UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UN SISTEMA DE VISIÓN DE BAJO COSTE PARA RECONSTRUCCIÓN 3D DE OBJETOS

AUTOR: PÉREZ ÁLVAREZ, JAIME

TUTOR: VALERA FERNÁNDEZ, ÁNGEL

COTUTOR: SÁNCHEZ SALMERÓN, ANTONIO JOSÉ

Curso Académico: 2014-15

AGRADECIMIENTOS

Quisiera aprovechar la ocasión para agradecer todo el apoyo a mi familia, asimismo como el esfuerzo que han hecho para adaptarse a mis horarios con el fin de ayudarme en el proyecto.

También quisiera agradecer todo el esfuerzo, ayuda y dedicación a mi cotutor, Antonio, quien pese a mi insistencia en algunas ocasiones siempre me ha recibido con amistad y amabilidad, y del que he aprendido muchísimo durante este proyecto.

Por último, pero no menos importante, quisiera agradecer a mis compañeros del Laboratorio Mixto del ai2 toda su ayuda y apoyo a lo largo de todo el proyecto. También darles las gracias por mostrarme la importancia de tomar un café con buena y grata compañía.

ÍNDICE

Documentos del Trabajo Final de Grado

- Memoria
- Presupuesto
- Planos

Índice de la Memoria

Índice de Planos

DISEÑO DE UN SISTEMA DE VISIÓN DE BAJO COSTE PARA RECONSTRUCCIÓN 3D DE OBJETOS

Memoria

Jaime Pérez Álvarez

1. RESUMEN DEL PROYECTO

El proyecto tratará sobre el diseño de un sistema de visión con el cual se captarán imágenes de un objeto y se procesarán para la reconstrucción del mismo en 3D. Para ello se tomarán dos planos perpendiculares entre si que contengan la pieza a reconstruir para tomar las imágenes, con las cuales se reconstruirá posteriormente el objeto.

Se espera aplicar dicho proceso para la detección de errores en piezas imprimidas mediante técnicas de adición de plástico (ABS/PLA) o comúnmente llamada impresión 3D. Los errores a detectar son los más frecuentes que suelen aparecer en la técnica citada anteriormente, que son el despegue entre las capas de material y la deformación del contorno de la pieza.

La detección de errores se realizará mediante la detección de bordes tanto exteriores de la pieza, para detectar la deformación de la misma, como interiores, para detectar el despegue entre capas. Aplicándose tanto en las imágenes en 2D como en la pieza final reconstruida.

Para ello se utilizará un sistema con dos cámaras, una para el plano que contiene el contorno de la pieza o plano superior y otra para el plano que contiene el perfil de las capas de material o plano frontal, las cuales habrá que calibrar para poder realizar mediciones exactas en las imágenes. Una vez obtenidas las mediciones, para el plano superior compararemos el contorno real de la pieza con las coordenadas del contorno que nos da el software de la impresora. Para el plano frontal compararemos, del mismo modo, la coordenada exacta de la capa superior y si existen bordes dentro de la pieza (que indicará que las capas inferiores se han despegado).

Una vez comprobada la detección de errores en cada plano, procederemos, como ya se ha hablado, a reconstruir la pieza mediante OpenCV y, posteriormente, se aplicarán las mismas técnicas de detección de errores en la pieza reconstruida.

Se necesitará, asimismo, una correcta iluminación de la pieza, por lo que se colocarán dos focos halógenos, jugando con su posición e inclinación para obtener una luz adecuada y se procederá a aislar (en la medida de lo posible) el sistema del exterior para que la luz ambiental no influya en las imágenes.

Por último, debido a que se busca un sistema de bajo coste y alta resolución, se analizará la relación resolución/coste del sistema y se comparará con el mismo parámetro de un sistema de visión 3D conocido como es la cámara Kinect v2 desarrollada por la compañía Microsoft. En esta comparación se intentará obtener un sistema de visión con una resolución similar o superiór a la de la Kinect y con un coste significativamente inferior a la misma.

2. INTRODUCCIÓN

2.1. Objeto del Proyecto

El objetivo del presente trabajo es el diseño y la implementación de un sistema de visión para la adquisición y procesamiento de imágenes con el propósito de reconstruir objetos en 3D. Con ello se desea poder detectar posibles errores de impresión 3D usando como referencia una impresora de plástico (ABS/PLA) modelo Prusa Iteración 3.

Se evaluará la eficiencia del método comparando su precisión/coste frente a un sistema de reconstrucción de objetos ya existente y comercial como es el módulo de cámara Kinect v2, pretendiendo obtener una resolución similar abaratando considerablemente el coste.

2.2. Objetivos del Proyecto

El principal objetivo de este proyecto es aplicar el proceso de reconstrucción 3D de un objeto para la detección de errores en piezas imprimidas mediante la técnica de adición de plástico (ABS/PLA) o comúnmente llamada impresión 3D. Los errores a detectar son los más frecuentes que suelen aparecer en la técnica citada anteriormente, que son el despegue entre las capas de material y la deformación de la geometría exterior de la pieza.

Es posible que no sea factible arreglar todos los errores que se puedan detectar en una pieza imprimida en 3D, pero dado el alto grado de automatización del proceso es de gran importancia el hecho de poder detectar cuándo se produce un error y dónde está dicho error en la pieza para poder evaluar si puede ser reparado, si debido a su magnitud puede ser ignorado sin afectar a la calidad final de la pieza o si, por el contrario, el error es de tal elevada magnitud que se debe desechar la pieza. La evaluación de los errores detectados quedaría fuera del presente proyecto debido, principalmente, al tiempo que esto acarrearía, aunque podría resultar de gran interés para el desarrollo de la tecnología de impresión 3D a fin de poder obtener un mayor grado de automatización en la misma.

El proyecto tratará sobre el diseño de un sistema de visión con el cual se captarán imágenes de un objeto desde diferentes planos. Se tomarán dos planos perpendiculares entre si que contengan la pieza a reconstruir para tomar las imágenes, con ello se pretende tener información de cada punto perteneciente al objeto. Después de su captura, las imágenes serán almacenadas y procesadas de forma que se pueda obtener la reconstrucción parcial del objeto en 3D.

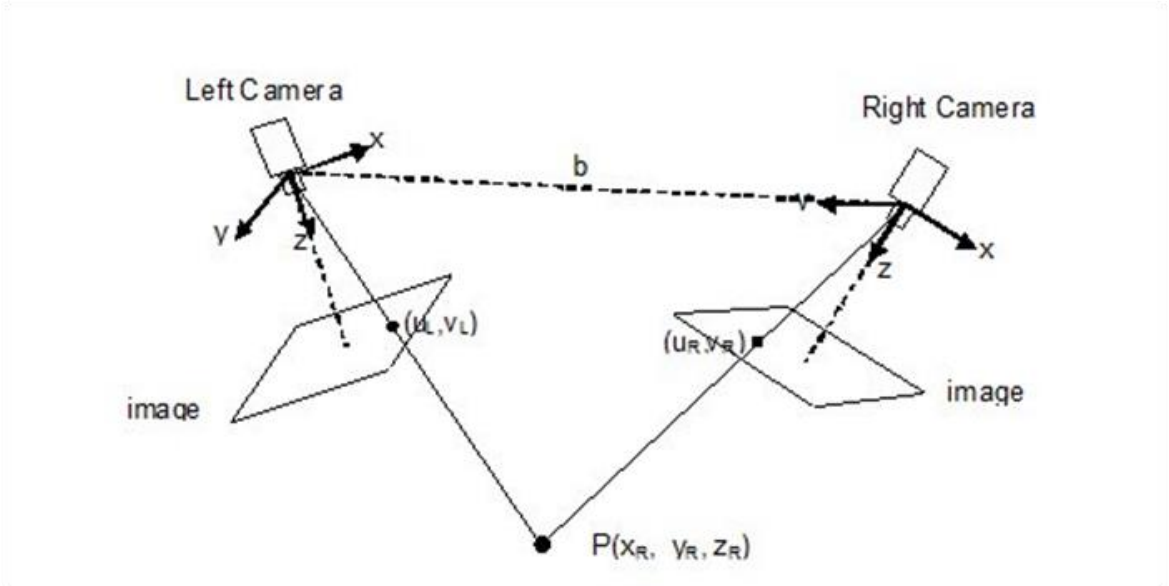


FIG. 1 PROYECCIÓN 3D DE UN PUNTO MEDIANTE DOS IMÁGENES

3. ANTECEDENTES, MOTIVACIÓN Y JUSTIFICACIÓN

3.1. Antecedentes

La impresión 3D es un grupo de tecnologías de fabricación por adición donde un objeto tridimensional es creado mediante la superposición de capas sucesivas de material. Las impresoras 3D son por lo general más rápidas, más baratas y más fáciles de usar que otras tecnologías de fabricación por adición, aunque como cualquier proceso industrial, estrarán sometidas a un compromiso entre su precio de adquisición y la tolerancia en las medidas de los objetos producidos. Las impresoras 3D ofrecen a los desarrolladores de producto, la capacidad para imprimir partes y montajes hechas de diferentes materiales con diferentes propiedades físicas y mecánicas, a menudo con un simple proceso de montaje. Las tecnologías avanzadas de impresión 3D, pueden incluso ofrecer modelos que pueden servir como prototipos de producto.

Existen un gran número de tecnologías para la impresión 3D, sus principales diferencias se encuentran en la forma en la que las diferentes capas son usadas para crear piezas.

Tipo	**Tecnologías**	**Materiales**
Extrusión	Modelado por deposición fundida (FDM)	Termoplásticos (por ejemplo PLA, ABS), HDPE, metales eutécticos, materiales comestibles
Hilado	Fabricación por haz de electrones (EBF)	Casi cualquier aleación
	Sinterizado directo de metal por láser (DMLS)	Casi cualquier aleación

	Fusión por haz de electrones (EBM)	Aleaciones de titanio
	Sinterizado selectivo por calor (SHS)	Polvo termoplástico
	Sinterizado selectivo por láser (SLS)	Termoplásticos, polvos metálicos, polvos cerámicos
	Proyección aglutinante (DSPC)	Yeso
Laminado	Laminado de capas (LOM)	Papel, papel de aluminio, capa de plástico
Fotoquímicos	Estereolitografía (SLA)	fotopolímero
	Fotopolimerización por luz ultravioleta (SGC)	fotopolímero

Actualmente existe una gran demanda en el mercado de impresoras 3D con tecnología de modelado por deposición fundida, según citaba Hilda Gómez (2014) en la web de noticias www.dealerworld.es *"Este año se venderán 44.000 impresoras 3D domésticas, y un millón en 2018"*, esto nos hace pronosticar una importante proyección económica de dicho mercado y asimismo nos hace pensar cuales son los posibles defectos de esta tecnología.

Se pueden establecer 4 categorias para los principales errores de impresión 3D:

1. Deformación de la Geometría
2. Despegue entre Capas
3. Problemas con el Acabado Superficial
4. Problemas en la extrusión

FIG. 2 – PIEZA CON ERROR TIPO 1

FIG. 3 – PIEZA CON ERROR TIPO 1

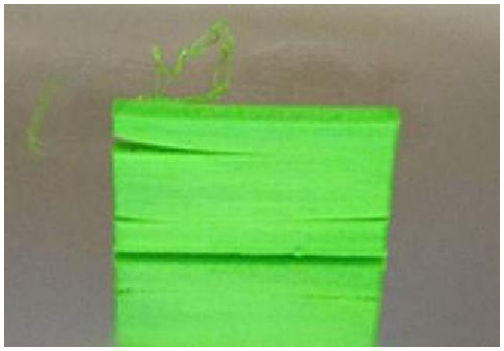


FIG. 4 – PIEZA CON ERROR TIPO 2

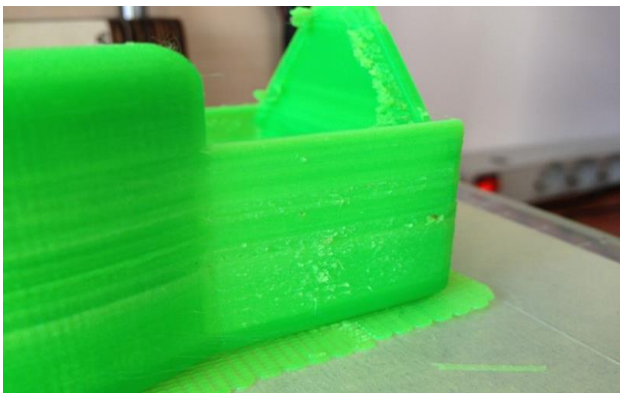


FIG. 5 – PIEZA CON ERROR TIPO 3

FIG. 6 – PIEZA CON ERROR TIPO 4

Los distintos errores mencionados pueden tener varias procedencias o causas distintas, ya que se trata de un sistema bastante complejo en el que intervienen muchos factores, pese a ello se pueden determinar unas causas principales para cada tipo de error, a continuación se procede a explicarlas.

Los errores denominados como Tipo 1 son básicamente debidos a un mal enfriamiento de las capas que produce que el material se dilate demasiado, provocando una deformación que puede ir acumulándose capa tras capa, llegando a producir una pieza visiblemente deformada (Fig. 2). También pueden ser debidos a que el sistema de transmisión del cualquier eje esté mal ajustado o bien a la pérdida de pasos (error muy común) de cualquiera de los motores paso a paso o "Step by step" que forman los ejes de la impresora (Fig. 3).

Los errores denominados como Tipo 2 tienen varios factores principales como son la mala adherencia entre las capas, la baja de temperatura del entorno de la pieza y las tensiones internas que se suelen crear debidas, básicamente, a las geometrías con muchas aristas pronunciadas (Fig. 4).

Los errores denominados como Tipo 3 son provocados, generalmente, por una mala temperatura en el extrusor para el tipo de material a imprimir, dejando así grumos en la superfície de las piezas (Fig. 5).

Por último, los errores denominados como Tipo 4 son debidos en gran parte a un mal ajuste del extrusor que puede dejar escapar material una vez se pausa la impresión o a un fallo en algún eje con la máquina funcionando por impresión desde ordenador y se haya parado el envío de comandos G-code (código con el que el software de impresión se comunica con la impresora) dejando así la máquina parada pero manteniendo la temperatura de la punta del extrusor, haciendo que el material fluya por gravedad sin detenerse en los engranajes que lo retienen por el hecho de encontrarse en estado líquido.

La existencia de estos errores dificulta altamente la automatización de las impresoras 3D para la producción en cadena o producción industrial, ya que actualmente no existe ninguna forma de poder detectar los errores que se producen de forma automática y en el caso de intentar una producción en masa nos encontraríamos con un procentaje demasiado elevado de piezas para desechar, con el gasto de material, electricidad y tiempo que ello conlleva, hecho inasumible e inaceptable para una empresa.

De entre los diferentes tipos de errores que se han hablado en este proyecto se van a detectar en las piezas los dos primeros (errores debidos a la deformación de la geometría y los errores debidos al despegue entre capas), ya que consideramos que se pueden abordar de forma directa con el sistema de visión para la reconstrucción de objetos en 3D que en este proyecto se trata.

3.2. Motivación

El principal motivo de realización de este proyecto es la aplicación tanto teórica como práctica por parte del proyectista de los conocimientos que a lo largo del Grado en Ingeniería en Tecnologías Industriales ha ido adquiriendo.

Es por eso que pasa a ser motivo de expresa mención la cantidad de diferentes áreas abarcadas por este proyecto, tratando disciplinas tanto de diseño mecánico, visión por computador, programación en C#, programación en Python, programación en C++, calibrado de cámaras y uso de sistemas embebidos, lo cual ofrece al proyectista un completo desarrollo profesional como ingeniero.

Es esta multidisciplinariedad, en conjunto con la misión de llevar a cabo una tarea con prespectiva industrial, la que lleva al elaborador de este proyecto a tener un interés y una motivación especial en realizarlo.

Cabe destacar también el auge tanto económico como tecnológico que está sufriendo en los últimos 5 años la técnica de impresión 3D con tecnología de modelado por deposición fundida y la proliferación de empresas (sobretodo en España) relacionadas con el mundo de la impresión 3D que llama a centrar la visión en busca de avances tecnológicos en ese ámbito para suplir las carencias existentes e incrementar su desarrollo.

“Madrid, capital mundial de la impresión 3D”, Pilar Ortega, Diario Metropoli (2015).

“La impresora 3D se convierte en un fenómeno de masas”, Javier Martín, El País (2013).

