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

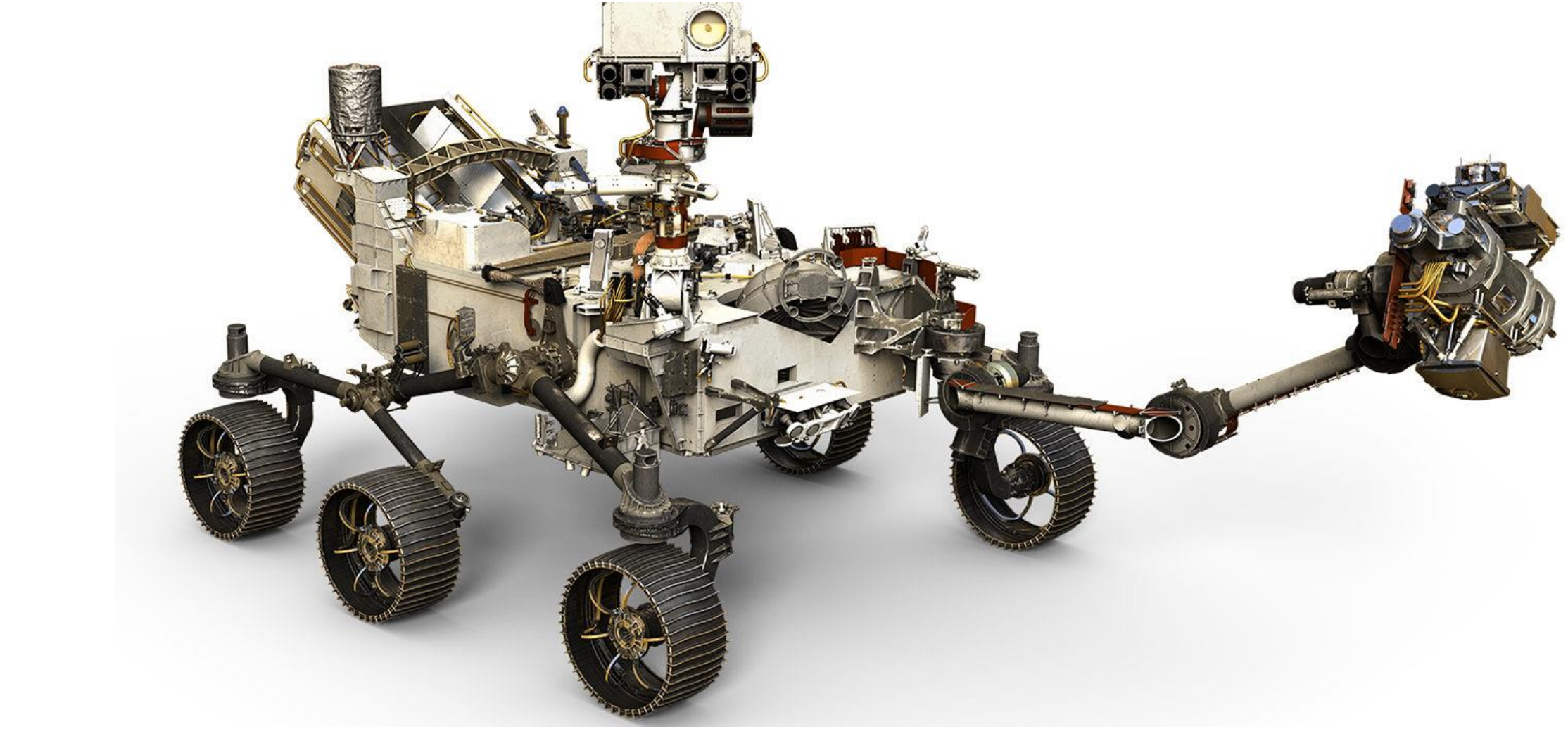
The first artificial robotic arm to be controlled by computer was designed at Rancho Los Amigos Hospital in Downey in 1963

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

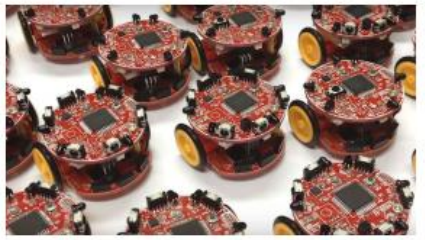
Robotic Arm *Walking Robots* *Wheeled Robots* *Underwater Vehicles* *Aerial Vehicles* *Swarm Robots*





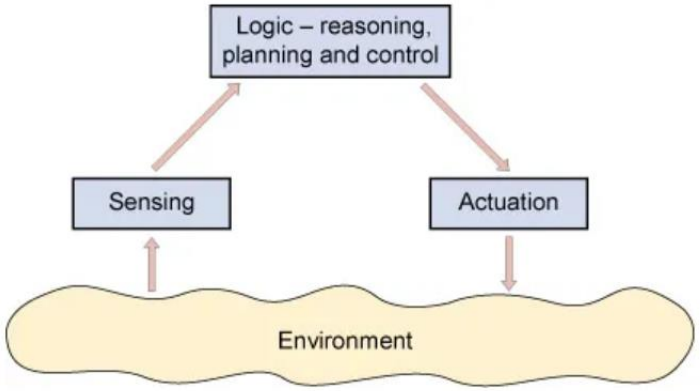


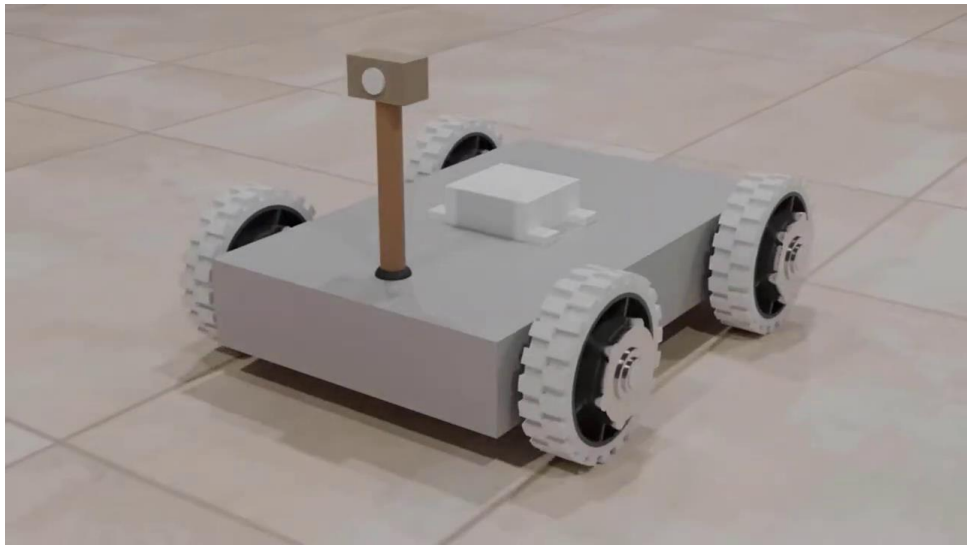



Robot

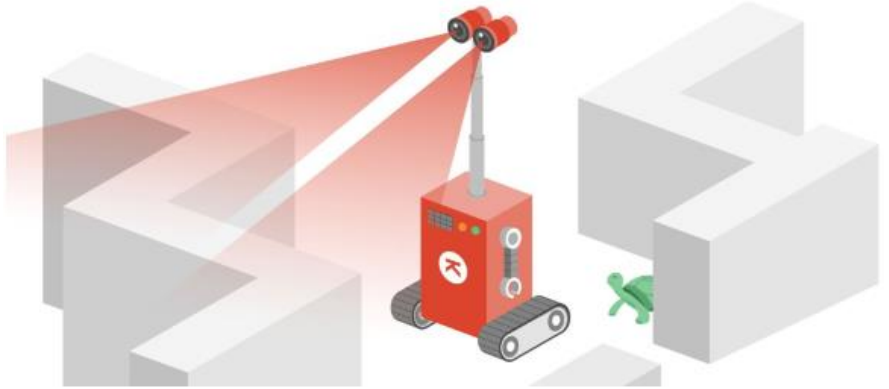
- A robot is any system that can perceive the environment i.e, its surroundings (using sensors), take decisions based on the state of the environment(using computation and algorithms) and is able to execute the instructions generated(using actuators).

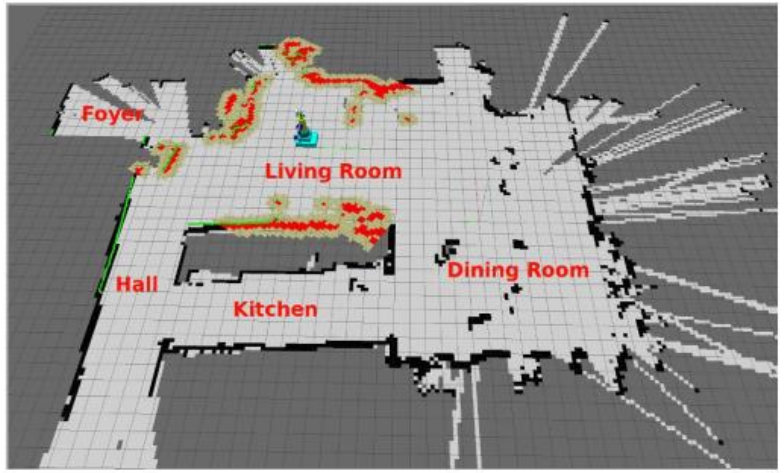




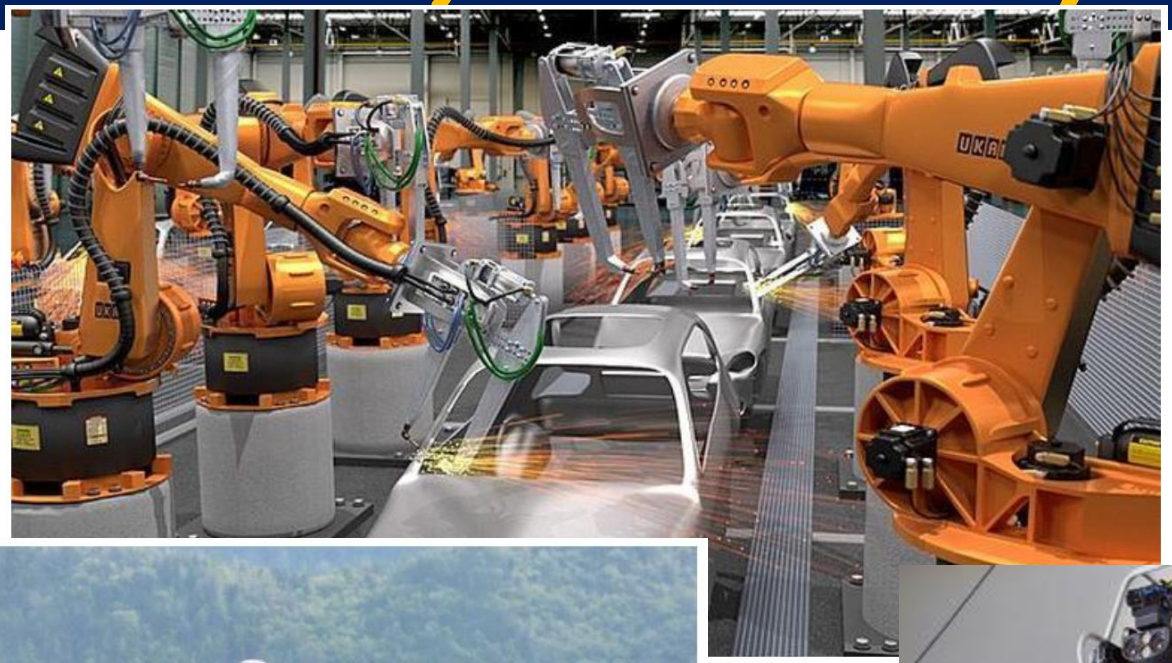
Subfields

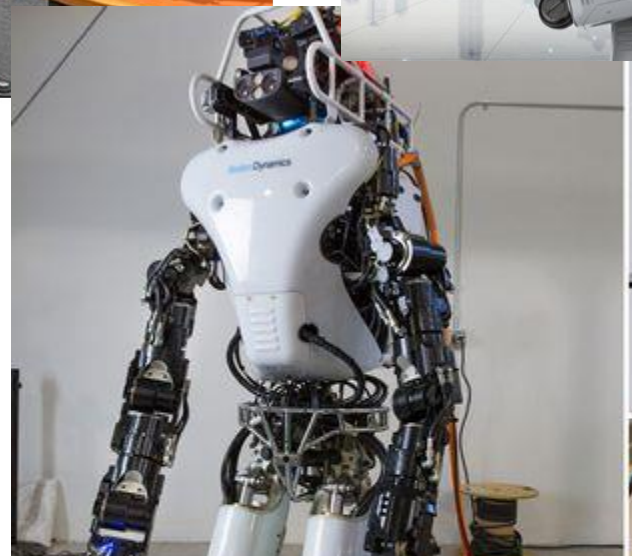
- The robotics field is somewhat unique in that it requires expertise in many fields across the engineering, sciences, and arts disciplines.
- **Kinematics:**
  - Describing the position and motion of components relative to each other without considering forces.
- **Dynamics:**
  - Describing the motion 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.





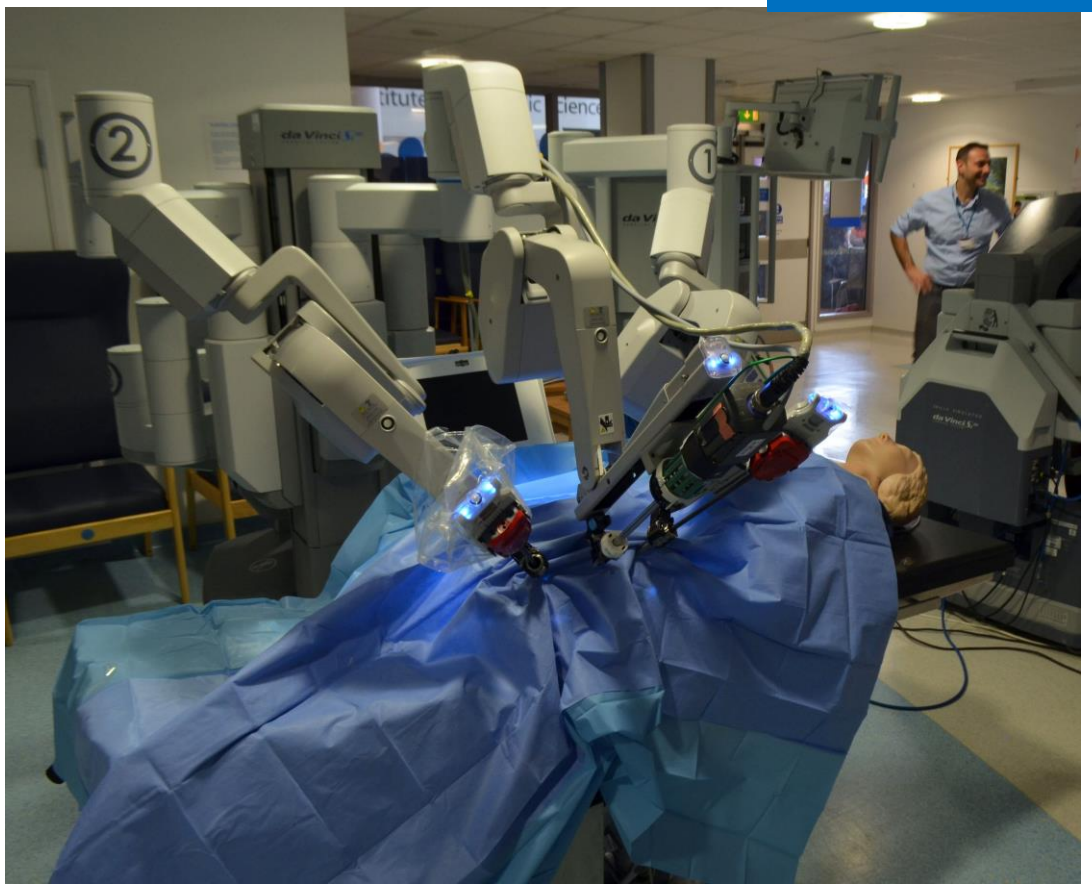
Types of Robots (e.g., industrial, service, humanoid)





