Introduction to ROS 2
(continued)

COMP3431/COMP9434

Robot Software Architectures

ROS2 Worskspace

colcon_ws

- build
- install
- log
- src (Your packages will go in here)
  - my_package
    - CMAKELists.txt
    - include
    - package.xml
    - src
      - myfile.cpp

•

Creating a package

First move into the src of your work space:

```
cd ~/turtlebot3_ws/src
```

To create a package run the code :

```
ros2 pkg create --build-type ament_cmake <package_name>
```

Example:

```
ros2 pkg create --build-type ament_cmake my_package
```

Building a workspace

Use colcon build to build the project:

```
$ colcon build
```

If installing natively you need to install colcon:

```
$ sudo apt install python3-colcon-common-extensions
```

package.xml

- This is where you put the dependencies
  - what packages does your package depend on?
- E.g. standard for C++

```
<depend>rclcpp</depend>

<depend>std_msgs
```

CMakeLists.txt

- If writing in C++, also need to edit CMakeLists.txt
- E.g. Add the packages we need to find:

```
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
```

CMakeLists.txt

- Also include the source files you will write your programs in
- A simple publisher might have:

```
add_executable(talker src/publisher_member_function.cpp)
ament_target_dependencies(talker rclcpp std_msgs)
```

- The executable called **talker**

CMakeLists.txt

Move the executable to a place where ROS can find and run it

```
install(TARGETS
  talker
  DESTINATION lib/${PROJECT_NAME})
```

Building the workspace

- First move into the root of your work space:

```
cd ~/turtlebot3_ws
```

- Check to see if you have any missing dependencies (good practice but not required)

```
rosdep install -i --from-path src --rosdistro foxy -y
```

- Use colcon build to build the project:

```
colcon build
```

Running the code

- Must source setup for your workspace in new terminal
- cd to workspace directory
- To source workspace run:

```
source install/setup.bash
```

- To run node:

```
ros2 run cpp_pubsub talker
```

ROS2 Visualisation

- To debug the connections between nodes use:

```
$ rqt_graph
```

  - Visualises the node graph – and topic connections
- Rviz2 is the main visualisation tool for ROS:

```
$ rviz2
```

  - Provides plugins architecture for visualising different topics:
    - Videos
    - Map of environment and localised robot
    - Point cloud within the map

ROS2 Visualisation

- To see if your node is punishing to a topic:

```
$ ros2 topic list
```

- To see what's being published on a topic:

```
$ ros2 topic echo/<topic>
```

RQT

- **rqt** is a graphical front end to commands like list and echo
- It let’s you generate message and listen to topic so you can debug individual nodes

ROS 2 Bags

- Possible to save the data produced by topics for later analysis and playback:

```
$ ros2 bag record -a
```

  - Creates a time stamped bag file in the current directory.
  - Warning: “-a” records all topics so will generate a lot of data.
- Often useful to only record only direct sensor inputs (e.g., laser scans and timing) because the other topics will be generated from processing sensor data.
- To replay:

```
$ ros2 bag play <bagfile>
```

- Useful if you are testing different interchangeable node (e.g., mapping with gmapping, hector SLAM, or Cartographer).
- Note: SLAM (Simultaneous Localisation and Mapping) algorithms build a map while at the same time localising. Very widely used in robotics.

ROS Tools – Simulator

Two standard simulators; Stage (2D) and Gazebo (3D)

https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/#gazebo-simulation

The Gazebo guide - easy guide to get simulator up and running.

```
$ export TURTLEBOT3_MODEL=waffle_pi

$ ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py
```

Launch Files

ros2 launch <package> <launch file>

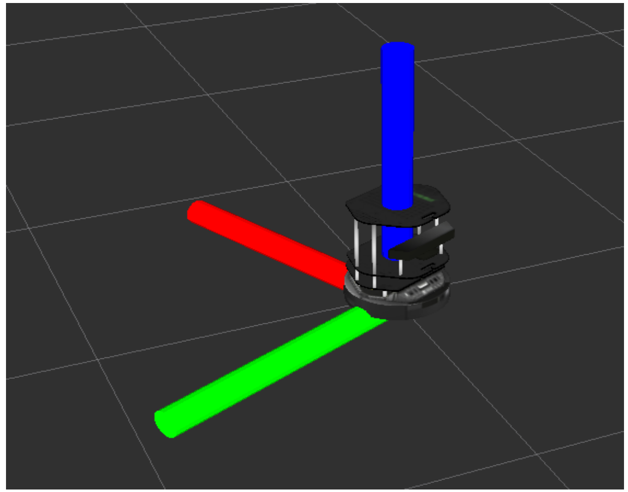
- Create two nodes from same executable
- Give each a separate namespace
- <u>turtlesim2</u> mimics <u>turtlesim1</u> movements

