UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

PROGRAMACIÓN GRIPPER DE VACÍO MATRICIAL

TRABAJO FIN DE GRADO

Junio -2023

AUTOR: José Antonio Ibáñez Maciá

DIRECTOR/RES: Carlos Pérez Vidal

Francisco José Martínez Peral

UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE
ESCUELA POLITÉCNICA SUPERIOR DE ELCHE
GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

TRABAJO FIN DE GRADO
JUNIO - 2023

PROGRAMACIÓN GRIPPER DE VACÍO MATRICIAL

AUTOR: JOSÉ ANTONIO IBÁÑEZ MACIÁ
DIRECTOR/ES: CARLOS PÉREZ VIDAL
FRANCISCO JOSÉ MARTÍNEZ PERAL

ÍNDICE

1. RESUMEN

La tecnología se encuentra en un proceso de innovación y optimización constante, llegando a la introducción de las pinzas de vacío. Se ha trabajado con este elemento para darle un mejor uso, ya que está siendo introducido a la industria por empresas de diversos sectores.

A la hora de realizar este trabajo, se ha buscado la mejor forma de emplear las ventosas de la matriz. Para ello, se ha buscado utilizar la mayor cantidad de ventosas posibles para la sujeción de un objeto, llevándose a cabo mediante programación en código Python.

La hipótesis empleada a la hora de realizar la investigación es que, si se detecta un punto dentro de una figura, se puede extrapolar a más puntos, haciendo que aquellos que estén dentro se activen.

Se empezó buscando si un punto se encontraba en el interior de una figura y a partir de eso, avanzar hasta llegar al resultado obtenido. Una matriz que se le activan o desactivan las ventosas, dependiendo de si estas están dentro o fuera de la figura.

Al realizar las pruebas, ciertas piezas permiten obtener más de una solución válida, pudiendo escoger una matriz de las opciones válidas para agarrar el objeto.

Palabras clave: Software, Matriz, Código, Robot, Python, Vacío, Gripper, Ventosas, Lectura ficheros DXF, Gripper matricial, Código Python grippers

2. INTRODUCCIÓN

Las máquinas se encuentran en una constante evolución, adaptándose a las nuevas tecnologías y desarrollando métodos específicos a las funciones a desempeñar y materiales con los que trabajar.

La industria tiene presente sistemas de ventosas para capa completa, destinado al paletizado en almacenes y logística interna. Sus garras pueden ser hechas a medida para el robot, diseñadas para aplicaciones específicas. Las empresas se encuentran en pleno desarrollo para la utilización de ventosas a vacío en diversas áreas, además del paletizado.

En el siguiente trabajo, se ha querido mostrar como un brazo robot matricial se adapta a un objeto, agarrándolo con el mayor número de ventosas posibles mediante un algoritmo realizado en Python. En él se enseña todo el proceso, mostrando el hardware empleado y el software realizado.

2.1. OBJETIVO

El objetivo consiste en la programación de un brazo robot matricial ventosa mediante lenguaje Python, con el fin de poder coger objetos con el mayor número de ventosas posibles.

Estos objetos son representados en AutoCAD empleando sus contornos como líneas, permitiendo trabajar con ellos en Python.

2.2. ESTADO DEL ARTE

Los robots industriales poseen diversos tipos de sistemas de agarre, adaptándose cada uno a una función específica para sacarle el mayor partido. Este caso debe ser enfocado a los gripper o pinzas a vacío, pero ¿Qué son los gripper?

Los gripper, conocidos como pinzas, son identificados como herramientas EOAT (End of Arm Tooling). Las pinzas de agarre son un accesorio robótico utilizado en máquinas automáticas, aumentando la productividad del proceso. Suelen imitar los movimientos de una mano humana, siendo el punto de unión entre un robot y la pieza [13].

En el caso de los grippers o pinzas de vacío (Figura 1), incorporan un sistema compuesto por ventosas, permitiendo agarrar piezas por medio de vacío. Puede emplearse para la recogida de objetos metálicos, vidrios, plásticos o cartón. Están especialmente indicados para todo tipo de sistemas de sujeción, ya sea para ensamblaje de piezas, para Pick & Place o paletizado (Figura 1). Están compuestos por un sistema neumático conectado a una red de aire (bomba de vacío, generadores o compresores) [13] y pueden integrar o no ventosas en función del sistema y la aplicación. En este caso, sí dispone de ventosas.

Figura 1. Pinza de vacío

También se está empezando a emplear en otros sectores, siendo estos el sector del calzado para la manipulación de pieles o el sector alimentario.

Este último fue mostrado hace unos años en una revista con el subtitular “Schmalz presenta nueva garra de vacío modular que garantiza al sector alimentario la manipulación de alimentos a gran velocidad” (Fernando O. & Tomás S. J., 2020) [5]. En el sector

alimentario es necesaria una manipulación rápida, pero suave y segura. Por ello, Schmalz ha desarrollado una garra con una ventosa de vacío de silicona [5]. Es adaptable a la forma de los alimentos, requiriendo conexión de aire comprimido.

Los sistemas de vacío proporcionan soluciones mecatrónicas [13]. Esto aumenta la resistencia de las operaciones, generando menores desgastes en la manipulación de materiales.

Las pinzas de vacío mediante sistema ventosa aportan precisión y seguridad de agarre, permitiendo mejorar la calidad del proceso y productividad. Estos sistemas requieren de bombas a vacío, tubos y mangueras, conectados a las instalaciones o tomas de red de aire procedentes de generadores o compresores.

El diseño de pinzas es un área de investigación interesante y emergente. En ciertos sectores se siguen realizando funciones manualmente, ya que se encuentran dificultades tecnológicas para el agarre y manipulación de materiales pequeños y flexibles.

Las pinzas pueden tener tres tipos de accionamiento [13]:

- Accionamiento neumático. Es el sistema más utilizado. Tiene la particularidad de ser muy ligeros y de reducido tamaño, siendo importante a la hora de integrar equipos. El ahorro de energía es considerable respecto a otros sistemas, siendo más rápidos que los eléctricos. Necesitan integrar electroválvulas, mangueras y un controlador (PLC).
- Accionamiento hidráulico. Suele ser recomendado para levantar grandes pesos. Aunque es necesaria una instalación más grande, es un sistema fiable y duradero. Como todo equipo hidráulico, necesita tubos para el circuito hidráulico, electroválvulas, un PLC y bombas de presión.
- Accionamiento servoeléctrico o eléctrico. Los gripper con pinzas eléctricas, utilizan un controlador adicional que suele ser muy intuitivo a la hora de programar. Este tipo de accionamiento apenas tiene mantenimiento.

Las pinzas de vacío pueden llegar a ofrecer una gran cantidad de ventajas, entre ellas se encuentra la reducción del tiempo de producción, evitar el rechazo de piezas por errores humanos o desajustes y la posibilidad de obtener un constante estándar de calidad.

Se ha desarrollado un modelo de recogida [1] para reemplazar la tradicional tecnología ventosa, siendo más pequeño, confiable y de bajo costo. Cada punto de recogida puede ser controlado independientemente.

Esto se ha realizado con el fin de superar los inconvenientes de la tecnología tradicional de ventosa como la alta caída de presión, debido a las pequeñas secciones de flujo de válvulas y tuberías; la baja eficiencia de agarre con material poroso; y el alto peso en general. Por el contrario, el módulo de agarre presenta una sección de alto flujo que permite altas fuerzas de retención en el caso de materiales con alto nivel de porosidad.

La aparición de needle gripper o pinzas de aguja (Figura 2) permite sustituir a las pinzas de vacío, dando lugar a la manipulación de materiales difíciles para estas, componentes flexibles y no rígidos [14], como es el caso de tejidos compuestos, filtros, fibras de vidrio tejida o de carbono o materiales de espuma. Es decir, cualquier tipo de material muy poroso, pudiendo encontrar en la industria fabricantes como SCHMALZ.

Figura 2. Needle gripper de SCHMALZ

La forma de trabajar de las pinzas de aguja [18] consiste en desplazar la pinza hasta estar en contacto con el material. Después se activan las agujas, cogiendo el material y permitiendo un desplazamiento seguro y fácil. Una vez desplazado al lugar deseado, las agujas se contraerán, depositando el material en el lugar deseado con facilidad. Mientras, para el resto de los materiales se emplearán las pinzas de vacío.

Las ventosas [15] que forman parte de las pinzas de vacío, son el miembro de unión entre la pieza y la instalación de manipulación. Estas están compuestas por una ventosa y una boquilla. Son empleadas para agarrar y mover la pieza mediante un robot. La ventosa es presionada contra la pieza y viceversa mediante la presión ambiental, también conocida como presión atmosférica. Para ello, la presión ambiental tiene que ser mayor a la presión entre la ventosa y la pieza, conectando un generador de vacío a la ventosa. Este generador lo que hace es aspirar aire entre la ventosa y la pieza, conocido como evacuación del aire.

Básicamente, se distingue entre dos tipos de ventosas, las ventosas planas y las ventosas de fuelle.

Las ventosas planas [15] se destinan para la manipulación de piezas con una superficie plana o ligeramente abombada. Estas ventosas pueden sufrir una evacuación rápido debido a su forma y tamaño interior. Gracias a esto, las piezas pueden llegar a ser aspiradas en un tiempo mínimo, permitiendo una manipulación con una dinámica alta.

Pueden ser utilizadas en la manipulación de piezas lisas o de rugosidad mínima como es el caso de la chapa, cajas de cartón, lunas de cristal, piezas de plástico y planchas de madera. También pueden ser empleadas en ciclos cortos de automatización. Además, tiene una amplia geometría de ventosas, pudiendo encontrar redondas, ovaladas, de labio sellador saliente inclinado o plano.

Las ventosas de fuelle [15] son empleadas en la manipulación de piezas de diversas alturas, superficies desniveladas o delicadas. Estas ventosas presentan una gran flexibilidad y adaptación, debido al fuelle, siendo así empleadas en chapas de carrocería, tubos, componentes eléctricos o productor envasados. Presentan un efecto de elevación al aspirar, compensando además las diferentes alturas. Como se ha dicho, permite manipular piezas delicadas, empleando cuidadosamente la aspiración.

Además, se pueden encontrar diversos subtipos dentro de estas (Tabla 1)[15].

Tabla 1. Tipos de ventosas

Ventosas de fuelle (redonda)	Manipulación de piezas con desnivel y abombadas. Agarre de la pieza con rapidez y en posición exacta. Forma plana y de pequeño volumen.	

Ventosas planas (redonda)	Manipulación de piezas planas. Rápido agarre de la pieza. Exactitud en la posición.	
Ventosas manipulación de chapa	Manipulación de chapa, superficies aceitadas, platinas y chapas convexas.	
Ventosas para embalajes	Manipulación de madera y tableros. Base estructural resistente. Labio sellador para la sujeción ajustada a deformaciones de envases.	
Ventosas de fuelle (ovalada)	Destinado a piezas alargadas y delgadas. Trabaja con diversas alturas, desniveles o zonas delicadas.	
Ventosas planas (ovalada)	Manipulación de piezas alargadas y delgadas. Agarre de la pieza con rapidez y en posición exacta.	

	Forma plana y de pequeño volumen.	
Ventosas para lámina y papel	Manipulación de cartón, bolsas, láminas y blisters. Labio sellador plano con interior reforzado. Aspiración de piezas con poca estabilidad. No deja huella.	

Schamalz, Robotiq y Onrobot son algunos fabricantes que se pueden encontrar en la actualidad teniendo pinzas de vacío en su catálogo. En Onrobot se pueden encontrar pinzas de vacío como VGP20 y VGP10.

3. MATERIALES Y MÉTODOS

A la hora de trabajar y enfrentarse al problema es necesario conocer las herramientas que se disponen y como usarlas de la mejor manera posible para un fin próspero. Además, de igual importancia es conocer el método que se empleará a la hora de trabajar, debiendo de favorecer el desarrollo del objeto.

3.1. METODOLOGÍA

Es necesario tener un método a la hora de afrontar el objeto, siendo utilizando para este caso el método de prueba y error, teniendo una base en la que trabajar.

En primer lugar, se ha realizado una búsqueda por internet de información relacionada al tema a tratar, siendo en este caso la programación de gripper de vacío matricial. Esta búsqueda ha proporcionado una serie de aplicaciones, ayudando al entendimiento del proceso, tanto en las funciones que se han realizado como en los paquetes empleados para su funcionamiento.

Conforme se ha avanzado en el código, se han ido encontrando ciertos problemas, siendo necesaria la modificación de este hasta llegar al presente. El proceso se ha ido desarrollando por etapas, siendo dividido en apartados.

Primero el concepto general, consistiendo en detectar un punto en el interior de una figura. Haciendo una exhaustiva búsqueda, se ha dado con un código en Python [6] empleándose como base y en el que se desarrolla el resto del programa.

Después, se ha añadido el proceso de traslación ya que presenta una complejidad inferior al proceso de rotación. A lo largo de este proceso, se ha ido modificando la forma de llevarlo a cabo, produciéndose comprobaciones hasta llegar a la forma más cercana a lo que se ha buscado.

A la hora de la rotación, aunque parezca similar a la traslación, se ha realizado de una manera completamente diferente, debiendo de estar constantemente empleando el punto inicial para todas las rotaciones.

Se ha añadido la aparición por pantalla y la animación de la gráfica. Esta animación se ha realizado un total de cuatro veces, ya que se han de mostrar todas las matrices empleadas tanto en traslación como en rotación, y todas las matrices máximas de traslación y rotación que se han escogido mediante una función.

Además, ha sido necesaria la realización de funciones para que se pueda llevar a cabo todos los procesos realizados como es el caso del cambio de matriz, para que se produzca el desplazamiento; o la elección de las matrices con mayor número de ventosas dentro de la figura, para así saber cuáles son las matrices válidas.

Por ello, como se ha comentado, se ha tenido que ir modificando conforme la marcha.

Para este proyecto se ha trabajado con código Python, enfocándose la mayoría del trabajo; y AutoCAD, para dibujar la figura y prepararla para su lectura en el código, siendo proporcionada por el departamento.

3.2. DESARROLLO DEL SISTEMA

Los elementos empleados para la realización del código se dividen en hardware, como es el caso del gripper matricial de ventosas a vacío, debiendo de conocer sus dimensiones para poder trabajar con él en el software; el robot al que se le implementaría la pinza para ensayos futuros; e incluso una computadora; y software, donde se presentan los programas empleados para el desarrollo del código y el mismo código.

3.2.1. HARDWARE

3.2.1.1. UR5

El robot proporcionado por el departamento es el UR5 (Figura 3), perteneciente a la empresa Universal Robots. Este robot está compuesto por juntas y tubos de aluminio extruido.

El UR5 es un robot colaborativo industrial de poco peso, permitiendo realizar funciones de hasta 5 kg de carga útil [3], siendo un producto versátil, flexible y adaptable. Son ligeros, capaces de adaptarse a aplicaciones diferentes sin cambiar el modelo de producción que tiene. Su flexibilidad es debida a los 6 ejes que presenta, permitiendo colocarse en múltiples posiciones. Presenta un periodo de retroceso más corto.

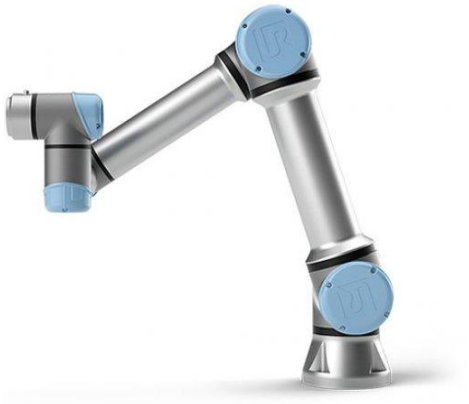


Figura 3. UR5 Universal Robot

El UR5 (Figura 3) está compuesto por 6 juntas robóticas y dos tubos de aluminio, conectando la base con la herramienta del robot, permitiendo que se mueva con un radio de trabajo de hasta 850 mm [3].

Este tipo de robots son diseñados con una intuitiva interfaz con visualización tridimensional. Puede mover el brazo del robot a las posiciones deseadas o empleando las flechas que aparecen en la tableta táctil de 12” (Figura 4). Además, puede comunicarse con diferentes máquinas mediante señales eléctricas.

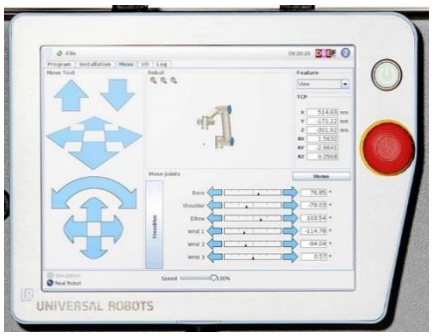


Figura 4. Tableta táctil UR5

Tiene una fácil programación y configuración, llegando a ser colaborativo y seguro, con una interfaz gráfica de usuario poliscópica (PolyScope) [17], pudiendo integrar sensores o pinzas adicionales. También se le permite instalar paquetes de software para una mejor adaptación a las funciones a desempeñar.

Las operaciones de colaboración que presenta el robot UR5 son 15 funciones avanzadas de seguridad regulable. Estos robots pueden trabajar sin vallas, con la posibilidad de tener a un usuario cerca. Cabe entender que únicamente es aplicado a funciones no peligrosas, según la evaluación de riesgo de la aplicación.

3.2.1.2. PINZA DE VACÍO

El gripper de vacío que se ha empleado para la obtención del objeto consta de 16 ventosas de fuelle redondas, ocupando poco volumen. Esto permite manipular piezas con desnivel y abombamiento.

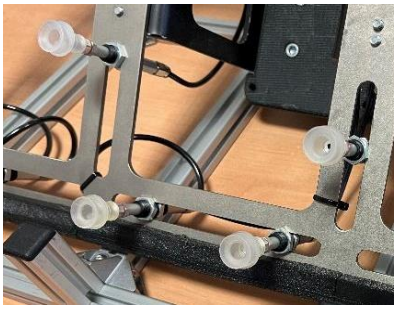


Figura 5. Ventosas del gripper

Las ventosas (Figura 5) permiten un agarre rápido, dotando al gripper de una exacta posición de trabajo a la hora de coger la figura.

Las 16 ventosas se encuentran distribuidas por una estructura de metal (Figura 6). Se ha requerido de la distancia de las ventosas a partir del centro de coordenadas de la estructura (Figura 6) para poder trabajar con la matriz, tomando como centro la pieza de unión de la estructura con el UR5.

La pieza de unión, perteneciente a Onrobot, es conocida como Quick Changer, capaz de soportar una carga útil de 20 kg [12].

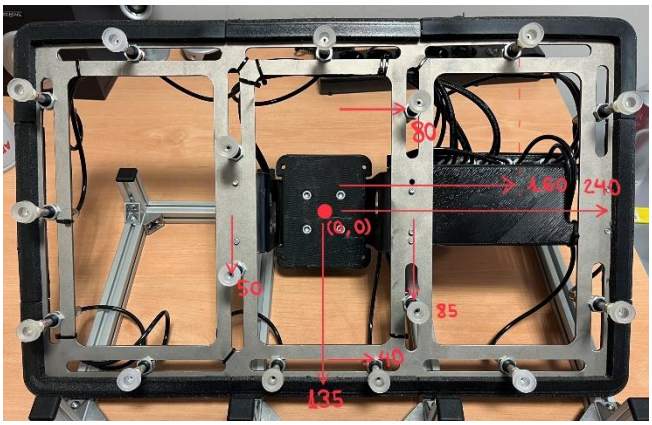


Figura 6. Gripper con el centro de coordenadas y medidas utilizadas (mm)

Los bordes de la pinza se encuentran protegidos por un plástico negro atornillado a la estructura metálica. Las ventosas tienen conectadas un tubo negro que las comunica con una electroválvula de 5 vías (Figura 7). Estas electroválvulas reducen el tamaño debido a su gran caudal, ahorrando espacio y reduciendo su peso. Se tiene presente la serie

JSY1000 [8]. A su derecha se encuentra conectado EX260, siendo un sistema de comunicación vía bus de campo.

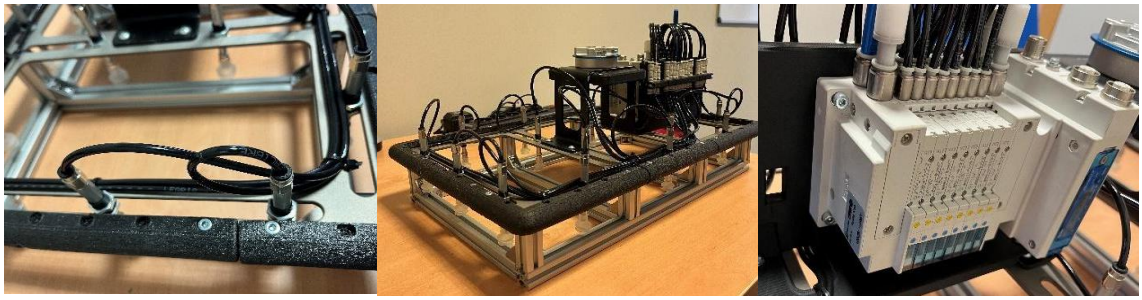


Figura 7. Gripper de vacío matricial y sus componentes

También se ha requerido de una computadora, siendo empleada para la realización del software. Su único requisito ha sido la posibilidad de poder realizar código Python sin ningún problema.

3.2.2. SOFTWARE

A la hora de realizar el código Python, se ha empleado el editor de código Visual Studio Code. En él se han ido añadiendo aquellos paquetes necesarios para alcanzar el objeto deseado, conforme se avanzaba en su desarrollo.

Los paquetes empleados para la realización del código Python han sido:

- ezdxf. Interfaz de Python para el formato DXF, desarrollado por Autodesk. Permite a los desarrolladores leer y modificar documentos DXF existentes o crear nuevos documentos DXF [7].
- math. Módulo que permite acceder a funciones matemáticas definidas por el estándar C. Las funciones que proporciona son funciones de representación y teoría de números, potencias y funciones logarítmicas, funciones trigonométricas, funciones hiperbólicas, funciones especiales y constantes [9].
- matplotlib. Librería completa para crear estáticas, animadas e interactivas visualizaciones en Python [10].
- numpy. Librería que define un tipo de dato representando matrices multidimensionales, teniendo funciones básicas para trabajar. Es una librería estable y rápida [11].

- shapely. Paquete de Python empleado para el análisis teórico de conjuntos y la manipulación de características planas utilizando funciones de la biblioteca GEOS [16].

Para conocer el programa, este ha sido dividido en tres secciones. Primero se conocerá la forma que deben de estar las imágenes para poder trabajar con ellas en el código Python, empleando la herramienta de AutoCAD. Después se mostrarán todas las funciones realizadas para el código y el Driver Code.

3.2.2.1. AUTOCAD

A la hora de emplear una figura, es necesario obtener su silueta. Para ello se ha trabajado de la mano de AutoCAD, debiendo de realizarse de una forma específica para así poder ser leída la silueta en el código Python y ser mostrada como polígono.