Applications of Robotics

Healthcare



Applications of Robots

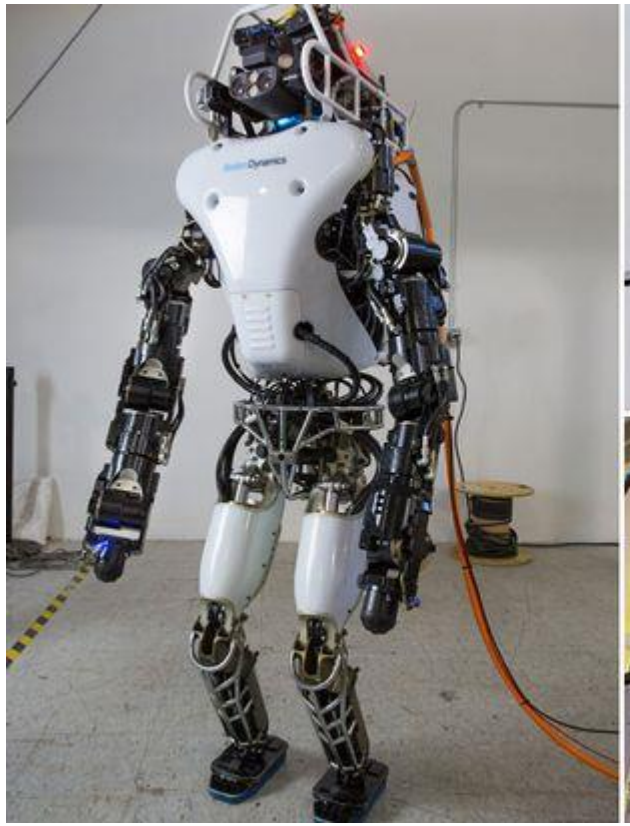
Manufacturing

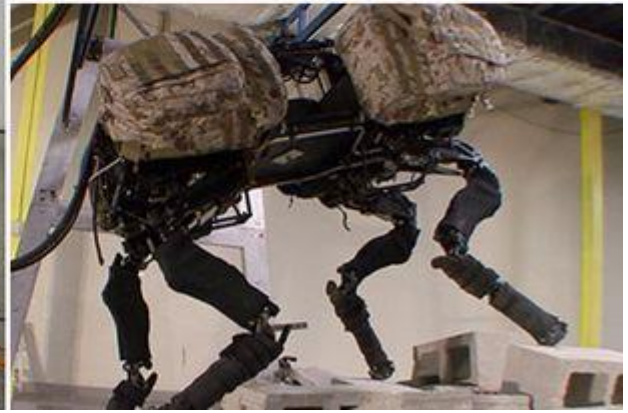
Applications of Robots

Automotive

Applications of Robots

Military



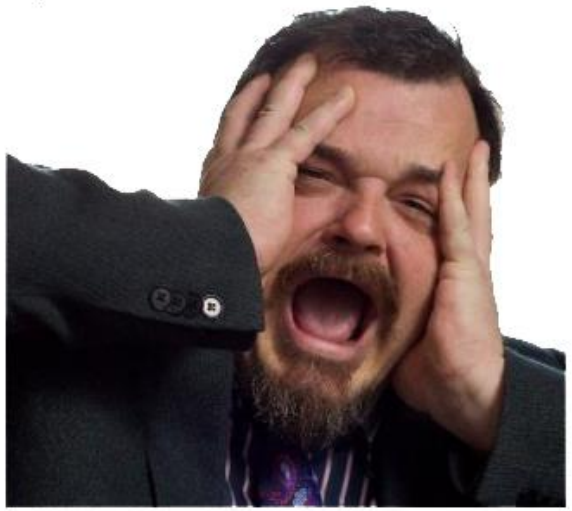


Applications

- 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

ROS

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

PLATFORM

Ubuntu Linux

OS X

PLATFORM

Ubuntu Linux

OS X

Windows

Inter Communication

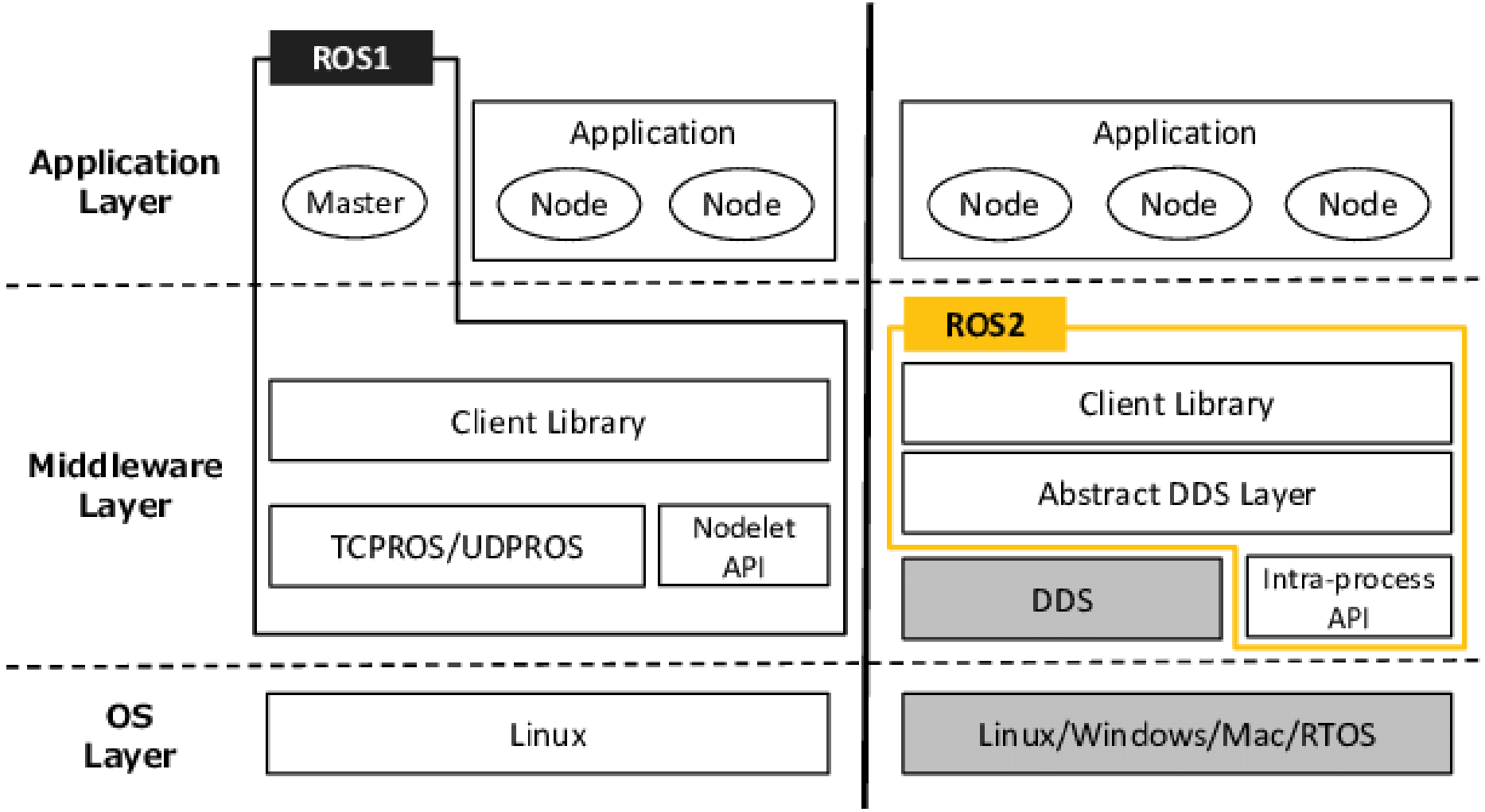
Using rosmaster

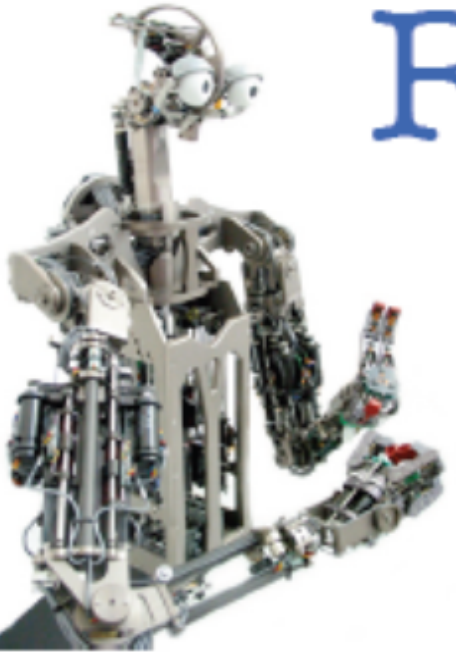
Nodes communicate through rosmaster

Inter Communication

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

ROS1 vs ROS2


ROS1
Application Layer
Application
Master
Node
Node
Application
Node
Node
Node
ROS2
Middleware Layer
Client Library
Abstract DDS Layer
TCPROS/UDPROS
Nodelet API
DDS
Intra-process API
Client Library
OS Layer
Linux
Linux/Windows/Mac/RTOS



ROS

navigation

task executive

visualization

simulation

perception

control

planning

data logging

message passing

device drivers

real-time capabilities

OS

web browser

email client

window manager

memory management

process management

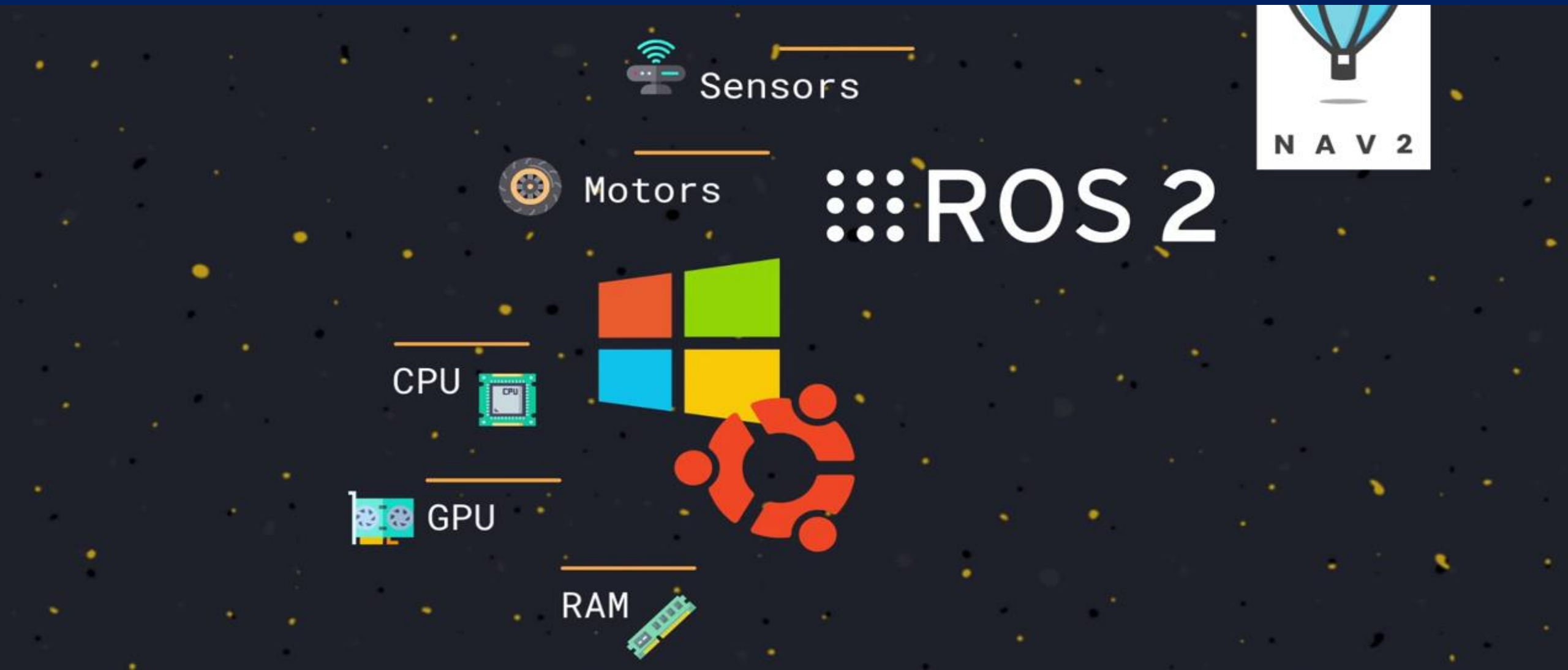
scheduler

device drivers

file system

ROS Benifits

Hardware Abstraction



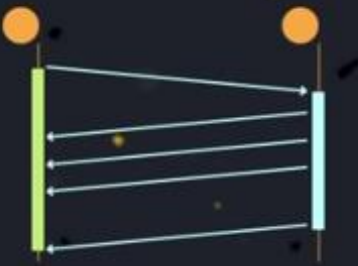
Low Level Device Control

Message Passing

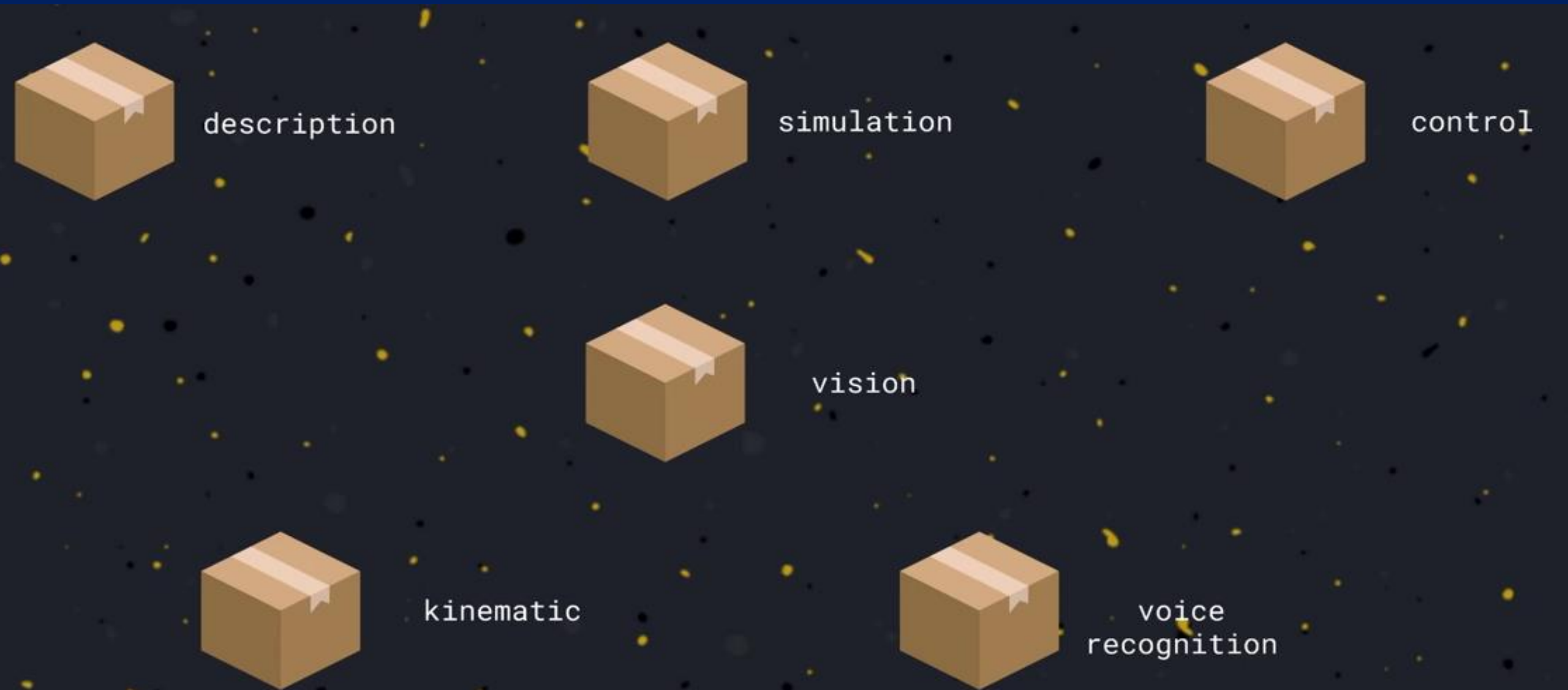
Topic

Service

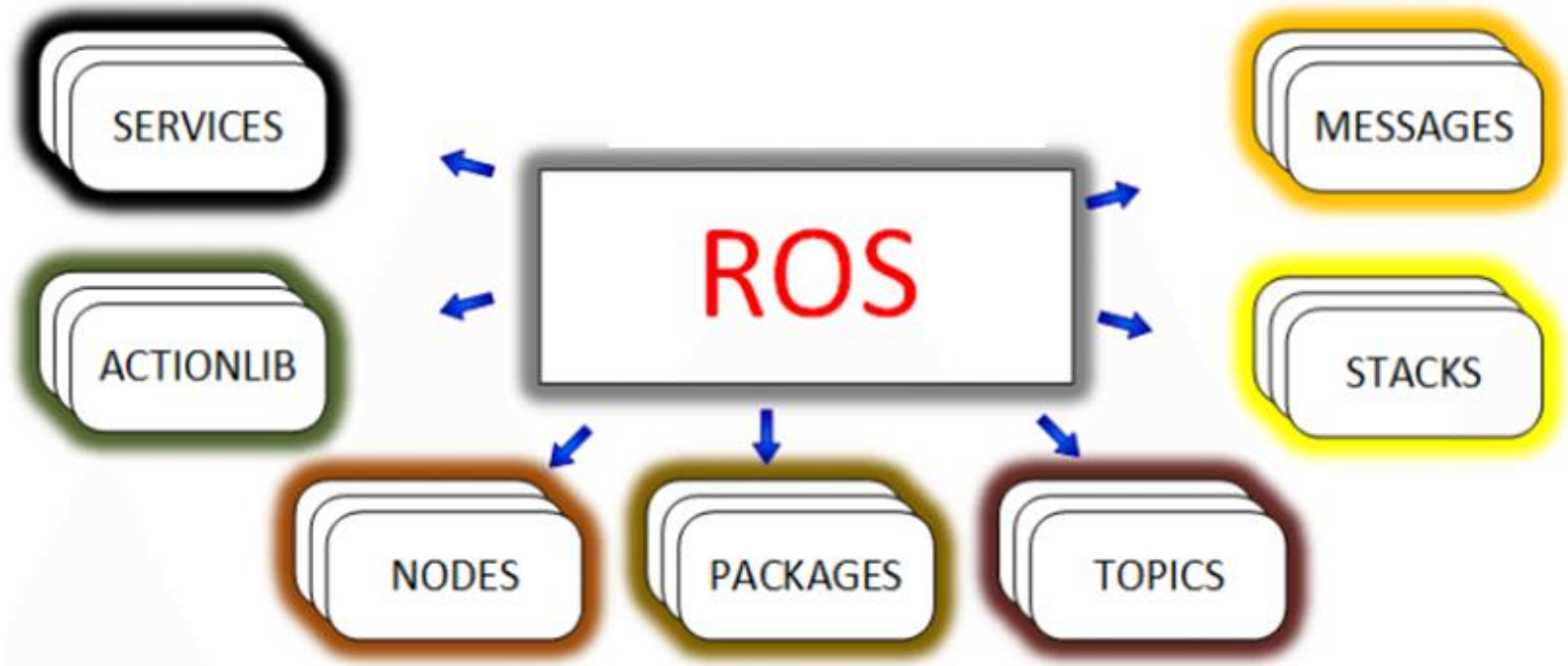
Action



Package Management



Architecture

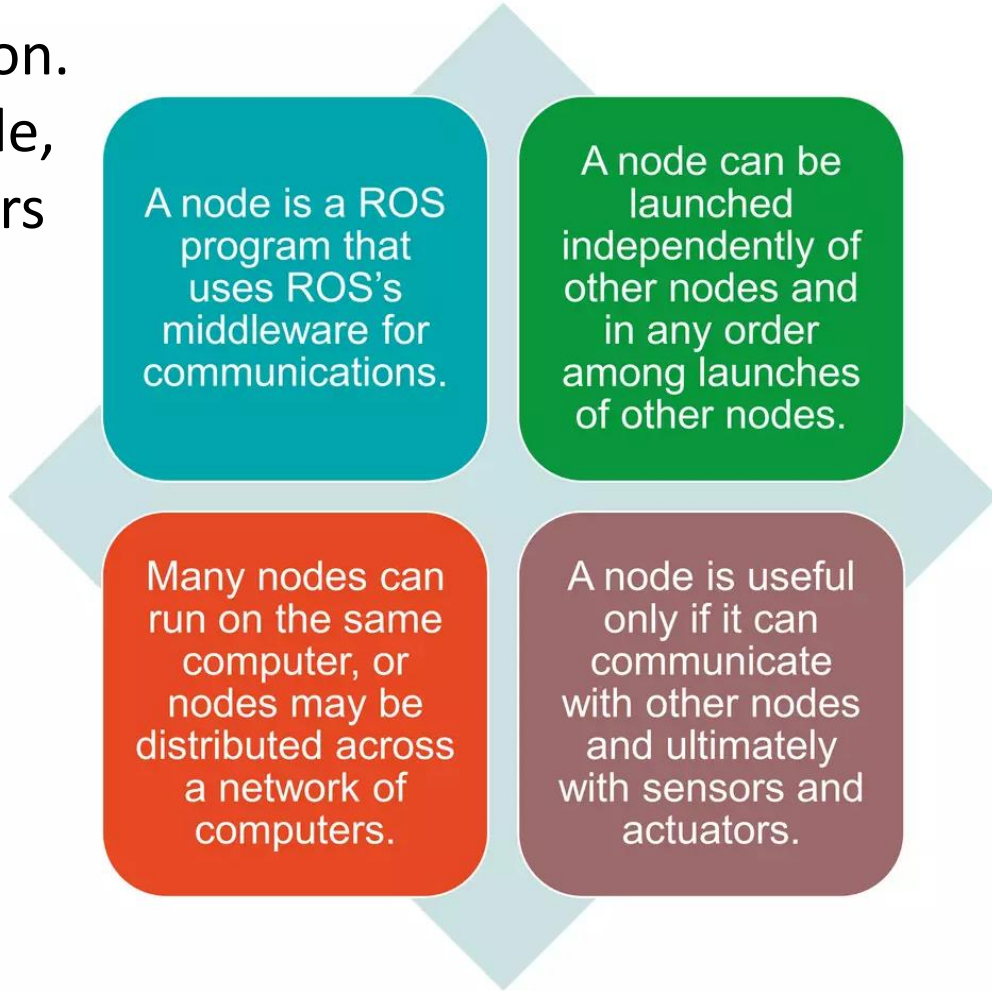


ROS Nodes

- **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.

