Imagen que contiene Diagrama

Descripción generada automáticamente Logotipo

Descripción generada automáticamente con confianza media

**DIARIO DE LABORATORIO**

**LABORATORIO DE INTEGRACIÓN SENSORIAL E INTERACCIÓN PERSONA-ROBOT**

MÁSTER EN INGENIERÍA INDUSTRIAL

**Malena Potesta González**

***Curso 2023-2024***

**Índice**

# Día 30/11/2023; Definición de etapas y objetivos

Definición de las etapas del trabajo. El diagrama de flujo del trabajo es el siguiente:

Diagrama

Descripción generada automáticamente

Figura 1.- Fases del trabajo

Para conectar RoboDK con Matlab tiene que estar abierta la simulación en RoboDK antes de ejecutar el código. Los códigos de ejemplo están en la web de RoboDK: <https://robodk.com/Matlab-API>

# Día 03/12/2023: Estudio del entorno de trabajo

Estudio de la conexión Matlab-RoboDK: En la *Comand Window* de Matlab poner doc Robolink y te lleva a una ayuda sobre la API. Para que funcione la programación desde Matlab tienen que estar en la misma carpeta el archivo .rdk y el .m. No olvidarse de establecerlos como *Current Folder* antes de ejecutar el código de Matlab.

En este vídeo se explica el funcionamiento de la API de la Matlab y cómo funcionan los movimientos en RoboDK. Es útil para saber programar los movimientos del robot: <https://www.youtube.com/live/gsmKY3Ine4w?si=XLI5ucRP2_L-fyIv> . Es necesario que en todos los scripts se incluyan las siguientes líneas, que son las que permiten establecer la conexión RoboDK-Matlab:

%%

% Generate a Robolink object RDK. This object interface with RoboDK

RDK = Robolink;

% Get the library path

path=RDK.getParam('PATH\_LIBRARY');

Consultar información sobre las matrices homogéneas y las rotaciones con RoboDK.

# Día 04/12/2023: Trabajos previos con RealSense

La aplicación de RealSense se descarga del siguiente link: <https://github.com/IntelRealSense/librealsense/releases/tag/v2.54.2> , el archivo está en Assets, es el Intel.Real.Sense.SDK-WIN10-2.53.1.4623.exe. Esta versión es la anterior a la 2.54.2, que aparece en el primer link. Es mejor descargar la versión anterior porque permite trabajar con Matlab (en la nueva no aparece esta opción). El link de descarga de la versión 2.53.1 es: <https://github.com/IntelRealSense/librealsense/releases/tag/v2.53.1>

En este link se encuentra una pequeña guía para trabajar con la cámara y Matlab: <https://github.com/adi226/Intel-RealSense-Guide/blob/master/Intel%20Realsense%20Guide.pdf>

Al trabajar con la cámara RealSense da un error al intentar capturar colores. La profundidad la captura correctamente. Probablemente, sea un problema de configuración de la misma. Aunque, tal vez sea más sencillo trabajar con una cámara RGB.

Computadora de escritorio sobre superficie de madera

Descripción generada automáticamente con confianza media

Figura 2.- Cámara RealSense

# 05/12/2023: Evaluación de cámaras y captura de movimiento con Mocap

La segunda parte del trabajo tiene que hacerse con una cámara RGB, que permite capturar colores y formas. El tratamiento de la imagen debe ser en blanco y negro. En el Campus Virtual está el fichero de Matlab con los pasos a seguir. El funcionamiento de la cámara es muy sencillo, basta con conectarla al ordenador a través del cable USB y tener en Matlab descargada la *toolbox* para trabajar con cámaras de vídeo.

Una cámara fotográfica

Descripción generada automáticamente con confianza media

Figura 3.- Cámara RGB

Captura de Mocap: Para una primera prueba del tratamiento de datos con Matlab para pasárselos al robot, se capturan una serie de movimientos con las cámaras creando un único sólido rígido. El objetivo es convertir este movimiento en puntos con coordenadas x, y, z que se le puedan pasar al robot como *targets*. El sistema de referencia de las cámaras y el robot no es el mismo, es por ello que se debe crear un nuevo sistema de referencia en RoboDK que coincida con el de las cámaras. Es necesario medir las distancias del *ground plane* de Mocap respecto de la base del robot.

Gráfico

Descripción generada automáticamente

Figura 4.- Ejes en Mocap. En rojo X, en verde Z y en azul Y

Diagrama

Descripción generada automáticamente

Figura 5.- Ejes de la base del UR3e. En rojo X, en verde Y y en azul Z

# Día 11/12/2023; Tratamiento de imágenes para machine learning

Dentro de la segunda parte del trabajo se busca que el robot sea capaz de distinguir piezas con grietas y sin ellas. Para evaluar el funcionamiento de la cámara RGB se utiliza el archivo de GitHub *script\_camara*. Con este archivo se realizan diferentes comprobaciones para garantizar la correcta captura de las imágenes.

