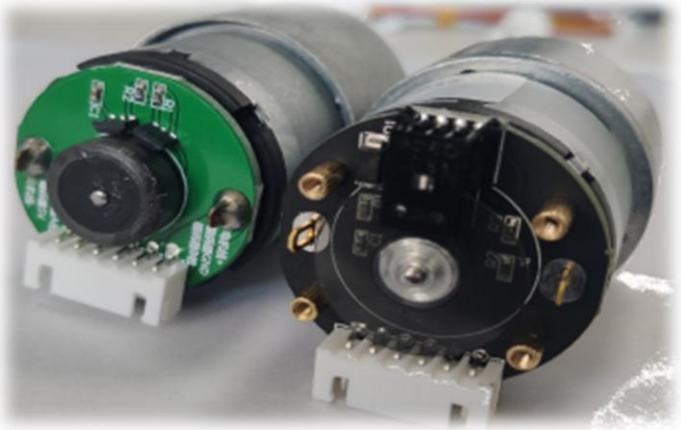
[Week 3]
Sensor fusion, localization and mapping

授課教師：郭重顯 教授
助教：Nguyen Thanh Thien Phuc、莊明洋
助教實驗室：工綜 106

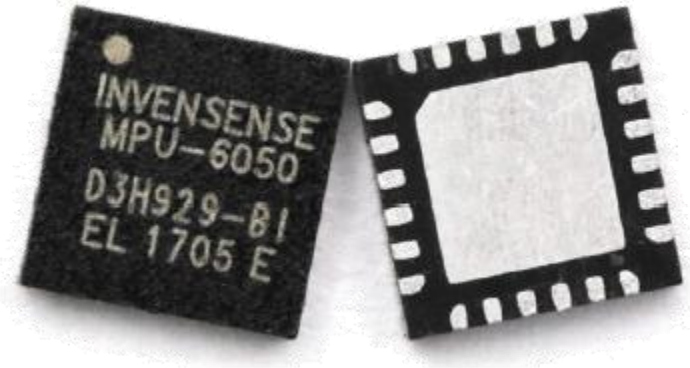
Sensors for the AMR

❖



Odometer

IMU



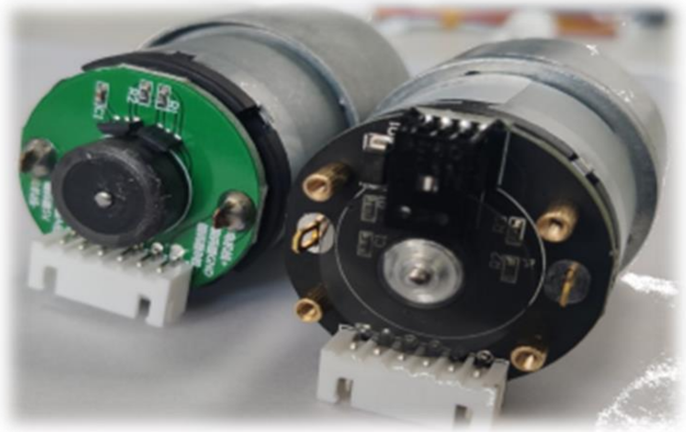
Camera

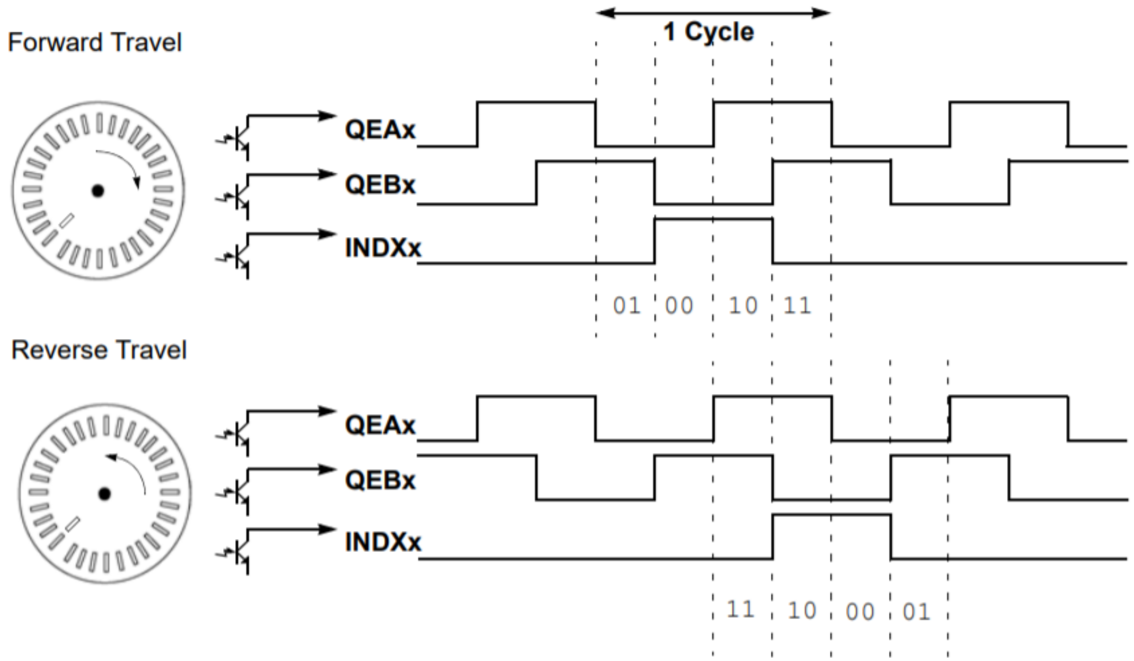
LiDAR

http://ww1.microchip.com/downloads/en/DeviceDoc/70208A.pdf

Encoder Decoding: Quadrature Encoder Interface

- The Quadrature Encoder Interface (QEI) module provides the interface to incremental encoders for obtaining mechanical position data.
  - If Phase A leads Phase B, the direction of the motor is deemed positive, or forward.
  - If Phase A lags Phase B, the direction of the motor is deemed negative or reverse.



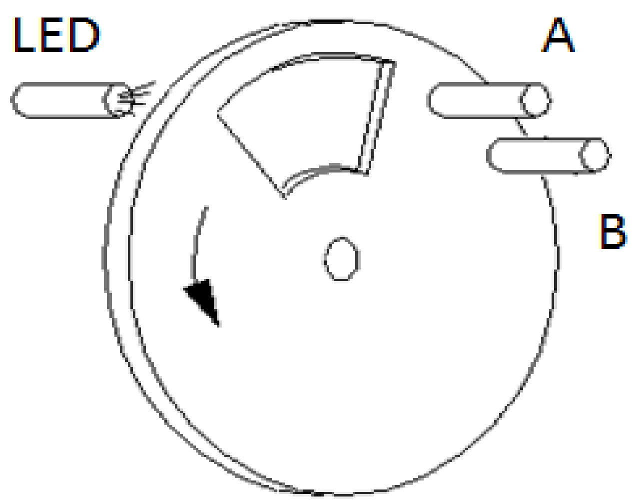


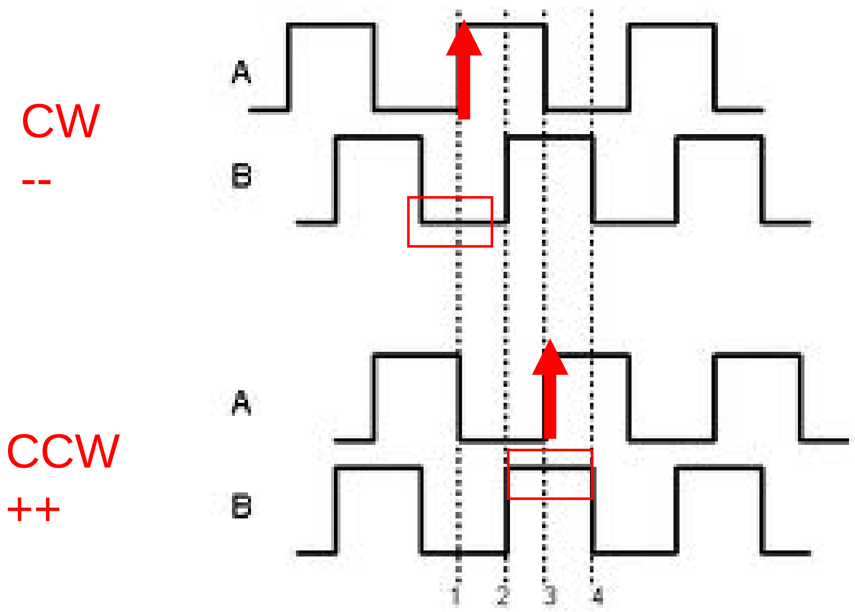
A/B Phases QEI

❖Direction determination and counting of pulses

❑Desirable for <u>**closed loop control (e.g., PID)**</u> of wheel angular position and velocity

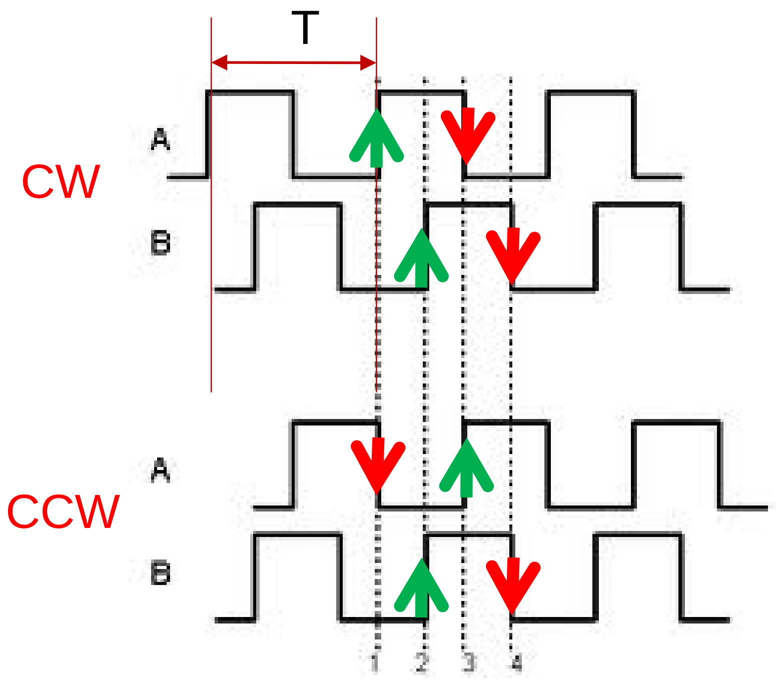
❑Applicable for **odometry** of AMR in terms of **kinematics**





