

Anexo I: Hazlo tú mismo

En este anexo se explica, paso a paso, todo lo necesario para poder reproducir y utilizar este robot a partir del contenido existente en el repositorio¹ de este proyecto.

Obtención de los ficheros STL

Para poder imprimir las distintas piezas, primeramente se requiere obtener los ficheros STL a partir de los ficheros fuente (piezas de FreeCAD) . Estos, los podemos encontrar en la carpeta `/src/design/FreeCad` del proyecto. Para generarlos se deben realizar los siguientes pasos:

1. Abrimos la pieza correspondiente con FreeCAD (pulsando encima de ella).
2. Cambiamos al banco de trabajo *Mesh Design*.
3. Hacemos click sobre la pieza para seleccionarla y, posteriormente, pulsamos el botón *Crear malla de forma* del panel superior.
4. Introducimos un valor de superficie de desviación de 0.01mm y damos en OK.
5. Click derecho del ratón sobre el objeto de malla creado y exportamos la malla con el botón *Exportar malla*.

Nota

No se recomienda generar directamente el STL con la opción de exportar del menú superior debido a que no se puede controlar la cantidad de polígonos de la malla y por defecto es bastante baja.

Una vez obtenidos todos los ficheros STL, se pueden importar en el laminador, en este caso se ha empleado Ultimaker Cura 5.2.1 para generar los ficheros que entiende la impresora 3D. La densidad requerida para cada pieza y la cantidad de ellas, se pueden ver en el Cuadro 5.8 del Capítulo 5.

¹<https://github.com/RoboticsURJC/tfg-vperez.git>

Montaje

Actualmente, no se dispone de una guía detallada de montaje. Aún así el montaje resulta bastante sencillo e intuitivo. La manera recomendada de proceder es fijándose en el fichero de montaje llamado `#0_ASSEMBLY.FCStd` que se encuentra en la misma carpeta que las piezas. En él aparece cada pieza, su posición y nombre. En la Sección 5.6 se puede ver la lista de las piezas que son necesarias para poder comprarlas o fabricarlas.

Software

Todo el software necesario para hacer funcionar este robot se encuentra en la carpeta `/src/software` del repositorio. Dentro de ella existen una serie de paquetes para utilizar el robot, los cuales son:

- Paquete `g_arm`: Es un paquete de ROS 2 que contiene el software necesario para comunicarse con el robot desde el ecosistema ROS. Dentro de él se encuentra la carpeta `g_arm_lib` con las clases de python necesarias para abstraer al ejecutable (driver del robot) de las comunicaciones.
- Paquete `g_arm_description`: Es un paquete de descripción creado para representar el robot real en el mundo virtual. No contiene código, únicamente una serie de lanzadores para poder visualizarlo.
- Paquete `g_arm_moveit2`: Es un paquete de MoveIt 2 que utiliza el anterior paquete para poder controlar el robot real desde este framework.
- Paquete `g_arm_python_examples`: Es un paquete que contiene ejemplos de código para mover el robot mediante la API de PyMoveit2.

Nota

Aunque es posible controlar las articulaciones del robot a bajo nivel enviando órdenes de código G por el puerto serie, se pierde la posibilidad de conocer la posición absoluta de cada articulación. Debido a esto, es recomendable utilizar al menos la capa de abstracción *Robot* que se puede encontrar en la carpeta `g_arm_lib` del paquete `g_arm`. Para conocer su funcionamiento, se puede analizar el código del ejecutable `driver.py` de esa misma carpeta.

Aun así, lo ideal es utilizar ROS 2 Humble y MoveIt 2 para controlar el robot. Para ello, es necesario tener instalado ambos y hacer uso de los cuatro paquetes mencionados.

Para utilizar estos paquetes es necesario añadirlos al *workspace* y compilar. En caso de que no se tenga uno, se deben seguir los siguientes pasos:

1. Se crea una carpeta llamada *workspace* en el directorio *home*, por ejemplo. Dentro de ella se debe de crear la carpeta *src*.
2. En *src*, se copian los paquetes mencionados anteriormente. Además de estos cuatro, es necesario incluir el paquete PyMoveit2².
3. Desde el nivel de la carpeta *workspace* ejecutamos el comando *colcon build -merge-install -symlink-install*.
4. Para que ROS lo encuentre debemos añadir la siguiente línea al final del fichero oculto *.bashrc* del directorio *home*: **source ~/workspace/install/setup.bash**