Es necesario que la figura esté formada por un conjunto de líneas rectas unidas entre sí en un orden específico, ya que no está permitida la utilización de circunferencias, semicircunferencias o splines.

Es de suma importancia exportar el archivo generado a DXF, ya que es el formato que se admite en el código Python para su lectura. En el caso de que no se haga así, el programa no encontrará el archivo introducido y, por ende, no podrá llevarse a cabo su lectura.

3.2.2.2. FUNCIONES

Se pueden encontrar múltiples funciones en el código realizado, incluso hay ciertas funciones que han de complementarse con otras para obtener el resultado deseado. En este apartado se muestran todas ellas, indicando su finalidad, una explicación de la función y en el caso de que se tenga que complementar, será indicado.

3.2.2.2.1. ONSEGMENT

La función “onSegment” mira si el punto “q”, descansa en la línea del segmento “pr”. Para que esto suceda, se han añadido tres puntos colineales, siendo tuplas. Estos puntos están nombrados como “p”, “q” y “r”, estando dispuestos en ese orden.

Devuelve un booleano dependiendo de si la condición se cumple o no. En el caso de que esta se cumpla, devuelve “True”. El punto “q[0]” debe ser menor o igual al máximo de “p[0]” o “r[0]”. Para ello, se ha utilizado la función “max”, proporcionada por uno de los paquetes de Python. Además, “q[0]” tiene que ser mayor o igual al mínimo de “p[0]” o

“r[0]”, mediante la función “min” proporcionada por Python. Lo mismo debe suceder en ambos casos descritos pero con “q[1]”, “p[1]” y “r[1]”.

Si alguna de las cuatro ecuaciones no se cumple, la función devuelve “False” (ver apartado 8.1. ANEXO *I*).

Función: onSegment

Variables:

p: tupla

q: tupla

r: tupla

Inicio

Si q[0] es menor o igual que máximo de p[0] o r[0] y

q[0] es mayor o igual que el mínimo de p[0] o r[0] y

q[1] es menor o igual que el máximo de p[1] o r[1] y

q[1] es mayor o igual que el mínimo de p[1] o r[1] Entonces:

Devuelve Verdadero

FinSi

Devuelve Falso

Fin

3.2.2.2.2. ORIENTATION

La función “orientation” trabaja con tres puntos distintos, como en el caso anterior, el punto “p”, el punto “q” y el punto “r”, siendo tupla cada uno y devolviendo un entero.

Esta función se ha empleado para encontrar la orientación del triplete ordenado. Para ello, un elemento, llamado “val”, es el resultado de una operación en la que la resta de “q[1]” con “p[1]” es multiplicada con la resta de “r[0]” con “q[0]”. Luego este resultado debe ser restado a la multiplicación realizada por la resta de “q[0]” con “p[0]” y la de “r[1]” con “q[1]”.

Dependiendo del valor de “val”, se devuelve un entero diferente. Si “val” es igual a 0, se devuelve el valor 0, indicando así que los puntos son colineales; en el caso de que “val”

sea mayor de 0, se devuelve 1, siendo en este caso la orientación en sentido horario; y si no, se devuelve un 2, dando a entender que el sentido de la orientación es antihorario (ver apartado 8.1. ANEXO *I*).

Función: orientation

Variables:

p: tupla

q: tupla

r: tupla

Inicio

val es igual a (((q[1] menos p[1]) multiplicado por

(r[0] menos q[0])) menos

((q[0] menos p[0]) multiplicado por

(r[1] menos q[1])))

Si val es igual a 0 Entonces:

Devuelve 0

Sino si val es mayor que 0 Entonces:

Devuelve 1

Sino:

Devuelve 2

FinSi

Fin

3.2.2.2.3. DOINTERSECT

La función “doIntersect” indica si alguno de los casos especiales o el general se produce, devuelve el booleano “True” si es así, y si no, “False”. Además, cabe destacar que en esta función tiene lugar la utilización de las dos funciones mencionadas anteriormente para así llegar a su objeto.

Se han empleado cuatro entradas conocidas como “p1”, “q1”, “p2” y “q2”. Con estas entradas, se ha obtenido cuatro orientaciones distintas mediante la función “orientation”. La primera orientación, llamada “o1”, se ha obtenido con “p1”, “q1” y “p2”; la segunda, conocida como “o2”, es resultado de “p1”, “q1” y “q2”; la tercera, “o3”, proviene de “p2”, “q2” y “p1”; y la cuarta, “o4”, se ha conseguido a partir de “p2”, “q2” y “q1”.

Una vez se tienen las diferentes orientaciones, se miran los casos para comprobar si alguno de ellos es cierto y así devolver “True”.

Se tiene cinco casos, uno general y cuatro especiales. El caso general es cierto si “o1” es distinta de “o2” y si “o3” es distinta de “o4”. Para los casos especiales se ha empleado la función “onSegmente”, diciendo así si un punto descansa en la línea del segmento.

El primer caso especial consiste en que “p1”, “q1” y “p2” sean colineales y que “p2” descanse en el segmento “p1q1”. Esto se ha realizado comprobando si “o1” es igual a 0 y empleando la función “onSegment” con “p1”, p2” y “q1”. En el segundo caso, “p1”, “q1” y “q2” deben de ser colineales, teniendo que ser “o2” igual a 0 y realizar “onSegment” para comprobar si “q2” descansa en el segmento “p1q1”. Para el tercer caso especial, “o3” debe ser igual a 0, siendo “p2”, “q2” y “p1” colineales y comprobar si “p1” descansa en el segmento “p2q2” mediante “onSegment”. Para el último caso especial, es necesario que “p2”, “q2” y “q1” sean colineales y que “q1” descanse en el segmento “p2q2”, siendo así “o4” igual a 0 y utilizando “onSegment” con “p2”, “q1” y “q2”.

Si ninguno de los casos mostrados antes es cierto, se devuelve “False” (ver apartado 8.1. ANEXO *I*).

Función: doIntersect

Variables:

p1: tupla

q1: tupla

p2: tupla

q2: tupla

Inicio

o1 es igual a la función orientation (p1,q1,p2)

o2 es igual a la función orientation (p1,q1,q2)

o3 es igual a la función orientation (p2,q2,p1)

o4 es igual a la función orientation (p2,q2,q1)

Si o1 es distinto a o2 y o3 es distinto a o4 Entonces:

Devuelve Verdadero

FinSi

Si o1 es igual a 0 y devuelve Verdadero la función onSegmente (p1,p2,q1) Entonces:

Devuelve Verdadero

FinSi

Si o2 es igual a 0 y devuelve Verdadero la función onSegmente (p1,q2,q1) Entonces:

Devuelve Verdadero

FinSi

Si o3 es igual a 0 y devuelve Verdadero la función onSegmente (p2,p1,q2) Entonces:

Devuelve Verdadero

FinSi

Si o4 es igual a 0 y devuelve Verdadero la función onSegmente (p2,q1,q2) Entonces:

Devuelve Verdadero

FinSi

Devuelve Falso

Fin

3.2.2.2.4. IS_INSIDE_POLYGON

La función “is_inside_polygon” se ha empleado para comprobar si un punto descansa dentro de un polígono, devolviendo un booleano. En esta función se han empleado todas

las funciones anteriores. Como entrada se ha empleado una lista llamada en este caso “points”, donde se encuentran los diferentes puntos que forman el polígono; y una tupla llamada “p”, donde se ha guardado el punto a verificar si se encuentra en el interior del polígono.

Es necesario saber cuántos vértices tiene el polígono, para ello se ha guardado el valor en una variable “n” utilizando una función proporcionada por Python “len”, dando la dimensión de la lista. Esta función termina si la dimensión de la lista es menor a tres, ya que no se puede formar un polígono.

Se ha creado un punto para segmentos lineales desde el punto “p” hasta infinito, guardado en “extreme”, además de “decrease”, “count” e “i”, para contar el número de puntos que forman el polígono, siendo estos tres valores iguales a 0.

Después, mediante un bucle while se comprueba si la línea de “p” a “extreme” interseca con la línea de “points[i]” a “points[next]”, siendo estos los puntos que forman el polígono, empleando la función “doIntersect”. Además, el contador aumenta una cifra su valor.

Al principio del bucle while, se ha realizado una ecuación en la que se modifica el valor “next”, sumándole a “i” uno y haciendo el módulo respecto al tamaño del polígono.

Cuando se ha obtenido el valor de “next”, se produce una sentencia if, llegando a aumentar una cifra el valor de “decrease” si la condición es cierta. Para ello “point[i][1]” debe ser igual a “p[1]”.

En el caso de que el punto ‘p’ sea colineal con la línea del segmento ‘i-next’, se mira si descansa en el segmento utilizando la función “orientation”, devolviendo si es igual a 0 la función “OnSegment”.

Una vez terminado esto, el valor de “i” se actualiza a “next”.

Cuando “i” es igual a 0, el bucle se rompe, reduciendo a “count” el valor de “decrease”, siendo verdadero si el valor que se ha devuelto es igual a 1. Para esto se hace el módulo de “count” respecto a 2 (ver apartado 8.1. ANEXO *I*).

Función: is_inside_polygon

Variables:

points: lista

p: tupla

Inicio

n es igual a tamaño de la lista points

Si n es menor que 3 Entonces:

Devuelve Falso

FinSi

extreme es igual a (INT_MAX, p[1])

decrease es igual a 0

count es igual a i que es igual a 0

Mientras sea Verdadero Hacer:

next es igual a (i +1) módulo de n

Si points[i][1] es igual a p[1] Entonces:

decrease se le suma 1

FinSi

Si la función doIntersect (points[i],points[next],p,extreme) devuelve Verdadero Entonces:

Si la función orientation (points[i],p,points[next]) es igual a 0 Entonces:

Devuelve la función onSegment (points[i],p,points[next])

FinSi

count se le suma 1

FinSi

i es igual a next

Si i es igual a 0 Entonces:

break

FinSi

Siguiente

count se le resta decrease

Devuelve Verdadero si (count módulo 2 es igual a 1) sino Falso

Fin

3.2.2.2.5. PRINT_LINE

La función “print_line” se ha empleado para obtener la figura, teniendo como entrada “e”, que es la línea de puntos que va formando al polígono; y “lista”, siendo una lista donde se guardan todos los puntos que forman el polígono.

Esta función es sencilla, pues lo que se ha hecho es añadir a la lista el valor del punto de inicio de cada línea que se envía. Se ha utilizado “append”, para ir añadiendo a la lista las coordenadas del punto en formato decimal (ver apartado 8.1. ANEXO *I*).

Función: print_line

Variables:

e: ezdxf.entities.line.Line

lista: tupla

Inicio

lista append (decimal de e.dxf.start[0], decimal de e.dxf.start[1])

Fin

3.2.2.2.6. MOVE

La función “move” se ha utilizado para modificar los puntos de la matriz que forman el gripper, realizando movimientos de traslación y rotación. Se tiene como entrada un punto de la matriz, llamado “puntos”; los valores de desplazamiento (x,y), llamados t1 y t2; y el valor de rotación, llamado t3.

En primer lugar, se han creado dos listas vacías, una denominada “lista”, en la que se guardará el punto de la matriz de la entrada; y otra que se ha llamado “resultado”, en la que se guardará la lista que devuelve la función. Se pasa a radianes el valor de rotación, ya que este se encuentra en grados. Para ello, se ha empleado la librería numpy, de donde se obtiene la función “radians”.

Mediante un bucle for, se han introducido los valores del punto de la matriz en “lista” con el método “append”. Esta lista se ha convertido en matriz, siendo traspuesta para poder trabajar con ella, guardándose en “AT”.

Después tiene lugar la definición de la matriz de traslación y rotación, multiplicándose ambas matrices y guardando el valor en “Val1”. Este valor, es multiplicado por “AT”, devolviendo “Val”, siendo este la nueva posición del punto dado.

Después, “Val” es convertido en lista y guardado en “resultado”. Es necesario añadirle el último valor del punto de la matriz que no ha sido ni modificado ni empleado en el proceso.

Una vez finalizado, se devuelve “resultado” (ver apartado 8.1. ANEXO *I*).

Función: move

Variables:

puntos: lista

t1, t2, t3: float / int

Inicio

Declaración de lista es igual a una lista vacía

Declaración de resultado es igual a una lista vacía

t3r es igual a t3 radians de numpy

Desde i hasta range(len(puntos)-1) Hacer:

lista append(puntos[i])

Siguiente

A es igual a lista array de numpy

AT es igual a A transpose de numpy

T es igual a numpy array ([[1, 0, t1],[0, 1, t2],[0, 0, 1]])

T2 es igual a numpy array ([[math.cos(t3r), -math.sin(t3r), 0],[math.sin(t3r), math.cos(t3r), 0],[0, 0,1]])

Val1 es igual a numpy dot (T1, T2)

Val es igual a numpy dot (Val1, AT)

resultado es igual a list (Val)

resultado append(puntos[3])

Devuelve resultado

Fin

3.2.2.2.7. BIGGER

Para encontrar las matrices más altas, las ventosas activadas, el total de matrices con mayor número de ventosas activadas y los valores de traslación y rotación de dichas posiciones, se ha empleado la función "bigger". Las entradas que se han empleado son "matriz", lista en la que se encuentran todas las matrices obtenidas; "matrizventosas", lista donde se encuentran las ventosas abiertas y cerradas de cada matriz; "cantidad", lista donde se encuentra el número total de ventosas activas de cada matriz; y "puntos", lista de los valores de rotación y traslación en matriz.

En primer lugar, se han creado 3 listas vacías, llamadas "posiciones" donde se guardan todas las matrices con valor máximo de ventosas activas; "ventosas" donde se guardan todas las ventosas de cada matriz con el máximo de ventosas activadas; y "matrizt" donde se guardan los valores de traslación y rotación que ofrecen la utilización del mayor número de ventosas de la matriz. También se han declarado "count", "i", "j" y "suma", igualando estos valores a 0.

Una vez se ha hecho la declaración de variables, se obtiene el valor máximo de puntos mediante la función "max" de numpy, utilizando la lista "cantidad" y guardando el valor en "maximo".

Luego se ha realizado un bucle for con la duración de cantidad, modificando el valor "z". Dentro del bucle se aumenta el valor de "count" con "num_coor" y se realiza una

sentencia if. Si “cantidad[z]” es igual al valor máximo, “j” será igual a “count” menos “num_coor” y “suma” incrementa su valor en más 1.

También se han declarado dos nuevas listas vacías, cuyo valor se vacía siempre que se produzca la igualdad. Estos valores llamados “matriztrabajop” y “matriztrabajov”, son empleados para guardar matrices del gripper y las ventosas.

Siguiendo dentro del if, se produce un while donde mientras que “j” sea menor que “count”, se añade a “matriztrabajop” la “matriz[j]” y en “matritrabajov” la “matrizventosas[j]”, incrementando el valor de “j” cada vez que vuelve a repetirse el bucle while. Cuando el bucle while ha finalizado, se añade a “posiciones” la “matriztrabajop”, “matriztrabajov” se introduce en “ventosas” y “puntos[z]” en “matrizt”.

Una vez que acabe el bucle for, se devuelve “posiciones”, “ventosas”, “suma” y “matrizt” (ver apartado 8.1. ANEXO *I*).

Función: bigger

Variables:

matriz: lista

matrizventosas: lista

cantidad: lista

puntos: lista

Inicio

Declaración de posiciones, ventosas y matrizt como listas vacías

Declaración de count, j y suma igual a 0

maximo es igual a numpy max de cantidad

Desde z hasta range(len(cantidad)) Hacer:

count es igual a count más num_coor

Si cantidad[z] es igual a maximo Entonces:

j es igual a count menos num_coor

suma es igual a suma más 1

matriztrabajop es igual a lista vacía

matriztrabajov es igual a lista vacía

Mientras j sea menor que count Hacer:

matriztrabajop append(matriz[j])

matriztrabajov append(matrizventosas[j])

j es igual a j más 1

Siguiente

FinSi

posiciones append(matriztrabajop)

ventosas append(matriztrabajov)

matrizt append(puntos[z])

Siguiente

Devuelve posiciones, ventosas, suma, matrizt

Fin

3.2.2.3. DECLARACIÓN DE VARIABLES

Han sido necesarias declaraciones de variables para las funciones y el Driver Code, teniendo declaraciones de variables fuera y dentro de este. Las declaraciones fuera del Driver Code no han sido modificadas por el programa cuando ha estado en uso, mientras que las que se encuentran dentro del Driver Code sí, ya que van cogiendo valores a lo largo del proceso.

3.2.2.3.1. DECLARACIÓN DE VARIABLES FUERA DEL DRIVER CODE

Fuera del Driver Code ha sido declarado “INT_MAX” con valor de 10000, siendo utilizado en la función “is_inside_polygon”. También ha sido declarado “num_coor”, este valor es el número de ventosas que tiene la matriz del gripper, empezando a contar desde 1. Por último, han sido declarados los escalones para el desplazamiento de la matriz de traslación y rotación, teniendo “stepx” para el eje x y “stepy” para el eje y, siendo empleados en la matriz de traslación; y “alpha”, que indica el ángulo (ver apartado 8.2. ANEXO *II*).

3.2.2.3.2. DECLARACIÓN DE VARIABLES DENTRO DEL DRIVER CODE

Dentro del Driver Code se ha declarado una larga serie de elementos (ver apartado 8.2. ANEXO *II*).

En primer lugar, se ha declarado la posición inicial de la matriz de puntos del gripper, indicando la posición de cada ventosa respecto del centro. Esto se ha guardado en una lista denominada "puntoinicial".

Después se han declarado una serie de listas vacías. La primera de ellas es "lista", donde se han guardado los puntos del polígono para así poder generarlos. Luego tenemos "puntoiniciotrabajo", donde se ha guardado el punto inicial de la traslación y el que se ha tomado como referencia para realizar el desplazamiento; "puntomodificado", es la matriz cuyo valor se modifica constantemente, siendo donde se guardan los puntos desplazados para luego poder trabajar con ellos; "guardarpuntos", se guardan todas las matrices que se han realizado hasta finalizar las traslaciones y rotaciones por separado; "puntosmaximost" y "puntosmaximosr" son bastante similares, pues se han utilizado para guardar todas las matrices con el máximo número de ventosas dentro de la figura, siendo separados en traslaciones ("puntosmaximost") y rotaciones ("puntosmaximosr"); lo mismo sucede con "ventosasmaximast" y "ventosasmaximasr", estas listas se han empleado para guardar en matriz el número de ventosas de las matrices con máximo número de ventosas dentro de la figura, volviendo a estar separado en traslación y rotación; "puntorotacion" se ha empleado para guardar las matrices con mayor número de ventosas en la traslación y así ser el inicio de la rotación, modificándose su valor constantemente; "ventosast" y "ventosasr" se han utilizado para guardar todas las ventosas activas e inactivas de cada matriz, siendo separado en traslación y rotación; "posiciones", se ha empleado para guardar los valores de "tx", "ty" y "tz" de todas las matrices; en "matriztrabajot" y "matriztrabajor" se han almacenado "tx","ty" y "tz" de las matrices con el máximo número de ventosas dentro de la figura; y "valores", donde se ha guardado el número total de ventosas activas dentro de la figura de cada matriz.

También se ha declarado una tupla, denominada "centro", indicando el centro de la figura.

Para finalizar las primeras declaraciones, se han asignado cuatro elementos igualados a 0, siendo estos "posicion", "max_sum", "val_maxt", donde se guarda el número total de matrices con valor máximo en traslación; y "val_maxr", que tiene la misma función que "val_maxt" pero en rotación.

3.2.2.4. DRIVER CODE

Después de las declaraciones, mediante “readfile” se ha leído el fichero DXF donde se encuentra la figura que se ha realizado en AutoCAD, guardándose en “doc”.

Con este nuevo elemento se ha creado un espacio modelo en “mps”, dividiendo la figura en diferentes partes. Utilizando un bucle for, se va cogiendo cada tramo generado. Si este coincide con una línea, se realiza la función “print_line” enviando “lista”, donde se guardan los puntos que forman la figura; y el valor con los datos de la línea, “e”.

Para poder presenciarlo en gráfica, se ha necesitado juntar los puntos guardados en “lista”. Para ello, se ha generado un polígono, guardándose en “pol”. Es necesario guardarlo en coordenadas “x” e “y” para poder representarlo.

Se ha obtenido el centro del polígono mediante “centroid”, pero antes de esto se ha declarado la gráfica a mostrar, indicando el tamaño de la ventana, siendo guardada para mostrar el centro de la figura, representado mediante una estrella.

Se ha guardado “puntoinicial” como lista en “puntomodificado”.

Mediante un bucle for, se ha modificado el valor de “puntomodificado”, siendo trasladado al centro del polígono con la función “move”. Este valor se ha guardado en el elemento “matrizcentro”.

Finalizando el bucle anterior, se realiza otro, este se ha empleado para mostrar la matriz “puntomodificado” en la gráfica. Luego, guarda la representación del polígono y se muestra por pantalla la gráfica con el polígono, la matriz del gripper y el centro del polígono.

Una vez mostrada la gráfica, “puntomodificado” vuelve a ser modificado, guardándose en una matriz que se ha llamado “puntoinicialtrabajo”, siendo la posición inicial de todas las traslaciones que se han realizado.

Ahora tiene lugar la primera parte de los desplazamientos, siendo la traslación.

Se han declarado dos elementos igualados a 0, siendo “tz” y “ty”. En este caso, el valor de “tz” no es modificado conforme avanza.

Se ha trabajado mediante dos bucles while. El primero de ellos corresponde a “ty”, debiendo ser su valor menor o igual 290. En este bucle while se ha declarado “tx”, siendo su valor 0 siempre que se repita. Y se pasa al otro bucle while que se encuentra dentro, teniendo lugar hasta que “tx” sea mayor que 480.

Dentro del bucle while se ha realizado en primer lugar la asignación de los valores “tx”, “ty” y “tz” a “posicion” como una matriz, mediante append.

Se ha declarado “sum” igualándolo a 0 y “puntomodificado” coge el valor de “puntoiniciotrabajo” para así poder modificarlo.

Se pasa a un bucle for, en el que el “puntomodificado” se cambia mediante la función “move”, utilizando “tx”, “ty” y “tz” y guardándose “puntomodifcado” una vez finalizado el bucle en “guardarpuntos”.

En otro bucle for se ha utilizado la función “is_inside_polygon”. En este bucle, se observa si las ventosas se encuentran dentro de la figura. Para ello, se ha guardado en “p” la posición x y la posición y de “puntomodificado”.

Mediante una sentencia if, siendo verdadero “is_inside_polygon”, se añade a “ventosast” el valor “puntomodificado[i][3]” y además “sum” se incrementa más 1 su valor. En el caso de que sea falso, se añade a “ventosast” el valor 0. Hay que tener en cuenta que este bucle se realiza tantos puntos tenga la matriz del gripper.

