Universidad Miguel Hernández de Elche
Máster Universitario en Robótica

“Clasificación robotizada de piezas: Aplicación a la industria del calzado”

Trabajo de Fin de Máster
Curso académico 2022-2023

Autor: Adrián Pascual Böhm
Tutor: Carlos Pérez Vidal

Agradecimientos

Me gustaría dar las gracias a todas las personas que, de una forma u otra, han contribuido al desarrollo de este trabajo:

A todos los profesores e investigadores del Máster en Robótica, por su dedicación en la labor docente y por proporcionarme todos los conocimientos necesarios para el desarrollo de este trabajo.

A mi tutor Carlos Pérez Vidal por depositar su confianza en mí ofreciéndome este proyecto, y ayudándome en todo lo posible.

A la fundación ValgrAI, por la beca recibida al cursar este máster, así como la invitación a conferencias en el ámbito de la inteligencia artificial.

Por último, agradecer también a mi familia por el apoyo incondicional, comprensión y ánimo constante durante toda mi trayectoria académica.

Resumen

Una de las ventajas más notorias de la etapa de robotización que estamos viviendo hoy en día, es sin duda la optimización de procesos industriales, con el objetivo de abaratar costes, y mejorar la productividad. Cada proceso es único, por lo que es necesario una labor de investigación que permita aplicar las tecnologías adecuadas y sacarles el máximo partido. El presente trabajo presenta un enfoque interdisciplinario que combina la mecánica, la programación y la visión por computadora a fin de desarrollar un sistema de agarre robótico diseñado para la manipulación precisa de piezas en la industria de calzado.

Este *gripper* robótico se caracteriza por su capacidad de ajustar su posición y orientación de manera dinámica, adaptándose a la geometría única de cada pieza en particular. Para lograr esta versatilidad, se empleará un script de Python que procesará archivos *DXF*, permitiendo así que el *gripper* pueda identificar y adaptarse a la forma específica de las piezas. Además, se incorporarán técnicas de visión por computadora para obtener información precisa sobre la posición de las piezas, lo que garantizará un proceso de manipulación eficiente y preciso.

Abstract

One of the most noticeable advantages of the ongoing stage of robotization we are experiencing today is undoubtedly the optimization of industrial processes, with the aim of reducing costs and improving productivity. Each process is unique, so research is necessary to apply the appropriate technologies and make the most of them. This work presents an interdisciplinary approach that combines mechanics, programming, and computer vision to develop a robotic gripper system designed for precise handling of pieces in the footwear industry.

This robotic gripper is characterized by its ability to dynamically adjust its position and orientation, adapting to the unique geometry of each particular piece. To achieve this versatility, a Python script will be employed to process DXF files, allowing the gripper to identify and adapt to the specific shape of the pieces. Furthermore, computer vision techniques will be incorporated to obtain accurate information about the position of the pieces, ensuring an efficient and precise manipulation process.

Índice general

Índice de figuras

1 Introducción

1.1 Contexto, objetivos y alcance del trabajo

La robótica industrial ha experimentado en los últimos años un crecimiento constante debido a varios factores, que han convergido para impulsar su desarrollo y adopción global:

- Avances tecnológicos: Los avances recientes en tecnología, como sensores, actuadores o sistemas de visión, han permitido el desarrollo de robots industriales más capaces y versátiles.
- Reducción de costos: A medida que la tecnología se asienta en la sociedad, se vuelve más accesible. Los costos de adquisición y mantenimiento de los robots industriales han disminuido considerablemente en los últimos años. Esto ha permitido que no solo las grandes empresas, sino también las medianas y pequeñas, sean capaces de acceder a robots industriales a fin de mejorar su productividad.
- Automatización y eficiencia: En un mundo cada vez más globalizado donde la competencia entre empresas es feroz, es imprescindible aumentar la eficiencia y la productividad en la fabricación. Esto ha llevado a las empresas a invertir en robótica y automatización.

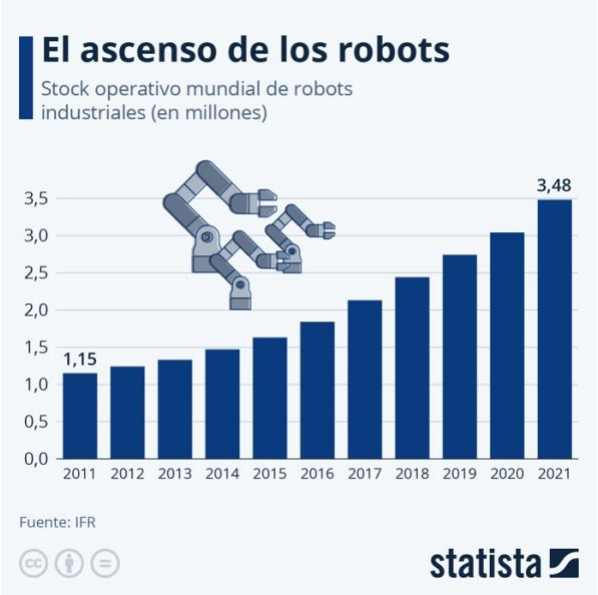


Ilustración 1 Stock mundial de robots industriales [16]

En este entorno industrial, cada vez más automatizado, se busca optimizar cualquier proceso para abaratar costes y mejorar tiempos de producción. En este trabajo se presenta una solución robotizada de aplicación real en la industria de calzado: el desarrollo e implementación de un *gripper* robótico para la manipulación de piezas metálicas utilizadas en la fabricación de suelas.

Un *gripper* robótico, es un dispositivo diseñado para agarrar, sujetar o manipular objetos de diversas formas y tamaños utilizando un robot. El desarrollo de *grippers* ha sido un proceso vital en la evolución de la robótica industrial, ya que una incorrecta sujeción o manipulación de las piezas puede conllevar desde una disminución de la productividad hasta graves accidentes en las cadenas de producción.

A la hora de diseñar la herramienta, es necesario hacer un estudio de todos los aspectos y características del proceso, a fin de optimizarlo. El aspecto principal a tener en cuenta en el desarrollo de este *gripper* radica en la necesidad de interpretar y adaptarse a la variabilidad en el tamaño y la forma de las piezas. Para ello, se han implementado una serie de algoritmos en Python, que permiten al robot leer archivos DXF (archivos 2D con información sobre la forma de las piezas), brindando una mayor flexibilidad y adaptabilidad en el proceso de manipulación. También se incorporará una cámara, para ubicar las piezas en el espacio de trabajo.

La primera parte de este trabajo se estructurará en torno al diseño y desarrollo y del *gripper* robótico, considerando todos los aspectos mecánicos. Después, se implementarán los algoritmos necesarios para el control de este usando Python.

1.2 Tecnologías involucradas

- Robot

Para llevar a cabo la robotización de esta tarea se ha elegido un robot tipo SCARA, este robot consta de 4 grados de libertad, número suficiente al tener que manipular piezas planas de manera horizontal, sin inclinación. Debido a su diseño, los robots SCARA son especialmente útiles a la hora de realizar movimientos rápidos y repetitivos en el plano horizontal, esto los hace muy eficientes en aplicaciones como manipulación de piezas, ensamblaje o *pick & place*.

Ilustración 2 Robot tipo SCARA [7]

Estos robots tienen una carga útil de 5 kg, suficiente para nuestra aplicación, aunque habrá que tenerlo en cuenta al diseñar el *gripper* para no sobrepasarla.

Ilustración 3 Controladora [7]

El brazo robótico se controla a través de una controladora de 4 ejes, estas controladoras cuentan con entradas y salidas, digitales y analógicas que permiten la conexión de actuadores, sensores, y otros dispositivos periféricos. En nuestro caso, se conectará el actuador de la bomba de vacío, y una cámara para detectar la posición de las piezas.

- Sistema de visión

Hoy en día prácticamente todos los procesos industriales cuentan con un sistema de captación de imagen, que permita obtener información del entorno. Para esta aplicación es necesario que el robot sea capaz de detectar la posición en el espacio de trabajo de las piezas, usando visión artificial. Para realizar las capturas, se usará una cámara USB de la marca ELP con las siguientes características:

- Distancia focal 5-50 mm
- Resolución 1080p
- Formato de vídeo H.264/MJPEG/YUY2
- Función de baja iluminación

Ilustración 4 Cámara [8]

- Software

A continuación, se presentan las herramientas de software usadas para la realización de este trabajo:

- *SolidWorks:* software utilizado para el diseño 3D.
- *Pycharm:* Entorno de desarrollo integrado elegido para escribir el código.
- *Python:* el lenguaje de programación para este proyecto será *Python.* Cabe destacar el uso de las siguientes librerías:

  - *Ezdxf:* Librería de Python que permite leer, crear y editar archivos DXF, un formato de archivo ampliamente utilizado para el intercambio de datos en dos dimensiones entre programas de diseño asistido por ordenador (CAD).
  - OpenCV: La librería OpenCV es una herramienta ampliamente utilizada en Python para el procesamiento de imágenes y visión artificial. Se utilizará para obtener y procesar imágenes de la cámara.
  - *Numpy*: librería diseñada para realizar cálculos numéricos y matemáticos avanzados, como por ejemplo la manipulación de matrices multidimensionales
  - *Shapely*: librería utilizada para realizar operaciones geométricas y análisis espacial en geometrías planas
  - *Matplotlib*: librería de muy utilizada para crear gráficos y visualizaciones de datos en 2D y 3D. Fue diseñada originalmente para emular las capacidades de visualización de MATLAB.

2 Estado del arte

Hoy en día la robotización está desempeñando un papel fundamental en la transformación de la todas las industrias. La necesidad de mejorar la eficiencia, la calidad de la producción, así como la reducción de costos, ha impulsado la adopción de soluciones robóticas en muchos procesos industriales.

La industria de textil y de calzado es un buen ejemplo de esta necesidad, ya que su línea de producción está llena de tareas repetitivas, fácilmente automatizables. Los robots industriales se utilizan para llevar a cabo tareas como el corte de materiales, la costura, el pegado y el ensamblaje de componentes. Esto, aparte de conllevar un beneficio económico directo, permite que los trabajadores puedan formarse y realizar tareas más amenas y menos demandantes físicamente.

Ilustración 5 Máquina de corte textil[9]

Uno de los desafíos más importantes en la robotización de la industria del calzado es la necesidad de manipular piezas flexibles y con gran variabilidad en tamaño y forma, como, por ejemplo, materiales textiles, cuero o piezas metálicas. Es por ello por lo que, el desarrollo de *grippers* robóticos capaces de adaptarse a cada tarea, se ha convertido en una prioridad para las empresas.

Los grandes avances de los últimos años en técnicas de visión artificial permiten a los robots identificar y manipular con precisión los distintos componentes del calzado, incluso en entornos cambiantes.

Existen multitud de tipos de *grippers* robóticos, a fin de adaptarse a cualquier proceso industrial. Muchos de ellos son fabricados directamente por los fabricantes de robots, aunque también hay empresas que se dedican únicamente al desarrollo de *grippers*, para casos muy específicos. A continuación, se detallan algunos de los más utilizados según su accionamiento:

Ilustración 6 Tipos de grippers robóticos [10]

- **Eléctricos:** Los *grippers* eléctricos son los más comunes debido a que se adaptan bien a la mayoría de las aplicaciones que requieran una fuerza de agarre media, y gran rapidez. Los grippers eléctricos suelen estar disponibles en configuraciones de dos y tres mordazas, siendo estos últimos elegidos a la hora de manipular objetos redondos o cilíndricos.
- **Hidráulicos:** Si es necesaria una fuerza de agarres mayor, y la velocidad de cierre no es determinante, el agarre óptimo será el de tipo hidráulico. Este agarre suele utilizarse mucho en la industria automovilística, en la que se trabaja con piezas pesadas.

- **Neumáticos:** Los *grippers* neumáticos ofrecen gran rapidez y repetibilidad, con una fuerza de agarre media-alta, aunque no son los más idóneos si se manipulan piezas con una alta variabilidad en el tamaño.
- **Vacío:** mecanismo seleccionado para este proyecto, se basa en la aplicación de una presión negativa, mediante ventosas, en la superficie de la pieza a manipular, de forma que esta quede pegada. Se usan cuando las piezas tienen una superficie lisa y amplia, lo que dificulta el agarre por otros métodos.

Otra de las tecnologías más importantes a la hora de automatizar y optimizar procesos industriales es la visión artificial. La visión artificial consiste en el desarrollo de algoritmos que permiten a las máquinas interpretar imágenes o videos digitales captados por cámaras. El objetivo principal de la visión artificial es extraer información del procesos y tomar decisiones en consecuencia. En el ámbito de la industria de calzado, la visión por computador puede aplicarse en:

- **Clasificación**: la visión artificial es muy eficaz a la hora de reconocer cambios en la forma y el tamaño de las piezas, lo que permite su clasificación.
- **Inspección de calidad:** la rapidez a la hora de ejecutar algoritmos que filtren las imágenes para detectar fallos permite a estos sistemas hacer un control de calidad en tiempo real. Defectos de forma, grietas, o el control de tolerancias son algunas de sus aplicaciones. Estos sistemas también se pueden combinar con sensores láser o rayos X para mejorar su precisión.
- **Seguimiento y etiquetado:** la implementación de estos sistemas puede ser muy útil en cintas de transporte, donde hay que realizar el seguimiento de un gran número de paquetes o piezas.

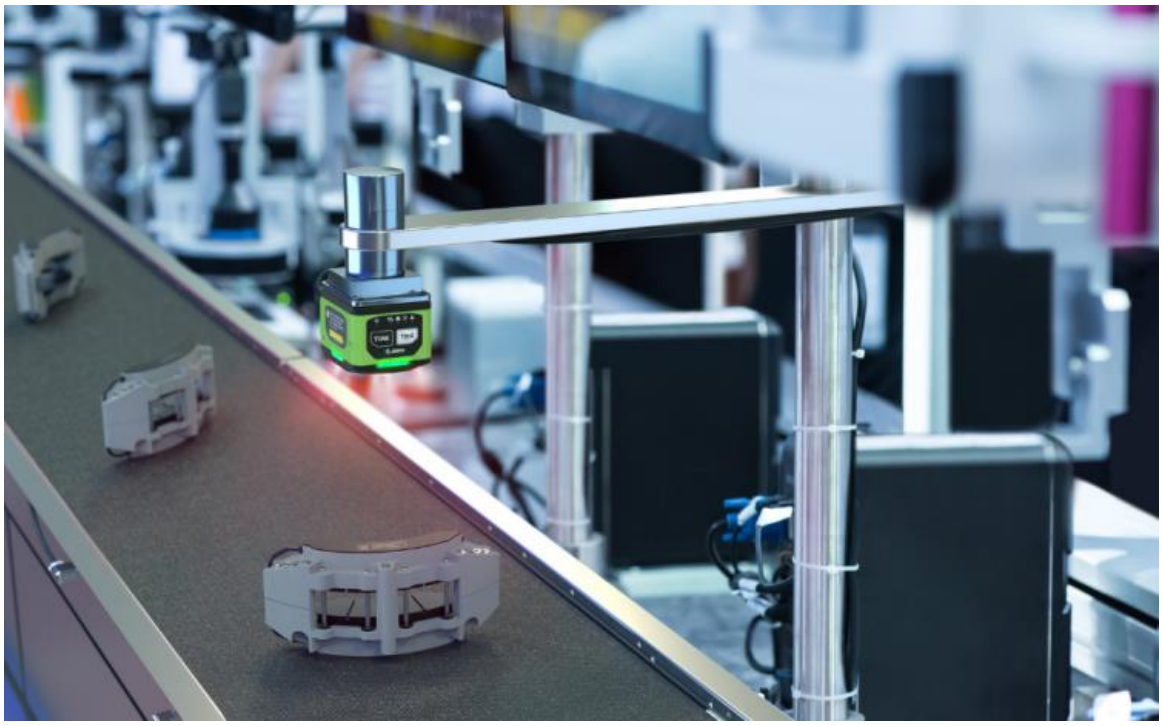


Ilustración 7 Inspección mediante visión artificial [11]

Algunas de las técnicas utilizadas en los programas de visión pueden ser, desde el uso de máscaras que detecten elementos característicos (contornos, esquinas…), hasta el uso de inteligencia artificial y aprendizaje automático para reconocer determinadas propiedades.