La estructura del *workspace* debería quedar igual a la mostrada en la Figura 1.

```

└── build
    ├── g_arm
    ├── g_arm_description
    ├── g_arm_moveit2
    ├── g_arm_python_examples
    └── pymoveit2
└── install
    ├── lib
    ├── local
    └── share
        └── log
            ├── build_2023-09-22_19-07-44
            ├── build_2023-09-23_19-16-34
            ├── build_2023-09-23_21-08-22
            ├── build_2023-09-23_21-13-38
            ├── build_2023-09-24_11-40-36
            ├── build_2023-09-24_15-28-48
            ├── build_2023-09-24_15-38-39
            ├── latest → latest_build
            ├── latest_build → build_2023-09-24_15-38-39
            ├── latest_list → list_2023-09-24_11-30-00
            └── list_2023-09-24_11-30-00
└── src
    ├── g_arm
    ├── g_arm_description
    ├── g_arm_moveit2
    ├── g_arm_python_examples
    └── pymoveit2

```

Figura 1: Resultado de ejecutar *tree -d -L 1* en la carpeta *workspace*

En cuanto al firmware interno del robot, utiliza la versión de Grbl v1.1. Existen numerosas guías en internet de cómo instalarlo. En el caso de utilizar la misma placa de este proyecto, no es necesario instalar nada ya que viene preinstalado de fábrica. Lo único a cambiar son los parámetros internos a los mostrados en el Cuadro 6.2. En esa misma sección se detalla como se cambian.

²<https://github.com/AndrejOrsula/pymoveit2>

Anexo II: Uso en docencia

En este anexo se aborda el cómo se pretende incluir este robot en la asignatura de Robótica Industrial del Grado de Robótica Software de la Universidad Rey Juan Carlos. Primeramente, se ha consultado con Julio Lora, actual profesor de esta asignatura, cómo se podría integrar este trabajo en el siguiente curso académico. Se llegó a la conclusión de que una buena manera de utilizar este robot en sus clases, sería para complementar al robot UR3, utilizado actualmente para enseñar a los alumnos a generar trayectorias de soldadura. Teniendo además el punto fuerte de aprender a programar un brazo robot a través de un framework de código abierto como es MoveIt2. En base a lo anterior, se ha planteado un posible ejercicio práctico.

Ejercicio: Ejecutando trayectorias en MoveIt 2

Crear un programa en Python para que el robot G-Arm ejecute una trayectoria con framework de MoveIt2. La trayectoria debe consistir en una lista de puntos de paso. Además, se debe hacer uso de la herramienta *Porta lápices* para dibujar la trayectoria sobre un folio con el robot real.

Competencias adquiridas

Al realizar este ejercicio los alumnos aprenderán las siguientes competencias:

1. Conocer la existencia del framework de MoveIt.
2. Aprender a generar los puntos de paso de distintas figuras.
3. Controlar y programar un brazo robot mediante la API de Python de MoveIt2.
4. Representar el recorrido de un eslabón con Rviz.

Conocimientos necesarios para resolverlo

Para solucionar el ejercicio propuesto, es necesario conocer las herramientas disponibles. Una de ellas es librería PyMoveIt2, la cuál proporciona una serie de clases y métodos que facilitan la interacción con este framework. Dentro de ella, existen una serie de clases que nos permiten controlar el robot de las siguientes formas:

1. ***Joint goal***: Permite controlar el robot en el espacio de articulaciones, es decir llevar cada articulación del robot a posición angular concreta.
2. ***Pose goal***: Permite controlar el robot en el espacio cartesiano, es decir llevar al extremo del robot a una posición XYZ con una cierta orientación.
3. ***Gripper action***: Permite interactuar con la herramienta del robot a través de una serie de cómodas funciones.
4. ***Servo***: Permite mandar comandos en tiempo real (sin la previa planificación que requieren los anteriores) para controlar las velocidad lineares y angulares del extremo del robot.

Como en este caso se pretenden realizar trayectorias, es necesario utilizar las funciones relativas a ***Pose goal***.

