



# **INTELLIGENT ROBOTICS**

## **Robot Operating System (ROS)**

Artificial Intelligence Technology Center (AITeC)

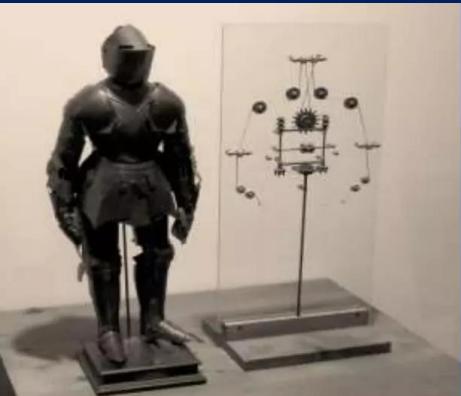
National Center for Physics (NCP)



**Autonomous AI & Decision Support Lab**



# History - Evolution



Da Vinci sketched the first humanoid robot in 1495



George Devol and Joseph Engelberger formed the world's first robot company in 1956



The Soviet Union launches the first artificial orbiting satellite in 1957



Unimate, the first industrial robot was designed in 1961



The first artificial robotic arm to be controlled by computer was designed at Rancho Los Amigos Hospital in Downey in 1963



First mobile robot controlled by artificial intelligence was designed in 1970

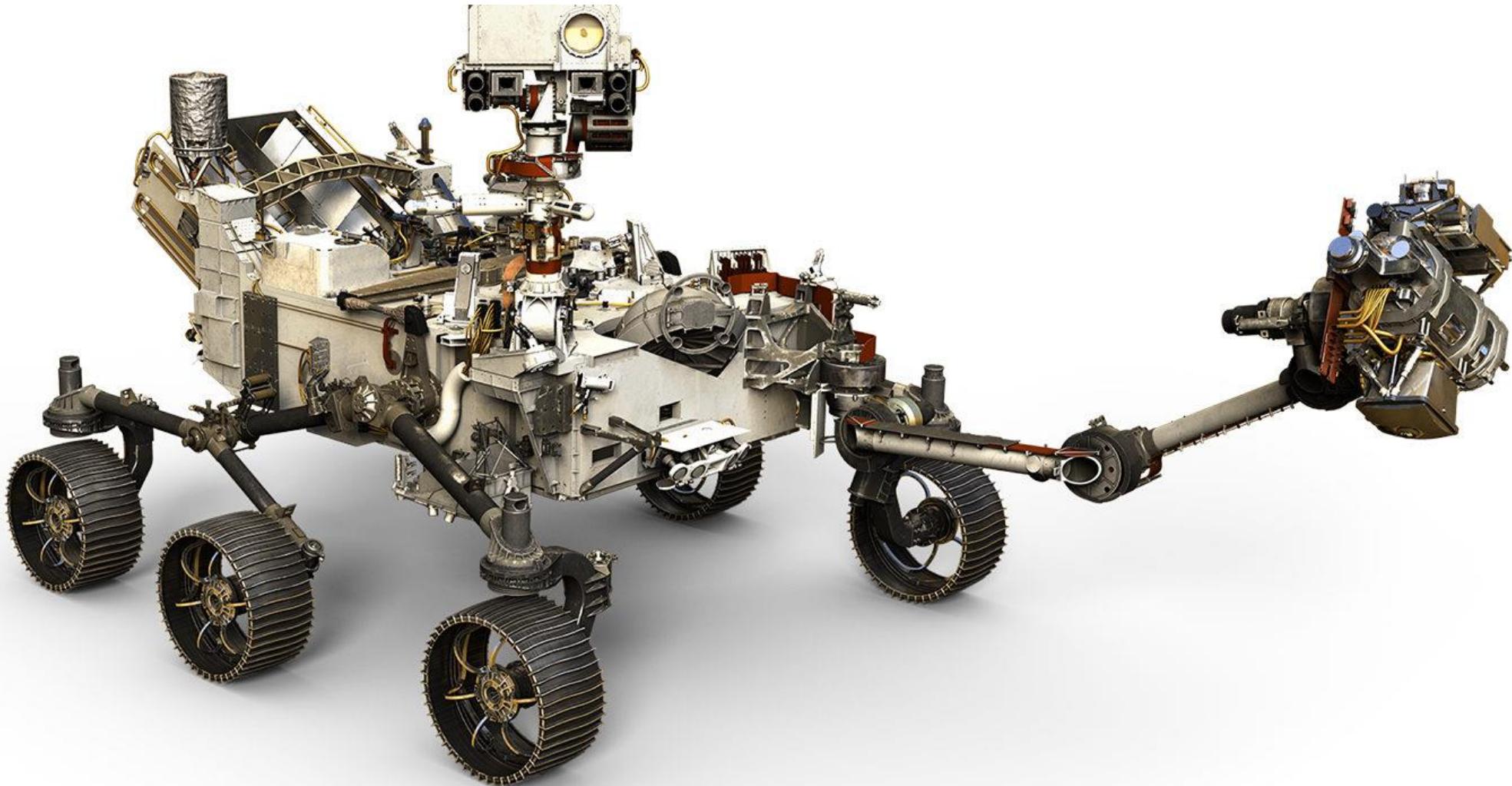


Mars Pathfinder's sojourner rover landed on Mars for the first time in 1977



Honda debuts a new humanoid robot called Asimo in 2002

# History - Evolution



# Robotics

- “Robot” is a broad term for any reprogrammable machine capable of..
  - Sensing the surrounding environment
  - Planning responses to inputs
  - Executing physical actions in the real world

*Robotic Arm*



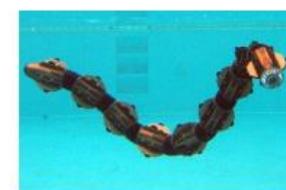
*Walking Robots*



*Wheeled Robots*



*Underwater Vehicles*



*Aerial Vehicles*

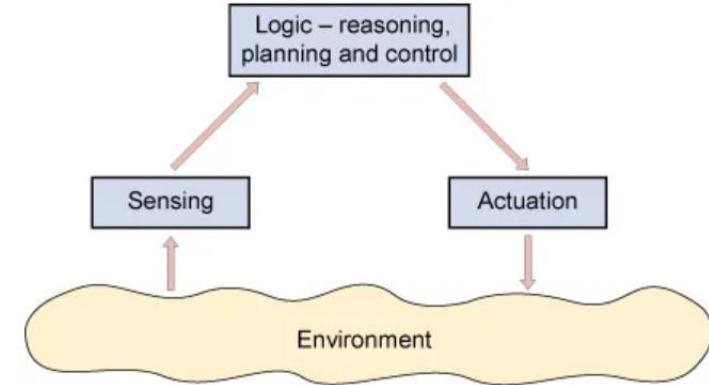
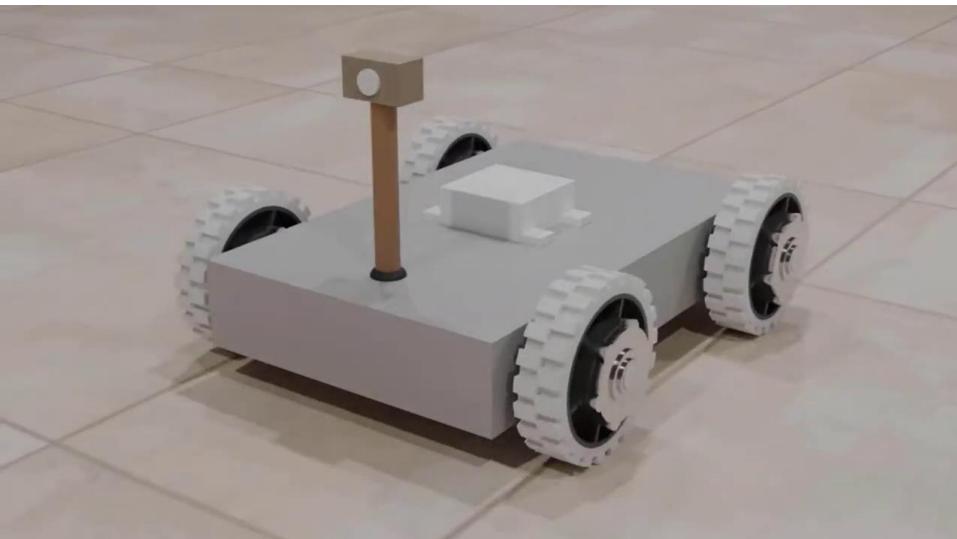


*Swarm Robots*



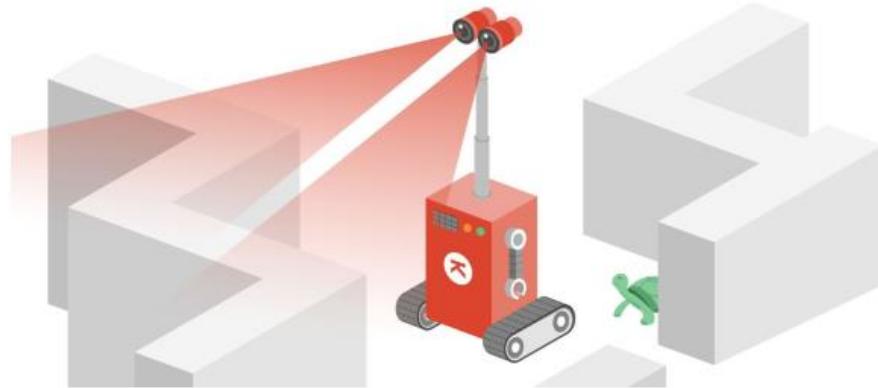
# Robot

- A robot is any system that can perceive the environment i.e, its surroundings (using sensors), take decisions based on the state of the environment(using computation and algorithms) and is able to execute the instructions generated(using actuators).

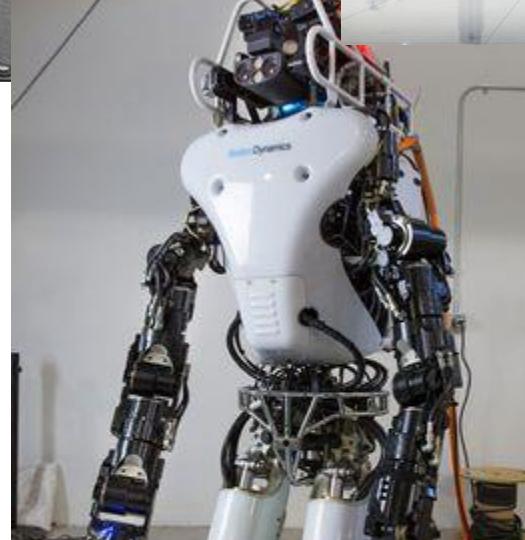


# Subfields

- The robotics field is somewhat unique in that it requires expertise in many fields across the engineering, sciences, and arts disciplines.
- **Kinematics:**
  - Describing the position and motion of components relative to each other without considering forces.
- **Dynamics:**
  - Describing the motion of components relative to each other using forces (Newtonian mechanics).
- **Localization:**
  - Where the robotic system is relative to the ‘world’ environment.
- **Mapping:**
  - Process of sensing and building a ‘map’ of the world environment.

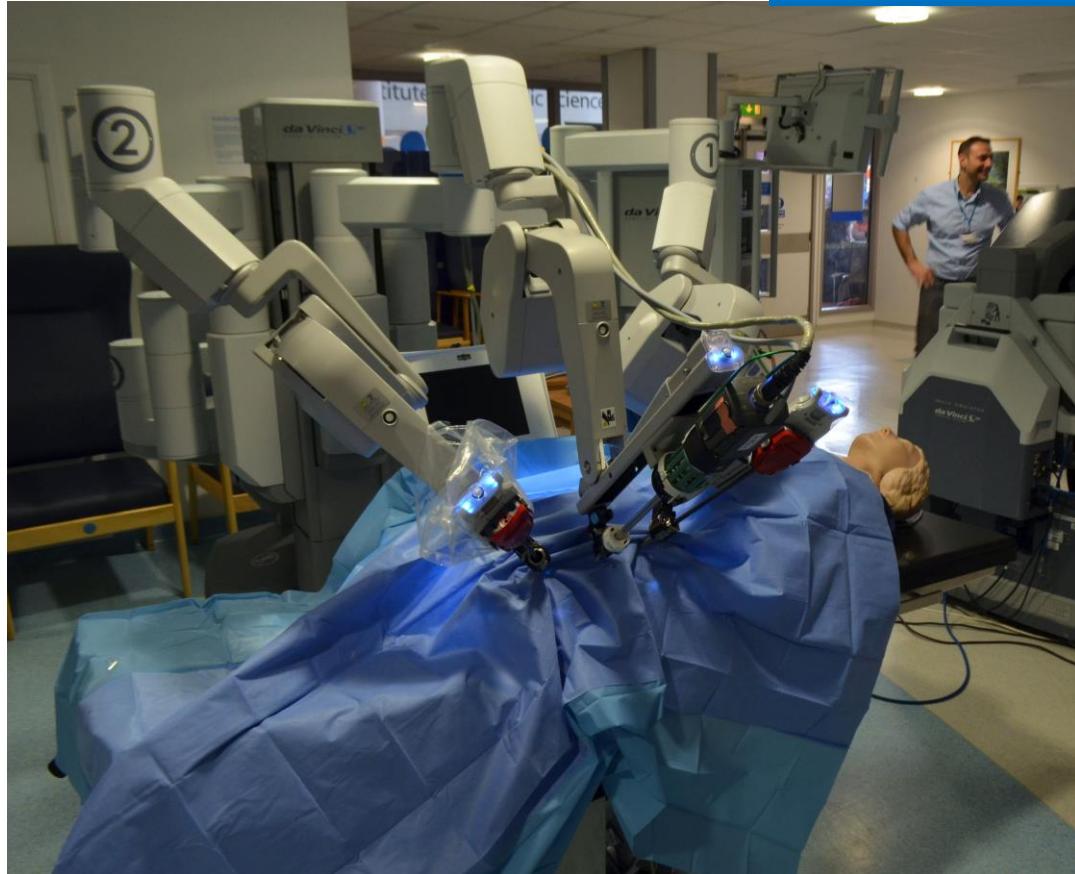


# Types of Robots (e.g., industrial, service, humanoid)



# Applications of Robotics

## Healthcare



# Applications of Robots

## Manufacturing



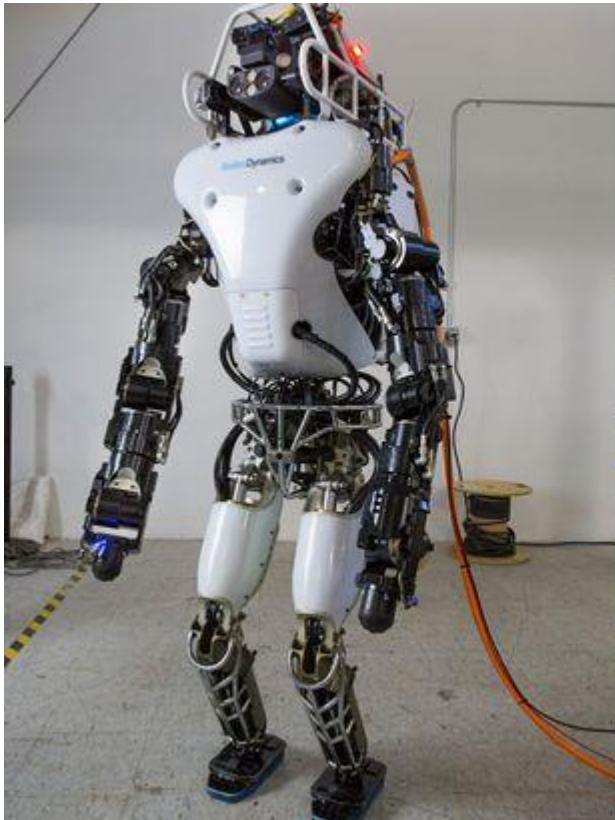
# Applications of Robots

## Automotive



# Applications of Robots

## Military



# **Applications**

- Manufacturing and Industry:
- Healthcare
- Agriculture
- Warehouse
- Automotive
- Space Exploration
- Military
- Construction

# In Robotics, before ROS

- Lack of standards
- Little code reusability
- Keeping reinventing (or rewriting) device drivers, access to robot's interfaces, management of onboard processes, inter-process communication protocols, ...
- Keeping re-coding standard algorithms
- New robot in the lab (or in the factory), start re-coding (mostly) from scratch



# Robot Operating System - History

- Originally developed in 2007 at the Stanford Artificial Intelligence Laboratory
- Since 2013 managed by OSRF (Open Source Robotics Foundation)
- Today used by many robots, universities and companies
- De facto standard for robot programming



# Robot operating System

- ROS (Robot Operating System) is a software framework that provides packages, and tools to better program, operate, and control robots.
- “*ROS is an open-source, meta-operating system for your robot.*
- *It provides the user with services such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.*
- *It also provides tools and libraries for obtaining, building, writing, and running code across multiple robots and multiple computers*“.

# Benefits

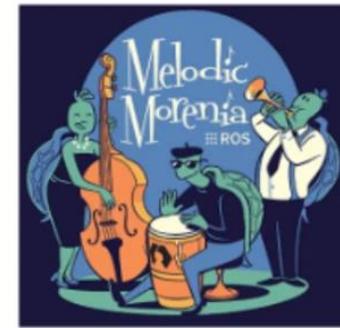
- **Modularity:** ROS encourages a modular approach to software development. You can break down complex robot systems into smaller, manageable components called nodes.
- **Middleware:** ROS provides middleware functionalities for communication between nodes, which simplifies inter-process communication.
- **Large Ecosystem:** ROS has a large and active community, resulting in a vast ecosystem of libraries, packages, and tools that are readily available to developers.