3 Diseño del *gripper* robótico

3.1 Requerimientos y descripción del diseño

El robot tendrá que manipular piezas con forma de suela de zapato, con distintos tamaños. A continuación, se muestran dos ejemplos:

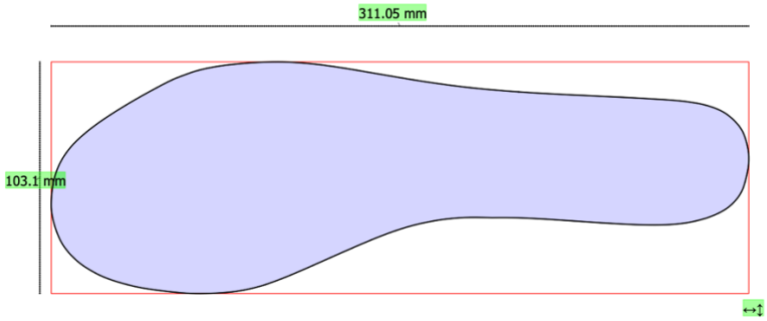


Ilustración 8 Suela pequeña

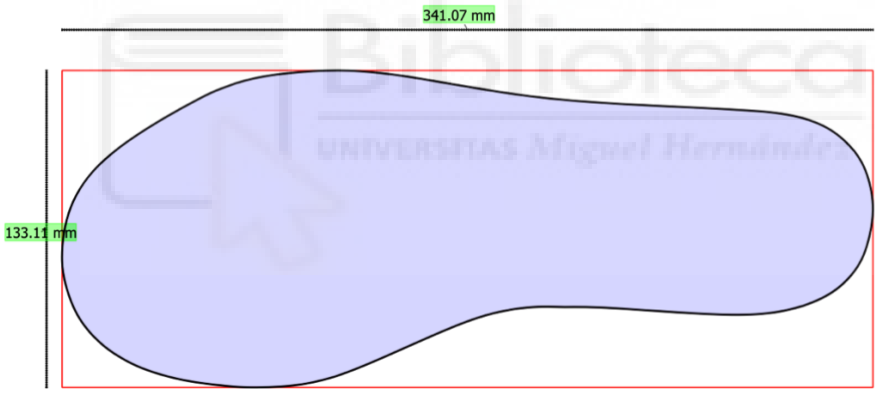


Ilustración 9 Suela grande

El cuadro exterior del *gripper* se ha dimensionado para la segunda imagen, que es la pieza de máximo tamaño que va a ser manipulada (341.07 x 133.11 mm).

Las piezas provienen de una máquina de corte por agua, esto se ha tenido en cuenta para el diseño ya que, al proceder a la extracción, es posible que éstas se queden encajadas en el contorno exterior. Es por eso por lo que se ha obtado por poner un cuadro exterior que ejerza presión sobre el contorno, facilitando así la extracción de la pieza. Como método de sujeción se ha elegido el de succión mediante una bomba de vacío y ventosas.

A continuación, se presenta el diseño del *gripper*:

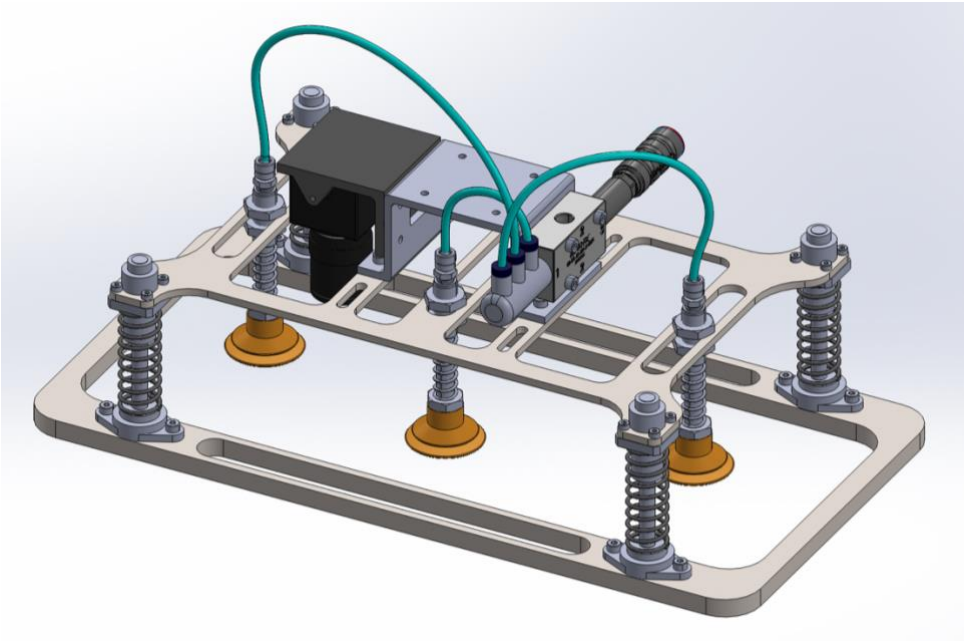


Ilustración 10 Propuesta de diseño

Dada la forma, tamaño y peso de la pieza, se ha considerado como suficiente la utilización de 3 ventosas. La distribución de estas respecto al centro de la herramienta es la siguiente:

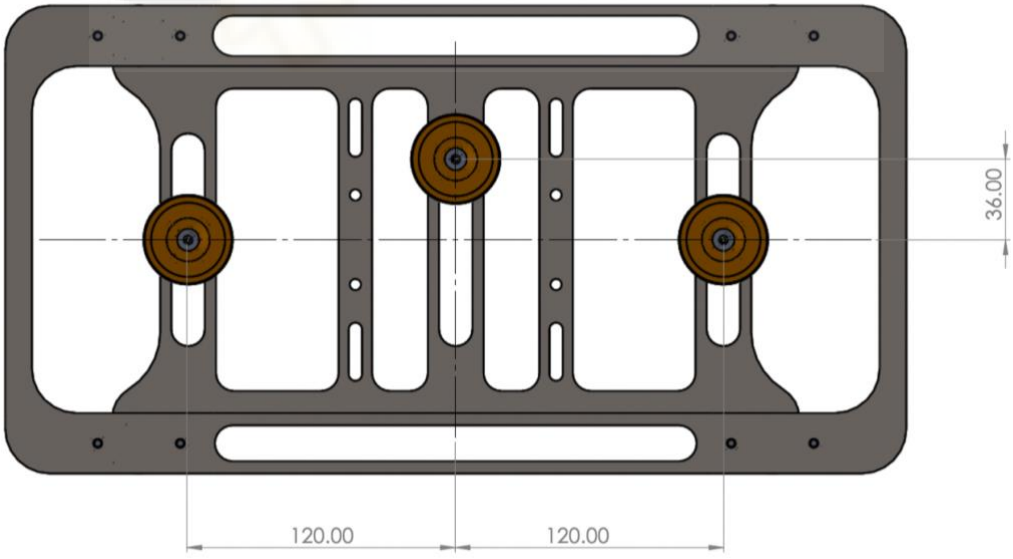


Ilustración 11 Disposición de las ventosas

Estas medidas hay que tenerlas en cuenta a la hora de programar los algoritmos correspondientes, para conocer la posición exacta de cada ventosa.

Por último, se esquematiza la distribución de las ventosas respecto a la pieza a manipular:

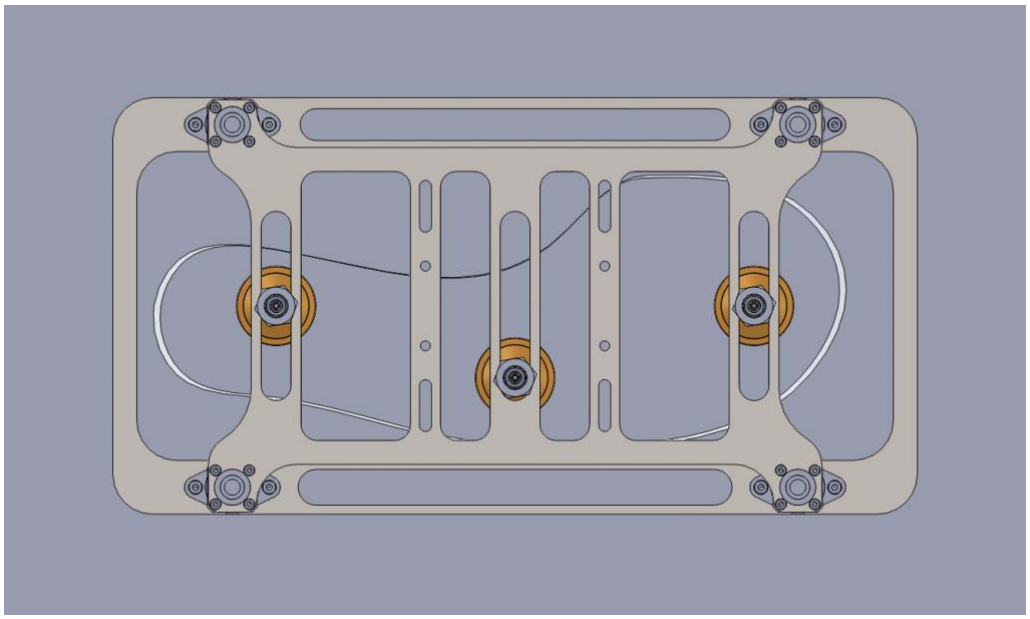


Ilustración 12 Disposición de las ventosas en la pieza

Tanto la elección de los materiales de fabricación, como el diseño de las piezas, se han elegido con el fin de reducir el peso, ya que el robot tiene una carga máxima de 5 Kg. Para verificar esto se ha hecho una simulación en SolidWorks, teniendo en cuenta los materiales de cada pieza. El peso total aproximado es de 2 Kg, a lo que habrá que sumarle el peso de la pieza (600 gr.) y la fuerza necesaria para extraer la pieza.

3.2 Piezas

- Matriz de ventosas

La matriz de ventosas se ha diseñado de manera que permita la incorporación de 3 portaventosas con posición regulable en un eje. El material de fabricación será titanio debido a su bajo peso y alta resistencia. El método de fabricación será corte por agua.

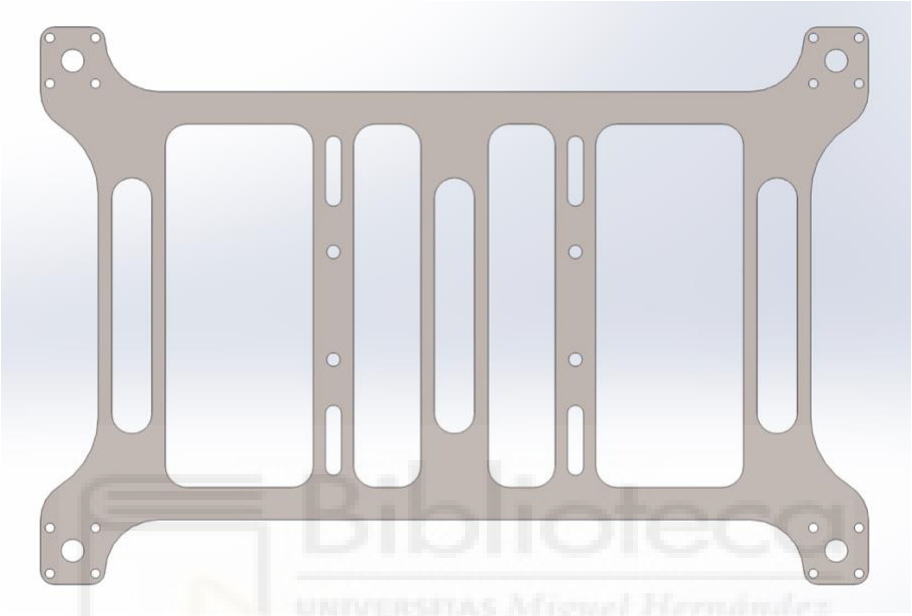


Ilustración 13 Matriz de ventosas

- Sistema de vacío

Dado que las plancha son metálicas y de superficie lisa, se ha obtado por el vacío como método de sujeción

  - Bomba de vacío

Se ha obtado por una bomba de vacío EJ-BA-M-LP-3-G1/8 de Gimatic. Esta bomba nos ofrece un caudal de 1.5 Nl/s y un vacío de -89 Kpa, suficiente para nuestra aplicación. La bomba se conectará directamente a la controladora del robot para su activación.

Ilustración 14 Bomba de vacío [12]

- Portaventosas, ventosas y racor neumático.

Para el portaventosas se ha elegido el modelo 203515 de la marca Vuototecnica. Las ventosas son estándar de la misma marca, con un diámetro de 40 mm.

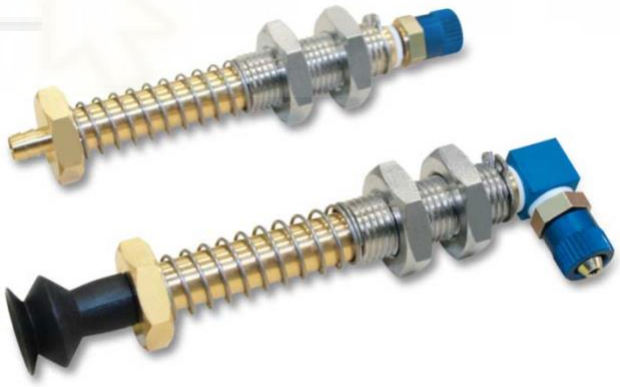


Ilustración 15 Portanvenstosa y ventosa [13]

Por último, para conectar los 3 portaventosas en paralelo, es necesario un racor neumático de 3 salidas, con sus respectivos tubos.

Ilustración 16 Racor neumático [13]

- Fijación efector

Esta pieza será la unión del *gripper* al efector final del robot, también servirá de sujeción al sistema de vacío y la cámara. El material deberá de ser también ligero, como titanio o aluminio.

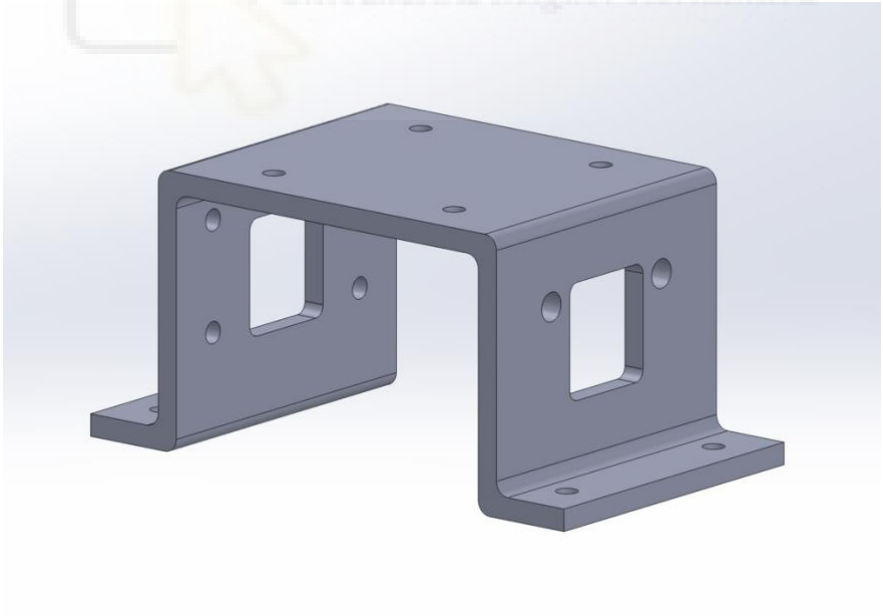


Ilustración 17 Sujección efector

- Soporte cámara

Otra de las piezas fabricadas a medidas será el soporte de la cámara. Ya que no debe tener una alta resistencia, se va a fabricar mediante impresión 3D, a fin de abaratar costes. El material será ABS o PLA.