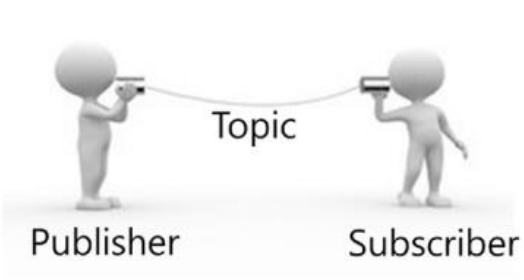
Nodes are written using a ROS **client library**

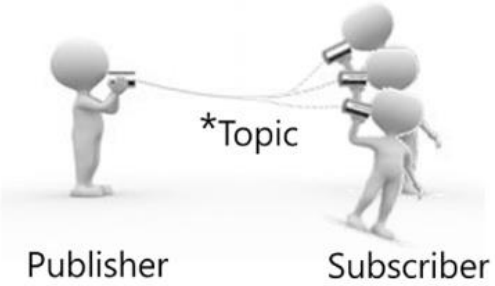
- **rclcpp** – C++ client library
- **rclpy** – python client library

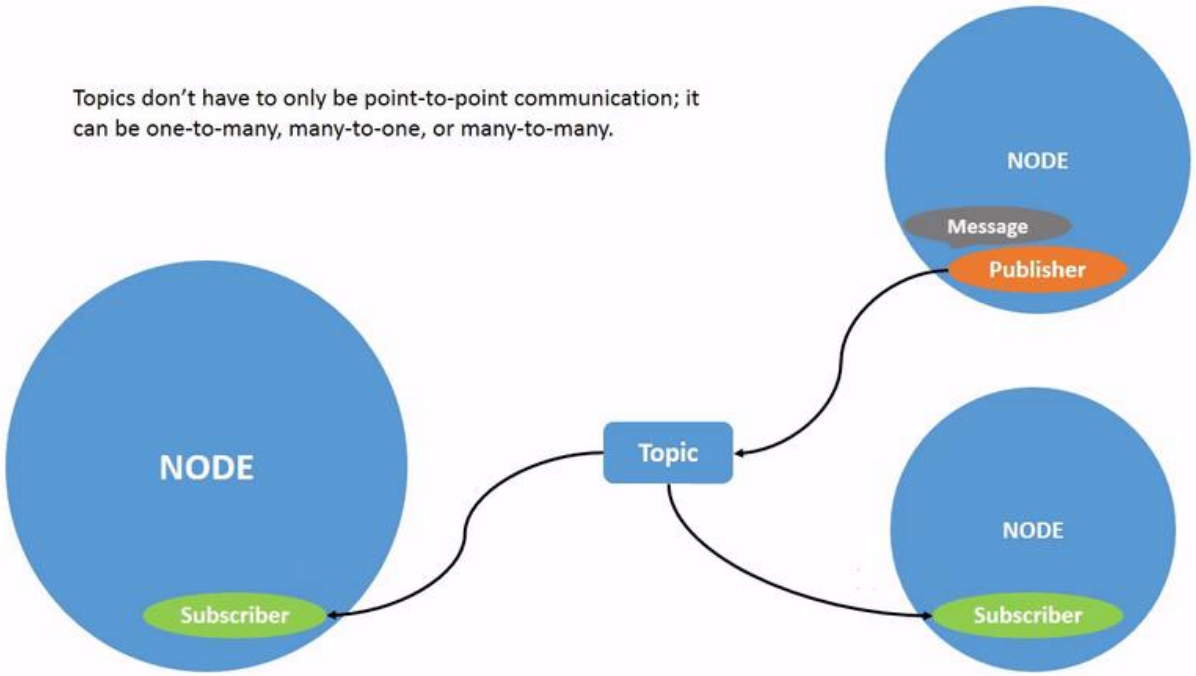


ROS Nodes and Topics

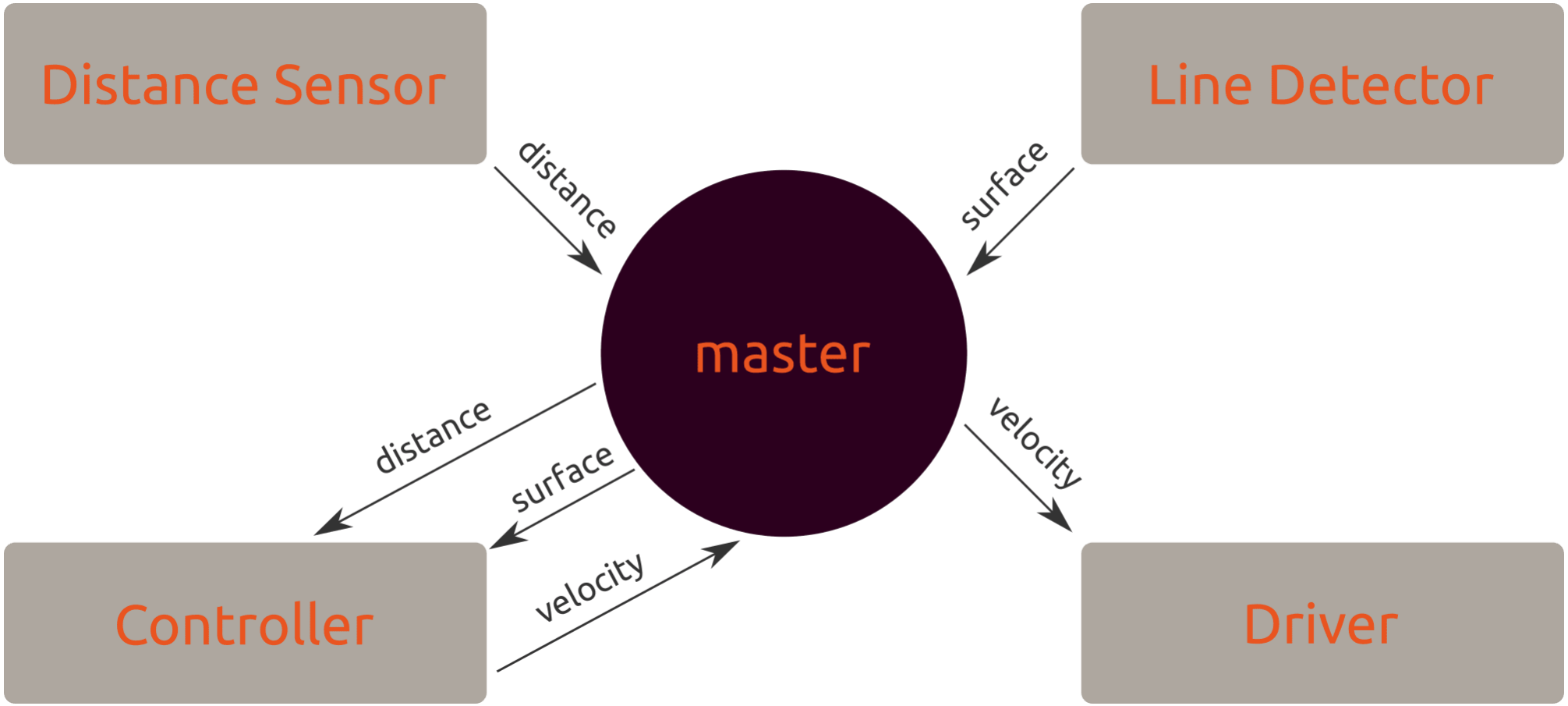
- A node may publish data to any number of topics and simultaneously have subscriptions to any number of topics.
- Nodes communicate with each other by publishing messages to topics
- Topics are communication channels that nodes use to talk to each other.
- Messages are the information robots send to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





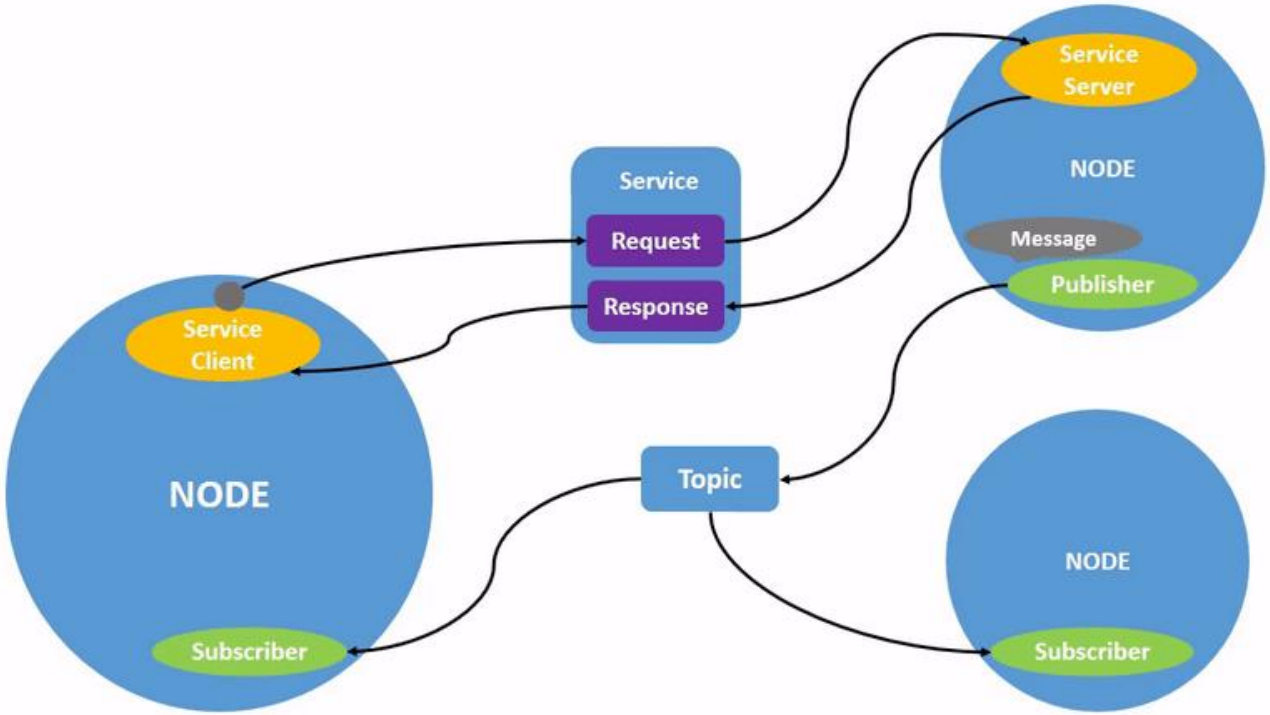


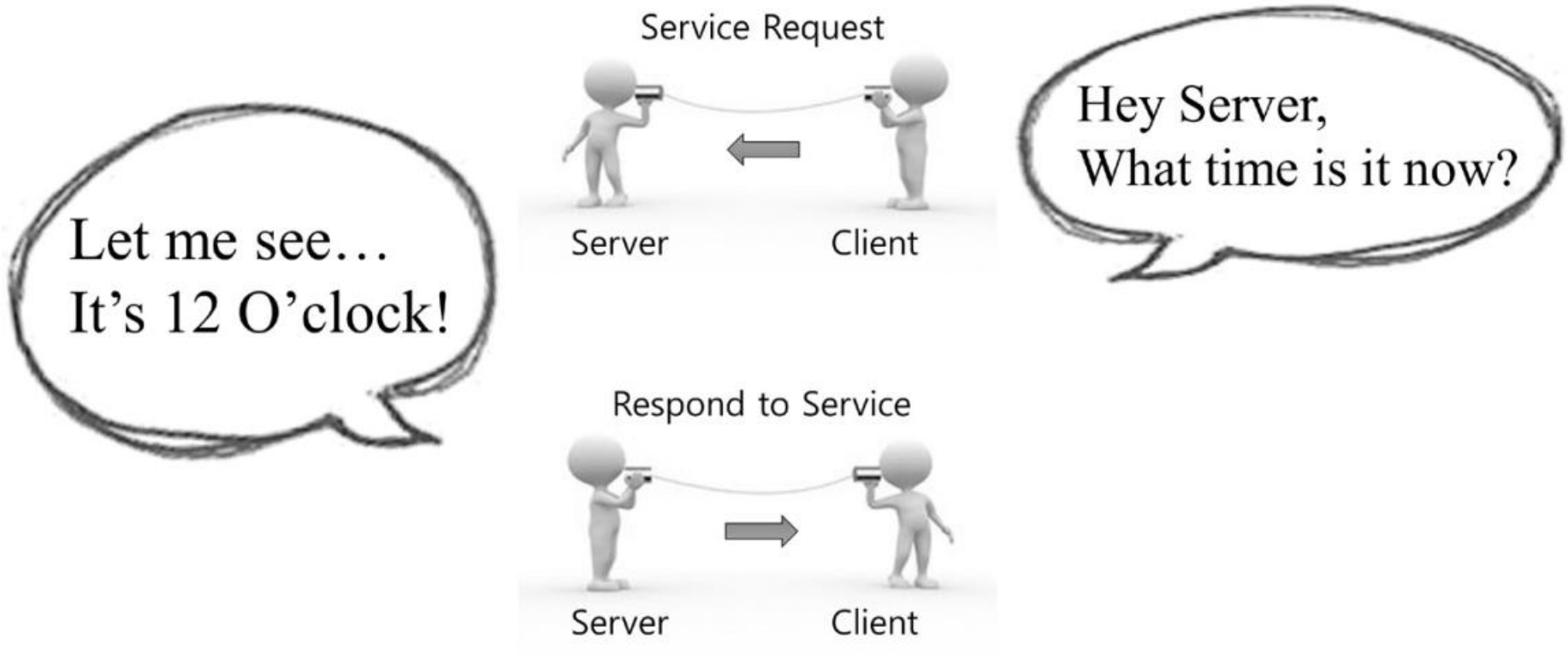
ROS Nodes


Distance Sensor
Line Detector
distance
surface
master
distance
surface
velocity
velocity
Controller
Driver

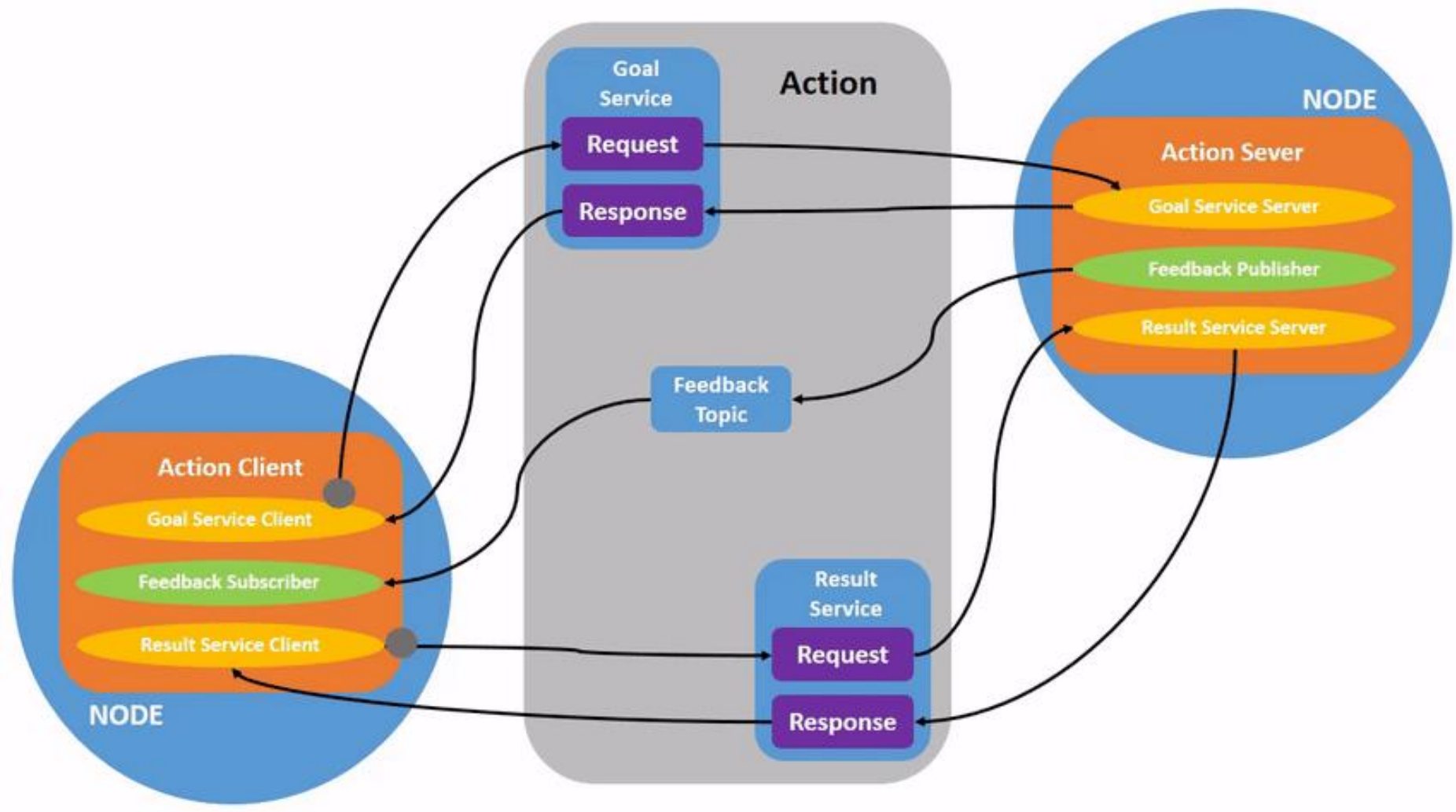
Service Server and Service Client

- 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.