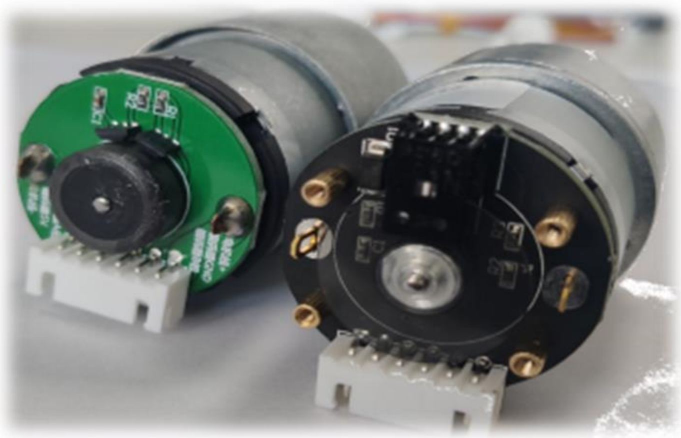
Decoding QEI with 1X, 2X and 4X

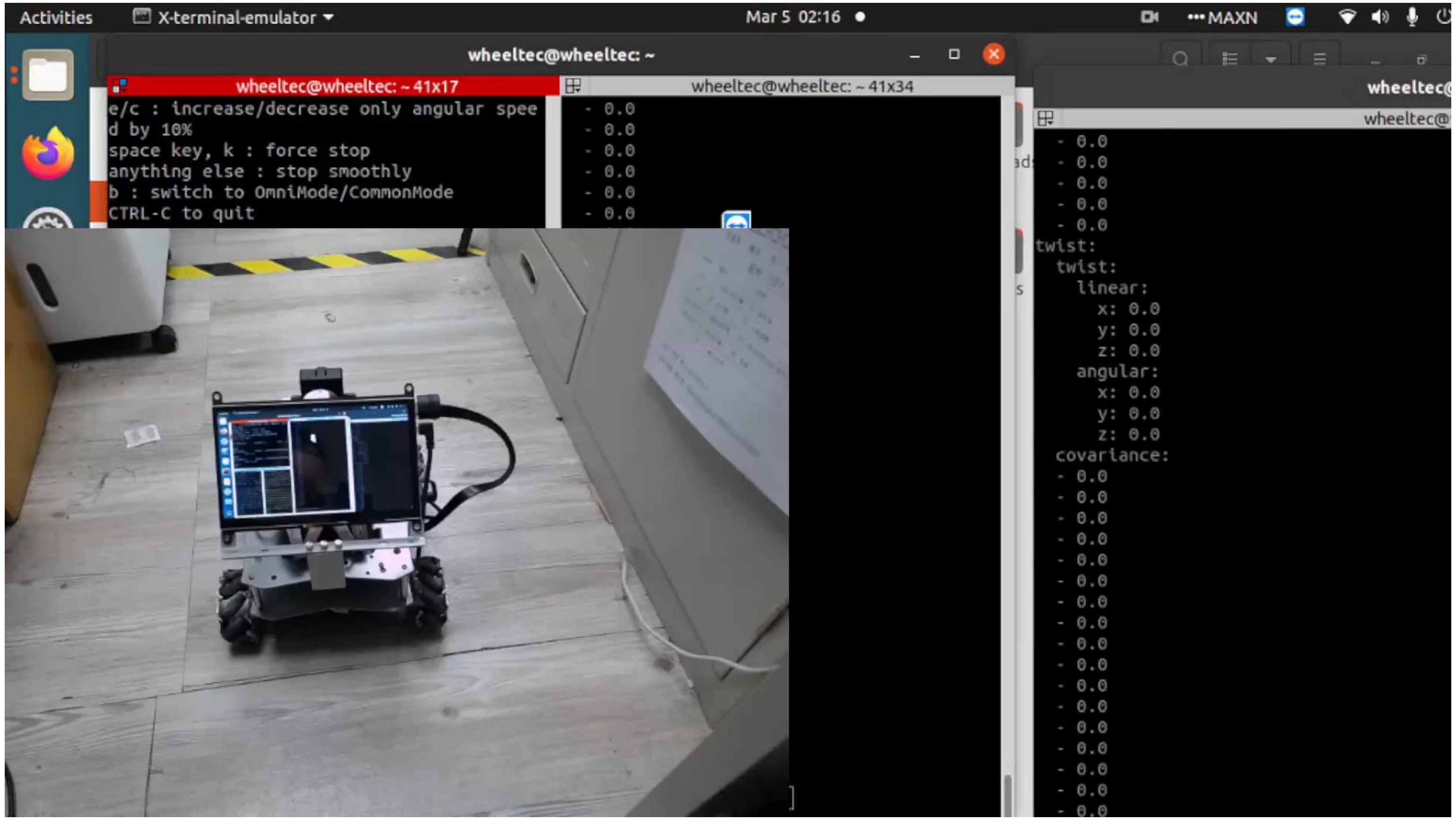
- A complete cycle, T (A & B Phases)
  - 4 Times Resolutions (90 Deg. Phase Diff.)
    - A channel rising
      - B channel is in low level
      - B channel is in high level
    - A channel falling
      - B channel is in low level
      - B channel is in high level
    - B channel rising
      - A channel is in low level
      - A channel is in high level
    - B channel falling
      - A channel is in low level
      - A channel is in high level



Encoders for odometry demonstration with ROS

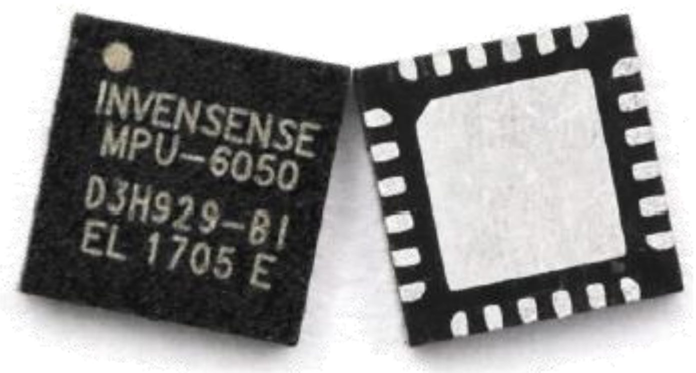
❖ **Odometer**

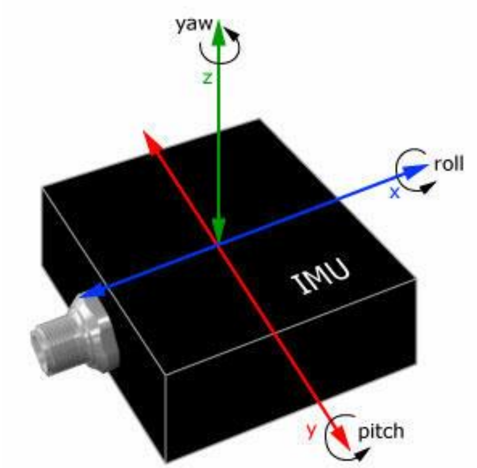


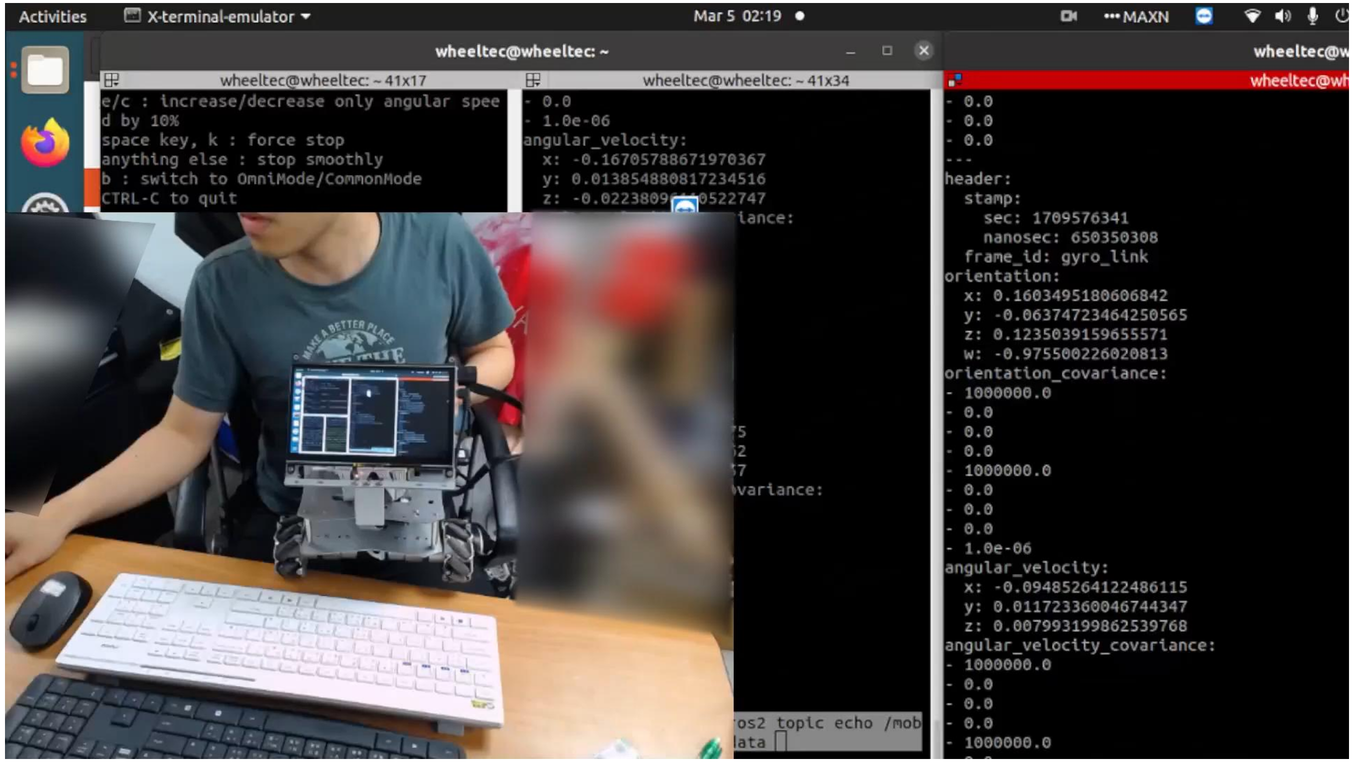


Poses of AMR with IMU

❖ **IMU:** inertial measurement unit







Credit:

https://www.usgs.gov/centers/pcmsc/science/inertial-measurement-unit-imu

Sensor Fusion

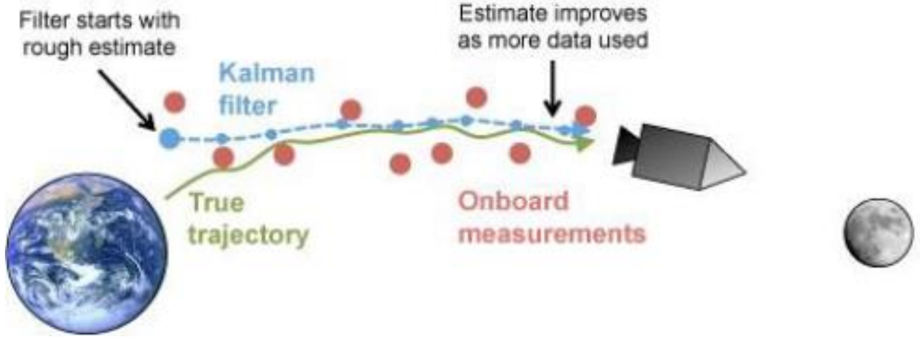
Credit:
https://www.wevolver.com/article/what-is-sensor-fusion-everything-you-need-to-know
https://www.analog.com/en/resources/analog-dialogue/raqs/raq-issue-139.html
https://micromega-dynamics.com/files/uploads/2020/03/TN01-MMD-Acceleration-Noise-Density.pdf

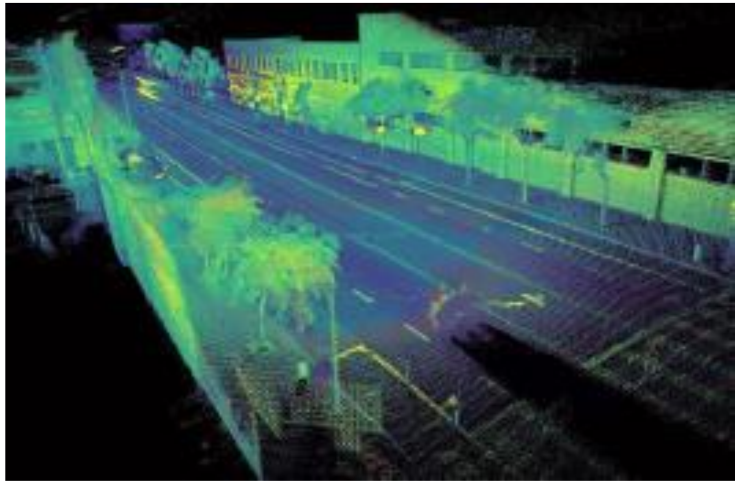
- Sensor fusion is a technique that combines data from **multiple sensors** to generate **a more accurate and reliable understanding** of the environment than what could be achieved using individual sensors alone.
  - Sensor Fusion example for IMU with **gyroscope** and **accelerometer**
    - **Gyroscopes** are subject to bias instabilities, in which the initial zero reading of the gyroscope will cause drift over time due to integration of inherent imperfections and noise within the device.
    - **Accelerometer exhibits high frequency noises**
    - **Solution: Kalman filter** algorithm could be desirable to reduce noise at the accelerometer and gyroscope sensor output.
  - AMR sensor fusion
    - Wheel encoder odometry: high resolution with accumulated odometry errors
    - LiDar SLAM: low resolution but less accumulated errors
    - Solution: **Kalman filter** algorithm could used for this purpose