# Benefits

- **Robustness:** ROS provides error-handling mechanisms, logging, and diagnostic tools to help developers create reliable and robust robot applications.

# ROS Versions

ROS



ROS2



22.04  
  
ubuntu

20.04  
  
ubuntu

18.04  
  
ubuntu

# ROS1 VS ROS2

## ROS 1

Provide the software tools for users who need to do R&D projects on different robots

## Languages



C++ 03  
C++ 11



Python 2

## ROS 2

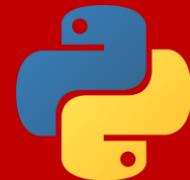
New requirements:

- Robot security
- Realtime control
- Increased distributed processing

## Languages



C++ 11  
C++ 14  
C++ 17



Python 3

# ROS1 VS ROS2



## Inter Communication

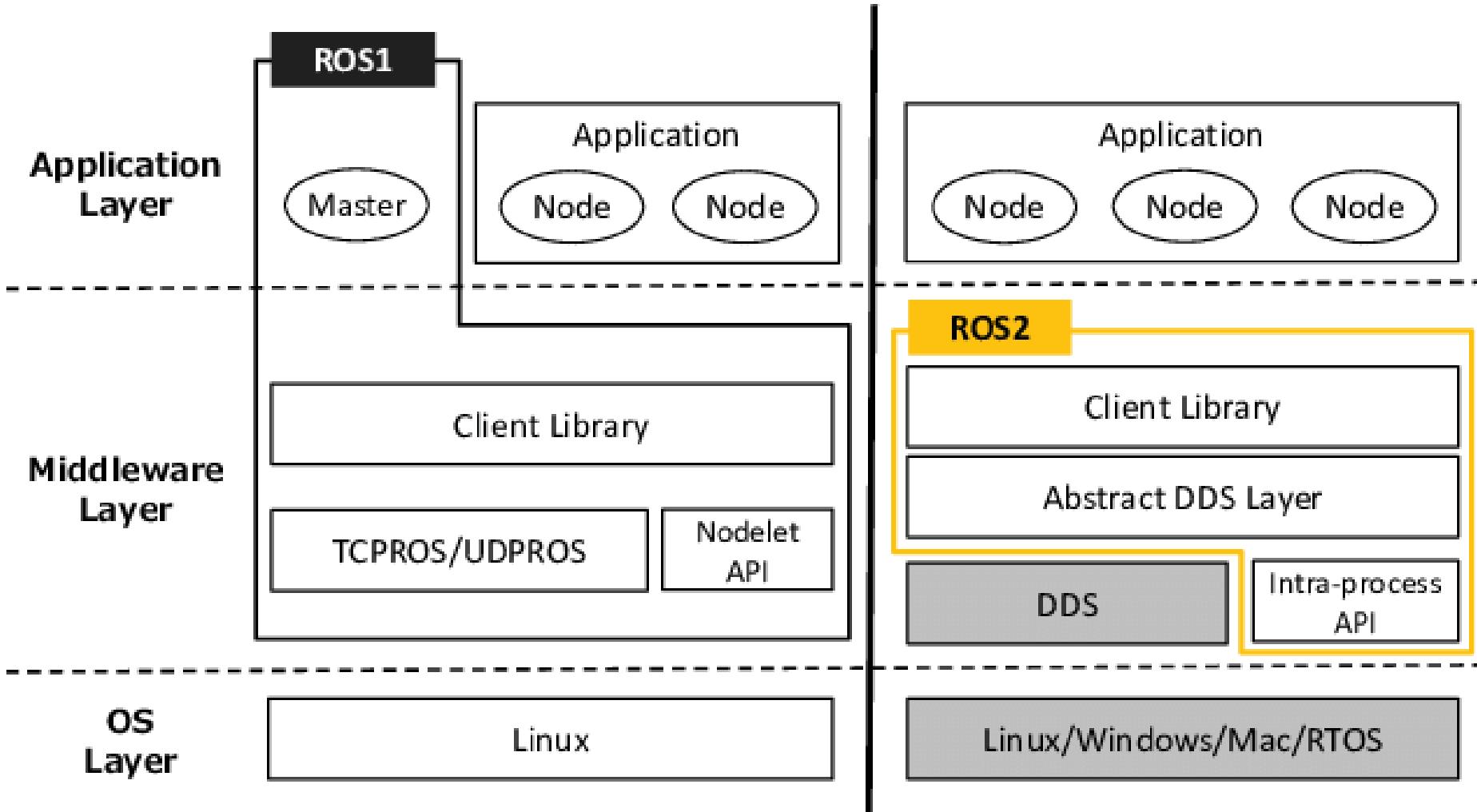
**Using  
rosmaster**

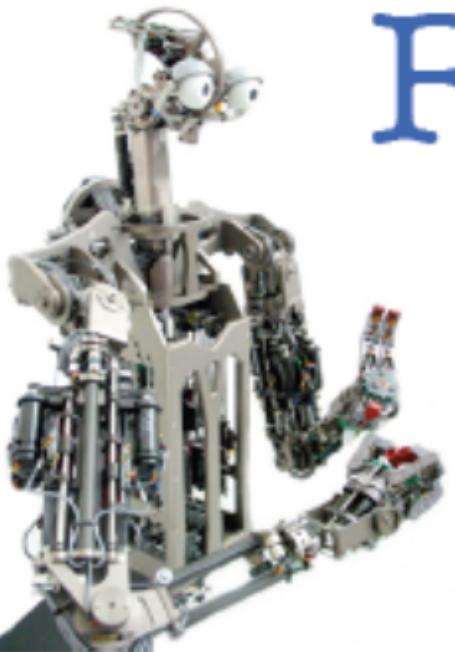
Nodes communicate  
through rosmaster

## Inter Communication

Allow nodes to communicate with each other  
in a peer-to-peer fashion without the need  
for a master

# ROS1 vs ROS2





# ROS

navigation

task executive

visualization

simulation

perception

control

planning

data logging

message passing

device drivers

real-time capabilities

web browser

# OS

email client

window manager

memory management

process management

scheduler

device drivers

file system



# ROS Benefits

Communication  
Between Process

Hardware  
Abstraction



Package  
Management

Low-Level  
Device Control

# Hardware Abstraction



# Low Level Device Control



# Message Passing

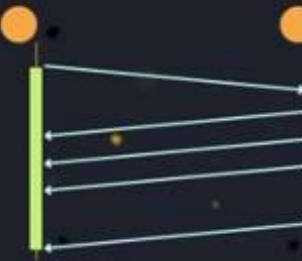
- Topic



- Service



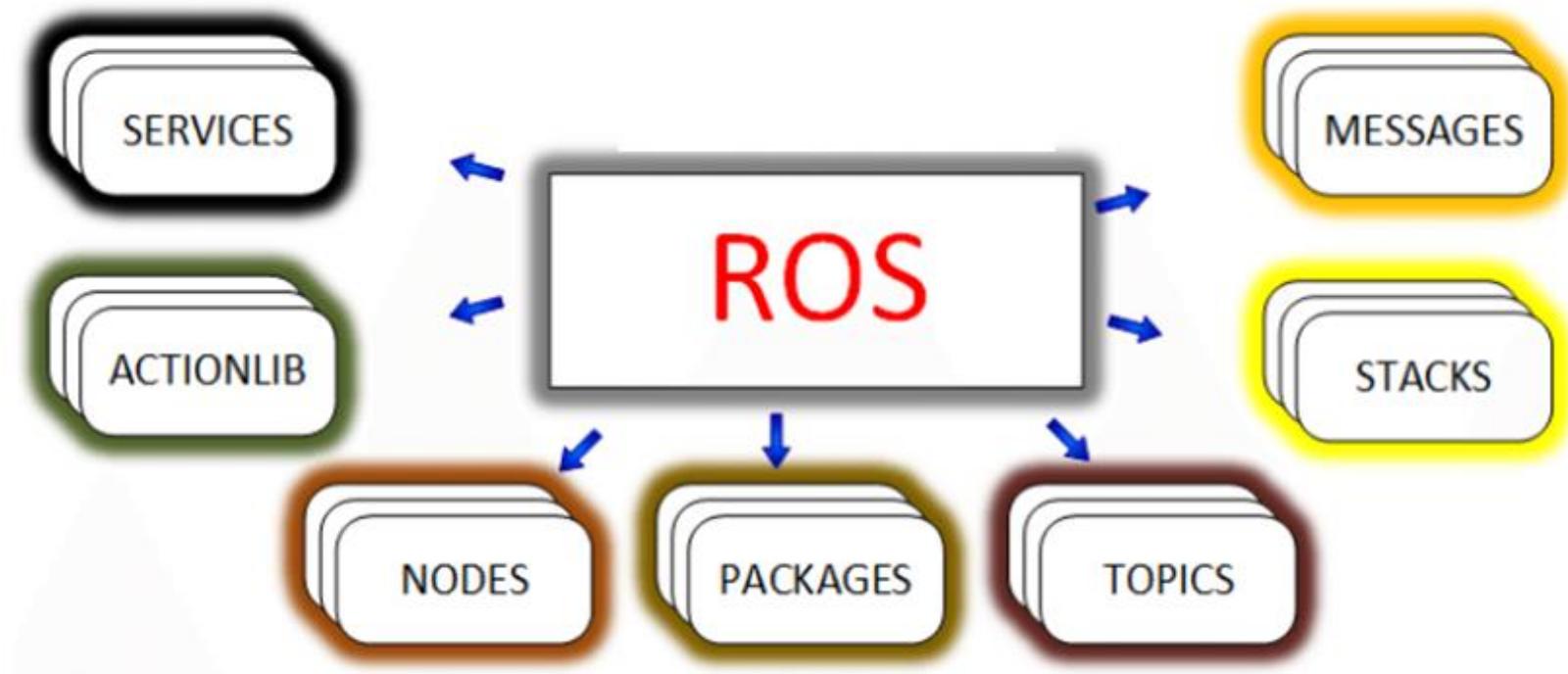
- Action



# Package Management



# Architecture



# ROS Nodes

- **Process that performs computation.** It is an **executable program** running inside your application.
- Each node in ROS should be responsible for a single, modular purpose, e.g. controlling the wheel motors or publishing the sensor data from a laser range-finder. Each node can send and receive data from other nodes via topics, services, actions, or parameters.

Nodes are written using a **ROS client library**

- **rclcpp** – C++ client library
- **rclpy** – python client library

A node is a ROS program that uses ROS's middleware for communications.

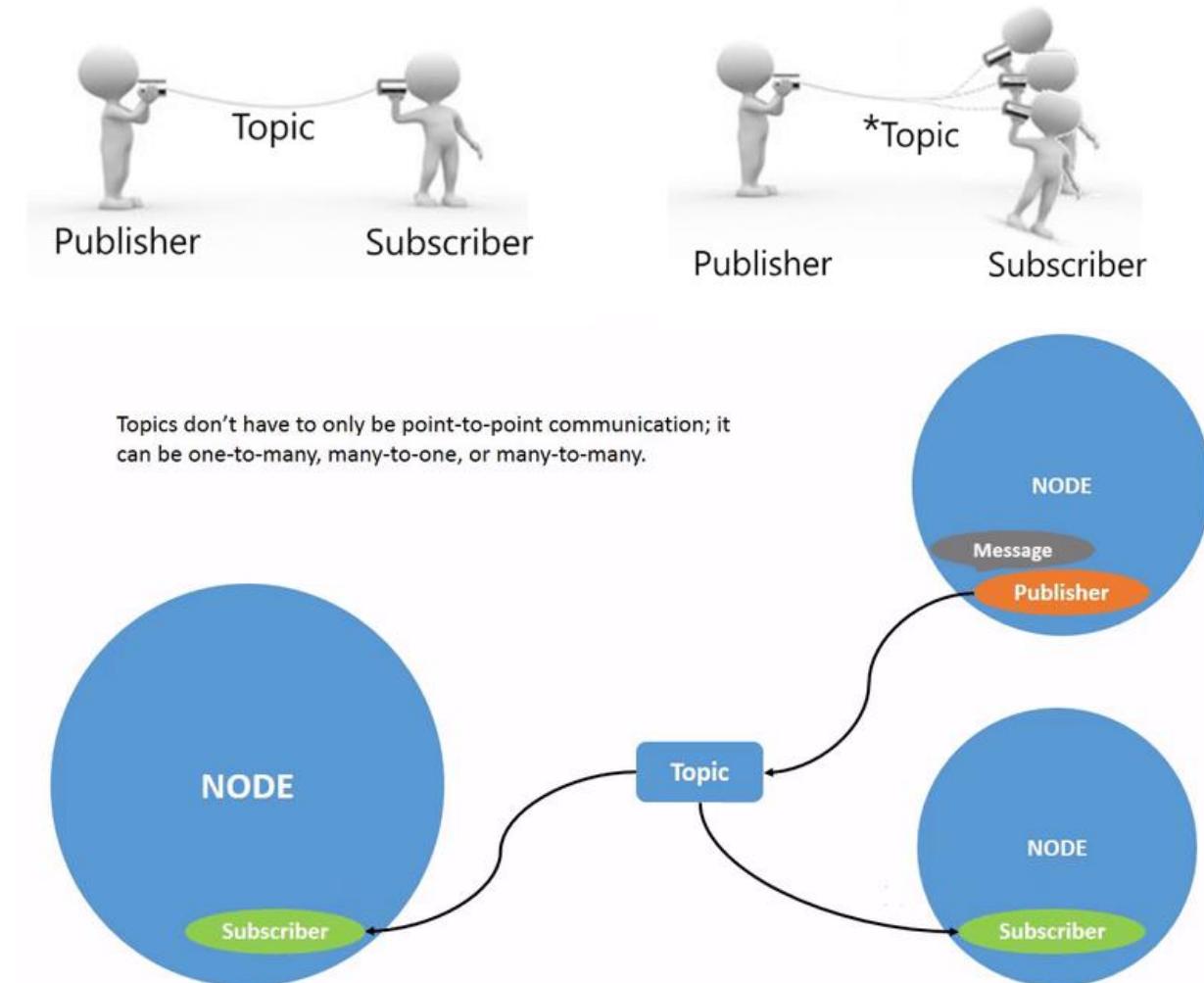
A node can be launched independently of other nodes and in any order among launches of other nodes.

Many nodes can run on the same computer, or nodes may be distributed across a network of computers.

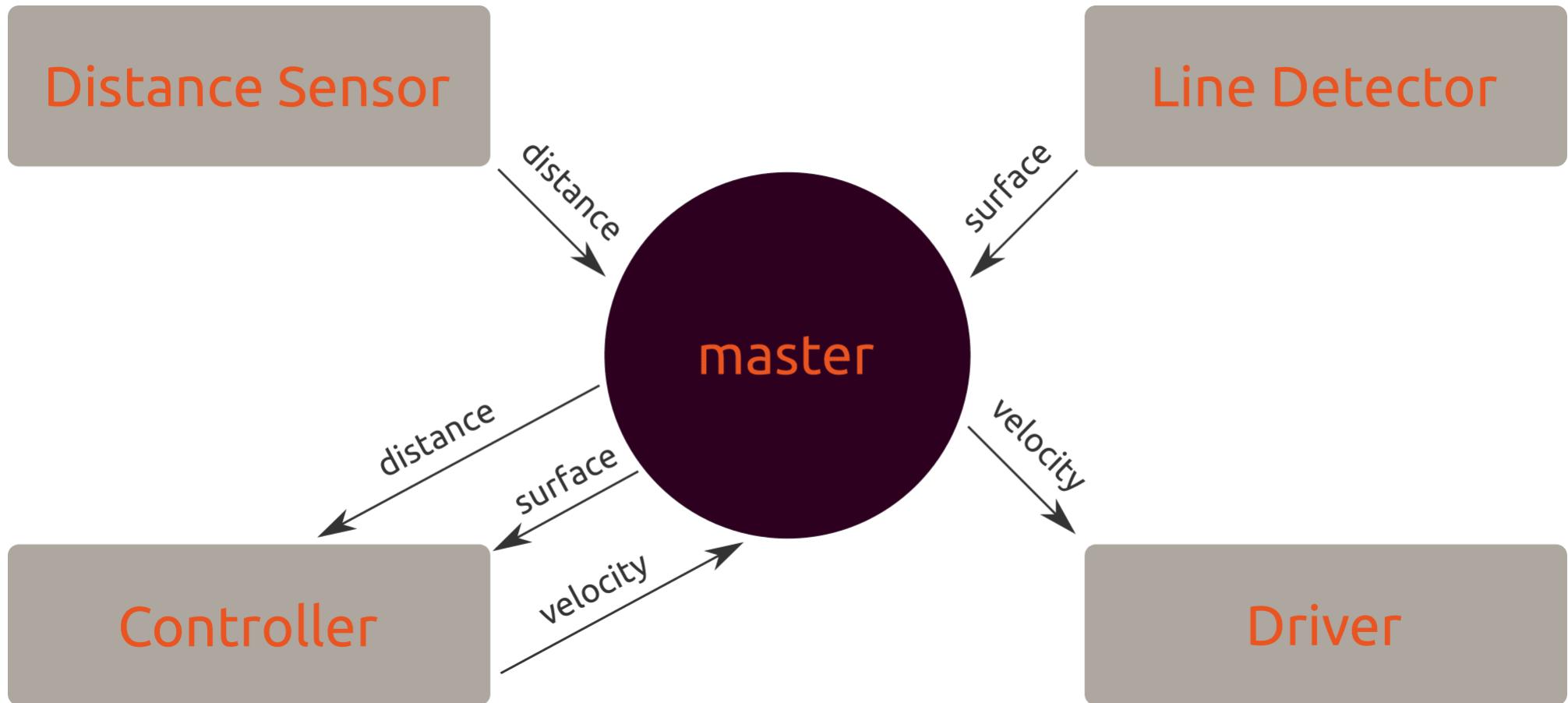
A node is useful only if it can communicate with other nodes and ultimately with sensors and actuators.

