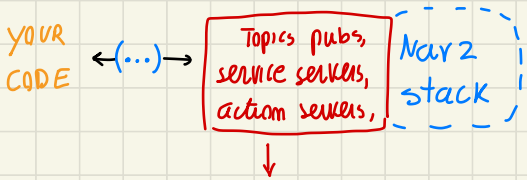
8) INTERACT PROGRAMMATICALLY WITH THE NAVIGATION STACK

So far, we use Rviz 2 to select initial pose and send Nav2 Goal
this is fine for testing → NEXT step is to automate this!

The objective is to interact programmatically with Nav2, to integrate Nav2 functionalities directly in your code and inside your ROS2 node → to use Nav stack directly from your custom ROS2 application



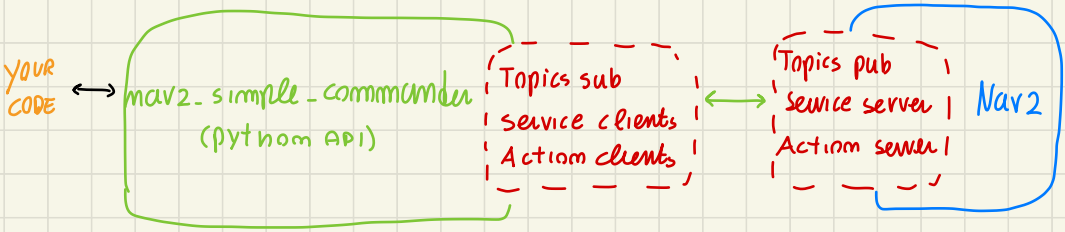
this is possible, in fact Navigation stack works by using topics, services, actions and so on...

- When you set 2D Pose Estimate in Rviz, behind the scene it has been used a topic
- When Nav2 Goal is sent, we use an action

We already have an INTERFACE to Nav2 stack, we need just to see which topic, services, actions are used and interact with it from your ROS2 code... → there is an easy to way to do it, by using an API developed for this:

nav2_simple_commander (python API)

let's you communicate with Nav2 stack with few lines of basic python code → all you need to do is to use this API in your own code to set initial pose, send nav2 Goal from your code



In this lecture we will:

- see what are those topics/actions used by Nav2
- learn how to use simple_commander API to interact programmatically with Nav2 stack

DISCOVER WHAT TOPICS and ACTIONS ARE USED

FIRST, let's understand what are the ROS2 interfaces used when 2D Pose Estimate is set or Nav2 Goal is sent.

↓

this give you a global understanding, to know what happens behind the scene and use smarter the nav2 API

With turtlebot3_gazebo and nav2 running, we can visualize the network communication topics:

There are a lot of topics,

- for 2D Pose Estimate, we are interested on /initial_pose topic

↓

in which you publish the Initial Robot pose of type

geometry_msgs/msg/PoseWithCovarianceStamped

↑ defined as Pose (position + orientation) stamped with time

So, to set initial pose without using Rviz2, you need a publisher on this topic

by listening on this /initial_pose topic, when sending 2D Pose Estimate we can see how this message is formatted

this will have:

```
ed@pc:~$ ros2 topic info /initialpose
Type: geometry_msgs/msg/PoseWithCovarianceStamped
Publisher count: 1
Subscription count: 1
ed@pc:~$ ros2 topic echo /initialpose

header:
  stamp:
    sec: 139
    nanosec: 778000000
  frame_id: map
pose:
  pose:
    position:
      x: -0.009042748250067234
      y: 0.011425078846514225
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: -0.012668822547163956
      w: 0.9999197472473821
  covariance:
  - 0.25
```

shape of initial pose

it is given as QUATERNION orientation

.... (covariance data)

⇒ quaternions are used instead of RPY angles because simpler to compute and manipulate

(it is possible to easily convert an euler orientation into quaternions)

- for Nav2 Goal, we need to refer to available actions,

especially /navigate_to_pose used when single Nav2 Goal Pose given

/follow_waypoints used when waypoint Follower is called

by looking at the characteristic of those actions

```
ed@pc:~$ ros2 action info /navigate_to_pose
Action: /navigate_to_pose
Action clients: 4
    /rviz2
    /rviz2
    /bt_navigator
    /waypoint_follower
Action servers: 1
    /bt_navigator
```

→ it is behavior tree navigator, which is inside Nav2 stack. This will call different global/local planner

Sending this messages, actions from terminal, it is complex due to the amount of fields we need to fill!