Kalman Filter

- The **Kalman** filter is an algorithm used for **estimating the state** of a **linear dynamic system** from a series of **noisy measurements**.
- It was developed by Rudolf E. Kálmán in the late 1950s and has since become a fundamental tool in various fields such as control theory, signal processing, and **navigation**.
  - Its initial focus was on developing **optimal estimation techniques** for aerospace applications, particularly for trajectory estimation problems in space exploration.
- Hint: Extended **Kalman** filter is for nonlinear dynamic systems
  - Usage:
    - Simply linearizing all the nonlinear models so that the traditional linear Kalman filter equations can be applied

Credit:
https://www.cs.unc.edu/~welch/kalman/media/pdf/Julier1997_SPIE_KF.pdf





Kalman Filter

❖ Model: Linear Time Varying System consisting of **process** and **measurement** noises

$$x_t = A_t x_{t-1} + B u_t + \varepsilon_t$$ Process noise

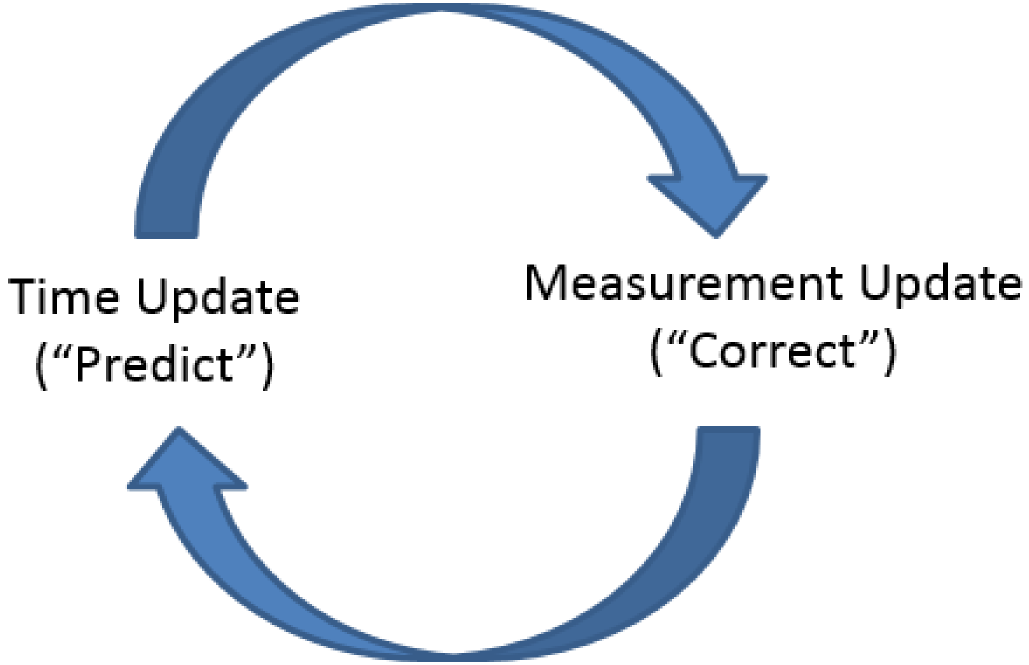
$$y_t = C_t x_t + \delta_t$$ Measurement noise

- x_t: Non-observable state
- y_t: Observable measured state
- A_t:(n*n)how state evolves without control or noise
- B_t:(n*l)how control u_t changes the state → **Motion model**
- C_t:(k*n)how to map state x to be observation z →**sensor model**
- ε_t, δ_t: process and measurement noise → **random variable**

Kalman Filter

Credit: https://www.kalmanfilter.net/multiSummary.html

- Kalman Filter operates in a "predict-correct" loop
  - Once **initialized**
    - the Kalman Filter **predicts** the system state at the next step.
    - It also provides the **uncertainty** of the prediction.
  - Once the **measurement** is **received**,
    - the Kalman Filter **updates** (or **corrects**) the prediction and the uncertainty of the current state. As well the Kalman Filter predicts the following states, and so on.



Prediction-and-Update Process Loop

Credit:
https://www.kalmanfilter.net/multiSummary.html
https://en.wikipedia.org/wiki/Covariance_matrix

Term	Name
$\boldsymbol{x}$	State Vector
$\boldsymbol{z}$	Measurements Vector
$\boldsymbol{F}$	State Transition Matrix
$\boldsymbol{u}$	Input Variable
$\boldsymbol{G}$	Control Matrix
$\boldsymbol{P}$	Estimate Covariance
$\boldsymbol{Q}$	Process Noise Covariance
$\boldsymbol{R}$	Measurement Covariance
$\boldsymbol{w}$	Process Noise Vector
$\boldsymbol{v}$	Measurement Noise Vector
$\boldsymbol{H}$	Observation Matrix
$\boldsymbol{K}$	Kalman Gain
$\boldsymbol{n}$	Discrete-Time Index

共變異數: 度量兩隨機變數關係的強弱

輸入系統狀態向量X_0、與系統共變異矩陣P_0之初始值以及Q、R矩陣 **(Covariance)**

Predict

估測系統狀態：$X_{k|k-1} = F(X_{k-1|k-1}) + G_k U_k + w_k$

更新系統共變異矩陣：$P_{k|k-1} = F P_{k-1|k-1} F^T + Q$

Update

計算卡爾曼增益：$K_k = P_{k|k-1} H^T [H P_{k|k-1} H^T + R]^{-1}$

利用測量值更新估測值：$X_{k|k} = X_{k|k-1} + K_k (Z_k - H(X_{k|k-1}))$

更新系統共變異矩陣：$P_{k|k} = (I - K_g H) P_{k|k}$

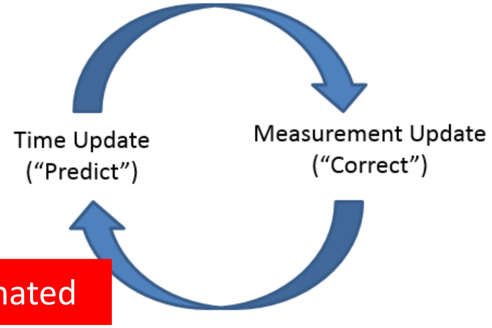
X_k estimated

P_k predicted

K_k obtained

X_k updated

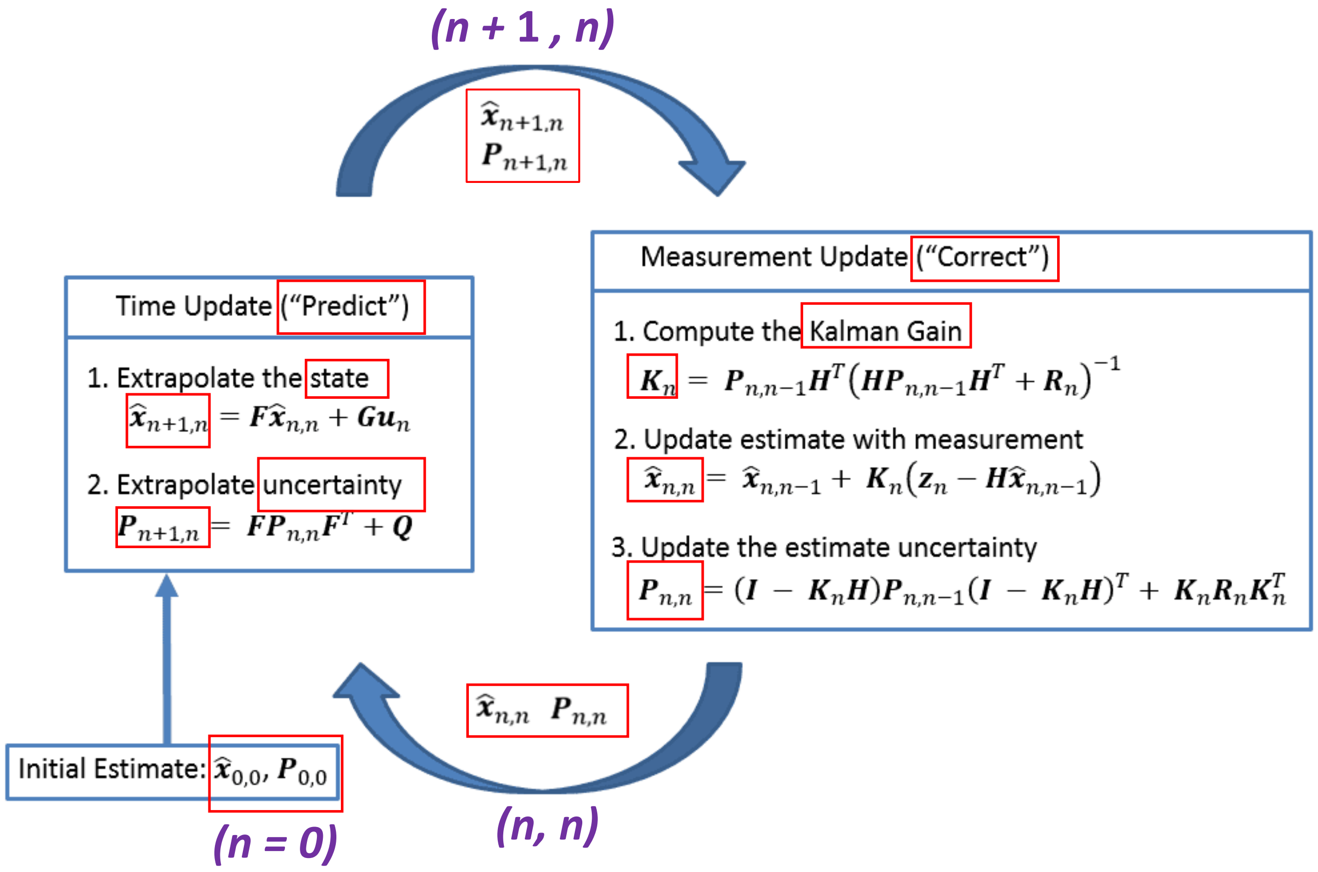
P_k updated



In probability theory and statistics, a **covariance matrix** (also known as **auto-covariance matrix**, **dispersion matrix**, **variance matrix**, or **variance–covariance matrix**) is a square matrix giving the covariance between each pair of elements of a given random vector.

