UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño y desarrollo de un sistema robotizado capaz de resolver un cubo de Rubik mediante un robot colaborativo y visión artificial

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Górriz Aliaga, Abel

Tutor/a: Ivorra Martínez, Eugenio

CURSO ACADÉMICO: 2022/2023

Resumen

El proyecto consiste en la resolución de un cubo de Rubik 3x3 utilizando el robot colaborativo UR3e, y el IDLE de Python y OpenCV. El objetivo es, mediante un sistema de visión artificial detectar el estado actual del cubo de Rubik y, a continuación, enviar instrucciones al robot para resolverlo.

Para lograr esto, el sistema utilizará una cámara conectada a un ordenador, mediante OpenCV procesará el estado de las 6 caras del cubo de Rubik para detectar su estado actual.

Una vez que se haya determinado el estado del cubo, se utilizará un algoritmo para calcular la solución óptima para resolverlo. Estas instrucciones serán enviadas al robot colaborativo, que, con la ayuda de un soporte para apoyar y girar el cubo, resolverá el problema.

El proyecto fusiona la visión artificial y la robótica para desarrollar un sistema autónomo que resuelve un difícil puzle como es el cubo de Rubik. Esta tecnológica ofrece un amplio espectro de aplicaciones, abarcando campos como la automatización en la producción, la robótica de servicios y la logística, entre otros.

Agradecimientos

Quiero aprovechar este espacio para agradecer de corazón a todas las personas que han sido parte de mi viaje universitario. Primero, a mi familia, por su amor y apoyo incondicional que me ha impulsado en cada paso, en especial a mi hermano Omar que me enseñó a resolver el cubo de Rubik que sirvió de inspiración a la hora de realizar este proyecto. También a mis profesores y mentores, que me han brindado su sabiduría y paciencia para crecer académicamente, en especial a mi tutor Eugenio Ivorra que me ha ayudado al desarrollo del trabajo y por despertar mi interés en la rama de visión artificial. Agradezco a mis compañeros de clase por los momentos compartidos y el apoyo mutuo. Y por último, pero no menos importante, agradezco a mi novia Patri por aguantarme y ayudarme en todo lo posible en estos cuatro años de grado.

Índice de Documentos

UNIVERSITAT POLITÉCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño y desarrollo de un sistema robotizado capaz de resolver un cubo de Rubik mediante un robot colaborativo y visión artificial

DOCUMENTO Nº1: Memoria

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Górriz Aliaga, Abel

Tutor/a: Ivorra Martínez, Eugenio

CURSO ACADÉMICO:2022-2023

Índice de contenidos

1. OBJETO

El presente documento tiene como objetivo el diseño y desarrollo de un programa capaz de resolver un cubo de Rubik 3x3 mediante la utilización de un robot industrial colaborativo, el UR3e de Universal Robots, para realizar los movimientos y con la utilización de Python y OpenCV para el reconocimiento del estado del cubo y los movimientos para resolverlo.

2. ANTECEDENTES

Para crear un contexto actual del proyecto, conviene conocer aspectos básicos como, el origen del cubo de Rubik y su estado actual, la utilidad de los robots colaborativos y el concepto de visión artificial

2.1. CUBO DE RUBIK

El cubo de Rubik fue creado por escultor, diseñador y profesor de arquitectura húngaro Ernő Rubik. Patentado en 1974 y comercializado en 1977 rápidamente se convirtió en un éxito, consiguiendo en 1980 con el premio alemán al Mejor Juego del Año en la categoría de mejor rompecabezas.

Originariamente Ernő Rubik estaba investigando un problema estructural sobre cómo se podían mover los bloques de forma independiente sin que se desmontara el cubo, la idea del puzle surgió cuando al girar los cubos en una de sus pruebas vio cómo se reorganizaban cada vez de forma distinta.

El que se considera el juguete más vendido del mundo, es a la vez un reto matemático. En el año 2010 un grupo de investigadores logró demostrar que no existía ninguna de las más de 43 trillones posiciones que requiriera más de 20 movimientos, quedando establecido este número de movimientos como el número de Dios.

El algoritmo Kociemba es un método para resolver el cubo de Rubik, utilizando la teoría de grupos y la búsqueda en árboles consigue encontrar la solución más corta posible en el menor tiempo posible. Este algoritmo fue desarrollado por Herbert Kociemba en 1992 y es uno de los métodos más populares y eficientes para resolver el cubo en la actualidad.

El algoritmo Kociemba se basa en la teoría de grupos, esta es una rama de las matemáticas que estudia la estructura de los conjuntos y las operaciones que se pueden realizar sobre ellos. En el caso del Cubo de Rubik, los grupos se refieren a los movimientos posibles que se pueden hacer para cambiar la posición de los cubos sin desmontarlo.

Fig 1: Cubo de Rubik

2.2. VISIÓN ARTIFICIAL

La visión artificial engloba tanto aplicaciones industriales y no industriales en donde gracias a la combinación de hardware y software se obtienen datos e información útil basándose en la captura y el procesamiento de imágenes para reconocer y analizar características específicas de dichas imágenes como formas, colores, texturas y contornos.

La visión artificial se usa en diversas áreas, como la robótica, la automatización industrial, la seguridad, la medicina, entre otros. Algunas de las técnicas utilizadas en la visión artificial incluyen el reconocimiento de patrones, el aprendizaje automático, la segmentación de imágenes, la detección de bordes, la extracción de características y la fusión de imágenes.

Fig 2: Logo OpenCV

OpenCV es una biblioteca de código abierto para el procesamiento de imágenes y visión artificial. Es una de las bibliotecas más populares y utilizadas en la industria para desarrollar aplicaciones y sistemas que requieren análisis de imágenes en tiempo real.

OpenCV proporciona una amplia variedad de funciones y algoritmos para el procesamiento de imágenes, la biblioteca es compatible con varios lenguajes de programación, lo que hace accesible para desarrolladores con diferentes habilidades y conocimientos en programación.

2.3. ROBOT COLABORATIVO

Fig 3: Robot colaborativo UR5e

Los robots colaborativos o cobots tienen la finalidad de automatizar procesos industriales con el propósito de optimizar dichos procesos. Estos robots son un tipo de robot industrial, articulado de pequeño tamaño muy novedosos debido a su factor colaborativo, por el cual podemos encontrarlos trabajando en con junto con el personal humano sin que ello ponga en riesgo su seguridad.

Estos robots son ideales para realizar tareas repetitivas y evitar lesiones en los trabajadores, así como para trabajar en industrias que requieran altas temperaturas, materiales tóxicos, entre otros. La seguridad de este tipo de robots es gracias a los sensores integrados que les permiten responder a estímulos externos y evitar accidentes. Además, estos robots pueden trabajar como una extensión de otros equipos o asistir a los trabajadores en sus tareas diarias.

2.4. VISIÓN ARTIFICIAL Y ROBÓTICA

En la actualidad gracias a la Industria 4.0 se está optando por la digitalización de las líneas de producción con elementos como la realidad virtual, inteligencia artificial, internet de las cosas y Big Data.

Las innovaciones tecnológicas han permitido que gracias a la unión de visión artificial y robots colaborativos hoy en día estos puedan realizar tareas impensables hasta ahora revolucionando el trabajo en sectores industriales.

Las ventajas de la visión artificial en robots incluyen garantizar la trazabilidad de los procesos de producción, un guiado automático de los brazos robóticos, rapidez y constancia. Las aplicaciones de la visión artificial incluyen pick & place, empaquetado y paletizado, control de calidad y montaje, y son ideales para sectores como el de automoción, alimentario, electrónica y tecnología, entre otros.

3. ESTUDIO DE NECESIDADES

Con relación al estudio de necesidades, se pretende resolver un cubo de Rubik tradicional con unas dimensiones de $56mm^3$ y un peso de $88g$ del fabricante Rubik's.

En cuanto a la programación se utilizará el IDLE de Python que conectado a la cámara Intel RealSense D415, del fabricante Intel, se desarrollará un programa capaz de captar la posición de las 6 caras del cubo de Rubik y mediante el optimizado algoritmo Kociemba, calcular los movimientos necesarios para la correcta resolución del puzle.

Para la manipulación del objeto será necesario programar el robot UR3e de Universal Robots. El sistema es un robot pequeño y compacto, capaz de manejar una carga de 3 kg, suficiente a la hora de interaccionar con el cubo, además de una fácil programación y capacidad de comunicación con el programa informático a la hora de recibir los diferentes movimientos a realizar.

Se utilizará la pinza paralela DHPS-20-A del fabricante FESTO con un actuador neumático donde será capaz de abrir y cerrar esta en un tiempo mínimo. Debido a que la medida de la pinza no es la adecuada para el cubo, se diseñara mediante impresión 3D dos pinzas con las medidas necesarias para acoplar a los dedos de la pinza que permita la manipulación del objeto sin ningún problema.

Para permitir al robot que pueda aplicar los movimientos al cubo se diseñara un soporte mediante impresión 3D. Este soporte debe de ser capaz de mantener el cubo en una determinada posición y a su vez dar libertad al robot de girar una de sus 6 caras y de la extracción de el mismo desde diferentes puntos.

4. PLANTEAMIENTO DE SOLUCIONES ALTERNATIVAS Y JUSTIFICACIÓN DE LA SOLUCIÓN ADOPTADA

La primera idea para la resolución del problema fue la construcción de un mecanismo que fuera capaz de insertar el cubo dentro de él, este mediante diferentes motores activados fuera capaz de realizar los giros en las 6 capas.

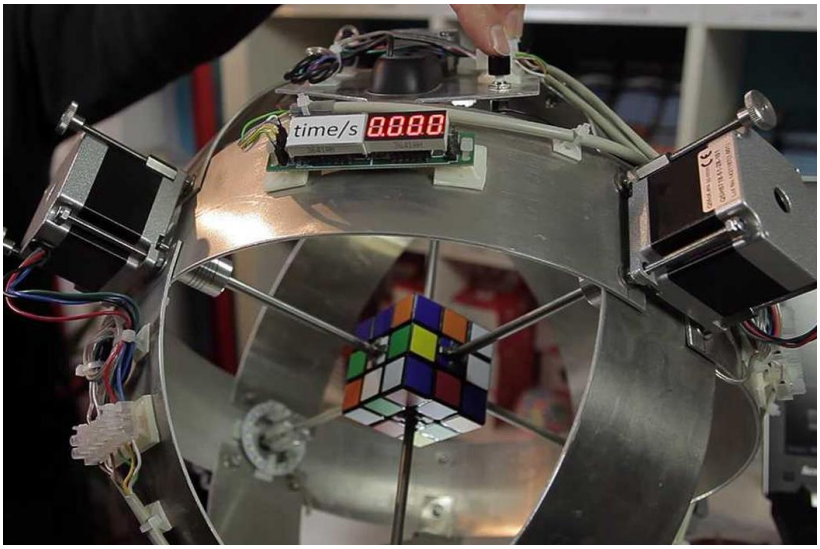


Fig 4: Maquina capaz de mover 6 capas

A pesar de que quizá sea la opción más rápida, la construcción del habitáculo no era sencilla teniendo en cuenta la necesidad de que una cámara fuera capaz de ver todo el cubo.

Una vez rechazada la opción de construir una maquina apareció la idea de utilizar robots colaborativos, gracias a que estos pueden mover el cubo a la perfección para que la cámara haga la identificación sin problemas.

Al principio se pensó en utilizar dos robots a modo de brazos de una persona y que el cubo vaya pasando de uno a otro cuando fuera necesario para realizar los movimientos.

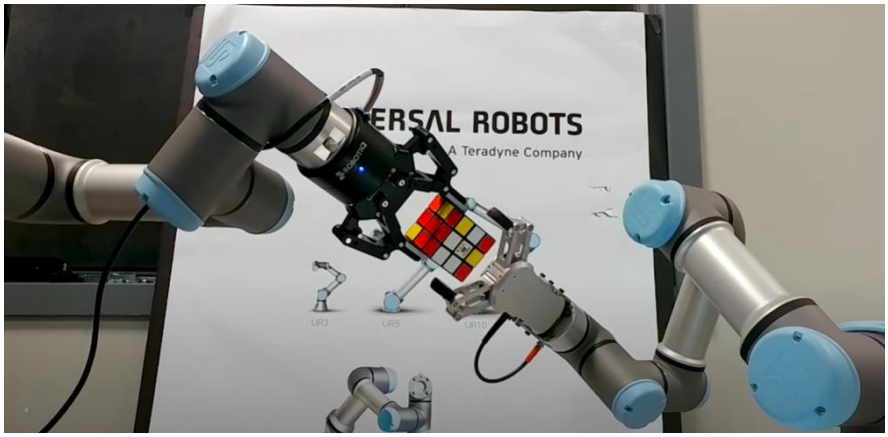


Fig 5: Dos robots resolviendo el cubo

Los problemas de realizar esta opción es la necesidad de dos robots y la dificultad de movimientos y comunicación entre ellos, además debido a los pequeños errores que puede realizar el robot, el cubo después de tantos movimientos es fácil que caiga y no se pueda finalizar la resolución.

Finalmente se ha optado por la utilización de un solo robot UR3e y el diseño de un soporte, que sirva tanto para girar el cubo como para reposarlo. De esta forma se puede mostrar todas las caras del cubo a la cámara sin problema no como en la primera solución. Además, los pequeños errores que se podían producirse al pasar el cubo de un robot a otro, ahora no se producen ya que al dejarlo en el soporte, estos errores no se acumulan.

5. DESCRIPCIÓN DETALLADA DE LA SOLUCIÓN ADOPTADA

La solución adoptada final se basa en la utilización de un robot industrial colaborativo URe3, que, gracias al diseño de un soporte que permita girar la capa inferior del cubo, dejar este reposando y poder cogerlo desde diferentes puntos, se permite la resolución del puzle. Se va a dividir el sistema en tres partes diferenciadas para, una vez comprendidas todas las partes, explicar la relación y secuenciación de estas a la hora de la resolución.

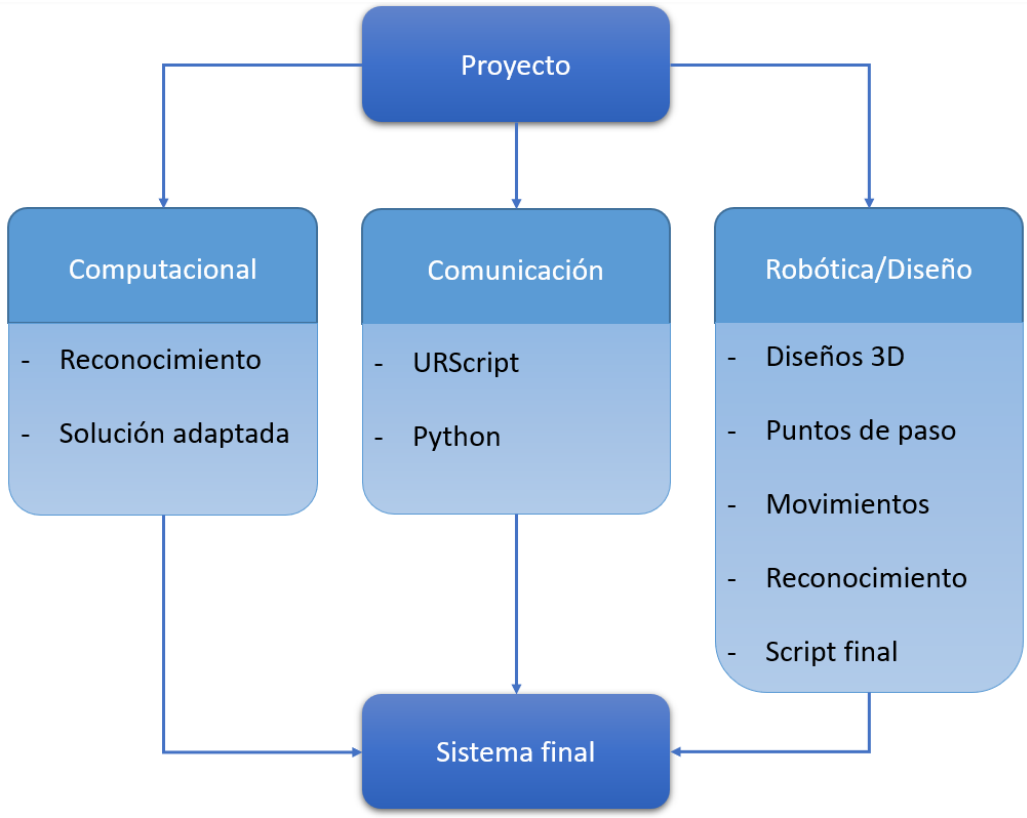


Fig 6: Organigrama del proyecto

5.1. PARTE COMPUTACIONAL

En la parte informática se usó como base un programa obtenido desde GitHub, donde mediante OpenCV se identifica el estado de las caras del cubo y gracias al algoritmo Kociemba, implementado en Python se proporciona la solución más corta posible en un instante de tiempo.

En muy pocas ocasiones, con un mayor tiempo de búsqueda de la solución, el algoritmo sería capaz de obtener una secuencia más corta. En la solución adoptada no se ha tenido en cuenta esta búsqueda de una secuencia de movimientos mejor debido a que no se sabe con certeza cuanto tiempo puede tardar y lo más común es que no obtenga ningún resultado mejor.

5.1.1. Reconocimiento

Para que el reconocimiento sea correcto, se ha de colocar cada cara del en una orientación predeterminada además de tener una luz correcta a la hora de la toma de datos.

5.1.1.1. Iluminación

Fig 7: Diferencia debido a la iluminación

Como se puede apreciar en la *Fig7* hay una diferencia notoria en cuanto a los reflejos. La identificación del cubo de la izquierda no se ha aplicado ningún tipo de iluminación además de la existente en la sala, en este caso se generan ciertos reflejos que dificultan la identificación de los colores. En el segundo cubo no se aprecia cuanto apenas reflejo alguno, para lograr este efecto se coloca un foco de luz con bastante inclinación en el lateral del cubo. Gracias a esta iluminación se eliminan los reflejos producidos por otras luces y esta no genera ninguno ya que los stickers, pegatinas de colores, son lisos y gracias al grado de inclinación del foco no consiguen reflejarse en la lente de la cámara.

5.1.1.2. Orientación

Con los colores previamente calibrados y una iluminación correcta se puede pasar a la identificación del cubo por completo. Para orientar correctamente el cubo hay que pensar en el cómo un plano.

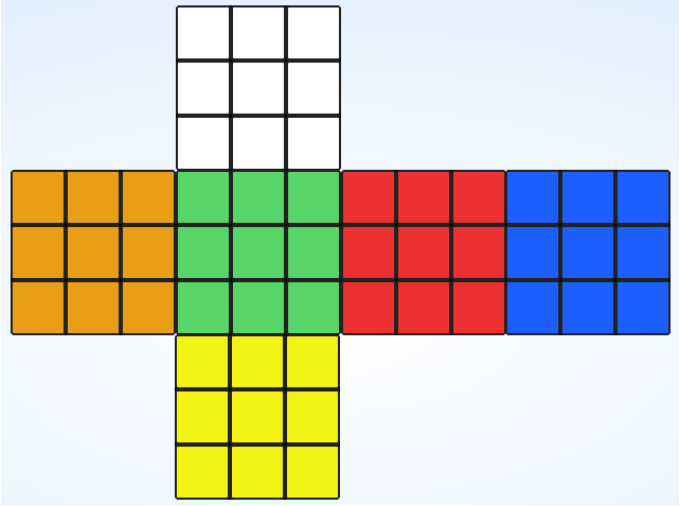


Fig 8: Cubo de Rubik en 2D

La orientación estándar es la que aparece en la *Fig8* y tal y como se vayan mostrando las caras el programa las irá reconociendo, dependiendo del color de la pieza central, debido a que esta no cambia de posición. No será necesario llevar un orden en concreto.

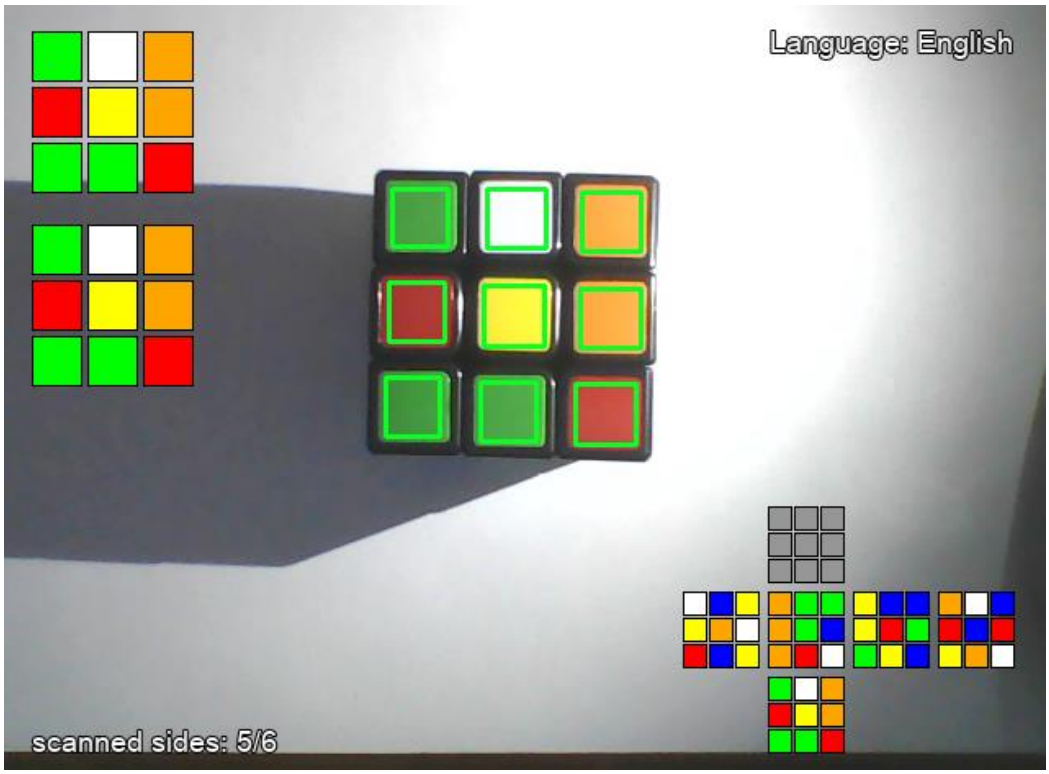


Fig 9: Interfaz de monitorización

En este caso, *Fig9*, se han escaneado 5 de las 6 caras. Una vez se obtenga los datos de la última cara de cubo, el programa utilizara dichos datos y mediante el algoritmo Kociemba proporcionara una solución al cubo mostrado.

```
Moves: 21
Solution: R U B R F D B' R L2 D2 B U2 R2 F2 R2 U' L2 B2 U L2 D'
```

Fig 10: Solución del cubo

Mediante la solución mostrada en la *Fig10* se resolvería el cubo que se ha identificado.

5.1.2. Solución adaptada

El algoritmo Kociemba proporciona unos movimientos a un cubo fijo donde cada cara del cubo tiene asignada una letra dependiendo de la posición en las manos de un humano.

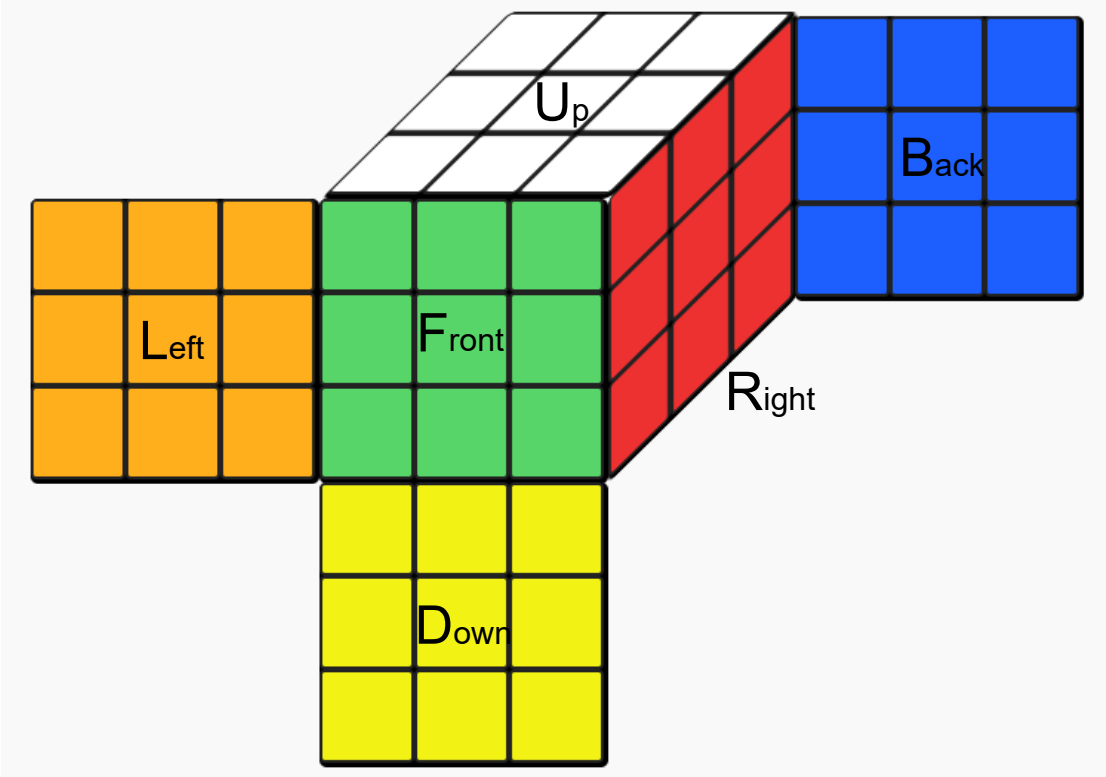


Fig 11: Notación básica del cubo de Rubik

Para la resolución del cubo dada una solución como la obtenida en la *Fig10* donde cada giro viene determinado por una letra que a su vez esta asignada a cada una de las caras de un cubo. Por ejemplo, como se muestra en la figura anterior, *Fig11*, un giro en U significa girar la cara Up, un giro en F significa girar la cara Front y así sucesivamente.

En el caso de que el giro no tenga nada después significa un giro de 90º horario, si tiene un 2 significa que debe realizar un giro de 180º, y finalmente si aparece una comilla significa que debe realizar un giro de 270º o -90º en sentido horario.

La idea de resolver el problema es que el robot gire las capas con ayuda del soporte, para hacer dicho giro el robot coge el cubo por un lateral, lo extrae del soporte y lo introduce de forma vertical para girar la cara inferior. De esta forma se ha cambiado la orientación del cubo.

Siguiendo con el ejemplo de la *Fig10* una vez realizado el primer giro de 90º horario (R) el cubo quedará con una orientación diferente a la inicial. De este modo la secuencia inicial dada por el algoritmo deja de ser válida si no se hace una un paso intermedio.

Ante esta problemática hay 2 maneras de resolverlo:

- Cada vez que se realiza un giro volver a colocar el cubo en su orientación inicial.
- Modificar la secuencia teniendo en cuenta la nueva orientación.

Se ha elegido la modificación de la secuencia con el fin de optimizar la resolución y disminuir el tiempo de esta.

Para realizar esta conversión se ha identificado el cubo como un vector de 6 números.

En el vector se han identificado un número con cada uno de los colores. Si utilizamos la orientación inicial vista en la *Fig.11* la relación entre movimiento, número del vector, parte del cubo y color sería la siguiente:

Movimiento	Vector	Parte del cubo	Color
U	1	Superior	
F	2	Frontal	
R	3	Derecha	
B	4	Detrás	
L	5	Izquierda	
D	6	Inferior	

Fig 12: Tabla de la orientación inicial

En este caso el cubo se encuentra en la posición inicial, donde la orientación de las caras coincide con la que se desea rotar. Por el movimiento U gira la cara superior, F gira la cara frontal, y así sucesivamente. Dicho vector de orientación llamado “cubo” sería el siguiente:

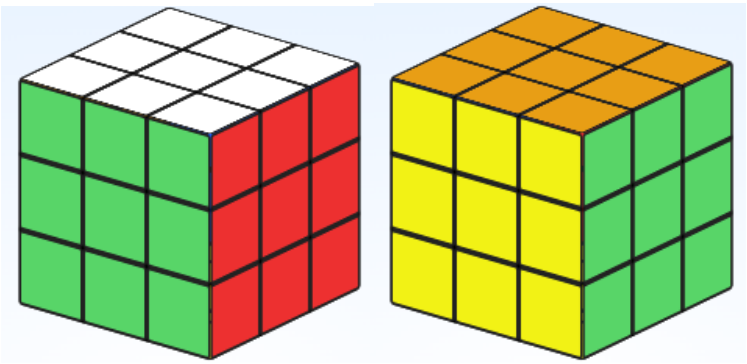
$$cubo = (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6)$$

Cuando se realiza un movimiento, la orientación del cubo cambia, por lo tanto, se ha de modificar el vector. Esto se realiza mediante la multiplicación 18 matrices que se han obtenido a partir del análisis de los movimientos del robot a la hora de la resolución. Estas matrices se corresponden con cada uno de todos los movimientos posibles teniendo en cuenta las 6 caras y los 3 distintos giros en cada una de ellas.

De esta forma al multiplicar una de estas matrices de transformación con el vector que contiene la orientación, se obtiene un nuevo vector:

$$(1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6) \cdot \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} = (5 \quad 6 \quad 2 \quad 1 \quad 4 \quad 3)$$

En el cálculo anterior se ha partido del cubo en su posición inicial y se ha realizado un movimiento R, dando como resultado una nueva orientación.



Orientación Inicial Nueva Orientación

Fig 13: Cambio de la orientación del cubo

Los nuevos valores del vector significan que ahora en la posición 1, la parte superior del cubo se corresponde con el número 5, el cual está asignado al color naranja. Y así sucesivamente con las demás caras.

Movimiento	Vector	Parte del cubo	Color	Movimiento esperado
U(1)	5	Superior		B
F(2)	6	Frontal		R
R(3)	2	Derecha		D
B(4)	1	Detrás		L
L(5)	4	Izquierda		U
D(6)	3	Inferior		F

Fig 14: Tabla de nueva orientación

Mediante la tabla anterior, *Fig14*, con los datos actualizados de la orientación y los colores se puede ver que cuando se quiere realizar un movimiento se ha de buscar relación con la cara que se desea rotar. Por ejemplo, si se desea realizar un movimiento en U se busca la ubicación del número 1 dentro del vector, que es la cara que inicialmente se desea rotar. Como este se encuentra en la posición 4 que originariamente se correspondía con un movimiento B, ahora el movimiento esperado es este último. Para obtener los otros movimientos esperados se procede de forma análoga.

Para seguir con la resolución de la *Fig10* después de haber realizado el giro en R y teniendo en cuenta la nueva orientación presentada en la *Fig13*, se desea realizar un movimiento en U, se debe calcular el movimiento esperado (B). Cuando ya se ha obtenido dicho movimiento se vuelve a realizar otra transformación como la anterior, ahora con la matriz que se corresponde al movimiento B se ha de multiplicar por el vector.

De esta forma se transforma el vector acorde al movimiento realizado y así sucesivamente se va eligiendo el movimiento correcto.

A la hora de realizar toda la conversión se va leyendo la solución proporcionada por el algoritmo y dependiendo de la información del vector de orientación y las 18 matrices, se va escribiendo la nueva secuencia de movimientos.