Hey Server,
What time is it now?
Service Request
Server
Client
Let me see…
It's 12 O'clock!
Respond to Service
Server
Client

Actions


Goal
Service
Request
Response
Action
NODE
Action Sever
Goal Service Server
Feedback Publisher
Result Service Server
Feedback
Topic
Action Client
Goal Service Client
Feedback Subscriber
Result Service Client
NODE
Result
Service
Request
Response

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
```

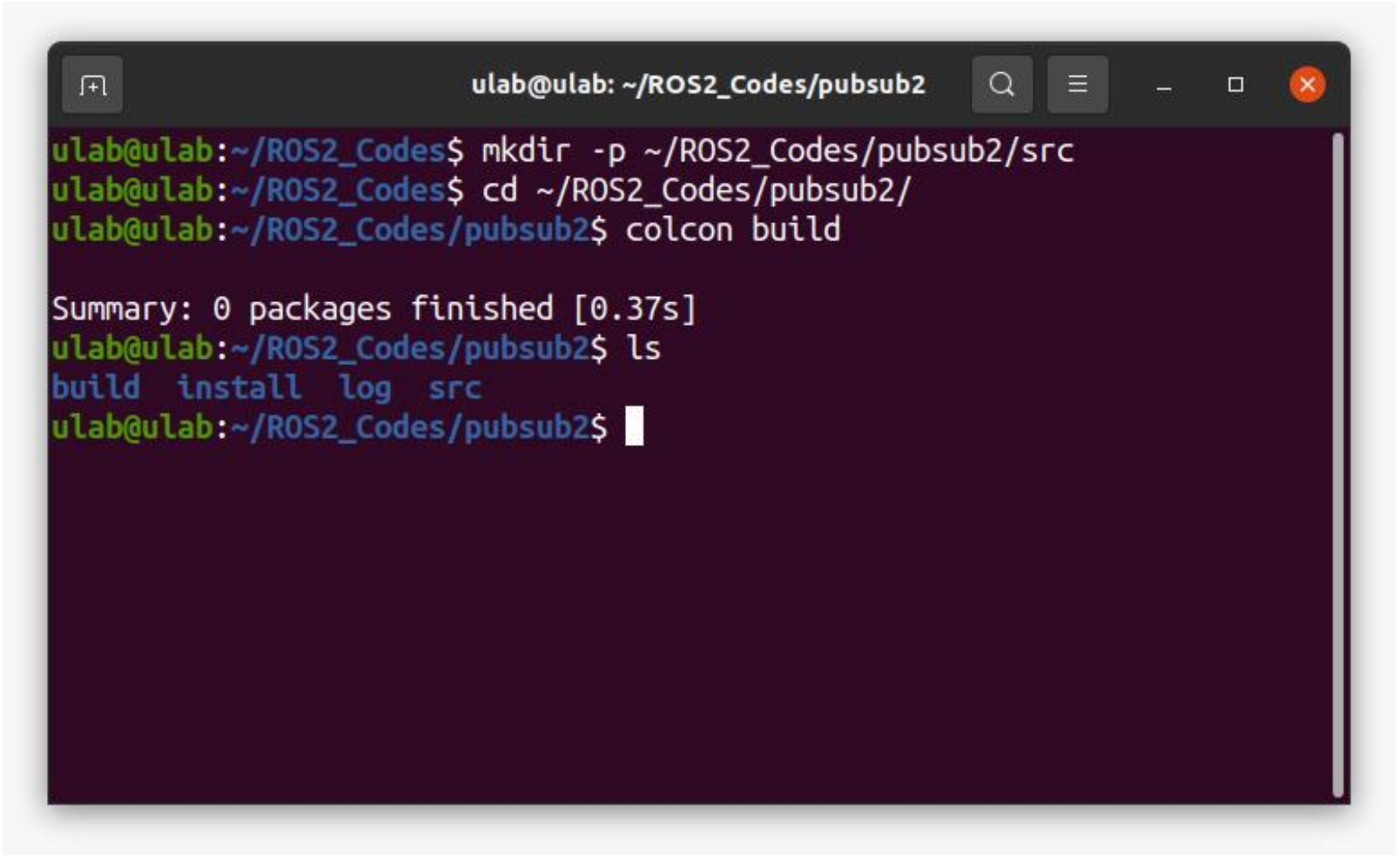
Simple Publisher & Subscriber

```
$ mkdir –p ~/simple_pub_sub

$ cd ~/simple_pub_sub

$ mkdir src
```

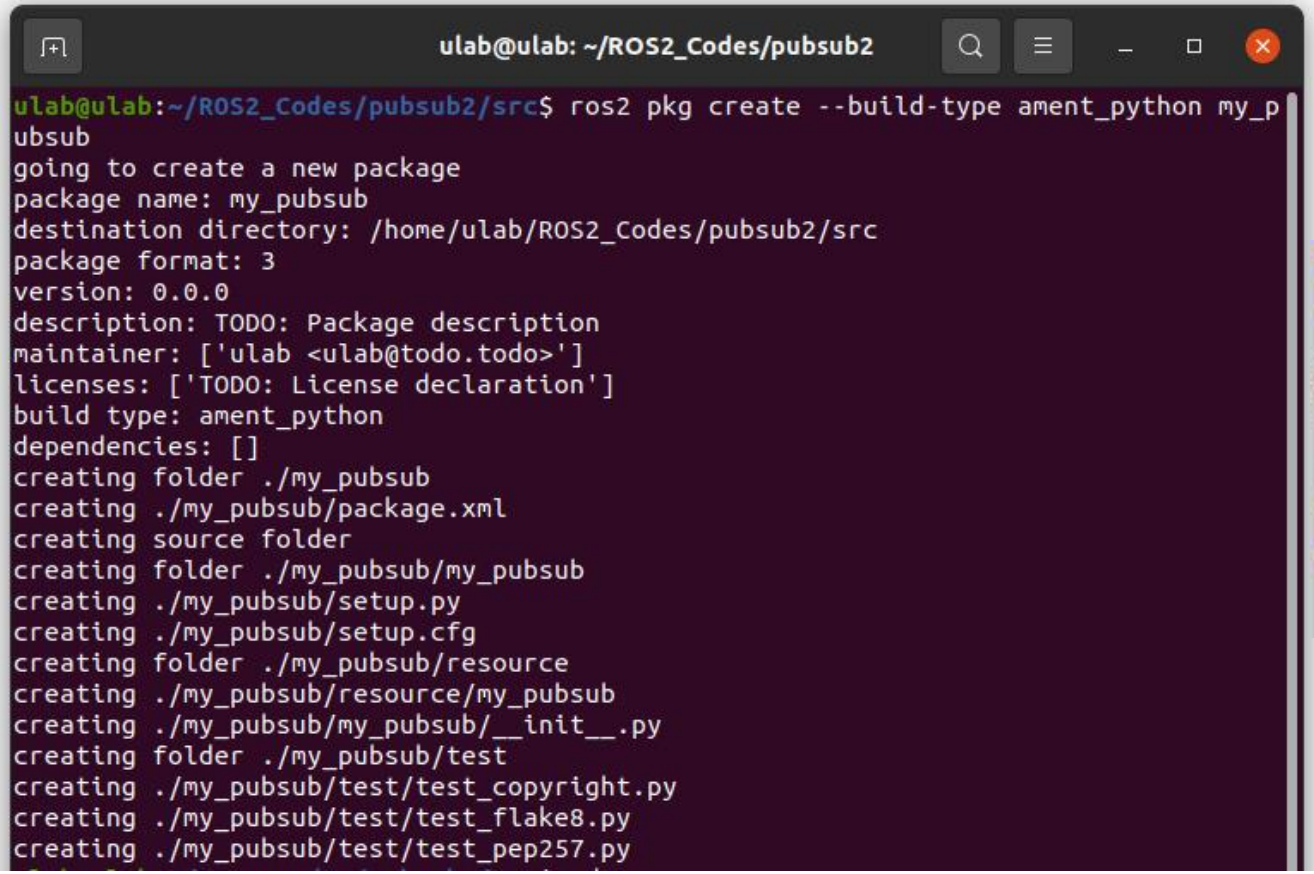
```
$ cd ~/simple_pub_sub/src
$ ros2 create pkg –build-type ament_python pub_sub
```

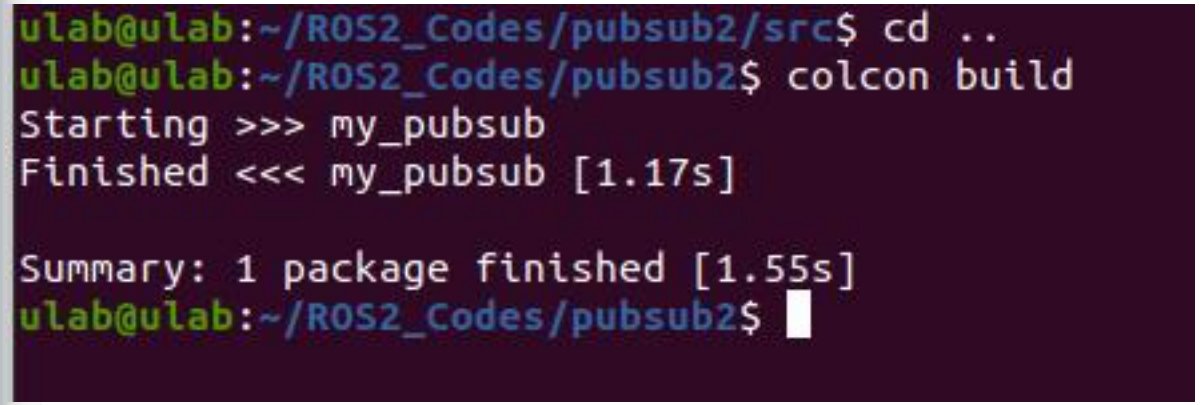
```
cd ..
$ colcon build
```

Simple Publisher & Subscriber



Simple Publisher & Subscriber





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

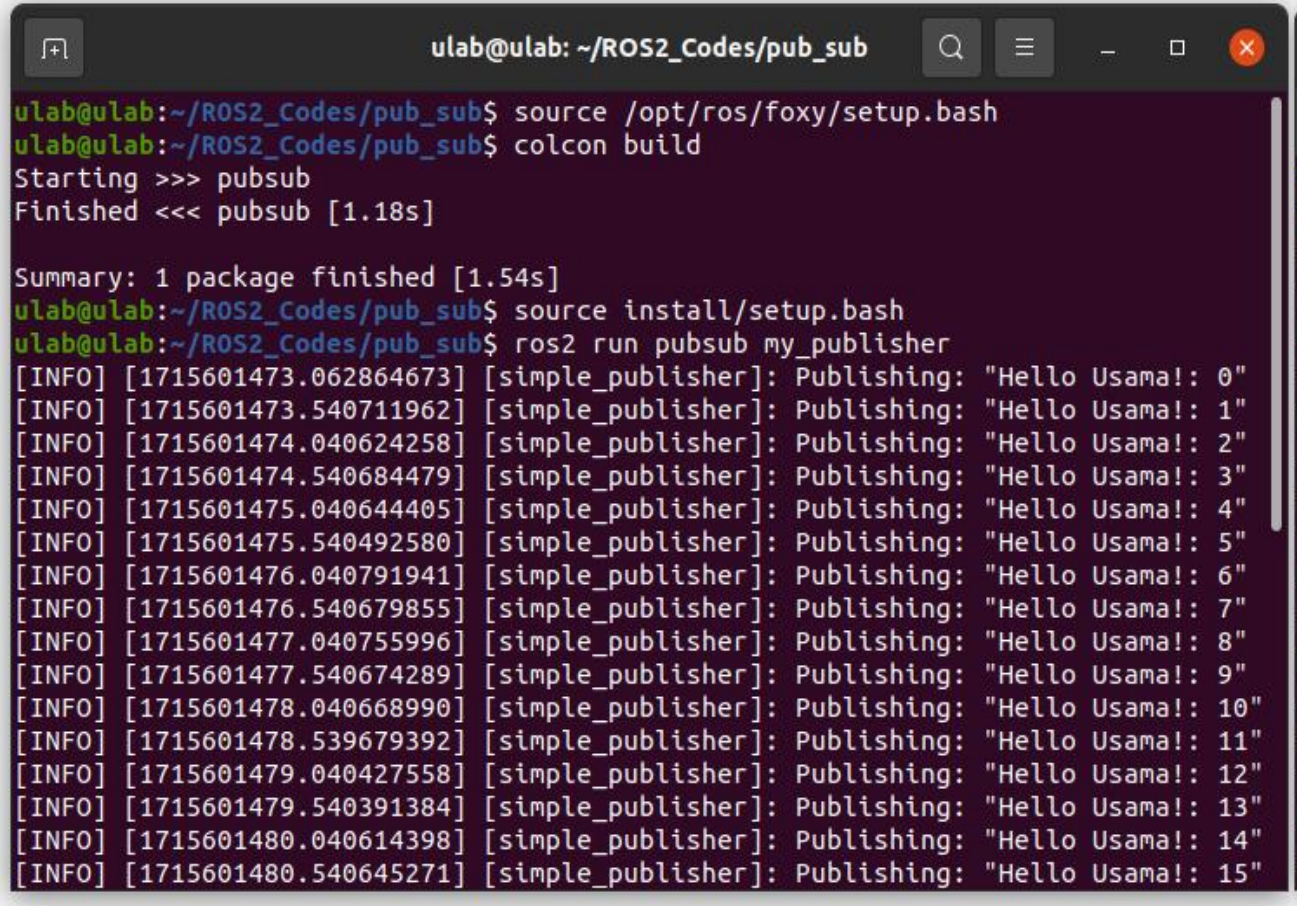
```
 entry_points={
        'console_scripts': ['my_publisher =
pubsub.my_publisher_node:main',
                            'my_subscriber =
pubsub.my_subscriber_node:main',
        ],
    },
```

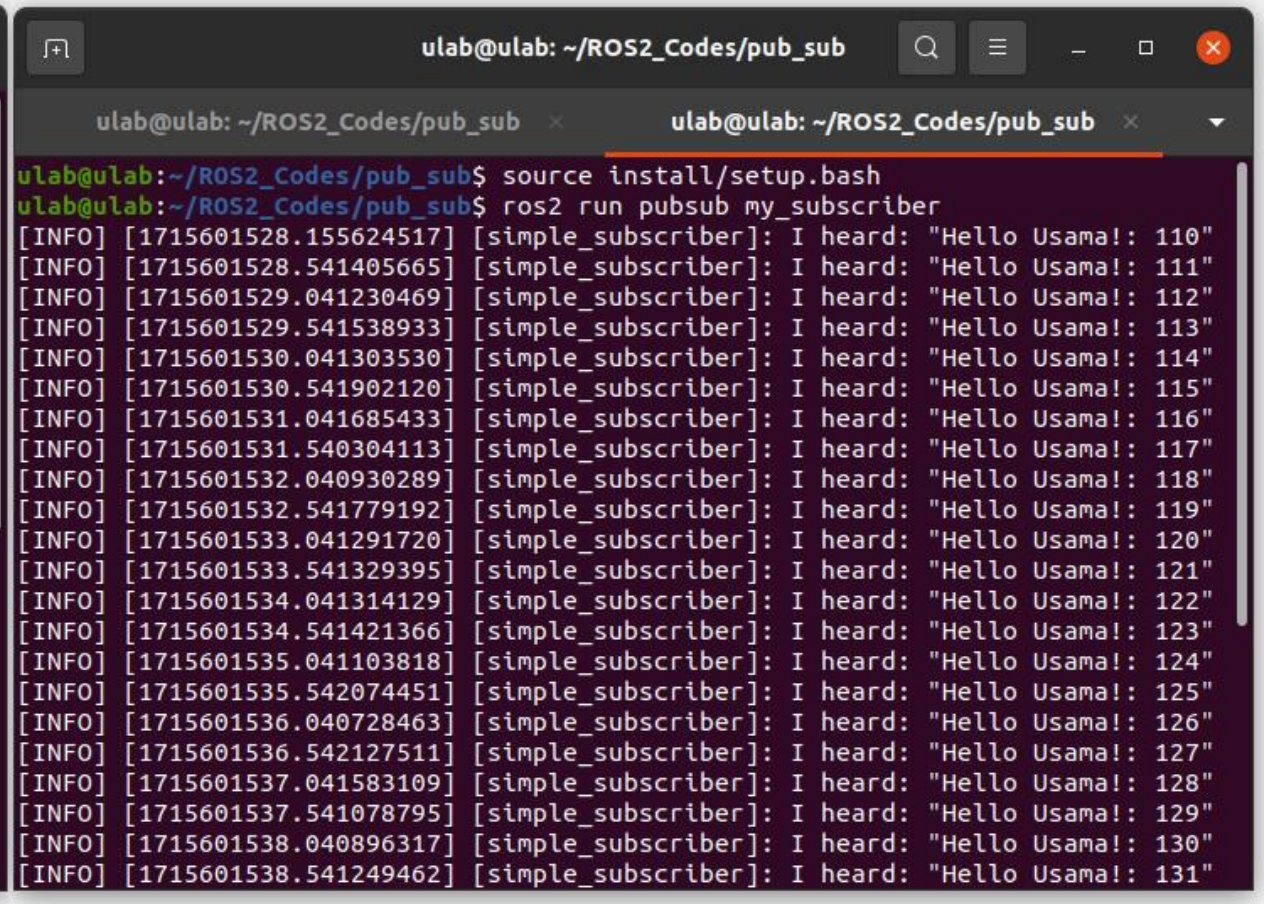
Simple Publisher & Subscriber

Setup.py

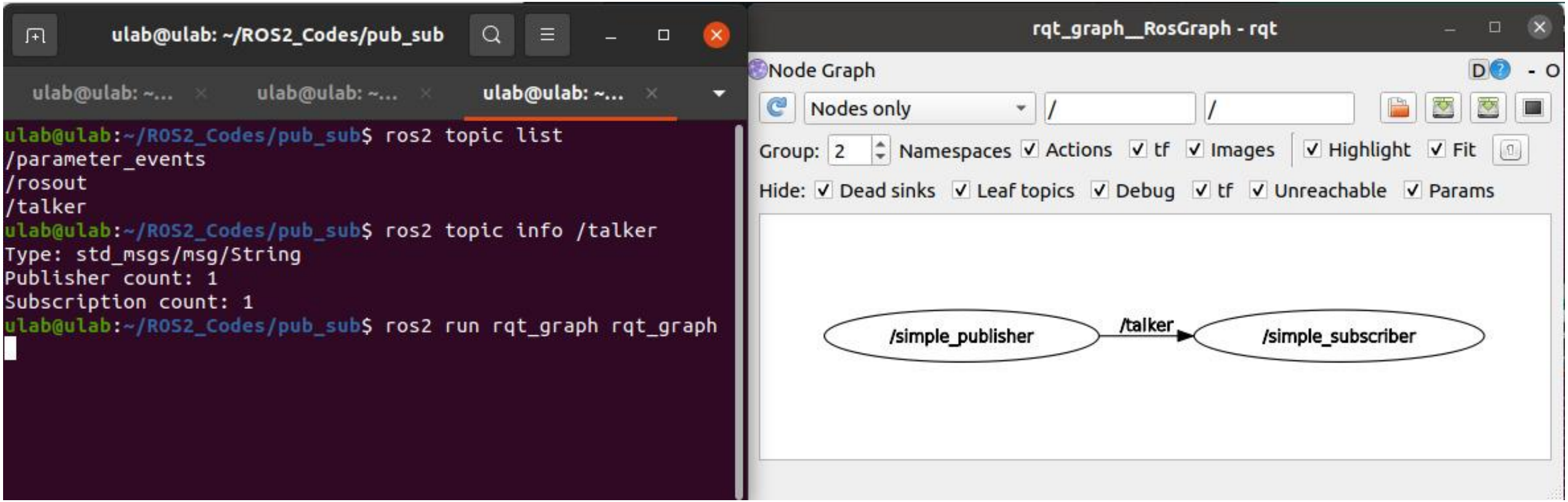
```
 entry_points={
        'console_scripts': ['my_publisher =
pubsub.my_publisher_node:main',
                            'my_subscriber =
pubsub.my_subscriber_node:main',
        ],
    },
```

Simple Publisher & Subscriber





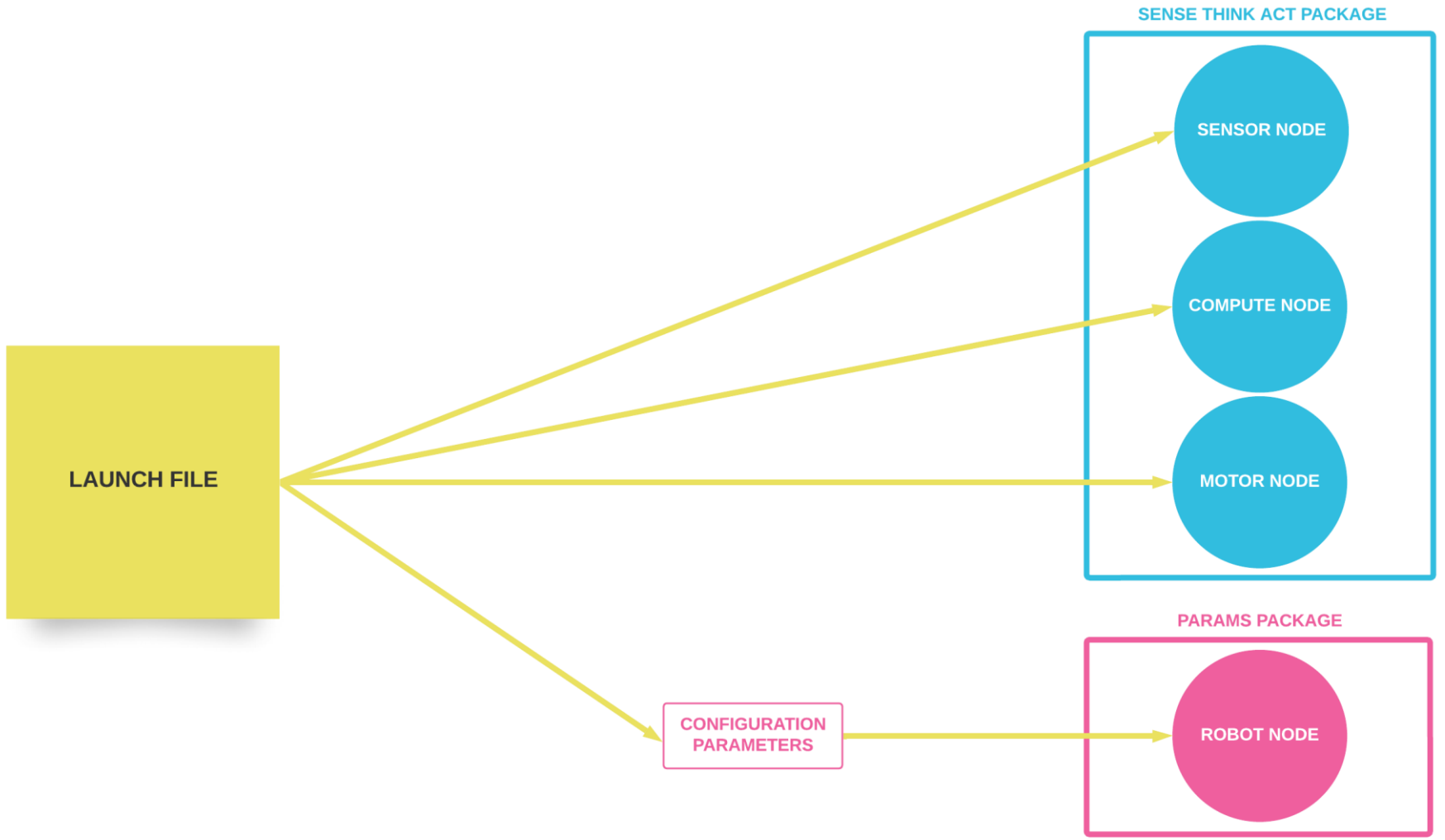
Simple Publisher & Subscriber



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


SENSE THINK ACT PACKAGE
SENSOR NODE
COMPUTE NODE
MOTOR NODE
LAUNCH FILE
CONFIGURATION PARAMETERS
PARAMS PACKAGE
ROBOT NODE

Launch File

Create launch folder → Make launchfile.launch.py

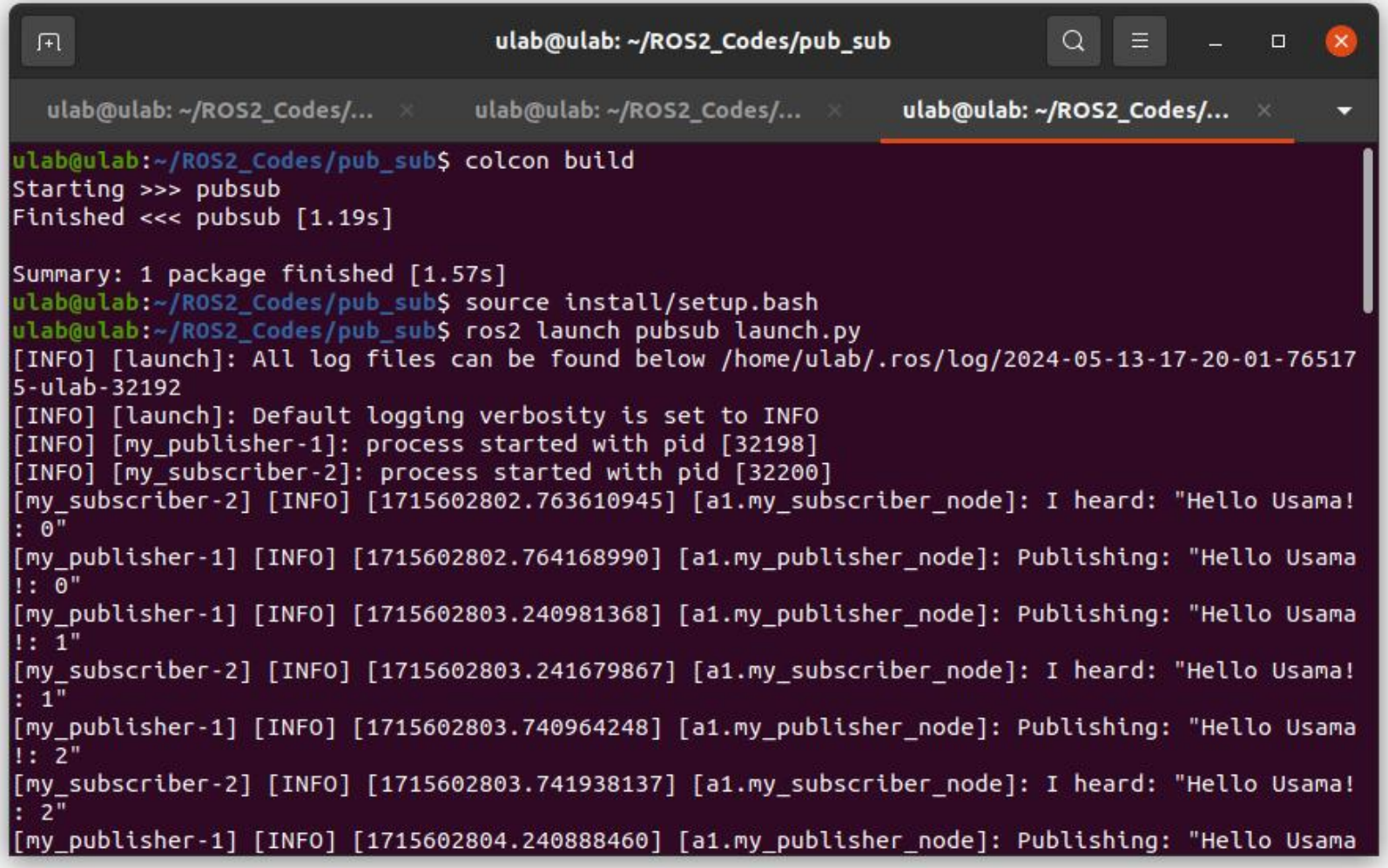
```
from launch import LaunchDescription
from launch_ros.actions import Node
def generate_launch_description():
    return LaunchDescription([
        Node(
        package='pubsub',
        namespace='a1',
        executable='my_publisher',
        name='my_publisher_node' ),
    Node(
        package='pubsub',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