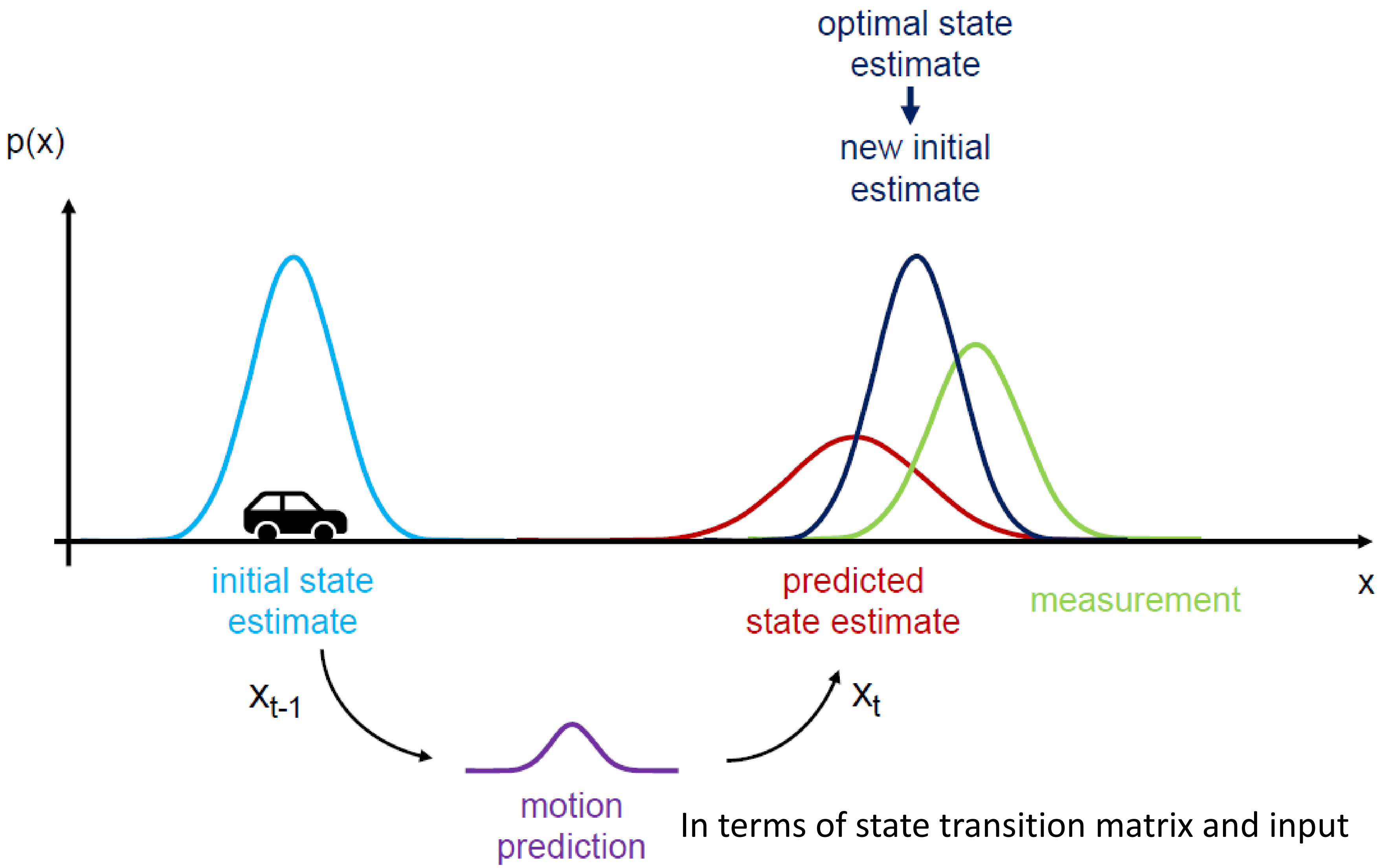
More Formal Kalman Filter Iteration Process

Term	Name
$\boldsymbol{x}$	State Vector
$\boldsymbol{z}$	Measurements Vector
$\boldsymbol{F}$	State Transition Matrix
$\boldsymbol{u}$	Input Variable
$\boldsymbol{G}$	Control Matrix
$\boldsymbol{P}$	Estimate Covariance
$\boldsymbol{Q}$	Process Noise Covariance
$\boldsymbol{R}$	Measurement Covariance
$\boldsymbol{w}$	Process Noise Vector
$\boldsymbol{v}$	Measurement Noise Vector
$\boldsymbol{H}$	Observation Matrix
$\boldsymbol{K}$	Kalman Gain
$\boldsymbol{n}$	Discrete-Time Index



Credit:
https://www.kalmanfilter.net/multiSummary.html

Kalman Filter

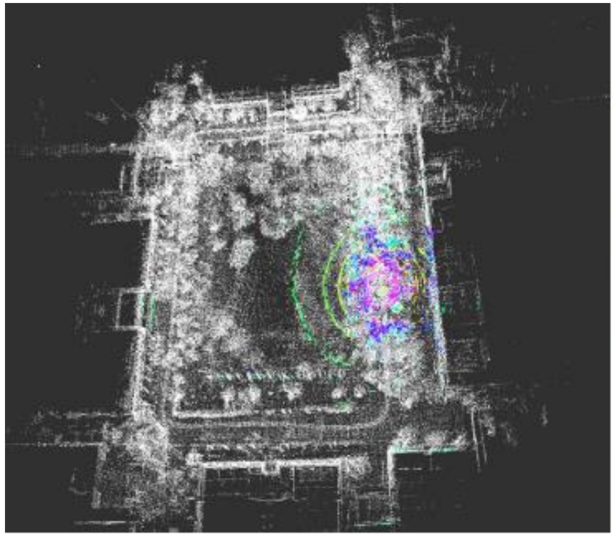


In terms of state transition matrix and input

SLAM

- Simultaneous Localization and Mapping (SLAM) is a technique for constructing maps while simultaneously determining the position of a sensor or device within the map.

- It integrates sensor data, such as vision, lidar, odometer or inertial measurements, to create a map of the environment and estimate the sensor's trajectory relative to the map (2D, 3D).



3D SLAM Example

Process of SLAM

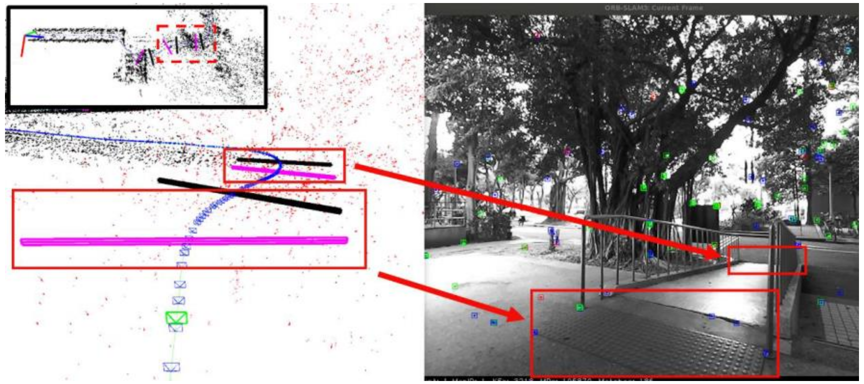
- The process of SLAM typically involves the following steps:
  - **Data Collection**: The robot will collect depth information about the environment through its sensors, such as LiDAR or stereo cameras.
  - **Feature extraction**: Extract useful features from the collected depth data, such as corners, edges, or other prominent landmarks.
  - **Location estimation**: Based on the feature points, use filters or other optimization methods to estimate the robot's position.
  - **Map construction**: Integrate the estimated position with sensor data to construct the map of the environment (accumulation).
  - **Closed-loop detection**: Identifying previously visited locations of the robot and **comparing them with current sensor** data to adjust the **consistency** of map information and trajectory
    - **Loop closure** is a sub-algorithm of SLAM that is about identifying previously visited locations and using them to **correct the accumulated errors** in the robot's pose estimation

Credit: https://www.thinkautonomous.ai/blog/loop-closure/

Common SLAM Methods

❖Common SLAM algorithms include:

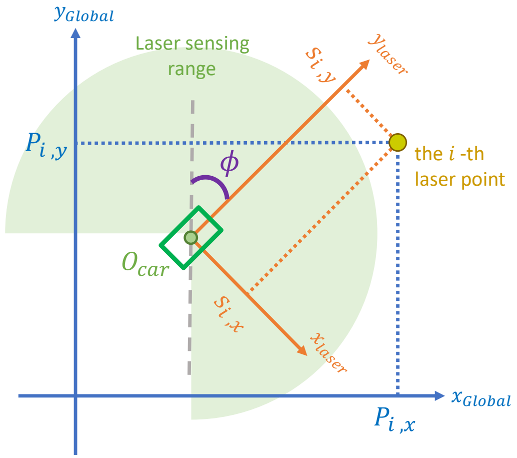
- ❑**Gmapping SLAM**: Utilizes particle filtering and grid mapping techniques for localization and mapping, suitable for long corridors and low-feature environments.
  - o Odometry data and the laser scan data
- ❑**Karto SLAM**: Relies on feature points for localization and mapping, typically used for constructing 2D maps with good mapping accuracy and localization precision.
- ❑**Cartographer SLAM**: An advanced SLAM system developed by Google, integrating various sensor data such as lidar and visual sensors to build high-quality 2D and 3D SLAM.
- ❑**ORB-SLAM**: A visual SLAM method based on ORB feature descriptors, known for its real-time performance and robustness, applicable to both indoor and outdoor environments.
- ❑**Hector SLAM**: A scan matching-based SLAM method primarily used for indoor mobile robot localization and map building.



ORB SLAM

Hector SLAM

- Hector SLAM, as one of the implementations of SLAM technology, can realize map construction **without the assistance of external position sensors**.
- Its principle is to update the map and **estimate the state** through the high-frequency scanning and **matching of a lidar**.
- In the scanning and matching stage, the current data will be compared with the original map data, and an **error function** will be generated , and finally the minimum value of this error function can be calculated by the **Gauss-Newton method**.
  - An iterative method regularly used for solving nonlinear least squares problems



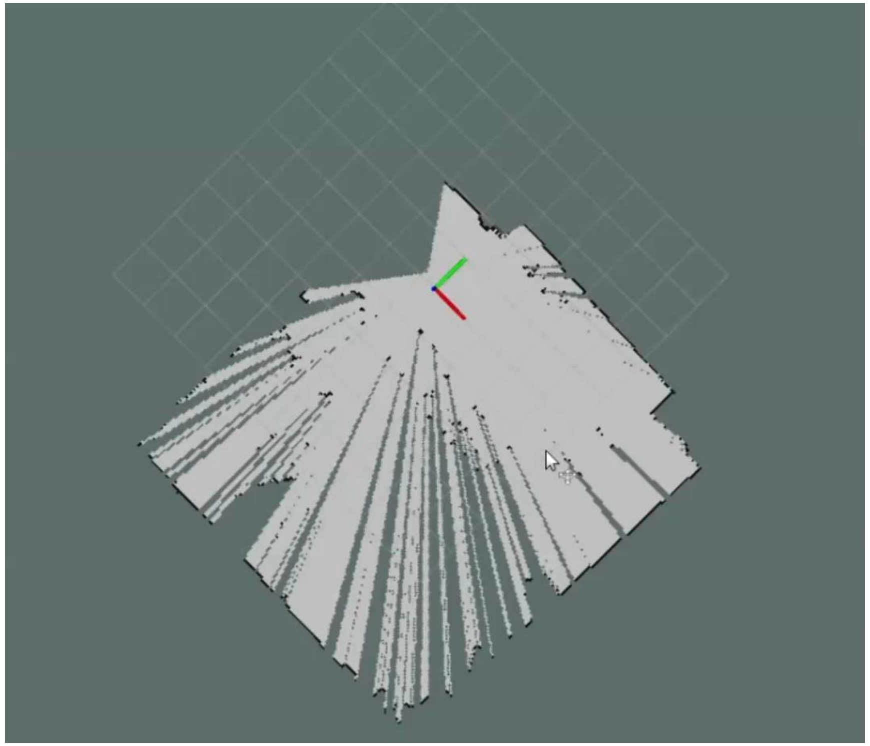
$$\xi^* = \underset{\xi}{\operatorname{argmin}} \sum_{i=1}^{n} \left[1 - M\left(S_i(\xi)\right)\right]^2$$

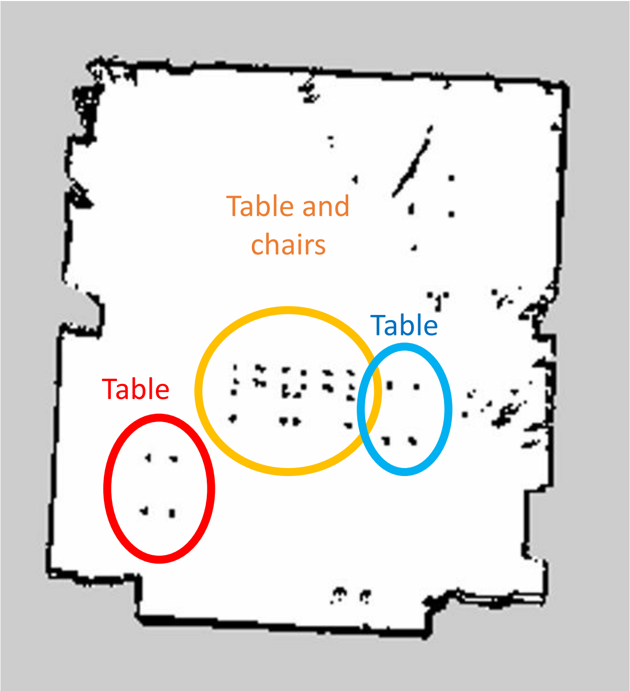
$$\boldsymbol{S}_i(\xi) = \begin{bmatrix} cos(\phi) & -sin(\phi) \\ sin(\phi) & cos(\phi) \end{bmatrix} \begin{bmatrix} s_{i,x} \\ s_{i,y} \end{bmatrix} + \begin{bmatrix} P_x \\ P_y \end{bmatrix}$$

Rotation and translation

Hector SLAM

❖ **Hector SLAM Demonstration**





Gmapping SLAM

❖ Based on the RBPF (Rao-Blackwellised particle filter) SLAM algorithm, the Gmapping algorithm uses **BOTH** Lidar data and odometer data to optimize the proposed distribution and introduce an adaptive resampling mechanism, which greatly reduces the number of required particles, improves computational efficiency, and satisfies most real-time application scenarios.