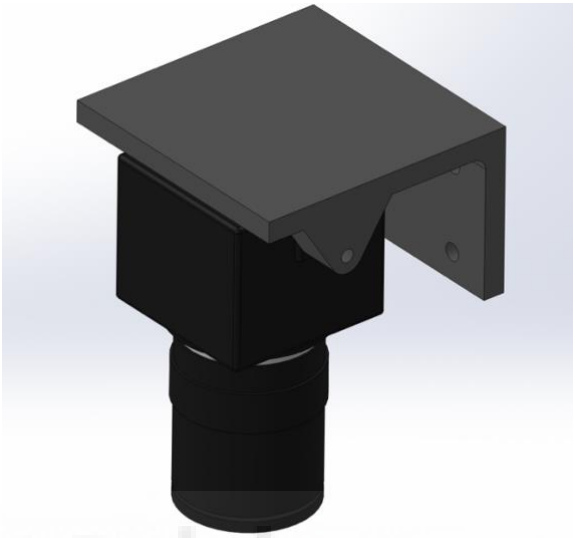


Ilustración 18 Soporte de la cámara

- Cuadro exterior

El cuadro exterior que sujeta el contorno de la pieza también será de fabricación propia por corte por agua. El material también será titanio para aligerar peso. En los agujeros se insertará una rosca M4x0.5 para la sujeción de los ejes.

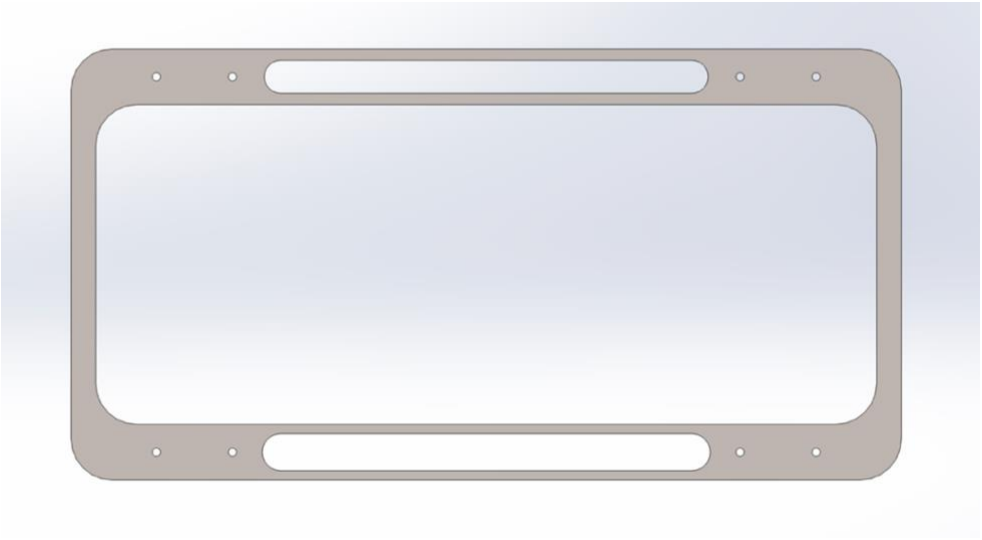


Ilustración 19 Cuadro exterior

- Muelles

A la hora de seleccionar los muelles, es importante calcular la fuerza ejercida por los mismos, para asegurar que no se exceda la fuerza máxima aplicable por el robot. En este caso, se han seleccionado muelles con una constante elástica de 0,25 N/mm. Teniendo en cuenta que el recorrido de los ejes será de 25 mm, y sumándole 5 mm más de pretensionado, podemos calcular la fuerza necesaria, que es 7,5 N por muelle, o 30 N (3 Kg) en total.

$$F = 0{,}25\frac{N}{mm} * 30\ mm = 7{,}5\ N$$

$$7{,}5\ N * 4 = 30\ N = 3\ Kg$$

Ilustración 20 Muelle

Teniendo en cuenta que el peso total del gripper es de unos 2 Kg, solo será necesario la aplicación de una fuerza equivalente a 1 Kg en dirección hacia abajo.

- Piezas estándar

Para terminar el diseño, también se usarán una serie de piezas estándar, que permitirán el movimiento del cuadro exterior.

- Ejes:

El tamaño de eje elegido para esta aplicación es de 8 mm este tamaño nos permite aligerar peso manteniendo una buena resistencia:

Ilustración 21 Ejes de 8 mm [14]

- Cojinetes

A fin de permitir el movimiento lineal de los ejes, se han elegido los siguientes cojinetes de 8 mm:

Ilustración 22 Cojinetes [14]

- Soportes y topes del eje

Para sujetar los ejes al cuadro exterior, se ha elegido el siguiente soporte:

Ilustración 23 Soporte del eje [14]

Por el otro extremo del eje, se necesitarán unos topes para evitar que éstos se salgan de los cojinetes.

Ilustración 24 Tope del eje [14]

- Tornillos y tuercas

Por último, para unir todas las piezas se usarán tornillos DIN912 de tamaño M4 y M3, con sus respectivas tuercas.

4 Implementación en Python

Python es uno de los lenguajes de programación más usados en la industria, esto se debe a su facilidad de uso y a la existencia de multitud de librerías, que agilizan el desarrollo de algoritmos. Para la tarea asociada a este proyecto, podemos dividir el desarrollo del código de Python en dos subtareas o fases, según el tipo de técnicas usadas. En la primera parte, se usarán técnicas relacionadas con la manipulación de archivos en formato *DXF*, que es el formato predominante para la exportación de formas en 2 dimensiones. Este archivo proviene directamente del ordenador que controla la máquina de corte, el robot será capaz de leer el archivo y cambiar la orientación de las ventosas para ajustarse a la forma.

En la segunda parte, se usarán técnicas de visión artificial para captar una imagen y detectar los bordes de la pieza de trabajo, para mover el efector del robot al punto necesario para extraer la pieza.

4.1 Suposiciones iniciales

A continuación, se muestra la pieza a manipular sobre el espacio de trabajo:

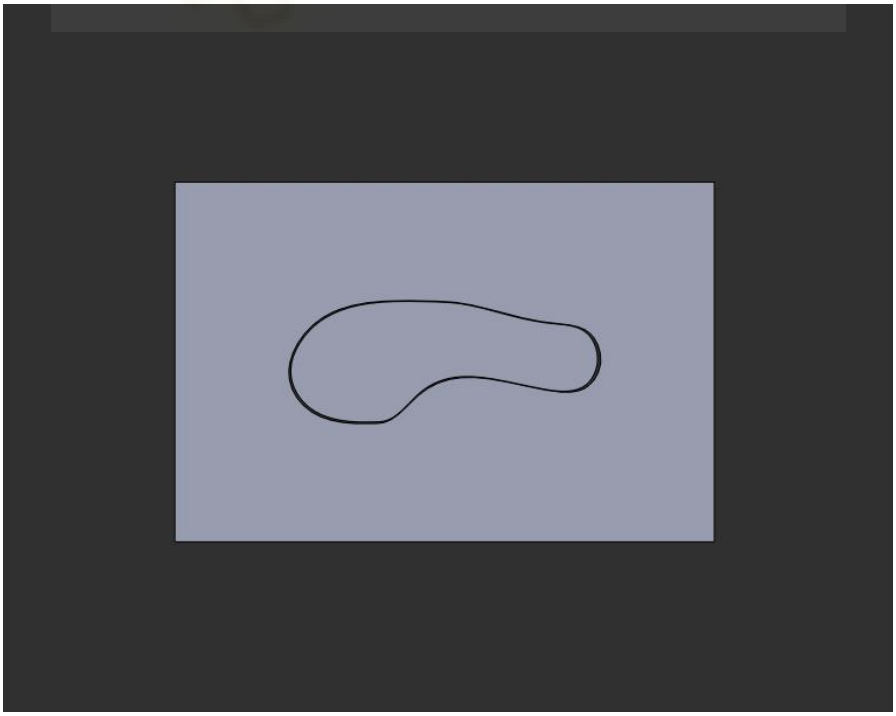


Ilustración 25 Pieza cortada

Esta representación no es realista, ya que, al ser cortada por agua, es probable que el borde de la pieza no sea visible a simple vista. Si el corte fuera visible, podrían usarse únicamente técnicas de visión artificial para detectar el borde y orientar el efecto, pero se ha considerado que, en este caso, usar estas técnicas no sería fiable. Es por ello por lo que se ha obtado por trabajar también con el archivo DXF.

La imagen con la que se trabajará en la parte de visión será, por tanto, la siguiente:

Ilustración 26 Plancha de acero

En un entorno real, es probable que las planchas no salgan de la máquina todas con idéntica posición y orientación. Por este motivo, se ha desarrollado un algoritmo que detecta los bordes de la plancha, y nos da una matriz de transformación que relaciona las coordenadas de la plancha con las coordenadas globales del espacio de trabajo. Esto asegurará la precisión del robot independientemente de la posición y orientación de la plancha.

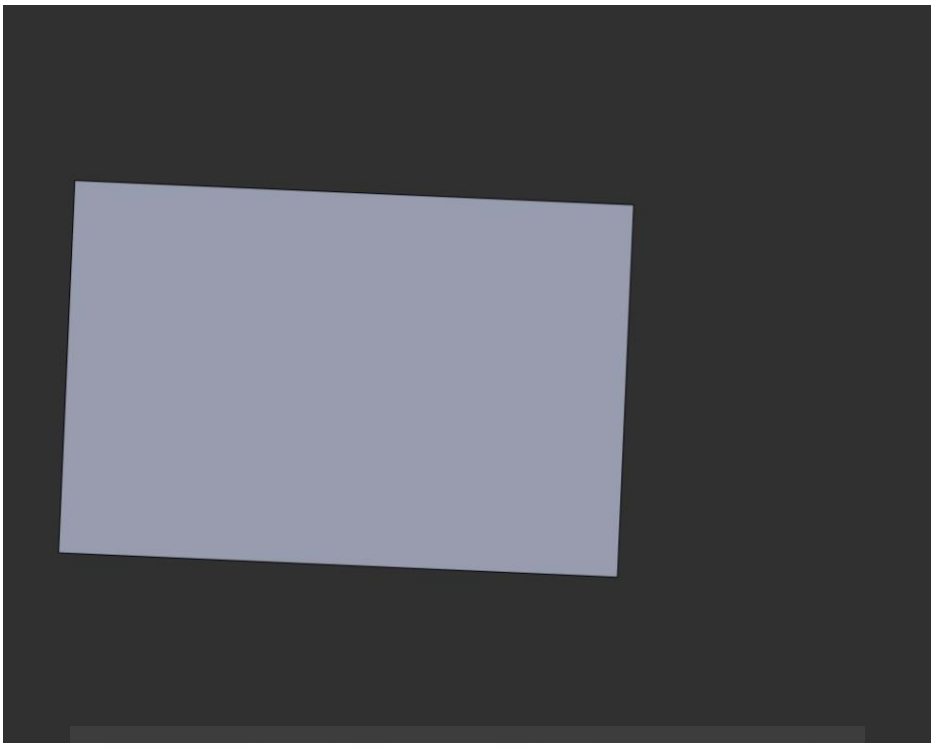


Ilustración 27 Plancha inclinada y desplazada

Por otro lado, tenemos el archivo DXF, representado en Python:

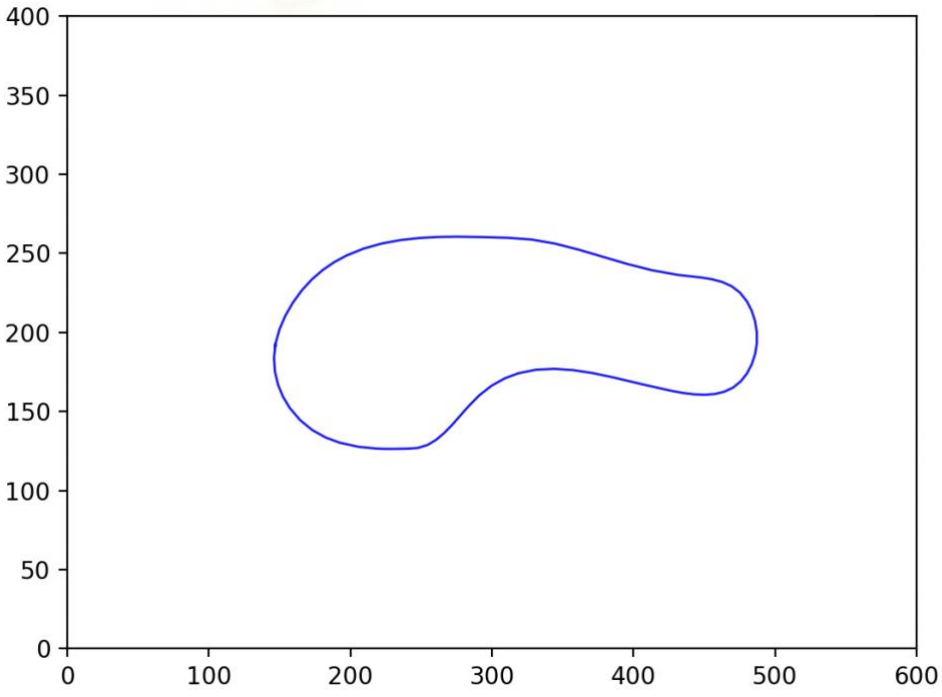


Ilustración 28 Pieza en formato DXF

Como suposición inicial y a fin de simplificar el código, el centroide de la pieza será coincidente con el centro de la plancha de acero. En cuanto a la orientación de la pieza, esta podrá tener cualquier valor.

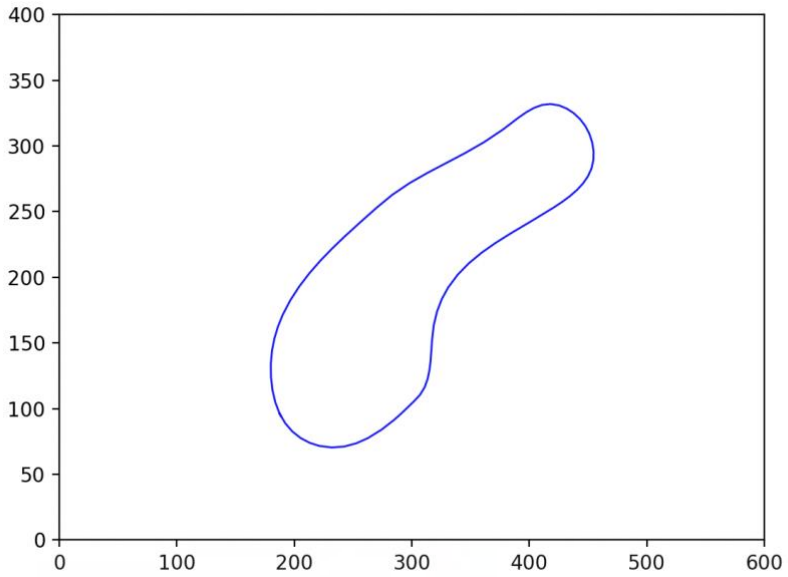


Ilustración 29 Pieza girada 40 grados.

