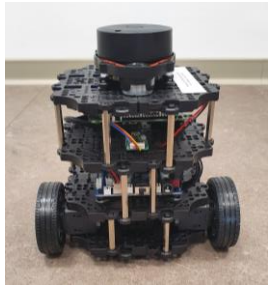
ROS 2 – Unity – Turtlebot3



Tutor: Francesco Trotti
francesco.trotti@univr.it

Agenda

- Background
- Working in ROS 2
- Working in Unity
- Simulator Unity – ROS 2
- Simulator bringup
- Turtlebot3 bringup

Background

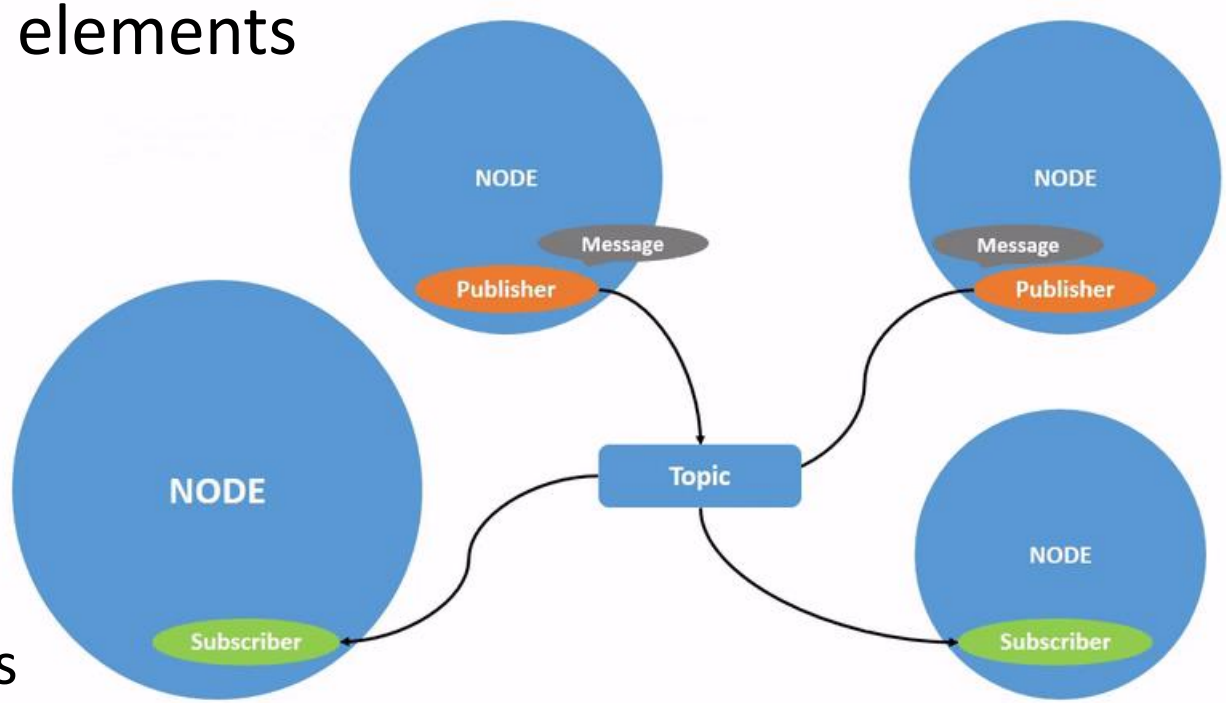
ROS 2
Unity

ROS 2

- ROS 2 is a framework that helps you building robotic applications
- ROS 2 includes several software, libraries and tools for robotics:
  - Visualizer - RVIZ
  - Simulator - Gazebo
  - Hardware interface for robots
  - Planner - MoveIt
- ROS 2 is a framework based on pub-sub system
- ROS 2 has a lot of distro, currently we use ROS 2 **Foxy**

ROS 2

- ROS 2 is based on two main elements
  - Nodes
  - Topics
- Node
  - Publisher
  - Subscriber
- Topics work like a bus
  - They connect different nodes
- Topics are not only point-to-point
- The nodes can be written in C++ or Python3