Desarrollo de la solución

Primeramente, se ha creado un programa sencillo en Python con cuatro funciones las cuales devuelven una lista de puntos que corresponden con los vértices de cuatro figuras: cuadrado, círculo, triángulo y un corazón. Para comprobar que los puntos son correctos, se puede utilizar la librería Turtle para dibujarlos. Por otro lado, es necesario implementar un bucle que itere sobre la lista de puntos y mueva el robot hasta el siguiente. Para hacer esto, es necesario crear una instancia de la clase *MoveIt2* y utilizar los métodos *move_to_pose()* y *wait_until_executed()* como se muestra en el fragmento de código 1.

```

from pymoveit2 import MoveIt2

# Object creation
moveit2 = MoveIt2(
    node=_node,
    joint_names=g_arm.joint_names(),
    base_link_name=g_arm.base_link_name(),
    end_effector_name=g_arm.end_effector_name(),
    group_name=g_arm.MOVE_GROUP_ARM,
    callback_group=callback_group,
    follow_joint_trajectory_action_name="/arm_controller/follow_joint_trajectory",
    execute_via_moveit=False
)

# Target position related to frame_id variable
position = (0.25, 0.0, 0.0)

moveit2.move_to_pose(position=position, quat_xyzw=(1.0, 0.0, 0.0, 0.0),
                     cartesian=True,
                     frame_id=g_arm.base_link_name(),
                     target_link=g_arm.end_effector_name(),
                     tolerance_orientation=3.14)

# Wait until there is no action running
moveit2.wait_until_executed()

```

Código 1: Uso básico de PyMoveIt2 para moverse a un punto

El código completo de la solución se encuentra, junto al resto de ejemplos, en el repositorio de github del proyecto³. Para probarlo, debemos ejecutar lo siguiente (cada comando en una terminal distinta):

```

ros2 launch g_arm_moveit2 show_end_effector_travel.launch
ros2 run g_arm_python_examples trajectory

```

³https://github.com/RoboticsURJC/tfg-vperez/blob/main/src/software/g_arm_python_examples/g_arm_python_examples/trajectory.py

Para visualizar el recorrido de un determinado eslabón se debe de activar la opción *Show Trail* de *RobotModel*. En este caso, interesa activar la del extremo del robot, como se muestra en la Figura 2.

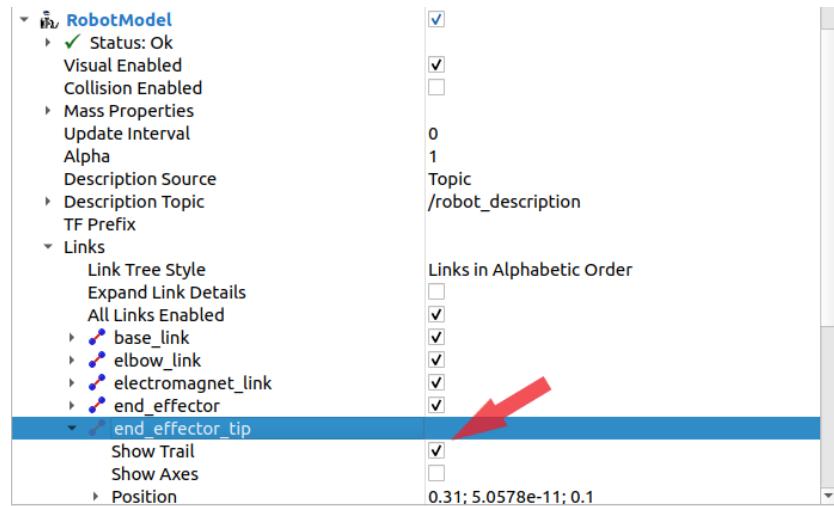


Figura 2: Activación de la opción Show Trail de un link en Rviz

En la Figura 3 se puede ver las distintas trayectorias de ejemplo ejecutadas .

