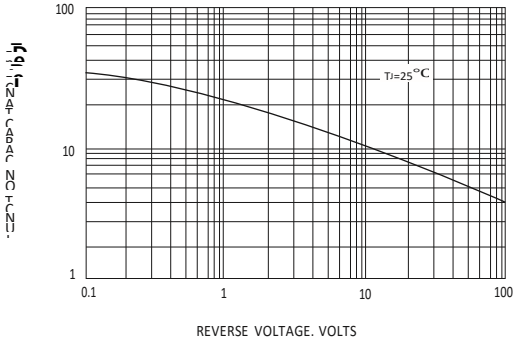


FIG.5-TYPICAL JUNCTION CAPACITANCE



INSTANTANEOUS FORWARD VOLTAGE (VOLTS)

PERCENT OF RATED PEAK REVERSE VOLTAGE (%)

2. PROGRAMAS DEL ESCÁNER 3D

A continuación se exponen todos los programas complementarios para el correcto funcionamiento del escáner 3D:

2.1. PINELEVACION.PY

Programa pinelevacion.py

```
#-*-coding: utf-8-*-
# Programa en el cual se definen una serie de funciones para controlar la rotación de la plataforma del escáner.
# Este  programa cuenta con cuatro funciones:
# initiation->Inicia y define los pines que controlan el motor vertical.
# subida->Desplazamiento de un número de pasos en sentido ascendente.
# bajada-> Desplazamiento de un número de pasos en sentido descendiente.
# liberation->Liberación de los pines.

import RPi.GPIO as GPIO #importación de la librería y asignación a "GPIO".
import time # librería utilizada para los delays.



def initiation():
    # Se establece el sistema de numeración que será empleado, en este caso BCM.
    GPIO.setmode(GPIO.BCM)

    # Se configuran los pines necesarios como salidas.
    GPIO.setup(29, GPIO.OUT)
    GPIO.setup(31, GPIO.OUT)
    GPIO.setup(33, GPIO.OUT)
    GPIO.setup(35, GPIO.OUT)

def subida(desplazamiento):

    for paso in range(0,desplazamiento):
```

```
        GPIO.output(29, GPIO.LOW)
        GPIO.output(31, GPIO.HIGH)
        GPIO.output(33, GPIO.HIGH)
        GPIO.output(35, GPIO.LOW)
        time.sleep(0.1)


        GPIO.output(29, GPIO.LOW)
        GPIO.output(31, GPIO.HIGH)
        GPIO.output(33, GPIO.LOW)
        GPIO.output(35, GPIO.HIGH)
        time.sleep(0.1)

        GPIO.output(29, GPIO.HIGH)
        GPIO.output(31, GPIO.LOW)
        GPIO.output(33, GPIO.LOW)
        GPIO.output(35, GPIO.HIGH)
        time.sleep(0.1)

        GPIO.output(29, GPIO.HIGH)
        GPIO.output(31, GPIO.LOW)
        GPIO.output(33, GPIO.HIGH)
        GPIO.output(35, GPIO.LOW)
        time.sleep(0.1)

def bajada(desplazamiento):

    for paso in range(0,desplazamiento):

        GPIO.output(29, GPIO.HIGH)
        GPIO.output(31, GPIO.LOW)
        GPIO.output(33, GPIO.HIGH)
        GPIO.output(35, GPIO.LOW)
        time.sleep(0.1)
```

```
        GPIO.output(29, GPIO.HIGH)
        GPIO.output(31, GPIO.LOW)
        GPIO.output(33, GPIO.LOW)
        GPIO.output(35, GPIO.HIGH)
        time.sleep(0.1)

        GPIO.output(29, GPIO.LOW)
        GPIO.output(31, GPIO.HIGH)
        GPIO.output(33, GPIO.LOW)
        GPIO.output(35, GPIO.HIGH)
        time.sleep(0.1)

        GPIO.output(29, GPIO.LOW)
        GPIO.output(31, GPIO.HIGH)
        GPIO.output(33, GPIO.HIGH)
        GPIO.output(35, GPIO.LOW)
        time.sleep(0.1)

def liberation():
    GPIO.cleanup() # Devuelve los pines a su estado inicial
```

2.2. PINROTATION.PY

Programa pinrotation.py

```
#-*-coding: utf-8-*-
# Programa en el cual se define una serie de funciones para controlar la rotación de la plataforma del escáner.
# Este programa cuenta con tres funciones:
# initiation->Inicia y define los pines que controlan el motor
# movement-> Desplazamiento de un número de pasos de la plataforma de escaneado.
# liberation->Liberación de los pines.


import RPi.GPIO as GPIO #importación de la librería y asignación a "GPIO".
import time # librería utilizada para los delays.

def initiation():
    # Se establece el sistema de numeración que será empleado, en este caso BCM.
    GPIO.setmode(GPIO.BCM)

    #configuramos el pin GPIO17 como una salida
    GPIO.setup(17, GPIO.OUT)
    GPIO.setup(18, GPIO.OUT)
    GPIO.setup(22, GPIO.OUT)
    GPIO.setup(23, GPIO.OUT)

def movement(desplazamiento):
    resto = desplazamiento % 4
    if resto == 0:
        GPIO.output(17, GPIO.LOW)
        GPIO.output(18, GPIO.HIGH)
        GPIO.output(22, GPIO.HIGH)
        GPIO.output(23, GPIO.LOW)
        time.sleep(0.1)
```

```
    if resto == 1:
        GPIO.output(17, GPIO.LOW)
        GPIO.output(18, GPIO.HIGH)
        GPIO.output(22, GPIO.LOW)
        GPIO.output(23, GPIO.HIGH)
        time.sleep(0.1)


    if resto == 2:
        GPIO.output(17, GPIO.HIGH)
        GPIO.output(18, GPIO.LOW)
        GPIO.output(22, GPIO.LOW)
        GPIO.output(23, GPIO.HIGH)
        time.sleep(0.1)

    if resto == 3:
        GPIO.output(17, GPIO.HIGH)
        GPIO.output(18, GPIO.LOW)
        GPIO.output(22, GPIO.HIGH)
        GPIO.output(23, GPIO.LOW)
        time.sleep(0.1)

def liberation():
    GPIO.cleanup() # Devuelve los pines a su estado inicial
```

2.3. MAXPIXEL.PY

Este programa contiene una función cuyo objetivo es la obtención del píxel característico de un vector que contiene el componente rojo de una fila de la imagen.

A

34	45	30	60	120	255	255	255	255	90	60	45	32	38
0	1	2	3	4	5	6	7	8	9	10	11	12	13

B

34	45	30	60	120	255	255	255	255	255	60	45	32	38
0	1	2	3	4	5	6	7	8	9	10	11	12	13

C

255	255	255	255	60	45	32	38	30	33	39	35	32	36
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Figura 1; Ejemplo de diferentes vectores.

Si al programa se le proporciona un vector como el A de la figura 1, el cual tiene un número par de píxeles con sus valores máximos, el programa tomará el valor siguiente y anterior al grupo de valores máximos (píxeles 4 y 9) y como en este caso el píxel 4 es mayor el programa nos devolverá que el píxel característico es el 6.

En caso de que se le proporcione un vector como el B de la figura 1, que un número impar de píxeles con su valor máximo (píxeles 5, 6, 7, 8 y 9), el programa devolverá como píxel característico el píxel intermedio (píxel 7).