Unity

- Unity is a well known multiplatform game engine
- Unity integrates physics and graphics engines suitable for
  - Game
  - Physics simulation
  - Industrial application
  - AR/VR application
  - Robotic application
- In the last version, Unity supports the integration with ROS 2
  - Import URDF file
  - Publish/subscribe message over topic

Working in ROS 2

ROS 2 base

- ROS 2 works on a specific folder structure
  - A workspace: where all packages will be built
  - A src folder: where all packages with your nodes will be placed
- Workspace structure in ROS 2

```
colcon_ws/
├── build
├── install
├── log
└── src/
    ├── package_cpp/
    │   ├── CMakeLists.txt
    │   ├── package.xml
    │   ├── include
    │   └── src
    └── package_python/
        ├── setup.py
        ├── package.xml
        └── resource/
            └── package_python
```

ROS 2 base

Create a Python3 package

- Move to src folder
- Create package
  - *ros2 pkg create --build-type ament_python <package_name>*
- Move to workspace directory and build

(If you prefer you can write your package in C++ but in this course we will use python language)

ROS 2 node example (publisher)

Example of ROS 2 publisher node

- The node name is "minimal_publisher"
- A message of type string is published
- The topic name is "topic"
- The data message is the string "Hello, world"
- The loop frequency is 500 ms

```python
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


class MinimalPublisher(Node):

    def __init__(self):
        super().__init__('minimal_publisher')
        self.publisher_ = self.create_publisher(String, 'topic', 10)
        timer_period = 0.5  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0

    def timer_callback(self):
        msg = String()
        msg.data = 'Hello World: %d' % self.i
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
        self.i += 1


def main(args=None):
    rclpy.init(args=args)

    minimal_publisher = MinimalPublisher()

    rclpy.spin(minimal_publisher)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_publisher.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

ROS 2 node example (subscriber)

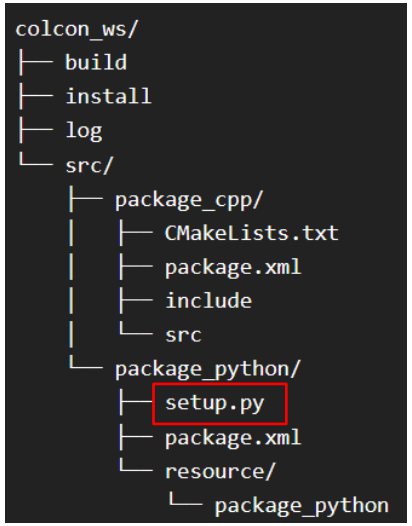
Example of ROS 2 subscriber node

- The node name is "minimal_subscriber"
- A message of type string is waited
- Subscribe to the topic called "topic"
- The callback "topic_callback" is called each time the message is published on topic

```python
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


class MinimalSubscriber(Node):

    def __init__(self):
        super().__init__('minimal_subscriber')
        self.subscription = self.create_subscription(
            String,
            'topic',
            self.listener_callback,
            10)
        self.subscription  # prevent unused variable warning

    def listener_callback(self, msg):
        self.get_logger().info('I heard: "%s"' % msg.data)


def main(args=None):
    rclpy.init(args=args)

    minimal_subscriber = MinimalSubscriber()

    rclpy.spin(minimal_subscriber)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_subscriber.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

ROS 2 setup.py

- To make the ROS 2 nodes executable it is necessary to write a setup.py file
- This file is autogenerated by ROS 2
- You can find it in your package folder
- In this file you can define the entry point for the interpreter and the name of the executable



ROS 2 setup.py example

- ROS 2 creates this file without entry points field
- To define the executable you have to add the entry point
- You have to specify where the main is
  - executable_name = package_name.file_name_with_main:main
  - Example: minimal_publisher_exec = minimal_publisher.minimal_publisher_node:main
- In this way ROS 2 creates the node executable

```python
from setuptools import setup

package_name = 'package_name'

setup(
    name=package_name,
    version='0.9.4',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    author='',
    author_email='',
    maintainer='',
    maintainer_email='',
    keywords=['ROS'],
    classifiers=[
        'Intended Audience :: Developers',
        'License :: OSI Approved :: Apache Software License',
        'Programming Language :: Python',
        'Topic :: Software Development',
    ],
    license='Apache License, Version 2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'executable_name = package_name.file_with_main_name:main',
        ],
    },
)
```