Hint:
Since there are many parameters that need to be set in the gmapping algorithm, so we often write the startup relatives of gmapping into the **launch** file.

Adaptive Monte Carlo Localization, AMCL

- Adaptive Monte Carlo localization (AMCL) is a modified version of the Monte Carlo localization method, the concept of which is to use a set of **randomly generated particles** (i.e., samples) to represent the possible position and orientation of a robot in the environment.
- As the robot moves in the environment, the **particle distribution state** is updated in an adaptive way, and then the actual position of the robot is estimated.
- The goal of the AMCL method is to maintain the position probability distribution of the robot at different times.
- Roughly speaking, particle filtering (PF) means to scatter a set of particles evenly in the map space at the beginning, and then move the particles by obtaining the motion of the robot.
  - For example, if the robot moves forward one meter, all particles will also move forward one meter, regardless of whether the current position of the particle is correct or not.
    - Use the position of each particle to simulate a sensor information and compare it with the observed sensor information (usually laser), thereby assigning a probability to each particle.
    - Then the particles are regenerated according to the probability of generation.
    - The higher the probability, the greater the probability of generation. After such iterations, all the particles will slowly converge together, and the possibly exact position of the robot will be calculated.

Some details of the AMCL

❖ Initialization:

❑ Build a hypothetical collection of particles that are evenly distributed across all possible positions and orientations.

❑ Each particle represents a possible location for the robot.

❖ Motion updates:

Estimate Robot's Position

$$\Rightarrow \hat{x} = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i}$$

❑ As the robot moves, we update the positions of these particles.

❑ We estimate the possible position of each particle based on the robot's motion model and the robot's control inputs (such as speed and rotation angle).

❖ Sensor updates:

❑ When the new observation was done, we update the particle weights.

❑ We compare the robot's observations (such as data from a laser scanner) with each particle's predicted observations, and then adjust the particle's weight based on the comparison.

❑ The higher the weight of a particle, the more consistent the position represented by the particle is with the observed data.

❖ Resampling:

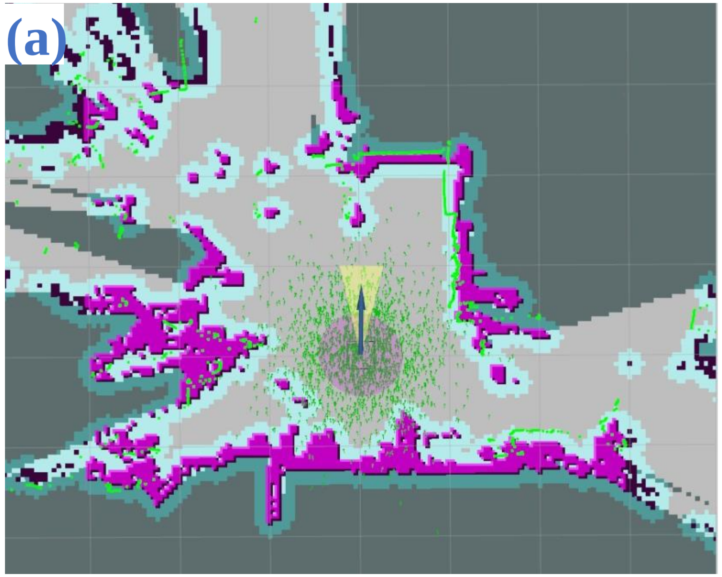
❑ Creating a new collection of particles. New particles will be selected from the old particle set, with a probability of selection proportional to the particle's weight.

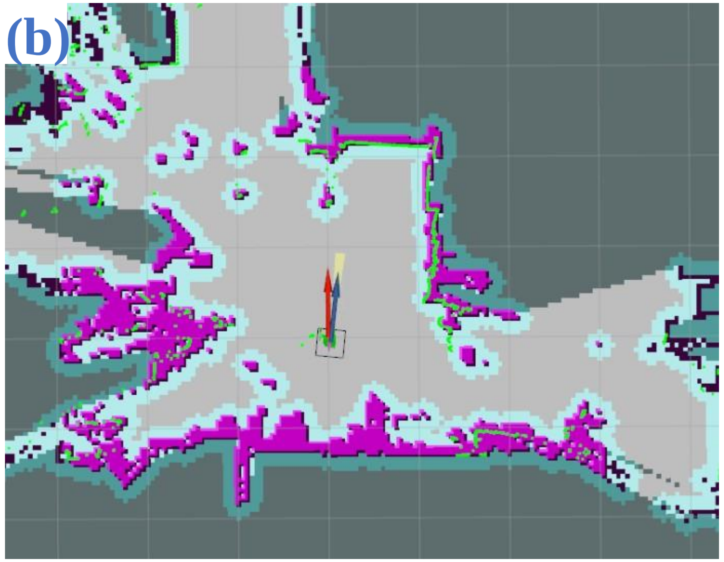
- This allows particles with lower weights (that do not match the observed data) to be excluded, while particles with higher weights are selected multiple times, thus concentrating the particles on locations where the robot is likely to be.

❖ Calculate ESS (Effective Sample Size)

❑ Evaluating the uncertainty of estimated positions and the effectiveness of particles according to this metric for dynamic adjustment of the number of particles

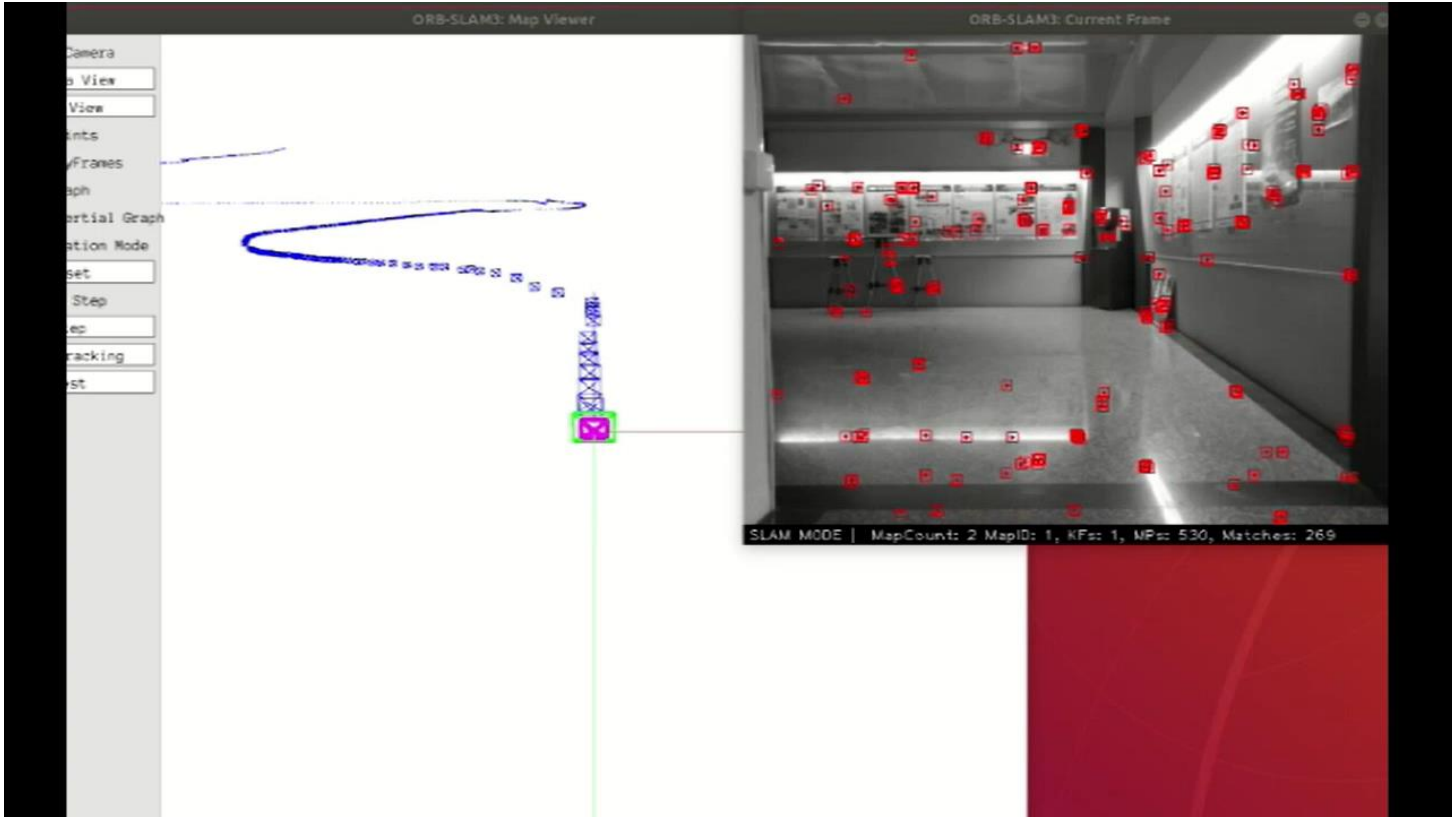
$$ESS = \frac{1}{\sum_{i=1}^{N} {w_i}^2}$$





3D LiDar SLAM

ORB-SLAM


ORB-SLAM3: Map Viewer
ORB-SLAM3: Current Frame
SLAM MODE | MapCount: 2 MapID: 1, KFs: 1, MPs: 530, Matches: 269

Practice Arrangement

Practice 1 - Controlling the robot using a keyboard

- **Introducing launch.py**
  - Launch.py is a command-line tool in ROS 2 used for launching nodes and composite launch files.
  - It offers a streamlined approach to starting and managing ROS 2 applications.
- Key Features:
  - Node Launching: Initiate/ startup multiple ROS 2 nodes with a single launch file, simplifying application startup.
  - Composite Launch Files: Combine multiple launch files to manage complicated application configurations effectively.
  - Centralized Configuration: Launch files allow for the central management of node startup parameters, namespaces, remappings, and other information.
  - Convenience: Provides a convenient way to organize and execute ROS 2 applications, enhancing development and maintenance efficiency.
- Example: ros2 launch [my_package] [my_launch_file.py]

Practice 1 - Controlling the robot using a keyboard

❖ Step 1: Run the launch file to start the motion-related nodes for the robot.

- ❑ Keyboard shortcut Ctrl+Alt+T: Create a new terminal.[Terminal A]
- ❑ Enter "ros2 launch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch.py" to launch the launch file.

❖ What inside turn_on_wheeltec_robot.launch.py?

- ❑ carto_slam_dec: Declares a launch argument for Cartographer SLAM.
- ❑ wheeltec_robot: Launches nodes related to robot.
- ❑ base_to_link: Configures the transformation between 'base' and 'link'.
- ❑ base_to_gyro: Configures the transformation between 'base' and 'gyro'.
- ❑ joint_state_publisher_node: Launches the joint state publisher node.
- ❑ choose_car: Includes configurations for choosing the car.
- ❑ robot_ekf: Launches the robot EKF node for sensor fusion.

Practice 1 - Controlling the robot using a keyboard

❖ turn_on_wheeltec_robot.launch.py

❖ launch_ros.actions.Node is an action class in ROS 2 used to launch a ROS node.

❑ With this action, a ROS node can be launched in ROS 2 to perform specific tasks.

