Robot simulator

Unity – ROS 2

Francesco Trotti
francesco.trotti@univr.it

Agenda

- ROS 2
- Unity
- Integration Unity – ROS 2
- Simulator installation

Background

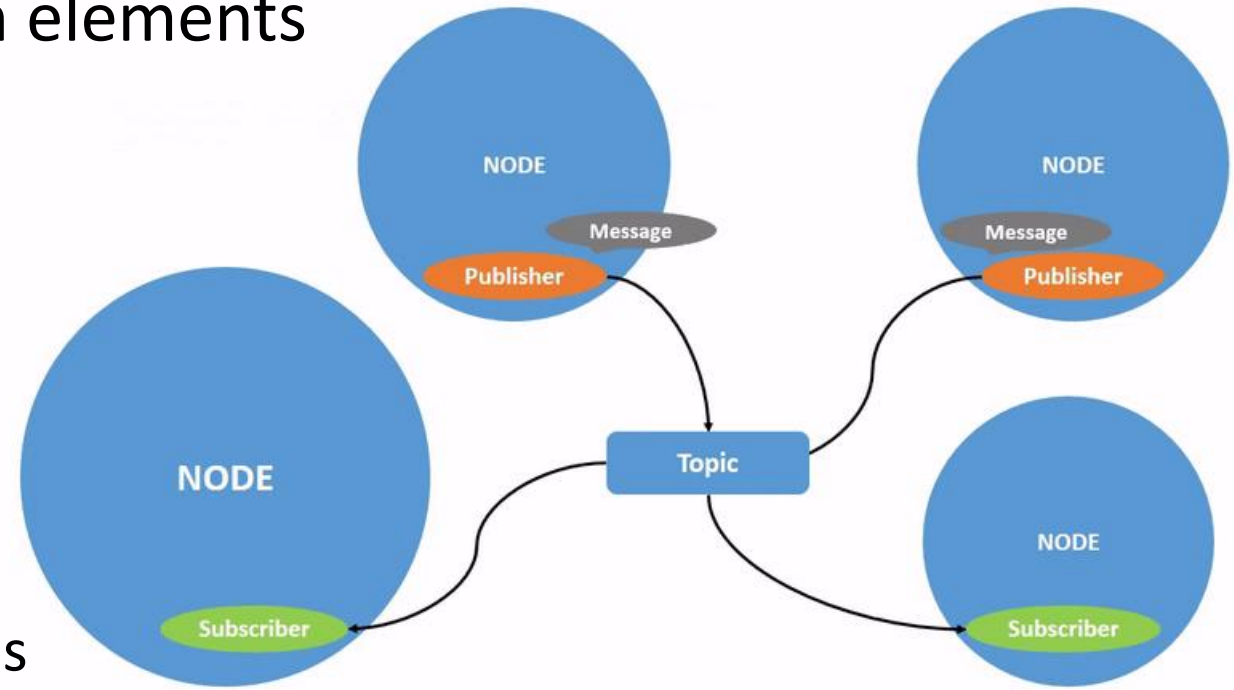
ROS 2
Unity

ROS 2

- ROS 2 is a framework that helps you building robot applications
- ROS 2 includes several softwares, 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 **Galactic**

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 engine 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
Robot Operating System

ROS 2 base

- Install ROS 2 and Colcon
- ROS 2 works on a specific folder structure
  - A workspace: where all package will be built
  - A src folder: where all package with your nodes will be placed
- Create a workspace in ROS 2:
  - Create a workspace and package folder
    - *mkdir –p /colcon_ws/src*
  - Build workspace and source the environment
    - *colcon build*
    - *. install/setup.bash*

ROS 2 base

Create a C++ package

- Move to src folder
- Create package
  - *ros2 pkg create --build-type ament_cmake <package_name>*

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

ROS 2 node example (publisher)