Después de varias pruebas, se concluye que la cámara es muy sensible a la luz, de modo que para que pueda apreciar bien las formas de los legos se debe poner un fondo negro detrás del objeto a capturar. A su vez, la definición de la cámara no es alta y en la imagen que debe guardarse para, posteriormente, clasificarla, no se ve bien la grieta, lo que puede dificultar el tratamiento de las imágenes. Probablemente, sea necesario utilizar una cámara con mayor definición.

# Día 12/12/2023: Conexión RoboDK-Matlab y Matlab-UR3e

Una vez comprobado que la conexión RoboDK Matlab funciona queda establecer la conexión con el robot real. En primer lugar, se debe establecer la conexión en remoto del UR3e y RoboDK. Se debe verificar que están conectados a la misma red y establecer en la consola del UR3 la opción de control remoto. En RoboDK, en la pestaña Conectar en la IP se debe establecer la misma IP a la que está conectado el UR3e y darle a conectar. En Matlab, se debe escribir el comando robot.Connect(), si el resultado es 1 la conexión se ha hecho correctamente, si es 0 es que no se ha establecido la conexión. A continuación, ejecutar el comando robot.RunProgram(). Se puede comprobar que el robot se mueve tanto en la simulación como en la realidad (en esta página se explica como establecer la conexión y da información sobre estos comandos: <https://robodk.com/forum/Thread-Using-Matlab-to-control-real-robot> ).

En esta sesión, se ejecutó sobre el robot el scrip titulado myUR3 y se comprobó su correcto funcionamiento. Finalmente, se grabó con Mocap un movimiento para, posteriormente, tratarlo desde Matlab y poder crear un programa para que el robot siga esos movimientos. El *ground plane* en Mocap se estableció en la base del robot, si bien, esta no es la mejor posición para colocar el eje de referencia en RoboDK debido a que el robot tiene problemas para alcanzar las posiciones:

Imagen que contiene interior, tabla, cocina, mostrador

Descripción generada automáticamente

Figura 6.- *Ground plane* de Mocap en la captura del día 12/12/2023

# Día 13/12/2023: Preparación del script en Matlab

En esta sesión se consiguió crear un *srcipt* de Matlab que genera un programa en RoboDK con todos los puntos objetivo a los que debe moverse el robot. Para ello*,* se toma como ejemplo el *script* de la web <https://robodk.com/Matlab-API> , del apartado How to generate a robot program. Además, en la web hay un ejemplo de colocación de la herramienta que puede ser útil a futuro.

Al *script* se le añade una sección en la que los datos obtenidos como *quaternions* se transforman en matrices homogéneas.

%%

% Creación del programa de movimiento del robot

prog=RDK.AddProgram('My Prog');

% Create a joint target home

target = RDK.AddTarget('Home',B0,robot);

target.setAsJointTarget();

target.setJoints(Jrest);

% Add joint movement into the program

prog.MoveJ(target);

% define the pose (H transforms) Cambiar las unidades

W=W;

X=X;

Y=Y;

Z=Z;

X1=X1\*1000;

Y1=Y1\*1000;

Z1=Z1\*1000;

q=quaternion(W,X,Y,Z);

longq=length(q);

%%

for i=1:100:longq

targetname = sprintf('Target%i',i);

target = RDK.AddTarget(targetname, B0, robot);

ad=[zeros(4,3), [X1(i);Y1(i);Z1(i);0]];

qh=quat2tform(q(i))+ ad;

disp(qh);

target.setPose(qh);

prog.MoveJ(target);

end

%%

robot.Connect();

% Run the program we just created

prog.RunProgram();

Mocap da problemas a la hora de capturar el movimiento como sólido rígido, pierde muchos datos y la trayectoria representada es muy pequeña, de modo que el robot no puede alcanzar esos targets.

# Día 14/12/2023: Solución de problemas con Mocap y creación del programa de movimiento en Matlab y RoboDK

La sesión comienza con la correcta colocación de las cámaras entorno al espacio de movimiento del robot. Además, se adecua el laboratorio para evitar reflejos indeseados y se realizan varias pruebas con las cámaras hasta que se consigue que dejen de captar como marcadores destellos del ambiente.

A continuación, se decide que para garantizar que los movimientos grabados por las cámaras son realizables por el UR3e lo mejor es colocar marcadores sobre el extremo del robot y grabar su propio movimiento. Para ello, desde el panel de control del UR3e, con el comando *Mover* se le manda a la posición indicada en la Figura 7 y se captura el movimiento del robot desde la posición de inicio al nuevo objetivo y desde el nuevo objetivo a inicio. El *ground plane* se coloca sobre la mesa, próximo al robot.