# ROS Nodes and Topics

- A node may publish data to any number of topics and simultaneously have subscriptions to any number of topics.
- Nodes communicate with each other by publishing messages to topics
- Topics are communication channels that nodes use to talk to each other.
- Messages are the information robots send to each other through topics.
- Reading data from a sensor
- Sending control signals to a motor

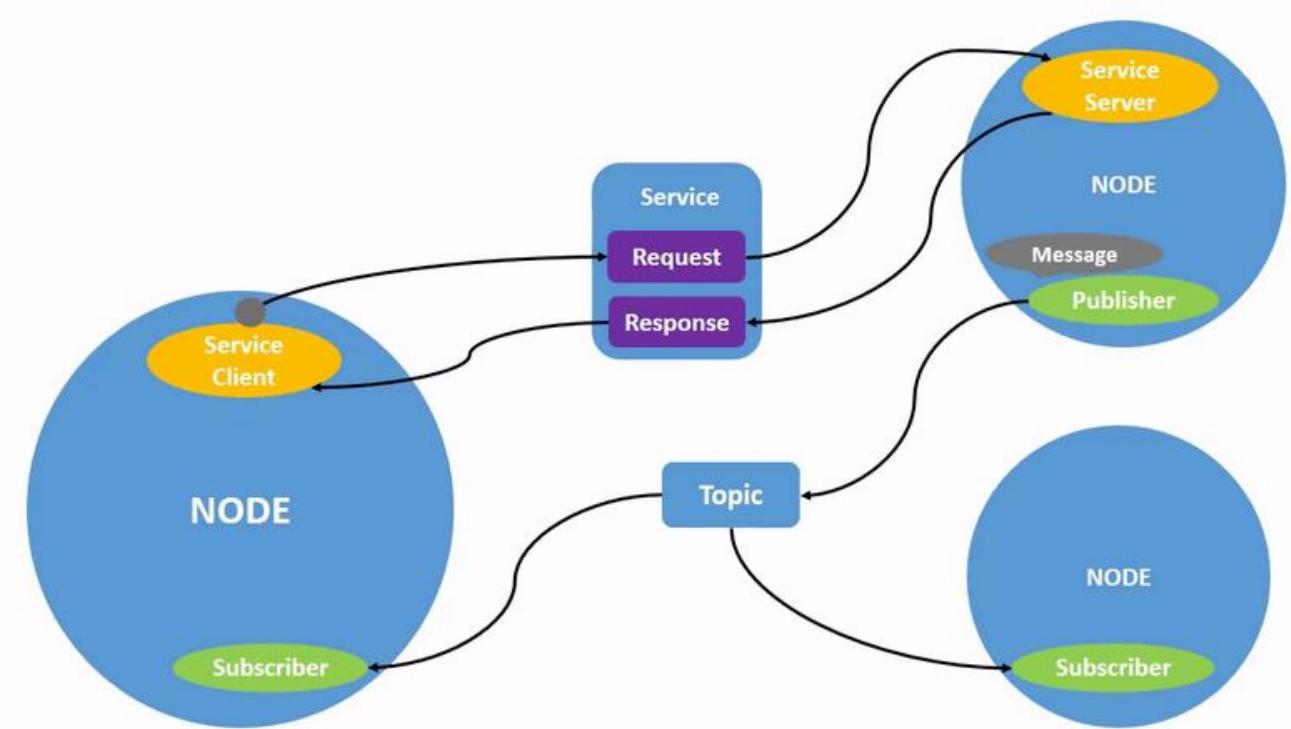


# ROS Nodes

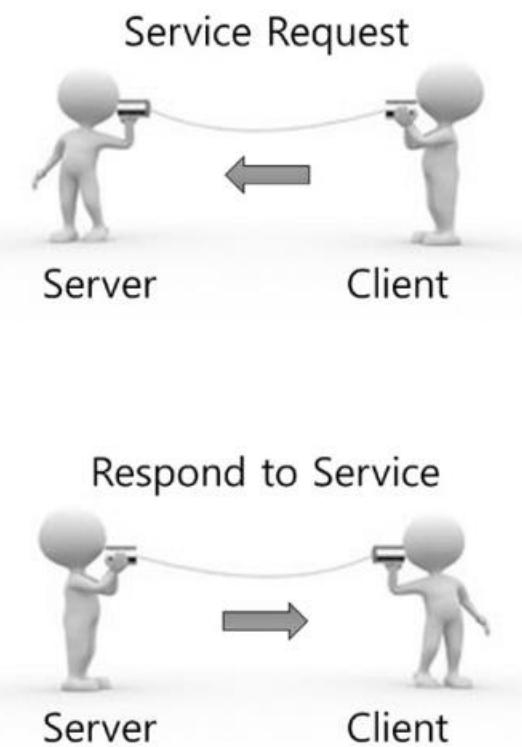


# Service Server and Service Client

- A service server is a ROS node or component that provides a specific service or functionality when requested by other nodes.
- When a service server receives a request from a service client, it processes the request, performs the desired task or computation, and sends back a response to the client.

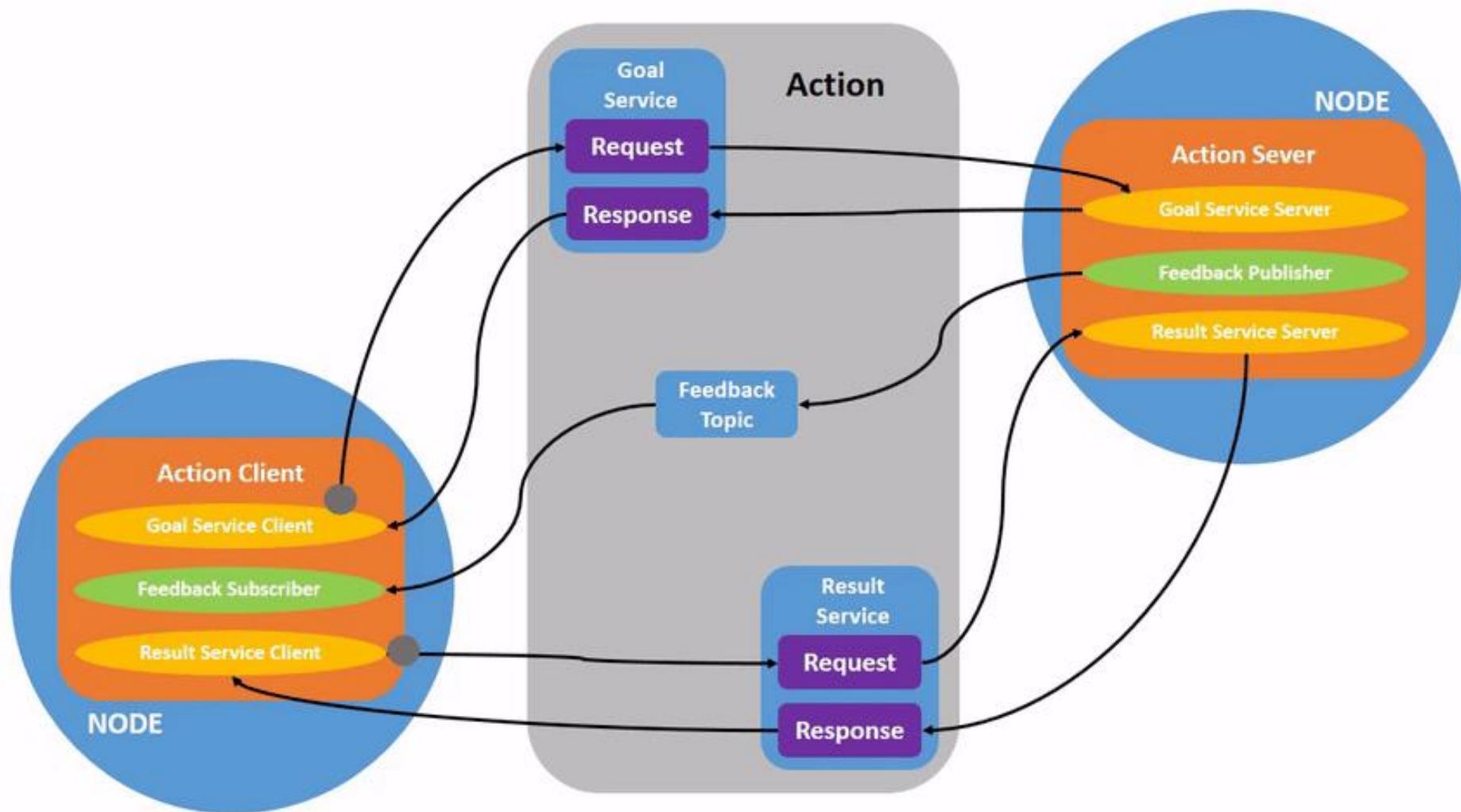


Let me see...  
It's 12 O'clock!



Hey Server,  
What time is it now?

# Actions



# Workspace

- **Colcon build** is the command that you use to build workspaces.

```
$ mkdir <workspace name>  
  
$ cd workspace_name  
  
$ mkdir src  
  
$ colcon build
```

- To start using this workspace you must first source this script by running the following command.

```
$ source ~/workspace_name/install/setup.bash
```

- To create a package, cd into the source space and run the **ros2 pkg create** command

```
$ cd ~/workspace_name/src
```

```
$ catkin_create_pkg [package name] [depend1] [depend2] ... [depend n]
```

```
cd ~/workspace_name
```

```
$ catkin_make
```

# Simple Publisher & Subscriber

```
$ mkdir -p ~/simple_pub_sub
```

```
$ cd ~/simple_pub_sub
```

```
$ mkdir src
```

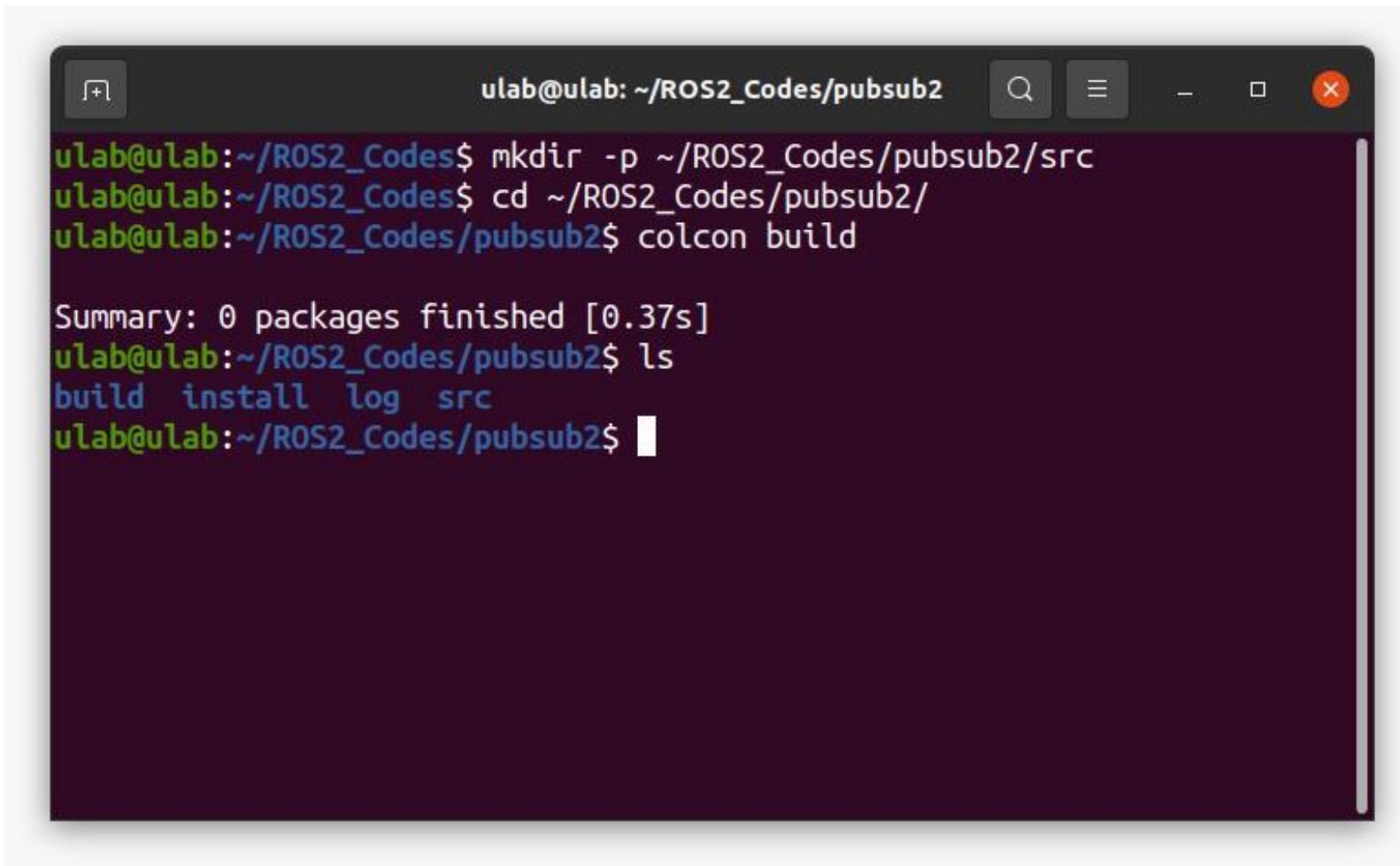
```
$ cd ~/simple_pub_sub/src
```

```
$ ros2 create pkg --build-type ament_python pub_sub
```

```
cd ..
```

```
$ colcon build
```

# Simple Publisher & Subscriber

A screenshot of a terminal window titled "ulab@ulab: ~/ROS2\_Codes/pubsub2". The terminal displays the following command-line session:

```
ulab@ulab:~/ROS2_Codes/pubsub2$ mkdir -p ~/ROS2_Codes/pubsub2/src
ulab@ulab:~/ROS2_Codes/pubsub2$ cd ~/ROS2_Codes/pubsub2/
ulab@ulab:~/ROS2_Codes/pubsub2$ colcon build

Summary: 0 packages finished [0.37s]
ulab@ulab:~/ROS2_Codes/pubsub2$ ls
build  install  log  src
ulab@ulab:~/ROS2_Codes/pubsub2$
```

The terminal has a dark background with light-colored text. It includes standard Linux terminal icons at the top: a maximize button, a search icon, a minimize button, a restore button, and a close button.