Example of ROS 2 publisher node

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

```
class MinimalPublisher : public rclcpp::Node
{
  public:
    MinimalPublisher()
    : Node("minimal_publisher"), count_(0)
    {
      publisher_ = this->create_publisher<std_msgs::msg::String>("topic", 10);
      timer_ = this->create_wall_timer(
      500ms, std::bind(&MinimalPublisher::timer_callback, this));
    }

  private:
    void timer_callback()
    {
      auto message = std_msgs::msg::String();
      message.data = "Hello, world! " + std::to_string(count_++);
      RCLCPP_INFO(this->get_logger(), "Publishing: '%s'", message.data.c_str());
      publisher_->publish(message);
    }
    rclcpp::TimerBase::SharedPtr timer_;
    rclcpp::Publisher<std_msgs::msg::String>::SharedPtr publisher_;
    size_t count_;
};

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<MinimalPublisher>());
  rclcpp::shutdown();
  return 0;
}
```

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"
- Run the callback "topic_callback" each time the message is published on topic

```cpp
class MinimalSubscriber : public rclcpp::Node
{
  public:
    MinimalSubscriber()
    : Node("minimal_subscriber")
    {
      subscription_ = this->create_subscription<std_msgs::msg::String>(
      "topic", 10, std::bind(&MinimalSubscriber::topic_callback, this, _1));
    }

  private:
    void topic_callback(const std_msgs::msg::String & msg) const
    {
      RCLCPP_INFO(this->get_logger(), "I heard: '%s'", msg.data.c_str());
    }
    rclcpp::Subscription<std_msgs::msg::String>::SharedPtr subscription_;
};

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<MinimalSubscriber>());
  rclcpp::shutdown();
  return 0;
}
```

ROS 2 cmake/setup

- To make the nodes executable it is necessary to write a cmake or setup file

C++ case

- Add in cmake file
  - The name of node and the name of executable
  - Add the link libraries for the compiler

