Programming Tool Induction

A Brief Introduction

- For this year we decided to provide a **homogenous experience** for the students of this master and we decided to use the same software for both COMP0242 and COMP245

A Brief Introduction

- For this year we decided to provide a homogenous experience for the students of this master and we decided to use the same software for both COMP0242 and COMP245
- We designed several **software tools** to provide simplified access to some programming tools that are currently used in robotics both for **control and simulation**

A Brief Introduction

- For this year we decided to provide a homogenous experience for the students of this master and we decided to use the same software for both COMP0242 and COMP245
- We designed several software tools to provide simplified access to some programming tools that are currently used in robotics both for control and simulation
- We have redesigned the **ROS2** interface to a **real robotic manipulator** that is used for didactical purposes

A Brief Introduction

- For this year we decided to provide a homogenous experience for the students of this master and we decided to use the same software for both COMP0242 and COMP245
- We designed several software tools to provide simplified access to some programming tools that are currently used in robotics both for control and simulation
- We have redesigned the ROS2 interface to a real robotic manipulator that is used for didactical purposes
- The ROS2 client is **integrated** with the **control and simulation** that we have developed

A Brief Introduction

- For this year we decided to provide a homogenous experience for the students of this master and we decided to use the same software for both COMP0242 and COMP245
- We designed several software tools to provide simplified access to some programming tools that are currently used in robotics both for control and simulation
- We have redesigned the ROS2 interface to a real robotic manipulator that is used for didactical purposes
- The ROS2 client is integrated with the control and simulation that we have developed
- This software will be used for the **lab sessions** both in **simulation** and (eventually) ono the **real robot**

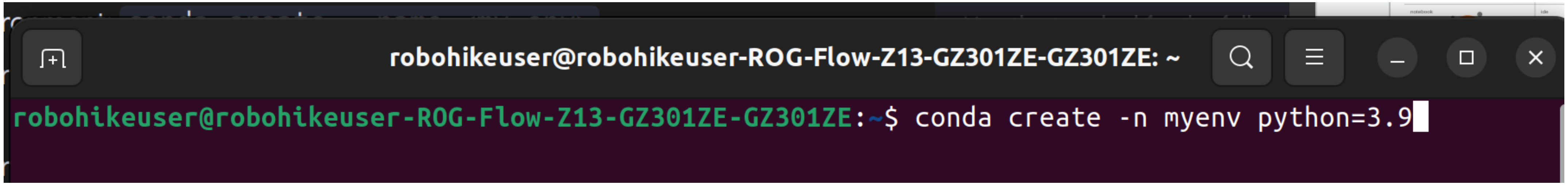
Programming Environment

Anaconda/Mamba Package Manager

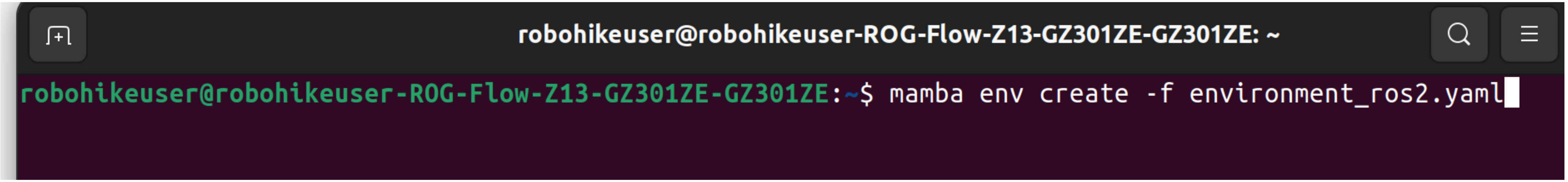
- What is Anaconda? from wikipedia
  "Conda is an open source cross-platform, language-agnostic package manager and environment management system that installs, runs, and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language (e.g., R), including multi-language projects"

Anaconda Basics: Creating environment

- Creating new environments in anaconda/mamba
- From scratch



- From file



Conda/Mamba Environments: Isolation for Project Dependencies

- **What is an Environment?**
  - A self-contained directory that holds a specific collection of libraries and versions.
- **Why Use Environments?**
  - **Isolation**: Ensures that different projects can have their own dependencies, without interference.
  - **Consistency**: Reproduces and shares project setups easily, enhancing compatibility and collaboration.
  - **Flexibility**: Easily switch between project settings without affecting global installations or other projects.

Installing Mamba miniforge

- If you did not installed Mamb please do that now!

- https://github.com/conda-forge/miniforge

Anaconda Basics: Managing Envs

- Once your environment is created you can activate it by typing: **conda activate myenv**
- once you have activated your new environment you can easily manage packages:
- **conda install scipy** → install package from conda
- **pip install package_name** → install package from pip
- **conda remove scipy** → remove package
- you can delete an environment, first activate the env and then type:
- **conda env remove --name environment_name**

Anaconda Cheat Sheet (1)

Conda basics	
Verify conda is installed, check version number	`conda info`
Update conda to the current version	`conda update conda`
Install a package included in Anaconda	`conda install PACKAGENAME`
Run a package after install, example Spyder*	`spyder`
Update any installed program	`conda update PACKAGENAME`
Command line help	`COMMANDNAME --help` `conda install --help`

*Must be installed and have a deployable command, usually PACKAGENAME

Using environments	
Create a new environment named py35, install Python 3.5	`conda create --name py35 python=3.5`
Activate the new environment to use it	`WINDOWS: activate py35` `LINUX, macOS: source activate py35`
Get a list of all my environments, active environment is shown with *	`conda env list`
Make exact copy of an environment	`conda create --clone py35 --name py35-2`
List all packages and versions installed in active environment	`conda list`
List the history of each change to the current environment	`conda list --revisions`

Anaconda Cheat Sheet (2)

Restore environment to a previous revision	`conda install --revision 2`
Save environment to a text file	`conda list --explicit > bio-env.txt`
Delete an environment and everything in it	`conda env remove --name bio-env`
Deactivate the current environment	`WINDOWS: deactivate` `macOS, LINUX: source deactivate`
Create environment from a text file	`conda env create --file bio-env.txt`
Stack commands: create a new environment, name it bio-env and install the biopython package	`conda create --name bio-env biopython`
Finding conda packages	
Use conda to search for a package	`conda search PACKAGENAME`
See list of all packages in Anaconda	https://docs.anaconda.com/anaconda/packages/pkg-docs

Anaconda Cheat Sheet (3)

Installing and updating packages	
Install a new package (Jupyter Notebook) in the active environment	`conda install jupyter`
Run an installed package (Jupyter Notebook)	`jupyter-notebook`
Install a new package (toolz) in a different environment (bio-env)	`conda install --name bio-env toolz`
Update a package in the current environment	`conda update scikit-learn`
Install a package (boltons) from a specific channel (conda-forge)	`conda install --channel conda-forge boltons`
Install a package directly from PyPI into the current active environment using pip	`pip install boltons`
Remove one or more packages (toolz, boltons) from a specific environment (bio-env)	`conda remove --name bio-env toolz boltons`
Managing multiple versions of Python	
Install different version of Python in a new environment named py34	`conda create --name py34 python=3.4`
Switch to the new environment that has a different version of Python	Windows: `activate py34` Linux, macOS: `source activate py34`

Anaconda Cheat Sheet (4)

Show the locations of all versions of Python that are currently in the path **NOTE**: The first version of Python in the list will be executed.	`Windows: where python` `Linux, macOS: which -a python`
Show version information for the current active Python	`python --version`