# Simple Publisher & Subscriber

The image shows two terminal windows side-by-side. The left window displays the command `ros2 pkg create --build-type ament_python my_pubsub` being run in a directory `~/ROS2_Codes/pubsub2/src`. The output shows the creation of a new ROS 2 package named `my_pubsub` with various files and folders being created. The right window shows the command `colcon build` being run in the same directory, which starts the build process for the `my_pubsub` package and completes it successfully in 1.17 seconds. A summary message indicates 1 package finished in 1.55 seconds.

```
ulab@ulab:~/ROS2_Codes/pubsub2$ ros2 pkg create --build-type ament_python my_pubsub
going to create a new package
package name: my_pubsub
destination directory: /home/ulab/ROS2_Codes/pubsub2/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['ulab <ulab@todo.todo>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: []
creating folder ./my_pubsub
creating ./my_pubsub/package.xml
creating source folder
creating folder ./my_pubsub/my_pubsub
creating ./my_pubsub/setup.py
creating ./my_pubsub/setup.cfg
creating folder ./my_pubsub/resource
creating ./my_pubsub/resource/my_pubsub
creating ./my_pubsub/my_pubsub/__init__.py
creating folder ./my_pubsub/test
creating ./my_pubsub/test/test_copyright.py
creating ./my_pubsub/test/test_flake8.py
creating ./my_pubsub/test/test_pep257.py

ulab@ulab:~/ROS2_Codes/pubsub2$ cd ..
ulab@ulab:~/ROS2_Codes$ colcon build
Starting >>> my_pubsub
Finished <<< my_pubsub [1.17s]

Summary: 1 package finished [1.55s]
ulab@ulab:~/ROS2_Codes$
```

# Simple Publisher & Subscriber

Open VsCode → Open Folder → Goto pub\_sub i-e → my\_publisher\_node.py

```
import rclpy # ROS Client Library for Python
# Handles the creation of nodes
from rclpy.node import Node
# Enables usage of the String message type
from std_msgs.msg import String
class Publisher(Node):
    #Create a Publisher class, which is a subclass of the Node class.
    def __init__(self):
        # Class constructor to set up the node
        # Initiate the Node class's constructor and give it a name
        super().__init__('simple_publisher')
        # Create the publisher. This publisher will publish a String message
        # to the talker topic. The queue size is 10 messages.
        self.publisher_ = self.create_publisher(String, 'talker', 10)
        # We will publish a message every 0.5 seconds
        timer_period = 0.5 # seconds
```

# Simple Publisher & Subscriber

```
# Create the timer
self.timer = self.create_timer(timer_period, self.timer_callback)
self.i = 0 # Initialize a counter variable
def timer_callback(self):
    # Callback function. This function gets called every 0.5 seconds.
    msg = String() # Create a String message
    # Set the String message's data
    msg.data = 'Hello Usama!: %d' % self.i
    # Publish the message to the topic
    self.publisher_.publish(msg)
    # Display the message on the console
    self.get_logger().info('Publishing: "%s"' % msg.data)
    self.i += 1 # Increment the counter by 1
```

# Simple Publisher & Subscriber

```
def main(args=None):
    rclpy.init(args=args) # Initialize the rclpy library
    simple_publisher = Publisher() # Create the node
    # Spin the node so the callback function is called.
    rclpy.spin(simple_publisher)
    rclpy.shutdown() # Shutdown the ROS client library for Python
if __name__ == '__main__':
    main()
```

# Simple Publisher & Subscriber

Pub\_sub → my\_subscriber\_node.py

```
import rclpy # ROS Client Library for Python
from rclpy.node import Node # Handles the creation of nodes
from std_msgs.msg import String # Handles string messages
class Subscriber(Node):
    def __init__(self):
        # Initiate the Node class's constructor and give it a name
        super().__init__('simple_subscriber')
        # The node subscribes to messages of type std_msgs/String,
        # over a topic named: /talker
        # The maximum number of queued messages is 10.
        self.subscription = self.create_subscription(String, 'talker', self.func, 10)
        self.subscription # prevents unused variable warning
    def func(self,msg):
        self.get_logger().info('I heard: "%s"' % msg.data)
```

# Simple Publisher & Subscriber

Pub\_sub → my\_subscriber\_node.py

```
def main(args=None):
    rclpy.init(args=args) # Initialize the rclpy library
    # Create a subscriber
    simple_subscriber = Subscriber()
    # Spin the node so the callback function is called.
    # Pull messages from any topics this node is subscribed to.
    rclpy.spin(simple_subscriber)
    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    simple_subscriber.destroy_node()
    # Shutdown the ROS client library for Python
    rclpy.shutdown()
if __name__ == '__main__':
    main()
```

# Simple Publisher & Subscriber

## Package.xml

```
<export>
  <build_type>ament_python</build_type>
  <exec_depend>rclpy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
</export>
```

# Simple Publisher & Subscriber

## Setup.py

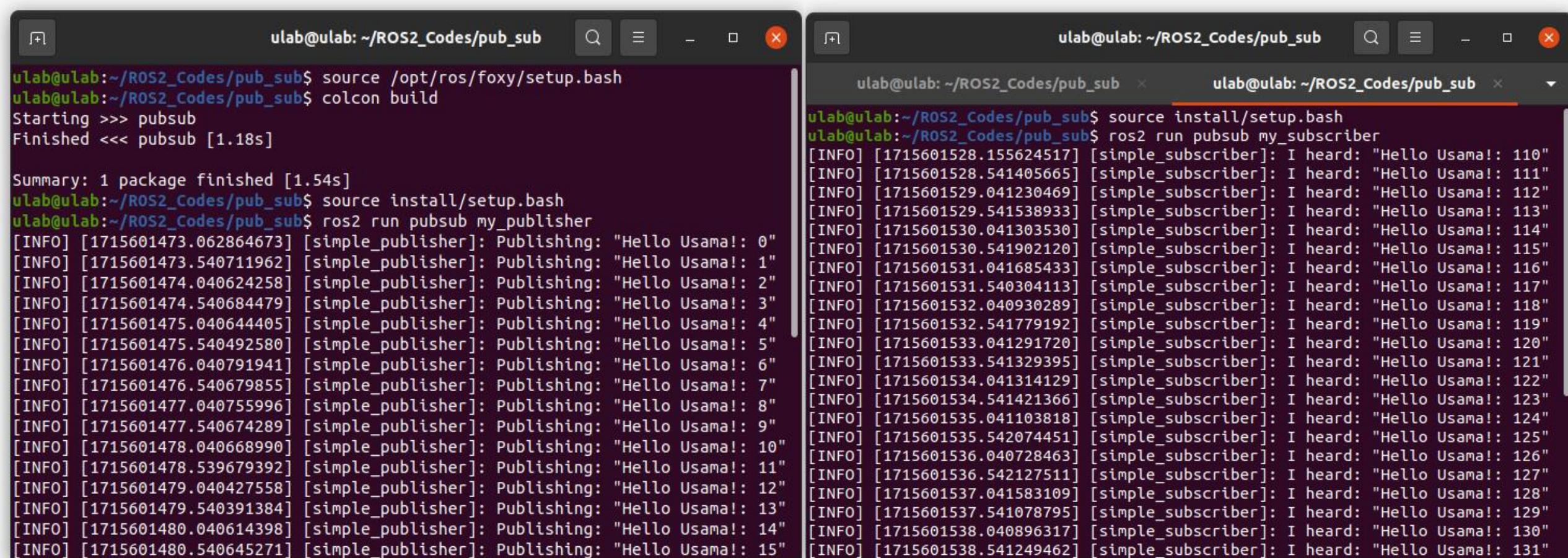
```
entry_points={  
    'console_scripts': ['my_publisher =  
pubsub.my_publisher_node:main',  
                      'my_subscriber =  
pubsub.my_subscriber_node:main',  
                      ],  
},
```

# Simple Publisher & Subscriber

## Setup.py

```
entry_points={  
    'console_scripts': ['my_publisher =  
pubsub.my_publisher_node:main',  
                      'my_subscriber =  
pubsub.my_subscriber_node:main',  
                      ],  
},
```

# Simple Publisher & Subscriber



The image shows two terminal windows side-by-side, both titled "ulab@ulab: ~/ROS2\_Codes/pub\_sub".

The left terminal window contains the following command-line session:

```
ulab@ulab:~/ROS2_Codes/pub_sub$ source /opt/ros/foxy/setup.bash
ulab@ulab:~/ROS2_Codes/pub_sub$ colcon build
Starting >>> pubsub
Finished <<< pubsub [1.18s]

Summary: 1 package finished [1.54s]
ulab@ulab:~/ROS2_Codes/pub_sub$ source install/setup.bash
ulab@ulab:~/ROS2_Codes/pub_sub$ ros2 run pubsub my_publisher
[INFO] [1715601473.062864673] [simple_publisher]: Publishing: "Hello Usama!: 0"
[INFO] [1715601473.540711962] [simple_publisher]: Publishing: "Hello Usama!: 1"
[INFO] [1715601474.040624258] [simple_publisher]: Publishing: "Hello Usama!: 2"
[INFO] [1715601474.540684479] [simple_publisher]: Publishing: "Hello Usama!: 3"
[INFO] [1715601475.040644405] [simple_publisher]: Publishing: "Hello Usama!: 4"
[INFO] [1715601475.540492580] [simple_publisher]: Publishing: "Hello Usama!: 5"
[INFO] [1715601476.040791941] [simple_publisher]: Publishing: "Hello Usama!: 6"
[INFO] [1715601476.540679855] [simple_publisher]: Publishing: "Hello Usama!: 7"
[INFO] [1715601477.040755996] [simple_publisher]: Publishing: "Hello Usama!: 8"
[INFO] [1715601477.540674289] [simple_publisher]: Publishing: "Hello Usama!: 9"
[INFO] [1715601478.040668990] [simple_publisher]: Publishing: "Hello Usama!: 10"
[INFO] [1715601478.539679392] [simple_publisher]: Publishing: "Hello Usama!: 11"
[INFO] [1715601479.040427558] [simple_publisher]: Publishing: "Hello Usama!: 12"
[INFO] [1715601479.540391384] [simple_publisher]: Publishing: "Hello Usama!: 13"
[INFO] [1715601480.040614398] [simple_publisher]: Publishing: "Hello Usama!: 14"
[INFO] [1715601480.540645271] [simple_publisher]: Publishing: "Hello Usama!: 15"
```

The right terminal window contains the following command-line session:

```
ulab@ulab:~/ROS2_Codes/pub_sub$ source install/setup.bash
ulab@ulab:~/ROS2_Codes/pub_sub$ ros2 run pubsub my_subscriber
[INFO] [1715601528.155624517] [simple_subscriber]: I heard: "Hello Usama!: 110"
[INFO] [1715601528.541405665] [simple_subscriber]: I heard: "Hello Usama!: 111"
[INFO] [1715601529.041230469] [simple_subscriber]: I heard: "Hello Usama!: 112"
[INFO] [1715601529.541538933] [simple_subscriber]: I heard: "Hello Usama!: 113"
[INFO] [1715601530.041303530] [simple_subscriber]: I heard: "Hello Usama!: 114"
[INFO] [1715601530.541902120] [simple_subscriber]: I heard: "Hello Usama!: 115"
[INFO] [1715601531.041685433] [simple_subscriber]: I heard: "Hello Usama!: 116"
[INFO] [1715601531.540304113] [simple_subscriber]: I heard: "Hello Usama!: 117"
[INFO] [1715601532.040930289] [simple_subscriber]: I heard: "Hello Usama!: 118"
[INFO] [1715601532.541779192] [simple_subscriber]: I heard: "Hello Usama!: 119"
[INFO] [1715601533.041291720] [simple_subscriber]: I heard: "Hello Usama!: 120"
[INFO] [1715601533.541329395] [simple_subscriber]: I heard: "Hello Usama!: 121"
[INFO] [1715601534.041314129] [simple_subscriber]: I heard: "Hello Usama!: 122"
[INFO] [1715601534.541421366] [simple_subscriber]: I heard: "Hello Usama!: 123"
[INFO] [1715601535.041103818] [simple_subscriber]: I heard: "Hello Usama!: 124"
[INFO] [1715601535.542074451] [simple_subscriber]: I heard: "Hello Usama!: 125"
[INFO] [1715601536.040728463] [simple_subscriber]: I heard: "Hello Usama!: 126"
[INFO] [1715601536.542127511] [simple_subscriber]: I heard: "Hello Usama!: 127"
[INFO] [1715601537.041583109] [simple_subscriber]: I heard: "Hello Usama!: 128"
[INFO] [1715601537.541078795] [simple_subscriber]: I heard: "Hello Usama!: 129"
[INFO] [1715601538.040896317] [simple_subscriber]: I heard: "Hello Usama!: 130"
[INFO] [1715601538.541249462] [simple_subscriber]: I heard: "Hello Usama!: 131"
```

# Simple Publisher & Subscriber

The screenshot displays two windows side-by-side. On the left is a terminal window titled 'ulab@ulab: ~/ROS2\_Codes/pub\_sub' showing ROS2 command outputs. On the right is the 'rqt\_graph' application showing a Node Graph.

**Terminal Output (ulab@ulab: ~/ROS2\_Codes/pub\_sub):**

```
ulab@ulab:~/.config$ ros2 topic list
/parameter_events
/rosout
/talker
ulab@ulab:~/.config$ ros2 topic info /talker
Type: std_msgs/msg/String
Publisher count: 1
Subscription count: 1
ulab@ulab:~/.config$ ros2 run rqt_graph rqt_graph
```

**rqt\_graph Node Graph:**

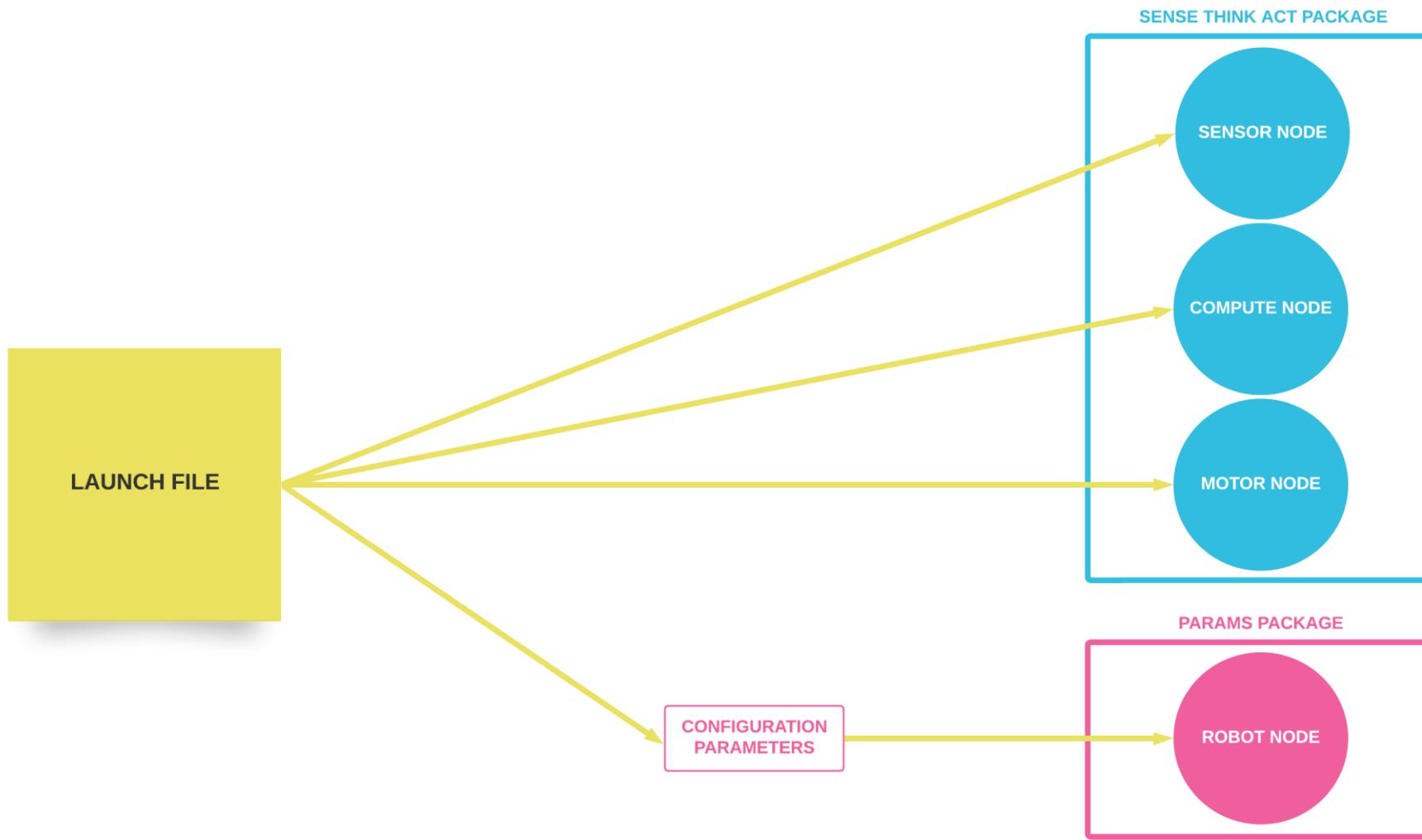
The Node Graph interface shows two nodes: '/simple\_publisher' and '/simple\_subscriber'. They are connected by an arrow labeled '/talker', indicating a publisher-subscriber relationship where '/simple\_publisher' publishes to '/simple\_subscriber' via the '/talker' topic.

```
graph LR; simple_publisher[/simple_publisher] -- "/talker" --> simple_subscriber[/simple_subscriber]
```

# Launch File

- Running many ROS 2 nodes takes a lot of time and many terminal windows.
- Even small projects or robots can have many nodes running simultaneously.
- Instead of running each of these nodes in a separate terminal window each time we startup the robot, we can use a launch file to execute them all at once – with a single command, in a single terminal window.

