UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería de Sistemas y Automática

Calibración y puesta en marcha de un sistema de decapado continuo con láser

Trabajo Fin de Máster

Máster Universitario en Automática e Informática Industrial

AUTOR/A: Gamir Artesero, Javier

Tutor/a: Ricolfe Viala, Carlos

CURSO ACADÉMICO: 2023/2024

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingenieros Industriales

Calibración y puesta en marcha de un sistema de decapado continuo con láser

Trabajo Fin de Máster

Máster en Automática e Informática Industrial

Autor

Javier Gamir Artesero

Tutor

Carlos Ricolfe Viala

CURSO ACADÉMICO: 2023/2024

Agradecimientos

A mis padres y mi hermano, por su constante apoyo en todo lo que hago, por escuchar mis quejas, dudas y quebraderos de cabeza durante estos meses, aun cuando no siempre entendieran de qué hablaba.

A mi tutor, por brindarme la oportunidad de participar en un proyecto real, que ha sido increíblemente gratificante y educativo, así como por su apoyo y, sobre todo, su infinita paciencia.

A mi compañero de laboratorio, Mario, por su consideración, por toda la ayuda que fue clave en momentos cruciales y por hacer más llevaderas las horas en el laboratorio.

Y a mis compañeros, con quienes no solo aprendí, sino que también disfruté del máster, especialmente a Abel Górriz, Marc Fontalba e Hipólit Raigal, de quienes me llevo una gran amistad y el valioso aprendizaje de lo que significa trabajar en equipo.

Índice General

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Calibración y puesta en marcha de un sistema de decapado continuo con láser

DOCUMENTO Nº1: MEMORIA

Máster Universitario en Automática e Informática Industrial

Autor

Javier Gamir Artesero

Tutor

Carlos Ricolfe Viala

CURSO ACADÉMICO: 2023/2024

Índice

Tabla de figuras

1 Objeto

1.1 Objeto

El presente proyecto tiene como objetivo desarrollar e implementar un sistema de decapado láser continuo, empleando una cámara, un láser industrial de fibra y una cinta transportadora. El sistema está diseñado para optimizar el proceso de decapado en una línea de producción, incrementando la eficiencia en comparación con los sistemas de decapado estático. Además, permite el procesamiento de piezas largas, que no pueden ser tratadas mediante métodos estáticos, mediante la integración de tecnologías robóticas y visión artificial.

El proyecto incluirá un análisis exhaustivo de las diferentes tecnologías de cámaras 3D disponibles en el mercado para la reconstrucción tridimensional en un entorno de producción industrial. Se seleccionarán la técnica y los componentes más adecuados, basándose en criterios como la resolución de imagen, la capacidad de zoom, el costo y el tamaño de los equipos.

El sistema de visión artificial desarrollado permitirá la identificación precisa de las áreas específicas de las piezas que requieren decapado, generando en tiempo real el patrón de decapado que será enviado al láser.

Para coordinar con precisión la zona de detección con la zona de decapado, se implementará un encoder que establecerá la sincronización entre ambas áreas y el movimiento de la cinta transportadora.

2 Antecedentes

2.1 Introducción

Actualmente, la visión artificial está ganando popularidad como una tecnología cada vez más adoptada por las empresas para automatizar procesos y aumentar la eficiencia en la producción. Esta tecnología proporciona a las máquinas la capacidad de "ver" y comprender su entorno, lo que les permite llevar a cabo tareas de manera autónoma y precisa.

La visión artificial se está implementando en diversos sectores, como la industria automotriz (como es el caso particular de la pieza utilizada en el trabajo), la alimentaria y la logística, entre otros. En estos campos, los robots

dotados de sistemas de visión artificial pueden ejecutar tareas como el ensamblaje de componentes, la clasificación de productos y la inspección de calidad, entre otras.

La integración de la visión artificial con la robótica ha facilitado el desarrollo de sistemas de automatización industrial más avanzados. Los robots equipados con sistemas de visión artificial tienen la capacidad de detectar y reconocer objetos y patrones complejos, lo que les permite realizar tareas con mayor precisión y rapidez.

También se pueden encontrar sistemas de visión artificial sin la necesidad de robots que permiten tareas como localización, reconstrucción 3D e identificación entre otros en tiempo real.

2.2 Motivo de Diseño

La razón fundamental que impulsa este proyecto radica en la necesidad de la empresa SRG Global de implementar un sistema de decapado continuo para una pieza específica en su línea de producción. La motivación técnica subyacente es optimizar el proceso de decapado estático, el cual ha sido trabajado e investigado por el laboratorio LASA del AI2.

De forma más concreta este proyecto busca mejorar velocidad de decapado por el decapado estático investigado y la posibilidad de inclusión de piezas largas al mismo.

2.3 Métodos de Reconstrucción 3D

Para el proyecto se han estudiado diferentes métodos de reconstrucción de piezas 3D, este apartado presenta dichas técnicas y algunas de sus características, posteriormente se presentará cual se ha escogido y el por qué.

2.4 Estereovisión

La técnica de estereovisión utiliza dos o más cámaras, colocadas a una distancia conocida entre sí, para capturar imágenes simultáneamente. A partir de las diferencias entre las imágenes obtenidas y mediante cálculos basados en

la disparidad de píxeles correspondientes a un mismo punto en el espacio, se puede determinar la profundidad de la escena.

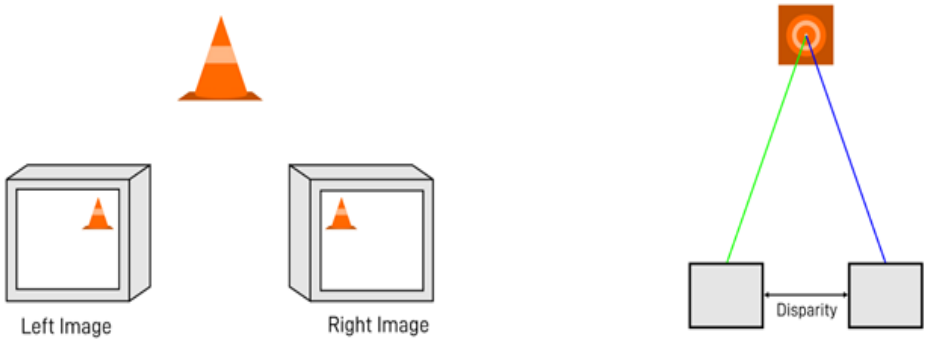


Figura 2.1 Aplicación Estereovisión

2.5 Cámara 3D con luz estructurada

Una cámara con luz estructurada proyecta un patrón de luz sobre un objeto y analiza las deformaciones del patrón en las imágenes capturadas para calcular la profundidad. En el decapado estático realizado en el Laboratorio LASA, se utiliza una cámara con estas características.

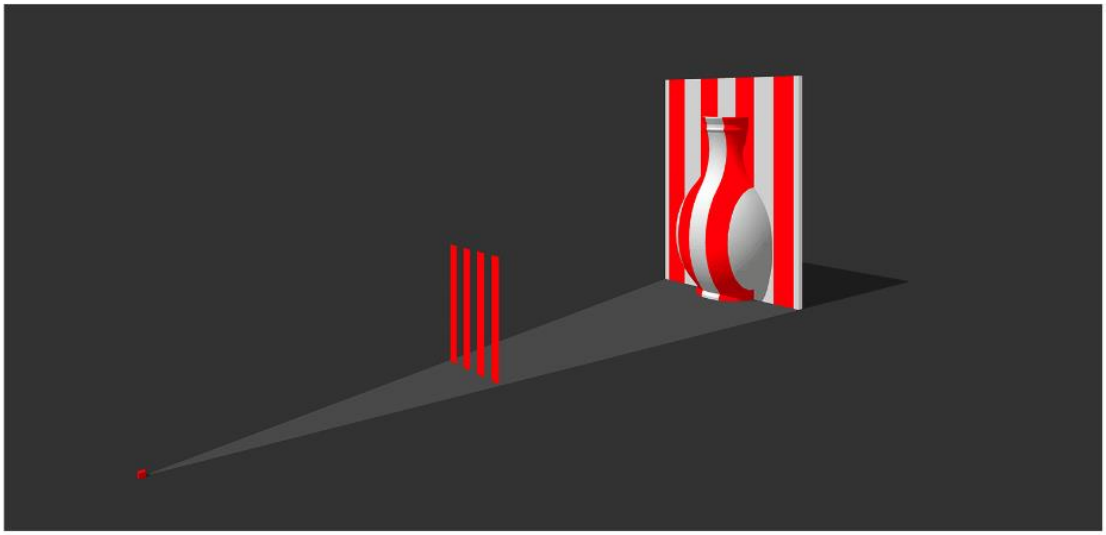


Figura 2.2 Aplicación Cámara con Luz Estructurada

2.6 Triangulación Láser

La triangulación láser emplea luz estructurada para realizar mediciones. Un rayo láser se proyecta sobre la superficie del objeto y es capturado por una cámara 2D desde un ángulo diferente al del láser. Las variaciones en la forma

del objeto causan un desplazamiento en la proyección del láser dentro de la imagen, lo que permite calcular la distancia y generar un perfil 3D preciso.

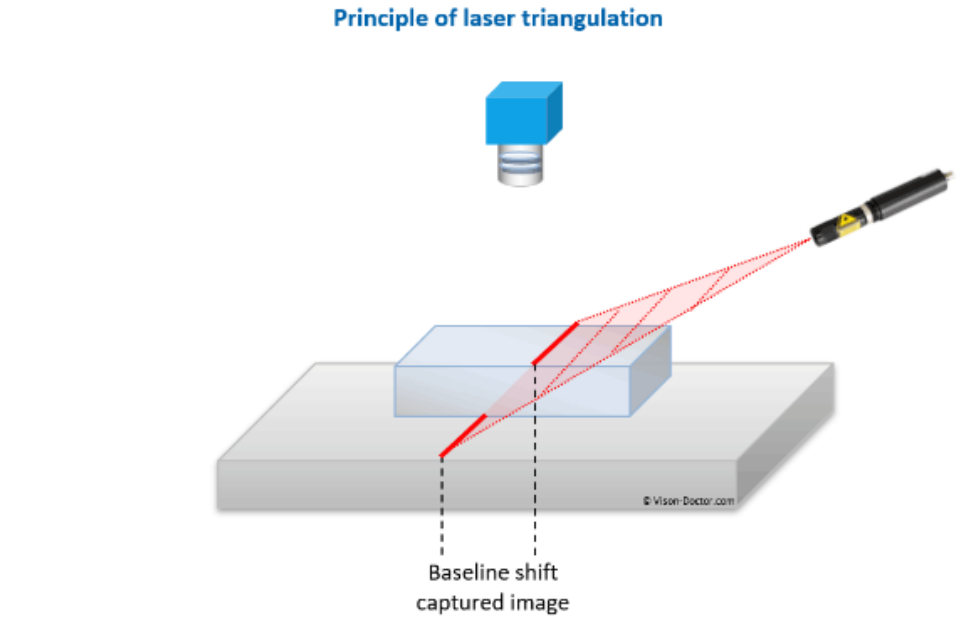


Figura 2.3 Aplicación Triangulación con Láser

2.7 Tiempo de vuelo

Esta técnica consiste en medir el tiempo de ida y vuelta que tarda la luz en viajar desde una fuente luminosa en la cámara, hasta un punto de la superficie reflectante. Este tiempo de viaje se denomina comúnmente tiempo de vuelo (ToF), con él se calcula la distancia que hay de la cámara al objeto.

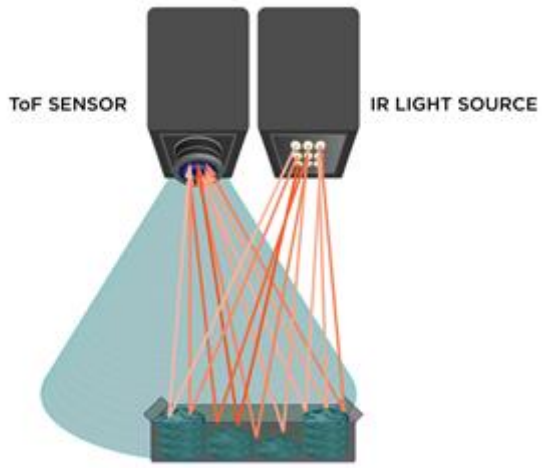


Figura 2.4 Aplicación Tiempo de vuelo (TOF)

Existen más técnicas de reconstrucción 3D, pero no fueron valoradas para el proyecto, entre ellas se encuentran la fotogrametría y reconstrucción 3D por las sombras (lo cuál era imposible de aplicar).

2.8 Métodos de Decapado láser

2.8.1 Decapado Estático

En el decapado estático, la pieza permanece en una posición fija mientras el láser actúa sobre ella, lo que permite una alta precisión en el control de las zonas de decapado, así como en parámetros como la velocidad y la profundidad del proceso.

No obstante, este método presenta desventajas, como la mayor duración del proceso debido a la falta de desplazamiento de la pieza, además de estar limitado por el tamaño de la pieza y la capacidad del láser y del espacio disponible.

2.8.2 Decapado Continuo

En el decapado continuo, el láser se desplaza a lo largo de la superficie del material mientras emite el haz, o bien, el objeto a decapar se mueve mientras el láser permanece estático. Este enfoque reduce la precisión en comparación con el decapado estático y está limitado por la velocidad de movimiento tanto del objeto como del láser.

Sin embargo, el movimiento continuo permite eliminar restricciones de tamaño, aumentar la velocidad del proceso y mitigar las posibles limitaciones relacionadas con el tamaño del haz láser, al permitir el desplazamiento de las piezas.

3 Estudio de Necesidades

3.1 Especificaciones del proyecto

El sistema ha de ser capaz de realizar el decapado de forma autónoma desde que se decide dar inicio al proceso hasta que se termine de decapar la zona deseada.

Será necesario emplear la celda de trabajo proporcionada por la empresa ya que trata de simular una estación de trabajo que pueda ser exportada/replicada en un posible futuro.

El sistema de visión artificial deberá de ser capaz de identificar la parte concreta de la pieza que se desea decapar en tiempo real y proporcionarle la información necesaria al láser de decapado.

Figura 3.1 Pieza Objeto de Decapado

Figura 3.2 Zona Objeto a Decapar en la pieza

3.2 Normativa tenida en cuenta para el proyecto

-Directiva 2014/35/UE del Parlamento Europeo y del Consejo de 26 de febrero de 2014 sobre la armonización de las legislaciones de los Estados miembros en materia de comercialización de material eléctrico destinado a utilizarse con determinados límites de tensión. **(D.C. 2014/35/UE)**

-Directiva 2006/42/CE del Parlamento Europeo y del Consejo, de 17 de mayo de 2006, relativa a las máquinas y por la que se modifica la Directiva 95/16/CE (refundición). **(D.C. 2006/42/CEE)**

-Real Decreto 186/2016, de 6 de mayo, por el que se regula la compatibilidad electromagnética de los equipos eléctricos y electrónicos. **(D.C. 2011/65/EU)**

-Real Decreto 187/2016, de 6 de mayo, por el que se regulan las exigencias de seguridad del material eléctrico destinado a ser utilizado en determinados límites de tensión. **(BOE-A-2016-4443)**

-Real Decreto 2200/1995, de 28 de diciembre, por el que se aprueba el Reglamento de la Infraestructura para la Calidad y la Seguridad Industrial. (**BOE-A-1996-2468).** El cual tiene como última modificación y por lo tanto vigente:

Real Decreto 1072/2015, de 27 de noviembre, por el que se modifica el Real Decreto 2200/1995, de 28 de diciembre, por el que se aprueba el Reglamento de la Infraestructura para la Calidad y la Seguridad Industrial. **(BOE-A-2015-13530)**

4 Planteamiento de Soluciones Alternativas

4.1 Elección de Soluciones Alternativas

En este apartado se presentan las distintas propuestas de solución para el proyecto de manera cuantitativa. La opción con mayor puntuación justificará la selección de los componentes. Los apartados destacados en verde indican las elecciones tomadas tras evaluar diferentes factores característicos de las distintas alternativas.

4.1.1 Sistema de Reconstrucción 3D

Alternativa Sistema 3D	Precio	Precisión	Facilidad de implementación	Robustez	Facilidad de calibración	Total
Estereovisión	1	1	1	1	1	5
3D Luz Estructurada	2	2	3	3	3	13
Triangulación Láser	4	4	2	2	2	14
Tiempo de vuelo	3	3	2	2	2	12

La valoración seguida para cada apartado ha sido la siguiente:

-Precio: El coste monetario de la adquisición de todos los componentes del sistema.

-Precisión: Tanto la precisión que relaciona la resolución píxel/unidades reales como mm, la precisión para medir la profundidad o altura.

- Facilidad de implementación: La facilidad de poner en marcha el sistema.

- Robustez: Como afecta al sistema errores en la calibración o el desplazamiento de los elementos una vez calibrados.

-Facilidad de calibración: Que requerimientos tiene el sistema para ser calibrado.

4.1.2 Cámaras 2D

Dado que se ha escogido el sistema de triangulación láser, uno de los elementos que componen el mismo es una cámara 2D.

Alternativa Cámara	Precio	Tamaño	Resolución	Accesibilidad	Total
Cognex In-Sight 2000	1	1	2	3	7
Basler acA1300-60gc	2	2	3	1	8
Mikotron EoSens CL color	4	3	3	2	12
Intel Realsense D435i	3	4	1	3	11

Para la valoración de estas alternativas se han seguido los siguientes criterios:

- Precio: De más económico a más caro. (Ya disponíamos de la cámara escogida por ello el precio es el mejor).

-Tamaño: De menor a mayor tamaño.

-Resolución: Se valora la calidad de imagen capturada y la proporción píxeles/mm.

-Accesibilidad: Se evalúan las conexiones requeridas por la cámara (como USB, Ethernet o Framegrabber), los softwares compatibles y los lenguajes de programación que se pueden utilizar con ella.

4.1.3 Lente

Para mejorar la resolución, se ha decidido utilizar una lente en la cámara para que el surco de la pieza ocupe un mayor número de píxeles en la imagen.

Alternativa Lente	Precio	Resolución	Compatibilidad	Total
RASPBERRY-PI RPI-16MM-LENS	4	1	1	6
Basler Lense C125-0418-5M F1.8 4/3	2	2	3	7
Basler Lense Zoom Computar H6Z0812C-MP	3	1	3	7
Lente de Zoom Tamron M111FM16	1	3	2	6

De las dos opciones la escogida es la lente Basler Lense C125-0418-5M F1.8 4/3 debido a que es la que ofrece una mayor resolución.

Los criterios escogidos y su valoración han sido:

-Precio: Coste de la lente, a menor precio mejor valoración.

-Resolución: Proporción mm/píxeles que ofrece la lente.

Compatibilidad: La adaptación de la lente a la cámara.

4.1.4 FrameGrabber

Alternativa FrameGrabber	Precio	Velocidad	Tamaño	Adaptación	Total
Matrox Radient eV-CXP	2	3	1	1	7
Silicon Software microEnable IV AD4-C	3	1	1	2	7
National Instruments NI 1492	1	2	1	2	6

Los criterios seguidos junto a su justificación son los siguientes:

-Precio: Coste del framegrabber, a menor precio mejor valoración.

-Velocidad: Velocidad en fps del framegrabber.

-Tamaño: Cuanto más pequeño mejor.

-Adaptación: Facilidad de conexiones a la cámara escogida.

Dado que la aplicación funciona correctamente a 40 fps, la velocidad adicional del Matrox Radient eV-CXP no es necesaria. Por lo tanto, se selecciona el Silicon Software microEnable IV AD4-C, que es más económico.

4.1.5 Haz Láser

Alternativa Láser	Precio	Tamaño	Potencia	Fuente de Alimentación	Total
Keyence LJ-V7000 Series	2	2	2	1	7
Micro-Epsilon scanCONTROL 2700	1	1	3	1	6
Z-LASER ZM18B	3	3	1	1	8
ZhonNa laser lineal	4	3	1	2	10

Los criterios para la selección han sido:

-Precio: Coste del láser, a menor coste mejor valoración.

-Tamaño: Cuanto más pequeño sea el láser mejor valoración.

-Potencia: Cuanta potencia lumínica tiene el láser, a mayor potencia mejor porque es necesario menor tiempo de exposición para la cámara y por lo tanto es más rápido entre disparos.

-Fuente de alimentación: si la fuente de alimentación es de 5 voltios la valoración es de 2 ya que no es necesario ocupar la fuente fija de 24.

4.1.6 Encoder

Para poder reconstruir la pieza en 3D es necesario utilizar las tres componentes del espacio, con el sistema láser y cámara 2D es posible obtener

dos coordenadas, la profundidad y otra más, para la tercera será necesario utilizar un encoder con el que medir cuanto se avanza con cada pulso.

Alternativa Láser	Precio	Resolución	Precisión	Robustez	Total
Heidenhain ERN 1387	1	3	3	2	9
Sick DGS60-E1B00250	2	3	3	2	10
RLS RM08	3	1	1	1	7
CUI AMT203	4	2	2	1	8

Para la valoración de estas alternativas se han seguido los siguientes criterios:

- Precio: De más económico a más caro. (Ya disponíamos de la cámara escogida por ello el precio es el mejor).

-Resolución: Se valora la cantidad de pulsos por vuelta para la aplicación.

-Precisión: El tipo de encoder ha sido tomado en consideración, ya que según sea óptico, magnético o capacitivo la precisión varía.

-Robustez: La susceptibilidad del dispositivo a perturbaciones físicas o electromagnéticas varía según el tipo de componente.

El lenguaje de programación es otro factor que considerar en la selección. La cámara incluye un SDK exclusivo para C++, lo cual es ventajoso, ya que este lenguaje ofrece mayor velocidad en comparación con otros y permite la implementación de un posible sistema embebido.

4.2 Resumen de la elección de las Soluciones Alternativas

Han sido elegidas soluciones para el sistema que se desea implementar:

- Reconstrucción 3D mediante triangulación láser, gracias a su gran precisión y precio asequible para su implementación.

- Cámara 2D Mikotron EoSens CL color por su resolución, precio y tamaño.

- El Haz láser ZhonNa laser lineal debido a su precio y su reducido tamaño.

-El encoder Sick DGS60-E1B00250 por su resolución y precisión.

-La lente Basler Lense C125-0418-5M F1.8 4/3 por su resolución y compatibilidad.

-El Framgrabber Silicon Software microEnable IV AD4-C por su precio y velocidad.

5 Descripción detallada de la solución adoptada

En el siguiente apartado se detalla la solución adoptada para el presente proyecto. Primero, se especificarán la arquitectura del sistema y sus componentes. A continuación, se describirán los elementos adicionales necesarios para el sistema, seguidos de una descripción general del proceso. Finalmente, se explicará el desarrollo del software.

Se ha optado por incluir una representación gráfica en forma de organigrama para ilustrar la organización empleada en el diseño del proceso.

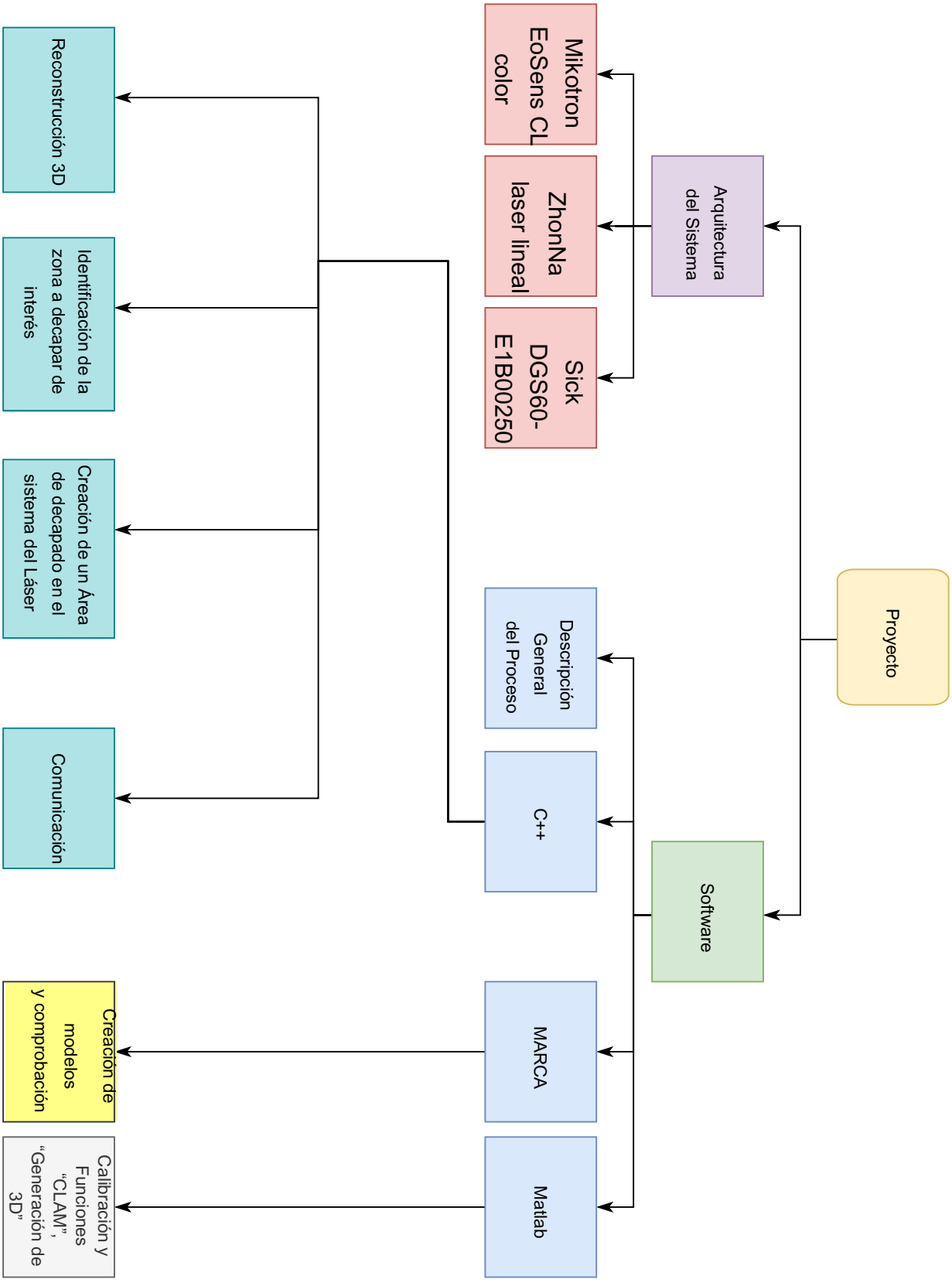


Figura 5.1 Organigrama del proyecto

5.1 Arquitectura del sistema

El sistema se encuentra compuesto de una cámara 2D Mikotron EoSens CL color, un haz láser de línea rojo ZhonNa laser lineal, una cinta transportadora, un encoder Sick DGS60-E1B00250, y un láser de decapado de MACSA ID F-9000 DUO MOPA.

Para la comunicación entre los elementos y su alimentación se ha realizado de la siguiente forma:

-El sistema central o nodo de comunicaciones es el PC, el cual se conecta al resto de elementos excepto la cinta transportadora.

-La cámara se conecta al PC mediante un framegrabber Silicon Software microEnable IV AD4-C conectados a los puertos A y B de ambos elementos. Para su alimentación se emplea una fuente alimentación.

-El encoder se encuentra conectado también al framegrabber, al pin que corresponde a la entrada 0.

-El encoder, para conectarse al framegrabber, se conecta a un puente electrónico que le proporciona la tensión y corriente necesarias, alimentado por la fuente de alimentación del PC. Este puente electrónico también se conecta al láser de decapado, permitiendo que el láser detecte el desplazamiento de la pieza una vez se recibe la señal de inicio de decapado. De esta manera, se asegura que el movimiento de la pieza no cause errores en el proceso de decapado.

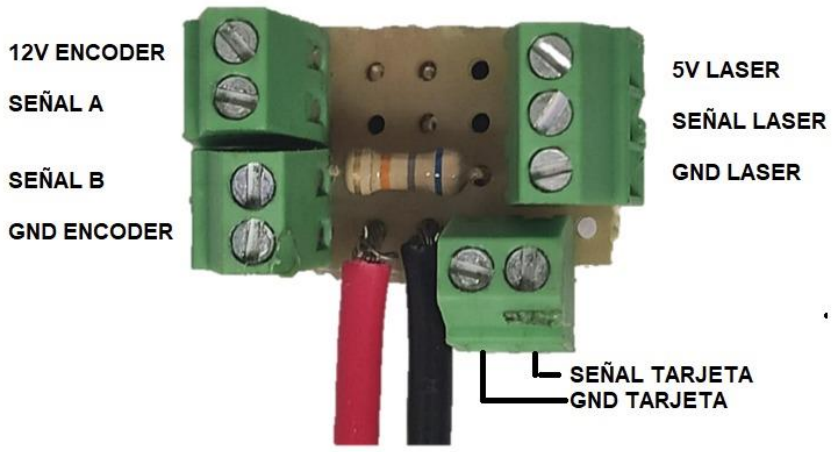


Figura 5.2 Puente Electrónico

-El haz láser es conectado a la fuente de alimentación.

-La conexión al láser de decapado es a través de Ethernet.

La comunicación del sistema sigue el siguiente flujo: la cámara adquiere una imagen y la transmite al PC, donde se procesa para identificar la zona a decapar. El PC genera un archivo con la información para el láser y lo envía al sistema de decapado. Simultáneamente, el PC recibe de manera continua los pulsos del encoder, utilizando esta información para ajustar la creación del archivo y determinar el momento preciso en el que enviar la señal de decapado al láser.

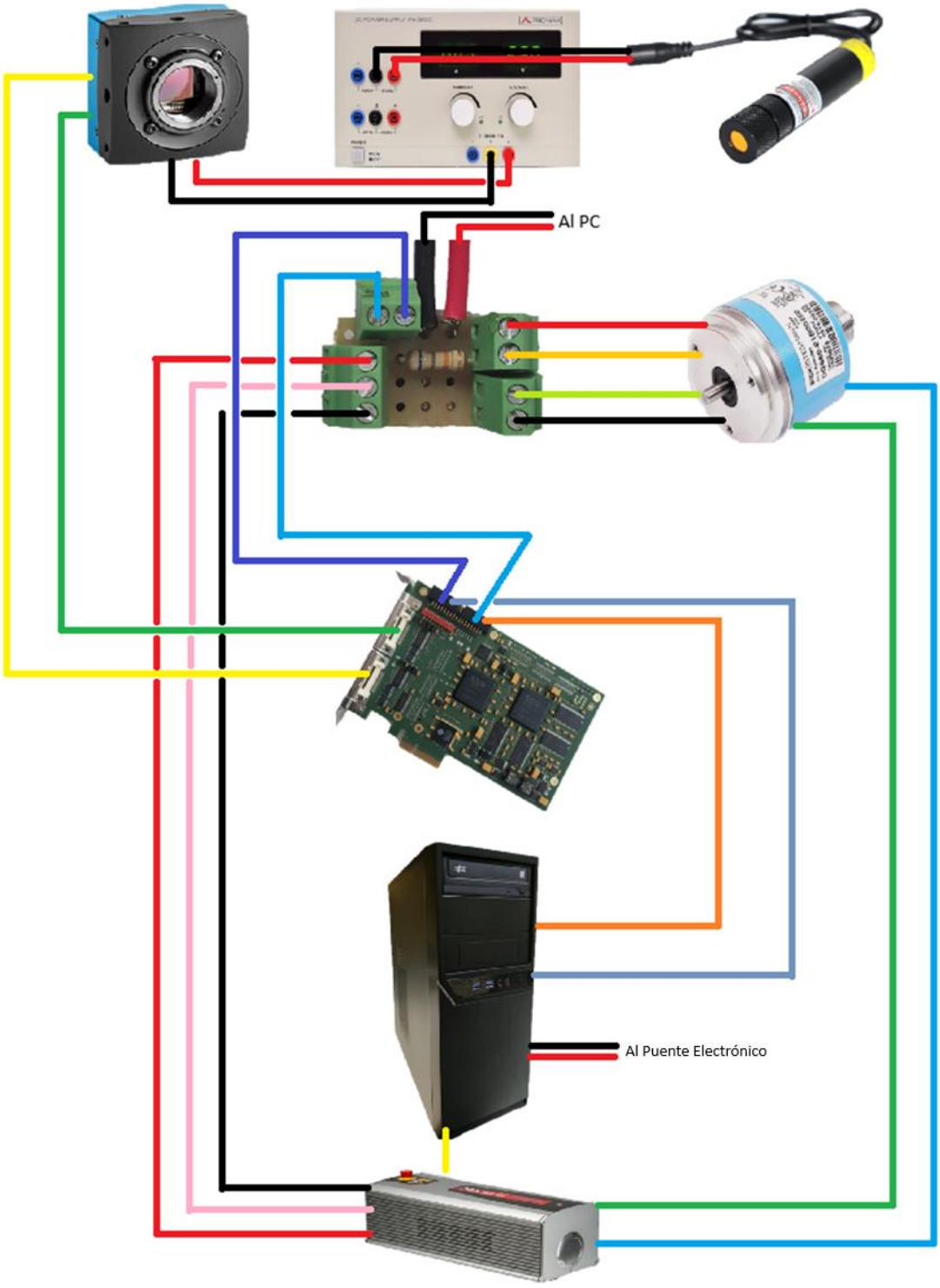


Figura 5.3 Conexionado Elementos

La configuración de cada componente del sistema se explica detalladamente en el anexo número 3 “Configuración de componentes”.

5.1.1 Elementos Extra

Para poder disponer de los elementos que componen el sistema en el espacio de la celda se han creado una serie de piezas para la cámara, el láser y el encoder. Se detallarán los planos de dichas piezas en la sección de planos del documento.

5.1.1.1 Elementos Extra-Encoder

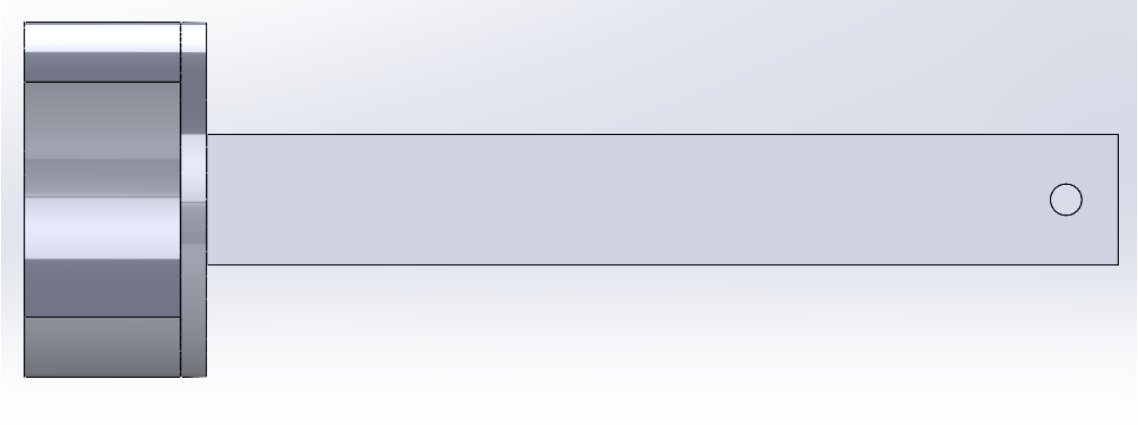


Figura 5.4 Soporte Movil para Encoder

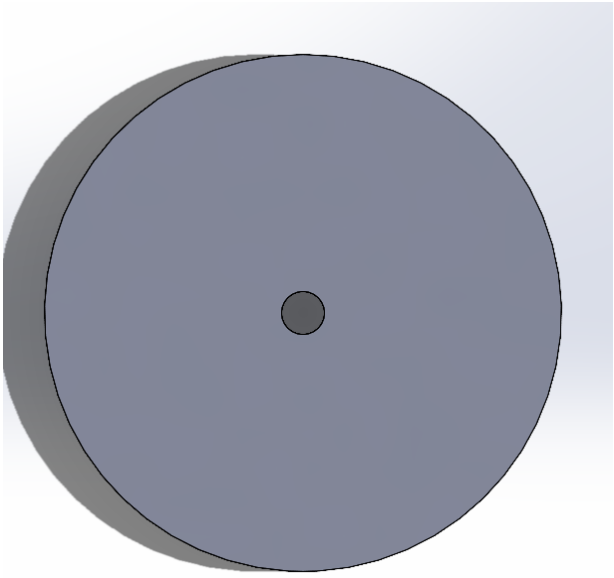


Figura 5.5 Rodamiento Encoder

5.1.1.2 Elementos Extra-Soporte para Barra Metálica

Ha sido necesario añadir unos soportes para fijar una barra de metal en la estructura de la celda a las ya existentes, se ha diseñado de forma que se disponga de una barra perpendicular a una ya existente:

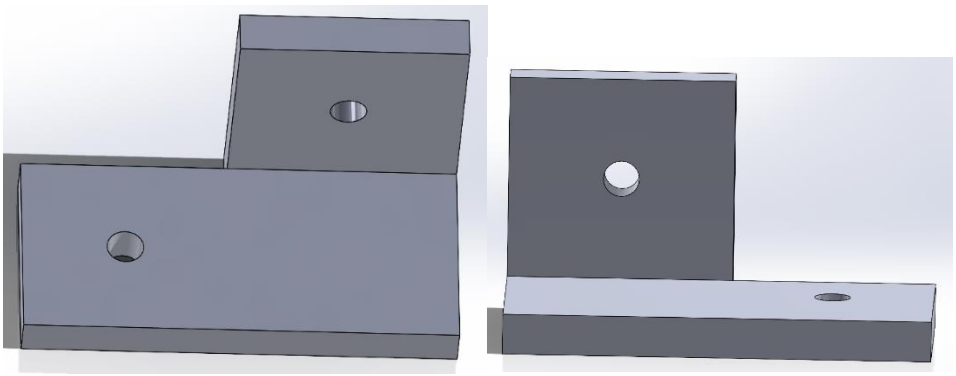


Figura 5.6 Soportes Izquierdo y Derecho para Barra de Metal

5.1.1.3 Elementos Extra-Cámara

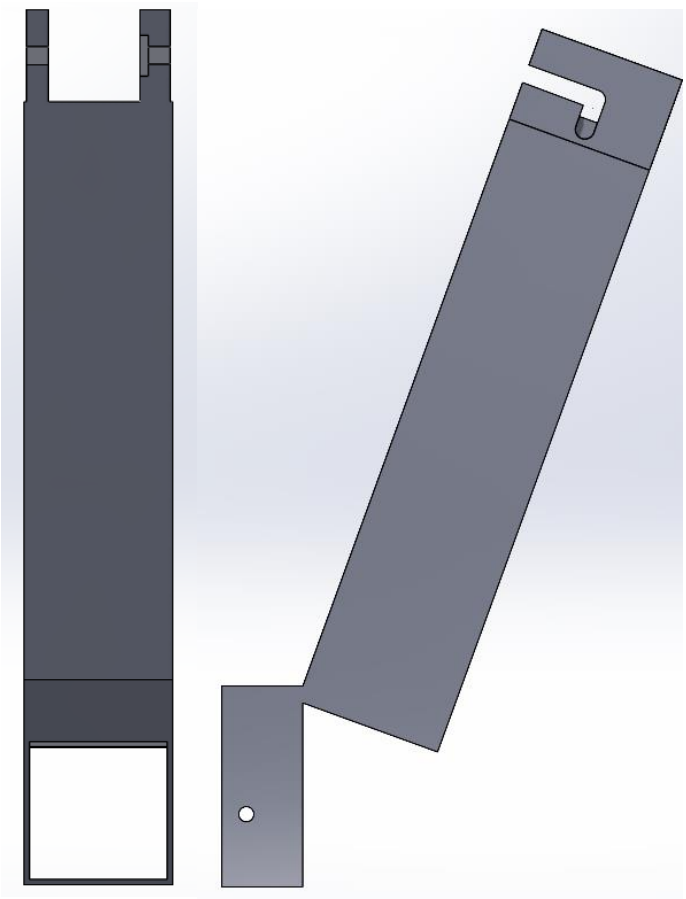


Figura 5.7 Soporte para la Cámara

5.1.1.4 Elementos Extra-Luz Estructurada

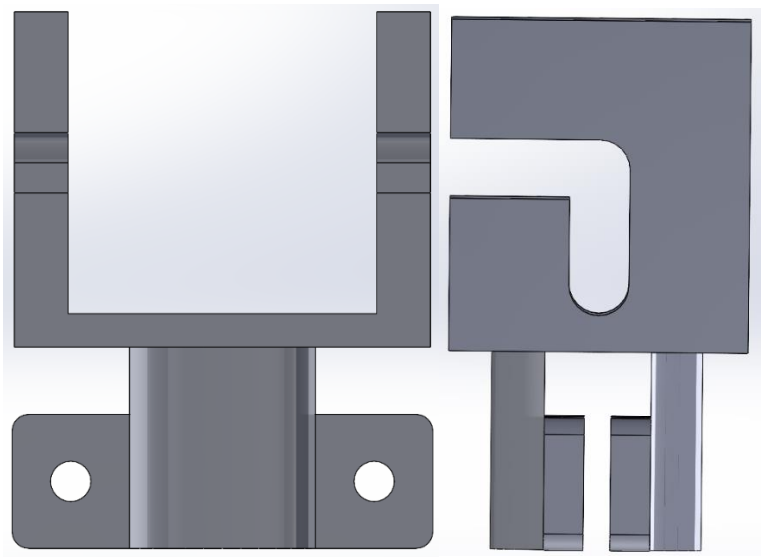


Figura 5.8 Soporte Luz Estructurada

5.2 Software

En el presente proyecto se han utilizado tres softwares distintos para la implementación del sistema. Matlab se ha empleado para la calibración del sistema y la generación de funciones, mientras que el funcionamiento del sistema se ha desarrollado en C++.

A continuación, se detallará el proceso de manera general, seguido de la calibración realizada en Matlab, para luego explicar el funcionamiento en C++. Todos los scripts utilizados en el proceso se adjuntan en el anexo 'Códigos'.

5.2.1 Descripción General del proceso

Para describir el proceso de manera general, se ha optado por utilizar un diagrama de flujo que ilustre de forma simplificada el funcionamiento completo del sistema.

En el proyecto se han implementado dos métodos de decapado distintos. El primero dispara líneas continuamente sobre la misma coordenada X mientras varía la coordenada Y, lo que, junto con el movimiento de la cinta transportadora, permite realizar un decapado continuo. El segundo método decapa un sector

completo, previamente definido, disparando varias veces de manera consecutiva para simular un decapado continuo.

Además, se han utilizado dos herramientas heredadas del proyecto de decapado estático. La primera es el LAM ("Linked Adaptive Matching"), que ha sido adaptada para este proyecto como CLAM ("Continuous Linked Adaptive Matching"), una herramienta desarrollada en Matlab por el investigador Carlos Ricolfe Viala para detectar la superficie plana en la que se desea decapar en una nube de puntos 2D. La segunda es una herramienta diseñada en Matlab, Mario Salvador, que genera una superficie 3D para el láser de decapado a partir de los puntos que la definen.

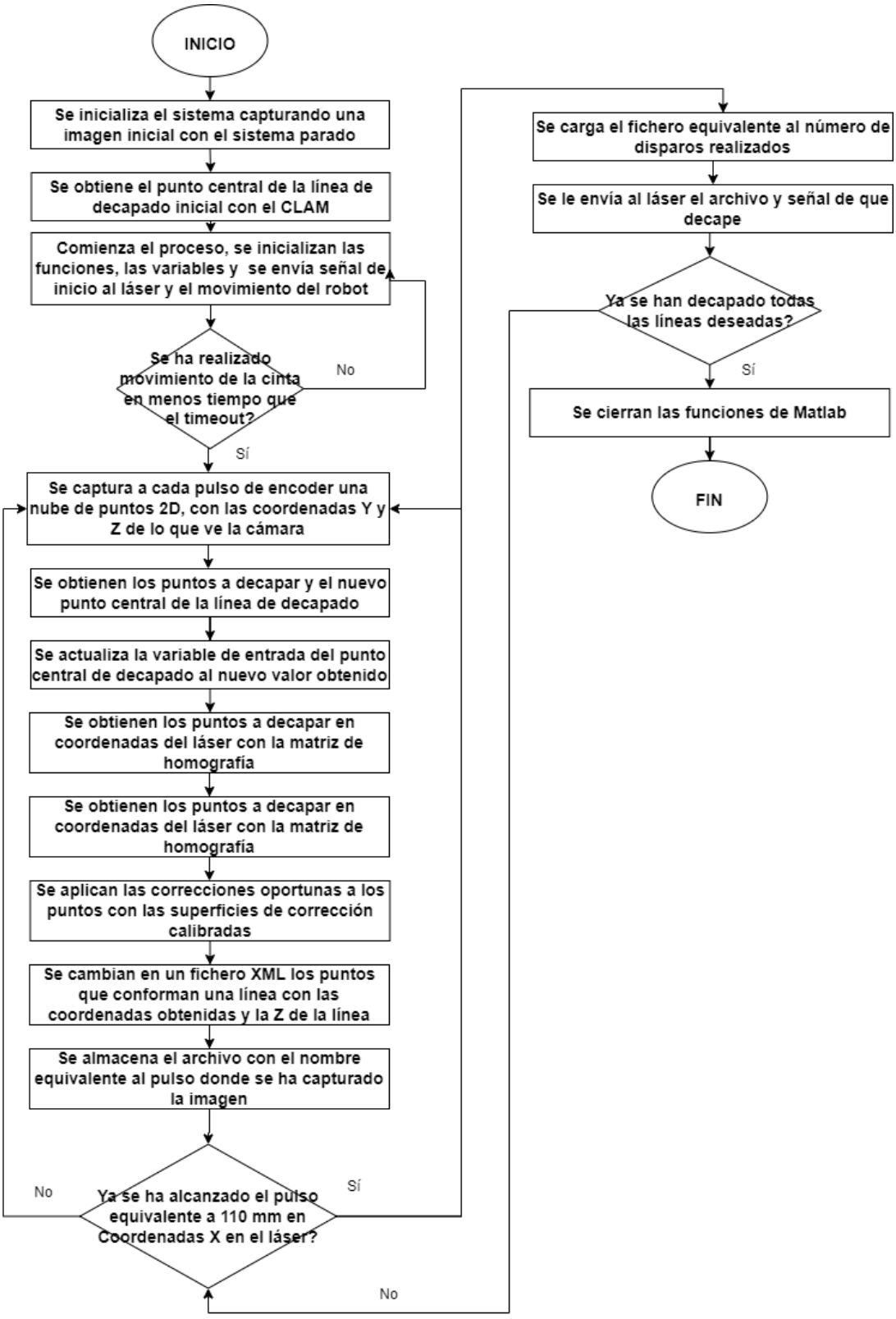


Figura 5.9 Diagrama General del proceso "Linea a Linea"

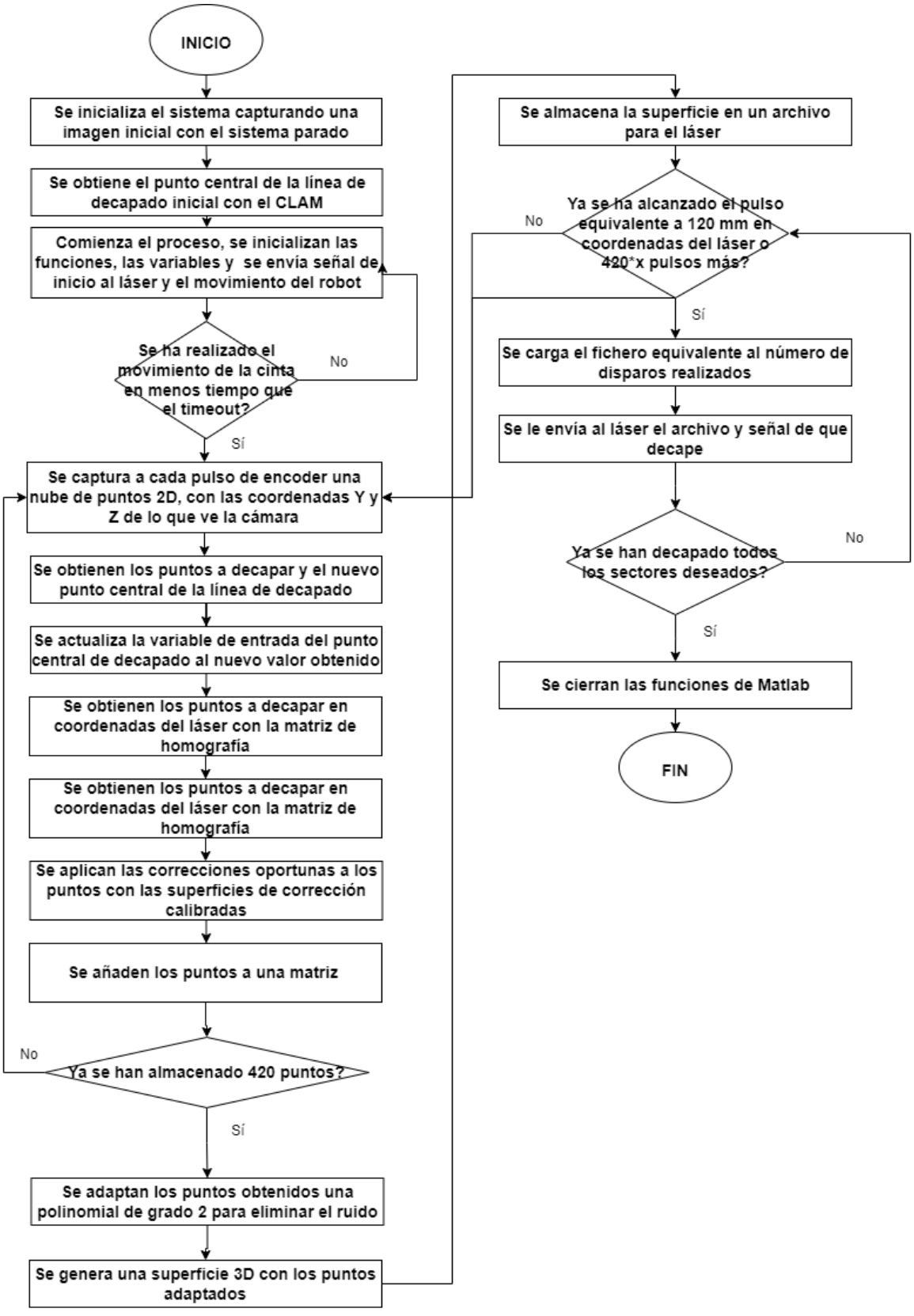


Figura 5.10 Diagrama General Decapado por Sectores

5.2.2 Matlab

Como se ha mencionado anteriormente, se ha utilizado el lenguaje Matlab para la calibración del sistema y para la generación de funciones que permiten identificar los puntos de interés a decapar en la pieza, así como para la creación de un archivo en formato XML para el láser.