Fuera del bucle for, “valores” coge el valor de “sum”, siendo almacenado como lista. También se produce el incremento de “tx” mediante “stepx”, y vuelta a empezar el bucle while hasta que “tx” sea mayor al valor indicado.

Fuera de este bucle while, se ha realizado un incremento de “ty” mediante “stepy”, repitiendo así el bucle while hasta que “ty” sea mayor al valor indicado.

Cuando se ha finalizado el bucle, se declara una gráfica en la que se va a mostrar el movimiento de la matriz en todas las posiciones obtenidas, pudiendo observar cuando pasa por la figura. También se ha guardado la figura para ser mostrada y se han realizado las declaraciones de “line,”, “xdata” e “ydata”.

Después se ha guardado en “matrices_traslacion” como lista “guardarpuntos” y en “num_trasl” como entero la cantidad de “matrices_traslacion” dividida por “num_coor”.

Se pasa a un bucle for mediante “num_trasl”. En este bucle, “avance” es igual a “j”, que es la variable del bucle, multiplicada por “num_coor”, indicando que “xdata” e “ydata” son dos listas vacías.

Dentro del bucle se tiene otro bucle for, donde se asigna la posición de las ventosas en “xdata” e “ydata”. Pero antes de asignar valores, se ha utilizado el valor “k” igualándolo

con “i”, la variable del bucle, sumándole “avance”, para seleccionar la posición de las ventosas.

Cuando se ha salido de este bucle, se realiza “set_data” en “line”, almacenando los valores de “xdata” e “ydata”. Tiene lugar una pausa de 0.5 entre cada gráfica y se sale del otro bucle for, mostrando así la gráfica en movimiento.

Después se ha realizado la función “bigger”, seleccionando así las matrices con mayor número de ventosas activadas, cuáles son esas ventosas, el número de matrices seleccionadas y las posiciones de desplazamiento en la traslación con el máximo número de ventosas activadas. Estos valores se han guardado en “puntosmaximost”, “ventosasmaximast”, “val_maxt” y “matriztrabajot”.

Estos valores se han mostrado por pantalla una vez obtenidos y se ha realizado una gráfica con movimiento como la anterior, para ver cuáles son aquellas matrices seleccionadas con el máximo número de ventosas activas.

La única diferencia es que, en este caso, se ha declarado “line” y “line2”, representando esta última a las ventosas dentro de la pieza, siendo cambiado de color al mostrarse en la gráfica. Además, estas ventosas serán guardadas en “xdata2” e “ydata2”. Para ello, dentro del primer bucle se ha añadido a “ventosas” la matriz “ventosasmaximast[j]” y en el segundo bucle for, dependiendo de si “ventosas[i]” es igual a 0 o no, se ha guardado en “xdata” e “ydata” o en “xdata2” e “ydata2”.

Luego se ha realizado la segunda parte, conocida como rotación. A partir de las matrices máximas obtenidas anteriormente, se comprueba si hay mayor número de ventosas activas rotándolas.

Se han vaciado las listas “guardarpuntos”, “posiciones” y “valores”.

Mediante un bucle for del tamaño del valor de “val_maxt”, se ha igualado “puntorotacion” con “puntosmaximost[l]”. La primera y segunda posición de este valor se ha utilizado para calcular “centrox” y “centroy”, ya que, para generar la rotación, es necesario realizar la traslación y rotación desde la posición inicial de la matriz, que ha sido introducida por el usuario.

Para calcular “centrox” y “centroy”, se ha restado “puntorotacion[0][0]” menos “puntoinicial[0][0]” y por otro lado, “puntorotacion[0][1]” menos “puntoinicial[0][1]”.

Se ha iniciado un bucle while que funciona mientras “tz” sea menor que 360. Dentro de este bucle se ha igualado “sum” a 0, cogiendo la variable “puntomodificado” la lista “puntoinicial”. Además, se ha guardado en “posiciones” el valor de “centrox”, “centroy” y “tz” como matriz.

En un bucle for dentro del while, se ha realizado la función “move” con los valores “puntomodificado[k]”, “centrox”, “centroy” y “tz”, siendo “k” la variable del bucle for. El nuevo valor se ha guardado en “puntomodificado[k]”, es decir, se ha sustituido el valor que se tenía antes por el nuevo.

Cuando se ha salido del bucle for, la nueva matriz “puntomodificado” es guardada en “guardarpuntos” y “tz” aumenta su valor en “alpha”.

Se ha vuelto a añadir un bucle for dentro del while, donde se comprueba si el punto está dentro de la figura o no. Para ello, se ha guardado en “p” como tupla el valor “puntomodificado[r][0]” y “puntomodificado[r][1]”. Con una sentencia if, se guarda en “ventosasr” el valor de “puntomodificado[r][3]” y aumenta el valor de “sum”, si la función “is_inside_polygon” es verdadera. En el caso de que sea falsa, “ventosasr” coge el valor de 0.

Fuera del bucle for, se ha almacenado en “valores” el valor “sum”.

Finalizada esta parte por completo, se muestra por pantalla mediante animación todas las matrices obtenidas. La única diferencia respecto a la traslación es que “num_trasl” se ha pasado a nombrar “num_rot”, desempeñando la misma función.

Después, mediante la función “bigger” se han escogido las matrices con el máximo número de ventosas dentro de la figura. Además de devolver el número de las ventosas activas, la cantidad de matrices con mayor número de ventosas activadas y las coordenadas empleadas para llegar a esas posiciones. Estos datos se han guardado en “puntosmaximosr”, “ventosasmaximasr”, “val_maxr” y “matriztrabajor”, siendo mostrados por pantalla.

Por último, se ha realizado una gráfica como en el caso anterior, mostrando todas las matrices con el máximo número de ventosas dentro de la figura. Además, las ventosas que se encuentran dentro de la figura se han mostrado de diferente color que las que están fuera (ver apartado 8.2. ANEXO *II*).

Función: Driver Code

Variables:

puntoinicial: tupla

lista, puntoiniciotrabajo, puntomodificado, guardarpuntos, puntosmaximost, ventosasmaximast, puntosmaximosr, ventosasmaximasr, puntorotacion, ventosast, ventosasr, posiciones, matriztrabajot, matriztrabajor, valores: lista

centro: tupla

posicion, max_sum, val_maxt, val_maxr: enteros

Inicio

Declaración de puntoinicial

Declaración de lista, puntoiniciotrabajo, puntomodificado, guardarpuntos, puntosmaximost, ventosasmaximast, puntosmaximosr, ventosasmaximasr, puntorotacion, ventosast, ventosasr, posiciones, matriztrabajot, matriztrabajor y valores como listas vacías

Declaración de centro como tupla vacía

Declaración de posicion, max_sum, val_maxt y val_maxr igualado a 0

doc es igual a ezdxf.readfile(insertar nombre archivo)

mps es igual a doc.modelspace()

Desde e hasta msp Hacer:

Si e.dxftype() es igual a ‘LINE’ Entonces:

Se hace la función print_line(e, lista)

FinSi

Siguiente

Imprimir lista por pantalla

pol es igual a Polygon de lista

x e y son iguales a pol.exterior.xy

fig y ax es igual a plt.subplots()

ax.set_xlim (valor inicial gráfica eje x, valor final gráfica eje x)

ax.set_ylim (valor inicial gráfica eje y, valor final gráfica eje y)

centro es igual a (pol.centroid.x, pol.centroid.y)

Imprimir centro por pantalla

Se hace un plt.scatter de la tupla centro

puntomodificado es igualado a puntoinicial como una lista

Desde k hasta range(len(puntomodificado)) Hacer:

    puntomodificado[k] es igual a la función move con puntomodificado[k], centro[0], centro[1] y 0

Siguiente

matrizcentro es igual a puntomodificado

Desde i hasta range(len(puntomodificado)) Hacer:

    p es igual a (puntomodificado[i][0], puntomodificado[i][1])

    Se hace plt.scatter de p

Siguiente

Se hace plt.plot de x e y

plt.show()

Desde k hasta range(len(puntomodificado)) Hacer:

    puntoiniciotrabajo.append de la función move con puntomodificado[k] y las posiciones de inicio de la matriz para trabajar

Siguiente

Se imprime por pantalla “Primera parte”

tz y ty son igual a 0

Mientras ty es menor o igual que 290 Hacer:

    tx es igual a 0

    Mientras tx sea menor o igual que 480 Hacer:

        posiciones.appen de [tx,ty,tz]

sum es igual a 0

puntomodificado es igual a puntoiniciotrabajo como lista

Desde k hasta range (len(puntomodificado)) Hacer:

puntomodificado[k] es igual a la función move con puntomodificado[k], tx, ty y tz

Siguiente

guardarpuntos.extend de puntomodificado

Desde i hasta range(len(puntomodificado)) Hacer:

p es igual a (puntomodificado[i][0], puntomodificado[i][1])

Si la función is_inside_polygon es verdadera con lista y p Entonces:

ventosast. append de puntomodificado[i][3]

sum es igual a sum más 1

Sino:

ventosast.append de 0

FinSi

Siguiente

valores.append de sum

tx es igual a tx más stepx

Siguiente

ty es igual a ty más stepy

Siguiente

fig y ax es igual a plt.subplots()

ax.set_xlim (valor inicial gráfica eje x, valor final gráfica eje x)

ax.set_ylim (valor inicial gráfica eje y, valor final gráfica eje y)

Se hace plt.plot de x e y

line es igual a ax.plot([],[],color rojo y +)

xdata e ydata son iguales a lista vacía cada uno

matrices_traslacion es igual a guardarpuntos como lista

num_trasl es igual a len(matrices_traslacion) dividido num_coor como entero

Desde j hasta range(num_trasl) Hacer:

    avance es igual a j por num_coor

    xdata e ydata son iguales a lista vacía cada uno

    Desde i hasta range(num_coor) Hacer:

        k es igual a i más avance

        new_xdata es igual a matrices_traslacion[k][0]

        new_ydata es igual a matrices_traslacion[k][1]

        xdata.append de new_xdata

        ydata.append de new_ydata

    Siguiente

    line.set_data de xdata e ydata

    Pausa de 0.5

Siguiente

plt.show()

puntosmaximost, ventosasmaximast, val_maxt y matriztrabajot es igual a la función bigger con guardarpuntos, ventosast, valores y posiciones

Mostrar por pantalla los valores obtenidos

fig y ax es igual a plt.subplots()

ax.set_xlim (valor inicial gráfica eje x, valor final gráfica eje x)

ax.set_ylim (valor inicial gráfica eje y, valor final gráfica eje y)

Se hace plt.plot de x e y

line es igual a ax.plot([],[],color rojo y +)

```
line2 es igual a ax.plot([],[], color verde y +)
xdata e ydata son iguales a lista vacía cada uno
xdata2 e ydata2 son iguales a lista vacía cada uno
matrices_traslacion es igual a puntosmaximost como lista
num_trasl es igual a len(matrices_traslacion) como entero
Desde j hasta range(num_trasl) Hacer:
    matrizavance es igual a matrices_traslacion[j]
    ventosas es igual a ventosasmaximast[j]
    xdata e ydata son iguales a lista vacía cada uno
    xdata2 e ydata2 son iguales a lista vacía cada uno
    Desde i hasta range(num_coor) Hacer:
        new_xdata es igual a matrizavance[i][0]
        new_ydata es igual a matrizavance[i][1]
        Si ventosas[i] es igual a 0 Entonces:
            xdata.append de new_xdata
            ydata.append de new_ydata
        Sino:
            xdata2.append de new_xdata
            ydata2.append de new_ydata
        FinSi
    Siguiente
    line.set_data de xdata e ydata
    line2.set_data de xdata2 e ydata2
    Pausa de 0.5
Siguiente
plt.show()
```

Se imprime por pantalla “Segunda parte”

guardarpuntos es igual a una lista vacía

posicion es igual a una lista vacía

valores es igual a una lista vacía

puntorotacion es igual a una lista vacía

Desde l hasta range(val_maxt) Hacer:

tz es igual a 0

puntorotacion es igual a puntosmaximost[l]

centrox es igual a puntorotacion[0][0] menos puntoinicial[0][0]

centroy es igual a puntorotacion[0][1] mens puntoinicial[0][1]

Mientras tz sea menor que 360 Hacer:

sum es igual a 0

puntomodificado es igual a puntoinicial como lista

posiciones.append de centrox, centroy y tz

Desde k hasta range(len(puntomodificado)) Hacer:

puntomodificado[k] es igual a la funcion move con puntomodificado[k], centrox, centroy y tz

Siguiente

guardarpuntos.extend de puntomodificado

tz es igual a tz más alpha

Desde r hasta range(len(puntomodificado)) Hacer:

p es igual a (puntomodificado[r][0], puntomodificado[r][1])

Si la función is_inside_polygon con lista y p es verdadera Entonces:

ventosasr.append de puntomodificado[r][3]

sum es igual a sum más 1

Sino:

ventosasr.append de 0

FinSi

Siguiente

valores.append de sum

Siguiente

Siguiente

fig y ax es igual a plt.subplots()

ax.set_xlim (valor inicial gráfica eje x, valor final gráfica eje x)

ax.set_ylim (valor inicial gráfica eje y, valor final gráfica eje y)

Se hace plt.plot de x e y

line es igual a ax.plot([],[],color rojo y +)

xdata e ydata son iguales a lista vacía cada uno

matrices_rotacion es igual a guardarpuntos como lista

num_rot es igual a len(matrices_rotacion) dividido num_coor como entero

Desde j hasta range(num_rot) Hacer:

avance es igual a j por num_coor

xdata e ydata son iguales a lista vacía cada uno

Desde i hasta range(num_coor) Hacer:

k es igual a i más avance

new_xdata es igual a matrices_rotacion[k][0]

new_ydata es igual a matrices_rotacion[k][1]

xdata.append de new_xdata

ydata.append de new_ydata

Siguiente

line.set_data de xdata e ydata

Pausa de 0.5

Siguiente

plt.show()

puntosmaximosr, ventosasmaximasr, val_maxr y matriztrabajor es igual a la función bigger con guardarpuntos, ventosasr, valores y posiciones

Mostrar por pantalla los valores obtenidos

fig y ax es igual a plt.subplots()

ax.set_xlim (valor inicial gráfica eje x, valor final gráfica eje x)

ax.set_ylim (valor inicial gráfica eje y, valor final gráfica eje y)

Se hace plt.plot de x e y

line es igual a ax.plot([],[],color rojo y +)

line2 es igual a ax.plot([],[], color verde y +)

xdata e ydata son iguales a lista vacía cada uno

xdata2 e ydata2 son iguales a lista vacía cada uno

matrices_rotacion es igual a puntosmaximosr como lista

num_rot es igual a len(matrices_rotacion) como entero

Desde j hasta range(num_rot) Hacer:

matrizavance es igual a matrices_rotacion[j]

ventosas es igual a ventosasmaximasr[j]

xdata e ydata son iguales a lista vacía cada uno

xdata2 e ydata2 son iguales a lista vacía cada uno

Desde i hasta range(num_coor) Hacer:

new_xdata es igual a matrizavance[i][0]

new_ydata es igual a matrizavance[i][1]

Si ventosas[i] es igual a 0 Entonces:

xdata.append de new_xdata

ydata.append de new_ydata

Sino:

xdata2.append de new_xdata

ydata2.append de new_ydata

FinSi

Siguiente

line.set_data de xdata e ydata

line2.set_data de xdata2 e ydata2

Pausa de 0.5

Siguiente

plt.show()

Fin

4. ANÁLISIS Y DISCUSIÓN

Mediante diferentes figuras se ha ido comprobando el código realizado, pudiéndose observar el proceso y resultado del programa creado. Para ello se ha empleado una serie de explicaciones acompañando a las gráficas que se han ido generando conforme avanza el proceso y los datos que se han obtenido por pantalla.

Las piezas empleadas para la comprobación del funcionamiento del código son corte1A, corte2B, corte3C, corte4D y corte5E. Cada una tiene una forma distinta, obteniendo resultados diferentes en cada caso.

Las variables utilizadas han sido las mismas para todas las figuras, siendo estas "INT_MAX" igual a 10000; "num_coor" igual a 16; "stepx" y "stepy" igual a 40; y "alpha" igual a 20.

4.1. FIGURA CORTE1A

En primer lugar, se ha leído el archivo DXF realizado en AutoCAD, el cual se ha llamado "corte1A.dxf". En este archivo se encuentra la figura con la que se ha trabajado para obtener las matrices con máximo número de ventosas activadas dentro de la figura.

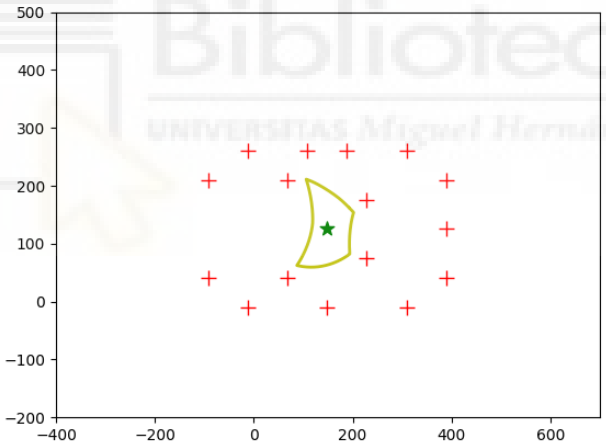
Cuando se ha guardado en un polígono la figura, son mostrados por pantalla los puntos que lo forman. Este polígono es representado en la gráfica mediante líneas amarillas, uniendo los puntos.

[(192.8149978636741, 162.0819837613552), (193.2634282171799, 161.7030888604331), (193.7108532041385, 161.3229763412265), …, (192.7357069286497, 162.1479365206543), (192.7759267562248, 162.1144084259808), (192.8149978636763, 162.0819837613541)]

Al obtenerse el polígono de la figura “corte1A.dxf”, se ha calculado su centro mediante “centroid”.

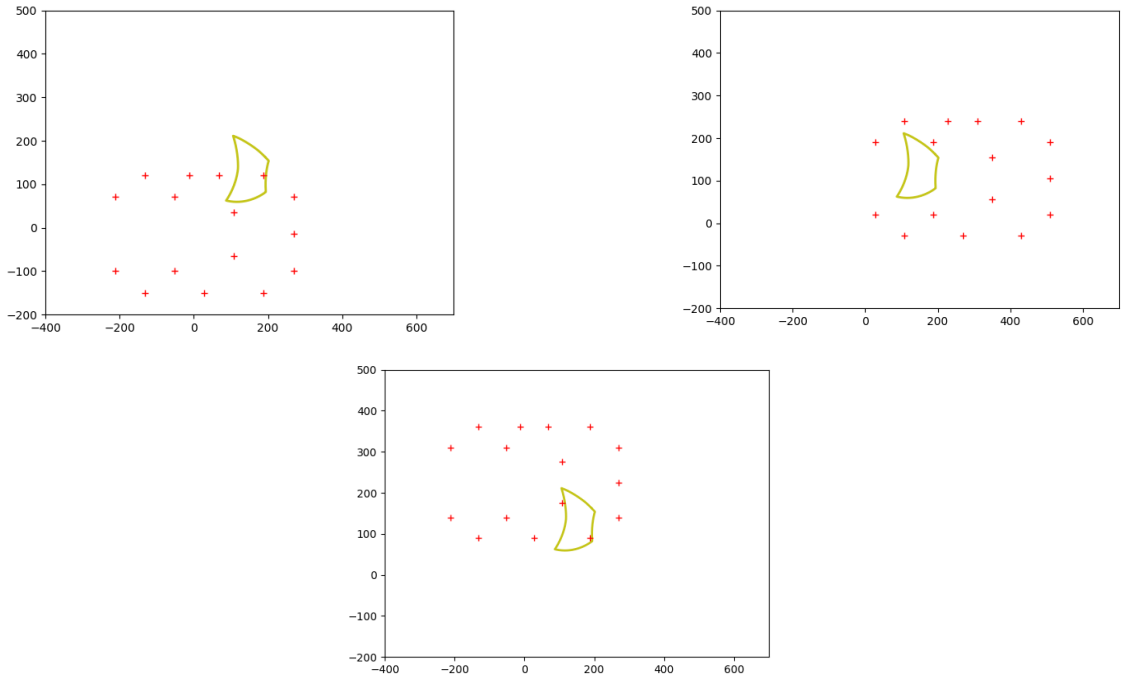
(148.68181480369222, 125.3055034546722)

El centro de la figura se ha mostrado en la gráfica mediante una estrella. Se ha desplazado la posición de la matriz de ventosas al centro de la figura, siendo estas ventosas representadas mediante el signo de sumar (Gráfica 1).



Gráfica 1. Figura corte1A.dxf con la matriz de ventosas en su centro

Después se ha producido la primera parte del desplazamiento, conocida como traslación. En esta parte, se ha modificado la matriz de ventosas, desplazándose por ciertas coordenadas para conocer cuales ventosas de cada matriz han pasado por dentro de la figura. Esto se ha mostrado en una gráfica animada, pudiendo captar el movimiento de la matriz (Gráfica 2).



Gráfica 2. Traslación de la matriz en la figura corte1A.dxf

Una vez se han obtenido todas las matrices posibles, se han buscado las matrices con mayor número de ventosas activas dentro de la figura, las ventosas activadas en cada matriz, la cantidad de matrices con mayor número de ventosas activas y las coordenadas empleadas para desplazar la matriz. Para ello, se ha empleado la función “bigger”, mostrándose por pantalla los resultados.

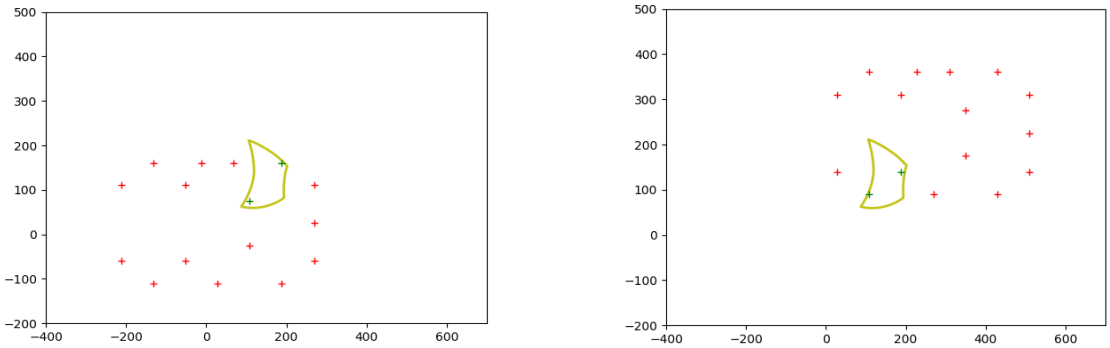
[[[428.6818148036922, 70.30550345467219, 1.0, 1], [428.6818148036922, -14.694496545327794, 1.0, 2], [428.6818148036922, -99.6944965453278, 1.0, 3], [348.6818148036922, 120.30550345467219, 1.0, 4], …, [188.68181480369222, 140.3055034546722, 1.0, 12], [108.68181480369222, 360.3055034546722, 1.0, 13], [108.68181480369222, 90.30550345467219, 1.0, 14], [28.68181480369219, 310.3055034546722, 1.0, 15], [28.68181480369219, 140.3055034546722, 1.0, 16]]]

[[0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 11, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 11, 0, 0, 0, 0, 0], …, [0, 0, 3, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0, 14, 0, 0]]

10

[[280, 0, 0], [320, 0, 0], [440, 0, 0], …, [0, 120, 0], [0, 200, 0], [40, 240, 0], [360, 240, 0]]

Las matrices con mayor número de ventosas activas se han mostrado en una gráfica animada. Las ventosas activadas se diferencian de las no activadas por el color (Gráfica 3).