Por último si el vector proporcionado tiene un conjunto de píxeles impar y se encuentra posicionado al final o al principio del vector como es el caso C de la figura 1. En este caso el programa interpretará que el píxel que se encuentra fuera del vector es nulo y por ello escogiendo de entre los dos píxeles centrales el cual se encuentre al lado contrario. En el caso C el programa devolverá el píxel que se encuentra la posición 2.

Una vez obtenido el píxel intermedio de cualquiera de estos vectores, se le aplica una aproximación parabólica. La cual corresponde a la fórmula:

$$\hat{\delta} = \frac{1}{2} * \frac{a - b}{c - 2 * b + a}$$

En esta fórmula a, b y c representan los 3 pixeles consecutivos del pico, donde b es el pixel de mayor intensidad.

Se ha decidido la utilización de la aproximación parabólica por los pocos recursos que consume de la Raspberry Pi. Ya que el uso de una aproximación mucho más precisa elevaría mucho la carga de la Raspberry Pi.

Programa maxpixel.py

```
#-*-coding: utf-8-*-
import numpy as np
import math



def medpixel(vector):
    line = np.array(vector)
    # Se obtiene el valor máximo de la fila
    linemax = np.amax(line)
    # se obtine el número de píxeles que componen la fila
    longi = len(line)
    initial = 1

    # Se realiza un bucle for para establecer en caso que halla varios valores seguidos que tienen el valor máximo
    # las variables final e init nos indican cuando termina y cuando comienza el conjunto de valores máximos.
    for x in range(0, longi):
        if line[x] == linemax:
            if initial == 1:
                init = x
                initial = 0

            if (x+1) != longi:
```

```
            if line[x] == linemax:
                final = x

        if x == longi-1 and line[x] == linemax:
            final = x

    # Obtengo el resto del cociente de 2 para saber si el número de píxeles es par o impar
    resto = (final-init) % 2

    # Si el resto es par hay píxel intermedio
    if resto == 0 or (final-init) == 0:
        punto = math.ceil((final-init)/2)

    # Si el resto es impar no lo hay y por ello se mira en los extremos cual es el valor más alto
    # y dependiendo de ello se posiciona el píxel característico
    if resto == 1 and (final-init) != 0:
        punto = math.floor((final-init)/2)

        # Si el píxel final no coincide con la longitud del vector y la el conjunto de valores máximos no comienza en el
        #  principio de vector secomprueba cual de los extremos del grupo es mayor y en el caso que el extremo derecho
        #  sea mayor se incremente en uno el valor de la variable punto. Tambien se incrementa en caso de que el
        #  conjunto de valores máximos comienze en el principio del vector.
        if final+1 != longi and init-1 != -1 and line[init-1]<line[final+1] or init-1 == -1:
            punto = punto+1

    # Se añade la variable punto a la posición init.
    fpoint = init+punto

    # Se procede a realizar una aproximación parabólica.
    if fpoint+1 != longi and fpoint-1 != -1:
        a = line[fpoint-1]
```

```
    b = line[fpoint]
    c = line[fpoint+1]

if fpoint+1 == longi:
    a = line[fpoint-1]
    b = line[fpoint]
    c = 0

if fpoint-1 == -1:
    a = 0
    b = line[fpoint]
    c = line[fpoint+1]

incremento = 0
if a != b or b != c:
    incremento = 0.5 * (a-b)/(c - 2 * b + a)
fpoint += incremento
return fpoint
```

2.4. LEERMATRIX.PY

Programa leermatrix.py

```
#-*-coding: utf-8-*-
# Programa en el cual se define una función que devuelve una matriz contenida en un archivo" .txt".
# Se define la función.
def lectura( num_filas, names):

    # Este programa solo sirve para las matrices que han sido creadas mediante la opción de calibrado.
    outfile = open(names)
    result = []
    if num_filas != 1:
        for col in range(0, num_filas):
            linea = outfile.readline()
            sep = linea.split()
            num = len(sep)
            if col == 0:
                c = []
                for x in range(0, num):
                    if x == 0:
                        long2 = len(sep[1])
                        c.append(float(sep[1][0:long2]))
                    if x != 1 and x != 0 and x != (num-1):
                        c.append(float(sep[x]))
                    if x == (num-1):
                        long2 = len(sep[num-1])
                        c.append(float(sep[num-1][0:long2-1]))
                result.append(c)
            if col != 0 and col != num_filas-1:
                c2= []
                for x in range(0, num):
                    if x == 0:
                        long2 = len(sep[1])
```

```
                    c2.append(float(sep[1][0:long2]))
                if x != 1 and x != 0 and x != (num-1):
                    c2.append(float(sep[x]))
                if x == (num-1):
                    long2 = len(sep[num-1])
                    c2.append(float(sep[num-1][0:long2-1]))
            result.append(c2)
        if col == num_filas-1:
            c2 = []
            for x in range(0, num):
                if x == 0:
                    c2.append(float(sep[1]))
                if x != 1 and x != 0 and x != (num-1) and names=='Refmat.txt':
                    c2.append(float(sep[x]))
                if x != 1 and x != 0 and x != (num-1) and x != (num-2) and
names=='CalibMat.txt':
                    c2.append(float(sep[x]))
                if x == (num-1) and names=='Refmat.txt':
                    long2 = len(sep[num-1])
                    c2.append(float(sep[num-1][0:long2-2]))
                if x == (num-1) and names=='CalibMat.txt':
                    c2.append(float(sep[num-2]))
            result.append(c2)
            outfile.close()
            return result
    if num_filas == 1:
        linea = outfile.readline()
        sep = linea.split()
        num = len(sep)
        c = []
        for x in range(1, num-1):
            c.append(float(sep[x]))
        result.append(c)
        outfile.close()
        return result
```

2.5. COORD.PY

Programa leermatrix.py

```
#-*-coding: utf-8-*-
# Programa en el cual se introduce un vector con los puntos normalizados y
devuleve un vector ya en el plano [x,y].
# La función tiene como entradas:
# cord->vector con los valores normalizados
# c1->valor de la cordenada que conozco por el plano "p"
# p->plano en el que se encuentran los puntos
# M->matriz de rotación con el vector de translación.[R|T]

import numpy as np

def coordxyz(cord,c1,p,M):



    for k in range (0,len(cord)):
        u = cord[k][0][0]
        v = cord[k][0][1]
        if p == 'x':
            colA = np.array([1, 2])
            colB = np.array([0, 3])

        if p == 'y':
            colA = np.array([0, 2])
            colB = np.array([1, 3])

        if p == 'z':
            colA = np.array([0, 1])
            colB = np.array([2, 3])

        AI = np.transpose(M)
        A2 = np.array([AI[colA[0]],AI[colA[1]]])
        A = np.transpose(A2)
        tvecs = np.array([[u],[v],[1]])
```

```
    AF = (np.c_[A, tvecs])

    B2 = np.array([AI[colB[0]],AI[colB[1]]])
    BF = np.transpose(B2)

    E1, resid, rank, s = np.linalg.lstsq(AF, BF)

    vec = np.array([[c1], [1]])
    Pxyz = np.dot(E1, vec)

    cord[k][0][0] = Pxyz[0][0]
    cord[k][0][1] = Pxyz[1][0]

return cord
```

3. INSTALACIÓN DEL SISTEMA OPERATIVO

A continuación se describen los diferentes sistemas operativos que se pueden utilizar en la Raspberry Pi, así como el sistema operativo requerido para el escáner y como instalarlo.