- OPTION 1

this can be done from a custom node to publish on topic, create an action client

- OPTION 2

use a simplified python API nav2_simple_commander

SIMPLE COMMANDER API - INSTALL AND SET THE INITIAL POSE

Now, we know the different ROS2 interfaces used to communicate with Nav2 stack, we want to use Python code to send Navigation commands! => 1ST step: set-up API and use it to send initial Robot pose...

install `ros-<distro>-nav2-simple-commander`

Then, create a new python scipt `nav2_test.py` wherever you want

remember to make it as executable: `chmod +x nav2_test.py`
and open with your text editor
(for example using VSCode)

Here, we will run the code from this python scipt, it is easy to create it as a py package, including this code in your own node...

`nav2_test.py`

FIRST we need to initialize ROS communication (before using any ROS library) by `import rclpy`

```
...
def main():
    rclpy.init()
    ...
    rclpy.shutdown()
```

We also need to initialize Nav2 simple commander API by importing `BasicNavigator`, then create an object of that type. This class contain all we need to interact with Nav2

start the scipt with `#!/usr/bin/env python3` to set the interpreter and run it from terminal

```python
#!/usr/bin/env python3
import rclpy
from nav2_simple_commander.robot_navigator import BasicNavigator

def main():
    # --- Init
    rclpy.init()
    nav = BasicNavigator()

    rclpy.shutdown()

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

~> ./nav2_test.py to check if library is properly installed and NO error occurs

If you have include error, be sure the library is properly installed

once ROS2 communication is initialized and BasicNavigator() object exists:

We can set initial pose easily by using nav giving Pose Stamped object to it!

calling setInitialPose(•) method with nav,

we can send as message a PoseStamped, inside the method, this is properly converted to Pose with covariance stamped which is the one used by Nav2 stack