“3D printing comes to the high street”, Carol Lewis, The Times (2014).

3.3. Justificación

El interés fundamental de este trabajo radica en el ahorro tanto de material (ABS/PLA), energía y tiempo que supone poder detectar los errores de impresión en el momento en el que estos se están produciendo. Es posible que dichos errores no puedan ser reparados aún siendo detectados en el momento de producirse, pero el hecho de detectar el error en una pieza que va a ser desechada conlleva el ahorro ya citado anteriormente por no tener que terminar la pieza.

Se encuentra pues una diversidad de intereses en este proyecto, a saber:

- Interés tecnológico, ya que actualmente no existe tecnología alguna que pueda ser capaz de detectar un error de impresión 3D in situ.
- Interés industrial, puesto que la detección de errores mientras la máquina trabaja proporcionaria un mayor grado de automatización a la misma, hecho de gran relevancia para una posible industrialización de la tecnología de impresión 3D con tecnología de modelado por deposición fundida.
- Interés económico, debido al ahorro que se produce cuando se dejan de desechar piezas tanto en material plástico, como en electricidad por el tiempo de trabajo perdido que se ahorra a la máquina como por ese mismo tiempo que puede ser utilizado en la producción de otra pieza. También cabe destacar en este apartado que el hecho de ahorrar tiempo no sólo afecta a la máquina en cuanto a producción a corto plazo, sino que también afecta a la producción total neta de la máquina, pudiendo tener un mayor número de horas útiles durante su ciclo de vida.

Se puede realizar un cálculo a estima del ahorro correspondiente al material desechado y a la energía perdida. Obteniendo un tiempo de impresión medio por volumen de pieza imprimida y un volumen de desecho de una pieza con un error detectado cuando se lleva imprimido la mitad del volumen total (usando como ejemplo el plano 2, apoyo de la cámara del plano XZ) y un precio medio de impresión por hora de trabajo (tras consultar a varios servicios profesionales de impresión 3D) se llega a concretar un ahorro por pieza de:

- Material:

$$10\left(\frac{euros}{hora}\right) * 763\left(\frac{horas}{dm^3}\right) * 0.0026213(dm^3) \;=\; 20\; euros$$

- Electricidad:

$$0.360(kW) * 0.12458\left(\frac{euros}{kWh}\right) = 0.045\;(\frac{euros}{hora})$$

Teniendo en cuenta que para la realización de la pieza se tardó aproximadamente unas 4 horas, se obtiene un consumo de electricidad de:

$$0.045 \left(\frac{euros}{hora}\right) * 4 \, (horas) = 0.18 \, euros$$

Lo que nos da un ahorro total por pieza de:

$$20 \, (euros) + 0.18 \, (euros) = 20.18 \, euros$$

Suma que, a priori, puede carecer de importancia pero que si se piensa en la tecnología de impresión 3D como un proceso industrial dicha suma pasaria a ser un factor determinante en la viabilidad de dicha producción.

4. NORMATIVA

La normativa aplicable al presente proyecto consta de las siguientes normas/leyes/recomendaciones:

- Normativa marco de Trabajos Fin de Grado y Fin de Máster de la Universitat Politécnica de Valencia.
- Recomendaciones sobre la presentación de Trabajo Final de Grado en la Escuela Técnica Superior de Ingenieros Industriales.
- Directiva europea 93/68/EEC de cumplimiento de mínimos requisitos de seguridad, tanto legales como técnicos.
- Directiva europea 2002/95/CE de Restricción de Sustancias Peligrosas en aparatos eléctricos y electrónicos.
- Normativa de Compatibilidad Electromagnética de conformidad con la Comisión Federal de Comunicaciones (Federal Communications Commission, FCC) de Estados Unidos.

FIG. 7 – LOGO DE CONFORMIDAD EUROPEA

FIG. 8 – LOGO DE LA COMISIÓN FEDERAL DE COMUNICACIONES

FIG. 9 – LOGO DE CUMPLIMIENTO DE RESTRICCIÓN DE SUSTANCIAS PELIGROSAS

5. ÁMBITO DE APLICACIÓN Y RANGO DE SOLUCIONES

5.1. Elección del sistema de adquisición de imágenes

Para la adquisición de las imágenes se plantearon varias alternativas:

- Webcams
- Módulos de cámara para Raspberry Pi

Las Webcams cuentan con una conexión muy sencilla mediante puerto USB, no requieren de instalación previa salvo los Drivers que se instalan, generalmente, de forma automática una vez se conecta el dispositivo, por tanto solo se debe tener en cuenta la resolución/precio que ofrecen, dado que los módulos de cámara para Raspberry Pi cuentan con una resolución máxima de 5 Megapíxeles se buscarán Webcams de la misma resolución para poder hacer una comparativa equánime.

Después de consultar diferentes fuentes se determina que el valor medio de una Webcam de 5 Megapíxeles es de 42,1 euros, por tanto el coste total del sistema de visión sería de 84.2 euros, ya que se necesitan dos cámaras. Este sistema cuenta con la desventaja de ser totalmente dependiente de un ordenador tanto para la adquisición como para el posterior tratamiento de las imágenes, el cuál podría incrementar significativamente el coste del sistema.

Por otro lado, los módulos de cámara para Raspberry Pi se conectan mediante un bus a la conexión habilitada para ello en la Raspberry Pi, no necesitan de instalación previa más que la habilitación del puerto una vez se ha inicializado el sistema operativo por primera vez.

Por tanto, para este sistema, sería necesario un módulo de cámara por cada plano, una Raspberry Pi modelo 2 (ya que cuesta lo mismo que el modelo anterior y ofrece mayores prestaciones) y un multiplexor para conmutar la adquisición de las imágenes entre los dos módulos. El coste de los módulos de cámara es de 25 euros por cada uno, la Raspberry Pi 2 tiene un coste de 52 euros (incluyendo tarjeta SD) y el del multiplexor es de 76 euros, con un total de 178 euros.

La ventaja que plantea este sistema frente al anterior expuesto es la total dependencia del mismo, ya que la misma Raspberry Pi 2 tiene la capacidad para adquirir y procesar las imágenes sin necesidad de procesamiento externo.

Una vez planteadas las dos opciones se opta por realizar el sistema de visión mediante módulos de cámara para Raspberry Pi, ya que aunque con la misma resolución el coste del sistema es, a priori, superior al formado por las Webcams, la flexibilidad que ofrece trabajar con el sistema operativo Raspbian proporciona una mayor libertad para el proyectista, además del mayor aprendizaje y aplicación de conocimientos que supone la puesta en marcha desde cero de la Raspberry Pi.

FIG. 10 – MÓDULO DE CÁMARA PARA RASPBERRY PI 2

5.2. Preparación de la Raspberry Pi y librerías a usar

Para la realización de este proyecto se ha sopesado usar la placa Raspberry Pi 2 B o la anterior versión de la misma, la Raspberry Pi B+ .El nuevo modelo ofrece todas las características del anterior con una mejora de la memoria del doble de capacidad (la versión 2 B tiene 1 Gigabyte, frente a los 512 Megabytes de la B+) y consta también de la mejora de poseer 4 núcleos de procesador en el caso de la 2 B, en lugar de 1 núcleo que posee la B+.

Todas estas mejoras son incrementadas en rendimiento económico, ya que los dos modelos cuestan lo mismo, aproximadamente 36 euros. Es por estas razones que se ha optado por usar la versión Raspberry Pi 2 B.

Una vez elegida la placa, se procede a elegir las librerías a usar para la realización del presente trabajo. Para ello se necesitará:

- Librería de tratamiento de imagen
- Librería para trabajar con vectores y matrices
- Librería para la inicialización de los pines de E/S de la placa

Para la librería de inicialización de los pines de E/S de la placa se va a utilizar la librería "RPIO.rpi", ya que está hecha específicamente para trabajar con Raspberry Pi.

La librería a utilizar para vectores y matrices será la librería "numpy", ya que permite crear vectores y matrices de cualquier forma y tamaño de una manera muy sencilla e intuitiva, a parte de poseer funciones que podrían ser de gran ayuda para operar con matrices y vectores en el caso de ser necesarias.

Por último, la librería para el procesamiento de las imágenes será la librería "OpenCV", exactamente la versión 3.0.0 de la misma, ya que cuenta con múltiples herramientas que serán de gran ayuda para la realización de éste proyecto, además de disponer de una amplia documentación, tutoriales y círculos de trabajo dónde encontrar ayuda si fuera menester. Además, dicha librería tiene una licencia de uso BSD, que otorga amplios permisos al usuario para su uso.

5.3. Diseño del sistema de visión

Partiendo de una estructura cúbica en la cual se encuentra montada la impresora 3D se opta por colocar las cámaras en la parte posterior del carro del extrusor, donde se encuentran los controles que determinan la posición de las coordenadas "X" y "Z" de la máquina, ya que en la parte delantera no queda disponible mucho espacio y condicionaría demasiado el diseño e incluso podría suponer que las cámaras se encontraran demasiado cerca, pudiendo perder así puntos de la pieza a reconstruir.

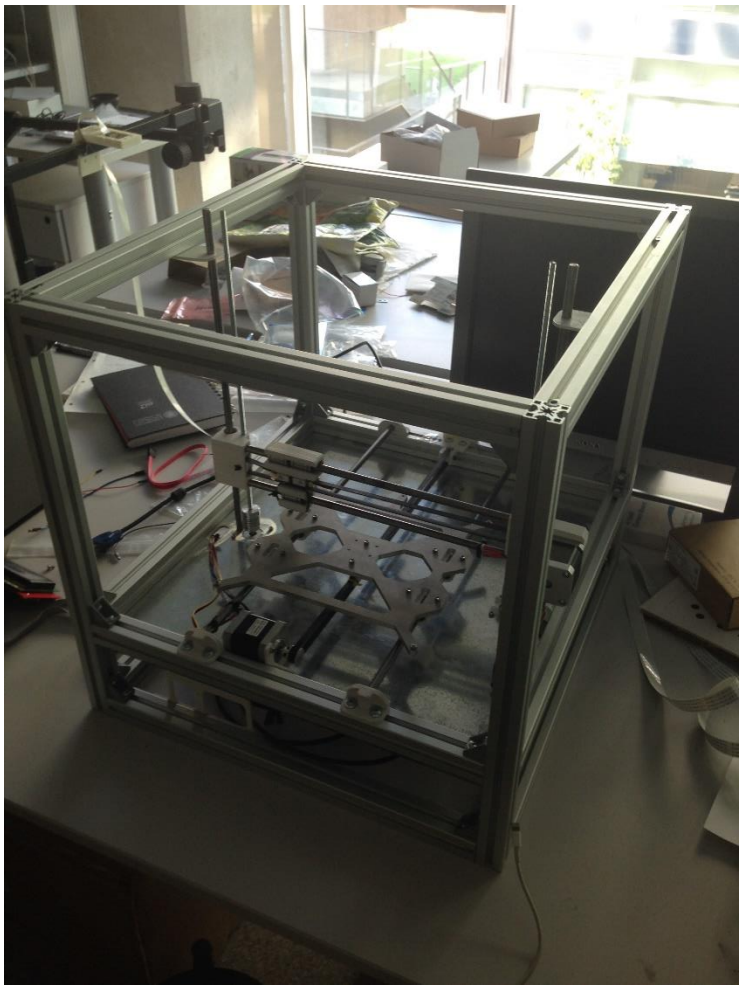


FIG. 11 – IMPRESORA PRUSA I3 MONTADA EN SU ESTRUCTURA

Una vez descartada la parte anterior al carro del extrusor se plantean dos ángulos de visión que aporten toda la información necesaria de la pieza para poder hacer su reconstrucción en 3D y que aporten la información característica que permita detectar los errores de impresión que se persiguen (ya comentados en otro apartado).Para ello se busca un plano que nos

otorgue información sobre el contorno de la pieza, determinándose así una vista superior de la misma.

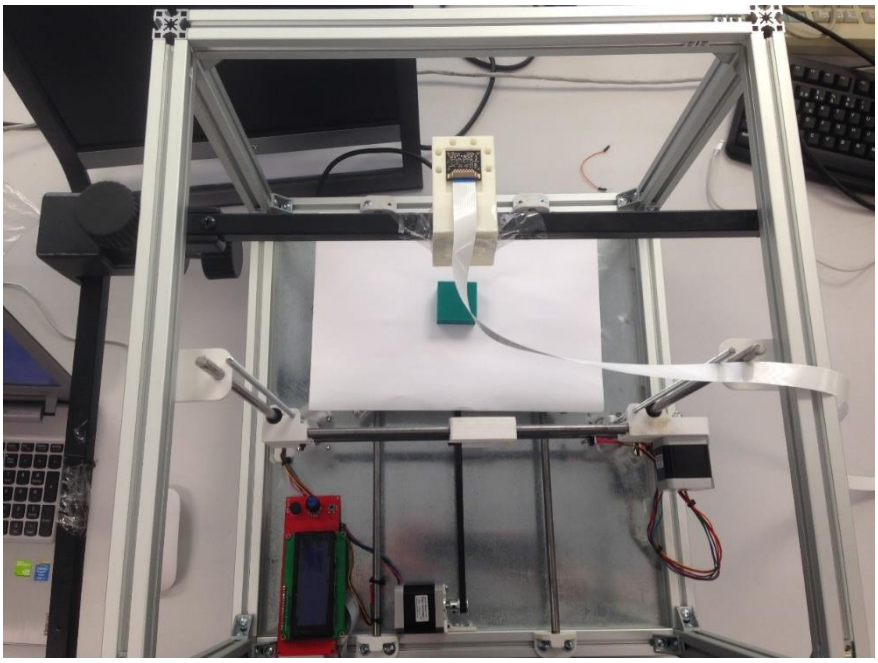


FIG. 12 – LOCALIZACIÓN DE LA CÁMARA SUPERIOR

Por otro lado, se busca obtener información sobre el perfil y las capas de la pieza, por lo cual se determina que cualquiera de los tres planos perpendiculares al plano superior y que ocupan tanto ambos lados de la impresora como la parte trasera de la misma son válidos, eligiéndose el último por disponerse de un espacio mayor para colocar la cámara y por la mayor proximidad a la cual puede ser colocada la pieza de la cámara en el caso de ser necesario, ya

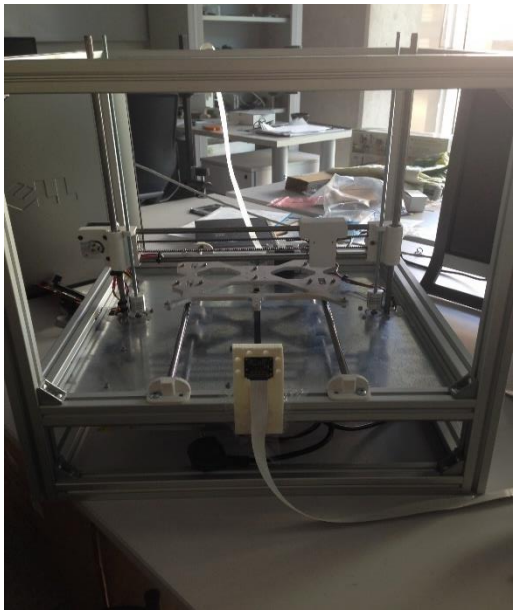


FIG. 13 – VISTA DE LA CÁMARA FRONTAL

que la cama de impresión puede moverse hasta llegar a la parte posterior de la impresora pero no puede moverse hacia los laterales, ya que para moverse en esa dirección durante la impresión ya está el carro del extrusor que, como se ha comentado anteriormente, determina la coordenada "X" a la que se situa la máquina.

Por último se determina el método de sujeción de las cámaras, para lo cual se elige diseñar mediante software de diseño 3D, concretamente mediante Inventor (creado por la compañía Autodesk), unas piezas de soporte para su acople en perfiles cuadrados como los que componen la estructura dónde se encuentra montada la impresora. En dichos soportes la cámara irá encajada en una posición diseñada para ello, aunque también podría anclarse con tornillos, ya que los soportes cuentan con agujeros que encajan con los que posee el módulo de cámara (para mayor detalle sobre los soportes consultar los planos 1 y 2).