5.2.2.1 Calibración Eje Y Cámara

Debido al ángulo de la cámara respecto al plano y a la altura del objeto, el ancho de este objeto se representará con una cantidad variable de píxeles. Por lo tanto, es necesario calcular la relación de píxeles a milímetros para cada columna de píxeles.

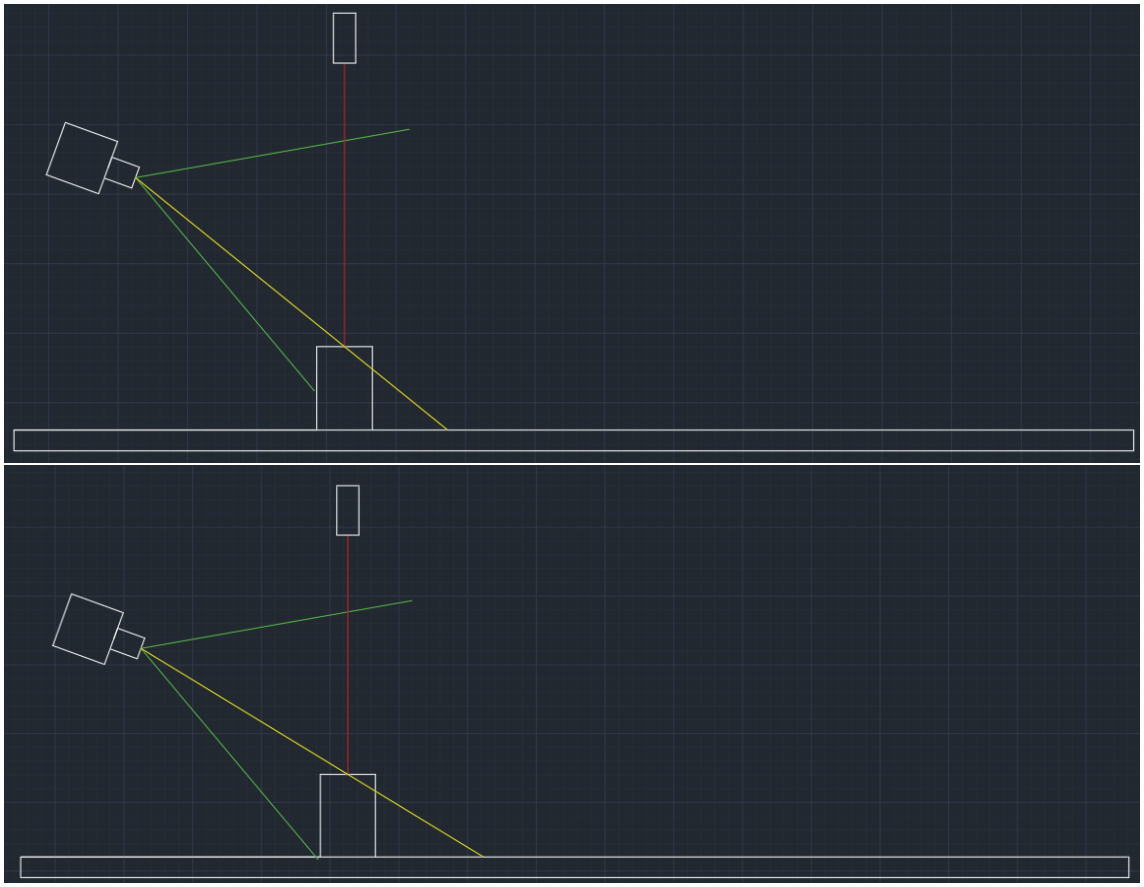


Figura 5.11 Qué capta la cámara según la altura del objeto

Para calibrar esta relación, se ha decidido capturar imágenes del mismo objeto a distintas alturas. A continuación, se presentan las dos alturas extremas que se han utilizado para la calibración::

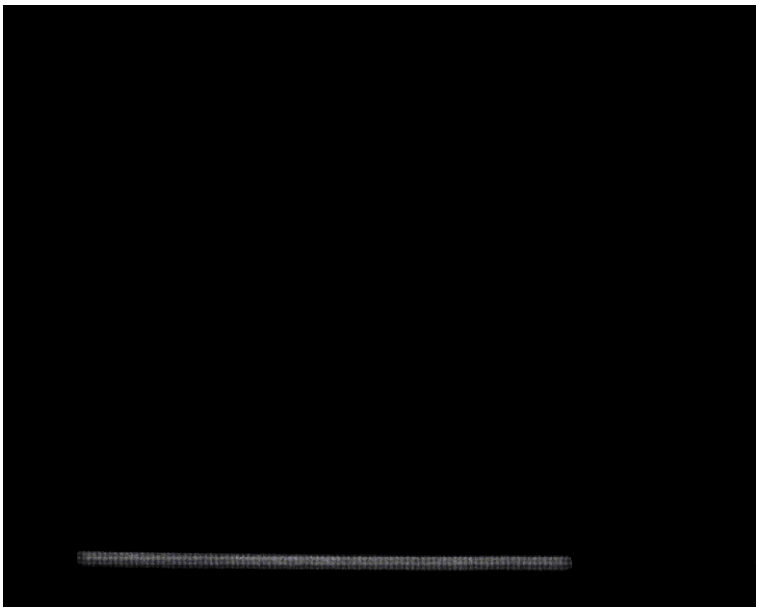


Figura 5.12 Imagen Calibración Ratio Pixeles/mm altura mínima

Figura 5.13 Imagen Calibración Ratio Pixeles/mm altura máxima

Como se puede observar, para una misma anchura del objeto, el número de píxeles que define esa anchura varía según la altura. Para calcular la relación de píxeles a milímetros en función de la altura, se ha creado un modelo que ajusta los datos a un polinomio de primer grado. En este modelo, la variable x representa la relación entre la anchura del objeto en milímetros y la anchura en píxeles, mientras que la variable y corresponde a la altura seleccionada. La altura se define como cero en el caso en que el láser de decapado está en su posición de referencia.

Como resultado se ha obtenido el siguiente polinomio:

$$ajusteY(x) = p1 * x + p2$$

$$p1 = 4.288e-06 \ (3.91e-06, 4.666e-06)$$

$$p2 = 0.05544 \ (0.0552, 0.05567)$$

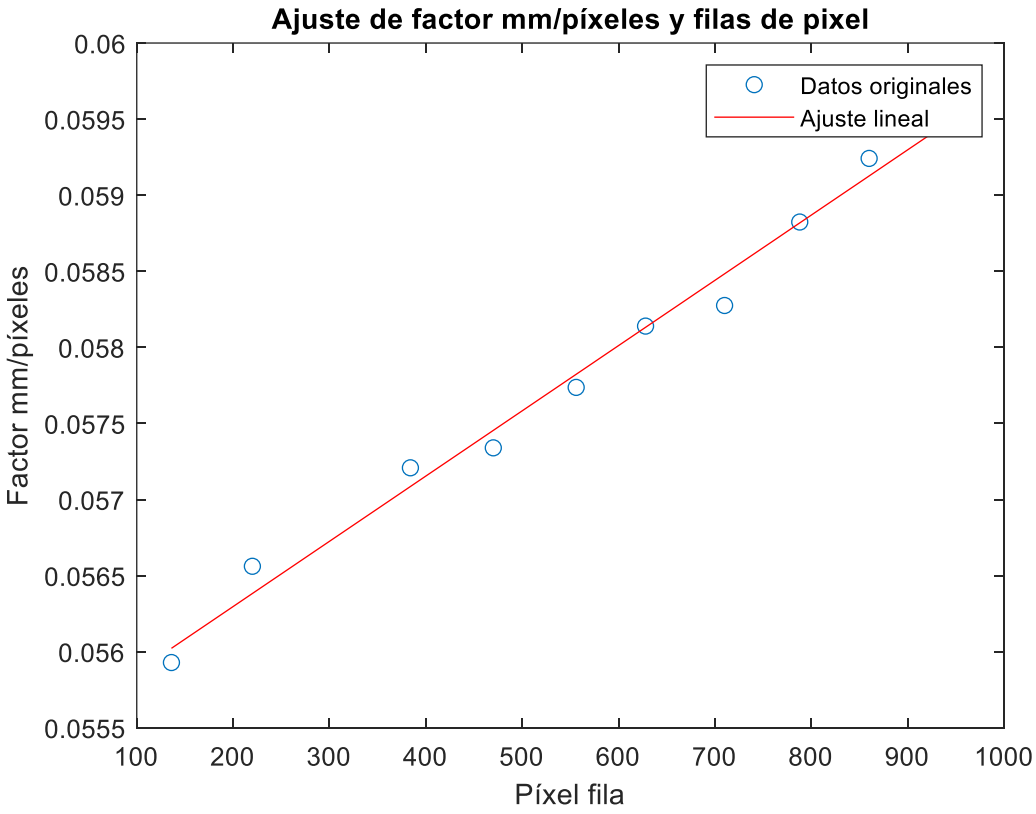


Figura.5.14 Modelo Obtenido junto a los resultados originales

Y se han obtenido un error medio de 1.9760e-04 y un error de desviación típica de 0.1076.

5.2.2.2 Calibración Eje Z Cámara

Para calibrar la relación píxeles/mm entre las columnas de la cámara y las dimensiones reales, se ha utilizado un método similar al empleado para calibrar el eje Y de la cámara. Se han utilizado las mismas imágenes y coordenadas en milímetros para la coordenada Z. Con estos datos, se ha desarrollado un modelo que relaciona los píxeles de la línea visible en la cámara con las alturas en milímetros.

A diferencia de la calibración en el eje Y el grado del polinomio a adaptar en la Z es de segundo orden:

$$ajusteZ(x) = p1 * x\text{^}2 + p2 * x + p3$$

$$p1 = -2.444e-05 \ (-3.403e-05, -1.485e-05)$$

$$p2 = -0.03007 \ (-0.04058, -0.01956)$$

$$p3 = 14.09 \ (11.55, 16.62)$$

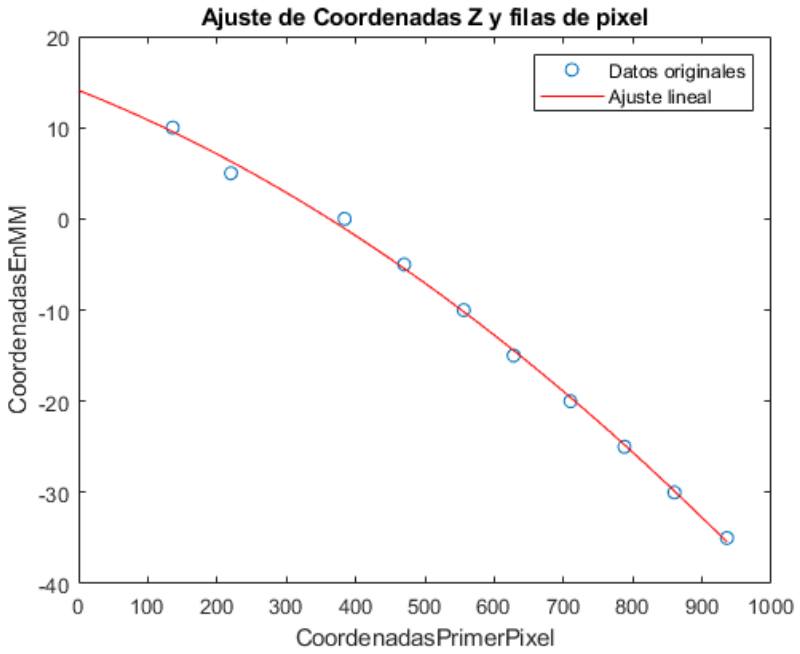


Figura 5.15 Modelo Obtenido junto a los resultados originales

Obteniendo un error medio de 1.5099e-15 y una desviación típica de 0.6679.

Como la nube de puntos para el CLAM se genera utilizando la coordenada Z en milímetros respecto a cierto píxel, y dado que la homografía se ha realizado con la calibración de Z descrita anteriormente, será necesario revertir la transformación a milímetros del resultado del CLAM. Debe devolverse al sistema de píxeles y aplicarse la calibración correspondiente.

Para eliminar la transformación previa para el CLAM sería:

$$CoordZPixel = 1009 - ResultadoZLAM * \frac{504}{330}$$

Y con esto se conseguiría la Z necesaria para aplicar la homografía de la siguiente forma:

$$ZHomografía = ajusteZ(CoordZPixel)$$

5.2.2.3 Calibración Centroide CCD

Para asegurar el uso correcto del CCD de la cámara, se ha decidido realizar una calibración que permita determinar el centro real en relación con el centro teórico.

Mediante la toma de dos nubes de puntos de placas decapadas a distintas alturas se puede observar que esto no se produce:

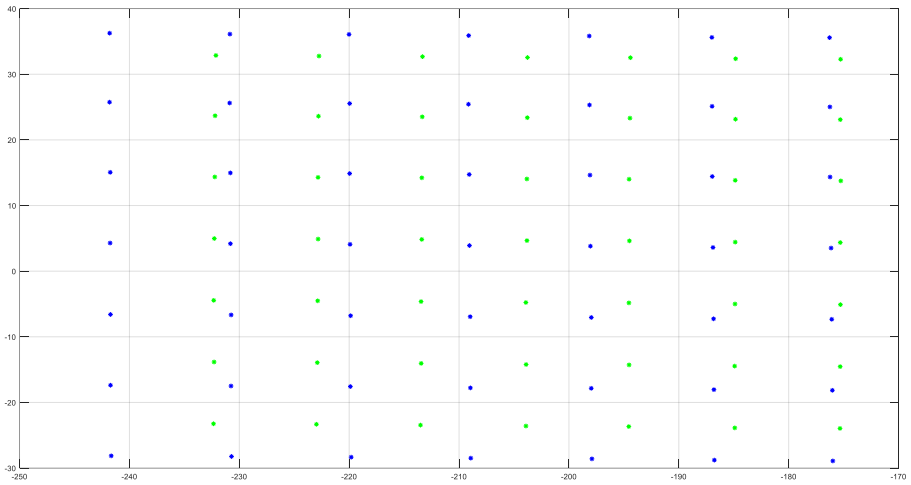


Figura 5.16 Vista de planta de los puntos de placas de calibración a distintas alturas

Los puntos se han representado en milímetros utilizando las funciones descritas anteriormente. Dado que la coordenada y se obtiene multiplicando el factor K_y (determinado mediante la calibración polinómica) por el valor del píxel en el que se encuentra el punto, se ha decidido proceder de la siguiente manera:

```
ptCloud_mm(:,2)=Ky.*(puntosCalibracionCamara(:,2)-640)
```

De forma que se genera la coordenada centrada, en este caso en el centro teórico en pixeles del CCD, para poder obtener el centroide real se ha modificado:

```
ptCloud_mm(:,2)=Ky.*(puntosCalibracionCamara(:,2)-640+offset)
```

Con esto se permite mediante iteraciones ir aumentando el offset y determinar para qué valor las nubes de puntos tienen alineados los puntos de la columna central de la nube, a continuación, se muestra el resultado, el código se encontrará en la sección de código:

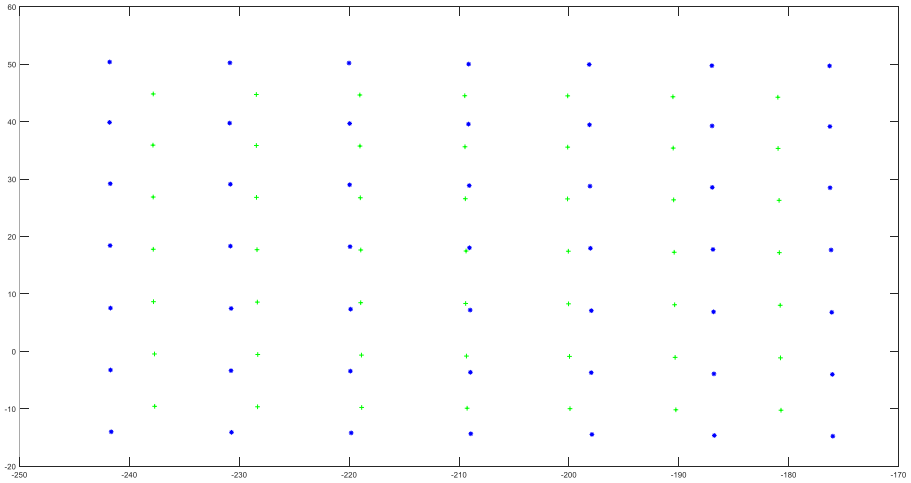


Figura 5.17 Vista de planta de los puntos de placas de calibración a distintas alturas corregido

5.2.2.4 Calibración Correspondencia Láser-Cámara

Para calibrar la relación Cámara-Luz Estructurada-Láser se ha empleado una homografía de una matriz de calibración realizada con el láser y vista con la cámara con su sistema de coordenadas.

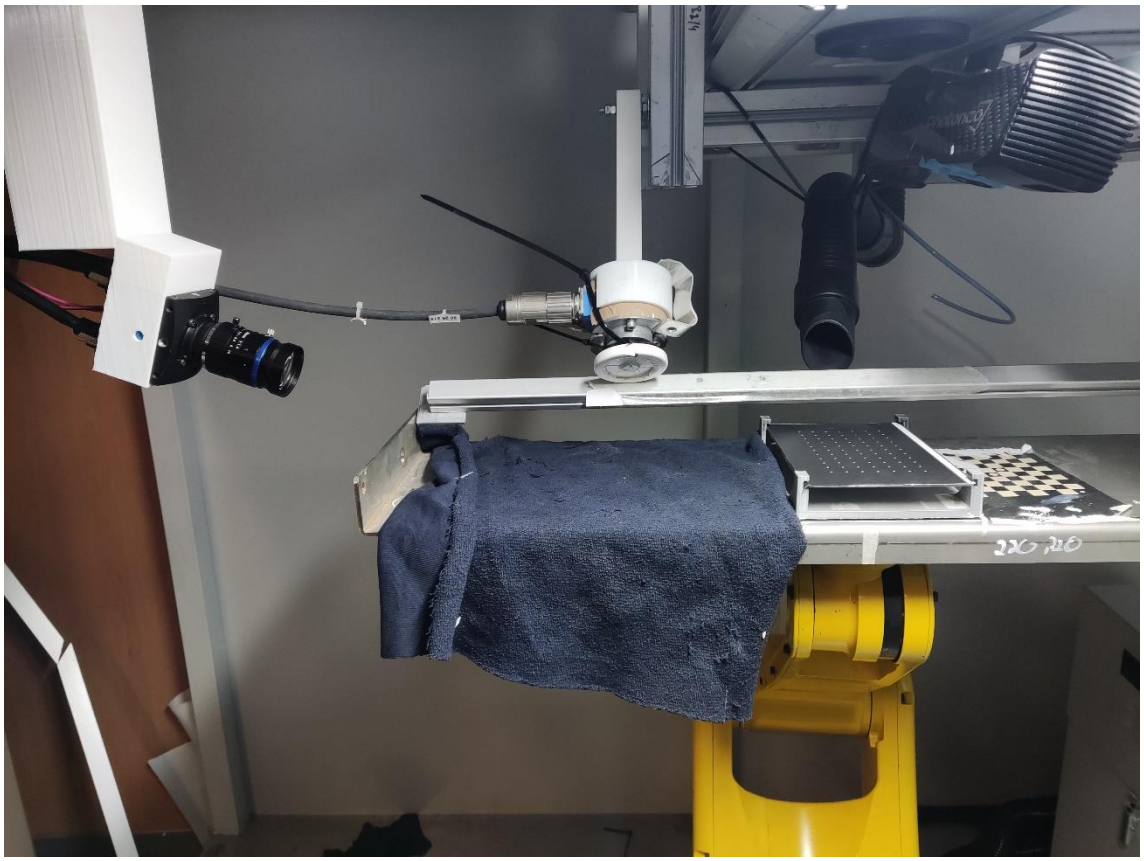


Figura 5.18 Disposición de los elementos para la calibración

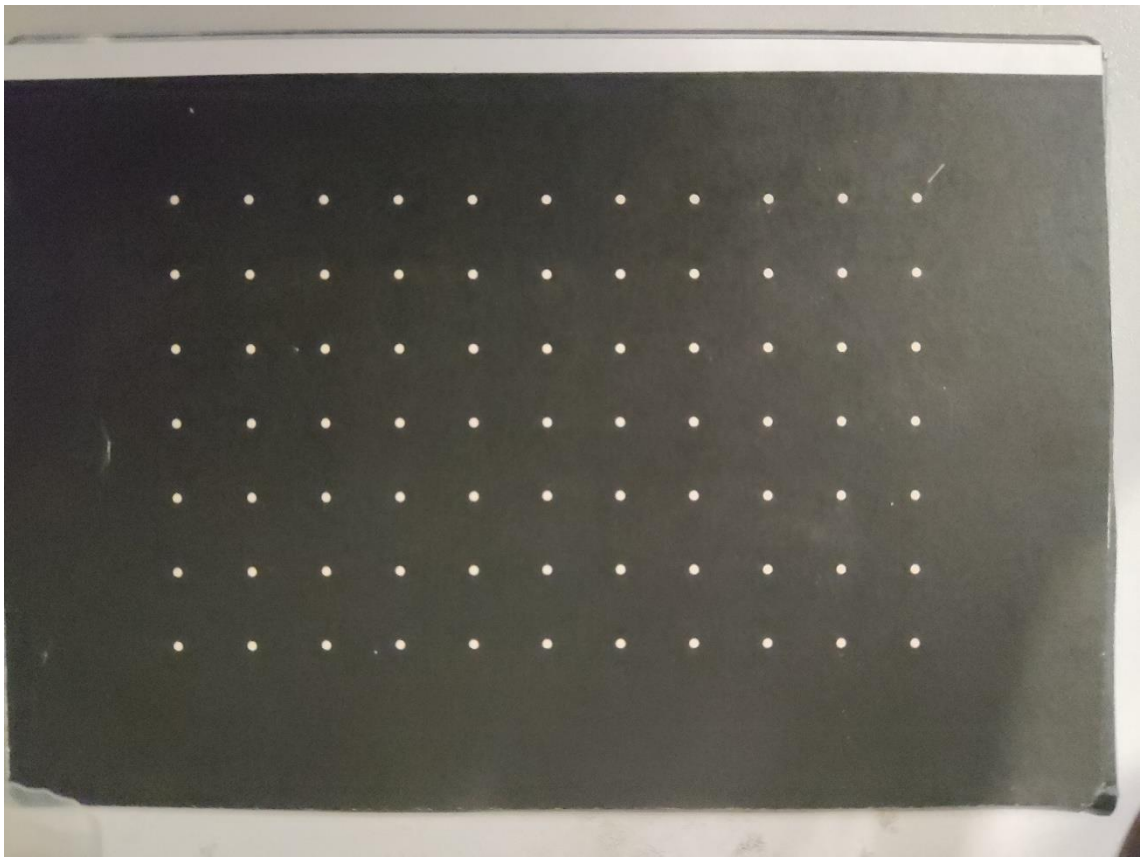


Figura 5.19 Plantilla de Calibración

El sistema de coordenadas de la cámara es, en x los pulsos que recibe el framegrabber del encoder, en la y la coordenada de la fila del píxel que ve la cámara y por último la z es en milímetros e inverso a la z de la cámara, concretamente es el resultado de $\left((1009 - PixelColumnna) * \frac{33}{504}\right) * 10$ esto es para favorecer a la herramienta CLAM a encontrar la superficie plana.

.

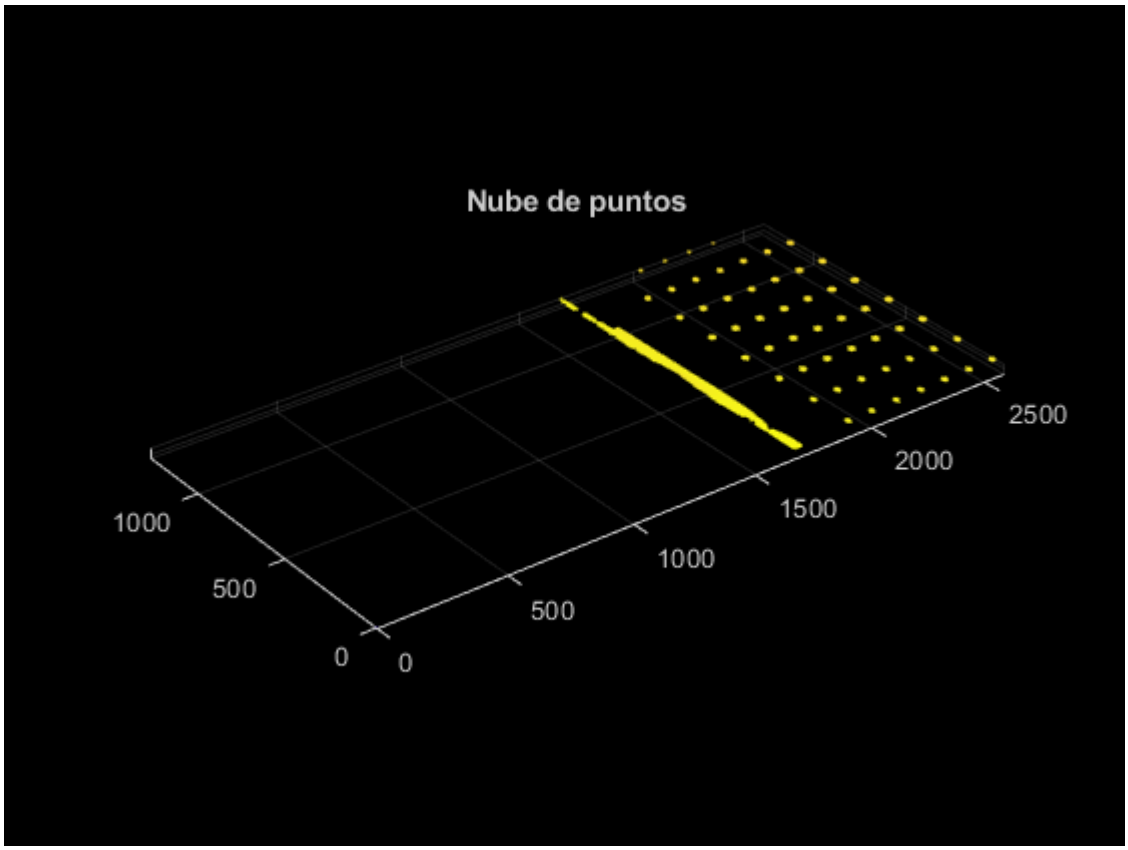
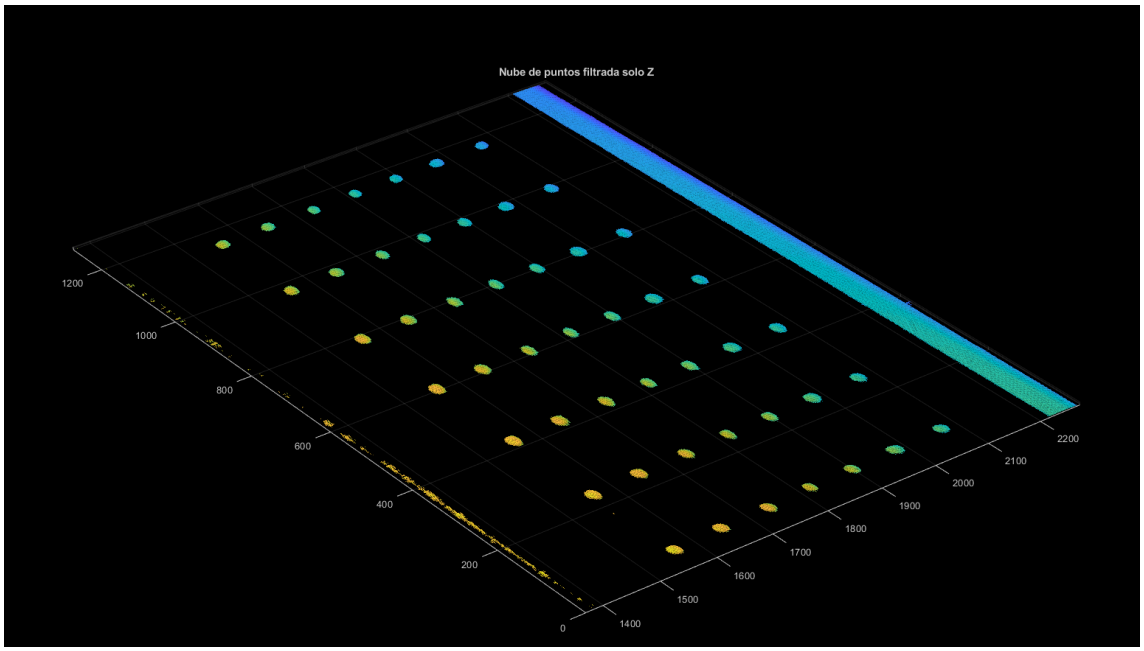
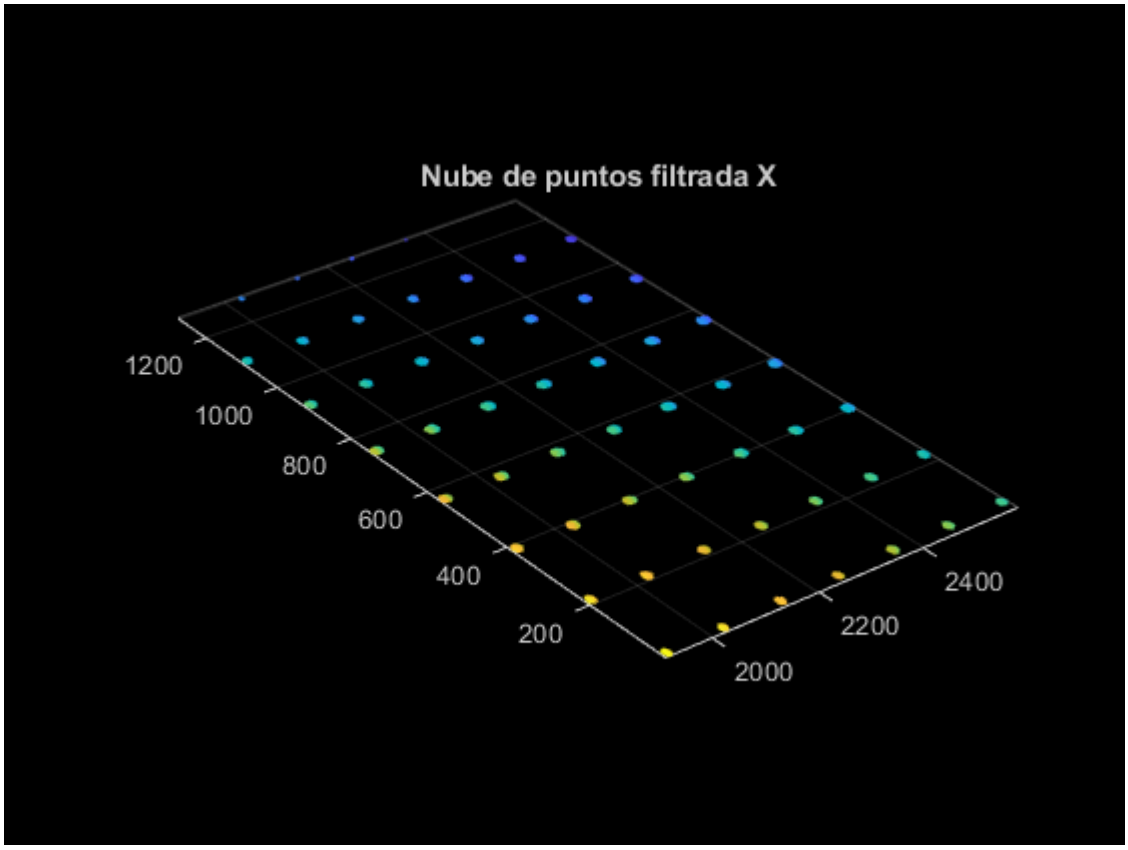


Figura 5.20 Nube de Puntos Captada por la cámara

Con la nube de puntos capturada por la cámara, es posible extraer una serie de centros de gravedad de puntos láser en coordenadas conocidas. Sin embargo, primero se debe segmentar la nube de puntos para optimizar las condiciones de análisis:


Nube de puntos filtrada solo Z
1200
1000
800
600
400
200
0
1400
1500
1600
1700
1800
1900
2000
2100
2200


Nube de puntos filtrada X
1200
1000
800
600
400
200
2000
2200
2400

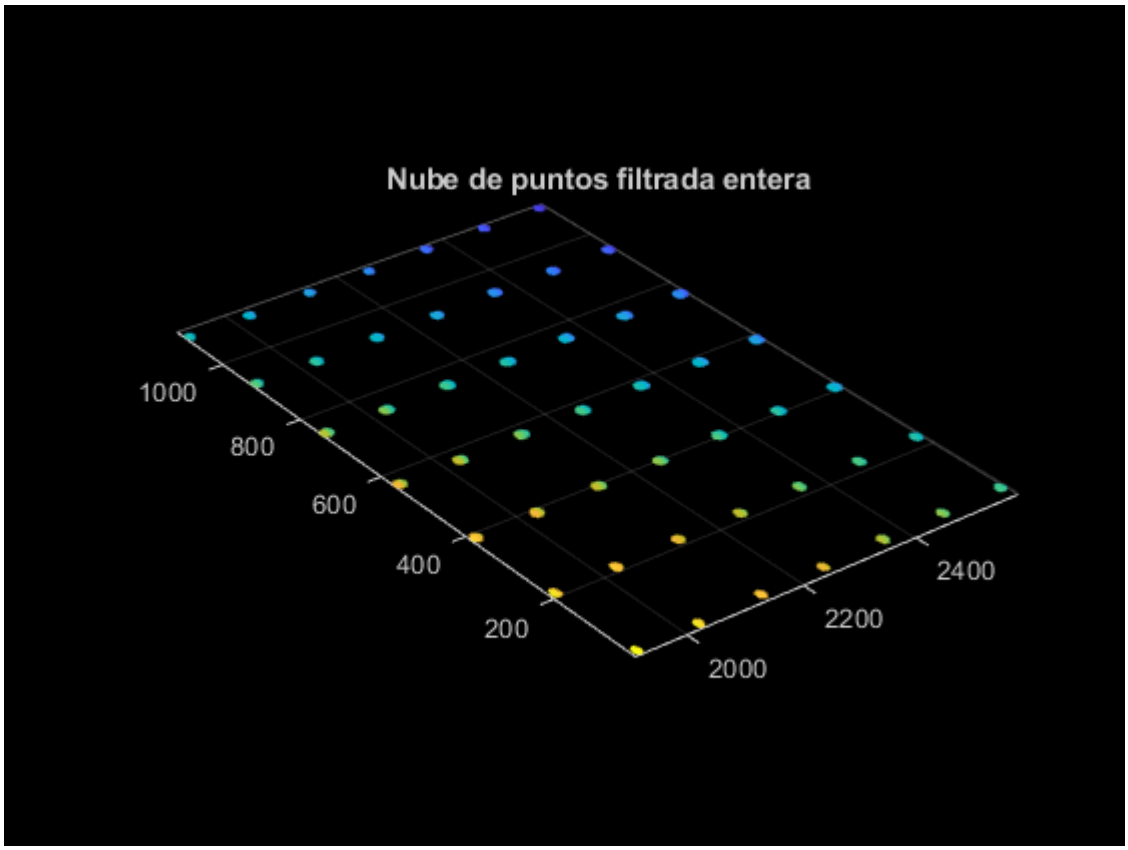


Figura 5.21 Nube de Puntos Captada por la cámara Segmentada

Una vez que se tiene la nube segmentada, se obtiene una imagen 2D para extraer las coordenadas x e y de los puntos de interés:

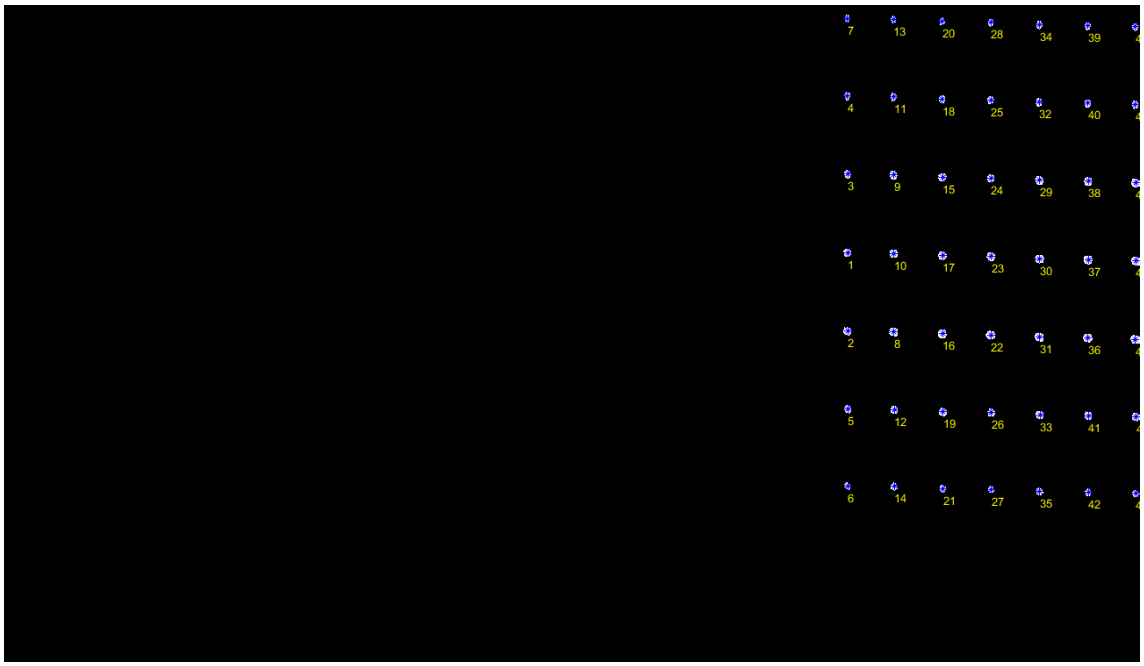


Figura 5.22 Imagen 2D Captada por la cámara

Una vez obtenida la imagen y los puntos de interés se ordenan estos para facilitar la homografía:

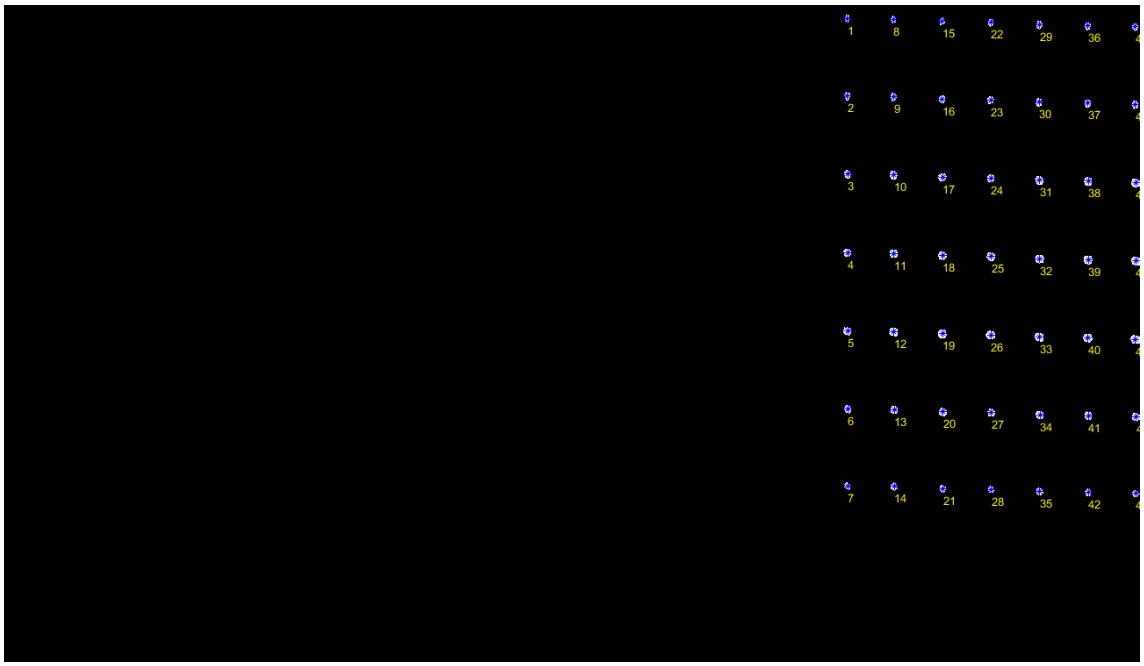


Figura 5.23 Imagen 2D Captada por la cámara Ordenada

La coordenada z se obtiene de la nube de puntos buscando el valor más cercano a las coordenadas x e y de cada punto de interés, lo que permite determinar las coordenadas en 3D.

La coordenada x obtenida se convierte a negativa, ya que se ha establecido que el eje z sea positivo en la dirección opuesta al láser, y el eje y sigue el mismo sentido que las filas de la cámara. Por lo tanto, el eje x está orientado en la dirección opuesta al acercamiento hacia el láser.

Para los puntos obtenidos de la imagen 2D, se aplica una conversión de unidades a milímetros. La coordenada x se multiplica por un factor obtenido de la imagen. La coordenada y se ajusta utilizando un factor determinado a partir de una línea que relaciona estos factores con la columna en la que se encuentran los puntos observados por la cámara, debido a la variación provocada por el ángulo de la cámara respecto al plano (esto se detalla en la sección "Calibración Eje Y Cámara"). Finalmente, para la coordenada z, la altura se obtiene mediante una polilínea que relaciona la altura en milímetros con la referencia del cero del láser y la columna vista por la cámara (esto se explica en la sección "Calibración Eje Z Cámara").

Una vez establecida la correspondencia entre los puntos y la matriz de homografía, se presenta el error asociado y se analiza la relación entre los sistemas de coordenadas.