Por último, se muestran las ventosas según la disposición seleccionada en el capítulo de diseño:

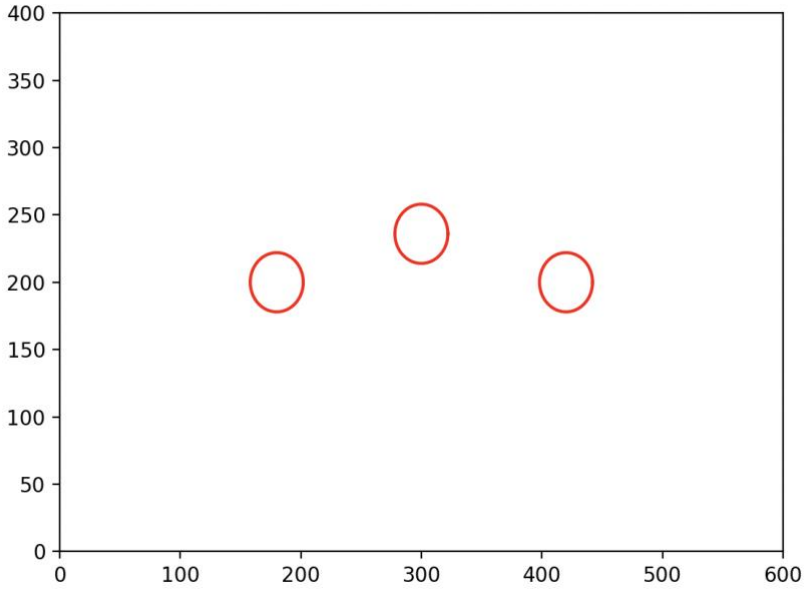


Ilustración 30 Representación de las ventosas

Estas tendrán las coordenadas (-120, 0), (120, 0) y (0, 36) respecto al centro del efector. Las ventosas tendrán un diámetro de 40 mm, pero se han representado con un diámetro de 44 mm a fin de asegurar que el agarre es el adecuado, y que ninguna ventosa está demasiado cerca del borde.

4.2 Orientación del *gripper* respecto a la forma de la pieza

En esta primera fase se supone que el centro del efecto coincide con el centroide de la pieza. A continuación,se exponen los algoritmos creados para la representación de las formas y el cálculo del ángulo de giro necesario. Se trabajará desde un script de Python principal, llamado *Orientation*, y se utilizarán algunas funciones auxiliares guardadas en un script llamado *Functions*, estas funciones nos servirán para representar y modificar la forma y las ventosas en el gráfico. Empezamos definiendo las funciones auxiliares:

```
def displacement(centroid_x,centroid_y,h_fig,w_fig):
    tx = w_fig / 2 - centroid_x
    ty = h_fig / 2 - centroid_y
    return tx, ty
```

La función *displacement*, tiene cómo input el centroide de la forma a mover (pieza), y el tamaño de la figura (plancha). Su objetivo es calcular la distancia necesaria para hacer coincidir el centroide de la pieza con el centro de la figura. Esto es importante ya que en el archivo DXF, sólo tenemos el contorno de la pieza, y no las dimensiones de la plancha.

```
def read_splines(doc):
    msp = doc.modelspace()
    for e in msp:
        if e.dxftype() == 'SPLINE':
            spline = msp.query('SPLINE')[0]
            control_points = spline.control_points  # Obtener
los puntos de control
            fit_points = spline.fit_points  # Obtener los
puntos de ajuste
        else:
            print('Forma no válida')
    return control_points,fit_points
```

La función *read_splines* es la que nos permite extraer la información del archivo DXF. Esta función usa la librería *ezdxf* para la manipulación de este tipo de archivos. La función *doc.modelspace()* asigna a la variable msp el espacio de modelos, que es donde están contenidos todos los objetos del archivo.

En el espacio de modelos hay hasta 13 categorías, a continuación, se nombran algunas de las más importantes:

- 'LINE': Segmentos en línea recta que conectan dos puntos del espacio
- 'CIRCLE': Círculos con centro y radio definidos
- 'ARC': Parte de un círculo definido con un ángulo de inicio y fin
- 'SPLINE': Los splines tienen la característica de ser curvas matemáticas suaves, no están compuestos por segmentos rectos o arcos, sino que están definidos por un conjunto de puntos de control que influyen en la forma de la curva.

Las suelas de zapato tienen una forma de *Spline* cerrado. Mediante la función *msp.query('SPLINE'),* podemos extraer el contorno y guardar los puntos de control y los puntos de ajuste que lo definen. En el caso de que la forma fuera recta o circular, la función nos devolvería "Forma no válida". Si se tuviera que adaptar el algoritmo para trabajar con otro tipo de piezas, rectangulares o con arcos, por ejemplo, no habría ningún problema, simplemente habría que adaptar la función para leer ese tipo de objetos y retornar los parámetros que los definen.

El resto del código se implementará en un *Script* principal llamado *Orientation*, el primer paso será cargar las librerías y funciones necesarias.

```
import ezdxf
import matplotlib.pyplot as plt
from shapely.geometry import Polygon, Point
from shapely.affinity import translate,rotate
from functions import *
import random
```

La librería *Shapely* nos permitirá crear objetos y hacer transformaciones geométricas, para después representar los resultados mediante *Matplotlib*. También se importa la librería *ezdxf*, que ya se ha definido anteriormente, así como las funciones auxiliares y una función *random* para generar valores aleatorios de giro

Inicializamos dos variables en las que guardamos las dimensiones de la plancha de acero:

```
h_fig = 400  # mm
w_fig = 600  # mm
```

Cargamos el archivo con la suela y extraemos los parámetros de la *spline* con la función *read_spline(),* definida anteriormente.

```
doc = ezdxf.readfile('./Suela1.dxf')
control_points, fit_points = read_splines(doc)
control_points = [coord[:2] for coord in control_points]
```

Esta función nos devuelve las coordenadas tridimensionales de cada punto de control. Al ser la tercera coordenadas siempre cero, la borramos para todos los puntos para trabajar mejor.

Convertimos los puntos de control en un polígono mediante la función *Polygon*. Esta función es muy interesante ya que nos permite acceder rápidamente a características del polígono como el centroide, que será usado a continuación.

```
pol = Polygon(control_points)
centroid = pol.centroid
cx = centroid.x
cy = centroid.y
```

El polígono creado a partir del archivo “DXF” únicamente contiene los datos del, *Spline*, es importante centrarlo en el espacio de trabajo ya que si no quedaría así al representarlo:

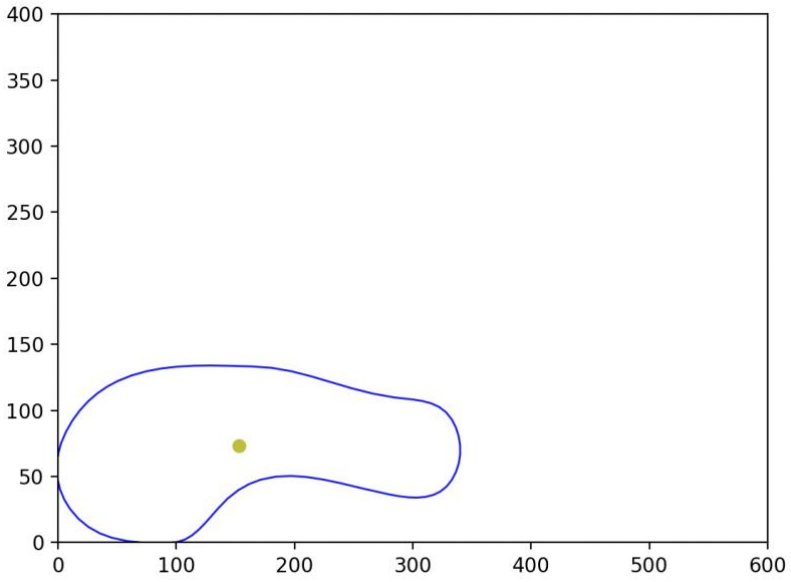


Ilustración 31 Forma sin centrar

La distancia que hay que desplazar el polígono depende de la posición de su centroide, y la calculamos a partir de la función *displacement(),* definida anteriormente

```
tx, ty = displacement(cx, cy, h_fig, w_fig)
```

Una vez calculada la distancia a desplazar, trasladamos el polígono a la posición de trabajo y guardamos las coordenadas del nuevo centroide.

```
pol = translate(pol, tx, ty)
centroid = pol.centroid
cx = centroid.x
cy = centroid.y
```

Es importante actualizar las coordenadas del nuevo centroide ya que son también el centro de giro a la hora de rotar la pieza.

```
pol = rotate(pol, theta_pieza, centroid, use_radians=False)
coord_x, coord_y = pol.exterior.xy
```

Para simular los cambios de orientación de la pieza en un entorno real, se rotará el polígono cierto ángulo Θ. A continuación, definimos 3 círculos que corresponderán a las 3 ventosas. Las coordenadas de cada ventosa están detalladas en los planos del anexo.

```
coord_vent1 = Point(120+w_fig / 2, h_fig / 2)
coord_vent2 = Point(-120+w_fig / 2, h_fig / 2)
coord_vent3 = Point(w_fig / 2, 36+h_fig / 2)
radius = 22
vent1 = coord_vent1.buffer(radius)
vent2 = coord_vent2.buffer(radius)
vent3 = coord_vent3.buffer(radius)
xv1, yv1 = vent1.exterior.xy
xv2, yv2 = vent2.exterior.xy
xv3, yv3 = vent3.exterior.xy
```

Para detectar si todas las ventosas están dentro del polígono, es necesario diseñar un algoritmo iterativo que lo compruebe. Para ello se inicia un bucle *for* con un número máximo de 90 iteraciones. Este número de iteraciones corresponde con el número de giros para dar una vuelta completa, siendo el ángulo de giro 4 grados. Para cada iteración, se comprueba que las 3 ventosas están dentro del polígono mediante la función *pol.contains().* Si no se cumple la condición, se vuelven a girar las ventosas 4 grados, hasta que se cumpla.

```
# Rotacion de las ventosas respecto al centro
num_iterations=90
for i in range(num_iterations):
    if pol.contains(vent1) and pol.contains(vent2) and
pol.contains(vent3):
        print('Ventosas dentro de la pieza')
        print('Numero de iteraciones:',i)
        break
    else:
        center = centroid
        theta_gripper = 4
        vent1 = rotate(vent1, theta_gripper, origin=center,
use_radians=False)
        vent2 = rotate(vent2, theta_gripper, origin=center,
use_radians=False)
        vent3 = rotate(vent3, theta_gripper, origin=center,
use_radians=False)
```

Una vez se cumple la condición, se finaliza el bucle y se muestra un aviso por pantalla indicando el número de iteraciones necesarias.

Por último, se representa la posición final de la pieza con su centroide (color azul y amarillo respectivamente) así como las 3 ventosas (color rojo) mediante la función *plt.plot().*

```
# Coordenadas rotadas ventosas
xv1, yv1 = vent1.exterior.xy
xv2, yv2 = vent2.exterior.xy
xv3, yv3 = vent3.exterior.xy

# Crear un gráfico
fig, ax = plt.subplots()
ax.set_xlim(0, w_fig)
ax.set_ylim(0, h_fig)


# Representar
plt.plot(xv1, yv1, color='r', alpha=0.9, linewidth=1,
solid_capstyle='round', zorder=2)
plt.plot(xv2, yv2, color='r', alpha=0.9, linewidth=1,
solid_capstyle='round', zorder=2)
plt.plot(xv3, yv3, color='r', alpha=0.9, linewidth=1,
solid_capstyle='round', zorder=2)
plt.plot(coord_x, coord_y, color='b', alpha=0.9, linewidth=1,
solid_capstyle='round', zorder=2)
plt.scatter(cx, cy, color='y', label='Centroide')
plt.show()
```

4.3 Posicionamiento del *gripper* en el espacio de trabajo

Las piezas que salen de la máquina de corte no tienen todas la misma posición. Los archivos DXF de cada pieza no nos dan información sobre las coordenadas de éstas respecto al robot, es por ello por lo que debemos hallarlas. En esta fase se desarrollarán algoritmos que, a partir de las imágenes tomadas por la cámara, sitúen el inicio de coordenadas en la pieza a manipular.

Importamos las librerías necesarias:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from functions import center_contour
```

Cv2 es un módulo de la biblioteca *OpenCV (Open Source Computer Vision Library),* que es una biblioteca de código abierto ampliamente utilizada para el procesamiento de imágenes y visión artificial en Python*. OpenCV* proporciona una gran variedad de funciones y herramientas para trabajar con imágenes y videos.

Cargamos la imagen de trabajo mediante la función *imread(),* y aplicamos un algoritmo para pasar la imagen a escala de grises. Trabajar en escala de grises en vez de en color hace mucho más eficiente el procesamiento de detección de bordes.

```
image = cv2.imread('Plancha1.png', cv2.IMREAD_GRAYSCALE)
```

Esta imagen es un renderizado digital y no presenta algunas de las características que tendría una imagen tomada por una cámara real, como puede ser el ruido. Para ajustar los algoritmos del proyecto se ha decido utilizar la imagen incorporando un ruido Gaussiano.

```
# Aplicacion ruido
mean = 50
std_dev = 11 # Ajusta el valor según la intensidad del ruido
deseado
gauss_noise = np.random.normal(mean, std_dev
image.shape).astype(np.uint8)
image = cv2.add(image, gauss_noise)
```

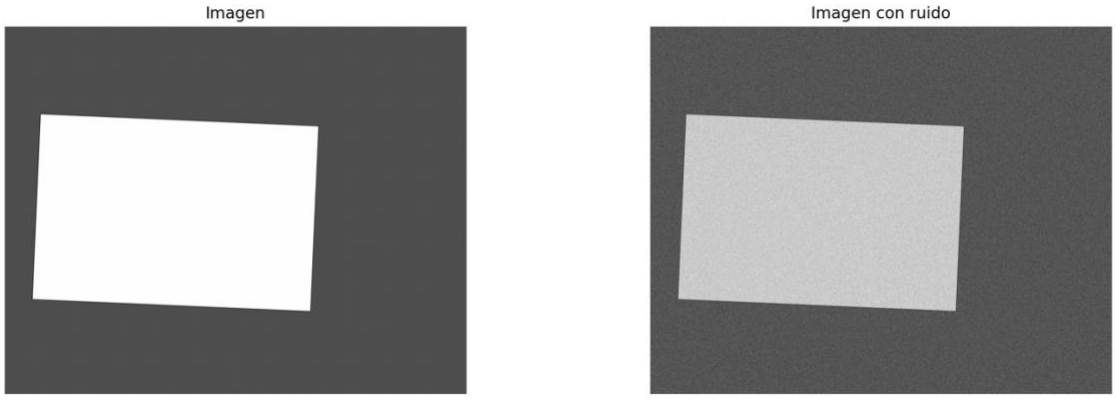


Ilustración 32 Imagen con ruido Gaussiano

Una vez cargada la imagen en escala de grises, podemos proceder con la detección de bordes. Para ello se aplicará el algoritmo *Canny*, que consta de 4 partes:

- **Suavizado:** Se aplica un filtro Gaussiano para eliminar ruido y suavizar la imagen, haciendo más fácil la detección de bordes.
- **Cálculo de los gradientes de intensidad:** La característica más importante de los bordes, es que existe un cambio brusco de intensidad en los píxeles vecinos, en la dirección perpendicular al borde. El operador Sobel aplica técnicas de derivación para hallar la magnitud y la dirección del cambio en la intensidad de píxeles en la imagen.
- **Supresión de no máximos:** Para cada píxel en la imagen, se compara su magnitud de gradiente con la de sus vecinos a lo largo de la dirección del gradiente. Si el píxel tiene el mayor valor de magnitud de gradiente en esa dirección, se mantiene como un candidato a borde; de lo contrario, se suprime. Esto reduce el ancho del borde detectado.
- **Umbralización:** La umbralización se utiliza para convertir los candidatos a bordes en bordes definitivos. Se hace una doble umbralización con un umbral superior y otro inferior. Los píxeles con magnitudes de gradiente por encima del umbral superior se consideran parte de un borde, mientras que los píxeles por debajo del umbral inferior se descartan. Los píxeles con magnitudes de gradiente entre los dos umbrales se mantienen si están conectados a píxeles por encima del umbral superior, lo que ayuda a cerrar los bordes. Estos dos umbrales son los parámetros del filtro que habrá que ajustar manualmente para que la detección sea óptima.

- **Conexión de bordes por histéresis:** A fin de obtener bordes continuos y sin fragmentación, se lleva a cabo una conexión de bordes por histéresis. Esto implica detectar y conectar los píxeles que están por encima del umbral inferior y que están conectados a píxeles por encima del umbral superior.