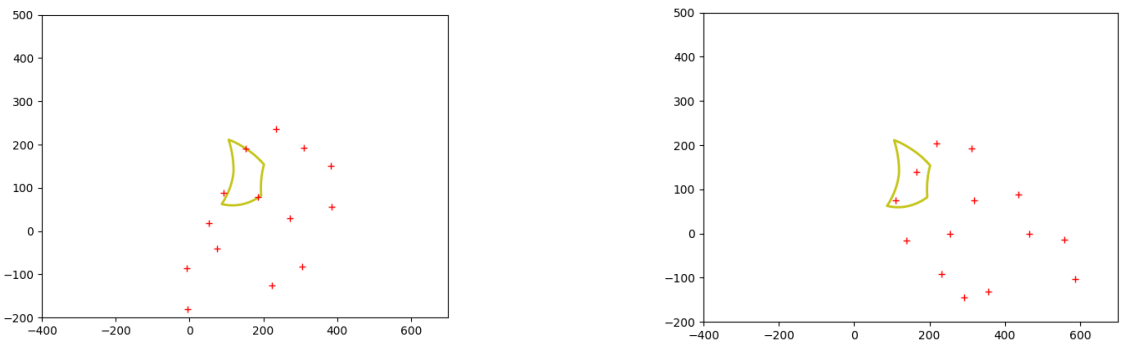
Gráfica 3. Matrices de traslación con mayor número de ventosas activas en la figura corte1A.dxf

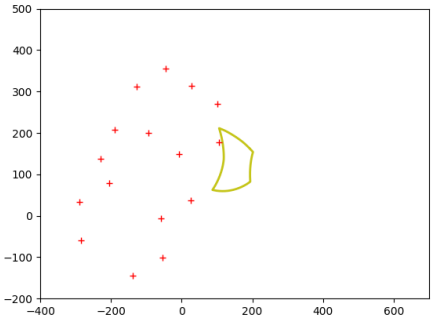
El resultado obtenido muestra que únicamente son guardadas 10 matrices de todas las que se han generado. Esto, como se ha comentado, es debido a que no todas cumplen el máximo de ventosas activas posibles, que en este caso ha sido 2. Por ello, mediante la función “bigger” se han escogido aquellas matrices que cumplen este requisito.

Que hayan sido seleccionadas 10 matrices, no quiere decir que al realizar las rotaciones el valor vaya a quedarse igual o disminuir. Dependiendo de la pieza, pueden llegar a activarse más ventosas mediante rotación.

Al finalizar el proceso de traslación, se pasa al de rotación. Para ello, se ha trabajado con las matrices obtenidas de la traslación, con el mayor número de ventosas activas dentro de la figura.

Estas matrices se han rotado respecto a su eje, obteniendo así nuevas matrices (Gráfica 4). Las rotaciones han sido mostradas en una gráfica animada, pudiendo así apreciar su movimiento.





Gráfica 4. Rotación de la matriz en la figura corte1A.dxf

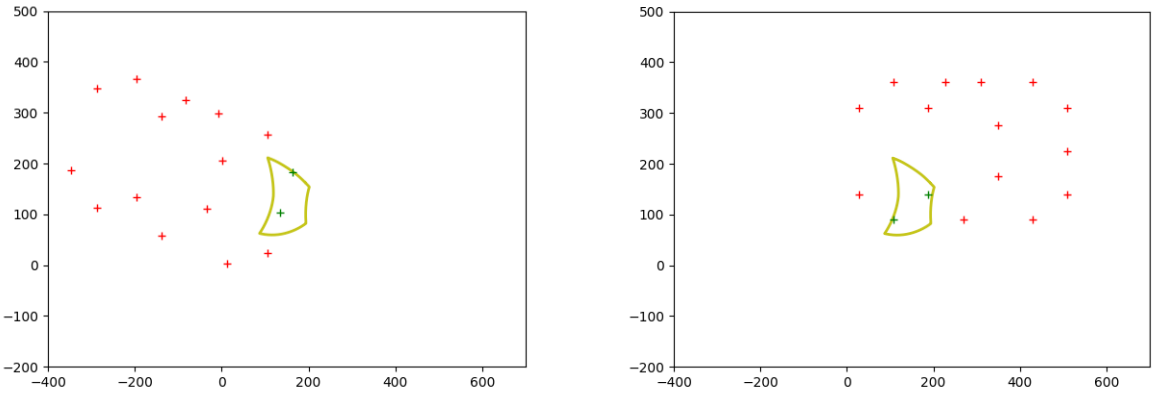
Una vez se han obtenido todas las matrices de rotación, se ha realizado la función “bigger” para obtener las matrices con mayor número de ventosas activas dentro de la figura, las ventosas activas de cada matriz, el número de matrices seleccionadas y las coordenadas para llegar a las matrices. Esto se ha mostrado por pantalla y las matrices se han representado en una gráfica animada (Gráfica 5).

[[[428.6818148036922, 70.30550345467219, 1.0, 1], [428.6818148036922, -14.694496545327809, 1.0, 2], [428.6818148036922, -99.69449654532781, 1.0, 3], [348.6818148036922, 120.30550345467219, 1.0, 4], …, [382.29397412536946, 198.52347115191708, 1.0, 12], [231.76838529279294, 19.241438849162023, 1.0, 13], [465.59524431459135, 154.24143884916197, 1.0, 14], [315.0696554820148, -25.04059345359309, 1.0, 15], [462.29397412536946, 59.95940654640688, 1.0, 16]]]

[[0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 11, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 11, 0, 0, 0, 0, 0], …, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 0, 13, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 8, 0, 10, 0, 0, 0, 0, 0, 0]]

32

[[188.6818148036922, -14.694496545327809, 0], [228.6818148036922, -14.694496545327809, 0], …, [268.6818148036922, 225.3055034546722, 0], [268.6818148036922, 225.3055034546722, 80], [268.6818148036922, 225.3055034546722, 120]]



Gráfica 5. Matrices de rotación con mayor número de ventosas activas en la figura corte1A.dxf

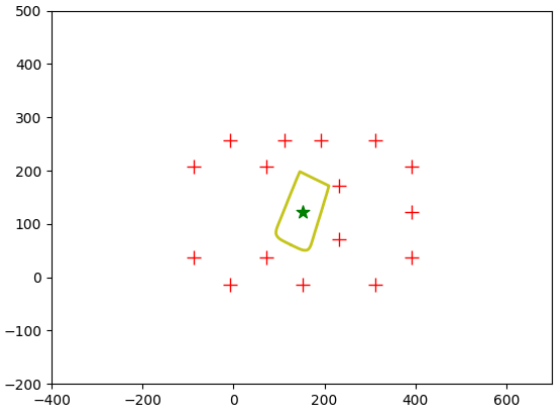
En este caso, el número de ventosas activadas máximo es el mismo en rotación y traslación, siendo este 2. Al realizar la rotación, se han generado nuevas posiciones para la matriz. Esto ha llevado a que el número de matrices que se han guardado en la función "bigger" sea 32, teniendo así más matrices con el número máximo de ventosas activas al realizar la rotación. Por ello, a la hora de coger la pieza, se puede escoger la posición que más guste entre todas las soluciones obtenidas.

4.2. FIGURA CORTE2B

Mediante la figura que se ha proporcionado con AutoCAD, llamada "corte2B.dxf", se ha creado un polígono. Este polígono se ha representado mediante líneas amarillas en una gráfica, junto al centro de la figura, siendo mostrado como una estrella. La matriz de ventosas se ha transportado desde el centro de su eje hasta el centro de la figura, representándose con el signo de sumar (Gráfica 6).

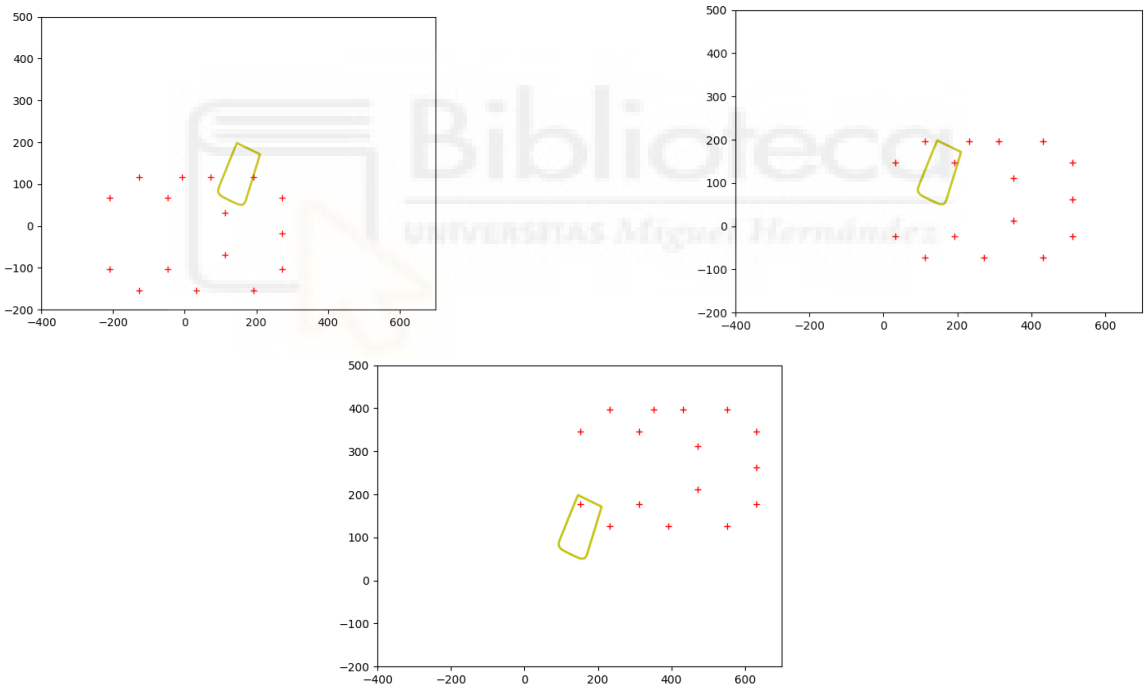
[(126.3991578063517, 60.68744770219621), (134.2957183566238, 57.51309241712078), (138.9442733147332, 55.61758805114422), …, (111.1622585619971, 67.22672911664723), (113.9448985003703, 66.0011211151353), (118.5665073356289, 64.04101097183735)]

(151.76151132698487, 121.76435528019796)



Gráfica 6. Figura corte2B.dxf con matriz de ventosas en su centro

Luego, la matriz de ventosas se ha desplazado a una posición inicial de trabajo, generando traslaciones a lo largo y ancho de unas coordenadas determinadas. Esto se ha realizado para obtener un amplio número de matrices y seleccionar después aquellas que interesan (Gráfica 7).



Gráfica 7. Traslación de la matriz en la figura corte2B.dxf

Una vez se han obtenido todas las posiciones posibles, se han buscado aquellas matrices con un mayor número de ventosas activas dentro de la figura. También se ha obtenido cuáles son esas ventosas activas, el número de matrices seleccionadas y el desplazamiento que ha tenido lugar para llegar a esos puntos. Siendo mostrado por pantalla.

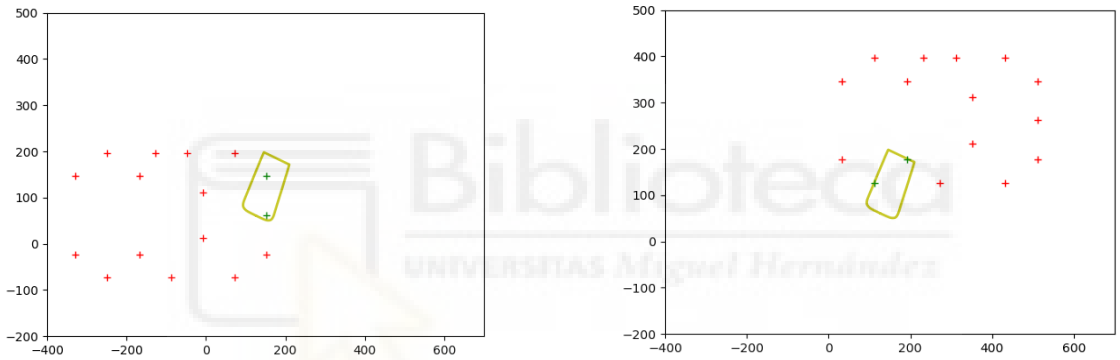
[[[271.7615113269849, 106.76435528019795, 1.0, 1], [271.7615113269849, 21.76435528019796, 1.0, 2], [271.7615113269849, -63.23564471980204, 1.0, 3], [191.76151132698487, 156.76435528019795, 1.0, 4], ..., [111.76151132698487, 126.76435528019795, 1.0, 14], [31.761511326984873, 346.76435528019795, 1.0, 15], [31.761511326984873, 176.76435528019795, 1.0, 16]]]

[[0, 0, 0, 4, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 11, 0, 0, 0, 0, 0], …, [0, 0, 3, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0, 14, 0, 0]]

15

[[120, 40, 0], [280, 40, 0], [320, 40, 0], …, [360, 240, 0], [40, 280, 0], [360, 280, 0]]

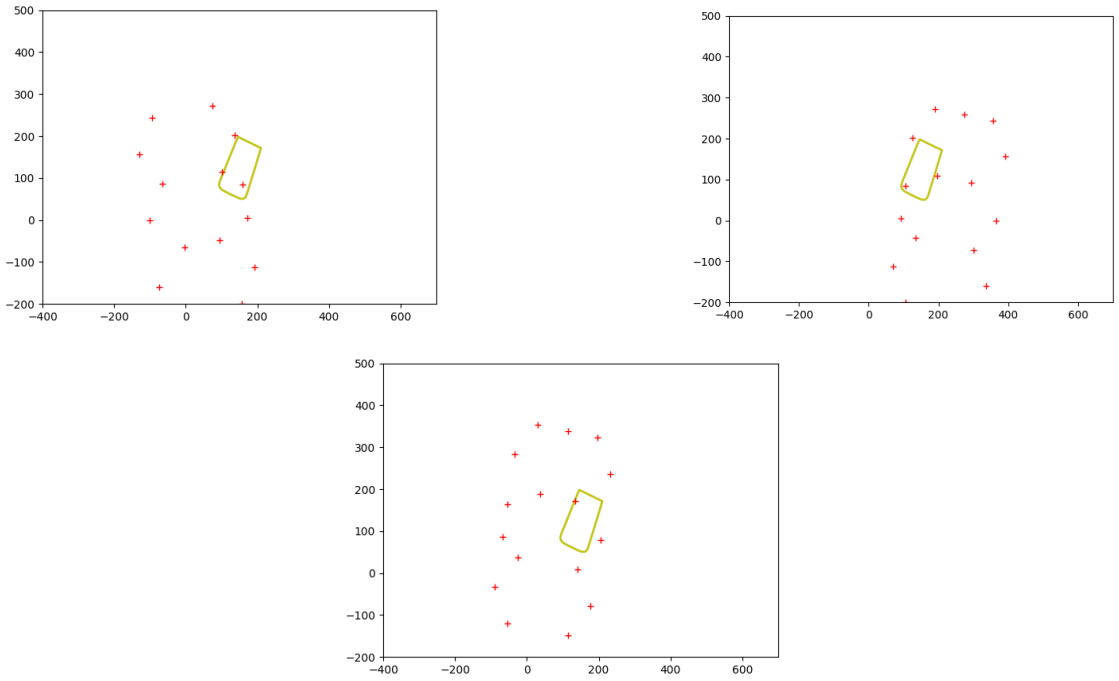
Las matrices seleccionadas son mostradas mediante una gráfica animada, distinguiéndose entre ventosas dentro y fuera de la figura (Gráfica 8).



Gráfica 8. Matrices de traslación con mayor número de ventosas activas en la figura corte2B.dxf

En este caso como en el anterior, al realizar la traslación se ha conseguido un máximo de 2 ventosas activas. Entre todas las matrices generadas, únicamente 15 han sido seleccionadas en la función “bigger”, las cuales son empleadas para la rotación y así comprobar si puede haber más matrices a la hora de realizar la sujeción de la figura o más ventosas activas.

Después de realizarse la traslación, se pasa a la rotación. Las matrices con mayor número de ventosas activas seleccionadas anteriormente sufren una serie de rotaciones, buscando así una mayor ocupación de las ventosas dentro de la figura. Cuando se han obtenido todas las matrices, estas son mostradas mediante una gráfica animada (Gráfica 9).



Gráfica 9. Rotación de la matriz en la figura corte2B.dxf

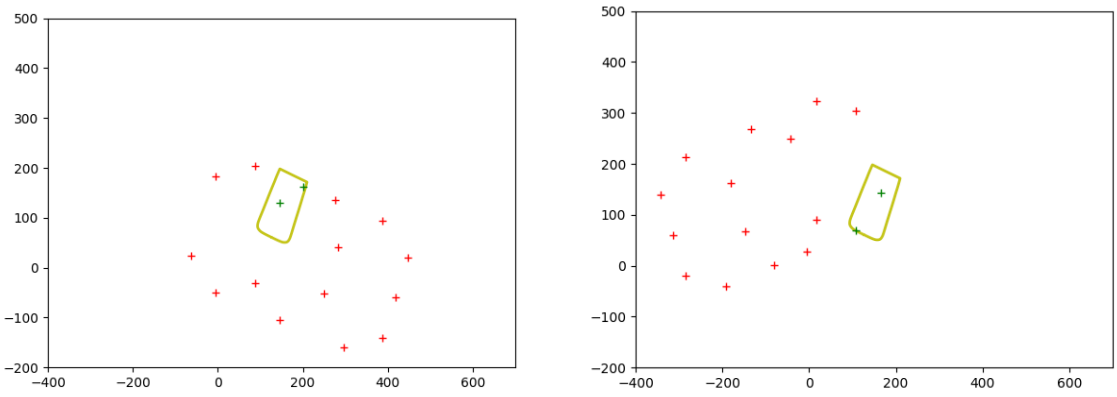
Luego se han buscado aquellas matrices con mayor cantidad de ventosas activas, siendo mostradas por pantalla y gráficamente (Gráfica 10). También se ha obtenido las ventosas activas de esas matrices, la cantidad de matrices escogidas y las posiciones empleadas para llegar a ellas.

[[[271.7615113269849, 106.76435528019795, 1.0, 1], [271.7615113269849, 21.764355280197947, 1.0, 2], [271.7615113269849, -63.23564471980205, 1.0, 3], …, [432.49426641034177, 127.63761878328026, 1.0, 14], [229.72841496101046, 10.650399455578935, 1.0, 15], [397.14573297308584, 40.17058965895711, 1.0, 16]]]

[[0, 0, 0, 4, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0, 14, 0, 0], …, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0, 14, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 0, 13, 0, 0, 0]]

55

[[31.761511326984873, 21.764355280197947, 0], [31.761511326984873, 21.764355280197947, 180], [31.761511326984873, 21.764355280197947, 260], …, [-48.23848867301513, 261.76435528019795, 340], [271.7615113269849, 261.76435528019795, 0], [271.7615113269849, 261.76435528019795, 100]]



Gráfica 10. Matrices de rotación con mayor número de ventosas activas en la figura corte2B.dxf

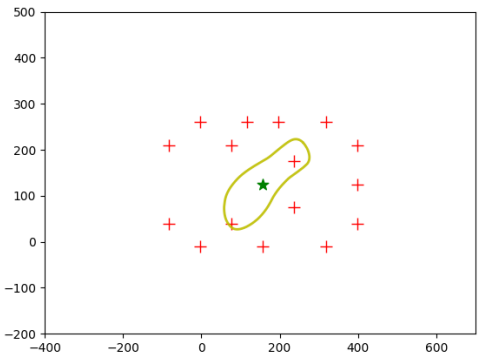
En la rotación ha sucedido lo mismo que en el caso anterior. Se ha seguido manteniendo el valor de ventosas activas máximo mostrado al realizar la traslación. Eso sí, el número de matrices ha aumentado a 55, pudiendo agarrar la figura cualquiera de las matrices guardadas.

4.3. FIGURA CORTE3C

En AutoCAD se ha obtenido una figura, llamada "corte3C.dxf". Con esta figura se ha generado un polígono, guardando los puntos de la figura en Python y siendo representado en la gráfica mediante líneas amarillas. También se ha obtenido el centro de la figura, representado con una estrella. Las ventosas de la matriz se han representado mediante el signo de sumar, desplazando la matriz de ventosas al centro de la figura (Gráfica 11).