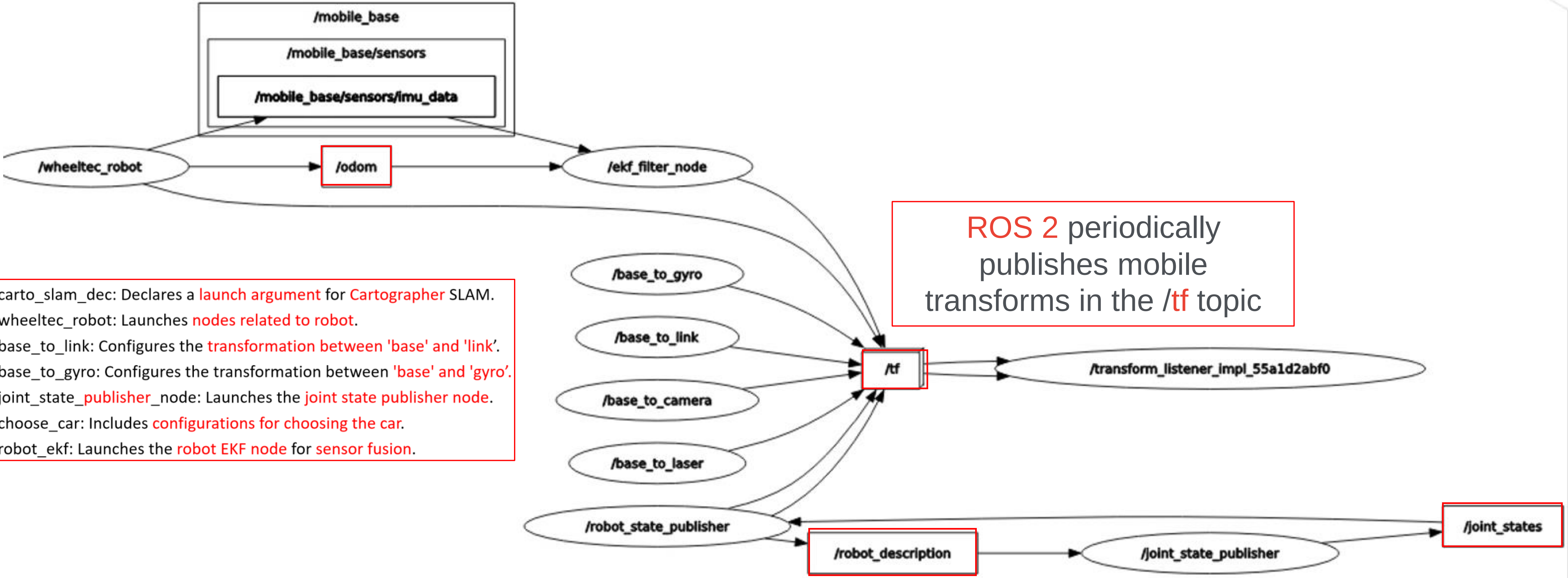
- This action allows setting parameters such as package name, executable name, node name, parameters, remappings, etc., for flexible configuration.

❖ LaunchDescription : Create a LaunchDescription object to define the launch configuration.

```
base_to_link = launch_ros.actions.Node(
        package='tf2_ros', # Use functionality from the tf2_ros package
        executable='static_transform_publisher', # Execute the static_transf
        name='base_to_link', # Node name is base_to_gyro
        arguments=['0', '0', '0','0', '0','0','base_footprint','base_link'],
)
base_to_gyro = launch_ros.actions.Node(
        package='tf2_ros',
        executable='static_transform_publisher',
        name='base_to_gyro',
        arguments=['0', '0', '0','0', '0','0','base_footprint','gyro_link'],
)

joint_state_publisher_node = launch_ros.actions.Node(
        package='joint_state_publisher',
        executable='joint_state_publisher',
        name='joint_state_publisher',
)

ld = LaunchDescription()  # Create a LaunchDescription object to define the

ld.add_action(carto_slam_dec)
ld.add_action(wheeltec_robot)
ld.add_action(base_to_link)
ld.add_action(base_to_gyro)
ld.add_action(joint_state_publisher_node)
ld.add_action(choose_car)
ld.add_action(robot_ekf)
```

Practice 1 - Controlling the robot using a keyboard

❖ Enter“rqt_graph” to visualize the relationships between nodes.



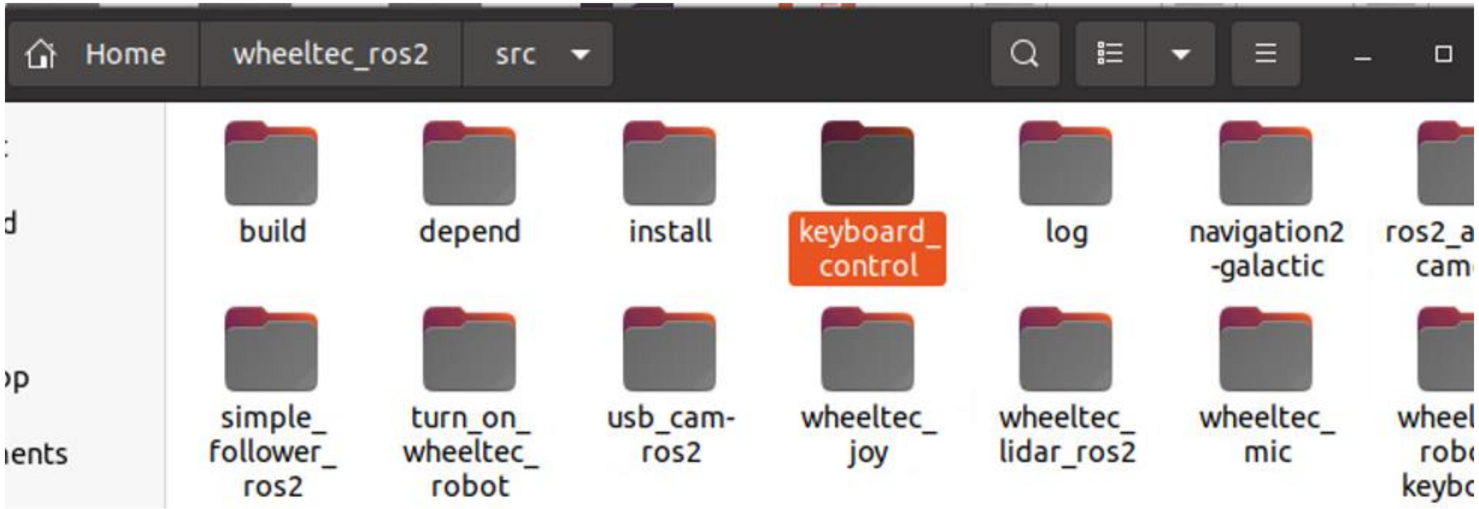
- carto_slam_dec: Declares a launch argument for Cartographer SLAM.
- wheeltec_robot: Launches nodes related to robot.
- base_to_link: Configures the transformation between 'base' and 'link'.
- base_to_gyro: Configures the transformation between 'base' and 'gyro'.
- joint_state_publisher_node: Launches the joint state publisher node.
- choose_car: Includes configurations for choosing the car.
- robot_ekf: Launches the robot EKF node for sensor fusion.

Practice 1 - Controlling the robot using a keyboard

❖ **Step 2: Create a new node to control the robot.**

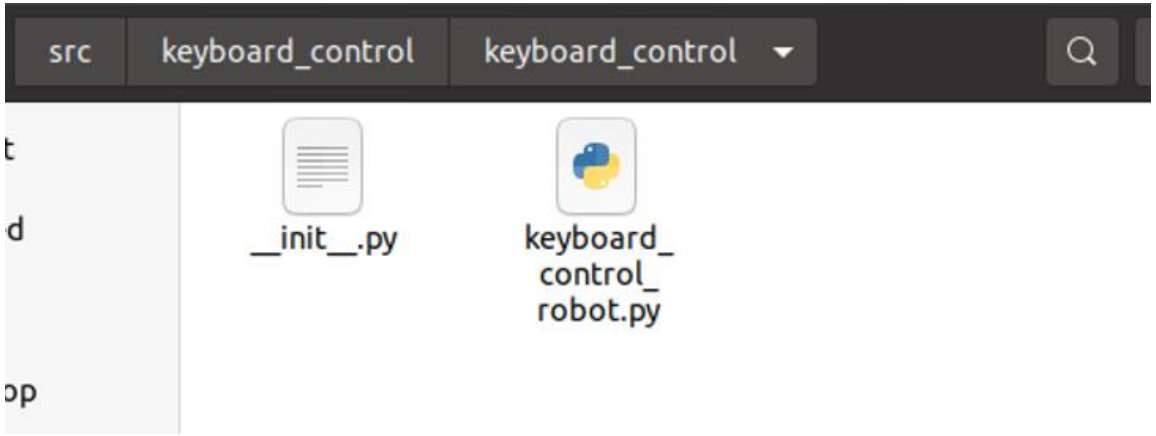
❑How to create the new node?

- Usage Ctrl+Alt+T: Create a new terminal. [Terminal B]
- Enter “cd ~/wheeltec_ros2/src” :This command changes the current directory to the 'wheeltec_ros2/src' directory located in the user's home directory .
- Enter “ros2 pkg create --build-type ament_python --node-name keyboard_control_robot keyboard_control” to create a new node.



Practice 1 - Controlling the robot using a keyboard

- ros2 pkg create --build-type ament_python --node-name [node name] [package name]
  - The command is used to create a new Python node package in ROS 2.
    - By specifying the --build-type ament_python parameter, we instruct ROS 2 to use the Ament build system to build this package.
  - Additionally, the --node-name parameter allows us to specify the name of the node.
    - Upon executing this command, ROS 2 will create a new package in the current directory and generate a basic Python node within it, which can be further developed and extended by the user.



Practice 1 - Controlling the robot using a keyboard

❖ Step 3: Coding the node program.

- ❑You can find code in “wheeltec_ros2/src/keyboard_control/keyboard_control”
- ❑ Importing related necessary resources:

```
#!/usr/bin/env python
# coding=utf-8

# Import necessary modules for ROS 2 communication
import os  # Operating system module
import select  # I/O multiplexing module
import sys  # System-specific parameters and functions module
import rclpy  # ROS 2 client library

# Import message types and Quality of Service (QoS) profile for ROS 2
from geometry_msgs.msg import Twist  # ROS 2 message type for robot velocity
from rclpy.qos import QoSProfile  # Quality of Service profile for ROS 2

# Import modules for controlling terminal I/O settings
import tty  # Terminal control module
```

Practice 1 - Controlling the robot using a keyboard

- Define a function to obtain information about user keyboard input:

```
def get_key():
    """
    Function to get a key press from the user.
    Returns:
        str: Key pressed by the user.
    Explanation:
        - tty.setraw(sys.stdin.fileno()): Set the terminal to raw mode, disabling line buffering.
        - select.select([sys.stdin], [], [], 0.1): Monitor the standard input (keyboard) for any activity
          with a timeout of 0.1 seconds.
        - if rlist: Check if there's any activity on the standard input.
        - key = sys.stdin.read(1): Read a single character from the standard input.
        - return key: Return the key pressed by the user.
    """
    tty.setraw(sys.stdin.fileno())
    rlist, _, _ = select.select([sys.stdin], [], [], 0.1)
    if rlist:
        key = sys.stdin.read(1)
    else:
        key = ''
    return key
```

Practice 1 - Controlling the robot using a keyboard

- Coding the ***main*** program.
  - rclpy.init(): Initialize the ROS 2 client library.
  - qos =QoSProfile(depth=10): Define a Quality of Service profile with a queue depth of 10.
    - This sets how messages are queued in the system.
  - node=rclpy.create_node('wheeltec_keyboard’):
    - Create a ROS 2 node with the name 'wheeltec_keyboard’.
  - pub = node.create_publisher(Twist, 'cmd_vel', qos): Create a publisher that publishes messages of type Twist to the topic 'cmd_vel'. The publisher will use the specified QoS profile.
  - Twist: This is a message type in ROS 2 used to represent velocity commands.
    - It consists of linear and angular velocity components.
    - twist.linear.x = [The speed at which the robot moves along the X-axis.]
    - twist.linear.y = [The speed at which the robot moves along the Y-axis.]
    - twist.linear.z = 0.0
    - twist.angular.x = 0.0
    - twist.angular.y = 0.0
    - twist.angular.z = [The Angular velocity at which the robot moves along the RX-axis.]