Specifying version numbers

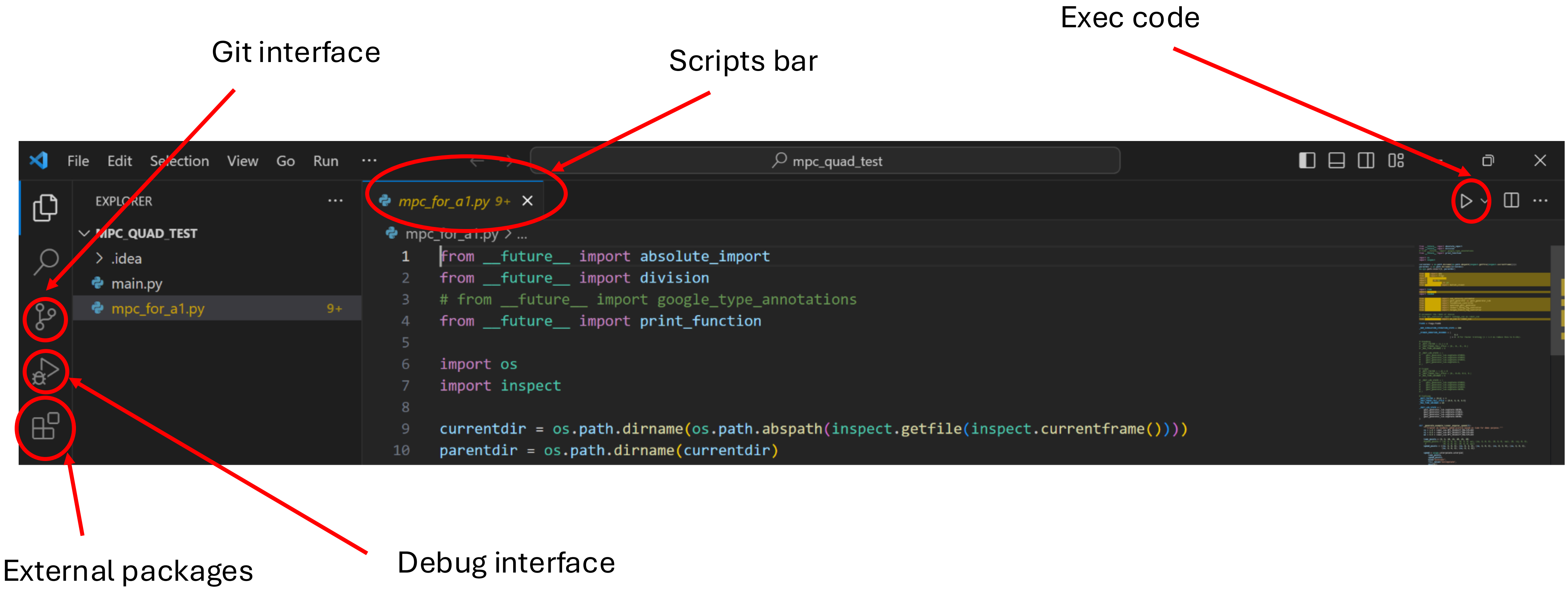
Ways to specify a package version number for use with conda create or conda install commands, and in meta.yaml files.

<table>
<tr><th>Constraint type</th><th>Specification</th><th>Result</th></tr>
<tr><td>Fuzzy</td><td><code>numpy=1.11</code></td><td>1.11.0, 1.11.1, 1.11.2, 1.11.18 etc.</td></tr>
<tr><td>Exact</td><td><code>numpy==1.11</code></td><td>1.11.0</td></tr>
<tr><td>Greater than or equal to</td><td><code>"numpy>=1.11"</code></td><td>1.11.0 or higher</td></tr>
<tr><td>OR</td><td><code>"numpy=1.11.1|1.11.3"</code></td><td>1.11.1, 1.11.3</td></tr>
<tr><td>AND</td><td><code>"numpy>=1.8,<2"</code></td><td>1.8, 1.9, not 2.0</td></tr>
</table>

NOTE: Quotation marks must be used when your specification contains a space or any of these characters: > < | *

The IDE: Visual Studio Code

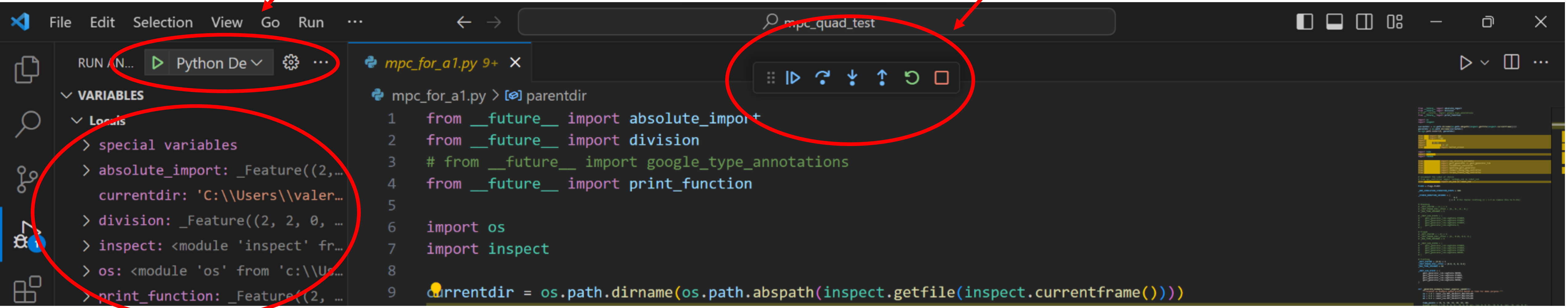
- Visual Studio code is a free editor developed by Microsoft.



Visual Studio Code: The Debugger

Run debugger

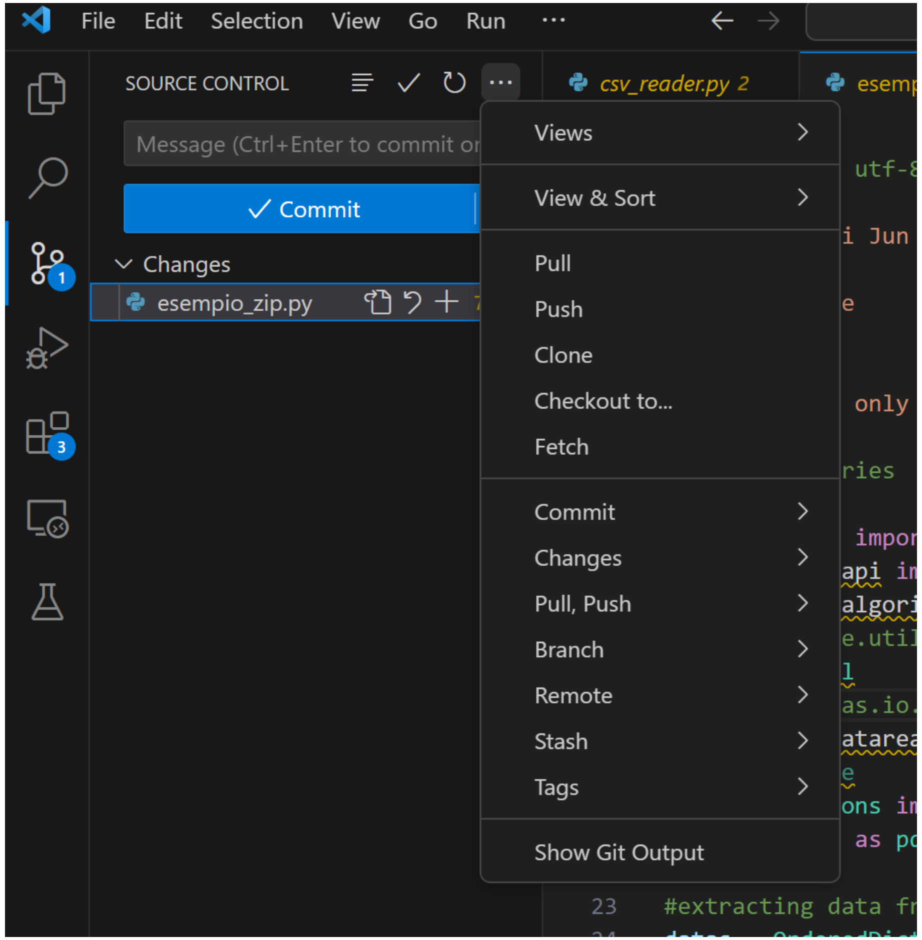
Debugging commands



Workspace, once you place a checkpoint

Visual Studio Code: the Git interface

- in this screenshot on the left you can briefly check out the Git interface offered by by VSC you can perform all the classical action provided by Git such as:
- commit
- push
- pull ...