[(259.832027449134, 215.5595131770094), (259.6595345332068, 215.7133372945402), (259.4834295367533, 215.8681399572888), …, (260.8790582187298, 214.5134749368704), (260.3562247484187, 215.0370731632917), (259.832027449134, 215.5595131770094)]

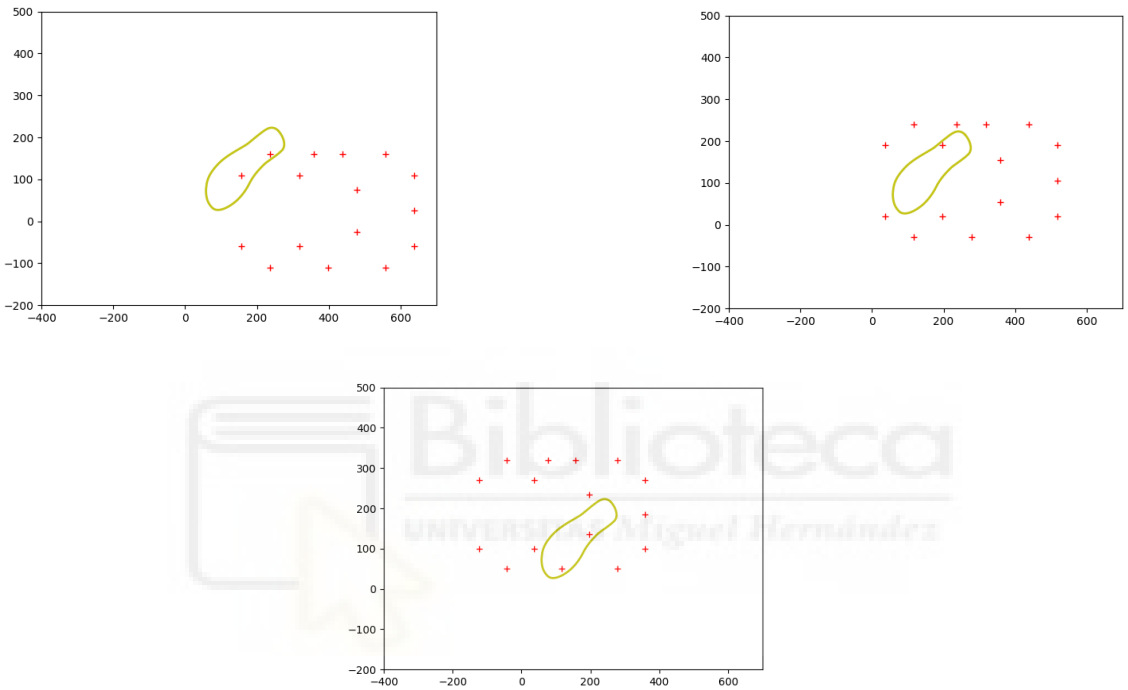
(157.5623341019758, 124.92772295528019)



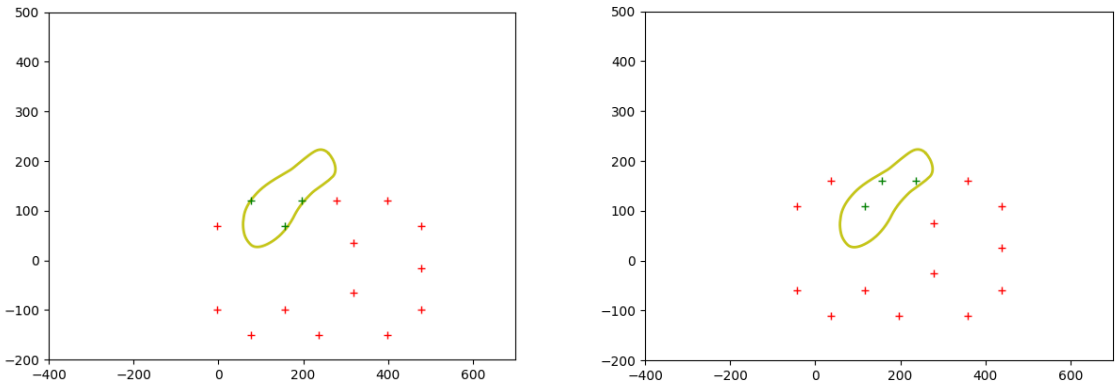
Gráfica 11. Figura corte3C.dxf con matriz de ventosas en su centro

La matriz se ha desplazado a una posición inicial de trabajo, produciéndose después una serie de traslaciones a lo largo y ancho de la gráfica, obteniendo diversas posiciones de la matriz (Gráfica 12). La traslación se ha mostrado en una gráfica animada, en la que se ha podido ver el desplazamiento completo.

Luego, se han seleccionado aquellas matrices con el mayor número de ventosas dentro de la figura, diferenciándose las ventosas que están dentro de la figura respecto de las que están fuera (Gráfica 13).



Gráfica 12. Traslación de la matriz en la figura corte3C.dxf



Gráfica 13. Matrices de traslación con mayor número de ventosas activas en la figura corte3C.dxf

Además, se ha mostrado por pantalla los valores de las matrices con mayor número de ventosas activas, las ventosas que se han activado, la cantidad de matrices seleccionadas y los valores de desplazamiento que se han empleado para llegar a cada una de las matrices seleccionada.

[[[277.5623341019758, 69.92772295528019, 1.0, 1], [277.5623341019758, -15.072277044719812, 1.0, 2], [277.5623341019758, -100.07227704471981, 1.0, 3], [197.5623341019758, 119.92772295528016, 1.0, 4], …, [37.56233410197581, -110.07227704471981, 1.0, 14], [-42.43766589802419, 109.92772295528019, 1.0, 15], [-42.43766589802419, -60.07227704471981, 1.0, 16]]]

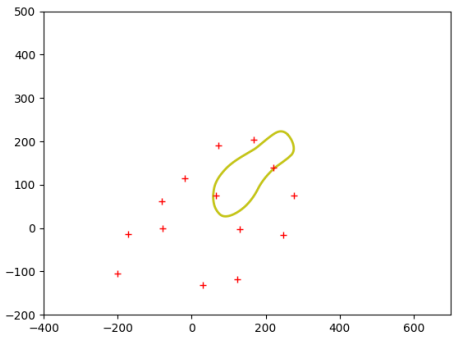
[[0, 0, 0, 4, 0, 6, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 8, 0, 10, 11, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 11, 0, 13, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 8, 0, 10, 11, 0, 0, 0, 0, 0]]

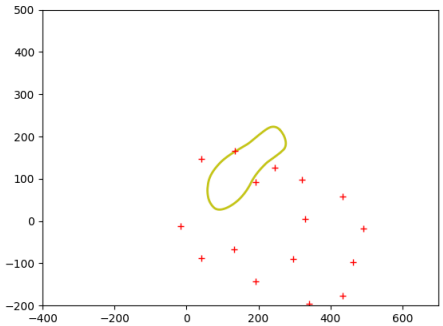
4

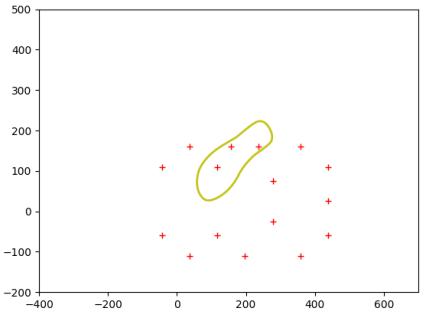
[[120, 0, 0], [240, 0, 0], [320, 0, 0], [280, 40, 0]]

En la traslación se ha obtenido un total de 4 matrices con el mayor número de ventosas activas, las cuales se utilizan en el proceso de rotación. Partiendo de estas matrices, se comprueba si se pueden obtener más soluciones para coger el objeto. El número máximo de ventosas activas en la traslación es de 3, por lo que se van a desechar aquellas inferiores a este valor en la rotación.

Una vez se han obtenidos las matrices con mayor número de ventosas dentro de la figura mediante la traslación, se pasa a la rotación. Para ello, utilizando estas matrices, se ha ido modificando su valor para realizar una rotación de estas. Estas rotaciones se han mostrado en una gráfica animada (Gráfica 14).







Gráfica 14. Rotación de la matriz en la figura corte3C.dxf

Cuando se han obtenidos las rotaciones, se han buscado todas las matrices con el máximo número de ventosas dentro de la figura. Además, se ha obtenido la ubicación de las ventosas activas, el número total de matrices seleccionadas y las coordenadas que se han empleado para desplazar las matrices.

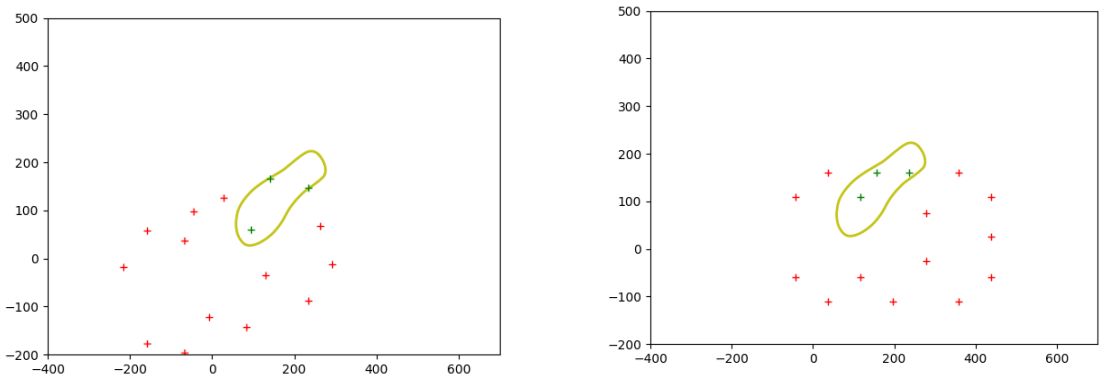
[[[277.5623341019758, 69.92772295528019, 1.0, 1], [277.5623341019758, -15.072277044719812, 1.0, 2], [277.5623341019758, -100.07227704471981, 1.0, 3], …, [234.475763612875, -181.13634165023, 1.0, 14], 3.9501747802985108, -140.41837395298506, 1.0, 15], [151.17449342365308, -225.4183739529851, 1.0, 16]]]

[[0, 0, 0, 4, 0, 6, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0], [1, 0, 0, 4, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0, 14, 0, 16], …, [0, 0, 0, 0, 0, 0, 0, 8, 0, 10, 11, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 6, 0, 8, 0, 10, 0, 0, 0, 0, 0, 0]]

12

[[37.56233410197581, -15.072277044719812, 0], [37.56233410197581, -15.072277044719812, 20], [37.56233410197581, -15.072277044719812, 200], [157.5623341019758, -15.072277044719812, 0], …, [197.5623341019758, 24.92772295528019, 20], [197.5623341019758, 24.92772295528019, 40], [197.5623341019758, 24.92772295528019, 60]]

Las matrices seleccionadas se han mostrado en una gráfica animada, diferenciándose las ventosas activas de las inactivas (Gráfica 15).



Gráfica 15. Matrices de rotación con mayor número de ventosas activas en la figura corte3C.dxf

Al realizar la rotación y escoger aquellas matrices con el máximo número de ventosas dentro de la figura, se puede comprobar que la cantidad de matrices ha aumentado, siendo esta 12.

Por otro lado, la cantidad de ventosas activas se ha mantenido, siendo este valor 3. Por ello, todas las matrices obtenidas en la traslación y rotación que tengan 3 ventosas activas pueden ser empleadas para la sujeción de la figura.

4.4. FIGURA CORTE4D

En primer lugar, se ha leído el archivo de AutoCAD guardado en DXF. Este archivo se ha llamado “corte4D.dxf”, donde se encuentra una figura diseñada por un conjunto de líneas.

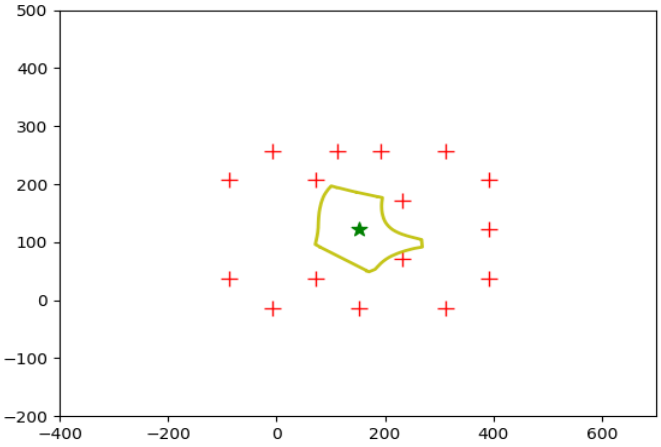
Una vez se ha leído, se ha mostrado por pantalla los puntos que forman el polígono generado para poder emplear esa figura en el código Python. La figura se ha mostrado en las gráficas mediante líneas amarillas.

[(146.7946044062666, 186.2865749733025), (152.1692663049677, 185.1837560966724), (162.821770053228, 183.0709939488435), ..., (124.5164477458676, 191.222508098328), (133.7551272010039, 189.1088141930416), (137.8414485046429, 188.2069393661247)]

Después, se ha calculado el centro de la figura mediante “centroid”. Este valor se ha mostrado por pantalla y en la gráfica, representado con una estrella.

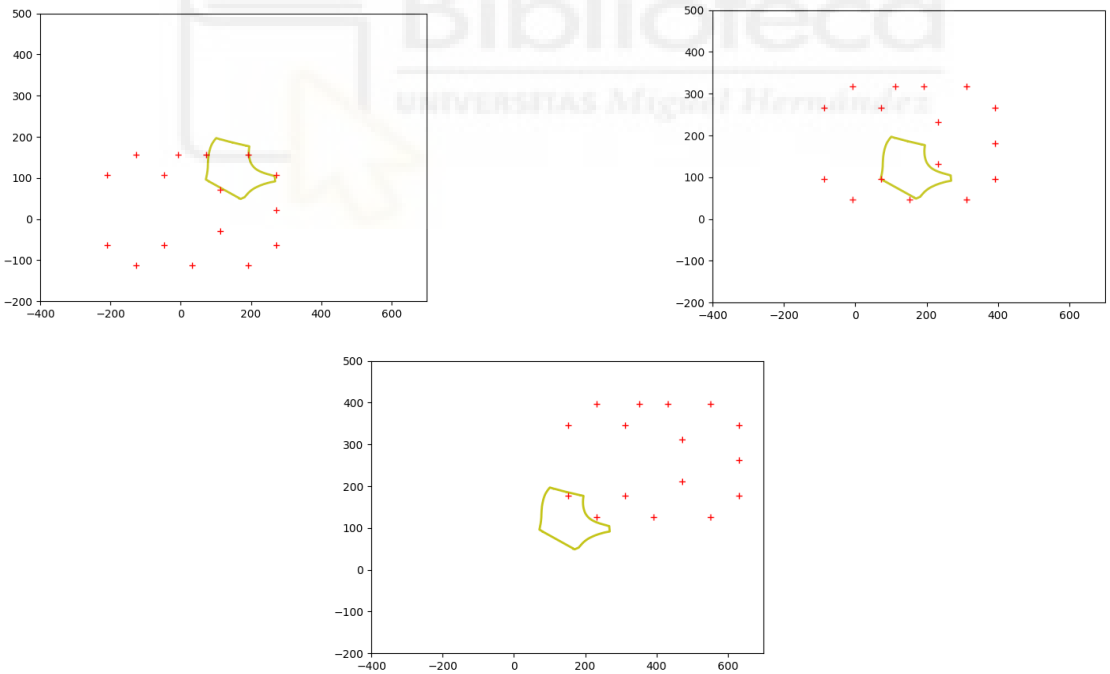
(151.76151132698763, 121.76435528019644)

La matriz de ventosas se ha transportado al centro de la figura, siendo representada con el signo de la operación suma y de color rojo (Gráfica 16).



Gráfica 16. Figura corte4D.dxf con matriz de ventosas en su centro

Se pasa a la primera parte del código, mostrando la traslación de la matriz por una superficie, guardando todas las posiciones obtenidas. Esto es mostrado mediante una gráfica animada (Gráfica 17).



Gráfica 17. Traslación de la matriz en la figura corte4D.dxf

Una vez que la matriz ha pasado por todas las posibles posiciones que se le han indicado, se ha enviado a una función para escoger aquellas matrices con mayor número de ventosas dentro de la figura, conocida como “bigger”.

Cuando la función ha finalizado, se muestra por pantalla las matrices con el máximo número de ventosas dentro de la figura, las ventosas activadas en cada matriz, el número total de matrices con el máximo número de ventosas activadas y los valores empleados en la traslación para el desplazamiento de la matriz.

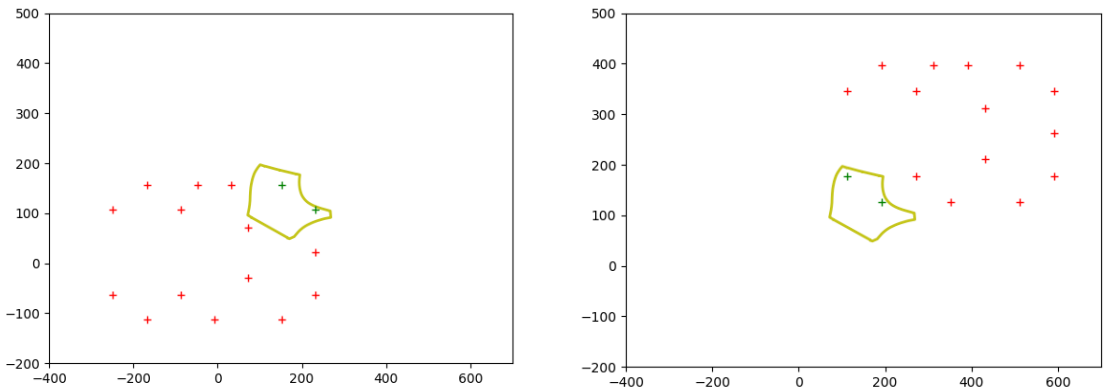
[[[191.7615113269876, 66.76435528019644, 1.0, 1], [191.7615113269876, -18.23564471980356, 1.0, 2], [191.7615113269876, -103.23564471980356, 1.0, 3], [111.7615113269876, 116.76435528019647, 1.0, 4], …, [191.76151132698763, 396.76435528019647, 1.0, 13], [191.76151132698763, 126.76435528019644, 1.0, 14], [111.7615113269876, 346.76435528019647, 1.0, 15], [111.7615113269876, 176.76435528019644, 1.0, 16]]]

[[1, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 8, 0, 10, 0, 0, 0, 0, 0, 0], …, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0, 14, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 14, 0, 16]]

30

[[40, 0, 0], [240, 0, 0], [320, 0, 0], …, [40, 280, 0], [280, 280, 0], [360, 280, 0], [440, 280, 0]]

Después se ha mostrado por pantalla mediante una animación todas las matrices con mayor número de ventosas dentro de la figura. Distinguiéndose las ventosas dentro de la figura y fuera de ella (Gráfica 18).

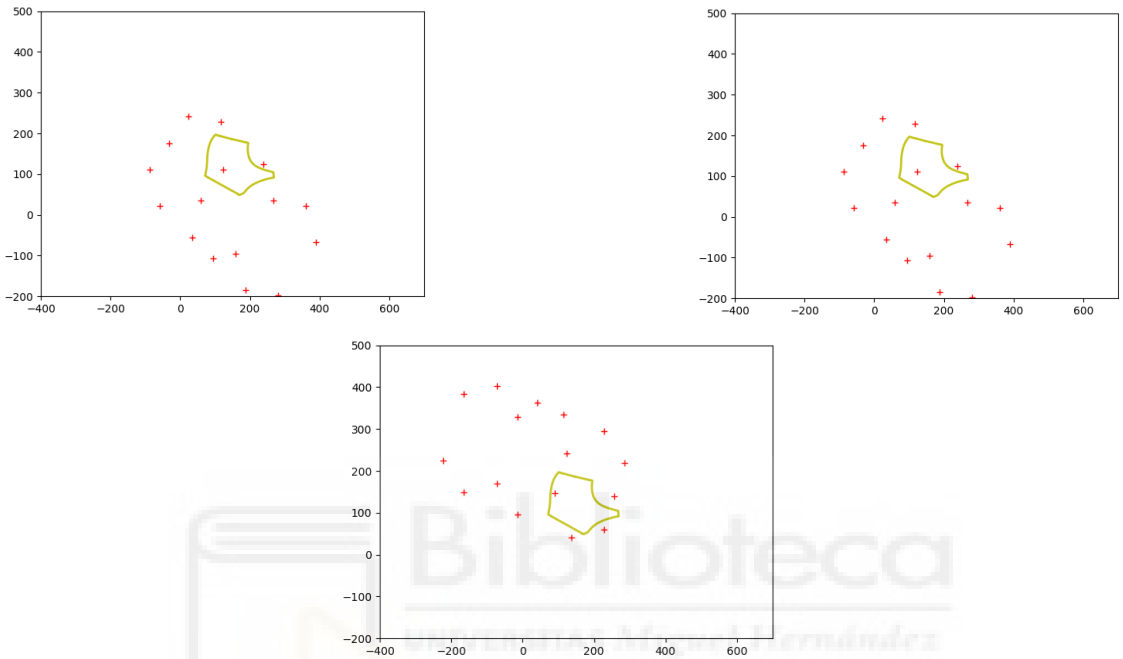


Gráfica 18. Matrices de traslación con mayor número de ventosas activas en la figura corte4D.dxf

Al realizar la traslación, el valor más alto de ventosas dentro de la figura es 2. Por ello, todas las matrices que se han obtenido mediante la función “bigger” cumplen con esa condición, siendo un total de 30 matrices. Hasta el momento, esas matrices son las que se pueden emplear a la hora de coger la pieza.

Una vez se tienen todos los valores máximos, se pasa a la segunda parte del código. Es bastante similar a la anterior, siendo su diferencia que el movimiento será de rotación.

En primer lugar, se ha mostrado en una gráfica las diferentes matrices de rotación, partiendo siempre de las matrices de traslación obtenidas anteriormente con el máximo número de ventosas dentro la figura.



Gráfica 19. Rotación de la matriz en la figura corte4D.dxf

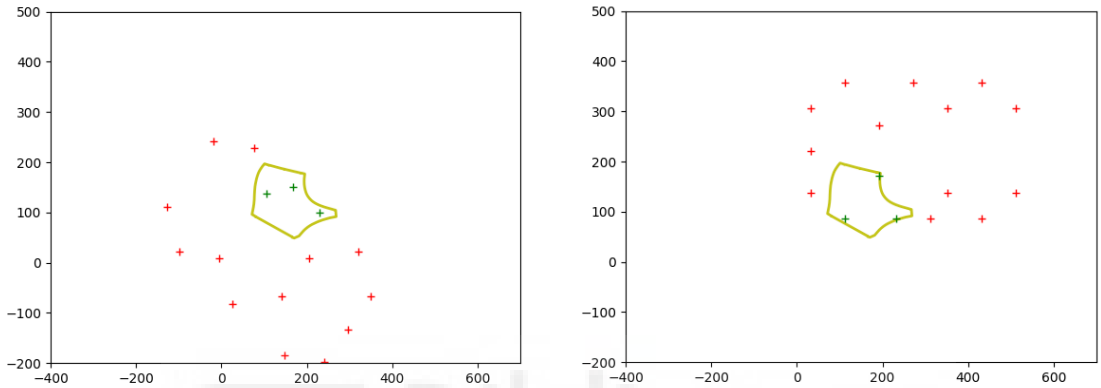
Cuando se han obtenido todas las posiciones de las matrices posibles, se han enviado a una función para obtener las matrices con el mayor número de ventosas dentro de la figura, las ventosas activas de cada matriz, el número de matrices máximas y las posiciones que se han empleado para que se realice la rotación. Estos datos se han mostrado por pantalla una vez obtenidos.

[[[390.24912449889814, -107.39089337945994, 1.0, 1], [335.6121776755423, -172.50467104457306, 1.0, 2], [280.9752308521864, -237.6184487096862, 1.0, 3], [361.1049495337069, -17.66566244858789, 1.0, 4], …, [431.7615113269876, 86.76435528019644, 1.0, 13], [431.7615113269876, 356.76435528019647, 1.0, 14], [511.7615113269876, 136.76435528019644, 1.0, 15], [511.7615113269876, 306.76435528019647, 1.0, 16]]]

[[0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 11, 0, 13, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 8, 0, 10, 11, 0, 0, 0, 0, 0], …, [0, 0, 0, 0, 0, 0, 0, 8, 0, 10, 11, 0, 0, 0, 0, 0], [0, 0, 0, 4, 0, 6, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0]]

[[151.7615113269876, -18.23564471980356, 320], [151.7615113269876, -18.23564471980356, 340], …, [231.7615113269876, 221.76435528019647, 140], [271.7615113269876, 221.76435528019647, 180]]

Las matrices con mayor número de ventosas dentro de la figura se han mostrado en una gráfica animada, en la que se distinguen las ventosas de dentro y fuera de la figura (Gráfica 20).