Gracias a esta modificación de la solución, la información cambia, ya que al principio se relacionaba el movimiento al color de la cara del cubo en una posición concreta. Ahora la información proporcionada al robot no es girar un color específico como sería un movimiento en U que correspondería a la capa blanca del cubo, sino que, el robot deberá girar la cara superior indiferentemente de que esta sea la capa blanca u otra. De esta forma el robot resolverá el cubo sin importar la orientación que vaya tomando el cubo.

Siguiendo con la solución de la *Fig10*, tras ejecutar este fichero se obtendrá esta nueva solución:

$$Adapted\ Solution = 11\ 51\ 51\ 51\ 41\ 31\ 43\ 51\ 12\ 22\ 41\ 42\ 22\ 42\ 52\ 23\ 22\ 42\ 41\ 52\ 33$$

11	12	13	21	22	23	31	32	33	41	42	43	51	52	53	61	62	63
U	U2	U3	F	F2	F3	R	R2	R3	B	B2	B3	L	L2	L3	D	D2	D3

Fig 15: Tabla del significado

Ahora estos números están relacionados a los subprogramas que van a ejecutar, se explica estos subprogramas en el punto 5.3.3.

5.2. COMUNICACIÓN

La comunicación en el proyecto es necesaria para enviar el vector de números con la información de los movimientos para resolver el puzle.

La conexión se realiza mediante TCP/IP el cual es un protocolo de enlace de datos que gracias a internet permite enviar y recibir datos, es decir, posibilita que los dispositivos conectados a internet se comuniquen entre sí en varias redes.

Funciona dividiendo los datos en paquetes más pequeños que se envían por separado y luego se vuelven a ensamblar en el destino. Esto facilita mantener la precisión de la comunicación. El modelo TCP/IP utiliza cuatro capas para enviar y recibir datos, asegurando un proceso estandarizado y eficiente. Es una forma confiable y eficiente de transferir datos en Internet.

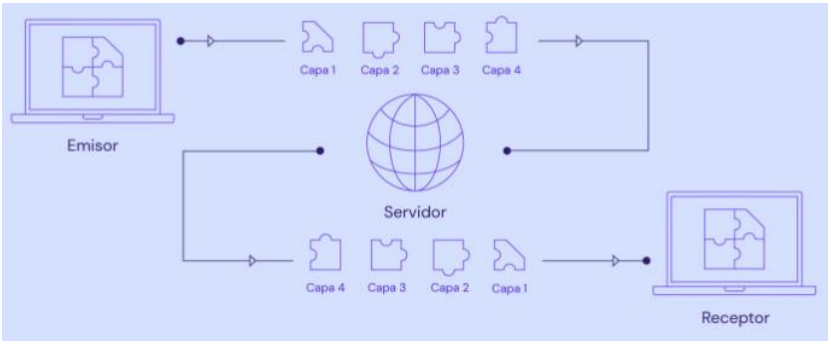


Fig 16: Esquema del funcionamiento TCP/IP

5.2.1. URScript

Para realizar la conexión se ha tratado al robot como un cliente.

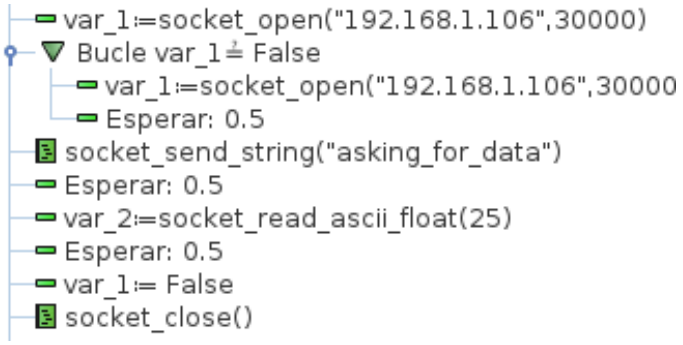


Fig 17: Funciones para la comunicación

Primero, mediante la función socket_open(), se conecta al servidor. Cuando está conectado establece la variable var_1 a TRUE, por lo cual avanzara con la secuencia de comunicación.

Para asegurar el correcto funcionamiento mediante la funión socket_send_string(), se envia un mensaje al servidor, para que este reconozca al cliente y le envie la información.

La funciona utilizada para recibir los datos es socket_read_ascii_float(), esta función necesita como dato el número de floats que va a recibir. La información será guardada como un vector de números en la variable var_2.

Para finalizar la comunicación se cierra la conexión mediante la función socket_close()

De esta forma ahora en la variable var_2 se encuentra toda la información necesaria que se encontrara con el siguiente formato:

$$Var\_2 = (25,62,52,53,51,41,23,31,31,53,41,22,41,22,33,22,33,52,42,42,0,0,0,0,0,0)$$

El vector con la información estará compuesto por un número principal que será predefinido por el valor introducido en la función socket_read_ascii_float(), que siempre será 25. Después de este primer valor se encontrarán los datos de los movimientos del cubo y luego se rellenaran los huecos con 0 hasta alcanzar el rango pedido.

Se ha rellenado el vector con 0 debido a que la función de lectura ha de recibir el mismo número de datos que el escrito en ella. Como cada resolución puede tener un número diferente de movimientos, se ha definido 25, debido a que en ningún caso se alcanzaría esa longitud y así no habrá ningún problema de perdida de información.

5.2.2. Python

Para realizar la conexión se ha tratado al ordenador como el servidor, con una IP de 192.168.1.106:30000. Esta IP a sido proporcionada por la red local que se dispone en el laboratorio donde se realizó el proyecto.

Las funciones utilizadas para la comunicación están incluidas en el módulo *socket*, y ha sido necesario un determinado formato del vector de movimientos para poder ser enviado y leído:

$$MovRobot = (62,52,53,51,41,23,31,31,53,41,22,41,22,33,22,33,52,42,42,0,0,0,0,0,0)$$

Como se ha comentado previamente este debe de tener una longitud de 25 valores, esto se consigue rellenando con 0.

Una vez se tiene el vector adaptado se comienza con la comunicación.

Primero se importan los módulos necesarios y se definen el servidor y el puerto:

```
#Comienza la conexión
import socket
import time

HOST = "192.168.1.106"
PORT = 3000
```

El segundo paso es crear el servidor y dejarlo a la espera de que se conecte un cliente:

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT))  # Enlazar al puerto
s.listen(5)  # Esperar conexión del cliente
c, addr = s.accept()  # Establecer conexión con el cliente
```

Finalmente una vez se recibe el mensaje desde el cliente, el servidor envía los datos mediante la función *sendal()* y si no ha ocurrido ningún problema el servidor se cierra para comenzar con la resolución por parte del robot.

```
    try:
    msg = c.recv(1024)
    print(msg)
    time.sleep(0.5)

    # Enviar los datos a través del socket
    c.sendall(MovRobot.encode('utf-8'))

except socket.error as socketerror:
    print("Error")