# Launch File



# Launch File

Create launch folder → Make `launchfile.launch.py`

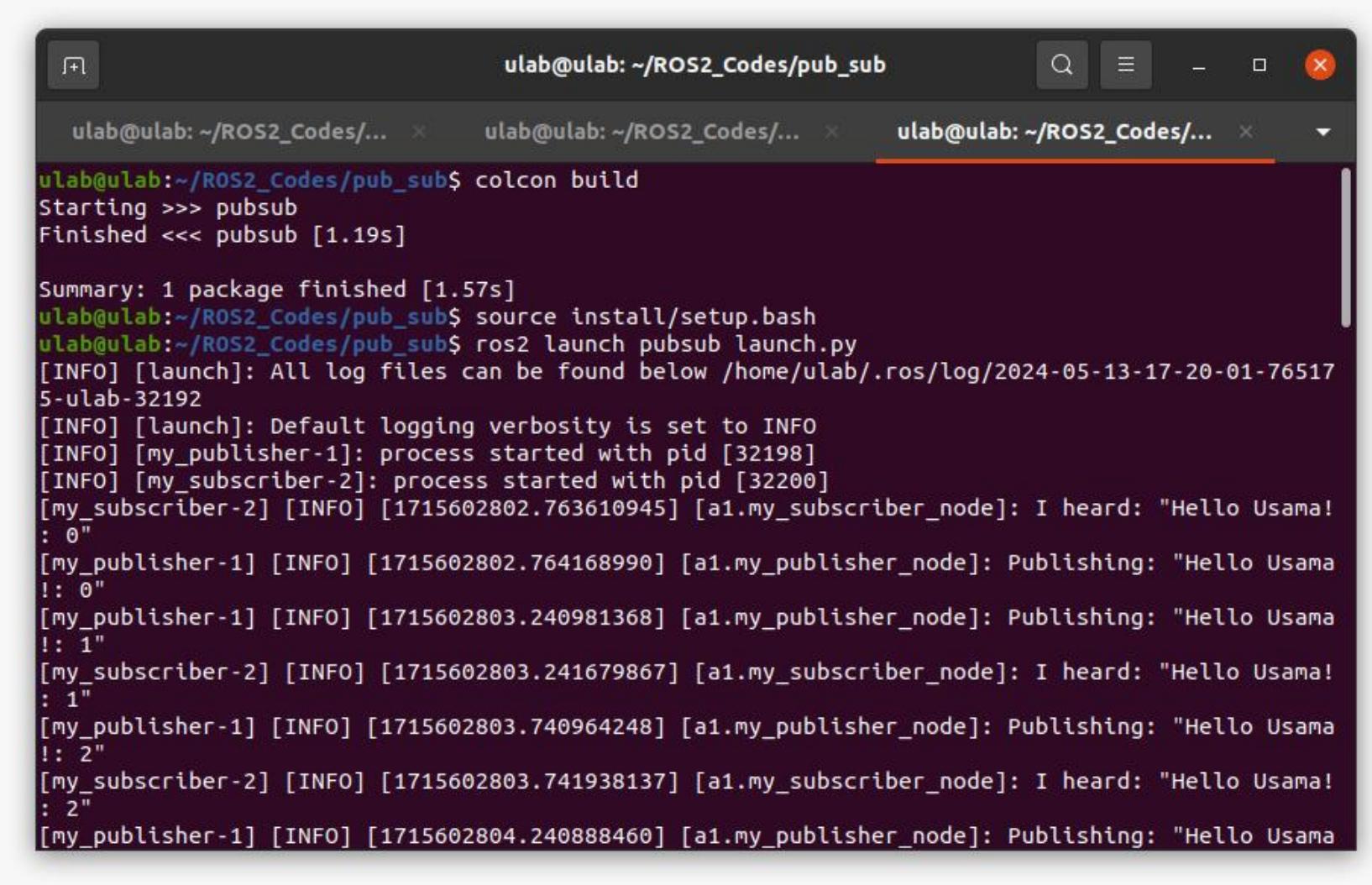
```
from launch import LaunchDescription
from launch_ros.actions import Node
def generate_launch_description():
    return LaunchDescription([
        Node(
            package='pubsub',
            namespace='a1',
            executable='my_publisher',
            name='my_publisher_node'),
        Node(
            package='pubsub',
            executable='my_subscriber',
            namespace='a1',
            name='my_subscriber_node')
    ])
```

# Launch File

## Edit setup.py

```
import os
from glob import glob
# Other imports ...
package_name = 'pub_sub'
setup(
    # Other parameters ...
    data_files=[
        # ... Other data files
        # Include all launch files.
        (os.path.join('share', package_name, 'launch'),
glob.glob(os.path.join('launch', 'launch.')))    ]
)
```

# Launch File



The screenshot shows a terminal window with three tabs open, all displaying the same command-line session. The terminal title is "ulab@ulab: ~/ROS2\_Codes/pub\_sub". The session starts with:

```
ulab@ulab:~/ROS2_Codes/... > ulab@ulab:~/ROS2_Codes/... > ulab@ulab:~/ROS2_Codes/... >
```

Then the user runs:

```
ulab@ulab:~/ROS2_Codes/pub_sub$ colcon build
```

Output:

```
Starting >>> pubsub
Finished <<< pubsub [1.19s]
```

Summary:

```
Summary: 1 package finished [1.57s]
```

Then the user runs:

```
ulab@ulab:~/ROS2_Codes/pub_sub$ source install/setup.bash
```

```
ulab@ulab:~/ROS2_Codes/pub_sub$ ros2 launch pubsub launch.py
```

Output:

```
[INFO] [launch]: All log files can be found below /home/ulab/.ros/log/2024-05-13-17-20-01-765175-ulab-32192
```

```
[INFO] [launch]: Default logging verbosity is set to INFO
```

```
[INFO] [my_publisher-1]: process started with pid [32198]
```

```
[INFO] [my_subscriber-2]: process started with pid [32200]
```

```
[my_subscriber-2] [INFO] [1715602802.763610945] [a1.my_subscriber_node]: I heard: "Hello Usama!
: 0"
```

```
[my_publisher-1] [INFO] [1715602802.764168990] [a1.my_publisher_node]: Publishing: "Hello Usama
!: 0"
```

```
[my_publisher-1] [INFO] [1715602803.240981368] [a1.my_publisher_node]: Publishing: "Hello Usama
!: 1"
```

```
[my_subscriber-2] [INFO] [1715602803.241679867] [a1.my_subscriber_node]: I heard: "Hello Usama!
: 1"
```

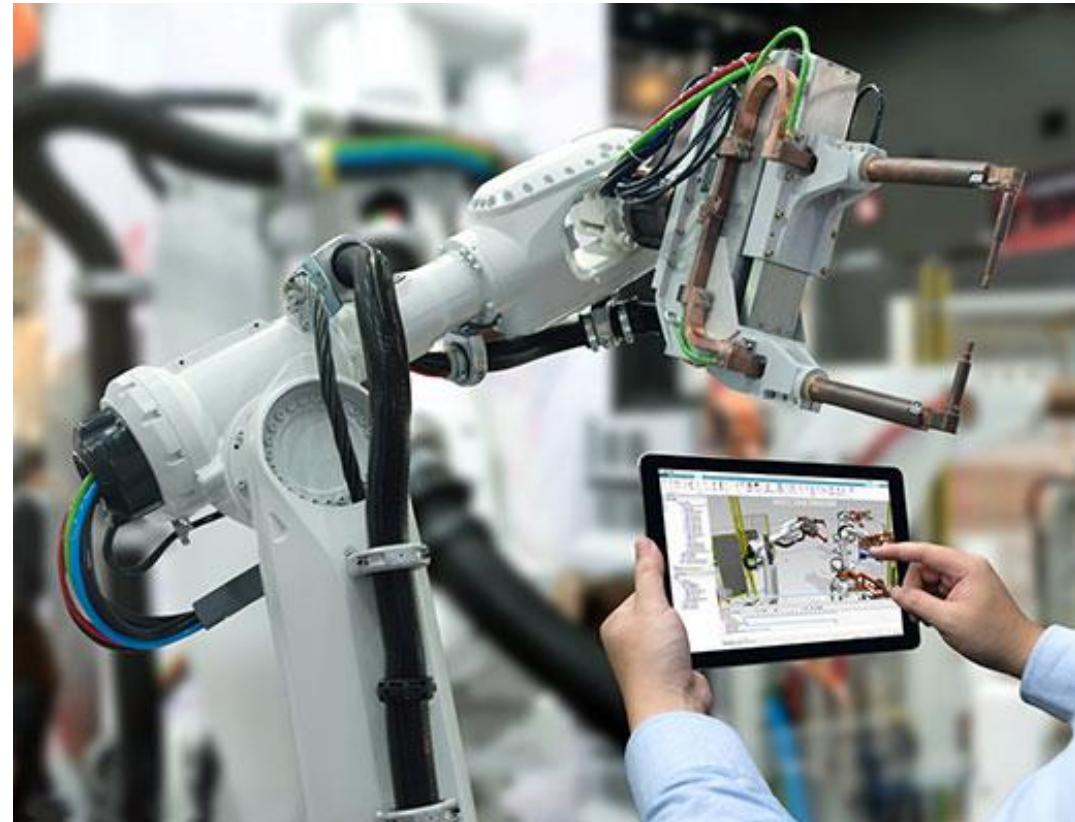
```
[my_publisher-1] [INFO] [1715602803.740964248] [a1.my_publisher_node]: Publishing: "Hello Usama
!: 2"
```

```
[my_subscriber-2] [INFO] [1715602803.741938137] [a1.my_subscriber_node]: I heard: "Hello Usama!
: 2"
```

```
[my_publisher-1] [INFO] [1715602804.240888460] [a1.my_publisher_node]: Publishing: "Hello Usama
```

# Digital Twin

- Digital twins are virtual representations of physical products and processes.
- Digital twins are also vital in robotic manipulation. They help virtualize hardware motion from raw material to the final product.
- Application:
  - Healthcare
  - Manufacturing
  - Aerospace
  - Automotive

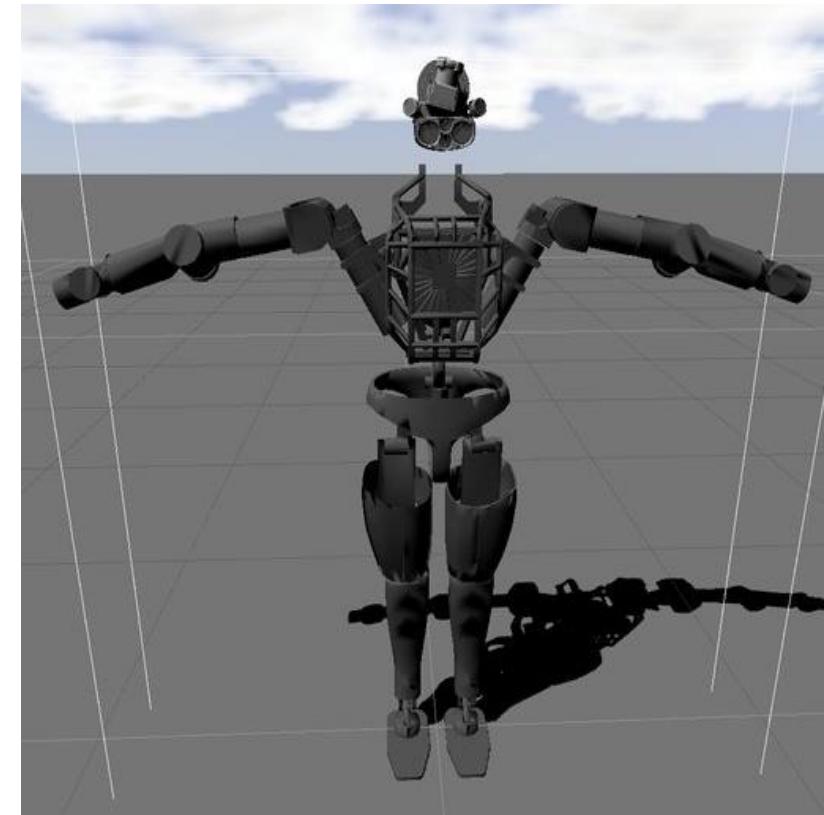
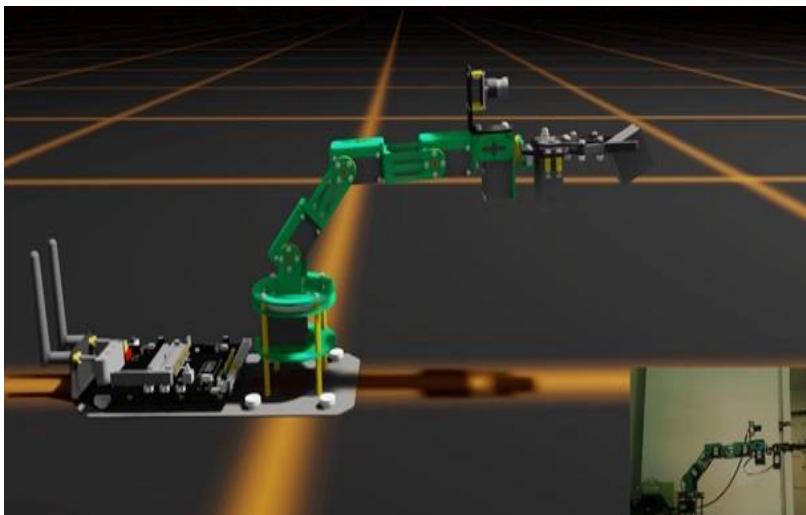
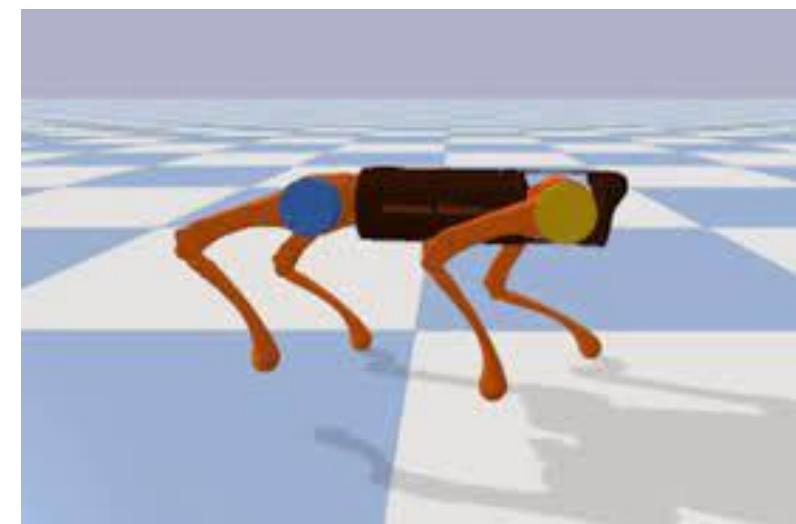
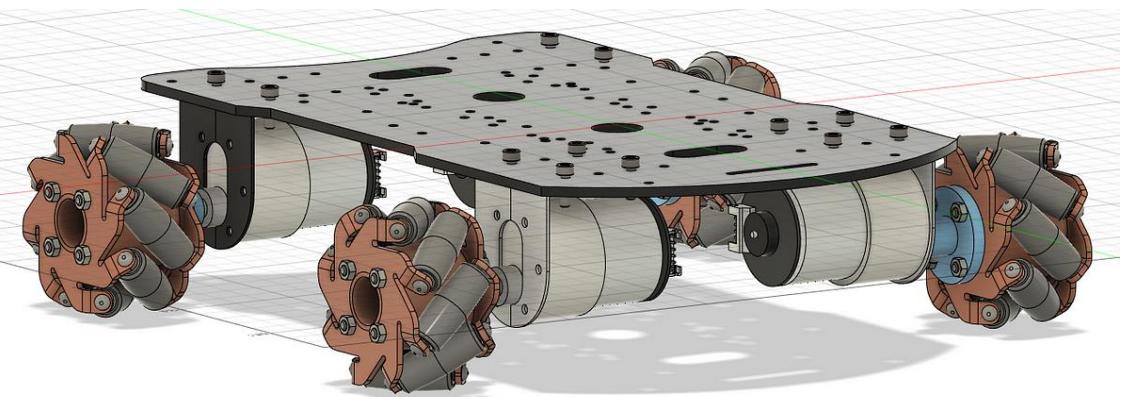