Lo primero que se tiene que tener en cuenta es que la Raspberry Pi es una tarjeta con gran variedad de aplicaciones lo cual ha producido un gran número de sistemas operativos que se pueden meter dentro de ella. Es tal su utilidad que no solo la empresa ha sacado sus propios sistemas operativos sino que diferentes usuarios han hecho por su cuenta adaptaciones para esta tarjeta. A continuación según la página Raspberry para torpes se presenta una tabla con todos los sistemas operativos diseñados para esta tarjeta:

Oficiales	**No oficiales**

• GNU/ **Linux** – Uso PC o servidor
  • **RASPBIAN** Debian Jessie
  • usuario: pi | contraseña: raspberry
  • RASPBIAN **Lite** Debian Jessie
  • usuario: pi |
	• GNU/**Linux** – Uso PC o servidor
	  • **ARCH LINUX**
	  • usuario: root | password: root
	  • **Occidentalis** especial para proyectos de electrónica
	  • **HypriotOS** especial para

<table>
<tr>
<td>
contraseña: raspberry
• PIDORA Fedora Remix
• GNU/Linux – Uso Media Center XBMC
• OSMC
• usuario: osmc | contraseña: osmc
• OPENELEC
• Solo Raspberry Pi 2
• Ubuntu Snappy Core solo servidor sin escritorio gráfico
• No Linux – Uso PC
• RISC OS
</td>
<td>
usar Docker ver post con
• MOEBIUS se ha reactivado y la <u>versión 2</u> cabe en una SD de 128MB no depende de otras distros
• usuario: root | password: moebius
• Minibian cabe en una SD de 512MB
• usuario: root | password: raspberry
• Minibian-wifi. LA anterior versión era tan minimalista que no incluye ni drivers Wi-Fi, esta sí.
• VoidLinux minimalista y desarrollada desde cero
• usuario: anon o también root | password: voidlinux
• openSUSE
• pipaOS
• PiBang no se actualiza desde mayo de 2014
• no incluye usuario por defecto se configura con raspi-setup no confundir con raspi-config de Raspbian
• OpenMediaVault interesante versión adaptada para crear un NAS desde cero
• Kano especial para los más pequeños y con software educativo
</td>
</tr>
</table>

	• LinuTOP se trata de una **versión de pago** especial para montar redes de información al público o kioskos de acceso a Internet todo configurable gráficamente. Incluye **VLC con aceleración gráfica**. • Q4OS basado en **Raspbian** pero con **escritorio KDE** • DietPi nueva web oficial y en el antiguo foro. Versión mínima de **Raspbian** comprimido ocupa solo **128MB** y al instalar **es suficiente con una microSD d 1GB**. Incluye su propia herramienta de configuración para instalar conjuntos completos de utilidades. • GNU/Linux – **solo Raspberry Pi 2** • **Ubuntu 14.04** sin escritorio predefinido, pero se puede instalar luego: • KDE = Kubuntu – sudo apt-get install kubuntu-desktop • XFCE = Xubuntu – sudo apt-get install xubuntu-desktop • Lxde = Lubuntu – sudo apt-get install lubuntu-desktop • **Ubuntu MATE** 15.04 con el

	escritorio MATE, ligero pero potente • Gentoo • GNU/Linux – Uso Media Center **XBMC** • Xbian • **OSMC** • GNU/Linux – **EMULADORES** • **Recalbox** • MAME4all • PiPlay • OpenMSX • EmulaStation • Chameleon Pi • **Retropie** • Happi Game Center • Lakka • GNU/Linux – Uso Media Center **Audio** • **Volumio** • Rune Audio • PiMusicBox • piCorePlayer • SqueezePlug • Linux – **Android o similar** • Android **NO FUNCIONAL** • Android Lollipop **NO FUNCIONAL pero casi** para **Raspberry Pi** • **Tizen** OS solo para **Raspberry Pi 2** • **SailPi** versión de SailfishOS solo para **Raspberry Pi 2** • **No Linux** – Uso PC o servidor • **FreeBSD** • **NetBSD**

	• **Windows 10 IoT** solo como **servidor** y solo para **Raspberry Pi** • **Plan9** • **Inferno OS** • **Inferno Pi** • **XinuPI** para enseñanza de Sistemas Operativos

Tabla 1; Listado de Sistemas operativos para Raspberry Pi.

En este caso, lo que se va a necesitar es un sistema operativo que permita utilizar Python, por ello se utilizará Raspbian. Para obtenerlo nos dirigimos a la página oficial de Raspberry Pi:

URL https://www.raspberrypi.org/

Este sistema operativo requiere una tarjeta micro SD que tenga como mínimo una capacidad de 8GB y para su instalación se requiere del programa auxiliar "SD Formatter 4.0". Que puede ser descargado de forma gratuita en el siguiente enlace:

URL https://www.sdcard.org/downloads/formatter_4/

Una vez obtenido el sistema operativo y el programa de formateo. Se procede a formatear la tarjeta SD que será instalada en la Raspberry Pi, para a continuación instalar el sistema operativo simplemente descomprimiendo el archivo descargado de la página oficial de Raspberry Pi dentro de la tarjeta de memoria.

Una vez instalado el sistema operativo y al conectar la Raspberry Pi a un monitor por primera vez se deberán ejecutar una serie de comandos.

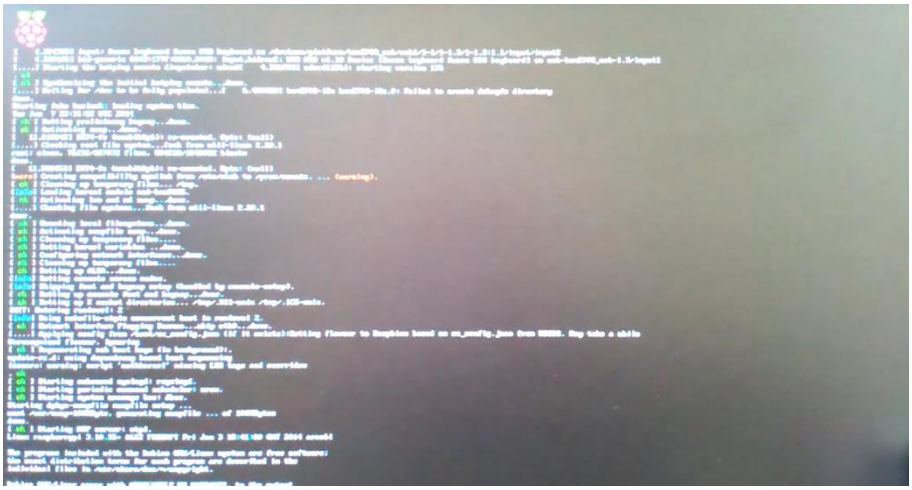


Figura 2; Pantalla del primer arranque de una Raspberry Pi

En este punto debemos introducir el comando “install” y a continuación nos van a pedir usuario y contraseña, las cuales son:

Usuario:	pi
Contraseña:	raspberry

Tabla 2; Usuario y contraseña del sistema operativo Raspbian

Una vez introducido el usuario y la contraseña se abrirá una ventana de configuración del sistema. Realizado esto volveremos a la pantalla en negro de códigos, en la cual para terminar la instalación se escribirá el comando “startx”.