Launch File

Edit setup.py

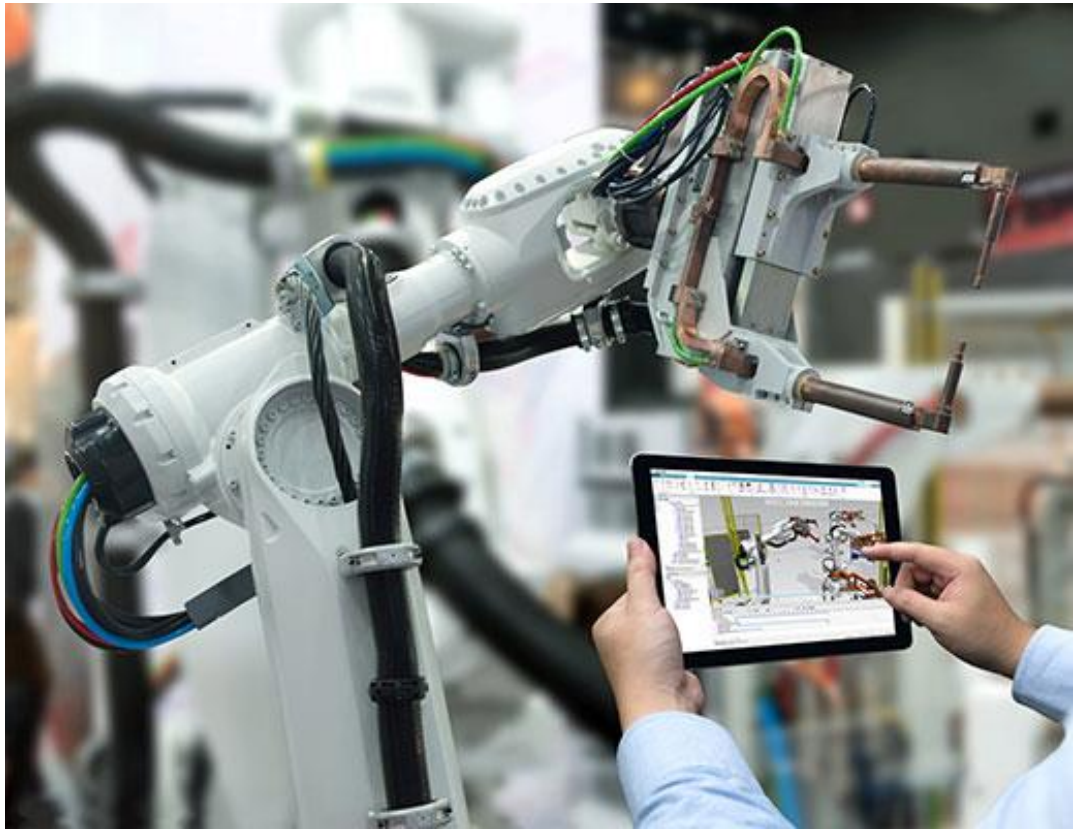
```
import os
from glob import glob
# Other imports ...
package_name = 'pub_sub'
setup(
    # Other parameters ...
    data_files=[
        # ... Other data files
        # Include all launch files.
        (os.path.join('share', package_name, 'launch’),
glob.glob(os.path.join('launch', 'launch.')))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

Launch File


ulab@ulab: ~/ROS2_Codes/pub_sub
ulab@ulab: ~/ROS2_Codes/...
ulab@ulab: ~/ROS2_Codes/...
ulab@ulab: ~/ROS2_Codes/...
ulab@ulab:~/ROS2_Codes/pub_sub$ colcon build
Starting >>> pubsub
Finished <<< pubsub [1.19s]
Summary: 1 package finished [1.57s]
ulab@ulab:~/ROS2_Codes/pub_sub$ source install/setup.bash
ulab@ulab:~/ROS2_Codes/pub_sub$ ros2 launch pubsub launch.py
[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

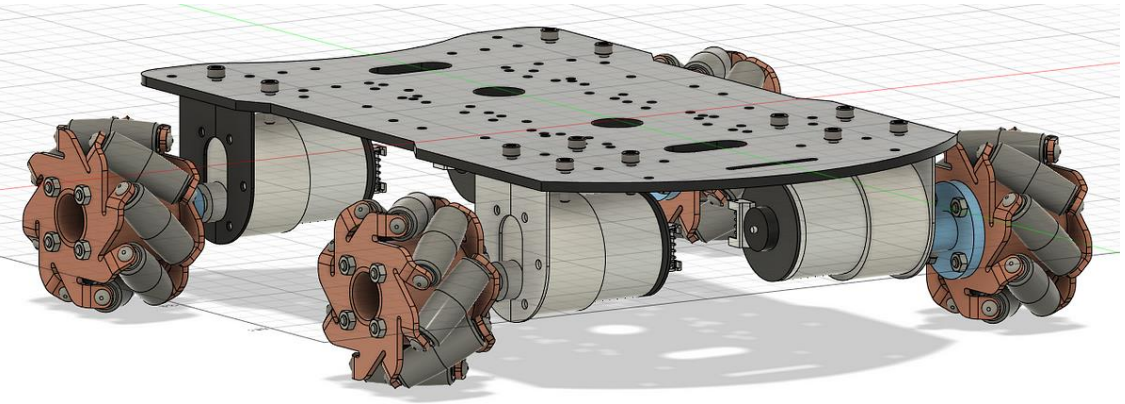
Digital Twin

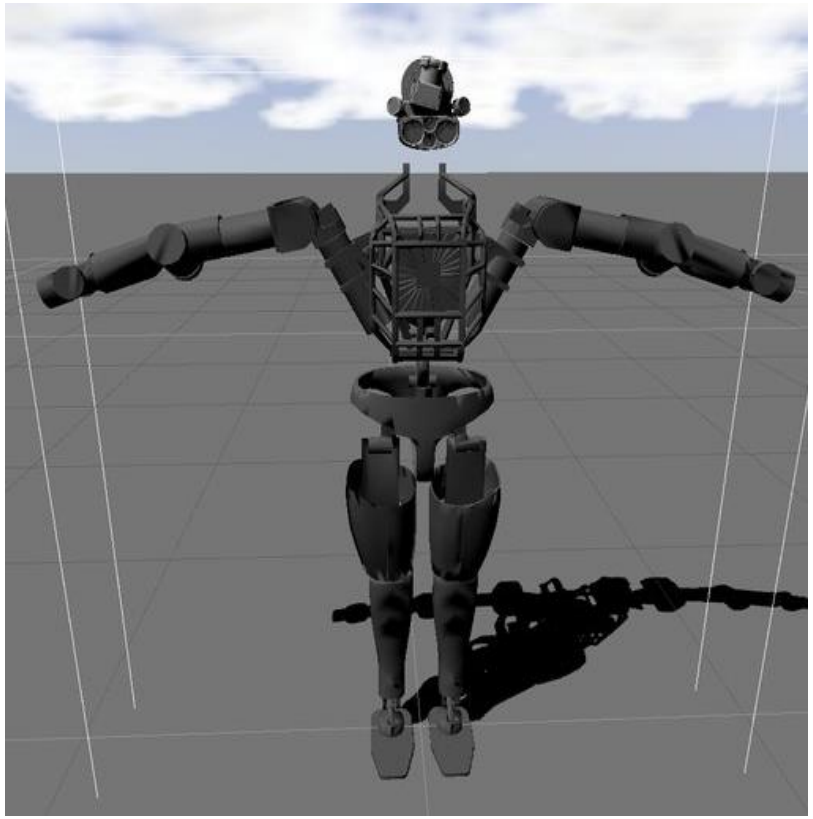
- Digital twins are virtual representations of physical products and processes.
- Digital twins are also vital in robotic manipulation. They help virtualize hardware motion from raw material to the final product.
- Application:
  - Healthcare
  - Manufacturing
  - Aerospace
  - Automotive

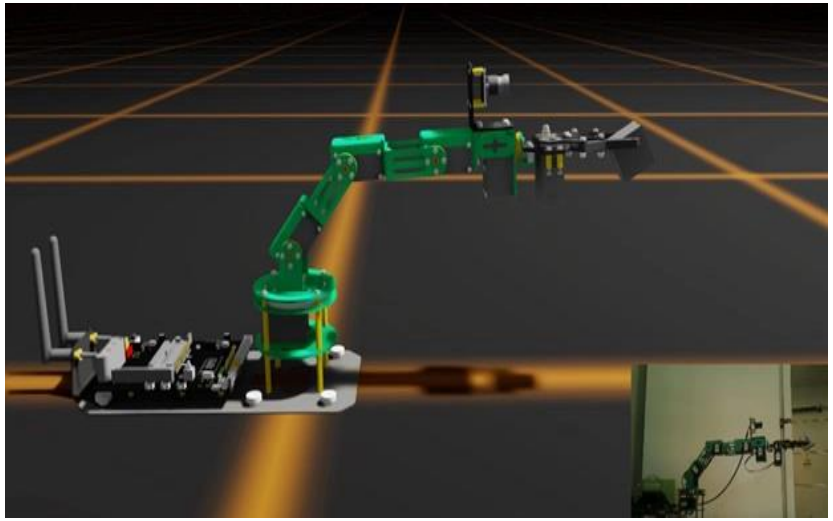


Digital Twin

Digital Twin

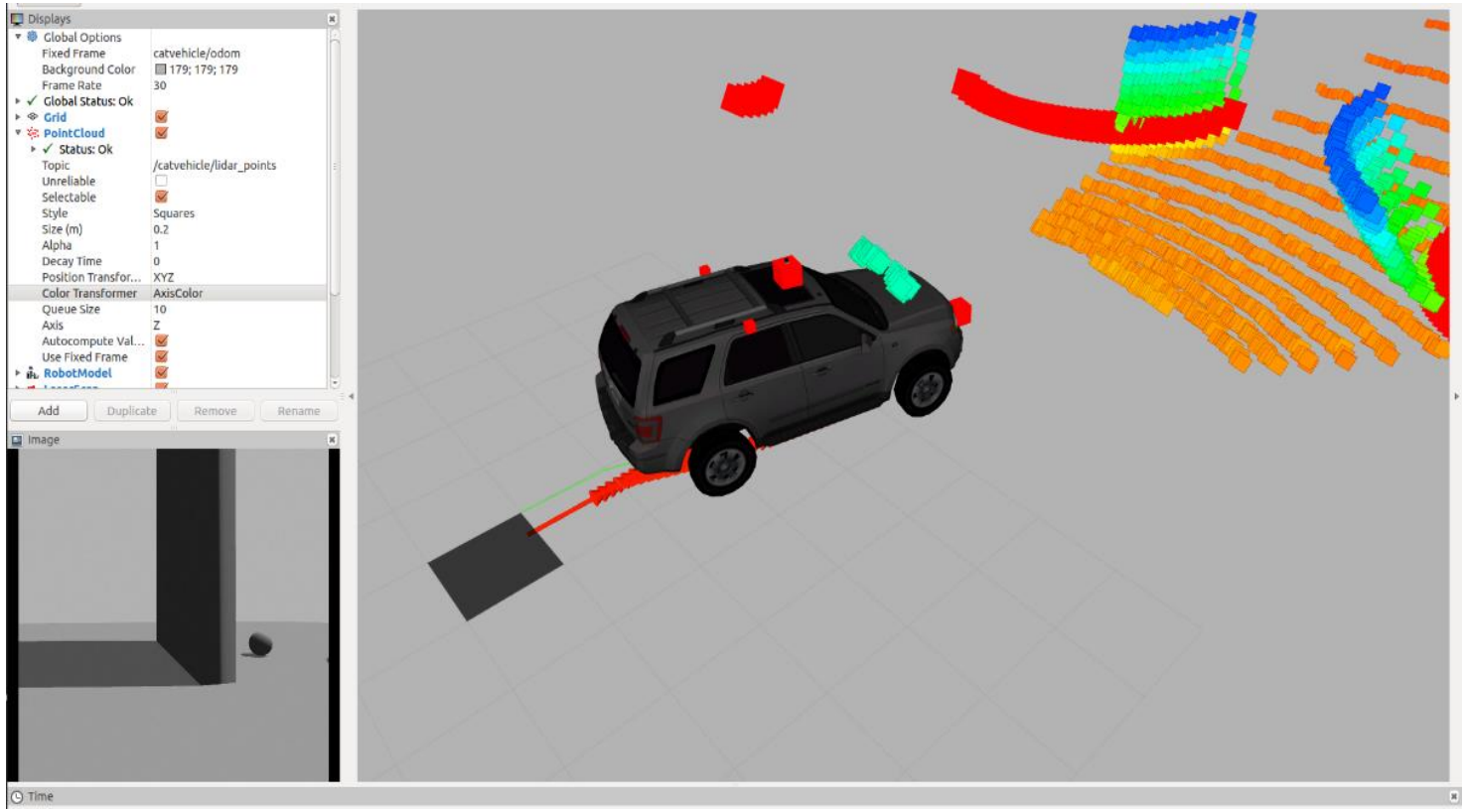






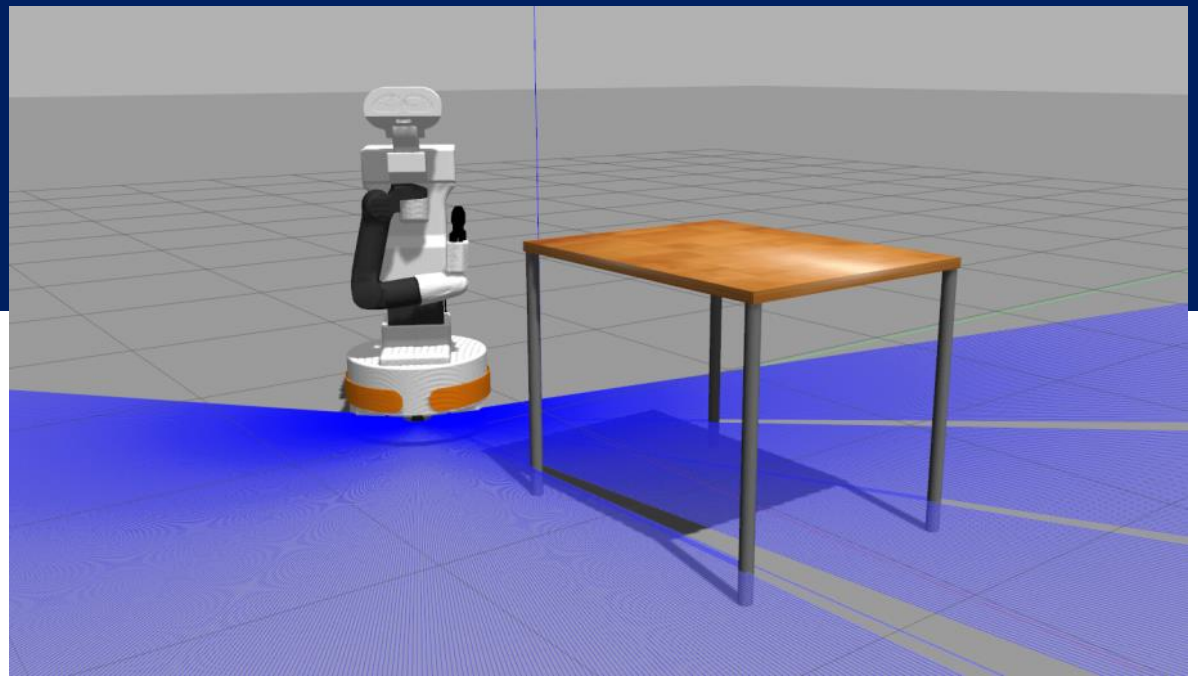
Robot Visualization (Rviz)

- **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**

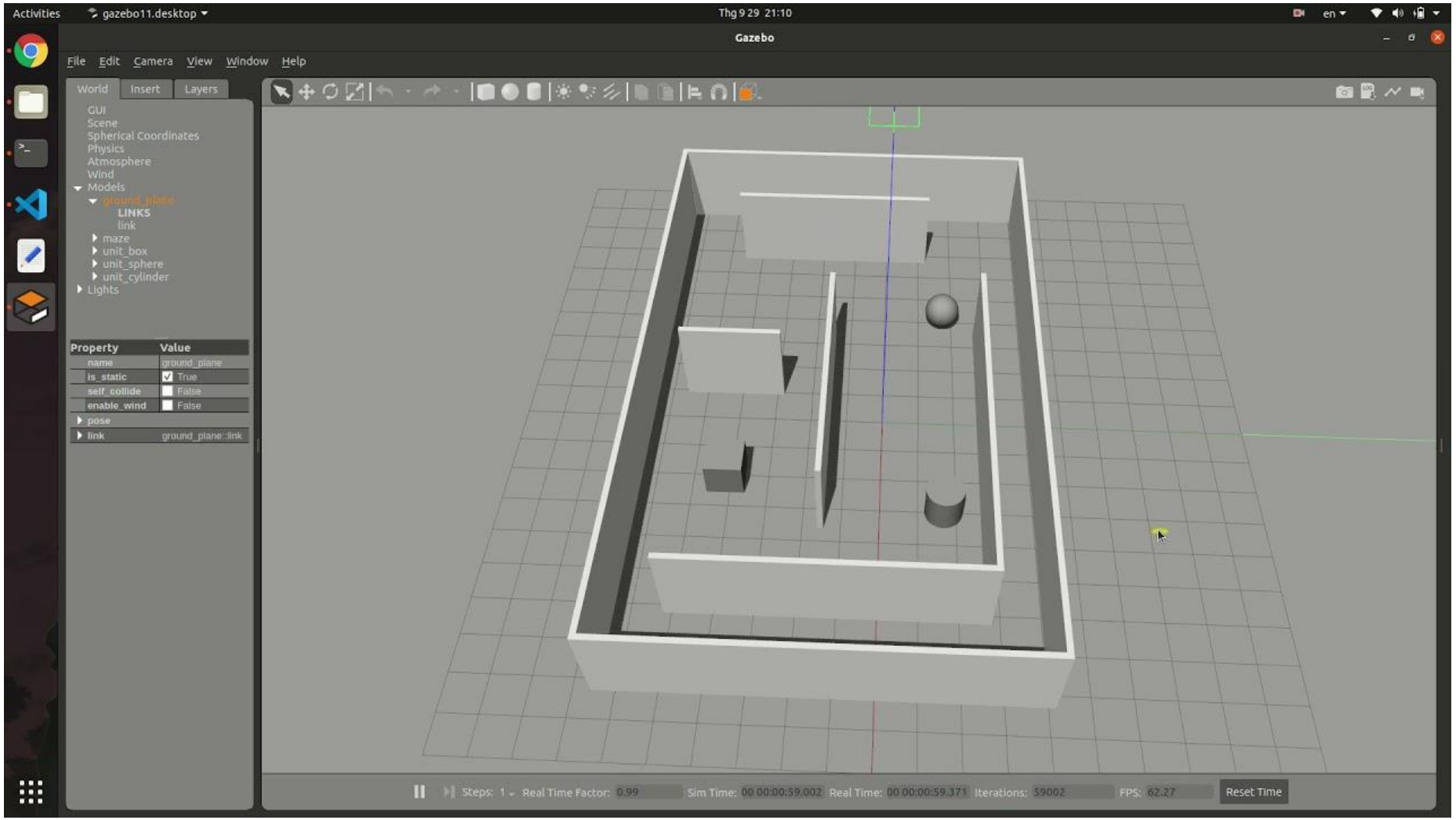


Gazebo

- **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.

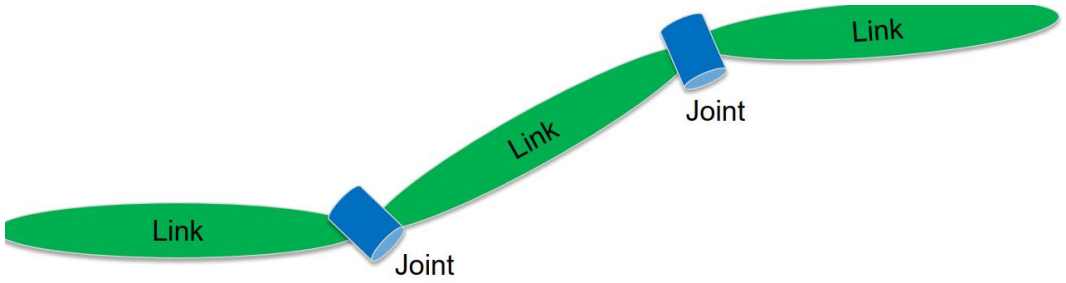