Practice 1 - Controlling the robot using a keyboard

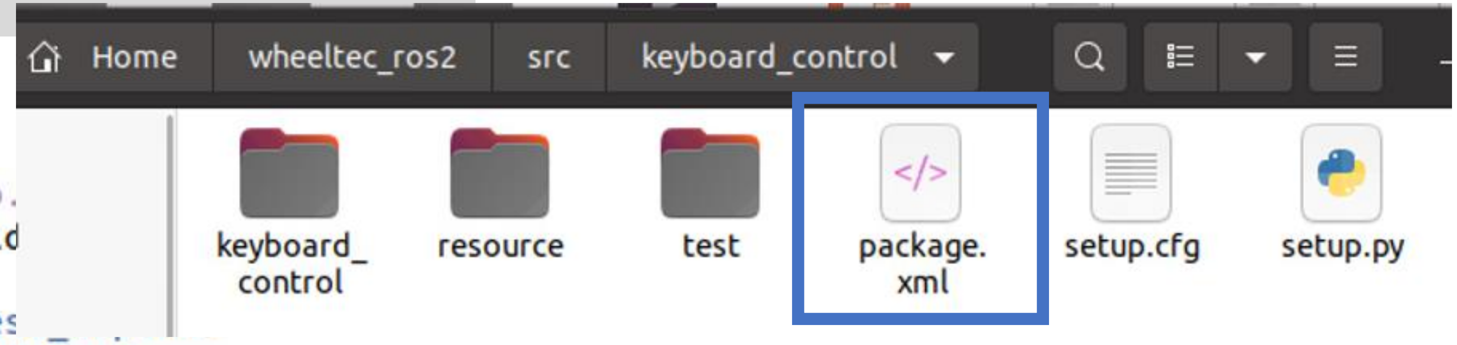
You can modify the program to perform more robot functionalities.

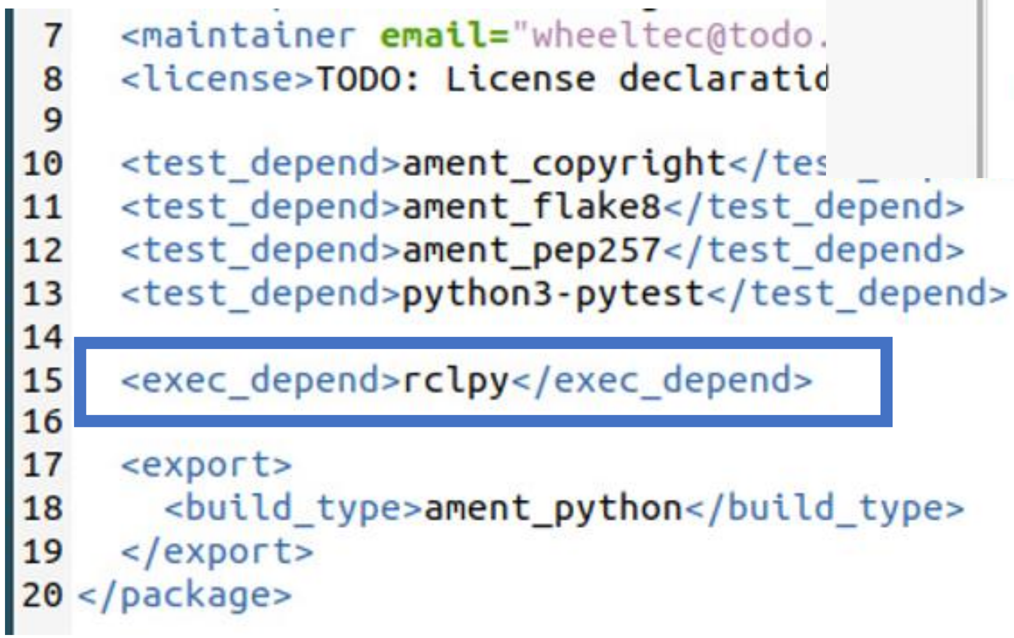
```
def main():
   rclpy.init()
   qos = QoSProfile(depth=10)
   node = rclpy.create_node('wheeltec_keyboard')
   pub = node.create_publisher(Twist, 'cmd_vel', qos)
   try:
      while(1):
         key = get_key()
         twist = Twist()
         if (key == 'w’): #The user presses the 'w' key, robot go forward.
            twist.linear.x = 0.15; twist.linear.y = 0.0;twist.linear.z = 0.0
            twist.angular.x = 0.0;twist.angular.y = 0.0;twist.angular.z = 0.0
         elif (key == 's’): #The user presses the 'w' key, robot go backward.
            twist.linear.x = -0.15;twist.linear.y = 0.0;twist.linear.z = 0.0
            twist.angular.x = 0.0;twist.angular.y = 0.0;twist.angular.z = 0.0
         elif (key == '\x03’): #The user presses the ‘Ctrl+C' key, stop program.
            break
         else :
            twist.linear.x = 0.0;twist.linear.y = 0.0;twist.linear.z = 0.0
            twist.angular.x = 0.0;twist.angular.y = 0.0;twist.angular.z = 0.0
         pub.publish(twist)
   except Exception as e:
      print('error!!’)
if __name__ == '__main__':
     main()
```

Practice 1 - Controlling the robot using a keyboard

❖ Step 4: Edit the package.xml file to add the dependency on rclpy.

```
…
<exec_depend>rclpy</exec_depend>
…
```





Practice 1 - Controlling the robot using a keyboard

❖ **Step 5: Go back to the Workspace directory and compile the program.**

❑Enter "cd ../" :Change directory to the previous directory. [Terminal B]

❑Enter "colcon build --packages-select keyboard_control" to compile the program.

❖**Step 6: Source**

❑Enter " source install/setup.bash "

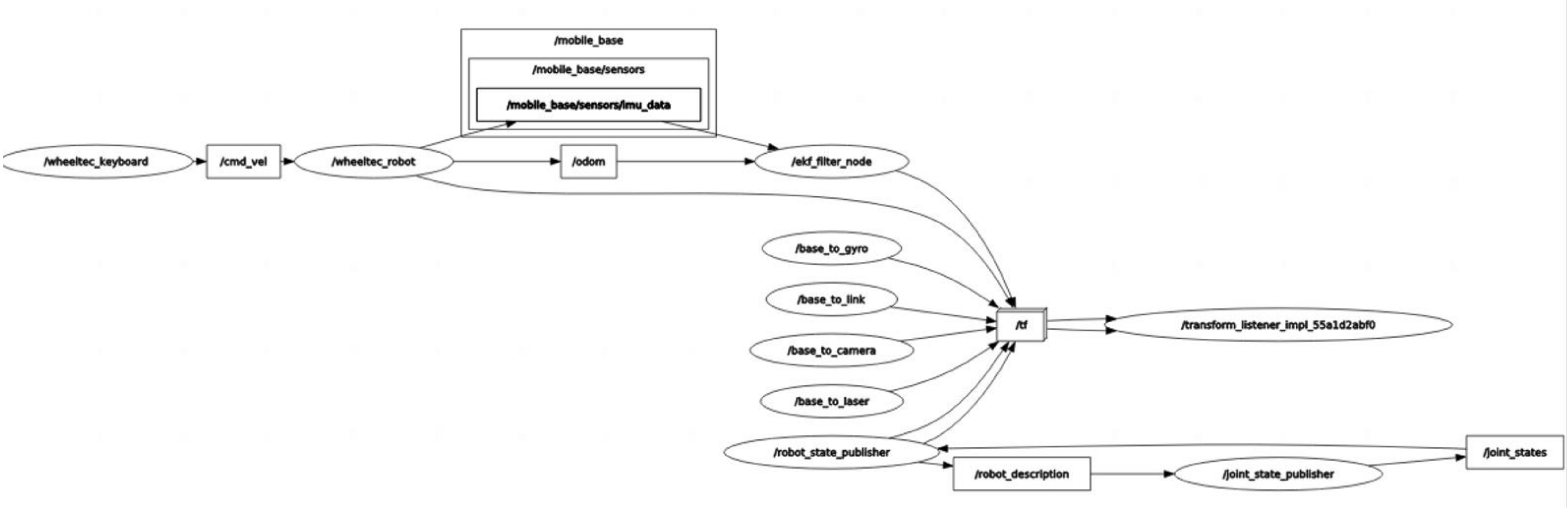
❑The "source" command is used in ROS 2 to load ROS 2 environment variables into the current shell session. When you run this command, it executes the necessary scripts to set up the environment for working with ROS 2, including configuring paths and variables required by ROS 2 tools and packages. This ensures that ROS 2 commands and tools are available and properly configured for use within the current shell session.

❖**Step 7: Run the program**

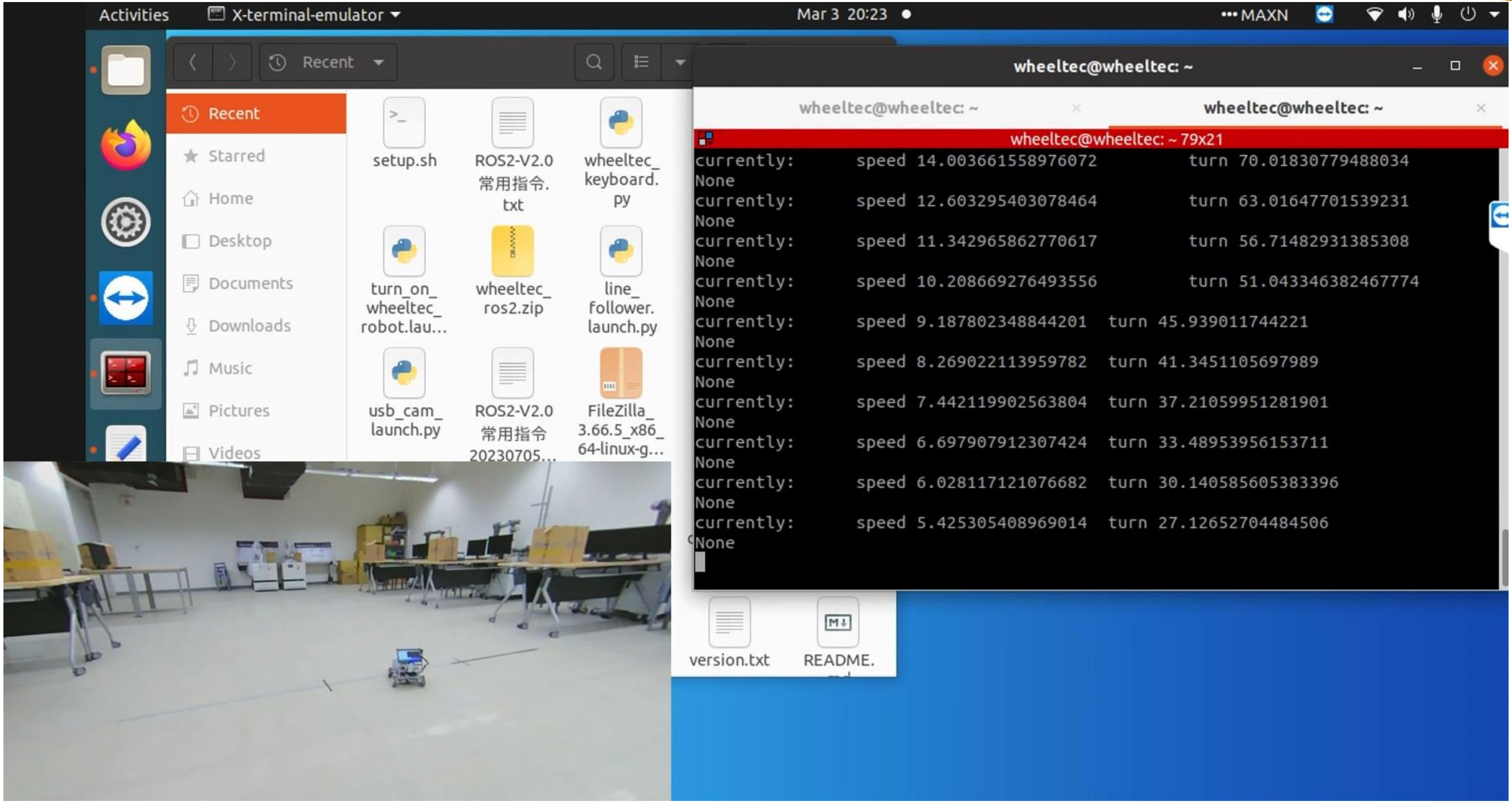
❑Enter "ros2 run keyboard_control keyboard_control_robot "

Practice 1 - Controlling the robot using a keyboard

❖ Enter“rqt_graph” to visualize the relationships between nodes.