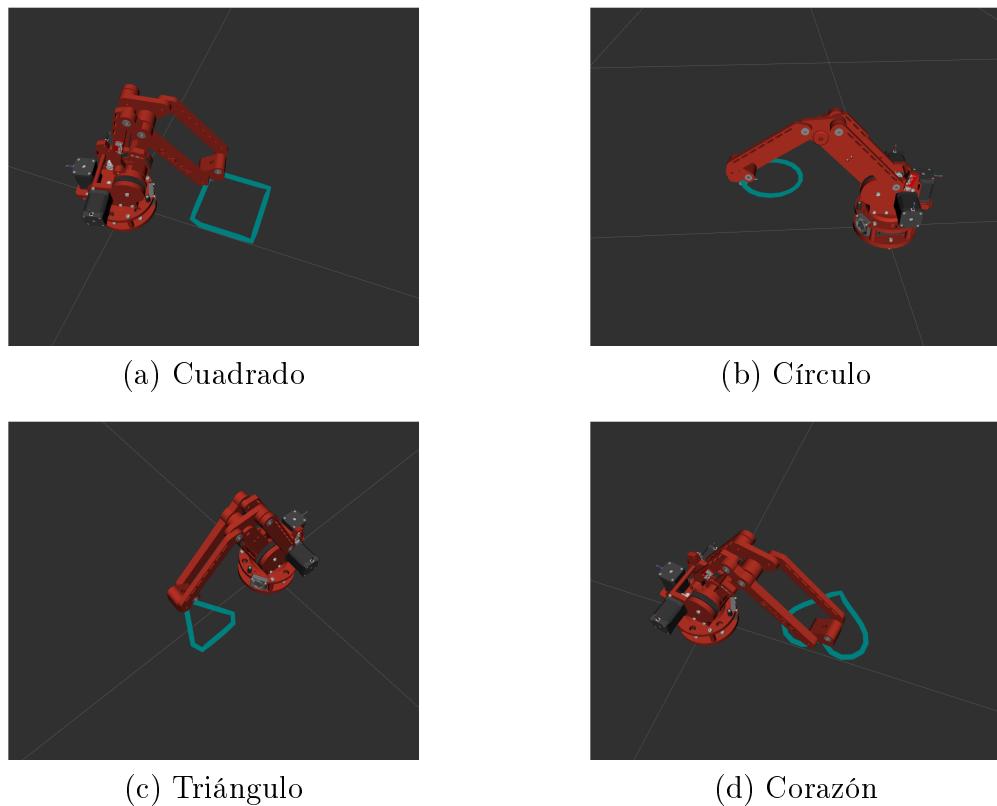
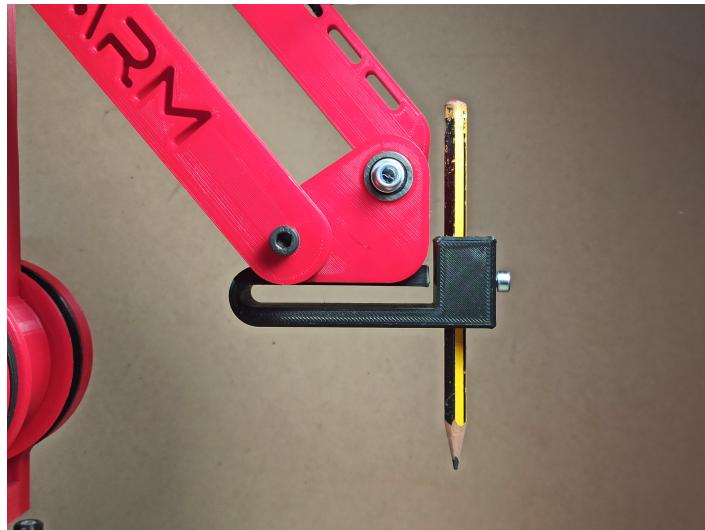