Gráfica 20. Matrices de rotación con mayor número de ventosas activas en la figura corte4D.dxf

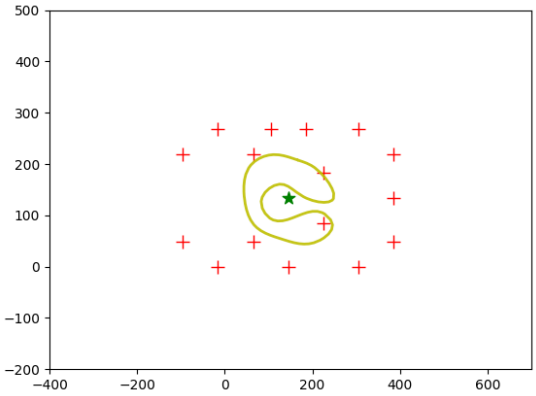
En esta figura, la cantidad de matrices que se pueden emplear para la cogida de la figura ha disminuido a 12. Pero lo que más llama la atención es que el número de ventosas activas ha aumentado a 3, por lo que las matrices obtenidas anteriormente no podrán ser empleadas para coger la figura, ya que el número de ventosas activas era menor. A eso es debido que el valor de las matrices seleccionadas por la función “bigger” haya disminuido.

4.5. FIGURA CORTE5E

En primer lugar, se ha obtenido un polígono mediante el archivo DXF, “corte5E.dxf”, obtenido en AutoCAD. Se ha calculado el centro del polígono, siendo mostrado mediante una estrella en la gráfica. También se ha desplazado la matriz de ventosas de su centro de coordenadas al centro de la figura (Gráfica 21).

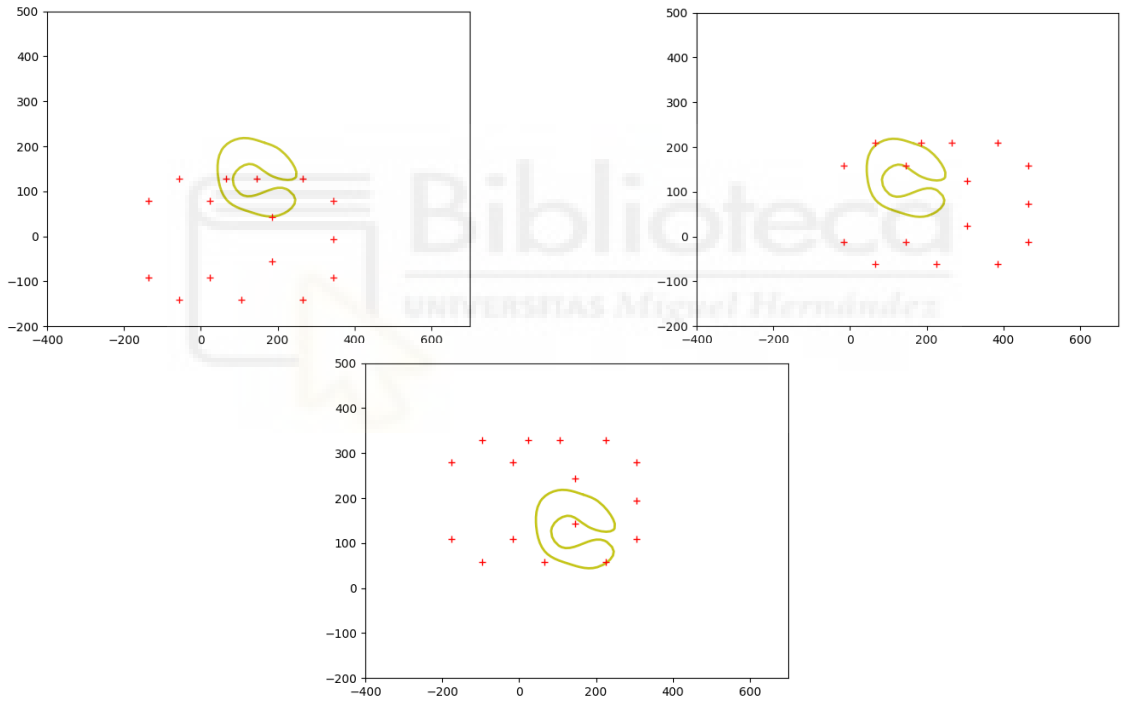
[(166.7273121575006, 207.6976792811605), (156.2750982507291, 210.7054009341267), (145.6221489009399, 213.654772741702), ..., (197.6781658619963, 197.1629576233685), (187.2891996017215, 201.5578369938161), (177.0223557193365, 204.8282824234149)]

(144.6509868936636, 133.71474186387871)



Gráfica 21. Figura corte5E.dxf con matriz de ventosas en su centro

Después, la matriz se ha llevado a una posición inicial de trabajo para empezar la traslación, obteniendo así diversas matrices (Gráfica 22).



Gráfica 22. Traslación de la matriz en la figura corte5E.dxf

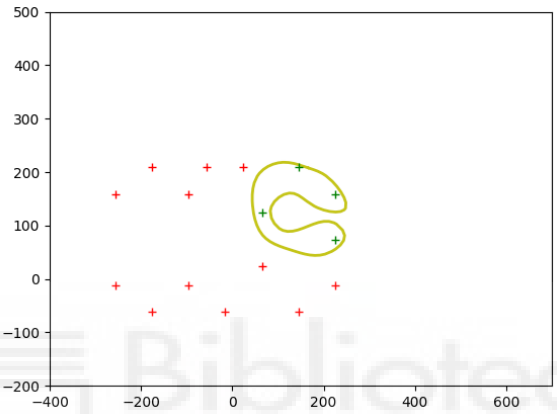
Una vez se han obtenido todas las matrices, se ha buscado cuales tienen un mayor número de ventosas dentro de la figura, además de la ubicación de esas ventosas, el número de matrices seleccionadas y las coordenadas que se han empleado para desplazar la matriz. Estos valores se han mostrado por pantalla, incluyendo una gráfica en la que aparecen las matrices seleccionadas por la función “bigger” con el mayor número de ventosas dentro de la figura.

[[[384.65098689366357, 118.71474186387871, 1.0, 1], [384.65098689366357, 33.714741863878714, 1.0, 2], [384.65098689366357, -51.285258136121286, 1.0, 3], …, [-175.3490131063364, -61.285258136121286, 1.0, 14], [-255.34901310633643, 158.71474186387871, 1.0, 15], [-255.34901310633643, -11.285258136121286, 1.0, 16]]]

[[0, 0, 0, 0, 0, 6, 0, 8, 0, 10, 11, 0, 0, 0, 0, 0], [1, 2, 0, 4, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

2

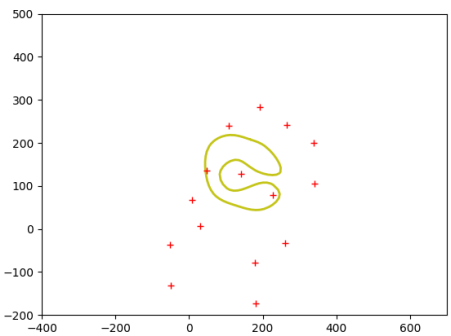
[[240, 40, 0], [80, 80, 0]]

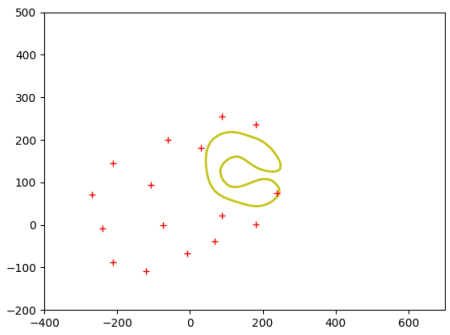


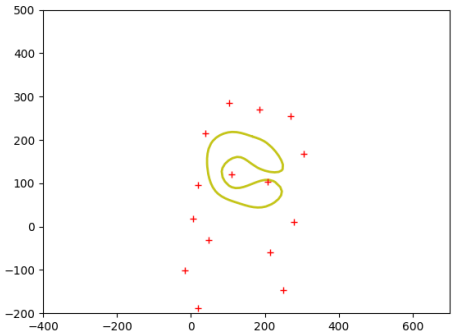
Gráfica 23. Matrices de traslación con mayor número de ventosas activas en la figura corte5E.dxf

El número máximo de ventosas activas en esta figura es 4, teniendo únicamente 2 matrices que cumplen la condición, por lo que únicamente se pueden utilizar esas 2 matrices para coger el objeto de momento.

Cuando se han obtenido todas las matrices con el máximo número de ventosas activas en la traslación, se realiza la rotación de esas mismas matrices, permitiendo la búsqueda de nuevas matrices con un mayor número de ventosas activas (Gráfica 24).







Gráfica 24. Rotación de la matriz en la figura corte5E.dxf

Una vez se han obtenido todas las matrices, se buscan de nuevo aquellas que tengan un mayor número de ventosas dentro de la figura, además de las ventosas que se han activado, el número total de matrices seleccionadas y las coordenadas que se han empleado para obtener las matrices seleccionadas.

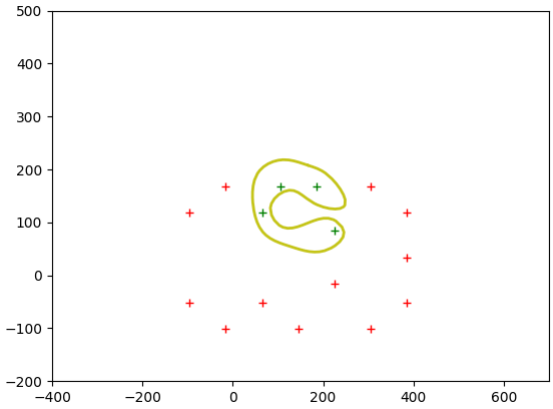
Estos valores se han mostrado por pantalla y las matrices seleccionadas se han mostrado gráficamente. Las ventosas de estas matrices se diferencian en ventosas activas y ventosas inactivas (Gráfica 25).

[[[384.65098689366357, 118.71474186387871, 1.0, 1], [384.65098689366357, 33.714741863878714, 1.0, 2], [384.65098689366357, -51.285258136121286, 1.0, 3], …, [-211.87255178104704, 1.579460989888048, 1.0, 14], [-211.8035299122726, 235.6734490288414, 1.0, 15], [-269.9469542776363, 75.92570349523696, 1.0, 16]]]

[[0, 0, 0, 0, 0, 6, 0, 8, 0, 10, 11, 0, 0, 0, 0, 0], [1, 2, 0, 4, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 0, 0, 4, 0, 6, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0]]

3

[[144.65098689366357, 33.714741863878714, 0], [-15.349013106336429, 73.71474186387871, 0], [-15.349013106336429, 73.71474186387871, 340]]



Gráfica 25. Matriz de rotación con mayor número de ventosas activas en la figura corte5E.dxf

A la hora de realizar la rotación, se ha seguido manteniendo el número máximo de ventosas activas. En cambio, la cantidad de matrices con el máximo número de ventosas dentro de la figura ha aumentado a 3, por lo que las matrices de traslación y la incluida mediante la rotación pueden ser empleadas para la sujeción de la figura.

5. MODIFICACIÓN DEL SOFTWARE

Una vez se tiene el software completado, es la hora de modificar ciertas partes y así mejorar el código realizado, siendo más próximo al objeto buscado. Estas mejoras pueden ayudar a optimizar el código y dotarlo de un coste computacional inferior. También puede pasar todo lo contrario y que su coste computacional se dispare.

La mayoría de las modificaciones se han realizado en el Driver Code, enfocándose los cambios a la hora de mostrarlos por pantalla y al comprobar si un objeto se encuentra dentro o fuera de la figura.

5.1. CAMBIO DE PUNTO A CIRCUNFERENCIA

La primera modificación realizada consiste en cambiar la función “is_inside_polygon”, ya que esta función comprueba si un punto está dentro de la figura o no. El problema es que las ventosas no son simples puntos, sino que están representadas como circunferencias de un radio determinado. Este dato es importante, ya que, si la ventosa representada como un punto en el código se ha indicado que se encuentra en el interior de la figura, pero está próxima al contorno de esta, puede ser que gran parte de la ventosa se encuentre fuera de la figura a la hora de trabajar.

Por ello, el código se ha modificado para comprobar si una circunferencia de radio determinado se encuentra dentro de la figura y si es así, se active.

En primer lugar, se han eliminado las funciones “onSegment”, “orientation”, “doIntersect” e “is_inside_polygon”. Esta última siendo sustituida por “contains”.

El método “contains” se ha empleado para comprobar si un objeto se encuentra dentro de otro objeto o no, devolviendo un valor booleano como verdadero o falso. Para ello se ha comprobado si en “pol”, siendo aquí donde se ha guardado el polígono, se encuentra en su interior la circunferencia, guardada en “circle”.

5.2. MOSTRAR POR PANTALLA

A la hora de mostrar la matriz de ventosas trasladada al centro de la figura, se ha utilizado un bucle for hasta la cantidad de “puntomodificado”. Dentro de este bucle se ha igualado “center” a “Point” de “puntomodificado[i][0]” y “puntomodificado[i][1]”. Después, se ha realizado “center.buffer”, empleando el radio definido fuera del Driver Code, guardado como “radius”. También se ha empleado “center” para obtener “centros” con “xy”.

Una vez que se tienen todos los valores, se genera la figura para representarla, mediante “plt.Circle”, añadiendo a éste “centros”, “radius”, “fill = True”, para que se coloreé el interior del círculo y el color a emplear. Esto se ha guardado en “circulo”, siendo introducido en “ax.add_patch”. Y así finaliza el bucle for.

Función: Driver Code

Inicio

.

.

.

Desde i hasta range(len(puntomodificado)) Hacer:

center es igual a Point() con puntomodificado[i][0] y puntomodificado[i][1]

centros es igual a center.xy

circulo es igual a plt.Circle() con centros, radius, fill igual a Verdadero y color igual a azul

ax.add_patch() de circulo

Siguiente

.

.

.

Fin

.

.

.

```
for i in range(len(puntomodificado)):
  center = Point(puntomodificado[i][0], puntomodificado[i][1])
  centros= center.xy
  circulo= plt.Circle(centros, radius, fill= True, color='b')
  ax.add_patch(circulo)
```

.

.

.

A la hora de mostrar todas las matrices compuestas por circunferencias, se declara una gráfica en la que se va a mostrar la matriz en todas las posiciones que ha adquirido, pudiendo observar cuando pasa por la figura. Se ha guardado en “matrices_traslacion” como lista “guardarpuntos” y en “num_trasl” como entero la cantidad de “matrices_traslacion” dividida por “num_coor”.

Después, se ha empleado un bucle for hasta el valor total de “num_trasl”. Dentro de este bucle se ha igualado “avance” con la variable empleada en el bucle multiplicada por “num_coor”.

Además de este bucle, dentro se encuentran dos bucles for por separado. El primero se ha empleado para eliminar los datos guardados en “circulo”, para que así no se solapen las matrices a la hora de representarlas. Para ellos, se tiene un bucle for de “circulo” hasta “ax.patches”. Dentro de este, se utilizará la función “remove” con “circulo”.

El otro bucle for llega hasta el valor total de “num_coor”. Primero, se tiene “k” igualado a la variable del bucle más “avance”. Después, se han generado las circunferencias para representarlas, igualando “center” a “Point”, teniendo dentro “matrices_traslacion[k][0]” y “matrices_traslacion[k][1]”. Luego se ha creado “centros” con “center.xy”.

Una vez se han obtenido estos dos datos, se ha guardado en “circulo” su representación mediante “plt.Circle”, añadiendo “centros”, “radius”, “fill=True” y el color que se ha utilizado.

Cuando se ha terminado el bucle, se produce una pausa de 0.5 y fuera del primer bucle se ha realizado “plt.show”, para mostrarlo por pantalla.

Función: Driver Code

Inicio

.

.

.

fig y ax es igual a plt.subplot()

ax.set_xlim (valor inicial gráfica eje x, valor final gráfica eje x)

ax.set_ylim (valor inicial gráfica eje y, valor final gráfica eje y)

Se hace plt.plot de x e y

matrices_traslacion es igual a lista de guardarpuntos

num_trasl es igual al entero de len(matrices_traslacion) dividido entre num_coor

Desde j hasta range(num_trasl) Hacer:

avance es igual a j multiplicado por num_coor

Desde circulo hasta ax.patches Hacer:

circulo.remove()

Siguiente

Desde i hasta range(num_coor) Hacer:

k es igual a i más avance

center es igual a Point() con matrices_traslacion[k][0] y matrices_traslacion[k][1]

centros es igual a center.xy

circulo es igual a plt.Circle() con centros, radius, fill igual a Verdadero y color igual a azul

ax.add_patch() de circulo

Siguiente

Pausa de 0.5

Siguiente

plt.show()

.

.

.

Fin

.

.

.

```
fig, ax = plt.subplots()
ax.set_xlim(-400, 700)
ax.set_ylim(-200, 500)
plt.plot(x, y, color='y', alpha=0.9, linewidth=2, solid_capstyle='round', zorder=2)
matrices_traslacion = list(guardarpuntos)
num_trasl = int(len(matrices_traslacion) / num_coor)

for j in range(num_trasl):
  avance = j * num_coor
```

```
  for circulo in ax.patches:
      circulo.remove()


  for i in range(num_coor):
      k = i + avance
      center = Point(matrices_traslacion[k][0], matrices_traslacion[k][1])
      centros = center.xy
      circulo = plt.Circle(centros, radius, fill=True, color='b')
      ax.add_patch(circulo)


  plt.pause(0.5)
plt.show()
```

.

.

.

En la parte de rotación es igual. Únicamente cambia “matrices_traslacion” por “matrices_rotacion” y “num_tras” por “num_rot”.

También se tiene la representación de las matrices con el mayor número de ventosas dentro de la figura. Para ello, se ha declarado una gráfica en la que se muestra el movimiento de la matriz en todas las posiciones que ha adquirido. Se ha guardado en “matrices_traslacion” una lista de “puntosmaximost” y en “num_trasl” como entero la longitud de “matrices_traslacion”.

Se ha generado un bucle for hasta el valor “num_trasl”. En el bucle, se ha guardado “matrices_traslacion[j]” en “matrizavance” y “ventosasmaximast[j]” en “ventosas”, siendo “j” la variable escogida para el bucle.

Después, se han realizado tres bucles for, dos de ellos empleados para eliminar el valor de “circuloout”, que son los valores que se encuentran fuera de la figura; y “circuloin”,

siendo los valores que están dentro de la figura. Estos bucles se utilizan para que una vez se muestre la matriz, esta no se solape con las siguientes.

En el primer bucle for se ha utilizado “circuloout” hasta “ax.patches”. Dentro de este bucle, se ha empleado “remove” en “circuloout”. El otro bucle for es igual, la única diferencia es que en vez de “circuloout” se utiliza “circuloin”.

El tercer bucle for es hasta el rango de “num_coor”. Dentro, se ha obtenido “center” mediante “Point”, introduciendo “matrizavance[i][0]” y “matrizavance[i][1]”; y “centros”, igualándolo a “center” con “xy”.

Después, dentro del bucle, se ha introducido una sentencia if, en la que dependiendo del valor de “ventosas”, sucede una cosa u otra. Si “ventosas[i]” es igual a 0, “circuloout” dibuja el círculo de ese punto utilizando “plt.Circle”, siendo introducido en “ax.add_patch”. Pero si sucede lo contrario, el círculo es dibujado por “circuloin”. A la hora de graficarlos, se diferencian por colores.

Fuera de este bucle, se hace una pausa de 0.5 y una vez que ha terminado el primer bucle de “num_trasl”, se muestra mediante “plt.show”.

Función: Driver Code

Inicio

.

.

.

fig y ax es igual a plt.subplot()

ax.set_xlim (valor inicial gráfica eje x, valor final gráfica eje x)

ax.set_ylim (valor inicial gráfica eje y, valor final gráfica eje y)

Se hace plt.plot de x e y

matrices_traslacion es igual a lista de puntosmaximost

num_trasl es igual a entero de len(matrices_traslacion)

Desde j hasta range(num_trasl) Hacer:

matrizavance es igual a matrices_traslacion[j]

ventosas es igual a ventosasmaximast[j]

Desde circuloout hasta ax.patches Hacer:

circuloout.remove()

Siguiente

Desde circuloin hasta ax.patches Hacer:

circuloin.remove()

Siguiente

Desde i hasta range(num_coor) Hacer:

center es igual a Point() con matrizavance[i][0] y matrizavance[i][1]

centros es igual a center.xy

Si ventosas[i] es igual a 0 Entonces:

circuloout es igual a plt.Circle() con centros, radius, fill igual a Verdadero y color igual a azul

ax.add_patch() de circuloout

Sino:

circuloin es igual a plt.Circle() con centros, radius, fill igual a Verdadero y color igual a verde

ax.add_patch() de circuloin

FinSi

Siguiente

Pausa de 0.5

Siguiente

plt.show()

.

.

.

```
Fin
.
.
.
fig, ax = plt.subplots()
ax.set_xlim(-400, 700)
ax.set_ylim(-200, 500)
plt.plot(x, y, color='y', alpha=0.9, linewidth=2, solid_capstyle='round', zorder=2)
matrices_traslacion = list(puntosmaximost)
#print(len(matrices_traslacion))
num_trasl = int(len(matrices_traslacion))

for j in range(num_trasl):
  matrizavance = matrices_traslacion[j]
  ventosas = ventosasmaximast[j]

  for circuloout in ax.patches:
    circuloout.remove()

  for circuloin in ax.patches:
    circuloin.remove()

  for i in range(num_coor):
    center = Point(matrizavance[i][0], matrizavance[i][1])
    centros = center.xy
```

```
        if ventosas[i] == 0:
            circuloout = plt.Circle(centros, radius, fill=True, color='b')
            ax.add_patch(circuloout)
        else:
            circuloin = plt.Circle(centros, radius, fill=True, color='g')
            ax.add_patch(circuloin)


    plt.pause(0.5)
plt.show()
```

.

.

.

La rotación es completamente igual, siendo la única diferencia el nombre de algunos datos como es el caso de “matrices_rotacion”, “puntosmaximosr” y “num_rot”.

5.3. COMPROBACIÓN DE LA CIRCUNFERENCIA DENTRO DE LA FIGURA

El único cambio realizado para comprobar si la circunferencia se encuentra dentro o fuera de la figura ha sido sustituir la función “is_inside_polygon” por “contains”, introduciendo dentro de este “circle”, como se ha explicado en el apartado del cambio de punto a circunferencia.