c.close()
s.close()
```

5.3. PARTE ROBÓTICA

En la parte robótica se ha utilizado el robot UR3e previamente comentado. La idea principal para resolver el problema es mediante la utilización de un soporte donde reposar el cubo. Para realizar un giro en una cara del cubo, el robot extraerá el cubo del soporte y lo introducirá de forma que, la cara objetivo quede en la parte inferior para, con la ayuda del soporte mantenerla fija y girar todo el resto del cubo.

La toma de imágenes también la realiza el robot, este irá mostrando el cubo a la cámara para que se vayan tomando los datos de las caras, también utiliza el soporte para poder coger de distintas formas el objeto.

5.3.1. Diseño 3D

5.3.1.1. Acoples para la pinza

Para poder agarrar el cubo se ha diseñado dos acoples para la pinza FESTO DHPS-20-A. Al ser una pinza neumática donde el cierre y la apertura no se puede modificar, es necesario de estos acoples para tener a la hora del cierre de la pinza, que la distancia entre los acoples sea exactamente la del cubo.

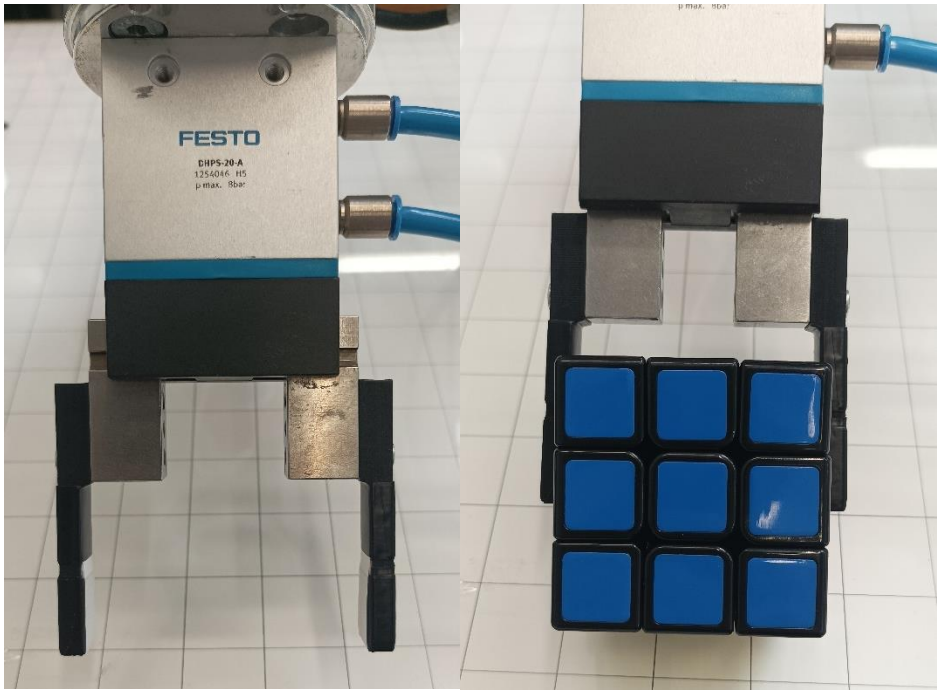


Fig 18: Diseño de nueva pinza

En la instalación del diseño solo se ha necesitado de un tornillo por pieza, aunque se ha diseñado con el mismo formato que la pinza neumática para permitir el máximo anclaje. En la siguiente imagen, *Fig18*, se puede apreciar la forma del diseño y como se ha atornillado.

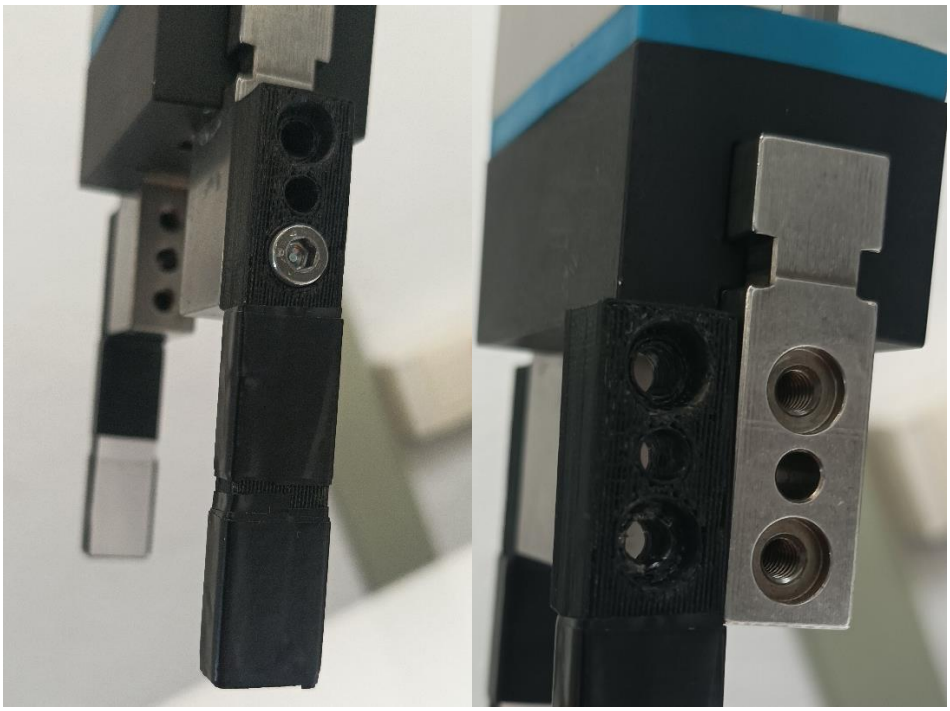


Fig 19: Anclaje del diseño

Uno de los problemas del acople fue el agarre con el cubo, al ser los dos objetos de plástico, el cubo se caía. Para resolver este problema se le ha añadido papel cogido con cinta aislante. Gracias a la superficie más rugosa del papel es suficiente para el correcto funcionamiento del acople.

Fig 20: Agarre del acople

Además de tener la medida exacta a la hora del cierre este acople tiene la longitud necesaria para bloquear dos capas. A la hora de levantar el cubo es importante la posición que toman los acoples en este.

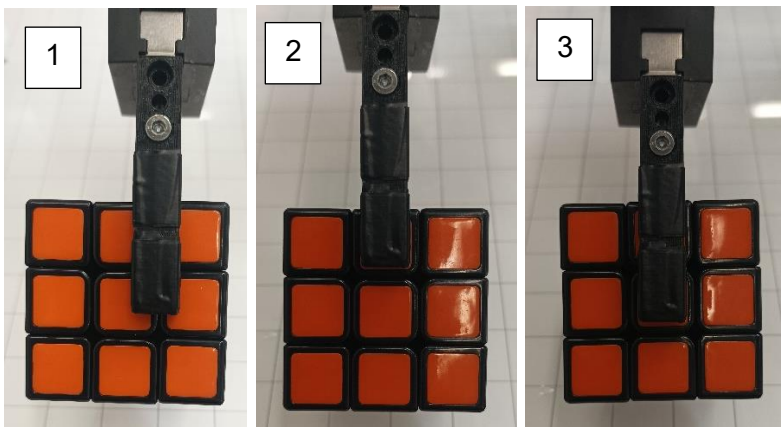


Fig 21: Correcto uso de la pinza

En la imagen anterior, *Fig21*, se muestran 3 formas de coger el cubo. La primera podría ser una opción pero a la hora de girar el cubo respecto a su centro, dificulta el movimiento del robot. La segunda opción no es realizable ya que, a la hora de rotar la capa inferior del cubo también se podría mover la capa intermedia, generando una nueva e indeseada configuración. La tercera opción es la mas adecuada, de esta forma la unica capa libre es la inferior, ademas de mayor facilidad de giro del cubo.

5.3.1.2. Soporte del cubo

Como ya previamente se ha comentado, se ha diseñado un soporte para el cubo. El funcionamiento del diseño es poder reposar el cubo, permitir agarrarlo y extraerlo desde distintos puntos y el bloqueo de la capa inferior del cubo para permitir el giro.

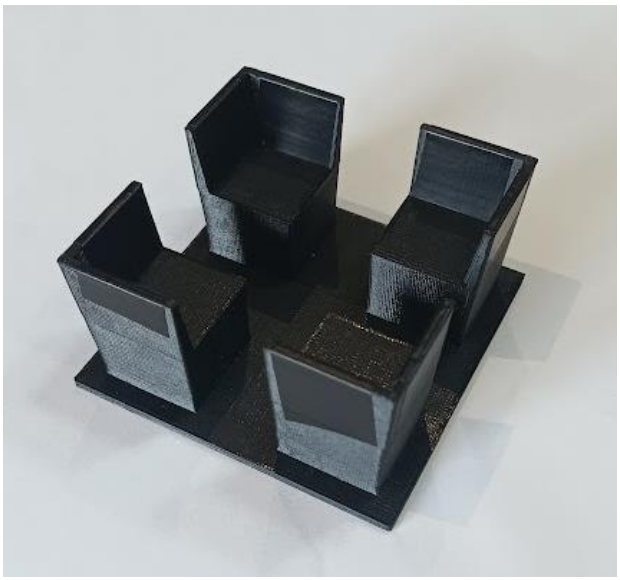


Fig 22: Soporte

También se han creado unos salientes en la parte inferior para facilitar el anclaje a la mesa mediante adhesivos, además de cierta forma de embudo para corregir posibles errores en la introducción del cubo.

Que el soporte permita coger el cubo desde distintos puntos es fundamental para la resolución, ya que es necesario que el robot coja el cubo desde los lados para realizar los giros necesarios.

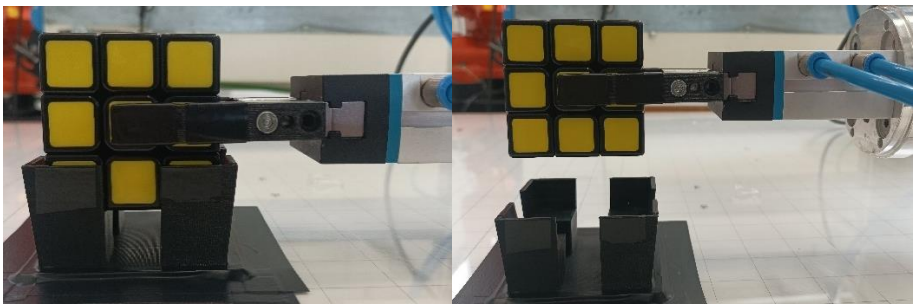


Fig 23: Extracción del cubo desde un lateral

Como se muestra en la *Fig23*, el soporte no supone ningún problema para las pinzas del robot.

A la hora de realizar los giros, el robot introduce el cubo de forma vertical y rota el efector final los grados necesarios.

Fig 24: Giro de la capa inferior

En la *Fig24* el soporte mantiene la cara inferior inmóvil y permite el giro del resto del cubo.

5.3.2. Puntos de paso

Un punto de paso son ubicaciones predeterminadas en la trayectoria de un robot, con una determinada posición y orientación, donde se realizan acciones o se ejecutan tareas específicas durante el proceso.

Respecto a los puntos de paso necesarios para la implementación hay que tener en cuenta las limitaciones donde solo se puede recoger el cubo por tres de sus caras. *Front*, *Right* y *Up*.

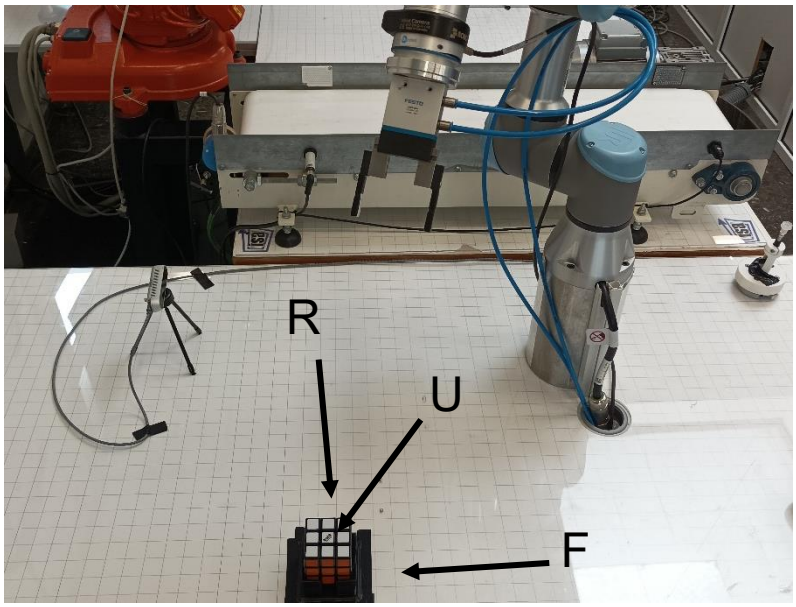


Fig 25: Posibilidades de extracción

Se plantea una nomenclatura para facilitar la programación donde los puntos de paso están definidos en partes.

La primera parte de los puntos de paso define su posicionamiento con respecto al soporte. Si está posicionado en la parte del soporte donde se encuentra el cubo, en esta posición se denomina In. Si posicionado fuera del soporte se denomina *Out*.

La segunda parte de la denominación de los puntos de paso viene definida dependiendo de la orientación en la que esta direccionada la herramienta. En el caso de que la herramienta este perpendicular a la capa superior del cubo, denominada *Up*, este punto de paso será *Up*. Pasa lo mismo con las dos capas horizontales que el robot es capaz de manipular, en este caso *Front* y *Right*.

La última parte solo es necesaria incluirla en los puntos de paso *Up* (tanto *In* como *Out*). En estas posiciones es donde el robot debe ser capaz de rotar para girar la capa, por lo que esta última parte puede ser 90,180,270 (horarios) y se corresponde con el grado de giro del efector final.

Por último, se crean dos puntos fuera de esta nomenclatura como son *AboveFront* y *AboveRight*. Estos dos puntos son necesarios a la hora de la extracción del cubo, debido a que dicha extracción siempre es vertical y es necesario que sea un movimiento lineal y suave para no golpear contra el soporte y modificar las capas del cubo.

Gracias a esta nomenclatura se definen los siguientes puntos para la resolución:

InUp: Dentro del soporte dirección capa *Up*

Fig 26: Punto de paso InUp

OutUp: Fuera del soporte dirección capa *Up*

Fig 27: Punto de paso OutUp

InFront: Dentro del soporte dirección capa *Front*

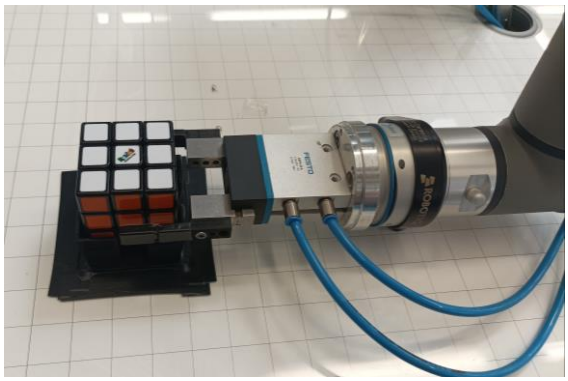


Fig 28: Punto de paso InFront

OutFront: Fuera del soporte dirección capa *Front*

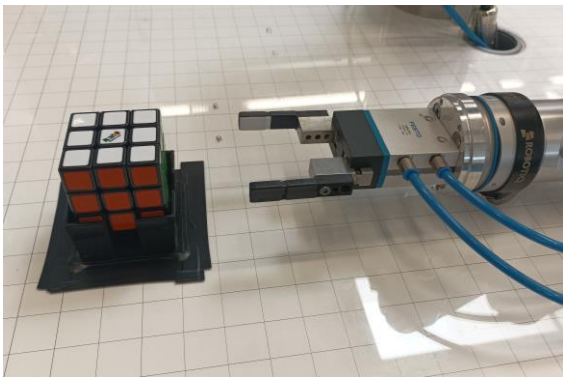


Fig 29: Punto de paso OutFront

InRight: Dentro del soporte dirección capa *Right*

Fig 30: Punto de paso InRight

OutRight: Fuera del soporte dirección capa *Right*

Fig 31: Punto de paso OutUp

InUp90: *InUp* con una rotación de 90º

InUp180: *InUp* con una rotación de 180º

InUp270: *InUp* con una rotación de 270º

Fig 32: Punto de paso InUp90 InUp270

Fig 33: Punto de paso InUp180

Fig 34: Punto de paso

OutUp90: *OutUp* con una rotación de 90º

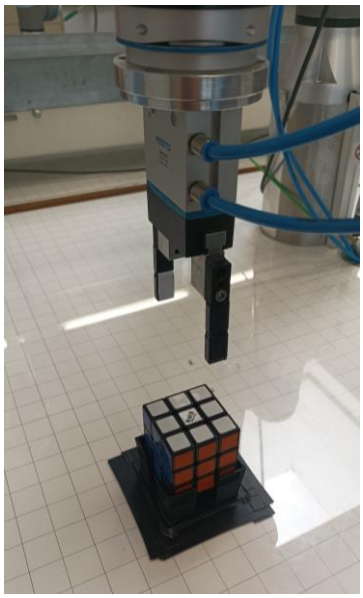


Fig 35: Punto de paso OutUp90

OutUp180: *OutUp* con una rotación de 180º

Fig 36: Punto de paso OutUp180

OutUp270: *OutUp* con una rotación de 270º

Fig 37: Punto de paso OutUp270

AboveFront: Arriba del soporte orientación capa *Front*

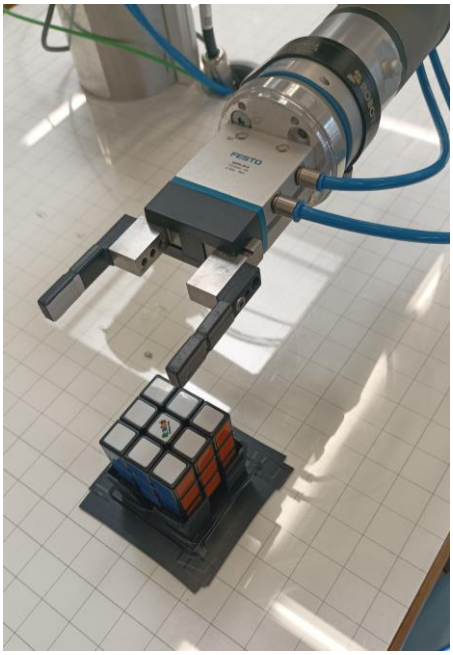


Fig 38: Punto de paso AboveFront

AboveRight: Arriba del soporte orientación capa *Right*

Fig 39: Punto de paso AboveRight

A parte de estos puntos principales se definen unos puntos intermedios para evitar colisiones y realizar los movimientos más fluidos:

- *OutUptoRight*: Punto intermedio entre *OutUp* y *OutRight.*
- *OutUptoFront*: Punto intermedio entre *OutUp* y *OutFront.*
- *AboveFronttoUp*: Punto intermedio entre *AboveFront* y *OutUp.*

También se han implementado 4 puntos más para el reconocimiento del cubo. Estos puntos se encuentran justo enfrente de la cámara con distintas orientaciones.

ToCamera1: En frente de la cámara con dirección vertical y 0º de rotación

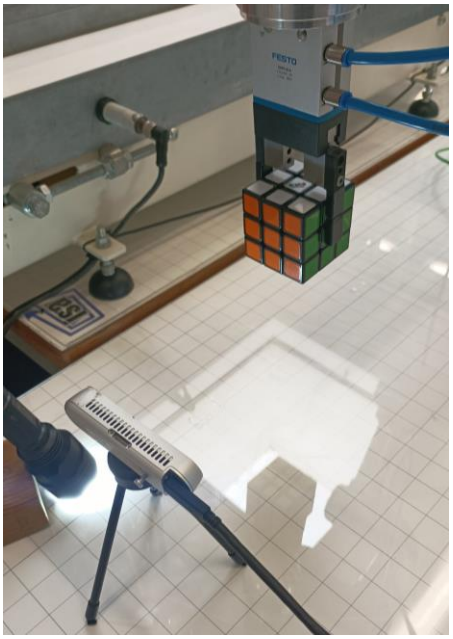


Fig 40: Punto de paso ToCamera1

ToCamera2: En frente de la cámara con dirección vertical y 180º de rotación

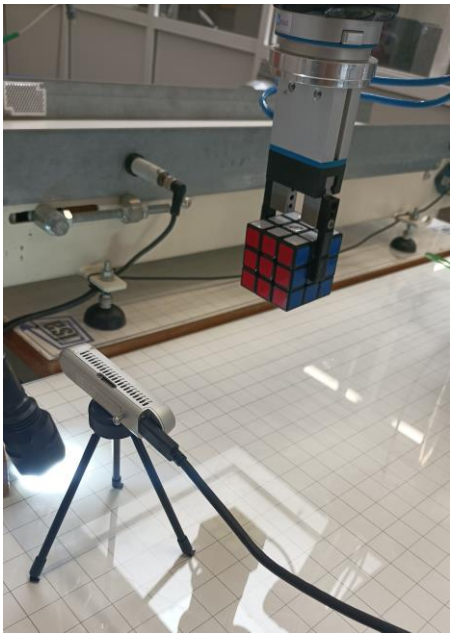


Fig 41: Punto de paso ToCamera2

ToCamera3: En frente de la cámara con dirección horizontal y 0º de rotación

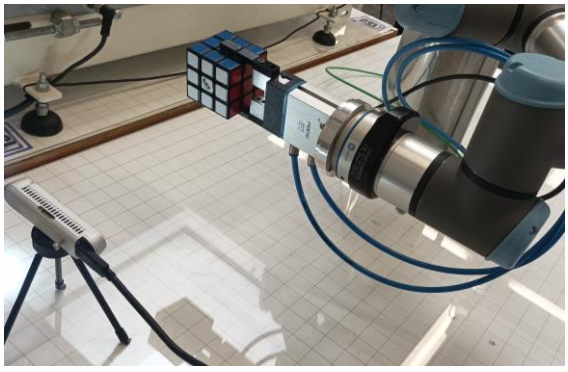


Fig 42: Punto de paso ToCamera3

ToCamera4: En frente de la cámara con dirección horizontal y 180º de rotación

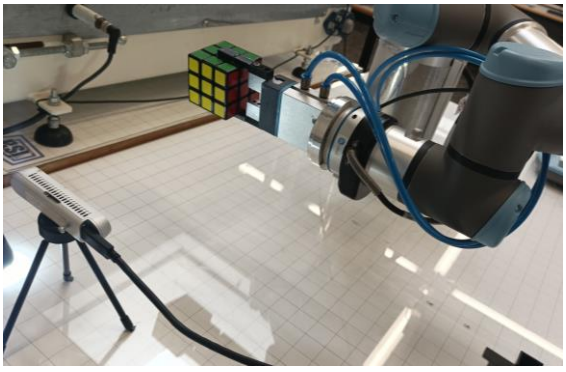


Fig 43: Punto de paso ToCamera4

5.3.3. Movimientos

Para poder realizar los 18 giros posibles en el cubo de Rubik se realiza un subprograma con cada uno de ellos para su utilización dentro de un Script. Dentro de estos subprogramas se encuentra la secuencia necesaria para girar las diferentes caras. Hay 3 subprogramas por cada cara del cubo, donde se contemplan los 3 posibles giros de cada una de ellas.

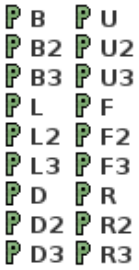


Fig 44: 18 posibles giros

Estos 18 subprogramas están divididos en 4 bloques.

5.3.3.1. Movimientos B, B2, B3, L, L2, L3

El primer bloque son los 6 primeros B, B2, B3, L, L2 y L3. Estos giros corresponden a las capas opuestas a las que el robot puede acceder con facilidad, *Front* y *Right*.

La idea principal para rotar las capas se basa en la extracción del cubo desde una determinada dirección, volver a introducirlo desde la vertical y finalmente rotar el cubo, de esta forma la capa inferior está bloqueada y gira respecto al resto del cubo.

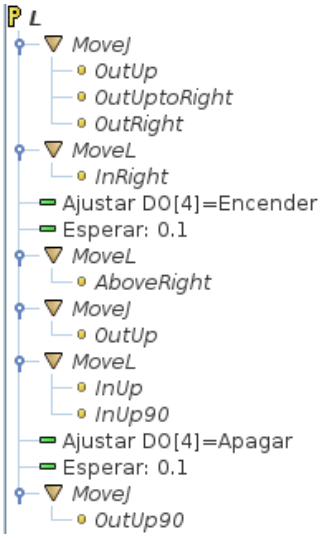


Fig 45: Subprograma L

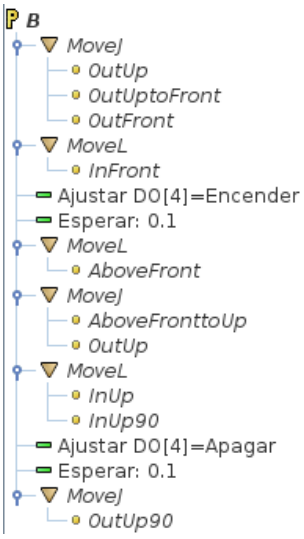


Fig 46: Subprograma B

Con la ejecución de estos subprogramas se rotará la parte del cubo que le corresponda

Para realizar un giro básico de 90º (sentido horario) en la capa *Left*(L), sin importar que color se encuentre en esa posición, el robot deberá alcanzar los siguientes puntos de paso y cerrar o abrir la pinza en el momento correcto.

En este subprograma el robot partirá de la posición inicial, *OutUp*(1), en la parte superior del cubo y avanzara hasta *OutRight*(2), que en este caso es la capa opuesta a *Left*. Para realizar este movimiento sin que se produzca ningún choque se ha hecho pasar, con un determinado radio, la herramienta por un punto intermedio, *OutUptoRight*.

El siguiente paso es la entrada al soporte, punto de paso *InRight*(3), con un movimiento lineal. Una vez alcanzado la posición, activara la pinza para atrapar el cubo.

Después de atrapar el cubo se debe extraer en dirección vertical para evitar colisiones, el siguiente punto de paso es *AboveRight*(4), justo en la vertical del soporte.

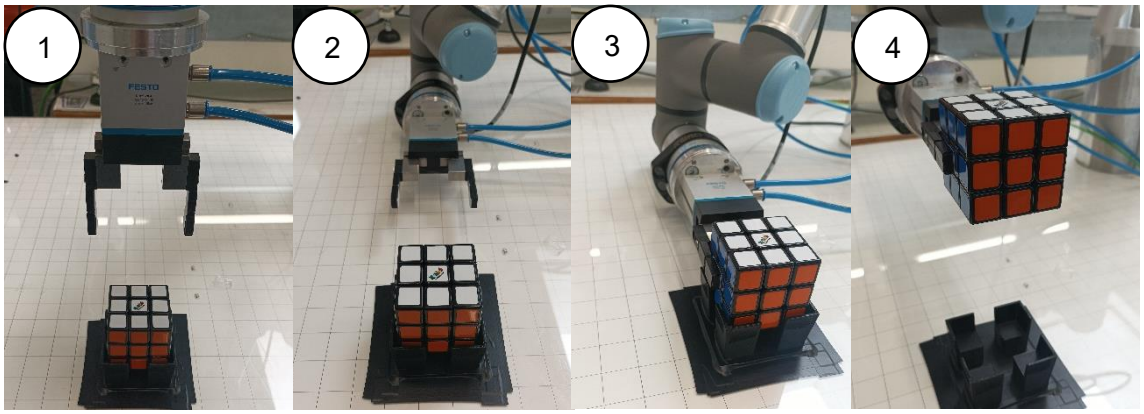


Fig 47: Secuencia Subprograma L 1/2

Una vez extraído correctamente, ahora se debe orientar e introducir en el soporte de nuevo, esto se realiza pasando por el punto de paso *OutUp*(5), y mediante un movimiento lineal, introducir el objeto en el soporte, *InUp*(6).

El siguiente paso es rotar el efector final 90º para así mover el cubo y rotar la cara L, *InUp90*(7). Finalmente se abre la pinza y la herramienta sale con la misma orientación una vez girado el cubo para evitar colisiones, *OutUp90*(8)

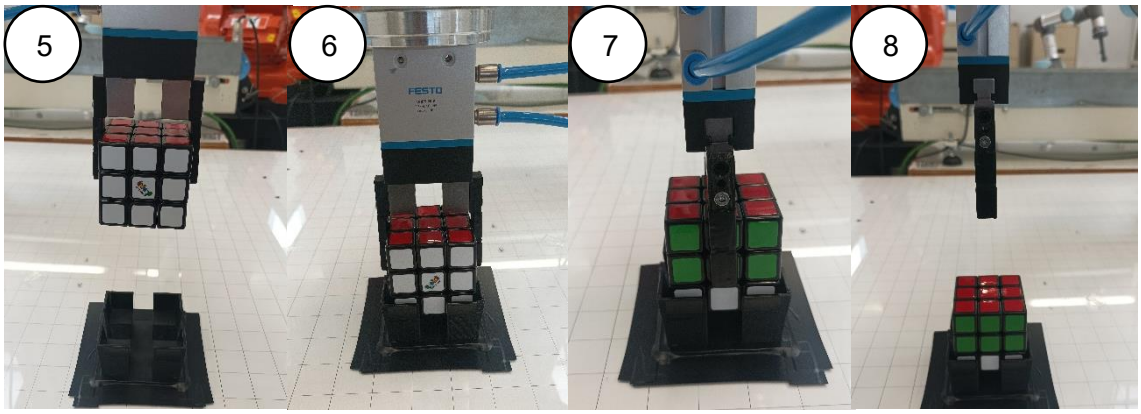


Fig 48: Secuencia Subprograma L 2/2

En el caso de los subprogramas L2 y L3 el único cambio es el ángulo de rotación de la herramienta al final de la secuencia.

- L2: *InUp180*(7) *OutUp180*(8)
- L3: *InUp270*(7) *OutUp270*(8)

El subprograma B es similar al explicado anteriormente. En este caso se deberá extraer el cubo desde otra posición. Para realizar un giro básico de 90º (sentido horario) en la capa *Back*(B), el robot deberá alcanzar los siguientes puntos de paso y cerrar o abrir la pinza en el momento correcto.

En este subprograma el robot partirá de la posición inicial, *OutUp*, en la parte superior del cubo y avanzara hasta *OutFront*(1), que en este caso es la capa opuesta a *Back*. Para realizar este movimiento sin que se produzca ningún choque se ha hecho pasar, con un determinado radio, la herramienta por un punto intermedio, *OutUptoFront*.

El siguiente paso es la entrada al soporte, punto de paso *InFront*(2), con un movimiento lineal. Una vez alcanzado la posición, activara la pinza para atrapar el cubo.

Después de atrapar el cubo se debe extraer en dirección vertical para evitar colisiones, el siguiente punto de paso es *AboveFront*(3), justo en la vertical del soporte.

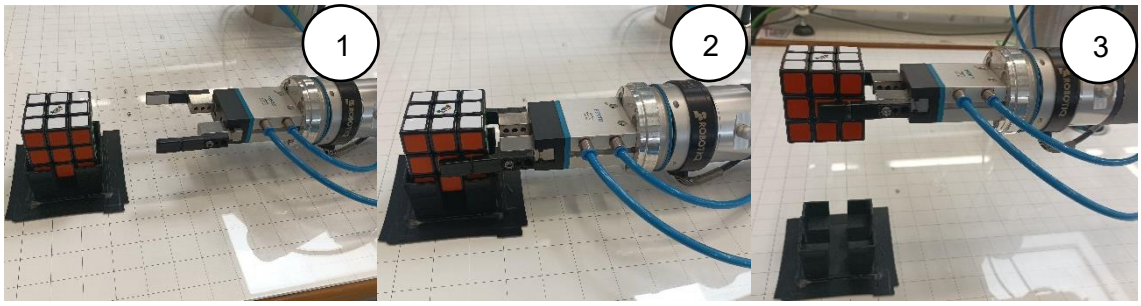


Fig 49: Secuencia Subprograma B 1/2

Una vez extraído correctamente, ahora se debe orientar e introducir en el soporte de nuevo, esto se realiza pasando por el punto de paso con un determinado radio, *AboveFronttoUp*, se alcanza finalmente *OutUp*(4), y mediante un movimiento lineal, introducir el objeto en el soporte, *InUp*(6).

El siguiente paso es rotar el efector final 90º para así mover el cubo y rotar la cara L, *InUp90*(6). Finalmente se abre la pinza y la herramienta sale con la misma orientación una vez girado el cubo para evitar colisiones, *OutUp90*(7).

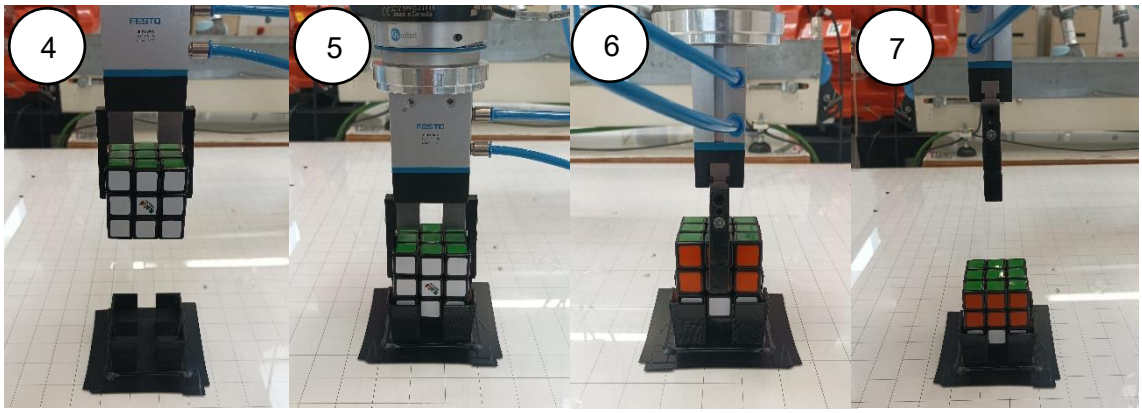


Fig 50: Secuencia Subprograma B 2/2

En el caso de los subprogramas B2 y B3 el único cambio es el ángulo de rotación de la herramienta al final de la secuencia, como pasaba anteriormente con L2 y L3.

- B2: *InUp180*(6) *OutUp180*(7)
- B3: *InUp270*(6) *OutUp270*(7)

5.3.3.2. Movimientos D, D2, D3

El siguiente bloque de subprogramas lo componen D, D2 y D3.

Esta situación solo se puede dar como primer giro de la resolución general, ya que es el único momento donde el cubo pueda necesitar girar la capa inferior. Esto es debido a que en todos los demás giros acaban colocando la capa que necesitan girar en la posición inferior, por lo tanto, el siguiente paso para la resolución no será volver a girar esa capa.

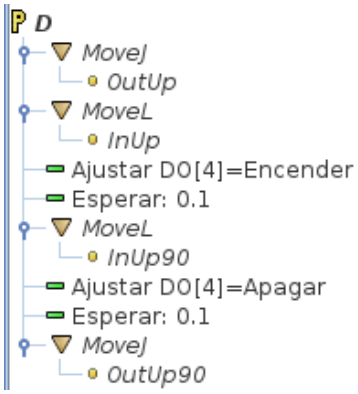


Fig 51: Subprograma B

Para resolver este movimiento solo será necesario comenzar desde *OutUp*(1), introducirse al soporte, *InUp*(2), cerrar la pinza y realizar el giro con el ángulo deseado.

En el caso del subrograma D, se gira la capa 90º, por lo cual el siguiente punto es *InUp90*(3). Finalmente se abre la pinza y la herramienta se aleja con la misma orientación *OutUp90*(4).

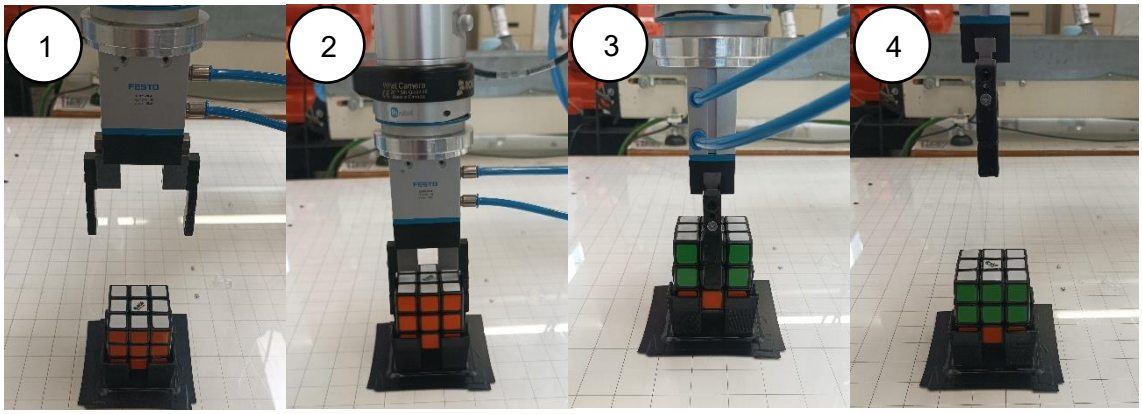


Fig 52: Secuencia Subprograma D

En el caso de los subprogramas D2 y D3 el único cambio es el ángulo de rotación de la herramienta al final de la secuencia, como pasaba anteriormente con los otros giros

- D2: *InUp180*(3) *OutUp180*(4)
- D3: *InUp270*(3) *OutUp270*(4)

5.3.3.3. Movimientos U, U2, U3

El siguiente bloque corresponde a U, U2 y U3. Estos bloques de subprogramas consisten en girar la capa superior del cubo. Para poder realizar este movimiento será necesario orientar el cubo hacia una de las caras contrarias a la que es posible su acceso.

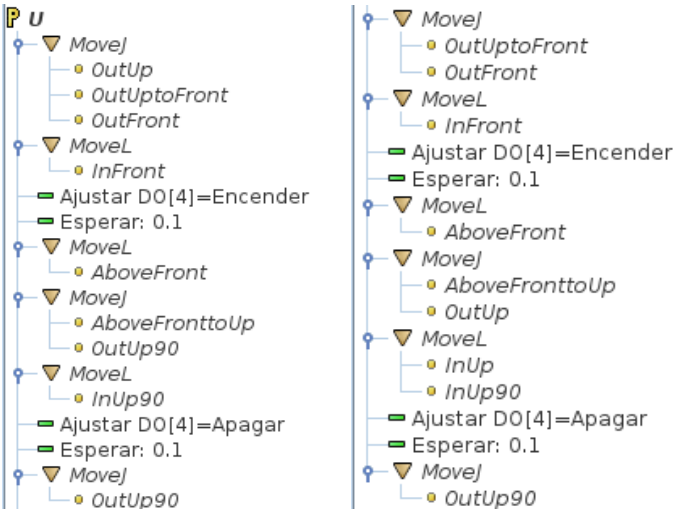


Fig 53: Subprograma U

Este bloque de subprogramas es similar al bloque de B, B2 y B3.

La forma de resolver este problema de acceso se resuelve orientando el cubo de una forma diferente. En este caso se va a extraer el cubo de manera análoga al movimiento en B de la *Fig47.*

Una vez extraído se introduce de nuevo al soporte con el cubo rotado 90º, esto significa llevar el cubo hasta *OutUp90*(1) e introducirlo, *InUp90*(2).

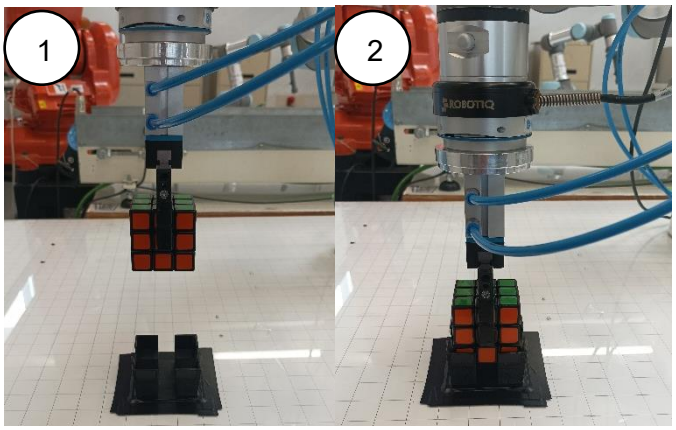


Fig 54: Introducción en subprograma U

Fig 55: Nueva orientación del cubo

Una vez se ha dejado el cubo en el soporte y el robot ha vuelto a la posición inicial, se realiza un movimiento en B explicado anteriormente, ya que ahora la cara objetivo es accesible gracias a su nueva orientación, *Fig53*, y realizable con este movimiento.

5.3.3.4. Movimientos R, R2, R3, F, F2, F3

El último bloque de giros está compuesto por R, R2, R3, F, F2 y F3.

Para la resolución de estos giros se ha de orientar el cubo de una forma adecuada, debido a que las limitaciones del brazo robótico no permiten realizar una secuencia similar al primer bloque de giros.

El primer paso para cumplir con el giro deseado será orientar el cubo. En el caso de un giro en R la herramienta comenzara desde la posición de reposo *OutUp*(1), se introducirá al soporte, *InUp*(2), cerrara la pinza y extraerá el cubo hasta *OutUp* de nuevo. Una vez fuera del soporte este rotara 180º, *OutUp180*(3). Finalmente será introducido de nuevo en el soporte, *InUp180*(4), y se abrirá la pinza y volverá a *OutUp180*.

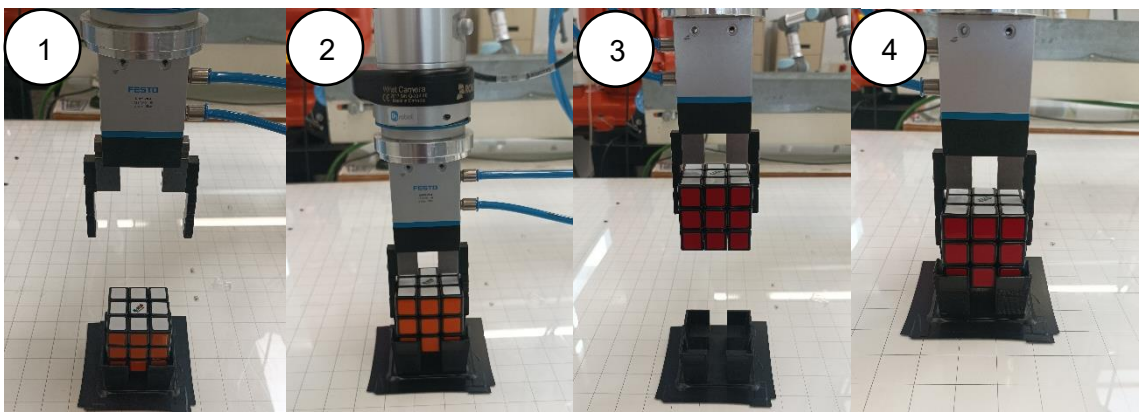


Fig 56: Primer paso subprograma R

A partir de este punto el giro se resuelve de forma idéntica a un giro en la capa L.

Finalmente, para resolver un giro en F en vez de rotar el cubo 180º, como anteriormente en un giro en R, solo es necesario rotar 90º

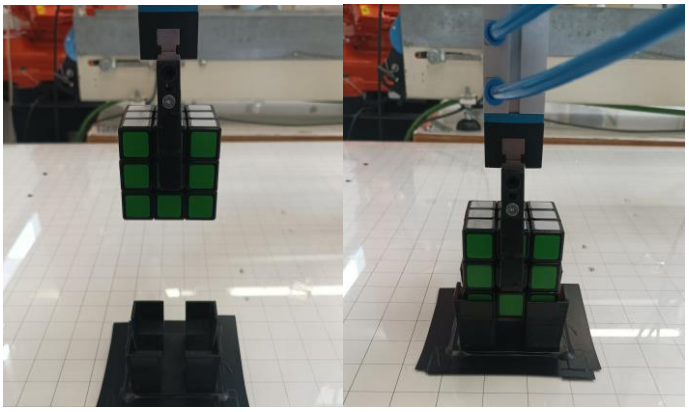


Fig 57: Primer paso subprograma F

A partir de este punto, como pasaba con el subprograma R, es idéntico a un giro en L.

5.3.4. Reconocimiento del cubo

Para el correcto reconocimiento del estado del cubo, el robot deberá extraer el cubo y mostrar las seis caras a la cámara. Esto se realiza con la implementación de un subprograma llamado *Camara*.

El funcionamiento del subporgrama se basa en extraer el cubo del soporte y mediante los puntos de paso ToCamera cuyos puntos se encuentran delante de la camara permiten el reconocimiento.

Cuando el robot alcance estos puntos se esperará un tiempo determinado para que el algoritmo procese adecuadamente el estado del cubo

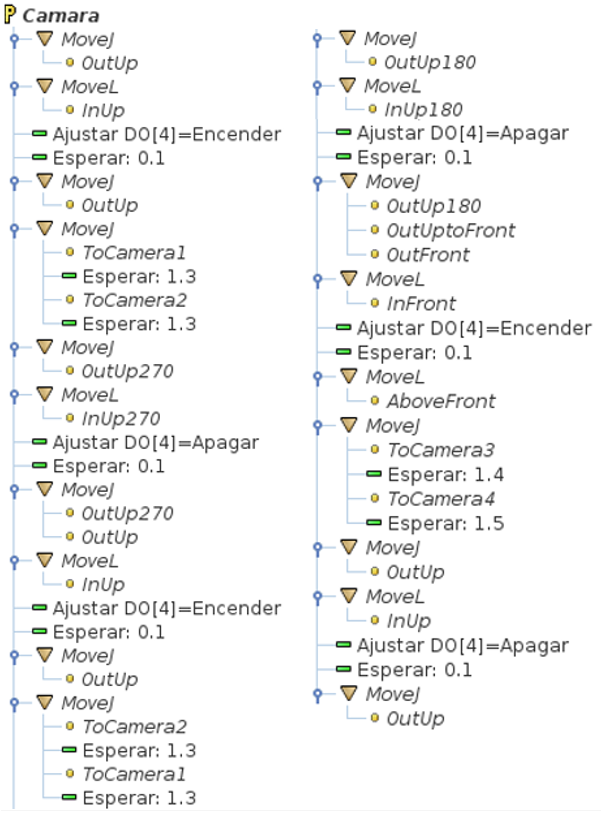


Fig 58: Subprograma Camara

La secuencia de movimientos del subprograma es el siguiente:

Teniendo el cubo en su posición inicial, el robot lo extrae desde la vertical con un movimiento *InUp*(1), una vez fuera del soporte lo mueve hasta *ToCamera1*(2) para el reconocimiento de la primera cara*, Left*, tras una espera de 1.3s rota la herramienta hasta alcanzar *ToCamera2*(3) para escanear la siguiente cara, *Right*.

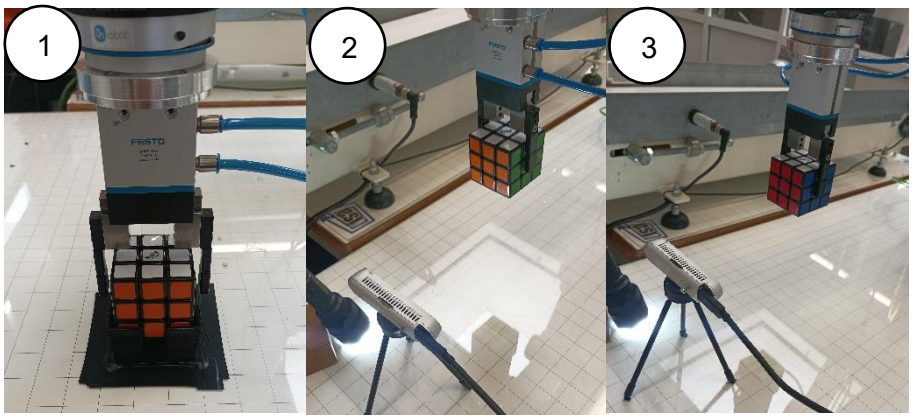


Fig 59: Secuencia Subprograma Camara 1/4

Una vez a pasado el tiempo necesario el cubo vuelve a ser insertado en el soporte, pero esta vez rotado, *OutUp270*(4) y *InUp270*(5). Mediante este giro ahora se permite volver a extraer el cubo, pero mostrando las otras dos caras horizontales. De forma análoga se lleva el cubo hasta *ToCamera2*(6), para escanear la cara *Front*.

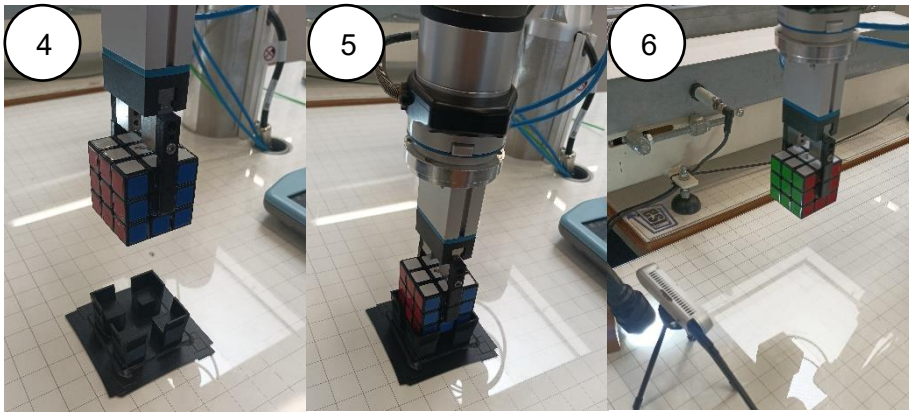


Fig 60: Secuencia Subprograma Camara 2/4

Tras la espera correspondiente el robot vuelve a girar la herramienta 180º, *ToCamera1*(7), para escanear la cuarta cara, *Back*. Después, el cubo vuelve a ser insertado de nuevo con una nueva orientación, *OutUp180*(8) y *InUp180*(9).

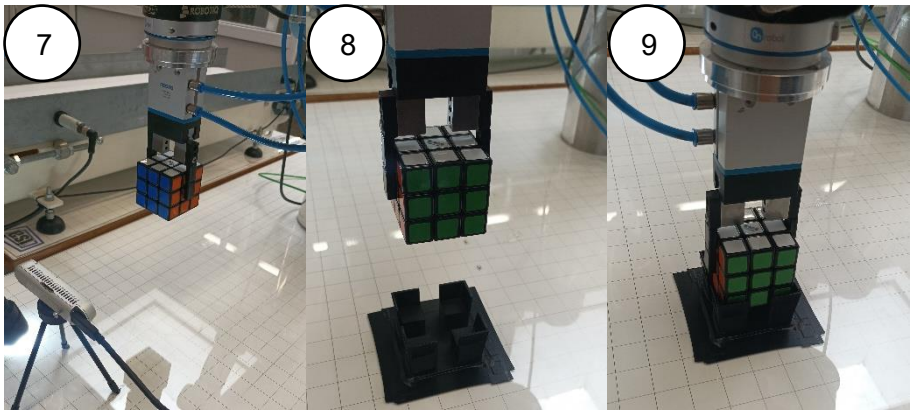


Fig 61: Secuencia Subprograma Camara 3/4

En este momento faltan por escanear las caras superior e inferior del cubo. Para poder realizar el reconocimiento hace falta extraer el cubo desde la posición *InFront*(10). Una vez fuera del soporte el cubo se dirige al siguiente punto, *ToCamera3*(11), y se toman datos de la cara superior, *Up*. Tras una espera como ocurría anteriormente, se rota la herramienta, *ToCamera4*(12), para así comenzar la última cara de la toma de datos, la cara inferior, *Down*.

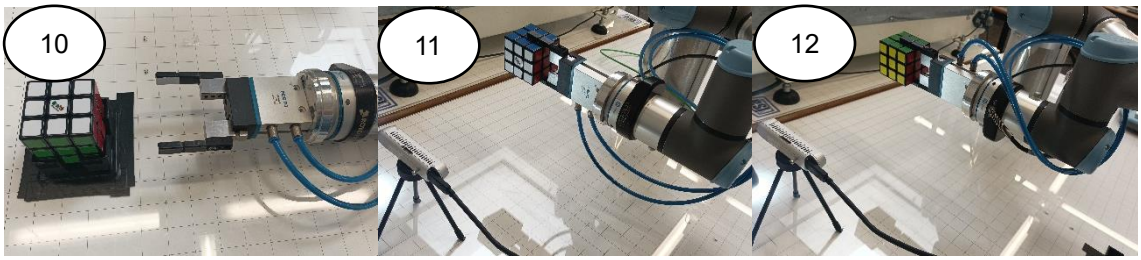


Fig 62: Secuencia Subprograma Camara 4/4

El último paso del subprograma es depositar el cubo en el soporte y llevar la herramienta a su posición inicial, *OutUp*.

cubo=[1, 2, 3, 4, 5, 6]

cubo=[3, 2, 6, 4, 1, 5]

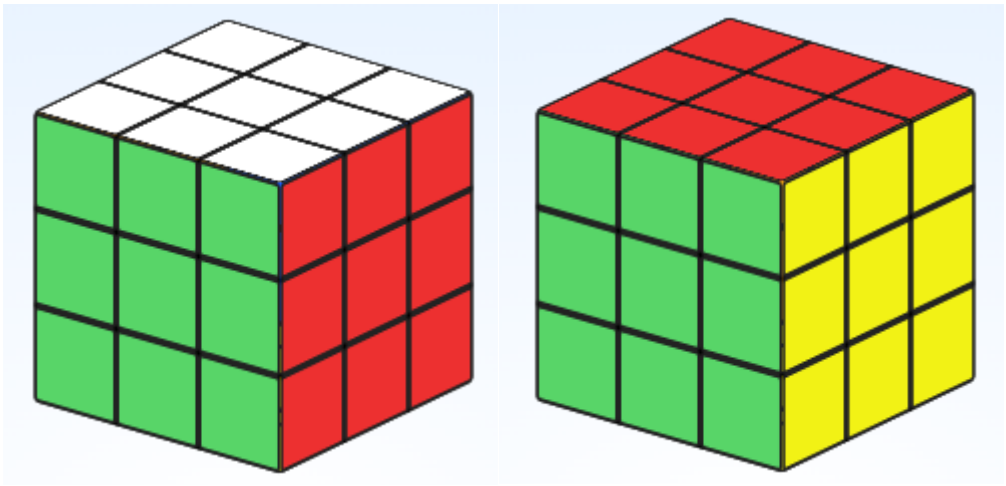


Fig 63: Nueva orientación debido al reconocimiento

Cabe mencionar que al realizar el subprograma *Camara* la orientación del cubo cambia y hay que tenerla en cuenta a la hora de realizar la adaptación de la secuencia al robot explicada en el punto 5.1.2. e inicializar correctamente el cubo en el fichero.

5.3.5. Script final

A la hora de automatizar todos los subprogramas con los diferentes giros se necesita la programación de un Script. A partir del vector de números obtenido gracias a la comunicación con el ordenador, se diseña el script de forma que vaya recorriendo el vector mediante un bucle. Gracias a diferentes if se llama a los diferentes subprogramas para la resolución del problema. Una vez finalizado el Script, el robot se moverá hasta la posición inicial y se podrá recoger el cubo resuelto que estará apoyado en el soporte.