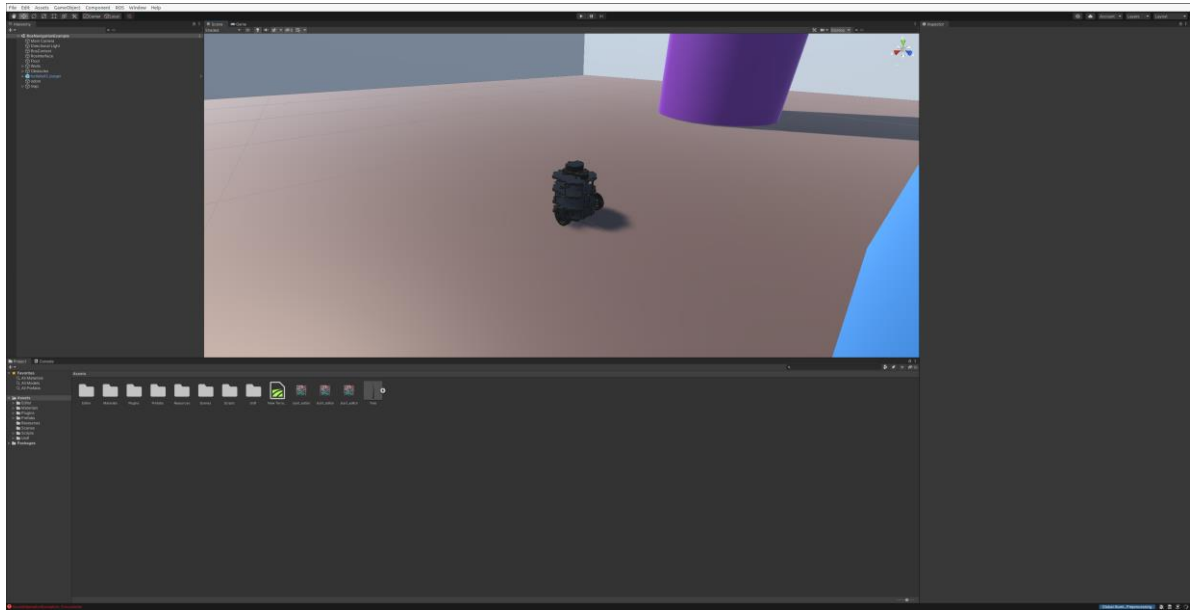
ROS 2 command line

- In ROS 2 there are some commands to inspect the state of the topics/nodes
  - To see the list of the active topics
    - *ros2 topic list*
  - *To read the message of one topic*
    - *ros2 topic echo "topic_name"*
  - *To know the info about one topic*
    - *ros2 topic info "topic_name"*
  - *To have a list of active nodes*
    - *ros2 node list*

Working in Unity

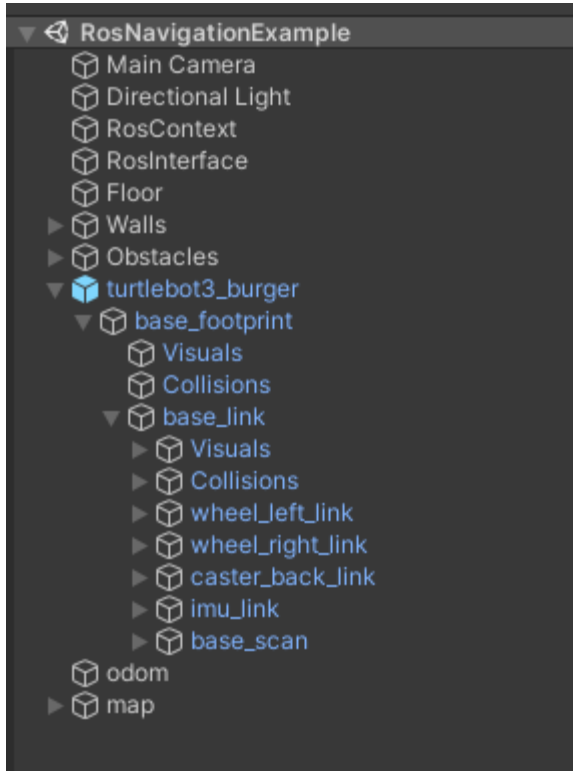
Unity - Environment

- Unity system is based on the creation of a scene where the physics solver works
- In Unity the objects in the scene are called Gameobject
- It is possible to attach some scripts to each Gameobject to modify its behavior in the world