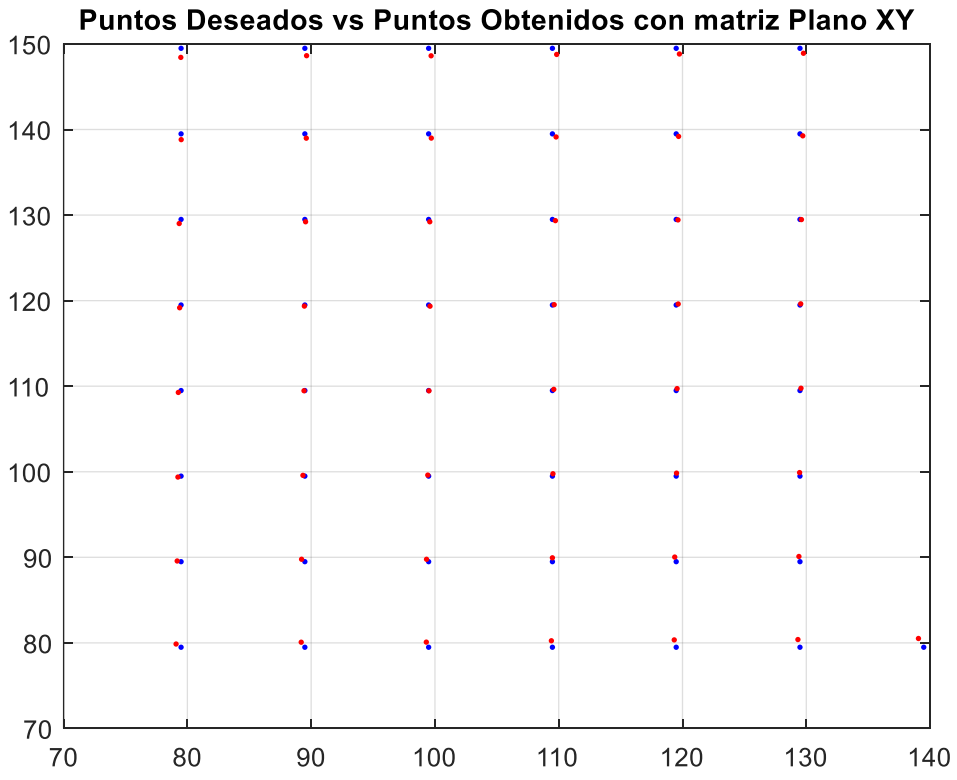


Figura 5.24 Error de la transformación Plano XY

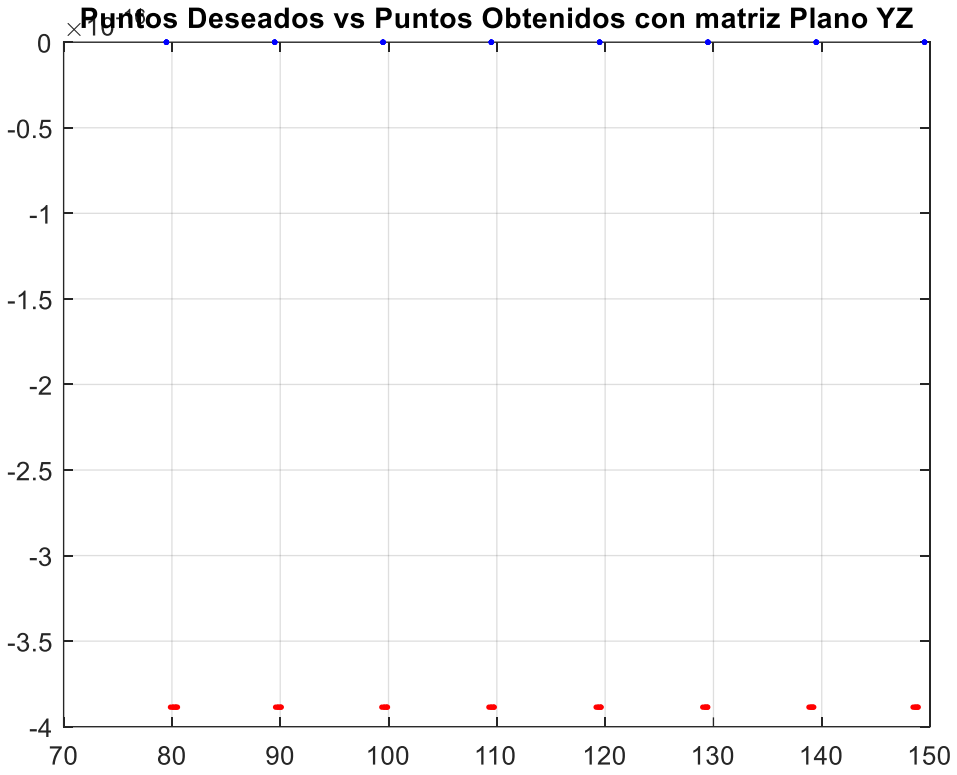


Figura 5.25 Error de la transformación Plano XZ

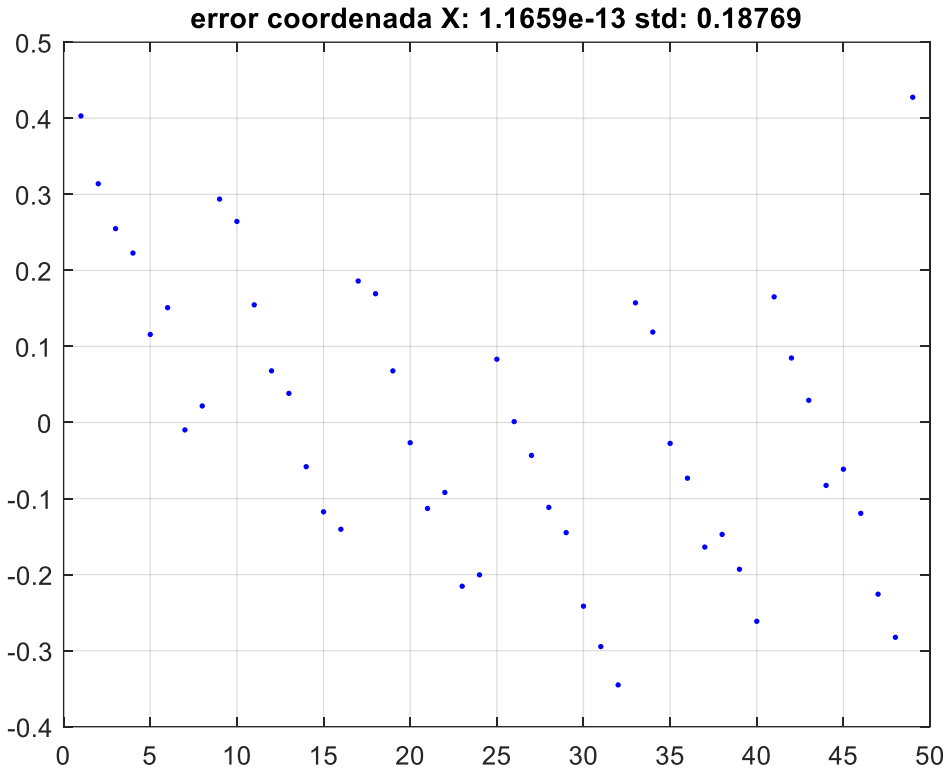


Figura 5.26 Error Calibracion Coordenada X

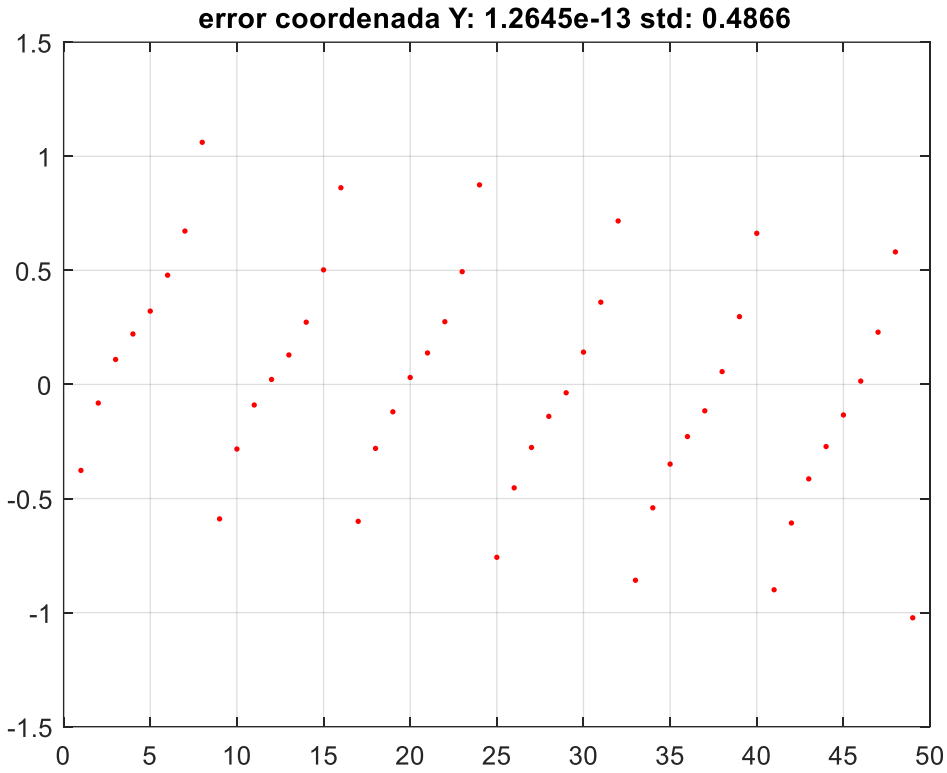


Figura 5.27 Error Calibración Coordenada Y

Como se puede observar, el error es mínimo y la calibración se considera correcta. Sin embargo, para verificar esta calibración, se realizó un experimento en el que se intentó recrear unos puntos generados por el láser, vistos por la

cámara y transformados. A continuación, se presenta el resultado del experimento:

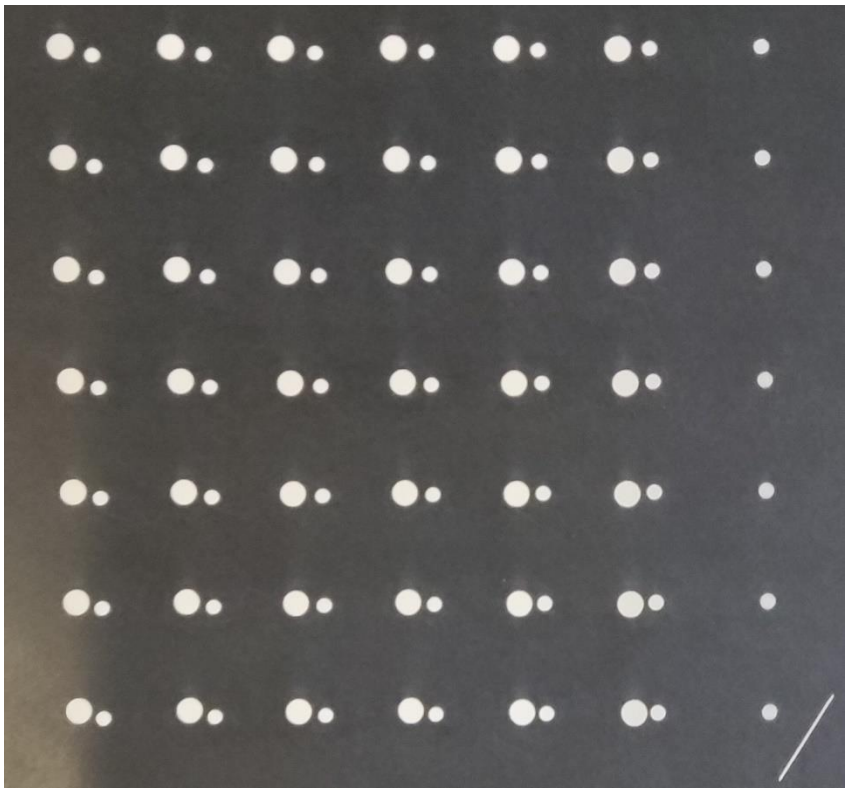


Figura 5.28 Comprobación calibrado

El resultado obtenido muestra un error inaceptable para el proyecto. El error en la coordenada X es debido al montaje del encoder, que se realizó en un robot antropomórfico que simula el comportamiento sobre una cinta transportadora; en la realidad, dicho error no debería presentarse. El error en la coordenada Y, que es mayor en los puntos que se encuentran más a la izquierda, está relacionado con el error en la coordenada X.

Para corregir los errores en X e Y, se ha decidido crear superficies que compensen las diferencias entre los puntos deseados y los puntos obtenidos para cada posición en X e Y. Al sumar estas correcciones a las coordenadas obtenidas con la transformación, se obtendrán los puntos correctos.

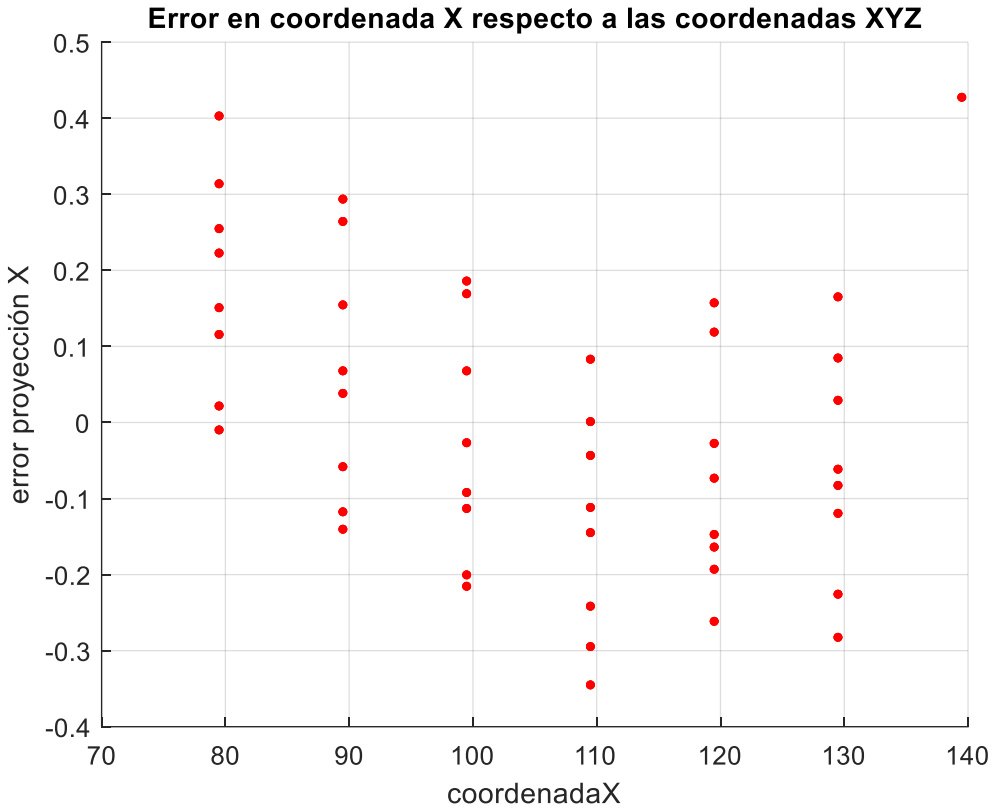


Figura 5.29 Error Proyección X Respecto Coordenada X

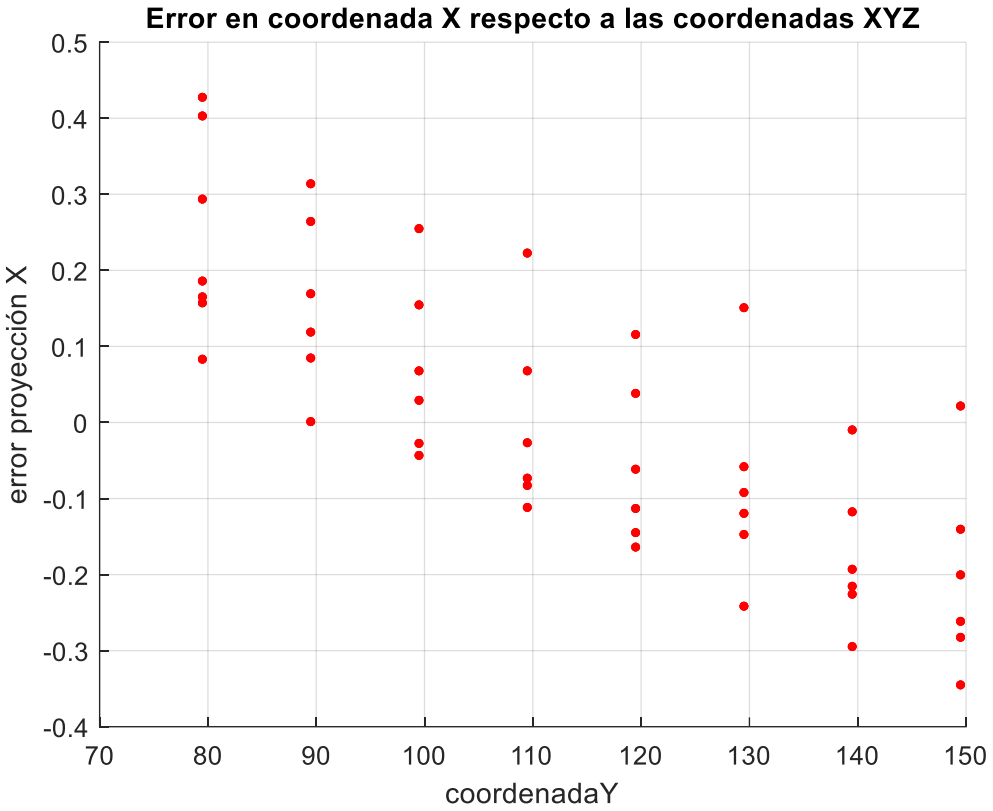


Figura 5.30 Error Proyección X Respecto Coordenada Y

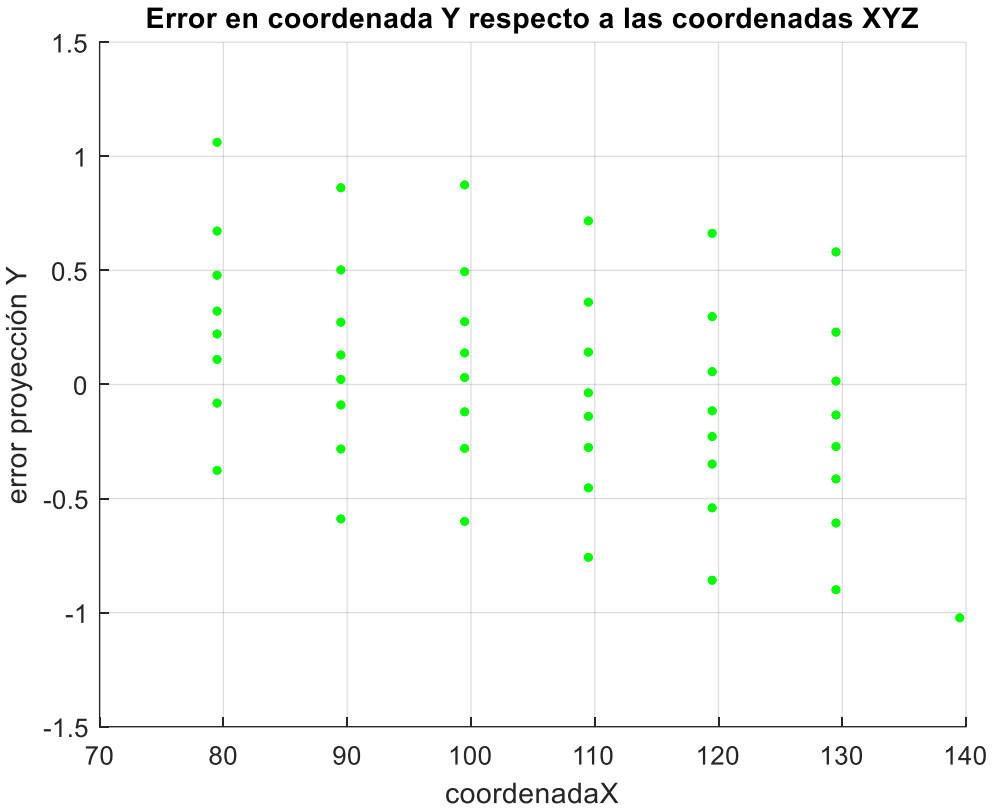


Figura 5.31 Error Proyección Y Respecto Coordenada X

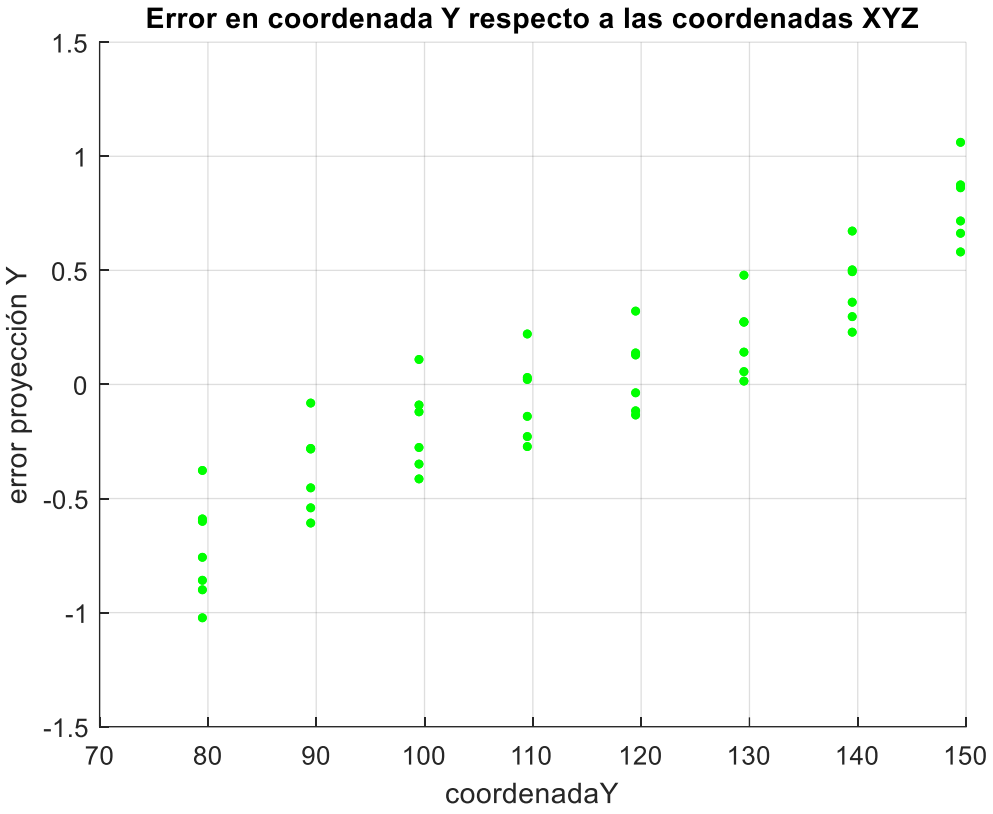


Figura 5.32 Error Proyección Y Respecto Coordenada Y

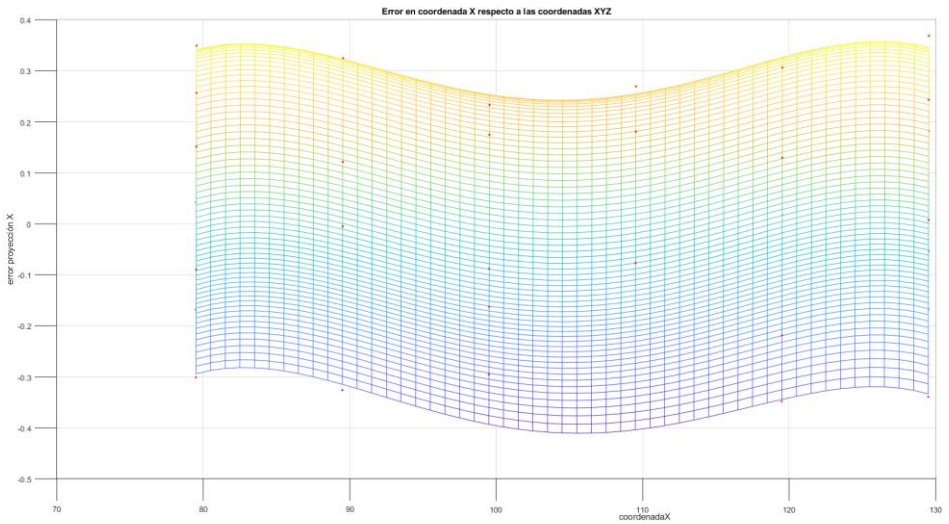


Figura 5.33 Superficie que se adapta al error de proyección en X respecto a coordenada X

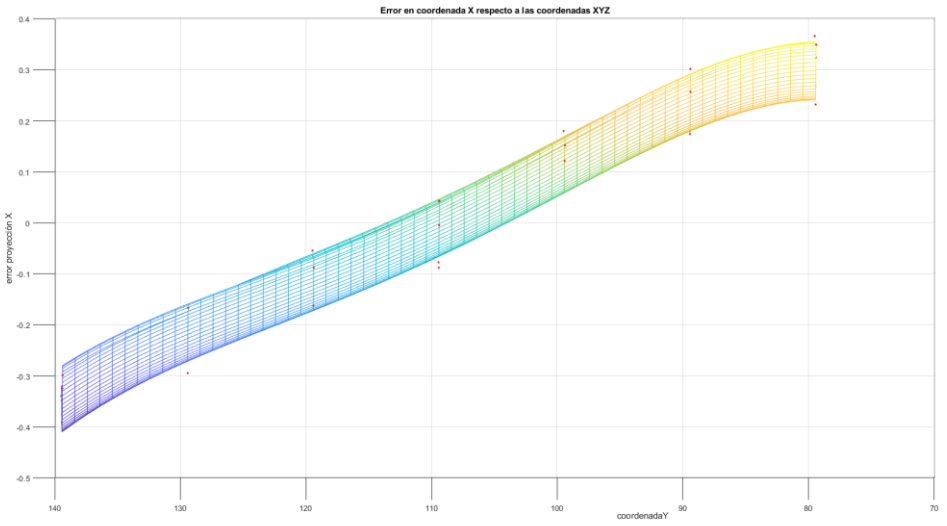


Figura 5.34 Superficie que se adapta al error de proyección en X respecto a coordenada Y

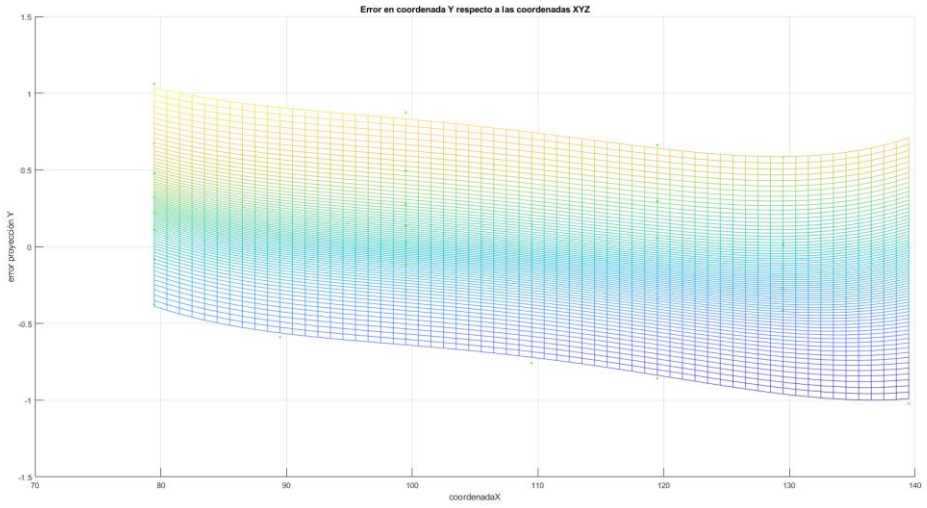


Figura 5.35 Superficie que se adapta al error de proyección en Y respecto a coordenada X

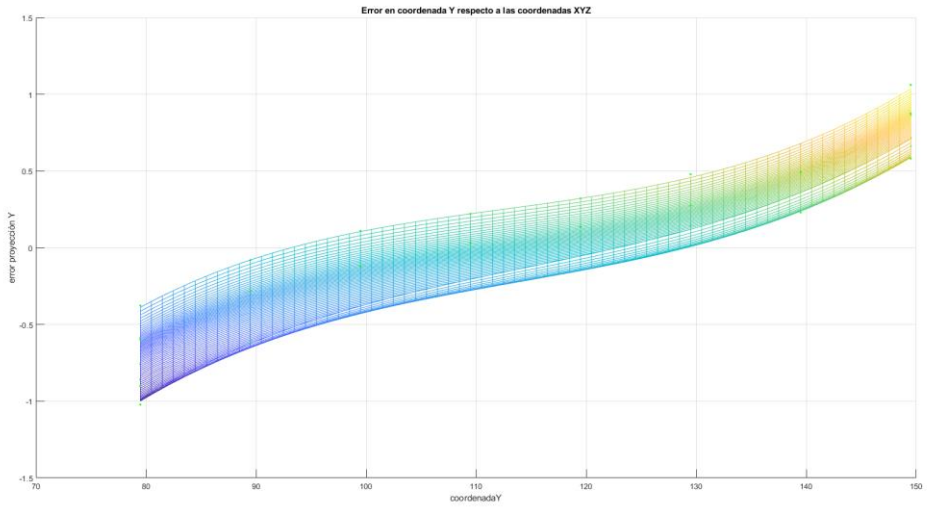


Figura 5.36 Superficie que se adapta al error de proyección en Y respecto a coordenada Y

Con estas superficies se obtiene una corrección con la cual se puede obtener la diferencia entre usar o no estas superficies:

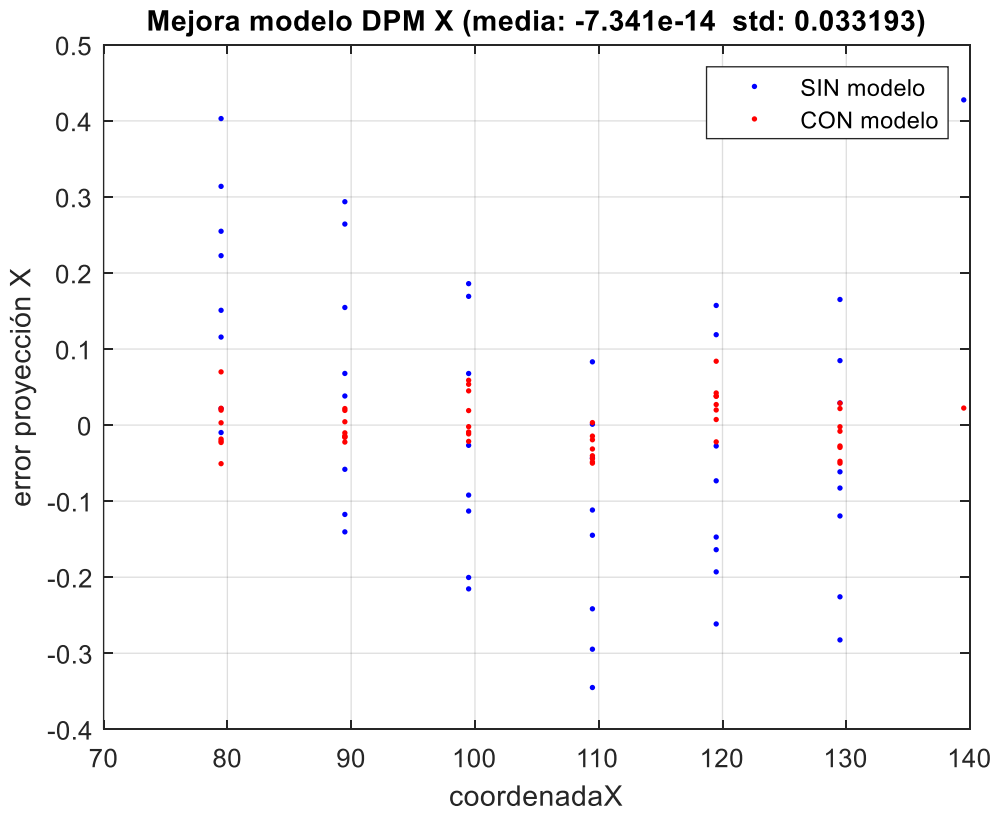


Figura 5.37 Mejora del modelo Coordenada X

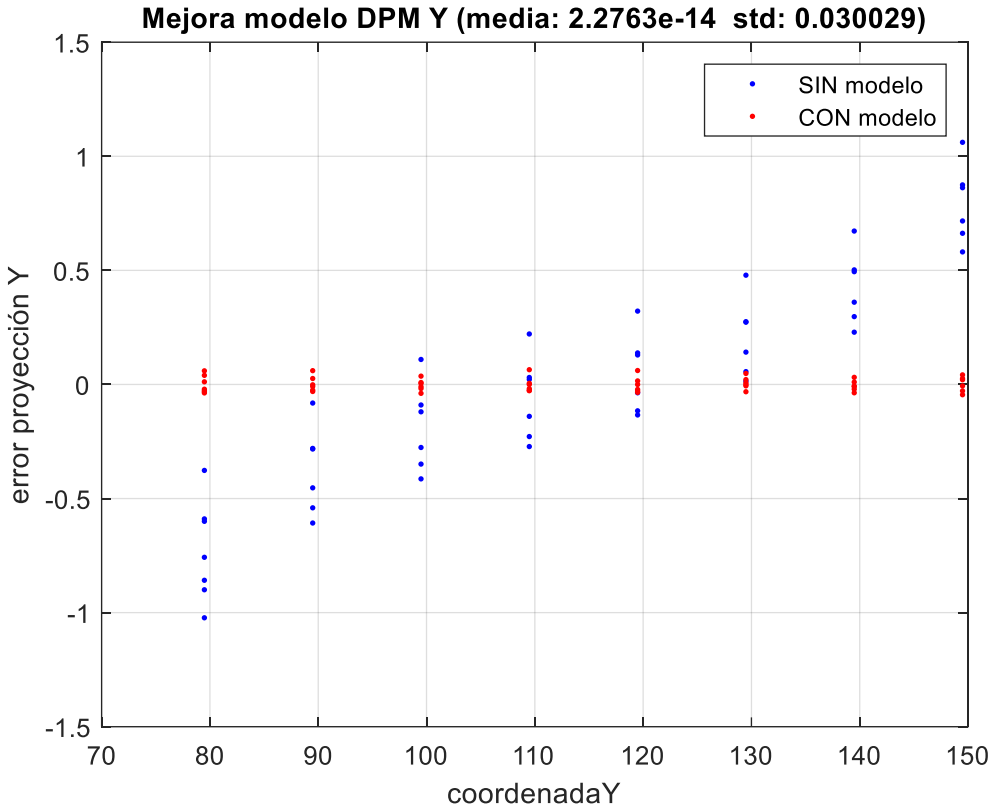


Figura 5.38 Mejora del modelo Coordenada Y

En ambos casos el error se aproxima a 0 y presenta una mejora notoria de los resultados, pero para su verificación se repitió el experimento anterior:

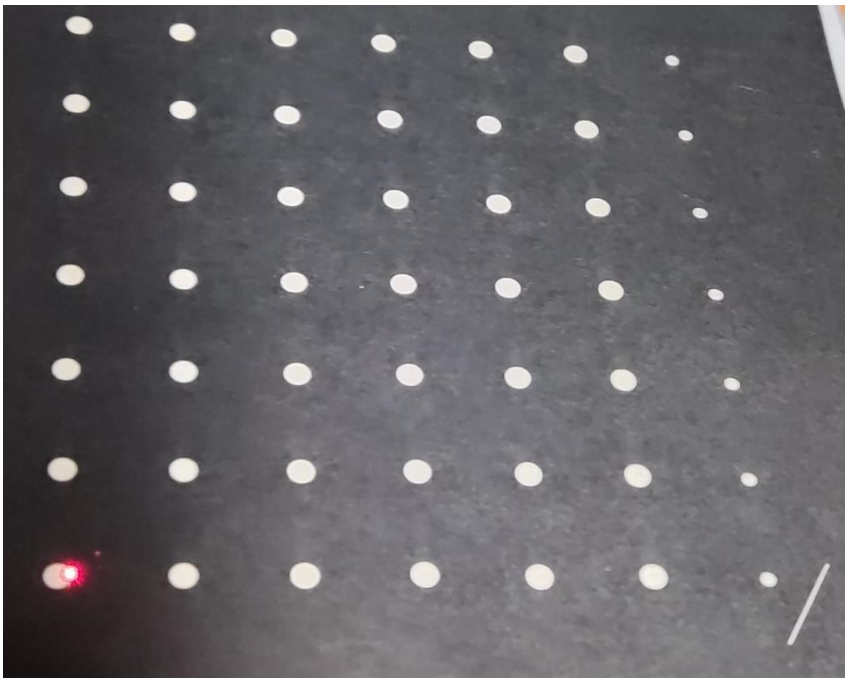


Figura 5.39 Comprobación calibrado completo

Con esta última implementación, se ha desarrollado un método fiable para disparar con el láser en la posición deseada, conocido como DPM.

Los sistemas de coordenadas están dispuestos de la siguiente manera: el sistema L representa el láser y el sistema C la cámara. Para que esta relación coincida con la disposición real, sería necesario girar la imagen 180 grados. Sin embargo, esta representación es suficiente para ilustrar cómo se encuentran ubicados en el espacio.

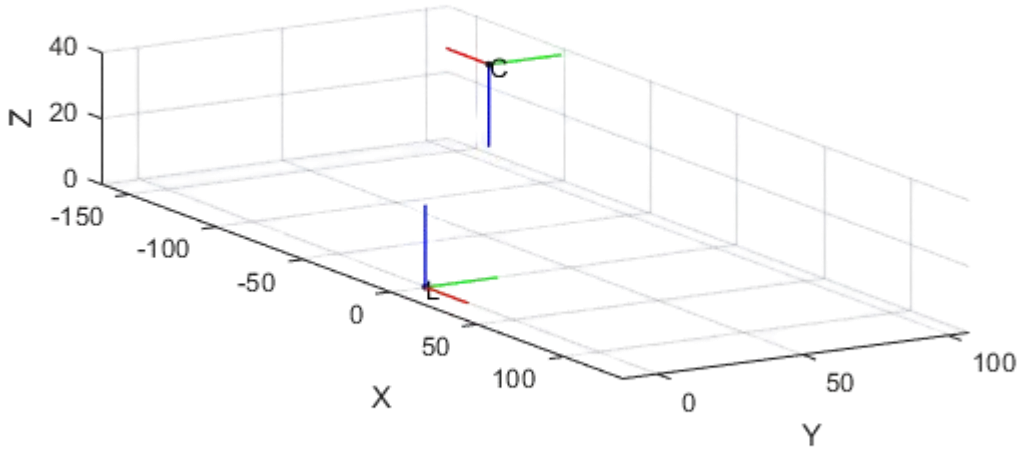


Figura 5.40 Sistemas de coordenadas en el espacio

5.2.2.5 Herramienta CLAM

La herramienta CLAM, desarrollada por el investigador Carlos Ricolfe Viala, es un algoritmo diseñado para identificar regiones planas en una nube de puntos 2D. Gracias a esta herramienta, es posible determinar la zona de decapado en la pieza.

Para utilizar CLAM, el algoritmo requiere como entrada la nube de puntos, un punto inicial para comenzar la búsqueda (lo que reduce la probabilidad de error y el tiempo necesario para encontrar la superficie), un umbral de error, un ancho teórico en las unidades de la nube de puntos, y las reducciones fuera y dentro, que corresponden a ajustes en los puntos izquierdo y derecho. Además, se puede optar por representar la nube de puntos junto con los puntos obtenidos.

El resultado de la herramienta incluye el punto central entre los puntos a decapar, los puntos a decapar y los puntos a decapar sin las reducciones. Los

puntos seleccionados para el decapado se transformarán a las coordenadas del láser, y el punto central de estos se utiliza como entrada para el algoritmo en la siguiente iteración. Una vez establecido un punto inicial, el proceso se automatiza.

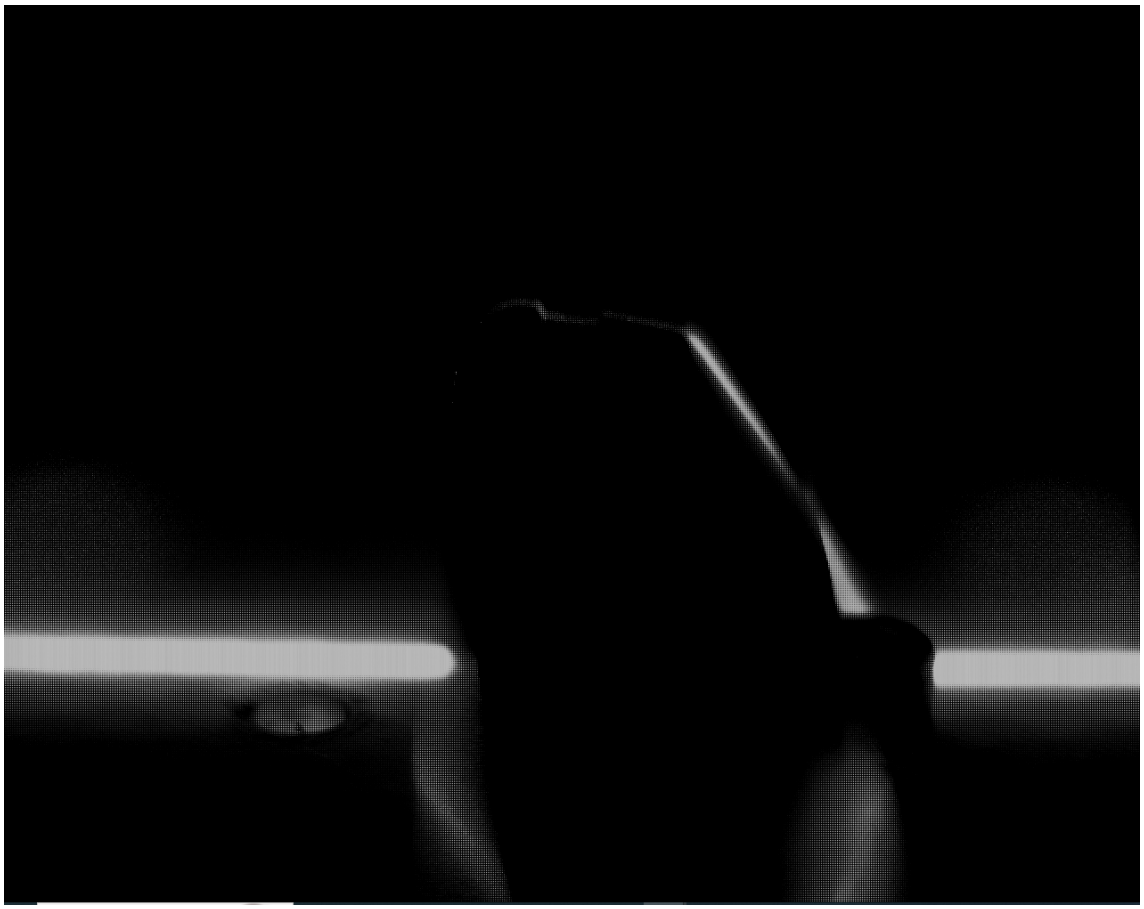


Figura 5.41 Ejemplo de qué ve la cámara en el proceso

Para una imagen como la figura superior se genera una nube de puntos en C++ de la siguiente manera:

Figura 5.42 Nube de Puntos C++ con PCL

Es esta nube de puntos la que se introduce al CLAM:

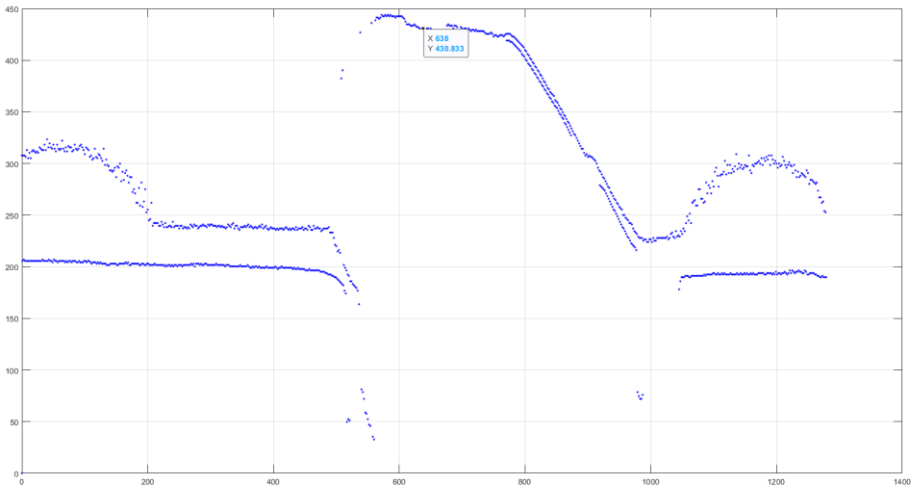


Figura 5.43 Nube de Puntos en Matlab y el punto central de la zona de interés

Como respuesta a estos datos el CLAM devuelve el siguiente resultado:

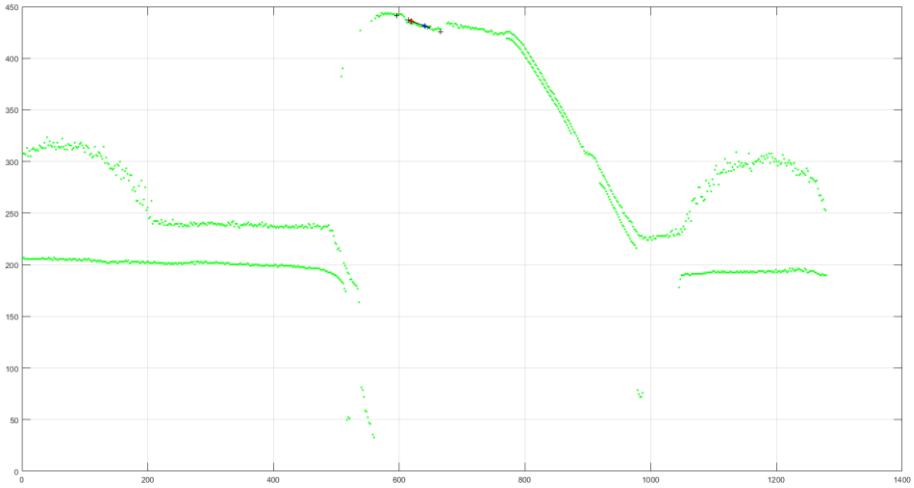


Figura 5.44 Puntos a Decapar en la Nube de Puntos

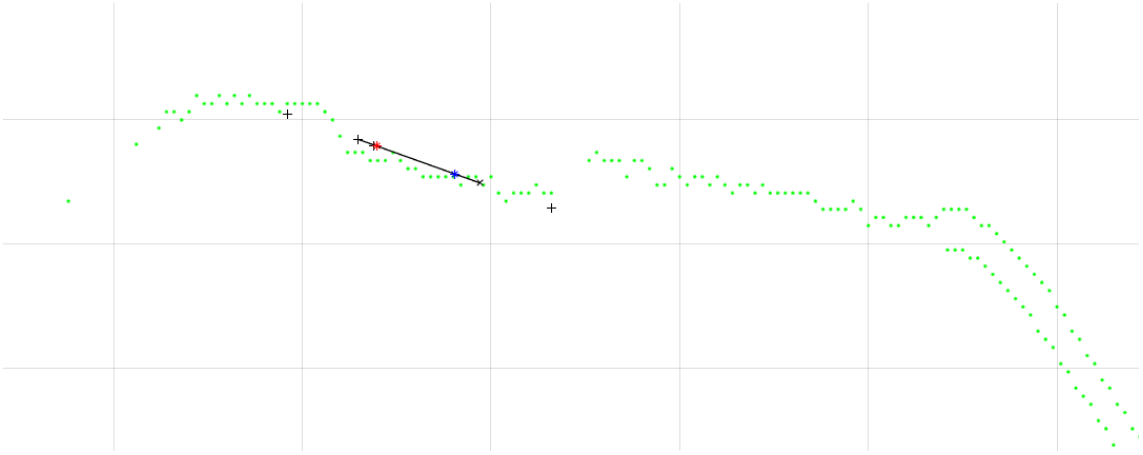


Figura 5.45 Zoom a los Puntos a Decapar

En la imagen se representan los puntos a decapar como los puntos rojo y azul, estos tienen una distancia entre sí de 1.2 mm que es el ancho deseado para el decapado de la pieza.

5.2.2.6 Adaptación Puntos Para Decapar

Esta parte del proceso se utiliza exclusivamente en el decapado por sectores. Los puntos obtenidos para el decapado pueden presentar un ligero error entre ellos, lo que puede resultar en una presentación incorrecta del decapado.

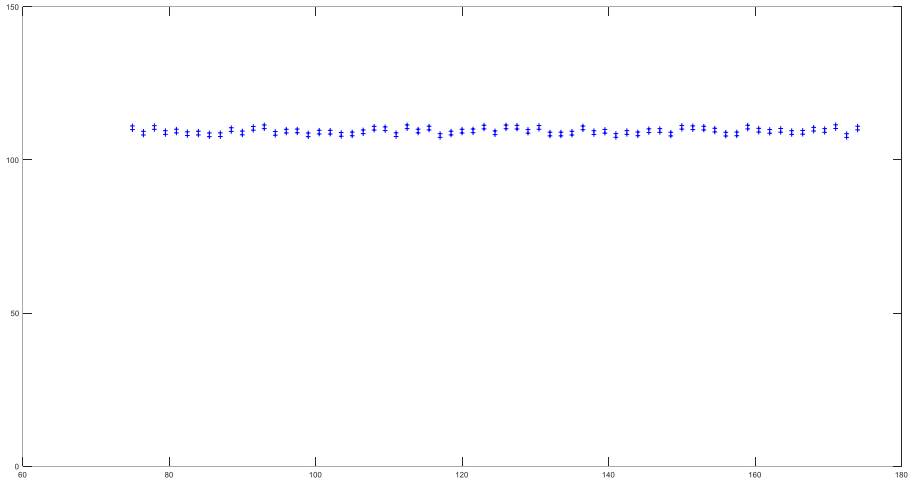


Figura 5.46 Segmento a decapar sin Adaptación

Por lo tanto, antes de construir un sector, es necesario adaptar y corregir estos puntos ajustándolos a un polinomio de grado 2. Esto asegura una uniformidad en el decapado, permitiendo a la vez la presencia de curvas.