```
variable=var_2          //Lectura de la variable de la comunicación
n = length(variable)    //Calculo de la longitud
i=1                     //Inicialización a 1 debido a la función de la comunicación
while i<n:              //Comienza el bucle
        if variable[i]==11:
                U()
        …….
        …….

        elif variable[i]==63:
                D3()
        elif variable[i]==0:

        end

        i=i+1

end                     //Fin del bucle una vez se haya recorrido todo el vector
```

5.4. SISTEMA FINAL

Una vez ya explicadas las 3 partes fundamentales del proyecto, es momento de relacionarlas entre ellas.

El sistema final tiene dos partes, donde la primera es siempre idéntica y la segunda depende de la complejidad del cubo

5.4.1. Reconocimiento del cubo

La primera de las dos partes es el reconocimiento del cubo, donde el robot se encarga de mostrar el cubo a la cámara, mientras se van tomando datos del estado actual del puzle.

El programa del robot comienza comprobando que está en la posición de reposo, *OutUp*, y abre la pinza para no causar coques al aproximarse al cubo

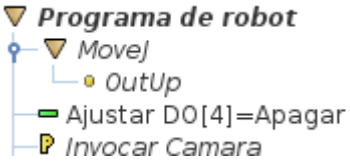


Fig 64: Inicio del programa del robot

Al final de este proceso de seguridad se invoca el subprograma *Camara* que trabajara simultáneamente con el programa desarrollado en Python.

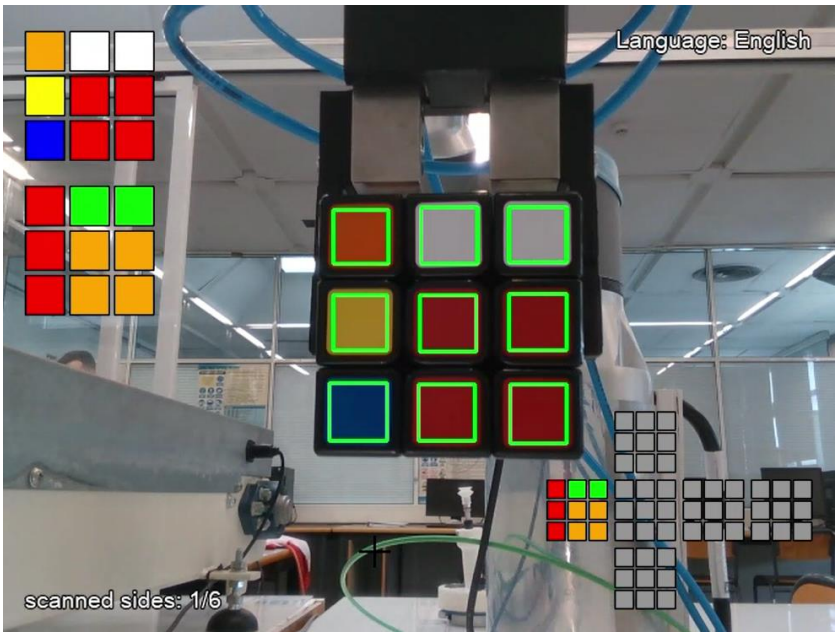


Fig 65: Vista del cubo desde la cámara

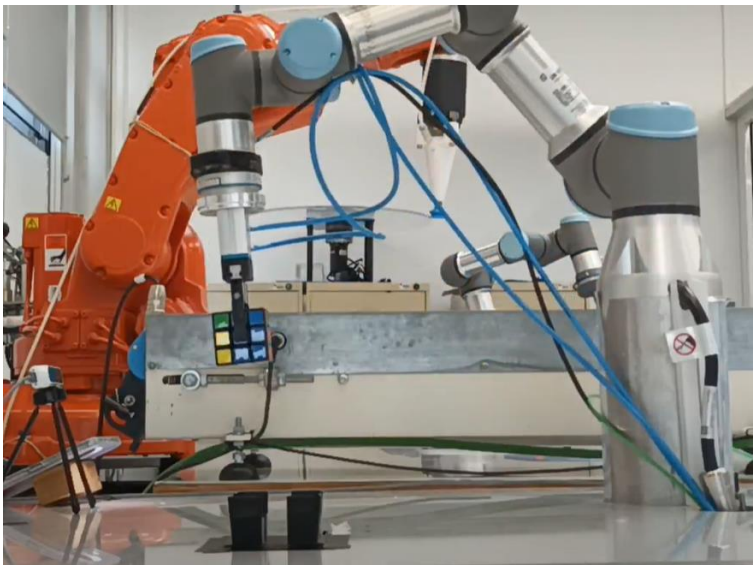


Fig 66: Vista del cubo desde el exterior

Las dos figuras anteriores corresponden al mismo instante de tiempo, la primera imagen es tomada a partir de lo que capta la cámara, y la segunda es una fotografía donde se puede apreciar la disposición de los elementos.

Como se ve en la *Fig65* el robot está mostrando el cubo a la cámara y esta va reconociendo las caras, en este caso ya ha tomado los datos de la cara *Left* y la siguiente es la opuesta, *Right.*

En la segunda imagen, *Fig66*, se puede comprobar el correcto funcionamiento de los dos sistemas en conjunto, el robot y la cámara.

5.4.2. Resolución del cubo

Una vez terminado la identificación del puzle, la cámara termina su función y se pasan los datos por el algoritmo Kociemba para obtener un resultado. Una vez dada una solución, esta se modifica para ser enviable y realizable por el robot.

Después de realizar la conexión entre el ordenador y el robot, se finaliza el programa informático dejando por consola información de interés

```
Moves: 6
Solution: L U F U L R
Adapted Solution: 61 41 51 21 31 11
Final orientation of the cube: [5 6 2 1 4 3]
```

Fig 67: Información de la ejecución

Gracias a esta información, *Fig67*, se puede conocer el número de movimientos y la solución, tanto para una persona humana como para el robot. También se proporciona la orientación final del cubo, por si desea colocar el cubo con una determinada orientación

En este instante una vez se cierra la conexión entre los sistemas, el robot comienza a movilizar el cubo gracias al script interno que gobierna los movimientos a realizar.

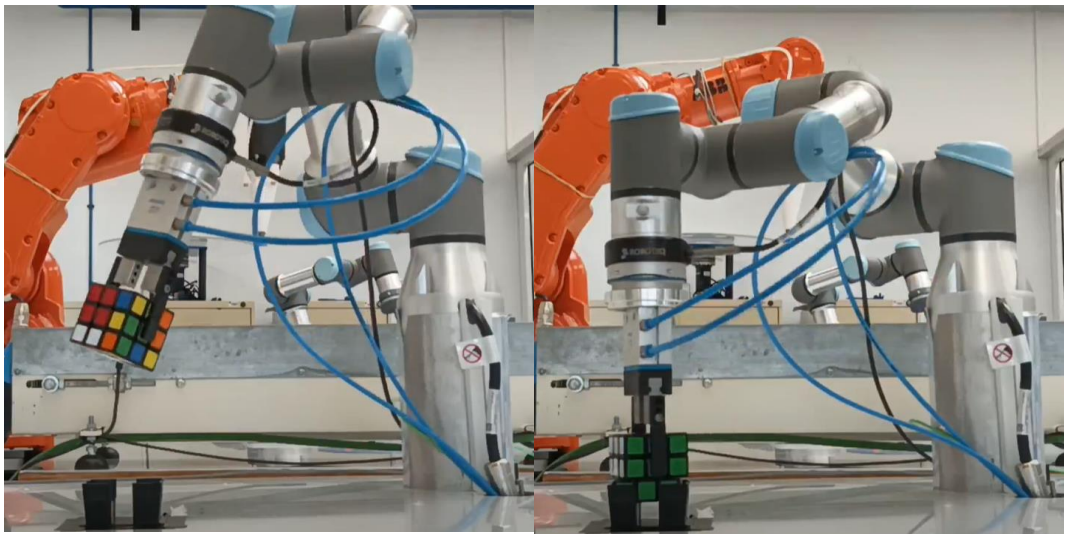


Fig 68: Resolviendo el cubo

Una vez se ha resuelto el cubo el robot espera en la posición inicial, *OutUp*, ya que se ha inicializado una variable a *true*, y hay una espera hasta que esta variable sea *false*. De esta forma el robot se esperará, hasta que se reinicie el programa o se cambie el valor de esta variable.

Si se desea, es el momento de recoger el cubo totalmente resuelto.

Video de la resolución de un cubo:

https://media.upv.es/player/?id=b8ed4180-0c69-11ee-835b-c18acbbb6e72

6. JUSTIFICACIÓN DETALLADA DE LA SELECCIÓN O DIMENSIONAMIENTO DE ELEMENTOS

Para el proyecto se han seleccionado ciertos elementos frente a otros posibles, así como el dimensionamiento de ciertos componentes necesarios para la resolución final.

6.1. ELEMENTOS ALTERNATIVOS

6.1.1. Robot ABB IRB 140

El robot ABB IRB 140 es un robot industrial antropomórfico fabricado por ABB, empresa líder en robótica y automatización. Diseñado para aplicaciones repetitivas de manipulación y ensamblaje en entornos industriales, el IRB 140 es compacto, versátil y altamente preciso.

Este modelo de ABB tiene una carga útil máxima de 6 kg y un alcance máximo de 810 mm, lo que le permite manejar una gran variedad de tareas de montaje y manipulación en espacios reducidos. Su estructura de brazo articulado consta de seis ejes de movilidad que le brindan una gran flexibilidad de movimiento y permiten un posicionamiento y orientación preciso en el espacio de trabajo.

Fig 69: Robot ABB IRB 140

El IRB 140 cuenta con una serie de características y tecnologías avanzadas que mejoran su rendimiento y eficiencia. Equipado con tecnologías avanzadas y opciones personalizables, el IRB 140 es una solución eficiente para automatizar procesos en diferentes industrias.

El robot IRB 140 es una opción válida para casi cualquier ámbito industrial, pero en el caso del proyecto debido a el gran tamaño del robot, dificulta mucho los movimientos y las posiciones tan cerca del suelo a la hora de la extracción del cubo, por ello queda descartado en la resolución.

6.1.1. Cubo de Rubik 3x3 Qiyi

Qiyi es una reconocida marca de cubos de Rubik y rompecabezas de velocidad, conocida por su alta calidad, bajo precio y rendimiento en la comunidad de cuberos.

Los cubos 3x3 de Qiyi son especialmente populares entre los cuberos aficionados y profesionales debido a su suavidad de giro, estabilidad y velocidad. Estos cubos están diseñados con mecanismos internos avanzados que permiten un movimiento fluido y sin enganches, lo que facilita la resolución rápida de los rompecabezas.

Este tipo de cubo fue la primera opción al ser más suave y flexible, absorbiendo mejor los errores que se pudieran producir.

Las dos diferencias con el cubo de Rubik original son un tamaño menor, de $1mm$ por lado, pasando a ser de $55mm^3$, y la forma más redondeada de los stickers centrales de cada cara. Esto ultimo dificulta el reconocimiento de la camara, al ser cada pegatina de una forma diferente. Ademas, la gama de colores tiene unos tonos de rojo naranja y amarillo no my diferenciados, dificultando tambien el reconocimiento.

Fig 70: Cubo 3x3 Qiyi

6.2. ELEMENTOS SELECCIONADOS

6.2.1. Robot Colaborativo UR3e

El robot UR3e es un robot colaborativo de la serie e-Series fabricado por Universal Robots. Está diseñado para trabajar de forma segura y eficiente junto a personas, el UR3e es compacto y versátil. Es útil y eficiente para gran número de aplicaciones en entornos industriales y de laboratorio.

El UR3e tiene una capacidad de carga útil de 3 kg y un alcance máximo de 500 mm, lo que le permite manipular objetos con precisión en áreas de trabajo más pequeñas. Con seis grados de libertad, el robot puede moverse de manera flexible y alcanzar diferentes posiciones y ángulos.

Está equipado con sensores y tecnología de detección de fuerza que le permiten interactuar de forma segura con los operadores humanos sin necesidad de barreras físicas de seguridad.

Fig 71: Robot UR3e

A diferencia con el robot ABB IRB 140, este al ser de menor tamaño puede alcanzar esas posiciones más difíciles en la extracción del cubo y moverse más rápido a la hora de realizar los movimientos. Además, el cobot es fácil de programar y de usar gracias a su interfaz intuitiva. Utilizando el software de programación de Universal Robots, se pueden programar movimientos y tareas mediante la programación manual o la simulación en 3D desde la interfaz.

6.2.2. Cubo de Rubik 3x3 Rubik's

A la hora de pensar en alguna marca de cubos, Rubik's es la más famosa y reconocida en todo el mundo, que gracias a su diseño básico del cubo 3x3 ha conseguido ser la número 1 en ventas de este puzle.

Los cubos de Rubik 3x3 son famosos por su diseño compacto y su bonita apariencia con colores brillantes en cada cara. Estos se caracterizan por una calidad adecuada y gracias a su mecanismo interno, permitir un giro suave y preciso de las capas. La marca Rubik's se ha esforzado por mantener altos estándares de calidad en sus productos, especialmente asegurándose de que los cubos sean duraderos y puedan soportar un uso intensivo.

Fig 72: Cubo 3x3 Rubik

Se ha seleccionado este cubo debido a que en este caso todas las pegatinas de sus caras son exactamente iguales. Además de tener una gama de colores bastante diferenciados, este cubo es mucho más sencillo a la hora del reconocimiento mediante visión artificial.

6.2.3. Intel RealSense D415

La Intel RealSense D415 es una cámara de profundidad que ofrece alta precisión en aplicaciones como el escaneo 3D. Utiliza un obturador giratorio (Rolling shutter) en el sensor de profundidad, lo que proporciona una calidad de profundidad superior por grado. Además, cuenta con un sensor RGB integrado, lo que la hace adecuada para autenticación facial, escaneo 3D y captura volumétrica.

Esta cámara es ligera, potente y de bajo coste, lo que la convierte en una opción ideal para el desarrollo y la producción de soluciones de detección. Puede integrarse fácilmente gracias a su software personalizable, lo que permite la creación de dispositivos que comprenden e interactúan mejor con su entorno.

La cámara D415 ofrece un sensor de profundidad con una resolución de hasta 1280x720 y una frecuencia de 90 fps. El sensor RGB integrado cuenta con una resolución de 1920x1080 a 30 fps.

La D415 es adecuada para el correcto reconocimiento del cubo gracias a la alta resolución del sensor RGB, el único que se va a utilizar en el proyecto. Además, gracias a un pequeño trípode es fácilmente integrable en el espacio de trabajo.

Fig 73: Intel Realsense D415

6.3. ELEMENTOS DISEÑADOS

Las medidas más detalladas de los elementos diseñados se encuentran en el documento nº2 planos.

Número de plano	Descripción
1	Soporte
2	Acoples

Fig 74: Tabla de planos

6.3.1. Acople para las pinzas

Se ha diseñado unos acoples a medida para la pinza FESTO DHPS-20-A.

Se ha obtenido las medidas de la pinza a partir de su datasheet, Anexo nº1.

A partir de las medidas de la pinza y sabiendo que todas las aristas del cubo miden 56mm, se ha diseñado los acoples, imitando las perforaciones y la anchura de los dedos, *Fig75*.

Fig 75: Acople junto al dedo de la pinza

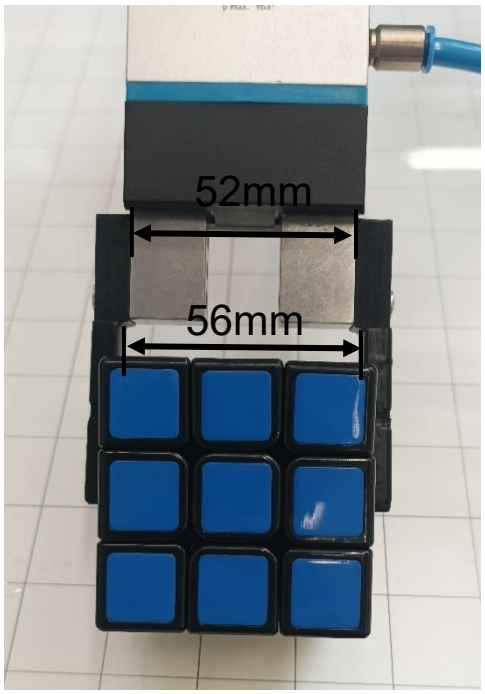


Fig 76: Esquema de la pinza

En la imagen anterior, *Fig76,* se puede apreciar el incremento de $4mm$ que se necesita para lograr agarrar el cubo. Este incremente se ha repartido entre los dos acoples que aportan $3mm$ de separación cada uno. Luego cuando se ha colocado papel en la parte interior para aumentar al agarre se ha disminuido $1mm$ por cada acople, por lo tanto, de esta forma se alcanza la medida necesaria.

Fig 77: Acoples finales

Finalmente se ha logrado el diseño de dos acoples fácilmente integrables en la pinza y con un correcto funcionamiento.

6.3.2. Soporte del cubo

Para el diseño del soporte, se ha tenido en cuenta la medida de las aristas de cubo, 56mm, de esta forma se puede obtener las medidas.

El soporte está compuesto por dos partes:

La primera es la base donde se debe proporcionar cierta altura para que el robot pueda acceder cómodamente al cubo. Se ha optado por unos $20mm$ de altura.

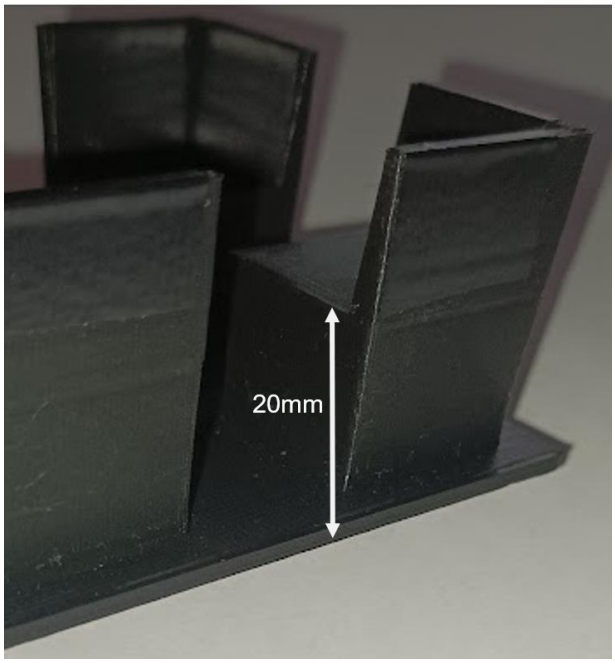


Fig 78: Altura del soporte

La segunda parte es la cesta que va a acoger al cubo, esta tiene forma de embudo donde su base es un cuadrado de $58mm$ por lado. Las paredes del soporte están inclinadas, de esta forma en el extremo de este se forma un cuadrado de 64mm de lado.

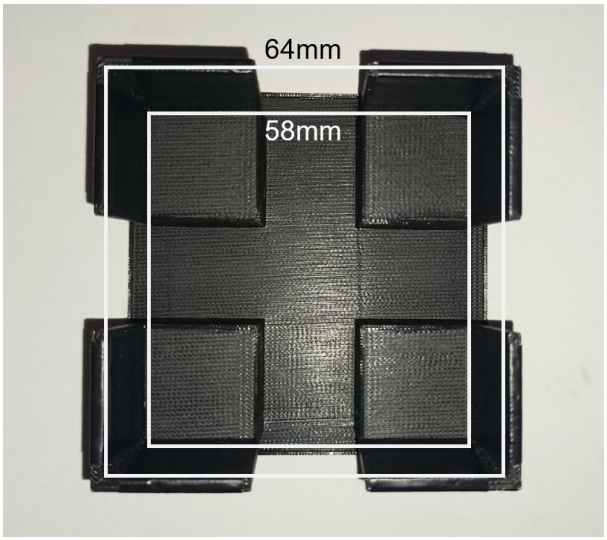


Fig 79: Medidas del interior del soporte

Como el soporte debe bloquear una capa, que en un cubo de $56mm$ mide aproximadamente $19mm$, se ha optado por darle una altura a la cesta de $17mm$. Quedando de esta forma un soporte con una altura total de $37mm$. También se ha añadido cinta aislante en el interior de las paredes, ya que esta es mas deslizante que el material de la impresión y ayuda a la preservación del objeto.

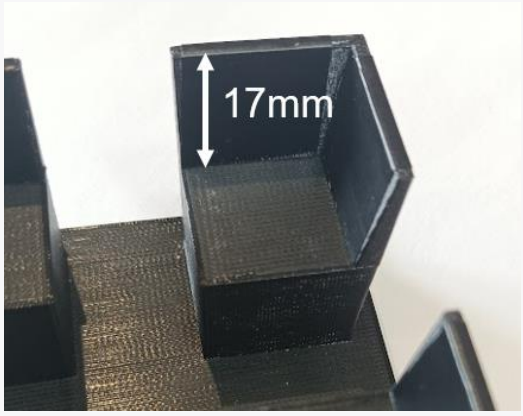


Fig 75: Altura de la cesta

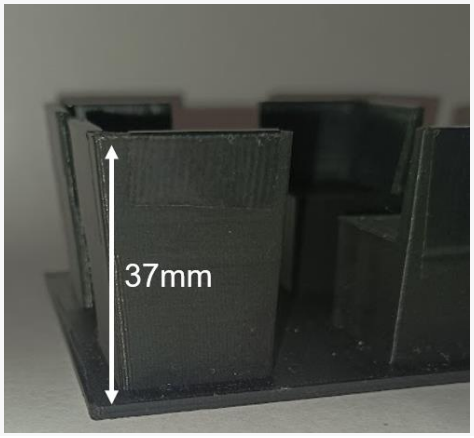


Fig 76: Altura del soporte

Gracias al diseño del soporte el cubo queda perfectamente dentro de él.

Fig 80: Cubo y soporte

7. PRUEBAS Y RESULTADOS

7.1. PROBLEMAS

Durante el desarrollo del proyecto se han ido efectuando pruebas en las cuales han surgido diferentes problemas tanto de diseño de piezas como de programación.

En este apartado se van a comentar los problemas o errores que han ido apareciendo durante el desarrollo.

El primer problema está relacionado con el alcance del robot. Una vez se había diseñado y pensado una forma de realizar los movimientos, a la hora de la implementación se vio que el robot no podía extraer el cubo por dos de sus caras.

Fig 81: Caras inalcanzables

Este problema dificultaba los movimientos para rotar las caras *Front* y *Right*. Se diseñó una nueva forma de rotar esas dos caras que queda reflejada en la explicación del punto 5.3.3.4.

El único aspecto negativo de este problema es el aumento de tiempo de estos movimientos respecto a otros más sencillos.

El segundo problema fue de diseño. Este primer soporte de prueba era más pequeño y con las paredes totalmente verticales.

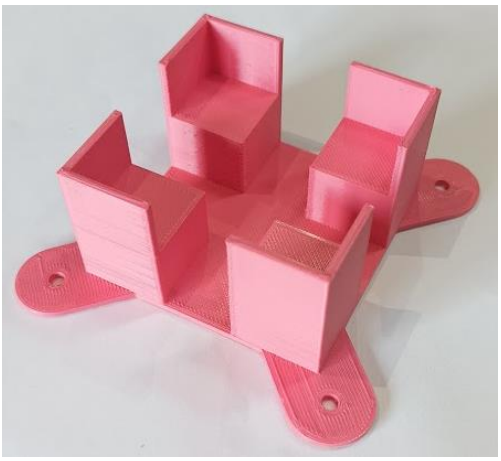


Fig 82: Primer prototipo del soporte

El principal problema que se generaba con este soporte era que, en ciertas ocasiones debido a errores milimétricos, una parte del cubo chocaba y no se introducía adecuadamente. Esto hacía que cuando rotará la herramienta el cubo sufriera una deformación o hasta su ruptura.

Tras ver la problemática de hacer un soporte tan justo se diseñó el segundo y último soporte para eliminar todo problema en el insertado del cubo

7.2. RESULTADOS FINALES

Se han llevado a cabo un total de 15 pruebas reales, con cubos totalmente desordenados, con el objetivo de evaluar tanto el tiempo de resolución como si se producía algún error durante el proceso.

Se han obtenido 15 algoritmos de desordenamiento del cubo para asegurar una complejidad máxima.

	Secuencia de desorden	Secuenda de resolución	Secuencia Robot
1	R' F R B R' U F2 R B' L' B2 U R2 F2 U'	L2 D2 L' B R B' D L F' L F2 D F2 U' B2 D' L2 F2 D2	62 52 53 51 41 23 31 31 53 41 22 41 22 33 22 33 52 42 42
2	D2 B L' R2 U2 R2 U' F' B' U B2 L2 U' F D'	R B' R U F2 R' D' L2 U' R D2 L2 F2 U' R2 B2 U F2 D R2	11 23 41 51 32 23 33 22 33 21 42 32 42 23 32 22 21 42 31 32
3	U' F2 L2 U' F' D2 F2 U2 R2 D L B2 R' F R'	R F' R B2 L' D' R2 F2 D2 F B2 D' B2 L2 D2 F2 D'	11 43 41 42 33 33 22 22 22 51 12 33 42 42 42 22 53
4	U' L F2 U B2 U R2 F' R' D' L2 B2 D2 R F'	F R D2 L F U F2 D2 R' L U2 D' B2 U' R2 F2 B2 D L2 U	21 41 52 31 51 31 22 32 23 11 42 13 22 33 52 42 12 31 32 31
5	B' L U' F D B U2 F' L2 B2 U L R2 F2 D2	L' B' D2 F B' L' U F2 B' U F' D' R2 F2 B2 D B2 R2 U B2 U'	63 53 52 31 13 53 51 52 13 31 43 23 52 22 12 31 22 22 41 52 53
6	F' R' U' L' F U2 R D' L F2 U B D2 R2 F'	F2 R' U2 B' U D2 L' D L' D2 B' U F2 L2 U2 R2 D' L2 D' B2 U	22 33 52 43 41 12 43 41 23 42 43 21 42 42 42 32 33 22 53 32 31
7	U' L B2 U2 F R D F2 B U' R' U2 D' F' L'	U F2 L F' B D' B L U' R2 F' R2 U F2 U L2 F2 R2 U2 R2 F2	51 52 41 23 11 33 41 51 33 22 43 42 21 32 51 32 22 32 22 52 42
8	B2 L F' U2 D2 B2 R2 U F' D2 L U2 F2 R' D	R2 U' B2 R F U2 R' L2 F' U F D R2 U B2 R2 D2 L2 D' F2 U'	12 43 32 41 41 52 43 12 23 51 21 41 32 31 52 42 22 32 53 52 33
9	F' D2 U2 B' L2 F2 D2 U2 R U2 L F' B2 R2 D'	U R U2 F D R' F2 D F U' L D2 R2 U R2 F2 L2 D L2 B2 D'	51 41 22 21 41 33 52 21 21 43 51 42 32 31 22 42 32 21 22 22 23
10	R D2 F R2 U L B R U F2 D' U2 F D L2 U'	U' F U2 D2 F2 L F' D2 R D R' L2 U2 R2 U B2 L2 F2 L2 D	53 31 22 12 42 21 23 32 21 21 23 12 32 32 51 52 22 32 52 21
11	B' L2 U R' D B R2 U' L2 B U' D2 F R2 B2	R U R2 L D L2 U F' R L U2 F2 D2 L2 U F2 B2 D' R2 U	11 51 22 11 31 22 31 53 51 11 32 42 32 42 31 52 12 23 32 31
12	U' L2 U B R2 U2 F R2 U2 D2 R' B L U B2	R U F2 R' L' D R' F B2 R2 L' F2 B2 U D F2 U' L2 D' R2 U2	11 51 32 23 13 51 43 51 12 32 13 52 12 31 11 22 33 52 33 22 32
13	F' B U2 D' F2 D L B' D2 F2 L B R' F' U'	B D B' R' U D F' R' B' L D2 R2 B2 U2 R2 D' L2 U' R2 D	41 31 23 53 31 11 33 33 23 21 52 32 42 22 22 33 22 33 22 31
14	R' B' F' L' F D U' B R U F2 L' D' B U'	R F B U2 R U' R' B' L2 U' R' U F2 U' R2 U' F2 D2 L2 F2 B2	11 41 11 52 21 23 43 53 22 23 23 41 32 53 32 53 52 32 42 42 12
15	L2 B2 U2 F2 U R' F D2 U' F R2 U' F L2 R'	R L2 F' U R2 F' U D2 F' R U' F2 U2 B2 L2	11 12 53 51 52 43 31 12 33 31 33 52 52 32 22

Fig 83: Tabla con los movimientos

En la tabla anterior, *Fig83*, quedan reflejados:

- La secuencia de desordenamiento del puzle.
- Solución proporcionada por el algoritmo Kociemba.
- Solución adaptada y lista para enviar al robot.

	Nº movimientos	Tiempo Total	Tiempo Identificación	Tiempo Resolución
1	19	3:56	0:36	3:20
2	20	4:56	0:36	4:20
3	17	3:55	0:36	3:19
4	20	4:52	0:36	4:16
5	21	4:30	0:36	3:54
6	21	4:37	0:36	4:01
7	21	4:42	0:36	4:06
8	21	4:37	0:36	4:01
9	21	4:48	0:36	4:12
10	20	4:57	0:36	4:21
11	20	4:56	0:36	4:20
12	21	5:10	0:36	4:34
13	20	5:00	0:36	4:24
14	21	4:40	0:36	4:04
15	15	3:41	0:36	3:05
MEDIA	19,87	4:37	0:36	4:01

Fig 84: Tabla con tiempos

En la tabla anterior, *Fig84*, quedan reflejados:

- El número de movimientos necesarios para resolver el estado del cubo.
- El tiempo total del proceso en minutos.
- El tiempo del reconocimiento del cubo con la cámara en minutos.
- El tiempo de resolución del robot con ayuda del soporte en minutos.

Analizando los resultados, como estaba programado, el tiempo de reconocimiento es idéntico en todos los casos, obteniendo un tiempo de 36 segundos.

La primera prueba de resolución fue de 6 minutos, después se aplicó velocidad a los movimientos hasta el punto de alcanzarse una media de 4:01 minutos en las pruebas finales.

El tiempo medio para la resolución final, tanto el reconocimiento como la secuencia de movimientos, es de 4:37 minutos.

También se ha calculado el tiempo que necesita el robot para rotar una cara del cubo, dando como media 12.1 segundos

Las 15 pruebas fueron realizadas con un 100% de éxito. Sin embargo, durante el transcurso de estas, los cables de la pinza FESTO, se iban descolgando cada vez más debido al ligero anclaje que tenían, y quizá después de muchas más pruebas podrían suponer un problema.

Fig 85: Sujeción de los cables

Para solucionar este posible problema, bastaría con aumenta la sujeción al brazo robótico para que los cables no sobresalgan tanto y exista la posibilidad de enredarse con algún objeto del entorno.

8. CONCLUSIONES

En resumen, se han cumplido las especificaciones del proyecto, se deseaba resolver un cubo de Rubik con un robot y una cámara.

Se ha desarrollado un sistema robusto con un 100% de efectividad en las pruebas realizadas. Además de cumplir con los objetivos planteados, se ha desarrollado una técnica novedosa a la hora de utilizar el soporte como ayudante del robot para la resolución del cubo.

Aunque el proyecto haya sido un éxito, existen áreas de mejora, como por ejemplo la implementación de nuevos subprogramas de movimientos, que tengan en cuenta también el siguiente movimiento para dejar el cubo de una forma ventajosa. Quizá de esta forma se podría reducir el tiempo unos 45 segundos.

Gracias a este proyecto se han aprendido diferentes aspectos en distintas áreas tales como:

- La integración de distintos elementos entre sí en un sistema mayor
- La utilización de Python para la visión artificial
- La utilización de un robot colaborativo
- La utilización de impresoras 3D
- La resolución de problema reales
- La resolución y entendimiento de un puzle complejo como el cubo de Rubik

9. BIBLIOGRAFÍA

Herbert Kociemba. (2020) Acerca del algoritmo kociemba. Recuperado de http://kociemba.org/

kkoomen. (2023) A webcam-based 3x3x3 rubik's cube solver written in Python 3 and OpenCV. GitHub. Recuperado de https://github.com/kkoomen/qbr

Sara Gonzalez.(2020) Los misterios matemáticos del cubo de Rubik. BBVAOpenMind. Recuperado de https://www.bbvaopenmind.com/ciencia/matematicas/los-misterios-matematicos-del-cubo-rubik/#:~:text=El%20cubo%20de%20Rubik%20toma,a%20comercializar%20en%20el%20pa%C3%ADs.

¿Qué es la visión artificial y cuáles son sus aplicaciones? Iberdrola. Recuperado de https://www.iberdrola.com/innovacion/vision-artificial#:~:text=La%20visi%C3%B3n%20artificial%20es%20una,actuar%20de%20una%20manera%20determinada.

Todo sobre los Robots Colaborativos. Neobotik. Recuperado de https://www.neobotik.com/robots-colaborativos/#:~:text=%C2%BFQu%C3%A9%20son%20los%20robots%20colaborativos,laboral%2C%20de%20ah%C3%AD%20su%20nombre

Ventajas y aplicaciones de visión artificial en los robots colaborativos. Universal Robots. Recuperado de https://www.universal-robots.com/es/blog/vision-artificial-en-robots/

El Robot UR3e. Universal Robots. Recuperado de https://www.universal-robots.com/products/ur3-robot/

Universal Robots e-Series Manual de usuario. (2018). Universal Robots. Recuperado de https://cfzcobots.com/wp-content/uploads/2018/06/UR3e_User_Manual_es_Global.pdf

The URScript Programming Language for e-Series. (2021). Recuperado de https://s3-eu-west-1.amazonaws.com/ur-support-site/115824/scriptManual_SW5.11.pdf

Pinza paralela DHPS-20-A. Festo. Recuperado de https://www.festo.com/es/es/a/1254046/?q=~:sortByFacetValues-asc

ABB IRB140. Eurobots. Recuperado de https://www.eurobots.es/irb-140-es.html

QiYi Warrior 3x3 Speedcube. Kubekings. Recuperado de https://kubekings.com/cubos-de-rubik-3x3/qiyi-warrior-3x3-s.html

Intel RealSense Depth Camera D415. Intel. Recuperado de https://www.intelrealsense.com/depth-camera-d415/

Danielle Bodnar. (2021). ¿Qué es TCP/IP y cómo funciona? AVG. Recuperado de https://www.avg.com/es/signal/what-is-tcp-ip#:~:text=TCP%2FIP%20es%20un%20protocolo,transmisi%C3%B3n%2Fprotocolo%20de%20Internet).

Hoja de datos pinza paralela DHPS-20-A. Festo. Recuperado de https://www.festo.com/es/es/a/download-document/datasheet/1254046

Robot

UR3e Technical Specifications. Universal Robots. Recuperado de https://www.universal-robots.com/media/1807464/ur3e-rgb-fact-sheet-landscape-a4.pdf

ANEXO 1. RELACIÓN DEL TRABAJO CON LOS OBJETIVOS DE DESARROLLO SOSTENIBLE DE LA AGENDA 2030

Anexo al Trabajo de Fin de Máster: Relación del trabajo con los Objetivos de Desarrollo Sostenible de la agenda 2030

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

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

El sistema diseñado tiene gran relación con la industria, la innovación y las infraestructuras.

La implementación de este sistema implica el desarrollo y aplicación de tecnologías s avanzadas, como la robótica y la visión artificial. Al aplicar estos avances tecnológicos en la resolución del cubo de Rubik, se demuestra la capacidad de la industria para adaptar y utilizar tecnologías innovadoras en diferentes contextos.

Además, este sistema tiene un impacto directo en la innovación, ya que combina diferentes disciplinas y conocimientos para lograr un objetivo específico. La integración de la robótica, la visión artificial y la capacidad de resolver un cubo de Rubik representa una solución innovadora en el campo de la automatización y la inteligencia artificial.

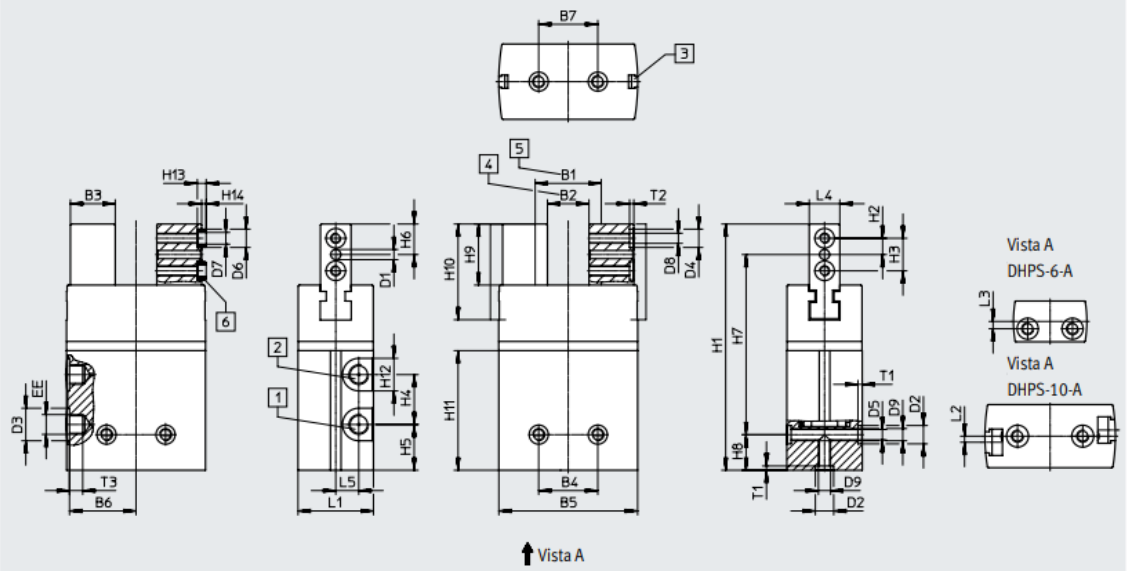
Finalmente, en cuanto a las infraestructuras, este sistema puede tener implicaciones en la optimización de procesos y la mejora de la productividad en diferentes sectores. La capacidad de resolver un cubo de Rubik podría ser un indicador de la flexibilidad y habilidad del robot para manejar múltiples tareas. Esto podría traducirse en una mayor eficiencia y productividad en la cadena de producción.

ANEXO 2. FICHAS TÉCNICAS DE LOS ELEMENTOS

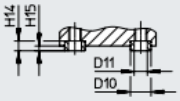
Pinza paralela DHPS-20-A:

Hoja de datos

Dimensiones Descargar datos CAD ➔ www.festo.com



[1] Conexión de aire comprimido para apertura
[2] Conexión de aire comprimido para cierre
[3] Ranura para sensor de proximidad
[4] Posición básica en DHPS-...-A y DHPS-...-A-NC
[5] Posición básica en DHPS-...-A-NO
[6] Casquillos para centrar ZBH (a partir del tamaño 10: 4 unidades incluidas en el suministro)
[7] Interfaz de fijación: Casquillos para centrar ZBH para la fijación de la pinza (2 unidades incluidas en el suministro)



Tamaño	B1	B2	B3	B4[1]	B5	B6	B7[1]	D1 Ø H8	D2 Ø H8	D3 Ø	D4 Ø H8
[mm]	±0,5	±0,5	−0,03		±0,1						
20	30	17	17,5	25	55,6	26,65	25	4	7	10	7

Tamaño	D5 Ø +0,1	D6 Ø h7	D7 Ø	D8	D9	D10 Ø h7	D11 Ø	EE	H1	H2	H3[1]
[mm]											
20	3,3	7	5,3	M4	M4	7	5,3	M5	101	7	14

Tamaño	H4	H5	H6	H7 ±0,2	H8[2]	H9	H10	H11	H12	H13 −0,2	H14 −0,3
[mm]											
20	23	16	12,5	81	7,5	25	39,5	50	10	3	1,4

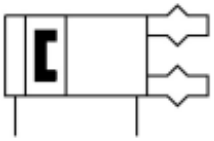
Tamaño	H15	H16	L1	L2	L3[1]	L4	L5	T1	T2	T3
[mm]	−0,2	−0,3				−0,05		+0,1	+0,1	+0,5
20	3	1,4	30±0,1	–	–	12	9	1,6	1,6	6

1) Tolerancia para taladro centrador ±0,02 mm; tolerancia para rosca ±0,1 mm
2) Tolerancia para taladro centrador −0,05 mm; tolerancia para rosca ±0,1 mm

Pinza paralela DHPS-20-A

FESTO

Número de artículo: 1254046



Hoja de datos

Característica	Valor
Tamaño	20
Carrera por mordaza	6.5 mm
Precisión máx. de sustitución	≤0.2 mm
Juego angular máximo de las mordazas ax, ay	<0.5 deg
Holgura máxima Sz de las mordazas	<0.02 mm
Simetría de rotación	≤0.2 mm
Precisión de repetición de las pinzas	<0.02 mm
Número de mordazas	2
Tipo de actuador	neumático
Posición de montaje	Cualquiera
Modo de funcionamiento	Doble efecto
Función de sujeción	Paralelo
Aseguramiento de la fuerza de sujeción	Sin
Forma constructiva	Palanca Movimiento guiado forzado
Guía	Guía deslizante
Detección de posición	Para sensor de proximidad
Símbolo	00991894
Fuerza total de sujeción a 6 bar durante la apertura	320 N
Fuerza de fijación a 6 bar en cierre	290 N
Presión de funcionamiento	2 bar ... 8 bar
Frecuencia de trabajo máxima de la pinza	3 Hz
Tiempo de apertura mínimo con 6 bar	59 ms
Tiempo de cierre mínimo con 6 bar	87 ms
Masa máx. por dedo externo	250 g
Medio de funcionamiento	Aire comprimido según ISO 8573-1:2010 [7:4:4]
Nota sobre el medio de trabajo/mando	Admite funcionamiento con lubricación (lo cual requiere seguir utilizándolo)
Clase de resistencia a la corrosión CRC	1 - riesgo de corrosión bajo
Conformidad PWIS	VDMA24364-B2-L
Idoneidad para la producción de baterías de iones de litio	No pueden utilizarse metales con más de un 5 % de cobre en masa. Excepciones: placas de circuito impreso, cables, conectores eléctricos y bobinas
Temperatura ambiente	5 °C ... 60 °C
Fuerza de sujeción por mordazas a 6 bar, abriendo	162 N
Fuerza de sujeción por mordaza con 6 bar en cierre	147 N
Momento de inercia de la masa	1.489 kgcm²

Característica	Valor
Fuerza estática Fz máxima en la mordaza	250 N
Momento estático Mx máximo en la mordaza	14 Nm
Momento estático My máximo en la mordaza	14 Nm
Momento estático Mz máximo en la mordaza	14 Nm
Intervalos de lubricación para componentes guiados	10 MioCyc
Peso del producto	380 g
Tipo de fijación	A elegir: Con rosca interior y casquillo para centrar Con taladro pasante y casquillos para centrar
Conexión neumática	M5
Nota sobre el material	Conformidad con la Directiva RoHS
Material de la tapa ciega	PA
Material del cuerpo	Aleación de forja de aluminio, anodizado duro
Material de las mordazas	Acero inoxidable de alta aleación

Robot UR3e:

UNIVERSAL ROBOTS

UR3e Technical Specifications

With a 3 kg payload capacity and 500 mm reach, the compact form factor of the UR3e makes it a fit for tight workspaces.

Today, more than 50,000 UR collaborative industrial robots have been delivered to customers across industries and around the world. UR3e is one of four e-Series cobots, each with a different payload and reach combination. e-Series brings incredible flexibility and unparalleled ease of use to your application.

Contact

Universal Robots A/S
Energivej 25
5260 Odense
Denmark
+45 89 93 89 89
sales@universal-robots.com
universal-robots.com

UR3e

Specification

Payload	3 kg (6.6 lbs)
Reach	500 mm (19.7 in)
Degrees of freedom	6 rotating joints
Programming	12 inch touchscreen with polyscope graphical user interface

Performance

Power, Consumption, Maximum Average	300 W
Power, Consumption, Typical with moderate settings (approximate)	100 W
Safety	17 configurable safety functions
Certifications	EN ISO 13849-1, PLd Category 3, and EN ISO 10218-1

Force Sensing, Tool Flange	Force, x-y-z	Torque, x-y-z
Range	30.0 N	10.0 Nm
Precision	2.0 N	0.1 Nm
Accuracy	3.5 N	0.1 Nm

Movement

Pose Repeatability per ISO 9283	± 0.03 mm

Axis movement	Working range	Maximum speed
Base	± 360°	± 180°/s
Shoulder	± 360°	± 180°/s
Elbow	± 360°	± 180°/s
Wrist 1	± 360°	± 360°/s
Wrist 2	± 360°	± 360°/s
Wrist 3	Infinite	± 360°/s

Typical TCP speed	1 m/s (39.4 in/s)

Features

IP classification	IP54
ISO 14644-1 Class Cleanroom	5
Noise	Less than 60 dB(A)
Robot mounting	Any orientation
I/O ports	
Digital in	2
Digital out	2
Analog in	2
Tool I/O Power Supply Voltage	12/24 V
Tool I/O Power Supply	600 mA

Physical

Footprint	Ø 128 mm
Materials	Aluminium, Plastic, Steel
Tool (end-effector) connector type	M8 | M8 8-pin
Cable length robot arm	6 m (236 in) cable included. 12 m (472 in) and high-flex options available.
Weight including cable	11.2 kg (17.7 lbs)
Operating temperature range	0-50°C
Humidity	90%RH (non-condensing)

Control Box

Features

IP classification	IP44
ISO 14644-1 Class Cleanroom	6
Operating temperature range	0-50°C
Humidity	90%RH (non-condensing)
I/O ports	
Digital in	16
Digital out	16
Analog in	2
Analog out	2
Quadrature Digital Inputs	4
I/O Power Supply	24V 2A
Communication	500 Hz Control frequency Modbus TCP PROFINET Ethernet/IP USB 2.0, USB 3.0
Power source	100-240VAC, 47-440Hz

Physical

Control box size (W x H x D)	460 mm x 449 mm x 254 mm (18.2 in x 17.6 in x 10 in)
Weight	12 kg (26.5 lbs)
Materials	Powder Coated Steel

The control box is also available in an OEM version.

Teach Pendant

Features

IP classification	IP54
Humidity	90%RH (non-condensing)
Display resolution	1280 x 800 pixels

Physical

Materials	Plastic, PP
Weight	1.6 kg (3.5 lbs) including 1m of TP cable
Cable length	4.5 m (177.17 in)

The teach pendant is also available in a 3PE option.

ANEXO 3. CÓDIGO IMPLEMENTADO

Archivo URScrip, *Cubo.script*:

```
def Cubo():
  global _hidden_verificationVariable=0
  step_count_fa42f50a_d613_4f81_8ae3_b83f6001bb28 = 0.0
  thread Step_Counter_Thread_526d1e95_23e6_4e28_a0c2_22e7a703906f():
    while (True):
      step_count_fa42f50a_d613_4f81_8ae3_b83f6001bb28 =
step_count_fa42f50a_d613_4f81_8ae3_b83f6001bb28 + 1.0
      sync()
    end
  end
  run Step_Counter_Thread_526d1e95_23e6_4e28_a0c2_22e7a703906f()
  set_standard_analog_input_domain(0, 1)
  set_standard_analog_input_domain(1, 1)
  set_tool_analog_input_domain(0, 1)
  set_tool_analog_input_domain(1, 1)
  set_analog_outputdomain(0, 0)
  set_analog_outputdomain(1, 0)
  set_input_actions_to_default()
  set_tool_communication(False, 115200, 0, 1, 1.5, 3.5)
  set_tool_output_mode(0)
  set_tool_digital_output_mode(0, 1)
  set_tool_digital_output_mode(1, 1)
  set_tool_voltage(0)
  set_target_payload(0.500000, [0.000000, 0.000000, 0.000000], [0.000000, 0.000000,
0.000000, 0.000000, 0.000000, 0.000000])
  set_tcp(p[0.0,0.0,0.205,0.0,0.0,0.0])
  set_safety_mode_transition_hardness(0)
  set_gravity([0.0, 0.0, 9.82])
  global variable_fin=0
  # begin: URCap Installation Node
  #   Source: CobotVNC, 1.2.1, NUTAI S.L.
```

```
#   Type: CobotVNC
# end: URCap Installation Node
# begin: URCap Installation Node
#   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
#   Type: Camera

##########################################
#######Vision urcap preamble start#########

logging_service = rpc_factory("xmlrpc","http://127.0.0.1:4747")
# Converts a pose relative to the flange in the base frame.
def get_T_in_base_from_flange(T_x_in_flange):

  T_flange_in_base = get_actual_tool_flange_pose()

  T_x_in_base = pose_trans(T_flange_in_base, T_x_in_flange)

  return T_x_in_base
end

# Search pose cartesian (camera pose)
T_camera_in_flange = p[0.0, 0.05, 0.05, -0.5, 0.0, 0.0]
snapshot_position_offset = p[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
ignore_snapshot_position = False

# Open connection with vision service
xmlrpc_server=rpc_factory("xmlrpc","http://127.0.0.1:4242")

#######Vision urcap preamble end##########
##########################################

# end: URCap Installation Node
global OutUp_p=p[.399639745892, .350512013999, .159986211776, 2.636137275729,
1.702062587543, -.000236488648]
```

global OutUp_q=[0.9737436771392431, -2.6661082706846244, -0.5378177165987257, -1.5079024893094255, 1.5791442394256872, -1.748467747365174]

global OutUptoFront_p=p[.138568236319, .317911512497, .062890642685, -2.610480289915, .221018415354, -1.438259628904]

global OutUptoFront_q=[1.8083434104919434, -2.2647954426207484, -2.209547519683838, -0.49119921148333745, 0.5816393494606018, 0.5431037545204163]

global OutFront_p=p[.118940926962, .365330855815, -.153797866172, -1.991597406752, .465660123464, -2.064762986115]

global OutFront_q=[1.867083951682483, -3.3076746027544015, -0.8700330355745338, -2.1375338384942726, 0.3642782850996875, 1.991706508092724]

global InFront_p=p[.206959210317, .362866226737, -.153146116262, -2.030592740532, .456288841845, -2.043294397392]

global InFront_q=[1.623467206954956, -3.294346948663229, -0.8911729454994202, -1.9464098415770472, 0.11809254437685013, 1.8003736734390259]

global AboveFront_p=p[.210840660477, .351852460891, -.029809241991, -2.004699952347, .420696667267, -2.016492907354]

global AboveFront_q=[1.6142029762268066, -2.784358640710348, -1.3959176540374756, -1.9746724567809046, 0.06143363565206528, 1.8384953737258911]

global AboveFronttoUp_p=p[.259493794028, .295173976220, .153926974846, -2.724550037599, .361731121630, -.929378437140]

global AboveFronttoUp_q=[1.3399238586425781, -2.0635582409300746, -2.0215039253234863, -0.272793249492981, 0.9664493203163147, -0.07861739793886358]

global InUp_p=p[.399655139229, .350496233057, .040998871460, 2.636118718168, 1.702029815719, -.000152394621]

global InUp_q=[0.9735542555686365, -2.8650851742265324, -0.6977787870872882, -1.1489153404032848, 1.5799107498523144, -1.7452771084423313]

global InUp90_p=p[.400067672766, .350410716888, .040999401614, 3.068149995907, -.662482794779, .000002516629]

global InUp90_q=[0.9723680763805218, -2.867269981589364, -0.6925357760102786, -1.152957991026618, 1.5823144856898983, -0.17453292519943295]

global OutUp90_p=p[.399638979781, .350510589020, .159988432496, 3.067473922687, -.665462232081, .004034454230]

global OutUp90_q=[0.9737522602081299, -2.6660930118956507, -0.5378336906433105, -1.5079127487591286, 1.579094648361206, -0.17455321947206 67]

global InUp180_p=p[.399644995695, .350477908194, .040959400647, 1.704039890535, -2.638998816887, .005908837786]

global InUp180_q=[0.9735420942306519, -2.865117212335104, -0.6978521943092346, -1.148943470125534, 1.5799587965011597, 1.3962223529815674]

global OutUp180_p=p[.399630540467, .350512802660, .159979394107, 1.699901335253, -2.641696568837, .005788990456]

global OutUp180_q=[0.9737486839294434, -2.6661044559874476, -0.5378373861312866, -1.5079365533641358, 1.5792039632797241, 1.3962054252624512]

```
  global InUp270_p=p[.399658455419, .350502326132, .040998902362, .660875562586,
3.068674683250, -.004204747362]

  global InUp270_q=[0.9735477573556732, -2.8651207652514277, -0.6977136025366066, -
1.148914377014381, 1.5799540631593745, -3.315998958783369]

  global OutUp270_p=p[.399652339237, .350497872592, .159988776964, .660937184535,
3.068731937306, -.004273737836]

  global OutUp270_q=[0.973701000213623, -2.6661154232420863, -0.5377913117408752, -
1.5079215106419106, 1.5791831016540527, -3.3191843668567103]

  global OutUptoRight_p=p[.456071409504, .124513304698, .127993182236, -
1.766051536896, -1.268621625889, .023848081377]

  global OutUptoRight_q=[0.39377880096435547, -2.5857039890685023, -
0.9047408699989319, -1.5717369518675746, 2.503070116043091, -2.603776518498556]

  global OutRight_p=p[.381254411455, .028095263068, -.157324347329, -1.394380464468, -
.897718460433, -.981180051489]

  global OutRight_q=[0.19135013222694397, -3.3241735897459925, -0.8315820097923279, -
2.09171785930776, 3.3966221809387207, -4.3088136355029505]

  global InRight_p=p[.386733323880, .157296024348, -.155542499191, -1.403292150147, -
.868471488155, -.977555209609]

  global InRight_q=[0.5261510014533997, -3.3357368908324183, -0.7859976291656494, -
2.1360355816283167, 3.7451350688934326, -4.297172848378317]

  global AboveRight_p=p[.387114589595, .157311894772, -.001247180142, -1.403206010697,
-.868550728564, -.977527351248]

  global AboveRight_q=[0.5248897075653076, -2.7399150333800257, -1.287880301475525, -
2.2325736484923304, 3.739853620529175, -4.30041748682131]

  global ToCamera1_p=p[.437526309716, -.095319953526, .007937526244, 3.051368654872, -
.662215013446, -.106468367913]

  global ToCamera1_q=[0.08565061539411545, -2.286234518090719, -1.2010836601257324, -
1.2932038468173523, 1.582958459854126, -1.0622885862933558]

  global ToCamera2_p=p[.437500276613, -.095354518163, .007915812674, -.674607793974, -
2.998431185390, .027580396109]

  global ToCamera2_q=[0.08559184521436691, -2.286239286462301, -1.2011620998382568, -
1.2931828957847138, 1.5829299688339233, -4.188751761113302]

  global ToCamera3_p=p[.417781546176, -.051433150437, -.006824615073, 1.583415399982,
-.212671539309, .128252783918]

  global ToCamera3_q=[0.7199845314025879, -3.3455387554564417, 0.11270362535585576,
-2.9973980389037074, 0.7695940732955933, 0.1688547134399414]

  global ToCamera4_p=p[.417783727372, -.051447440894, -.006822533307, .276934345788,
2.158316130473, -2.115384039197]

  global ToCamera4_q=[0.7199622988700867, -3.3455540142455042, 0.1126940886126917, -
2.997435232202047, 0.7695596814155579, 3.298701763153076]

  def B():

    $ 40 "B" "noBreak"
```

```
$ 41 "MoveJ"
$ 42 "OutUp" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976, r=0.05)
$ 43 "OutUptoFront" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUptoFront_p, get_tcp_offset()),
qnear=OutUptoFront_q), t=1.2, r=0.05)
$ 44 "OutFront" "breakAfter"
movej(get_inverse_kin(pose_trans(OutFront_p, get_tcp_offset()), qnear=OutFront_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)
$ 45 "MoveL"
$ 46 "InFront" "breakAfter"
movel(pose_trans(InFront_p, get_tcp_offset()), a=1.2, v=0.25)
$ 47 "Set DO[4]=On"
set_standard_digital_out(4, True)
$ 48 "Wait: 0.1"
sleep(0.1)
$ 49 "MoveL"
$ 50 "AboveFront" "breakAfter"
movel(pose_trans(AboveFront_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)
$ 51 "MoveJ"
$ 52 "AboveFronttoUp" "breakAfter"
movej(get_inverse_kin(pose_trans(AboveFronttoUp_p, get_tcp_offset()),
qnear=AboveFronttoUp_q), t=1.0, r=0.05)
$ 53 "OutUp" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)
$ 54 "MoveL"
$ 55 "InUp" "breakAfter"
movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
$ 56 "InUp90" "breakAfter"
movel(pose_trans(InUp90_p, get_tcp_offset()), t=0.8)
$ 57 "Set DO[4]=Off"
set_standard_digital_out(4, False)
$ 58 "Wait: 0.1"
sleep(0.1)
```

```
  $ 59 "MoveJ"
  $ 60 "OutUp90" "breakAfter"
  movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q),
a=1.3962634015954636, v=1.0471975511965976)
 end
 def B2():
  $ 62 "B2" "noBreak"
  $ 63 "MoveJ"
  $ 64 "OutUp" "breakAfter"
  movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976, r=0.05)
  $ 65 "OutUptoFront" "breakAfter"
  movej(get_inverse_kin(pose_trans(OutUptoFront_p, get_tcp_offset()),
qnear=OutUptoFront_q), t=1.2, r=0.05)
  $ 66 "OutFront" "breakAfter"
  movej(get_inverse_kin(pose_trans(OutFront_p, get_tcp_offset()), qnear=OutFront_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)
  $ 67 "MoveL"
  $ 68 "InFront" "breakAfter"
  movel(pose_trans(InFront_p, get_tcp_offset()), a=1.2, v=0.25)
  $ 69 "Set DO[4]=On"
  set_standard_digital_out(4, True)
  $ 70 "Wait: 0.1"
  sleep(0.1)
  $ 71 "MoveL"
  $ 72 "AboveFront" "breakAfter"
  movel(pose_trans(AboveFront_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)
  $ 73 "MoveJ"
  $ 74 "AboveFronttoUp" "breakAfter"
  movej(get_inverse_kin(pose_trans(AboveFronttoUp_p, get_tcp_offset()),
qnear=AboveFronttoUp_q), t=1.0, r=0.05)
  $ 75 "OutUp" "breakAfter"
  movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)
  $ 76 "MoveL"
  $ 77 "InUp" "breakAfter"
  movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
```

```
    $ 78 "InUp180" "breakAfter"
    movel(pose_trans(InUp180_p, get_tcp_offset()), t=1.2)
    $ 79 "Set DO[4]=Off"
    set_standard_digital_out(4, False)
    $ 80 "Wait: 0.1"
    sleep(0.1)
    $ 81 "MoveJ"
    $ 82 "OutUp180" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp180_p, get_tcp_offset()), qnear=OutUp180_q),
a=1.3962634015954636, v=1.0471975511965976)
  end
  def B3():
    $ 84 "B3" "noBreak"
    $ 85 "MoveJ"
    $ 86 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976, r=0.05)
    $ 87 "OutUptoFront" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUptoFront_p, get_tcp_offset()),
qnear=OutUptoFront_q), t=1.2, r=0.05)
    $ 88 "OutFront" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutFront_p, get_tcp_offset()), qnear=OutFront_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)
    $ 89 "MoveL"
    $ 90 "InFront" "breakAfter"
    movel(pose_trans(InFront_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 91 "Set DO[4]=On"
    set_standard_digital_out(4, True)
    $ 92 "Wait: 0.1"
    sleep(0.1)
    $ 93 "MoveL"
    $ 94 "AboveFront" "breakAfter"
    movel(pose_trans(AboveFront_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)
    $ 95 "MoveJ"
    $ 96 "AboveFronttoUp" "breakAfter"
```

```
    movej(get_inverse_kin(pose_trans(AboveFronttoUp_p, get_tcp_offset()),
qnear=AboveFronttoUp_q), t=1.0, r=0.05)

    $ 97 "OutUp" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)

    $ 98 "MoveL"

    $ 99 "InUp" "breakAfter"

    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)

    $ 100 "InUp270" "breakAfter"

    movel(pose_trans(InUp270_p, get_tcp_offset()), t=0.8)

    $ 101 "Set DO[4]=Off"

    set_standard_digital_out(4, False)

    $ 102 "Wait: 0.1"

    sleep(0.1)

    $ 103 "MoveJ"

    $ 104 "OutUp270" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUp270_p, get_tcp_offset()), qnear=OutUp270_q),
a=1.3962634015954636, v=1.0471975511965976)

  end

  def L():

    $ 106 "L" "noBreak"

    $ 107 "MoveJ"

    $ 108 "OutUp" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976, r=0.05)

    $ 109 "OutUptoRight" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUptoRight_p, get_tcp_offset()),
qnear=OutUptoRight_q), t=1.5, r=0.1)

    $ 110 "OutRight" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutRight_p, get_tcp_offset()), qnear=OutRight_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)

    $ 111 "MoveL"

    $ 112 "InRight" "breakAfter"

    movel(pose_trans(InRight_p, get_tcp_offset()), a=1.2, v=0.25)

    $ 113 "Set DO[4]=On"

    set_standard_digital_out(4, True)

    $ 114 "Wait: 0.1"
```

```
    sleep(0.1)
    $ 115 "MoveL"
    $ 116 "AboveRight" "breakAfter"
    movel(pose_trans(AboveRight_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)
    $ 117 "MoveJ"
    $ 118 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.7,
r=0.01)
    $ 119 "MoveL"
    $ 120 "InUp" "breakAfter"
    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 121 "InUp90" "breakAfter"
    movel(pose_trans(InUp90_p, get_tcp_offset()), t=0.8)
    $ 122 "Set DO[4]=Off"
    set_standard_digital_out(4, False)
    $ 123 "Wait: 0.1"
    sleep(0.1)
    $ 124 "MoveJ"
    $ 125 "OutUp90" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q),
a=1.3962634015954636, v=1.0471975511965976)
  end
  def L2():
    $ 127 "L2" "noBreak"
    $ 128 "MoveJ"
    $ 129 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976, r=0.05)
    $ 130 "OutUptoRight" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUptoRight_p, get_tcp_offset()),
qnear=OutUptoRight_q), t=1.5, r=0.1)
    $ 131 "OutRight" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutRight_p, get_tcp_offset()), qnear=OutRight_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)
    $ 132 "MoveL"
    $ 133 "InRight" "breakAfter"
```

```
    movel(pose_trans(InRight_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 134 "Set DO[4]=On"
    set_standard_digital_out(4, True)
    $ 135 "Wait: 0.1"
    sleep(0.1)
    $ 136 "MoveL"
    $ 137 "AboveRight" "breakAfter"
    movel(pose_trans(AboveRight_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)
    $ 138 "MoveJ"
    $ 139 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.7,
r=0.01)
    $ 140 "MoveL"
    $ 141 "InUp" "breakAfter"
    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 142 "InUp180" "breakAfter"
    movel(pose_trans(InUp180_p, get_tcp_offset()), t=1.2)
    $ 143 "Set DO[4]=Off"
    set_standard_digital_out(4, False)
    $ 144 "Wait: 0.1"
    sleep(0.1)
    $ 145 "MoveJ"
    $ 146 "OutUp180" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp180_p, get_tcp_offset()), qnear=OutUp180_q),
a=1.3962634015954636, v=1.0471975511965976)
  end
  def L3():
    $ 148 "L3" "noBreak"
    $ 149 "MoveJ"
    $ 150 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976, r=0.05)
    $ 151 "OutUptoRight" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUptoRight_p, get_tcp_offset()),
qnear=OutUptoRight_q), t=1.5, r=0.1)
    $ 152 "OutRight" "breakAfter"
```