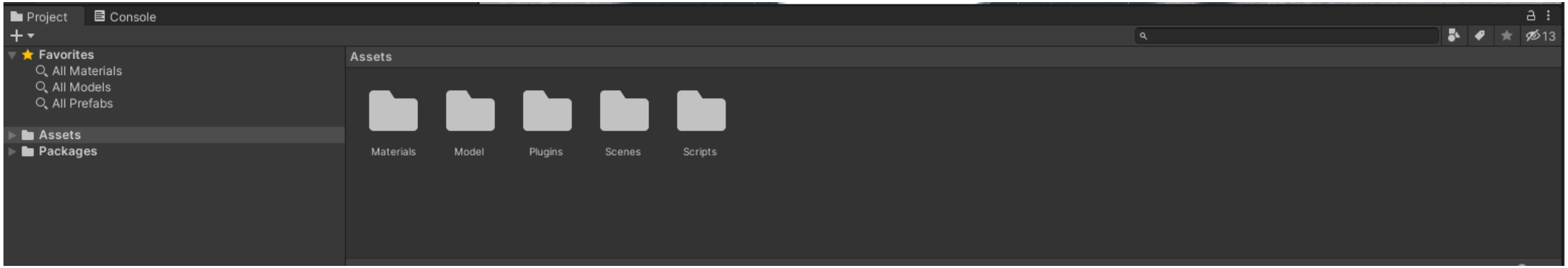
Unity - Hierarchy

- On the left side of the environment
  - GameObject tree placed in scene
  - Sub-tree between different GameObjects
  - Defining the parent frame for trasformation
  - Each GameObject has a world frame with position and rotation



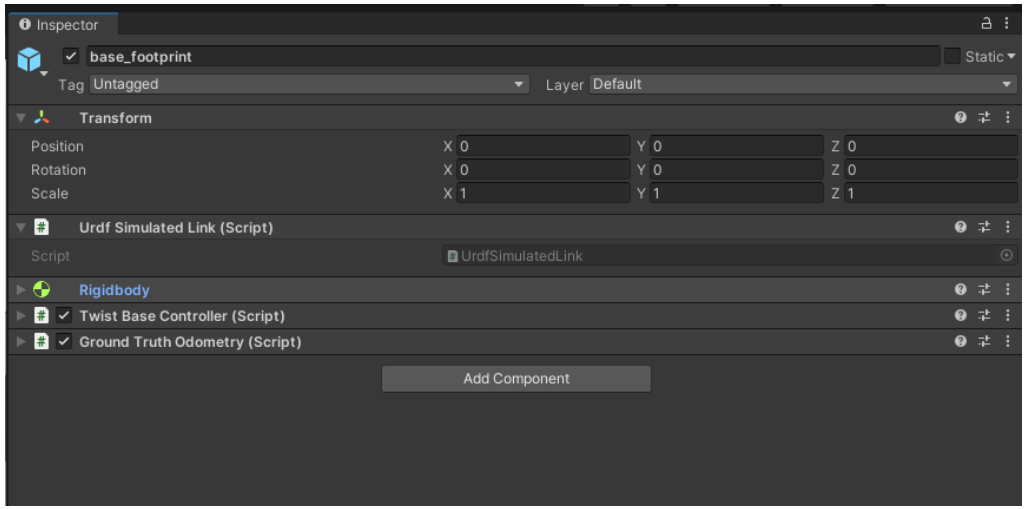
Unity - Directory

- On the bottom of environment
  - Structure of the Unity package
  - Main folders
    - Scene
    - Script
    - Model



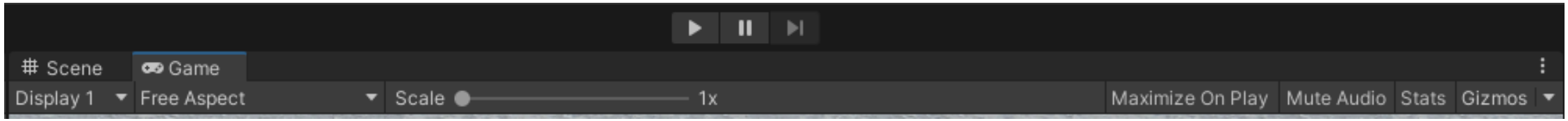
Unity - Inspector

- On the right of the environment
  - Position and rotation of selected GameObject
  - All property of GameObject
    - Material
    - Articulation/joint
    - Script
    - Collider
    - Physics settings