Pantalla de juego de computadora

Descripción generada automáticamente con confianza media

Figura 7.- Posición objetivo del UR3e

Una sala de estar

Descripción generada automáticamente con confianza media

Figura 8.- Colocación de los marcadores de Mocap en el UR3e

Lo siguiente es importar los datos a Matlab y eliminar aquellos que no son válidos. Además, se adaptó el *script* para que de los más de 1800 datos solo se consideraran como objetivo 18. Debe tenerse en cuenta que en Mocap las distancias se miden en m y en RoboDK en mm, de modo que los datos calculados por Mocap deben pasarse a mm. Otro punto a tener en cuenta es que los ejes deben colocarse adecuadamente en RoboDK porque pequeñas variaciones en su posición hacen que el robot no pueda alcanzar los objetivos.

for i=1:100:longq

targetname = sprintf('Target%i',i);

target = RDK.AddTarget(targetname, B0, robot);

ad=[zeros(4,3), [X1(i);Y1(i);Z1(i);0]];

qh=quat2tform(q(i))+ ad;

disp(qh);

target.setPose(qh);

prog.MoveJ(target);

end

El resultado final en RoboDK es:

Diagrama, Esquemático

Descripción generada automáticamente

Figura 9.- Espacio de trabajo en RoboDK

# Día 18/12/2023: Preparación del script de Matlab y conexión con RoboDK

RoboDK genera una advertencia con los puntos generados como *targets* desde Matlab y el robot no puede moverse a ellos, ya que los considera como inalcanzables. Para tratar de solucionar este problema se recolocaron las cámaras y se tomó una nueva captura de datos con Motive utilizando un único marcador, en lugar de tres para formar un sólido rígido. En esta ocasión, el plano del suelo se colocó delante del UR3e. De este modo, los *targets* carecen de orientaciones, pero sí se tienen sus posiciones. Aún así, RoboDK sigue sin permitir mover el robot a esos puntos objetivos. No obstante, en el momento en el que se seleccionan los *targets* desde RoboDK y se mueven ligeramente la advertencia desaparece, pero RoboDK no ejecuta el programa.

Imagen que contiene interior, tabla, cama, cuarto

Descripción generada automáticamente

Figura 10.- Colocación del plano del suelo

Una sala de estar

Descripción generada automáticamente con confianza media

Figura 11.- Colocación del marcador el UR3e

# Día 19/12/2023: Solución de problemas RoboDK-Matlab y creación de red neuronal

En esta sesión se continúa trabajando en la conexión RoboDK-Matlab, pues los *targets* que se crean desde el *script* de Matlab aparecen como inalcanzables en RoboDK. Para abordar este problema, se realizan varias pruebas con los *script* del GitHub movimientomocap\_14dic y myUR3. El objetivo de estas pruebas es determinar si el fallo está en el formato de los datos capturados por Motive (fallo de redondeo), en el tratamiento que hace Matlab de ellos o en el propio RoboDK. Se llega a la conclusión de que el fallo, muy probablemente, esté en las orientaciones del extremo del robot y las orientaciones de los *targets*, ya que estas no coinciden. También, anotar que en próximas actualizaciones de Matlab (posteriores a la 2023b) la función *tcpip* dejará de funcionar, lo que puede afectar a la conexión con RoboDK.

En la segunda parte de esta sesión se crea una red neuronal de 2 neuronas que permite distinguir entre piezas rectangulares y cuadradas. Para ello, se dispone el laboratorio según la siguiente imagen:

Imagen que contiene interior, tabla, cuarto, computadora

Descripción generada automáticamente

Figura 12.- Setup del machine learning

Se capturan 25 imágenes de piezas lego rectangulares y cuadradas y se genera la red mediante el *script* de Matlab, ubicado en Github, piezas.mlx. Se le dan tres datos de entrenamiento.

layers = [

imageInputLayer([480 640 3])

convolution2dLayer(3, 2, 'Padding', 'same')

batchNormalizationLayer

reluLayer

maxPooling2dLayer(2, 'Stride', 2)

fullyConnectedLayer(2)

softmaxLayer

classificationLayer];

Al ser una tarea sencilla, el programa tiene una exactitud de 1. Al hacer una prueba con una imagen se ve el programa funciona.

# Día 20/12/2023: Solución de problemas RoboDK\_Matlab y continuación de la red neuronal

En esta sesión se concluye que el problema de la conexión RoboDK-Matlab es la orientación de los ejes en el extremo del UR3 y las orientaciones de los ejes en los punto objetivo, ya que al no coincidir estas el robot o no puede posicionarse en el objetivo o lo hace mediante un movimiento que no corresponde al real. Para abordar el problema se toma una nueva captura de datos con Motive. El plano del suelo se muestra en esta imagen:

Imagen que contiene interior, tabla, pequeño, pastel

Descripción generada automáticamente

Figura 13.- Colocación del plano del suelo

Imagen que contiene pequeño, verde, morado, papalote

Descripción generada automáticamente

Figura 14.- Orientación de los ejes en Motive. En azul Z, en rojo X y en verde Y

Se captura un cuerpo sólido con los marcadores colocados del siguiente modo:

Un par de personas con instrumentos musicales y micrófonos en un escritorio

Descripción generada automáticamente con confianza baja

Figura 15.- Marcadores para la captura de datos con Motive

En Motive los ejes tienen una orientación diferente a los de la base del UR3e en RoboDK. Es por ello que en RoboDK se crea un nuevo sistema de ejes rotando el sistema de ejes de la base del UR3e 90 grados en X.