```
    movej(get_inverse_kin(pose_trans(OutRight_p, get_tcp_offset()), qnear=OutRight_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)
    $ 153 "MoveL"
    $ 154 "InRight" "breakAfter"
    movel(pose_trans(InRight_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 155 "Set DO[4]=On"
    set_standard_digital_out(4, True)
    $ 156 "Wait: 0.1"
    sleep(0.1)
    $ 157 "MoveL"
    $ 158 "AboveRight" "breakAfter"
    movel(pose_trans(AboveRight_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)
    $ 159 "MoveJ"
    $ 160 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.7,
r=0.01)
    $ 161 "MoveL"
    $ 162 "InUp" "breakAfter"
    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 163 "InUp270" "breakAfter"
    movel(pose_trans(InUp270_p, get_tcp_offset()), t=0.8)
    $ 164 "Set DO[4]=Off"
    set_standard_digital_out(4, False)
    $ 165 "Wait: 0.1"
    sleep(0.1)
    $ 166 "MoveJ"
    $ 167 "OutUp270" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp270_p, get_tcp_offset()), qnear=OutUp270_q),
a=1.3962634015954636, v=1.0471975511965976)
  end
  def D():
    $ 169 "D" "noBreak"
    $ 170 "MoveJ"
    $ 171 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.0)
```

```
  $ 172 "MoveL"
  $ 173 "InUp" "breakAfter"
  movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
  $ 174 "Set DO[4]=On"
  set_standard_digital_out(4, True)
  $ 175 "Wait: 0.1"
  sleep(0.1)
  $ 176 "MoveL"
  $ 177 "InUp90" "breakAfter"
  movel(pose_trans(InUp90_p, get_tcp_offset()), t=0.8)
  $ 178 "Set DO[4]=Off"
  set_standard_digital_out(4, False)
  $ 179 "Wait: 0.1"
  sleep(0.1)
  $ 180 "MoveJ"
  $ 181 "OutUp90" "breakAfter"
  movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q),
a=1.3962634015954636, v=1.0471975511965976)
 end
 def D2():
  $ 183 "D2" "noBreak"
  $ 184 "MoveJ"
  $ 185 "OutUp" "breakAfter"
  movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.0)
  $ 186 "MoveL"
  $ 187 "InUp" "breakAfter"
  movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
  $ 188 "Set DO[4]=On"
  set_standard_digital_out(4, True)
  $ 189 "Wait: 0.1"
  sleep(0.1)
  $ 190 "MoveL"
  $ 191 "InUp180" "breakAfter"
  movel(pose_trans(InUp180_p, get_tcp_offset()), t=1.2)
  $ 192 "Set DO[4]=Off"
```

```
  set_standard_digital_out(4, False)
  $ 193 "Wait: 0.1"
  sleep(0.1)
  $ 194 "MoveJ"
  $ 195 "OutUp180" "breakAfter"
  movej(get_inverse_kin(pose_trans(OutUp180_p, get_tcp_offset()), qnear=OutUp180_q),
a=1.3962634015954636, v=1.0471975511965976)
 end
 def D3():
  $ 197 "D3" "noBreak"
  $ 198 "MoveJ"
  $ 199 "OutUp" "breakAfter"
  movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.0)
  $ 200 "MoveL"
  $ 201 "InUp" "breakAfter"
  movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
  $ 202 "Set DO[4]=On"
  set_standard_digital_out(4, True)
  $ 203 "Wait: 0.1"
  sleep(0.1)
  $ 204 "MoveL"
  $ 205 "InUp270" "breakAfter"
  movel(pose_trans(InUp270_p, get_tcp_offset()), t=0.8)
  $ 206 "Set DO[4]=Off"
  set_standard_digital_out(4, False)
  $ 207 "Wait: 0.1"
  sleep(0.1)
  $ 208 "MoveJ"
  $ 209 "OutUp270" "breakAfter"
  movej(get_inverse_kin(pose_trans(OutUp270_p, get_tcp_offset()), qnear=OutUp270_q),
a=1.3962634015954636, v=1.0471975511965976)
 end
 def U():
  $ 211 "U" "noBreak"
  $ 212 "MoveJ"
```

$ 213 "OutUp" "breakAfter"

movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), a=1.3962634015954636, v=1.0471975511965976, r=0.05)

$ 214 "OutUptoFront" "breakAfter"

movej(get_inverse_kin(pose_trans(OutUptoFront_p, get_tcp_offset()), qnear=OutUptoFront_q), t=1.2, r=0.05)

$ 215 "OutFront" "breakAfter"

movej(get_inverse_kin(pose_trans(OutFront_p, get_tcp_offset()), qnear=OutFront_q), a=1.8325957145940461, v=1.3962634015954636, r=0.01)

$ 216 "MoveL"

$ 217 "InFront" "breakAfter"

movel(pose_trans(InFront_p, get_tcp_offset()), a=1.2, v=0.25)

$ 218 "Set DO[4]=On"

set_standard_digital_out(4, True)

$ 219 "Wait: 0.1"

sleep(0.1)

$ 220 "MoveL"

$ 221 "AboveFront" "breakAfter"

movel(pose_trans(AboveFront_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)

$ 222 "MoveJ"

$ 223 "AboveFronttoUp" "breakAfter"

movej(get_inverse_kin(pose_trans(AboveFronttoUp_p, get_tcp_offset()), qnear=AboveFronttoUp_q), t=1.0, r=0.05)

$ 224 "OutUp90" "breakAfter"

movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q), a=1.8325957145940461, v=1.3962634015954636, r=0.01)

$ 225 "MoveL"

$ 226 "InUp90" "breakAfter"

movel(pose_trans(InUp90_p, get_tcp_offset()), a=1.2, v=0.25)

$ 227 "Set DO[4]=Off"

set_standard_digital_out(4, False)

$ 228 "Wait: 0.1"

sleep(0.1)

$ 229 "MoveJ"

$ 230 "OutUp90" "breakAfter"

movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q), a=1.3962634015954636, v=1.0471975511965976, r=0.01)

$ 231 "MoveJ"

$ 232 "OutUptoFront" "breakAfter"

movej(get_inverse_kin(pose_trans(OutUptoFront_p, get_tcp_offset()), qnear=OutUptoFront_q), t=1.2, r=0.05)

$ 233 "OutFront" "breakAfter"

movej(get_inverse_kin(pose_trans(OutFront_p, get_tcp_offset()), qnear=OutFront_q), a=1.8325957145940461, v=1.3962634015954636, r=0.01)

$ 234 "MoveL"

$ 235 "InFront" "breakAfter"

movel(pose_trans(InFront_p, get_tcp_offset()), a=1.2, v=0.25)

$ 236 "Set DO[4]=On"

set_standard_digital_out(4, True)

$ 237 "Wait: 0.1"

sleep(0.1)

$ 238 "MoveL"

$ 239 "AboveFront" "breakAfter"

movel(pose_trans(AboveFront_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)

$ 240 "MoveJ"

$ 241 "AboveFronttoUp" "breakAfter"

movej(get_inverse_kin(pose_trans(AboveFronttoUp_p, get_tcp_offset()), qnear=AboveFronttoUp_q), t=1.0, r=0.05)

$ 242 "OutUp" "breakAfter"

movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), a=1.8325957145940461, v=1.3962634015954636, r=0.01)

$ 243 "MoveL"

$ 244 "InUp" "breakAfter"

movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)

$ 245 "InUp90" "breakAfter"

movel(pose_trans(InUp90_p, get_tcp_offset()), t=0.8)

$ 246 "Set DO[4]=Off"

set_standard_digital_out(4, False)

$ 247 "Wait: 0.1"

sleep(0.1)

$ 248 "MoveJ"

$ 249 "OutUp90" "breakAfter"

```
    movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q),
a=1.3962634015954636, v=1.0471975511965976)

  end

  def U2():

    $ 251 "U2" "noBreak"

    $ 252 "MoveJ"

    $ 253 "OutUp" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976, r=0.05)

    $ 254 "OutUptoFront" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUptoFront_p, get_tcp_offset()),
qnear=OutUptoFront_q), t=1.2, r=0.05)

    $ 255 "OutFront" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutFront_p, get_tcp_offset()), qnear=OutFront_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)

    $ 256 "MoveL"

    $ 257 "InFront" "breakAfter"

    movel(pose_trans(InFront_p, get_tcp_offset()), a=1.2, v=0.25)

    $ 258 "Set DO[4]=On"

    set_standard_digital_out(4, True)

    $ 259 "Wait: 0.1"

    sleep(0.1)

    $ 260 "MoveL"

    $ 261 "AboveFront" "breakAfter"

    movel(pose_trans(AboveFront_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)

    $ 262 "MoveJ"

    $ 263 "AboveFronttoUp" "breakAfter"

    movej(get_inverse_kin(pose_trans(AboveFronttoUp_p, get_tcp_offset()),
qnear=AboveFronttoUp_q), t=1.0, r=0.05)

    $ 264 "OutUp90" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)

    $ 265 "MoveL"

    $ 266 "InUp90" "breakAfter"

    movel(pose_trans(InUp90_p, get_tcp_offset()), a=1.2, v=0.25)

    $ 267 "Set DO[4]=Off"

    set_standard_digital_out(4, False)
```

```
$ 268 "Wait: 0.1"
sleep(0.1)
$ 269 "MoveJ"
$ 270 "OutUp90" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q),
a=1.3962634015954636, v=1.0471975511965976, r=0.01)
$ 271 "MoveJ"
$ 272 "OutUptoFront" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUptoFront_p, get_tcp_offset()),
qnear=OutUptoFront_q), t=1.2, r=0.05)
$ 273 "OutFront" "breakAfter"
movej(get_inverse_kin(pose_trans(OutFront_p, get_tcp_offset()), qnear=OutFront_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)
$ 274 "MoveL"
$ 275 "InFront" "breakAfter"
movel(pose_trans(InFront_p, get_tcp_offset()), a=1.2, v=0.25)
$ 276 "Set DO[4]=On"
set_standard_digital_out(4, True)
$ 277 "Wait: 0.1"
sleep(0.1)
$ 278 "MoveL"
$ 279 "AboveFront" "breakAfter"
movel(pose_trans(AboveFront_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)
$ 280 "MoveJ"
$ 281 "AboveFronttoUp" "breakAfter"
movej(get_inverse_kin(pose_trans(AboveFronttoUp_p, get_tcp_offset()),
qnear=AboveFronttoUp_q), t=1.0, r=0.05)
$ 282 "OutUp" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)
$ 283 "MoveL"
$ 284 "InUp" "breakAfter"
movel(pose_trans(InUp_p, get_tcp_offset()), t=1.2)
$ 285 "InUp180" "breakAfter"
movel(pose_trans(InUp180_p, get_tcp_offset()), t=1.2)
$ 286 "Set DO[4]=Off"
```

```
    set_standard_digital_out(4, False)
    $ 287 "Wait: 0.1"
    sleep(0.1)
    $ 288 "MoveJ"
    $ 289 "OutUp180" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp180_p, get_tcp_offset()), qnear=OutUp180_q),
a=1.3962634015954636, v=1.0471975511965976)
  end
  def U3():
    $ 291 "U3" "noBreak"
    $ 292 "MoveJ"
    $ 293 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976, r=0.05)
    $ 294 "OutUptoFront" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUptoFront_p, get_tcp_offset()),
qnear=OutUptoFront_q), t=1.2, r=0.05)
    $ 295 "OutFront" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutFront_p, get_tcp_offset()), qnear=OutFront_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)
    $ 296 "MoveL"
    $ 297 "InFront" "breakAfter"
    movel(pose_trans(InFront_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 298 "Set DO[4]=On"
    set_standard_digital_out(4, True)
    $ 299 "Wait: 0.1"
    sleep(0.1)
    $ 300 "MoveL"
    $ 301 "AboveFront" "breakAfter"
    movel(pose_trans(AboveFront_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)
    $ 302 "MoveJ"
    $ 303 "AboveFronttoUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(AboveFronttoUp_p, get_tcp_offset()),
qnear=AboveFronttoUp_q), t=1.0, r=0.05)
    $ 304 "OutUp90" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)
```

$ 305 "MoveL"

$ 306 "InUp90" "breakAfter"

movel(pose_trans(InUp90_p, get_tcp_offset()), a=1.2, v=0.25)

$ 307 "Set DO[4]=Off"

set_standard_digital_out(4, False)

$ 308 "Wait: 0.1"

sleep(0.1)

$ 309 "MoveJ"

$ 310 "OutUp90" "breakAfter"

movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q), a=1.3962634015954636, v=1.0471975511965976, r=0.01)

$ 311 "MoveJ"

$ 312 "OutUptoFront" "breakAfter"

movej(get_inverse_kin(pose_trans(OutUptoFront_p, get_tcp_offset()), qnear=OutUptoFront_q), t=1.2, r=0.05)

$ 313 "OutFront" "breakAfter"

movej(get_inverse_kin(pose_trans(OutFront_p, get_tcp_offset()), qnear=OutFront_q), a=1.8325957145940461, v=1.3962634015954636, r=0.01)

$ 314 "MoveL"

$ 315 "InFront" "breakAfter"

movel(pose_trans(InFront_p, get_tcp_offset()), a=1.2, v=0.25)

$ 316 "Set DO[4]=On"

set_standard_digital_out(4, True)

$ 317 "Wait: 0.1"

sleep(0.1)

$ 318 "MoveL"

$ 319 "AboveFront" "breakAfter"

movel(pose_trans(AboveFront_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)

$ 320 "MoveJ"

$ 321 "AboveFronttoUp" "breakAfter"

movej(get_inverse_kin(pose_trans(AboveFronttoUp_p, get_tcp_offset()), qnear=AboveFronttoUp_q), t=1.0, r=0.05)

$ 322 "OutUp" "breakAfter"

movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), a=1.8325957145940461, v=1.3962634015954636, r=0.01)

$ 323 "MoveL"

```
    $ 324 "InUp" "breakAfter"
    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 325 "InUp270" "breakAfter"
    movel(pose_trans(InUp270_p, get_tcp_offset()), t=0.8)
    $ 326 "Set DO[4]=Off"
    set_standard_digital_out(4, False)
    $ 327 "Wait: 0.1"
    sleep(0.1)
    $ 328 "MoveJ"
    $ 329 "OutUp270" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp270_p, get_tcp_offset()), qnear=OutUp270_q),
a=1.3962634015954636, v=1.0471975511965976)
  end
  def F():
    $ 331 "F" "noBreak"
    $ 332 "MoveJ"
    $ 333 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.2)
    $ 334 "MoveL"
    $ 335 "InUp" "breakAfter"
    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 336 "Set DO[4]=On"
    set_standard_digital_out(4, True)
    $ 337 "Wait: 0.1"
    sleep(0.1)
    $ 338 "MoveJ"
    $ 339 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976)
    $ 340 "OutUp90" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q), t=0.8)
    $ 341 "MoveL"
    $ 342 "InUp90" "breakAfter"
    movel(pose_trans(InUp90_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 343 "Set DO[4]=Off"
```

```
    set_standard_digital_out(4, False)

    $ 344 "Wait: 0.1"

    sleep(0.1)

    $ 345 "MoveJ"

    $ 346 "OutUp90" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q),
a=1.3962634015954636, v=1.0471975511965976, r=0.01)

    $ 347 "OutUptoRight" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUptoRight_p, get_tcp_offset()),
qnear=OutUptoRight_q), t=1.5, r=0.1)

    $ 348 "OutRight" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutRight_p, get_tcp_offset()), qnear=OutRight_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)

    $ 349 "MoveL"

    $ 350 "InRight" "breakAfter"

    movel(pose_trans(InRight_p, get_tcp_offset()), a=1.2, v=0.25)

    $ 351 "Set DO[4]=On"

    set_standard_digital_out(4, True)

    $ 352 "Wait: 0.1"

    sleep(0.1)

    $ 353 "MoveL"

    $ 354 "AboveRight" "breakAfter"

    movel(pose_trans(AboveRight_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)

    $ 355 "MoveJ"

    $ 356 "OutUp" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.7,
r=0.01)

    $ 357 "MoveL"

    $ 358 "InUp" "breakAfter"

    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)

    $ 359 "InUp90" "breakAfter"

    movel(pose_trans(InUp90_p, get_tcp_offset()), t=0.8)

    $ 360 "Set DO[4]=Off"

    set_standard_digital_out(4, False)

    $ 361 "Wait: 0.1"

    sleep(0.1)
```

```
    $ 362 "MoveJ"
    $ 363 "OutUp90" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q),
a=1.3962634015954636, v=1.0471975511965976)
  end
  def F2():
    $ 365 "F2" "noBreak"
    $ 366 "MoveJ"
    $ 367 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.2)
    $ 368 "MoveL"
    $ 369 "InUp" "breakAfter"
    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 370 "Set DO[4]=On"
    set_standard_digital_out(4, True)
    $ 371 "Wait: 0.1"
    sleep(0.1)
    $ 372 "MoveJ"
    $ 373 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976)
    $ 374 "OutUp90" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q), t=0.8)
    $ 375 "MoveL"
    $ 376 "InUp90" "breakAfter"
    movel(pose_trans(InUp90_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 377 "Set DO[4]=Off"
    set_standard_digital_out(4, False)
    $ 378 "Wait: 0.1"
    sleep(0.1)
    $ 379 "MoveJ"
    $ 380 "OutUp90" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q),
a=1.3962634015954636, v=1.0471975511965976, r=0.01)
    $ 381 "OutUptoRight" "breakAfter"
```

```
    movej(get_inverse_kin(pose_trans(OutUptoRight_p, get_tcp_offset()),
qnear=OutUptoRight_q), t=1.5, r=0.1)

    $ 382 "OutRight" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutRight_p, get_tcp_offset()), qnear=OutRight_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)

    $ 383 "MoveL"

    $ 384 "InRight" "breakAfter"

    movel(pose_trans(InRight_p, get_tcp_offset()), a=1.2, v=0.25)

    $ 385 "Set DO[4]=On"

    set_standard_digital_out(4, True)

    $ 386 "Wait: 0.1"

    sleep(0.1)

    $ 387 "MoveL"

    $ 388 "AboveRight" "breakAfter"

    movel(pose_trans(AboveRight_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)

    $ 389 "MoveJ"

    $ 390 "OutUp" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.7,
r=0.01)

    $ 391 "MoveL"

    $ 392 "InUp" "breakAfter"

    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)

    $ 393 "InUp180" "breakAfter"

    movel(pose_trans(InUp180_p, get_tcp_offset()), t=1.2)

    $ 394 "Set DO[4]=Off"

    set_standard_digital_out(4, False)

    $ 395 "Wait: 0.1"

    sleep(0.1)

    $ 396 "MoveJ"

    $ 397 "OutUp180" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUp180_p, get_tcp_offset()), qnear=OutUp180_q),
a=1.3962634015954636, v=1.0471975511965976)

  end

  def F3():

    $ 399 "F3" "noBreak"

    $ 400 "MoveJ"
```

```
$ 401 "OutUp" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.2)
$ 402 "MoveL"
$ 403 "InUp" "breakAfter"
movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
$ 404 "Set DO[4]=On"
set_standard_digital_out(4, True)
$ 405 "Wait: 0.1"
sleep(0.1)
$ 406 "MoveJ"
$ 407 "OutUp" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976)
$ 408 "OutUp90" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q), t=0.8)
$ 409 "MoveL"
$ 410 "InUp90" "breakAfter"
movel(pose_trans(InUp90_p, get_tcp_offset()), a=1.2, v=0.25)
$ 411 "Set DO[4]=Off"
set_standard_digital_out(4, False)
$ 412 "Wait: 0.1"
sleep(0.1)
$ 413 "MoveJ"
$ 414 "OutUp90" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q),
a=1.3962634015954636, v=1.0471975511965976, r=0.01)
$ 415 "OutUptoRight" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUptoRight_p, get_tcp_offset()),
qnear=OutUptoRight_q), t=1.5, r=0.1)
$ 416 "OutRight" "breakAfter"
movej(get_inverse_kin(pose_trans(OutRight_p, get_tcp_offset()), qnear=OutRight_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)
$ 417 "MoveL"
$ 418 "InRight" "breakAfter"
movel(pose_trans(InRight_p, get_tcp_offset()), a=1.2, v=0.25)
$ 419 "Set DO[4]=On"
```

```
    set_standard_digital_out(4, True)
    $ 420 "Wait: 0.1"
    sleep(0.1)
    $ 421 "MoveL"
    $ 422 "AboveRight" "breakAfter"
    movel(pose_trans(AboveRight_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)
    $ 423 "MoveJ"
    $ 424 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.7,
r=0.01)
    $ 425 "MoveL"
    $ 426 "InUp" "breakAfter"
    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 427 "InUp270" "breakAfter"
    movel(pose_trans(InUp270_p, get_tcp_offset()), t=0.8)
    $ 428 "Set DO[4]=Off"
    set_standard_digital_out(4, False)
    $ 429 "Wait: 0.1"
    sleep(0.1)
    $ 430 "MoveJ"
    $ 431 "OutUp270" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp270_p, get_tcp_offset()), qnear=OutUp270_q),
a=1.3962634015954636, v=1.0471975511965976)
  end
  def R():
    $ 433 "R" "noBreak"
    $ 434 "MoveJ"
    $ 435 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.2)
    $ 436 "MoveL"
    $ 437 "InUp" "breakAfter"
    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 438 "Set DO[4]=On"
    set_standard_digital_out(4, True)
    $ 439 "Wait: 0.1"
```

```
    sleep(0.1)
    $ 440 "MoveJ"
    $ 441 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976)
    $ 442 "OutUp180" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp180_p, get_tcp_offset()), qnear=OutUp180_q),
t=1.2)
    $ 443 "MoveL"
    $ 444 "InUp180" "breakAfter"
    movel(pose_trans(InUp180_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 445 "Set DO[4]=Off"
    set_standard_digital_out(4, False)
    $ 446 "Wait: 0.1"
    sleep(0.1)
    $ 447 "MoveJ"
    $ 448 "OutUp180" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp180_p, get_tcp_offset()), qnear=OutUp180_q),
a=1.3962634015954636, v=1.0471975511965976)
    $ 449 "OutUptoRight" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUptoRight_p, get_tcp_offset()),
qnear=OutUptoRight_q), t=1.7, r=0.1)
    $ 450 "OutRight" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutRight_p, get_tcp_offset()), qnear=OutRight_q),
a=1.8325957145940461, v=1.3962634015954636)
    $ 451 "MoveL"
    $ 452 "InRight" "breakAfter"
    movel(pose_trans(InRight_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 453 "Set DO[4]=On"
    set_standard_digital_out(4, True)
    $ 454 "Wait: 0.1"
    sleep(0.1)
    $ 455 "MoveL"
    $ 456 "AboveRight" "breakAfter"
    movel(pose_trans(AboveRight_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)
    $ 457 "MoveJ"
```

```
    $ 458 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.7,
r=0.01)
    $ 459 "MoveL"
    $ 460 "InUp" "breakAfter"
    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 461 "InUp90" "breakAfter"
    movel(pose_trans(InUp90_p, get_tcp_offset()), t=0.8)
    $ 462 "Set DO[4]=Off"
    set_standard_digital_out(4, False)
    $ 463 "Wait: 0.1"
    sleep(0.1)
    $ 464 "MoveJ"
    $ 465 "OutUp90" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp90_p, get_tcp_offset()), qnear=OutUp90_q),
a=1.3962634015954636, v=1.0471975511965976)
  end
  def R2():
    $ 467 "R2" "noBreak"
    $ 468 "MoveJ"
    $ 469 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.2)
    $ 470 "MoveL"
    $ 471 "InUp" "breakAfter"
    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 472 "Set DO[4]=On"
    set_standard_digital_out(4, True)
    $ 473 "Wait: 0.1"
    sleep(0.1)
    $ 474 "MoveJ"
    $ 475 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976)
    $ 476 "OutUp180" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp180_p, get_tcp_offset()), qnear=OutUp180_q),
t=1.2)
```

```
$ 477 "MoveL"
$ 478 "InUp180" "breakAfter"
movel(pose_trans(InUp180_p, get_tcp_offset()), a=1.2, v=0.25)
$ 479 "Set DO[4]=Off"
set_standard_digital_out(4, False)
$ 480 "Wait: 0.1"
sleep(0.1)
$ 481 "MoveJ"
$ 482 "OutUp180" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUp180_p, get_tcp_offset()), qnear=OutUp180_q),
a=1.3962634015954636, v=1.0471975511965976)
$ 483 "OutUptoRight" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUptoRight_p, get_tcp_offset()),
qnear=OutUptoRight_q), t=1.7, r=0.1)
$ 484 "OutRight" "breakAfter"
movej(get_inverse_kin(pose_trans(OutRight_p, get_tcp_offset()), qnear=OutRight_q),
a=1.8325957145940461, v=1.3962634015954636)
$ 485 "MoveL"
$ 486 "InRight" "breakAfter"
movel(pose_trans(InRight_p, get_tcp_offset()), a=1.2, v=0.25)
$ 487 "Set DO[4]=On"
set_standard_digital_out(4, True)
$ 488 "Wait: 0.1"
sleep(0.1)
$ 489 "MoveL"
$ 490 "AboveRight" "breakAfter"
movel(pose_trans(AboveRight_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)
$ 491 "MoveJ"
$ 492 "OutUp" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.7,
r=0.01)
$ 493 "MoveL"
$ 494 "InUp" "breakAfter"
movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
$ 495 "InUp180" "breakAfter"
movel(pose_trans(InUp180_p, get_tcp_offset()), t=1.2)
```

```
    $ 496 "Set DO[4]=Off"
    set_standard_digital_out(4, False)
    $ 497 "Wait: 0.1"
    sleep(0.1)
    $ 498 "MoveJ"
    $ 499 "OutUp180" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp180_p, get_tcp_offset()), qnear=OutUp180_q),
a=1.3962634015954636, v=1.0471975511965976)
  end
  def R3():
    $ 501 "R3" "noBreak"
    $ 502 "MoveJ"
    $ 503 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.2)
    $ 504 "MoveL"
    $ 505 "InUp" "breakAfter"
    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 506 "Set DO[4]=On"
    set_standard_digital_out(4, True)
    $ 507 "Wait: 0.1"
    sleep(0.1)
    $ 508 "MoveJ"
    $ 509 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976)
    $ 510 "OutUp180" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp180_p, get_tcp_offset()), qnear=OutUp180_q),
t=1.2)
    $ 511 "MoveL"
    $ 512 "InUp180" "breakAfter"
    movel(pose_trans(InUp180_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 513 "Set DO[4]=Off"
    set_standard_digital_out(4, False)
    $ 514 "Wait: 0.1"
    sleep(0.1)
```

$ 515 "MoveJ"

$ 516 "OutUp180" "breakAfter"

movej(get_inverse_kin(pose_trans(OutUp180_p, get_tcp_offset()), qnear=OutUp180_q), a=1.3962634015954636, v=1.0471975511965976)

$ 517 "OutUptoRight" "breakAfter"

movej(get_inverse_kin(pose_trans(OutUptoRight_p, get_tcp_offset()), qnear=OutUptoRight_q), t=1.7, r=0.1)

$ 518 "OutRight" "breakAfter"

movej(get_inverse_kin(pose_trans(OutRight_p, get_tcp_offset()), qnear=OutRight_q), a=1.8325957145940461, v=1.3962634015954636)

$ 519 "MoveL"

$ 520 "InRight" "breakAfter"

movel(pose_trans(InRight_p, get_tcp_offset()), a=1.2, v=0.25)

$ 521 "Set DO[4]=On"

set_standard_digital_out(4, True)

$ 522 "Wait: 0.1"

sleep(0.1)

$ 523 "MoveL"

$ 524 "AboveRight" "breakAfter"

movel(pose_trans(AboveRight_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)

$ 525 "MoveJ"

$ 526 "OutUp" "breakAfter"

movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.7, r=0.01)

$ 527 "MoveL"

$ 528 "InUp" "breakAfter"

movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)

$ 529 "InUp270" "breakAfter"

movel(pose_trans(InUp270_p, get_tcp_offset()), t=0.8)

$ 530 "Set DO[4]=Off"

set_standard_digital_out(4, False)

$ 531 "Wait: 0.1"

sleep(0.1)

$ 532 "MoveJ"

$ 533 "OutUp270" "breakAfter"

```
    movej(get_inverse_kin(pose_trans(OutUp270_p, get_tcp_offset()), qnear=OutUp270_q),
a=1.3962634015954636, v=1.0471975511965976)

  end

  def Camara():

    $ 535 "Camara" "noBreak"

    $ 536 "MoveJ"

    $ 537 "OutUp" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976)

    $ 538 "MoveL"

    $ 539 "InUp" "breakAfter"

    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)

    $ 540 "Set DO[4]=On"

    set_standard_digital_out(4, True)

    $ 541 "Wait: 0.1"

    sleep(0.1)

    $ 542 "MoveJ"

    $ 543 "OutUp" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976, r=0.02)

    $ 544 "MoveJ"

    $ 545 "ToCamera1" "breakAfter"

    movej(get_inverse_kin(ToCamera1_p, qnear=ToCamera1_q), t=1.5)

    $ 546 "Wait: 1.3"

    sleep(1.3)

    $ 547 "ToCamera2" "breakAfter"

    movej(get_inverse_kin(ToCamera2_p, qnear=ToCamera2_q), t=1.2)

    $ 548 "Wait: 1.3"

    sleep(1.3)

    $ 549 "MoveJ"

    $ 550 "OutUp270" "breakAfter"

    movej(get_inverse_kin(pose_trans(OutUp270_p, get_tcp_offset()), qnear=OutUp270_q),
t=1.5, r=0.02)

    $ 551 "MoveL"

    $ 552 "InUp270" "breakAfter"

    movel(pose_trans(InUp270_p, get_tcp_offset()), a=1.2, v=0.25)
```

```
    $ 553 "Set DO[4]=Off"
    set_standard_digital_out(4, False)
    $ 554 "Wait: 0.1"
    sleep(0.1)
    $ 555 "MoveJ"
    $ 556 "OutUp270" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp270_p, get_tcp_offset()), qnear=OutUp270_q),
a=1.3962634015954636, v=1.0471975511965976)
    $ 557 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=0.8)
    $ 558 "MoveL"
    $ 559 "InUp" "breakAfter"
    movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
    $ 560 "Set DO[4]=On"
    set_standard_digital_out(4, True)
    $ 561 "Wait: 0.1"
    sleep(0.1)
    $ 562 "MoveJ"
    $ 563 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976)
    $ 564 "MoveJ"
    $ 565 "ToCamera2" "breakAfter"
    movej(get_inverse_kin(ToCamera2_p, qnear=ToCamera2_q), t=1.5)
    $ 566 "Wait: 1.3"
    sleep(1.3)
    $ 567 "ToCamera1" "breakAfter"
    movej(get_inverse_kin(ToCamera1_p, qnear=ToCamera1_q), t=1.2)
    $ 568 "Wait: 1.3"
    sleep(1.3)
    $ 569 "MoveJ"
    $ 570 "OutUp180" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp180_p, get_tcp_offset()), qnear=OutUp180_q),
t=1.5, r=0.02)
    $ 571 "MoveL"
```