But… what is Git? (1)

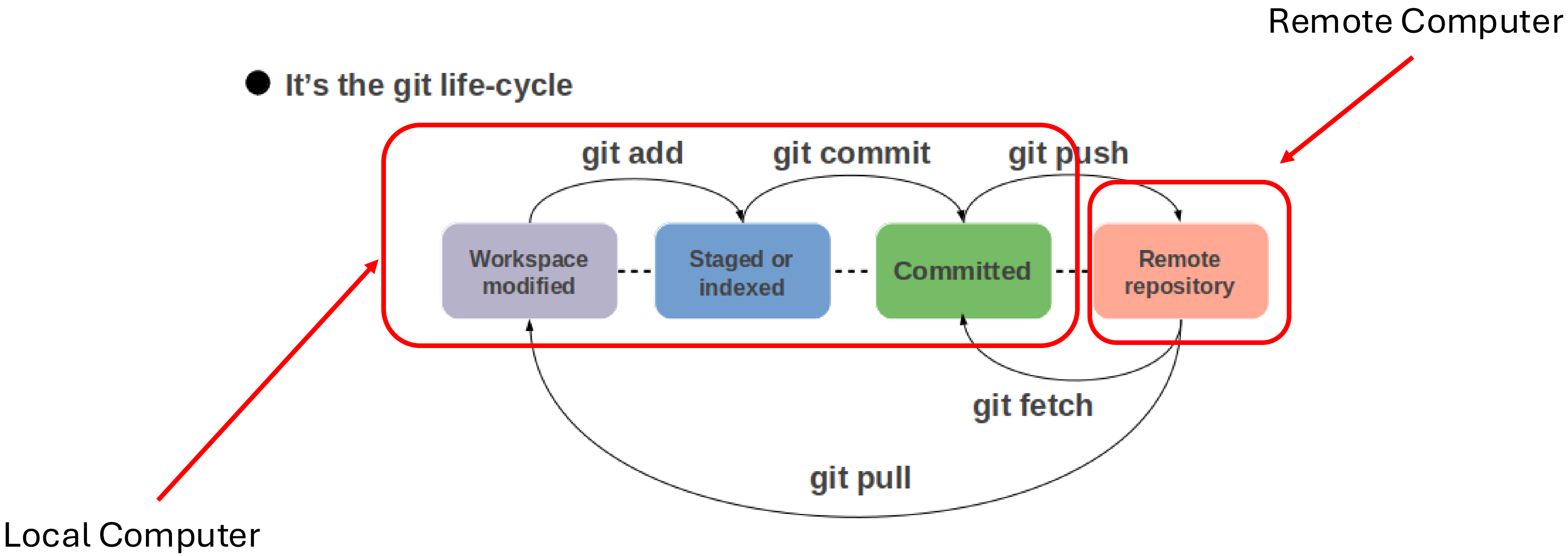
- Probably now many of you are asking yourself: what is Git? What is used for? According to the first Git commit:

> "git" can mean anything, depending on your mood.
>
> - random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronounciation of "get" may or may not be relevant.
> - stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.
> - "global information tracker": you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.
> - "goddamn idiotic truckload of sh*t": when it breaks

… What is Git? (2)

- Git is a **distributed version control system** that **tracks changes** in any set of computer files, usually used for coordinating work among programmers
- .In practice Git works **by taking "snapshots" of your files**, allowing you to track who changed what and when.
- Git makes it easy to collaborate without conflicting and allows for
- experiment **with new ideas in branches**
- **revert back** to older versions if mistakes are made.
- every code project under Git version control is called “**Repository**”

Git Life Cycle



Remote Git Hosting Service

- There exist numerous services online that offer a Git service for free. You can create one or more repositories on each of these services

Github

Bitbucket

Gitlab

ROS

- **What is ROS?**
- An open-source, flexible framework for writing robot software.
- Provides tools, libraries, and conventions to simplify the task of creating complex and robust robot behavior
- Features:
  - **Modularity**: Software in ROS is organized in packages and nodes for easy reusability and testing.
  - **Tools**: Includes powerful utilities for building, debugging, and visualizing robot applications.
  - **Ecosystem**: Large community contributions of tools, algorithms, and drivers that support rapid development and integration.
- **Why ROS?**
  - Facilitates **code reuse** in robotics research and development.
  - ROS is ideal for **educational purposes** and **industrial integration**, bridging the gap between research and practical applications.

ROS 2 Humble

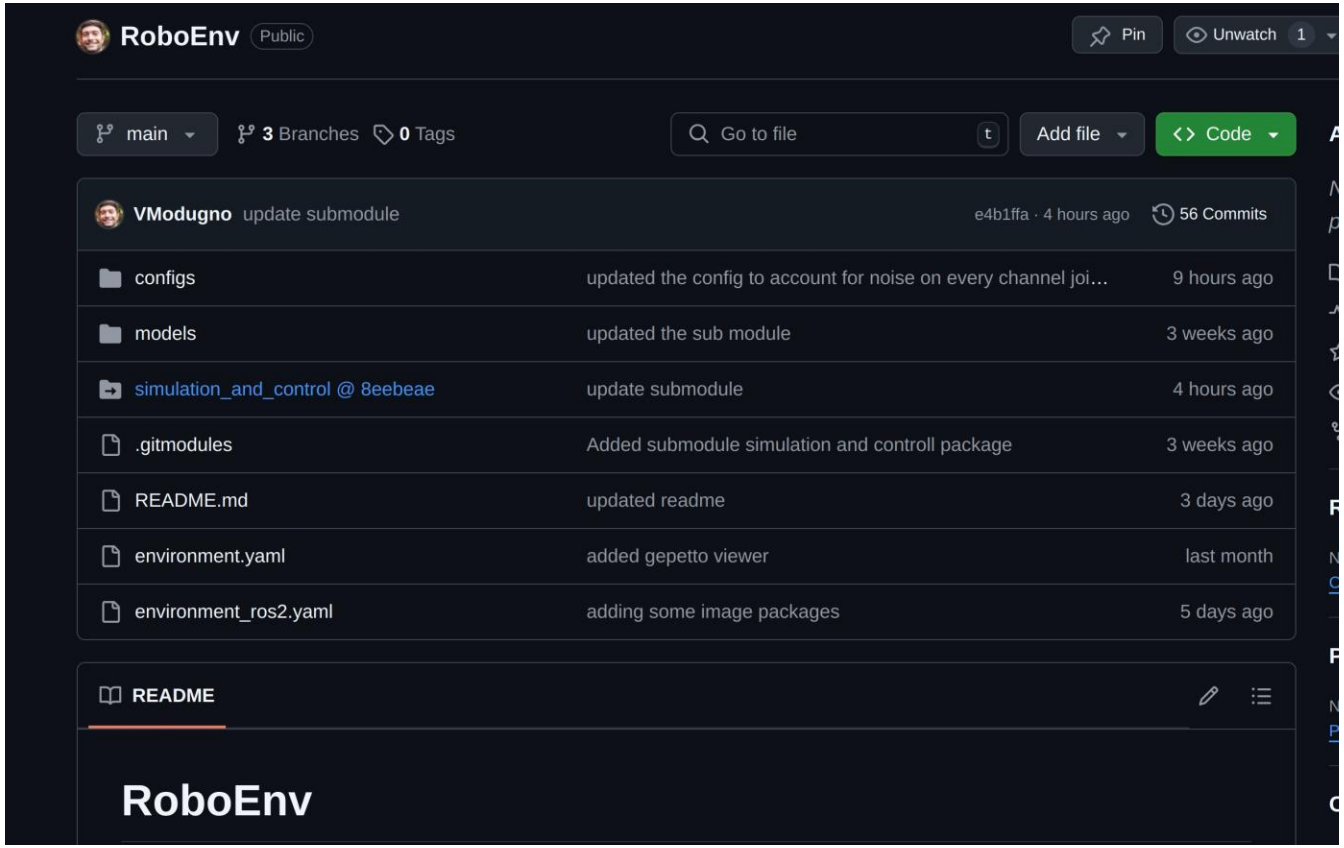
- In this course we will use ROS2 Humble

- We will delve more into it during one of the next Thursday during the innovation lab session

The Library for Sim and Control

RoboEnv

- The repository is here https://github.com/VModugno/RoboEnv



RoboEnv

- Let's try to download it using git!