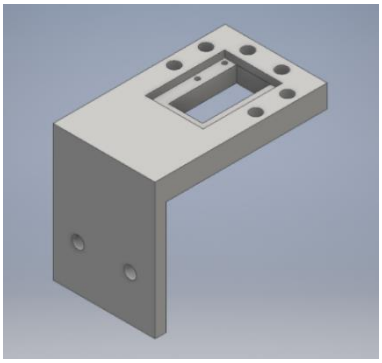


FIG. 14 – APOYO CÁMARA SUPERIOR

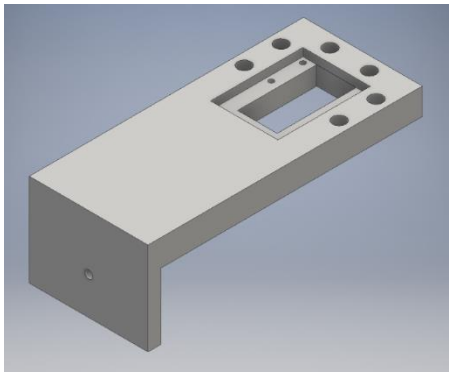


FIG. 15 – APOYO CÁMARA FRONTAL

Quedando las dos cámaras acopladas sobre la estructura de forma que es posible obtener toda la información necesaria para poder detectar errores de deformación gracias a la cámara superior, que comprobará el contorno de la pieza y a la cámara frontal, que será capaz de detectar errores de despegue entre capas de material. A su vez se consigue la información necesaria para poder reconstruir en 3D la pieza, obteniendose finalmente el siguiente sistema:

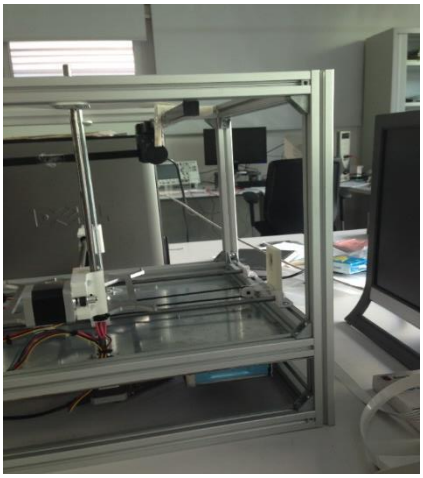


FIG. 16 – ESTRUCTURA CON LAS DOS CÁMARAS ACOPLADAS

5.4. Extracción de la pieza teórica

Cabe destacar que el sistema empleado funciona de manera idéntica tanto para la cámara frontal como para la cámara superior, por lo tanto, desde ahora en adelante en todos los apartados se mencionará la implantación del sistema para una cámara (la cámara superior), siendo trivial la implantación del mismo para la otra cámara (cámara frontal).

La necesidad de obtener las coordenadas exactas de la pieza teórica a imprimir es debida a la comparación que se realizará posteriormente con las mismas coordenadas de la pieza real, extraídas de las imágenes de las cámaras.

En este caso la extracción de las coordenadas de la pieza teórica se hace mediante el archivo de Código G o también llamado Código Máquina, que es mandado por el software de impresión 3D, el Repetier. En dicho archivo se pueden encontrar una serie de órdenes que debe seguir la impresora, como el calentamiento a determinada temperatura del extrusor, la fijación de unidades en milímetros o el tipo de movimiento y la cantidad de material que debe extruir cuando realice dicho movimiento. En la Figura 17 se puede observar un fragmento de dicho código.

```
G21 ; set units to millimeters
M107
M104 S200 ; set temperature
G28 ; home all axes
G1 Z5 F5000 ; lift nozzle
M109 S200 ; wait for temperature to be reached
G90 ; use absolute coordinates
G92 E0
M82 ; use absolute distances for extrusion
G1 F1800.000 E-1.00000
G92 E0
G1 Z0.350 F7800.000
G1 X72.384 Y65.741 F7800.000
G1 E1.00000 F1800.000
G1 X74.137 Y64.255 E1.07480 F1080.000
G1 X76.292 Y63.453 E1.14960
G1 X77.500 Y63.343 E1.18906
```

FIG. 17 – FRAGMENTO DE CÓDIGO G

El descifrado del Código G es muy sencillo y el significado de cada orden se puede encontrar en la página (http://reprap.org/wiki/G-code/es).

La pieza teórica será extraída del código mediante un programa de creación propia que transforma las coordenadas dadas a la máquina en cada capa en un formato entendible y accesible de tipo (x, y, z), donde “x”, “y” y “z” corresponden a las coordenadas de cada punto donde debe ir la máquina durante la realización de la pieza.

5.5. Calibración de las cámaras

La calibración de las cámaras se hará mediante el módulo de calibración de cámaras de la librería OpenCV mediante el uso de un patrón de círculos asimétricos como el de la Figura 18. La calibración se realiza mediante la toma de imágenes del patrón mencionado colocándolo en diferentes ángulos respecto la cámara con la tónica de que se observen de forma lo más clara posible los puntos que lo componen y que puedan verse todos ellos en cada imagen.

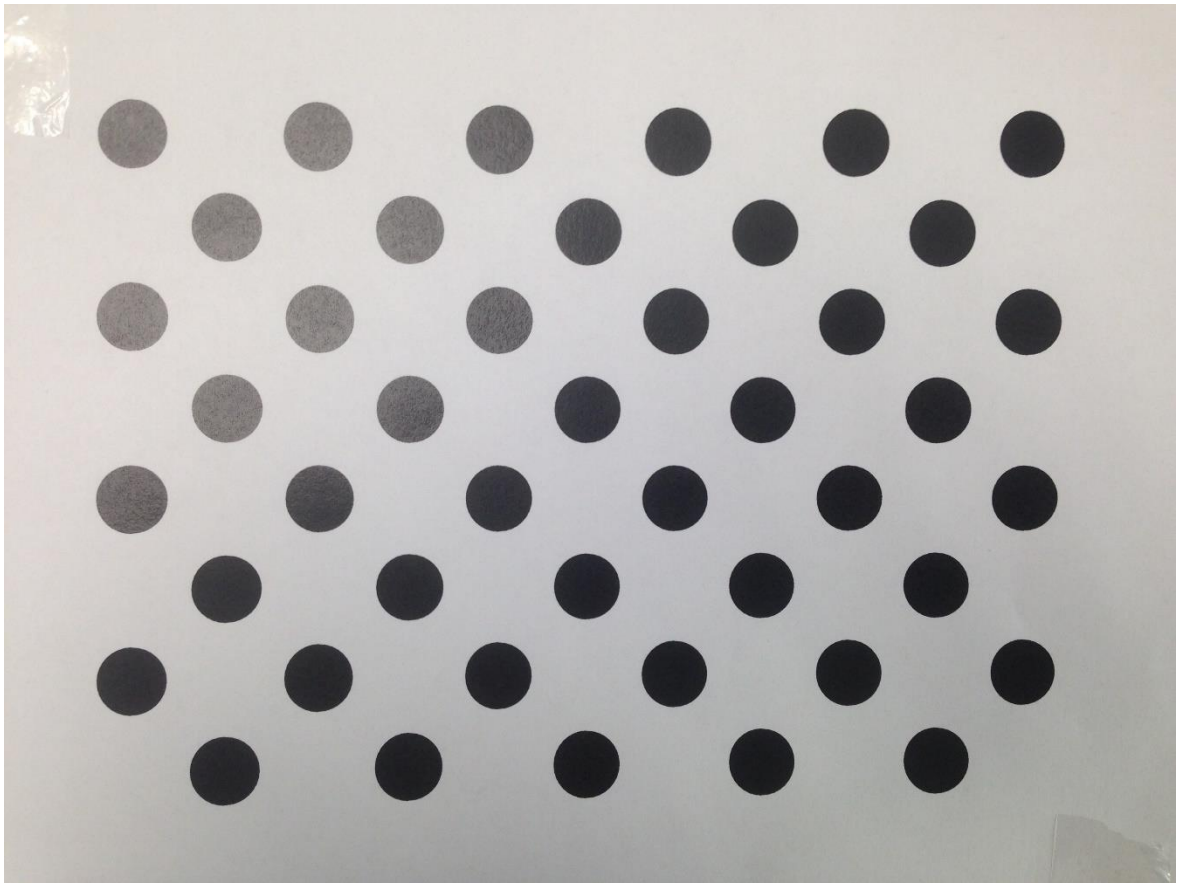


FIG. 18 – PATRÓN USADO PARA LA CALIBRACIÓN DE LAS CÁMARAS

Con ello se pretende obtener las matrices de parámetros de la cámara para, mediante una función también de la librería OpenCV se pueda obtener una proyección de cada punto del mundo real que sea captado por la cámara en el sistema de coordenadas del mismo haciendo una correspondencia a un píxel determinado de la imagen.

La calibración se realizará para cada cámara de manera independiente, usando el mismo tipo de patrón y las mismas funciones de calibración para ambas.

5.6. Medición en las imágenes

La medición en las imágenes será posible gracias a la calibración de cada cámara mencionada en el apartado anterior. Una vez extraídas las matrices de parámetros de la cámara podrá ser posible la proyección de un punto perteneciente al mundo y que esté dentro de la imagen haciendo una relación con un píxel de la misma.

Las matrices de parámetros de la cámara corresponden a la matriz de parámetros intrínsecos, dónde residen los parámetros de la propia cámara y coeficientes de distorsión de la imagen y a la matriz de parámetros extrínsecos, dónde se puede encontrar los datos pertenecientes a la rotación y translación de la imagen.

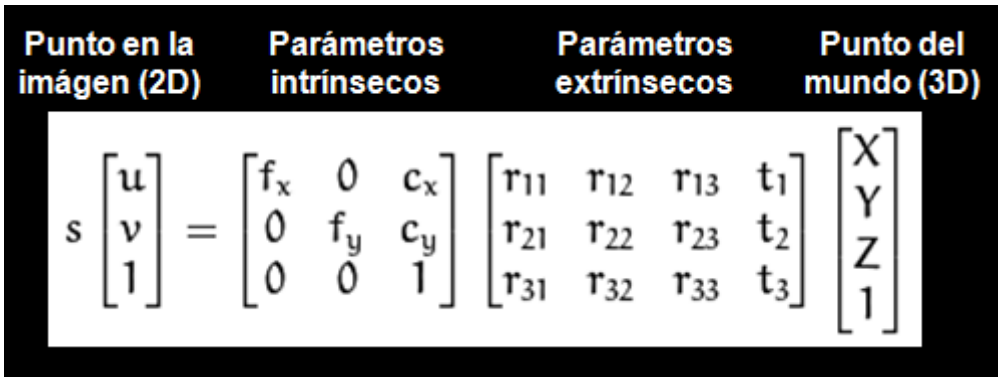


FIG. 19 – MODELO MATEMÁTICO PARA LA PROYECCIÓN DE UN PUNTO

A partir de estas matrices se podrá construir el modelo matemático que permite, dado un punto del mundo captado por la cámara definido por sus coordenadas (x, y, z), obtener su correspondiente píxel en la imagen. Con esto la medida en la misma imagen se convierte en una cuestión bastante trivial, ya que dadas las coordenadas de dos puntos separados una distancia en el mundo (y captados por la cámara) será posible obtener los píxeles a los que pertenece cada punto y medir, así, la distancia entre ellos en la imagen.

5.7. Extracción de la pieza real

El paso de conocer las coordenadas reales de la pieza es fundamental, ya que serán las coordenadas que, posteriormente, se compararán con las coordenadas teóricas y permitirá detectar si se ha producido un error durante la impresión.

Para determinar las coordenadas reales de la pieza es necesario conocer la gama a la que pertenece el color del material usado durante la impresión, crear un contraste fuerte entre el fondo de la imagen y mantener una iluminación adecuada, intentando aislar el sistema de la luz exterior (ver Figura 20), para así poder obtener un color puro y unos bordes definidos. Con ello, mediante un programa adaptado de un ejemplo de búsqueda de formas geométricas de un color (se puede encontrar el programa original en http://robologs.net/2015/01/25/deteccion-de-triangulos-con-opencv-y-python/) , se pretende detectar la pieza utilizando la diferencia de color entre ella y el resto de elementos de la imagen, pudiendo delimitar los bordes de la pieza y las coordenadas que los componen.

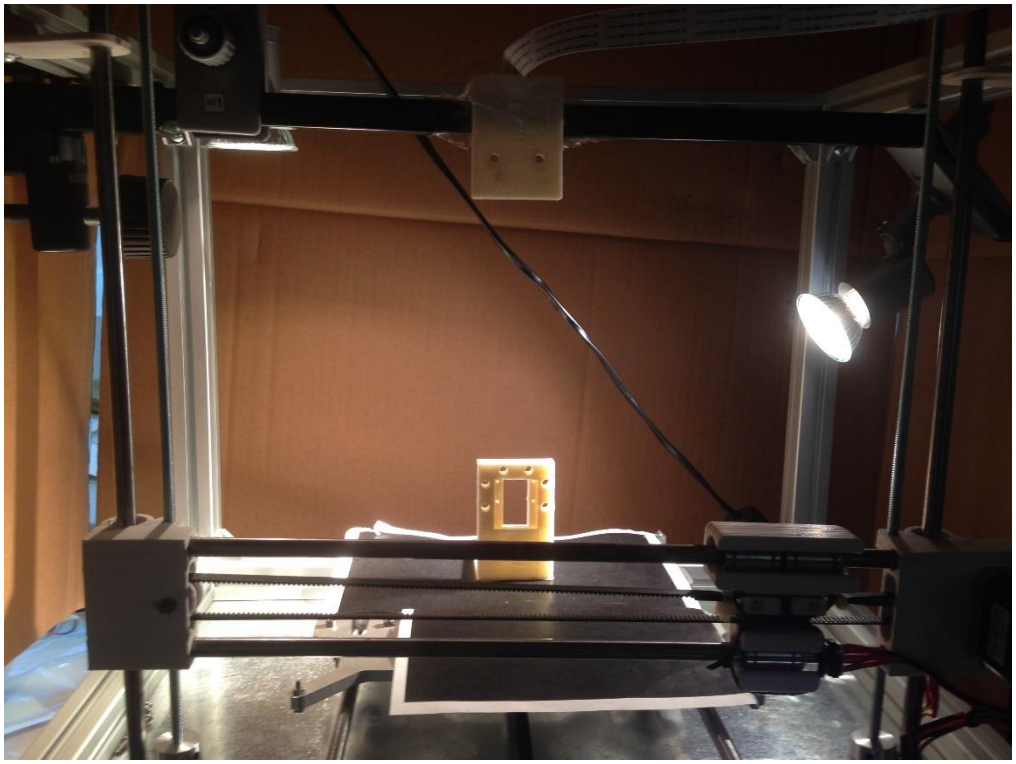


FIG. 20 – SISTEMA AISLADO CON ILUMINACIÓN DIRECTA Y PIEZA BLANCA SOBRE FONDO NEGRO

De este modo, una vez conocidas las coordenadas de la pieza real, que en este caso se trata de píxeles directos pertenecientes a la imagen, es posible hacer una comparación entre dónde debería estar cada punto de la pieza y dónde ha sido imprimida.

5.8. Proyección de la pieza teórica en las imágenes

La proyección de la pieza teórica en las imágenes se realiza mediante una función de la librería OpenCV llamada "projectPoints", la cuál dadas las matrices de parámetros intrínsecos de la cámara y extrínsecos de la imagen patrón usada en la calibración y un punto teórico de la pieza que se encuentre dentro del cuadro de la imagen, definido por sus coordenadas en 3D (respecto al sistema de coordenadas del mundo de la cámara), devuelve sus coordenadas en 2D pertenecientes al píxel donde se encuentra proyectado dicho punto (respecto al sistema de coordenadas de la imagen) .

Aplicando ésta función para todos los puntos que componen la pieza teórica, se obtiene dicha pieza proyectada en la imagen, donde posteriormente se podrá comparar con la pieza real para la detección de los errores.

Para la proyección de los puntos se ha empleado la siguiente ecuación:

$$\begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = [P_{3x4}] * \begin{bmatrix} 0 & 1 & 0 & -50 \\ 1 & 0 & 0 & -46.1 \\ 0 & 0 & -1 & 55 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} O_x^m \\ O_y^m \\ O_z^m \\ 1 \end{bmatrix}$$

Dónde el vector columna 4x1 "O" son las coordenadas del punto a proyectar expresadas en el sistema de coordenadas de la impresora. La matriz 4x4 contiene el cambio de coordenadas entre el sistema de coordenadas de la impresora y el sistema de coordenadas del mundo de la cámara (3 vectores 4x1 para traslación y 1 vector 4x1 para las coordenadas del origen del sistema de la impresora expresadas en el sistema del mundo de la cámara. La matriz P de tamaño 3x4 es dónde se encuentran agrupadas las matrices de parámetros (extrínsecos e intrínsecos) y el vector columna 3x1 son las coordenadas del píxel al que corresponde el punto proyectado expresadas en el sistema de coordenadas de la imagen.