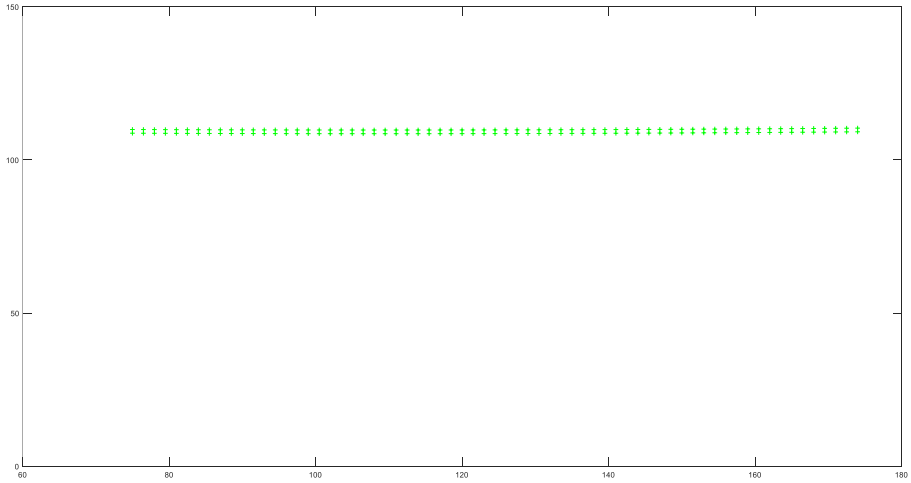


Figura 5.47 Segmento a decapar Adaptado

5.2.2.7 Sector en formato Láser

Para el decapado por sectores, se suministran a una función los puntos adaptados previamente junto con el nombre del archivo, la potencia del láser, la frecuencia del láser, la resolución, la velocidad de pasada del láser, el número de repeticiones y el desenfoque. Todos estos parámetros son fijos para la aplicación específica debido a factores como la pintura a decapar y el ancho de la superficie a tratar.

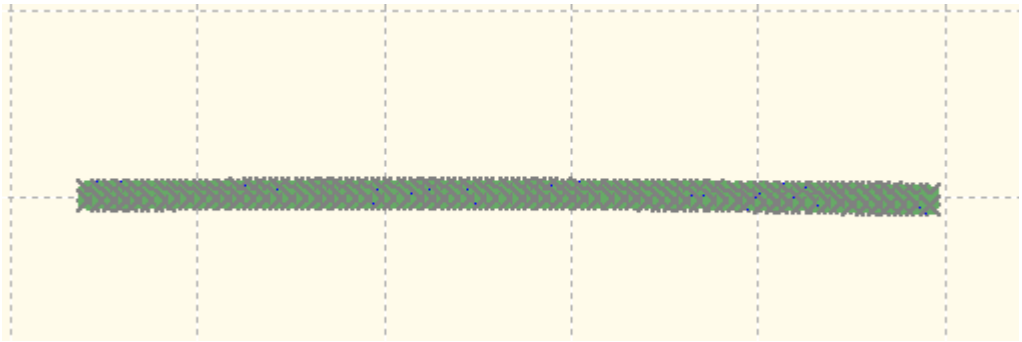


Figura 5.48 Segmento a decapar Adaptado en Archivo Láser

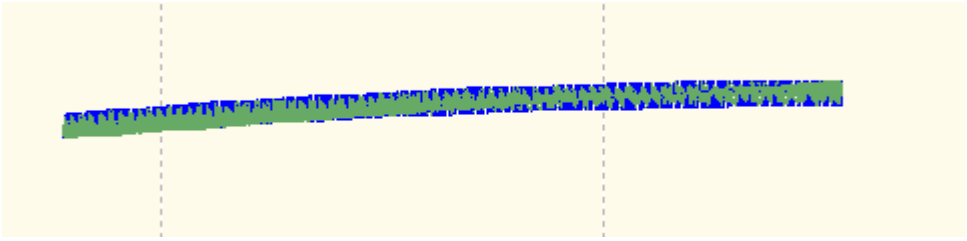


Figura 5.49 Ejemplo de Sector que se envía al láser

Figura 5.50 Ejemplo de unión de sectores continuos

La figura superior muestra el resultado que se apreciaría al decapar de forma continua el resultado de tres sectores consecutivos.

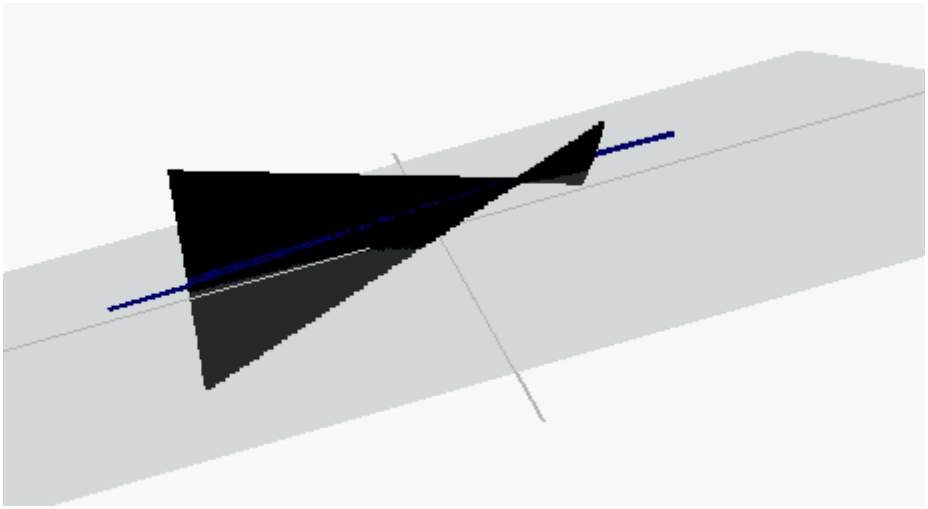


Figura 5.51 Representación 3D de los 3 sectores

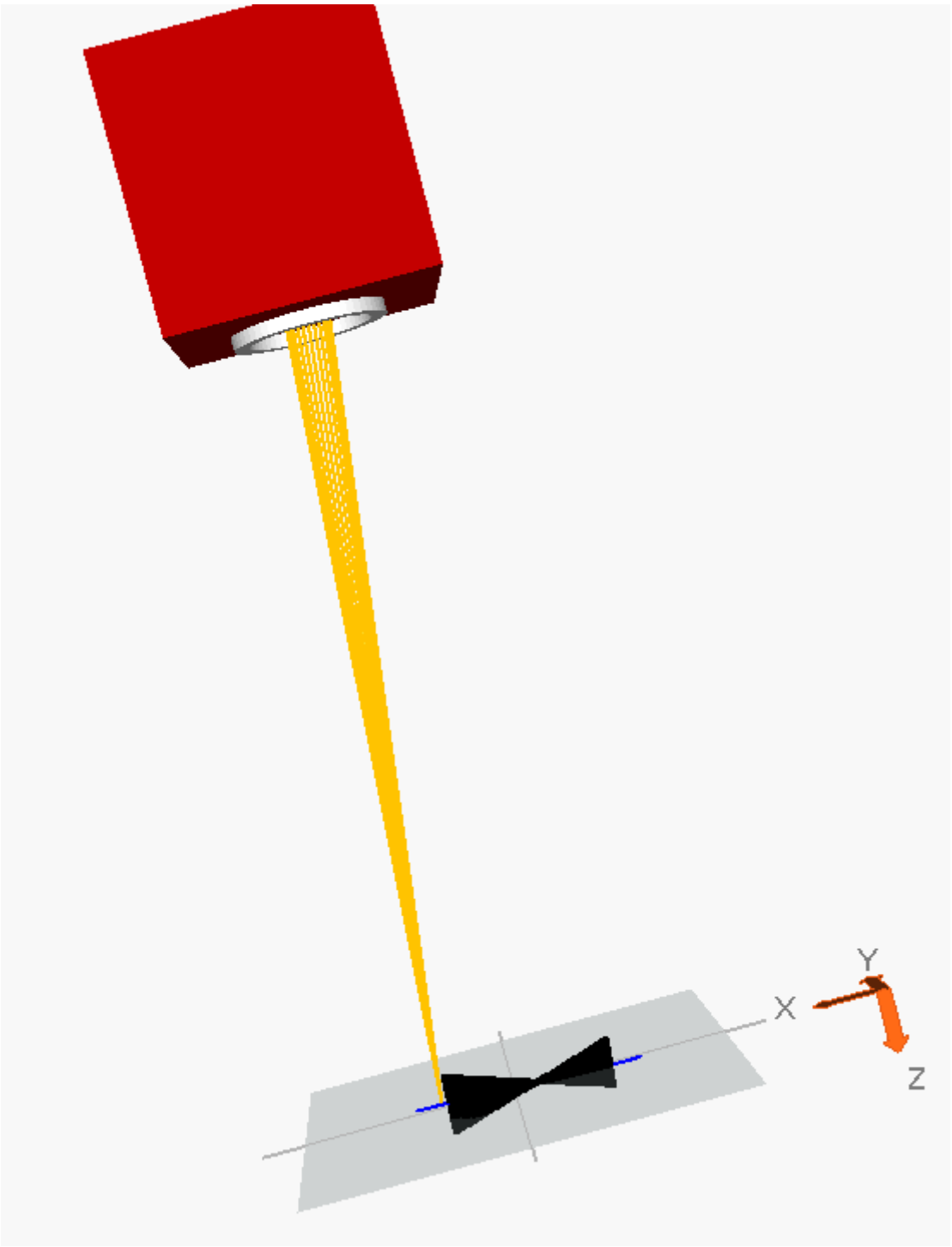


Figura 5.52 Representación de decapado de 3 Sectores en 3D

5.2.3 C++

El programa principal está desarrollado en C++. Se han importado a C++ como librerías las funciones de CLAM, adaptación de puntos y creación de sectores en formato XML, previamente desarrolladas en Matlab. Además, los polinomios obtenidos de las calibraciones realizadas en Matlab se han incorporado como funciones en el código.

Por ejemplo, para obtener las correcciones de proyección en el eje X:

```
double DPM_X(double x, double y) {

    double p00 = 64.66;
    double p10 = -3.136;
    double p01 = 0.5291;
    double p20 = 0.05009;
    double p11 = -0.003873;
    double p02 = -0.00576;
    double p30 = -0.0003419;
    double p21 = 1.936e-05;
    double p12 = 1.845e-05;
    double p03 = 2.961e-05;
    double p40 = 8.544e-07;
    double p31 = -4.213e-08;
    double p22 = -3.16e-08;
    double p13 = -3.752e-08;
    double p04 = -5.861e-08;

    double CorreccionX = p00 + p10 * x + p01 * y + p20 * x * x + p11 * x * y +
p02 * y * y + p30 * x * x * x + p21 * x * x * y + p12 * x * y * y + p03 * y * y * y +
p40 * x * x * x * x + p31 * x * x * x * y + p22 * x * x * y * y + p13 * x * y * y * y +
p04 * y * y * y * y;
    return CorreccionX;
}
```

Con el resto de los elementos de la calibración se ha seguido el mismo proceso. En ambas aplicaciones, tanto para el decapado por líneas como por sectores, se inicializan las variables y librerías necesarias para el funcionamiento de la aplicación.

Entre estas variables se incluyen el número de imágenes a capturar, el pulso del encoder correspondiente al momento en que se desea iniciar el decapado, y el punto central donde CLAM debe comenzar la búsqueda. El código

completo se encuentra en el apartado “Código” del documento. Comunicación con el Láser

En ambas aplicaciones es necesario comunicarse con el láser, para ello se emplea una librería proporcionada por MACSA. Primeramente, es necesario iniciar el láser:

```
MInit(sock, name, ip, path);

p = MStartClient(sock);
  if (p != 0) {
      printf("Error al coenctarse al laser");
      return 1;
      }

p = MLaser_StartPrintSession(sock, 1);
```

Para iniciar el láser, se necesita su dirección IP. El resto de las variables deben ser declaradas por el usuario, y el valor de “sock” se modifica en la función “MInit” para permitir la conexión en “MStartClient”. Si la conexión se establece con éxito, se envía una señal al láser indicando el inicio de una sesión de decapado, lo que permite al láser comenzar a funcionar cuando se le envíen archivos y señales de decapado.

Para la aplicación de decapado por líneas, se envía un archivo al láser con cada pulso del encoder que supere el pulso calibrado. En la aplicación de decapado por sectores, se envía un archivo al láser cuando el pulso del “encoder” sea igual o superior al pulso calibrado y hayan transcurrido 420 pulsos, equivalentes a 40 mm. Esto significa que cada sector tendrá una longitud de 40 mm, aunque este valor es completamente modificable.

Para el decapado por líneas:

```
if (cur_pic_nr >= PulsoCalibracion) {
//Si se han alcanzado el número de fotos necesario se envía un archivo al LAM

 ArchivoEnviar  =  "C:\\Users\\usuario\\Desktop\\PracticasJavi\\XMLLaser\\" +
std::to_string(ArchivoEnviado) + ".xml";
resultado = documentoLaser.load_file(ArchivoEnviar.c_str());
          if (!resultado) {
                cout << "No se ha podido cargar el archivo para el laser";
                return -1;
          }
```

```
        else {
              documentoLaser.save_file(FicheroLaserSTR.c_str());
              p = MLaser_CopyFile(sock, FicheroLaser, FilePath2, option);
                p = MLaser_Reload(sock);
                MLaser_Start(sock, FicheroLaser, 1);
                ArchivoEnviado++;
                }
            }
```

Primero se carga el archivo a enviar, el cuál debe empezar siendo el 0 y aumentando en número con la con la función de load file sobre la variable "documentoLaser".

Si se ha cargado el archivo correctamente se sobre escribe el fichero "FicheroLaserSTR" con la información del fichero cargado y se envía al láser, al tener siempre el mismo nombre, estaremos sobre escribiendo el archivo en el láser, esto se hace para no sobrecargar la memoria del láser.

Cuando el fichero haya sido sobre escrito se le manda una señal de recarga al láser para que lea la información nueva con "MLaser_Reload" y se le envía la señal de que decape el fichero enviado con MLaser_Start.

Para el decapado por Sectores sólo cambia la condición inicial:

```
if (cur_pic_nr >= PulsoCalibracion && (cur_pic_nr - PulsoCalibracion) % 420 == 0)
```

5.2.3.1 Adquisición de nube de puntos

En ambas aplicaciones la adquisición es la misma, se accede a la imagen y mediante un threshold se recorren las columnas de la imagen y se almacena el primer punto encontrado con una intensidad superior al threshold:

```
for (k = 0; k < width; k = k + 1)
{
  for (j = 0; j < height; j = j + 1)
   {
     if (punteroImagen[k + j * width] >= tvalue)
        {

          //Para calibrar
```

```
            /*cloud->points[k + currentPicture * width].x = double(coordX) + 1;
              cloud->points[k + currentPicture * width].y = double(k);
              cloud->points[k + currentPicture * width].z = double(j);*/
            //cloud->points[k + currentPicture * width].z = ((double(pixelReferencia0) - double(j)) * 33.0 / 504.0);

            //Si solo se quiere crear una nube por imagen
              cloud->points[k + width].x = PulsoCalibracion;
              cloud->points[k + width].y = double(k);
              cloud->points[k + width].z = ((double(pixelReferencia0) - double(j)) * 33.0 / 504.0) * 10;

              NubeDePuntos[k].x=PulsoCalibracion;// X
              NubeDePuntos[k].y=double(k);   // Y
              NubeDePuntos[k].z=(((double(pixelReferencia0) - double(j)) * 33.0 / 504.0) * 10);   // Z
               j = height; //Para pasar a la siguiente columna
        }
    }
}
```

Si se desea calibrar y adquirir una nube de puntos completa como es el caso para calibrar la matriz de transformación entre ambos sistemas de coordenadas la coordenada X necesita ir aumentando en 1 cada vez que se produce un pulso.

Si tan solo se desea adquirir la nube de puntos de la pieza para la aplicación no es necesario. Se ha empleado cloud->points para obtener la pcd con la librería Point Cloud Library y poder exportarla, pero para la aplicación final esto no es necesario y con la variable “NubeDePuntos” es suficiente.

5.2.3.2 Obtención Puntos para Decapar

De nuevo el procedimiento es el mismo para ambas aplicaciones, se adaptan los puntos obtenidos de la nube en el paso previo para poder pasarlos al CLAM y este devuelve como resultado los puntos deseados:

```
//Se adaptan solo las coordenadas Z e Y
for (size_t i = 0; i < NubeDePuntos.size(); ++i) {
                    NubeDePuntos1fila[i * 2 + 0] = NubeDePuntos[i].y;
                    NubeDePuntos1fila[i * 2 + 1] = NubeDePuntos[i].z;
                }
```

```
//Se asignan los puntos de la nube de puntos a la variable de matlab
try {
// Utilizar SetData para copiar los datos a la matriz mwArray
puntosLAM.SetData(NubeDePuntos1fila.data(), NubeDePuntos1fila.size());
}
catch (const mwException& e) {
	std::cerr << "mwException: " << e.what() << std::endl;
	return -1;
}

catch (...) {
	std::cerr << "Error desconocido al usar SetData." << std::endl;
	return -1;
}

LAM_DC(nargout, puntosCorregidos, puntosDecapar, centroBordesActual, puntosLAM, centroBordes, umbralError, anchoTeorico, reduccionFuera, reduccionDentro, debugCorreccion);

//Se le asigna el valor a centroBordes de centroBordesActual para la proxima iteración
centroBordes(1, 1) = centroBordesActual(1, 1);
centroBordes(1, 2) = centroBordesActual(2, 1);
```

Como se puede observar además se deja preparada la variable para que el CLAM empiece a buscar a la siguiente iteración. A continuación, se transforman los puntos a coordenadas del láser:

```
puntosDecapar.GetData(VectorAuxiliar, 8);

//primer punto
PixelColumna = pixelReferencia0 - VectorAuxiliar[1] * 504 / 330;

VectorCamara[0] = -PulsoCalibracion * R_pulso_mm; para el decapado por líneas

VectorCamara[0] = float(PulsoTramoFinal + pointCount) * R_pulso_mm; para el decapado continuo

VectorCamara[1] = ObtenerY(VectorAuxiliar[0], PixelColumna);
```

```
VectorCamara[2] = ObtenerZ(PixelColumna);
        for (int l = 0; l < 4; ++l) {
             for (int d = 0; d < 4; ++d) {
                  VectorLaser1(l) += MT[l][d] * VectorCamara[d];
                  }
        }

VectorLaser1[0]=VectorLaser1[0]+ DPM_X(VectorLaser1[0], VectorLaser1[1]);

VectorLaser1[1]=VectorLaser1[1]+ DPM_Y(VectorLaser1[0], VectorLaser1[1]);

//segundo punto
PixelColumna = pixelReferencia0 - VectorAuxiliar[5] * 504 / 330;

VectorCamara[1]= ObtenerY(VectorAuxiliar[4], PixelColumna);
VectorCamara[2]= ObtenerZ(PixelColumna);
          for (int l = 0; l < 4; ++l) {
               for (int d = 0; d < 4; ++d) {
                    VectorLaser2(l) +=MT[l][d] * VectorCamara[d];
                   }
          }

VectorLaser2[0]=VectorLaser2[0]+ DPM_X(VectorLaser2[0], VectorLaser2[1]);
VectorLaser2[1]=VectorLaser2[1]+ DPM_Y(VectorLaser2[0], VectorLaser2[1]);
```

Primero, los puntos a decapar se cargan en una variable vector auxiliar. Luego, se obtienen las coordenadas en píxeles de cada punto para aplicar las funciones de calibración en las coordenadas Z e Y. Para la coordenada X, en el caso del decapado por líneas, se dispara siempre en la misma coordenada. En el decapado continuo, se crea el sector comenzando en la coordenada de 120 mm y se resta hasta 80 mm en cada pulso del encoder.

Con las coordenadas de los puntos en milímetros, se utiliza la matriz de transformación para obtener las coordenadas del láser. Finalmente, se aplica la corrección en X e Y utilizando el calibrado DPM.

5.2.3.3 Creación de Documento para el láser

En este apartado sí que existen diferencias más notorias para cada caso de decapado.

Comenzando con el decapado por líneas:

```
// Obtener el nodo de la línea (<line>)
    xml_node lineNode = doc.select_node("//line").node();

// Obtener el nodo de la línea (<layer>)
     xml_node layerNode = doc.select_node("//layer").node();

// Modificar los valores de sy, ey y zdefocus con variables personalizadas

     newSy = (VectorLaser1[1]) /KSven;    // Nuevo valor de sy
     newEy =( VectorLaser2[1]) /KSven;    // Nuevo valor de ey
     int newZDefocus = ((VectorLaser1[2] + VectorLaser2[2]) / 2);  // Nuevo valor
de zdefocus

// Actualizar los valores en el archivo XML

     while (lineNode && contadornodos < 2) {
             lineNode.attribute("sy").set_value(newSy);
             lineNode.attribute("ey").set_value(newEy);
             lineNode = lineNode.next_sibling("line"); // Mover al siguiente nodo
"line"
                 contadornodos++;
               }

     contadornodos = 0;

     if (!layerNode.attribute("zdefocus")) {
                 std::cerr << "El atributo zdefocus no está presente en el nodo
line." << std::endl;;
               }

      layerNode.attribute("zdefocus").set_value(newZDefocus);

      newFilename = "C:\\Users\\usuario\\Desktop\\PracticasJavi\\XMLLaser\\" +
std::to_string(ArchivoGuardado) + ".xml"; // Nombre del nuevo archivo XML

       if (!doc.save_file(newFilename.c_str())) {
          cerr << "Error al guardar el archivo XML modificado." << endl;
          return 1;
        }

         else {
          ArchivoGuardado = ArchivoGuardado + 1;
```

```
            }

        VectorLaser1[0] = 0;
        VectorLaser1[1] = 0;
        VectorLaser1[2] = 0;
        VectorLaser2[0] = 0;
        VectorLaser2[1] = 0;
        VectorLaser2[2] = 0;
```

Para poder comprender y explicar correctamente el código es necesario ver el archivo que se usa como plantilla para la creación de otros archivos correspondientes a cada línea:

```
<?xml version="1.0"?>
<laserfile version="0x2">
        <layers>

<layer name="Nueva capa0" id="0" mask="0x0" printable="1" editable="1" visible="1" power="50" speed="1500000" resolution="200" frequency="50" zpos="0" zdefocus="-8401" color="0xff0000" delay="0" repeat="0" signalmask="0x0" signalstatus="0x0" scannerset="0" pulselen="0" pixmap="0" />
        </layers>
        <objects>
                <line sx="50000" sy="43954" ex="50000" ey="44389" id="0">
                        <generic layer_id="0" printable="1" editable="1" linewidth="0" wobblefreq="0" wobbleratio="1" />
                </line>
                <line sx="50009" sy="43954" ex="50009" ey="44389" id="1">
                        <generic layer_id="0" printable="1" editable="1" linewidth="0" wobblefreq="0" wobbleratio="1" />
                </line>
        </objects>
</laserfile>
```

Este archivo describe para el láser dos líneas verticales, especificadas por los puntos iniciales en X e Y con “line sx” y “line sy” respectivamente, y los puntos finales con “line ex” y “line ey”. Además, contiene las características para el decapado de estas dos líneas en 2D, como la potencia del láser, la resolución, la frecuencia, entre otros parámetros. El más importante de estos parámetros es Zdefocus, que permite decapar la línea 2D a diferentes alturas, proporcionando así una capacidad limitada de decapado en 3D.

En el código presentado se emplea la librería pugi. Con ella, se declara la intención de buscar en el archivo de plantilla los nodos “line”, de forma que, por ejemplo, “line sx” sería el nodo line con el atributo sx. Una vez encontrados estos atributos, se sustituyen por las coordenadas obtenidas en el apartado anterior, transformadas a coordenadas Sven, que son las utilizadas internamente por el láser.

Una vez definidas las coordenadas Y de cada punto, se busca en el archivo el nodo “layer”, que corresponde a las características del decapado, y se actualiza el atributo Zdefocus con la altura correspondiente.

Una vez realizados todos los cambios en el archivo, se guarda con un nombre nuevo que corresponde al número del pulso en el que se ha realizado la creación.

Para el decapado continuo el proceso se presenta de la siguiente forma:

```
// Agregar el vector como una nueva fila a la matriz
   MatrizPuntos->col(columncount) = VectorLaser1;
   try {
        MatrizPuntos->col(columncount + 1) = VectorLaser2;
        }

   catch (const std::out_of_range& e) {
        std::cout << "Se ha salido del tamaño de la matriz: " << e.what() << std::endl;
        }

   catch (const std::exception& e) {
         std::cout << "Se ha producido una excepción: " << e.what() << std::endl;
          }

   catch (...) {
         std::cout << "Se ha producido una excepción desconocida." << std::endl;
          }

   respuesta = CrearPolinomial(MatrizPuntos, &ArchivoGuardado, &pointCount, &potencia, &frecuencia, &resolucion, &velocidad, &repeticiones, &desenfoque, &gridSize, &debug);
```

A cada pulso que se recibe del encoder al final del proceso se añaden ambos puntos obtenidos a una matriz creada mediante la librería “armadillo” y esta se pasa como input a la función “CrearPolinomial”, la cual está definida de la siguiente forma:

```
int CrearPolinomial(mat* Matriz,int* numeroArchivo, int* ContadorPuntos, mwArray* potencia, mwArray* frecuencia, mwArray* resolucion, mwArray* velocidad, mwArray* repeticiones, mwArray* desenfoque, mwArray* gridSize, mwArray* debug) {

   if(*ContadorPuntos==420-1){//Porque empiezo desde 0, entonces de 0 a 419 son los 420 pulsos

      // Guardamos la penúltima y última columna ya que serán las coordenadas Y, y Z de los primeros puntos del siguiente disparo.

       arma::vec penultimaColumna = Matriz->col(Matriz->n_cols - 2);
       arma::vec ultimaColumna = Matriz->col(Matriz->n_cols - 1);

       // Guardamos la primera y segunda columna ya que serán la coordenada X de los primeros puntos del siguiente disparo.

       arma::vec primeraColumna = Matriz->col(0);
       arma::vec segundaColumna = Matriz->col(1);

       // Cambiamos la primera coordenada de la penúltima y última columna con la de la primera y segunda respectivamente para quedarnos con la X de interés

       penultimaColumna(0) = primeraColumna(0);
       ultimaColumna(0) = segundaColumna(0);

       mwArray matlabArray(Matriz->n_rows, Matriz->n_cols, mxDOUBLE_CLASS);
       mwArray matlabArrayAdaptada(Matriz->n_rows, Matriz->n_cols, mxDOUBLE_CLASS);
       mwArray paso(3.0);

       // Llamar a la función de Matlab para copiar datos
       matlabArray.SetData(Matriz->memptr(), Matriz->n_elem);

       // Cambio en el nombre del archivo

       std::string nuevoNombreArchivo = "C:\\Users\\usuario\\Desktop\\PracticasJavi\\XMLLaser\\" + std::to_string(*numeroArchivo);
       mwArray nombreArchivoMw(nuevoNombreArchivo.c_str());
```

```
        *numeroArchivo = *numeroArchivo + 1;

        try {
            AdaptarPuntosLAM(1,matlabArrayAdaptada,matlabArray,paso);
        }

        catch (const mwException& e) {
            std::cerr << "Error al llamar a la función de Matlab: " << e.what() <<
std::endl;
            return -1; // Salir de la función si ocurre un error
        }

        try {
            CrearXML3Ddef(matlabArrayAdaptada, nombreArchivoMw, *potencia,
*frecuencia, *resolucion, *velocidad, *repeticiones, *desenfoque, *gridSize,
*debug);
        }
        catch (const mwException& e) {
            std::cerr << "Error al llamar a la función de Matlab: " << e.what() <<
std::endl;
            return -1; // Salir de la función si ocurre un error
        }

        //Se reinicia la matriz
        Matriz->zeros();

        // Se asignan las columnas guardadas a la primera y segunda columna
        Matriz->col(0) = penultimaColumna;
        Matriz->col(1) = ultimaColumna;

        *ContadorPuntos = 1;
        return 0; // Devuelve 0 para indicar éxito
    }
    else {
        *ContadorPuntos = *ContadorPuntos + 1;
        return 0;
    }

}
```

En la función, primero se verifica si han transcurrido 420 pulsos, que corresponden a la evaluación de un sector. Si no se han alcanzado estos pulsos, se incrementa el contador de pulsos.

Una vez se alcanzan los 420 pulsos, se almacenan las coordenadas Z e Y de los últimos dos puntos que componen el sector. En un decapado continuo, el último punto de un sector debe ser el primer punto del siguiente. Del mismo modo, dado que se volverán a contar 420 pulsos, la coordenada X de los dos primeros puntos del siguiente sector será la misma que la del sector actual.

A continuación, se crea un array en Matlab para volcar los datos de la matriz de entrada. Estos datos se pasan por la función de adaptación de puntos de la calibración, generando un polinomio que se almacena en otra matriz.

Con la matriz adaptada, se crea el archivo XML utilizando la función correspondiente. Después, se vacía la matriz y se reintegran las coordenadas almacenadas al principio. Finalmente, se ajusta la variable que cuenta los pulsos restantes a 420, dado que al agregar el primer punto del siguiente sector se cuenta como un pulso (para evitar desbordamientos en la matriz).

6 Resultados, conclusiones y mejoras del sistema

6.1 Resultados

Se han llevado a cabo múltiples pruebas de decapado en la pieza de interés, y se han obtenido los siguientes resultados:

1. **Identificación del Sector**: La identificación del sector de la pieza ha demostrado ser muy fiable, teniendo en cuenta las características del

prototipo, tales como la iluminación, la intensidad del láser y la lente empleada.

2. **Comunicación con el Láser**: La comunicación con el láser ha sido efectiva, cumpliendo con las expectativas de disparar en el lugar y momento deseado.

3. **Decapado en 2D**: El sistema en 2D ha sido testeado con resultados correctos y positivos. Esta prueba se realizó limitando la pieza a una distancia específica en la coordenada Z. Aunque esta prueba ha confirmado la calibración en X e Y, el proyecto requiere un funcionamiento en 3D, por lo que estos resultados sólo sirvieron como referencia preliminar.

4. **Decapado Continuo**: El decapado continuo ha cumplido con el objetivo de presentar un proceso ininterrumpido. No se aprecian discontinuidades entre sectores, lo que sugiere un disparo continuo del láser sin interrupciones visibles.

Problemas Encontrados y Causas Posibles

A pesar de los resultados positivos, se han identificado problemas con la transformación a coordenadas del láser, que han mostrado variabilidad en los resultados. Las posibles causas de estos problemas incluyen:

1. **Inclinación de la Cámara**: La inclinación de la cámara con respecto al plano de trabajo ha llevado a la necesidad de una calibración adicional. La altura de la pieza afecta el número de píxeles que ocupa, lo que puede introducir errores en la medición. A pesar de las calibraciones realizadas, este fenómeno podría seguir afectando los resultados.

2. **Desviación del Encoder**: El encoder, montado sobre una pieza 3D conectada a una barra de metal con tornillos y tuercas, ha presentado un ligero "valgo". Este problema se manifestó en la diferencia de distancia de movimiento de izquierda a derecha en el robot, lo cual es crucial para la calibración del movimiento. Este error se está corrigiendo mediante el método DPM, como se explicó en la calibración del eje X.

   Este error aparece por el uso del robot antropomórfico usado como prototipo, con el uso de la cinta transportadora en la aplicación final se eliminará.

3. **Calibración 3D**: La calibración 3D presentada en el proyecto se basa en un método que asume ciertas limitaciones. La relación entre la distancia

de decapado y el origen del láser no ha sido completamente considerada, lo que podría restringir la precisión de la calibración en 3D.

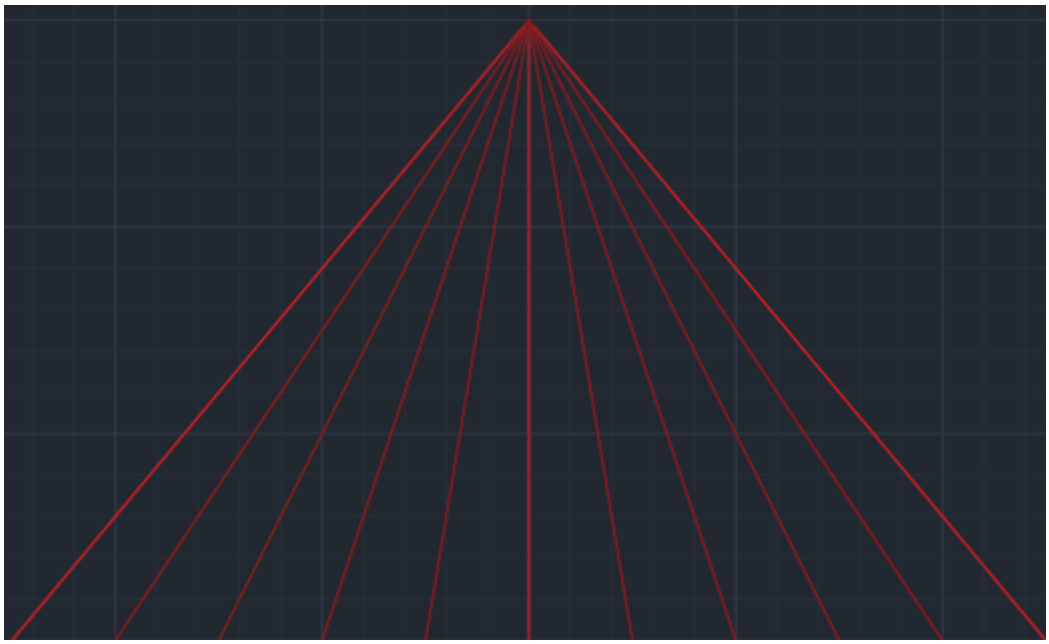


Figura 6.1 Representación del disparo del láser

Se ha observado que el haz láser presenta coordenadas distintas según la altura, lo que indica que el comportamiento de proyección del láser varía con la altura de la pieza. Se intentó realizar una calibración para modelar este comportamiento de proyección, sin embargo, los resultados obtenidos no mejoraron significativamente.

Este problema se abordará con mayor detalle en el apartado de **Mejoras**, donde se discutirá la metodología empleada y las posibles soluciones para optimizar la calibración y mejorar la precisión del sistema.

A continuación, se presentan algunos resultados de decapado sobre la pieza:

Figura 6.2 Decapado por líneas resultado 1

Figura 6.3 Decapado por líneas resultado 2

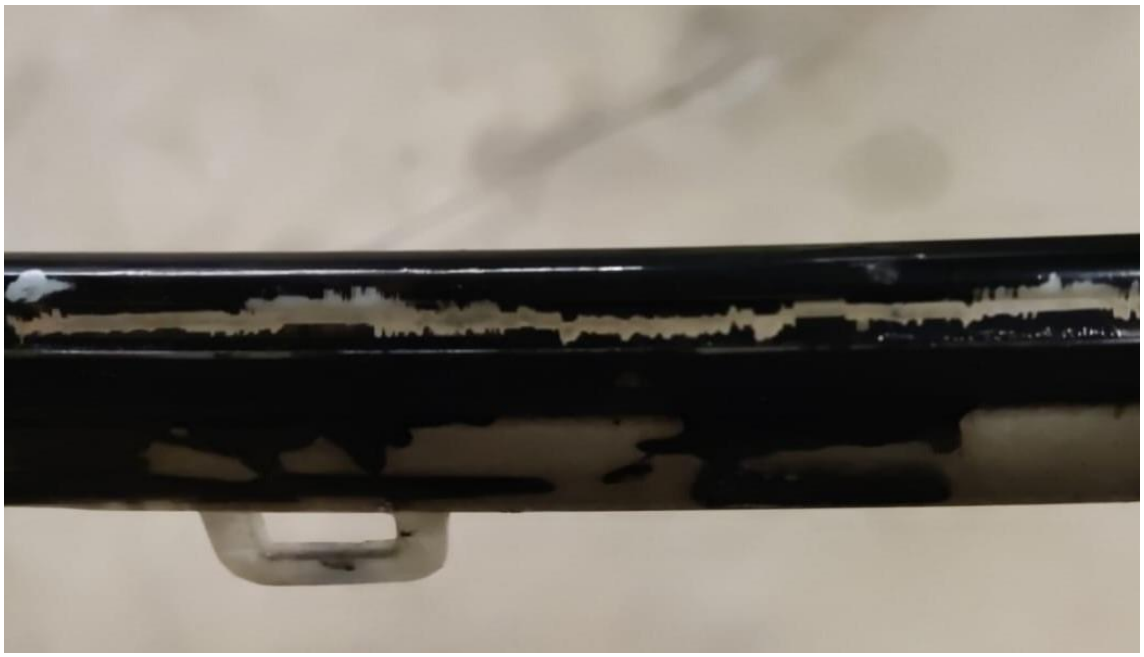


Figura 6.4 Decapado Contínuo sin adaptación

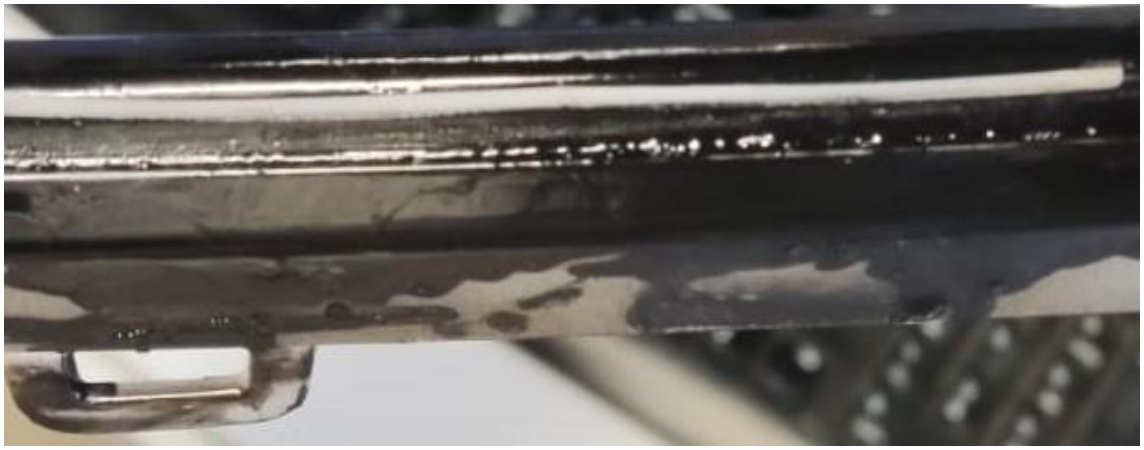


Figura 6.5 Decapado Contínuo Adaptado sin corrección DPM

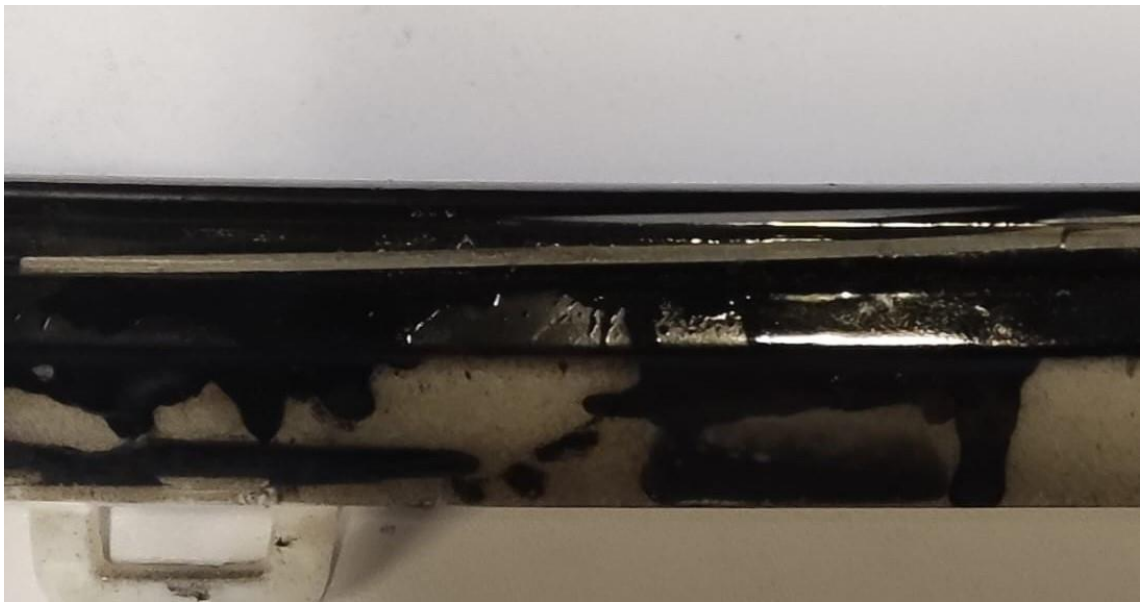


Figura 6.6 Decapado Continuo resultado 1

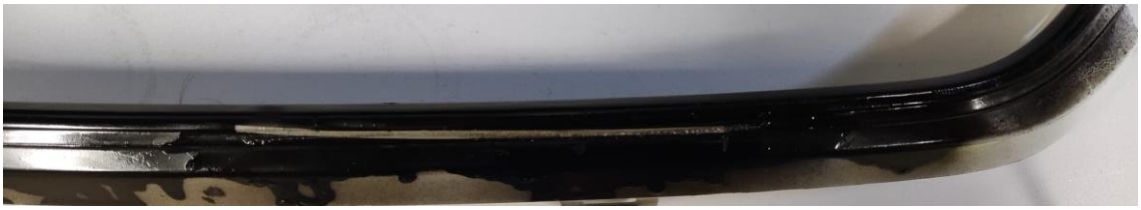


Figura 6.7 Decapado Contínuo resultado final

6.2 Conclusiones

A partir de los diversos resultados obtenidos, se han llegado a las siguientes conclusiones:

- **Decapado por líneas:** Este método no ofrece continuidad entre los disparos, lo que resulta en una falta de cohesión en el decapado. Aunque se podrían explorar mejoras en la velocidad de disparo con diferentes prototipos para lograr un resultado continuo, se ha decidido no continuar con esta técnica debido a sus limitaciones actuales.
- **Decapado continuo:** Este método presenta características prometedoras, como una correcta coordinación entre sectores y uniones no perceptibles. Sin embargo, el disparo del láser no se alinea adecuadamente con el surco y tiende a desviarse, lo que hace que, en su forma actual, el método tampoco sea factible.
- **Investigación futura:** Dado que el decapado continuo muestra aspectos positivos, se continuará investigando un método mejorado de reconstrucción 3D para optimizar la correspondencia entre la imagen

capturada por la cámara y las coordenadas de disparo del láser. Esta investigación tiene el objetivo de mejorar la precisión y lograr un resultado final adecuado.

Estado Actual del Proyecto: El proyecto, en su estado actual, no cumple con los estándares requeridos para su presentación a la empresa solicitante. No obstante, con una mejora que garantice un decapado preciso en el surco captado por la cámara, el proyecto estará en condiciones de ser presentado.

6.3 Mejoras del sistema

Con base en lo mencionado anteriormente, se propone la siguiente línea de investigación para asegurar el resultado deseado.

1.**Implementación del Movimiento Automático de la cinta**:

- Se integrará una funcionalidad en el software para controlar automáticamente el movimiento de la cinta transportadora. Esto permitirá que la cinta se mueva y detenga sin intervención manual, facilitando una operación más fluida y precisa.

2.**Nuevo Método de Calibración**:

- **Primera Opción: Modelo de Proyección del Láser**:
    - Se utilizarán plantillas de calibración 2D a diferentes alturas para ajustar el sistema de acuerdo con el modelo de proyección del láser. Este método, desarrollado y en proceso de patente por el investigador Carlos Ricolfe Viala, ha mostrado éxito en el decapado estático. Sin embargo, en las pruebas para el decapado continuo, el método no logró la distribución adecuada, posiblemente debido a errores en la reconstrucción 3D de las nubes de puntos. Dado que el modelo está en proceso de patente, no se puede presentar en detalle.
- **Segunda Opción: Calibración 3D Directa**:
    - Se empleará un patrón con piezas de dimensiones conocidas para establecer una calibración precisa entre el plano del láser y el de la cámara. Se utilizará una placa similar a las usadas en procesos previos, pero con una inclinación en el eje Y para captar correctamente la proyección del láser. Este proceso de calibración será repetido para asegurar la precisión de la correspondencia entre el láser y la cámara.

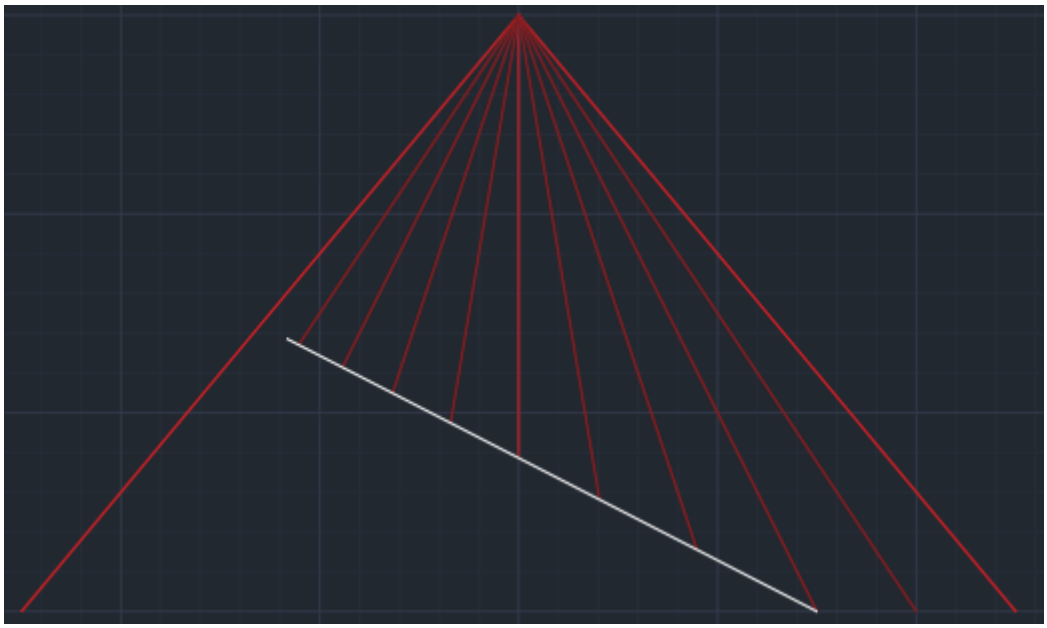


Figura 6.8 Marcaje de la Placa de Calibración Segundo Método 3D

Además, se cambiará la disposición de los elementos de forma que se eliminen errores como el cambio de correspondencia en mm/píxeles según la altura del objeto:

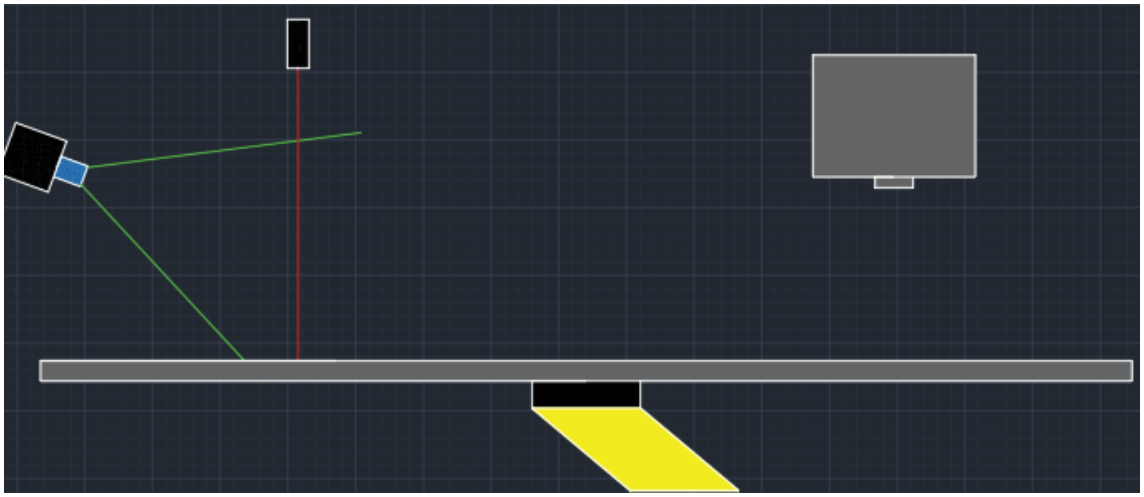


Figura 6.9 Disposición Actual de los objetos

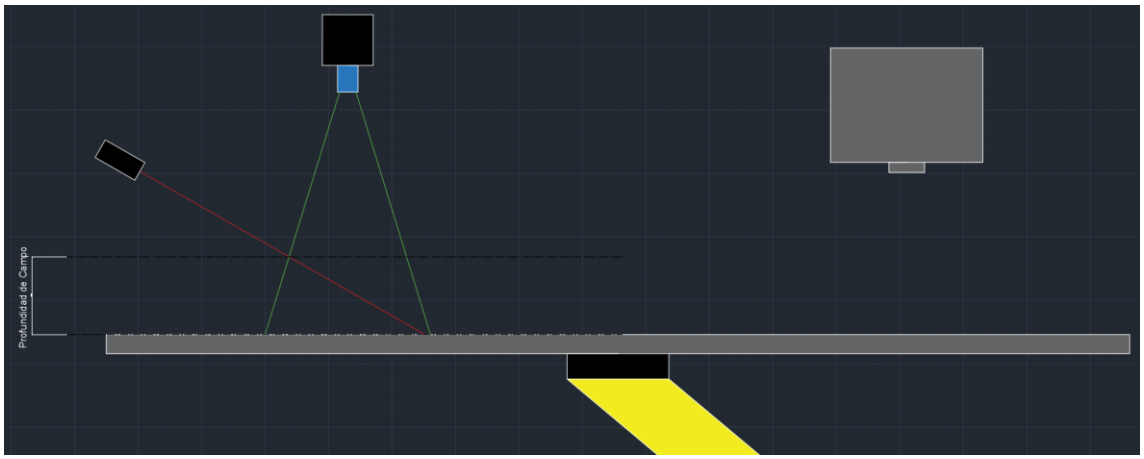


Figura 6.10 Nueva Disposición de los elementos

Correspondencia entre Planos y Transformación de Coordenadas

1. **Correspondencia entre el Plano de Luz Estructurada y el Plano de Imagen**:
    - Se logrará mediante una matriz de transformación.
2. **Transición del Plano de Luz Estructurada al Plano del Mundo**:
    - Se aplicará una rotación en el plano Y para alinear ambos planos.
3. **Conversión de Coordenadas del Mundo a Coordenadas del Láser**:
    - Se aplicará una matriz de transformación específica para convertir las coordenadas del mundo a las del láser.

$$P_{Láser} = T_{Mundo}^{Láser} \cdot R_Y \cdot T_{Imagen}^{Luz} \cdot P_{Cámara}$$

La cota Z del plano láser captado por la cámara está limitada por la profundidad de campo de la cámara. Para adaptar este método al decapado continuo, se procederá de la siguiente manera:

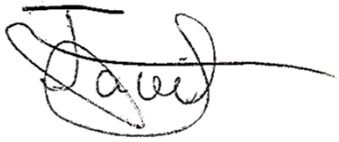
1. **Cálculo de Pulso a Milímetros**:
    - Se determinará cuántos milímetros en el eje X corresponden a cada pulso del encoder.
    - A cada pulso detectado se le sumará a la coordenada X del plano del mundo obtenida, y se multiplicará por la conversión a milímetros.
2. **Aplicación de la Transformación**:

- La matriz de transformación se empleará para ajustar la correspondencia entre el plano del láser y el plano de imagen, teniendo en cuenta la rotación y la profundidad de campo.