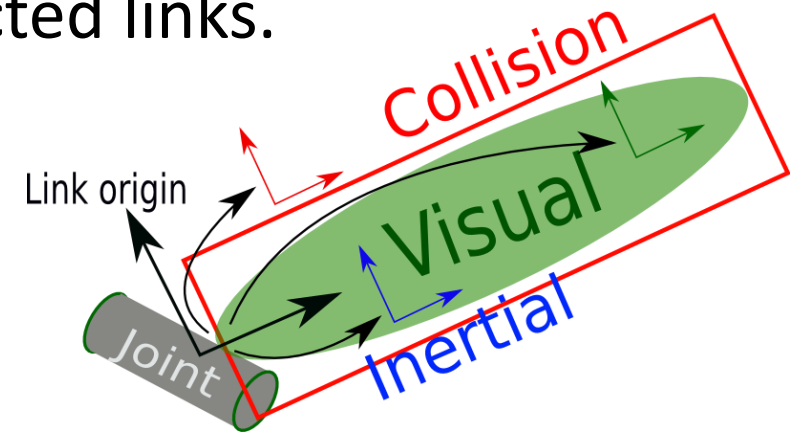
Gazebo


Activities
gazebo11.desktop
Thg 9 29 21:10
en
Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
ground_plane
LINKS
link
maze
unit_box
unit_sphere
unit_cylinder
Lights
Property Value
name ground_plane
is_static True
self_collide False
enable_wind False
pose
link ground_plane::link
Steps: 1
Real Time Factor: 0.99
Sim Time: 00 00:00:59.002
Real Time: 00 00:00:59.371
Iterations: 59002
FPS: 62.27
Reset Time

How Robot made

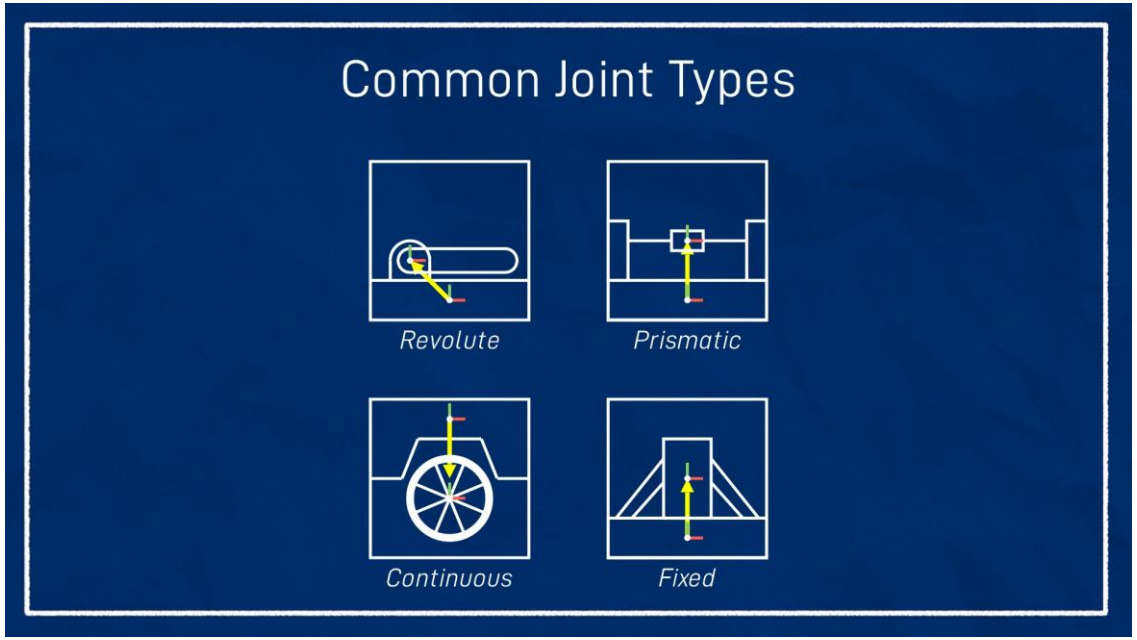
- Most robotic mechanisms may be simplified to a series of links connected by motorized joints.
- Links are approximated as rigid bodies.
- A rigid body is an object whose deformation is assumed to be negligible such that any two points on the body will always be the same distance from one another.
-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



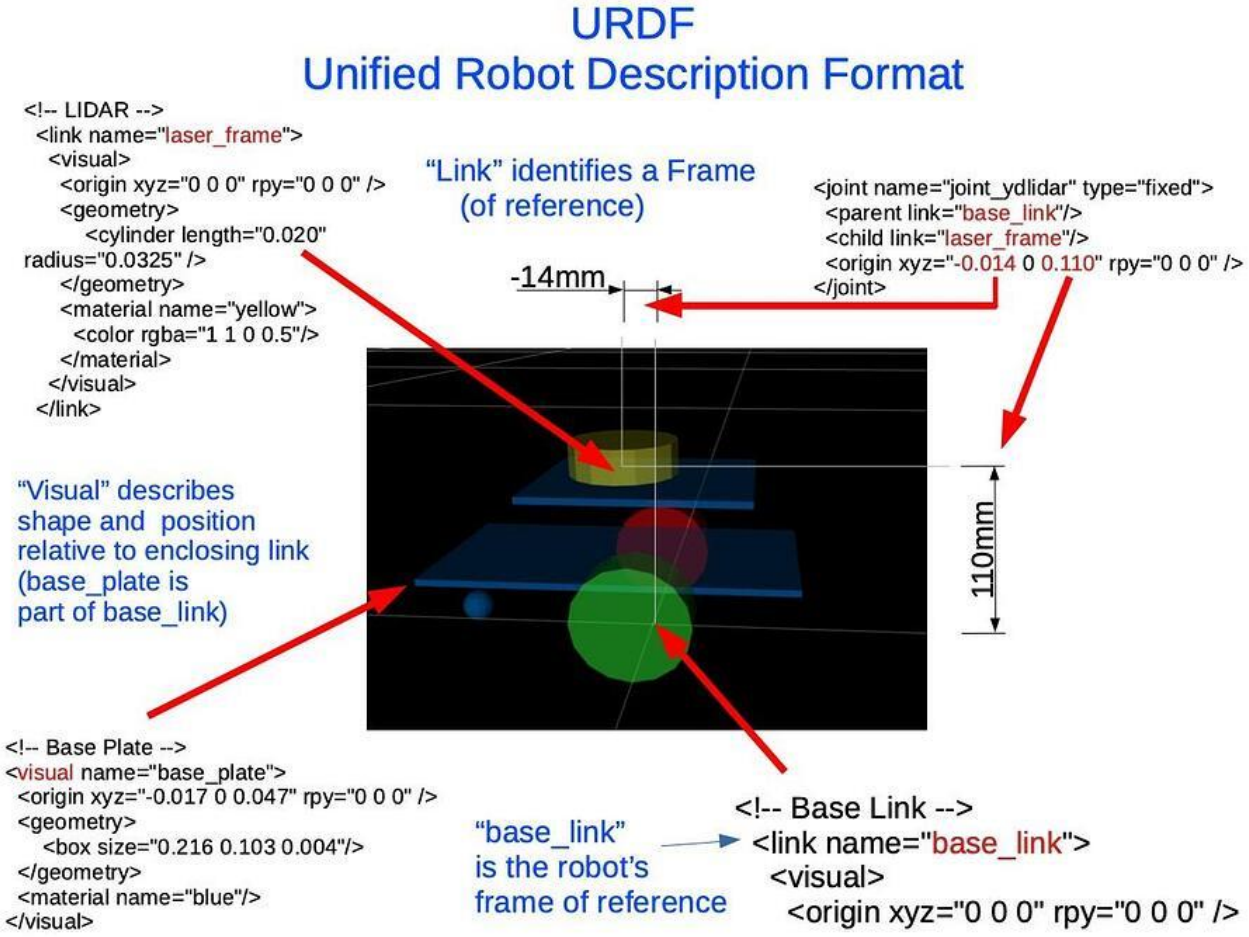
URDF (Unified Robot Description Format)

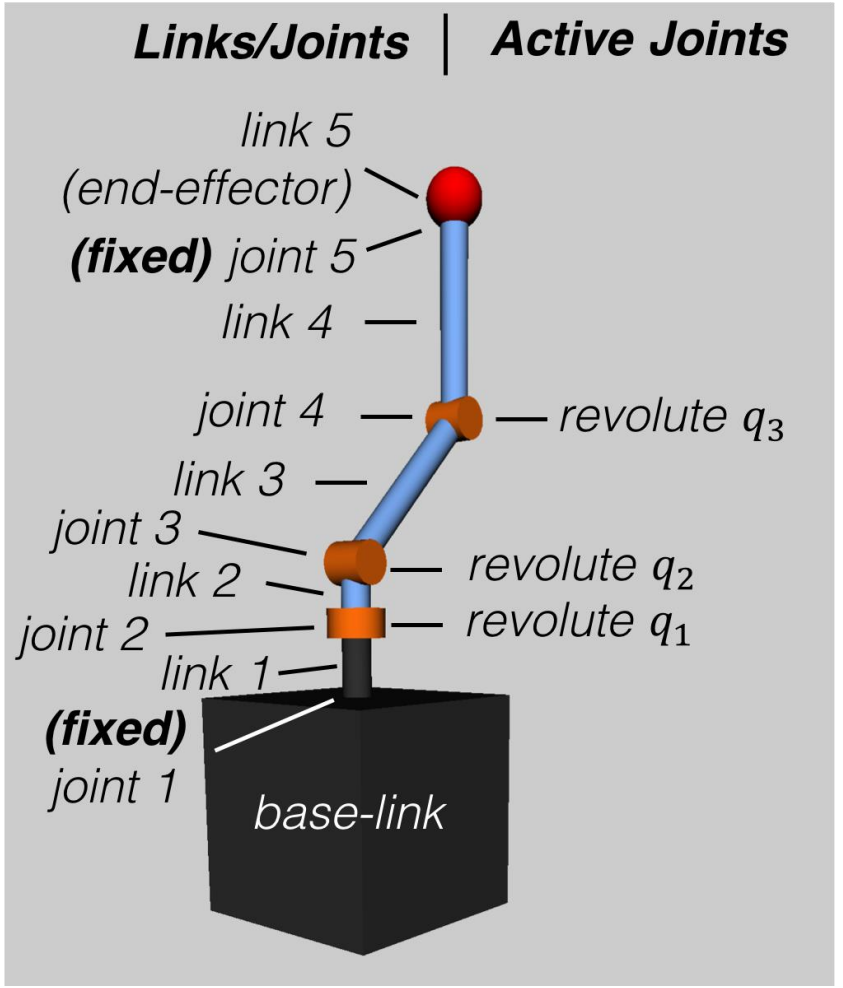
```
<robot>
        <link>
                ...
        </link>
        <link>
                ...
        </link>
        <joint>
                ...
        </joint>
</robot>
```

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

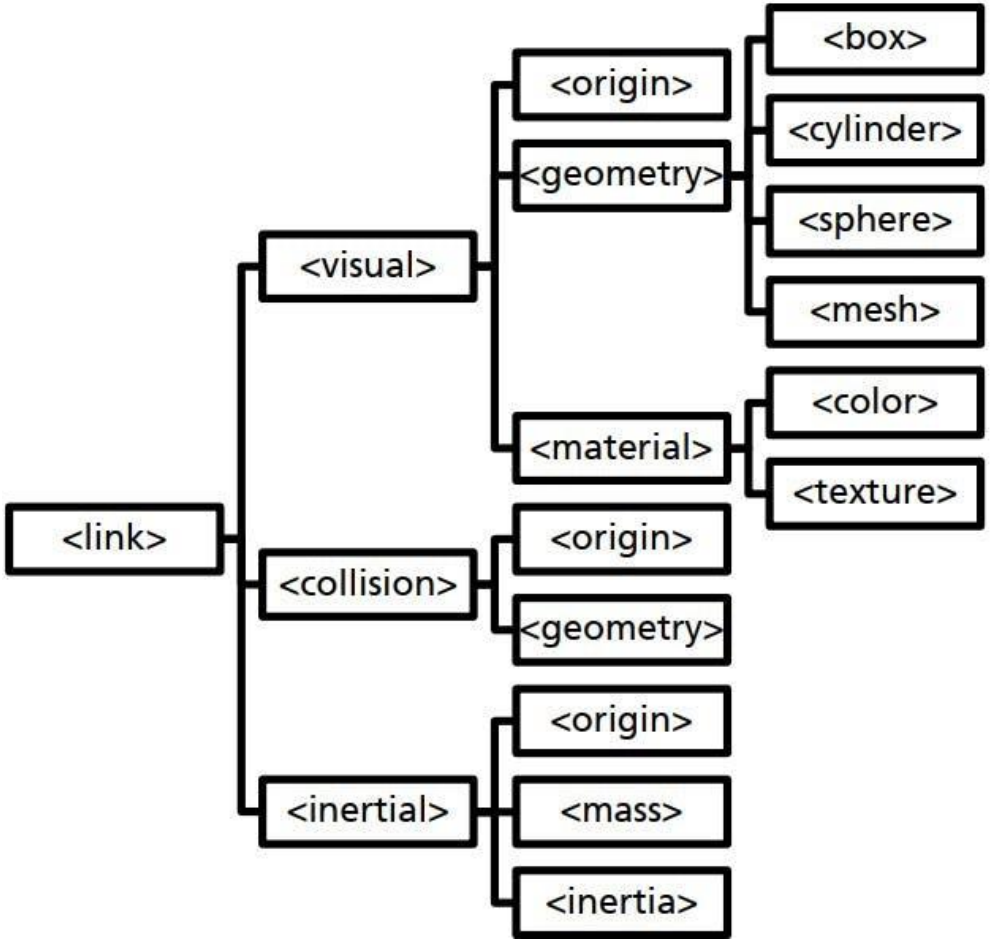
```
<robot name = "linkage">
        <joint name = "joint A ... >
                <parent link = "link A" />
                <child link = "link B" />
        </joint>
        <joint name = "joint B ... >
                <parent link = "link A" />
                <child link = "link C" />
        </joint>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