- For downloading it follow the instruction on the Readme.md which is in the repo
- Caveat: the library contains a **submodule** so you will not be able to download all of it
- Read the instruction carefully!

RoboEnv

- After downloading it using Git check if there are stuff in the folder simulation_and_control
- If the folder is empty check at the end of the readme. You can still download the submodule!

RoboEnv: Creating the Mamba env

- If the download has been successful we can proceed by installing the mamba/conda environment.
- **If you did not do it before do it now!**
  ensure that you have mamba installed. If you do not have mamba installed go to https://github.com/conda-forge/miniforge
  and install the version which is compatible with your operating system
- After this follow the instructions to install the environment using the **environment_ros2.yaml** file and activate the newly created environment

RoboEnv: Installing simulation_and_control

- Simulation and control is a python package.
- Ensure that roboenv2 is activated!
- You should see something like this op your terminal

```
(base) robohikeuser@robohikeuser-ROG-Flow-Z13-GZ301ZE-GZ301ZE:~/git_teaching/RoboEnv$ conda activate roboenv2
(roboenv2) robohikeuser@robohikeuser-ROG-Flow-Z13-GZ301ZE-GZ301ZE:~/git_teaching/RoboEnv$
```

- If you are in the right environment and you are in the roboenv folder you can install the library inside roboenv2 python environment if you type in your terminal
- **pip install .**

Congratulations!

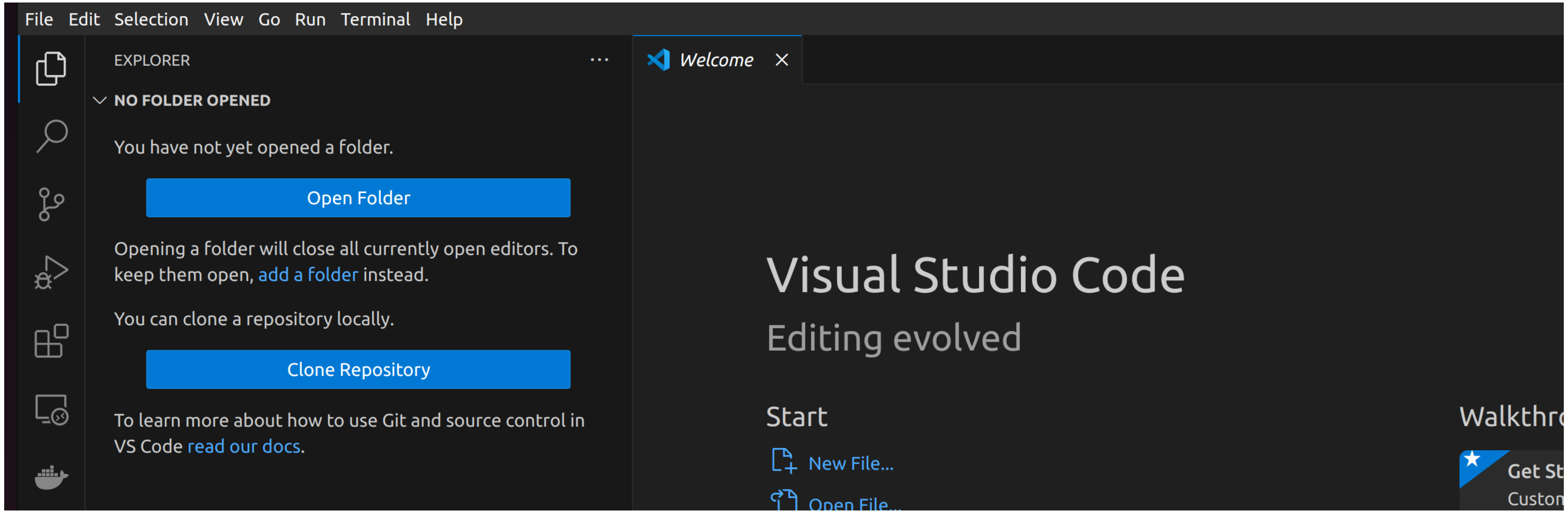
You have successfully installed all the roboenv dependencies

Testing if Everything Works

- Now to test if everything works you can go to:
- https://github.com/VModugno/lab_sessions_COMP0245_PUBLIC
- Download this repository by typing in your terminal
- **Git clone https://github.com/VModugno/lab_sessions_COMP0245_PUBLIC**
- **This apply for all the courses that use roboenv (the induction is always the same)**

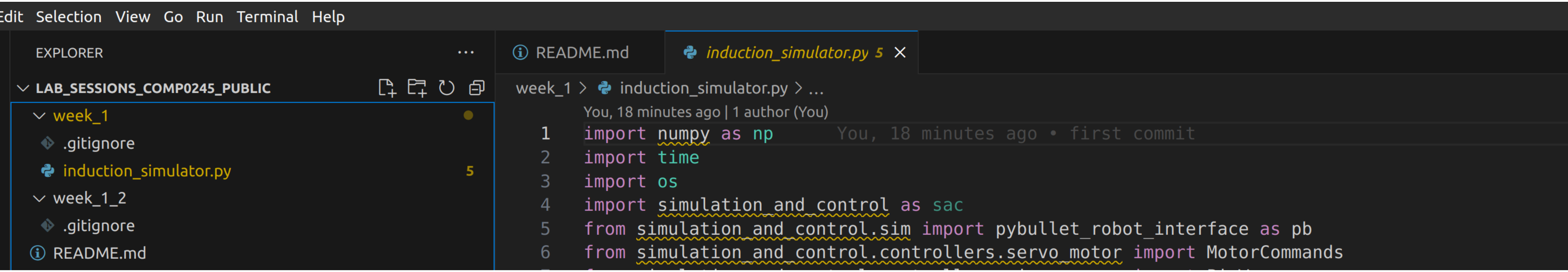
Create a Project with your IDE

- In this class I will use VSC as primary IDE but you can use any IDE you feel confident with
- From now on all the instructions are for visual studio code
- Open VSC and click on Open Folder



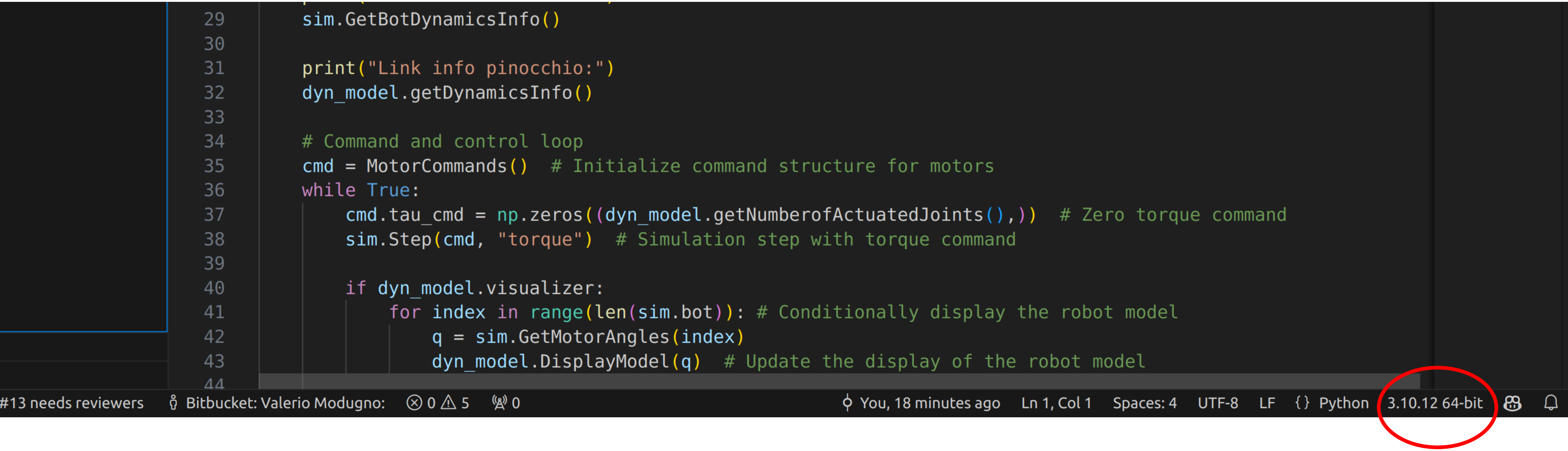
Create a Project with your IDE

- After clicking open folder you need to find where you have downloaded lab_sessions_COMP0245_PUBLIC
- If you have done it correctly you should see something like this:



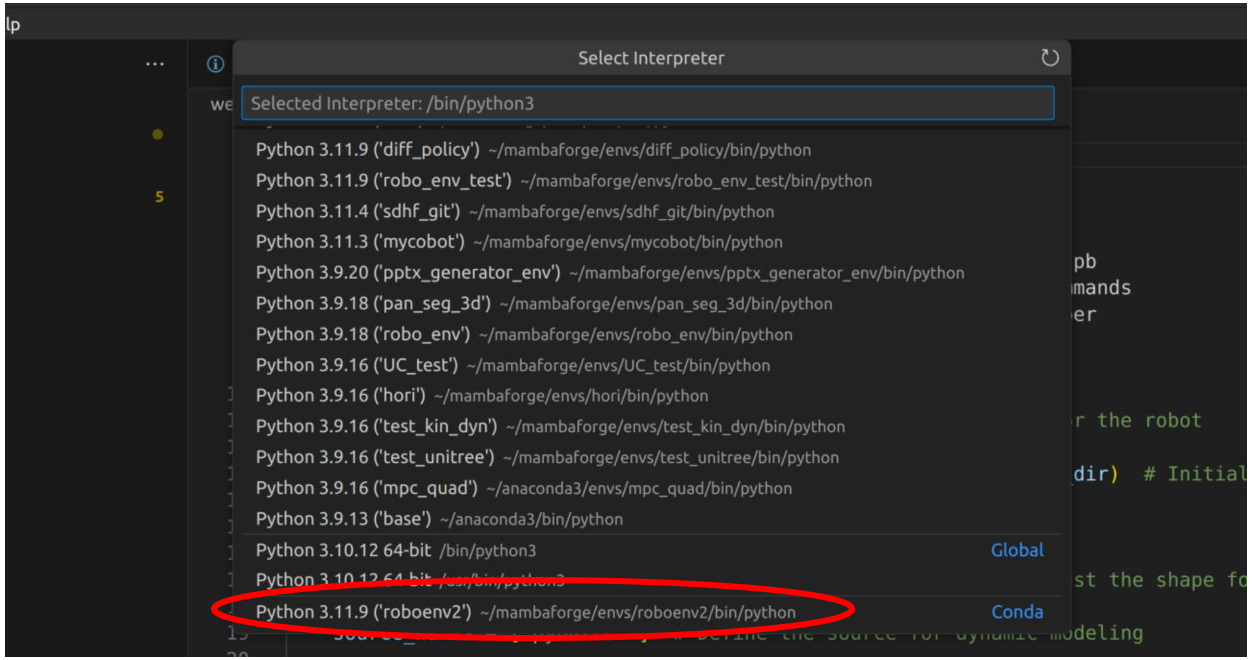
Select the Correct Python Env in VSC (1)

- As you can see some dependancies are not recognized
- To connect VSC to the right python env click in the bottom left corner



Select the Correct Python Env in VSC (2)

- Once you click there a menu like this will appear at the top



- Select the right environment by clicking on it

Execute the Test Code

- Before executing the code copy inside week one the folder called **configs** and **models** which are located in RoboEnv. This contains the URDF of the robots and the configuration file which **are essential** to exec the code
- Now if everything has been done correctly if you execute the code (one of the 2 examples) by clicking on the top left green arrow you should see the simulation start!
- **Congratulations again**! Now you can start to do some real robotic simulation, control, and learning!

Simulation and Control Lib

Simulation and Control Brief

- The library is built upon two major robotics libraries which are commonly used for robotics research:

Pybullet

Pinocchio

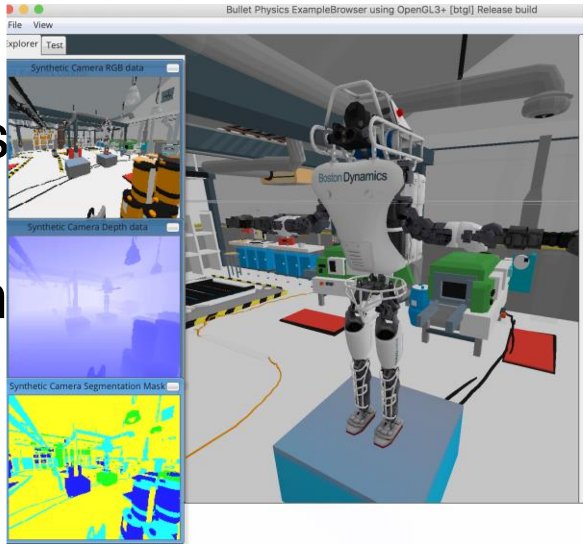
Pinocchio

Efficient and versatile rigid body dynamics algorithms

PyBullet

- PyBullet is an open-source Python library for robotics simulation

- Real-time **physics** simulation for robotics,
- Support for importing **URDF**, **SDF**, and **MJCF** formats
- It comes with many algorithm such as **inverse kinematics**, **path planning**, and **collision detection**
- Integrated support for **deep learning** and **reinforcement learning** algorithms.
- **Cross-platform** compatibility
- widely used in academic and industrial research



Wait a Second, What is an URDF?

- **URDF: Unified Robot Description Format**
- URDF (Unified Robot Description Format) is an XML format used in robotics for defining the **physical and visual properties** of a robot.
- **Modular Design**: URDF allows the representation of a robot's structure in a tree-like hierarchy of **links** and **joints**
- **Visual and Collision Elements**: Each link can have associated visual and collision properties, enabling visualization and contact

An Example of URDF

```xml
<?xml version="1.0"?>
<robot name="origins">
  <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
    </visual>
  </link>

  <link name="right_leg">
    <visual>
      <geometry>
        <box size="0.6 0.1 0.2"/>
      </geometry>
      <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>
    </visual>
  </link>

  <joint name="base_to_right_leg" type="fixed">
    <parent link="base_link"/>
    <child link="right_leg"/>
    <origin xyz="0 -0.22 0.25"/>
  </joint>

</robot>
```

This URDF represent simple object which has 2 links connected with a single fixed joint

Base link

Right leg

Joint

Pinocchio

- Pinocchio is an open-source C++ library designed for efficiently computing the dynamics of articulated rigid-body models,
- **Efficient Dynamics Algorithms**: Offers fast forward and inverse dynamics computations, Lagrangian Dynamics terms
- **Robot Modeling**: Supports loading robot models from standard formats like URDF, SDF, and others.
- **Versatility**: Integrates easily with software, such ROS and it has a Python interface
- **Application Scope**: Ideal for motion planning, simulation, optimization, and **control** tasks, making it a robust tool for robotics researchers and engineers.

How to Use the Library

- Simulation_and_control provide simplified access to PyBullet and Pinocchio.
- The class has to main access point:
- The class **SimInterface**
- The class **PinWrapper**

The SimInterface Class (1)

- The `SimInterface` class is designed to interface with the PyBullet physics simulator, specifically for robotic simulation environments. Here’s a breakdown of its functionality and structure:
- Initialization
  - **Configuration Loading**: On initialization, the class loads a JSON configuration file that contains various simulation parameters including time steps, environment settings, and robot configurations.
  - **Simulator Setup**: It sets up the PyBullet simulator with specific physics engine parameters like solver iterations, time step, and gravity.
  - **Environment and Robot Initialization**: Loads the ground plane, initializes the visual environment, and creates robot instances based on the URDF paths defined in the configuration. It also manages robot state initialization like joint angles and velocities.
- Simulation Control
  - **Simulation Step Control**: It provides methods to advance the simulation by applying motor commands, handling the simulation timestep, and fetching new observations from the simulated robots.
  - **Environment Script Execution**: If specified, executes additional Python scripts for environment customization directly within the simulator’s context.