Unity – Run simulation

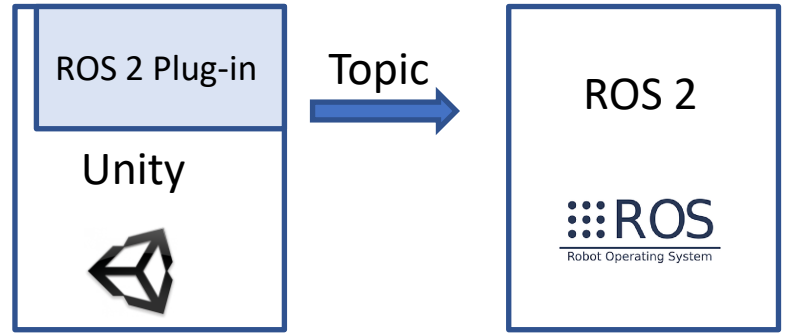
- On top of the environment
  - Play button to run the simulation
  - Two tab "scene" and "game"
    - Game is where the physics engine works when clicking play
    - Scene is the editor part to see the behavior of the GameObject
  - Camera view selector
    - The main camera is in "Display 1"



Integration Unity – ROS 2

Unity – ROS 2

- Unity includes a ROS2 version built-in
  - Can create publisher and subscriber node
  - All messages of ROS 2 are included
- ROS 2 can read topics of Unity and can create other topics
- It is possible to use ROS 2 tools with Unity topics



Unity – ROS 2

- Unity
  - Turtlebot3 waffle/burger robot
  - Odometry
  - Lidar driver
  - Motion controller (velocity)
- ROS 2
  - Navigation stack
  - SLAM stack
  - Teleoperation system

Turtlebot3 node

Turtlebot3 node

- Clone the Unity project from repository
  - https://gitlab.com/TrottiFrancesco/mobile_robotics_lab.git
- Run *./turtlebot3_nodes.sh*
- *source ~/.bashrc*
- Move to workspace directory
- colcon build
- . install/setup.bash

Simulation vs Reality

- Using this configuration you can test the same code on the simulator and on the real robotic platform.
- The simualtor emulates all behaviours of real turtlebot3
- The simulator emulates all topics and ROS2 component of real robot
- The algorithms test in simulator are easy to deploy to the real robot

Simulator bringup

Load and setup scene

- Clone the Unity project from repository
  - https://gitlab.com/TrottiFrancesco/mobile_robotics_lab.git
- In Unity Hub open the project cloned before
  - Move to Turtlebot3UnityROS2 folder and click "open"

- In Unity go to scene folder
- Drag turtlebot3.scene in hierarchy area
- Delete default Simplescene from hierarchy

Simulation bringup

- In Unity, click on play button to run the simulation
- In .bashrc comments
  - export ROS_DOMAIN_ID=<ros_id>
- In .bashrc add
  - export TURTLEBOT3_MODEL=burger
- reload .bashrc
- Run the command to see the active topics
  - */cmd_vel:* velocity command for turtlebot3
  - */scan:* laser scanner distance

Turtlebot3 teleoperation

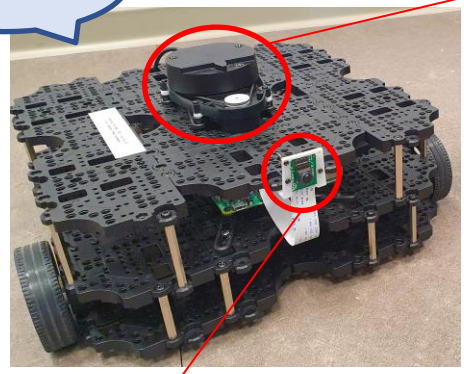
- In command line run the teleoperation node
  - ros2 run turtlebot3_teleop teleop_keyboard
  - With "WASD" you can move the turtlebot3 in the environment
- This node publishes a velocity commands on topic /cmd_vel
- To see the message on topic run
  - *ros2 topic echo /cmd_vel*
- To see the state of the robot and lidar collision in ROS2 you can use RVIZ2
  - ros2 launch turtlebot3_visualizer turtlebot3_visualizer