%%

% robot frame, hide it

Br=robot.Parent();

Br.setVisible(0);

%%

% define a new rotated frame to be base frame

B0=RDK.AddFrame('B0 frame', Br);

Hr0=transl(0,-90,0)\*rotx(pi/2);

B0.setPose(Hr0);

% from now on, all motions are taken with respect to B0

robot.setPoseFrame(B0);

Puesto que en el movimiento capturado el robot comienza parado y finaliza unos segundos parado, se eliminan de los vectores las partes del movimiento en las que el robot no se mueve se eliminan de los datos.

% load('WS\_punto\_18dic.mat')

% define the pose (H transforms) Cambiar las unidades

% Download date

load('WS\_20dic.mat')

W=W(480:1380);

X=X(480:1380);

Y=Y(480:1380);

Z=Z(480:1380);

X1=X1(480:1380);

Y1=Y1(480:1380);

Z1=Z1(480:1380);

% figure

% plot(X1)

q=quaternion(W,X,Y,Z);

longq=length(q);

Para orientar correctamente los *targets* se le aplica una rotación en Z de 180 grados a la matriz homogénea con las posiciones de estos. Los puntos deben establecerse como Cartesian Targets (es la opción por defecto) y el movimiento para llegar a ellos de tipo articular.

for i=1:200:longq

targetname = sprintf('Target%i',i);

target = RDK.AddTarget(targetname, B0, robot);

ad=[zeros(4,3), [X1(i);Y1(i);Z1(i);1]];

qh=quat2tform(q(i))+ ad;

qh=qh\*rotz(pi);

disp(qh);

target.setPose(qh);

prog.MoveJ(target);

end

Ahora el programa funciona correctamente y tanto la simulación como el UR3e realizan el movimiento. No obstante, el robot en ciertos puntos realiza un movimiento más brusco que en otros. Esto puede deberse a que en esta primera prueba no se le han pasado muchos puntos objetivos o a que el eje B0 no está colocado de forma del todo correcta, pues el punto que se le marca desde Matlab como Home no coincide exactamente con los *targets* en los que el robot está quieto grabados con Motive.

