# Lab03 - Depth Camera and Moveit

MTRN4231 - UNSW School of Mechanical and Manufacturing Engineering

## Preliminaries

## Introduction

This lab can be conducted in a group and aims to bridge the gap between the Ros2 concepts taught in the previous weeks with the practical equipment. Completing all these lab tasks is highly advised as this will give you a good platform to start the development of your project.

## Task 1 - Object detection

A depth camera provides an RGBD image, which is the standard monocular RGB image with an additional depth image. For this task, you will use it to estimate the location of a coloured ball with colour masking. Next lab you will learn about YOLO which is a machine-learning algorithm capable of detecting custom objects.

```
ros2 launch realsense2_camera rs_launch.py align_depth.enable:=true enable_color:=true enable_depth:=true
```

Launch RQT to view the monocular image (no depth). in RQT go `Plugins -> Visualization -> Image View` Refresh the image topics and select the image_raw topic to view the image.

Additionally, we can view the point cloud in rviz.

- Open Rviz.
- Go to `Add -> By topic -> /depth/color/points/PointCloud2`.
- Under global options change the fixed frame to `camera_link`.

Complete the provided stub depth code to mask a coloured OOI and broadcast a dynamic transformation frame from the static camera frame completed in lab 2. Verify your code is correct by confirming the transformation frames in Rviz match the real-world location of the object of interest.

## Task 2 - Connecting to the robot

Connect to the control box using the orange Ethernet cable. Execute the setup script in `4231_scripts`:

```
./setupRealur5e.sh
```

Alternatively, if you want to use visualised hardware run the command.

```
./setupFakeur5e.sh
```

This will spawn two nodes and should open RViz displaying the current configuration of the arm. On the robotic arm teach the tablet in manual mode, and create a program that only has the external control URCap program. Run the program to establish communication between ROS2 and the UR5e. The script should open an RVIZ window with a plug-in to directly interface with Moveit. Change the `Goal State` to be `test_configuration` and hit plan and execute.

## Task 3 - Moveit Table Plane

Move it is the inverse kinematic planning library. It is important to understand how and why this is different to the RTDE interface used in 4230. Consider MoveJ, Move J solves a desired joint position and moves each joint to the desired location, during this movement there is no consideration of collisions. In contrast, moving it solves an inverse kinematic path which is constrained by a set of rules. A local controller then directly controls the joints to match the desired trajectory.

The first constraints that will be used are plane constraints which ensure all the joints do not collide with planes during movement. Two planes have already been provided in the sample function, However, there is no table plane. It is your task to add a table plane to the provided code.

- Navigate to `'/4231/lab3_workspace/src'` and open the existing a moveit Package.
- Add a new table plane at z = 0.05.

## Task 4 - Move to ball location

Use your completed code from lab 2 to set up transformations from the map to the depth camera, and the arm. At a frequency of 0.1Hz look up the transformation to the object of interest and move the arm to its location on the table. There is no need to alter the orientation of the arm at this stage, it can remain pointing down.

When launching move it, it is important to launch using the provided launch file (covered in lab4) this is because the node requires additional libraries to function correctly. Launch the node with the code below.

```
ros2 launch lab3_moveit move_to_marker.launch.py
```

Example code can be found in `'/4231/4231_demo_packages/src/demo_moveit_ur'`. In addition, examples of other movement methods can be found such as servo control, obstacle avoidance, and contained box movement.