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            namespace='turtlesim1',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            namespace='turtlesim2',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            executable='mimic',
            name='mimic',
            remappings=[
                ('/input/pose', '/turtlesim1/turtle1/pose'),
                ('/output/cmd_vel', '/turtlesim2/turtle1/cmd_vel'),
            ]
        )
    ])
```

Frames of Reference

- ROS standardises the transformation model between different coordinate frames of reference.
- Right Hand Rule, X forward (XYZ ↔ RGB)
- Tree structure:
  - /map
    - /base_link
      - /base_footprint
      - /laser



- Example: laser detected object is relative to **laser** frame. Need to transform to **map** coordinate to know where it is on the map.
- https://docs.ros.org/en/humble/Tutorials/Tf2/Tf2-Main.html#tf2main

Many Different Sensors

- Laser Scanner
- Camera
- IR Cameras
- Depth Cameras
- Motor
- Pressure Sensor
- Compass
- Accelerometer
- IMU (Inertial Measurement Unit) – detects linear acceleration using accelerometer and rotation using gyroscope
- Audio

ROS provides standardised data structures for some of these sensors.

Laser Scanners

- A laser is rotated through a plane
- Distance (& intensity) measurements taken periodically
- 180-270 degrees

```
sensor_msgs/LaserScan

std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

Cameras

- Stream images
- Various encodings used (RGB, Mono, UYVY, Bayer)
- ROS has no conversion functions

```
sensor_msgs/Image

std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

```
#include <sensor_msgs/image_encodings.h>
```

Depth Cameras

- Usually produce Mono16 images
- Typically turned into point clouds
- Depth measurements can be radial or axial

```
sensor_msgs/PointCloud

std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Point32[] points
  float32 x
  float32 y
  float32 z
sensor_msgs/ChannelFloat32[] channels
  string name
  float32[] values
```

Motor Positions

- Many motors report their positions
- Used to produce transformations between frames of reference

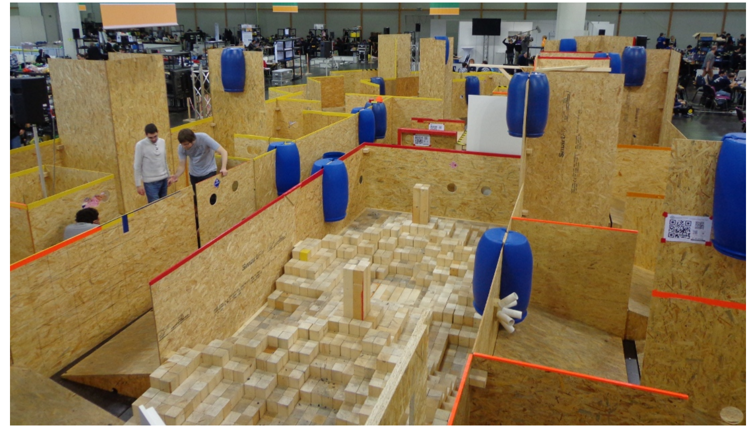
```
sensor_msgs/JointState