Imagen que contiene Interfaz de usuario gráfica

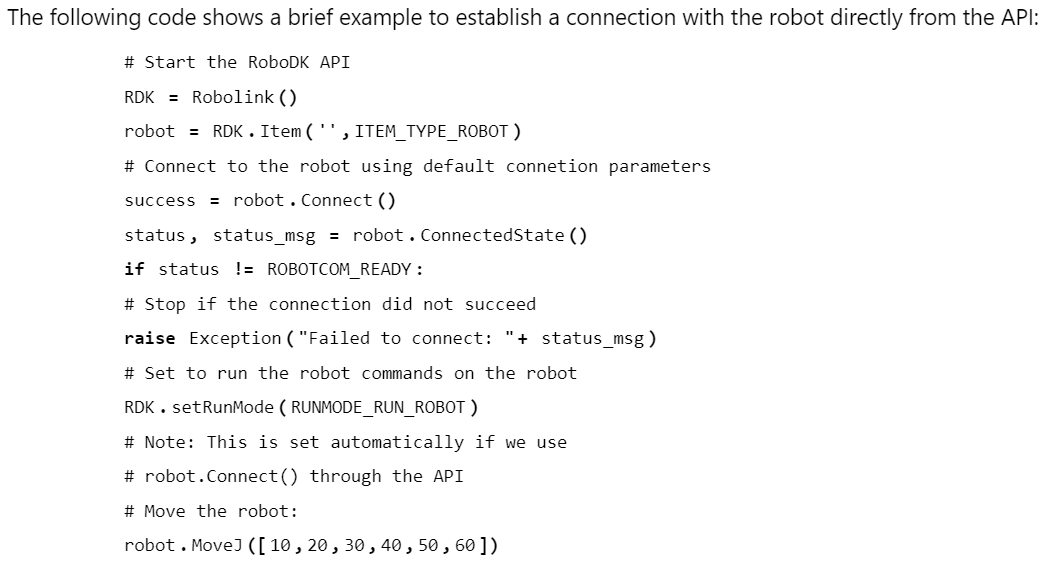
Descripción generada automáticamente

Figura 16.- Resultado final de la simulación

En esta sesión se continúa trabajando con las redes neuronales. Se comienza tomando fotos de piezas lego rotas. Partiendo del ejemplo hecho el día anterior se crean dos redes neuronales nuevas de dos neuronas. La primera permite distinguir entre legos rotos y legos válidos y la segunda entre legos rotos de forma rectangular, legos válidos de forma rectangular, legos rotos de forma cuadrada y legos rotos de forma rectangular. En ambos casos se tienen cuarenta y una imágenes, de las cuales cinco se utilizan para entrenar la red. La exactitud en ambos casos es próxima a 0,95. La red acierta con la imagen de prueba que se le pasa. Los nuevos archivos se encuentran accesibles en GitHub dentro de la carpeta TRABAJO, PARTE 2 MACHINE LEARNING, Redes\_neuronales, cuadrado\_rectangulo\_roto, piezas\_rotas.

# Día 21/12/2023: Solución de problemas con la conexión RoboDK-UR3e y primeros pasos con Turtlebot4

Tras solucionar los problemas de posicionamiento de RoboDK, se intenta establecer una conexión entre el UR3e y RoboDK a través de Matlab. En base a la bibliografía consultada en sesiones previas, la orden que establece que los comandos de RoboDK se ejecuten en Matlab es robot.Connect(). Se supone que de ese modo se cambia el modo de trabajo a “RUNMODE\_RUN\_ROBOT” y así los comandos se ejecutan en el robot real. En la propia página de RoboDK se especifica la información anterior (<https://robodk.com/doc/en/Robot-Drivers.html> ).



No obstante, a la hora de establecer la conexión RoboDK se conecta correctamente con el UR3e, pero no se ejecuta en el programa ni en la simulación ni en UR3e. En la clase de Robolink se especifican los siguientes modos de funcionamiento:

% Script execution types

RUNMODE\_SIMULATE=1; % performs the simulation moving the robot (default)

RUNMODE\_QUICKVALIDATE=2; % performs a quick check to validate the robot movements

RUNMODE\_MAKE\_ROBOTPROG=3; % makes the robot program

RUNMODE\_MAKE\_ROBOTPROG\_AND\_UPLOAD=4; % makes the robot program and updates it to the robot

RUNMODE\_MAKE\_ROBOTPROG\_AND\_START=5; % makes the robot program and starts it on the robot (independently from the PC)

RUNMODE\_RUN\_ROBOT=6; % moves the real robot from the PC (PC is the client, the robot behaves like a server)

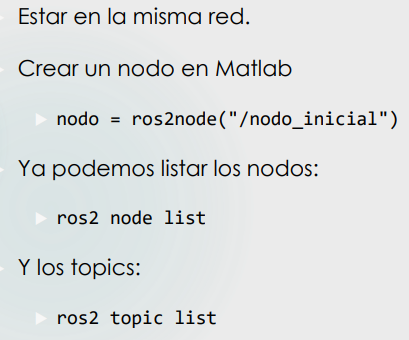
% Program execution type

PROGRAM\_RUN\_ON\_SIMULATOR=1; % Set the program to run on the simulator

PROGRAM\_RUN\_ON\_ROBOT=2; % Set the program to run on the robot

A pesar de buscar bibliografía y de utilizar la orden de “RDK**.**setRunMode**(**RUNMODE\_RUN\_ROBOT**)”** no se consigue establecer la conexión con el UR3e correctamente. Para que se ejecute el programa en el UR3e y en la simulación se debe marcar en RoboDK la orden de “Ejecutar en el robot”, en caso y al dar la orden de prog.RunProgram() sí se ejecuta el programa en la simulación y en el UR3e. Este fallo puede ser debido a la propia API de Matlab, ya que en las páginas de RoboDK estos comandos se dan en Python.

Paralelamente a la búsqueda de una solución a este problema se comenzó a trabajar en la programación en Matlab y ROS2 del Turtlebot4. El objetivo es que este robot haga un mapa del laboratorio para luego poder hacerle navegar hasta un punto próximo al UR3e y programar en Matlab la interacción entre ambos. En primer lugar se enciende el Turtlebot4 alejándolo de la base. A continuación, se establece la conexión con Matlab y el robot mediante las siguientes órdenes:



La parte de navegación del robot por el laboratorio es bastante sencilla. Se debe utilizar el paquete “turtlebot4\_navigation”. Abrir RVIZ y en RVIZ habilitar las imágenes, el lidar (topic /scan) y el propio robot. Mediante un “teleop” se controla la navegación por el laboratorio hasta que se ha mapeado toda la sala. El siguiente paso es llevar el robot a un punto concreto, para ello cuando se ven los topic en Matlab se tienen los siguientes que pueden ayudar a este propósito:

Texto

Descripción generada automáticamente

Imagen que contiene Texto

Descripción generada automáticamente

En estos links se tiene información acerca del funcionamiento de estos topic:

<https://blog.csdn.net/ncnynl/article/details/127306827>

<https://blog.51cto.com/u_1790502/6084578>

<https://get-help.robotigniteacademy.com/t/topic-nav2-goal-from-rviz2/23047/4>

# Día 08/01/2024: Últimas acciones con RoboDK-Matlab y trabajos previos con Turtlebot4

Se comienza la sesión trabajando con la conexión RoboDK-Matlab para poder hacer que se ejecute en el robot real el programa creado por Matlab sin tener que activar en la aplicación de RoboDK la opción de *Ejecutar en el robot*. Una de las posibles soluciones consiste en tener un archivo de RoboDK con el programa *My prog* ya creado, pero vacío y añadirle desde Matlab las instrucciones. Se deben añadir las siguientes líneas:

% Get all programs

allprograms = RDK.ItemList(RDK.ITEM\_TYPE\_PROGRAM);

if numel(allprograms) < 1

error('No programs found in the station!')

end

% Select first program:

prog = allprograms{1};

fprintf('Selecting program: %s\n', prog.Name())

Y eliminar esta:

prog=RDK.AddProgram('My Prog');

En el ordenador portátil de Illán funciona con el nuevo código, pero en el ordenador de mesa del laboratorio, a pesar de haber actualizado Matlab y la API a la misma versión que las del portátil, genera un error. El error se debe a que las variables *allprograms* y *prog* no las crea como un *Robolink* sino como un array de contenido ‘My prog’. Al no crearlas como una clase *Robolink* no se pueden añadir instrucciones como *prog.MoveJ(target)*. Este problema queda sin resolver, pues resulta extraño que utilizando las mismas versiones de todos los programas y t*oolbox* y el mismo código funcione en un ordenador, pero no en otro.

En la segunda parte de la sesión se trabaja con el Turtlebot4. En primer lugar, se debe dar la orden de *undock* para que el robot salga de la base. Al darle esta orden en RViz ya se puede acceder al topic de la cámara.

Texto

Descripción generada automáticamenteLo siguiente es hacer un mapa del laboratorio, para lo cual se siguen las instrucciones que se encuentran en el siguiente link: <https://turtlebot.github.io/turtlebot4-user-manual/tutorials/navigation.html>. La versión que se utiliza de ROS2 es Humble. En concreto, las órdenes que se le dan al robot para realizar el mapeo son:

Figura 17.- Orden de undock

Para crear un mapa mediante técnicas SLAM:

ros2 launch turtlebot4\_navigation slam.launch.py

Para lanzar RViz y tener ya configurado el entorno sin necesidad de hacerlo manualmente:

ros2 launch turtlebot4\_viz view\_robot.launch.py

Una captura de pantalla de una computadora

Descripción generada automáticamente con confianza media

Figura 18.- Entorno de RViz

En terminator se lanza un teleop y se puede controlar el robot para que mapee la sala. Debido a la posición del lidar y su rango de visión los elementos que se encuentran a una baja altura, como la barra de la parte frontal de la mesa donde está el UR3e, no los detecta bien.

Una silla de madera

Descripción generada automáticamente con confianza baja

Figura 19.- Adaptación del entorno de trabajo para mapeo con el Turtlebot4

En esta sesión se investiga, también, el funcionamiento del mando del Turtlebot4. Si se presiona el botón de house y el botón único a la izquierda se puede conectar el mando por Bluetooth al ordenador. En el ordenador portátil funciona correctamente y se puede controlar la pantalla con el mando. Lo siguiente es tratar de establecer una conexión con el robot, lo que no se ha hecho en esta sesión debido a que el Turtlebot4 se quedó sin batería.

Imagen que contiene arma, tabla, remoto, sostener

Descripción generada automáticamente

Figura 20.- Mando del Turtlebot4

# Día 09/01/2024: Mapeo y navegación con el Turtlebot4

Se comienza la sesión investigando acerca del funcionamiento del mando del Turtlebot4. No se puede establecer una conexión entre el mando y el robot debido que en el ordenador del laboratorio no está instalado el paquete *turtlebot4\_bringup*, el cual contiene un launchfile para utilizar el mando, como se explica en este link:

<https://turtlebot.github.io/turtlebot4-user-manual/tutorials/driving.html>

El paquete se puede descargar de GitHub:

<https://github.com/turtlebot/turtlebot4_robot/tree/humble/turtlebot4_bringup>

Al intentar descargarlo en la carpeta /opt/ros/humble mediante el comando git clone y el link al repositorio de GitHub se obtiene un error de permisos en dicha carpeta: “fatal: could not create work tree dir 'turtlebot4\_bringup': Permission denied”. Los comandos para tratar de descargar el paquete se obtienen de este link: <https://support.franka.de/docs/franka_ros2.html> . En esta sesión no se trabaja más con el mando (pues no es de especial relevancia para el proyecto) y se pasa a la parte de mapeo y navegación.