5.9. Comparación entre la pieza teórica y la pieza real. Detección de errores en el plano

La comparación entre la pieza teórica (borde teórico) y la pieza real (borde real) se realizará mediante un programa elaborado por el autor de este proyecto y que tratará sobre una comparación punto a punto entre el borde extraído de la pieza real en la imagen y el borde teórico que ofrece el software de la impresora 3D mediante el código G. Dicho programa mostrará una medida porcentual de coincidencias entre bordes y imprimirá dichas coincidencias sobre la imagen real, pudiéndo observarse dónde se encuentran discrepancias entre bordes teóricos y reales, claro indicador de que existe un defecto en la pieza.

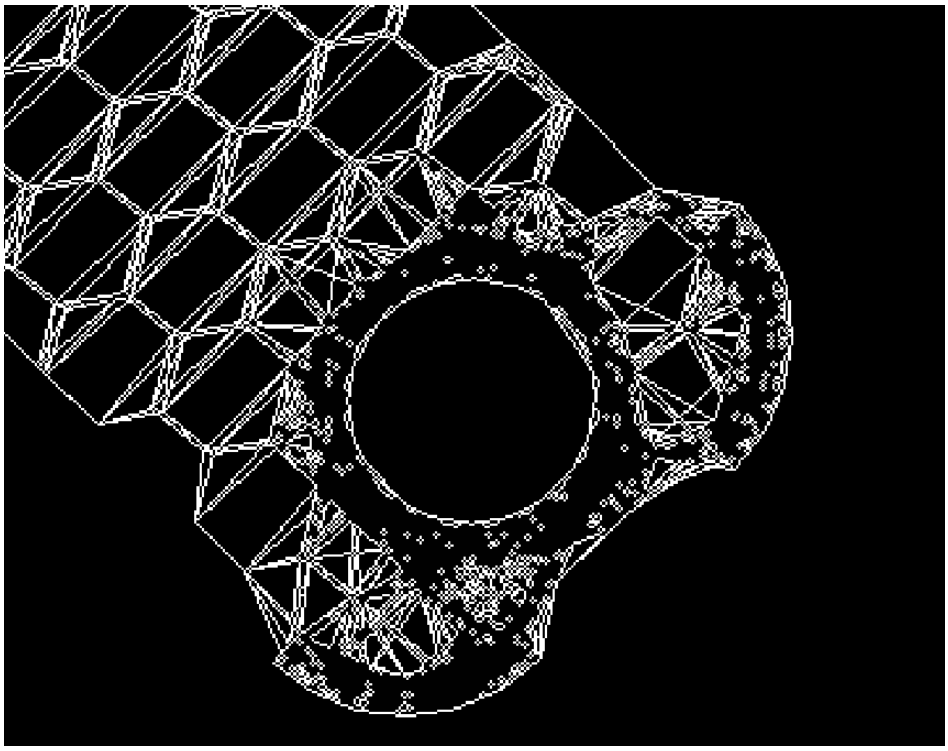


FIG. 21 – EJEMPLO, BORDE 1 A COMPARAR

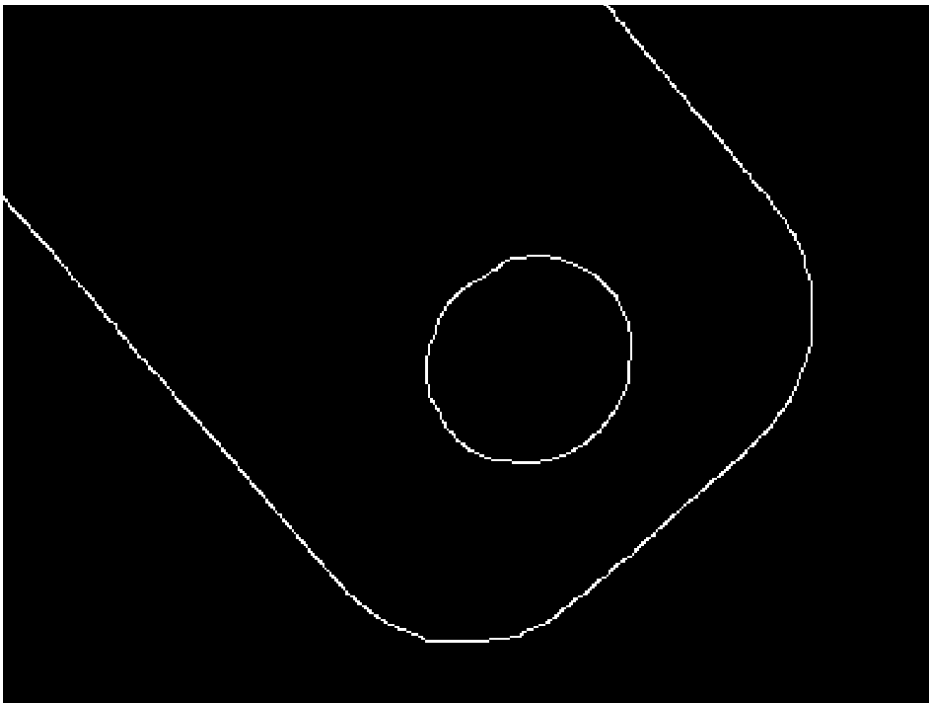


FIG. 22 – EJEMPLO, BORDE 2 A COMPARAR

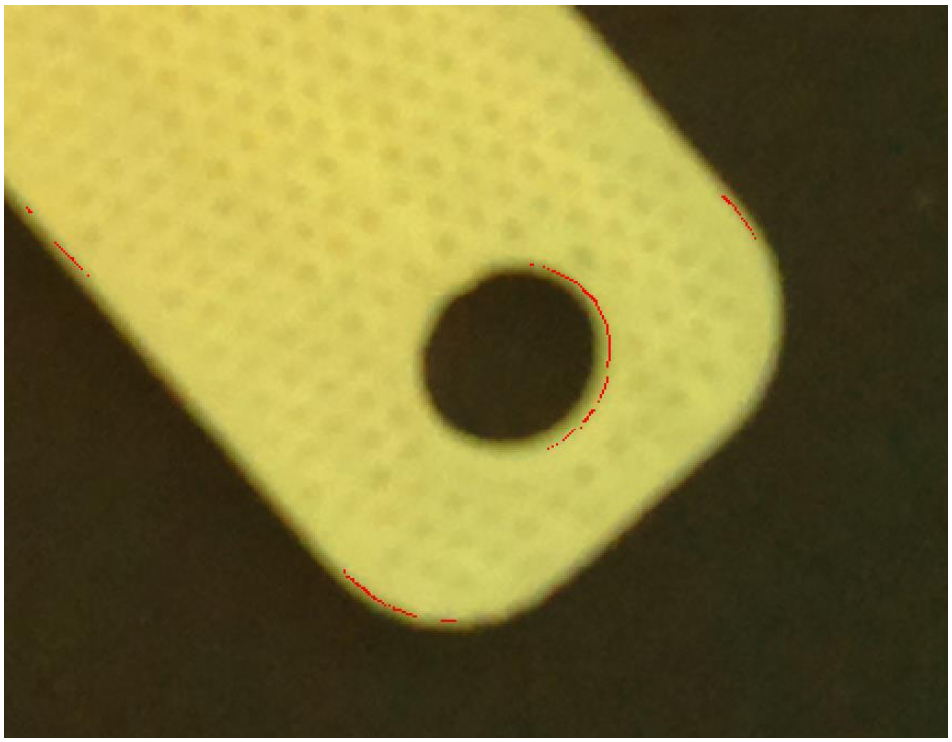


FIG. 23 – EJEMPLO, COINCIDENCIAS ENTRE BORDE 1 Y BORDE 2

5.10. Reconstrucción 3D de la pieza

La reconstrucción 3D de la pieza se realizará mediante la reconstrucción de las coordenadas de los puntos obtenidos como borde real, teniendo en cuenta que dichos puntos se encuentran todos situados en el mismo plano y que dicho plano es conocido, ya que corresponde a la altura de la capa que se está imprimiendo.

Mediante un programa de implementación propia se realizará la extracción de los puntos del borde real de la pieza desde la imagen tomada por la cámara y se hará una transformación (o proyección inversa) de dichos puntos desde sus coordenadas en píxeles de la imagen a coordenadas 3D del mundo real, para ello se utilizarán las matrices de parámetros extrínsecos e intrínsecos de la cámara y una matriz de cambio de coordenadas desde el plano donde se encuentran los puntos a el plano XY del origen de coordenadas del mundo de la cámara.

Una vez aplicada la transformación mencionada se obtendrán las coordenadas "(x, y, z)" de cada punto del contorno real de la pieza, pudiéndose así reconstruir la pieza aplicando este procedimiento para las "n" capas que componen la pieza (dato suministrado por el software de la impresora).

La transformación se realizará mediante la siguiente ecuación:

$$\begin{bmatrix} O_x^w \\ O_y^w \\ 1 \end{bmatrix} = [Q_{3x3}] * \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix}$$

Dónde el vector columna "O" será el vector de coordenadas "X Y" de los puntos del plano (dado que su coordenada "Z" es conocida, ya que conocemos el plano en el que se encuentran) expresadas en el sistema de coordenadas del mundo de la cámara. La matriz 3x3 "Q" será la matriz invertida del producto entre la matriz de parámetros extrínsecos e intrínsecos eliminando la columna correspondiente a la "Z" (en la primera de ellas). Por último el vector columna "p" será el correspondiente a las coordenadas en píxeles de cada punto del contorno, referidas al sistema de coordenadas de la imagen.

El cambio de coordenadas del sistema del mundo de la cámara al sistema de la impresora se haría multiplicando el vector "O" por la inversa de la matriz 4x4 del apartado 5.8.

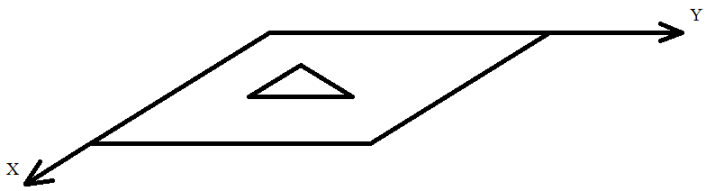


FIG. 24 – CONTORNO CONTENIDO EN UN PLANO

5.11. Comparativa con el sistema de visión Kinect v2

Se pretende comparar el sistema de visión diseñado en este proyecto con un sistema comercial, la cámara Kinect v2, cuyo coste es de aproximadamente 150 euros (149,99 euros exactamente), para ello se compararán parámetros como coste en relación a la resolución de la imagen, independencia del sistema y flexibilidad del mismo.

Empezando por el primer factor, por una parte está la cámara Kinect v2, cuya resolución de imagen es de 1080p (1944 píxeles de ancho por 1080 píxeles de alto) lo cual le otorga una resolución de imagen de 2.1 Megapíxeles, dando un parámetro de 71.429 euros por cada megapíxel. Por otra parte, el sistema de visión diseñado en este proyecto, cuyo coste es de 178 euros y cuya resolución es de 5 Megapíxeles, obteniendo así un parámetro de 35.6 euros por cada megapíxel. Por tanto, en la relación coste del sistema frente a resolución del mismo gana considerablemente el sistema de visión compuesto por los módulos de cámara de Raspberry Pi, por un poco menos de la mitad que el sistema Kinect v2.

En relación a la independencia del sistema hay que considerar que la cámara Kinect necesita de un procesador externo para el proceso de las imágenes y reconstrucción del objeto en 3D, mientras que el sistema con Raspberry Pi es totalmente independiente para procesar las imágenes y obtener el resultado final.

Por último y en relación a la flexibilidad del sistema cabe destacar que Raspberry Pi pertenece a la filosofía “Open Source”, que significa que tanto Software como Hardware pueden ser modificados y usados libremente por cualquier persona, sin ninguna restricción. Por otro lado, Kinect v2 pertenece a la compañía Microsoft, con patente registrada, conllevando unas limitaciones en cuanto a su uso.

FIG. 25 – CÁMARA KINECT V2, DE LA COMPAÑÍA MICROSOFT

6. MANUAL DEL USUARIO

El sistema de visión diseñado en este proyecto está implementado para requerir la mínima interacción por parte del usuario y ejecutar la detección de errores de forma automática, ayudando así a la automatización de la tecnología de impresión 3D con modelado por deposición fundida.

En linia con lo comentado, se requiere para el funcionamiento del sistema que el usuario envie, mediante el software de la impresora (como puede ser el Repetier), una pieza a imprimir, transformándose la misma (por parte del software) en código G, una vez hecho tendrá que ser copiado el archivo con el código G en la carpeta donde se encuentra el ejecutable en formato .txt con el nombre “pieza.txt”, posteriormente, se deberá ejecutar el script ejecutable que contiene toda la secuencia de programas para la detección de errores.

Para empezar se deberá ejecutar el script ejecutable usando la siguiente linea de código por terminal “sudo ./TFG.sh”. Una vez ejecutado “TFG.sh” las cámaras tomarán las imágenes de la pieza, preguntándo posteriormente al usuario qué color se desea detectar, es decir, de qué el color es el material de impresión (Fig. 26), pudiendo seleccionar por teclado un color de entre una lista de posibles colores. El siguiente paso será indicar por parte del usuario el número de capa que está siendo analizado en las imágenes (Fig. 27), este paso puede ser evitado en un posible proyecto futuro que integre la detección de los errores en el software de la impresora, pudiéndose enviar el dato directamente desde la misma.

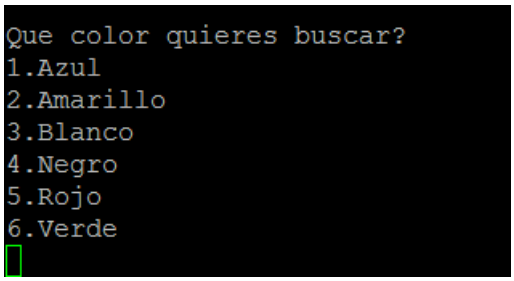


FIG. 26 – PETICIÓN AL USUARIO DEL COLOR DEL MATERIAL

Pasado esto, ya no se necesitaría interacción por parte del usuario hasta que el proceso finalice, una vez esto suceda se creará un fichero de imagen con extensión .png donde se encontrará el perfil resultante de coincidir el contorno teórico de la pieza con el contorno real de la misma, pudiéndose ver así los errores de impresión que existan (en el caso de que las hubiera).

Una vez finalizado el proceso, el usuario deberá considerar si los errores detectados se pueden despreciar, reparar o se debe parar la impresión y desechar la pieza, aunque esto también puede formar parte de otro proyecto encargado de la valoración automática de los errores detectados.

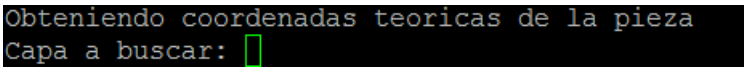


FIG. 27 – PETICIÓN DEL NÚMERO DE CAPA A ANALIZAR

Los posibles errores que puedan aparecer en la ejecución del programa pueden ser debidos a diferentes causas, como por ejemplo:

- Introducir un número que no esté en la lista de colores a elegir
- Introducir un número de capa a analizar menor que 0 o mayor que el número total de capas de la pieza.
- Incorrecta colocación de la cámara en el puerto de la Raspberry Pi
- Error de compilación debido a que se ha modificado algún archivo del programa de manera indebida.

Para evitar dichos errores se deberá indicar un valor que se encuentre dentro de los rangos tanto de colores a elegir como de capas de la pieza, comprobar bien la colocación de la cámara en su lugar correspondiente y no modificar ningún archivo salvo conocimiento en la materia y consultando en todo caso el manual del programador.