La disposición final del sistema puede ser modificada en el futuro, pero el modelo de reconstrucción 3D se basará en este enfoque inicial.

VALENCIA, SEPTIEMBRE 2024

Javier Gamir Artesero



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Calibración y puesta en marcha de un sistema de decapado continuo con láser

ANEXO Nº1: CÓDIGOS

Máster Universitario en Automática e Informática Industrial

Autor

Javier Gamir Artesero

Tutor

Carlos Ricolfe Viala

CURSO ACADÉMICO: 2023/2024

Índice

1 Matlab

1.1 Calibración Píxeles mm coordenadas Z e Y

```
%%Calibración función coordenada Z
% Especificar la carpeta donde se encuentran las imágenes
carpeta = 'C:\Users\javie\OneDrive\Escritorio\Master\TFM\DLLs
Matlab\Calibracion\Calibrar funcion Z\SegundoIntento';

% Obtener la lista de archivos BMP en la carpeta
archivos = dir(fullfile(carpeta, '*.bmp'));

% Inicializar una celda para almacenar las imágenes
imagenes = cell(1, length(archivos));

CoordenadasPrimerPixel=[];
CoordenadasUltimoPixel=[];
% Leer cada imagen y almacenarla en la celda
for i = 1:length(archivos)
    % Obtener el nombre completo del archivo
    nombreArchivo = fullfile(carpeta, archivos(i).name);

    % Leer la imagen
    disp(nombreArchivo)
    imagen = imread(nombreArchivo);

    % Almacenar la imagen en la celda
    imagenes{i} = imagen;

    [height,width]=size(imagen);
    threshold=175;
    bw=imagen>threshold;

    found=0;
    PrimerPunto=[];
    UltimoPunto=[];
    for col=1:width
        for row=1:height
            if bw(row,col)
                if isempty (PrimerPunto)
                    PrimerPunto=[row,col];
                else
                    UltimoPunto=[row,col];
                    break
                end
            end
        end

    end

    CoordenadasPrimerPixel=[CoordenadasPrimerPixel;PrimerPunto];
    CoordenadasUltimoPixel=[CoordenadasUltimoPixel;UltimoPunto];
```

```
end

CoordenadasEnMM=[0;5;10;15;20;25;30;35;40;45;50;55];
CoordenadasEnMMSCCamara=[0;-10;-15;-20;-25;-30;-35;-5;+10;+5];


% Mostrar las imágenes para verificar que se han leído correctamente
for i = 1:length(imagenes)
    hFig = figure;
    hAx = axes('Parent', hFig);
    threshold=175;
    bw=imagenes{i}>threshold;
    hIm = imshow(bw, 'Parent', hAx);
    title(['Imagen ', num2str(i)]);
    % Agregar el cursor de píxeles
    hPixelInfo = impixelinfo(hIm);
    set(hPixelInfo, 'Position', [5 5 200 20]);
end
```

1.2 Coordenada Z

```
%%CalibraciónZ
% Ajuste de una línea recta
modelo = fittype('poly2'); % Polinomio de primer grado
ajusteZ = fit(CoordenadasPrimerPixel(:,1), CoordenadasEnMMSCCamara,
modelo);
save('ajusteZ.mat', 'ajusteZ');

% Mostrar los coeficientes del ajuste
disp(ajusteZ);

% Crear una figura
figure;

% Graficar los datos originales
plot(CoordenadasPrimerPixel(:,1), CoordenadasEnMMSCCamara, 'o',
'DisplayName', 'Datos originales');

errormedio=mean(ajusteZ(CoordenadasPrimerPixel(:,1))-
CoordenadasEnMMSCCamara);
errorDesvTipica=std(ajusteZ(CoordenadasPrimerPixel(:,1))-
CoordenadasEnMMSCCamara);


% Sostener la gráfica para superponer el ajuste
hold on;

% Graficar la línea de ajuste
% Generar puntos para la línea de ajuste
x_fit = linspace(0, max(CoordenadasPrimerPixel(:,1)), 100);
```

```
y_fit = ajusteZ(x_fit);

% Graficar la línea de ajuste
plot(x_fit, y_fit, 'r-', 'DisplayName', 'Ajuste lineal');

% Añadir leyenda
legend('show');

% Añadir etiquetas a los ejes
xlabel('CoordenadasPrimerPixel');
ylabel('CoordenadasEnMM');
title('Ajuste de Coordenadas Z y filas de pixel');

% Liberar la gráfica
hold off;
```

1.3 Coordenada Y

```
%%CalibraciónY
AnchuraEnMM=50;
AnchuraPixeles=[];
for i=1:size(CoordenadasUltimoPixel,1)
    AnchuraPixeles=[AnchuraPixeles;CoordenadasUltimoPixel(i,2)-
CoordenadasPrimerPixel(i,2)];
end
ProporcionAnchuraMMPixeles=AnchuraEnMM./AnchuraPixeles;
% Ajuste de una línea recta
modelo = fittype('poly1'); % Polinomio de primer grado
ajusteY = fit(CoordenadasPrimerPixel(:,1), ProporcionAnchuraMMPixeles,
modelo);
save('ajusteY.mat', 'ajusteY');

% Mostrar los coeficientes del ajuste
disp(ajusteY);

% Crear una figura
figure;

% Graficar los datos originales
plot(CoordenadasPrimerPixel(:,1), ProporcionAnchuraMMPixeles, 'o',
'DisplayName', 'Datos originales');

errores=(ajusteY(CoordenadasPrimerPixel(:,1)).*CoordenadasUltimoPixel(:,2)-
ajusteY(CoordenadasPrimerPixel(:,1)).*CoordenadasPrimerPixel(:,2))-
AnchuraEnMM;
errormedio=mean(errores)
errorDesvTipica=std(errores)


% Sostener la gráfica para superponer el ajuste
```

```
hold on;

% Graficar la línea de ajuste
% Generar puntos para la línea de ajuste
x_fit = linspace(min(CoordenadasPrimerPixel(:,1)),
max(CoordenadasPrimerPixel(:,1)), 100);
y_fit = ajusteY(x_fit);

% Graficar la línea de ajuste
plot(x_fit, y_fit, 'r-', 'DisplayName', 'Ajuste lineal');

% Añadir leyenda
legend('show');

% Añadir etiquetas a los ejes
xlabel('CoordenadasPrimerPixel');
ylabel('CoordenadasEnMM');
title('Ajuste de Coordenadas y columnas de pixel');

% Liberar la gráfica
hold off;
```

1.4 Obtención modelos de corrección de proyección en X e Y

```
function [modeloDPM_X, modeloDPM_Y] = calcularDPM(puntosDeseados,
PuntosReales, distanciaCorrespondencias)

puntosDeseadosArray = [];
puntosRealesArray = [];

nPuntos = length(PuntosReales(:,1));
for i=1:nPuntos
    error = puntosDeseados - PuntosReales(i,:);
    d = vecnorm(error');
    [valor, indice] = min(d);
    if valor < distanciaCorrespondencias
        puntosDeseadosArray = [puntosDeseadosArray;
puntosDeseados(indice,:)];
        puntosRealesArray = [puntosRealesArray; PuntosReales(i,:)];
    end
end

figure(40)
hold on
grid on
plot3(puntosDeseadosArray(:,1),
puntosDeseadosArray(:,2),puntosDeseadosArray(:,3), 'r.', 'MarkerSize', 10);
plot3(puntosRealesArray(:,1),
puntosRealesArray(:,2),puntosRealesArray(:,3), 'g.', 'MarkerSize', 10);
```

```
daspect ([1 1 1])

diferencias = puntosDeseadosArray - puntosRealesArray;

figure(41)
hold on
grid on
plot3(puntosDeseadosArray(:,1), puntosDeseadosArray(:,2),diferencias(:,1),
'r.', 'MarkerSize', 10);
title ("Error en coordenada X respecto a las coordenadas XYZ")
xlabel("coordenadaX")
ylabel("coordenadaY")
zlabel("error proyección X")

figure(42)
hold on
grid on
plot3(puntosDeseadosArray(:,1), puntosDeseadosArray(:,2),diferencias(:,2),
'g.', 'MarkerSize', 10);
title ("Error en coordenada Y respecto a las coordenadas XYZ")
xlabel("coordenadaX")
ylabel("coordenadaY")
zlabel("error proyección Y")

figure(43)
hold on
grid on
plot3(puntosDeseadosArray(:,1), puntosDeseadosArray(:,2),diferencias(:,3),
'b.', 'MarkerSize', 10);
title ("Error en coordenada Z respecto a las coordenadas XYZ")
xlabel("coordenadaX")
ylabel("coordenadaY")
zlabel("error proyección Z")

% Estimación del Projection Distortion Model Extendio (DPME)
modeloDPM_X = fit(puntosDeseadosArray(:,1:2), diferencias(:,1), 'poly44');
modeloDPM_Y = fit(puntosDeseadosArray(:,1:2), diferencias(:,2), 'poly44');

% Dibujo de la superficie estimada
xMinVentana = min(puntosDeseadosArray(:,1));
xMaxVentana = max(puntosDeseadosArray(:,1));
yMinVentana = min(puntosDeseadosArray(:,2));
yMaxVentana = max(puntosDeseadosArray(:,2));

[X,Y] = meshgrid(xMinVentana:1:xMaxVentana, yMinVentana:1:yMaxVentana);
ZX = zeros(size(Y));
ZY = zeros(size(Y));
for u = 1:size(Y,1)
    for v = 1:size(Y,2)
        ZX(u,v) = modeloDPM_X(X(u,v), Y(u,v));
        ZY(u,v) = modeloDPM_Y(X(u,v), Y(u,v));
    end
end
```

```
figure(11)
plot3(puntosDeseadosArray(:,1), puntosDeseadosArray(:,2), diferencias(:,1),
'r.');
grid on
hold on
mesh(X,Y,ZX);
title ("Error en coordenada X respecto a las coordenadas XYZ")
xlabel("coordenadaX")
ylabel("coordenadaY")
zlabel("error proyección X")

figure(12)
plot3(puntosDeseadosArray(:,1), puntosDeseadosArray(:,2), diferencias(:,2),
'g.');
grid on
hold on
mesh(X,Y,ZY);
title ("Error en coordenada Y respecto a las coordenadas XYZ")
xlabel("coordenadaX")
ylabel("coordenadaY")
zlabel("error proyección Y")

% Comprobación de errores
correccionX_DPM = modeloDPM_X(puntosDeseadosArray(:,1),
puntosDeseadosArray(:,2))
correccionY_DPM = modeloDPM_Y(puntosDeseadosArray(:,1),
puntosDeseadosArray(:,2))

errores = diferencias(:,1:2) - [correccionX_DPM correccionY_DPM];

figure(13)
plot(puntosDeseadosArray(:,1), diferencias(:,1), 'b.');
grid on
hold on
plot(puntosDeseadosArray(:,1), errores(:,1), 'r.');
title ("Mejora modelo DPM X (media: " + mean(errores(:,1)) + "  std: " +
std(errores(:,1)) + ")")
xlabel("coordenadaX")
ylabel("error proyección X")
legend("SIN modelo", "CON modelo")

figure(14)
plot(puntosDeseadosArray(:,2), diferencias(:,2), 'b.');
grid on
hold on
plot(puntosDeseadosArray(:,2), errores(:,2), 'r.');
title ("Mejora modelo DPM Y (media: " + mean(errores(:,2)) + "  std: " +
std(errores(:,2)) + ")")
xlabel("coordenadaY")
ylabel("error proyección Y")
legend("SIN modelo", "CON modelo")
```

1.5 Calibración Matriz Transformación

1.5.1 Filtración de la nube de Puntos

Primero se ha creado una función que devuelva los puntos de interés de la nube de puntos que se pase como input.

```
function Puntos=ObtenerRaw(ptCloud)
umbral_Z = 28;
umbral_X_sup = 2590;
umbral_X_inf = 1700;
umbral_Y_sup = 1300;
umbral_Y_inf = 100;

ptCloud_filtrada = ptCloud;
% Filtrar puntos con X superior al umbral que equivale a la linea final
indices = ptCloud_filtrada.Location(:,1) >= umbral_X_inf &
ptCloud_filtrada.Location(:,1) <= umbral_X_sup;
ptCloud_filtrada2 = select(ptCloud_filtrada, indices);

% Filtrar puntos con Y superior al umbral que equivale a la linea final
indices = ptCloud_filtrada2.Location(:,2) <= umbral_Y_sup &
ptCloud_filtrada2.Location(:,2) >= umbral_Y_inf;
ptCloud_filtrada3 = select(ptCloud_filtrada2, indices);

Puntos(:,1)=ptCloud_filtrada3.Location(:,1);
Puntos(:,2)=ptCloud_filtrada3.Location(:,2);
Puntos(:,3)=ptCloud_filtrada3.Location(:,3);

figure()
pcshow(ptCloud_filtrada3);
title('Nube de puntos');

end
```

1.5.2 Obtener Centroides

Una vez se han obtenido los puntos de interés de la nube de puntos, se extraen los centroides de los círculos decapados por el láser.

```
function
[Centroides,pulso_mm,pixel_mm]=ObtenerCentroides(Puntos3D,filas,columnas)
```

```
for i=1:size(Puntos3D,1)

        coorX = round(Puntos3D(i,1));
        coorY = round(Puntos3D(i,2));
        if coorX == 0
            coorX = 1;
        end
        if coorY == 0
            coorY = 1;
        end
        imagen(coorY, coorX) = 255;
end

se = strel('cube',2);
imagen = imdilate(imagen, se);
imagen = imdilate(imagen, se);
imagen = imerode(imagen, se);
imagen = imerode(imagen, se);


bw = imbinarize(imagen, 0.5);

s = regionprops(bw, "Area", "Centroid",
"MajorAxisLength","MinorAxisLength");

centroids = cat(1,s.Centroid);
area = cat(1,s.Area);
mAxis = cat(1,s.MinorAxisLength);
MAxis = cat(1,s.MajorAxisLength);

buenos = (area>150) & (area < 10000) & (mAxis./MAxis) > 0.5;

centroidesBuenos=centroids(buenos,:);

puntosCalibracionCamara = [];


%Se ordenan los puntos de forma lógica:
Dist_X_puntos_2D=centroidesBuenos(2+filas,1)-centroidesBuenos(2,1);
offset_min_X_2D=min(centroidesBuenos(:,1))-Dist_X_puntos_2D/2;
offset_max_X_2D=min(centroidesBuenos(:,1))+Dist_X_puntos_2D/2;

for i=0:columnas-1 %Número de columnas-1

% Filtrar puntos dentro del intervalo especificado de coordenadas X
indices_filtrados = centroidesBuenos(:, 1) >=
(offset_min_X_2D+i*Dist_X_puntos_2D) & centroidesBuenos(:, 1) <=
(offset_max_X_2D+i*Dist_X_puntos_2D);
puntos_filtrados = centroidesBuenos(indices_filtrados, :);

% Ordenar los puntos restantes según su coordenada Y
[~, indices_ordenados] = sort(puntos_filtrados(:, 2), 'ascend');
puntos_ordenados = puntos_filtrados(indices_ordenados, :);
puntosCalibracionCamara=[puntosCalibracionCamara;puntos_ordenados];
```

```
end

 figure
imshow(imagen)
hold on
plot(centroids(buenos,1),centroids(buenos,2),'b*')
hold off

centroidesBuenos=centroids(buenos,:);
nPuntos = size(centroidesBuenos,1);
%Los puntos deben+n de ser negativos porque van a la izquierda
puntosCalibracionCamaraConvertidos=[];
puntosCalibracionCamaraConvertidos(:,1)=puntosCalibracionCamara(:,1)*-1;

for i = 1:nPuntos
    texto = sprintf("%d",i);

text(puntosCalibracionCamara(i,1)*,puntosCalibracionCamara(i,2)+25,texto,"C
olor",[1 1 0]);
end


%INTRODUCCIÓN AJUSTE Y y AJUSTE Z
ajusteZ=load('ajusteZ.mat');
ObtenerZ=ajusteZ.ajusteZ;
ajusteY=load('ajusteY.mat');
ObtenerY=ajusteY.ajusteY;

PixelColumna=1009-Puntos3D(:,3)*504/33;

AlturasAjustadas=ObtenerZ(PixelColumna);

Ky=ObtenerY(PixelColumna);
puntosCalibracionCamaraConvertidos(:,2)=Ky.*(puntosCalibracionCamara(:,2)-
536);

Centroides=[puntosCalibracionCamaraConvertidos,AlturasAjustadas];

%Ahora se obtiene la conversión de pulsos a mm y de píxeles a mm
Sum=0;
nelementos=0;
for i=1:filas*(columnas-1) %filas*columnas-1
    Sum=Sum+(10/(puntosCalibracionCamara(i+filas,1)-
puntosCalibracionCamara(i,1)));
    (10/(puntosCalibracionCamara(i+filas,1)-puntosCalibracionCamara(i,1)));
    % puntosCalibracionCamara(i+filas,1)
    % puntosCalibracionCamara(i,1)
    nelementos=nelementos+1;
end
pulso_mm=abs(Sum/nelementos);

Sum=0;
contador=0;
```

```
for i=0:columnas-1 %Numero de columna -1
    for j=1:filas-1%Numero de fiera-1
        Sum=Sum+(10/(puntosCalibracionCamara(i*filas+j+1,2)-
puntosCalibracionCamara(i*filas+j,2)));
        %puntosCalibracionCamara(i*6+j+1,2)-
puntosCalibracionCamara(i*6+j,2)
        contador=contador+1;
    end
end
pixel_mm=Sum/contador;


end
```

La función también devuelve las proporciones de pixel/mm y pulso/mm, sin embargo, no es realmente necesario excepto para el caso en el que la distancia de decapado al láser sea 275 mm, distancia a la que el decapado láser en 2D no tiene deformaciones por el modelo de disparo del láser, en tal caso la proporción pixel/mm es importante y la proporción pulso/mm permite comprobar que la función para obtener la Y en mm daba un resultado correcto.

1.5.3 Obtener Matriz de Transformación

Con los centroides obtenidos se obtiene la calibración

```
function Matriz=ObtenerMT(Centroides,minPosx,minPosy,filas,columnas)
%Se crean los puntos en coordenadas del láser que van desde el
(minposx,minposy) hasta el (minposx+10*columnas, minposy+10*filas):
puntosLaser=[];
for i=0:columnas-1
    for j=0:filas-1
    puntosLaser=[puntosLaser;[minPosx+i*10,minPosy+j*10,0]];
    end
end

figure()
for i=1:filas*columnas
    x=Centroides(i,1);
    y=Centroides(i,2);
    z=Centroides(i,3);

    plot3(x,y,z,'k.')
    hold on
```

```
     % Añadir el índice debajo de cada punto
    text(x, y - 0.1, z, num2str(i), 'VerticalAlignment', 'top',
'HorizontalAlignment', 'center')
end


for i=1:filas*columnas
    x=puntosLaser(i,1);
    y=puntosLaser(i,2);
    z=puntosLaser(i,3);
    plot3(x,y,z,'r.')
    hold on
     % Añadir el índice debajo de cada punto
    text(x, y - 0.1, z, num2str(i), 'VerticalAlignment', 'top',
'HorizontalAlignment', 'center')
end



[regParams,Bfit,Bfit1,ErrorStats]=absor(Centroides',puntosLaser');
Matriz=regParams.M;

figure
plot3(puntosLaser(:,1),puntosLaser(:,2),puntosLaser(:,3),'.b')
hold on
grid on
plot3(Bfit(1,:),Bfit(2,:),Bfit(3,:),'r.')

figure
plot(puntosLaser(:,2),puntosLaser(:,3),'.b')
hold on
grid on
plot(Bfit(2,:),Bfit(3,:),'r.')
title('Puntos Deseados vs Puntos Obtenidos con matriz Plano YZ')


errores = puntosLaser - Bfit';

figure
plot(errores(:,1),'b.')
hold on
grid on
title("error coordenada X: " + mean(errores(:,1)) + " std: "+
std(errores(:,1)) )

figure
plot(errores(:,2),'r.')
hold on
grid on
title("error coordenada Y: " + mean(errores(:,2)) + " std: "+
std(errores(:,2)) )

figure
plot3(puntosLaser(:,1),puntosLaser(:,2),errores(:,1),'.b')
```

```
hold on
grid on

figure
plot3(puntosLaser(:,1),puntosLaser(:,2),errores(:,2),'.r')
hold on
grid on

end
```

1.6 Obtención Centro del CCD

Mediante el uso de 2 de dos nubes de puntos de placas de calibración a distintas alturas obtenidas mediante el uso de las funciones expuestas previamente se obtiene el centro del CCD gracias al siguiente código:

```
Error=[];
Offsets=[];

for offset=-640:0.25:640
Offsets=[Offsets,offset];
clear ptCloud_mm
clear ptCloud_mm2

ptCloud_mm(:,1)=-NubeDePuntos1(:,1)*R_Pulso_mm;
PixelColumna=1009-NubeDePuntos1(:,3)*504/33;
Ky=ObtenerY(PixelColumna);
ptCloud_mm(:,2)=Ky.*( NubeDePuntos1(:,2)-640+offset);
ptCloud_mm(:,3)=ObtenerZ(PixelColumna);

ptCloud2_mm(:,1)=-NubeDePuntos2(:,1)*R_Pulso_mm;
PixelColumna=1009-NubeDePuntos2(:,3)*504/33;
Ky=ObtenerY(PixelColumna);
ptCloud2_mm(:,2)=Ky.*( NubeDePuntos2(:,2)-640+offset);
ptCloud2_mm(:,3)=ObtenerZ(PixelColumna);

%Se extraen los puntos pertenecientes a la columna central
puntosCalibracionCamaraExtraidos=[];
puntosCalibracionCamara2Extraidos=[];
for i=4:7:49

puntosCalibracionCamaraExtraidos=[puntosCalibracionCamaraExtraidos;ptCloud_
mm(i,2)];

puntosCalibracionCamara2Extraidos=[puntosCalibracionCamara2Extraidos;ptClou
d2_mm(i,2)];
end
```

```
error=(puntosCalibracionCamaraExtraidos-puntosCalibracionCamara2Extraidos);
Error=[Error,abs(mean(error))];
end

[Error,indice]=min(Error)

offset=Offsets(indice)

ptCloud_mm(:,1)=-NubeDePuntos1(:,1)*R_Pulso_mm;
PixelColumna=1009-NubeDePuntos1(:,3)*504/33;
Ky=ObtenerY(PixelColumna);
ptCloud_mm(:,2)=Ky.*( NubeDePuntos1(:,2)-640+offset);
ptCloud_mm(:,3)=ObtenerZ(PixelColumna);

ptCloud2_mm(:,1)=-NubeDePuntos2(:,1)*R_Pulso_mm;
PixelColumna=1009-NubeDePuntos2(:,3)*504/33;
Ky=ObtenerY(PixelColumna);
ptCloud2_mm(:,2)=Ky.*( NubeDePuntos2(:,2)-640+offset);
ptCloud2_mm(:,3)=ObtenerZ(PixelColumna);

figure()
plot(ptCloud_mm(:,1),ptCloud_mm(:,2),'b*')
hold on
plot(ptCloud2_mm(:,1),ptCloud2_mm(:,2),'g+')
```

1.7 CLAM

```
function [puntosCorregidos, puntosDecapar, centroBordesActual] =
LAM_DC(puntosPerfil2DPlanoFiltrados, centroBordes, umbralError,
anchoTeorico, reduccionFuera, reduccionDentro, debugCorreccion)
% Adapta los puntosDecapar a una nube de puntos ptCloud en la que se supone
que hay un borde plano
%%disp("Continuous Linked Adaptive Matching (CLAM) v 1.0 released 16:08:16
25/04/2024")

indices = puntosPerfil2DPlanoFiltrados(1,:) ~= 0;
puntosPerfil2DPlanoFiltrados = puntosPerfil2DPlanoFiltrados(:,indices);

% CORRECCIÓN POR AJUSTE DE LINEA A LA BASE DEL DECAPADO Y AHÍ SE BUSCA LA
DISTANCIA DE DECAPADO CENTRADA
% El centro del surco se hereda el la sección anterior
[val, indiceCentro] = min(abs(centroBordes(1) -
puntosPerfil2DPlanoFiltrados(1,:)));

if debugCorreccion == 1
    % se dibujan los puntos transformados para hacer el cálculo
    figure(3);
```

```
plot(puntosPerfil2DPlanoFiltrados(1,:),puntosPerfil2DPlanoFiltrados(2,:),'g
.');
    hold on
    grid on
end

indiceExtremo1 = indiceCentro - 5;
indiceExtremo2 = indiceCentro + 5;

% se buscan los extremos de la zona plana de la pieza
error = 0;
while (error < umbralError) && (indiceExtremo1 > 1)
    lineaDecapado =
polyfit(puntosPerfil2DPlanoFiltrados(1,indiceExtremo1:indiceCentro),
puntosPerfil2DPlanoFiltrados(2,indiceExtremo1:indiceCentro),1);
    indiceExtremo1 = indiceExtremo1 - 1;
    yEstimada =
polyval(lineaDecapado,puntosPerfil2DPlanoFiltrados(1,indiceExtremo1));
    error = abs(yEstimada -
puntosPerfil2DPlanoFiltrados(2,indiceExtremo1));
end
indiceExtremo1 = indiceExtremo1 + 1;


error = 0;
nPuntosPerfil2DPlanoFiltrados = length(puntosPerfil2DPlanoFiltrados) - 1;
while (error < umbralError) && (indiceExtremo2 <
nPuntosPerfil2DPlanoFiltrados)
    lineaDecapado =
polyfit(puntosPerfil2DPlanoFiltrados(1,indiceExtremo1:indiceExtremo2),
puntosPerfil2DPlanoFiltrados(2,indiceExtremo1:indiceExtremo2),1);
    indiceExtremo2 = indiceExtremo2 + 1;
    yEstimada =
polyval(lineaDecapado,puntosPerfil2DPlanoFiltrados(1,indiceExtremo2));
    error = abs(yEstimada -
puntosPerfil2DPlanoFiltrados(2,indiceExtremo2));
end
indiceExtremo2 = indiceExtremo2 - 1;


bordeX = [puntosPerfil2DPlanoFiltrados(1,indiceExtremo1)
puntosPerfil2DPlanoFiltrados(1,indiceExtremo2)];
bordeY = polyval(lineaDecapado,bordeX);
bordes = [bordeX; bordeY; 0 0; 1 1];
centroBordesActual = sum(bordes,2)/2;
vector = bordes(1:2,2) - bordes(1:2,1);
vector = vector / norm(vector);
diferencia = anchoTeorico / 2;

punto1Corregido = [centroBordesActual(1:2) - vector * diferencia; 0; 1];
punto1CCorregido = [centroBordesActual(1:2) + vector * diferencia; 0; 1];
puntosCorregidos = [punto1Corregido punto1CCorregido];
```

```
punto1Decapar = [centroBordesActual(1:2) - vector * (diferencia -
reduccionFuera); 0; 1];
punto1CDecapar = [centroBordesActual(1:2) + vector * (diferencia -
reduccionDentro); 0; 1];
puntosDecapar = [punto1Decapar punto1CDecapar];

if debugCorreccion == 1
    figure(3)
    plot(punto1Corregido(1),punto1Corregido(2),'k+');
    plot(punto1CCorregido(1),punto1CCorregido(2),'kx');

    linea=[punto1Corregido punto1CCorregido];
    plot(linea(1,:),linea(2,:),'k','LineWidth',1);
    plot(bordes(1,:),bordes(2,:),'k+');
    plot(centroBordes(1),centroBordes(2),'k+');

    plot(punto1Decapar(1),punto1Decapar(2),'r*');
    plot(punto1CDecapar(1),punto1CDecapar(2),'b*');
end
```

1.8 Adaptación de Puntos

```
function PuntosAdaptados=AdaptarPuntosLAM(Puntos2Adapt,paso)

indices = Puntos2Adapt(1,:) ~= 0;
puntosFiltrados = Puntos2Adapt(:,indices);
[~,tamano]=size(puntosFiltrados);

puntosR(:,1)=puntosFiltrados(1,1:paso:tamano);
puntosR(:,2)=puntosFiltrados(2,1:paso:tamano);
puntosR(:,3)=puntosFiltrados(3,1:paso:tamano);
puntosR(:,4)=puntosFiltrados(4,1:paso:tamano);

[tamano2,~]=size(puntosR);

Puntos_Medios_y=[]
for i=1:2:tamano2-1
    Media_y=(puntosR(i,2)+puntosR(i+1,2))/2;
    Puntos_Medios_y=[Puntos_Medios_y;Media_y;Media_y];
end

for j=1:6
    [tamano3,~]=size(Puntos_Medios_y);
    if(tamano2~=tamano3)
        Puntos_Medios_y(tamano3+j,1)=puntosR(j,2);
    end
end

modelo = fittype('poly2');
AdaptadorPuntos=fit(puntosR(:,1),Puntos_Medios_y(:,1),modelo);

PuntosAdaptados(1,:)=puntosFiltrados(1,:);
```

```
PuntosAdaptados(2,:)=AdaptadorPuntos(puntosFiltrados(1,:));
PuntosAdaptados(3,:)=puntosFiltrados(3,:);
PuntosAdaptados(4,:)=puntosFiltrados(4,:);

%Mantener la alternancia de los puntos originales
for i = 1:tamano
    if mod(i,2)==0
        PuntosAdaptados(2,i) = PuntosAdaptados(2,i)+0.6;
    else
        PuntosAdaptados(2,i) = PuntosAdaptados(2,i)-0.6;
    end

end

end
```

1.9 Creación Polinomial XML

```
function [] = CrearXML3Ddef(puntos3D, nombreArchivo, potencia, frecuencia,
resolucion, velocidad, repeticiones, desenfoque, gridSize,debug)

    disp("XML3D_DEF v1.0 released 13:08 26/02/2024")
    kSven = 100000/220;
    puntos3D = round(puntos3D*kSven);
    gridSize = round(gridSize*kSven);
    nPuntos = floor(size(puntos3D,2)/2);

    % La estimación de la superficie debe hacerse con la nube de puntos
centrada en su centroide.
    % Por lo tanto será necesario restar el centroide a la nube de puntos
antes de estimar la superficie.
    centroide = mean(puntos3D(1:3,:),2);
    puntos3D = puntos3D(1:3,:)-centroide ;

    xMin = min(puntos3D(1,:)) - 10000;
    xMax = max(puntos3D(1,:)) + 10000;
    yMin = min(puntos3D(2,:)) - 10000;
    yMax = max(puntos3D(2,:)) + 10000;

    % como los sectores son alargados de añade puntos a los laterales para
que
    % la estimación de la superficie sea más estable
    puntos1 = puntos3D(1:3,1:nPuntos);
    puntos2 = puntos3D(1:3,nPuntos:end);
    puntos2 = flip(puntos2,2);
    puntosSatelite = [];
    distanciaExterior = 2000;
    for i = 1:nPuntos
        direccion = puntos1(:,i) - puntos2(:,i);
        direccion = direccion/norm(direccion);
```

```
        puntosSatelite = [puntosSatelite
puntos1(:,i)+direccion*distanciaExterior puntos2(:,i)-
direccion*distanciaExterior];
    end

    puntosEstimarSuperficie = [puntos3D puntosSatelite];

    superficie = fit([puntosEstimarSuperficie(1,:)',
puntosEstimarSuperficie(2,:)'], puntosEstimarSuperficie(3,:)','poly22');
    [X,Y] = meshgrid(xMin:gridSize:xMax, yMin:gridSize:yMax);
    Z = zeros(size(Y));
    for v = 1:size(Y,2)
        for u = 1:size(Y,1)
            Z(u,v) = superficie(X(u,v), Y(u,v));
        end
    end

    % Normales
    [Nx,Ny,Nz] = surfnorm(X,Y,Z);

    % se añade el centroide
    puntos3D=puntos3D + [110 110 0]' + centroide;
    X = X + centroide(1);
    Y = Y + centroide(2);
    Z = Z + centroide(3);
    xMin = xMin + centroide(1);
    xMax = xMax + centroide(1);
    yMin = yMin + centroide(2);
    yMax = yMax + centroide(2);

    puntos1 = puntos1 + centroide;
    puntos2 = puntos2 + centroide;
    puntosSatelite = puntosSatelite + centroide;

    disp(nombreArchivo);

    fileID = fopen(nombreArchivo + ".xml",'w','n','UTF-8');
    fprintf(fileID,'<?xml version = "1.0" encoding="UTF-8"?>\n');
    fprintf(fileID,'<laserfile version="0x2">\n');
    fprintf(fileID,'    <layers>\n');
    fprintf(fileID,'        <layer name="Nueva capa" id="0" mask="0x0"
printable="1" editable="1" visible="1" power="%d" speed="%d"
resolution="%d" frequency="%d" zpos="0" zdefocus="%d" color="0xff0000"
delay="0" repeat="%d" signalmask="0x0" signalstatus="0x0" scannerset="0"
pulselen="0" pixmap="0"/>\n',potencia, round(velocidad), resolucion,
frecuencia, desenfoque, repeticiones);
    fprintf(fileID,'    </layers>\n');
    fprintf(fileID,'    <objects>\n');
    fprintf(fileID,'        <polyline type="closed"');
    fprintf(fileID,' points=" ');

    for i = 1: length(puntos3D(1,:))
        if(i == length(puntos3D))
            valText = string(puntos3D(1,i)) + " " + string(puntos3D(2,i)) ;
```

```
        else
            valText = string(puntos3D(1,i)) + " " + string(puntos3D(2,i)) +
', ';
        end
        fprintf(fileID,'%s', valText);
    end

    %fill type=1 single lines, 2=cross lines angle=90º
    fprintf(fileID,'" id="%d">\n', 0);
    fprintf(fileID,'            <generic layer_id="0" printable="1"
editable="1" linewidth="0" wobblefreq="0" wobbleratio="1"/>\n');
    fprintf(fileID,'            <fill type="2" separation="23" edge="0"
angle="0" include="0" repeat="0"/>\n');
    fprintf(fileID,'            <mask hexvalue="0x1"/>\n');
    fprintf(fileID,'            <mapped3d index="0" mode="2" ref="0"
refx="0" refy="0"/>\n');
    fprintf(fileID,'        </polyline>\n');
    fprintf(fileID,'    </objects>\n');
    fprintf(fileID,'    <objects3d>\n');
    fprintf(fileID,'        <generic3d filename="" filepath="" factor="1"
locked="0x2" locked_cx="50000" locked_cy="50000" locked_cz="0"
locked_vyx="0" locked_vyy="1" locked_vyz="0" locked_vzx="0" locked_vzy="0"
locked_vzz="1">\n');
    fprintf(fileID,'            <generic name="solid binary" cx="50000"
cy="50000" cz="0" vyx="0" vyy="1" vyz="0" vzx="0" vzy="1" vzz="0"/>\n');
    fprintf(fileID,'            <map3d brx="%.0f" bry="%.0f" tlx="%.0f"
tly="%.0f" size="%.0f" entries="',xMax,yMin,xMin,yMax,gridSize);

    contador = 1;
    for j = 1: length(Z(1,:))
        for i = 1: length(Z(:,1))
            fprintf(fileID,'%d %d %d %d, ', Nx(i,j), Ny(i,j), Nz(i,j),
Z(i,j));
            texto = sprintf("%d", contador);
%             text(X(i,j), Y(i,j), Z(i,j), texto);
            contador = contador+1;
%             plot3(X(i,j), Y(i,j), Z(i,j),'.g')
        end
    end

    fprintf(fileID,'" />\n');
    fprintf(fileID,'        </generic3d>\n');
    fprintf(fileID,'    </objects3d>\n');
    fprintf(fileID,'</laserfile>\n');

    fclose(fileID);

end
```