```
initial_pose = PoseStamped()
initial_pose.header.frame_id = 'map'      ← because we want to give pose relative to the map frame
initial_pose.header.stamp = nav.get_clock().now().to_msg()      ← to take the time
initial_pose.pose.position.x = 0.0  ┐
    //      //      //     .y = 0.0  ├  we will fix x, y, z real position later on
    //      //      //     .z = 0.0  ┘
    //      //  .orientation.x =
    //      //      //     .y =
    //      //      //     .z =
    //      //      //     .w =
```

orientation is given as (x, y, z, w) as QUATERNION, it is easyer to give as EULER RPY and convert it using a proper library

to compute x, y, z, w just install

- ros-<distro>-tf-transformations
- python3-transforms3d

Then import tf_transformations inside nav2_test.py

and use the method `tf_transformations.quaternion_from_euler()`
and assign it to
q_x, q_y, q_z, q_w
and use it to fill the orientation fields

{ quaternion (x,y,z,w) from euler (x,y,z) }

finally we are ready for: `nav.setInitialPose(initial_pose)`

And then we need Nav2 to be active, to be sure initial pose is set, once `nav2.waitUntilNav2Active()`
we are sure Initial Pose is set up correctly!

We are ready to run this, first of all let's start the Robot and Nav stack for example by using the usual turtlebot3 packages

Launching `turtlebot3_gazebo` and `turtlebot3_navigation2` packages
⇓
by looking in Rviz2 you can see what is going on when using the nav2 API from your code.

NOW you can run `nav2_test.py`
be sure ROBOT initial pose is sent as the correct one, in our case, pose.position is at map origin (0,0,0)
and pose.orientation is RPY = (0,0,0), NO ORIENTATION

This will set Nav2 Pose correctly!
So, we can interact with Nav2 without using Rviz2, in fact if we do the same using `nav2_bringup` package, without Rviz, by running our script we can set initial pose correctly!

NEXT STEP: Use the API to send Nav2 Goal to the Robot!

NOTE IF you have errors with import `tf_transformations`, try to reinstall and upgrade the library with

`pip install --upgrade transforms3d`

SIMPLE COMMANDER API - SEND A NAV2 GOAL

Let's continue to write our script nav2_test.py, now to give nav2 Goal!
To test the API, for example we can define

Goal as { x = 3.5, y = 1.0, z = 0.0
R = 0.0, P = 0.0, Y = 1.57

move to (3.5, 1) in xy plane with 90° orientation on the left

⇓

to send a Goal we need to:

1) create a goal_pose as PoseStamped()
Initialize this as before, with proper header and pose
for example position.x = 3.5
position.y = 1.0
orientation.yaw = π/2 = 1.57

using right hand anti-clock wise convention
+ orientation positive

2) use method nav.goToPose(goal_pose)
This JUST send the pose, but we need to wait for navigation finish:
we can wait for navigation to goal by

⇓

3) While not nav.isTaskComplete():
this method return True if task given is completed

feedback = nav.getFeedback() ← we get current position as feedback just to check what is going on
print(feedback)

We can now launch the Robot stack (Gazebo) and nav2, and run the script in another terminal

BUT: if we try to re-run nav2_test.py... the initial pose will be again set up to (0,0,0) map, and it will mess up everything!

↓

the initial pose is wrongly set-up

Be carefull! once initial pose is setted once, DON'T try to set it again

Next time you run it, don't use nav.setInitialPose(initial_pose)
We can manage it by adding main() arguments, to
set initial_pose true or false when we run from terminal
(by using argv, argc of main...)

We can also get the final result of the navigation, by calling
nav.getResult() after the wait!
You can see if task is SUCCEDED or ABORTED

SIMPLE COMMANDER API - WAY POINT FOLLOWER

Let's see how to use the API to send some waypoints using waypoint follower mode directly from our script

⇓

OPTION 1: use goToPose method in eoop, for any desired goal

OPTION 2: use followWaypoints method, specific for this task

↓

this require a list of goal poses, each as PoseStamped() as before

↓

to make the code more effcient, let's define a function that return a pose stamped instead of rewriting all the code any time for any pose...

required to get the timestamp

def create_pose_stamped(navigator, position_x, position_y, orientation_z)

only pose info that can be specified in a 2D plane

here the code is the same used to define goal_pose before, using the arguments properly in the initialization, finish with a return pose

Then, we can use this function to make the code cleaner. Using it to create the initial_pose as x=0.0, y=0.0, yaw=0.0
And also to define a nav2 Goal with proper x, y, yaw arguments

↓

so, I can use create_pose_stamped(nav, x, y, yaw) to define multiple goal_pose i i=1,2,3,...,N

store those poses in an array waypoints = [goal_pose1, goal_pose2, ...] and finally call

nav.followWaypoints(waypoints)

remember from to: while not nav.isTaskComplete() as before...

Then you can run the code and way point follower functionality will be started...

Always be carefull of avoidim reinitialize Pose in (0,0,0) after runnimg the first time...

NOTICE: I personally decided to customize the code to avoid commentimg out part of the scupt anytime.
I manage the functionality from the argoument given when runnimg the scupt in the terminal:

» ./nav2_test.py argoument
this will be argv[1]

- Initialize: only the first time to set initial Pose
- Goal: to send a simple Goal
- Waypoints: to send multiple Goal followed with waypoint follower

It is easy to further modify the scupt to give the goal pose [x y yaw] as argv arag ouments...

• ASSIGNMENT 4: SIMPLE COMMANDER API - CREATE A ROBOT PATROL

Practice a bit with this API.
usimg turtlebot3_house as Robot stack, simulation environment
previously we map it and patrol araund usimg Rviz Nav2 Goal

Task: stant simulation + Nav2 stack (with / without Rviz)
then, with a custom pythom scupt set initial Robot pose and Patrol araund the house environment

SOLUTION: the solution is easy, launch turtlebot3_house.launch.py and navigation2.launch.py, choose some points (x, y, yaw) in each room (by looking at Rviz)
choose waypoints wisely to cross doors without "U-shape" turn that doesn't work very well with turtlebot3 navigation (by usimg additional points to go through). use the same nav2_test.py scupt, just Redefine the goal_pose i composimg waypoints array
(you can then stant to waypoint follow in loop)