A continuación se muestran los errores comentados, respectivamente:

```
Capturando imagenes y obteniendo borde real
mmal: mmal_vc_component_create: failed to create component 'vc.ril.camera' (1:EN
OMEM)
mmal: mmal_component_create_core: could not create component 'vc.ril.camera' (1)
mmal: Failed to create camera component
mmal: main: Failed to create camera component
mmal: Camera is not detected. Please check carefully the camera module is instal
led correctly
```

FIG. 28 – ERROR DE MALA CONEXIÓN DE LA CÁMARA

```
Que color quieres buscar?
1.Azul
2.Amarillo
3.Blanco
4.Negro
5.Rojo
6.Verde
7
Traceback (most recent call last):
  File "bordes.py", line 75, in <module>
    mask = cv2.inRange(hsv, bajos, altos)
NameError: name 'bajos' is not defined
```

FIG. 29 – ERROR FUERA DE RANGO DE COLORES

```
Capa a buscar: -1
Proyectando coordenadas
Obteniendo bordes teoricos en la imagen
Detectando errores
```

FIG. 30 – ERROR FUERA DE RANGO DE CAPAS

```
Traceback (most recent call last):
  File "bordes.py", line 41, in <module>
    imagen = cv2.imread('/home/pi/imagen1.png')
NameError: name 'cv2' is not defined
```

FIG. 31 – ERROR MODIFICACIÓN INDEBIDA DE ARCHIVO

7. MANUAL DEL PROGRAMADOR

7.1. Programación de la Raspberry Pi (desde Windows)

Para empezar a trabajar con la Raspberry Pi primero se debe descargar sistema operativo desde la página del fabricante (https://www.raspberrypi.org/downloads/) escogiendo el S.O. NOOBS Offline, una vez descargado descomprimir el contenido (se trata de un archivo comprimido con formato zip). El siguiente paso será formatear la tarjeta SD con formato FAT32, para ello usaremos la aplicación Fat32Format que podemos encontrar en muchos sitios, por ejemplo la podemos descargar desde (http://fat32-format.en.softonic.com/), ya que puede que Windows no permita formatear en dicho formato para tarjetas SD de gran capacidad.

El proceso es muy sencillo, se trata de abrir el archivo guiformat.exe que nos habremos descargado de la página anteriormente mencionada, seleccionar la tarjeta SD a formatear, marcar la opción "Quick Format" y luego "Start", una vez finalizado el formateo marcar "Done" para cerrar la aplicación.

Después de formatear la tarjeta SD se deben copiar los archivos anteriormente descomprimidos dentro de la misma.

El siguiente paso será inicializar la placa, para ello se debe colocar la tarjeta SD en la ranura situada en la cara posterior. Si se tiene acceso a una pantalla con conexión HDMI la inicialización es un proceso sencillo y directo, en el caso de este proyecto se ha seguido otro camino debido a la falta de disposición de pantalla y conexión a internet en el mismo lugar. Por eso se ha utilizado una conexión SSH para poder acceder a la Raspberry Pi 2 desde un ordenador externo con el simple hecho de conectar ambos a un router común via Ethernet.

La conexión SSH se ha realizado con el programa PuTTy, que se puede adquirir, por ejemplo, desde (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html). Una vez instalado el programa (instalación estándar), se ejecutará y se incluirá la dirección IP de la Raspberry Pi (la placa debe estar conectada tanto a la corriente mediante el cable micro usb que incluye como al router por cable ethernet, como ya se ha dicho) en el campo "Host Name (or IP address)" tal que así (ver página siguiente):

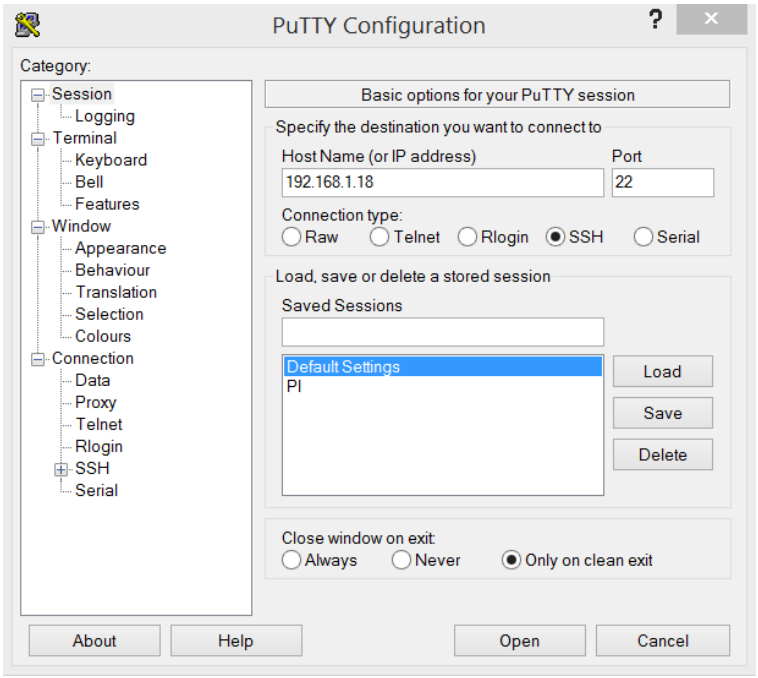


FIG. 32 – CONFIGURACIÓN PUTTY PARA CONEXIÓN SSH

Una vez introducida la dirección IP se seleccionará "Open", abriéndose así un terminal en el cual la Raspberry se iniciará automáticamente hasta que pida al usuario elegir qué sistema operativo desea instalar, se debe escoger Raspbian, ya que ha sido creado expresamente para trabajar con la placa, aunque se podría elegir sin ningún problema cualquier otra distribución de Linux. Una vez instalado el sistema operativo accederá a la pantalla de configuración de la Raspberry o "raspi-config", en la cual podrá cambiar, si se desea, el usuario y la contraseña (por defecto el usuario es "pi" y la contraseña "raspberry"), se deberá habilitar la cámara, lo cual hará que quede habilitada para siempre, sin necesidad de volverlo a hacer y, finalmente, se dividirá la memoria de la placa entre memoria RAM y memoria de la GPU (ya que en este modelo consta 1 Gigabyte de memoria la cual utilizan tanto CPU como GPU, asi que reservaremos una cantidad fija, 512 Kilobytes, para cada unidad de proceso).

Una vez terminada la configuración básica de la Raspberry Pi se hará una actualización del sistema, para ello se escribirá por terminal "sudo apt-get" y se pasará a instalar las librerías necesarias para el desarrollo del proyecto.

Se instalará la librería CMAKE escribiendo por terminal "sudo apt-get install cmake", luego la librería OpenCV siguiendo el tutorial que ofrecen desarrolladores de la misma (http://docs.opencv.org/doc/tutorials/introduction/linux_install/linux_install.html#linux-installation). Seguidamente se instalará la librería Wiring Pi siguiendo, como en el caso anterior, el tutorial de sus desarrolladores (http://wiringpi.com/download-and-install/) y la

última librería a instalar será la librería ncurses.h, escribiendo por terminal "sudo apt-get install libncurses5-dev".

Una vez instaladas las librerías se apagará la Raspberry Pi para instalar la cámara (escribiendo "sudo shutdown –h now").

Una vez apagada se instalará la cámara introduciendo el bus del módulo de cámara en el puerto destinado a ello.

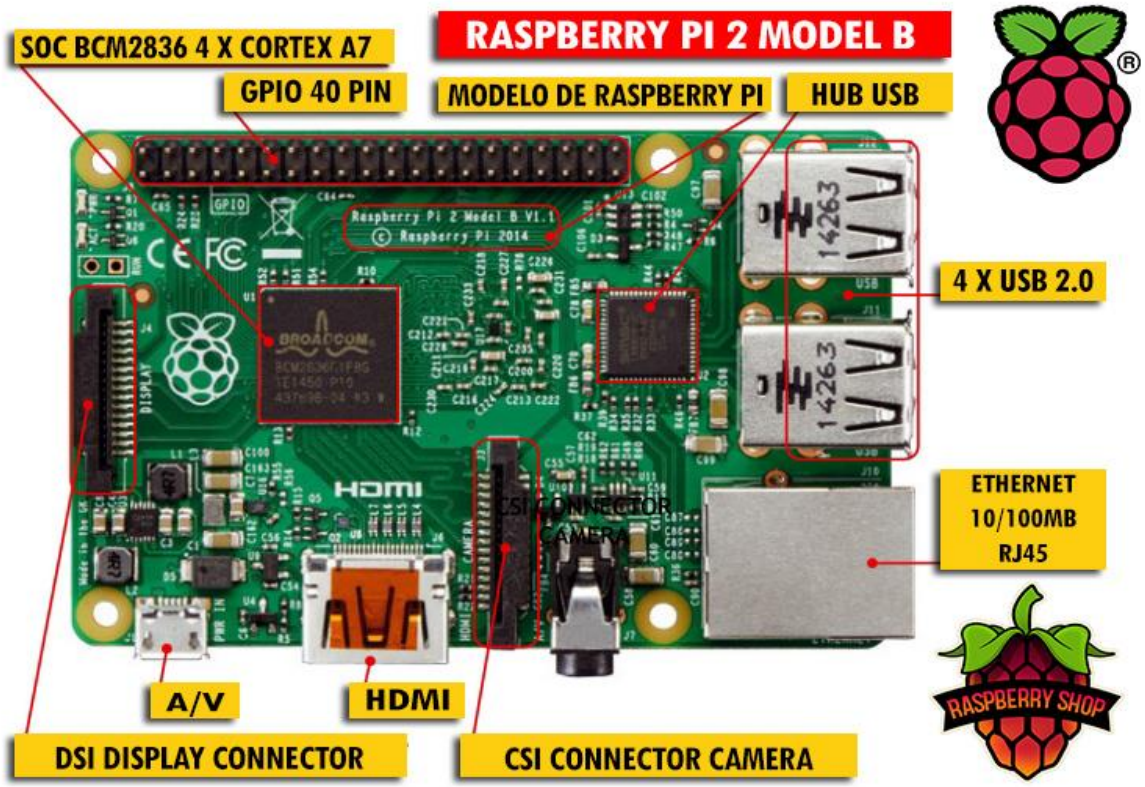


FIG. 33 – LOCALIZACIÓN DE LOS PUERTOS DE RASPBERRY PI 2

Una vez conectado el módulo de cámara se procederá a encender la placa de nuevo, se dejará inicializar hasta que pida el usuario y la contraseña, que serán introducidos (según si se han cambiado desde raspi-config o siguen por defecto). Una vez encendida se hará una actualización del firmware escribiendo desde terminal "sudo apt-get update", seguido de "sudo apt-get upgrade". El siguiente paso será descargar e instalar los drivers de la cámara (userland-master) desde github (https://github.com/raspberrypi/userland) y se descomprimirá el archivo descargado para posteriormente copiar el resultado de la descompresión en el directorio de la raspberry "/opt/vc". Para la transferencia de archivos desde el ordenador a la Raspberry Pi se puede usar una conexión SCP, por ejemplo con el programa WinSCP (aunque también se podría hacer, por ejemplo, con un dispositivo USB), se puede ver un tutorial de cómo usarlo en (https://siliconhosting.com/kb/questions/129/Usar+WinSCP+para+subir+archivos).

Una vez hecha la transferencia nos podemos situar desde terminal en la carpeta que se habrá creado "/opt/vc/userland-master " usando el comando "cd" para movernos entre directorios y se escribirá por terminal "sudo mkdir build", se entrará dentro de ese directorio creado con el comando ya citado y se escribirá "sudo cmake –DCMAKE_BUILD_TYPE=Release .." seguido de "sudo make" y finalmente "sudo make install", con ello habremos instalado los drivers de la cámara.

Para terminar, se puede hacer un reinicio del sistema para la completa instalación de los drivers escribiendo "sudo reboot".

7.2. Obtención de los Bordes Reales

Para la obtención de los bordes reales se ha utilizado un programa escrito en Python llamado "Bordes.py", se trata de una modificación de un programa original (ya mencionado en el apartado 5.7.). Para ahorrar espacio se ha obviado el proceso que se aplica a la imagen capturada por la cámara frontal, por ser idéntico al de la cámara superior, sí se muestran las configuraciones para poder tomar las dos imágenes con un multiplexor. El código es el siguiente:

```
#Importar las librerías necesarias
import cv2
import numpy as np
import RPi.GPIO as gp
import os

#Inicializar pines
gp.setwarnings(False)
gp.setmode(gp.BOARD)
gp.setup(7, gp.OUT)
gp.setup(11, gp.OUT)
gp.setup(12, gp.OUT)

#Obtener la imagen de la camara 1
gp.output(7, False)
gp.output(11, False)
gp.output(12, True)
cmd = "raspistill -w 2592 -h 1944 -o imagen1.png"
os.system(cmd)

#Obtener la imagen de la camara 2
gp.output(7, False)
gp.output(11, True)
gp.output(12, False)
cmd = "raspistill -w 2592 -h 1944 -o imagen2.png"
os.system(cmd)

#Leer la imagen 1
imagen = cv2.imread('/home/pi/imagen1.png')

#Desdistorsionamos la imagen
cameraMatrix = np.matrix([[2574.7565286761178, 0, 1345.6535861508519],
[0, 2573.3889599185100, 1008.1544386232972], [0, 0, 1]])
distCoeffs = np.array([0.12906482320138898, -0.41559007944622650,
0.0075376687939254505, 0.0097244756785408015, 0.30318378725808615])

imagen = cv2.undistort(imagen, cameraMatrix, distCoeffs)
```

```
#Convertir la imagen en HSV
hsv = cv2.cvtColor(imagen,cv2.COLOR_BGR2HSV)

#Pedir el color a buscar (color del material de impresión)
color = int(input('Que color quieres
buscar?\n1.Azul\n2.Amarillo\n3.Blanco\n4.Negro\n5.Rojo\n6.Verde\n'))

#Colores almacenados 1.Azul 2.Amarillo 3.Blanco 4.Negro 5.Rojo 6.Verde
if color == 1 :
  bajos = np.array([67,40,105], dtype=np.uint8)
  altos = np.array([129,255,182], dtype=np.uint8)

elif color == 2 :
  bajos = np.array([25,50,0], dtype=np.uint8)
  altos = np.array([65,255,255], dtype=np.uint8)

elif color == 3 :
  bajos = np.array([0,0,170], dtype=np.uint8)
  altos = np.array([255,255,255], dtype=np.uint8)

elif color == 4 :
  bajos = np.array([0,0,0], dtype=np.uint8)
  altos = np.array([255,255,0], dtype=np.uint8)

elif color == 5 :
  bajos = np.array([0,100,0], dtype=np.uint8)
  altos = np.array([20,255,255], dtype=np.uint8)

elif color == 6 :
  bajos = np.array([60,100,0], dtype=np.uint8)
  altos = np.array([110,255,255], dtype=np.uint8)

#Crear una mascara que detecte los colores
mask = cv2.inRange(hsv, bajos, altos)

#Filtrar el ruido con un filtro de imagen CLOSE seguido de un OPEN
kernel = np.ones((6,6),np.uint8)
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

#Difuminar la mascara para suavizar los contornos y aplicar filtro canny
blur = cv2.GaussianBlur(mask, (5, 5), 0)
edges = cv2.Canny(mask,1,2)

#Buscar y bordear los contornos y filtrar ruido según el área del
contorno encontrado
contours, hier =
cv2.findContours(edges,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

```
areas = [cv2.contourArea(c) for c in contours]
i = 0
for extension in areas:
    if extension > 600:
      actual = contours[i]
      approx =
cv2.approxPolyDP(actual,0.05*cv2.arcLength(actual,True),True)
      cv2.drawContours(imagen,[actual],0,(0,0,255),1)
    i = i+1

#Crea una imagen en negro donde se guardará el contorno encontrado
maskxy = np.zeros((1944,2592), np.uint8)
cv2.drawContours(maskxy,contours,-1,(255,255,255),1)
cv2.imwrite('/home/pi/mask1xy.png',maskxy)