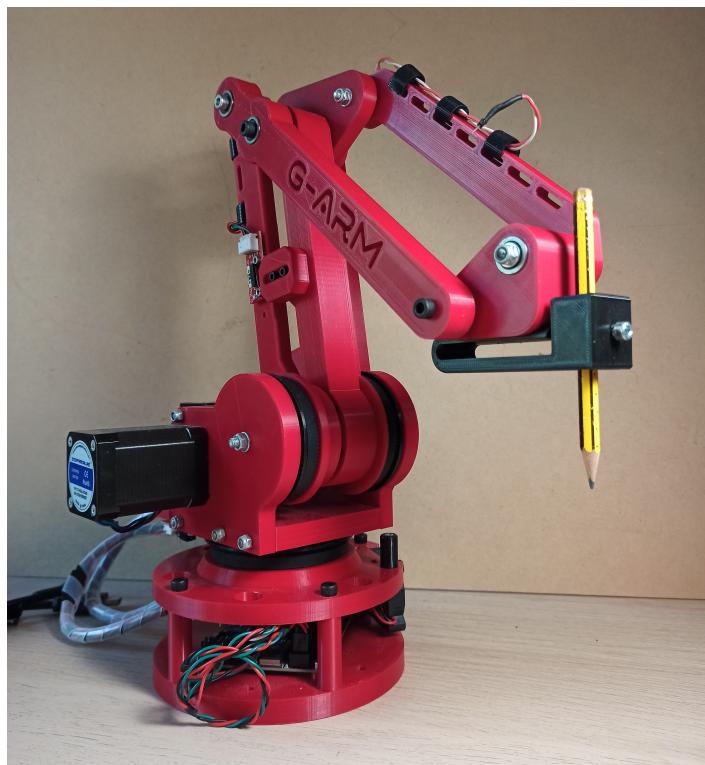
Figura 3: Trayectorias realizadas

Finalmente, con el objetivo de probar la trayectoria en el robot real, se ha diseñado una herramienta nueva (Figura 4). Se trata de un porta lápices que utiliza la flexión del

PLA para acoplar un lápiz al extremo del robot de una forma robusta y sin holguras, siendo a su vez flexible. Se puede descargar el modelo desde la carpeta de piezas del repositorio (24_TOOL_Pen.FCStd).



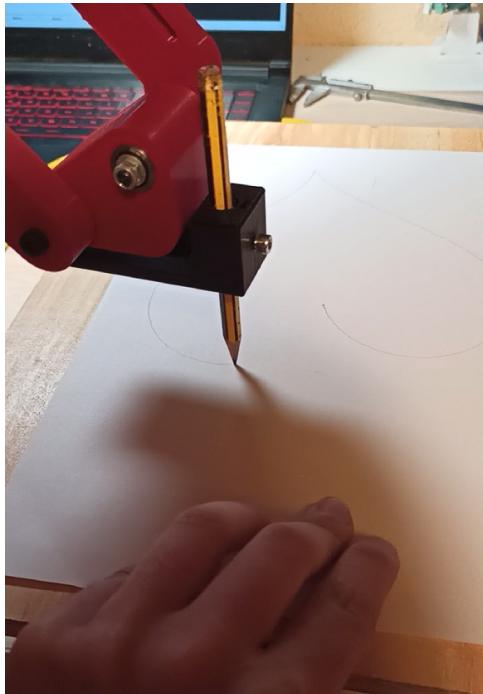
(a) Vista lateral



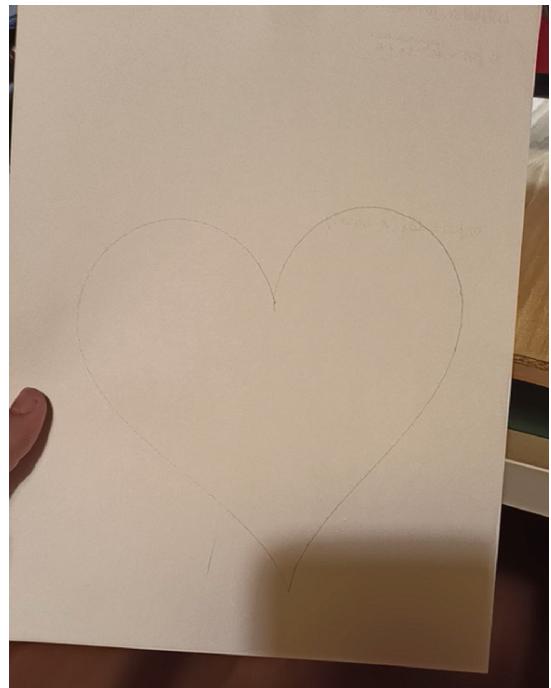
(b) Vista completa

Figura 4: Herramienta porta lápices

Se ha escogido un plano con una altura en Z igual a 10cm, ya que en este plano es capaz de alcanzar una gran cantidad de puntos. En él, se ha dispuesto una base plana y un folio en el que se ha dibujado la figura más compleja del ejemplo, el corazón. El resultado de todo el proceso de dibujado puede verse en el siguiente vídeo⁴. Adicionalmente, en la Figura 5 se muestra algunas imágenes extraídas de él.



(a) Durante la trayectoria



(b) Resultado final

Figura 5: Prueba en el robot real

Como se puede ver, el trazado es excelente, por lo que puede ser utilizado para realizar todo tipo de trayectorias asegurando unos buenos resultados.