# Digital Twin

SIEMENS

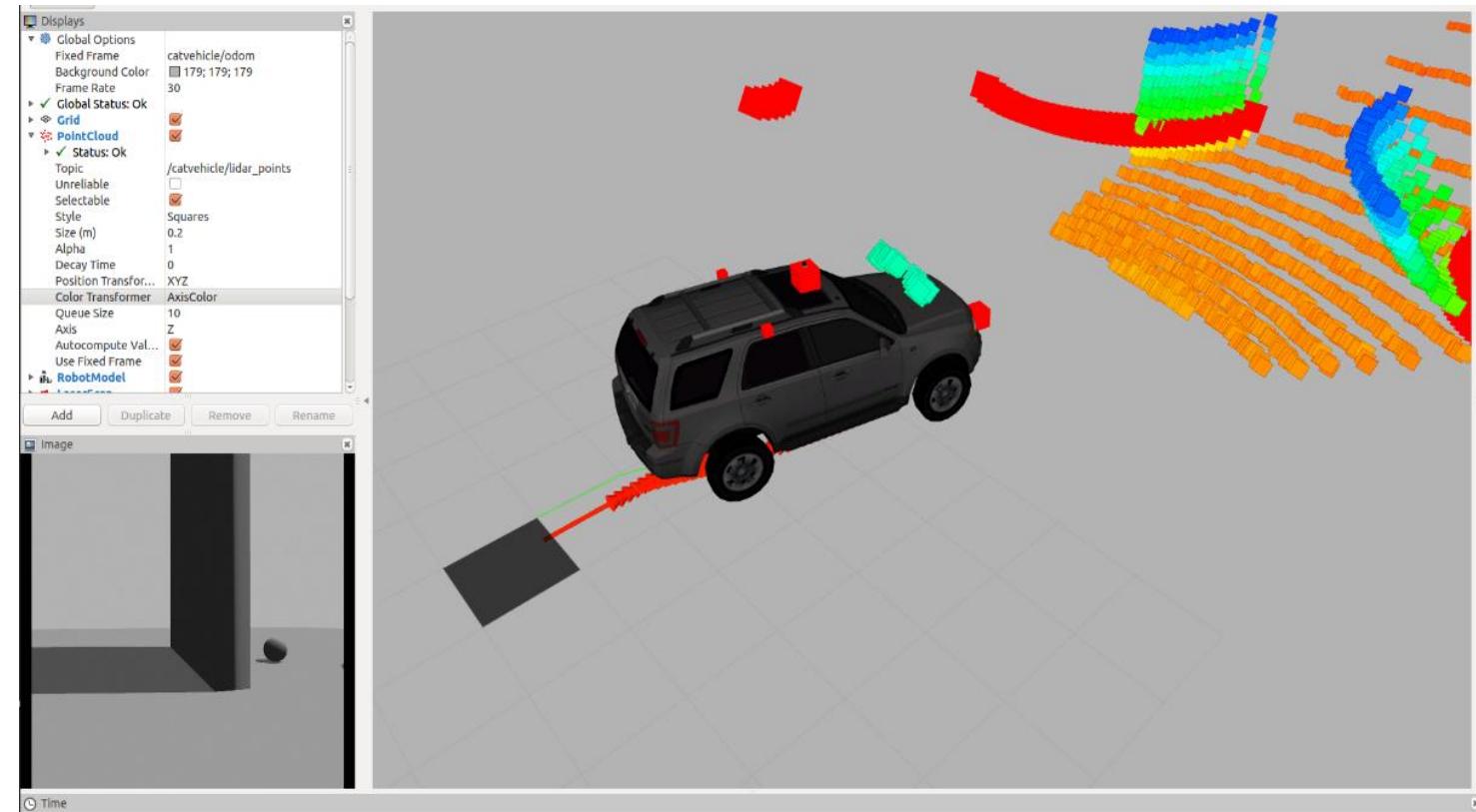


# Digital Twin



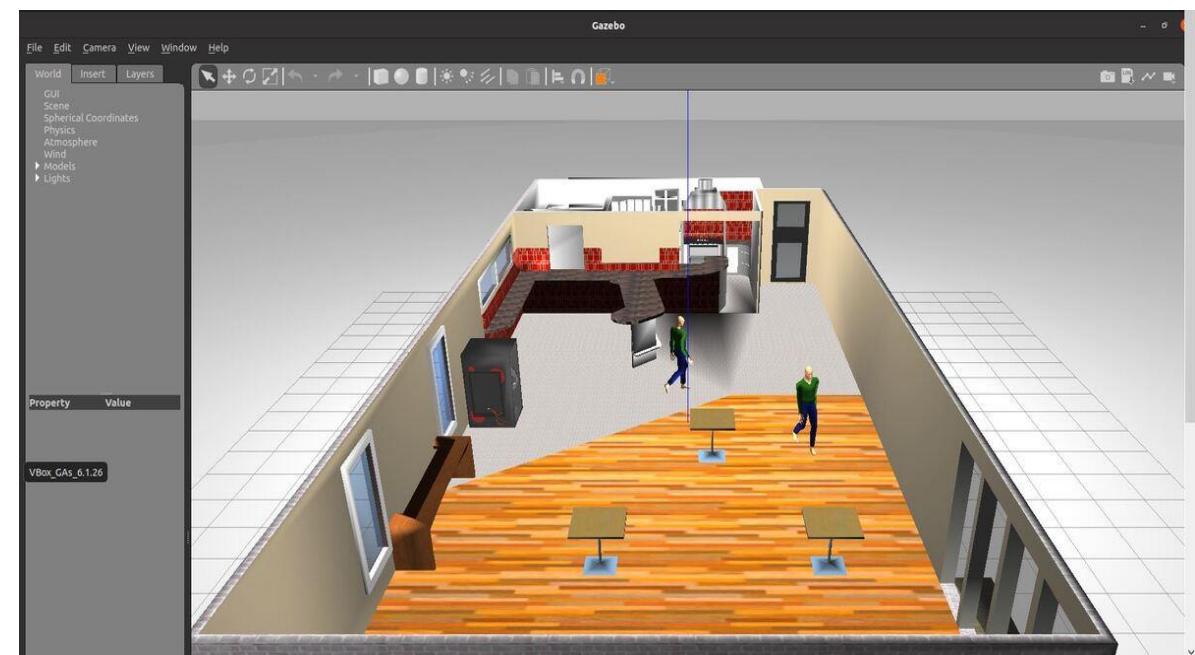
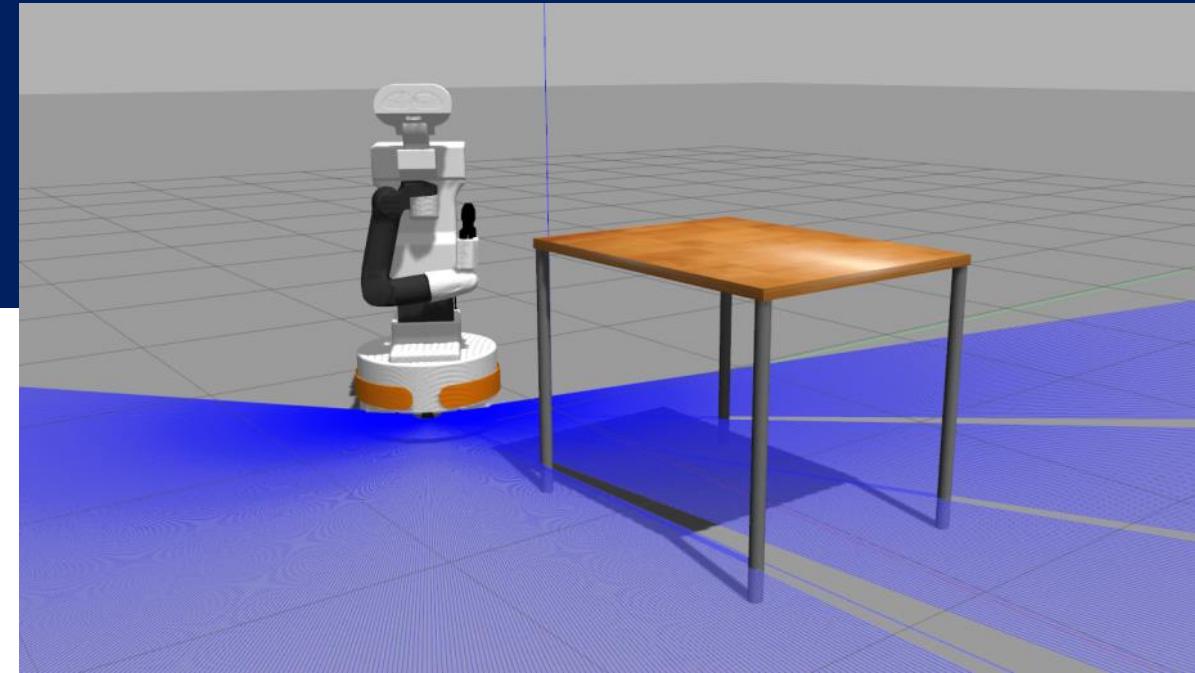
# Robot Visualization (Rviz)

- RViz is a 3D Visualization tool for ROS. Ros Vizualization.
- It takes in a topic as input and visualizes that based on the message type being published.
- **Sensor Data Visualization**
- **Robot Model Visualization**
- **Laser Scan and Point Cloud Visualization**

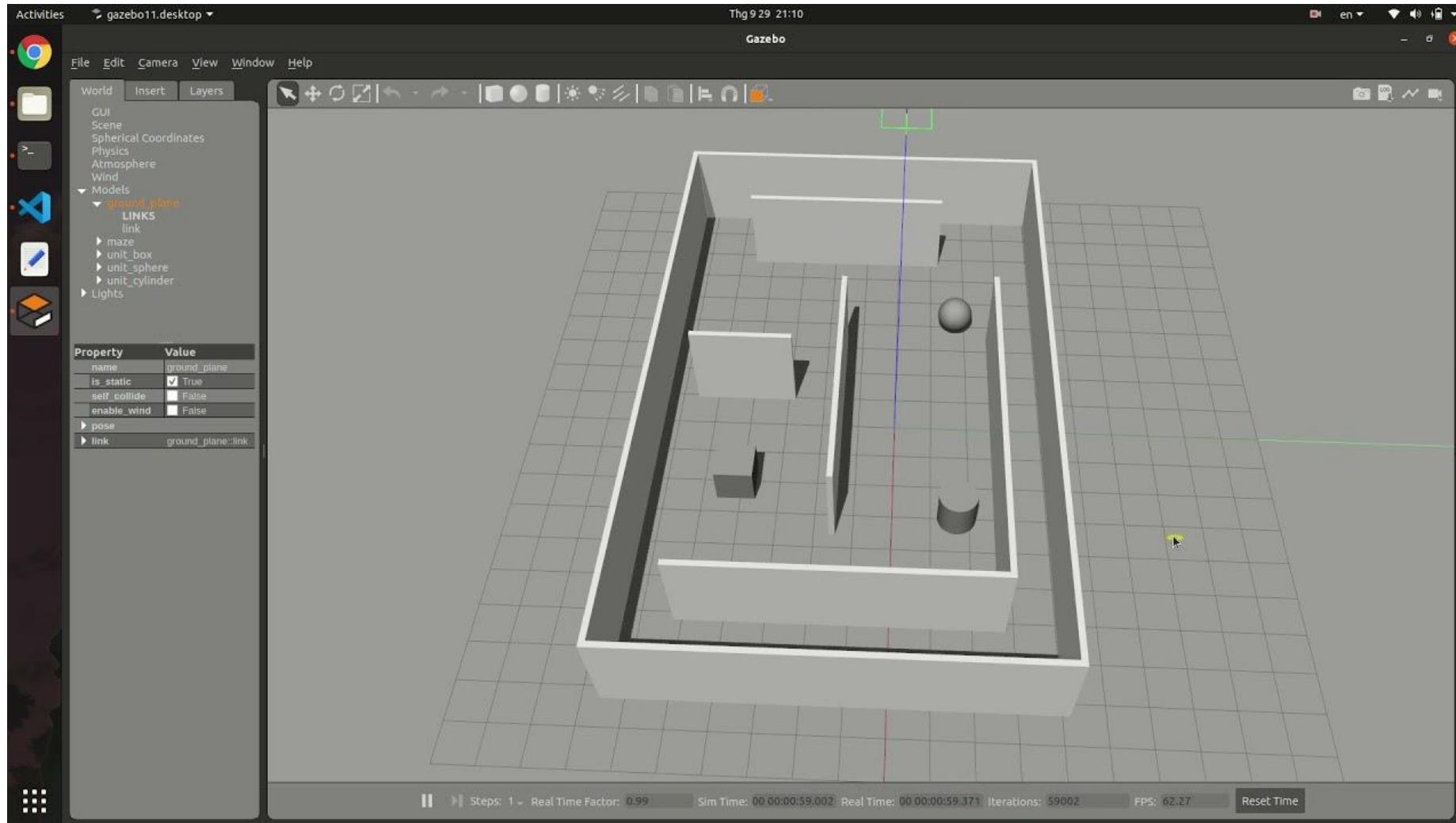


# Gazebo

- **Gazebo** is an open-source 3D physics simulator used in robotics and autonomous systems development.
- Gazebo provides high-fidelity physics simulation for testing robot behaviors in a virtual environment.
- Gazebo can simulate various sensors like LIDAR, cameras, and IMUs.

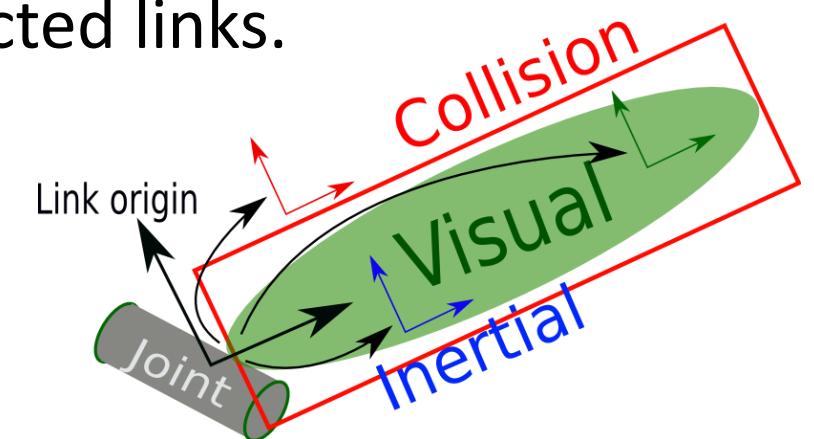
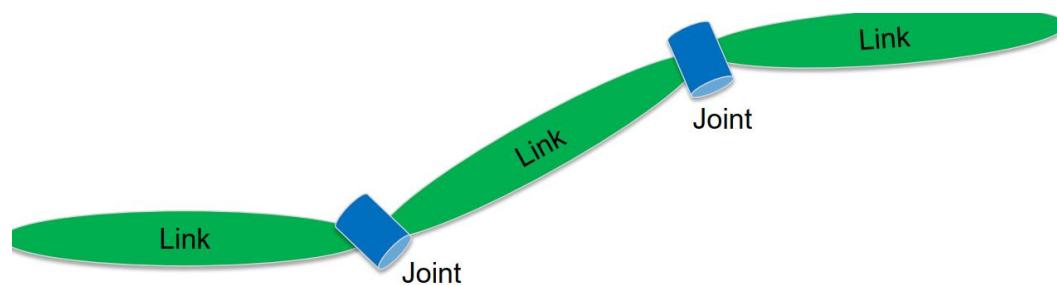


# Gazebo



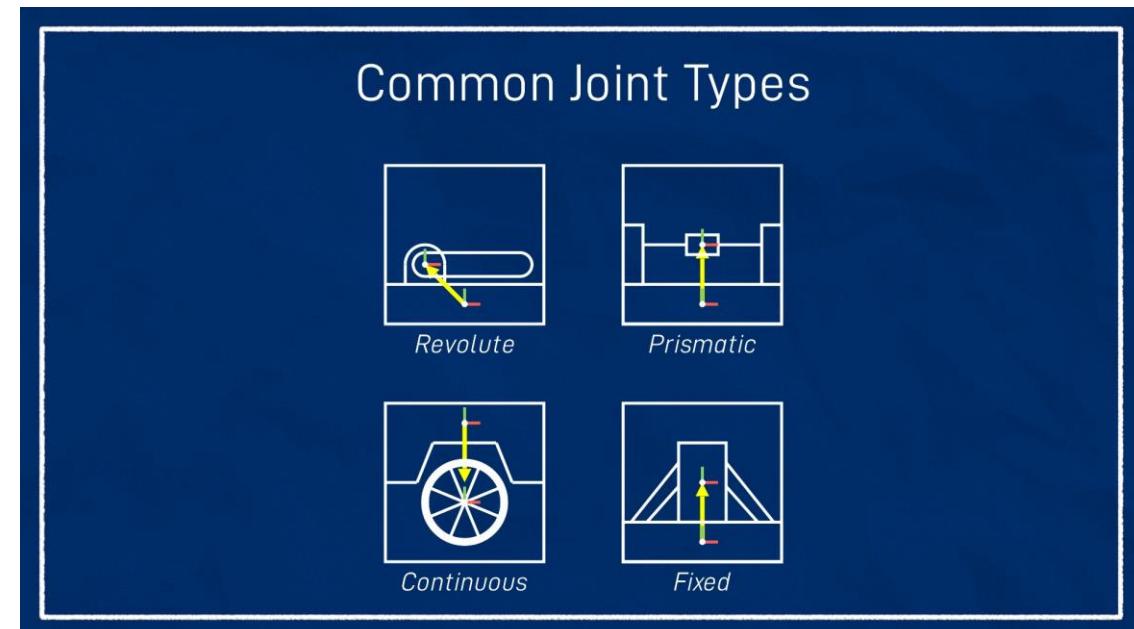
# How Robot made

- Most robotic mechanisms may be simplified to a series of **links** connected by motorized joints.
- Links are approximated as **rigid bodies**.
- A **rigid body** is an object whose deformation is assumed to be negligible such that any two points on the body will always be the same distance from one another.
- **Joints** are actuated by motors to move connected links.



# URDF (Unified Robot Description Format)

- **Revolute** - A rotational motion, with minimum/maximum angle limits.
- **Continuous** - A rotational motion with no limit (e.g. a wheel).
- **Prismatic** - A linear sliding motion, with minimum/maximum position limits.
- **Fixed** - The child link is rigidly connected to the parent link. This is what we use for those “convenience” links.