#Guardamos la imagen desdistorsionada con el contorno encontrado
cv2.imwrite('/home/pi/planoxy.png',imagen)
```

Con este programa se obtienen los siguientes resultados:

FIG. 34 – IMAGEN DE LA CÁMARA SUPERIOR CON DISTORSIÓN DEBIDA A LA CURVATURA DE LA LENTE

FIG. 35 – BORDE ENCONTRADO EN LA IMAGEN DE LA CÁMARA SUPERIOR

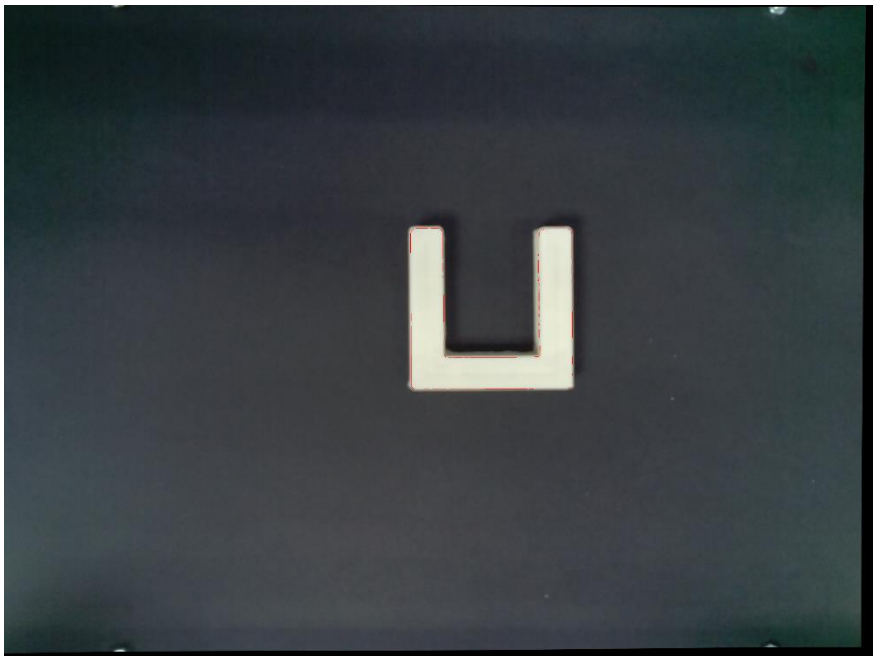


FIG. 36 – IMAGEN DE LA CÁMARA SUPERIOR DESDISTORSIONADA CON EL CONTORNO MARCADO

7.3. Obtención de los Bordes Teóricos

Los bordes teóricos de la pieza se obtendrán a partir del archivo de código G que procede del software de la impresora 3D, mediante un programa llamado "Write.py" creado en Python que filtrará las coordenadas que sigue la máquina y las almacena en un fichero llamado "coord.txt", el código es el siguiente:

```
#Importar las librerías necesarias
import numpy as np
import cv2

#Abrir el fichero con el código G y crear el fichero de destino de las
coordenadas
f = open("pieza.txt")
g = open("coord.txt", "w")

#Inicializar variables
letra = None
z = 0
e = 0
e_1 = 0
x_2 = 0
y_2 = 0
x_3 = 0
y_3 = 0
c = -1

#Pedir la capa a buscar (capa actual)
capa = int(input('Capa a buscar: '))

#Bucle de lectura del fichero y obtención de coordenadas
while letra == None :

    linea = f.readline()
    dato = list(linea)
    largo = len(dato)
    i = 0
    x = 0
    y = 0
    resta = 0

#Obtención de la coordenada "Z", si la capa cambia muestra en cuál se
encuentra buscando
    numeroz = [None]*10
    if len(dato) > 0:
      if dato[0] == "G":
        if dato[1] == "1":
```

```
            if dato[2] == " ":
              if dato[3] == "Z" :
                i = 4
                n = 0
                while dato[i] != " ":
                    numeroz[n] = dato[i]
                    n = n+1
                    i = i+1
                for a in range(0,9):
                    m = 9-a
                    if numeroz[m] == None :
                      numeroz.pop(m)
                z_2 = ''.join(numeroz)
                if z_2 != z :
                  c = c+1
                  if c > 0:
                    print(c)
                z = z_2

#Obtención de la coordenada "X"
      numerox = [None]*7
      if dato[0] == "G":
        if dato[1] == "1":
          if dato[2] == " ":
            if dato[3] == "X" :
              i = 4
              n = 0
              while dato[i] != " ":
                  numerox[n] = dato[i]
                  n = n+1
                  i = i+1
              for a in range(0,6):
                  m = 6-a
                  if numerox[m] == None :
                    numerox.pop(m)
              x = ''.join(numerox)

#Obtención de la coordenada "Y"
      numeroy = [None]*7
      if dato[0] == "G":
        if dato[1] == "1":
          if dato[2] == " ":
            i = i+1
            if dato[i] == "Y" :
              i = i+1
              n = 0
              while dato[i] != " " :
                  numeroy[n] = dato[i]
                  n = n+1
```

```
                i = i+1
                if dato[i] == "\n" :
                  dato[i] = " "
            for a in range(0,6):
                m = 6-a
                if numeroy[m] == None :
                  numeroy.pop(m)
            y = ''.join(numeroy)

#Obtención de la longitud de material imprimido entre dos coordenadas
consecutivas
      numeroe = [None]*8
      if dato[0] == "G":
        if dato[1] == "1":
          if dato[2] == " ":
            i = i+1
            if dato[i] == "E" :
              i = i+1
              n = 0
              while dato[i] != " " :
                  numeroe[n] = dato[i]
                  n = n+1
                  i = i+1
                  if dato[i] == "\n" :
                    dato[i] = " "
              for a in range(0,7):
                  m = 7-a
                  if numeroe[m] == None :
                    numeroe.pop(m)
              e = float(''.join(numeroe))
              resta = e-e_1
              e_1 = e

#Escritura de la coordenada en coord.txt si ésta pertenece a la capa
actual de impresión
      if capa == c :
        if x != 0 :
          if c >= 1:
            if e >= 0:
              x_1 = float(y)-50
              x_2 = x_1
              y_1 = float(x)-46.1
              y_2 = y_1
              z_1 = float(z_2)*(-1)+55
              x = str(x_1)
              y = str(y_1)
              z = str(z_1)
              g.write(x)
              g.write(" ")
```

```
                g.write(y)
                g.write(" ")
                g.write(z)
                g.write(" ")

#Algoritmo que crea puntos intermedios entre dos coordenadas
consecutivas muy distantes entre si
                if resta > 0.07187 :
                  if abs(x_2-x_3) > 3.1 :
                    if abs(y_2-y_3) < 3.1 :
                      puntosx = int((x_2-x_3)/3.1)
                      puntosy = int((y_2-y_3)/puntosx)
                      for i in range(1, puntosx) :
                          x = x_2-i*3.1
                          y = y_2+puntosy*i
                          x = str(x)
                          y = str(y)
                          g.write(x)
                          g.write(" ")
                          g.write(y)
                          g.write(" ")
                          g.write(z)
                          g.write(" ")
                  if abs(y_2-y_3) > 3.1 :
                    if abs(x_2-x_3) < 3.1 :
                      puntosy = int((y_2-y_3)/3.1)
                      puntosx = int((x_2-x_3)/puntosy)
                      for i in range(1, puntosy) :
                          x = x_2+puntosx*i
                          y = y_2-i*3.1
                          x = str(x)
                          y = str(y)
                          g.write(x)
                          g.write(" ")
                          g.write(y)
                          g.write(" ")
                          g.write(z)
                          g.write(" ")
                  if abs(x_2-x_3) > 3.1 :
                    if abs(y_2-y_3) > 3.1 :
                      puntosx = int((x_2-x_3)/3.1)
                      puntosy = int((y_2-y_3)/3.1)
                      med = (puntosx+puntosy)/2
                      for i in range(1, med) :
                          x = x_2-i*3.1
                          y = y_2-i*3.1
                          x = str(x)
                          y = str(y)
                          g.write(x)
```

```
                g.write(" ")
                g.write(y)
                g.write(" ")
                g.write(z)
                g.write(" ")
            x_3 = x_2
            y_3 = y_2


#Finalizar el bucle cuando se ha llegado a la capa buscada
      elif capa < c :
        letra = "a"

#Indicar siguiente linea de lectura
      f.tell()

#Finalizar el bucle si hemos llegado al final del fichero de lectura
    elif len(dato) <= 0:
      letra = "a"

#Cerrar los ficheros de lectura y de escritura
f.close()
g.close()
```

Finalmente se obtiene un fichero con las coordenadas en forma “X Y Z ” expresadas en el sistema de coordenadas del mundo de la cámara, como se puede ver a continuación:

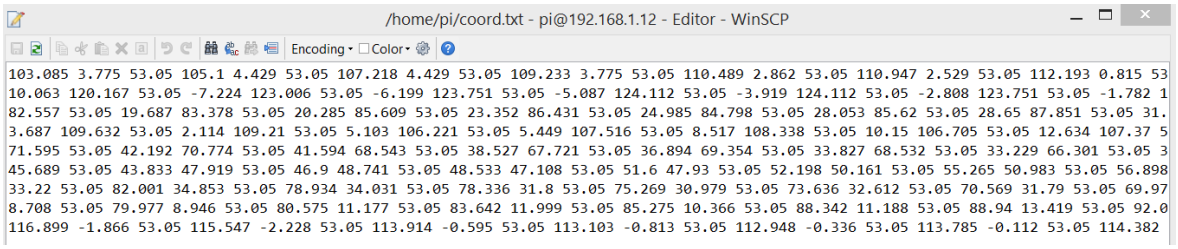


FIG. 37 – COORDENADAS QUE SIGUE LA IMPRESORA 3D EN LA CAPA ACTUAL

Éstas coordenadas son extraídas por un ejecutable llamado “proyeccion” que es obtenido a partir del ejemplo de calibración de cámara de la librería OpenCV, hecho en C++, al cuál se le realizan unas modificaciones, dicho ejecutable crea un fichero llamado “pixels.txt” donde guarda los píxeles correspondientes a las coordenadas de “coord.txt” proyectadas en la imagen tomada por la cámara superior, usando las matrices obtenidas en la calibración de la misma. La modificación realizada al ejemplo de calibración es la siguiente:

```
//Crear las variables que servirán de fichero de lectura y escritura y
crear dichos ficheros
FILE *read, *write;
```

```
        read = fopen("/home/pi/coord.txt", "r");
        write = fopen("/home/pi/pixels.txt", "w");

//Crear un vector para guardar las coordenadas extraídas de coord.txty
bucle de lectura de las mismas
        Point3f m;

        while (!feof(read)){

        fscanf(read, "%f", &m.x);
        fscanf(read, "%f", &m.y);
        fscanf(read, "%f", &m.z);

        int i = 0;
        for (i = 0; i < 44; i++){
            objectPoints[0][i] = m;
        }

//Proyectar las coordenadas usando las matrices de calibración de la
cámara y obteniendo los píxeles correspondientes a dichas coordenadas
        projectPoints(objectPoints[0], rvec, tvec, cameraMatrix,
distCoeffs, projectedPoints);

//Escribir los píxeles en el archivo “pixels.txt”
        fprintf(write, "%f ", projectedPoints[0].x);
        fprintf(write, "%f ", projectedPoints[0].y);

        }

//Cerrar los archivos de lectura y escritura
        fclose(read);
        fclose(write);
```

Con este programa obtenemos el archivo “pixels.txt con los píxeles en formato “X Y ” expresados en el sistema de coordenadas de la imagen, como se puede observar a continuación:

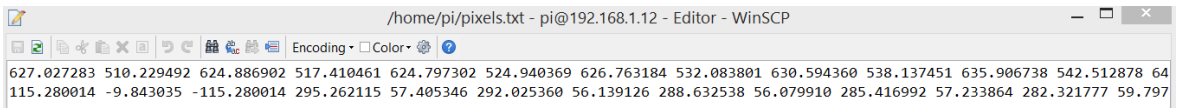


FIG. 38 – PÍXELES CORRESPONDIENTES AL BORDE TEÓRICO DE LA PIEZA

Finalmente, los píxeles obtenidos son leídos por un programa propio llamado “read.py” hecho en Python que pinta el borde teórico en la imagen desdistorsionada de la cámara y crea una imagen en negro y pinta los bordes en ella también. El código es el siguiente:

```
#Importar las librerías necesarias
import cv2
import numpy as np

#Crear la imagen en negro y abrir la imagen desdistorsionada y el
archivo que contiene los píxeles
img = cv2.imread("planoxy.png")
negro = np.zeros((1944,2592), np.uint8)
f = open("pixels.txt")

#Bucle de lectura de los píxeles
linea = f.readline()
dato = list(linea)
length = len(dato)

i = 0

while i < length :
#Lectura de la coordenada “X”
    numerox = [None]*11
    n = 0
    while dato[i] != " " :
        numerox[n] = dato[i]
        n = n+1
        i = i+1

    for a in range(0,10):
        m = 10-a
        if numerox[m] == None :
          numerox.pop(m)

    if numerox[0] != None:
      x = ''.join(numerox)
      x = int(float(x))

#Lectura de la coordenada “Y”
    numeroy = [None]*11
    i = i+1
    n = 0

    while dato[i] != " ":
        numeroy[n] = dato[i]
        n = n+1
        i = i+1

    for a in range(0,10):
        m = 10-a
        if numeroy[m] == None :
          numeroy.pop(m)
```

```
    if numeroy[0] != None:
      y = ''.join(numeroy)
      y = int(float(y))

    i = i+1

#Pintar los puntos en la imagen en negro creada anteriormente
    x_3 = int(x)+45
    y_3 = int(y)-55
    cv2.circle(negro, (x_3, y_3), 0, (255, 255, 255), -1)

#Detectar los píxeles blancos en la imagen
pixelpoints = np.nonzero(negro)
length = len(pixelpoints[0])

#Algoritmo de unión de lineas a partir de la nube de píxeles blancos
coord = [None]*length
for i in range(0, length) :
    coord[i] = (pixelpoints[1][i], pixelpoints[0][i])
for i in range(0, length) :
    for j in range(0, length-1) :
        if j != i :
          if abs(coord[i][0]-coord[j][0]) < 35 :
            if abs(coord[i][1]-coord[j][1]) < 35 :
              cv2.line(negro, (coord[i][0], coord[i][1]), (coord[j][0],
coord[j][1]), (255,255,255), 1)

#Detección de los bordes a partir de las lineas pintadas anteriormente
negro1 = negro
contours, hier =
cv2.findContours(negro1,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(negro,contours,-1,(255,255,255),1)
cv2.drawContours(img,contours,-1,(0,0,255),1)

#Cerrar el fichero de lectura
f.close()

#Guardar las imágenes con los bordes pintados
cv2.imwrite("teobordes.png", img)
cv2.imwrite("mask2xy.png", negro)
```

Finalmente se obtiene una imagen con los bordes teóricos sobre la imagen desdistorsionada de la pieza real y otra similar con los bordes teóricos en blanco sobre fondo negro, como puede verse a continuación (cabe destacar que la máquina no recorre cada punto de la pieza en cada capa):

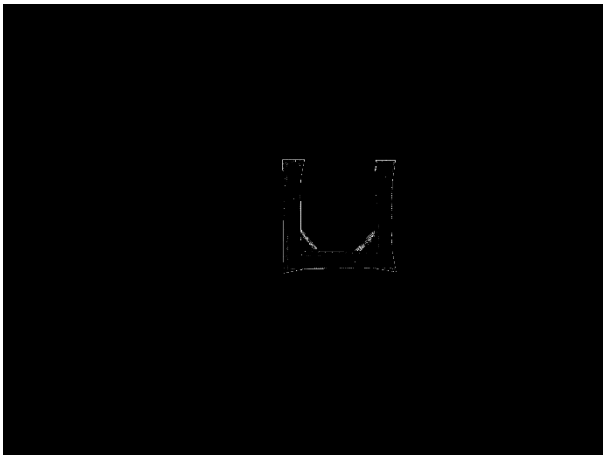


FIG. 39 – BORDES TEÓRICOS QUE SIGUE LA IMPRESORA 3D EN LA CAPA ACTUAL

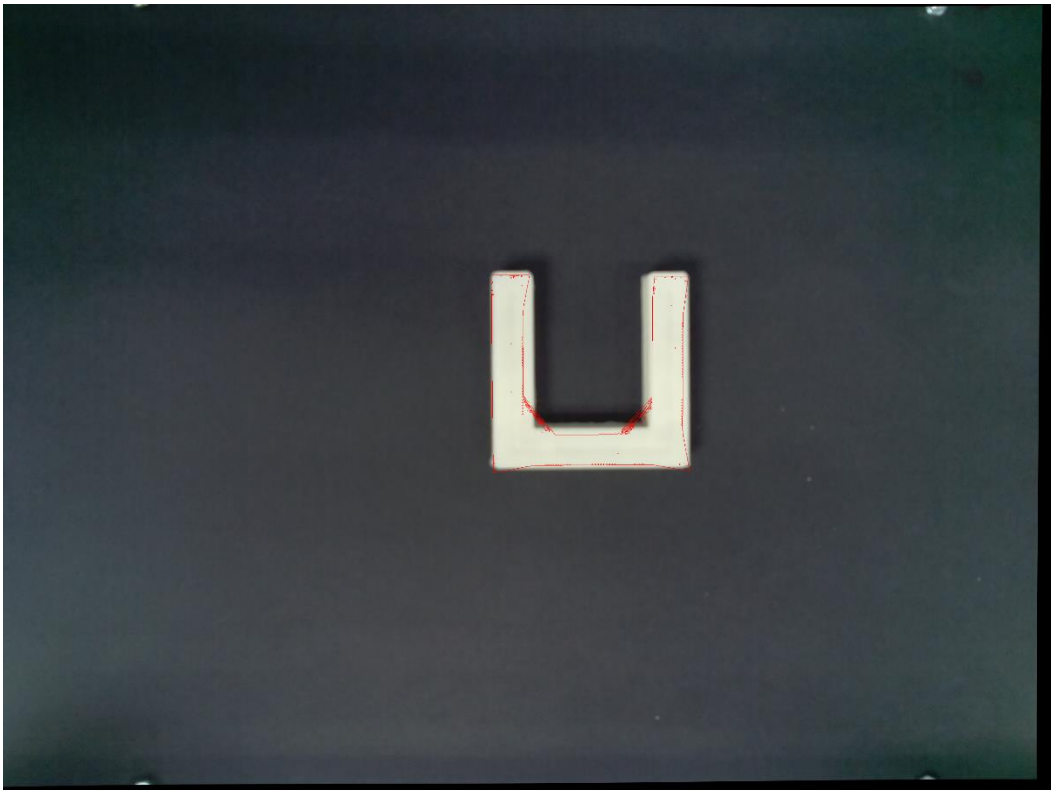


FIG. 40 – BORDES TEÓRICOS SOBRE LA IMAGEN DESDISTORSIONADA DE LA PIEZA

7.4. Comparación de Bordes. Detección de Errores

El programa usado para la detección de errores se trata de un programa de elaboración propia llamado "compbordes.py" hecho en Python que analiza las máscaras de bordes creadas tanto para el borde teórico como para el borde real píxel a píxel, dibujando sobre la imagen original desdistorsionada las coincidencias que encuentra y ofreciendo una medida porcentual de dichas coincidencias basada en el porcentaje de puntos del borde real que coinciden con el borde teórico. El código del programa es el siguiente:

```
#Importar las librerías necesarias
import cv2
import numpy as np

#Abrir las imágenes que se van a usar, máscara con el borde real, máscara
con el borde teórico e imagen original desdistorsionada, respectivamente
img = cv2.imread("mask1xy.png")
image = cv2.imread("mask2xy.png")
final = cv2.imread("planoxy1.png")
med = np.zeros((1944,2592), np.uint8)

#Binarizar las máscaras
imag = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
imagen = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

#Consultar la cantidad de puntos pertenecientes a cada borde
pixelpoints = np.nonzero(imag)
length = len(pixelpoints[0])

pixelpoint = np.nonzero(imagen)
large = len(pixelpoint[0])

#Bucle de comparación de bordes y escritura de coincidencias
c = 0

for i in range(0, length-1) :
    for j in range(0, large-1) :
        if abs(pixelpoints[0][i] - pixelpoint[0][j]) < 22 :
          if abs(pixelpoints[1][i] - pixelpoint[1][j]) < 7 :
            cv2.circle(med, (pixelpoints[1][i], pixelpoints[0][i]), 0,
(255, 255, 255), -1)
            c = c+1

pixelmed = np.nonzero(med)
```

```
long = len(pixelmed[0])
imagen, contours, hier =
cv2.findContours(med,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(final,contours,-1,(0,0,255),2)

#Cálculo porcentual de coincidencias, muestra del mismo y cierre de la
imagen de coincidencias
valor = (100*long)/length
print valor,"% de coincidencias entre bordes"

cv2.imwrite("coincidencias.png", final)
```

El resultado final con el ejemplo que se está tratando se muestra a continuación (las coincidencias se han redondeado considerando un 5% de error de posicionamiento entre ambos bordes):

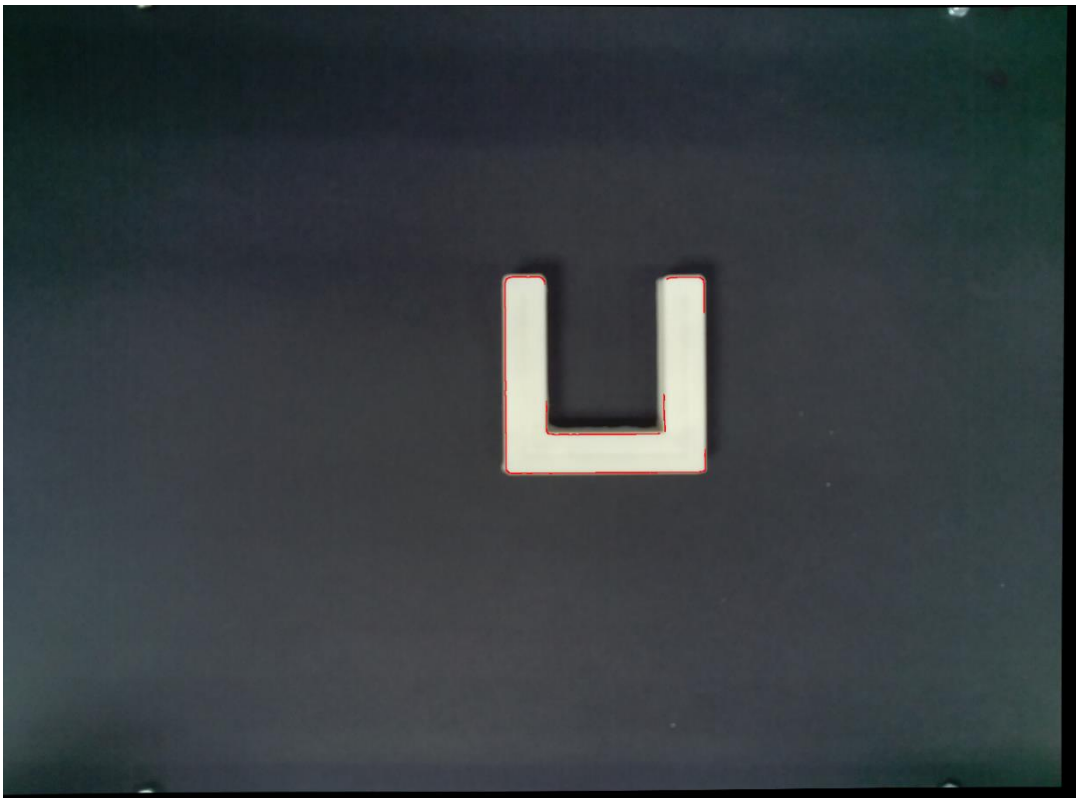


FIG. 41 – COINCIDENCIAS ENCONTRADAS ENTRE LOS BORDES TEÓRICO Y REAL DEL EJEMPLO

```
pi@raspberrypi ~ $ python compbordes.py
65 % de coincidencias entre bordes
```

FIG. 42 – VALOR DE COINCIDENCIA DE BORDES DEL EJEMPLO

7.5. Reconstrucción 3D

El programa que realiza la reconstrucción 3D de los puntos de cada capa es un programa de elaboración propia llamado "pixtocoord.py" hecho en Python. Éste programa extrae los puntos pertenecientes a la capa de la máscara que contiene el borde real de la pieza y utiliza la ecuación ya comentada en el apartado 5.10. para convertir dichos píxeles de imagen en coordenadas del tipo "X Y Z" expresadas en el sistema de coordenadas de la impresora 3D, que posteriormente son almacenadas en un fichero llamado "puntos.py". El código del programa es el que se puede ver a continuación:

```
#Importar librerías necesarias
import cv2
import numpy as np

#Abrir la máscara con el borde real y los ficheros donde escribe los puntos
y de dónde extrae la coordenada "Z" del plano
img = cv2.imread("mask1xy.png")
f = open("puntos.txt", 'w')
g = open("coord.txt", 'r')

#Inicializar variables y algoritmo de lectura de la coordenada "Z"
c = 0
j = 0
n = 0

linea = g.readline()
dato = list(linea)

while c < 2 :
    if dato[j] == " " :
      c = c+1
    j = j+1

numeroz = [None]*7

while dato[j] != " ":
    numeroz[n] = dato[j]
    n = n+1
    j = j+1
for a in range(0,6):
    m = 6-a
    if numeroz[m] == None :
      numeroz.pop(m)

z_2 = float(''.join(numeroz))
z = 55-z_2
```

```
#Obtención de los puntos pertenecientes al borde real de la pieza e
introducción de las matrices de parámetros extrínsecos e intrínsecos
pixelpoints = np.nonzero(img)
length = len(pixelpoints[0])

intrinsec = np.matrix([[2574.7565768766349, 0, 1345.6535252315264], [0,
2573.3890315858821, 1008.1543963732877], [0, 0, 1]])

rotation2 = cv2.Rodrigues(np.array([[0.043823279307342464], [-
0.086726318561831814], [1.5636925199758283]]))

rotation = rotation2[0]

extrinsec = np.matrix([[rotation[0, 0], rotation[0, 1], rotation[0, 2],
61.431050845413381], [rotation[1, 0], rotation[1, 1], rotation[1, 2], -
50.237816224637925], [rotation[2, 0], rotation[2, 1], rotation[2, 2],
212.26062789775935], [0, 0, 0, 1]])

#Bucle de cálculo de pixel a coordenadas
for i in range(0, length-1) :

    matriz = np.matrix([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, z], [0, 0,
0, 1]])

    pixel = np.matrix([[pixelpoints[1][i]], [pixelpoints[0][i]], [1]])

    resultante = extrinsec*matriz

    resultante2 = np.matrix([[resultante[0, 0], resultante[0, 1],
resultante[0, 3]], [resultante[1, 0], resultante[1, 1], resultante[1,
3]], [resultante[2, 0], resultante[2, 1], resultante[2, 3]]])

    resultante3 = intrinsec*resultante2

    retval, invertida = cv2.invert(resultante3)

    if retval == 1 :
      points = invertida*pixel

      points = points/points[2]

      x = float(points[1])+46.1
      y = float(points[0])+50
      x = str(x)
      y = str(y)
      z = str(z)
```

```
#Escritura de las coordenadas en el fichero “puntos.txt”
      f.write(x)
      f.write(" ")
      f.write(y)
      f.write(" ")
      f.write(z)
      f.write("\n")

    z = float(z)

#Cierre de los ficheros utilizados
f.close()
g.close()
```

Con este código se obtiene un fichero como el que se muestra a continuación (coordenadas expresadas en milímetros):

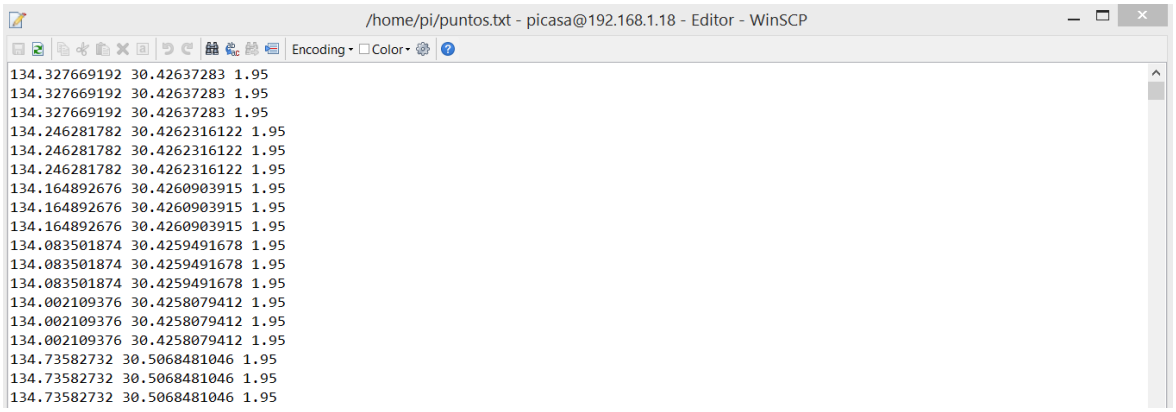


FIG. 43 – FICHERO DE COORDENADAS 3D RECONSTRUIDAS A PARTIR DE UNA IMAGEN

7.6. Ejecución del Sistema

Para ejecutar todos los programas explicados en los apartados anteriores se ha creado un script ejecutable llamado "TFG.sh", el cuál hará la llamada a cada programa e indicará al usuario qué programa se encuentra en ejecución. Para iniciar el script se escribirá por consola (como ya se ha dicho en el manual de usuario) "sudo ./TFG.sh". cada linea va comentada indicando el programa a llamar. El código usado es el siguiente:

```
#!/bin/bash

echo "Capturando imagenes y obteniendo borde real"

sudo python bordes.py

echo "Obteniendo coordenadas teoricas de la pieza"

python write.py

echo "Proyectando coordenadas"

cd calibracion

gcc proyeccion.cpp -o proyeccion -
L/home/pi/opencv/opencv/release/lib -lopencv_calib3d -lopencv_core
-lopencv_imgproc -lopencv_imgcodecs -lopencv_videoio -
lopencv_highgui -lopencv_features2d -L/lib/arm-linux-gnueabihf -lm
-lstdc++

sudo ./proyeccion

echo "Obteniendo bordes teoricos en la imagen"

cd ..

python read.py

echo "Detectando errores"

python compbordes.py

echo "Reconstruccion 3D de la capa"