URDF (Unified Robot Description Format)

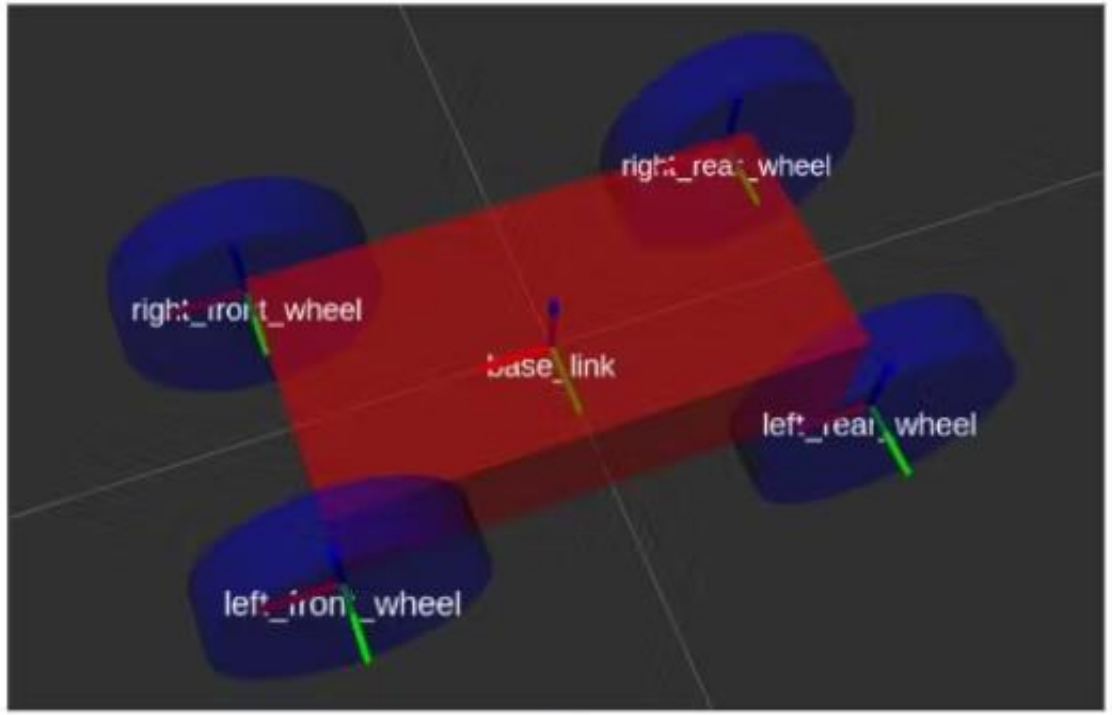




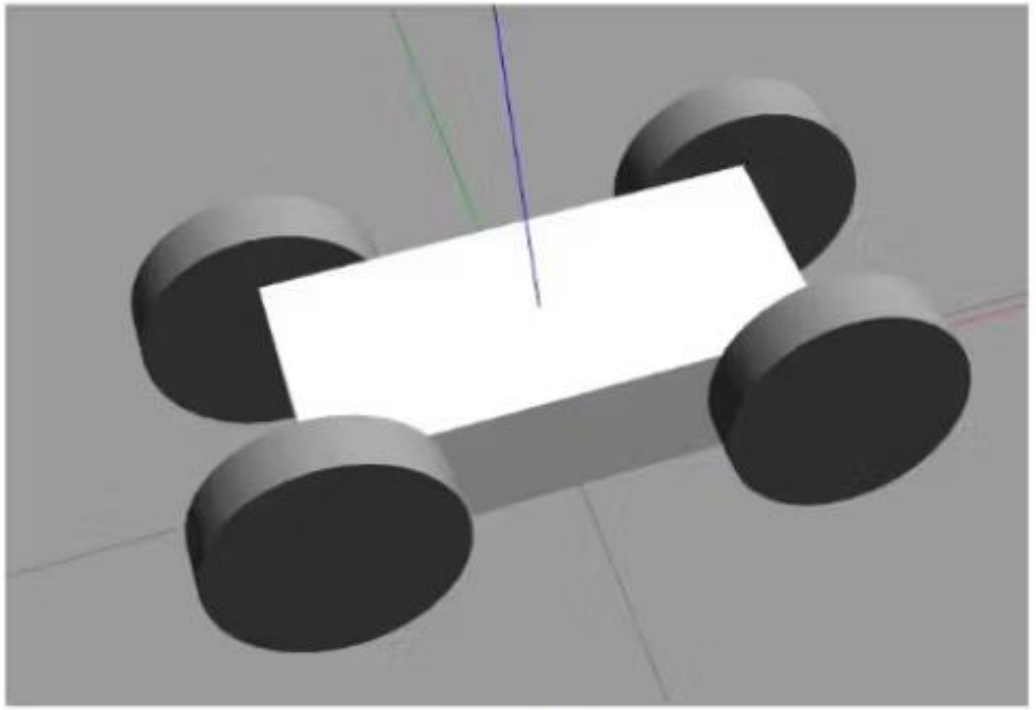
URDF (Unified Robot Description Format)



URDF (Unified Robot Description Format)

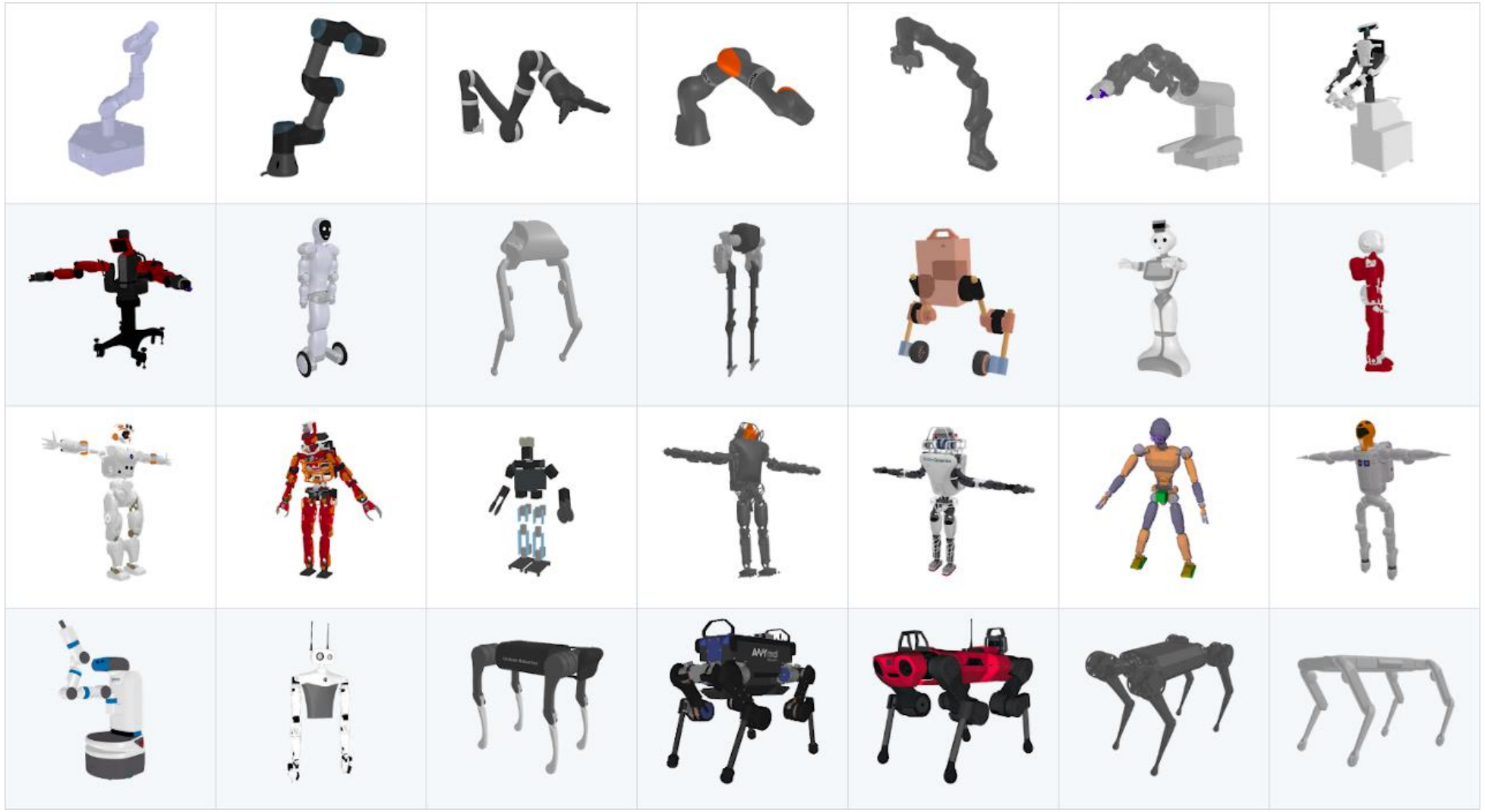


rviz



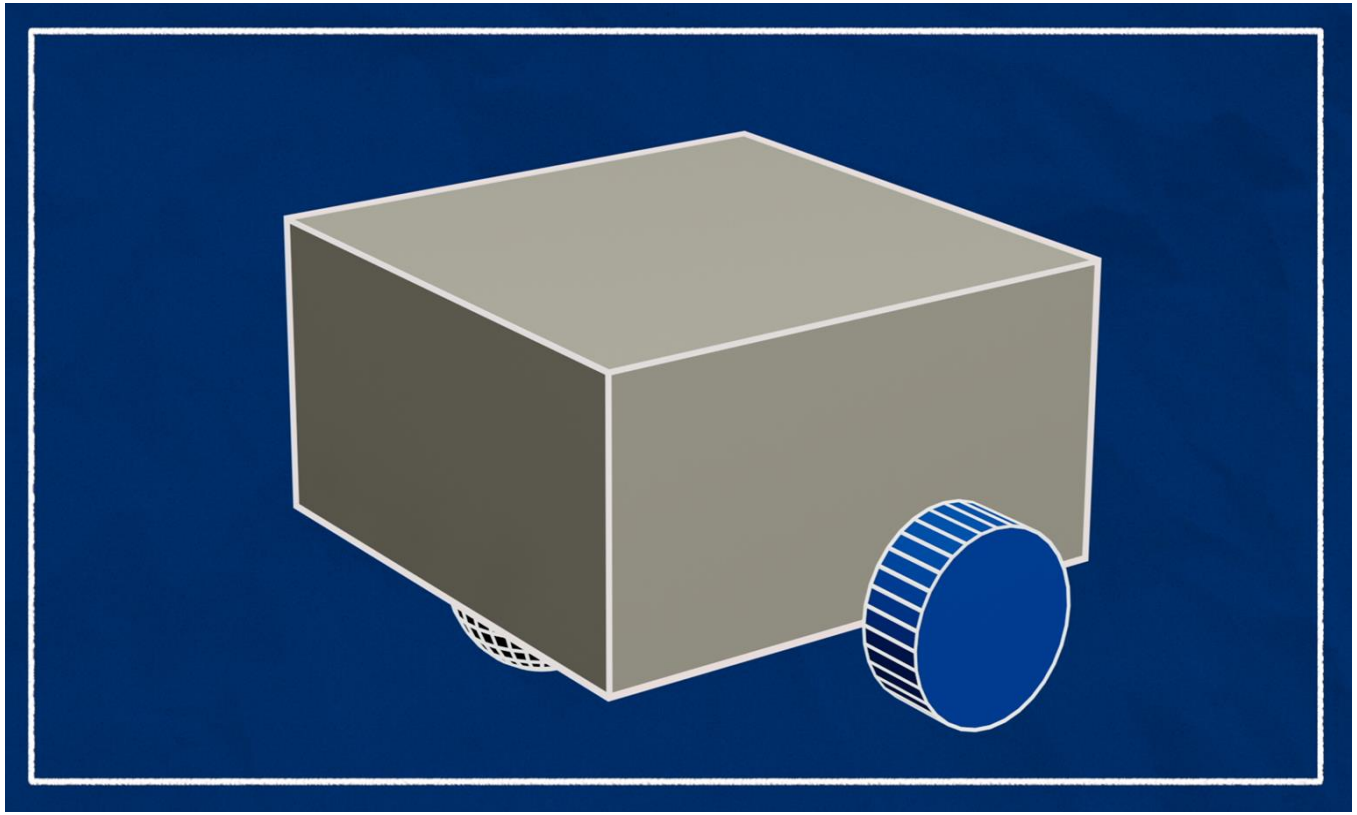
Gazebo

URDF (Unified Robot Description Format)