La página web oficial de Raspberry Pi, nos ofrece las siguientes indicaciones:

Indicaciones de la página oficial de Raspberry Pi
DOWNLOAD: Se recomienda usar una memoria SD con un mínimo de capacidad de 8GB. 1-Usar un ordenador con un lector de tarjeta SD. 2-Click en el “Download zip” debajo de “Noobs”. 3-Estraer los archivos del zip. FORMATEAR TU TARJETA SD: 1-Visitar la página web de SD Association y descargar SD Formatter 4.0. 2-Seguir las instrucciones y descargar el software. 3-Conectar la tarjeta SD al ordenador. 4-En el programa SD Formatter, seleccionamos el puerto de la tarjeta SD y formatearlo.

ARRASTRAR Y SOLTAR LOS ARCHIVOS NOOBS:

1-Una vez formateado tu tarjeta SD, arrastra los ficheros extraídos del fichero Noobs y soltarlo dentro de la tarjeta SD.

2-Los ficheros necesarios serán transferidos en la tarjeta SD.

3-Cuando el proceso ha terminado, sacar de forma segura la tarjeta SD e insertarla dentro de la tarjeta Raspberry Pi.

PRIMER ARRANQUE:

1-Conectar a la tarjeta Raspberry un teclado, ratón y un monitor.

2-Conectar la tarjeta a la fuente de alimentación.

3-La Raspberry Pi arrancara, y una ventana aparecerá con una lista de diferentes sistemas operativos que tú puedes instalar. Nosotros recomendamos que tú uses Raspbian- tick en la caja al lado y da click en “Install”.

4-Raspbian comenzara a instalarse. Esto puede tardar un tiempo.

5-Cuando el proceso de instalación se complete, la Raspberry Pi cargara un menú de configuración. En el cual se selecciona la hora, el día, el idioma…

LOGGING IN Y ACCESO A LA INTERFAZ GRAFICA:

La default login para Raspbian es username “pi” y la password “raspberry”. Hay que tener en cuenta que no veras ningún tipo de símbolo cuando escribas la contraseña por motivos de seguridad de la propia Raspbian. Esta es una de las características de Linux. Para cargar la interfaz gráfica del usuario, escribe “startx” y pulsa Enter.

4. INSTALACIÓN DEL SOFTWARE

A continuación se describen todos los pasos para la instalación de las librerías necesarias para el funcionamiento del escáner como el programa utilizado para programarlo.

1) Instalación de la librería GPIO (controladores de pines):

Para controlar los motores paso a paso mediante los chips “L293” necesitamos ser capaces de controlar la tensión de los pines. Para controlar un motor paso a paso mediante este chip necesitaremos controlar cuatro pines y como son dos motores en total serán 8 pines.

Para descargar este software procederemos a realizar los siguientes pasos:

- Abrimos el terminal. No el de Python; si no el del propio sistema.

Figura 3; Icono de Terminal

El símbolo del “Terminal” lo podemos encontrar a la derecha del botón “Menu”; este tiene forma de pantalla de ordenador.

Al abrirlo se nos abrirá una ventana como la siguiente:

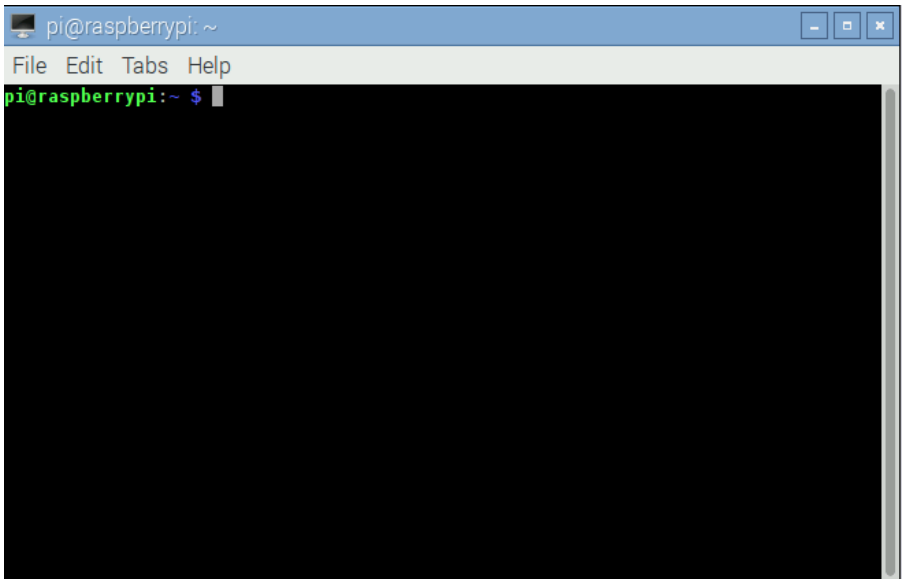


Figura 4; Terminal

- Dentro de este terminal ejecutamos el siguiente código:

sudo python

Dentro de la consola de Python comprobamos la versión

import RPi.GPIO

RPi.GPIO.VERSION

Si nuestra versión es inferior a la 0.5.4 la actualizamos (y ya de paso todos los paquetes del sistema que lo requieran)

sudo apt-get update

sudo apt-get upgrade

Si no tenemos instalada RPi.GPIO, la descargamos e instalamos. Para ello ejecutamos el siguiente código:

sudo apt-get install python-dev python-rpi.gpio

2) Instalación de OpenCV:

Este software “OpenCV” es un software especializado para la toma de imágenes y su tratamiento. Para la instalación procedemos con los siguientes pasos:

- Abrimos el terminal del sistema (mirar la instalación de GPIO).

- Introducimos el siguiente código:

sudo apt-get update

sudo apt-get install libopencv-dev python-opencv

Con esto teóricamente ya funcionaría pero sin embargo es posible que no se haya instalado correctamente con lo cual ejecutamos el siguiente código:

sudo apt-get -f install

Una vez instalado, escribimos de nuevo:

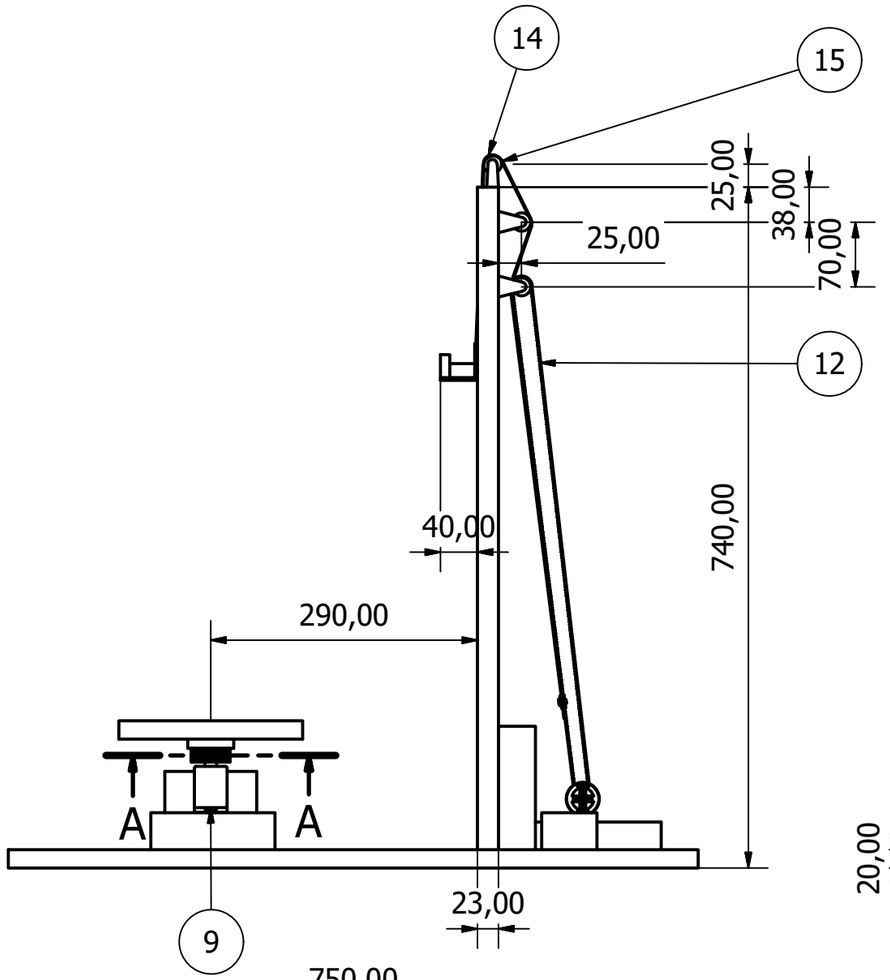
sudo apt-get install libopencv-dev python-opencv

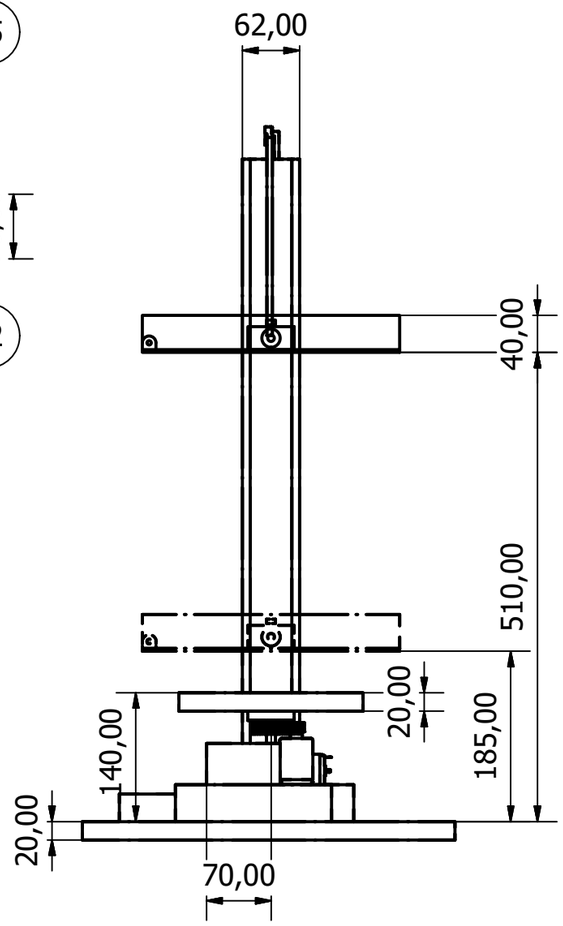
Y ya se habrían instalado todas las librerías.