Python3

- Add in setup.py file
  - The name of the entry point for the interpreter

- If you have to run more than one node you can use the launch file

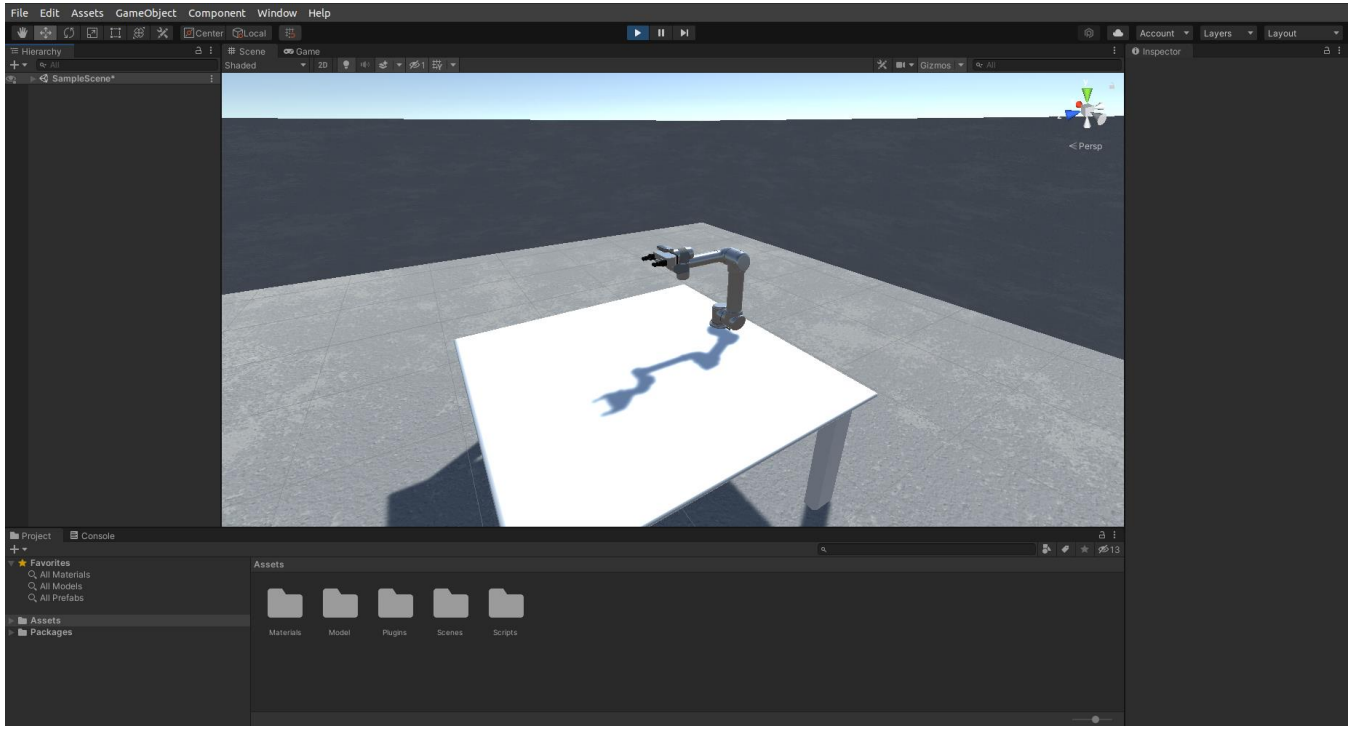
ROS 2 command line

- In ROS 2 there are some commands to see 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