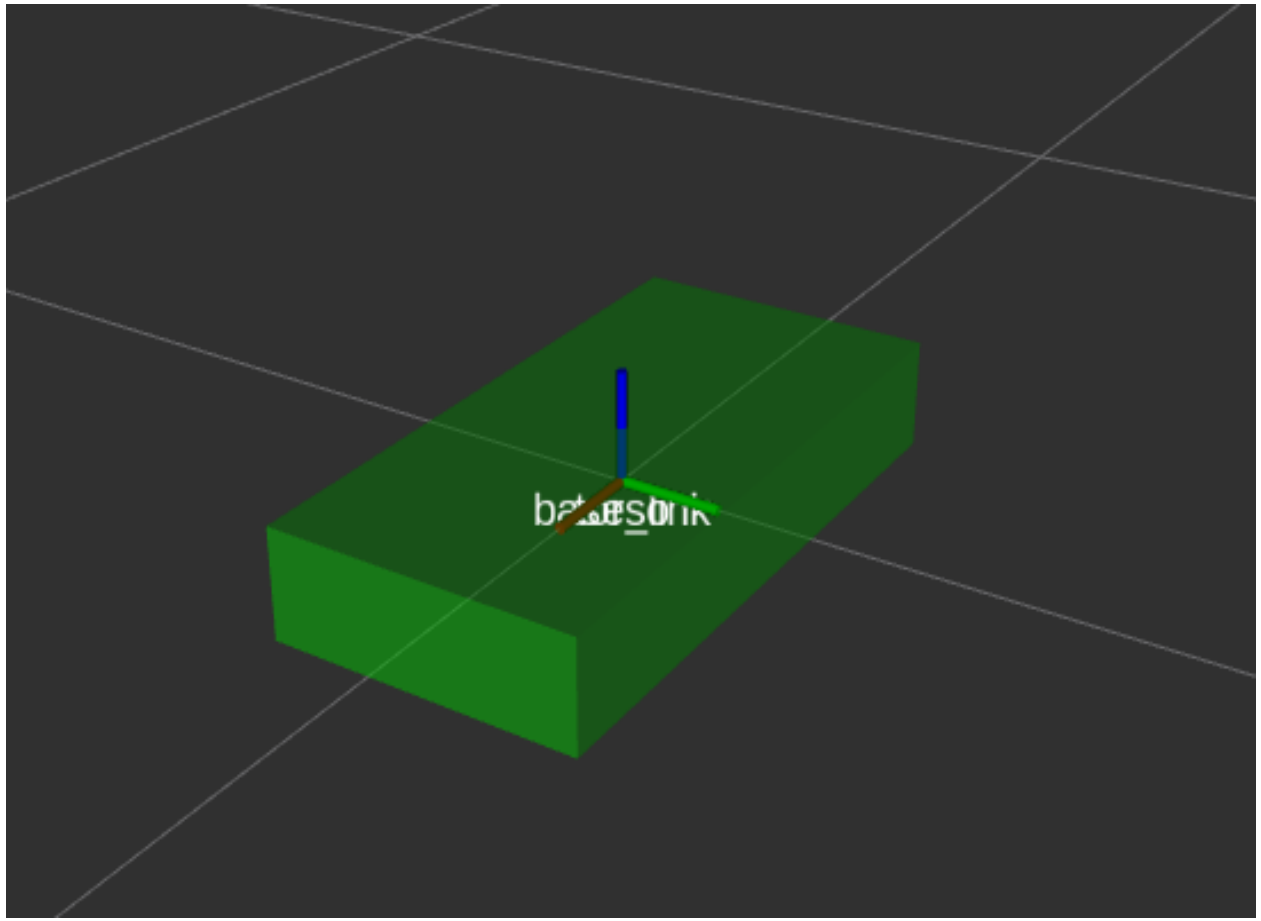
Differential Drive Robot

- Th 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 direction (these are called caster wheels).



Differential Drive Robot

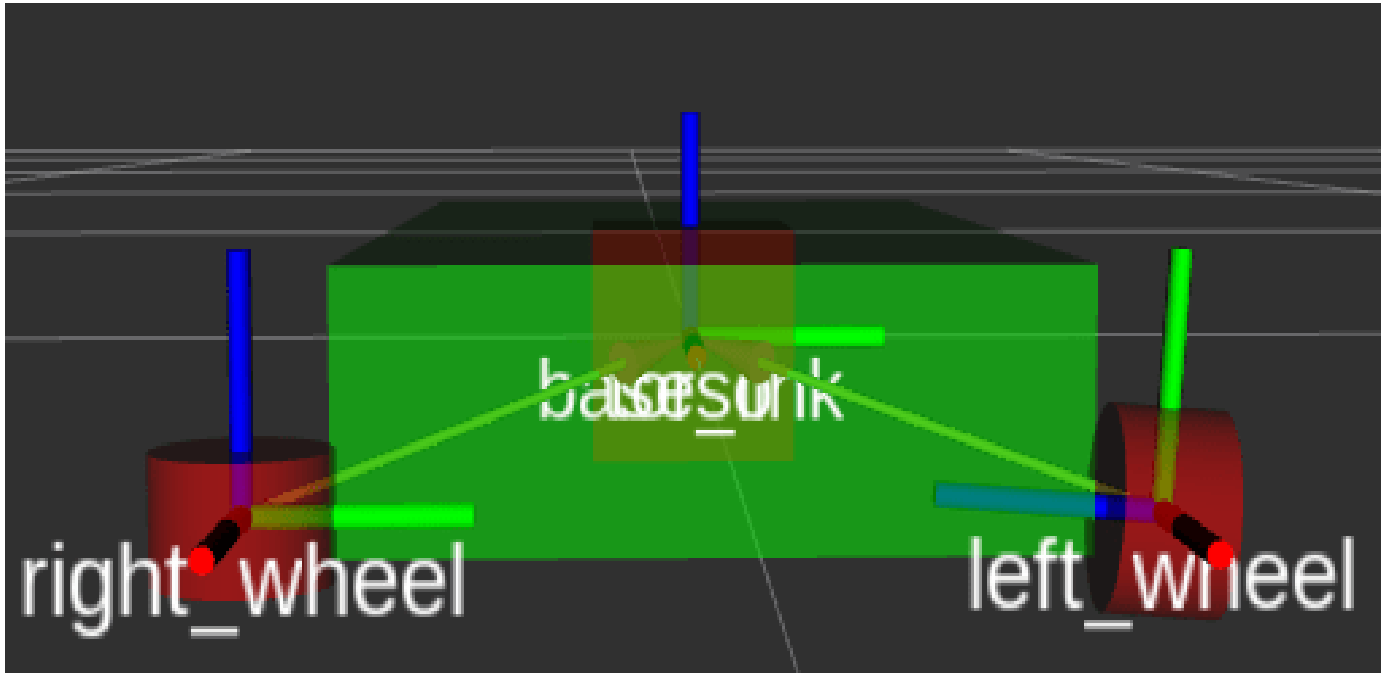
```
<?xml version="1.0"?>
<robot name="bot">
  <link name="base_link">
    <visual>
      <geometry>
        <box size="0.6 0.3 0.1"/>
      </geometry>
      <material name="green">
        <color rgba="1.0 1.0 0.0 1"/>
      </material>
    </visual>
  </link>
</robot>
```



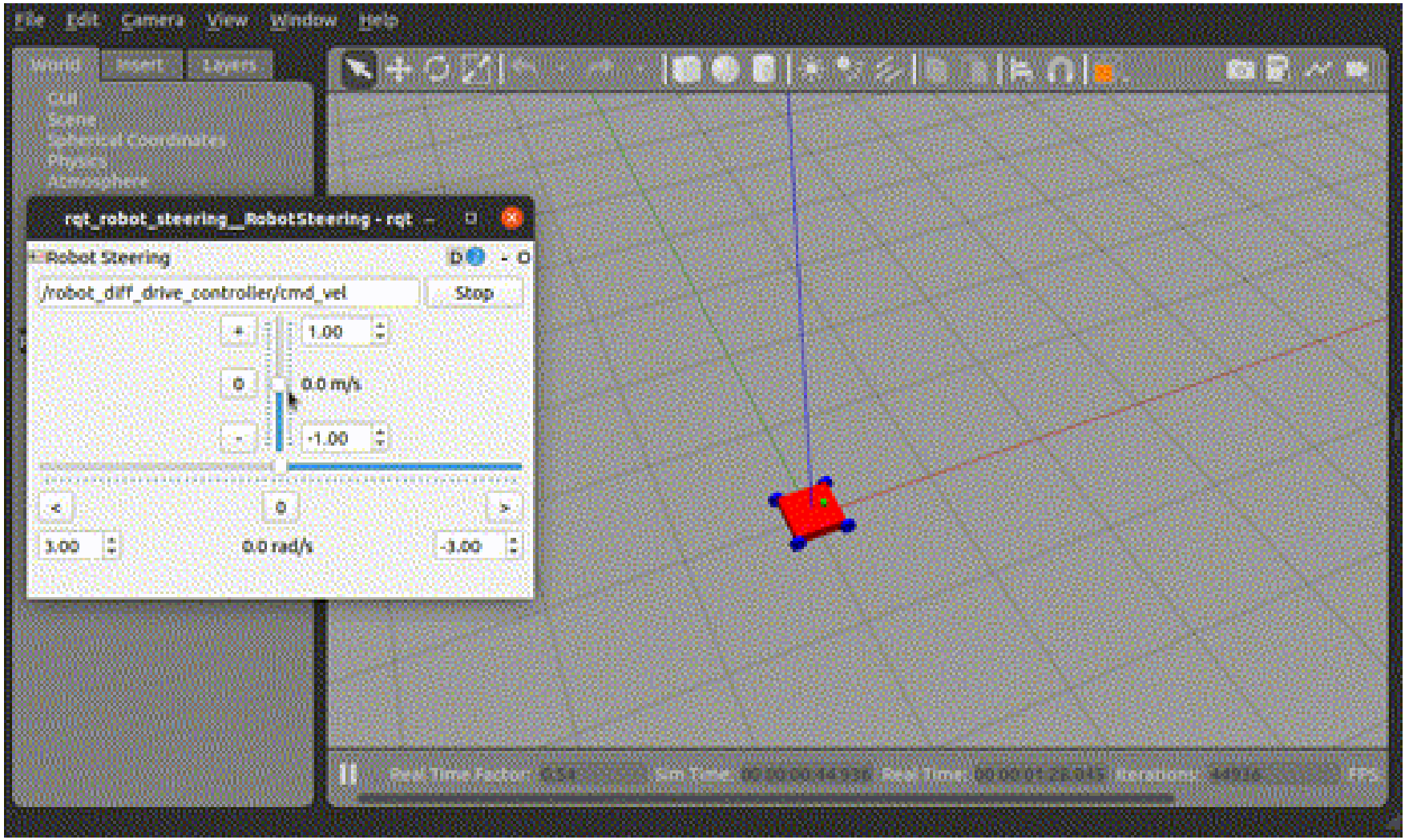
Differential Drive Robot

```
<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>
```

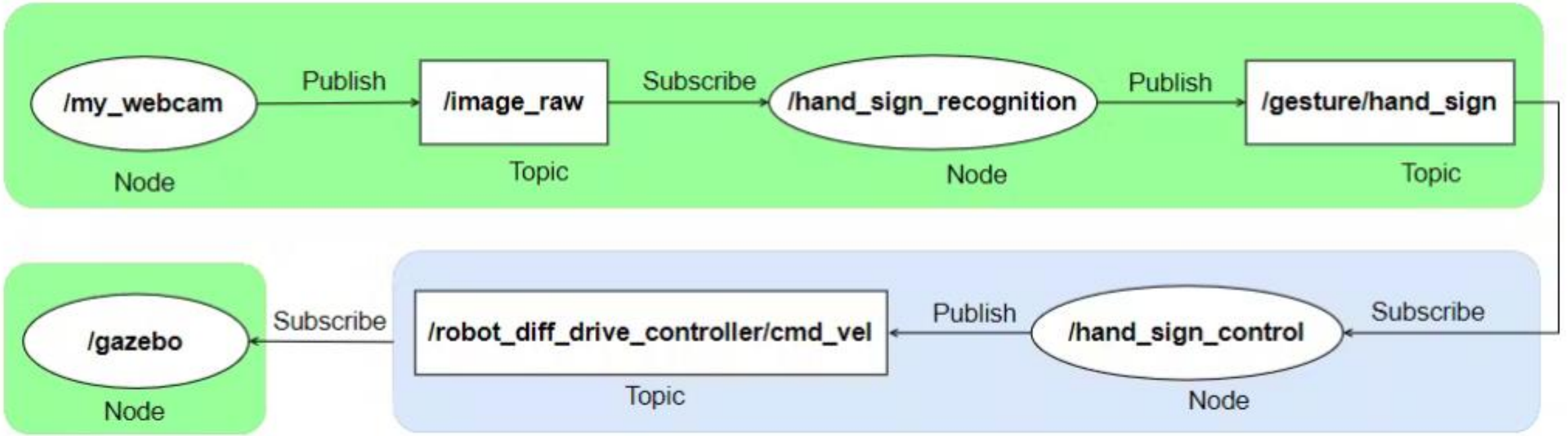
```
<joint name="base_link_right_wheel" type="continuous">
  <parent link="base_link" />
  <child link="right_wheel" />
  <origin xyz="0.2 -0.2 -0.05" rpy="1.570796  0 0"/>
</joint>
```



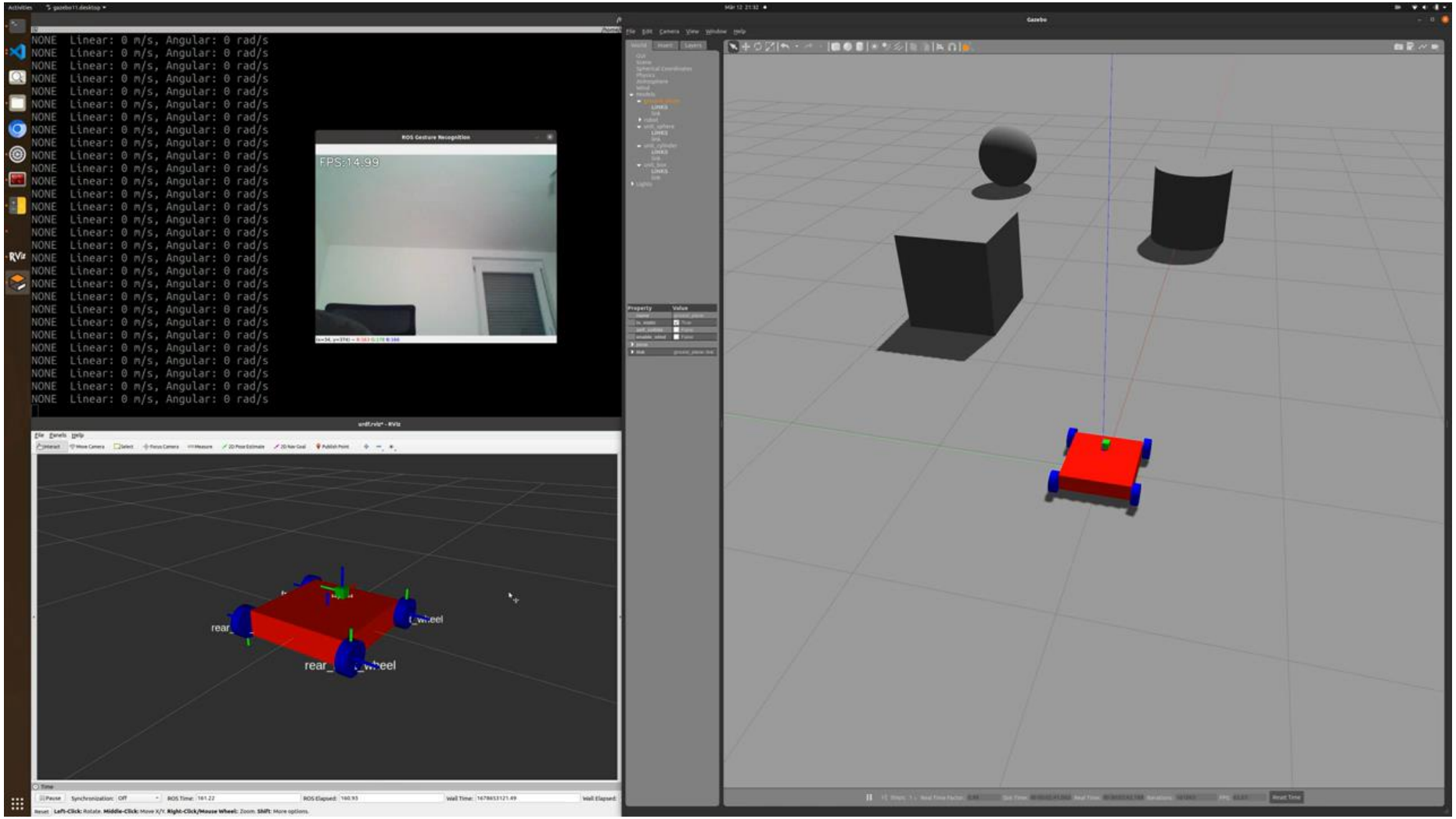
Driving the Robot



Hand Gesture Driving



Hand Gesture Driving



URDF (Unified Robot Description Format)