Para la traslación, dentro de la sentencia while de “tx” se encuentra un bucle for hasta el rango de “puntomodificado”. Dentro de este bucle, se ha obtenido “center” mediante “Point” de “puntomodificado[i][0]” y “puntomodificado[i][1]”, siendo “i” la variable del bucle. Después se ha obtenido “circle” con “center” empleando “buffer” con “radius”.

Una vez se ha conseguido “circle”, se ha realizado una sentencia if en la que si “pol.contains(circle)” es verdadero, se añade en “ventosast” el “puntomodificado[i][3]” y “sum” aumenta un valor. En el caso de que sea falso, se añade 0 a “ventosast”, indicando así las ventosas activas e inactivas.

Función: Driver Code

Inicio

.

.

.

Desde i hasta range(len(puntomodificado)) Hacer:

center es igual a Point() con puntomodificado[i][0] y puntomodificado[i][1]

circle es igual a center.buffer() de radius

Si pol.contains() de circle es igual a verdadero Entonces:

ventosast.append de puntomodificado[i][3]

sum es igual a sum más 1

Sino:

ventosast.append de 0

FinSi

Siguiente

.

.

.

Fin

.

.

.

```
for i in range(len(puntomodificado)):
  center = Point(puntomodificado[i][0], puntomodificado[i][1])
  circle = center.buffer(radius)
```

```
  if (pol.contains(circle)):
     # print('Yes')
     ventosast.append(puntomodificado[i][3])
     sum = sum + 1
  else:
     ventosast.append(0)
```

.

.

.

Para la rotación es completamente igual, cambiando únicamente la variable “i” por “r”.

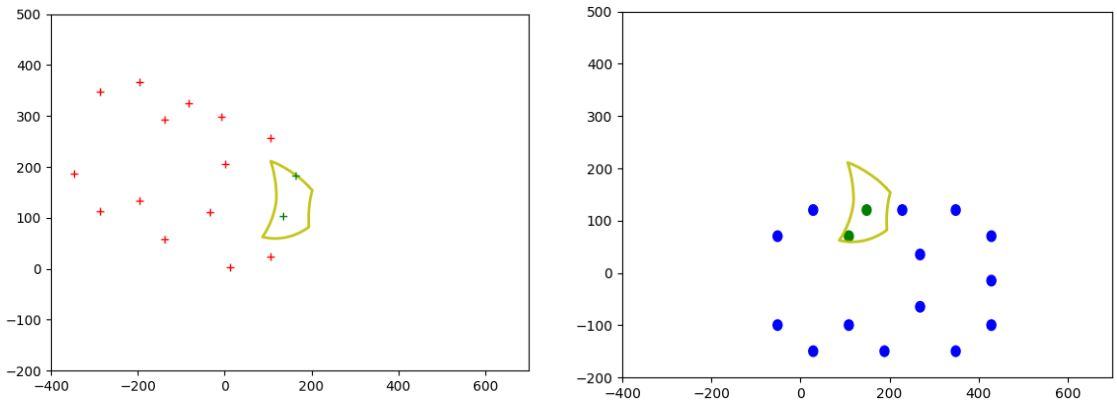
El software completo modificado se encuentra en el apartado 8.3. ANEXO *III*.

5.4. COMPARACIÓN DE SOFTWARES

Al modificar el software principal, las ventosas activas pueden haber variado. Esto es debido a que en el software sin modificar se ha comprobado si un punto se encuentra dentro de la figura. En el caso de que estuviera, la ventosa se activaría y esta agarraría la figura. En el nuevo software no se trabaja con puntos, sino con circunferencias de un radio determinado, por lo que la circunferencia debe estar completamente dentro de la figura para que esta se active. En el caso de que se salga un poco la circunferencia, la ventosa queda invalidada. Por ello, en el software modificado el valor de las ventosas activadas es menor o igual al valor de las ventosas activadas del software inicial.

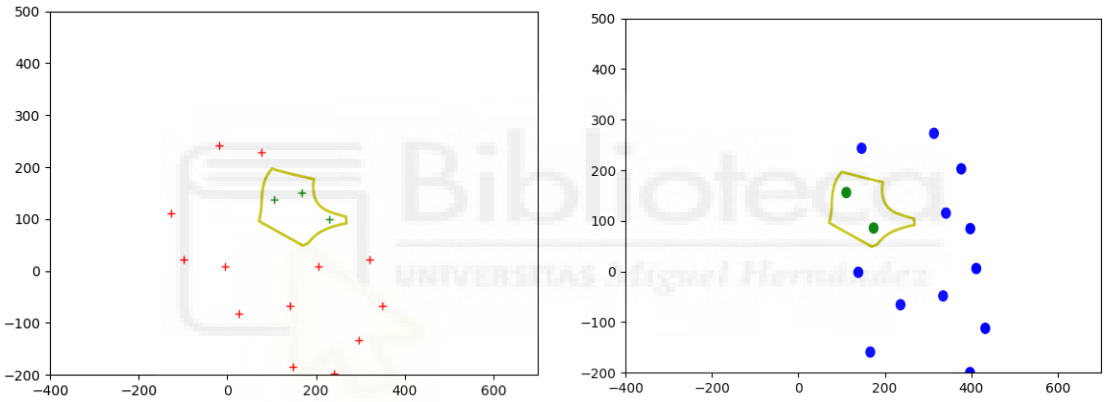
Esto se puede comprobar a la hora de ejecutarlo. Incluso aunque el número máximo de ventosas activadas sea el mismo, la cantidad de matrices con mayor número de ventosas dentro de la figura puede variar.

Si se trabaja con “corte1A.dxf”, sucede que el número máximo de ventosas activas es el mismo, siendo este de 2, pero cuando se realiza la traslación o rotación de las matrices, únicamente será correcta 1 opción (Gráfica 26). Esto con el software inicial no sucede, dotando al usuario con diversas soluciones.



Gráfica 26. Comparación del Software inicial (izquierda) con el Software modificado (derecha) con corte1A.dxf

En cambio, en la figura “corte4D.dxf”, lo que sucede es que con el software modificado el valor de ventosas activadas ha disminuido, pasando de 3 en el caso del software inicial, a 2 en el software modificado (Gráfica 27).



Gráfica 27. Comparación del Software inicial (izquierda) con el Software modificado (derecha) con corte4D.dxf

Estos cambios realizados permiten obtener un mejor funcionamiento del software, llegando al objeto deseado con mayor exactitud. Al modificar el código se ha obtenido la cantidad de ventosas correctas que podrán ser activadas y trabajar para coger la pieza sin salirse de su contorno.

6. CONCLUSIÓN

El objetivo planteado ha sido llevado a cabo con el método de prueba y error. Esto ha hecho que el proceso de creación del código se haya alargado, ya que se han ido realizando cambios constantemente para llegar al resultado final.

La realización del código en Python ha llevado una serie de modificaciones a lo largo de este proceso. Se empezó únicamente comprobando si un punto se encontraba dentro de una figura, concretamente, dentro de su contorno. En realidad, ese era el objeto principal,

pero se ha tenido que agrandar para completarlo, ya que no se partía de un punto, sino de 16. Además, al ser empleados para coger un objeto, se tenía que buscar la mayor cantidad de puntos posibles entre esos 16 y luego mostrarse como una matriz para así poder seleccionarlos.

Que únicamente trabajara con puntos se quedaba muy corto respecto a lo que se buscaba, ya que como se ha comprobado, algunos de esos puntos dados como válidos si se extrapolaban a la realidad no servían. Así que tocó buscar otra vía para conseguir el objeto deseado, por ello se utilizaron circunferencias, representando a las ventosas. Estas al tener un radio, el cual se puede modificar para otras ventosas, han permitido un criterio más exacto a la hora de darse como válidas o desecharse para coger la pieza.

La lectura de la figura ha sido un reto a la hora de decidir como tendría que ser. Al final se ha optado por que fuera realizada mediante líneas rectas unidas, pudiendo generar cualquier forma sin ser perceptible por el ojo humano de que eran un conjunto de rectas. La utilización de semicircunferencias se vio declinada por los problemas que acarreaba a la hora de plasmarlo en un polígono, mostrando una complicada lectura y representación en el código, ya que estas se dibujaban con diferentes ángulos.

A la hora de mostrar el proceso gráficamente, se ha optado por una serie de animaciones. Antes se tenía la aparición de ventanas conforme se cerraban otras manualmente. Al ser tedioso de esa forma ya que en el caso de que fueran muchas matrices llevaría un gran periodo de trabajo, se optó por la opción final.

Al utilizar el código modificado con las figuras proporcionadas para hacer las pruebas de funcionamiento, su tiempo de trabajo ha rondado los 2 minutos aproximadamente entre la media de todas piezas. Todo dependerá de la pieza que se use, pudiendo alargar o disminuir el tiempo de trabajo.

En definitiva, el código realiza el objeto que se buscaba después de tener diversas modificaciones. Eso no quiere decir que no se pueda mejorar, llegando a un estado de optimización máximo y con un menor tiempo de trabajo respecto al que ha tenido.

7. TRABAJOS FUTUROS

El código obtenido es sólo el principio de la investigación. A partir de este, se puede realizar una serie de investigaciones y trabajos futuros para mejorarlo.

Por ello, se propone la investigación de una nueva forma para leer los archivos DXF con el fin de crear un polígono y ser mostrado por pantalla en una gráfica. Buscar nuevas formas en las que, en vez de emplear líneas rectas unidas de una forma concreta, se pueda trabajar con polilíneas, circunferencias, semicircunferencias, splines, etc, siendo leídas correctamente y seleccionando aquellos datos necesarios para ser mostrada la figura.

Para comprobar si es posible optimizar el código realizado, se propone la trascripción del código a C/C++ y comprobar si se produce una gran diferencia. Para ello se requerirá de conocimiento de este lenguaje, siendo plasmado de una forma diferente al realizado en código Python.

Relacionado con C/C++, se puede trabajar con Codon [4]. Codon es una implementación de Python que compila el código Python en código de máquina nativo. Su rendimiento es similar al de C/C++.

Así mismo, es importante comprobar su funcionamiento en la vida real. Para ello, es interesante la integración de la pinza con el robot y trabajar con diferentes elementos, pudiendo realizar futuras modificaciones en el software. Como el caso anterior, lleva unas horas de cómputo, debiendo de trabajar con las URcaps, dejándose así para trabajos futuros.

8. ANEXOS

8.1. ANEXO I. FUNCIONES

A la hora de llevar a cabo el código, se han tenido que realizar funciones, complementando al Driver Code.

En este apartado se tiene presente las líneas de código realizadas y una breve explicación de su función, dando lugar a un mejor entendimiento.

8.1.1. ONSEGMENT

```
#Se dan tres puntos colineales
#La función mira si el punto q descansa en la línea del segmento 'pr'
def onSegment(p: tuple, q: tuple, r: tuple) -> bool:
    if ((q[0] <= max(p[0], r[0])) &
            (q[0] >= min(p[0], r[0])) &
            (q[1] <= max(p[1], r[1])) &
            (q[1] >= min(p[1], r[1]))):
        return True

    return False
```

La función “onSegment” es empleada para trabajar dentro de la función “is_inside_polygon”. En esta función se comprueba si un punto descansa en un segmento generado por otros dos puntos distintos. Para ello, estos puntos deberán ser colineales.

8.1.2. ORIENTATION

```
#Para encontrar la orientación del triplete ordenado (p,q,r)
#La función devuelve:
#0-> p,q y r son colineales
#1-> sentido horario
#2-> sentido anti-horario
def orientation(p: tuple, q: tuple, r: tuple) -> int:
    val = (((q[1] - p[1]) *
```

```
        (r[0] - q[0])) -
        ((q[0] - p[0]) *
        (r[1] - q[1])))

    if val == 0:
        return 0
    elif val > 0:
        return 1
    else:
        return 2
```

La función “orientation” es empleada para trabajar dentro de las funciones “doIntersect” e “is_inside_polygon”. En esta función se busca la orientación del triplete ordenado, devolviendo un resultado diferente dependiendo de los puntos. En el caso de que los puntos sean colineales, se devuelve 0; si se encuentra en sentido horario, devuelve un 1; y si están en sentido antihorario, devuelve un 2.

8.1.3. DOINTERSECT

```
def doIntersect(p1, q1, p2, q2):

    #Encontrar 4 orientaciones necesarias
    #para casos generales y especiales
    o1 = orientation(p1, q1, p2)
    o2 = orientation(p1, q1, q2)
    o3 = orientation(p2, q2, p1)
    o4 = orientation(p2, q2, q1)

    #caso general
    if (o1 != o2) and (o3 != o4):
```

```
    return True

#Casos especiales
#p1, q1 y p2 son colineales y
# p2 descansa en el segmento p1q1
if (o1 == 0) and (onSegment(p1, p2, q1)):
    return True

#p1,q1 y q2 son colineales y
#q2 descansa en el segmento p1q1
if (o2 == 0) and (onSegment(p1, q2, q1)):
    return True

#p2, q2 y p1 son colineales y
#p1 descansa en el segmento p2q2
if (o3 == 0) and (onSegment(p2, p1, q2)):
    return True

#p2, q2 y q1 son colineales y
#q1 descansa en el segmento p2q2
if (o4 == 0) and (onSegment(p2, q1, q2)):
    return True

return False
```

La función “doIntersect” es empleada para trabajar dentro de la función “is_inside_polygon”, utilizando la función “orientation” y “onSegment”. En esta función

se comprueba si es cierto el caso general o alguno de los especiales. Para ello, se emplearán 4 orientaciones para poder trabajar.

8.1.4. IS_INSIDE_POLYGON

```
#Devuelve True si el punto p descansa dentro del polígono con n vértices
def is_inside_polygon(points: list, p: tuple) -> bool:

    n = len(points)

    #Es necesario saber cuántos vértices
    #tiene el polígono n = len(points)
    #teniendo como mínimo 3
    if n < 3:
        return False

    #Se crea un punto para segmentos lineales de p a infinito
    extreme = (INT_MAX, p[1])

    #Para contar el número de puntos que forman el polígono
    decrease = 0
    count = i = 0

    while True:
        next = (i + 1) % n

        if (points[i][1] == p[1]):
            decrease += 1
```

```
    #Se ve si la línea de p al 'extremo'
    #interseca con la línea de points[i]
    #a points[next]
    if (doIntersect(points[i],
                    points[next],
                    p, extreme)):

        #Si el punto 'p' es colineal con la línea del segmento 'i-next'
        #se mira si descansa en el segmento
        if orientation(points[i], p,
                       points[next]) == 0:
            return onSegment(points[i], p,
                             points[next])

        count += 1

    i = next

    if (i == 0):
        break

#Reducir el contador
count -= decrease

return (count % 2 == 1)
```

La función “is_inside_polygon” se utiliza para comprobar si un punto se encuentra dentro de un polígono. Para ello, esta función trabajará con todas las anteriores, “onSegment”, “orientation” y “doIntersect”. Lo que sucede es que dentro de un bucle while, se comprueba si la línea interseca y si es cierto, se comprueba si la orientación es igual a 0 y se realiza la función “onSegment”, devolviendo verdadero o falso. En el caso de que no devuelva nada dentro del bucle, se devuelve verdadero si el módulo del valor “count” de 2 es igual a 1.

8.1.5. PRINT_LINE

```
#Obtener datos del archivo DXF
def print_line(e,lista):

    #Añade el punto inicial a la lista
    lista.append((float(e.dxf.start[0]), float(e.dxf.start[1])))
```

La función “print_line” es empleada para guardar los datos necesarios del archivo DXF y emplearlos después para realizar un polígono.

8.1.6. MOVE

```
#Realización de traslación y rotación
def move(puntos,t1,t2,t3):

    lista = []
    resultado = []
    t3r = np.radians(t3)

    #Se hace una lista del nuevo punto
    #Se introducen los puntos como una matriz
    for i in range(len(puntos)-1):
        lista.append(puntos[i])
```

```
    #Se convierte en matriz y se transpone para poder trabajar
    A = np.array(lista)
    AT = np.transpose(A)

    #Matriz de traslación y rotación
    T1 = np.array([[1, 0, t1],[0, 1, t2],[0, 0, 1]])
    T2 = np.array([[math.cos(t3r), -math.sin(t3r), 0],[math.sin(t3r), math.cos(t3r), 0], [0, 0, 1]])

    #Multiplición de ambas matrices y la matriz del punto
    Val1 = np.dot(T1, T2)
    Val = np.dot(Val1, AT)
    resultado = list(Val)
    resultado.append(puntos[3])

    return resultado
```

La función “move” es empleada para modificar la posición de los puntos de la matriz. Esto permite que la matriz realice la función de traslación y rotación. Es necesario hacerlo punto a punto.

8.1.7. BIGGER

```
#Encontrar las matrices máximas
def bigger(matriz, matrizventosas, cantidad, puntos):
    posiciones = []
    ventosas = []
    matrizt = []
    count = 0
    j = 0
```

```
suma = 0 #valores con el número mayor

#Busqueda del valor más alto en la lista
maximo = np.max(cantidad)

for z in range(len(cantidad)):
    count= count + num_coor

    if cantidad[z]==maximo:
        j = count - num_coor
        suma = suma + 1
        matriztrabajop = []
        matriztrabajov = []

        while j<count:
            matriztrabajop.append(matriz[j])
            matriztrabajov.append(matrizventosas[j])
            j=j+1

        posiciones.append(matriztrabajop)
        ventosas.append(matriztrabajov)
        matrizt.append(puntos[z])

#Devuelve todas las matrices máximas, las ventosas,
#la cantidad y las posiciones que se han usado
return posiciones, ventosas, suma, matrizt
```

La función “bigger” es empleada para obtener las matrices con el número máximo de ventosas activadas, las ventosas que se activan de cada matriz, el número máximo de matrices seleccionadas y los datos empleados para que los puntos hayan llegado a esas posiciones. Esta función se utilizada tanto en el proceso de traslación como el de rotación.

8.2. ANEXO II. DRIVER CODE

A parte de las funciones, es necesario tener un código principal, conocido como Driver Code. En este código se trabaja y se emplean todas las funciones mostradas en el apartado 8.1. ANEXO *I*. Sin el Driver Code, el programa no funcionaría.

En el Driver Code, se produce la llamada de funciones como es la función “print_line”, la función “is_inside_polygon” o la función “bigger”. Además, todas las declaraciones de las variables, empleadas en el código principal, son declaras dentro de este.

También trabaja con las variables declaradas fuera del Driver Code.

```
#Obtener las matrices con mayor número de ventosas dentro de la figura

INT_MAX = 10000
num_coor = 16   #puntos del gripper

#Coordenadas de avance de matriz de tralación y rotación
stepx = 40
stepy = 40
alpha = 20
.

.

.
```

Por otro lado, se ha realizado la muestra mediante gráficas de los procesos que han ido sucediendo, como es el caso de la traslación y rotación de la matriz.

Después de la declaración de variables fuera del Driver Code, se encuentras las funciones del apartado 8.1. ANEXO *I*, dando después paso al Driver Code.

.

.

.

```
# Driver code
if __name__ == '__main__':

    import matplotlib.pyplot as plt
    from shapely.geometry import Polygon
    import ezdxf
    import math
    import numpy as np

    #4X4
    puntoinicial = ([240,85,1,1],
            [240,0,1,2],
            [240,-85,1,3],
            [160,135,1,4],
            [160,-135,1,5],
            [80,50,1,6],
            [80,-50,1,7],
            [40,135,1,8],
            [0,-135,1,9],
            [-40,135,1,10],
            [-80,85,1,11],
            [-80,-85,1,12],
            [-160,135,1,13],
            [-160,-135,1,14],
            [-240,85,1,15],
```

```
        [-240,-85,1,16])


    lista = []  #guarda los puntos del polígono
    puntoiniciotrabajo = [] #guarda el punto inicial de traslación
    puntomodificado =[] #matriz que se modifica constantemente
    guardarpuntos = []  #guarda todas las matrices
    puntosmaximost = [] #guarda todas las matrices con los máximos puntos dentro de la figura traslacion
    ventosasmaximast = []
    puntosmaximosr = [] #guarda todas las matrices con los máximos puntos dentro de la figura rotación
    ventosasmaximasr = []
    puntorotacion = []  #punto inicial de la rotación
    ventosast =[]
    ventosasr =[]
    posiciones = []
    matriztrabajot = [] #almacena tx,ty,tz en matriz de traslación máximos
    matriztrabajor = [] #almacena tx,ty,tz en matriz de rotación máximos
    valores = []    #guarda los puntos dentro de la figura de cada matriz


    centro =()


    posicion = 0
    max_sum = 0
    val_maxt = 0 #guarda cuantas matrices tiene el valor máximo en traslación
    val_maxr = 0 #guarda cuantas matrices tiene el valor máximo en rotación
```

```
#Poner el DXF que se va a usar
doc= ezdxf.readfile("corte5E.dxf")

msp = doc.modelspace()
for e in msp:
    if e.dxftype()== 'LINE':
        print_line(e, lista)

#Comprobación de que todos los puntos se encuentren en la lista
print(lista)
#Generar un polígono para poder visualizarlo
pol = Polygon(lista)
x, y = pol.exterior.xy

fig, ax = plt.subplots()
ax.set_xlim(-400, 700)
ax.set_ylim(-200, 500)
#centro del polígono
centro = (pol.centroid.x, pol.centroid.y)
print(centro)
plt.scatter(centro[0], centro[1], marker='*', linewidths=1, c="g", s=90)

#Mover el gripper al centro de la pieza
puntomodificado = list(puntoinicial)
for k in range(len(puntomodificado)):
```

```
    puntomodificado[k] = move(puntomodificado[k], centro[0], centro[1], 0)

matrizcentro = puntomodificado  #posición de la matriz en el centro de la pieza

for i in range(len(puntomodificado)):
    p = (puntomodificado[i][0], puntomodificado[i][1])
    plt.scatter(p[0], p[1], marker='+', linewidths=1, c="r", s=90)

plt.plot(x, y, color='y', alpha=0.9, linewidth=2, solid_capstyle='round', zorder=2)
plt.show()
#Mover a posición de trabajo inicial para empezar la traslación
for k in range(len(puntomodificado)):
    puntoiniciotrabajo.append(move(puntomodificado[k], -240, -140, 0))

print("Primera parte:")     #traslación
#Se desplaza la matriz para trabajar la traslación en su posición inicial de trabajo
#Se mira el movimiento de traslación de la matriz por la figura

tz = 0
ty = 0
while ty<=290:

    tx= 0
    while tx<=480:

        posiciones.append([tx,ty,tz])   #valores matriz traslación
```

```
        sum = 0
        puntomodificado = list(puntoiniciotrabajo)
        #Se modifican los puntos
        for k in range(len(puntomodificado)):
            puntomodificado[k] = move(puntomodificado[k],tx,ty,tz)

        guardarpuntos.extend(puntomodificado)

        #Comprobación si el punto está dentro o fuera de la figura
        for i in range(len(puntomodificado)):

            p = (puntomodificado[i][0], puntomodificado[i][1])

            if (is_inside_polygon(lista,p)):

                ventosast.append(puntomodificado[i][3])
                sum = sum + 1
            else:
                ventosast.append(0)

        valores.append(sum)
        tx = tx + stepx

    ty = ty + stepy

#Representación de las matrices obtenidas
```

```
fig, ax = plt.subplots()
ax.set_xlim(-400, 700)
ax.set_ylim(-200, 500)
plt.plot(x, y, color='y', alpha=0.9, linewidth=2, solid_capstyle='round', zorder=2)
line, = ax.plot([], [], 'r+')
xdata, ydata = [], []
matrices_traslacion = list(guardarpuntos)
num_trasl = int(len(matrices_traslacion) / num_coor)

for j in range(num_trasl):
    avance = j * num_coor
    xdata, ydata = [], []

    for i in range(num_coor):
        k = i + avance
        new_xdata = matrices_traslacion[k][0]
        new_ydata = matrices_traslacion[k][1]
        xdata.append(new_xdata)
        ydata.append(new_ydata)

    line.set_data(xdata, ydata)
    plt.pause(0.5)
plt.show()

#Obtener las matrices con mayor cantidad de puntos dentro de la figura
```

```
        puntosmaximost, ventosasmaximast, val_maxt, matriztrabajot =
bigger(guardarpuntos,ventosast, valores, posiciones)
    print(puntosmaximost)
    print(ventosasmaximast)
    print(val_maxt)
    print(matriztrabajot)

    #Representación de las matrices obtenidas
    fig, ax = plt.subplots()
    ax.set_xlim(-400, 700)
    ax.set_ylim(-200, 500)
    plt.plot(x, y, color='y', alpha=0.9, linewidth=2, solid_capstyle='round', zorder=2)
    line, = ax.plot([], [], 'r+')
    line2, = ax.plot([],[],'g+')
    xdata, ydata = [], []
    xdata2, ydata2 = [], []
    matrices_traslacion = list(puntosmaximost)
    num_trasl = int(len(matrices_traslacion))

    for j in range(num_trasl):
        matrizavance = matrices_traslacion[j]
        ventosas = ventosasmaximast[j]
        xdata, ydata = [], []
        xdata2, ydata2 = [], []

        for i in range(num_coor):
```

```
        new_xdata = matrizavance[i][0]
        new_ydata = matrizavance[i][1]

        if ventosas[i]== 0:
            xdata.append(new_xdata)
            ydata.append(new_ydata)
        else:
            xdata2.append(new_xdata)
            ydata2.append(new_ydata)

    line.set_data(xdata, ydata)
    line2.set_data(xdata2, ydata2)
    plt.pause(0.5)
plt.show()
```

En la primera parte del código se encuentra la traslación. La matriz coge valores moviéndose a lo largo y ancho de unos valores determinados. Una vez se han obtenido todas las matrices posibles, se emplea la función “bigger” para descartar aquellas matrices que no tienen el número más alto de puntos activados.