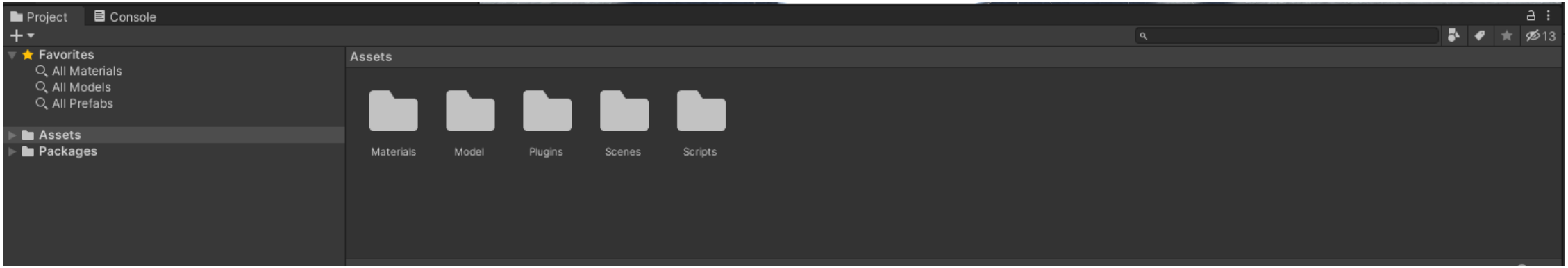


Unity - GameObject

- 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

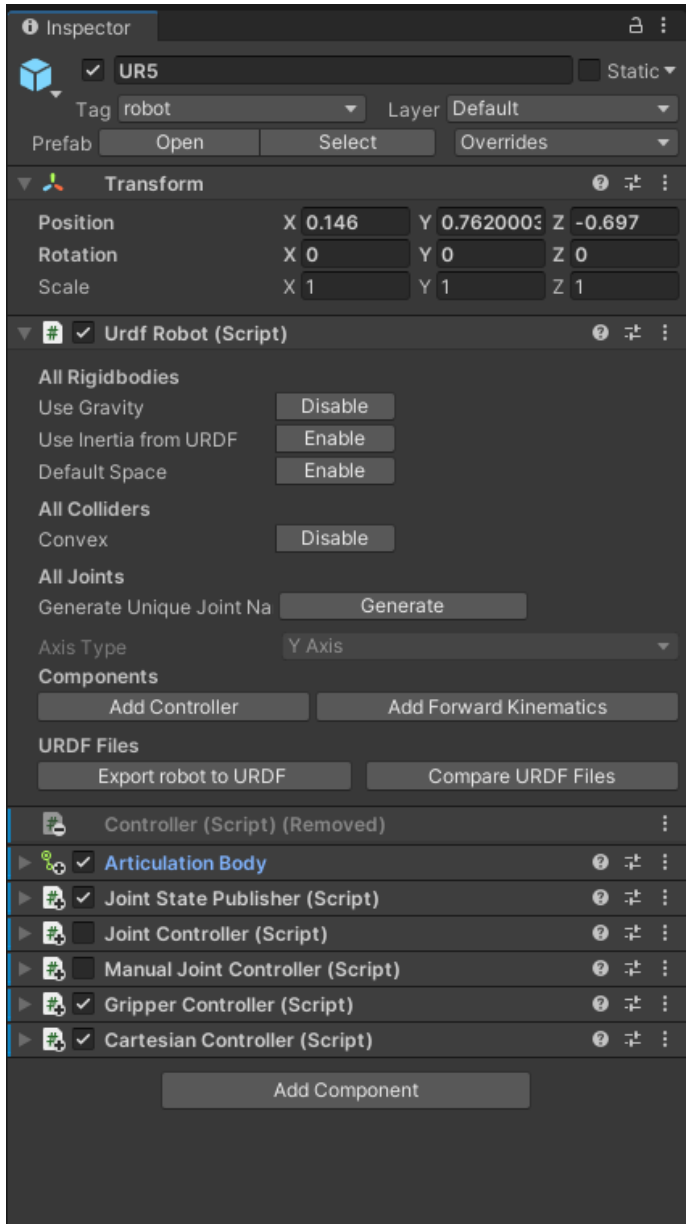
Unity - Directory

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



Unity - Inspector

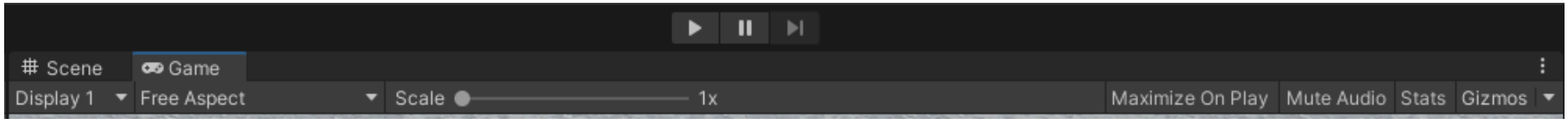
- On the right of the environment
  - Position and rotation of selected GameObject
  - All property of GameObject
    - Material
    - Articulation/joint
    - Script
    - Collider
    - Physics settings



UNIVERSITÀ di VERONA
Dipartimento di INFORMATICA

Unity – Run simulation

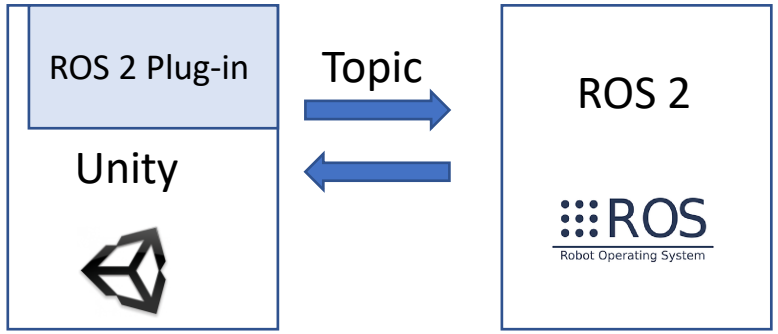
- On top of environment
  - Play button to run the simulation
  - Two tab "scene" and "game"
    - Game is where the physics engine works when clicking play
    - Scene is the editor part to see the behavior of the GameObject
  - Camera view selector
    - The main camera is in "Display 1"



Integration Unity – ROS 2

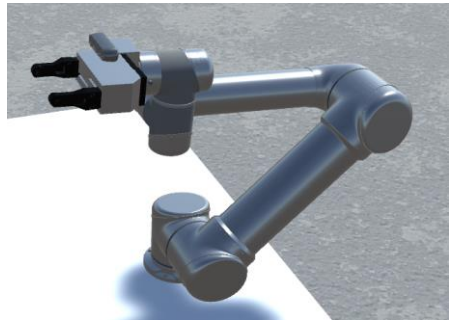
Unity – ROS 2

- Unity includes a ROS2 version build-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

- In Unity simulation
  - UR5 robot
  - WSG 50 gripper
  - Camera on arm
- In Unity there are some script to control the robot
  - Joint control
  - Cartesian control
  - Manual control
- In ROS 2 there are some nodes to control the robot