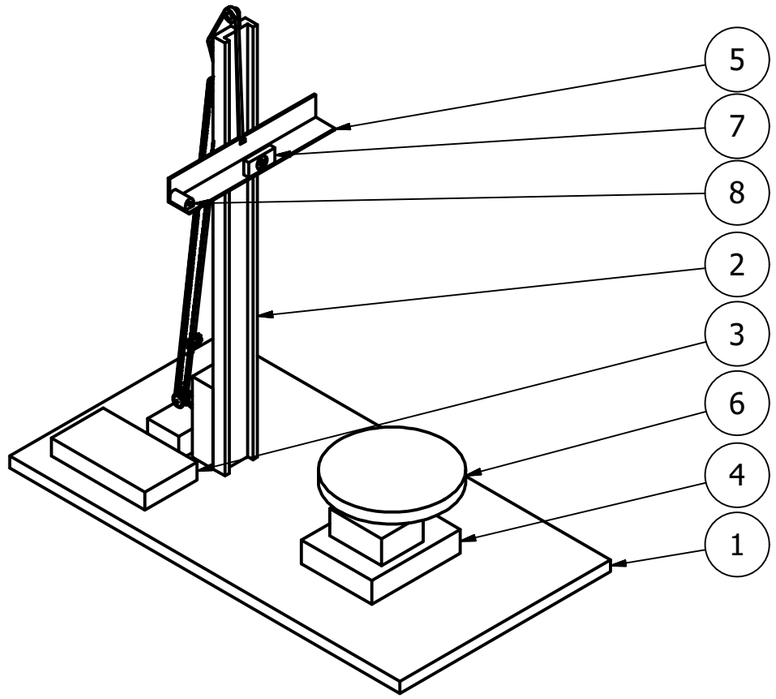
DOCUMENTO 3

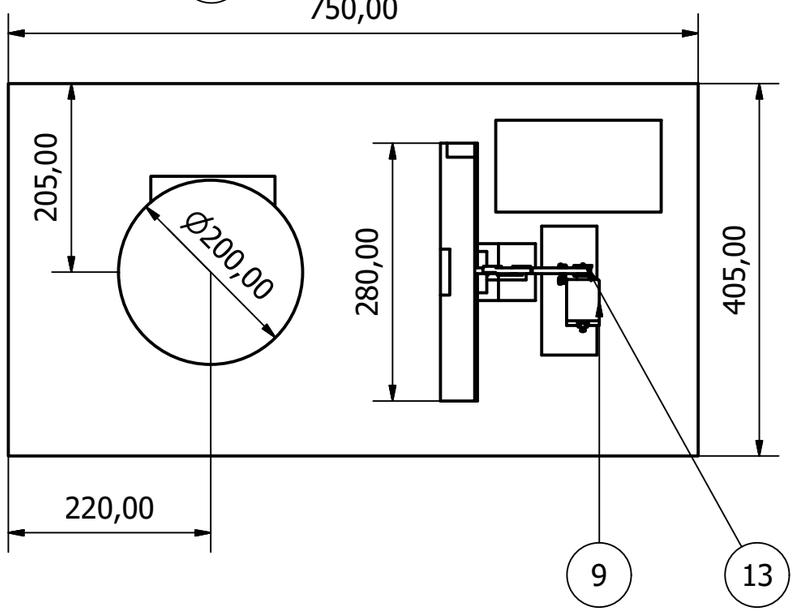
PLANOS

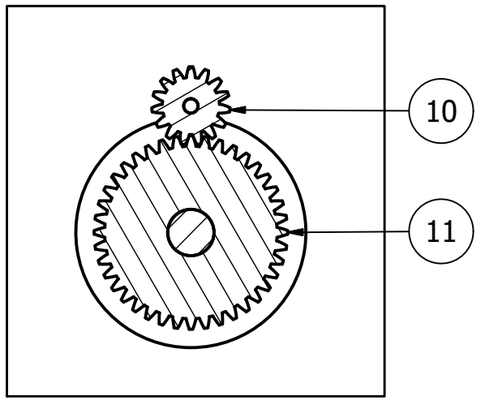
ÌNDICE









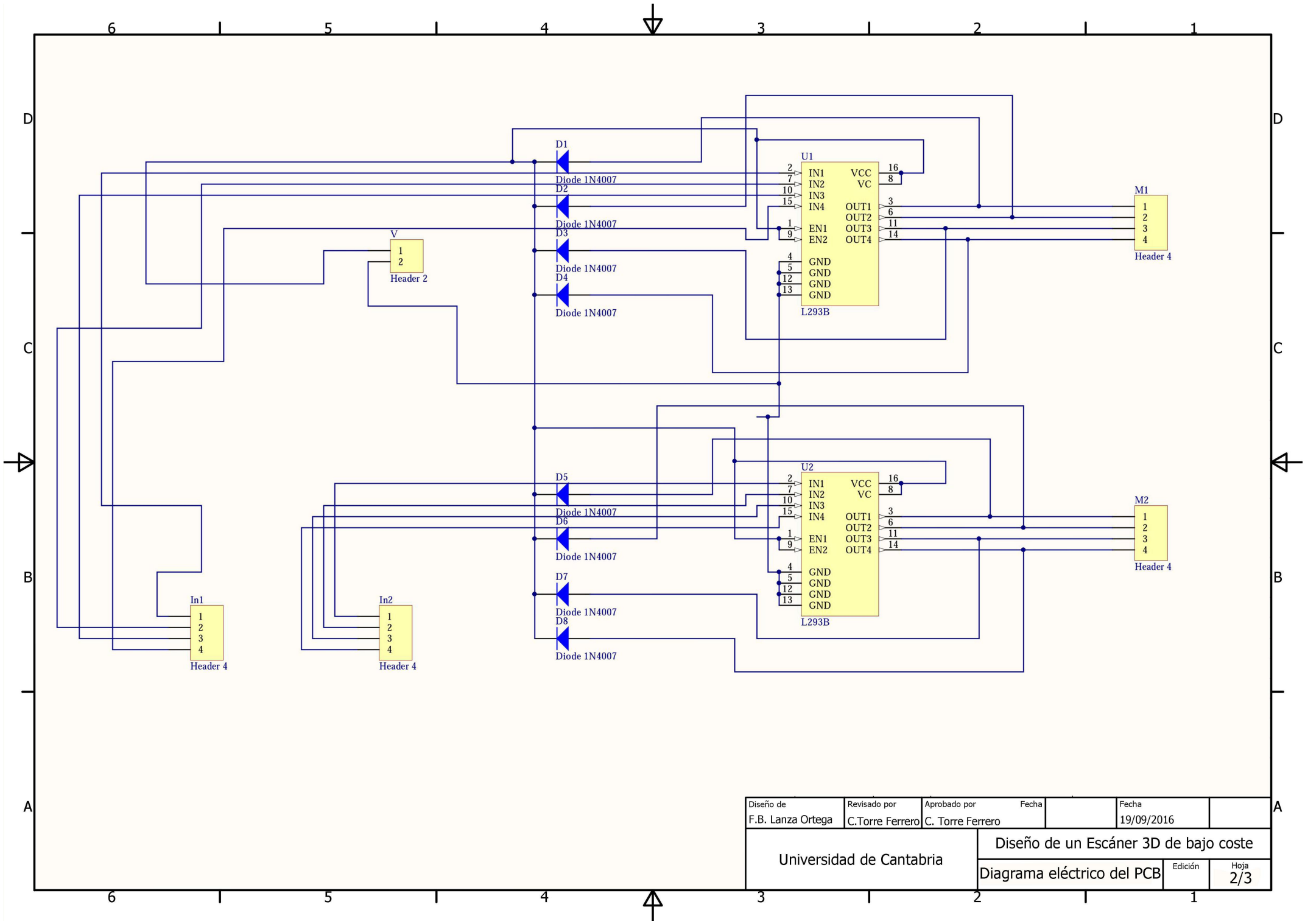


A-A (0.75:1)

LISTA DE PIEZAS			
ELEMENTO	CTDAD	Nº DE PIEZA	DESCRIPCIÓN
1	1	Tabla	
2	1	Columna	Guia integrada
3	1	Procesador	Medidas no especificadas por variedad de elementos
4	1	Soporte-Base- Motor	
5	1	Plataforma	
6	1	Base	
7	1	Camara	
8	1	Laser	
9	2	Motor-NEMA 17	Motor paso a paso bipolar
10	1	Engranaje Motriz	Z= 10 m=1
11	1	Engranaje Movil	Z= 40 m=1
12	1	Correa síncrona	Ancho: 6mm m=1.5
13	1	Polea síncrona	Z=15 m= 1.5
14	3	Polea plana	
15	1	correa-2	Sujeción con correa sincrona mediante brida

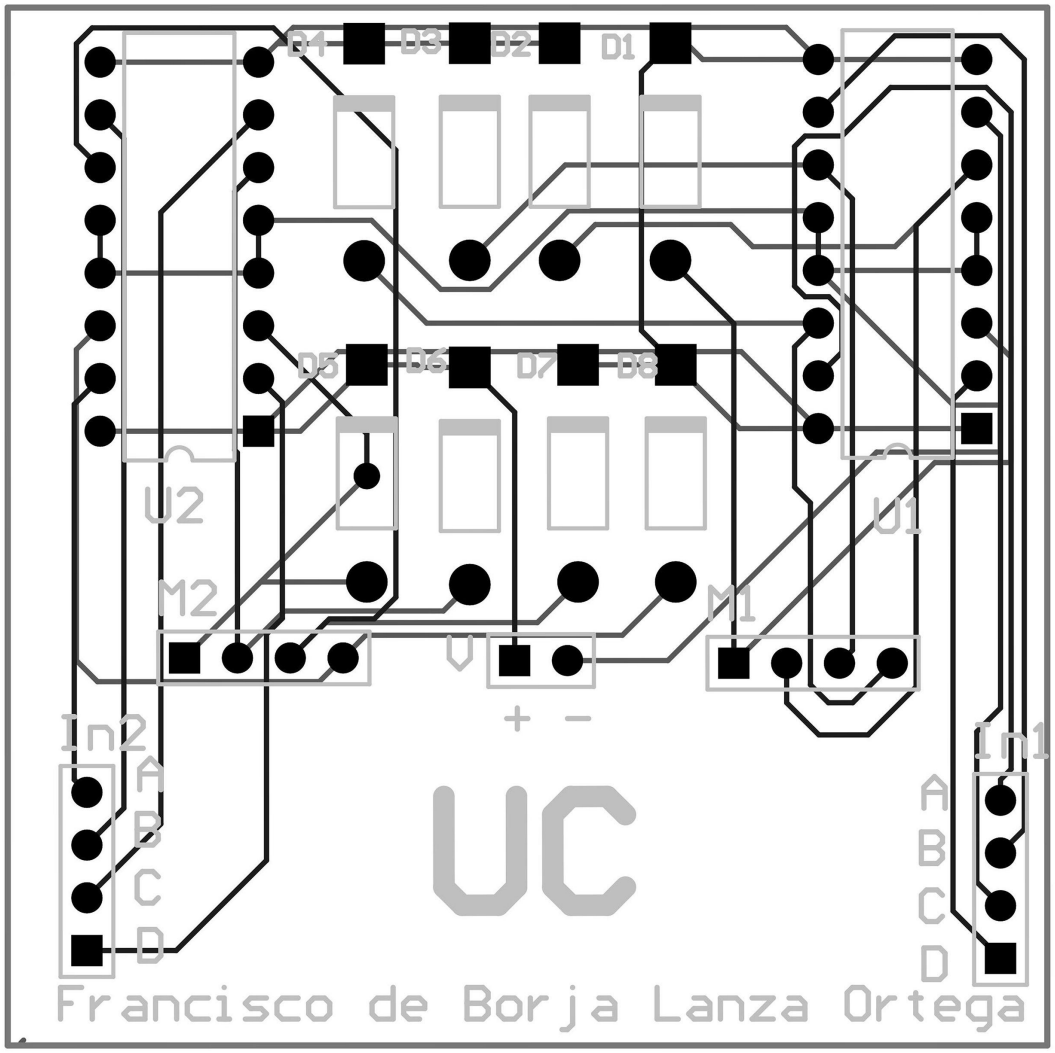
Diseño de	Revisado por	Aprobado por	Fecha	Fecha	
F.B. Lanza Ortega	C.Torre Ferrero	C. Torre Ferrero		19/09/2016	