2 C++

2.1 Decapado Por Líneas

```
#include <stdio.h>
#include <iostream>
//#include <WS2tcpip.h>
//#include <WinSock2.h>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/visualization/cloud_viewer.h>
#include <pcl/point_cloud.h>
#include <fgrab_struct.h>
#include <fgrab_prototyp.h>
#include <fgrab_define.h>
#include <SisoDisplay.h>
#include <vector>
#include
<C:\\Users\\usuario\\Desktop\\PracticasJavi\\TCPIP\\PruebaTCPIP2\\libs\
\x64\\SocketCommDllExport.h>
#include "pugixml.cpp"

#include <boost/filesystem.hpp>
#include <C:\\Users\\usuario\\Desktop\\PracticasJavi\\DLLs
Matlab\\DLLs\\LAM_DC.h>

//#include <conio.h>
//#pragma comment (lib, "ws2_32.lib")

void printMwArray(const mwArray& array, const std::string& name) {
    std::cout << "Valores de " << name << ":" << std::endl;
    if (array.NumberOfElements() == 0) {
        std::cout << "Array vacío." << std::endl;
        return;
    }

    mwArray dimensions = array.GetDimensions();
    int numDims = dimensions.NumberOfElements();
    std::vector<mwSize> dims(numDims);
```

```
    dimensions.GetData(&dims[0], numDims);

    if (numDims == 2 && dims[0] == 1 && dims[1] == 1) {
        // Es un escalar
        std::cout << array(1, 1) << std::endl;
    }
    else {
        // Es una matriz
        for (mwSize i = 1; i <= dims[0]; ++i) {
            for (mwSize j = 1; j <= dims[1]; ++j) {
                std::cout << array(i, j) << " ";
            }
            std::cout << std::endl;
        }
    }
}


double ObtenerZ(double puntoZ) {

    double puntoZmm = - 2.444e-05 * puntoZ* puntoZ - 0.03007 * puntoZ +
14.09;
    return puntoZmm;

}

double ObtenerKY(double puntoZ) {

    double ky = 4.288e-06 * puntoZ + 0.0561;
    return ky;
}

double ObtenerY(double puntoY, double puntoZ){

    double ky = ObtenerKY(puntoZ);
    double puntoYmm = ky * (puntoY - 536);
    return puntoYmm;
}

double DPM_X(double x, double y) {

    double p00 = 64.66;
    double p10 = -3.136;
    double p01 = 0.5291;
```

```
    double p20 = 0.05009;
    double p11 = -0.003873;
    double p02 = -0.00576;
    double p30 = -0.0003419;
    double p21 = 1.936e-05;
    double p12 = 1.845e-05;
    double p03 = 2.961e-05;
    double p40 = 8.544e-07;
    double p31 = -4.213e-08;
    double p22 = -3.16e-08;
    double p13 = -3.752e-08;
    double p04 = -5.861e-08;

    double CorreccionX = p00 + p10 * x + p01 * y + p20 * x * x + p11 *
x * y + p02 * y * y + p30 * x * x * x + p21 * x * x * y + p12 * x * y *
y + p03 * y * y * y + p40 * x * x * x * x + p31 * x * x * x * y + p22 *
x * x * y * y + p13 * x * y * y * y + p04 * y * y * y * y;
    return CorreccionX;
}

double DPM_Y(double x, double y) {

    double p00 = -50.81;
    double p10 = 1.64;
    double p01 = 0.3519;
    double p20 = -0.02329;
    double p11 = -0.001963;
    double p02 = -0.00301;
    double p30 = 0.0001469;
    double p21 = 8.673e-06;
    double p12 = 1.104e-05;
    double p03 = 7.521e-06;
    double p40 = -3.456e-07;
    double p31 = -1.899e-08;
    double p22 = -1.558e-08;
    double p13 = -2.44e-08;
    double p04 = 7.187e-09;

    double CorreccionY = p00 + p10 * x + p01 * y + p20 * x * x + p11 *
x * y + p02 * y * y + p30 * x * x * x + p21 * x * x * y + p12 * x * y *
y + p03 * y * y * y + p40 * x * x * x * x + p31 * x * x * x * y + p22 *
x * x * y * y + p13 * x * y * y * y + p04 * y * y * y * y;
    return CorreccionY;
```

```
}

int main(int argc, char* argv[], char* envp[])
{
    //int ecoms;
    Fg_Struct* fg = NULL;
    int boardNr = 0;
    int camPort = PORT_A;
    frameindex_t nrOfPicturesToGrab = 2; //2 Para calibrar
    frameindex_t nbBuffers = 16;
    unsigned int width = 1280;
    unsigned int height = 1024;
    unsigned int CC0 = 10;
    int samplePerPixel = 1, k, j;
    size_t bytePerSample = 1;

    //int Valor de threshold=10 para calibrar, 40 para proceso

    int tvalue = 10, coordX = 0, pixelReferencia0 = 1009, nfotosLAM =
1730;

    //Variables para el laser
    using namespace pugi;
    const wchar_t* FilePath =
L"C:\\Users\\usuario\\Desktop\\PracticasJavi\\XMLLaser\\";
    unsigned char option = '0';
    wchar_t name[16] = L"Laser";
    wchar_t ip[16] = L"10.253.174.100";
    wchar_t path[16] = L".\\";
    const wchar_t* FilePath2 =
L"C:\\Users\\usuario\\Desktop\\PracticasJavi\\DLLs Macsa\\";
    const char* FicheroXML =
"C:\\Users\\usuario\\Desktop\\PracticasJavi\\DLLs
Macsa\\PruebaEncoderBIS.xml"; // Nombre del archivo XML original
    const wchar_t* FicheroLaser = L"XMLLaser.xml"; // Nombre del
archivo XML LASer
    std::string ArchivoEnviar;
    std::string FicheroLaserSTR =
"C:\\Users\\usuario\\Desktop\\PracticasJavi\\DLLs Macsa\\XMLLaser.xml";
    std::string newFilename =
"C:\\Users\\usuario\\Desktop\\PracticasJavi\\XMLLaser\\"; // Nombre del
nuevo archivo XML
```

```
    int sock = 0, p;

    //Variables Documentos
    // Cargar el documento XML
    xml_document doc, documentoLaser;
    //Para poder comprobar si se ha cargado bien
    xml_parse_result resultado;
    //Contador Para cambiar las variables de altura de las tres lineas
    int contadornodos = 0;

    if (!doc.load_file(FicheroXML)) {
        cerr << "Error al cargar el archivo XML." << endl;
        return -1;
    }

    //Matriz de transformación coordenadas de la cámara al láser y
ratios de conversión
    float MT[4][4] = {
        {-1.0000f, -0.0029f, -0.0000f, -97.7287f},
        {-0.0029f, 1.0000f, 0.0000f, 98.7702f},
        {0.0000f, -0.0000f, -1.0000f, 0.6430f},
        {0.0000f, 0.0000f, 0.0000f, 1.0000f}
    };
    double R_pixel_mm = 0.0571;
    float R_pulso_mm = 0.0926;

    //Pulso donde X láser=110
    int PulsoCalibracion = 2102;//1730
    int ArchivoEnviado = 0;
    int ArchivoGuardado = 0;
    double VectorCamara[4] = { -PulsoCalibracion,0.0,0.0,1.0 };
    double VectorAuxiliar[8];
    double VectorLaser1[4] = { 0.0,0.0,0.0,0.0 }, VectorLaser2[4] = {
0.0,0.0,0.0,0.0 };
    double PixelColumna;
    int newSy = 0, newEy = 0, newZDefocus = 0;

    //ConversionMMaSven = 50000 / 110;
    float KSven = 0.0022;

    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new
pcl::PointCloud<pcl::PointXYZ>);
    cloud->width = width * (nrOfPicturesToGrab);
    cloud->height = 1;
```

```
    cloud->is_dense = false;
    cloud->points.resize(cloud->width * cloud->height);

    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud2(new
pcl::PointCloud<pcl::PointXYZ>);
    cloud2->width = 420*5;
    cloud2->height = 1;
    cloud2->is_dense = false;
    cloud2->points.resize(cloud2->width * cloud2->height);
    int contadorCloud2 = 0;



    float min_y = 10000000, min_z = std::numeric_limits<float>::max(),
max_y = std::numeric_limits<float>::min();
    float max_z = std::numeric_limits<float>::min();

    //inicializar Matlab
     // Asegúrate de inicializar el entorno de MATLAB
    if (!mclInitializeApplication(NULL, 0)) {
        std::cerr << "No se pudo inicializar la aplicación MCR." <<
std::endl;
        return -1;
    }

    try {
        // Inicializar la librería de MATLAB
        if (!LAM_DCInitialize()) {
            std::cerr << "No se pudo inicializar la librería LAM_DC."
<< std::endl;
            // Obtener y mostrar la traza del error
            const char* errorMsg = mclGetLastErrorMessage();
            if (errorMsg) {
                std::cerr << "Error message: " << errorMsg <<
std::endl;
                mxFree((void*)errorMsg);  // Liberar el mensaje de
error
            }

            return -1;
        }
    }
    catch (const mwException& e) {
        std::cerr << "MWException caught: " << e.what() << std::endl;
```

```
    }
    catch (const std::exception& e) {
        std::cerr << "Standard exception caught: " << e.what() <<
std::endl;
    }
    catch (...) {
        std::cerr << "Unknown exception caught" << std::endl;
    }

    //VariablesMatlab LAMDC
    mwArray umbralError(4.0);
    mwArray anchoTeorico(33.0);
    mwArray reduccionFuera(5.0);
    mwArray reduccionDentro(7.0);
    mwArray debugCorreccion(0);
    double centroBordesData[2] = { 709.0000,438.0 };
    mwArray centroBordes(1, 2, mxDOUBLE_CLASS);
    mwArray puntosLAM(2, width, mxDOUBLE_CLASS);
    mwArray dimensions;
    int numDimensions;
    centroBordes.SetData(centroBordesData, 2); //Hay que inicializarlo
correctamente obteniendo primero una nube de puntos inicial y
calculandolo con Matlab
    //Variables Para el LAM
    double InicializacioncBA[4] = { 0.0, 0.0, 0.0, 1.0 };
    // Datos para inicializar puntosDecapar
    double InicializacionPuntos[8] = { 0.0, 0.0,0.0,1.0,0.0, 0.0,0.0,
1.0 };

    //Variables Para el LAM
    mwArray centroBordesActual, puntosCorregidos, puntosDecapar;

    //Variables función LAM
    struct Point {
        double x = 0.0;
        double y = 0.0;
        double z = 0.0;
    };
    std::vector<Point> NubeDePuntos;
    // Redimensionar el vector si es necesario
    if (NubeDePuntos.size() <= width) {
        NubeDePuntos.resize(width+1);
    }
```

```
    std::vector<double> NubeDePuntos1fila;
    if (NubeDePuntos1fila.size() <= width*2) {
        NubeDePuntos1fila.resize(width*2);
    }

    int nargout = 3;

    const char* applet;
    switch (Fg_getBoardType(boardNr)) {
    case PN_MICROENABLE4AS1CL:
        printf("1");
        applet = "Acq_FullAreaGray8";
        break;
    case PN_MICROENABLE4AD1CL:
    case PN_MICROENABLE4AD4CL:
    case PN_MICROENABLE4VD1CL:
    case PN_MICROENABLE4VD4CL:
        printf("2");
        applet = "Acq_FullAreaGray8";
        break;
    case PN_MICROENABLE4AQ4GE:
    case PN_MICROENABLE4VQ4GE:
        printf("3");
        applet = "Acq_FullAreaGray8";
        break;
    case PN_MICROENABLE3I:
        printf("4");
        applet = "Acq_FullAreaGray8";
        break;
    case PN_MICROENABLE3IXXL:
        printf("5");
        applet = "Acq_FullAreaGray8";
        break;
    default:
        applet = "DualAreaGray16";
        break;
    }

    MInit(sock, name, ip, path);
    p = MStartClient(sock);
    if (p != 0) {
        printf("Error al coenctarse al laser");
        return 1;
    }
```

```
    p = MLaser_StartPrintSession(sock, 1);

    if ((fg =
Fg_InitConfig("C:\\Users\\usuario\\Desktop\\PracticasJavi\\Codigo
PCLyCamara\\CodigoPCLconCamara - SoloUnaLinea\\Perfil6.mcf", boardNr))
== NULL) {
        fprintf(stderr, "error in Fg_Init: %s\n",
Fg_getLastErrorDescription(NULL));
        return FG_ERROR;
    }

    int dispId0 = CreateDisplay((unsigned int)(8 * bytePerSample *
samplePerPixel), width, height);
    SetBufferWidth(dispId0, width, height);

    /*Calculate buffer size (careful to avoid integer arithmetic
overflows!) and allocate memory.*/
    size_t totalBufferSize = (size_t)width * height * samplePerPixel *
bytePerSample * nbBuffers;
    dma_mem* memHandle = Fg_AllocMemEx(fg, totalBufferSize, nbBuffers);
    if (memHandle == NULL) {
        fprintf(stderr, "error in Fg_AllocMemEx: %s\n",
Fg_getLastErrorDescription(fg));
        CloseDisplay(dispId0);
        Fg_FreeGrabber(fg);
        return FG_ERROR;
    }

    /*Image width of the acquisition window.*/
    if (Fg_setParameter(fg, FG_WIDTH, &width, camPort) < 0) {
        fprintf(stderr, "Fg_setParameter(FG_WIDTH) failed: %s\n",
Fg_getLastErrorDescription(fg));
        Fg_FreeMemEx(fg, memHandle);
        CloseDisplay(dispId0);
        Fg_FreeGrabber(fg);
        return FG_ERROR;
    }

    /*Image height of the acquisition window.*/
    if (Fg_setParameter(fg, FG_HEIGHT, &height, camPort) < 0) {
        fprintf(stderr, "Fg_setParameter(FG_HEIGHT) failed: %s\n",
Fg_getLastErrorDescription(fg));
        Fg_FreeMemEx(fg, memHandle);
```

```
        CloseDisplay(dispId0);
        Fg_FreeGrabber(fg);
        return FG_ERROR;
    }

    /*cc FOR THE CAMERA INPUT.*/
    if (Fg_setParameter(fg, FG_TRIGGERCC_SELECT0, &CC0, camPort) < 0) {
        fprintf(stderr, "Fg_setParameter(FG_CCSEL0) failed: %s\n",
Fg_getLastErrorDescription(fg));
        Fg_FreeMemEx(fg, memHandle);
        CloseDisplay(dispId0);
        Fg_FreeGrabber(fg);
        return FG_ERROR;
    }



    int bitAlignment = FG_LEFT_ALIGNED;
    if (Fg_setParameter(fg, FG_BITALIGNMENT, &bitAlignment, camPort) <
0) {
        fprintf(stderr, "Fg_setParameter(FG_BITALIGNMENT) failed:
%s\n", Fg_getLastErrorDescription(fg));
        Fg_FreeMemEx(fg, memHandle);
        CloseDisplay(dispId0);
        Fg_FreeGrabber(fg);
        return FG_ERROR;
    }

    //Aquí empieza la adquisición
    if ((Fg_AcquireEx(fg, camPort, nrOfPicturesToGrab, ACQ_STANDARD,
memHandle)) < 0) {
        fprintf(stderr, "Fg_AcquireEx() failed: %s\n",
Fg_getLastErrorDescription(fg));
        Fg_FreeMemEx(fg, memHandle);
        CloseDisplay(dispId0);
        Fg_FreeGrabber(fg);
        return FG_ERROR;
    }

    frameindex_t last_pic_nr = 0;
    //int undistorted_pic[640][480];
    frameindex_t cur_pic_nr;
    int currentPicture = 0;
    int timeout = 60;
```

```
    /*
    if (ecoms = ComSetup() < 0) {
        return ecoms;
    }
     */
    printf("Que empiece el movimiento!");
    while ((cur_pic_nr = Fg_getLastPicNumberBlockingEx(fg, last_pic_nr
+ 1, camPort, timeout, memHandle)) < nrOfPicturesToGrab) {

        if (cur_pic_nr < 0) {
            Fg_stopAcquire(fg, camPort);
            Fg_FreeMemEx(fg, memHandle);
            CloseDisplay(dispId0);
            Fg_FreeGrabber(fg);
            return FG_ERROR;
        }

        last_pic_nr = cur_pic_nr;
        if (cur_pic_nr >= 1) {
                if (cur_pic_nr >= PulsoCalibracion) {
    //Si se han alcanzado el número de fotos necesario se envia archivo
al LAM
                    ArchivoEnviar =
"C:\\Users\\usuario\\Desktop\\PracticasJavi\\XMLLaser\\" +
std::to_string(ArchivoEnviado) + ".xml";
                    /*cout << ArchivoEnviar << endl;*/
                    resultado =
documentoLaser.load_file(ArchivoEnviar.c_str());
                    if (!resultado) {
                        cout << "No se ha podido cargar el archivo para
el laser";
                        return -1;
                    }
                    else {
                        documentoLaser.save_file(FicheroLaserSTR.c_str(
));
                        p = MLaser_CopyFile(sock, FicheroLaser,
FilePath2, option);
                        printf("%d \n", p);
                        p = MLaser_Reload(sock);
                        printf("%d \n", p);
                        MLaser_Start(sock, FicheroLaser, 1);
                        ArchivoEnviado++;
```

```
                    }
                    /*ArchivoEnviar <<
"C:/Users/usuario/Desktop/PrácticasJavi/ArchivosLAM" << ArchivoVacio <<
".xml";} */
                }


                else {

                    //DrawBuffer(dispId0, Fg_getImagePtrEx(fg,
undistorted_pic, camPort, memHandle), static_cast<int>(last_pic_nr),
"Cámara Lasa");

                    unsigned char* punteroImagen = (unsigned
char*)Fg_getImagePtrEx(fg, last_pic_nr, camPort, memHandle);
                    //if (cur_pic_nr < PulsoCalibracion) {
                        //buenpixel = 0;
                        //int kant = -1;
                    for (k = 0; k < width; k = k + 1)
                    {
                        for (j = 0; j < height; j = j + 1)
                        {
                            if (punteroImagen[k + j * width] >= tvalue)
                            {
                                //Para calibrar
                                /*cloud->points[k + currentPicture *
width].x = double(coordX) + 1;
                                cloud->points[k + currentPicture *
width].y = double(k);
                                cloud->points[k + currentPicture *
width].z = double(j);*/
                                //cloud->points[k + currentPicture *
width].z = ((double(pixelReferencia0) - double(j)) * 33.0 / 504.0);

                                //Si solo se quiere crear una nube por
imagen
                                cloud->points[k + width].x =
PulsoCalibracion;
                                cloud->points[k + width].y = double(k);
                                cloud->points[k + width].z =
((double(pixelReferencia0) - double(j)) * 33.0 / 504.0) * 10;
                                NubeDePuntos[k].x=PulsoCalibracion;// X
                                NubeDePuntos[k].y=double(k);    // Y
```

```
                                NubeDePuntos[k].z=(((double(pixelRefere
ncia0) - double(j)) * 33.0 / 504.0) * 10);   // Z
                                j = height;
                                //}
                            }

                        }
                    }

                    //Para calibrar
                    coordX = coordX + 1;

                    for (size_t i = 0; i < NubeDePuntos.size(); ++i) {
                        NubeDePuntos1fila[i * 2 + 0] =
NubeDePuntos[i].y;
                        NubeDePuntos1fila[i * 2 + 1] =
NubeDePuntos[i].z;

                    //Se asignan los puntos de la nube de puntos a la
variable de matlab
                    try {
                        // Utilizar SetData para copiar los datos a la
matriz mwArray
                        puntosLAM.SetData(NubeDePuntos1fila.data(),
NubeDePuntos1fila.size());
                    }
                    catch (const mwException& e) {
                        std::cerr << "mwException: " << e.what() <<
std::endl;
                        return -1;
                    }
                    catch (...) {
                        std::cerr << "Error desconocido al usar
SetData." << std::endl;
                        return -1;
                    }
                   // //Pasar Nube de Puntos al LAM
                    LAM_DC(nargout, puntosCorregidos, puntosDecapar,
centroBordesActual, puntosLAM, centroBordes, umbralError, anchoTeorico,
reduccionFuera, reduccionDentro, debugCorreccion);

                    //Se le asigna el valor a centroBordes de
centroBordesActual para la proxima iteración
                    centroBordes(1, 1) = centroBordesActual(1, 1);
```

```
                    centroBordes(1, 2) = centroBordesActual(2, 1);
                    /*printMwArray(puntosDecapar, "puntosDecapar")*/;
                    puntosDecapar.GetData(VectorAuxiliar, 8);



                    PixelColumna = pixelReferencia0 -
VectorAuxiliar[1]*504/330;


                    VectorCamara[0] = -PulsoCalibracion * R_pulso_mm;
                    VectorCamara[1] =
ObtenerY(VectorAuxiliar[0],PixelColumna);
                    VectorCamara[2] = ObtenerZ(PixelColumna);



                    for (int l = 0; l < 4; ++l) {
                        for (int d = 0; d < 4; ++d) {
                            VectorLaser1[l] += MT[l][d] *
VectorCamara[d];
                        }
                    }

                    VectorLaser1[0] = VectorLaser1[0] +
DPM_X(VectorLaser1[0], VectorLaser1[1]);
                    VectorLaser1[1] = VectorLaser1[1] +
DPM_Y(VectorLaser1[0], VectorLaser1[1]);

                    PixelColumna = pixelReferencia0 - VectorAuxiliar[5]
* 504 / 330;
                    /*printf("%f", PixelColumna)*/;

                    VectorCamara[1] = ObtenerY(VectorAuxiliar[4],
PixelColumna);
                    VectorCamara[2] = ObtenerZ(PixelColumna);
                    for (int l = 0; l < 4; ++l) {
                        for (int d = 0; d < 4; ++d) {
                            VectorLaser2[l] += MT[l][d] *
VectorCamara[d];
                        }
                    }
                    VectorLaser2[0] = VectorLaser2[0] +
DPM_X(VectorLaser2[0], VectorLaser2[1]);
```

```
                    VectorLaser2[1] = VectorLaser2[1] +
DPM_Y(VectorLaser2[0], VectorLaser2[1]);

                    // Obtener el nodo de la línea (<line>)
                    xml_node lineNode =
doc.select_node("//line").node();

                    // Obtener el nodo de la línea (<layer>)
                    xml_node layerNode =
doc.select_node("//layer").node();

                    // Modificar los valores de sy, ey y zdefocus con
variables personalizadas
                    newSy = (VectorLaser1[1]) /KSven;    // Nuevo valor
de sy
                    newEy =( VectorLaser2[1]) /KSven;    // Nuevo valor
de ey
                    int newZDefocus = ((VectorLaser1[2] +
VectorLaser2[2]) / 2);  // Nuevo valor de zdefocus
                    // Actualizar los valores en el archivo XML
                    /*lineNode.attribute("sy").set_value(newSy);
                    lineNode.attribute("ey").set_value(newEy);*/
                    while (lineNode && contadornodos < 2) {
                        lineNode.attribute("sy").set_value(newSy);
                        lineNode.attribute("ey").set_value(newEy);
                        lineNode = lineNode.next_sibling("line"); //
Mover al siguiente nodo "line"
                        contadornodos++;
                    }
                    contadornodos = 0;

                    if (!layerNode.attribute("zdefocus")) {
                        std::cerr << "El atributo zdefocus no está
presente en el nodo line." << std::endl;;
                    }
                    layerNode.attribute("zdefocus").set_value(newZDefoc
us);

                    newFilename =
"C:\\Users\\usuario\\Desktop\\PracticasJavi\\XMLLaser\\" +
std::to_string(ArchivoGuardado) + ".xml"; // Nombre del nuevo archivo
XML
                    if (!doc.save_file(newFilename.c_str())) {
```

```
                        cerr << "Error al guardar el archivo XML
modificado." << endl;
                        return 1;
                    }
                    else {
                        ArchivoGuardado = ArchivoGuardado + 1;
                    }
                    VectorLaser1[0] = 0;
                    VectorLaser1[1] = 0;
                    VectorLaser1[2] = 0;
                    VectorLaser2[0] = 0;
                    VectorLaser2[1] = 0;
                    VectorLaser2[2] = 0;
                    buenpixel = 0;

                }
                //Comentar la siguiente línea si se está calibrando
                cloud->clear();
            }
                //}
                DrawBuffer(dispId0, Fg_getImagePtrEx(fg, last_pic_nr,
camPort, memHandle), static_cast<int>(currentPicture), "Cámara Lasa");
                currentPicture = currentPicture + 1;


        }


    //... populate cloud
    // Finalizar la biblioteca de componentes de MATLAB Compiler
Runtime
    LAM_DCTerminate();
    // Finalizar la aplicación de MATLAB Compiler Runtime
    mclTerminateApplication();

    pcl::visualization::PCLVisualizer::Ptr viewer(new
pcl::visualization::PCLVisualizer("3D Viewer"));
    viewer->setBackgroundColor(0, 0, 0);

    viewer->addPointCloud<pcl::PointXYZ>(cloud, "cloud");

    while (!viewer->wasStopped())
    {
        viewer->spinOnce(100);
```

```
    }



    // Especificar la ruta completa de la carpeta para guardar el
archivo de texto para calibrar
    std::string folderPath =
"C:/Users/usuario/Desktop/PracticasJavi/ArchivosLAM/"; // Cambia
"/ruta/a/la/carpeta/" por la ruta de la carpeta deseada

    // Abrir un archivo de texto para escritura
    std::ofstream outfile(folderPath + "Segmento.txt");

    // Verificar si el archivo se abrió correctamente
    if (!outfile.is_open()) {
        std::cerr << "no se pudo abrir el archivo para escritura." <<
std::endl;
        return -1;
    }



    // Iterar sobre cada punto en la nube de puntos y escribir sus
coordenadas en el archivo de texto
    for (const auto& point : *cloud) {
        outfile << point.x << " " << point.y << " " << point.z <<
std::endl;
    }

// Si se desea guardar la nube de puntos como PCD:

   /* pcl::io::savePCDFile(folderPath + "PCDCLoud.pcd", *cloud);
    std::cerr << "Saved " << cloud->size() << " data points to
test_pcd.pcd." << std::endl;*/

   /* pcl::io::savePCDFile(folderPath + "ComprobarCalibracion.pcd",
*cloud);*/

    // Cerrar el archivo
    outfile.close();

   /* std::cout << "Nube de puntos guardada en " << folderPath <<
"output_cloud.txt" << std::endl;*/
```

```
    CloseDisplay(dispId0);
    Fg_stopAcquire(fg, camPort);
    Fg_FreeMemEx(fg, memHandle);
    Fg_FreeGrabber(fg);

    return FG_OK;
}
```

2.2 Decapado Continuo

```
#include <stdio.h>
#include <iostream>
//#include <WS2tcpip.h>
//#include <WinSock2.h>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/visualization/cloud_viewer.h>
#include <pcl/point_cloud.h>
#include <fgrab_struct.h>
#include <fgrab_prototyp.h>
#include <fgrab_define.h>
#include <SisoDisplay.h>
#include <vector>
#include
<C:\\Users\\usuario\\Desktop\\PracticasJavi\\TCPIP\\PruebaTCPIP2\\libs\
\x64\\SocketCommDllExport.h>
#include "pugixml.cpp"

#include "LAM_DC.h"
#include "CrearXML3Ddef.h"
#include "AdaptarPuntosLAM.h"
#include <armadillo>
#include <matrix.h>
#include <Eigen/Core>

using namespace arma;
using namespace pugi;

int CrearPolinomial(mat* Matriz,int* numeroArchivo, int*
ContadorPuntos,mwArray* potencia, mwArray* frecuencia, mwArray*
```

```
resolucion, mwArray* velocidad, mwArray* repeticiones, mwArray*
desenfoque, mwArray* gridSize, mwArray* debug) {
    if(*ContadorPuntos==420-1){//Porque empiezo desde 0, entonces de 0
a 419 son los 420 pulsos

        // Guardamos la penúltima y última columna ya que serán las
coordenadas Y, y Z de los primeros puntos del siguiente disparo
        arma::vec penultimaColumna = Matriz->col(Matriz->n_cols - 2);
        arma::vec ultimaColumna = Matriz->col(Matriz->n_cols - 1);

        // Guardamos la primera y segunda columna ya que serán la
coordenada X de los primeros puntos del siguiente disparo
        arma::vec primeraColumna = Matriz->col(0);
        arma::vec segundaColumna = Matriz->col(1);

        // Cambiamos la primera coordenada de penúltima y última
columna con la de la primera y segunda respectivamente para quedarnos
con la X de interés
        penultimaColumna(0) = primeraColumna(0);
        ultimaColumna(0) = segundaColumna(0);

        mwArray matlabArray(Matriz->n_rows, Matriz->n_cols,
mxDOUBLE_CLASS);
        mwArray matlabArrayAdaptada(Matriz->n_rows, Matriz->n_cols,
mxDOUBLE_CLASS);
        mwArray paso(3.0);
        // Llamar a la función de Matlab
        // Llamar a la función de Matlab
        matlabArray.SetData(Matriz->memptr(), Matriz->n_elem);

        // Cambio en el nombre del archivo

        std::string nuevoNombreArchivo =
"C:\\Users\\usuario\\Desktop\\PracticasJavi\\XMLLaser\\" +
std::to_string(*numeroArchivo);
        mwArray nombreArchivoMw(nuevoNombreArchivo.c_str());
        *numeroArchivo = *numeroArchivo + 1;

        try {
            AdaptarPuntosLAM(1,matlabArrayAdaptada,matlabArray,paso);
        }
        catch (const mwException& e) {
            std::cerr << "Error al llamar a la función de Matlab: " <<
e.what() << std::endl;
```

```
            return -1; // Salir de la función si ocurre un error
        }

        try {
            CrearXML3Ddef(matlabArrayAdaptada, nombreArchivoMw,
*potencia, *frecuencia, *resolucion, *velocidad, *repeticiones,
*desenfoque, *gridSize, *debug);
        }
        catch (const mwException& e) {
            std::cerr << "Error al llamar a la función de Matlab: " <<
e.what() << std::endl;
            return -1; // Salir de la función si ocurre un error
        }
        //Se reinicia la matriz
        Matriz->zeros();

        // Se asignan las columnas guardadas a la primera y segunda
columna
        Matriz->col(0) = penultimaColumna;
        Matriz->col(1) = ultimaColumna;

        *ContadorPuntos = 1;
        return 0; // Devuelve 0 para indicar éxito
    }
    else {
        *ContadorPuntos = *ContadorPuntos + 1;
        return 0;
    }

}


double ObtenerZ(double puntoZ) {

    double puntoZmm = -2.444e-05 * puntoZ * puntoZ - 0.03007 * puntoZ +
14.09;
    return puntoZmm;

}

double ObtenerKY(double puntoZ) {

    double ky = 4.288e-06 * puntoZ + 0.0561;
    return ky;
```

```
}

double ObtenerY(double puntoY, double puntoZ) {

    double ky = ObtenerKY(puntoZ);
    double puntoYmm = ky * (puntoY - 536);
    return puntoYmm;
}

double DPM_X(double x, double y) {

    double p00 = 64.66;
    double p10 = -3.136;
    double p01 = 0.5291;
    double p20 = 0.05009;
    double p11 = -0.003873;
    double p02 = -0.00576;
    double p30 = -0.0003419;
    double p21 = 1.936e-05;
    double p12 = 1.845e-05;
    double p03 = 2.961e-05;
    double p40 = 8.544e-07;
    double p31 = -4.213e-08;
    double p22 = -3.16e-08;
    double p13 = -3.752e-08;
    double p04 = -5.861e-08;

    double CorreccionX = p00 + p10 * x + p01 * y + p20 * x * x + p11 *
x * y + p02 * y * y + p30 * x * x * x + p21 * x * x * y + p12 * x * y *
y + p03 * y * y * y + p40 * x * x * x * x + p31 * x * x * x * y + p22 *
x * x * y * y + p13 * x * y * y * y + p04 * y * y * y * y;
    return CorreccionX;
}

double DPM_Y(double x, double y) {

    double p00 = -50.81;
    double p10 = 1.64;
    double p01 = 0.3519;
    double p20 = -0.02329;
    double p11 = -0.001963;
    double p02 = -0.00301;
    double p30 = 0.0001469;
    double p21 = 8.673e-06;
```

```
    double p12 = 1.104e-05;
    double p03 = 7.521e-06;
    double p40 = -3.456e-07;
    double p31 = -1.899e-08;
    double p22 = -1.558e-08;
    double p13 = -2.44e-08;
    double p04 = 7.187e-09;

    double CorreccionY = p00 + p10 * x + p01 * y + p20 * x * x + p11 *
x * y + p02 * y * y + p30 * x * x * x + p21 * x * x * y + p12 * x * y *
y + p03 * y * y * y + p40 * x * x * x * x + p31 * x * x * x * y + p22 *
x * x * y * y + p13 * x * y * y * y + p04 * y * y * y * y;
    return CorreccionY;


}

//#include <conio.h>
//#pragma comment (lib, "ws2_32.lib")

void printMwArray(const mwArray& array, const std::string& name) {
    std::cout << "Valores de " << name << ":" << std::endl;
    if (array.NumberOfElements() == 0) {
        std::cout << "Array vacío." << std::endl;
        return;
    }

    mwArray dimensions = array.GetDimensions();
    int numDims = dimensions.NumberOfElements();
    std::vector<mwSize> dims(numDims);
    dimensions.GetData(&dims[0], numDims);

    if (numDims == 2 && dims[0] == 1 && dims[1] == 1) {
        // Es un escalar
        std::cout << array(1, 1) << std::endl;
    }
    else {
        // Es una matriz
        for (mwSize i = 1; i <= dims[0]; ++i) {
            for (mwSize j = 1; j <= dims[1]; ++j) {
                std::cout << array(i, j) << " ";
            }
            std::cout << std::endl;
        }
```

```
    }
}


int main(int argc, char* argv[], char* envp[])
{
    // Habilitar el heap de depuración de CRT
    _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);

    //int ecoms;
    Fg_Struct* fg = NULL;
    int boardNr = 0;
    int camPort = PORT_A;
    frameindex_t nrOfPicturesToGrab = 3600;
    frameindex_t nbBuffers = 16;
    unsigned int width = 1280;
    unsigned int height = 1024;
    unsigned int CC0 = 10;
    int samplePerPixel = 1, k, j;
    size_t bytePerSample = 1;

    //Valor de threshold para calibrar=10, para proceso=40

    int tvalue = 40, coordX = 0, pixelReferencia0 = 1009;

    //Variables para el laser

    const wchar_t* FilePath =
L"C:\\Users\\usuario\\Desktop\\PracticasJavi\\XMLLaser\\";
    unsigned char option = '0';
    wchar_t name[16] = L"Laser";
    wchar_t ip[16] = L"10.253.174.100";
    wchar_t path[16] = L".\\";
    const wchar_t* FilePath2 =
L"C:\\Users\\usuario\\Desktop\\PracticasJavi\\DLLs Macsa\\";
    const char* FicheroXML =
"C:\\Users\\usuario\\Desktop\\PracticasJavi\\DLLs
Macsa\\PruebaEncoderDobleRect.xml"; // Nombre del archivo XML original
    const wchar_t* FicheroLaser = L"XMLLaser.xml"; // Nombre del
archivo XML LASer
    std::string ArchivoEnviar;
    std::string FicheroLaserSTR =
"C:\\Users\\usuario\\Desktop\\PracticasJavi\\DLLs Macsa\\XMLLaser.xml";
```

```
    std::string newFilename =
"C:\\Users\\usuario\\Desktop\\PracticasJavi\\XMLLaser\\"; // Nombre del
nuevo archivo XML
    int sock = 0,p;
    // Cargar el documento XML
    xml_document doc,documentoLaser;

    if (!doc.load_file(FicheroXML)) {
        cerr << "Error al cargar el archivo XML." << endl;
        return -1;
    }

    //Matriz de transformación coordenadas de la cámara al láser
    float MT[4][4] = {
        {-1.0000f, -0.0029f, -0.0000f, -97.7287f},
        {-0.0029f, 1.0000f, 0.0000f, 98.7702f},
        {0.0000f, -0.0000f, -1.0000f, 0.6430f},
        {0.0000f, 0.0000f, 0.0000f, 1.0000f}
    };
    double R_pixel_mm = 0.0575;
    float R_pulso_mm = 0.0926;

    //Pulso donde X láser=120 para hacer 40 mm de sector son 420 pulsos
    int PulsoCalibracion = 2316;
    float PulsoTramoFinal = -2316;
    int ArchivoEnviado = 0;
    int ArchivoGuardado = 0;
    double VectorCamara[4] = { -PulsoCalibracion,0,0,1 };
    double PixelColumna;
    double VectorAuxiliar[8];
    //ConversionMMaSven = 50000 / 110;


    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new
pcl::PointCloud<pcl::PointXYZ>);
    cloud->width = width* nrOfPicturesToGrab;
    cloud->height = 1;
    cloud->is_dense = false;
    cloud->points.resize(cloud->width * cloud->height);

    if (!mclInitializeApplication(NULL, 0)) {
        std::cerr << "No se pudo inicializar la aplicación MCR." <<
std::endl;
        return -1;
```

```
    }

    try {
        // Inicializar la librería de MATLAB
        if (!LAM_DCInitialize()) {
            std::cerr << "No se pudo inicializar la librería LAM_DC."
<< std::endl;
            // Obtener y mostrar la traza del error
            const char* errorMsg = mclGetLastErrorMessage();
            if (errorMsg) {
                std::cerr << "Error message: " << errorMsg <<
std::endl;
                mxFree((void*)errorMsg);  // Liberar el mensaje de
error
            }

            return -1;
        }
    }
    catch (const mwException& e) {
        std::cerr << "MWException caught: " << e.what() << std::endl;
    }
    catch (const std::exception& e) {
        std::cerr << "Standard exception caught: " << e.what() <<
std::endl;
    }
    catch (...) {
        std::cerr << "Unknown exception caught" << std::endl;
    }

    try {
        // Inicializar la segunda librería de MATLAB
        if (!CrearXML3DdefInitialize()) {
            std::cerr << "No se pudo inicializar la librería
CrearXML3D." << std::endl;
            // Obtener y mostrar la traza del error
            const char* errorMsg = mclGetLastErrorMessage();
            if (errorMsg) {
                std::cerr << "Error message: " << errorMsg <<
std::endl;
                mxFree((void*)errorMsg);  // Liberar el mensaje de
error
            }
```

```
            return -1;
        }
    }
    catch (const mwException& e) {
        std::cerr << "MWException caught: " << e.what() << std::endl;
    }
    catch (const std::exception& e) {
        std::cerr << "Standard exception caught: " << e.what() <<
std::endl;
    }
    catch (...) {
        std::cerr << "Unknown exception caught" << std::endl;
    }

    try {
        // Inicializar la tercera librería de MATLAB
        if (!AdaptarPuntosLAMInitialize()) {
            std::cerr << "No se pudo inicializar la librería
CrearXML3D." << std::endl;
            // Obtener y mostrar la traza del error
            const char* errorMsg = mclGetLastErrorMessage();
            if (errorMsg) {
                std::cerr << "Error message: " << errorMsg <<
std::endl;
                mxFree((void*)errorMsg);  // Liberar el mensaje de
error
            }

            return -1;
        }
    }
    catch (const mwException& e) {
        std::cerr << "MWException caught: " << e.what() << std::endl;
    }
    catch (const std::exception& e) {
        std::cerr << "Standard exception caught: " << e.what() <<
std::endl;
    }
    catch (...) {
        std::cerr << "Unknown exception caught" << std::endl;
    }


    //Variables Matlab CrearPolinomial
```

```
    mwArray puntos3D;
    mwArray nombreArchivo =
"C:\\Users\\usuario\\Desktop\\PracticasJavi\\XMLLaser\\output.xml";
    mwArray potencia(50.0);          // Potencia laser
    mwArray frecuencia(50.0);         // Frecuencia laser
    mwArray resolucion(100);           // Resolucion
    mwArray velocidad(1500000);            // Velocidad de pasada del
láser
    mwArray repeticiones(1.0);        // Número de veces a hacer el
fichero
    mwArray desenfoque(0);           // Desenfoque en Z
    mwArray gridSize(1.0);             //
    mwArray debug(0.0);                //

    //VariablesMatlab LAMDC
    mwArray umbralError(4.0);
    mwArray anchoTeorico(33.0);
    mwArray reduccionFuera(5.0);
    mwArray reduccionDentro(7.0);
    mwArray debugCorreccion(0);
    double centroBordesData[2] = { 751.0000, 390.8196 };
    mwArray centroBordes(1, 2, mxDOUBLE_CLASS);
    mwArray puntosLAM(2, width, mxDOUBLE_CLASS);
    mwArray dimensions;
    int numDimensions;
    centroBordes.SetData(centroBordesData, 2); //Hay que inicializarlo
correctamente obteniendo primero una nube de puntos inicial y
calculandolo con Matlab

    //Variables Para el LAM
    mwArray centroBordesActual, puntosCorregidos, puntosDecapar;


    //Variables función CrearPolinomial
    mat *MatrizPuntos = new mat(4, 420*2, fill::zeros);
    vec VectorLaser1(4, fill::zeros);
    vec VectorLaser2(4, fill::zeros);
    int pointCount = 0;
    int columncount = 0;
    float coordenadaX = 120;
    rowvec VectorPuntos = {coordenadaX,0,0 };
    int respuesta = 0;

    //Variables función LAM
```

```
struct Point {
    double x = 0.0;
    double y = 0.0;
    double z = 0.0;
};
std::vector<Point> NubeDePuntos;
int nargout=3;

if (NubeDePuntos.size() <= width) {
    NubeDePuntos.resize(width + 1);
}

std::vector<double> NubeDePuntos1fila;
if (NubeDePuntos1fila.size() <= width * 2) {
    NubeDePuntos1fila.resize(width * 2);
}

const char* applet;
switch (Fg_getBoardType(boardNr)) {
case PN_MICROENABLE4AS1CL:
    printf("1");
    applet = "Acq_FullAreaGray8";
    break;
case PN_MICROENABLE4AD1CL:
case PN_MICROENABLE4AD4CL:
case PN_MICROENABLE4VD1CL:
case PN_MICROENABLE4VD4CL:
    printf("2");
    applet = "Acq_FullAreaGray8";
    break;
case PN_MICROENABLE4AQ4GE:
case PN_MICROENABLE4VQ4GE:
    printf("3");
    applet = "Acq_FullAreaGray8";
    break;
case PN_MICROENABLE3I:
    printf("4");
    applet = "Acq_FullAreaGray8";
    break;
case PN_MICROENABLE3IXXL:
    printf("5");
    applet = "Acq_FullAreaGray8";
    break;
default:
```

```
        applet = "DualAreaGray16";
        break;
    }

    MInit(sock, name, ip, path);
    p = MStartClient(sock);
    if (p != 0) {
        printf("Error al coenctarse al laser");
        return 1;
    }

    p = MLaser_StartPrintSession(sock, 1);

    if ((fg =
Fg_InitConfig("C:\\Users\\usuario\\Desktop\\PracticasJavi\\Codigo
PCLyCamara\\CodigoPCLconCamara - SoloUnaLinea\\Perfil6.mcf", boardNr))
== NULL) {
        fprintf(stderr, "error in Fg_Init: %s\n",
Fg_getLastErrorDescription(NULL));
        return FG_ERROR;
    }

    int dispId0 = CreateDisplay((unsigned int)(8 * bytePerSample *
samplePerPixel), width, height);
    SetBufferWidth(dispId0, width, height);

    /*Calculate buffer size (careful to avoid integer arithmetic
overflows!) and allocate memory.*/
    size_t totalBufferSize = (size_t)width * height * samplePerPixel *
bytePerSample * nbBuffers;
    dma_mem* memHandle = Fg_AllocMemEx(fg, totalBufferSize, nbBuffers);
    if (memHandle == NULL) {
        fprintf(stderr, "error in Fg_AllocMemEx: %s\n",
Fg_getLastErrorDescription(fg));
        CloseDisplay(dispId0);
        Fg_FreeGrabber(fg);
        return FG_ERROR;
    }

    /*Image width of the acquisition window.*/
    if (Fg_setParameter(fg, FG_WIDTH, &width, camPort) < 0) {
        fprintf(stderr, "Fg_setParameter(FG_WIDTH) failed: %s\n",
Fg_getLastErrorDescription(fg));
        Fg_FreeMemEx(fg, memHandle);
```

```
        CloseDisplay(dispId0);
        Fg_FreeGrabber(fg);
        return FG_ERROR;
    }

    /*Image height of the acquisition window.*/
    if (Fg_setParameter(fg, FG_HEIGHT, &height, camPort) < 0) {
        fprintf(stderr, "Fg_setParameter(FG_HEIGHT) failed: %s\n",
Fg_getLastErrorDescription(fg));
        Fg_FreeMemEx(fg, memHandle);
        CloseDisplay(dispId0);
        Fg_FreeGrabber(fg);
        return FG_ERROR;
    }

    /*cc FOR THE CAMERA INPUT.*/
    if (Fg_setParameter(fg, FG_TRIGGERCC_SELECT0, &CC0, camPort) < 0) {
        fprintf(stderr, "Fg_setParameter(FG_CCSEL0) failed: %s\n",
Fg_getLastErrorDescription(fg));
        Fg_FreeMemEx(fg, memHandle);
        CloseDisplay(dispId0);
        Fg_FreeGrabber(fg);
        return FG_ERROR;
    }



    int bitAlignment = FG_LEFT_ALIGNED;
    if (Fg_setParameter(fg, FG_BITALIGNMENT, &bitAlignment, camPort) <
0) {
        fprintf(stderr, "Fg_setParameter(FG_BITALIGNMENT) failed:
%s\n", Fg_getLastErrorDescription(fg));
        Fg_FreeMemEx(fg, memHandle);
        CloseDisplay(dispId0);
        Fg_FreeGrabber(fg);
        return FG_ERROR;
    }

    //Aquí empieza la adquisición
    if ((Fg_AcquireEx(fg, camPort, nrOfPicturesToGrab, ACQ_STANDARD,
memHandle)) < 0) {
        fprintf(stderr, "Fg_AcquireEx() failed: %s\n",
Fg_getLastErrorDescription(fg));
        Fg_FreeMemEx(fg, memHandle);
```

```
        CloseDisplay(dispId0);
        Fg_FreeGrabber(fg);
        return FG_ERROR;
    }

    frameindex_t last_pic_nr = 0;
    //int undistorted_pic[640][480];
    frameindex_t cur_pic_nr;
    int currentPicture = 0;
    int timeout = 80;

    /*
    if (ecoms = ComSetup() < 0) {
        return ecoms;
    }
     */
    while ((cur_pic_nr = Fg_getLastPicNumberBlockingEx(fg, last_pic_nr
+ 1, camPort, timeout, memHandle)) < nrOfPicturesToGrab) {
        if (cur_pic_nr >= PulsoCalibracion && (cur_pic_nr -
PulsoCalibracion) % 420 == 0) {
            //Si se han alcanzado el número de fotos necesario se envia
archivo al LAM
            ArchivoEnviar =
"C:\\Users\\usuario\\Desktop\\PracticasJavi\\XMLLaser\\" +
std::to_string(ArchivoEnviado) + ".xml";
            documentoLaser.load_file(ArchivoEnviar.c_str());
            documentoLaser.save_file(FicheroLaserSTR.c_str());
            p = MLaser_CopyFile(sock, FicheroLaser, FilePath2, option);
            printf("%d \n", p);
            p = MLaser_Reload(sock);
            printf("%d \n", p);
            MLaser_Start(sock, FicheroLaser, 1);
            ArchivoEnviado++;
            //ArchivoEnviar <<
"C:/Users/usuario/Desktop/PrácticasJavi/ArchivosLAM" << ArchivoVacio <<
".xml";}
        }

        if (cur_pic_nr < 0) {
            Fg_stopAcquire(fg, camPort);
            Fg_FreeMemEx(fg, memHandle);
            CloseDisplay(dispId0);
            Fg_FreeGrabber(fg);
            return FG_ERROR;
```

```
        }

        last_pic_nr = cur_pic_nr;


        //DrawBuffer(dispId0, Fg_getImagePtrEx(fg, undistorted_pic,
camPort, memHandle), static_cast<int>(last_pic_nr), "Cámara Lasa");

        unsigned char* punteroImagen = (unsigned
char*)Fg_getImagePtrEx(fg, last_pic_nr, camPort, memHandle);
        if (cur_pic_nr < PulsoCalibracion && cur_pic_nr>=1)
        {
            //buenpixel = 0;
            int kant = -1;
            for (k = 0; k < width; k = k + 1)
            {
                for (j = 0; j < height; j = j + 1)
                {
                    if (punteroImagen[k + j * width] >= tvalue)
                    {

                        NubeDePuntos[k].x = PulsoCalibracion;// X
                        NubeDePuntos[k].y = double(k);   // Y
                        NubeDePuntos[k].z = (((double(pixelReferencia0)
- double(j)) * 33.0 / 504.0) * 10);   // Z
                        j = height;
                        //}
                    }

                }
            }

            for (size_t i = 0; i < NubeDePuntos.size(); ++i) {
                NubeDePuntos1fila[i * 2 + 0] = NubeDePuntos[i].y;
                NubeDePuntos1fila[i * 2 + 1] = NubeDePuntos[i].z;
            }


            // Copiar los datos desde el vector a mwArray usando
SetData
            puntosLAM.SetData(NubeDePuntos1fila.data(),
NubeDePuntos1fila.size());

            try {
```

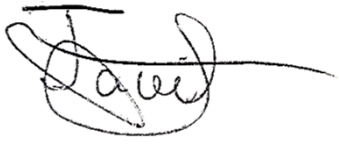
```
                // Utilizar SetData para copiar los datos a la matriz
mwArray
                LAM_DC(3, puntosCorregidos, puntosDecapar,
centroBordesActual, puntosLAM, centroBordes, umbralError, anchoTeorico,
reduccionFuera, reduccionDentro, debugCorreccion);
            }
            catch (const mwException& e) {
                std::cerr << "mwException: " << e.what() << std::endl;
                return -1;
            }
            catch (...) {
                std::cerr << "Error desconocido al usar SetData." <<
std::endl;
                return -1;
            }

            centroBordes(1, 1) = centroBordesActual(1, 1);
            centroBordes(1, 2) = centroBordesActual(2, 1);


            //primer punto
            puntosDecapar.GetData(VectorAuxiliar, 8);
            PixelColumna = pixelReferencia0 - VectorAuxiliar[1] * 504 /
330;

            VectorCamara[0] = float(PulsoTramoFinal + pointCount) *
R_pulso_mm;
            VectorCamara[1] = ObtenerY(VectorAuxiliar[0],
PixelColumna);
            VectorCamara[2] = ObtenerZ(PixelColumna);
                        for (int l = 0; l < 4; ++l) {
                            for (int d = 0; d < 4; ++d) {
                                VectorLaser1(l) += MT[l][d] *
VectorCamara[d];
                            }
                        }
            VectorLaser1[0] = VectorLaser1[0] + DPM_X(VectorLaser1[0],
VectorLaser1[1]);
            VectorLaser1[1] = VectorLaser1[1] + DPM_Y(VectorLaser1[0],
VectorLaser1[1]);

            //segundo punto
            PixelColumna = pixelReferencia0 - VectorAuxiliar[5] * 504 /
330;
```

```
            VectorCamara[1]= ObtenerY(VectorAuxiliar[4], PixelColumna);
            VectorCamara[2]= ObtenerZ(PixelColumna);
            for (int l = 0; l < 4; ++l) {
                            for (int d = 0; d < 4; ++d) {
                                VectorLaser2(l) +=MT[l][d] *
VectorCamara[d];
                            }
                        }
            VectorLaser2[0] = VectorLaser2[0] + DPM_X(VectorLaser2[0],
VectorLaser2[1]);
            VectorLaser2[1] = VectorLaser2[1] + DPM_Y(VectorLaser2[0],
VectorLaser2[1]);

            // Agregar el vector como una nueva fila a la matriz
            MatrizPuntos->col(columncount) = VectorLaser1;
            try {
                MatrizPuntos->col(columncount + 1) = VectorLaser2;
            }
            catch (const std::out_of_range& e) {
                std::cout << "Se ha salido del tamaño de la matriz: "
<< e.what() << std::endl;
            }
            catch (const std::exception& e) {
                std::cout << "Se ha producido una excepción: " <<
e.what() << std::endl;
            }
            catch (...) {
                std::cout << "Se ha producido una excepción
desconocida." << std::endl;
            }

            respuesta = CrearPolinomial(MatrizPuntos, &ArchivoGuardado,
&pointCount, &potencia, &frecuencia, &resolucion, &velocidad,
&repeticiones, &desenfoque, &gridSize, &debug);
            VectorLaser1[0] = 0;
            VectorLaser1[1] = 0;
            VectorLaser1[2] = 0;
            VectorLaser2[0] = 0;
            VectorLaser2[1] = 0;
            VectorLaser2[2] = 0;

        }
        cloud->clear();
```

```
        DrawBuffer(dispId0, Fg_getImagePtrEx(fg, last_pic_nr, camPort,
memHandle), static_cast<int>(last_pic_nr), "Cámara Lasa");
        currentPicture = currentPicture + 1;


    }

    // Finalizar la biblioteca de componentes de MATLAB Compiler
Runtime
    LAM_DCTerminate();
    CrearXML3DdefTerminate();
    AdaptarPuntosLAMTerminate();
    // Finalizar la aplicación de MATLAB Compiler Runtime
    mclTerminateApplication();

    //... populate cloud
    pcl::visualization::PCLVisualizer::Ptr viewer(new
pcl::visualization::PCLVisualizer("3D Viewer"));
    viewer->setBackgroundColor(0, 0, 0);

    viewer->addPointCloud<pcl::PointXYZ>(cloud, "cloud");

    while (!viewer->wasStopped())
    {
        viewer->spinOnce(100);
    }

    CloseDisplay(dispId0);
    Fg_stopAcquire(fg, camPort);
    Fg_FreeMemEx(fg, memHandle);
    Fg_FreeGrabber(fg);

    return FG_OK;
}
```

VALENCIA, SEPTIEMBRE 2024

Javier Gamir Artesero



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Calibración y puesta en marcha de un sistema de decapado continuo con láser

ANEXO Nº2: BIBLIOGRAFÍA

Máster Universitario en Automática e Informática Industrial

Autor

Javier Gamir Artesero

Tutor

Carlos Ricolfe Viala

CURSO ACADÉMICO: 2023/2024

1. *Decapado láser: ¡toda la información!* | *Cryoblaster®*. (s. f.). Recuperado 2 de septiembre de 2024, de https://cryoblaster.com/es/decapado-laser/

2. Franca, J. G. D. M., Gazziro, M. A., Ide, A. N., & Saito, J. H. (2005). A 3D scanning system based on laser triangulation and variable field of view. *IEEE International Conference on Image Processing 2005*, *1*, I-425. https://doi.org/10.1109/ICIP.2005.1529778

3. *Laser triangulation*. (s. f.). Teledyne E2v Imaging. Recuperado 2 de septiembre de 2024, de https://imaging.teledyne-e2v.com/products/applications/3d-imaging-solutions/laser-triangulation/

4. lesagegp. (2013, diciembre 4). Laser scanning explained. *G. P. Le Sage Blog*. https://lesagegp.wordpress.com/2013/12/04/laser-scanning-explained/

5. *¿Para qué sirve el decapado por láser?* (s. f.). Recuperado 2 de septiembre de 2024, de https://www.trumpf.com/es_ES/soluciones/aplicaciones/tratamiento-de-superficies-con-el-laser/decapado-por-laser/

6. *Principle of triangulation*. (s. f.). Recuperado 2 de septiembre de 2024, de https://www.vision-doctor.com/en/laser-illumination/principle-of-triangulation.html

7. Resolución de las interferencias multipath en las cámaras ToF. (s. f.). *Innotransfer*. Recuperado 2 de septiembre de 2024, de https://innotransfer.org/eoi/resolucion-de-las-interferencias-multipath-en-las-camaras-tof/

8. *Robot industrial FANUC LRMate 200iD*. (s. f.). Recuperado 2 de septiembre de 2024, de https://www.fanuc.eu/es/es/robots/página-filtro-robots/serie-lrmate/lrmate-200-id

9. Schonhardt, B. (2023, marzo 30). *Fundamentals of industrial image processing and laser technology*. QuellTech. https://quelltech.de/en/image-processing-laser-triangulation/

10. *Setting up c++ development environment—Matlab & simulink—Mathworks españa*. (s. f.). Recuperado 2 de septiembre de 2024, de https://es.mathworks.com/help/compiler_sdk/cxx/cpp-development-environment.html

11. Spurgeon, W. (s. f.-a). *Tiempo de vuelo: Afrontando desafíos 3d*. Recuperado 2 de septiembre de 2024, de https://www.clearview-imaging.com/es/blog/tiempo-de-vuelo-afrontando-desafíos-3d

12. Spurgeon, W. (s. f.-b). *Visión Estéreo 3D para Aplicaciones de Visión Artificial*. Recuperado 2 de septiembre de 2024, de https://www.clearview-imaging.com/es/blog/visión-estéreo-3d-para-aplicaciones-de-visión-articicial

13. Zivid. (s. f.). *3D vision technology principles*. Recuperado 2 de septiembre de 2024, de https://www.zivid.com/3d-vision-technology-principles

14. Antonio Sánchez Salmerón. *Tema 12 Visión Por Computador en la Industria* [Diapositivas de PowerPoint]. DISA, Universidad Politécnica de Valencia. Diapositivas de 20 a 25.