# URDF (Unified Robot Description Format)

```
<robot>
  <link>
    ...
  </link>
  <link>
    ...
  </link>
  <joint>
    ...
  </joint>
</robot>
```

```
<robot>
  <link>
    <inertial>
      ...
    </inertial>
    <visual>
      <geometry>
        ...
      </geometry>
      <material>
        <color />
      </material>
    </visual>
  </link>
  ...
</robot>
```

```
<robot name = "linkage">
  <joint name = "joint A ... >
    <parent link = "link A" />
    <child link = "link B" />
  </joint>
  <joint name = "joint B ... >
    <parent link = "link A" />
    <child link = "link C" />
  </joint>
  <joint name = "joint C ... >
    <parent link = "link C" />
    <child link = "link D" />
  </joint>
</robot>
```

# URDF (Unified Robot Description Format)

**URDF**  
Unified Robot Description Format

```

<!-- LIDAR -->
<link name="laser_frame">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <cylinder length="0.020" radius="0.0325" />
    </geometry>
    <material name="yellow">
      <color rgba="1 1 0 0.5"/>
    </material>
  </visual>
</link>

"Link" identifies a Frame (of reference)

"Visual" describes shape and position relative to enclosing link (base_plate is part of base_link)

<!-- Base Plate -->
<visual name="base_plate">
  <origin xyz="-0.017 0 0.047" rpy="0 0 0" />
  <geometry>
    <box size="0.216 0.103 0.004"/>
  </geometry>
  <material name="blue"/>
</visual>

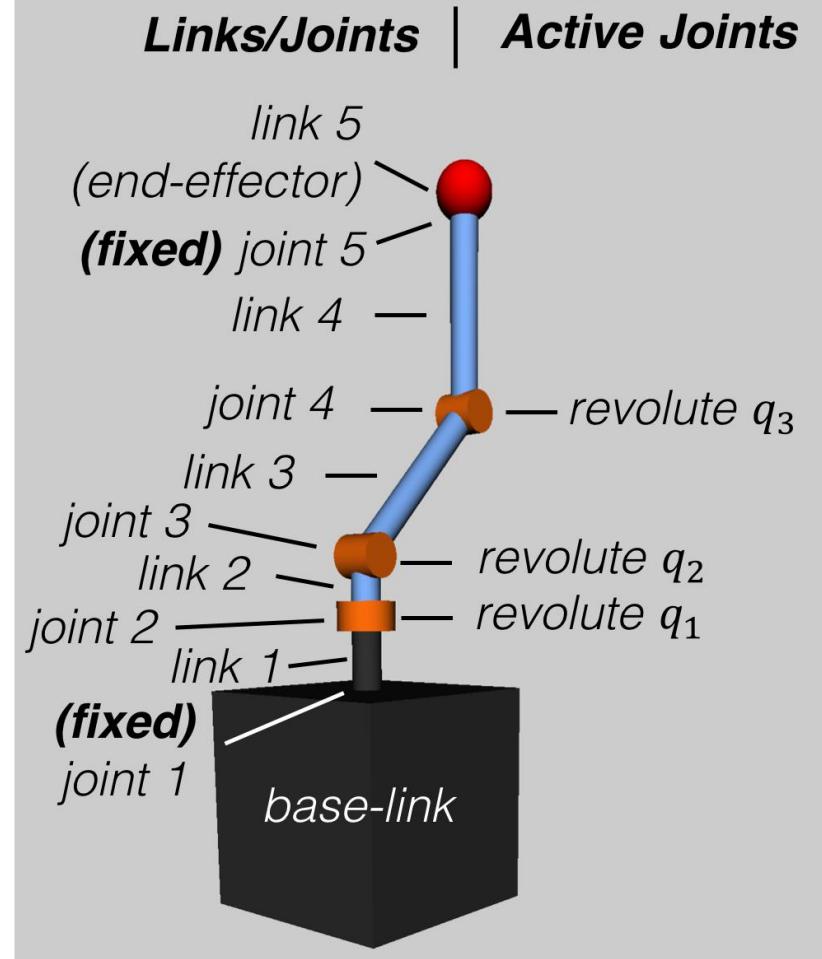
<!-- Base Link -->
<link name="base_link">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
  </visual>

```

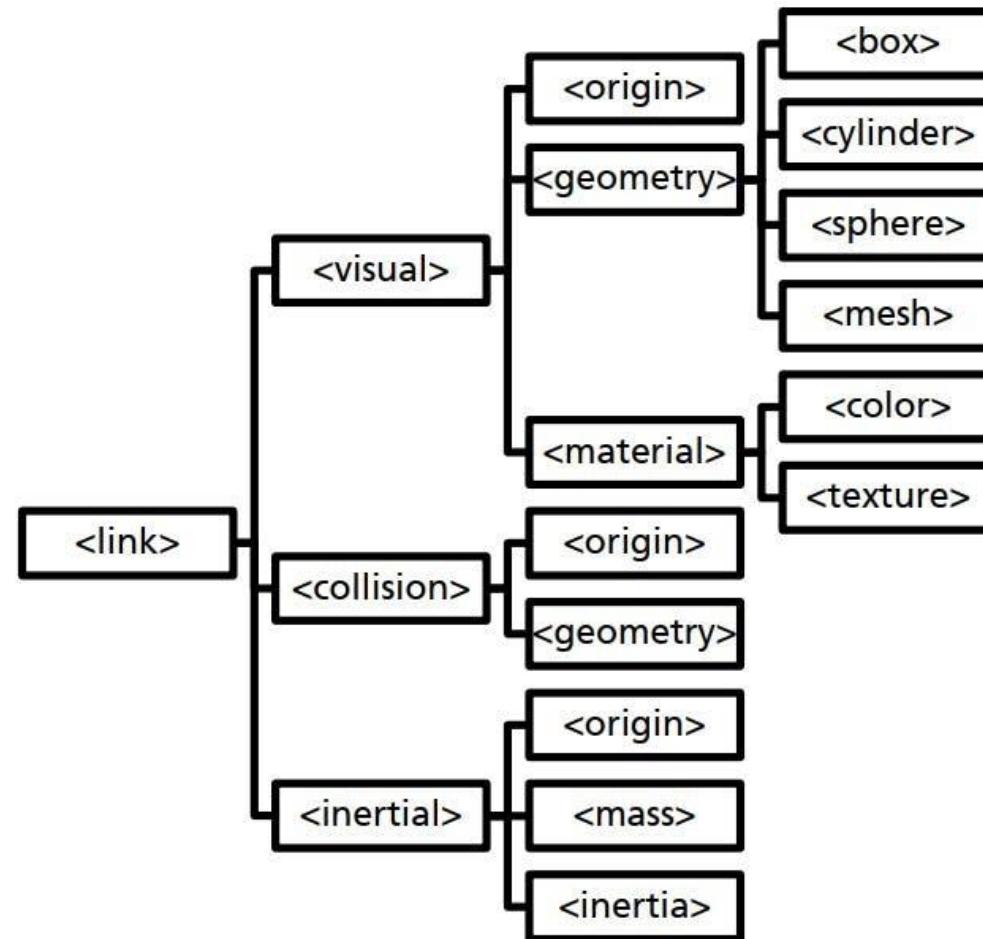
-14mm

110mm

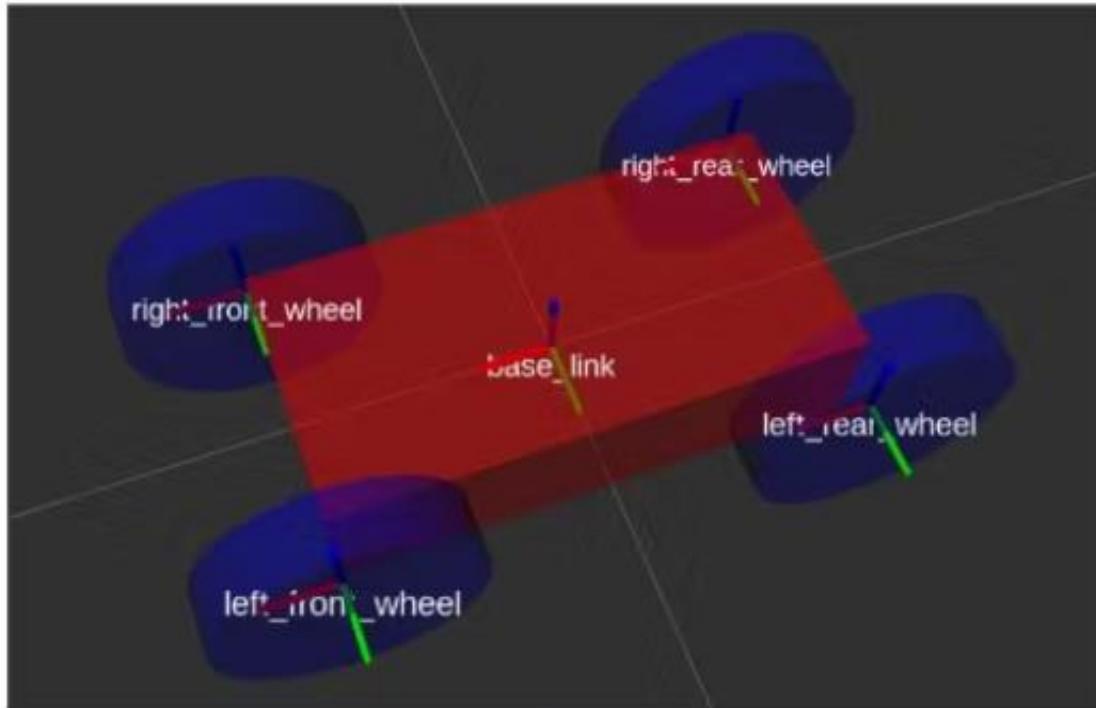
"base\_link" is the robot's frame of reference



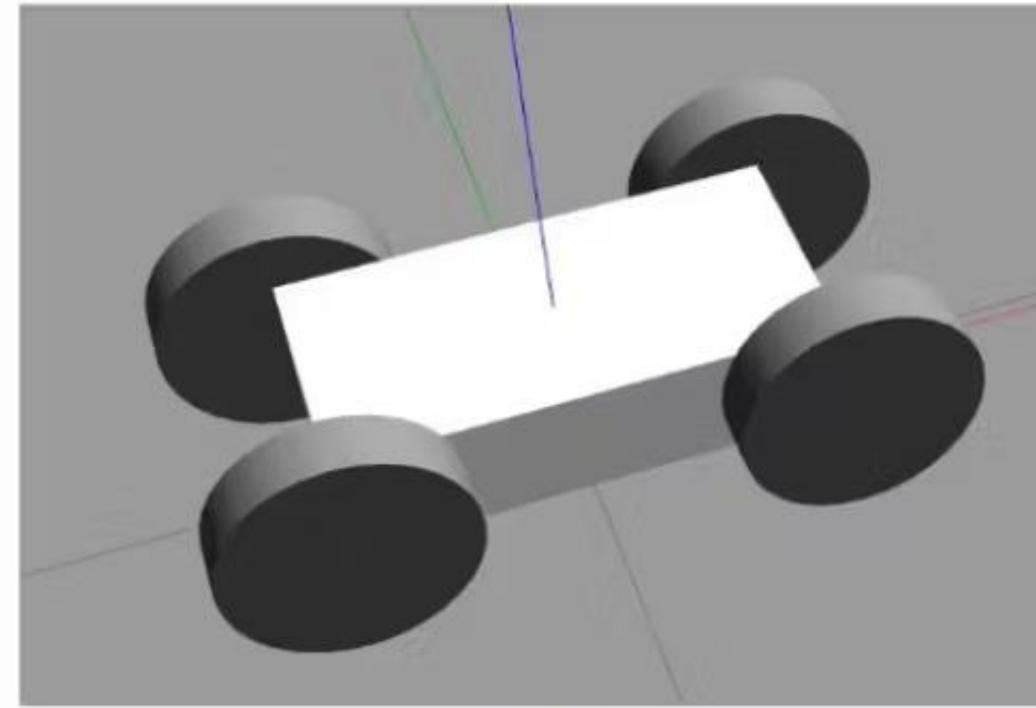
# URDF (Unified Robot Description Format)