```
edges = cv2.Canny(image, threshold1, threshold2)
```

Los dos parámetros *threshold* corresponden a los dos umbrales comentados anteriormente, el *threshold1* corresponde al umbral inferior y el *threshold2* al umbral superior. Estos habrá que ajustarlos manualmente para detectar los bordes.

Para umbrales bajos, el algoritmo detecta el ruido como borde:

```
threshold1=80
threshold2=120
```

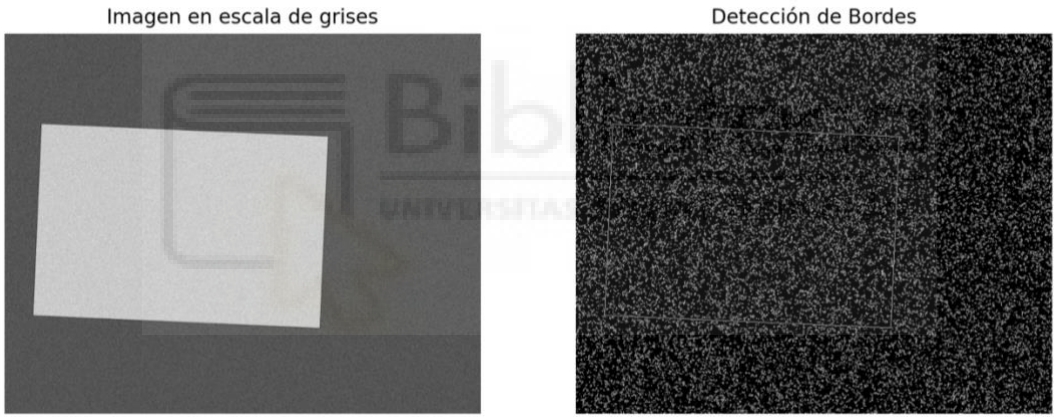


Ilustración 33 Detección de bordes 1

```
threshold1=100
threshold2=140
```

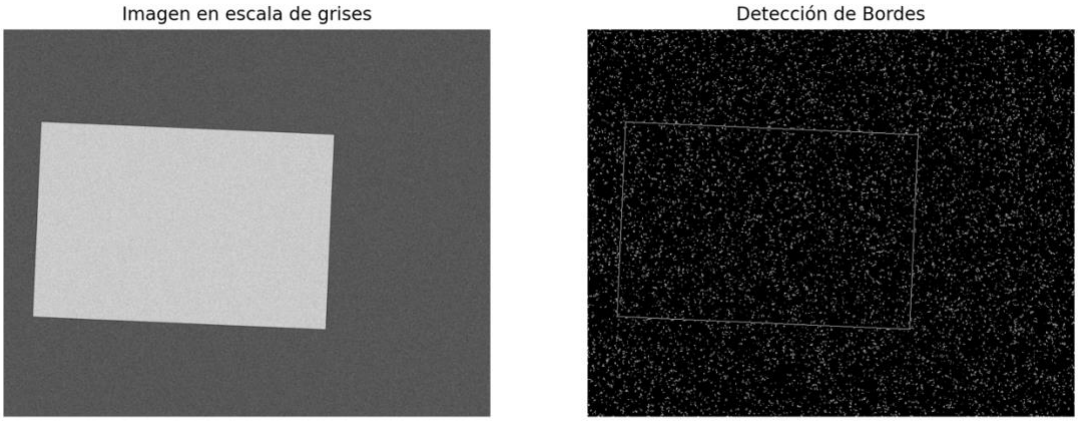


Ilustración 34 Detección de bordes 2

Finalmente, para valor de 220 (Umbral 1) y 240 (Umbral 2), se detecta un borde continuo sin apenas ruido.

```
threshold1=220
threshold2=240
```

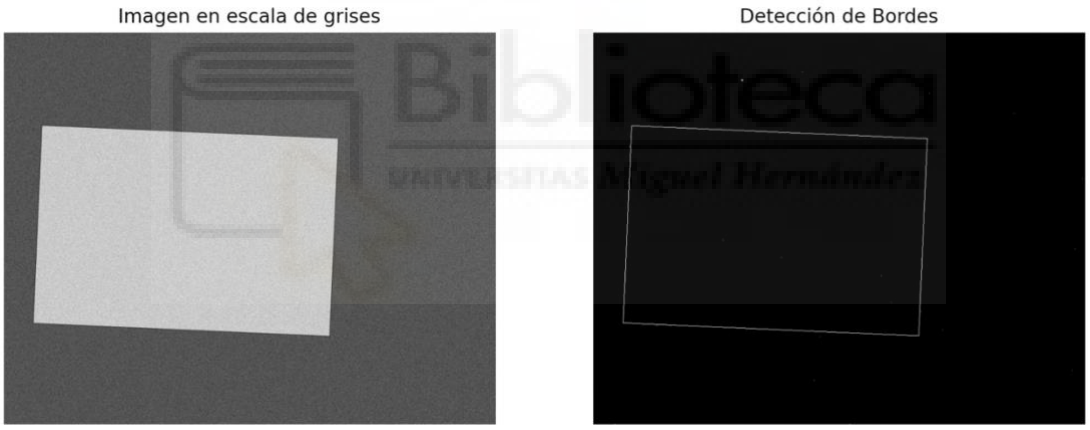


Ilustración 35 Detección de bordes 3

En un entorno real, en el que el borde quizá no sea tan evidente, habría que ser mucho más cuidadoso para la selección de los umbrales ya que un umbral muy alto podría conllevar una discontinuidad en el borde.

Una vez detectados los bordes en la imagen, podemos extraerlos en formato contorno, con el que será más fácil trabajar. Para ello usamos la función *find.contours()* de cv2, y guardamos en una variable el contorno de mayor tamaño. Esto evita posibles errores si se ha detectado bordes pequeños debido al ruido o imperfecciones.

```
# Encontrar contornos en los bordes detectados
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
            cv2.CHAIN_APPROX_SIMPLE)

# Encontrar el contorno más grande, el borde
rectangle = max(contours, key=cv2.contourArea)
centroid_rectangle =center_contour(rectangle)
```

Después, hallamos el centroide del contorno mediante la siguiente función auxiliar:

```
def center_contour(contour):
    M = cv2.moments(contour)
    center_x = int(M['m10'] / M['m00'])
    center_y = int(M['m01'] / M['m00'])
    return [center_x, center_y]
```

Una vez hallado tanto el rectángulo que forman los bordes como su centroide, se debe definir una matriz de transformación que relacione las coordenadas del rectángulo con las coordenadas globales.
Para ellos se aplicará un algoritmo de detección de esquinas sobre el contorno del borde.

```
# Deteccion de esquinas
corners = cv2.approxPolyDP(rectangle,
            0.04 * cv2.arcLength(rectangle, True), True)
```

A partir de las coordenadas de las esquinas, podemos calcular la distancia entre ellas y representar todo en una nueva imagen.

```
# Dimensiones del rectangulo de esquinas
x, y, w, h = cv2.boundingRect(corners)
image_edges = np.zeros((image.shape[0],
                        image.shape[1], 1), dtype=np.uint8)
cv2.drawContours(image_edges, [corners], -1, (255, 0, 0), 1)
cv2.circle(image_edges, tuple(centroid_rectangle), 4, (255, 0, 0),
                        -1)

# Dibujar las esquinas del rectángulo en la imagen
for corner in corners:
    cv2.circle(image_edges, tuple(corner[0]), 5, (255, 0, 0), -1)
```

A partir de las coordenadas de las esquinas podemos hallar la matriz de transformación mediante la función *cv2.getPerspectiveTransform(),* que relaciona estas coordenadas con las que tendría el rectángulo si tuviera el origen de coordenadas en (0,0).

```
# Crear la matriz de transformación afín para cambiar de coordenadas
dst_pts = np.array([corners[0], corners[1], corners[2], corners[3]],
                   dtype=np.float32)
src_pts = np.array([(0, 0), (0, h), (w, h), (w, 0)],
                   dtype=np.float32)
transform_matrix = cv2.getPerspectiveTransform(src_pts, dst_pts)
print('Matriz de transformacion:',transform_matrix)
print('Centroide:',centroid_rectangle)
```

Finalmente, representamos ambas imágenes, la tomada por la cámara y la que contiene los bordes, centroide y esquinas:

```
# Tamano de las figuras
plt.figure(figsize=(16, 5))

# Crear cuadricula de parcelas con una fila y dos columnas, imagen
se muestra en parcela 1
plt.subplot(121)
plt.imshow(image, cmap='gray')
plt.title('Imagen en escala de grises')
plt.axis('off')