Las matrices que sí tienen activados la cantidad de puntos más alta son empleadas en la segunda parte.

```
print("Segunda parte:")     #rotacion
#Se mira el movimiento de rotación de la matriz por la figura
#de las matrices máximas

guardarpuntos = []
posiciones = []
valores = []
```

```
puntorotacion =[]

for l in range(val_maxt):
    tz = 0

    puntorotacion = puntosmaximost[l]

    #Distancia de (0,0) al nuevo centro de la matriz
    centrox = puntorotacion[0][0]-puntoinicial[0][0]
    centroy = puntorotacion[0][1]-puntoinicial[0][1]

    while tz<360:
        sum = 0
        puntomodificado = list(puntoinicial)
        posiciones.append([centrox, centroy, tz])

        #Se modifican los puntos
        for k in range(len(puntomodificado)):
            puntomodificado[k] = move(puntomodificado[k], centrox, centroy,tz)

        guardarpuntos.extend(puntomodificado)
        tz = tz + alpha

        #Comprobación si el punto está dentro o fuera de la figura
        for r in range(len(puntomodificado)):
```

```
            p = (puntomodificado[r][0], puntomodificado[r][1])

            if (is_inside_polygon(lista, p)):
                ventosasr.append(puntomodificado[r][3])
                sum = sum + 1
            else:
                ventosasr.append(0)

        valores.append(sum)

#Representación de las matrices obtenidas
fig, ax = plt.subplots()
ax.set_xlim(-400, 700)
ax.set_ylim(-200, 500)
plt.plot(x, y, color='y', alpha=0.9, linewidth=2, solid_capstyle='round', zorder=2)
line, = ax.plot([], [], 'r+')
xdata, ydata = [], []
matrices_rotacion = list(guardarpuntos)
num_rot = int(len(matrices_rotacion) / num_coor)

for j in range(num_rot):
    avance = j * num_coor
    xdata, ydata = [], []

    for i in range(num_coor):
        k = i + avance
```

```
            new_xdata = matrices_rotacion[k][0]
            new_ydata = matrices_rotacion[k][1]
            xdata.append(new_xdata)
            ydata.append(new_ydata)

        line.set_data(xdata, ydata)
        plt.pause(0.5)
    plt.show()

    #Obtener las matrices con mayor cantidad de puntos dentro de la figura
            puntosmaximosr, ventosasmaximasr, val_maxr, matriztrabajor =
bigger(guardarpuntos, ventosasr, valores, posiciones)

    print(puntosmaximosr)
    print(ventosasmaximasr)
    print(val_maxr)
    print(matriztrabajor)

    #Representación de las matrices obtenidas
    fig, ax = plt.subplots()
    ax.set_xlim(-400, 700)
    ax.set_ylim(-200, 500)
    plt.plot(x, y, color='y', alpha=0.9, linewidth=2, solid_capstyle='round', zorder=2)
    line, = ax.plot([], [], 'r+')
    line2, = ax.plot([],[],'g+')
    xdata, ydata = [], []
```

```
xdata2, ydata2 = [], []
matrices_rotacion = list(puntosmaximosr)
num_rot = int(len(matrices_rotacion))

for j in range(num_rot):
    matrizavance = matrices_rotacion[j]
    ventosas = ventosasmaximasr[j]
    xdata, ydata = [], []
    xdata2, ydata2 = [], []

    for i in range(num_coor):
        new_xdata = matrizavance[i][0]
        new_ydata = matrizavance[i][1]

        if ventosas[i]== 0:
            xdata.append(new_xdata)
            ydata.append(new_ydata)
        else:
            xdata2.append(new_xdata)
            ydata2.append(new_ydata)

    line.set_data(xdata, ydata)
    line2.set_data(xdata2, ydata2)
    plt.pause(0.5)
plt.show()
```

En la segunda parte del código se produce la rotación. Esta consiste en utilizar aquellas matrices de la primera parte, con el máximo número de puntos activados, rotándolas sobre su eje a la búsqueda de más posiciones válidas de la matriz.

Una vez realizas todas las rotaciones de las distintas posiciones se utiliza la función "bigger", que como en la primera parte, se guardan aquellas matrices con el valor máximo de puntos activados.

Estas matrices son las que se pueden emplear para coger la pieza deseada, pudiendo tener más de una solución el código.

8.3. ANEXO III. SOFTWARE MODIFICADO

El código ha sido modificado para tener una mayor proximidad al objeto buscado. Para ello se han eliminado una serie de funciones y en vez de puntos, se han empleado circunferencias de un radio determinado.

Estas modificaciones ayudan a que el número máximo de ventosas seleccionadas sea más preciso, activándose únicamente las ventosas que su completa circunferencia esté dentro del polígono.

```
#Obtener las matrices con mayor número de ventosas dentro de la figura


INT_MAX = 10000

num_coor = 16   #puntos del gripper


#Coordenadas de avance de matriz de traslación y rotación

stepx = 40

stepy = 40

alpha = 20

radius = 10


#Obtener datos del archivo DXF

def print_line(e,lista):
```

```
    #Añade el punto inicial a la lista
    lista.append((float(e.dxf.start[0]), float(e.dxf.start[1])))    #añade el punto inicial a la lista

#Realización de traslación y rotación
def move(puntos,t1,t2,t3):

    lista = []
    resultado = []
    t3r = np.radians(t3)

    #Se hace una lista del nuevo punto
    #Se introducen los puntos como una matriz
    for i in range(len(puntos)-1):
        lista.append(puntos[i])
    #Se convierte en matriz y se transpone para poder trabajar
    A = np.array(lista)
    AT = np.transpose(A)

    #Matriz de traslación y rotación
    T1 = np.array([[1, 0, t1],[0, 1, t2],[0, 0, 1]])
    T2 = np.array([[math.cos(t3r), -math.sin(t3r), 0],[math.sin(t3r), math.cos(t3r), 0], [0, 0, 1]])

    #Multiplicación de ambas matrices y la matriz del punto
```

```
    Val1 = np.dot(T1, T2)
    Val = np.dot(Val1, AT)
    resultado = list(Val)
    resultado.append(puntos[3])

    return resultado

#Encontrar las matrices máximas
def bigger(matriz, matrizventosas, cantidad, puntos):
    posiciones = []
    ventosas = []
    matrizt = []
    count = 0
    j = 0
    suma = 0 #valores con el número mayor

    #Busqueda del valor más alto en la lista
    maximo = np.max(cantidad)

    for z in range(len(cantidad)):
        count= count + num_coor

        if cantidad[z]==maximo:
            j = count - num_coor
            suma = suma + 1
            matriztrabajop = []
```

```
        matriztrabajov = []

        while j<count:
            matriztrabajop.append(matriz[j])
            matriztrabajov.append(matrizventosas[j])
            j=j+1

        posiciones.append(matriztrabajop)
        ventosas.append(matriztrabajov)
        matrizt.append(puntos[z])

    #Devuelve todas las matrices máximas, las ventosas,
    #la cantidad y las posiciones que se han usado
    return posiciones, ventosas, suma, matrizt

# Driver code
if __name__ == '__main__':

    import matplotlib.pyplot as plt
    from shapely.geometry import Polygon, Point
    import ezdxf
    import math
    import numpy as np

    #4X4
    puntoinicial = ([240,85,1,1],
```

```
            [240,0,1,2],
            [240,-85,1,3],
            [160,135,1,4],
            [160,-135,1,5],
            [80,50,1,6],
            [80,-50,1,7],
            [40,135,1,8],
            [0,-135,1,9],
            [-40,135,1,10],
            [-80,85,1,11],
            [-80,-85,1,12],
            [-160,135,1,13],
            [-160,-135,1,14],
            [-240,85,1,15],
            [-240,-85,1,16])

    lista = []  #guarda los puntos del polígono
    puntoiniciotrabajo = [] #guarda el punto inicial de traslación
    puntomodificado =[] #matriz que se modifica constantemente
    guardarpuntos = []  #guarda todas las matrices
    puntosmaximost = [] #guarda todas las matrices con los máximos puntos dentro de la figura traslación
    ventosasmaximast = []
    puntosmaximosr = [] #guarda todas las matrices con los máximos puntos dentro de la figura rotación
    ventosasmaximasr = []
```

```
puntorotacion = []  #punto inicial de la rotación
ventosast =[]
ventosasr =[]
posiciones = []
matriztrabajot = [] #almacena tx,ty,tz en matriz en traslación máximos
matriztrabajor = [] #almacena tx,ty,tz en matriz de rotación máximos
valores = []    #guarda los puntos dentro de la figura de cada matriz

centro =()

posicion = 0
max_sum = 0
val_maxt = 0 #guarda cuantas matrices tienen el valor máximo en traslación
val_maxr = 0 #guarda cuantas matrices tienen el valor máximo en rotación

#Poner el DXF que se va a usar
doc= ezdxf.readfile("corte_linea.dxf")

msp = doc.modelspace()
for e in msp:
    if e.dxftype()== 'LINE':
        print_line(e, lista)



#comprobación de que todos los puntos se encuentren en la lista
print(lista)
```

```
#Generar un polígono para poder visualizarlo
pol = Polygon(lista)
x, y = pol.exterior.xy

fig, ax = plt.subplots()
ax.set_xlim(-400, 700)
ax.set_ylim(-200, 500)
#centro del polígono
centro = (pol.centroid.x, pol.centroid.y)
print(centro)
plt.scatter(centro[0], centro[1], marker='*', linewidths=1, c="g", s=90)

#Mover el gripper al centro de la pieza
puntomodificado = list(puntoinicial)
for k in range(len(puntomodificado)):
    puntomodificado[k] = move(puntomodificado[k], centro[0], centro[1], 0)

matrizcentro = puntomodificado  #posición de la matriz en el centro de la pieza

#Modificado para círculos
for i in range(len(puntomodificado)):

    center = Point(puntomodificado[i][0], puntomodificado[i][1])
    centros= center.xy
    circulo= plt.Circle(centros, radius, fill= True, color='b')
    ax.add_patch(circulo)
```

```
plt.plot(x, y, color='y', alpha=0.9, linewidth=2, solid_capstyle='round', zorder=2)
plt.show()
#Mover a posición de trabajo inicial para empezar la traslación
for k in range(len(puntomodificado)):
    puntoiniciotrabajo.append(move(puntomodificado[k], -240, -140, 0))

print("Primera parte:")     #traslación
#Se desplaza la matriz para trabajar la traslación en su posición inicial de trabajo
#Se mira el movimiento de traslación de la matriz por la figura

tz = 0
ty = 0
while ty<=290:

    tx= 0
    while tx<=480:

        posiciones.append([tx,ty,tz])   #valores matriz traslación
        sum = 0
        puntomodificado = list(puntoiniciotrabajo)
        #Se modifican los puntos
        for k in range(len(puntomodificado)):
            puntomodificado[k] = move(puntomodificado[k],tx,ty,tz)

        guardarpuntos.extend(puntomodificado)
```

```
        #Comprobación si el punto está dentro o fuera de la figura
        for i in range(len(puntomodificado)):
            center = Point(puntomodificado[i][0], puntomodificado[i][1])
            circle = center.buffer(radius)

            if (pol.contains(circle)):

                ventosast.append(puntomodificado[i][3])
                sum = sum + 1
            else:
                ventosast.append(0)

        valores.append(sum)
        tx = tx + stepx

    ty = ty + stepy

#Representación de las matrices obtenidas
fig, ax = plt.subplots()
ax.set_xlim(-400, 700)
ax.set_ylim(-200, 500)
plt.plot(x, y, color='y', alpha=0.9, linewidth=2, solid_capstyle='round', zorder=2)
matrices_traslacion = list(guardarpuntos)
num_trasl = int(len(matrices_traslacion) / num_coor)
```

```
    for j in range(num_trasl):
        avance = j * num_coor

        for circulo in ax.patches:
            circulo.remove()

        for i in range(num_coor):
            k = i + avance
            center = Point(matrices_traslacion[k][0], matrices_traslacion[k][1])
            centros = center.xy
            circulo = plt.Circle(centros, radius, fill=True, color='b')
            ax.add_patch(circulo)

        plt.pause(0.5)
    plt.show()


    #Obtener las matrices con mayor cantidad de puntos dentro de la figura
            puntosmaximost, ventosasmaximast, val_maxt, matriztrabajot =
bigger(guardarpuntos,ventosast, valores, posiciones)
    print(puntosmaximost)
    print(ventosasmaximast)
    print(val_maxt)
    print(matriztrabajot)

    #Representación de las matrices obtenidas
```

```
fig, ax = plt.subplots()
ax.set_xlim(-400, 700)
ax.set_ylim(-200, 500)
plt.plot(x, y, color='y', alpha=0.9, linewidth=2, solid_capstyle='round', zorder=2)
matrices_traslacion = list(puntosmaximost)
num_trasl = int(len(matrices_traslacion))

for j in range(num_trasl):
    matrizavance = matrices_traslacion[j]
    ventosas = ventosasmaximast[j]

    for circuloout in ax.patches:
        circuloout.remove()

    for circuloin in ax.patches:
        circuloin.remove()

    for i in range(num_coor):
        center = Point(matrizavance[i][0], matrizavance[i][1])
        centros = center.xy

        if ventosas[i] == 0:
            circuloout = plt.Circle(centros, radius, fill=True, color='b')
            ax.add_patch(circuloout)
        else:
            circuloin = plt.Circle(centros, radius, fill=True, color='g')
```

```
            ax.add_patch(circuloin)


    plt.pause(0.5)
plt.show()


print("Segunda parte:")  #rotacion
#Se mira el movimiento de rotación de la matriz por la figura
#de las matrices máximas


guardarpuntos = []
posiciones = []
valores = []
puntorotacion =[]


for l in range(val_maxt):
    tz = 0


    puntorotacion = puntosmaximost[l]


    #Distancia de (0,0) al nuevo centro de la matriz
    centrox = puntorotacion[0][0]-puntoinicial[0][0]
    centroy = puntorotacion[0][1]-puntoinicial[0][1]


    while tz<360:
        sum = 0
        puntomodificado = list(puntoinicial)
```

```
        posiciones.append([centrox, centroy, tz])

        #Se modifican los puntos
        for k in range(len(puntomodificado)):
            puntomodificado[k] = move(puntomodificado[k], centrox, centroy,tz)

        guardarpuntos.extend(puntomodificado)
        tz = tz + alpha

        #Comprobación si el punto está dentro o fuera de la figura
        for r in range(len(puntomodificado)):
            center = Point(puntomodificado[r][0], puntomodificado[r][1])
            circle = center.buffer(radius)

            if (pol.contains(circle)):
                ventosasr.append(puntomodificado[r][3])
                sum = sum + 1
            else:
                ventosasr.append(0)

        valores.append(sum)

#Representación de las matrices obtenidas
fig, ax = plt.subplots()
ax.set_xlim(-400, 700)
ax.set_ylim(-200, 500)
```

```
    plt.plot(x, y, color='y', alpha=0.9, linewidth=2, solid_capstyle='round', zorder=2)
    matrices_rotacion = list(guardarpuntos)
    num_rot = int(len(matrices_rotacion) / num_coor)

    for j in range(num_rot):
        avance = j * num_coor

        for circulo in ax.patches:
            circulo.remove()

        for i in range(num_coor):
            k = i + avance
            center = Point(matrices_rotacion[k][0], matrices_rotacion[k][1])
            centros = center.xy
            circulo = plt.Circle(centros, radius, fill=True, color='b')
            ax.add_patch(circulo)

        plt.pause(0.5)
    plt.show()

    #Obtener las matrices con mayor cantidad de puntos dentro de la figura
        puntosmaximosr, ventosasmaximasr, val_maxr, matriztrabajor =
bigger(guardarpuntos, ventosasr, valores, posiciones)

    print(puntosmaximosr)
    print(ventosasmaximasr)
```

```
print(val_maxr)
print(matriztrabajor)

#Representación de las matrices obtenidas
fig, ax = plt.subplots()
ax.set_xlim(-400, 700)
ax.set_ylim(-200, 500)
plt.plot(x, y, color='y', alpha=0.9, linewidth=2, solid_capstyle='round', zorder=2)
matrices_rotacion = list(puntosmaximosr)
num_rot = int(len(matrices_rotacion))

for j in range(num_rot):
    matrizavance = matrices_rotacion[j]
    ventosas = ventosasmaximasr[j]

    for circuloout in ax.patches:
        circuloout.remove()

    for circuloin in ax.patches:
        circuloin.remove()

    for i in range(num_coor):
        center = Point(matrizavance[i][0], matrizavance[i][1])
        centros = center.xy

        if ventosas[i] == 0:
```

```
            circuloout = plt.Circle(centros, radius, fill=True, color='b')
            ax.add_patch(circuloout)
        else:
            circuloin = plt.Circle(centros, radius, fill=True, color='g')
            ax.add_patch(circuloin)

    plt.pause(0.5)
plt.show()
```

9. BIBLIOGRAFÍA

[1] Acaccia, G. M., Bruzzone, L. E., Molfino, R. M., & Zoppi, M. (2004, julio). Modular SMA-based matrix gripper for grasping and handling of limp materials. En Intelligent Manipulation and Grasping IMG04. https://www.researchgate.net/profile/Rezia-Molfino/publication/228796103_Modular_SMA-based_matrix_gripper_for_grasping_and_handling_of_limp_materials/links/0f317535e68ef6f354000000/Modular-SMA-based-matrix-gripper-for-grasping-and-handling-of-limp-materials.pdf

[2] Andrino, R. G. (2021, 12 abril). Python - Extraer información de un DXF. https://es.linkedin.com/pulse/python-extraer-informaci%C3%B3n-de-un-dxf-ram%C3%B3n-gonz%C3%A1lez-andrino

[3] Brazo Robotizado | Universal Robots. (s. f.). https://www.universal-robots.com/es/productos/robot-ur5/

[4] Exaloop. (s. f.). GitHub - exaloop/codon: A high-performance, zero-overhead, extensible Python compiler using LLVM. GitHub. https://github.com/exaloop/codon

[5] Fernando O., & Tomás S. J. (Directores). (2020, 1 noviembre). Schmalz diseña garra de vacío SFG para el sector alimentario. Revista de Robots, 1, 12.

[6] GeeksforGeeks. (2023, 16 marzo). How to check if a given point lies inside or outside a polygon. https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/

[7] Introduction — ezdxf 1.0.3 documentation. (s. f.).

https://ezdxf.readthedocs.io/en/stable/introduction.html

[8] JSY1000, Plug-in | SMC España. (s. f.). https://www.smc.eu/es-es/productos/jsy1000-plug-in%7E163780%7Enav

[9] math — Mathematical functions. (s. f.). Python documentation. https://docs.python.org/3/library/math.html

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

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

[12] QUICK CHANGER | El cambiador de herramientas más rápido del mundo. (s. f.-b). OnRobot. https://onrobot.com/es/productos/quick-changer

[13] R. (2022, 4 septiembre). Gripper con pinzas de vacío robóticas. REVISTA DE ROBOTS. https://revistaderobots.com/sistemas-de-agarre/garras-y-gripper-con-pinzas-de-vacio-roboticas/

[14] Schmalz. (s. f.). Needle Grippers. SCHMALZ. https://www.schmalz.com/en/vacuum-technology-for-automation/vacuum-components/special-grippers/needle-grippers/

[15] Schmalz. (s. f.). Ventosas de vacío. SCHMALZ. https://www.schmalz.com/es-es/saber-de-vacio/el-sistema-de-vacio-y-sus-componentes/ventosas-de-vacio/

[16] The Shapely User Manual — Shapely 2.0.1 documentation. (s. f.). https://shapely.readthedocs.io/en/stable/manual.html

[17] UNIVERSAL ROBOTS UR5/UR5E. (s. f.). MYBOTSHOP. https://www.mybotshop.de/Universal-Robots-UR5-UR5e_3

[18] VMECA Vacuum Tech. (2014, 2 julio). VMECA Needle Gripper for handling porous Insoles for footwear [Vídeo]. YouTube. https://www.youtube.com/watch?v=tR8j9jNzB2Q