```
$ 572 "InUp180" "breakAfter"
movel(pose_trans(InUp180_p, get_tcp_offset()), a=1.2, v=0.25)
$ 573 "Set DO[4]=Off"
set_standard_digital_out(4, False)
$ 574 "Wait: 0.1"
sleep(0.1)
$ 575 "MoveJ"
$ 576 "OutUp180" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUp180_p, get_tcp_offset()), qnear=OutUp180_q),
a=1.3962634015954636, v=1.0471975511965976)
$ 577 "OutUptoFront" "breakAfter"
movej(get_inverse_kin(pose_trans(OutUptoFront_p, get_tcp_offset()),
qnear=OutUptoFront_q), t=1.2, r=0.05)
$ 578 "OutFront" "breakAfter"
movej(get_inverse_kin(pose_trans(OutFront_p, get_tcp_offset()), qnear=OutFront_q),
a=1.8325957145940461, v=1.3962634015954636, r=0.01)
$ 579 "MoveL"
$ 580 "InFront" "breakAfter"
movel(pose_trans(InFront_p, get_tcp_offset()), a=1.2, v=0.25)
$ 581 "Set DO[4]=On"
set_standard_digital_out(4, True)
$ 582 "Wait: 0.1"
sleep(0.1)
$ 583 "MoveL"
$ 584 "AboveFront" "breakAfter"
movel(pose_trans(AboveFront_p, get_tcp_offset()), a=1.2, v=0.25, r=0.01)
$ 585 "MoveJ"
$ 586 "ToCamera3" "breakAfter"
movej(get_inverse_kin(ToCamera3_p, qnear=ToCamera3_q), t=1.5)
$ 587 "Wait: 1.4"
sleep(1.4)
$ 588 "ToCamera4" "breakAfter"
movej(get_inverse_kin(ToCamera4_p, qnear=ToCamera4_q), t=1.2)
$ 589 "Wait: 1.5"
sleep(1.5)
```

```
  $ 590 "MoveJ"
  $ 591 "OutUp" "breakAfter"
  movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.8,
r=0.02)
  $ 592 "MoveL"
  $ 593 "InUp" "breakAfter"
  movel(pose_trans(InUp_p, get_tcp_offset()), a=1.2, v=0.25)
  $ 594 "Set DO[4]=Off"
  set_standard_digital_out(4, False)
  $ 595 "Wait: 0.1"
  sleep(0.1)
  $ 596 "MoveJ"
  $ 597 "OutUp" "breakAfter"
  movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q),
a=1.3962634015954636, v=1.0471975511965976)
 end
 while (True):
  $ 1 "Robot Program"
  $ 2 "MoveJ"
  $ 3 "OutUp" "breakAfter"
  movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.2)
  $ 4 "Set DO[4]=Off"
  set_standard_digital_out(4, False)
  $ 5 "Call Camara"
  Camara()
  $ 6 "Loop var_1≟ False "
  while (var_1 ==  False ):
   $ 7 "var_1≔socket_open('192.168.1.106',30000)"
   global var_1=socket_open("192.168.1.106",30000)
   $ 8 "Wait: 0.5"
   sleep(0.5)
  end
  $ 9 "socket_send_string('asking_for_data')"
  socket_send_string("asking_for_data")
  $ 10 "Wait: 0.5"
```

```
sleep(0.5)
$ 11 "var_2:=socket_read_ascii_float(25)"
global var_2=socket_read_ascii_float(25)
$ 12 "Wait: 0.5"
sleep(0.5)
$ 13 "var_1:= False "
global var_1=  False
$ 14 "socket_close()"
socket_close()
$ 15 "Wait: 1.0"
sleep(1.0)
$ 16 "Script: Script"
variable=var_2
n = length(variable)
i=1
while i<n:

    if variable[i]==11:
        U()
    elif variable[i]==12:
        U2()
    elif variable[i]==13:
        U3()
    elif variable[i]==21:
        F()
    elif variable[i]==22:
        F2()
    elif variable[i]==23:
        F3()
    elif variable[i]==31:
        R()
    elif variable[i]==32:
        R2()
    elif variable[i]==33:
```

```
                R3()
            elif variable[i]==41:
                B()
            elif variable[i]==42:
                B2()
            elif variable[i]==43:
                B3()
            elif variable[i]==51:
                L()
            elif variable[i]==52:
                L2()
            elif variable[i]==53:
                L3()
            elif variable[i]==61:
                D()
            elif variable[i]==62:
                D2()
            elif variable[i]==63:
                D3()
            elif variable[i]==0:

            end

            i=i+1
    end
    $ 17 "MoveJ"
    $ 18 "OutUp" "breakAfter"
    movej(get_inverse_kin(pose_trans(OutUp_p, get_tcp_offset()), qnear=OutUp_q), t=1.2)
    $ 19 "Wait variable_fin==1"
    while (not(variable_fin==1)):
      sync()
    end
  end
end
```

Archivo añadido a la carpeta src del programa de Python, *adaptation.py*:

```
import numpy as np
cubo = np.array([3, 2, 6, 4, 1, 5])
U = np.array([[0,0,0,0,0,1],[0,0,0,1,0,0],[0,0,1,0,0,0],[0,1,0,0,0,0],[0,0,0,0,1,0],[1,0,0,0,0,0]])
U2 = np.array([[0,0,0,0,0,1],[0,0,1,0,0,0],[0,1,0,0,0,0],[0,0,0,0,1,0],[0,0,0,1,0,0],[1,0,0,0,0,0]])
U3 = np.array([[0,0,0,0,0,1],[0,1,0,0,0,0],[0,0,0,0,1,0],[0,0,0,1,0,0],[0,0,1,0,0,0],[1,0,0,0,0,0]])
F = np.array([[0,0,0,1,0,0],[0,0,0,0,0,1],[0,0,0,0,1,0],[1,0,0,0,0,0],[0,0,1,0,0,0],[0,1,0,0,0,0]])
F2 = np.array([[0,0,0,1,0,0],[0,0,0,0,1,0],[1,0,0,0,0,0],[0,0,1,0,0,0],[0,0,0,0,0,1],[0,1,0,0,0,0]])
F3 = np.array([[0,0,0,1,0,0],[1,0,0,0,0,0],[0,0,1,0,0,0],[0,0,0,0,0,1],[0,0,0,0,1,0],[0,1,0,0,0,0]])
R = np.array([[0,0,0,0,1,0],[0,0,0,0,0,1],[0,1,0,0,0,0],[1,0,0,0,0,0],[0,0,0,1,0,0],[0,0,1,0,0,0]])
R2 = np.array([[0,0,0,0,1,0],[0,1,0,0,0,0],[1,0,0,0,0,0],[0,0,0,1,0,0],[0,0,0,0,0,1],[0,0,1,0,0,0]])
R3 = np.array([[0,0,0,0,1,0],[1,0,0,0,0,0],[0,0,0,1,0,0],[0,0,0,0,0,1],[0,1,0,0,0,0],[0,0,1,0,0,0]])
B = np.array([[0,1,0,0,0,0],[0,0,0,0,0,1],[0,0,1,0,0,0],[1,0,0,0,0,0],[0,0,0,0,1,0],[0,0,0,1,0,0]])
B2 = np.array([[0,1,0,0,0,0],[0,0,1,0,0,0],[1,0,0,0,0,0],[0,0,0,0,1,0],[0,0,0,0,0,1],[0,0,0,1,0,0]])
B3 = np.array([[0,1,0,0,0,0],[1,0,0,0,0,0],[0,0,0,0,1,0],[0,0,0,0,0,1],[0,0,1,0,0,0],[0,0,0,1,0,0]])
L = np.array([[0,0,1,0,0,0],[0,0,0,0,0,1],[0,0,0,1,0,0],[1,0,0,0,0,0],[0,1,0,0,0,0],[0,0,0,0,1,0]])
L2 = np.array([[0,0,1,0,0,0],[0,0,0,1,0,0],[1,0,0,0,0,0],[0,1,0,0,0,0],[0,0,0,0,0,1],[0,0,0,0,1,0]])
L3 = np.array([[0,0,1,0,0,0],[1,0,0,0,0,0],[0,1,0,0,0,0],[0,0,0,0,0,1],[0,0,0,1,0,0],[0,0,0,0,1,0]])
D = np.array([[1,0,0,0,0,0],[0,0,1,0,0,0],[0,0,0,1,0,0],[0,0,0,0,1,0],[0,1,0,0,0,0],[0,0,0,0,0,1]])
D2 = np.array([[1,0,0,0,0,0],[0,0,0,1,0,0],[0,0,0,0,1,0],[0,1,0,0,0,0],[0,0,1,0,0,0],[0,0,0,0,0,1]])
D3 = np.array([[1,0,0,0,0,0],[0,0,0,0,1,0],[0,1,0,0,0,0],[0,0,1,0,0,0],[0,0,0,1,0,0],[0,0,0,0,0,1]])

MovRobot=""
f=open("Movimientos.txt","r")
Mov=f.read()
f.close()
#print(Mov)
Mov=Mov.replace("'", "3")
Mov=Mov.split()
n=len(Mov)

j=0
for i in Mov:
```

```
# Up
    if Mov[j] == 'U':
        if cubo[0] == 1:
            MovRobot = MovRobot + "11,"
            cubo = U @ cubo
        elif cubo[1] == 1:
            MovRobot = MovRobot + "21,"
            cubo = F @ cubo
        elif cubo[2] == 1:
            MovRobot = MovRobot + "31,"
            cubo = R @ cubo
        elif cubo[3] == 1:
            MovRobot = MovRobot + "41,"
            cubo = B @ cubo
        elif cubo[4] == 1:
            MovRobot = MovRobot + "51,"
            cubo = L @ cubo
        elif cubo[5] == 1:
            MovRobot = MovRobot + "61,"
            cubo = D @ cubo

    elif Mov[j] == 'U2':
        if cubo[0] == 1:
            MovRobot = MovRobot + "12,"
            cubo = U2 @ cubo
        elif cubo[1] == 1:
            MovRobot = MovRobot + "22,"
            cubo = F2 @ cubo
        elif cubo[2] == 1:
            MovRobot = MovRobot + "32,"
            cubo = R2 @ cubo
        elif cubo[3] == 1:
            MovRobot = MovRobot + "42,"
            cubo = B2 @ cubo
```

```
            elif cubo[4] == 1:
                MovRobot = MovRobot + "52,"
                cubo = L2 @ cubo
            elif cubo[5] == 1:
                MovRobot = MovRobot + "62,"
                cubo = D2 @ cubo

        elif Mov[j] == 'U3':
            if cubo[0] == 1:
                MovRobot = MovRobot + "13,"
                cubo = U3 @ cubo
            elif cubo[1] == 1:
                MovRobot = MovRobot + "23,"
                cubo = F3 @ cubo
            elif cubo[2] == 1:
                MovRobot = MovRobot + "33,"
                cubo = R3 @ cubo
            elif cubo[3] == 1:
                MovRobot = MovRobot + "43,"
                cubo = B3 @ cubo
            elif cubo[4] == 1:
                MovRobot = MovRobot + "53,"
                cubo = L3 @ cubo
            elif cubo[5] == 1:
                MovRobot = MovRobot + "63,"
                cubo = D3 @ cubo

    # Front
        elif Mov[j] == 'F':
            if cubo[0] == 2:
                MovRobot = MovRobot + "11,"
                cubo = U @ cubo
            elif cubo[1] == 2:
                MovRobot = MovRobot + "21,"
```

```
            cubo = F @ cubo
        elif cubo[2] == 2:
            MovRobot = MovRobot + "31,"
            cubo = R @ cubo
        elif cubo[3] == 2:
            MovRobot = MovRobot + "41,"
            cubo = B @ cubo
        elif cubo[4] == 2:
            MovRobot = MovRobot + "51,"
            cubo = L @ cubo
        elif cubo[5] == 2:
            MovRobot = MovRobot + "61,"
            cubo = D @ cubo

    elif Mov[j] == 'F2':
        if cubo[0] == 2:
            MovRobot = MovRobot + "12,"
            cubo = U2 @ cubo
        elif cubo[1] == 2:
            MovRobot = MovRobot + "22,"
            cubo = F2 @ cubo
        elif cubo[2] == 2:
            MovRobot = MovRobot + "32,"
            cubo = R2 @ cubo
        elif cubo[3] == 2:
            MovRobot = MovRobot + "42,"
            cubo = B2 @ cubo
        elif cubo[4] == 2:
            MovRobot = MovRobot + "52,"
            cubo = L2 @ cubo
        elif cubo[5] == 2:
            MovRobot = MovRobot + "62,"
            cubo = D2 @ cubo
```

```
    elif Mov[j] == 'F3':
        if cubo[0] == 2:
            MovRobot = MovRobot + "13,"
            cubo = U3 @ cubo
        elif cubo[1] == 2:
            MovRobot = MovRobot + "23,"
            cubo = F3 @ cubo
        elif cubo[2] == 2:
            MovRobot = MovRobot + "33,"
            cubo = R3 @ cubo
        elif cubo[3] == 2:
            MovRobot = MovRobot + "43,"
            cubo = B3 @ cubo
        elif cubo[4] == 2:
            MovRobot = MovRobot + "53,"
            cubo = L3 @ cubo
        elif cubo[5] == 2:
            MovRobot = MovRobot + "63,"
            cubo = D3 @ cubo

# Right
    elif Mov[j] == 'R':
        if cubo[0] == 3:
            MovRobot = MovRobot + "11,"
            cubo = U @ cubo
        elif cubo[1] == 3:
            MovRobot = MovRobot + "21,"
            cubo = F @ cubo
        elif cubo[2] == 3:
            MovRobot = MovRobot + "31,"
            cubo = R @ cubo
        elif cubo[3] == 3:
            MovRobot = MovRobot + "41,"
            cubo = B @ cubo
```

```
        elif cubo[4] == 3:
            MovRobot = MovRobot + "51,"
            cubo = L @ cubo
        elif cubo[5] == 3:
            MovRobot = MovRobot + "61,"
            cubo = D @ cubo

    elif Mov[j] == 'R2':
        if cubo[0] == 3:
            MovRobot = MovRobot + "12,"
            cubo = U2 @ cubo
        elif cubo[1] == 3:
            MovRobot = MovRobot + "22,"
            cubo = F2 @ cubo
        elif cubo[2] == 3:
            MovRobot = MovRobot + "32,"
            cubo = R2 @ cubo
        elif cubo[3] == 3:
            MovRobot = MovRobot + "42,"
            cubo = B2 @ cubo
        elif cubo[4] == 3:
            MovRobot = MovRobot + "52,"
            cubo = L2 @ cubo
        elif cubo[5] == 3:
            MovRobot = MovRobot + "62,"
            cubo = D2 @ cubo

    elif Mov[j] == 'R3':
        if cubo[0] == 3:
            MovRobot = MovRobot + "13,"
            cubo = U3 @ cubo
        elif cubo[1] == 3:
            MovRobot = MovRobot + "23,"
            cubo = F3 @ cubo
```

```
        elif cubo[2] == 3:
            MovRobot = MovRobot + "33,"
            cubo = R3 @ cubo
        elif cubo[3] == 3:
            MovRobot = MovRobot + "43,"
            cubo = B3 @ cubo
        elif cubo[4] == 3:
            MovRobot = MovRobot + "53,"
            cubo = L3 @ cubo
        elif cubo[5] == 3:
            MovRobot = MovRobot + "63,"
            cubo = D3 @ cubo

    # Back
    elif Mov[j] == 'B':
        if cubo[0] == 4:
            MovRobot = MovRobot + "11,"
            cubo = U @ cubo
        elif cubo[1] == 4:
            MovRobot = MovRobot + "21,"
            cubo = F @ cubo
        elif cubo[2] == 4:
            MovRobot = MovRobot + "31,"
            cubo = R @ cubo
        elif cubo[3] == 4:
            MovRobot = MovRobot + "41,"
            cubo = B @ cubo
        elif cubo[4] == 4:
            MovRobot = MovRobot + "51,"
            cubo = L @ cubo
        elif cubo[5] == 4:
            MovRobot = MovRobot + "61,"
            cubo = D @ cubo
```

```
elif Mov[j] == 'B2':
    if cubo[0] == 4:
        MovRobot = MovRobot + "12,"
        cubo = U2 @ cubo
    elif cubo[1] == 4:
        MovRobot = MovRobot + "22,"
        cubo = F2 @ cubo
    elif cubo[2] == 4:
        MovRobot = MovRobot + "32,"
        cubo = R2 @ cubo
    elif cubo[3] == 4:
        MovRobot = MovRobot + "42,"
        cubo = B2 @ cubo
    elif cubo[4] == 4:
        MovRobot = MovRobot + "52,"
        cubo = L2 @ cubo
    elif cubo[5] == 4:
        MovRobot = MovRobot + "62,"
        cubo = D2 @ cubo

elif Mov[j] == 'B3':
    if cubo[0] == 4:
        MovRobot = MovRobot + "13,"
        cubo = U3 @ cubo
    elif cubo[1] == 4:
        MovRobot = MovRobot + "23,"
        cubo = F3 @ cubo
    elif cubo[2] == 4:
        MovRobot = MovRobot + "33,"
        cubo = R3 @ cubo
    elif cubo[3] == 4:
        MovRobot = MovRobot + "43,"
        cubo = B3 @ cubo
    elif cubo[4] == 4:
```

```
            MovRobot = MovRobot + "53,"
            cubo = L3 @ cubo
        elif cubo[5] == 4:
            MovRobot = MovRobot + "63,"
            cubo = D3 @ cubo

# Left
    elif Mov[j] == 'L':
        if cubo[0] == 5:
            MovRobot = MovRobot + "11,"
            cubo = U @ cubo
        elif cubo[1] == 5:
            MovRobot = MovRobot + "21,"
            cubo = F @ cubo
        elif cubo[2] == 5:
            MovRobot = MovRobot + "31,"
            cubo = R @ cubo
        elif cubo[3] == 5:
            MovRobot = MovRobot + "41,"
            cubo = B @ cubo
        elif cubo[4] == 5:
            MovRobot = MovRobot + "51,"
            cubo = L @ cubo
        elif cubo[5] == 5:
            MovRobot = MovRobot + "61,"
            cubo = D @ cubo

    elif Mov[j] == 'L2':
        if cubo[0] == 5:
            MovRobot = MovRobot + "12,"
            cubo = U2 @ cubo
        elif cubo[1] == 5:
            MovRobot = MovRobot + "22,"
            cubo = F2 @ cubo
```

```
        elif cubo[2] == 5:
            MovRobot = MovRobot + "32,"
            cubo = R2 @ cubo
        elif cubo[3] == 5:
            MovRobot = MovRobot + "42,"
            cubo = B2 @ cubo
        elif cubo[4] == 5:
            MovRobot = MovRobot + "52,"
            cubo = L2 @ cubo
        elif cubo[5] == 5:
            MovRobot = MovRobot + "62,"
            cubo = D2 @ cubo

    elif Mov[j] == 'L3':
        if cubo[0] == 5:
            MovRobot = MovRobot + "13,"
            cubo = U3 @ cubo
        elif cubo[1] == 5:
            MovRobot = MovRobot + "23,"
            cubo = F3 @ cubo
        elif cubo[2] == 5:
            MovRobot = MovRobot + "33,"
            cubo = R3 @ cubo
        elif cubo[3] == 5:
            MovRobot = MovRobot + "43,"
            cubo = B3 @ cubo
        elif cubo[4] == 5:
            MovRobot = MovRobot + "53,"
            cubo = L3 @ cubo
        elif cubo[5] == 5:
            MovRobot = MovRobot + "63,"
            cubo = D3 @ cubo

# Down
```

```
elif Mov[j] == 'D':
    if cubo[0] == 6:
        MovRobot = MovRobot + "11,"
        cubo = U @ cubo
    elif cubo[1] == 6:
        MovRobot = MovRobot + "21,"
        cubo = F @ cubo
    elif cubo[2] == 6:
        MovRobot = MovRobot + "31,"
        cubo = R @ cubo
    elif cubo[3] == 6:
        MovRobot = MovRobot + "41,"
        cubo = B @ cubo
    elif cubo[4] == 6:
        MovRobot = MovRobot + "51,"
        cubo = L @ cubo
    elif cubo[5] == 6:
        MovRobot = MovRobot + "61,"
        cubo = D @ cubo

elif Mov[j] == 'D2':
    if cubo[0] == 6:
        MovRobot = MovRobot + "12,"
        cubo = U2 @ cubo
    elif cubo[1] == 6:
        MovRobot = MovRobot + "22,"
        cubo = F2 @ cubo
    elif cubo[2] == 6:
        MovRobot = MovRobot + "32,"
        cubo = R2 @ cubo
    elif cubo[3] == 6:
        MovRobot = MovRobot + "42,"
        cubo = B2 @ cubo
    elif cubo[4] == 6:
```

```
            MovRobot = MovRobot + "52,"
            cubo = L2 @ cubo
        elif cubo[5] == 6:
            MovRobot = MovRobot + "62,"
            cubo = D2 @ cubo

    elif Mov[j] == 'D3':
        if cubo[0] == 6:
            MovRobot = MovRobot + "13,"
            cubo = U3 @ cubo
        elif cubo[1] == 6:
            MovRobot = MovRobot + "23,"
            cubo = F3 @ cubo
        elif cubo[2] == 6:
            MovRobot = MovRobot + "33,"
            cubo = R3 @ cubo
        elif cubo[3] == 6:
            MovRobot = MovRobot + "43,"
            cubo = B3 @ cubo
        elif cubo[4] == 6:
            MovRobot = MovRobot + "53,"
            cubo = L3 @ cubo
        elif cubo[5] == 6:
            MovRobot = MovRobot + "63,"
            cubo = D3 @ cubo

    j=j+1

#Fin del cambio de la secuencia

#Adaptando el resultado para poder codificar y enviar al robot

x=25-n
MovRobot=MovRobot.rstrip(',')
```

```
MovRobotPrint=MovRobot.replace(",", " ")
print("Adapted Solution: "+str(MovRobotPrint))
print("Final orientation of the cube: "+ str(cubo))
# Paso 1: Separar el string en una lista
lista = MovRobot.split(',')

# Paso 2: Añadir x repeticiones de "0" a la lista
lista.extend(['0'] * x)

# Paso 3: Unir los elementos de la lista en un solo string
MovRobot = ','.join(lista)

# Paso 4: Agregar paréntesis al principio y al final del string
MovRobot = '(' + MovRobot + ')'

#Comienza la conexión
import socket
import time

HOST = "192.168.1.106"
PORT = 30000

print("Starting connection")
count = 0


while (count < 1):

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind((HOST, PORT))  # Bind to the port
    s.listen(5)  # Now wait for client connection.
    c, addr = s.accept()  # Establish connection with client.
    print("Connected")
```

```
    try:
        msg = c.recv(1024)
        print(msg)
        time.sleep(0.5)

        # Enviar los datos a través del socket
        c.sendall(MovRobot.encode('utf-8'))
        count=1
    except socket.error as socketerror:
        print("error de conexión")