The SimInterface Class (2)

- Observation Management
  - **Observation Handling**: Manages the history and retrieval of sensory data and robot states, ensuring that data like joint angles, velocities, and robot base states are updated and accessible at each timestep. It also supports handling delayed observations to simulate real-world sensing delays.
- Dynamics Computations
  - **Physics Calculations**: Includes methods for computing dynamics-related quantities like mass matrices, Coriolis forces, and other physics-based simulations directly through PyBullet's APIs.
- Utility Functions
  - **Utility and Debugging**: Provides numerous utilities for robot control and debugging, such as setting joint positions, computing transformations between coordinate frames, and extracting or setting physical properties like link masses or motor torques.

PinWrapper Class (1)

- The `PinWrapper` class is designed for interfacing with the **Pinocchio library**. The class is initialized with various parameters to set up the environment and configuration for a robot simulation.
- **Initialization Parameters**:
  - `conf_file_name`: Specifies the configuration file for setting up the robot model.
  - `simulator`: Specifies the simulation environment to use.
  - `list_link_name_for_reodering`: An optional numpy array specifying the order of link names for joint state conversion.
  - `data_source_names`: List of names corresponding to data sources for joint state conversion.
  - `visualizer`: A boolean flag to enable or disable visualization.
  - `index`: Index used to specify configurations when multiple setups are defined in a configuration file.
  - `conf_file_path_ext`: Optional path extension for configuration files.

PinWrapper Class (2)

- **File and Configuration Management**:
  - Constructs paths for configuration and URDF (Unified Robot Description Format) files based on provided parameters.
  - Loads configuration settings from a JSON file.
- **Robot Model Setup**:
  - Depending on the robot's base type (fixed, floating, or on-rack), the appropriate URDF model is loaded and the robot model is built using the Pinocchio library. The robot dynamics (e.g., joint and velocity dimensions) are set up accordingly.

PinWrapper Class (3)

- **Dynamic Computations**:
  - Provides methods to compute various dynamics properties such as **Jacobians**, **mass matrices**, **Coriolis forces**, and **gravitational forces** using the Pinocchio library. These computations are crucial for simulating and controlling robot movements accurately.
- **Utility Methods**:
  - Includes methods to compute kinematics, handle joint state reordering, and perform dynamic simulations based on the actuated parts of the robot.

The Config File (1)

Has you have already notice both classes require a config file.

- The config file needs to be located in a folder called configs.
- You can copy the folder from the roboenv repo.
- The config file is a json file which is made by several different block
- The sim part which does not require to be modified

```
{
    "sim": {
        "time_step": 0.001,
        "experiment_duration": 5,
        "FL": [["FL_foot","FL_foot_fixed"]],
        "FR": [["FR_foot","FR_foot_fixed"]],
        "RL": [["RL_foot","RL_foot_fixed"]],
        "RR": [["RR_foot","RR_foot_fixed"]],
        "feet_contact_names": [["FR_foot","FL_foot","RL_foot","RR_foot"]]
    },
```

This parameter defines the time step of the simulator. In the simulator the time step determine the simulation fidelity and its stability

The Config File (2)

- The pybullet sections contains numerous parameters of interest

```
"robot_pybullet": {
    "base_type": ["fixed"],
    "collision_enable": [0],
    "robot_description_model":[""],
    "urdf_path": ["panda_description/panda.urdf"],
    "ros_urdf_path": "",
    "init_link_base_position": [[0, 0, 0]],
    "init_link_base_vel": [[0.0, 0.0, 0.0]],
    "init_link_base_orientation": [[0.0, 0.0, 0.0, 1]],
    "init_link_base_ang_vel": [[0.0, 0.0, 0.0]],
    "init_motor_angles": [[0.0000, 1.0323, 0.0000, 0.8247, 0.0000, 1.57, 0.0000]],
    "init_motor_vel":[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]],
    "motor_offset": [[]],
    "motor_direction": [[]],
    "motor_damping": [0],
    "motor_damping_coeff": [0],
    "motor_elastic_torque": [0],
    "motor_elastic_coeff": [0],
    "motor_inertia": [0],
    "motor_inertia_coeff": [0],
    "foot_friction": [2],
    "foot_restitution": [0],
    "enable_feet_joint_force_sensors": [["RL_foot_fixed", "RR_foot_fixed", "FL_foot_fixed", "FR_foot_fixed"]],
    "floating_base_name": ["floating_base"],
    "servo_pos_gains": [400],
    "servo_vel_gains": [3],
    "delay_measure_flag":[0],
    "delay_measure_steps":[3],
    "noise_flag":[0],
    "robot_noise":[
            {
                "joint_cov": 0.0001,
                "joint_vel_cov": 0.0001,
                "joint_acc_cov": 0.0001,
                "base_pos_cov": 0.0001,
                "base_ori_cov": 0.0001,
                "base_lin_vel_cov": 0.0001,
                "base_ang_vel_cov": 0.0001,
                "joint_torque_cov":0.0001
            }
```

Specify if the robot is fixed or mobile base

The Config File (3)

- The pybullet sections contains numerous parameters of interest

```
"robot_pybullet": {
    "base_type": ["fixed"],
    "collision_enable": [0],
    "robot_description_model":[""],
    "urdf_path": ["panda_description/panda.urdf"],
    "ros_urdf_path": "",
    "init_link_base_position": [[0, 0, 0]],
    "init_link_base_vel": [[0.0, 0.0, 0.0]],
    "init_link_base_orientation": [[0.0, 0.0, 0.0, 1]],
    "init_link_base_ang_vel": [[0.0, 0.0, 0.0]],
    "init_motor_angles": [[0.0000, 1.0323, 0.0000, 0.8247, 0.0000, 1.57, 0.0000]],
    "init_motor_vel":[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]],
    "motor_offset": [[]],
    "motor_direction": [[]],
    "motor_damping": [0],
    "motor_damping_coeff": [0],
    "motor_elastic_torque": [0],
    "motor_elastic_coeff": [0],
    "motor_inertia": [0],
    "motor_inertia_coeff": [0],
    "foot_friction": [2],
    "foot_restitution": [0],
    "enable_feet_joint_force_sensors": [["RL_foot_fixed", "RR_foot_fixed", "FL_foot_fixed", "FR_foot_fixed"]],
    "floating_base_name": ["floating_base"],
    "servo_pos_gains": [400],
    "servo_vel_gains": [3],
    "delay_measure_flag":[0],
    "delay_measure_steps":[3],
    "noise_flag":[0],
    "robot_noise":[
            {
                "joint_cov": 0.0001,
                "joint_vel_cov": 0.0001,
                "joint_acc_cov": 0.0001,
                "base_pos_cov": 0.0001,
                "base_ori_cov": 0.0001,
                "base_lin_vel_cov": 0.0001,
                "base_ang_vel_cov": 0.0001,
                "joint_torque_cov":0.0001
            }
```

This is the link to the URDF file that contains the robot model. Here the assumption is that all the model are contained in folder called model located with the main script

The Config File (4)

- The pybullet sections contains numerous parameters of interest

```
"robot_pybullet": {
    "base_type": ["fixed"],
    "collision_enable": [0],
    "robot_description_model":[""],
    "urdf_path": ["panda_description/panda.urdf"],
    "ros_urdf_path": "",
    "init_link_base_position": [[0, 0, 0]],
    "init_link_base_vel": [[0.0, 0.0, 0.0]],
    "init_link_base_orientation": [[0.0, 0.0, 0.0, 1]],
    "init_link_base_ang_vel": [[0.0, 0.0, 0.0]],
    "init_motor_angles": [[0.0000, 1.0323, 0.0000, 0.8247, 0.0000, 1.57, 0.0000]],
    "init_motor_vel":[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]],
    "motor_offset": [[]],
    "motor_direction": [[]],
    "motor_damping": [0],
    "motor_damping_coeff": [0],
    "motor_elastic_torque": [0],
    "motor_elastic_coeff": [0],
    "motor_inertia": [0],
    "motor_inertia_coeff": [0],
    "foot_friction": [2],
    "foot_restitution": [0],
    "enable_feet_joint_force_sensors": [["RL_foot_fixed", "RR_foot_fixed", "FL_foot_fixed", "FR_foot_fixed"]],
    "floating_base_name": ["floating_base"],
    "servo_pos_gains": [400],
    "servo_vel_gains": [3],
    "delay_measure_flag":[0],
    "delay_measure_steps":[3],
    "noise_flag":[0],
    "robot_noise":[
            {
                "joint_cov": 0.0001,
                "joint_vel_cov": 0.0001,
                "joint_acc_cov": 0.0001,
                "base_pos_cov": 0.0001,
                "base_ori_cov": 0.0001,
                "base_lin_vel_cov": 0.0001,
                "base_ang_vel_cov": 0.0001,
                "joint_torque_cov":0.0001
```