# Segunda parcela
plt.subplot(122)
plt.imshow(image_edges, cmap='gray')
plt.title('Detección de Bordes')
plt.axis('off')

#Mostrar graficos
plt.show()
```

5 Resultados

5.1 Rotación

A continuación, se muestran los resultados de rotar aleatoriamente la pieza. El valor aleatorio se genera mediante *random.uniform()* en un intervalo de 0 a 360 grados.

- Prueba 1

Forma válida
Angulo de giro: 307
Ventosas dentro de la pieza
Numero de iteraciones: 76

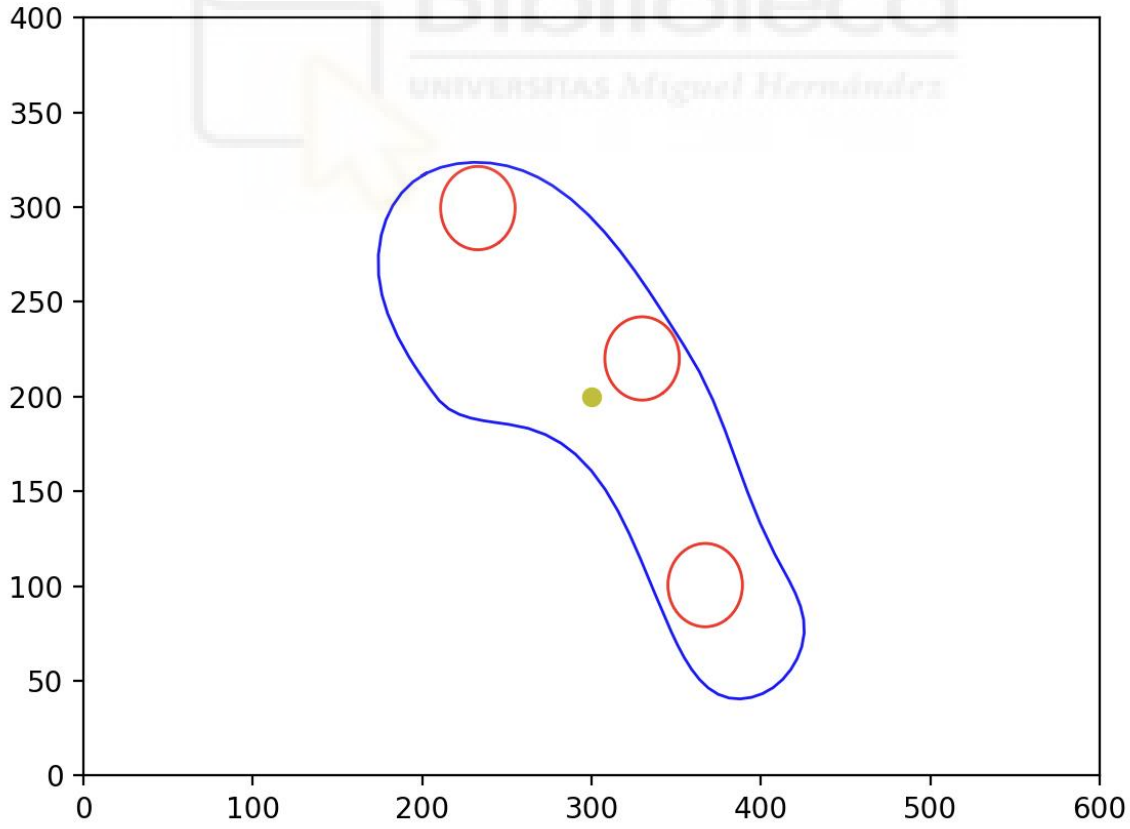


Ilustración 36 Prueba 1 rotación

- Prueba 2

  Forma válida

  Angulo de giro: 350

  Ventosas dentro de la pieza

  Numero de iteraciones: 87

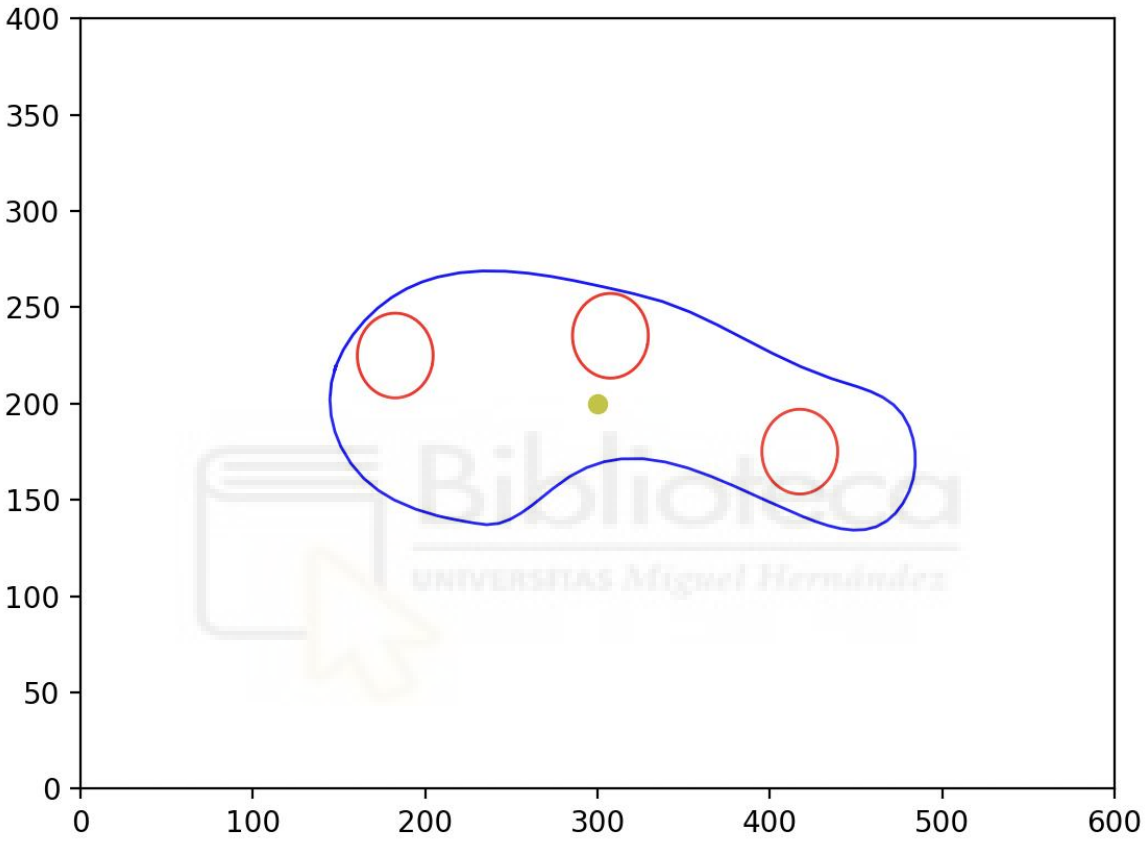


Ilustración 37 Prueba 2 rotación

- Prueba 3

Forma válida

Angulo de giro: 71

Ventosas dentro de la pieza

Numero de iteraciones: 17

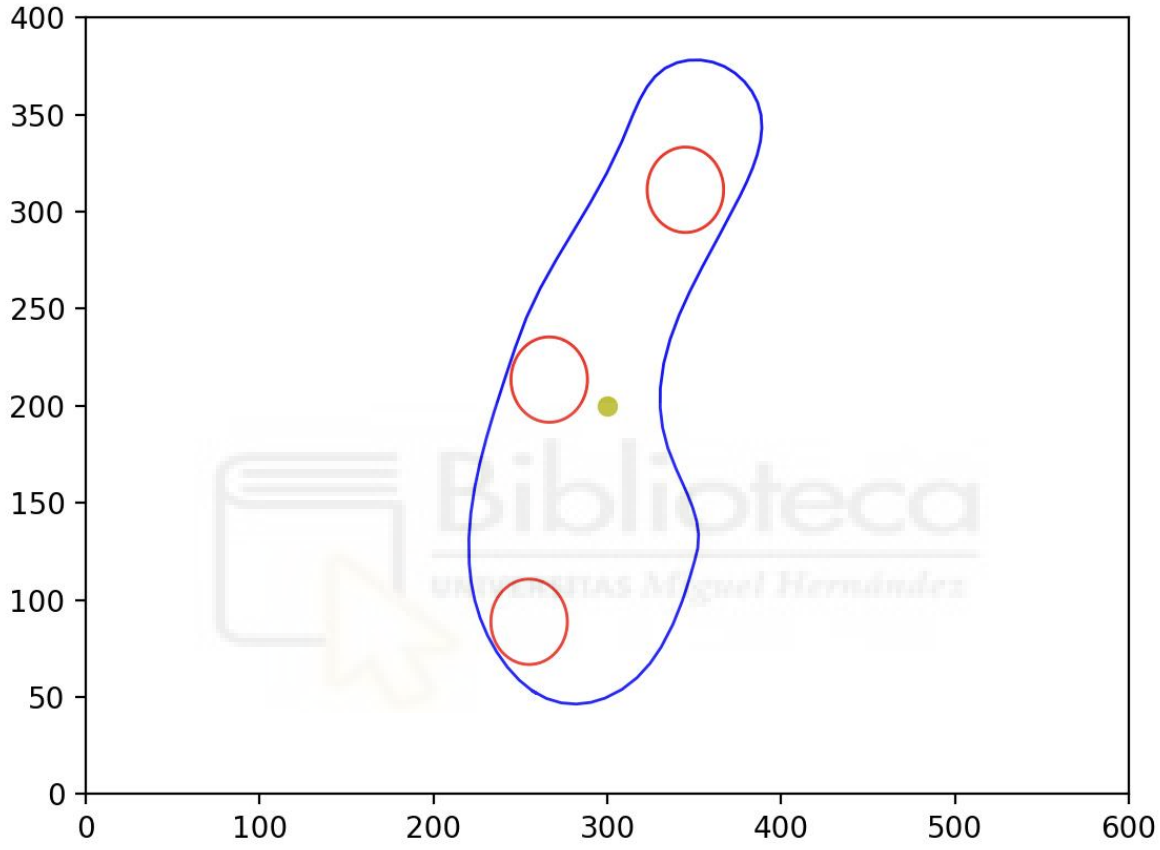


Ilustración 38 Prueba 3 rotación

- Prueba 4

Forma válida

Angulo de giro: 246

Ventosas dentro de la pieza

Numero de iteraciones: 61

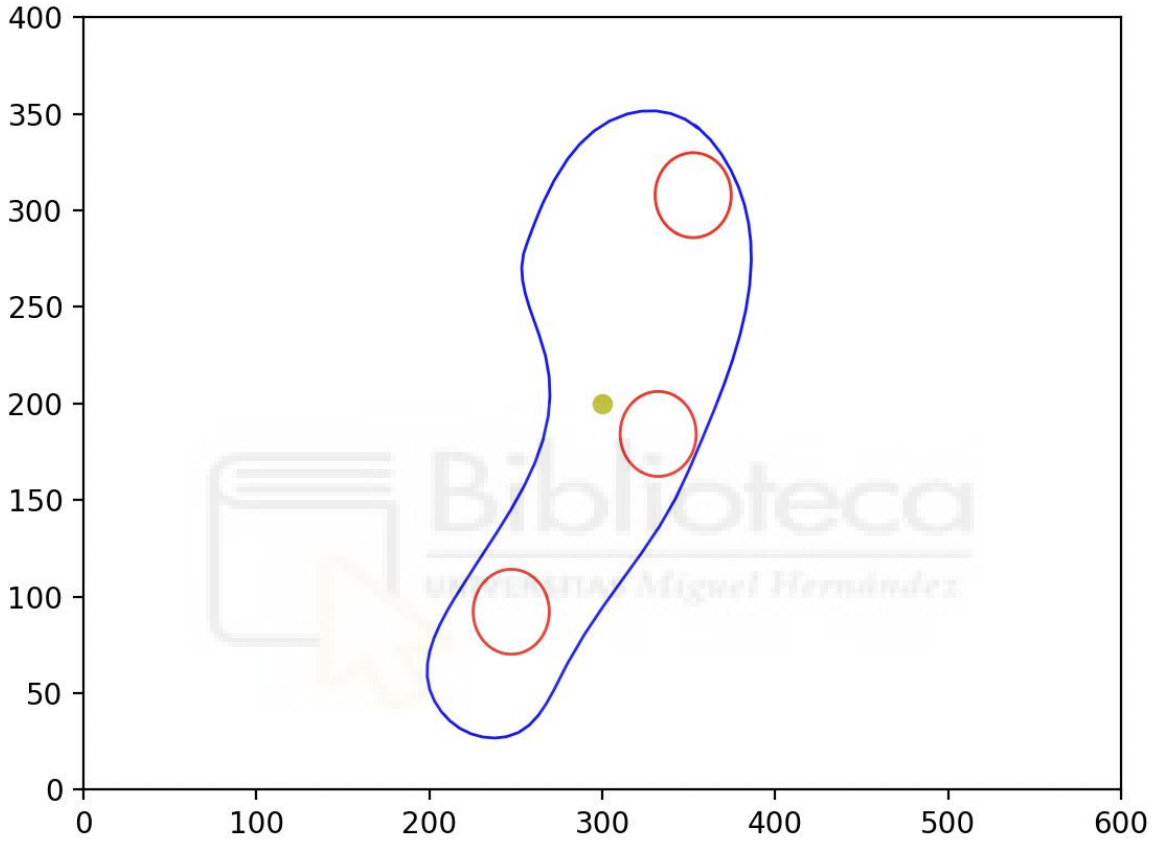


Ilustración 39 Prueba 4 rotación

5.2 Traslación

Los resultados de este código para la operación de traslación serán, una matriz de transformación que relaciona las coordenadas globales con las de la pieza, así como su centroide. Además, se representarán las imágenes generadas por el código.

- Prueba 1

Imagen: Plancha1
Matriz de transformación: [[9.731e-01 -4.070e-02 7.600e+01]
 [4.370e-02 9.379e-01 1.860e+02]
 [0.000e+00 0.000e+00 1.000e+00]]
Centroide: [361, 394]

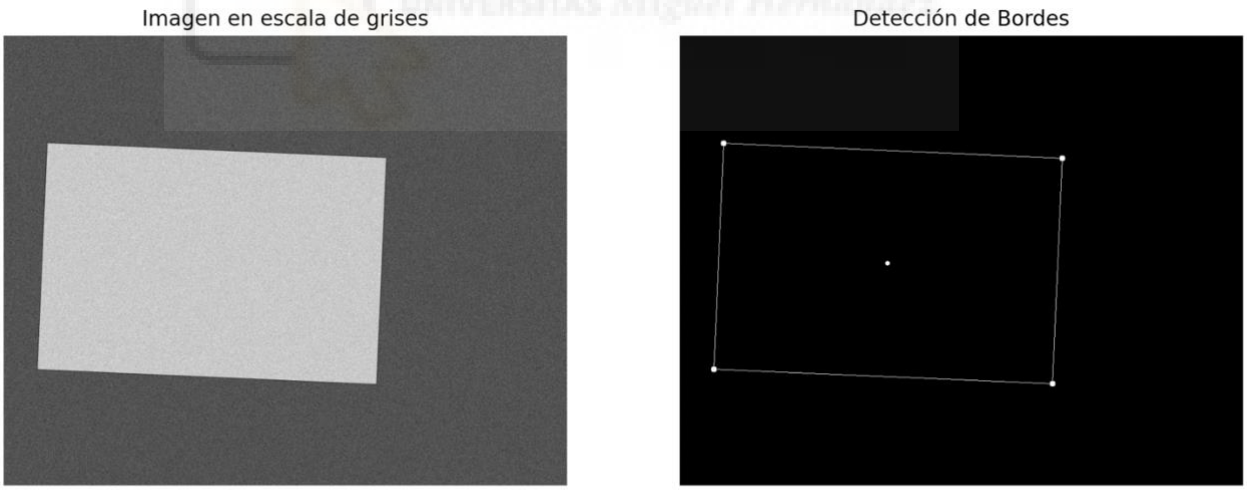


Ilustración 40 Prueba 1 traslación

- Prueba 2

Imagen: Plancha2

Matriz de transformación: [[2.0800e-02 -1.4184e+00 8.7900e+02]
 [6.5020e-01 5.2100e-02 1.9200e+02]
 [-0.0000e+00 0.0000e+00 1.0000e+00]]

Centroide: [591, 398]

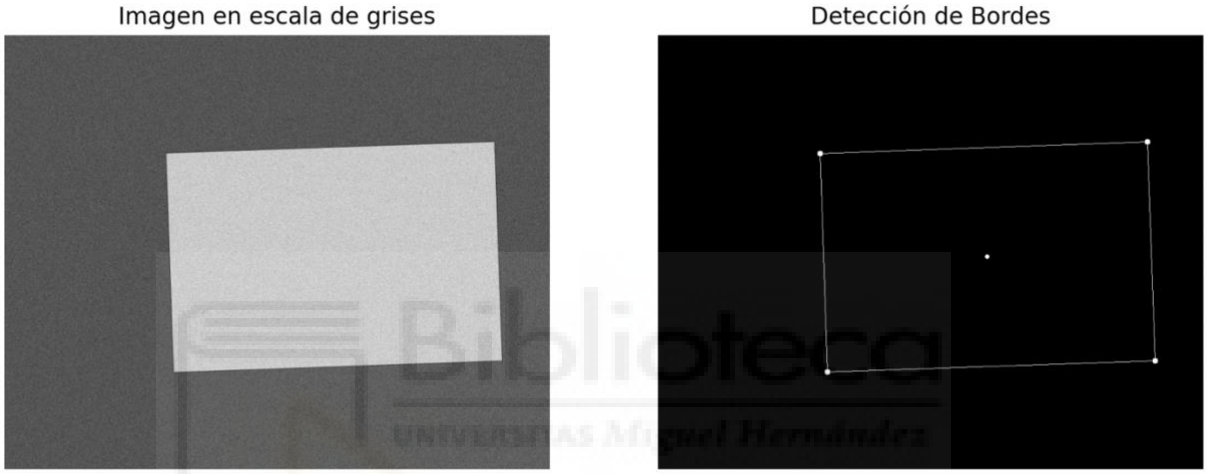


Ilustración 41 Prueba 2 traslación

- Prueba 3

Matriz de transformación: [[9.781e-01 -2.450e-02 2.600e+02]
 [2.360e-02 9.584e-01 3.130e+02]
 [-0.000e+00 -0.000e+00 1.000e+00]]
Centroide: [549, 516]

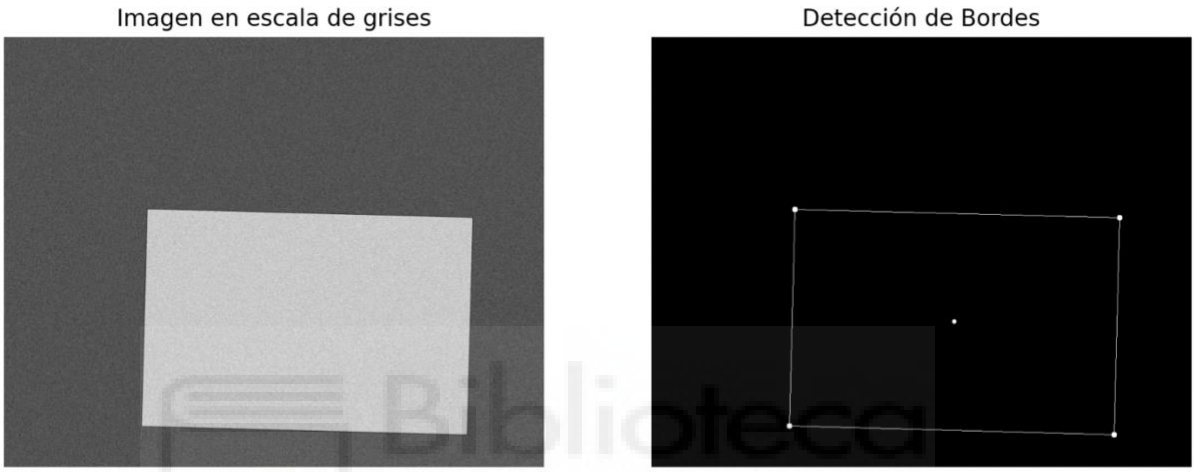


Ilustración 42 Prueba 3 traslación

- Prueba 4

Matriz de transformación: [[9.916e-01 -1.110e-02 2.570e+02]
 [8.400e-03 9.827e-01 1.400e+02]
 [0.000e+00 -0.000e+00 1.000e+00]]
Centroide: [549, 338]

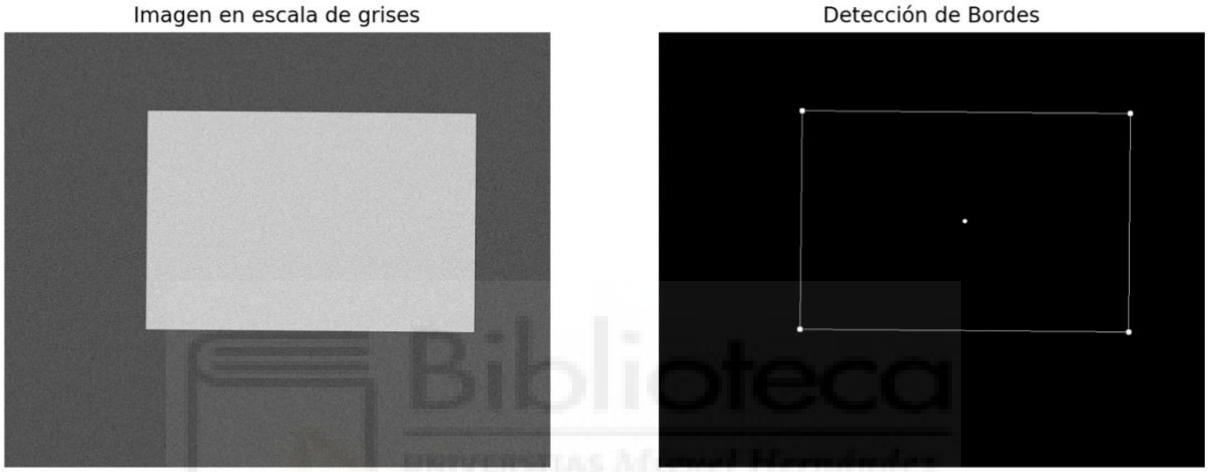


Ilustración 43 Prueba 4 traslación

6 Conclusiones y trabajos futuros

En este capítulo se exponen las conclusiones a las que se ha llegado al terminar este proyecto de fin de máster, así como algunas líneas de investigación para futuros trabajos.

6.1 Conclusiones

En este Trabajo de Fin de Máster se ha abordado de manera práctica un desafío muy común en la industria, como es la manipulación precisa de piezas. Para ello se ha diseñado con éxito un *gripper* con ventosas como método de sujeción, que junto al código en Python permite manipular piezas adaptándose a su posición y orientación en el espacio. Todas las decisiones tomadas en lo relativo al diseño han sido justificadas en el capítulo correspondiente. En la parte de programación, se ha realizado un programa sólido y funcional, que ha funcionado correctamente en las pruebas de rotación y traslación. Este código es válido para un primer test, aunque habrá que hacer modificaciones para la puesta en marcha del sistema.

Con un enfoque interdisciplinar, a través de una combinación de mecánica, programación y visión por computador, se ha desarrollado un sistema de agarre robótico adaptable y eficiente. Una de las ventajas de este trabajo es que sienta las bases para el desarrollo de otro tipo de *grippers,* pudiéndose adaptar a diferentes tamaños, formas, o tipos de piezas.

6.2 Trabajos futuros

En primer lugar, se propone realizar la puesta en marcha del sistema, para ello será necesario juntar los dos códigos, el de rotación y traslación, e implementarlo en la controladora del robot a fin de que este se mueve a la posición requerida cuando sea necesario. Una vez se hallan implementado los dos códigos, habrá que modificarlo de manera que pueda trabajar en tiempo real, tomando capturas de la cámara cada vez que el robot tenga que coger una pieza nueva. Para la puesta en marcha del sistema habrá que hacer numerosas pruebas a fin de buscar errores o formas de optimizar el código.

Una vez el sistema funcione con éxito, se propone adaptar el sistema para manipular otro tipo de piezas más complejas, considerando la posibilidad de incluir más ventosas si las piezas fuesen más grandes o tuviesen una forma distinta. En el caso de piezas complejas, se propone incluir el uso de otro tipo de sensores, como sensores láser, a fin de mejorar la percepción del entorno y la precisión del robot. Se propone también estudiar la posibilidad de aplicar esta tecnología en un entorno colaborativo, en el que un operador interactúe con el robot en la manipulación de las piezas. Para implementar este enfoque es necesario desarrollar protocolos que garanticen la seguridad del operario frente al robot.

7 Referencias

[1] Richard Szeliski, University of Washington, 2022: *Computer Vision: Algorithms and Applications*, *2nd edition*

[2] Comité Español de Automática 2016: *Conceptos y Métodos en Visión por Computador*

[3] Charles R. Severance,2016: *Python for everybody*

[4] Matplotlib — Visualization with Python. (s. f.). https://matplotlib.org

[5] Numpy — Bioinformatics at COMAV 0.1 documentation. (s. f.). https://bioinf.comav.upv.es/courses/linux/python/scipy.html

[6] José Antonio Ibáñez Maciá, 2023: *Programación gripper de vacío matricial*

[7] Robot SCARA: *www.epson.es*

[8] Cámara ELP: *www.aliexpress.es*

[9] Comelz, fabricación de calzado y textil. Máquina de corte textil: *www.comelz.es*

[10] Universal Robots, Tipos de *grippers* robóticos: *www.universalrobots.com*

[11] Multi Sytems, soluciones de inspección por visión artificial para la industria: www.*multisystems.com*

[12] Gimatic, sistemas de vacío*: www.gimatic.com*

[13] Vuototecnica, accesorios de vacío, ventosas y portaventosas: *www.vuototecnica.es*

[14] Norelem, piezas estandarizadas: *www.norelem.es*

[15] Gareth J. Monkman, Stefan Hesse, Ralf Steinmann, Henrik Schunk, 2007: Robot grippers

[16] Estadística en la industria www.statista.com

8 Anexos

8.1 Script *Functions*