Universidad de Cantabria	Diseño de un Escáner 3D de bajo coste		
	Escaner 3d de bajo coste	Edición	Hoja 1 / 2



Diseño de	Revisado por	Aprobado por	Fecha		Fecha	
F.B. Lanza Ortega	C.Torre Ferrero	C. Torre Ferrero			19/09/2016	

Universidad de Cantabria	Diseño de un Escáner 3D de bajo coste		
	Diagrama eléctrico del PCB	Edición	Hoja 2/3



Diseño de	Revisado por	Aprobado por	Fecha		Fecha	
F.B. Lanza Ortega	C.Torre Ferrero	C. Torre Ferrero			19/09/2016	
Universidad de Cantabria				Diseño de un Escáner 3D de bajo coste		
				Plano del PCB	Edición	Hoja 3/3

DOCUMENTO 4
MEDICIONES

ÍNDICE:

1. MEDICIONES DE MATERIALES

Unidades	Material	Cantidad
Uds.	Raspberry Pi 3.	1
Uds.	Chip: L293.	2
Uds.	Diodo: 1N4007 (Paquete de 25 Unidades).	1
Uds.	Motor paso a paso: Nema 17 2 Fases 4 Cables 1.8°.	2
Uds.	2M Correa Belt GT2 + 2x Poleas Pulley.	1
Uds.	Piezas de madera de diferentes tamaños.	1
Uds.	Rueda fija nylon negra diámetro 20 mm.	3
Uds.	Guía de acero.	1
Uds.	650nm 5mw rojo Focusable Laser Line Module. Módulo láser estándar industrial.	1
Uds.	Creative LIVE! Cam Chat HD.	1
Uds.	Abrazadera.	3
Uds.	Piñón, módulo 1, 10 piñones.	1
Uds.	Piñón, módulo 1, 40 piñones.	1
Uds.	Barra cromada rectificada F 114-10mm 10cm.	1
Uds.	Rodamiento.	1

2. MEDICIONES DE TIEMPOS

Unidades	Operación	Tiempo (min)
min	Corte de piezas.	30
min	Taladrado y atornillado de las piezas.	140
min	Colocación de la correa de transmisión.	30
min	Instalación de los motores.	20
min	Montaje de la sujeción del láser y su instalación.	15
min	Montaje de la cámara.	10
min	Instalación del hardware.	20
min	Total	265

DOCUMENTO 5

PRESUPUESTO

ÍNDICE:

1. JUSTIFICACIÓN DE LOS PRECIOS

1.1. JUSTIFICACIÓN DE LOS PRECIOS DE LOS MATERIALES

Material	Precio U(€/Unidad)
Raspberry Pi 3.	41.95
Chip: L293.	1.20
Diodo: 1N4007 (Paquete de 25 Unidades).	1.00
Motor paso a paso: Nema 17 2 Fases 4 Cables 1.8°.	10.85
2M Correa Belt GT2 + 2x Poleas Pulley.	5.89
Piezas de madera de diferentes tamaños.	9.00
Rueda fija nylon negra diámetro 20 mm.	0.89
Guía de acero.	13.56
650nm 5mw rojo Focusable Laser Line Module. Módulo láser estándar industrial.	2.79
Creative LIVE! Cam Chat HD.	22.95
Abrazadera.	0.85
Piñón, módulo 1, 10 piñones.	4.39
Piñón, módulo 1, 40 piñones.	10.44
Barra cromada rectificada F 114-10mm 10cm.	21.43
Rodamiento.	13.32

1.2. JUSTIFICACIÓN DEL COSTE DE LA MANO DE OBRA

Hora de trabajo de un peón es 15€/h o 0.25 €/minuto.

2. PRESUPUESTOS PARCIALES

2.1. PRESUPUESTO PARCIAL RELATIVO A MATERIALES

Material	Cantidad	Precio U(€/Unidad)	Precio Total(€)
Raspberry Pi 3.	1	41.95	41.95
Chip: L293.	2	1.20	2.40
Diodo: 1N4007 (Paquete de 25 Unidades).	1	1.00	1.00
Motor paso a paso: Nema 17 2 Fases 4 Cables 1.8°.	2	10.85	21.70
2M Correa Belt GT2 + 2x Poleas Pulley.	1	5.89	5.89
Piezas de madera de diferentes tamaños.	1	9	9
Rueda fija nylon negra diámetro 20 mm.	3	0.89	2.67
Guía de acero.	1	13.56	13.56
650nm 5mw rojo Focusable Laser Line Module. Módulo láser estándar industrial.	1	2.79	2.79
Creative LIVE! Cam Chat HD.	1	22.95	22.95
Abrazadera.	3	0.85	2.55
Piñón, módulo 1, 10 piñones.	1	4.39	4.39
Piñón, módulo 1, 40 piñones.	1	10.44	10.44
Barra cromada rectificada F 114-10mm 10cm.	1	21.43	6.43
Rodamiento.	1	13.32	13.32
Total		161.04	

2.2. PRESUPUESTO PARCIAL RELATIVO A OPERACIONES DE MONTAJE

Mano de obra				
Orden	Operación	Tiempo (min)	Precio mano de obra (€/min)	Precio total (€)
1	Corte de piezas.	30	0.25	7.5
2	Taladrado y atornillado de las piezas.	140		35
3	Colocación de la correa de transmisión.	30		7.5
4	Instalación de los motores.	20		5
5	Montaje de la sujeción del láser y su instalación.	15		3.75
6	Montaje de la cámara.	10		2.5
7	Instalación del hardware.	20		5
Total		265		66.25

3. PRESUPUESTO

Presupuesto	
Materiales	161.04 €
Mano de obra	66.25 €
Impuestos (21%)	47.73 €
Presupuesto Total	275.02 €

El presupuesto de ejecución material del prototipo asciende a un total de 275.02 € (doscientos setenta y cinco euros con dos céntimos).