# URDF (Unified Robot Description Format)

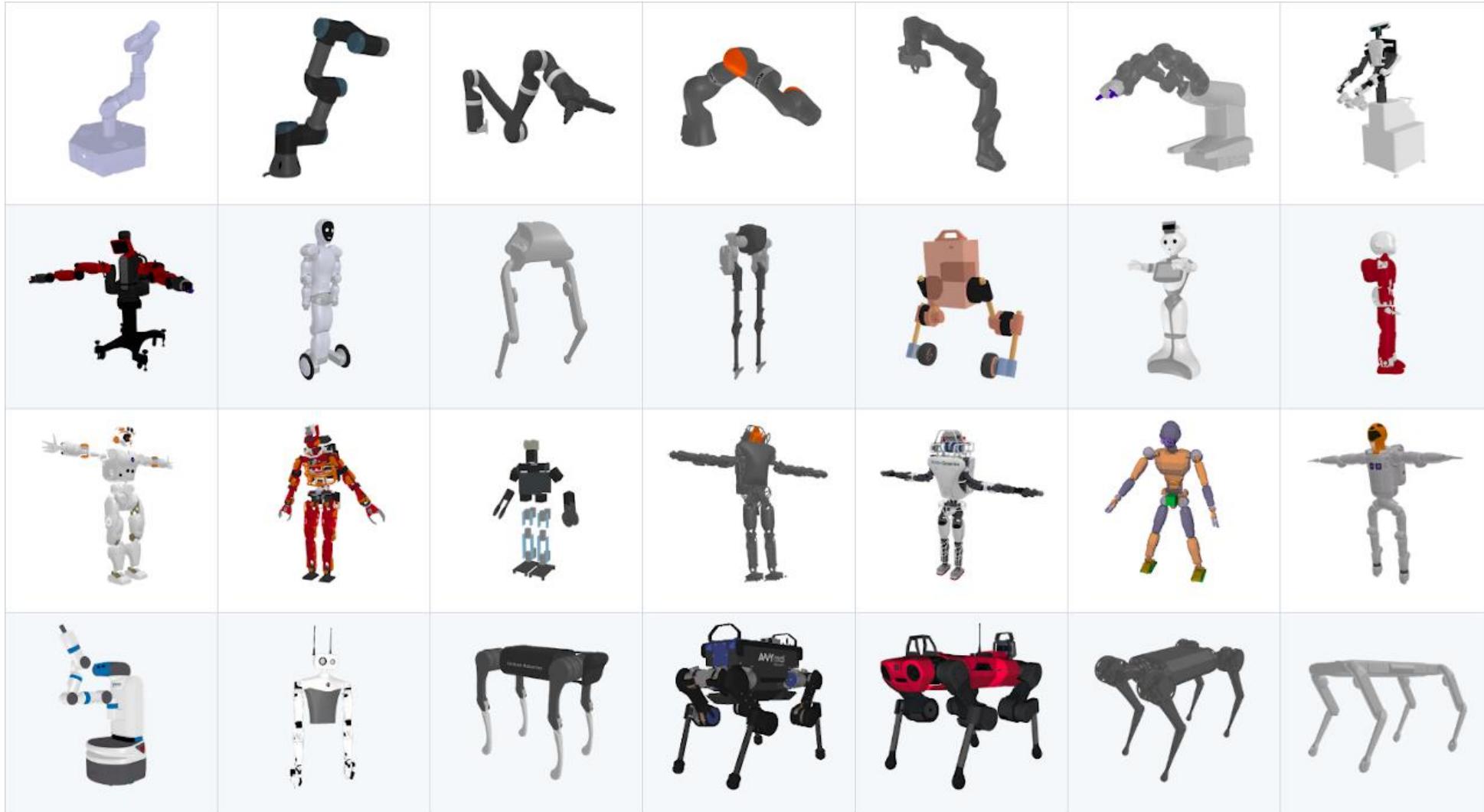


rviz



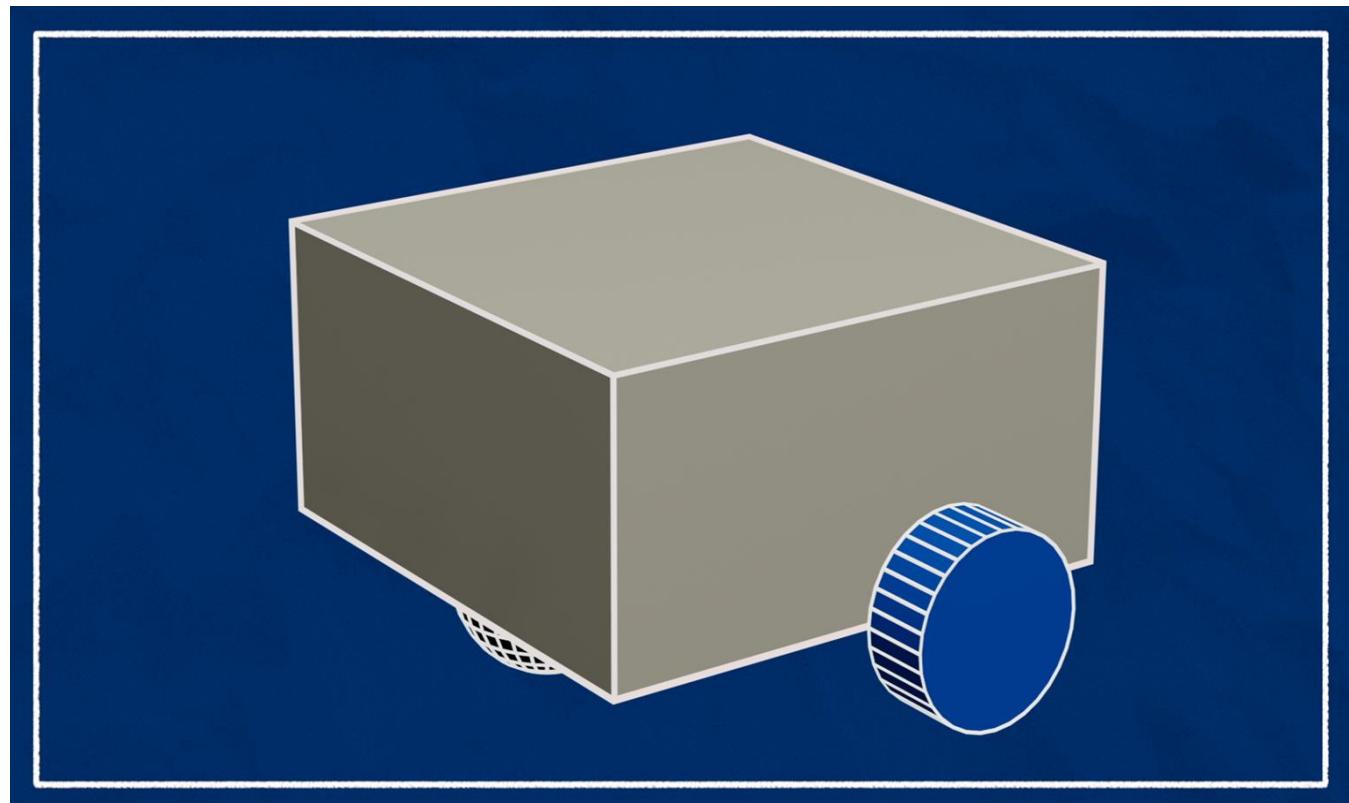
Gazebo

# URDF (Unified Robot Description Format)



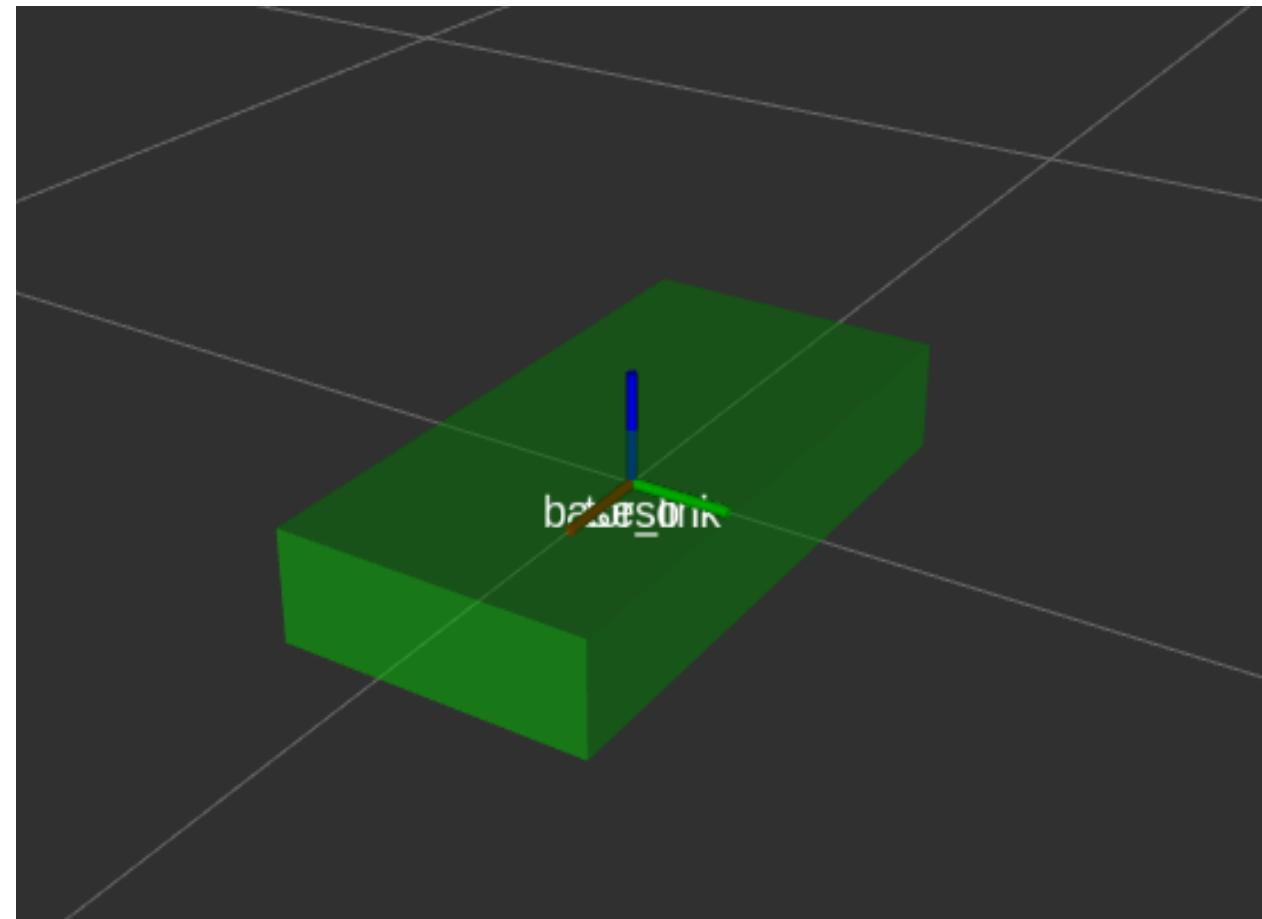
# Differential Drive Robot

- The robot is a differential-drive robot, which means the robot has two driven wheels, one on the left and one on the right.
- These two wheels control all motion, and any other wheels are just there to keep it stable and can spin freely in all directions (these are called caster wheels).



# Differential Drive Robot

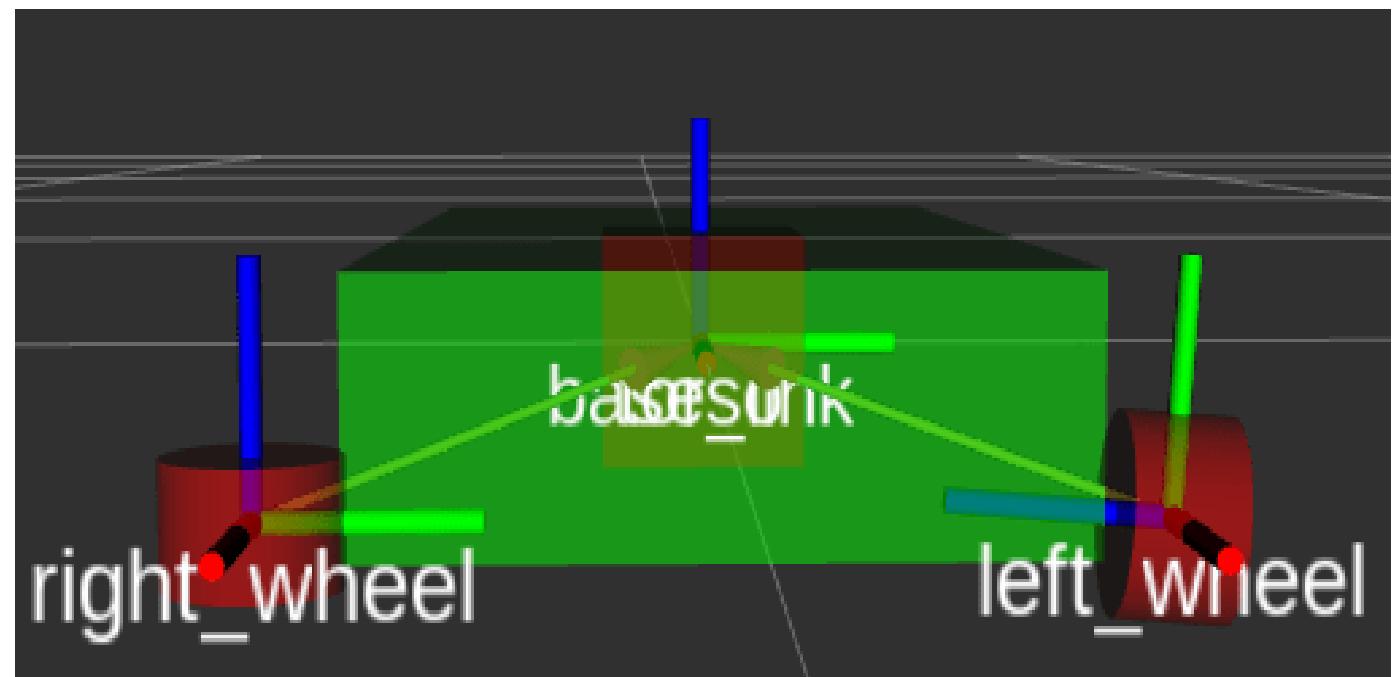
```
<?xml version="1.0"?>
<robot name="bot">
  <link name="base_link">
    <visual>
      <geometry>
        <box size="0.6 0.3 0.1"/>
      </geometry>
      <material name="green">
        <color rgba="1.0 1.0 0.0 1"/>
      </material>
    </visual>
  </link>
</robot>
```



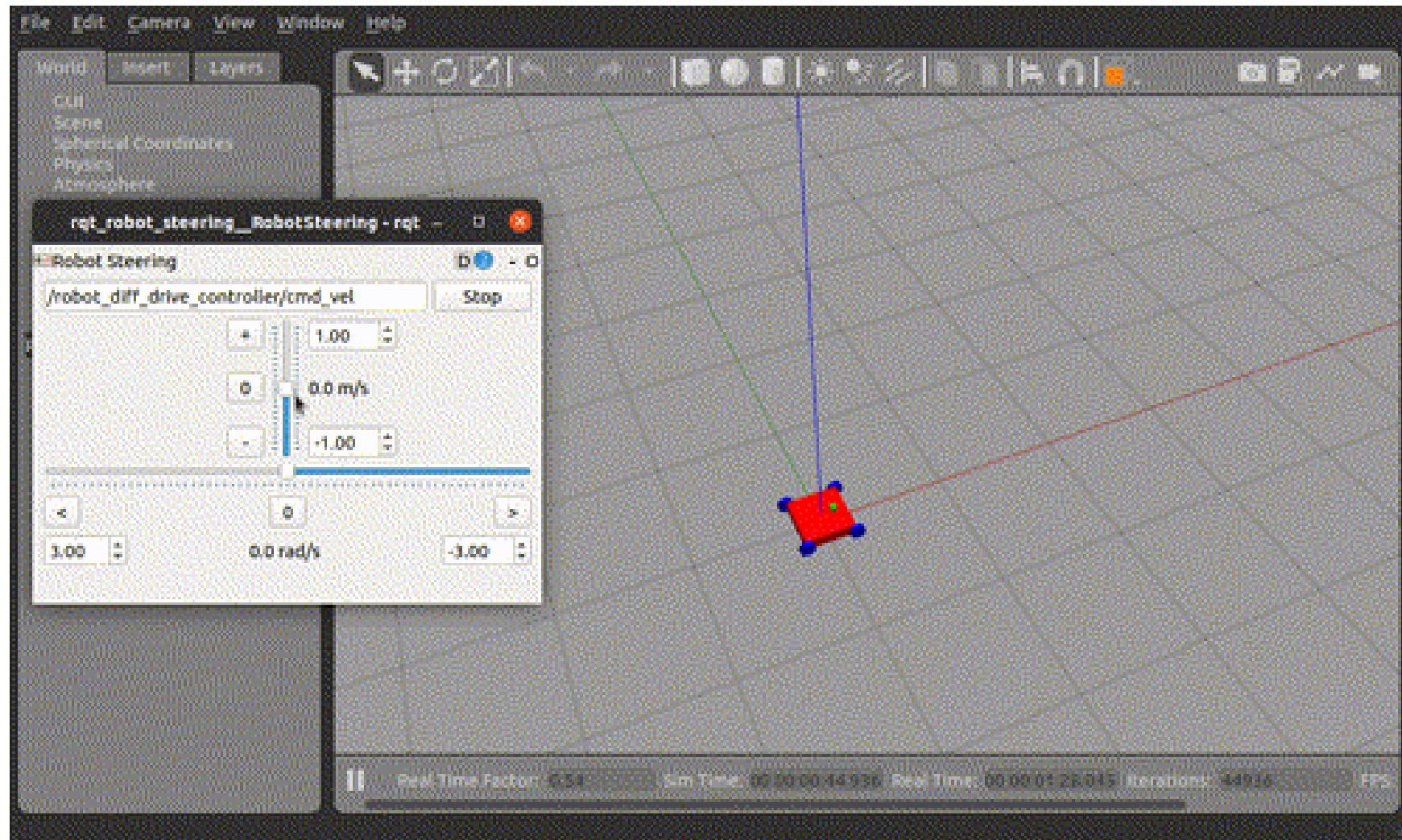
# Differential Drive Robot

```
<link name="left_wheel">
  <visual>
    <geometry>
      <cylinder radius="0.04" length="0.05" />
    </geometry>
    <material name="red">
      <color rgba="1 0 0 1"/>
    </material>
  </visual>
</link>
<link name="right_wheel">
  <visual>
    <geometry>
      <cylinder radius="0.04" length="0.05" />
    </geometry>
    <material name="red">
      <color rgba="1 0 0 1"/>
    </material>
  </visual>
</link>
```

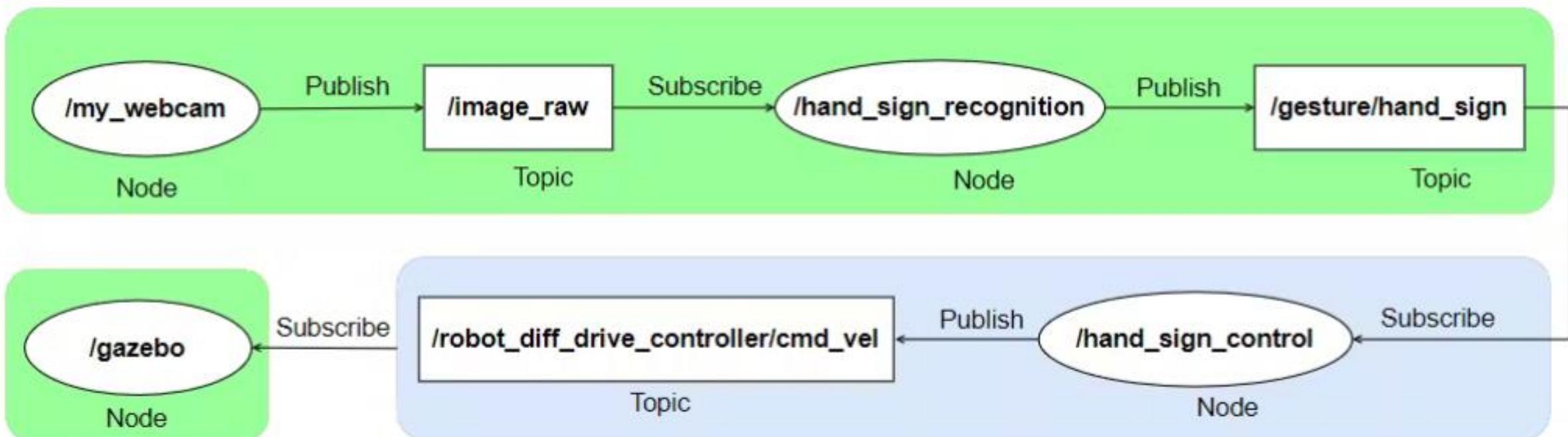
```
<joint name="base_link_right_wheel" type="continuous">
  <parent link="base_link" />
  <child link="right_wheel" />
  <origin xyz="0.2 -0.2 -0.05" rpy="1.570796 0 0"/>
</joint>
```



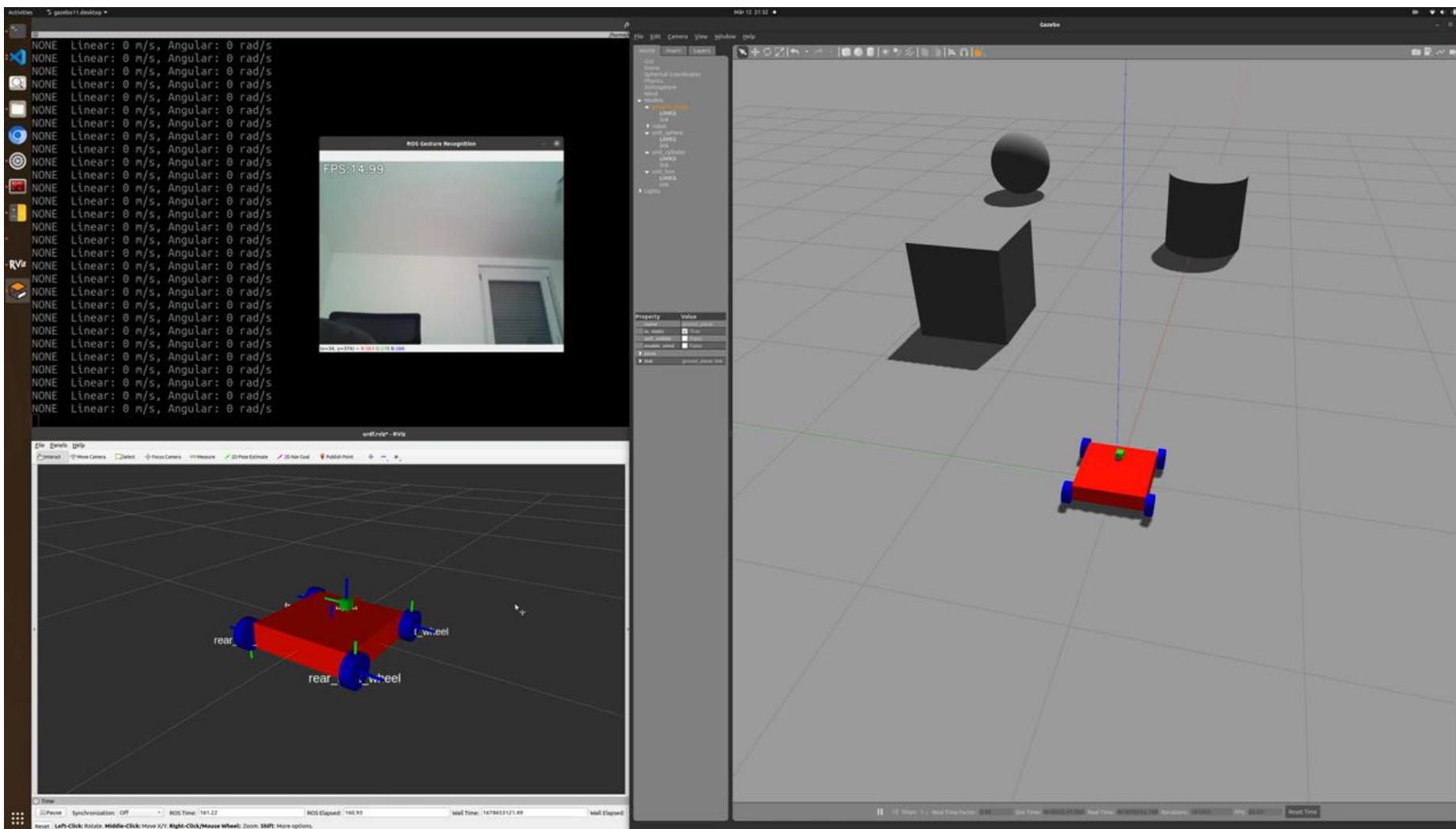
# Driving the Robot



# Hand Gesture Driving



# Hand Gesture Driving



# URDF (Unified Robot Description Format)