This are the initialization parameters with which you can set:

- Robot init position
- Robot init orientation
- Robot init lin velocity (if floating base)
- Robot init ang velocity (if floating base)
- Robot init joint positions
- Robot init joint velocities

The Config File (5)

- The pybullet sections contains numerous parameters of interest

```
"robot_pybullet": {
    "base_type": ["fixed"],
    "collision_enable": [0],
    "robot_description_model":[""],
    "urdf_path": ["panda_description/panda.urdf"],
    "ros_urdf_path": "",
    "init_link_base_position": [[0, 0, 0]],
    "init_link_base_vel": [[0.0, 0.0, 0.0]],
    "init_link_base_orientation": [[0.0, 0.0, 0.0, 1]],
    "init_link_base_ang_vel": [[0.0, 0.0, 0.0]],
    "init_motor_angles": [[0.0000, 1.0323, 0.0000, 0.8247, 0.0000, 1.57, 0.0000]],
    "init_motor_vel":[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]],
    "motor_offset": [[]],
    "motor_direction": [[]],
    "motor_damping": [0],
    "motor_damping_coeff": [0],
    "motor_elastic_torque": [0],
    "motor_elastic_coeff": [0],
    "motor_inertia": [0],
    "motor_inertia_coeff": [0],
    "foot_friction": [2],
    "foot_restitution": [0],
    "enable_feet_joint_force_sensors": [["RL_foot_fixed", "RR_foot_fixed", "FL_foot_fixed", "FR_foot_fixed"]],
    "floating_base_name": ["floating_base"],
    "servo_pos_gains": [400],
    "servo_vel_gains": [3],
    "delay_measure_flag":[0],
    "delay_measure_steps":[3],
    "noise_flag":[0],
    "robot_noise":[
            {
                "joint_cov": 0.0001,
                "joint_vel_cov": 0.0001,
                "joint_acc_cov": 0.0001,
                "base_pos_cov": 0.0001,
                "base_ori_cov": 0.0001,
                "base_lin_vel_cov": 0.0001,
                "base_ang_vel_cov": 0.0001,
                "joint_torque_cov":0.0001
            }
```

With this two parameters you can add a damping actrion to the robot motor

- **Motor_damping**: {0,1} can be used to activate the damping
- **Motor_damping_coefficient**: is an array to define the motor dampings

The Config File (6)

- The pybullet sections contains numerous parameters of interest

```
"robot_pybullet": {
    "base_type": ["fixed"],
    "collision_enable": [0],
    "robot_description_model":[""],
    "urdf_path": ["panda_description/panda.urdf"],
    "ros_urdf_path": "",
    "init_link_base_position": [[0, 0, 0]],
    "init_link_base_vel": [[0.0, 0.0, 0.0]],
    "init_link_base_orientation": [[0.0, 0.0, 0.0, 1]],
    "init_link_base_ang_vel": [[0.0, 0.0, 0.0]],
    "init_motor_angles": [[0.0000, 1.0323, 0.0000, 0.8247, 0.0000, 1.57, 0.0000]],
    "init_motor_vel":[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]],
    "motor_offset": [[]],
    "motor_direction": [[]],
    "motor_damping": [0],
    "motor_damping_coeff": [0],
    "motor_elastic_torque": [0]
    "motor_elastic_coeff": [0],
    "motor_inertia": [0],
    "motor_inertia_coeff": [0],
    "foot_friction": [2],
    "foot_restitution": [0],
    "enable_feet_joint_force_sensors": [["RL_foot_fixed", "RR_foot_fixed", "FL_foot_fixed", "FR_foot_fixed"]],
    "floating_base_name": ["floating_base"],
    "servo_pos_gains": [400],
    "servo_vel_gains": [3],
    "delay_measure_flag":[0],
    "delay_measure_steps":[3],
    "noise_flag":[0],
    "robot_noise":[
            {
                "joint_cov": 0.0001,
                "joint_vel_cov": 0.0001,
                "joint_acc_cov": 0.0001,
                "base_pos_cov": 0.0001,
                "base_ori_cov": 0.0001,
                "base_lin_vel_cov": 0.0001,
                "base_ang_vel_cov": 0.0001,
                "joint_torque_cov":0.0001
            }
```

With this two parameters you can add an elastic actrion to the robot motor

- **Motor_elastic_torque:** {0,1} can be used to activate the elastic torque
- **Motor_elastic_coefficient:** is an array to define the motor elastic contribution

The Config File (7)

- The pybullet sections contains numerous parameters of interest

```
"robot_pybullet": {
    "base_type": ["fixed"],
    "collision_enable": [0],
    "robot_description_model":[""],
    "urdf_path": ["panda_description/panda.urdf"],
    "ros_urdf_path": "",
    "init_link_base_position": [[0, 0, 0]],
    "init_link_base_vel": [[0.0, 0.0, 0.0]],
    "init_link_base_orientation": [[0.0, 0.0, 0.0, 1]],
    "init_link_base_ang_vel": [[0.0, 0.0, 0.0]],
    "init_motor_angles": [[0.0000, 1.0323, 0.0000, 0.8247, 0.0000, 1.57, 0.0000]],
    "init_motor_vel":[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]],
    "motor_offset": [[]],
    "motor_direction": [[]],
    "motor_damping": [0],
    "motor_damping_coeff": [0],
    "motor_elastic_torque": [0],
    "motor_elastic_coeff": [0],
    "motor_inertia": [0],
    "motor_inertia_coeff": [0],
    "foot_friction": [2],
    "foot_restitution": [0],
    "enable_feet_joint_force_sensors": [["RL_foot_fixed", "RR_foot_fixed", "FL_foot_fixed", "FR_foot_fixed"]],
    "floating_base_name": ["floating_base"],
    "servo_pos_gains": [400],
    "servo_vel_gains": [3],
    "delay_measure_flag":[0],
    "delay_measure_steps":[3],
    "noise_flag":[0],
    "robot_noise":[
            {
                "joint_cov": 0.0001,
                "joint_vel_cov": 0.0001,
                "joint_acc_cov": 0.0001,
                "base_pos_cov": 0.0001,
                "base_ori_cov": 0.0001,
                "base_lin_vel_cov": 0.0001,
                "base_ang_vel_cov": 0.0001,
                "joint_torque_cov":0.0001
            }
```

With this two parameters you can add a delay in the measurements from the robot

- **delay_measure_flag**: {0,1} can be used to activate the delay
- **Delay_measure_steps**: number of steps of delay with which we will read the system states. The actual delay time can be computed by doing
- Delay_measure_steps*time_steps

The Config File (8)

- The pybullet sections contains numerous parameters of interest

```
"robot_pybullet": {
    "base_type": ["fixed"],
    "collision_enable": [0],
    "robot_description_model":[""],
    "urdf_path": ["panda_description/panda.urdf"],
    "ros_urdf_path": "",
    "init_link_base_position": [[0, 0, 0]],
    "init_link_base_vel": [[0.0, 0.0, 0.0]],
    "init_link_base_orientation": [[0.0, 0.0, 0.0, 1]],
    "init_link_base_ang_vel": [[0.0, 0.0, 0.0]],
    "init_motor_angles": [[0.0000, 1.0323, 0.0000, 0.8247, 0.0000, 1.57, 0.0000]],
    "init_motor_vel":[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]],
    "motor_offset": [[]],
    "motor_direction": [[]],
    "motor_damping": [0],
    "motor_damping_coeff": [0],
    "motor_elastic_torque": [0],
    "motor_elastic_coeff": [0],
    "motor_inertia": [0],
    "motor_inertia_coeff": [0],
    "foot_friction": [2],
    "foot_restitution": [0],
    "enable_feet_joint_force_sensors": [["RL_foot_fixed", "RR_foot_fixed", "FL_foot_fixed", "FR_foot_fixed"]],
    "floating_base_name": ["floating_base"],
    "servo_pos_gains": [400],
    "servo_vel_gains": [3],
    "delay_measure_flag":[0],
    "delay_measure_steps":[3],
    "noise_flag":[0],
    "robot_noise":[
        {
            "joint_cov": 0.0001,
            "joint_vel_cov": 0.0001,
            "joint_acc_cov": 0.0001,
            "base_pos_cov": 0.0001,
            "base_ori_cov": 0.0001,
            "base_lin_vel_cov": 0.0001,
            "base_ang_vel_cov": 0.0001,
            "joint_torque_cov":0.0001
```