DOCUMENTO 6

BIBLIOGRAFÍA

Enlaces consultados:

1) DEPARTAMENTO DE INGENIERÍA INDUSTRIAL UNIVERSIDAD POLITÉCNICA DE MADRID. Reconstrucción 3D. En: DEPARTAMENTO DE INGENIERÍA INDUSTRIAL UNIVERSIDAD POLITÉCNICA DE MADRID [sitio web]. Madrid:UPM [Consulta: 15 junio 2016]. Disponible en: http://www.elai.upm.es/webantigua/spain/Investiga/GCII/personal/lrodriguez/web3D/reconstruccion_3d.htm

2) STACK OVERFLOW. 2012. How to format xy points for undistortPoints with the python cv2 api?. En: Stack Overflow [sitio web]. [Consulta: 1 mayo 2016]. Disponible en: http://stackoverflow.com/questions/11017984/how-to-format-xy-points-for-undistortpoints-with-the-python-cv2-api

3) GITHUB. 2012. Guia-Tkinter/Interfaz gráfica con Tkinter.wiki. En: GitHub [sitio web]. [Consulta: 18 julio 2016]. Disponible en: https://github.com/elilluminado/Guia-Tkinter/blob/master/Interfaz%20grafica%20con%20Tkinter.wiki

4) SIGHTATIONS. 2013. Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix. En: Sightations [sitio web]. [Consulta: 11 julio 2016]. Disponible en: http://ksimek.github.io/2016/08/13/intrinsic/

5) DEPARTAMENTO DE INGENIERÍA INDUSTRIAL UNIVERSIDAD COMPLUTENSE DE MADRID. Computación científica con Python para módulos de evaluación continua en asignaturas de ciencias aplicadas. En: DEPARTAMENTO DE INGENIERÍA INDUSTRIAL UNIVERSIDAD COMPLUTENSE DE MADRID [sitio web]. Madrid:UCM [Consulta: 7 mayo 2016]. Disponible en: http://pendientedemigracion.ucm.es/info/aocg/python/modulos_cientificos/numpy/index.html#matrices

6) OPENCV. Camera Calibration and 3D Reconstruction. En: OpenCV [sitio web]. [Consulta: 23 septiembre 2016]. Disponible en: http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

7) OPENCV. Camera Calibration. En: OpenCV [sitio web]. [Consulta: 5 junio 2016]. Disponible en: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html

8) MICROMOUSE. XYZ Files. En: Micromouse [sitio web]. [Consulta: 1 septiembre 2016]. Disponible en: http://www.micromouse.ca/xyz_files.html

9) PYIMAGESEARCH. OpenCV with Tkinter. En: Pyimagesearch [sitio web]. [Consulta: 14 septiembre 2016]. Disponible en: http://www.pyimagesearch.com/2016/05/23/opencv-with-tkinter/

10) UNIVERSIDAD DEL PAÍS VASCO. SECUENCIAS PARA MANEJAR MOTORES PASO A PASO. En: Universidad del País Vasco [sitio web]. País Vasco:UPV [Consulta: 30 agosto 2016]. Disponible en: http://server-die.alc.upv.es/asignaturas/lsed/2002-03/MotoresPasoaPaso/tipos.htm

11) MECATRÓNICA UASLP. Patrones de calibración para OpenCV. En: Mecatrónica UASLP [sitio web]. [Consulta: 25 mayo 2016]. Disponible en: https://mecatronicauaslp.wordpress.com/2014/03/08/patrones-de-calibracion-para-opencv/

12) RASPBERRY PARA TORPES. NOOBS paso a paso. Instalar el sistema operativo en la Raspberry Pi. En: Raspberry para torpes [sitio web]. [Consulta: 19 agosto 2016]. Disponible en: https://raspberryparatorpes.net/instalacion/noobs-paso-a-paso-instalar-el-sistema-operativo-en-la-raspberry-pi/

13) ROBOTLOGS. Tutorial de Raspberry Pi GPIO y Python. En: Robotlogs [sitio web]. [Consulta: 8 junio 2016]. Disponible en: http://robologs.net/2014/04/12/tutorial-de-raspberry-pi-gpio-y-python-i/

14) FRAMBUESA PI COLOMBIA. Tutorial 3 – Instalación y configuración de inicial del Raspberry Pi (raspi-config). En: Frambuesa Pi [sitio web]. [Consulta: 14 mayo 2016]. Disponible en: http://www.frambuesapi.co/2016/08/10/tutorial-3-instalacion-y-configuracion-de-inicial-del-raspberry-pi-raspi-config/

15) OPENCV. Install OpenCV-Python in Windows. En: OpenCV [sitio web]. [Consulta: 23 julio 2016]. Disponible en: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_setup_in_windows/py_setup_in_windows.html

16) TODOROBOT. Tutorial sobre Motores Paso a Paso (Stepper motors). En: Todorobot La web de Android [sitio web]. [Consulta: 20 junio 2016]. Disponible en: http://www.todorobot.com.ar/tutorial-sobre-motores-paso-a-paso-stepper-motors/

17) MINIBOTS. Notas sobre robótica, sistemas operativos y programación En: Minibots [sitio web]. [Consulta: 18 agosto 2016]. Disponible en: Minibots.wordpress.com

18) DEPARTAMENTO DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE CORUÑA. Escáner 3D. En: Escáner e Impresión 3D [sitio web]. Coruña:UDC [Consulta: 4 mayo 2016]. Disponible en: http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/Hardware/scanner3D/Escaner3D.html

19) DEPARTAMENTO DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DEL PAÍS VASCO. Reconstrucción 3D. En: DEPARTAMENTO DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DEL PAÍS VASCO [sitio web]. País Vasco:UPV [Consulta: 7 julio 2016]. Disponible en: http://server-die.alc.upv.es/asignaturas/lsed/2002-03/MotoresPasoaPaso/tipos.htm

Libros y artículos consultados:

20. TORRE FERRERO, C. [et al]. 2004. Fusión de datos para reconstrucción 3D mediante un sistema híbrido de visión. En: *XXV Jornadas de Automática Ciudad Real, del 8 al 10 de Septiembre de 2004.* Archivo pdf. Disponible en: intranet.ceautomatica.es/old/actividades/jornadas/XXV/.../109-areroauorr.PDF

21. FISHER, R.B.; NAIDU, D.K. 1991. A comparison of algorithms for subpixel peak detection. En: *British Machine Vision Conference.* Archivo pdf. Disponible en: http://webcache.googleusercontent.com/search?q=cache:D4R9Dl7VvE4J:citeseerx.ist.psu.edu/viewdoc/download%3Fdoi%3D10.1.1.380.5886%26rep%3Drep1%26type%3Dpdf+&cd=1&hl=es&ct=clnk&gl=es

22. FAUGERAS, O. 1993. *Three-Dimensional Computer Vision: A Geometric Viewpoint.* MIT Press 1993. ISBN 0262061589, 9780262061582.

23. FOREST, J. [et al]. 2004. Laser stripe peak detector for 3D sanners. A FIR filter approach. En:Procceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004. Disponible en: https://www.researchgate.net/publication/46771681_Laser_stripe_peak_detector_for_3D_scanners_A_FIR_filter_approach

24. WATKINS, D.S. [et all]. 2002. *Fundamentals of matrix Computations.* WILEY-INTERSCIENCE 2002. ISBN 0-471-21394-2.