std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string[] name
float64[] position
float64[] velocity
float64[] effort
```

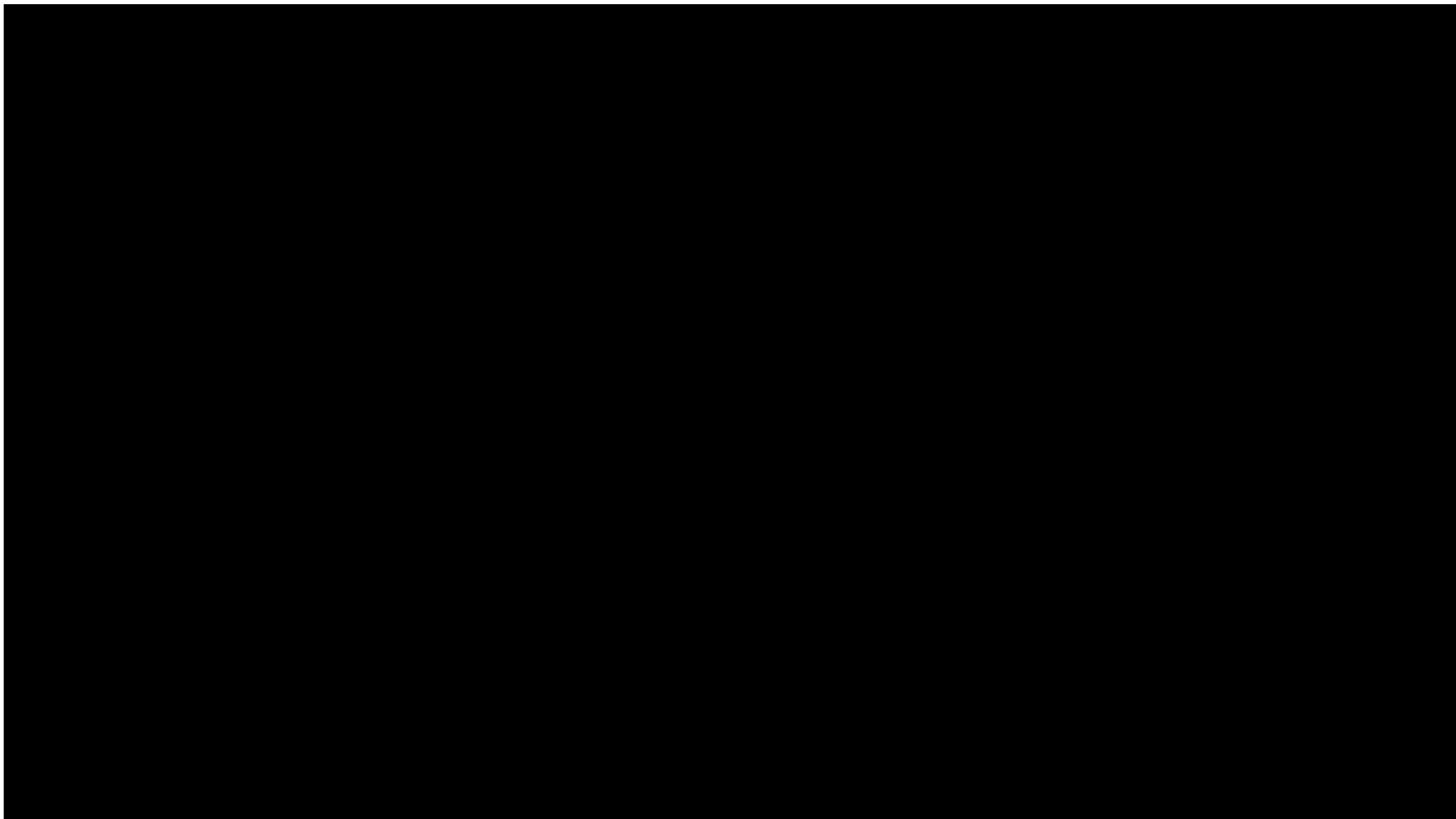
Lab Exercise

- Setup up connection to your robot
- Test that you can:
    - teleoperate
    - use Rviz to see state of robot
    - try adding camera image
    - run cartographer
- Begin working on wall follower

RoboCup



RoboCup@Home



RoboCup@Home

UT Austin Villa

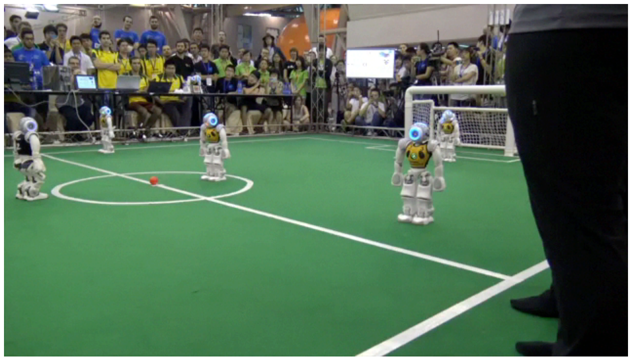
RoboCup@Home 2019 Qualification

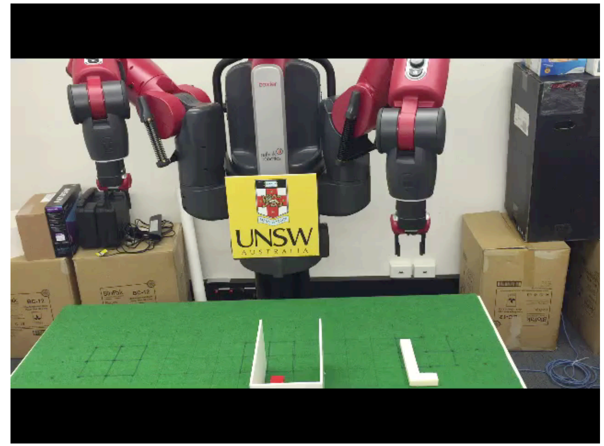
Project

Based on RoboCup@Home

1. Explore and map a home environment (first stage)
2. Use vision to recognise common objects in home
3. Given a goal, create and execute a plan (sequence of actions) to achieve the goal

Robotics Laboratory





Interested in RoboCup

- Soccer:
  - https://robotics.cse.unsw.edu.au/robocupsoccer-standard-platform-league/runswift-get-involved/
- @Home:
  - Talk to any of your COMP3431/9434 tutors