```
import cv2

def read_splines(doc):
    msp = doc.modelspace()
    for e in msp:
        if e.dxftype() == 'SPLINE':
            spline = msp.query('SPLINE')[0]
            control_points = spline.control_points  # Obtener los puntos de
                                    #control
            fit_points = spline.fit_points  # Obtener los puntos de ajuste
            print('Forma válida')
    return control_points,fit_points

def displacement(centroid_x,centroid_y,h_fig,w_fig):
    tx = w_fig / 2 - centroid_x
    ty = h_fig / 2 - centroid_y
    return tx, ty

def center_contour(contour):
    M = cv2.moments(contour)
    center_x = int(M['m10'] / M['m00'])
    center_y = int(M['m01'] / M['m00'])
    return [center_x, center_y]
```

8.2 Script *Rotation*

```
import ezdxf
import matplotlib.pyplot as plt
from shapely.geometry import Polygon, Point
from shapely.affinity import translate,rotate
from functions import *
import random

# Cargar imagen y obtener sus dimensiones
image = cv2.imread('Plancha1.png')
#h_fig, w_fig, channels = image.shape

h_fig = 400  # mm
w_fig = 600  # mm

# Cargar forma dxf y extraer parámetros
doc = ezdxf.readfile('./Suela1.dxf')
control_points, fit_points = read_splines(doc)
control_points = [coord[:2] for coord in control_points] # Elimina tercera
#componente

# Creamos poligono con los puntos de control y guardamos las coordenadas de
#su centroide
pol = Polygon(control_points)
centroid = pol.centroid
cx = centroid.x
cy = centroid.y

# Calcula el desplazamiento necesario para centrar el poligono
tx, ty = displacement(cx, cy, h_fig, w_fig)

# Trasladar y rotar el poligono
pol = translate(pol, tx, ty)
centroid = pol.centroid
cx = centroid.x
cy = centroid.y
theta_pieza = int(random.uniform(0, 360))
print('Angulo de giro:', theta_pieza)
pol = rotate(pol, theta_pieza, centroid, use_radians=False)
coord_x, coord_y = pol.exterior.xy

# Ventosas
coord_vent1 = Point(120+w_fig / 2, h_fig / 2)
coord_vent2 = Point(-120+w_fig / 2, h_fig / 2)
coord_vent3 = Point(w_fig / 2, 36+h_fig / 2)
radius = 22
vent1 = coord_vent1.buffer(radius)
vent2 = coord_vent2.buffer(radius)
vent3 = coord_vent3.buffer(radius)
xv1, yv1 = vent1.exterior.xy
xv2, yv2 = vent2.exterior.xy
xv3, yv3 = vent3.exterior.xy
```

```
# Rotacion de las ventosas respecto al centro
num_iterations=90
for i in range(num_iterations):
    if pol.contains(vent1) and pol.contains(vent2) and pol.contains(vent3):
        print('Ventosas dentro de la pieza')
        print('Numero de iteraciones:',i)
        break
    else:
        center = centroid
        theta_gripper = 4
        vent1 = rotate(vent1, theta_gripper, origin=center,
               use_radians=False)
        vent2 = rotate(vent2, theta_gripper, origin=center,
               use_radians=False)
        vent3 = rotate(vent3, theta_gripper, origin=center,
               use_radians=False)

# Coordenadas rotadas ventosas
xv1, yv1 = vent1.exterior.xy
xv2, yv2 = vent2.exterior.xy
xv3, yv3 = vent3.exterior.xy

# Crear un gráfico
fig, ax = plt.subplots()
ax.set_xlim(0, w_fig)
ax.set_ylim(0, h_fig)

# Representar
plt.plot(xv1, yv1, color='r', alpha=0.9, linewidth=1,
         solid_capstyle='round', zorder=2)
plt.plot(xv2, yv2, color='r', alpha=0.9, linewidth=1,
         solid_capstyle='round', zorder=2)
plt.plot(xv3, yv3, color='r', alpha=0.9, linewidth=1,
         solid_capstyle='round', zorder=2)
plt.plot(coord_x, coord_y, color='b', alpha=0.9,
         linewidth=1, solid_capstyle='round', zorder=2)
plt.scatter(cx, cy, color='y', label='Centroide')
plt.show()

plt.scatter(cx, cy, color='y', label='Centroide')
plt.show()
```

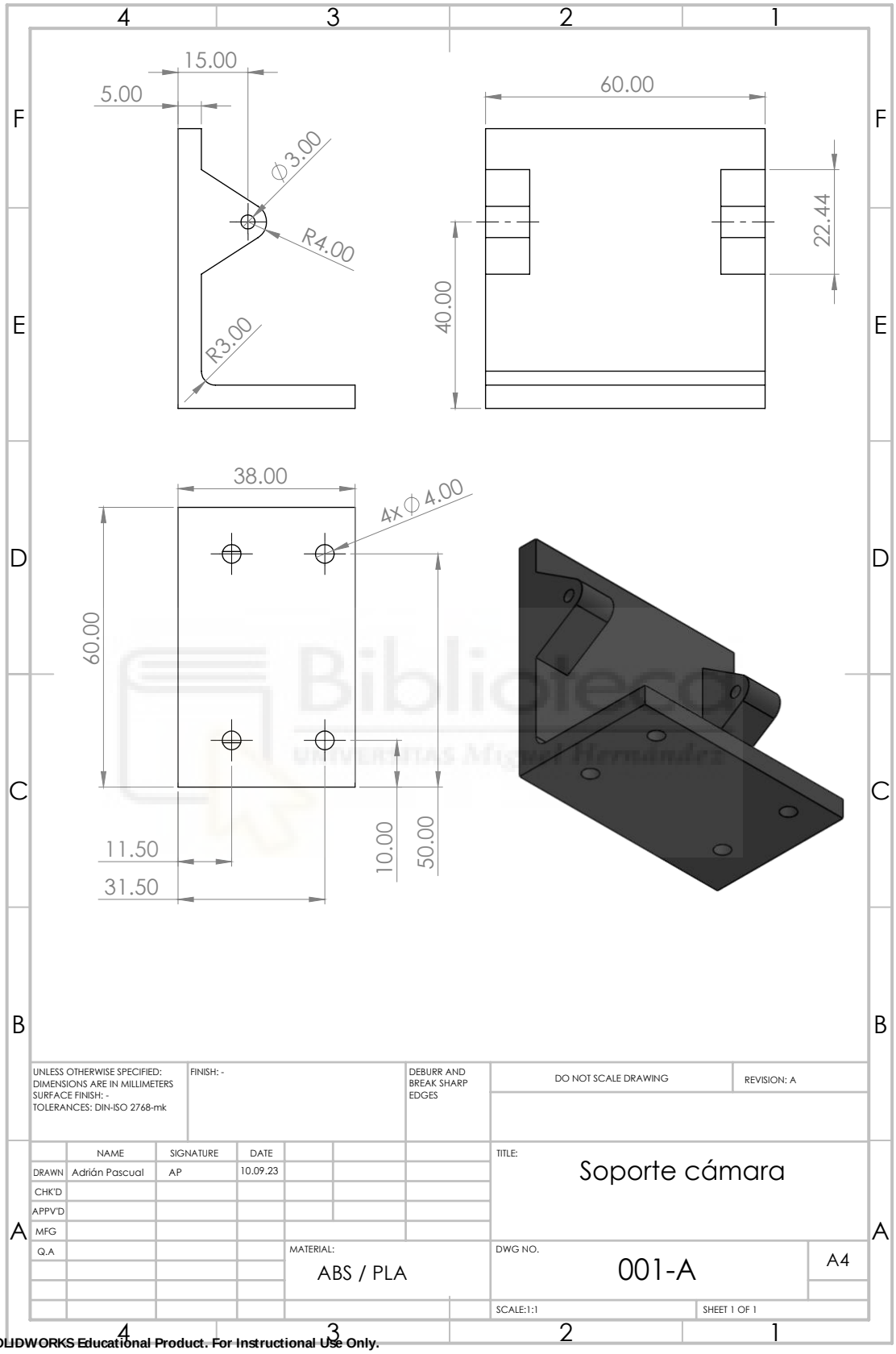

8.3 Script *Translation*

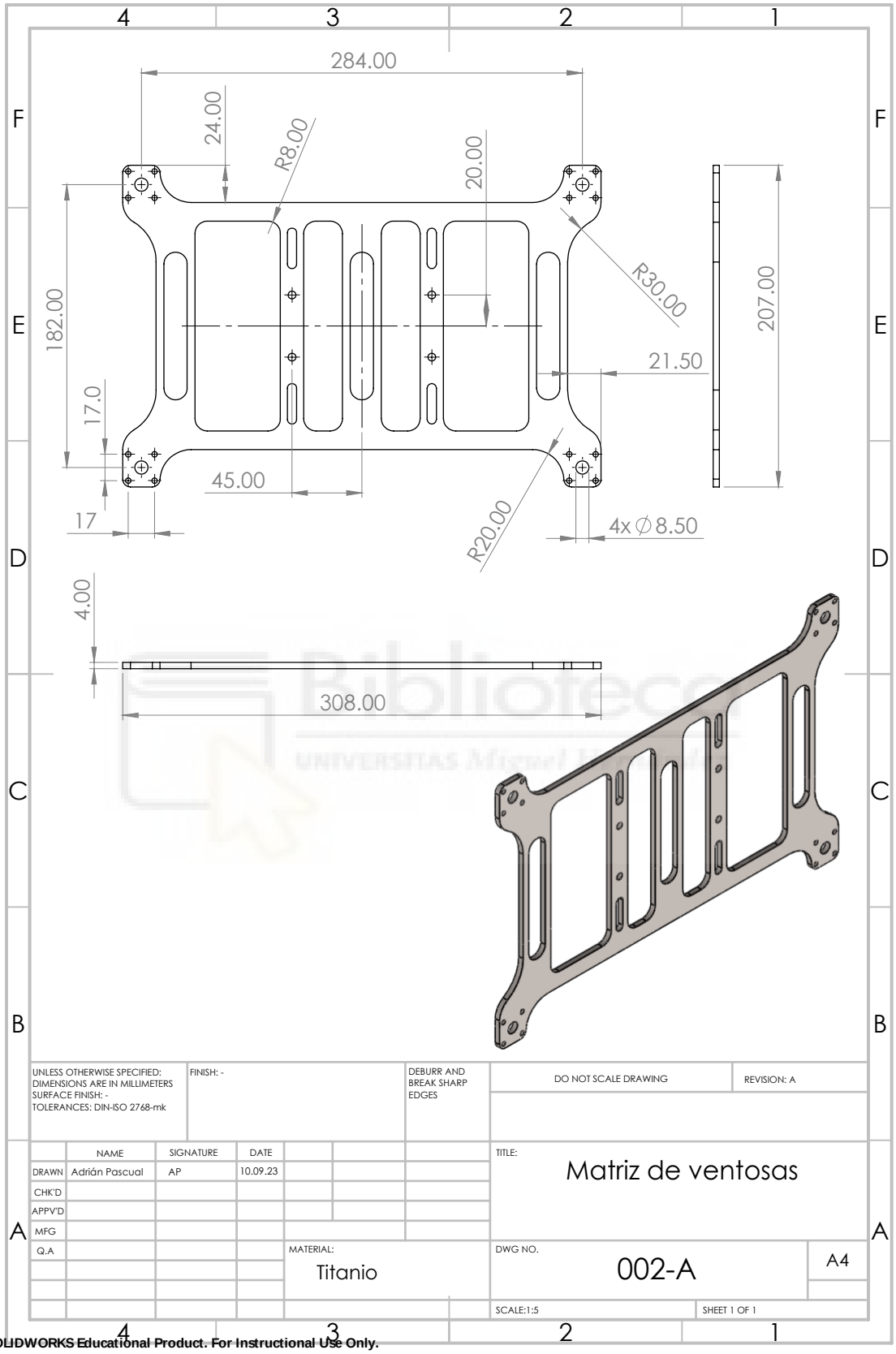
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from functions import center_contour

# Cargar imagen en escala de grises
image = cv2.imread('Plancha1.png', cv2.IMREAD_GRAYSCALE)

# Aplicacion ruido
mean = 50
std_dev = 11 # Ajusta el valor según la intensidad del ruido deseado
gauss_noise = np.random.normal(mean, std_dev, image.shape).astype(np.uint8)
image = cv2.add(image, gauss_noise)


# Aplicar un filtro de Canny para detección de bordes
edges = cv2.Canny(image, threshold1=220, threshold2=240)

# Encontrar contornos en los bordes detectados
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Encontrar el contorno más grande, el borde y hallar su centroide
rectangle = max(contours, key=cv2.contourArea)
centroid_rectangle =center_contour(rectangle)

# Deteccion de esquinas
corners = cv2.approxPolyDP(rectangle, 0.04 * cv2.arcLength(rectangle,
                                                           True), True)

# Dimensiones del rectangulo de esquinas
x, y, w, h = cv2.boundingRect(corners)
image_edges = np.zeros((image.shape[0],
                        image.shape[1], 1), dtype=np.uint8)
cv2.drawContours(image_edges, [corners], -1, (255, 0, 0), 1)
cv2.circle(image_edges, tuple(centroid_rectangle), 4, (255, 0, 0), -1)


# Dibujar las esquinas del rectángulo en la imagen
for corner in corners:
    cv2.circle(image_edges, tuple(corner[0]), 5, (255, 0, 0), -1)

# Crear la matriz de transformación afín para cambiar de coordenadas
dst_pts = np.array([corners[0], corners[1], corners[2], corners[3]],
dtype=np.float32)
src_pts = np.array([(0, 0), (0, h), (w, h), (w, 0)], dtype=np.float32)
transform_matrix = cv2.getPerspectiveTransform(src_pts, dst_pts)
transform_matrix = np.round(transform_matrix,4)
print('Matriz de transformacion:',transform_matrix)
print('Centroide:',centroid_rectangle)

# Tamano de las figuras
plt.figure(figsize=(16, 5))
```