python pixtocoord.py
```

8. BIBLIOGRAFÍA

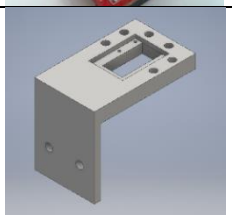
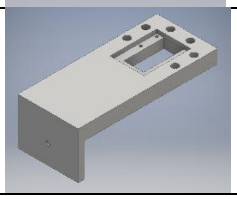
- Impresión 3D, Wikipedia (http://es.wikipedia.org/wiki/Impresi%C3%B3n_3D)
- Hilda Gómez, Dealerworld, 2014 (http://www.dealerworld.es/tendencias/este-ano-se-venderan-44000-impresoras-3d-domesticas-y-un-millon-en-2018)
- Fig. 1 (http://www.ni.com/white-paper/14103/es/)
- Defectos impresión 3D (http://www.dima3d.com/defectos-en-piezas-fabricadas-por-impresion-3d-fff-causas-y-soluciones/)
- Fig. 2 (http://www.dima3d.com/defectos-en-piezas-fabricadas-por-impresion-3d-fff-causas-y-soluciones/)
- Fig. 3 (https://www.flickr.com/photos/68272764@N05/15988584302/in/photostream/)
- Fig. 4 (http://spainlabs.com/foro/viewtopic.php?f=31&t=340)
- Fig. 5 (http://enthings.com/tutorial/como-mejorar-la-sujecion-de-las-piezas-la-cama/)
- Fig. 6 (http://www.dima3d.com/defectos-en-piezas-fabricadas-por-impresion-3d-fff-causas-y-soluciones-iii/)
- Fig. 7 (http://electronilab.co/tienda/camara-para-raspberry-pi-5mp/)
- Pilar Ortega, Metropoli, 2015 (http://www.metropoli.com/salir/2015/03/12/55006d9de2704e283c8b459f.html)
- Javier Martín, El País, 2013 (http://tecnologia.elpais.com/tecnologia/2013/01/09/actualidad/1357703388_558416.html)
- Carol Lewis, The Times, 2014 (http://www.thetimes.co.uk/tto/life/property/interiors/article4217636.ece)
- Fig. 25 (http://www.microsoftstore.com/store/mssg/en_SG/pdp/Kinect-for-Windows-v2-Sensor/productID.299057000)
- Fig. 33 (http://www.raspberryshop.es/)
- Documentación de las funciones de la librería OpenCV (http://docs.opencv.org/)
- Dudas y consultas varias sobre OpenCV y programación en Python (http://stackoverflow.com/)

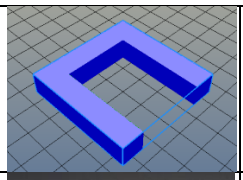
DISEÑO DE UN SISTEMA DE VISIÓN DE BAJO COSTE PARA RECONSTRUCCIÓN 3D DE OBJETOS

Presupuesto

Jaime Pérez Álvarez

El presupuesto del proyecto es el que viene detallado a continuación:

COMPONENTE	IMAGEN	PRECIO UNITARIO (€/UNIDAD)	CANTIDAD (UNIDADES)	PRECIO (IVA INCLUIDO, €)
Tarjeta SD (8Gb)		5	1	5
Raspberry Pi 2 B		32	1	32
Transformador 230V – 5V		7	1	7
Módulo de Cámara para Raspberry Pi 5Mpi		22	2	44
Multiplexor		110	1	110
Soporte impreso 3D plano XY		28.42	1	28.42
Soporte impreso 3D plano XZ		22.69	1	22.69

Pieza de ejemplo impresa 3D		3.46	1	3.46
Cable conexión cámara 2 metros		2	2	4
Conector HDMI-VDI		5.82	1	5.82
Foco iluminación 230V 50W		10.48	2	20.96
Hora de trabajo Ingeniero Industrial		25	300	7500
Presupuesto Total de Ejecución Material				**7783.35**
Gastos Generales			15%	1167.5
Beneficio Industrial			6%	467
Presupuesto Total de Inversión				**9417.85**
IVA			21%	1977.75
Presupuesto Base de Licitación				**11395.6**

Asciende el presupuesto total de ejecución material a la siguiente cantidad:

SIETE MIL SETECIENTOS OCHENTA Y TRES EUROS CON TREINTA Y CINCO CÉNTIMOS

Asciende el presupuesto total de inversión a la siguiente cantidad:

NUEVE MIL CUATROCIENTOS DIECISIETE EUROS CON OCHENTA Y CINCO CÉNTIMOS

Asciende el presupuesto base de licitación a la siguiente cantidad:

ONCE MIL TRESCIENTOS NOVENTA Y CINCO EUROS CON SESENTA CÉNTIMOS

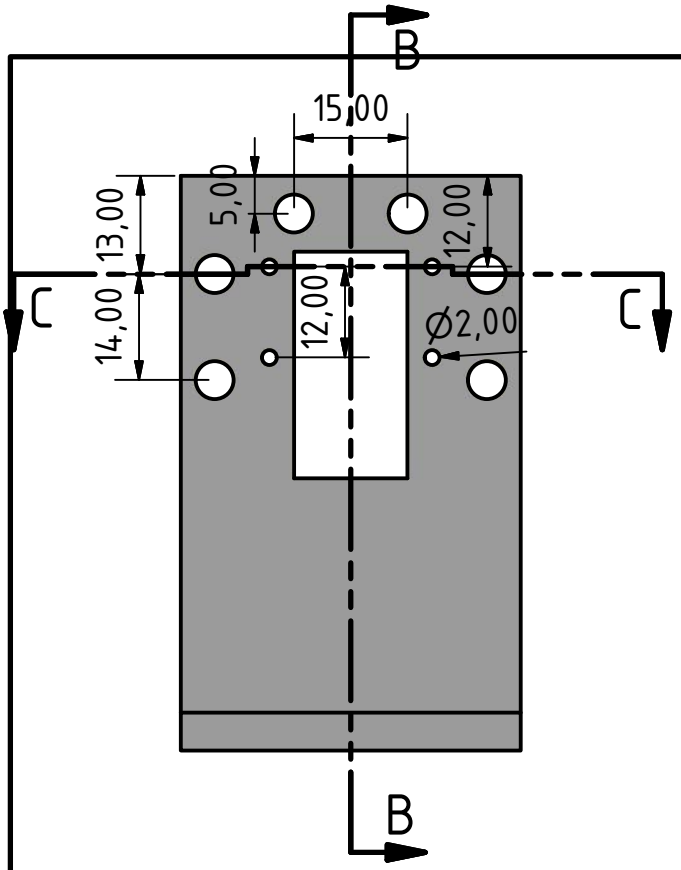
La suma total del proyecto es, a primer vista, bastante cuantiosa, pero se debe tener en cuenta lo comentado en el apartado de Justificación del Proyecto, y es que en una producción industrial, debido a ser un proceso automatizado, puede estar la impresora produciendo autónomamente todo el día. Haciendo cálculos tenemos que el ahorro por pieza al detectarse un error en una pieza es de 20€ aproximadamente, evitando 2 horas de impresión con error y según un estudio tecnológico hecho por la "Michigan Technical University", existe una tasa de fallos del 20% en las impresoras 3D actuales (http://www.sciencedirect.com/science/article/pii/S0957415813001153). Esto nos lleva a obtener un ahorro de 48€ diarios, siendo el periodo de retorno de la inversión del proyecto de 237.41 días.

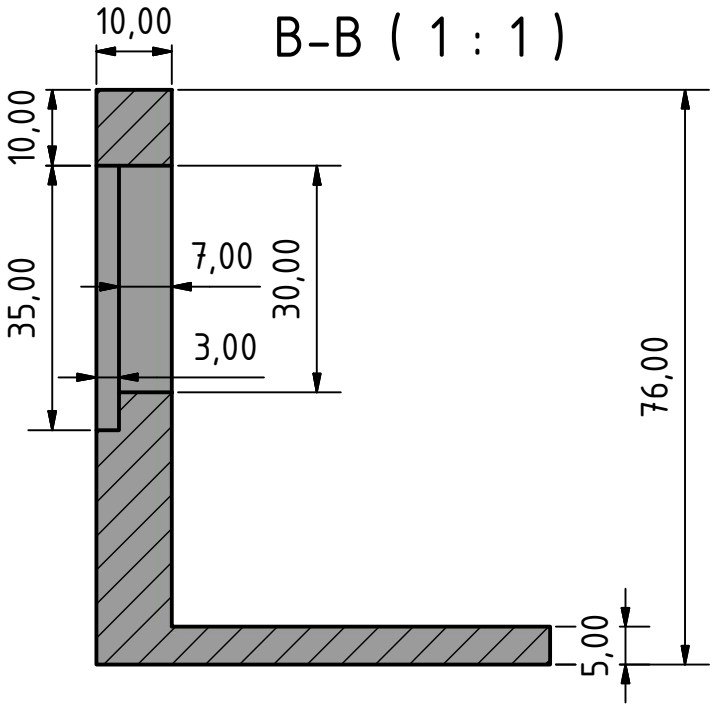
Ahora bien, la vida útil de una impresora 3D "no tiene fin", en el sentido que todas sus piezas pueden ser recambiables y la durabilidad de dichas piezas depende de múltiples factores, el más importante de ellos, el mantenimiento. Dicho esto, queda concluir que puede ser viable la implantación del sistema visión para la detección de errores de impresión 3D presentado en este proyecto para una empresa que se dedique a la impresión 3D de manera industrial, siendo éste más viable cuanto mayor sea la vida útil de la impresora.

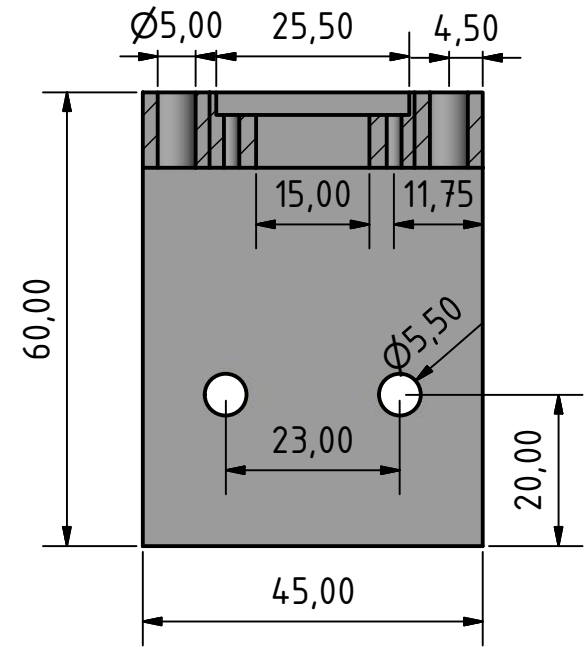
DISEÑO DE UN SISTEMA DE VISIÓN DE BAJO COSTE PARA RECONSTRUCCIÓN 3D DE OBJETOS

Planos

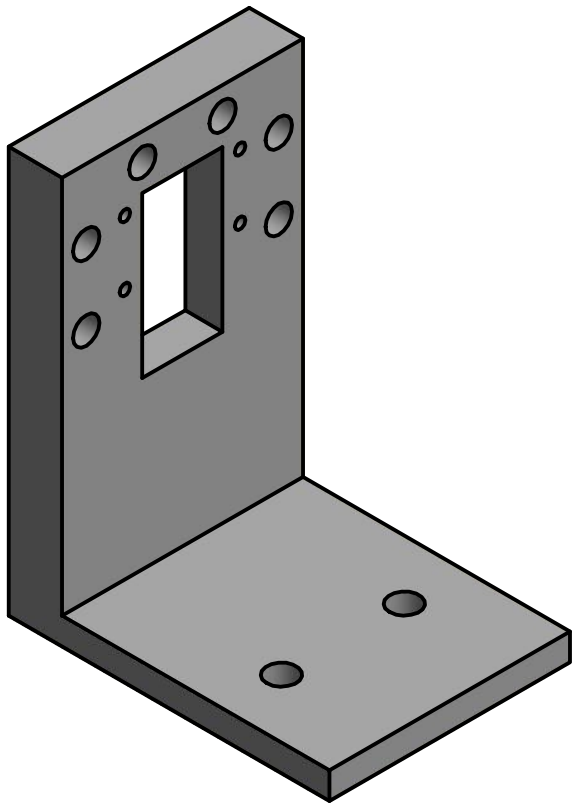
Jaime Pérez Álvarez







C-C (1 : 1)



TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA TÈCNICA SUPERIOR ENGINYERS INDUSTRIALS VALÈNCIA

Jaime Pérez Álvarez
Autor Proyecto

Proyecto:

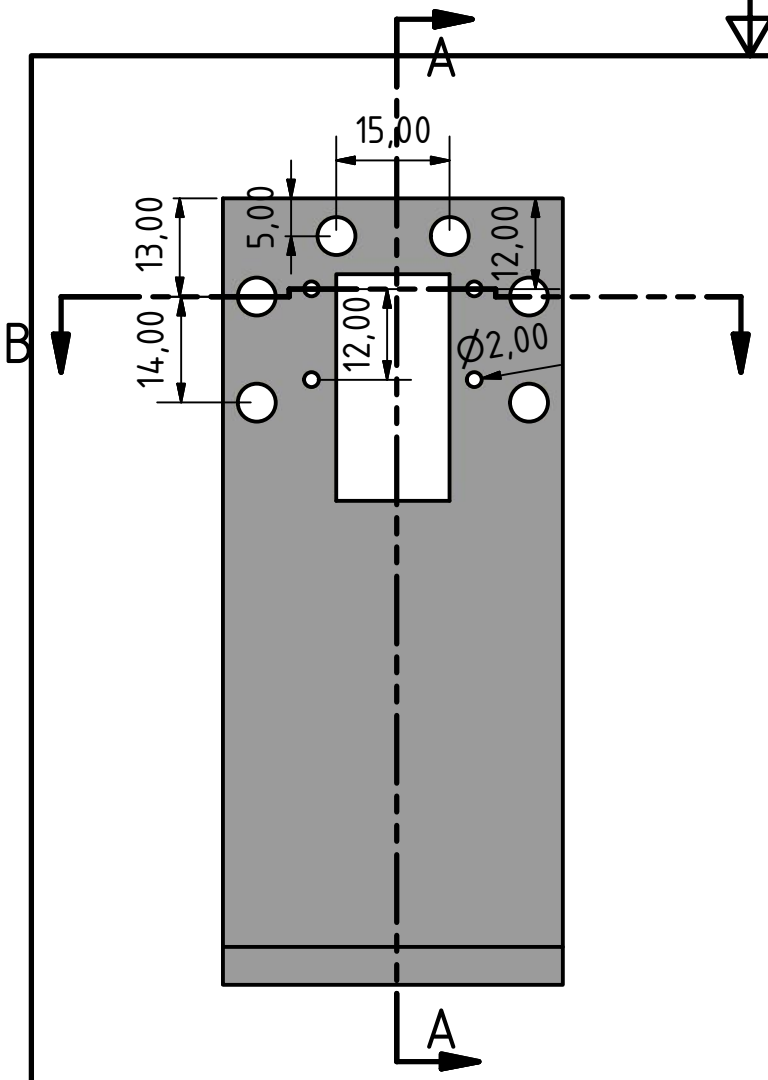
DESARROLLO DE UN SISTEMA DE VISIÓN DE BAJO COSTE PARA RECONSTRUCCIÓN 3D DE OBJETOS

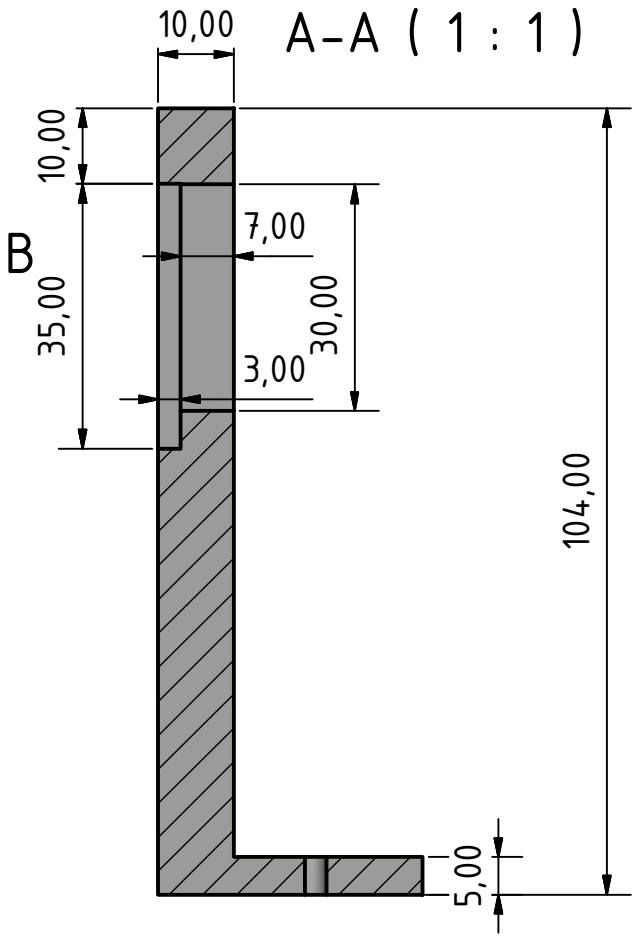
Fecha: Julio 2015

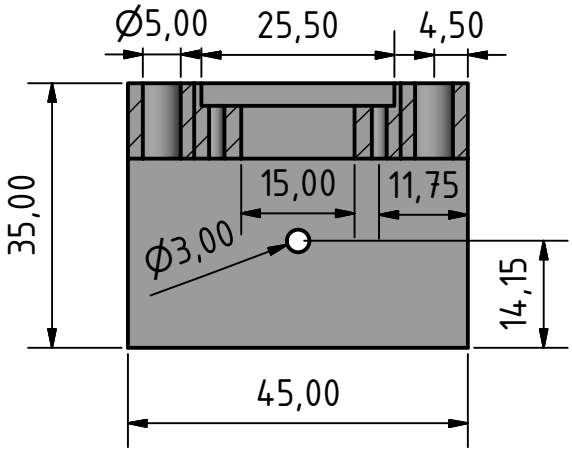
Escala: 1/1

Plano: Soporte. Cámara Superior.

Nº Plano: 1

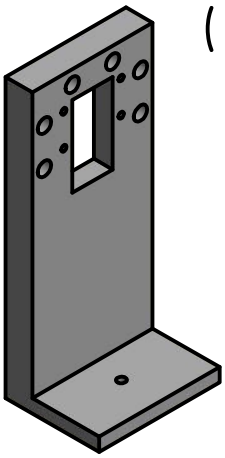






B-B (1 : 1)

Vista Isométrica
(1 : 2)



TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA TÈCNICA SUPERIOR ENGINYERS INDUSTRIALS VALÈNCIA

Jaime Pérez Álvarez
Autor Proyecto

Proyecto:

DESARROLLO DE UN SISTEMA DE VISIÓN DE BAJO COSTE PARA RECONSTRUCCIÓN 3D DE OBJETOS

Fecha: Julio 2015

Escala: 1/1

Plano: Soporte. Cámara Frontal.

Nº Plano: 2