⁴https://youtube.com/shorts/3rwK_FV3eSs

Ejercicio extra: Uso de una herramienta actuada

Crear un programa en Python para llevar a cabo una tarea de manipulación con el robot G-Arm. La tarea consiste en coger una moneda de una determinada posición y depositarla en otra distinta, haciendo uso de la herramienta *Electroimán*.

Conocimientos previos

La herramienta acoplada al robot debe ser controlada mediante un joint ficticio con un rango de movimiento de 0.0 a 1.0 radianes. Este valor es convertido por el driver de ROS en un valor entendible por la salida PWM del extremo del robot. En la configuración del paquete de MoveIt se estableció que este joint no pertenece a la cadena cinemática del brazo y no es usado a la hora de planificar. Este, se controla separadamente en un grupo separado llamado Tool. Para controlar el actuador del extremo del robot, la librería PyMoveit2 incorpora una clase llamada *MoveIt2Gripper* y principalmente dos funciones: *open()* y *close()*.

Desarrollo

En este ejercicio se ha utilizado la clase *MoveIt2Gripper* mencionada anteriormente, tal y como se muestra en el fragmento de código 2.

```
# Create MoveIt 2 gripper interface
tool = MoveIt2Gripper(
node=self._node,
gripper_joint_names=g_arm.tool_joint_name(),
open_gripper_joint_positions=g_arm.electromagnet_on(),
closed_gripper_joint_positions=g_arm.electromagnet_off(),
gripper_group_name=g_arm.MOVE_GROUP_GRIPPER,
follow_joint_trajectory_action_name="/tool_controller/follow_joint_trajectory",
callback_group=callback_group)

tool.open() # Open gripper (electromagnet on)
tool.wait_until_executed()
tool.close() # Close gripper (electromagnet off)
tool.wait_until_executed()
```

Código 2: Uso básico de PyMoveIt2 para moverse a un punto

Posteriormente, se han establecido una serie de puntos: P0, PA1, PA2, PB1, PB2 (Figura 6). Y se ha utilizado los conocimientos del anterior ejercicio para moverse entre ellos.

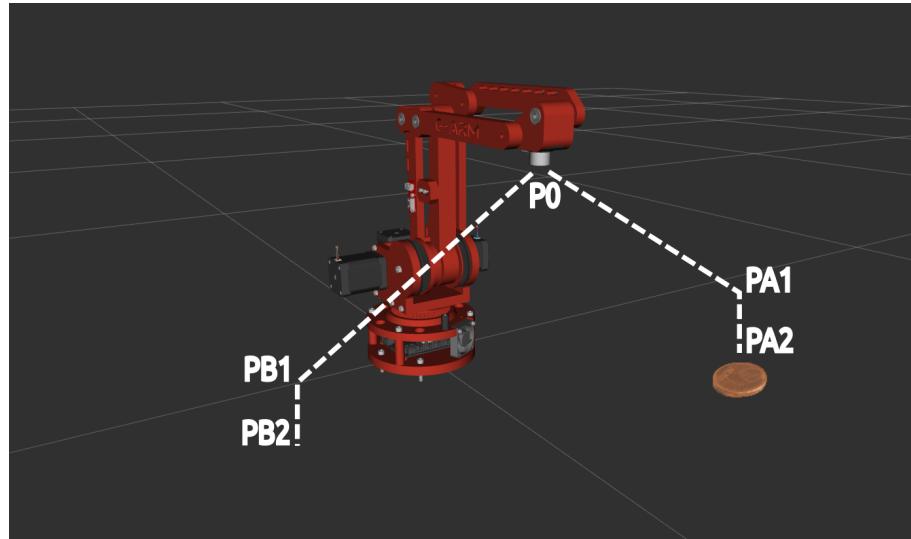


Figura 6: Puntos del ejercicio

El código completo de la solución se encuentra, junto al resto de ejemplos, en el repositorio de github del proyecto⁵. Para probarlo, debemos ejecutar lo siguiente (cada comando en una terminal distinta):

```
ros2 launch g_arm_moveit2 show_end_effector_travel.launch
ros2 run g_arm_python_examples electromagnet
```

En este enlace⁶ se puede ver el vídeo que se ha grabado de la ejecución en el robot real.

⁵https://github.com/RoboticsURJC/tfg-vperez/blob/main/src/software/g_arm_python_examples/g_arm_python_examples/electromagnet.py

⁶<https://youtu.be/iRcEg6AgwcU>