15. *Macsa id—Coding, marking and tracing solutions worldwide*. (2017, febrero 1). https://www.macsa.com/en/

16. Macsa id. *Guía usuario uso de bibliotecas para comunicación TCP.*

17. *Armadillo: C++ library for linear algebra & scientific computing*. (s. f.). Recuperado 2 de septiembre de 2024, de https://arma.sourceforge.net/

18. *Get started with matlab compiler—Mathworks españa*. (s. f.). Recuperado 2 de septiembre de 2024, de https://es.mathworks.com/help/compiler/getting-started-with-matlab-compiler.html

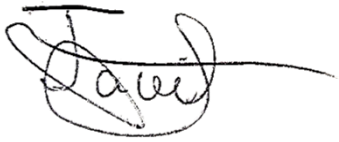
19. *How to integrate matlab generated dll into visual studio /c++*. (s. f.). Recuperado 2 de septiembre de 2024, de https://es.mathworks.com/matlabcentral/answers/72248-how-to-integrate-matlab-generated-dll-into-visual-studio-c

20. *MatLab C++ Shared Dll library initialization problem*. (s. f.). Recuperado 2 de septiembre de 2024, de https://es.mathworks.com/matlabcentral/answers/286624-matlab-c-shared-dll-library-initialization-problem

21. *Tipo de ajuste para curvas y superficies—MATLAB fittype—MathWorks España*. (s. f.). Recuperado 2 de septiembre de 2024, de https://es.mathworks.com/help/curvefit/fittype.html

VALENCIA, SEPTIEMBRE 2024

Javier Gamir Artesero



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Calibración y puesta en marcha de un sistema de decapado continuo con láser

ANEXO Nº3: CONFIGURACIÓN DE COMPONENTES

Máster Universitario en Automática e Informática Industrial

Autor

Javier Gamir Artesero

Tutor

Carlos Ricolfe Viala

CURSO ACADÉMICO: 2023/2024

Índice

1 Cámara 2D Mikotron EoSens CL color y Framegrabber

Para la cámara disponemos de una aplicación donde podemos configurarla y para el framegrabber disponemos de otra, comenzando por la de la cámara la aplicación a utilizar es MC Control Tool:

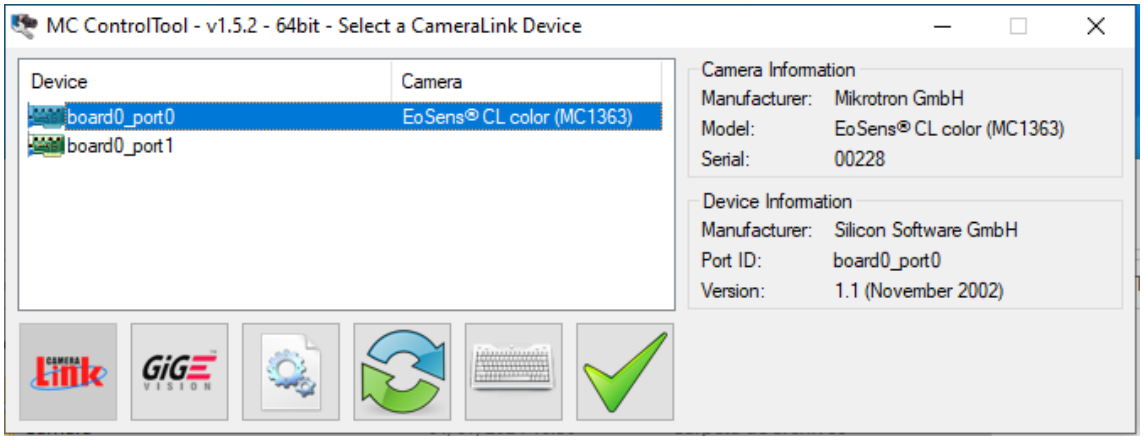


Figura 1.1 MC Control Tool Pantalla de Selección

En esta pantalla inicial se nos presentan las conexiones presentes al ordenador, en la figura presentada aparece la cámara escogida y los dos puertos del framegrabber al que se encuentra conectada.

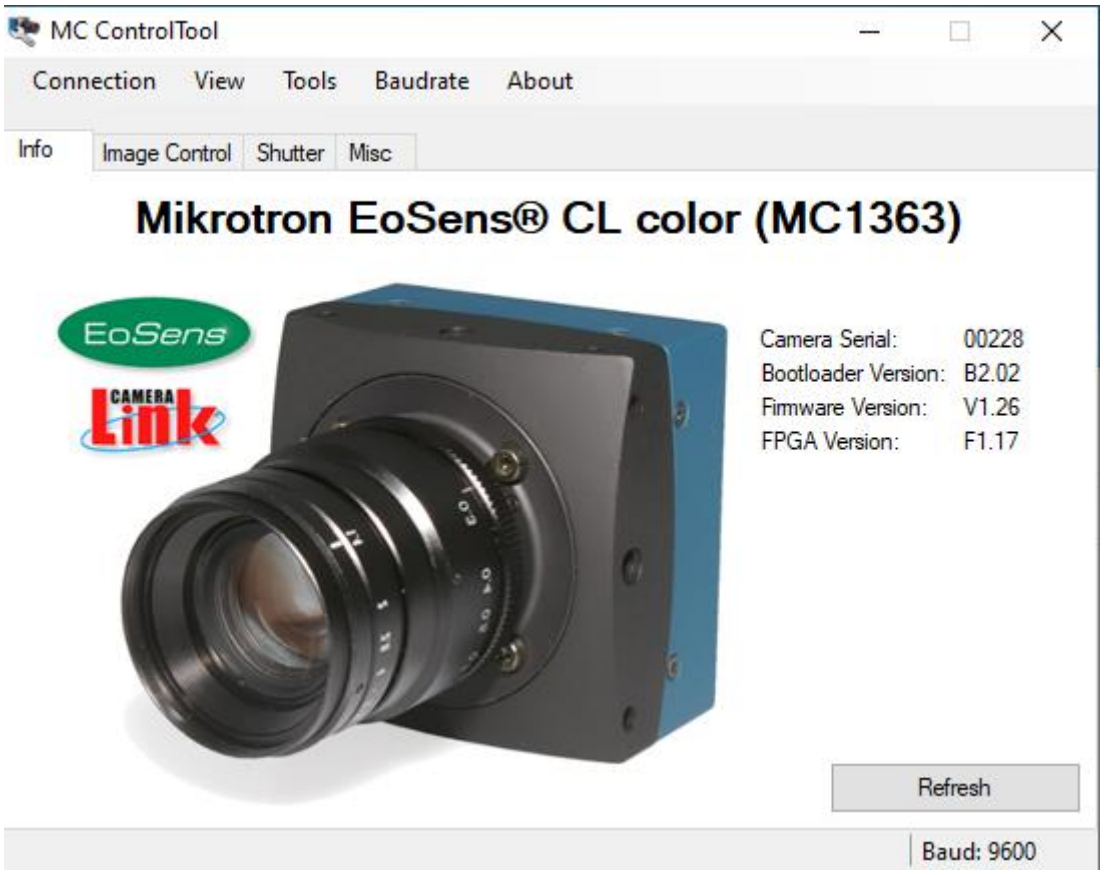


Figura 1.2 MC Control Tool Pantalla de Presentación de la Cámara Escogida

En esta pantalla se nos presenta información acerca de la cámara escogida.

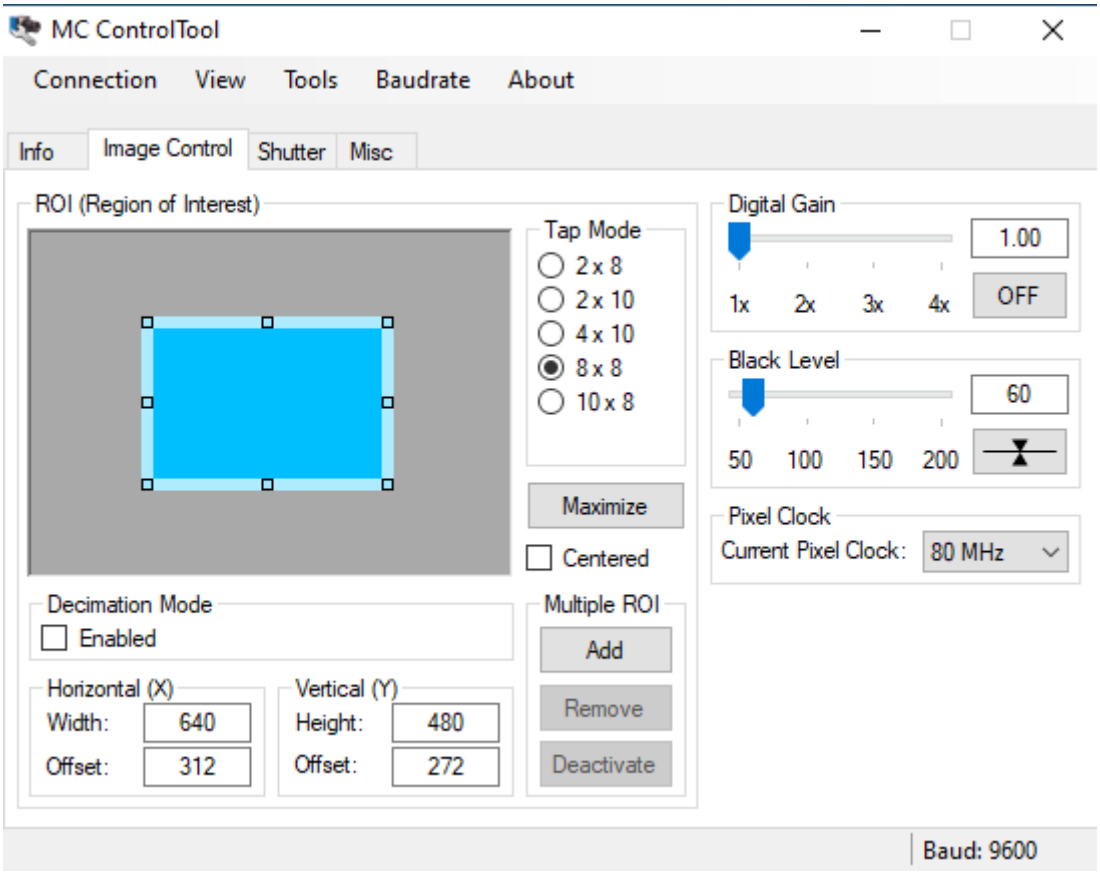


Figura 1.3 MC Control Tool Pantalla de Control de la Imagen

En la figura superior se muestra dónde podemos cambiar el ROI de la cámara, su ganancia digital, la configuración del CCD en “Tap Mode”, el nivel de negro, el reloj de la cámara y si se desea añadir distintos ROIs.

Para el proyecto se ha escogido utilizar toda la capacidad de la cámara, siendo esta 1280x1024, el Tap Mode en 8x8, y no se ha cambiado nada del resto de características que se presentan en la figura.

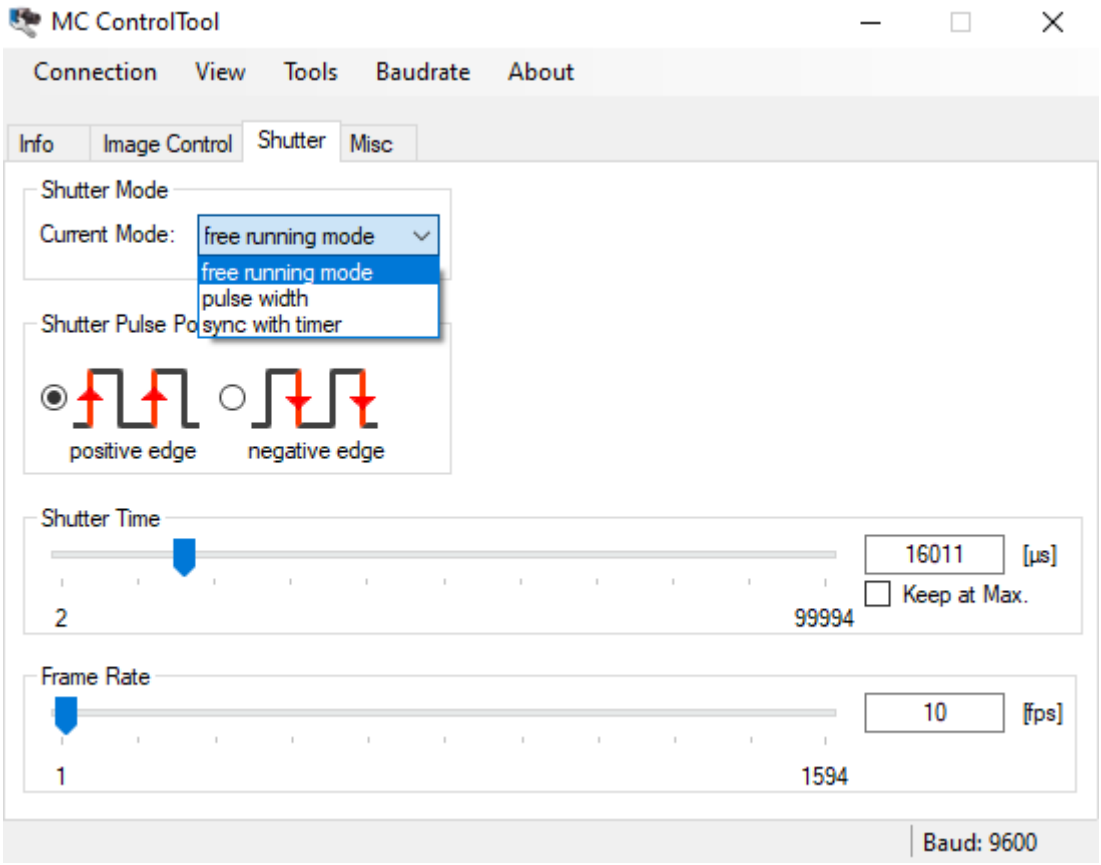


Figura 1.4 MC Control Tool Pantalla de Disparo

Mediante la pantalla de disparo podemos escoger entre los tres distintos modos de disparo de la cámara, el método libre que permite que la cámara dispare con una tasa de fps seleccionada y un tiempo de disparo elegido; un modo de disparo que relaciona estas dos selecciones con el ancho del pulso recibido en el framegrabber y un tercer modo que permite disparar cuando se recibe un pulso, pero se puede escoger el tiempo de captura de la cámara.

Para el proyecto junto el encoder se ha escogido utilizar el tercer modo, “Sync with Timer” para poder escoger el tiempo de captura, el cual deberá de ser escogido correctamente para que no haya retrasos en la misma.

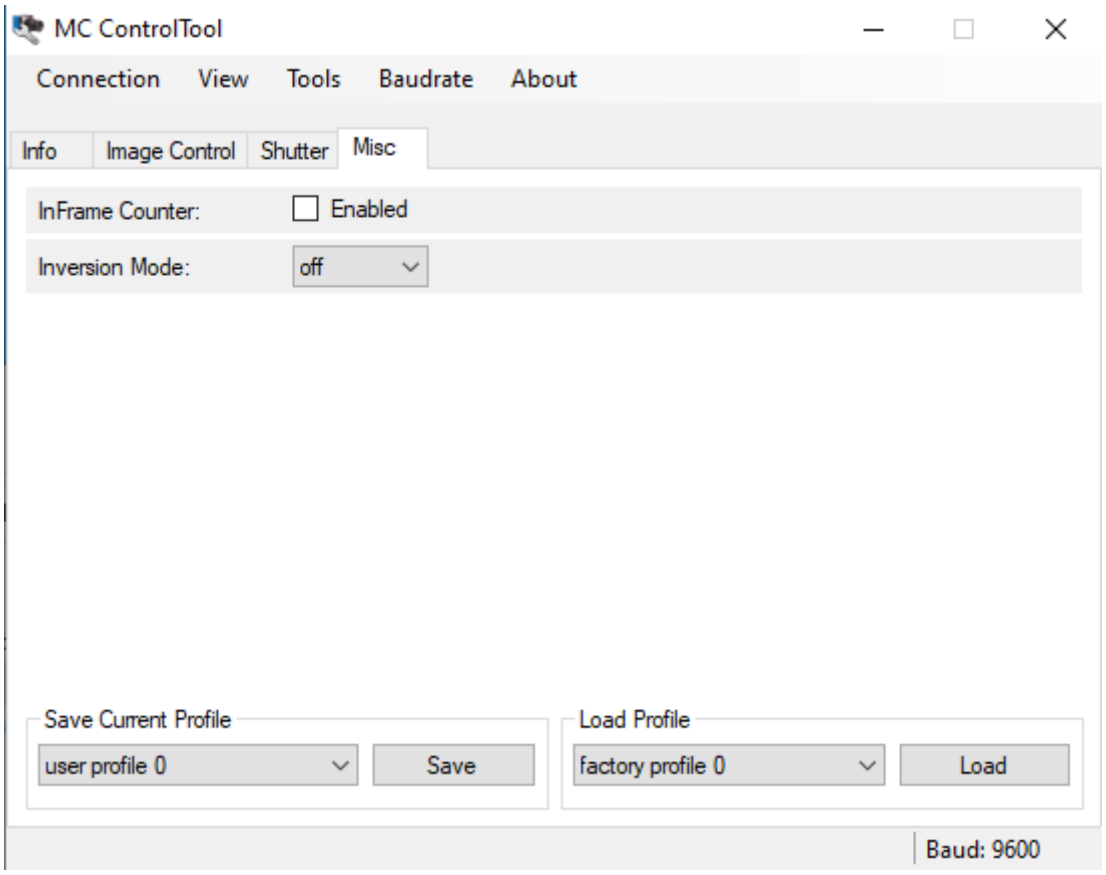


Figura 1.5 MC Control Tool Última Pantalla

En esta pantalla se pueden almacenar configuraciones, invertir el modo y seleccionar un contador de frames interno. No se ha empleado nada de lo presente.

Respecto al framegrabber:

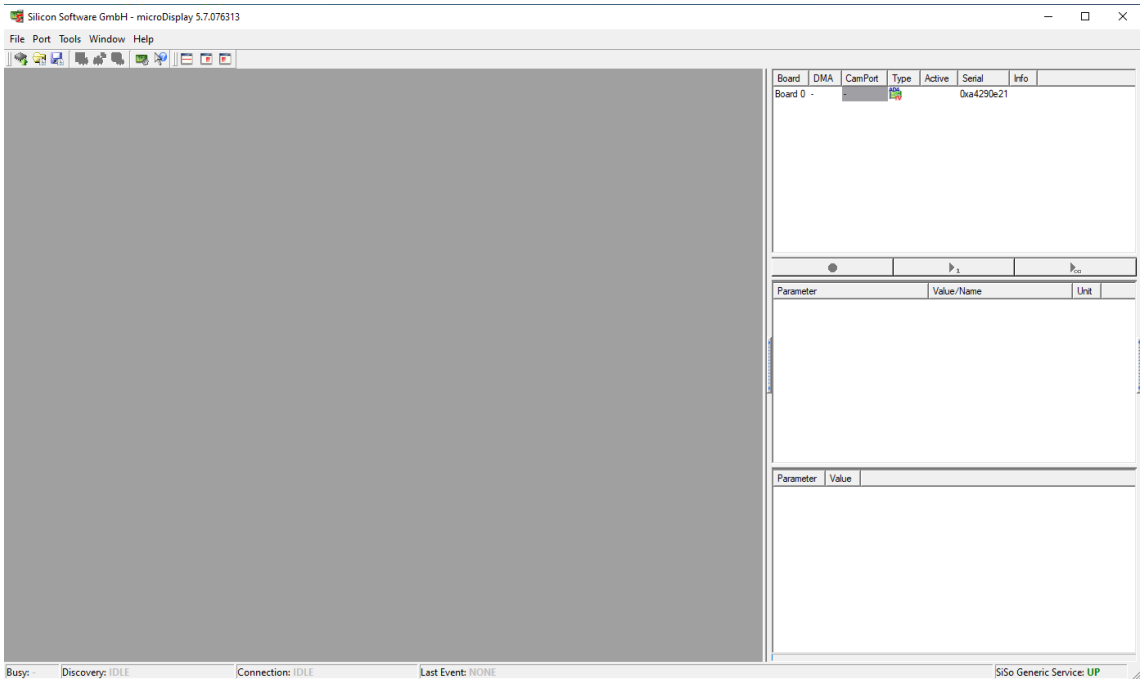


Figura 1.6 Pantalla General MicroDisplay

La aplicación empleada para el framegrabber es MicroDisplay y la figura superior muestra la pantalla principal de la misma.

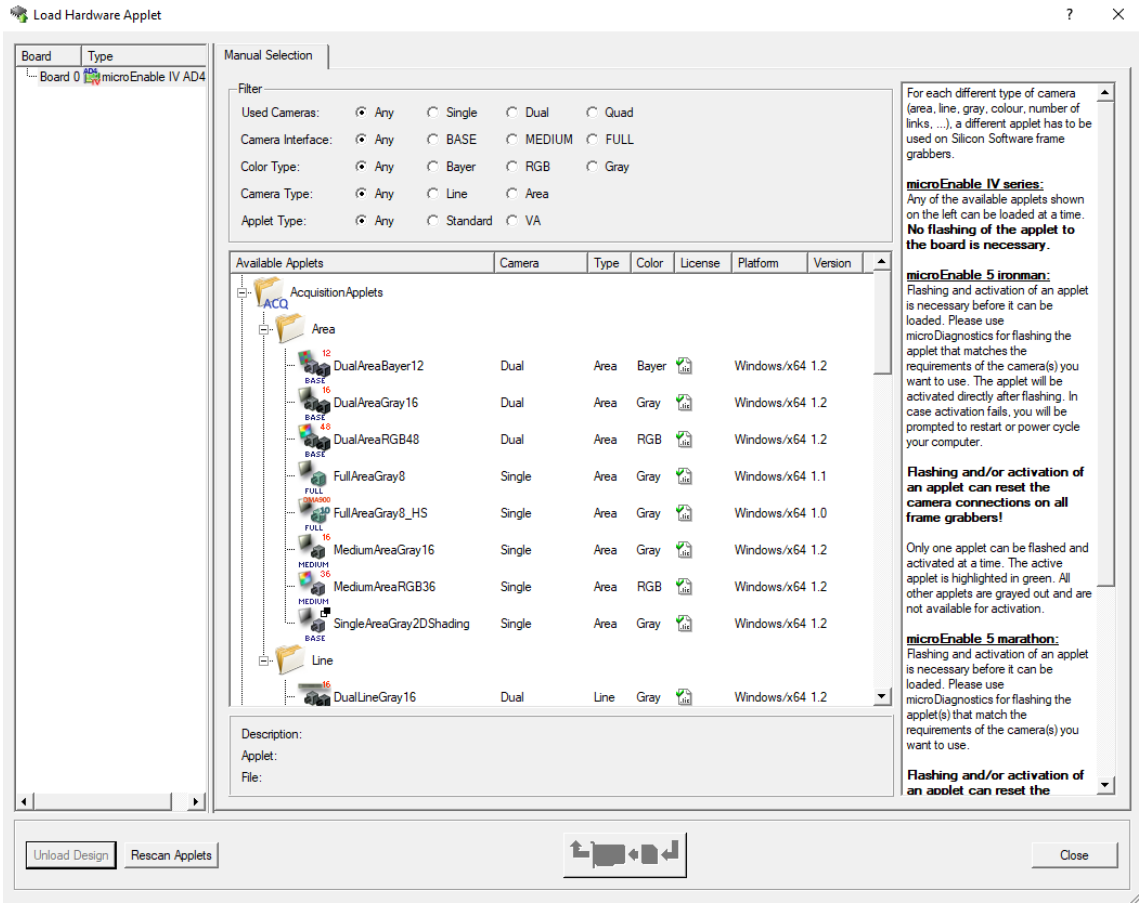


Figura 1.7 MicroDisplay Panatlla de Selección de Applets

Mediante esta selección es posible escoger el método de captura del framegrabber para la cámara, sin entrar mucho en detalle, se ha escogido el método de FullAreaGray8, ya que el método escogido del CCD previamente era 8x8, y la cámara es de blanco y negro y de Área.

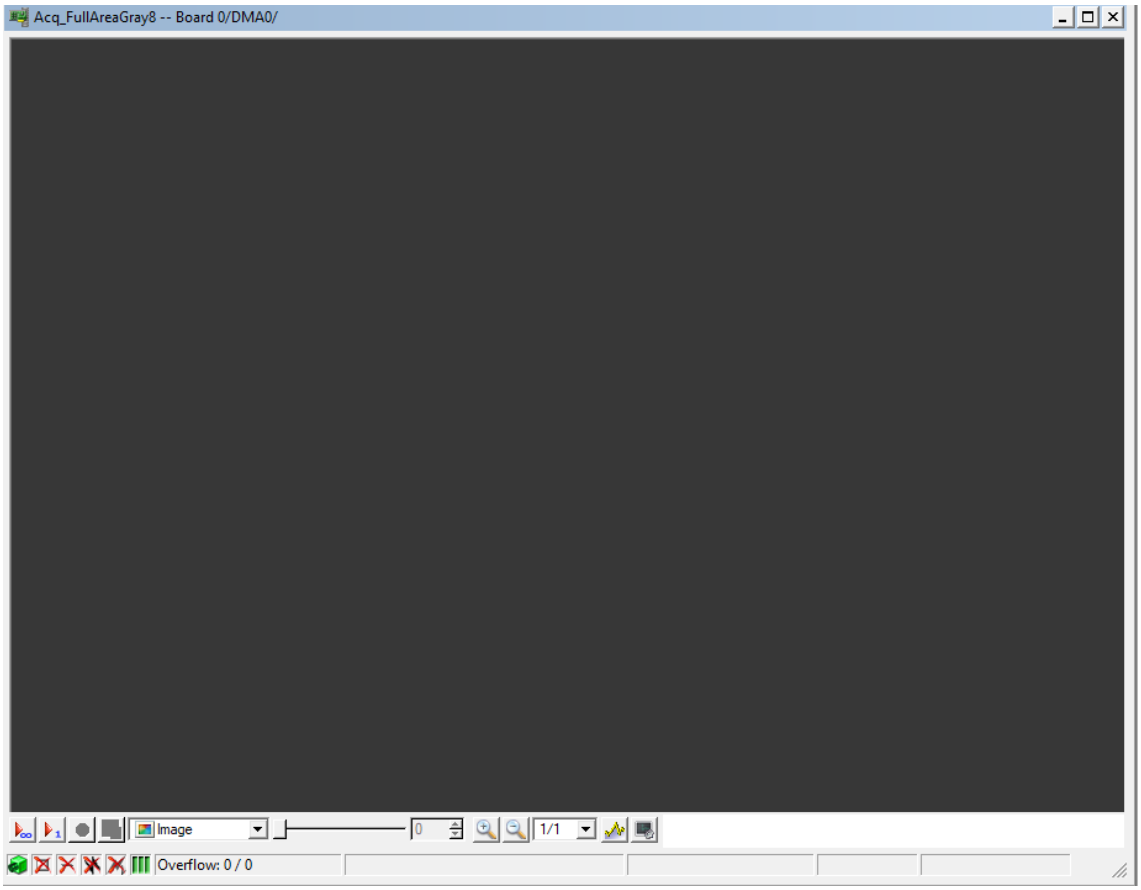


Figura 1.8 MicroDisplay Pantalla De Imagen

Es en esta sección de la pantalla principal donde se puede ver lo que ve la cámara, de forma continua o imagen a imagen según que se desee, y que permite almacenar imágenes vistas.

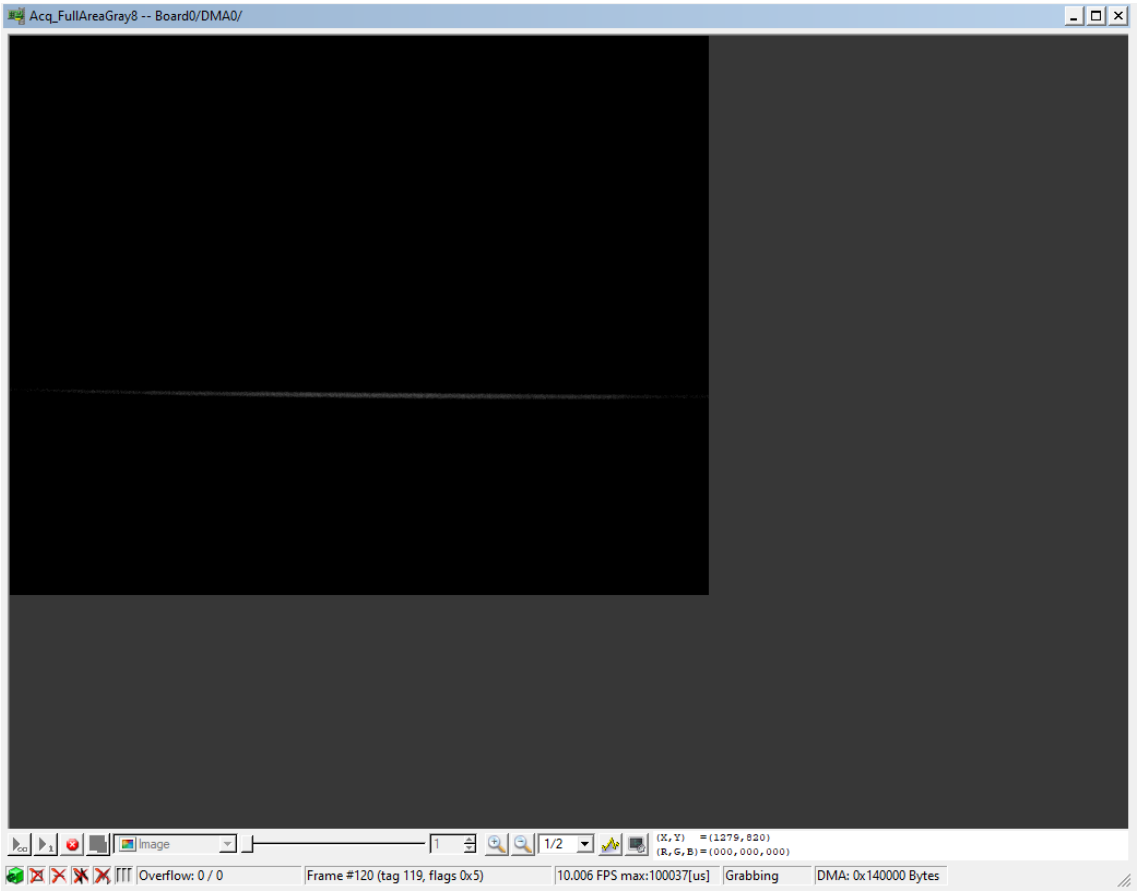


Figura 1.9 Ejemplo de Imagen

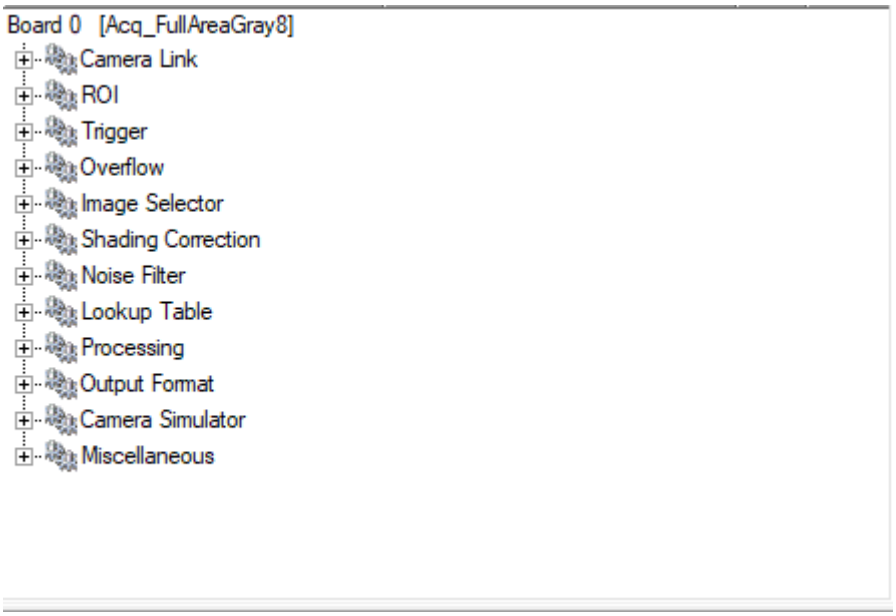


Figura 1.10 MicroDisplay Configuración

En la figura superior se muestran los distintos parámetros que se pueden modificar de la adquisición del framegrabber. Los importantes para el proyecto es la selección del ROI y la selección del disparo para capturar.

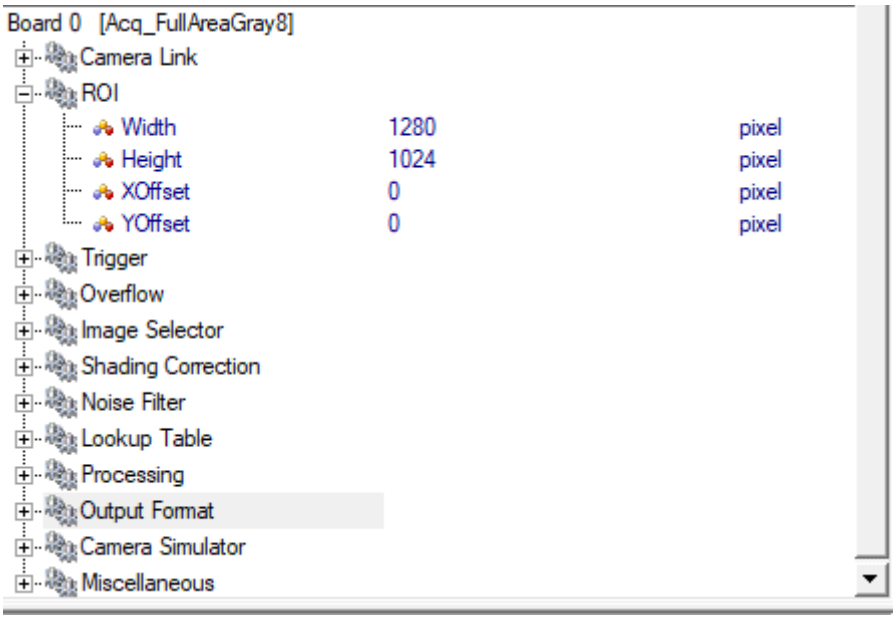


Figura 1.11 Configuración del ROI de captura

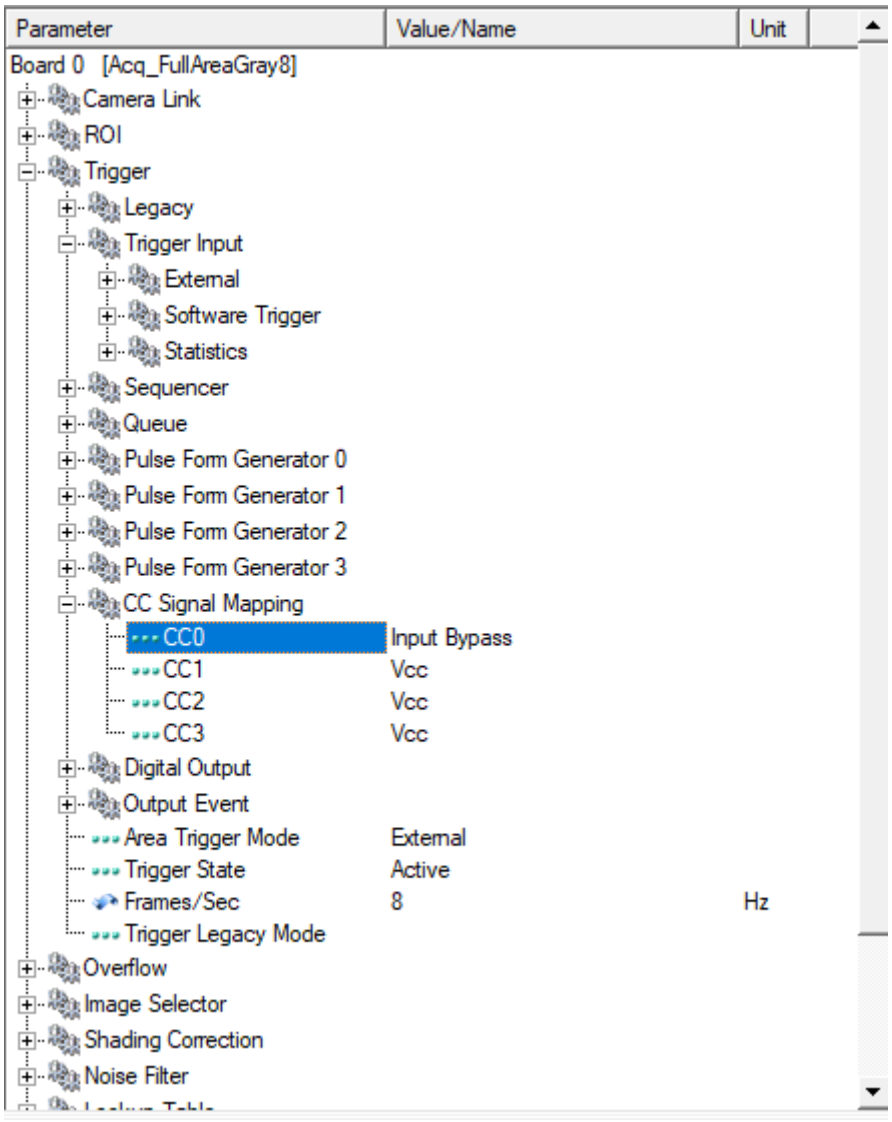


Figura 1.12 Configuración del Disparo

Para que el framegrabber capture con la señal recibida del encoder se debe configurar el pin CC al que este está conectado como “Input Bypass” y el método de “área Trigger Mode” como externa.

2 Láser F-9000 DUO MOPA

El láser ha debido de ser configurado correctamente para que este detecte el encoder y corrija la posición de decapado a medida que la pieza se mueva de forma que decape en la posición que se la ha suministrado.

Modo:	Estático-dynámico
Distancia (mm):	2200.0
Autoshift:	✔
Autosort:	✔
Autoexplotar:	✔
N-prints:	0
Fotocelula tipo:	Automático
Señal:	Trigger Up
Señal de paro:	No
Nivel de señal de paro:	Desactivado
Parar impresión actual:	No
Permitir trigger multiple:	✘
Senñal de Triggerenable:	No
Nivel de señal de triggerenable:	Desactivado
Trigger filtro [ms]:	0.0
Trigger FIFO [0..64]:	0
Offset (mm):	0.0
Encoder tipo:	Externo
A+B canal:	✘
Dirección:	Derecha
Señal de dirección	No
Invertir señal de dirección:	✘
Encoder posición:	Posición 2
Pulsos por vuelta:	2000
mm por vuelta:	180.000

N-prints:	0
Fotocelula tipo:	Automático
Señal:	Trigger Up
Señal de paro:	No
Nivel de señal de paro:	Desactivado
Parar impresión actual:	No
Permitir trigger multiple:	✘
Señal de Triggerenable:	No
Nivel de señal de triggerenable:	Desactivado
Trigger filtro [ms]:	0.0
Trigger FIFO [0..64]:	0
Offset (mm):	0.0
Encoder tipo:	Externo
A+B canal:	✘
Dirección:	Derecha
Señal de dirección	No
Invertir señal de dirección:	✘
Encoder posición:	Posición 2
Pulsos por vuelta:	2000
mm por vuelta:	180.000
Modo rotatorio:	✘
Velocidad(m/min):	0.121
Retraso sistemático (usec):	0
Min. pulsos:	10
Tiempo de validez (ms):	1

Figura 2.1 Configuración Láser Decapado

El modo de decapado establece la forma en la que el láser inicia el decapado y si ha de tener en cuenta fuentes externas, para el proyecto se han utilizado dos métodos:

-Estático: El láser decapa en las coordenadas marcadas sin ningún tipo de movimiento (este método se ha empleado para un estilo de decapado línea a línea)

-Estático-Dinámico: El láser comienza decapando en las coordenadas marcadas y se desplaza por cada pulso recibido los milímetros marcados en su configuración (este método se emplea en el decapado continuo ya que la pieza se desplaza mientras se decapa un sector).

Además del modo de decapado son también importantes las siguientes configuraciones:

-Encoder Tipo: Se ha seleccionado externo ya que el láser es capaz de emplear un encoder de tipo interno, pero la señal es de tipo externo.

-A+B canal: Es posible usar dos canales para el encoder, pero solo se ha empelado uno, por lo que se marca con una ‘X’.

-Dirección: Se establece que la pieza se desplazará hacia la derecha del láser.

-Encoder Posición: Se ha de seleccionar entre 1 y 2 de forma que coincida con la dirección escogida, sino se invierte.

-Pulsos por vuelta: El número de pulsos que el encoder tiene por vuelta.

-mm por vuelta: Cuantos mm se recorren por cada vuelta de encoder, esto permite calcular junto a pulsos por vuelta, cuantos mm equivalen a un pulso.

Las conexiones al láser con el encoder son de tipo TTL y vienen dadas por el siguiente esquema:

Figura 2.2 Conector Pin Encoder

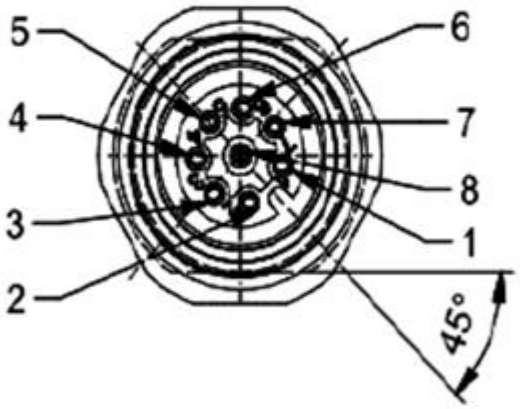


Figura 2.3 Esquema del Conector Pin Encoder

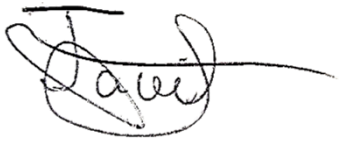
Número de Pin	Función	Color
#1	Tierra	Negro
#2	VCC +5V	Rojo
#3	-	-
#4	Salida B	Verde
#5	/Salida B	Azul
#6	Salida A	Amarillo
#7	/Salida A	Marrón
#8	-	-

Figura 2.4 Pin-Out Conector Pin Encoder

Se han conectado a los pines 4 y 5 que pertenecen al canal B.