c.close()
s.close()
print("Finished connection and start resolution")
```

Archivo modificado de la carpeta src del programa de Python, *video.py*:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# vim: fenc=utf-8 ts=4 sw=4 et
import socket


import cv2
from colordetection import color_detector
from config import config
from helpers import get_next_locale
import i18n
from PIL import ImageFont, ImageDraw, Image
import numpy as np
from constants import (
    COLOR_PLACEHOLDER,
    LOCALES,
    ROOT_DIR,
    CUBE_PALETTE,
    MINI_STICKER_AREA_TILE_SIZE,
    MINI_STICKER_AREA_TILE_GAP,
    MINI_STICKER_AREA_OFFSET,
    STICKER_AREA_TILE_SIZE,
    STICKER_AREA_TILE_GAP,
    STICKER_AREA_OFFSET,
    STICKER_CONTOUR_COLOR,
    CALIBRATE_MODE_KEY,
    SWITCH_LANGUAGE_KEY,
    TEXT_SIZE,
    E_INCORRECTLY_SCANNED,
    E_ALREADY_SOLVED
)
import time


class Webcam:
```

```
i=0
def __init__(self):
    print('Starting webcam... (this might take a while, please be patient)')
    self.cam = cv2.VideoCapture(2)
    print('Webcam successfully started')

    #self.cam.set(cv2.CAP_PROP_ZOOM,1)

    self.colors_to_calibrate = ['green', 'red', 'blue', 'orange', 'white', 'yellow']
    self.average_sticker_colors = {}
    self.result_state = {}

    self.snapshot_state = [(255,255,255), (255,255,255), (255,255,255),
                           (255,255,255), (255,255,255), (255,255,255),
                           (255,255,255), (255,255,255), (255,255,255)]
    self.preview_state = [(255,255,255), (255,255,255), (255,255,255),
                          (255,255,255), (255,255,255), (255,255,255),
                          (255,255,255), (255,255,255), (255,255,255)]

    self.cam.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
    self.cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
    self.width = int(self.cam.get(cv2.CAP_PROP_FRAME_WIDTH))
    self.height = int(self.cam.get(cv2.CAP_PROP_FRAME_HEIGHT))

    self.calibrate_mode = False
    #self.calibrated_colors = {}
    self.calibrated_colors = {
        'green': (206.46275, 214.83139, 60.609806),
        'blue': (129.4989, 121.357605, 203.6739),
        'red': (255.0, 173.78572, 67.5014),
        'orange': (124.50255, 160.29082, 214.07907),
        'white': (254.14253, 250.87357, 235.10115),
        'yellow': (177.67188, 232.62674, 227.67188)
    }
```

```
        self.current_color_to_calibrate_index = 0
        self.done_calibrating = False
        self.count=0
        self.sides=0


    def draw_stickers(self, stickers, offset_x, offset_y):
        """Draws the given stickers onto the given frame."""
        index = -1
        for row in range(3):
            for col in range(3):
                index += 1
                x1 = (offset_x + STICKER_AREA_TILE_SIZE * col) + STICKER_AREA_TILE_GAP *
col
                y1 = (offset_y + STICKER_AREA_TILE_SIZE * row) + STICKER_AREA_TILE_GAP *
row
                x2 = x1 +STICKER_AREA_TILE_SIZE
                y2 = y1 + STICKER_AREA_TILE_SIZE

                # shadow
                cv2.rectangle(
                    self.frame,
                    (x1, y1),
                    (x2, y2),
                    (0, 0, 0),
                    -1
                )

                # foreground color
                cv2.rectangle(
                    self.frame,
                    (x1 + 1, y1 + 1),
                    (x2 - 1, y2 - 1),
                    color_detector.get_prominent_color(stickers[index]),
                    -1
```

```
            )


    def draw_preview_stickers(self):
        """Draw the current preview state onto the given frame."""
        self.draw_stickers(self.preview_state, STICKER_AREA_OFFSET,
STICKER_AREA_OFFSET)


    def draw_snapshot_stickers(self):
        """Draw the current snapshot state onto the given frame."""
        y = STICKER_AREA_TILE_SIZE * 3 + STICKER_AREA_TILE_GAP * 2 +
STICKER_AREA_OFFSET * 2
        self.draw_stickers(self.snapshot_state, STICKER_AREA_OFFSET, y)


    def find_contours(self, dilatedFrame):
        """Find the contours of a 3x3x3 cube."""
        contours, hierarchy = cv2.findContours(dilatedFrame, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
        final_contours = []


        # Step 1/4: filter all contours to only those that are square-ish shapes.
        for contour in contours:
            perimeter = cv2.arcLength(contour, True)
            approx = cv2.approxPolyDP(contour, 0.1 * perimeter, True)
            if len (approx) == 4:
                area = cv2.contourArea(contour)
                (x, y, w, h) = cv2.boundingRect(approx)


                # Find aspect ratio of boundary rectangle around the countours.
                ratio = w / float(h)


                # Check if contour is close to a square.
                if ratio >= 0.8 and ratio <= 1.2 and w >= 30 and w <= 60 and area / (w * h) > 0.4:
                    final_contours.append((x, y, w, h))


        # Return early if we didn't found 9 or more contours.
```

```
if len(final_contours) < 9:
    return []

# Step 2/4: Find the contour that has 9 neighbors (including itself)
# and return all of those neighbors.
found = False
contour_neighbors = {}
for index, contour in enumerate(final_contours):
    (x, y, w, h) = contour
    contour_neighbors[index] = []
    center_x = x + w / 2
    center_y = y + h / 2
    radius = 1.5

    # Create 9 positions for the current contour which are the
    # neighbors. We'll use this to check how many neighbors each contour
    # has. The only way all of these can match is if the current contour
    # is the center of the cube. If we found the center, we also know
    # all the neighbors, thus knowing all the contours and thus knowing
    # this shape can be considered a 3x3x3 cube. When we've found those
    # contours, we sort them and return them.
    neighbor_positions = [
        # top left
        [(center_x - w * radius), (center_y - h * radius)],

        # top middle
        [center_x, (center_y - h * radius)],

        # top right
        [(center_x + w * radius), (center_y - h * radius)],

        # middle left
        [(center_x - w * radius), center_y],
```

```
            # center
            [center_x, center_y],

            # middle right
            [(center_x + w * radius), center_y],

            # bottom left
            [(center_x - w * radius), (center_y + h * radius)],

            # bottom middle
            [center_x, (center_y + h * radius)],

            # bottom right
            [(center_x + w * radius), (center_y + h * radius)],
        ]

        for neighbor in final_contours:
            (x2, y2, w2, h2) = neighbor
            for (x3, y3) in neighbor_positions:
                # The neighbor_positions are located in the center of each
                # contour instead of top-left corner.
                # logic: (top left < center pos) and (bottom right > center pos)
                if (x2 < x3 and y2 < y3) and (x2 + w2 > x3 and y2 + h2 > y3):
                    contour_neighbors[index].append(neighbor)

    # Step 3/4: Now that we know how many neighbors all contours have, we'll
    # loop over them and find the contour that has 9 neighbors, which
    # includes itself. This is the center piece of the cube. If we come
    # across it, then the 'neighbors' are actually all the contours we're
    # looking for.
    for (contour, neighbors) in contour_neighbors.items():
        if len(neighbors) == 9:
            found = True
            final_contours = neighbors
```

```
            break

    if not found:
        return []

    # Step 4/4: When we reached this part of the code we found a cube-like
    # contour. The code below will sort all the contours on their X and Y
    # values from the top-left to the bottom-right.

    # Sort contours on the y-value first.
    y_sorted = sorted(final_contours, key=lambda item: item[1])

    # Split into 3 rows and sort each row on the x-value.
    top_row = sorted(y_sorted[0:3], key=lambda item: item[0])
    middle_row = sorted(y_sorted[3:6], key=lambda item: item[0])
    bottom_row = sorted(y_sorted[6:9], key=lambda item: item[0])

    sorted_contours = top_row + middle_row + bottom_row
    return sorted_contours

def scanned_successfully(self):
    """Validate if the user scanned 9 colors for each side."""
    color_count = {}
    for side, preview in self.result_state.items():
        for bgr in preview:
            key = str(bgr)
            if key not in color_count:
                color_count[key] = 1
            else:
                color_count[key] = color_count[key] + 1
    invalid_colors = [k for k, v in color_count.items() if v != 9]
    return len(invalid_colors) == 0

def draw_contours(self, contours):
```

```
        """Draw contours onto the given frame."""
        if self.calibrate_mode:
            # Only show the center piece contour.
            (x, y, w, h) = contours[4]
            cv2.rectangle(self.frame, (x, y), (x + w, y + h), STICKER_CONTOUR_COLOR, 2)
        else:
            for index, (x, y, w, h) in enumerate(contours):
                cv2.rectangle(self.frame, (x, y), (x + w, y + h), STICKER_CONTOUR_COLOR, 2)
            #print("Aqui a detectado algo")
            self.count+=1
            #print(self.count)


    def update_preview_state(self, contours):
        """
        Get the average color value for the contour for every X amount of frames
        to prevent flickering and more precise results.
        """
        max_average_rounds = 8
        for index, (x, y, w, h) in enumerate(contours):
            if index in self.average_sticker_colors and len(self.average_sticker_colors[index]) ==
max_average_rounds:
                sorted_items = {}
                for bgr in self.average_sticker_colors[index]:
                    key = str(bgr)
                    if key in sorted_items:
                        sorted_items[key] += 1
                    else:
                        sorted_items[key] = 1
                most_common_color = max(sorted_items, key=lambda i: sorted_items[i])
                self.average_sticker_colors[index] = []
                self.preview_state[index] = eval(most_common_color)
                break


            roi = self.frame[y+7:y+h-7, x+14:x+w-14]
            avg_bgr = color_detector.get_dominant_color(roi)
```

```
            closest_color = color_detector.get_closest_color(avg_bgr)['color_bgr']
            self.preview_state[index] = closest_color
            if index in self.average_sticker_colors:
                self.average_sticker_colors[index].append(closest_color)
            else:
                self.average_sticker_colors[index] = [closest_color]


    def update_snapshot_state(self):
        """Update the snapshot state based on the current preview state."""
        self.snapshot_state = list(self.preview_state)
        center_color_name =
color_detector.get_closest_color(self.snapshot_state[4])['color_name']
        self.result_state[center_color_name] = self.snapshot_state
        self.draw_snapshot_stickers()
        #print("Aqui se ha hecho la captura")
        self.count=0
        time.sleep(1)
        self.sides+=1




    def get_font(self, size=TEXT_SIZE):
        """Load the truetype font with the specified text size."""
        font_path = '{}/assets/arial-unicode-ms.ttf'.format(ROOT_DIR)
        return ImageFont.truetype(font_path, size)


    def render_text(self, text, pos, color=(255, 255, 255), size=TEXT_SIZE, anchor='lt'):
        """
        Render text with a shadow using the pillow module.
        """
        font = self.get_font(size)


        # Convert opencv frame (np.array) to PIL Image array.
        frame = Image.fromarray(self.frame)
```

```
    # Draw the text onto the image.
    draw = ImageDraw.Draw(frame)
    draw.text(pos, text, font=font, fill=color, anchor=anchor,
              stroke_width=1, stroke_fill=(0, 0, 0))

    # Convert the pillow frame back to a numpy array.
    self.frame = np.array(frame)

def get_text_size(self, text, size=TEXT_SIZE):
    """Get text size based on the default freetype2 loaded font."""
    return self.get_font(size).getsize(text)

def draw_scanned_sides(self):
    """Display how many sides are scanned by the user."""
    text = i18n.t('scannedSides', num=len(self.result_state.keys()))
    self.render_text(text, (20, self.height - 20), anchor='lb')

def draw_current_color_to_calibrate(self):
    """Display the current side's color that needs to be calibrated."""
    offset_y = 20
    font_size = int(TEXT_SIZE * 1.25)
    if self.done_calibrating:
        messages = [
            i18n.t('calibratedSuccessfully'),
            i18n.t('quitCalibrateMode', keyValue=CALIBRATE_MODE_KEY),
        ]
        for index, text in enumerate(messages):
            _, textsize_height = self.get_text_size(text, font_size)
            y = offset_y + (textsize_height + 10) * index
            self.render_text(text, (int(self.width / 2), y), size=font_size, anchor='mt')
    else:
        current_color = self.colors_to_calibrate[self.current_color_to_calibrate_index]
        text = i18n.t('currentCalibratingSide.{}'.format(current_color))
```

```
            self.render_text(text, (int(self.width / 2), offset_y), size=font_size, anchor='mt')


    def draw_calibrated_colors(self):
        """Display all the colors that are calibrated while in calibrate mode."""
        #Aqui es donde calibra y guarda los valores en calibrated_color.item
        offset_y = 20
        for index, (color_name, color_bgr) in enumerate(self.calibrated_colors.items()):
            x1 = 90
            y1 = int(offset_y + STICKER_AREA_TILE_SIZE * index)
            x2 = x1 + STICKER_AREA_TILE_SIZE
            y2 = y1 + STICKER_AREA_TILE_SIZE

            # shadow
            cv2.rectangle(
                self.frame,
                (x1, y1),
                (x2, y2),
                (0, 0, 0),
                -1
            )

            # foreground
            cv2.rectangle(
                self.frame,
                (x1 + 1, y1 + 1),
                (x2 - 1, y2 - 1),
                tuple([int(c) for c in color_bgr]),
                -1
            )
            self.render_text(i18n.t(color_name), (20, y1 + STICKER_AREA_TILE_SIZE / 2 - 3),
anchor='lm')


    def reset_calibrate_mode(self):
        """Reset calibrate mode variables."""
        self.calibrated_colors = {}
```

```
    self.current_color_to_calibrate_index = 0
    self.done_calibrating = False


def draw_current_language(self):
    text = '{}: {}'.format(
        i18n.t('language'),
        LOCALES[config.get_setting('locale')]
    )
    offset = 20
    self.render_text(text, (self.width - offset, offset), anchor='rt')


def draw_2d_cube_state(self):
    """
    Create a 2D cube state visualization and draw the self.result_state.

    We're gonna display the visualization like so:
              -----
            | W W W |
            | W W W |
            | W W W |
      -----  -----  -----  -----
     | O O O | G G G | R R R | B B B |
     | O O O | G G G | R R R | B B B |
     | O O O | G G G | R R R | B B B |
      -----  -----  -----  -----
            | Y Y Y |
            | Y Y Y |
            | Y Y Y |
              -----
    So we're gonna make a 4x3 grid and hardcode where each side has to go.
    Based on the x and y in that 4x3 grid we can calculate its position.
    """
    grid = {
        'white' : [1, 0],
```

```
        'orange': [0, 1],
        'green' : [1, 1],
        'red'   : [2, 1],
        'blue'  : [3, 1],
        'yellow': [1, 2],
    }

    # The offset in-between each side (white, red, etc).
    side_offset = MINI_STICKER_AREA_TILE_GAP * 3

    # The size of 1 whole side (containing 9 stickers).
    side_size = MINI_STICKER_AREA_TILE_SIZE * 3 + MINI_STICKER_AREA_TILE_GAP *
2

    # The X and Y offset is placed in the bottom-right corner, minus the
    # whole size of the 4x3 grid, minus an additional offset.
    offset_x = self.width - (side_size * 4) - (side_offset * 3) - MINI_STICKER_AREA_OFFSET
    offset_y = self.height - (side_size * 3) - (side_offset * 2) - MINI_STICKER_AREA_OFFSET

    for side, (grid_x, grid_y) in grid.items():
        index = -1
        for row in range(3):
            for col in range(3):
                index += 1
                x1 = int(
                    (offset_x + MINI_STICKER_AREA_TILE_SIZE * col) +
                    (MINI_STICKER_AREA_TILE_GAP * col) +
                    ((side_size + side_offset) * grid_x)
                )
                y1 = int(
                    (offset_y + MINI_STICKER_AREA_TILE_SIZE * row) +
                    (MINI_STICKER_AREA_TILE_GAP * row) +
                    ((side_size + side_offset) * grid_y)
                )
                x2 = int(x1 + MINI_STICKER_AREA_TILE_SIZE)
```

```
                y2 = int(y1 + MINI_STICKER_AREA_TILE_SIZE)

                foreground_color = COLOR_PLACEHOLDER
                if side in self.result_state:
                    foreground_color =
color_detector.get_prominent_color(self.result_state[side][index])

                # shadow
                cv2.rectangle(
                    self.frame,
                    (x1, y1),
                    (x2, y2),
                    (0, 0, 0),
                    -1
                )

                # foreground color
                cv2.rectangle(
                    self.frame,
                    (x1 + 1, y1 + 1),
                    (x2 - 1, y2 - 1),
                    foreground_color,
                    -1
                )

    def get_result_notation(self):
        """Convert all the sides and their BGR colors to cube notation."""
        notation = dict(self.result_state)
        for side, preview in notation.items():
            for sticker_index, bgr in enumerate(preview):
                notation[side][sticker_index] = color_detector.convert_bgr_to_notation(bgr)

        # Join all the sides together into one single string.
        # Order must be URFDLB (white, red, green, yellow, orange, blue)
        combined = ''
```

```
    for side in ['white', 'red', 'green', 'yellow', 'orange', 'blue']:
        combined += ''.join(notation[side])
    return combined


def state_already_solved(self):
    """Find out if the cube hasn't been solved already."""
    for side in ['white', 'red', 'green', 'yellow', 'orange', 'blue']:
        # Get the center color of the current side.
        center_bgr = self.result_state[side][4]

        # Compare the center color to all neighbors. If we come across a
        # different color, then we can assume the cube isn't solved yet.
        for bgr in self.result_state[side]:
            if center_bgr != bgr:
                return False
    return True


def run(self):
    """
    Open up the webcam and present the user with the Qbr user interface.

    Returns a string of the scanned state in rubik's cube notation.
    """
    while True:
        _, frame = self.cam.read()
        self.frame = frame
        key = cv2.waitKey(10) & 0xff
        #27 es esc, 32 es espacio
        # Quit on escape.
        if key == 27:
            break
        if len(self.result_state.keys()) ==6:
            break
```

```
if not self.calibrate_mode:
    # Update the snapshot when space bar is pressed.
    # Aqui es donde se activa para hacer la captura
    if self.count == 20:
        self.update_snapshot_state()
    if key == 32:
        self.update_snapshot_state()

    # Switch to another language.
    if key == ord(SWITCH_LANGUAGE_KEY):
        next_locale = get_next_locale(config.get_setting('locale'))
        config.set_setting('locale', next_locale)
        i18n.set('locale', next_locale)

# Toggle calibrate mode.
if key == ord(CALIBRATE_MODE_KEY):
    self.reset_calibrate_mode()
    self.calibrate_mode = not self.calibrate_mode

grayFrame = cv2.cvtColor(self.frame, cv2.COLOR_BGR2GRAY)
blurredFrame = cv2.blur(grayFrame, (3, 3))
cannyFrame = cv2.Canny(blurredFrame, 30, 60, 3)
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 9))
dilatedFrame = cv2.dilate(cannyFrame, kernel)

contours = self.find_contours(dilatedFrame)
if len(contours) == 9:
    self.draw_contours(contours)
    if not self.calibrate_mode:
        self.update_preview_state(contours)
    elif key == 32 and self.done_calibrating is False:
        current_color = self.colors_to_calibrate[self.current_color_to_calibrate_index]
        (x, y, w, h) = contours[4]
        roi = self.frame[y+7:y+h-7, x+14:x+w-14]
```

```
                avg_bgr = color_detector.get_dominant_color(roi)
                self.calibrated_colors[current_color] = avg_bgr
                self.current_color_to_calibrate_index += 1
                self.done_calibrating = self.current_color_to_calibrate_index ==
len(self.colors_to_calibrate)
                if self.done_calibrating:
                    color_detector.set_cube_color_pallete(self.calibrated_colors)
                    config.set_setting(CUBE_PALETTE, color_detector.cube_color_palette)

            if self.calibrate_mode:
                self.draw_current_color_to_calibrate()
                self.draw_calibrated_colors()
            else:
                self.draw_current_language()
                self.draw_preview_stickers()
                self.draw_snapshot_stickers()
                self.draw_scanned_sides()
                self.draw_2d_cube_state()

            cv2.imshow("Qbr - Rubik's cube solver", self.frame)

        self.cam.release()
        cv2.destroyAllWindows()

        if len(self.result_state.keys()) != 6:
            return E_INCORRECTLY_SCANNED

        if not self.scanned_successfully():
            return E_INCORRECTLY_SCANNED

        if self.state_already_solved():
            return E_ALREADY_SOLVED
        return self.get_result_notation()

webcam = Webcam()
```

UNIVERSITAT POLITÉCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño y desarrollo de un sistema robotizado capaz de resolver un cubo de Rubik mediante un robot colaborativo y visión artificial

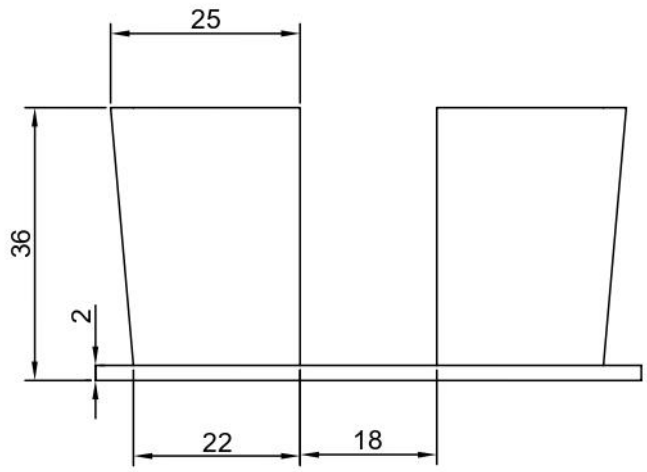
DOCUMENTO Nº2: Planos

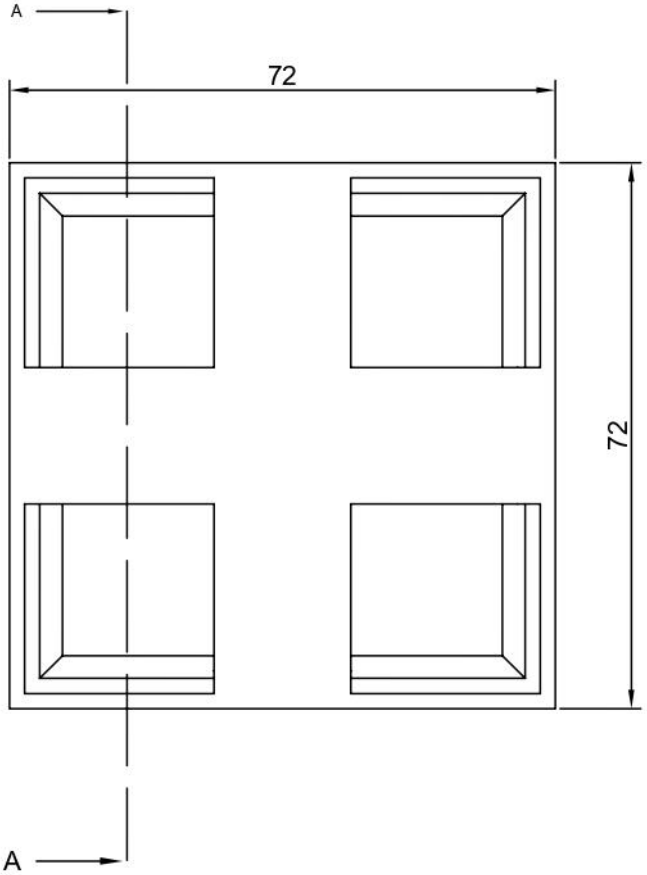
Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Górriz Aliaga, Abel

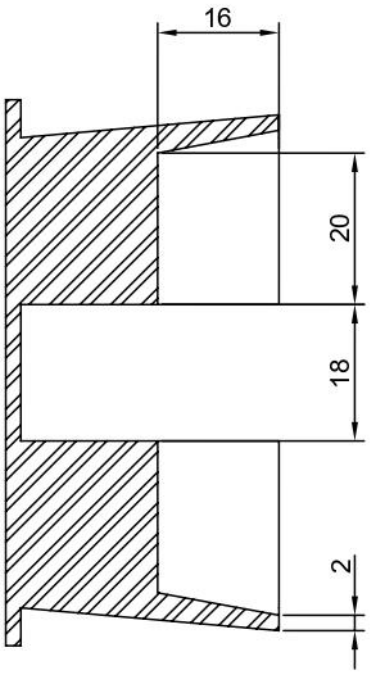
Tutor/a: Ivorra Martínez, Eugenio

CURSO ACADÉMICO:2022-2023



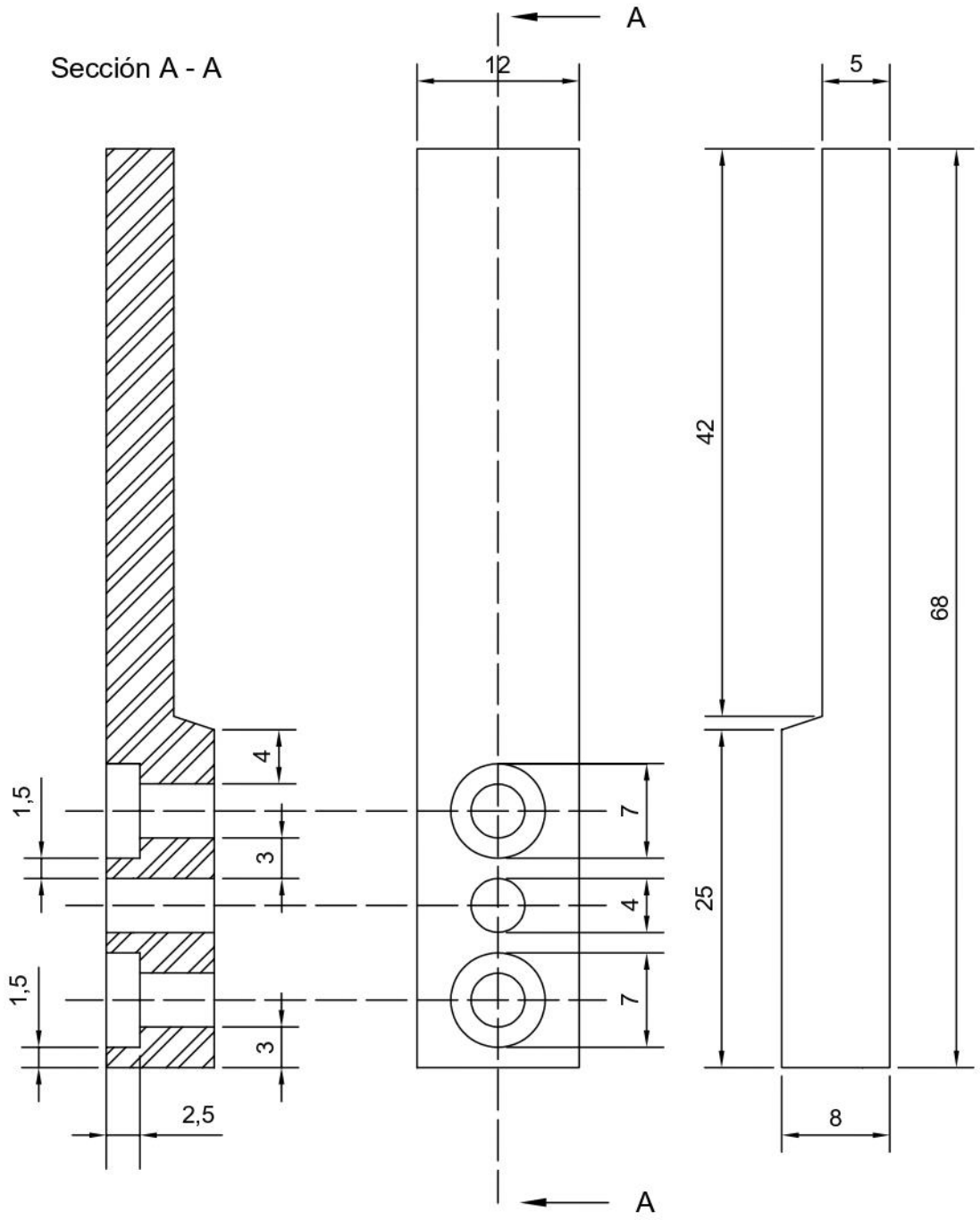


Sección A - A



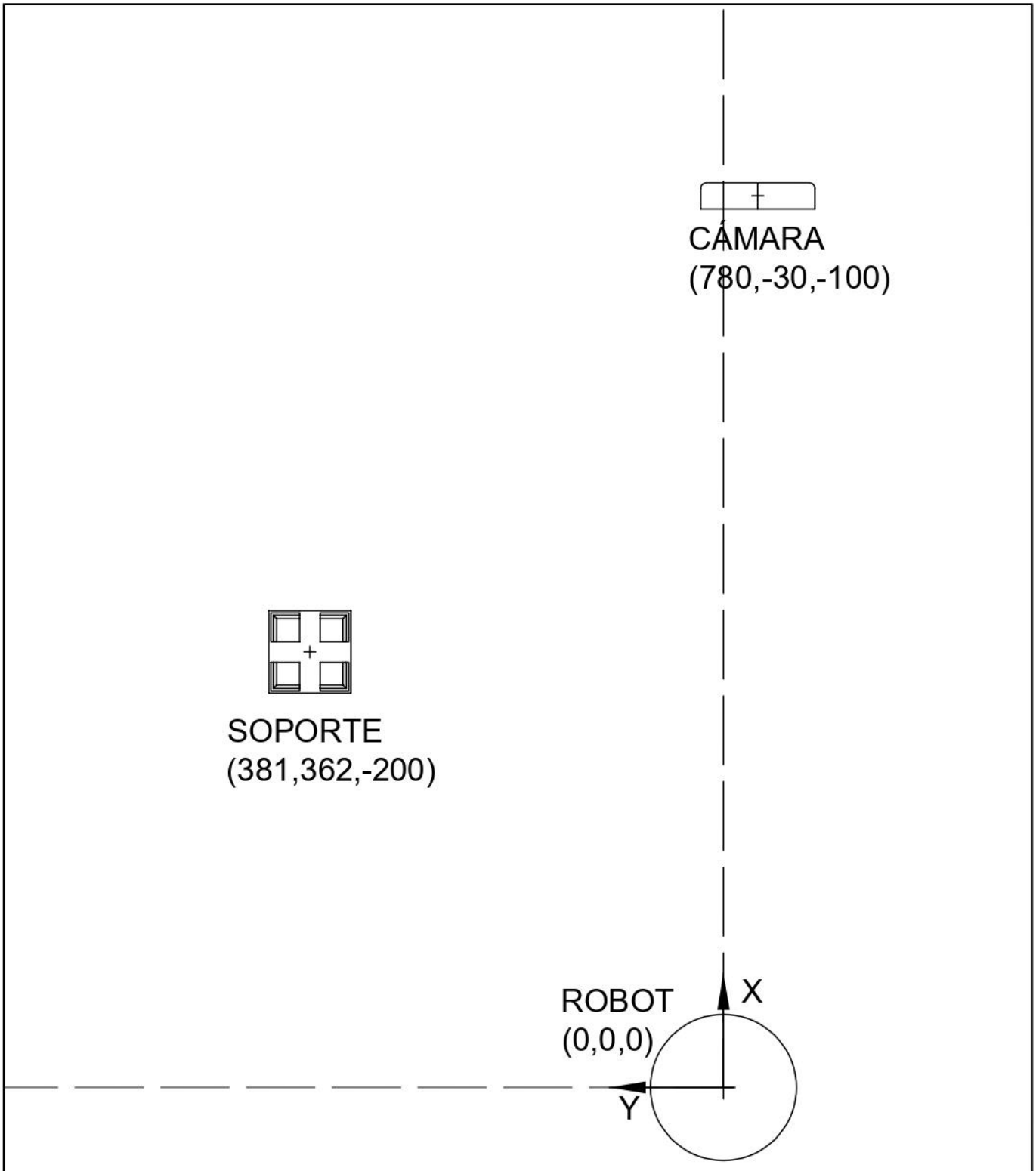
	Nombre	Fecha	Firma	
Diseñado	ABEL GÓRRIZ	12/06/2023		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO
Dibujado	ABEL GÓRRIZ	12/06/2023		Titulo del proyecto: Robot resolvedor de Cubos de Rubik
Comprobado				Autores: ABEL GÓRRIZ ALIAGA

ESCALA	Denominación del plano	Número del plano
1/1	Diseño de Soporte	1



	Nombre	Fecha	Firma	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO
Diseñado	ABEL GÓRRIZ	12/06/2023		Titulo del proyecto: Robot resolvedor de Cubos de Rubik
Dibujado	ABEL GÓRRIZ	12/06/2023		Autores: ABEL GÓRRIZ ALIAGA
Comprobado				

ESCALA	Denominación del plano	Número del plano
2/1	Diseño de Pinza	2



	Nombre	Fecha	Firma	
Diseñado	ABEL GÓRRIZ	12/06/2023		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO
Dibujado	ABEL GÓRRIZ	12/06/2023		Titulo del proyecto: Robot resolvedor de Cubos de Rubik
Comprobado				Autores: ABEL GÓRRIZ ALIAGA

ESCALA	Denominación del plano	Número del plano
1/5	Posicionamiento de los elementos	3

UNIVERSITAT POLITÉCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño y desarrollo de un sistema robotizado capaz de resolver un cubo de Rubik mediante un robot colaborativo y visión artificial

DOCUMENTO Nº3: Pliego de Condiciones

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Górriz Aliaga, Abel

Tutor/a: Ivorra Martínez, Eugenio

CURSO ACADÉMICO:2022-2023

Índice de contenidos

1. Condiciones generales

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

La presente especificación técnica se refiere al diseño e implementación de un sistema robotizado capaz de resolver un cubo de Rubik con la ayuda de una cámara.

1.1. Vigencia

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

1).-Planos.

2).- Pliego de Condiciones.

3).- Presupuesto.

4).- Memoria.

1.2. Descripción

Este proyecto regula la instalación e implementación de un sistema robotizado capaz de resolver un cubo de Rubik con la ayuda de una cámara.

1.3. Modificaciones

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

1.4. Dirección e inspección

La dirección de la instalación eléctrica estará a cargo del responsable de la dirección del proyecto, pudiendo éste delegar en personal a cargo de la ejecución práctica de la instalación.

2. Condiciones facultativas

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

3. Objeto

La presente especificación técnica se refiere al diseño e implementación de un sistema robotizado, basado en un robot UR3e, capaz de resolver un cubo de Rubik, de la marca Rubik, con la ayuda de una cámara, la Intel RealSense D415.

4. Condiciones de los materiales

4.1. Robot UR3e

- Alcance: 500 mm
- Carga útil: 3 kg
- Huella: Ø128 mm
- Peso: 11.2 kg

4.2. Pinza Paralela DHPS-20-A

- Tamaño: 20
- Carrera por mordaza: 6.5 mm
- Número de mordazas: 2
- Tipo de actuador: neumático
- Modo de funcionamiento: doble efecto
- Presión de funcionamiento: 2 bar…8 bar
- Peso: 380 g
- Fuerza total de sujeción a 6 bar en cierre: 290 N

4.3. Cubo de Rubik 3x3

- Longitud de arista: 56 mm
- Peso: 88 g

4.4. Intel RealSense D415

- RGB frame resolution: 1920x1080
- RGB frame rate: 30fps
- RGB sensor technology: Rolling Shuther

4.5. Diseños 3D

- Material de la impresión: ácido poliláctico (PLA)

4.6. Python

- Python 3.8.10

5. Condiciones de la ejecución

5.1. Materiales

Robot UR3e:

- Deberá ser colocado en un entorno seguro y en la posición y orientación que marca el plano nº3, en dicho plano se ha tomado como origen la base del robot.
- Se conectará todo el sistema a la electricidad correctamente.

Pinza paralela DHPS-20-A:

- Deberá ser acoplada al extremo final del robot.
- Se conectará el sistema al compresor de aire que proceda.

Cubo de Rubik 3x3:

- Se colocará el cubo encima del soporte.

Intel RealSense D415:

- Se colocará en la posición que marca el plano nº3, mirando hacia el robot.
- Se conectará mediante un cable USB 3.0 Type-C al ordenador que proceda.
- Se deberá colocar una iluminación adecuada, que no produzca reflejos en la identificación

Diseños 3D:

- El soporte se colocará en la posición que marca el plano nº3.
- El soporte se fijará al suelo mediante cinta aislante o un adhesivo diferente.
- Los acoples se fijarán a la pinza paralela DHPS-20-A mediante un tornillo.

5.2. Ejecución del sistema

Una vez colocada la cámara en su posición, se ejecutará el archivo de Python qbr y se seguirán las instrucciones para una correcta calibración de los colores. Una vez se haya realizado esta calibración, ya se puede ejecutar el sistema las veces que procedan.

Para la ejecución correcta del sistema robotizado sin ningún riesgo, se deberá proceder de la siguiente forma:

- 1º, se colocará el cubo de Rubik en su posición inicial, la cual será con la cara blanca en la parte superior, la verde en la frontal y la roja en la derecha.
- 2º, se ejecutará desde el ordenador el archivo de Python qbr, el cual hace que se active el proceso de reconocimiento de datos y este a la espera del cubo.
- 3º, se ejecutará el programa del robot.

De esta forma se resolverá el cubo de Rubik sin ningún problema.

UNIVERSITAT POLITÉCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño y desarrollo de un sistema robotizado capaz de resolver un cubo de Rubik mediante un robot colaborativo y visión artificial

DOCUMENTO Nº4: Presupuesto

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Górriz Aliaga, Abel

Tutor/a: Ivorra Martínez, Eugenio

CURSO ACADÉMICO:2022-2023

1. Presupuesto por costes según su naturaleza

MATERIALES				
Uds	Denominación	Cantidad	Precio (€)	Total
u	Robot UR3e	1,00	25000,00	25000,00
u	Pinza paralela DHPS-20-A	1,00	574,19	574,19
u	Intel RealSense Depth Camera D415	1,00	447,07	447,07
u	Cubo de Rubik	1,00	12,99	12,99
u	Piezas 3D	2,00	5,00	10,00
u	Ordenador	1,00	600,00	600,00
u	Python	1,00	0,00	0,00
		SUBTOTOTAL MATERIALES		26.644,25 €

MANO DE OBRA				
Uds	Denominación	Cantidad	Precio (€)	Total
h	Desarrollo del sistema por el ingeniero	250,00	15	3750,00
h	Instalacion del material	2,00	15	30,00
		SUBTOTOTAL MANO DE OBRA		3.780,00 €

TOTAL PRESUPUESTO DE EJECUCIÓN MATERIAL		30.424,25 €

Asciende el presente Presupuesto de ejecución material a la cantidad de treinta mil cuatrocientos veinticuatro euros con veinticinco céntimos (30.424,25 €)

2. Resumen del presupuesto

RESUMEN DEL PRESUPUESTO				
Uds	Denominación	Cantidad	Precio (€)	Total
u	Total presupuesto de ejecución material	1,0	30424,25	30424,25
%	Gastos Generales	0,13	30424,25	3955,1525
%	Beneficio Industrial	0,06	30424,25	1825,455
%	Honorarios	0,1	30424,25	3042,425
%	IVA	0,21	30424,25	6389,0925
		TOTAL PROYECTO		45.636,38 €

Asciende el presente Presupuesto Total del proyecto a la cantidad cuarentaicinco mil seiscientos treinta y seis euros con treinta y ocho céntimos (30.424,25 €)