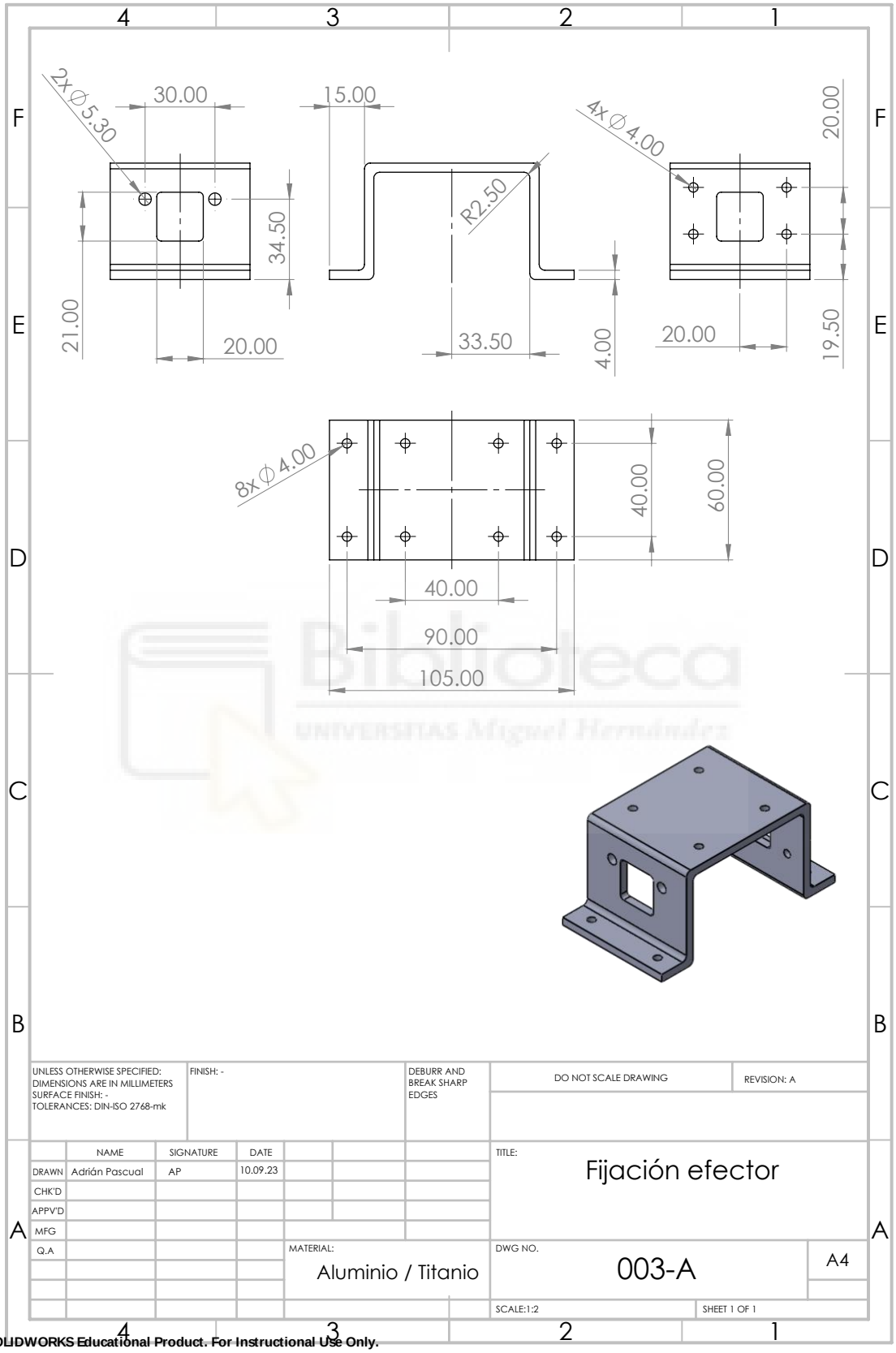
```
# Crear cuadricula de parcelas con una fila y dos columnas, imagen se
muestra en parcela 1
plt.subplot(121)
plt.imshow(image, cmap='gray')
plt.title('Imagen en escala de grises')
plt.axis('off')

# Segunda parcela
plt.subplot(122)
plt.imshow(image_edges, cmap='gray')
plt.title('Detección de Bordes')
plt.axis('off')

#Mostrar graficos
plt.show()
```

8.4 Planos

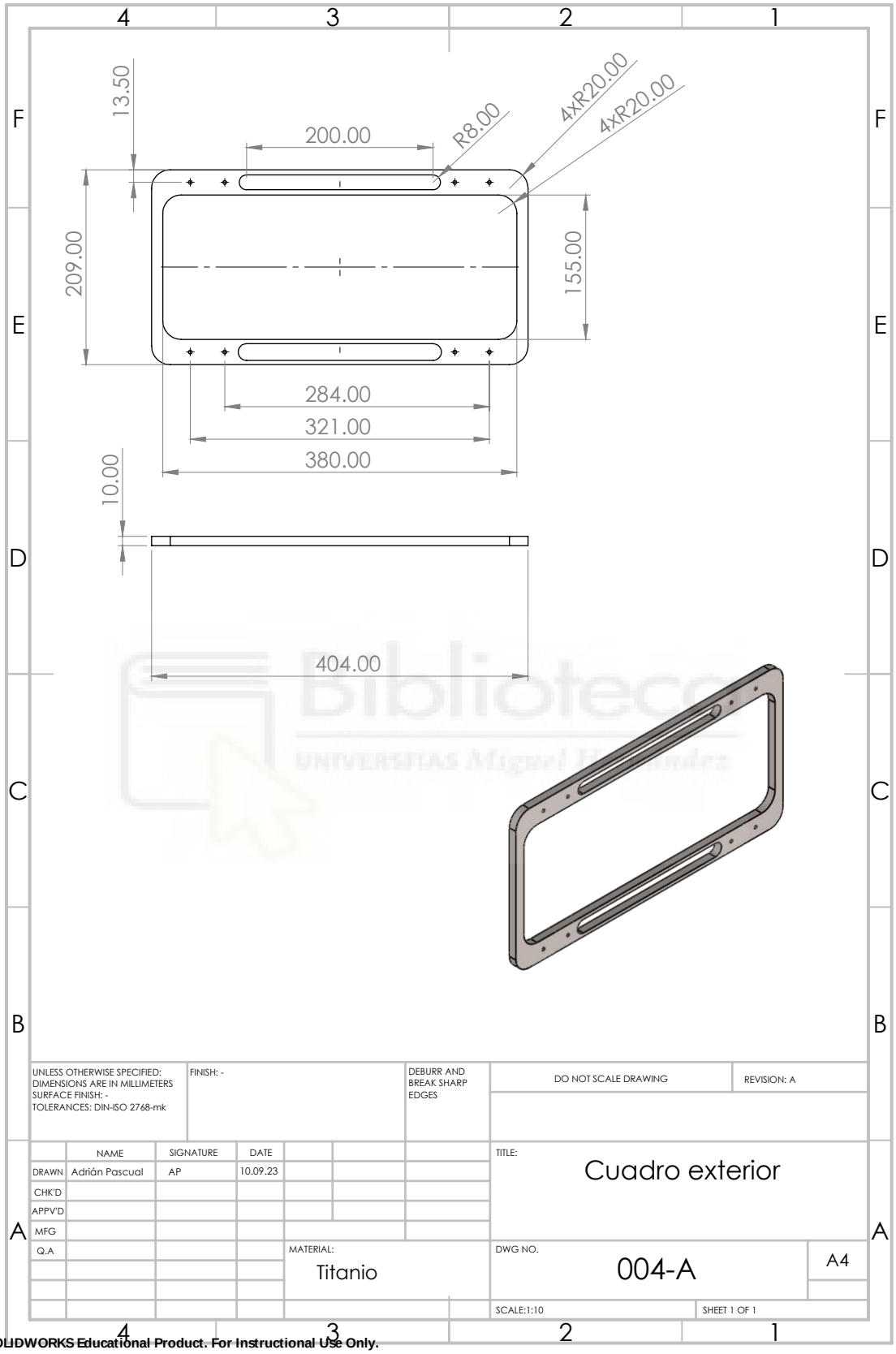
A continuación, se muestran los planos de las cuatro piezas a fabricar, así como un plano del ensamblaje final en formato explosión.

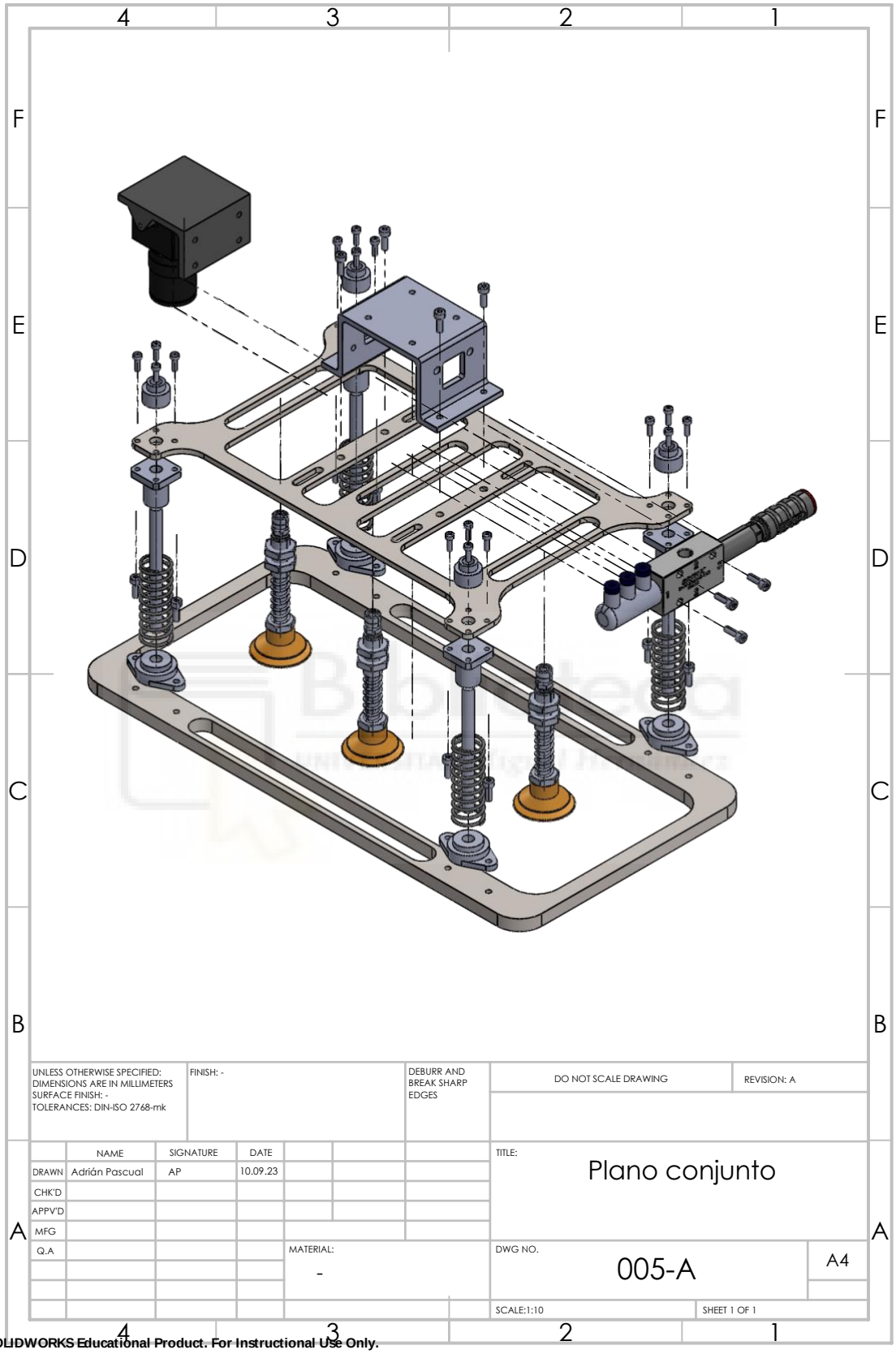






UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: - TOLERANCES: DIN-ISO 2768-mk		FINISH: -		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION: A	
	NAME	SIGNATURE	DATE			TITLE:		
DRAWN	Adrián Pascual	AP	10.09.23			Fijación efector		
CHK'D								
APPV'D								
MFG								
Q.A				MATERIAL:		DWG NO.		A4
				Aluminio / Titanio		003-A		
						SCALE:1:2	SHEET 1 OF 1	





UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: - TOLERANCES: DIN-ISO 2768-mk		FINISH: -		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION: A	
	NAME	SIGNATURE	DATE		TITLE:		
DRAWN	Adrián Pascual	AP	10.09.23		Plano conjunto		
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:	DWG NO.		A4
				-	005-A		
					SCALE:1:10	SHEET 1 OF 1	