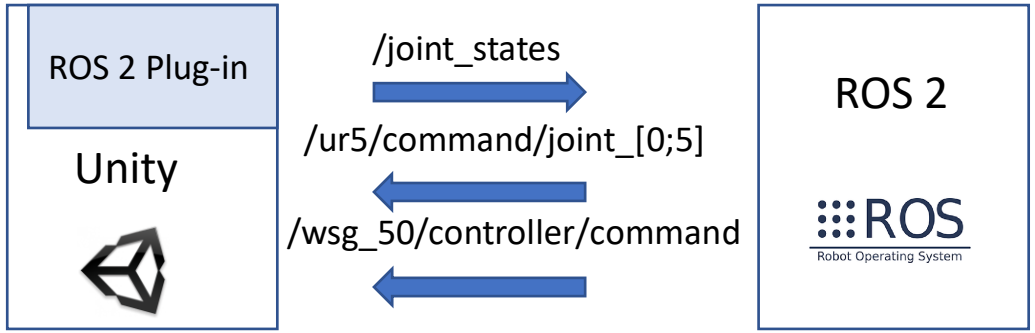


Unity – ROS 2 (Joint control)

- In Unity
  - On the UR5 GameObject enables scripts
    - Joint state publisher
    - Joint control
    - Gripper Control
- In ROS 2
  - Launch file
    - *ros2 launch rvc_bringup joint_bringup.launch.py*

Unity – ROS 2 (Joint control)

- Running the simulation on Unity
  - The robot goes in initial position
  - GUI with slider to change the joint configuration
- Running the launch file in ROS 2
  - Some topics appear
    - /joint_states
    - /ur5/command/joint_[0;5]
    - /wsg_50/controller/command
    - /ur5/ee_actual/pose

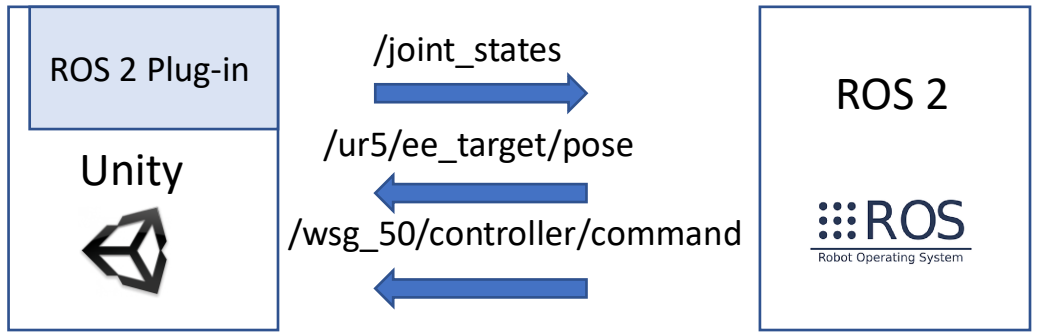


Unity – ROS 2 (Cartesian control)

- In Unity
  - On the UR5 GameObject enables scripts
    - Joint state publisher
    - Cartesian control
    - Gripper Control
- In ROS 2
  - Launch file
    - *ros2 launch rvc_bringup cartesian_bringup.launch.py*

Unity – ROS 2 (Cartesian control)

- Running the simulation on Unity
  - The robot goes in initial position
  - GUI with slider to change the joint configuration
- Running the launch file in ROS 2
  - Some topics appear
    - /joint_states
    - /wsg_50/controller/command
    - /ur5/ee_actual/pose
    - /ur5/ee_target/pose



Unity – ROS 2 Topics

- /joint_states
  - Standard topic for robotics application
  - Topic with joint name, position, velocity and effort
  - The message is a JointSatate type (sensor_msgs/JointState)
  - Publisher is Unity
  - Subsciber is ROS 2

Example JointState
- Heather
  - Frame id
  - Stamp
- Name[]
- Position[]
- Velocity[]
- Effort[]

Unity – ROS 2 Topics

- /wsg_50/controller/command
  - Topic to open and close the gripper
  - Boolean message (std_msgs/bool)
    - True open gripper
    - False close gripper
  - Publisher is ROS 2
  - Subscriber is Unity