With this two parameters you can add a noise in the measurements from the robot

- **noise_flag**: {0,1} can be used to activate the noise
- **robot_noise**: this is a structure that can be used to add a noise to different robot measurements channels
    - Joint
    - Joint vel
    - Joint acc
    - Etc…

The assumption is that the noise has **zero mean** and we can only set the **noise cov**

The Config File (9)

- The pybullet sections contains numerous parameters of interest

```
"robot_pybullet": {
    "base_type": ["fixed"],
    "collision_enable": [0],
    "robot_description_model":[""],
    "urdf_path": ["panda_description/panda.urdf"],
    "ros_urdf_path": "",
    "init_link_base_position": [[0, 0, 0]],
    "init_link_base_vel": [[0.0, 0.0, 0.0]],
    "init_link_base_orientation": [[0.0, 0.0, 0.0, 1]],
    "init_link_base_ang_vel": [[0.0, 0.0, 0.0]],
    "init_motor_angles": [[0.0000, 1.0323, 0.0000, 0.8247, 0.0000, 1.57, 0.0000]],
    "init_motor_vel":[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]],
    "motor_offset": [[]],
    "motor_direction": [[]],
    "motor_damping": [0],
    "motor_damping_coeff": [0],
    "motor_elastic_torque": [0],
    "motor_elastic_coeff": [0],
    "motor_inertia": [0],
    "motor_inertia_coeff": [0],
    "foot_friction": [2],
    "foot_restitution": [0],
    "enable_feet_joint_force_sensors": [["RL_foot_fixed", "RR_foot_fixed", "FL_foot_fixed", "FR_foot_fixed"]],
    "floating_base_name": ["floating_base"],
    "servo_pos_gains": [400],
    "servo_vel_gains": [3],
    "delay_measure_flag":[0],
    "delay_measure_steps":[3],
    "noise_flag":[0],
    "robot_noise":[
            {
                "joint_cov": 0.0001,
                "joint_vel_cov": 0.0001,
                "joint_acc_cov": 0.0001,
                "base_pos_cov": 0.0001,
                "base_ori_cov": 0.0001,
                "base_lin_vel_cov": 0.0001,
                "base_ang_vel_cov": 0.0001,
                "joint_torque_cov":0.0001
            }
```

All these parameters allow the addition of **multiple robots** to the simulator which is why they are define **between [] parenthesis**

The Config File (10)

- Finally the last part of the config defines the pinwrapper beahviour

```
"robot_pin": {
    "base_type": ["fixed"],
    "robot_description_model": [""],
    "urdf_path": ["panda_description/panda.urdf"],
    "ros_urdf_path": [""],
    "floating_base_name": ["floating_base"],
    "joint_state_conversion_active": [1]
}
```

This define if the robot is fixed base or mobile base

This gives the path to the URDF imported in Pinocchio. Having two different ones for the control and one for the simulator is a good practice because it forces the use of an internal model for the controller which is different from the simulated one

An Example: the Cartesian Kinematic Controller

- In this example we have two controls loops:
- **low-level controller:** A first **feedback linearization controller** that stabilize the joints dynamics
- **high-level controller**: A second controller that **tracks a Cartesian trajectory** and provides the desired joints positions and velocities reference for the low-level controller

the Cartesian_kinematic_controller.py (1)

- At the beginning we import all the necessary functions from the library

```
import numpy as np
import time
import os
import matplotlib.pyplot as plt
from simulation_and_control import pb, MotorCommands, PinWrapper, feedback_lin_ctrl, SinusoidalRefere
```

the Cartesian_kinematic_controller.py (2)

- In this block we specify the config file and create the sim object and the dyn_model object

b_sessions_COMP0211/week_2

```python

def main():
    # Configuration for the simulation
    conf_file_name = "pandaconfig.json"  # Configuration file for the robot
    cur_dir = os.path.dirname(os.path.abspath(__file__))
    sim = pb.SimInterface(conf_file_name, conf_file_path_ext = cur_dir)  # Initialize simulation inte

    # Get active joint names from the simulation
    ext_names = sim.getNameActiveJoints()
    ext_names = np.expand_dims(np.array(ext_names), axis=0)  # Adjust the shape for compatibility

    source_names = ["pybullet"]  # Define the source for dynamic modeling

    # Create a dynamic model of the robot
    dyn_model = PinWrapper(conf_file_name, "pybullet", ext_names, source_names, False,0,cur_dir)
```

the Cartesian_kinematic_controller.py (3)

- In this block we define some information for the cartesian controller
  - The link that we want to control
  - The initial cartesian position of the robot

```
controlled_frame_name = "panda_link8"
init_joint_angles = sim.GetInitMotorAngles()
init_cartesian_pos,init_R = dyn_model.ComputeFK(init_joint_angles,controlled_frame_name)
```

the Cartesian_kinematic_controller.py (4)

- In this block we setup the sinusoidal reference for the Cartesian controller
- The library comes with sinusoidal reference generator but it can be expanded with new reference generator

```python
    amplitudes = [0, 0.1, 0]  # Example amplitudes for 4 joints
    # Specify different frequency values for each joint
    frequencies = [0.4, 0.5, 0.4]  # Example frequencies for 4 joints

    # Convert lists to Nu (variable) amplitudes: list ation in computations
    amplitude = np.array(amplitudes)
    frequency = np.array(frequencies)
    ref = SinusoidalReference(amplitude, frequency,init_cartesian_pos)  # Initialize the reference
```

the Cartesian_kinematic_controller.py (5)

- In this block we setup the setup the gains for the high level controller and the low level controller
- We also initial the motor command object that is used to send commands to the simulated robots

```
    # Command and control loop
    cmd = MotorCommands()  # Initialize command structure for motors

    # P conttroller high level
    kp_pos = 100 # position
    kp_ori = 0   # orientation

    # PD controller gains low level (feedbacklingain)
    kp = 1000
    kd = 100
```

the Cartesian_kinematic_controller.py (6)

- From line 78 starts the control loop
- Line 80,81, and 82 are used for reading the current state of the robot from the sim class
- In line 86 we get the current desired Cartesian Pos and vel from our reference generator

```
78      while True:
79          # measure current state
80          q_mes = sim.GetMotorAngles(0)
81          qd_mes = sim.GetMotorVelocities(0)
82          qdd_est = sim.ComputeMotorAccelerationTMinusOne(0)
83          # Compute sinusoidal reference trajectory
84          # Ensure q_init is within the range of the amplitude
85
86          p_d, pd_d = ref.get_values(current_time)  # Desired position and velocity
```

the Cartesian_kinematic_controller.py (7)

- In this block we compute the control actions
- In line 91 the inverse differential kinematics compute the desired joint positions and velocity
- In line 94 given q_des an qd_des we compute the torque with the feedback lin controller
- In line 95 we advance the simulation by one using the Step function

```
90        ori_d_des = None
91        q_des, qd_des_clip = CartesianDiffKin(dyn_model,controlled_frame_name,q_mes, p_d, pd_d, ori_d
92
93        # Control command
94        cmd.tau_cmd = feedback_lin_ctrl(dyn_model, q_mes, qd_mes, q_des, qd_des_clip, kp, kd)  # Zero
95        sim.Step(cmd, "torque")  # Simulation step with torque command
96
```

Run It and see What Happens!

- Try to change the input Cartesian trajectory
- Try to change the gains for both high level and low-level controller
- Try to add a delay in the measurement by changing the config file
- Try to add measurement noise by changing the config file
- Try to add an elastic effect on the motors by changing the config file
- Try to add a damping to the motors by changing the config file
- Try to change the config file and simulate the elephant robot