Texto

Descripción generada automáticamenteEn primer lugar, se crea un mapa del laboratorio siguiendo los pasos que se indican en el siguiente link: <https://turtlebot.github.io/turtlebot4-user-manual/tutorials/generate_map.html> . Para ello se mueve el robot por la sala mediante un teleop. El mapa se guarda con el comando mostrado en la siguiente imagen, en el directorio ~/MapaRobolab (es más sencillo utilizar este comando que el que viene en el link).

Figura 21.- Comando para guardar el mapa

El mapa resultante es el mostrado a continuación:

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

Figura 22.- Mapa del laboratorio generado por Turtlebot4

Este mapa es más preciso que el generado en la sesión anterior. El siguiente paso consiste en hacer navegar al robot hasta el punto deseado. Para ello se siguen las instrucciones del siguiente link: <https://turtlebot.github.io/turtlebot4-user-manual/tutorials/navigation.html>

Una captura de pantalla de una computadora

Descripción generada automáticamenteEn primer lugar, se lanzan los comandos necesarios y se obtiene que en Rviz se abre el mapa anterior y se le pueden dar las órdenes de navegar al robot.

Figura 23.- Entorno de trabajo en Rviz

Interfaz de usuario gráfica

Descripción generada automáticamenteEn segundo lugar, se le indica al robot dónde se encuentra y mediante un e*cho* se obtienen las coordenadas de ese punto.

Figura 24.- Posición inicial

Interfaz de usuario gráfica

Descripción generada automáticamenteFinalmente, mediante el comando Nav2 Goal se le indica al robot la pose final a la que se le quiere enviar y se guardan sus coordenadas.

Figura 25.- Pose final

El robot llega correctamente al punto pedido, como se ve en la siguiente imagen. Si bien, en RViz se observa que el robot se representa con un elemento semicircular a su alrededor (Figura 27), se debe estudiar cómo eliminarlo, pues es lo que hace que el robot no llegue justo al punto deseado.

Imagen que contiene tabla

Descripción generada automáticamente

Figura 26Interfaz de usuario gráfica

Descripción generada automáticamente.- Robot real en la pose final

Figura 27.- Representación en RViz del Turtlebot4

Texto

Descripción generada automáticamenteAntes de controlar el Turtlebot4 desde Matlab se debe conocer cuál es el topic con el que se le puede enviar a la posición pedida. Se prueba primero con el topic /*goal*\_*pose*. Con este topic se consigue que el robot vaya a la posición pedida. Es importante cambiar el campo *frame\_id* a map para que el robot sepa que se está moviendo por el laboratorio. La sintaxis de este topic se muestra en la siguiente figura. El robot se mueve correctamente al punto pedido.

Figura 28.- Sintaxis del topic /goal\_pose

Se prueba, también, mediante acciones, en concreto, se utiliza la acción:

ros2 action send\_goal /navigate\_to\_position irobot\_create\_msgs/action/NavigateToPosition “{achieve\_goal\_heading: true, goal\_pose:{pose…{position:{x: 0.106, y: -2.52, z:0, orientation:{x:0.0, y:0.0, z:0.0, w:0.0}}}}”. Con esta acción se obtiene un mensaje de error y se aborta la acción, debido a que entra en conflicto con los comandos de RViz.

# Día 10/01/2024: Conexión Turtlebot4-Matlab y captura de datos con el UR3e

Se comienza la sesión trabajando con la conexión Turtlebot4-Matlab. Se desarrolla el *script* de Matlab que se muestra a continuación:

Con este *script* y siguiendo los pasos indicados en el link para navegar con el Turtlebot4 se consigue mandarle desde Matlab a diferentes puntos. Para saber la posición y orientación de los puntos se realiza un echo del topic /amcl\_pose, en donde se muestran los valores buscados, y se lleva desde RViz al Turtlebot4 al punto buscado. Es de destacar que debido al buffer que se muestra en la simulación en RViz y los propios errores de medida en los sensores, el robot no llega perfectamente al punto. La acumulación de errores al mandarlo varias veces a distintos puntos hace que se le aleje de los puntos objetivo. Esto no supone un gran impedimento para el desarrollo del trabajo, pero se estudiará como disminuirlos. Una buena práctica es tratar de ajustar lo máximo posible en RViz la estimación inicial de la pose. También, se disponen marcas en el suelo del laboratorio para trabajar siempre con las mismas referencias de puntos.