VALENCIA, SEPTIEMBRE 2024

Javier Gamir Artesero



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Calibración y puesta en marcha de un sistema de decapado continuo con láser

ANEXO Nº4: FICHAS DE DATOS

Máster Universitario en Automática e Informática Industrial

Autor

Javier Gamir Artesero

Tutor

Carlos Ricolfe Viala

CURSO ACADÉMICO: 2023/2024

Índice

1 LR Mate 200iD

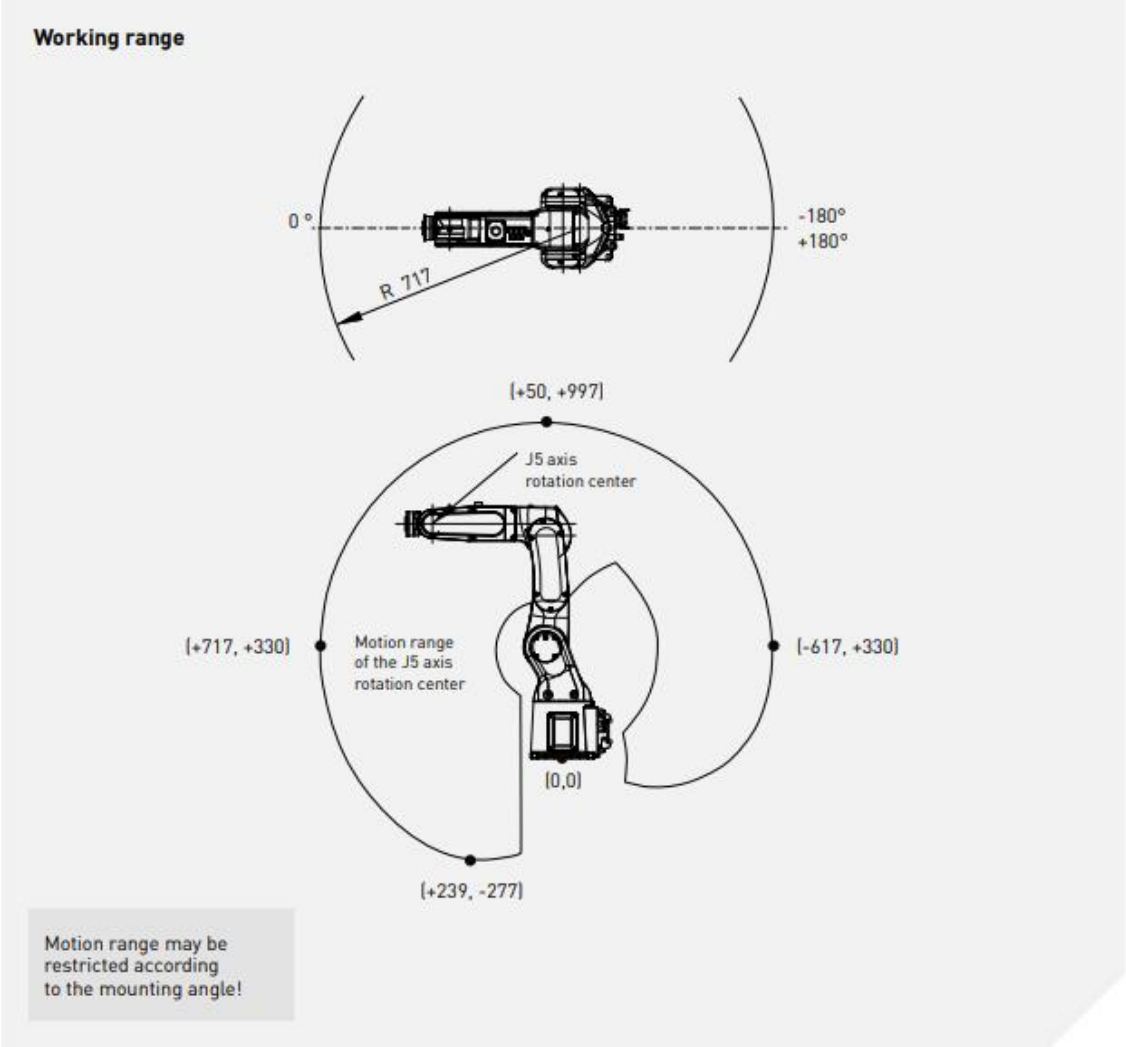


Figura 1.1 Rango de Trabajo del Robot Fanuc

LR Mate 200*i*D

Max. load capacity at wrist: **7 kg**

Max. reach: **717 mm**

Controlled axes	Repeatability (mm)	Mechanical weight (kg)	Motion range (°)							Maximum speed (°/s)							J4 Moment/ Inertia (Nm/kgm²)	J5 Moment/ Inertia (Nm/kgm²)	J6 Moment/ Inertia (Nm/kgm²)
			J1	J2	J3	J4	J5	J6	E1	J1	J2	J3	J4	J5	J6	E1			
6	± 0.01*	25	360	245	420	380	250	720	-	450	380	520	550	545	1000	-	16.6/0.47	16.6/0.47	9.4/0.15

Figura 1.2 Ejes del manipulador

Robot	**LR Mate 200*i*D**
Robot footprint [mm]	190 x 190
Mounting position Floor	●
Mounting position Upside down	●
Mounting position Angle	●
Controller	**R-30*i*B Plus**
Open air cabinet	○
Mate cabinet	●
A-cabinet	-
B-cabinet	-
*i*Pendant Touch	●
Electrical connections	
Voltage 50/60Hz 3phase [V]	-
Voltage 50/60Hz 1phase [V]	200-230
Average power consumption [kW]	0.5
Integrated services	
Integrated signals on upper arm In/Out	6/2
Integrated air supply	1
Environment	
Acoustic noise level [dB]	64.7
Ambient temperature [° C]	0-45
Protection	
Body standard/optional	IP67/IP69K
Wrist & J3 arm standard/optional	IP67/IP69K
Clean room (ISO Class 4)	-

Figura 1.3 Robot Fanuc Características

2 Mikrotron EO Sens CL

Technical Data

(More detailed specifications are available on request)

	EoSens® CL (monochrome / color)
Resolution	1.3 Mpix
Active pixels	1,280 x 1,024 px
Interface	Camera Link® Full
Frame rate (8 bit)	506 fps
Sensor	LUPA1300-2
Sensor type	CMOS global shutter
Sensor format	4/3'
Active sensor area (H x V)	17.92 x 14.34 mm
Pixel size	14 x 14 µm
Sensitivity (mono)	10.2 V/lux*s @ 550nm
Color depth	10 / 8 bit
Dynamic range	57 dB / up to 90 dB
Shutter time (steps)	2 µs
Shutter time range	2 µs – 1 s
Mount options	C-Mount / F-Mount
Dimensions (W x H x L w/o mount)	63 x 63 x 38 mm
Weight (C-Mount)	300 g
Power consumption	5 W
Power supply	8 – 24 V DC
Camera body temperature	+5 °C ... +50 °C
Shock / Vibration proof	70 g / 7 grms
Conformity	CE / RoHS / Camera Link®
EMVA 1288 reports	✓

Figura 2.1 Características de la cámara

3 Framegrabber Micro Enable IV AD4-CL

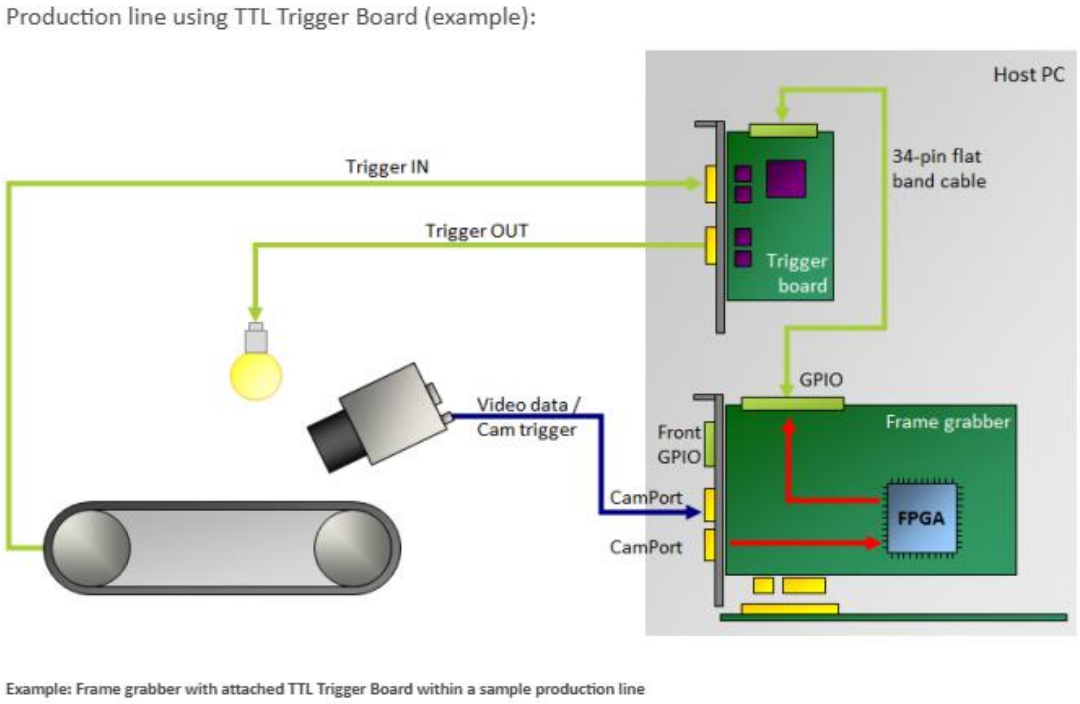


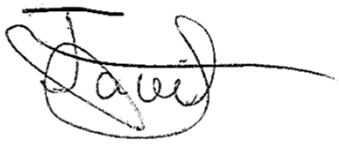
Figura 3.1 Ejemplo de uso de Framegrabber

1	Trigger Output 0 Port A *Flash Signal*	+3.3V	2
3	Trigger Output 1 Port A *For Area Cameras: Image Trigger (ExSync)* *For Line Cameras: Line Trigger (ExSync2)*	+3.3V	4
5	Trigger Output 2 Port A *For Area Cameras: HD Sync* *For Line Cameras: Line Trigger (ExSync)*	GND	6
7	Trigger Output 3 Port A *User Output (Digital Out Bit #0)*	GND	8
9	Trigger Input 0 Port A	GND	10
11	Trigger Input 1 Port A	GND	12
13	Trigger Input 2 Port A	GND	14
15	Trigger Input 3 Port A	GND	16
17	Trigger Output 4 Port B *Flash Signal*	GND	18
19	Trigger Output 5 Port B *For Area Cameras: Image Trigger (ExSync)* *For Line Cameras: Line Trigger (ExSync2)*	GND	20
21	Trigger Output 6 Port B *For Area Cameras: HD Sync* *For Line Cameras: Line Trigger (ExSync)*	GND	22
23	Trigger Output 7 Port B *User Output (Digital Out Bit #1)*	GND	24
25	Trigger Input 4 Port B	GND	26
27	Trigger Input 5 Port B	GND	28
29	Trigger Input 6 Port B	GND	30
31	Trigger Input 7 Port B	VCCIO (+2.5V)	32
33	Presence Detect	VCCIO (+2.5V)	34

Figura 3.2 Pin Layout Framegrabber

VALENCIA, SEPTIEMBRE 2024

Javier Gamir Artesero



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Calibración y puesta en marcha de un sistema de decapado continuo con láser

ANEXO Nº5: RELACIÓN DEL TRABAJO CON LOS OBJETIVOS DE DESARROLLO SOSTENIBLE DE LA AGENDA 2030

Máster Universitario en Automática e Informática Industrial

Autor

Javier Gamir Artesero

Tutor

Carlos Ricolfe Viala

CURSO ACADÉMICO: 2023/2024

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				✓
ODS 2. **Hambre cero.**				✓
ODS 3. **Salud y bienestar.**				✓
ODS 4. **Educación de calidad.**				✓
ODS 5. **Igualdad de género.**				✓
ODS 6. **Agua limpia y saneamiento.**				✓
ODS 7. **Energía asequible y no contaminante.**				✓
ODS 8. **Trabajo decente y crecimiento económico.**	✓			
ODS 9. **Industria, innovación e infraestructuras.**	✓			
ODS 10. **Reducción de las desigualdades**.				✓
ODS 11. **Ciudades y comunidades sostenibles**.				✓
ODS 12. **Producción y consumo responsables.**			✓	
ODS 13. **Acción por el clima.**				✓
ODS 14. **Vida submarina.**				✓
ODS 15. **Vida de ecosistemas terrestres.**				✓
ODS 16. **Paz, justicia e instituciones sólidas.**				✓
ODS 17. **Alianzas para lograr objetivos.**			✓	

Descripción de la alineación del TFG/TFM con los ODS

ODS 8. Trabajo decente y crecimiento económico: El proyecto permitirá eliminar una tarea cuya única forma de llevarse a cabo alternativamente a la propuesta es diseñar cunas de granito específicas para cada pieza, suponiendo un coste económico de gran cantidad y poco adaptable a cambios.

Lo recientemente explicado se encuentra alineado con los siguientes subapartados de la ODS 8:

-8.2: Se contribuye mediante innovación y modernización tecnológica.

-8.4: Mejora el consumo eficiente de recursos.

-8.6: Puede crear puestos de trabajo de jóvenes ingenieros.

ODS 9. Industria, innovación e infraestructuras: Impulsa la innovación en la industria al aumentar el número de aplicaciones en ella de elementos de robótica y visión artificial. Los subapartados que se encuentran alineados con el proyecto son:

-9.1: La implementación de una cinta transprotadora y un sistema de visión artificial infiere la necesidad de la creación de una infraestructura avanzada, sostenible y de calidad. Además, impulsa la creación de otras infraestructuras como puede ser de transporte de los productos empaquetados.

-9.4: Reduce el número de recursos utilizados en el proceso.

ODS 12. Producción y consumo responsables: Se reduce el número de cunas de granito cuyo único y exclusivo uso es para el decapado de las piezas, una vez se actualiza una pieza o cambia el modelo la cuna queda inútil. Esto se encuentra en consonancia con los siguientes subapartados:

-12.3: Permite reducir el desperdicio de recursos.

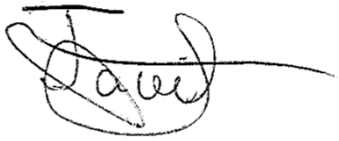
-12.6: Aporta un enfoque sostenible para el proceso de decapado automático.

ODS 17. Alianzas para lograr objetivos: Mediante este proyecto se promueve el intercambio de información entre la Universidad y la empresa SRG.

El subapartado que se encuentra alineado con esto es el 17.6 al mejorar la cooperación regional en materia de ciencia, tecnología e innovación aumentar el intercambio de conocimientos en condiciones mutuamente convenidas

VALENCIA, SEPTIEMBRE 2024

Javier Gamir Artesero



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Calibración y puesta en marcha de un sistema de decapado continuo con láser

DOCUMENTO Nº2: PLANOS

Máster Universitario en Automática e Informática Industrial

Autor

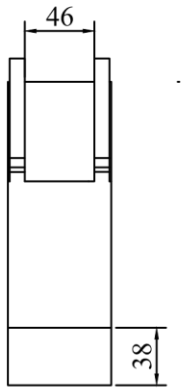
Javier Gamir Artesero

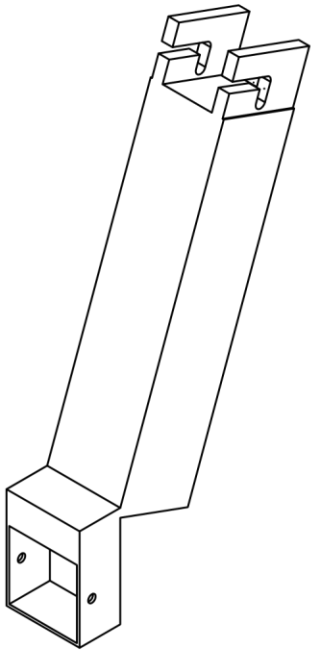
Tutor

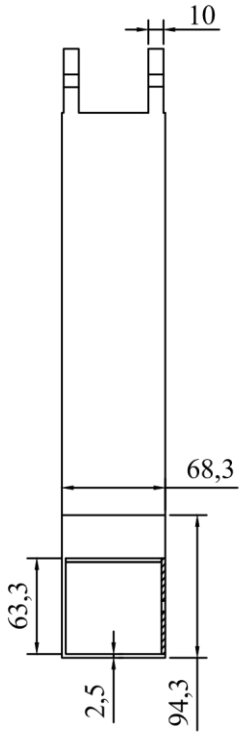
Carlos Ricolfe Viala

CURSO ACADÉMICO: 2023/2024

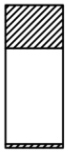
Índice

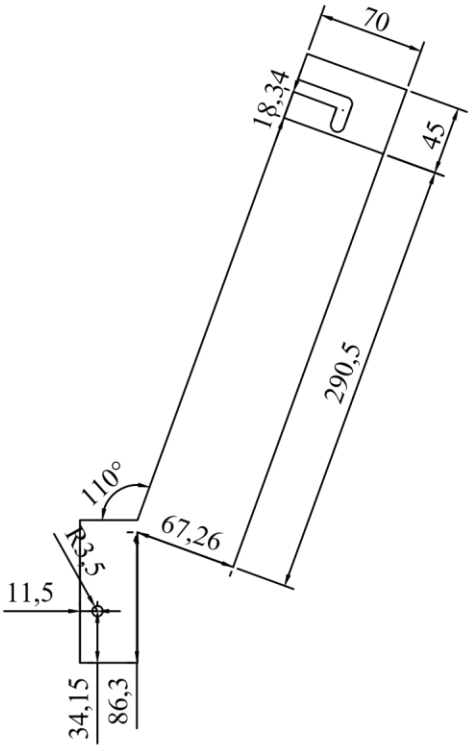




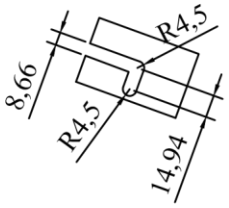


Sección
A-A

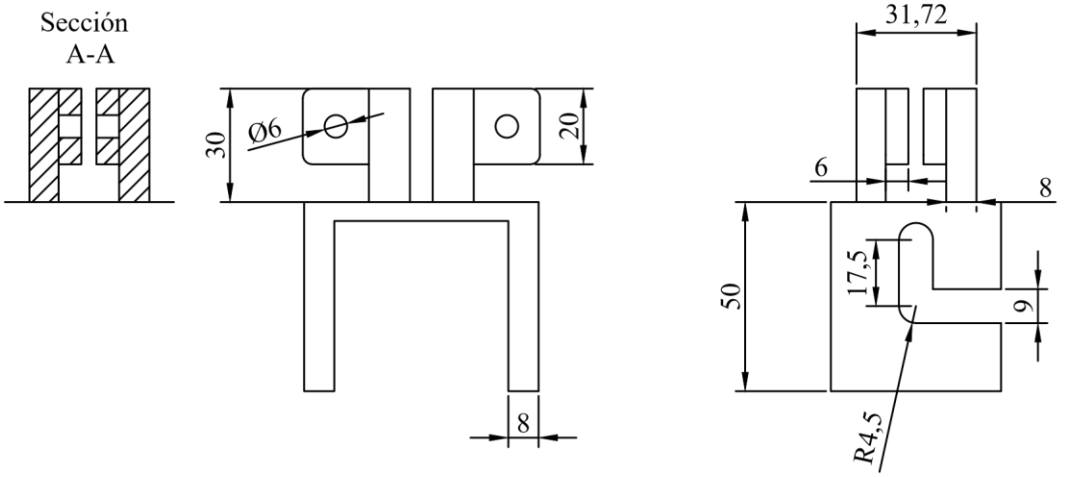


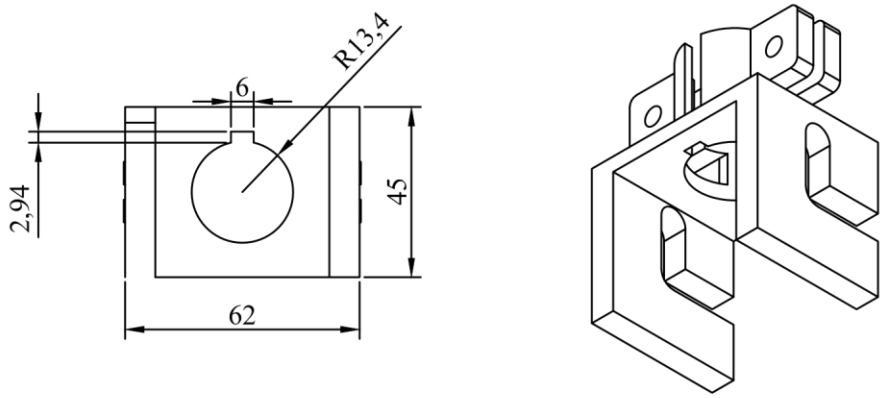


Detalle
A-A

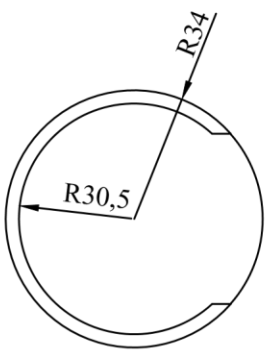


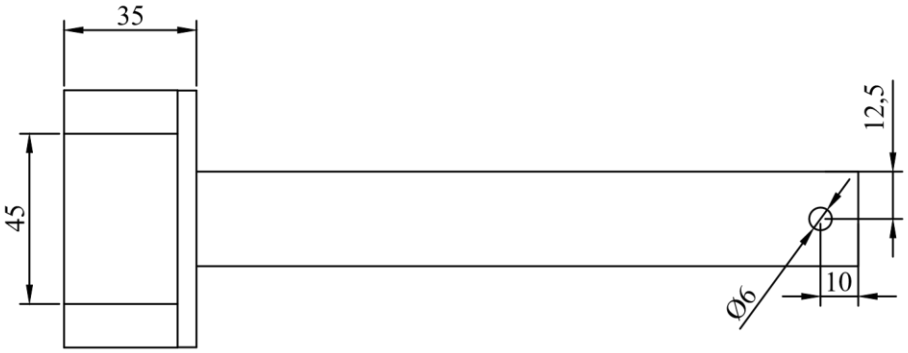
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Titulo del proyecto: Calibración y puesta en marcha de un sistea de decapado continuo con láser Titular: Javier Gamir Artesero		Fecha: 03/09/2024 Escala: 1:5
Autor: Javier Gamir Artesero	Denominación del plano Soporte Cámara Mikotron EoSens CL color	Número del plano 01

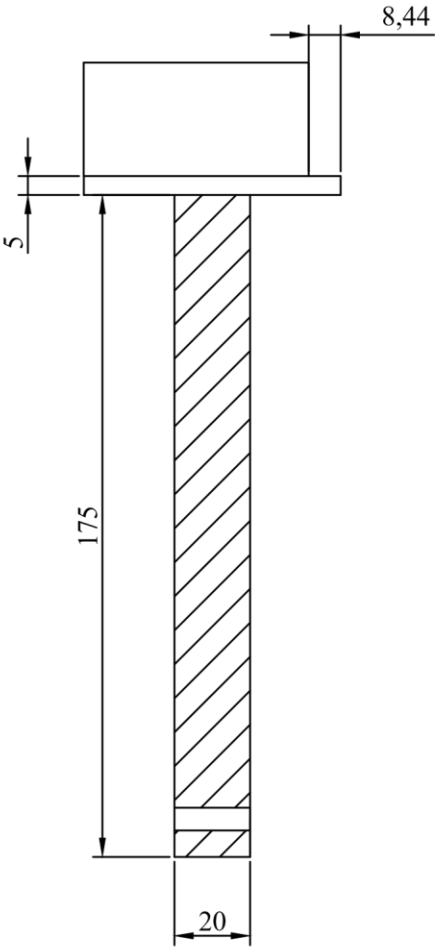


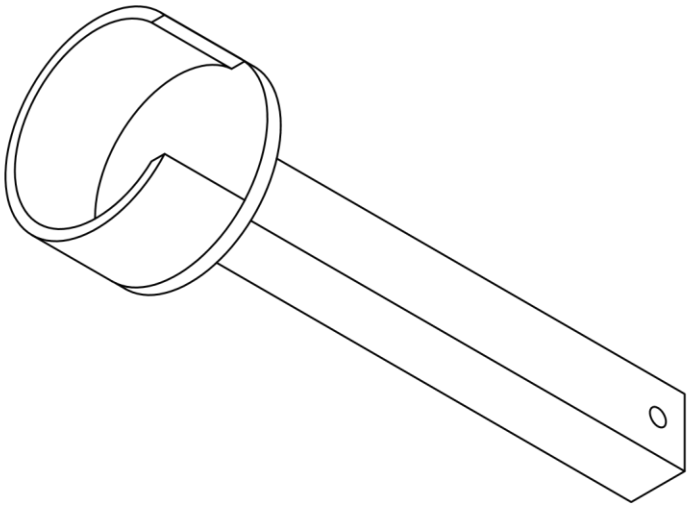


ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Titulo del proyecto: Calibración y puesta en marcha de un sistema de decapado continuo con láser Titular: Javier Gamir Artesero		Fecha: 03/09/2024 Escala: 1:2
Autor: Javier Gamir Artesero	Denominación del plano Soporte Láser	Número del plano 02

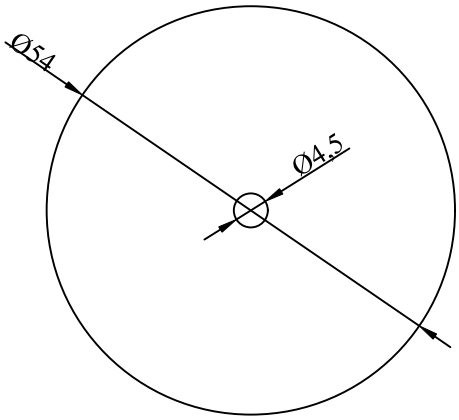


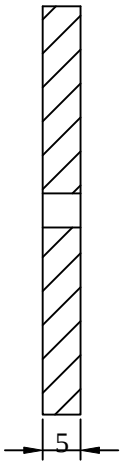


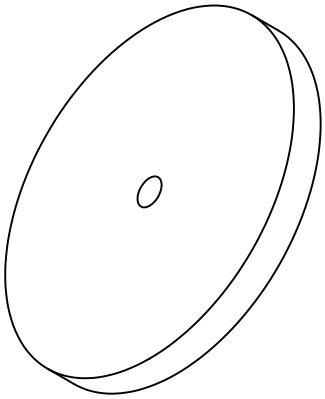




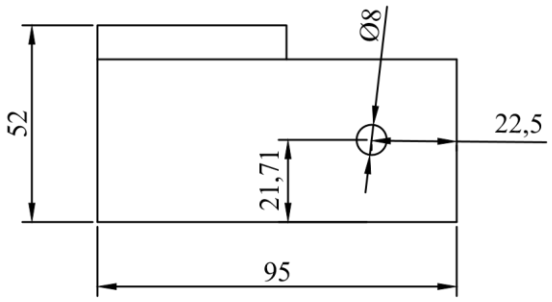
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Titulo del proyecto: Calibración y puesta en marcha de un sistea de decapado continuo con láser Titular: Javier Gamir Artesero		Fecha: 03/09/2024 Escala: 1:2
Autor: Javier Gamir Artesero	Denominación del plano Soporte Encoder	Número del plano 03

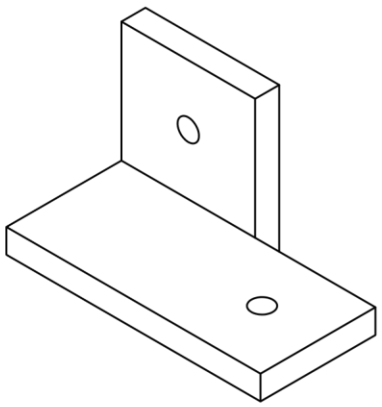


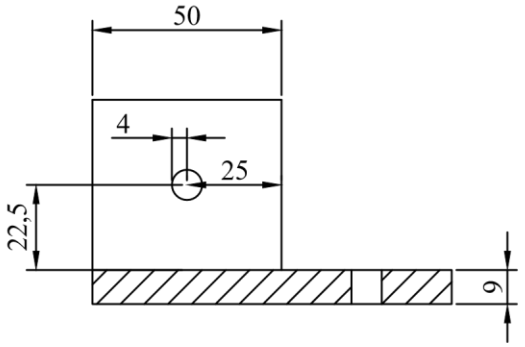


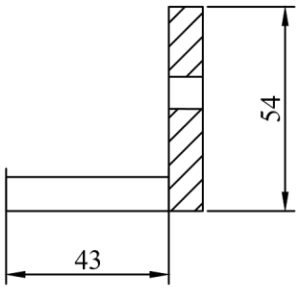


ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Título del proyecto: Calibración y puesta en marcha de un sistema de decapado continuo con láser Titular: Javier Gamir Artesero		Fecha: 03/09/2024 Escala: 1:1
Autor: Javier Gamir Artesero	Denominación del plano Rodamiento Encoder	Número del plano 04

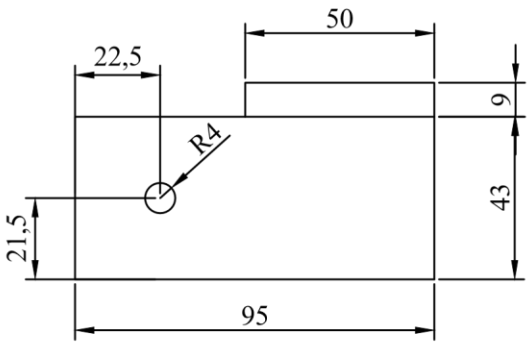


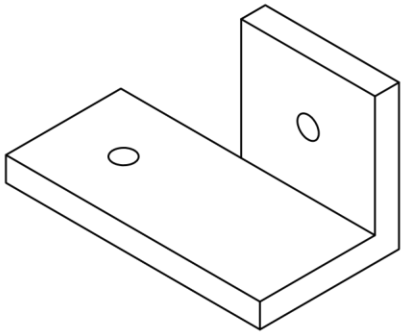


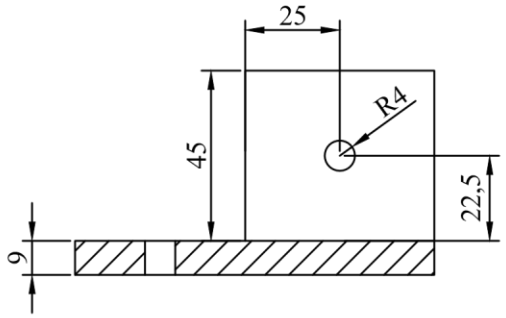


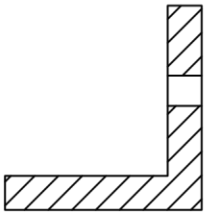


ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Titulo del proyecto: Calibración y puesta en marcha de un sistea de decapado continuo con láser Titular: Javier Gamir Artesero		Fecha: 03/09/2024 Escala: 1:2
Autor: Javier Gamir Artesero	Denominación del plano Soporte Derecho Unión de Barras	Número del plano 05

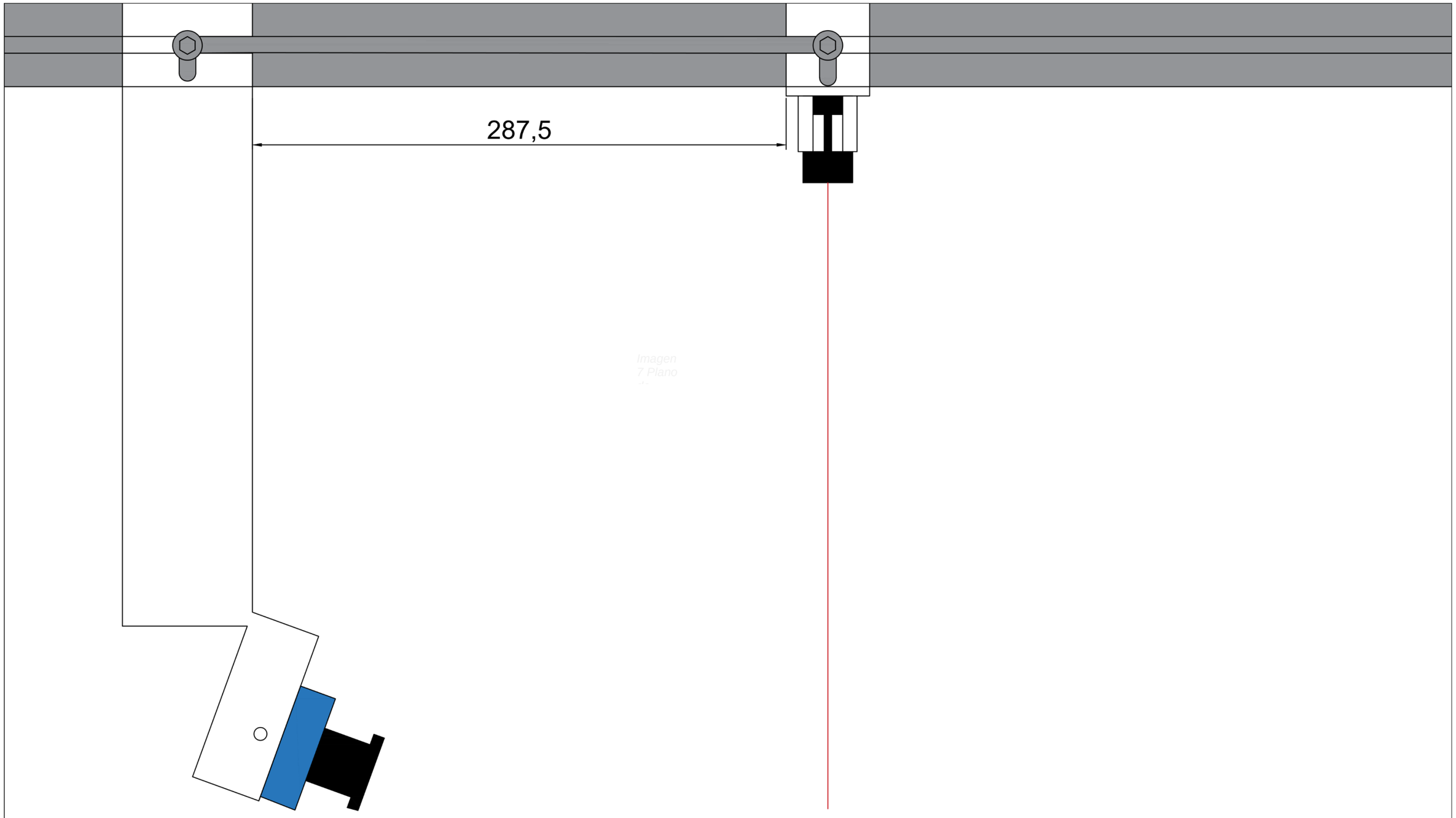








ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Titulo del proyecto: Calibración y puesta en marcha de un sistea de decapado continuo con láser Titular: Javier Gamir Artesero		Fecha: 03/09/2024 Escala: 1:2
Autor: Javier Gamir Artesero	Denominación del plano Soporte Izquierdo Unión de Barras	Número del plano 06



ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Titulo del proyecto: Calibración y puesta en marcha de un sistea de decapado continuo con láser Titular: Javier Gamir Artesero		Fecha: 03/09/2024 Escala: 1:2
Autor: Javier Gamir Artesero	Denominación del plano: Plano de Situación	Número del plano: 07 ~~17~~ 2

Calibración y puesta en marcha de un sistema de decapado continuo con láser

DOCUMENTO Nº3: PLIEGO DE CONDICIONES

Máster Universitario en Automática e Informática Industrial

Autor

Javier Gamir Artesero

Tutor

Carlos Ricolfe Viala

CURSO ACADÉMICO: 2023/2024

Índice

1 Condiciones Generales

Este proyecto tiene carácter de obligado cumplimiento una vez sellado y legalizado, debiendo ser objeto de aprobación previa todas aquellas modificaciones al mismo durante su ejecución.

1.1 Vigencia

Este Pliego de Condiciones, con todos sus articulados, estará en vigor durante la ejecución del desarrollo del proceso y hasta la terminación de este, entendiéndose que las partes a que hace referencia éste, se aceptarán en todos sus puntos por el adjudicatario del proceso. Frente a posibles discrepancias, el orden de prioridad de los documentos básicos del Proyecto será el siguiente:

1).- Planos.

2).- Pliego de Condiciones.

3).- Presupuesto.

4).- Memoria.

1.2 Descripción

Esta especificación se refiere al diseño e instalación de un sistema de reconstrucción 3D conformado por una cámara, un láser de luz estructurada y un encoder, así como la coordinación de este junto con láser de decapado y una cinta transportadora instalados en una celda.

Quedan excluidas de esta especificación la instalación de la cinta transportadora y el láser de decapado en la celda y la instalación eléctrica de los mismos para su funcionamiento.

1.3 Pliegos oficiales

Respecto al ámbito europeo, la normativa que hace referencia al proyecto es Directiva 2006/42/CE del Parlamento Europeo y del Consejo, de 17 de mayo de 2006, relativa a las máquinas y por la que se modifica la Directiva 95/16/CE.

Asimismo, la directiva 2014/35/UE del Parlamento Europeo y del Consejo de 26 de febrero de 2014 sobre la armonización de las legislaciones de los Estados miembros en materia de comercialización de material eléctrico destinado a utilizarse con determinados límites de tensión.

También el Real Decreto 187/2016, de 6 de mayo, por el que se regulan las exigencias de seguridad del material eléctrico destinado a ser utilizado en determinados límites de tensión.

Además, del Real Decreto 186/2016, de 6 de mayo, por el que se regula la compatibilidad electromagnética de los equipos eléctricos y electrónicos.

A su vez, el Real Decreto 1072/2015, de 27 de noviembre, por el que se modifica el Real Decreto 2200/1995, de 28 de diciembre, por el que se aprueba el Reglamento de la Infraestructura para la Calidad y la Seguridad Industrial.

1.4 Modificaciones

Durante la ejecución del proyecto, se podrán realizar cuantas modificaciones se estimen oportunas, siempre que las mismas sean aprobadas por el responsable de la Dirección del Proyecto, y en todo momento, de acuerdo con la entidad contratante.

2 Condiciones Técnicas

Las funciones de director del proyecto son las de revisión del trabajo realizado, programación de los trabajos, reconocimiento de los materiales utilizados y autorizaciones referentes al proyecto.

En el caso de que los materiales no fueran especificados, los que se utilicen deberán cumplir los requisitos mínimos de funcionamiento y tolerancia que se requiere, siendo obligatorio que sean normalizados y sometidos a la aprobación del director del proyecto.

Todos los trabajos se ejecutarán con estricta sujeción al proyecto que ha servido de base a la contratación y a las modificaciones que hayan sido aprobadas. En caso de dudas u omisiones, o con motivo de reforma del presupuesto, se formará un comité entre proyectistas, director del proyecto y, si se cree oportuno, el contratista, para decidir la solución más adecuada y económica.

3 Materiales

3.1 Arquitectura del sistema

3.1.1 Cinta Transportadora

La cinta deberá tener un ancho de banda de 600 mm, una longitud de 3 metros y una velocidad ajustable de hasta 112.8 m/min. El modelo deberá ser el Dorner Precision Move 2200 o equivalente, según el criterio del director del proyecto.

3.1.2 Cámara

La cámara tendrá unas dimensiones de 63 mm x 41 mm x 41 mm, deberá tener una resolución no menor de 1280x1024 y una tasa de mínimo 40 fps. Además, será necesario que esta cuente con entrada de canal A y B, con al menos una terminación de cada uno en MDR26 para su acople en el framegrabber. Además, deberá contar con disparo interno y externo por señal TTL. Por ello el modelo será la cámara Mikrotron EoSens Color o equivalente al juicio del director del proyecto.

3.1.3 Ordenador

El ordenador deberá tener disponibilidad de instalación de un software corredor de código C++ y la aplicación Matlab, a su vez deberá tener acceso a un switch para acceder a la red del láser y contar con una velocidad de procesamiento mínima de 3000 MHz y una RAM mínima de 8 GB. El modelo será un GREED® Intel Core i7 4790 Multimedia PC o equivalente al juicio del director del proyecto.

3.1.4 Encoder

El encoder debe ser de tipo incremental con una resolución máxima de 2000 pulsos por revolución. Además, la salida del encoder deberá ser de tipo TTL y la carcasa deberá tener un diámetro de 58 mm. En función de estas especificaciones, el encoder recomendado es el modelo Sick DGS60-E1B00250 o un equivalente, conforme a la evaluación del director del proyecto.

3.1.5 Láser de Luz Estructurada

El dispositivo láser con luz estructurada debe ser de tipo lineal, con un alcance mínimo de 2 metros y una precisión en el rango de milímetros. El láser deberá emitir luz de color rojo y funcionar con una alimentación de 5 voltios. En función de estas características, el modelo recomendado es el ZhonNa láser lineal o un producto equivalente, según la evaluación del director del proyecto.

3.1.6 Framegrabber

El framegrabber deberá disponer de conexión Camera Link con dos puertos, tipo A y B, compatibles con la cámara seleccionada. Además, deberá ofrecer una velocidad de hasta 40 fps a resolución completa y contar con un formato PCI Express, estándar para tarjetas de expansión. En base a estas especificaciones, el modelo recomendado es el Silicon Software microEnable IV AD4-C o un producto equivalente, de acuerdo con la evaluación del director del proyecto.

3.2 Software

3.2.1 Software Proceso

El programa relacionado con el sistema de reconstrucción 3D y coordinación con el láser deberá contar con librerías de acceso al framegrabber, matrices, manipulación de documentos y generación y control de nubes de puntos 3D.

Además, deberá ser compatible con la incorporación de DLLs de la aplicación Matlab donde se han creado herramientas específicas para el proceso.

También deberá de ser capaz de comunicarse con el láser de decapado haciendo uso de librerías ofrecidas por la compañía proveedora. El programa será C++ o equivalente al juicio del director del proyecto.

3.2.2 Software Calibración

El software utilizado para la calibración deberá tener la capacidad de calcular polinomios que se ajusten a una serie de datos. El programa deberá

permitir realizar análisis estadísticos, ajuste de curvas y modelos matemáticos complejos.

Además, deberá ser capaz de realizar la interpolación de datos, trabajar con nubes 3D, convertir estas nubes en imágenes 2D y realizar segmentaciones de estas. También deberá generar representaciones gráficas de los resultados obtenidos durante el proceso de calibración.

Asimismo, el software deberá permitir la obtención de matrices de transformación entre dos sistemas, dados una serie de puntos vistos por cada uno de ellos, y utilizar herramientas especializadas para la calibración y el ajuste de parámetros, como la optimización de mínimos cuadrados. Este software será Matlab o un equivalente aprobado por el director del proyecto.

4 Condiciones de uso, mantenimiento y seguridad

4.1 Obligaciones de usuario

El mantenimiento de la cinta transportadora, el encoder, la cámara, el framegrabber, el láser de luz estructurada y láser de decapado será a cargo de la unidad contratante. Estas operaciones se deben realizar siguiendo el manual que ofrece cada vendedor de cada producto.

5 Certificados y documentación

Con anterioridad al comienzo de los trabajos de la programación del presente proyecto, la Dirección del Proyecto podrá solicitar certificados de homologación de los materiales de que se compone el mismo, así como documentación y catálogos en los que se indiquen sus características principales.

6 Condiciones de ejecución

6.1 Descripción del proceso de ejecución

6.1.1 Disposición de materiales

La cinta transportadora debe estar ubicada en una zona segura dentro de una celda de trabajo. Esta debe situarse por debajo y en alineación directa con el láser de decapado y el sistema de reconstrucción 3D.

La cámara deberá instalarse en un soporte metálico, utilizando los dos soportes, derecho e izquierdo, indicados en el apartado de planos de la celda. Se empleará el soporte especificado en ese mismo apartado para su correcta fijación.

De manera similar, se instalará el láser de luz estructurada, respetando la distancia indicada en el plano de situación, también disponible en el apartado de planos.

El ordenador deberá estar ubicado cerca del switch de conexión a la red del láser de decapado, manteniendo una distancia de seguridad con la celda y asegurando la mínima longitud necesaria para interconectar los elementos como el framegrabber, la cámara y el encoder.

El encoder deberá fijarse de forma que se encuentre en contacto con la cinta transportadora.

Las instalaciones deberán de estar bien iluminadas ya sea de forma natural o de forma artificial.

6.1.2 Ejecución del software

En primer lugar, deberá instalarse el programa de procesamiento seleccionado. Si se utiliza la opción propuesta por el director del trabajo, C++ deberá estar acompañado de librerías para el manejo de matrices y manipulación de documentos, como las recomendadas "Armadillo" y "Pugi", en el ordenador de trabajo.

Además, deberán estar instaladas y correctamente integradas las DLLs creadas en Matlab para la localización de superficies planas, la adaptación de puntos a un polinomio y la creación de polinomios en archivos XML.

Una vez configurado el entorno, el proceso comienza con la captura de una imagen que contenga la zona de la pieza a decapar, la cual se analizará mediante la herramienta CLAM, colocando manualmente el punto central inicial.

Con los valores iniciales obtenidos del análisis de CLAM, se actualizarán en el programa de procesamiento, lo que permitirá iniciar la ejecución del software. Al aparecer en pantalla el mensaje “Que empiece el movimiento”, se procederá a desplazar la cinta transportadora a una velocidad de 2 mm/s, hasta que finalice el proceso de decapado.

Finalmente, si se está utilizando la aplicación de decapado continuo, será necesario detener la cinta durante un segundo antes de reanudar el movimiento al recibir el mensaje “XML3D_DEF v1.0 released 13:08 26/02/2024”, para evitar perder capturas de imagen mientras se genera el archivo XML.

7 Pruebas de Servicio

- Se deberá verificar que, al cambiar de modo manual a automático, el sistema solicite la confirmación del usuario antes de ejecutar el cambio.

- Se comprobará que, en el modo manual del robot, sea necesario activar el movimiento y que el usuario permanezca atento durante su operación.

- Se comprobará que la cámara esté correctamente calibrada con el láser. Para ello, se realizarán varias decapaciones puntuales con el láser, capturándolas con la cámara. Posteriormente, se transformarán las coordenadas y se verificará que los puntos captados y transformados coincidan con los decapados.

- Se verificará la calidad de la iluminación de la instalación y la intensidad del láser de luz estructurada, asegurando que la cámara detecte adecuadamente las áreas de decapado en las piezas y las placas de calibración utilizadas.

-Se deberá comprobar que la distancia entre un punto decapado por el láser y su correspondiente lectura en la cámara coincida con el número de pulsos calibrados en el encoder, y que esta medición sea estable y no varíe.

- Se deberá comprobar la precisión del sistema de reconstrucción 3D en condiciones de operación continua. Para esto, se capturarán y analizarán las nubes de puntos generadas por la cámara con las placas de calibración y se procederá a su decapado, verificando que no existan desfases entre la posición del láser y las coordenadas 3D reconstruidas por el sistema.

8 Certificados

8.1 Marcado CE

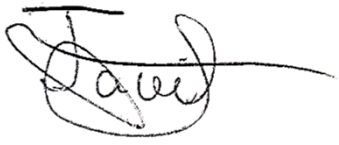
Todos los productos adquiridos deberán contar con su pertinente marcado CE. No se deberá admitir bajo ningún concepto aquellos productos que no cuenten con uno.

8.2 Ecodiseño

Todos los materiales y productos adquiridos deberán tener el “Certificado ecológico europeo” que acredite que los productos se producen siguiendo una estricta normativa basada en la protección del medio ambiente y del propio producto.

VALENCIA, SEPTIEMBRE 2024

Javier Gamir Artesero



DEPARTAMENTO DE INGENIERÍA
DE SISTEMAS Y AUTOMÁTICA

Calibración y puesta en marcha de un sistema de decapado continuo con láser

DOCUMENTO Nº4: PRESUPUESTO

Máster Universitario en Automática e Informática Industrial

Autor

Javier Gamir Artesero

Tutor

Carlos Ricolfe Viala

CURSO ACADÉMICO: 2023/2024

Índice

1 Objeto

Referencia	Unidades	Descripción	Precio (€)
Materiales			
m1	ud.	Dorner Precision Move 220	9500,00
m2	ud.	Cámara Mikrotron EoSens Color	2300,00
m3	ud.	ZhonNa laser lineal	16,00
m4	ud.	Plástico PVC	0,01
m5	ud.	GREED® Intel Core i7 4790 Multimedia PC	499,90
m6	ud.	FrameGrabber microenable iv ad4 cl	2160,00
m7	ud.	Lente Raspberrry-PI RPI-16MM	54,79
m8	ud.	Encoder	328,00
m9	ud.	Láser de decapado	60000,00
m10	ud.	Celda	5000,00
m11	ud.	C++	0,00
m12	ud.	Matlab	900,00
Secciones			
s1	ch	Impresión 3D	30,00
s2	h	Calibración del sistema de reconstrucción 3D	
s3	h	Programación de la comunicación láser-Cámara y del proceso de decapado	
M.O.D			
h1	h	Empresa de impresión 3d	30,00
h2	h	Programador	17,00

2 Cuadro de Precios Auxiliares

Referencia	Unidades	Descripción	Precio	Cantidad	Parcial
d1	ud.	Fabricación de soportes			
Materiales					
m4	ud.	Plástico PVC	0,01	2000,00	12,40
Secciones					
S1	ch	Impresión 3D	30,00	1	30
M.O.D.					
h1	h	Empresa de impresión 3d	30,00	1	30

Precio Total d1=72.40 €

Referencia	Unidades	Descripción	Precio	Cantidad	Parcial
d2	ud.	Programar Reconstruccción 3D			
Materiales					
m1	ud.	Dorner Precision Move 220	9500,00	1,00	9500,00
m2	ud.	Cámara Mikrotron EoSens Color	2300,00	1,00	2300,00
m3	ud.	ZhonNa laser lineal	16,00	1,00	16,00
m5	ud.	GREED® Intel Core i7 4790 Multimedia PC	499,90	1,00	499,90
m6	ud.	FrameGrabber microenable iv ad4 cl	2160,00	1,00	2160,00
m7	ud.	Lente Raspberrry-PI RPI-16MM	54,79	1,00	54,79
m8	ud.	Encoder	328,00	1,00	328,00
m10	ud.	Celda	5000,00	1,00	5000,00
m12	ud.	Matlab	900,00	1,00	900,00
Secciones					
s2	h	Calibración del sistema de reconstrucción 3D		250,00	0
M.O.D.					
h2	h	Programador	17,00	250,00	15508,69

Precio Total d2=25008,69 €

Referencia	Unidades	Descripción	Precio	Cantidad	Parcial
d3	ud.	Programar Comunicaciones y proceso decapado			
Materiales					
m9	ud.	Láser de decapado	60000,00	1,00	60000,00
d2	ud.	Sistema de Reconstrucción 3D Calibrado	0,00	1,00	0,00
m5	ud.	GREED® Intel Core i7 4790 Multimedia PC	0,00	1,00	0,00
m12	ud.	Matlab	900,00	1,00	900,00
Secciones					
S3	h	Programación de la comunicación láser-Cámara y del proceso de decapado		150,00	0
M.O.D.					
h2	h	Programador	17,00	150,00	2550,00

Precio Total d3=63450 €

3 Cuadro de Precios Descompuestos

D1	ud.	Desarrollo completo del sistema de decapado y reconstrucción 3D			
Materiales					
d1	ud.	Fabricación de soportes	72,40	1,00	72,40
d2	ud.	Programar Reconstruccción 3D	25008,69	1,00	25008,69
d3	ud.	Programar Comunicaciones y proceso decapado	63450,00	1,00	63450,00
CD02					
%	Porcentaje	Costes Directos	0,02		1770,62

Precio Total D1=90301,71 €

4 Resumen del presupuesto

Referencia	Unidades	Descripción	Precio	Cantidad	Importe
Ref	Ud	Descripción	Precio	Cantidad	IMPORTE
D1	ud	Desarrollo completo del sistema de decapado y reconstrucción 3D	90301,71	1,00	90301,71
Presupuesto de ejecución material			90301,71	1,00	90301,71
Gastos Generales	%		0,06	1,00	5418,10
Beneficio industrial	%		0,13	1,00	11739,22
Presupuesto de ejecución por contrata			114134,94	1,00	107459,04
Porcentaje de honorarios y trámites	%		0,10	1,00	9030,17
IVA sobre el porcentaje de honorarios	%		0,21	1,00	18963,36

TOTAL, PRESUPUESTO DE EJECUCIÓN GENERAL=

CIENTO TREINTA Y CINCO MIL CUATROCIENTOS CINCUENTA Y DOS CON CINCUENTA Y SIETE EUROS (135452,57)

VALENCIA, SEPTIEMBRE 2024

Javier Gamir Artesero