Turtlebot3 bringup

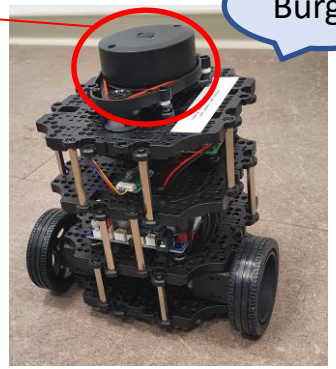
Turtlebot3

360 Lidar

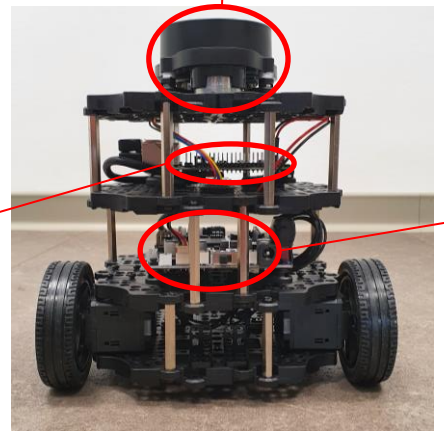
Waffle PI



Burger



PI camera

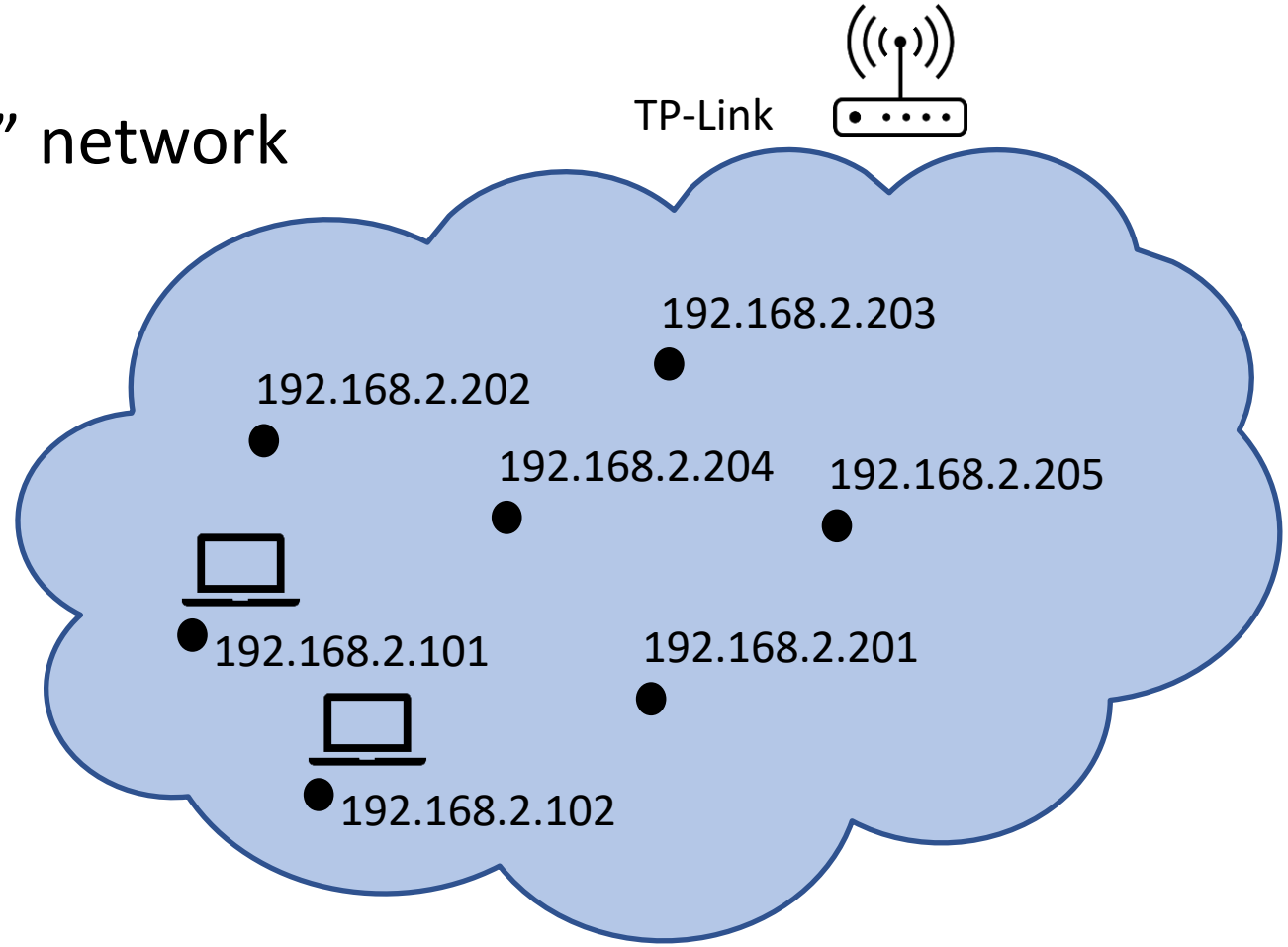


Raspberry 3 or 4
- Ubuntu 20.04
- ROS 2 Foxy

OpenCR board
- Motor driver
- Power system
- IMU
  - Gyroscope
  - Accelerometer

Network setup

- Connect your pc in “TP-Link” network
- Turtlebot3 IP
  - 192.168.2.201
  - 192.168.2.202
  - 192.168.2.203
  - 192.168.2.204
  - 192.168.2.205
- TB3 user: ubuntu
- TB3 password: turtlebot



ROS 2 setup

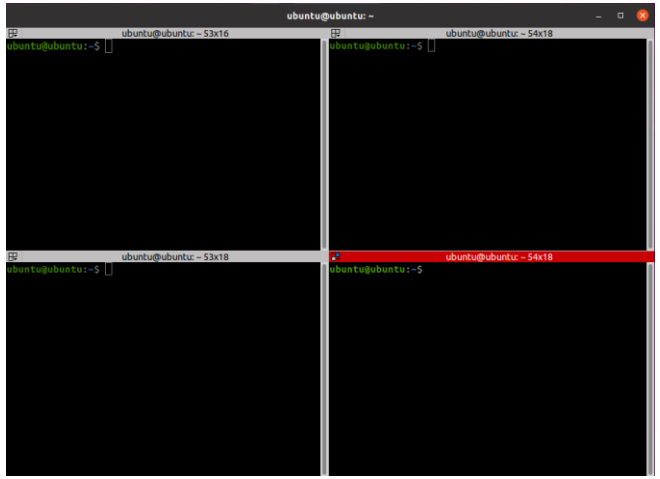
- Choose one turtlebot3
  - In your .bashrc change or add:
    - export ROS_DOMAIN_ID= ‘’number on robot label’’
    - If choose burger model set
      - export TURTLEBOT3_MODEL=burger
    - Else
      - export TURTLEBOT3_MODEL=waffle_pi
- Turn on the switch

Turtlebot3 bringup and teleop

- Enter with ssh in your robot
  - ssh ubuntu@<robot_ip>
- Run the bringup launch file
  - *ros2 launch turtlebot3_bringup robot.launch.py*
- If everything goes as planned in **your** machine you should see all topics of turtlebot3
- While bringup is running on turtlebot in **your** pc run
  - ros2 run turtlebot3_teleop teleop_keyboard
  - With "WASD" you can move the turtlebot3 in the environment

Help tools

- You could use visual studio code with sftp extension to facilitate the ROS 2 nodes implementation on turtlebot3
- You could download "terminator" to open multi terminal in same window
  - *sudo apt install terminator*



References

- Unity
  - https://unity.com/
- ROS 2
  - Installation
    - https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html#
  - Workspace and Package
    - https://docs.ros.org/en/foxy/Tutorials/Workspace/Creating-A-Workspace.html
    - https://docs.ros.org/en/foxy/Tutorials/Creating-Your-First-ROS2-Package.html
  - C++/Python3 example
    - https://docs.ros.org/en/foxy/Tutorials/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html
    - https://docs.ros.org/en/foxy/Tutorials/Writing-A-Simple-Py-Publisher-And-Subscriber.html