- /ur5/command/joint_[0;5]
  - Standard topic for joint position controller
  - The message is a float with the joint position in radians (std_msgs/float32)
  - Publisher is ROS 2
  - Subscriber is Unity

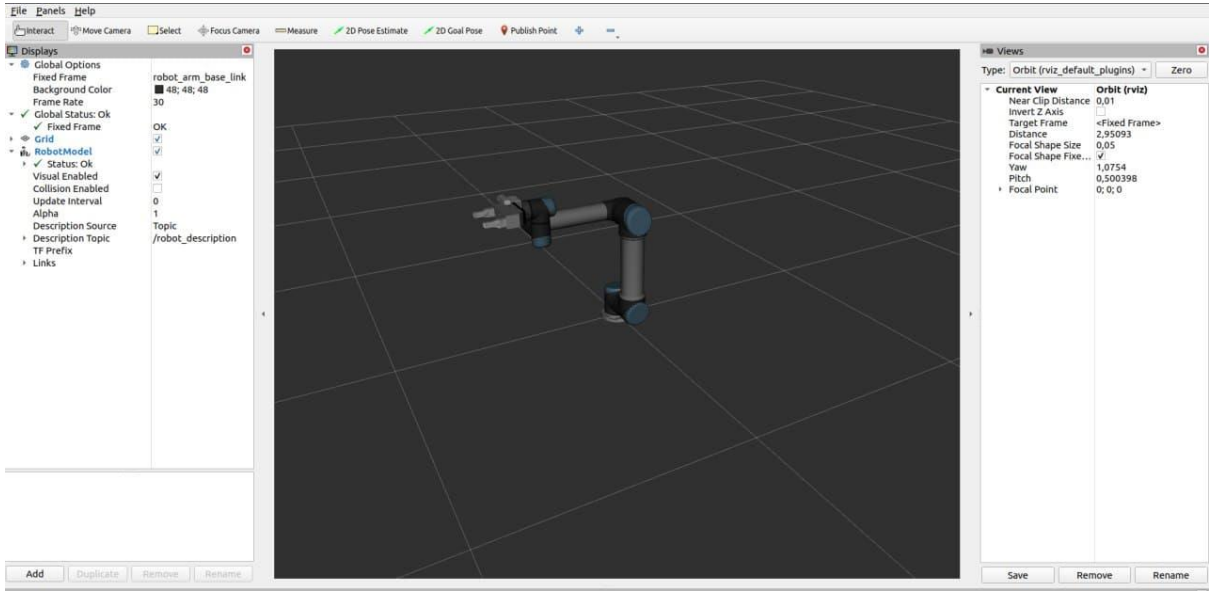
Unity – ROS 2 Topics

- /ur5/ee_actual/pose
  - Topic with actual end-effector pose (result of forward kinematic)
  - The message is a PoseStamped type (geometry_msgs/PoseStamped)
  - Publiser is ROS 2
- /ur5/ee_ target /pose
  - Topic with target end-effector pose (input for inverse kinematic)
  - The message is a PoseStamped type (geometry_msgs/PoseStamped)
  - Subscriber is ROS 2

Example PoseStamped
- Heather
- Pose
  - Position
    - x
    - y
    - z
  - Orientation
    - x
    - y
    - z
    - w

Unity – ROS 2 Tool

- To see the state of the robot in ROS 2, it is possibile to run RVIZ2
  - ros2 launch ur5_description ur5_wsg50.launch.py
- RVIZ 2 is only the ROS 2 visualizer
- RVIZ 2 subscribes to specific topic (e.g /joint_state) to show the state of robot



Installation and reference
Unity – ROS 2

Unity – ROS 2 Installation

Prerequisites:

- Ubuntu 20.04
- Unity 2020.3.22 Personal edition
- ROS 2 Galactic

- Install Unity, Unity hub and register on website
- Install ROS 2 via debian package

Unity – ROS 2 Test simulator

- Open Unity hub and add project
- Go to scene directory and drag it in GameObject section
- Click play button to run the simulation
- Open command line
- Source the environment (. install/setup.bash)
- Run *ros2 topic list*

References

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