Una vez consigo que el Turtlebot4 se mueva desde Matlab se pasa a trabajar con el UR3e, para poder crear un *script* en Matlab en el que se coordine el movimiento de ambos robots. En primer lugar, se toma una captura de datos con Motive del movimiento del robot. Se manda el UR3e a un punto de la mesa donde están colocadas las piezas lego y una vez que ha llegado a ese punto se mueve la base del robot para que arrastre la pieza, después se devuelve el robot a su posición de inicio. Se busca que cuando el robot UR3e vea una pieza defectuosa la arrastre por la mesa hasta dejarla caer sobre el Turtlebot4 para que este la lleve a otro punto del laboratorio. Puesto que las piezas van a estar en la mesa cubierta con lonas negras se debe ajustar bien la posición del UR3e para evitar que se choque contra la mesa. Para poder capturar esta pose exacta se utiliza, a modo de simulación de la pinza, una caja de cartón, como se ve en la siguiente figura.

Imagen que contiene interior, pequeño, tabla, cama

Descripción generada automáticamente

Figura 29.- Caja como simulación de pinza del UR3e

La pose a la que se lleva el robot es la siguiente:

Pantalla de juego de computadora

Descripción generada automáticamente con confianza media

Figura 30.- Pose del UR3e para empujar las piezas

Tras la captura de datos se crea un nuevo *script* de Matlab para establecer su conexión con RoboDK y poder programar las trayectorias. De nuevo, RoboDK genera ciertos problemas con las orientaciones de los puntos y genera trayectorias que no se corresponden con la captura de imágenes. Para esta toma de datos, los puntos capturados con Motive no se someten a ninguna rotación.

for i=1:300:longq

targetname = sprintf('Target%i',i);

target = RDK.AddTarget(targetname, B0, robot);

%target.setAsJointTarget();

%ad=[zeros(4,3), [X1(i);Y1(i);Z1(i);0]];

ad=[zeros(4,3), [X1(i);Y1(i);Z1(i);1]];

qh=quat2tform(q(i))+ ad;

qh=qh%\*rotz(pi);

disp(qh);

target.setPose(qh);

prog.MoveJ(target);

end

Puesto que RoboDK no genera correctamente las trayectorias, se deben incluir en el programa de RoboDK únicamente aquellos puntos para los que la trayectoria de la simulación es la misma que la original.

Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza baja

Figura 31.- Simulación en RoboDK con algunos de los puntos válidos

# Día 11/01/2024: Trabajo con RoboDK

El *script* de Matlab se modifica para que el número de puntos que se representan en RoboDK sea menor al de los puntos capturados con las cámaras.

for i=1:50:(longq-400)

targetname = sprintf('Target%i',i);

target = RDK.AddTarget(targetname, B0, robot);

%target.setAsJointTarget();

%ad=[zeros(4,3), [X1(i);Y1(i);Z1(i);0]];

ad=[zeros(4,3), [X1(i);Y1(i);Z1(i);1]];

qh=quat2tform(q(i))+ ad;

qh=qh%\*rotz(pi);

disp(qh);

target.setPose(qh);

prog.MoveJ(target);

end

Con los puntos generados se consigue una trayectoria similar a la original y el robot no tiene problemas en llegar a cada uno de los punto objetivo. No obstante, se plantea la posibilidad de volver a capturar los datos a una velocidad menor para asegurarse de que las cámaras no pierden tanto puntos.

Diagrama

Descripción generada automáticamente

Figura .- Programa en RoboDK con puntos válidos

Para conectar el UR3e con RoboDK se deben hacer una serie de cambios en la conexión de los router. El UR3e está conectado al router que le da acceso a la red de UniOvi, pero esta red es distinta de la red a la que está conectado el Turtlebot4. Si se quiere controlar el Turtlebot4 desde el ordenador del laboratorio es necesario que dicho ordenador esté en la misma red y, a la vez, debe estar en la misma red que el UR3e. Se decide conectar el UR3e al router que proporciona la red robolab\_5g. En principio, el ordenador del laboratorio se conecta por wifi a la red robolab\_5g, pero esto no permite conectarse al UR3e, aunque sí al Turtlebot4, al tener una dirección IP diferente por wifi que mediante la conexión Ethernet. Para que las conexiones funcionen, el UR3e debe estar conectado por Ethernet al router que proporciona la red robolab\_5g.

INSERTAR foto del router

La IP para poder conectarse al UR3e desde RoboDK es:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura .- Dirección IP para poder conectarse al U3e desde la red robolab\_5g

Al ejecutar el programa en el robot real se ve que en su trayectoria cuando empuja la pieza se acerca demasiado a la mesa del propio UR3e y eso hace que se dificulte su subida, si se le pone una pinza (u otro objeto que lo simule). En esta sesión se ha tratado de ajustar la posición del eje B0 para que que la representación de los puntos objetivo sea mejor, así como de tratar de buscar un punto en la trayectoria que permita que el robot se desplace por la mesa cubierta de lonas negras y no se acerque tanto a su propia mesa. Esto es sumamente complicado, pues en RoboDK una pequeña variación de las coordenadas de los puntos hacen que el robot cambie radicalmente su trayectoria. De nuevo, se plantea regrabar el movimiento en la próxima sesión.