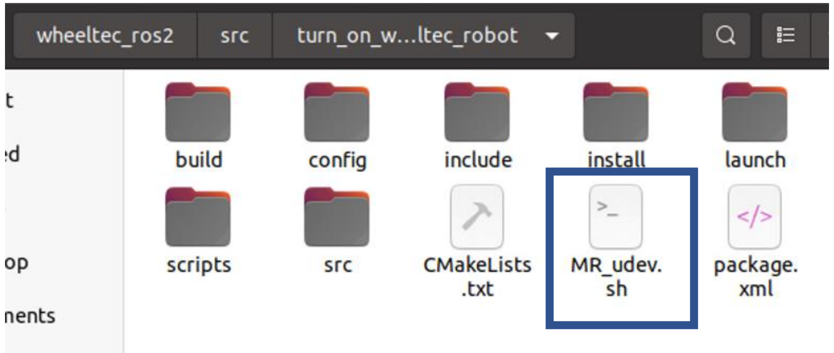
Practice 1 - Controlling the robot using a keyboard



Practice 2 - SLAM

❖ **Step 1: Setting up the information for the lidar.**

- ❑ Downland the code from NTU cool [MR_udev.sh].
- ❑ Put it into the /home/wheeltec/wheeltec_ros2/src/turn_on_wheeltec_robot.
- ❑ Run the following code. (Password :dongguan)
  - ```
    cd ~/wheeltec_ros2/src/turn_on_wheeltec_robot/
    ```
  - ```
    sudo sh MR_udev.sh
    ```



Practice 2 - SLAM

```
# CP2102 serial number 0001 Set the alias to wheeltec_lidar
echo  'KERNEL=="ttyUSB*", ATTRS{idVendor}=="10c4", ATTRS{idProduct}=="ea60",ATTRS{serial}=="0001", MODE:="0777",
GROUP:="dialout", SYMLINK+="wheeltec_laser"' >/etc/udev/rules.d/wheeltec_lidar.rules
# CH9102, with corresponding driver installed Serial number 0001 Set the alias to wheeltec_lidar
echo  'KERNEL=="ttyCH343USB*", ATTRS{idVendor}=="1a86", ATTRS{idProduct}=="55d4",ATTRS{serial}=="0001", MODE:="0777",
GROUP:="dialout", SYMLINK+="wheeltec_laser"' >/etc/udev/rules.d/wheeltec_lidar2.rules
# CH9102, with no corresponding driver installed Serial number 0001 Set the alias to wheeltec_lidar
echo  'KERNEL=="ttyACM*", ATTRS{idVendor}=="1a86", ATTRS{idProduct}=="55d4",ATTRS{serial}=="0001", MODE:="0777",
GROUP:="dialout", SYMLINK+="wheeltec_laser"' >/etc/udev/rules.d/wheeltec_lidar3.rules
```

❖This code segment is for setting up device aliases using udev rules.

- ❑When a serial device with a kernel name (KERNEL) matching ttyCH343USB* and having idVendor as 1a86, idProduct as 55d4, and serial number (serial) as 0001 is detected,
- ❑The permission for this serial device is set to 0777, with group ownership set to dialout, and a symbolic link named wheeltec_laser is created.
- ❑Finally, this rule is written into the file /etc/udev/rules.d/wheeltec_lidar2.rules.

※udev is a device manager for the Linux kernel. It dynamically creates or removes device nodes in the /dev directory, or changes their permissions and ownership, based on rules specified in udev configuration files.udev is responsible for handling the device nodes that represent hardware devices, such as USB devices, serial ports, network interfaces, and more. It allows for the automatic configuration of devices as they are detected or connected to the system, providing a flexible and predictable way to manage hardware devices in a Linux system.

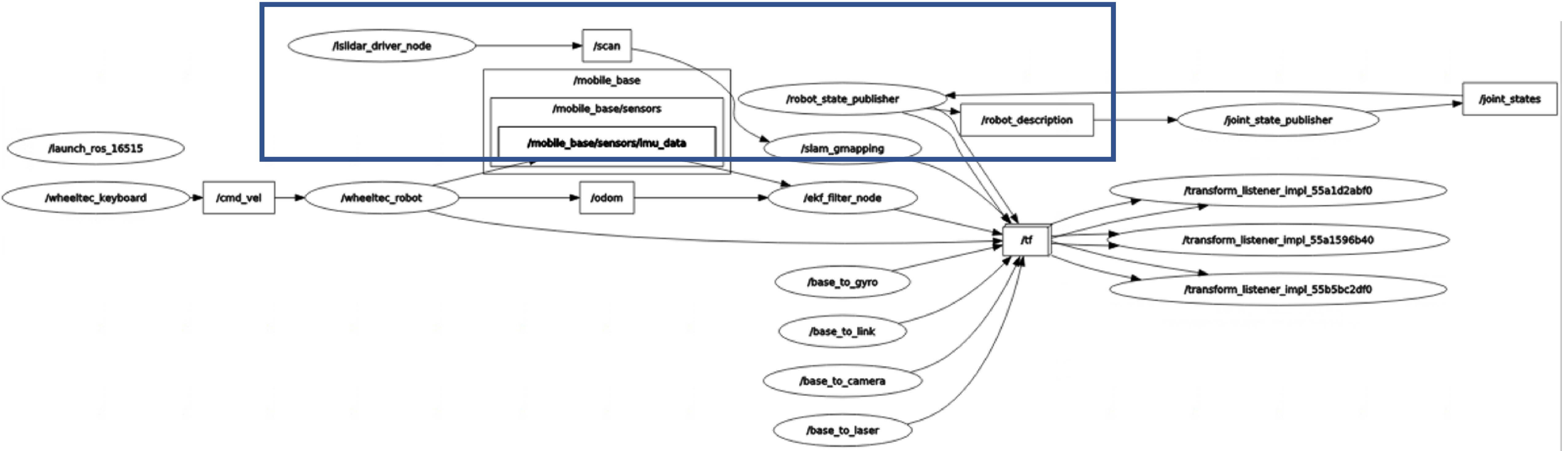
Practice 2 - SLAM

- **Step 2: Use the following commands to start SLAM.**
  - Keyboard shortcut Ctrl+Alt+T: Create a new terminal. [Terminal C]
  - Enter “ros2 launch slam_gmapping slam_gmapping.launch.py” to launch the launch file.
- IncludeLaunchDescription is an action in ROS 2 Launch used to include the contents of another launch file within the current launch file. This allows for organizing and combining the startup of multiple subsystems or nodes within a single launch file. By using IncludeLaunchDescription, code can be modularized and made more reusable, while also making the overall startup configuration clearer.

```python
def generate_launch_description():
    use_sim_time = launch.substitutions.LaunchConfiguration('use_sim_time', default='false')

    bringup_dir = get_package_share_directory('turn_on_wheeltec_robot')
    launch_dir = os.path.join(bringup_dir, 'launch')

    wheeltec_lidar = IncludeLaunchDescription(
            PythonLaunchDescriptionSource(os.path.join(launch_dir, 'wheeltec_lidar.launch.py')),
    )
    wheeltec_robot = IncludeLaunchDescription(
            PythonLaunchDescriptionSource(os.path.join(launch_dir, 'turn_on_wheeltec_robot.launch.py')),
    )
    return LaunchDescription([
        wheeltec_robot,wheeltec_lidar,
        SetEnvironmentVariable('RCUTILS_LOGGING_BUFFERED_STREAM', '1'),
        launch_ros.actions.Node(
            package='slam_gmapping', executable='slam_gmapping', output='screen', parameters=[{'use_sim_time':use_sim_time}]),
    ])
```

Practice 2 - SLAM

❖ Enter“rqt_graph” to visualize the relationships between nodes.

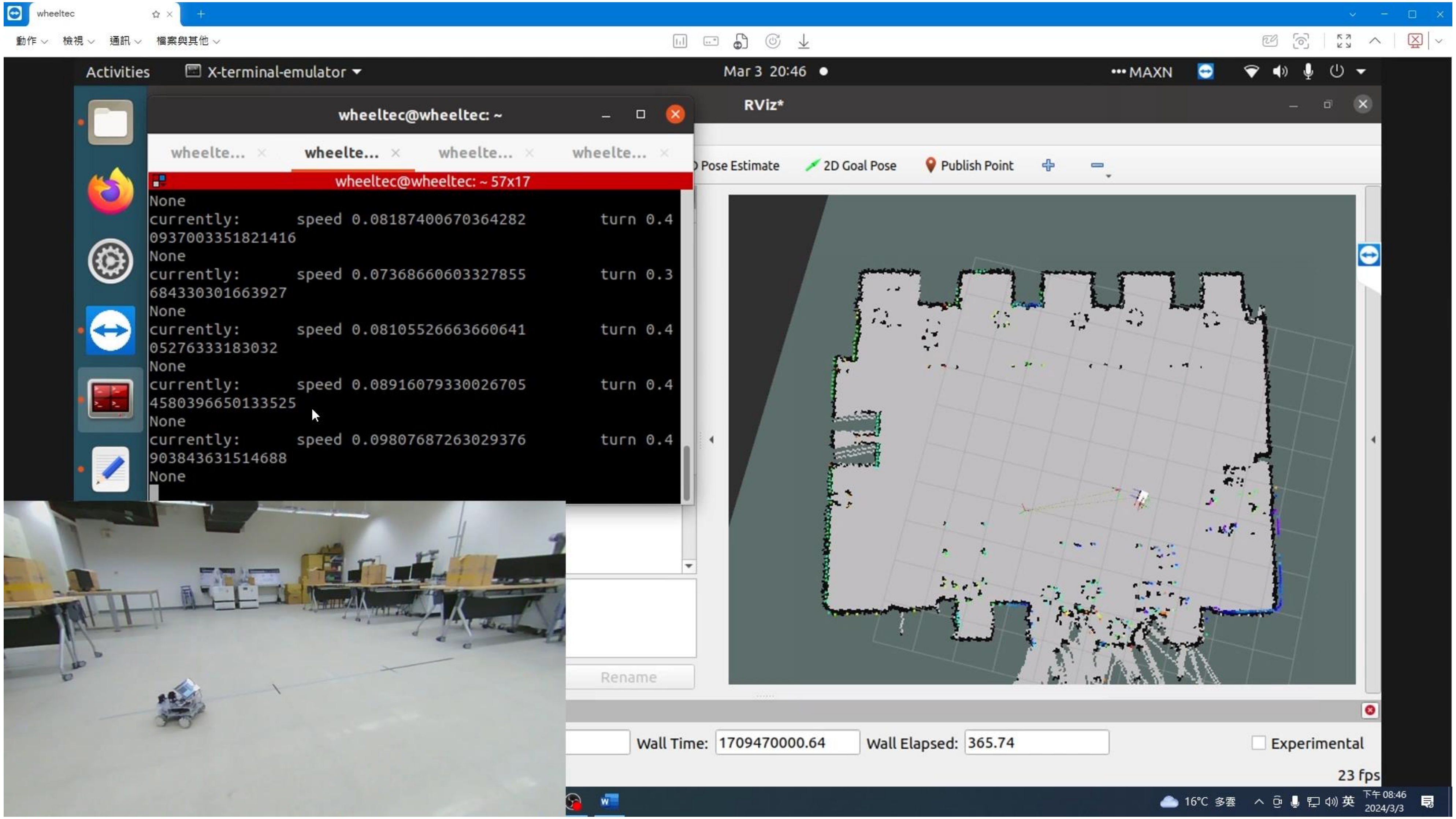


Practice 2 - SLAM

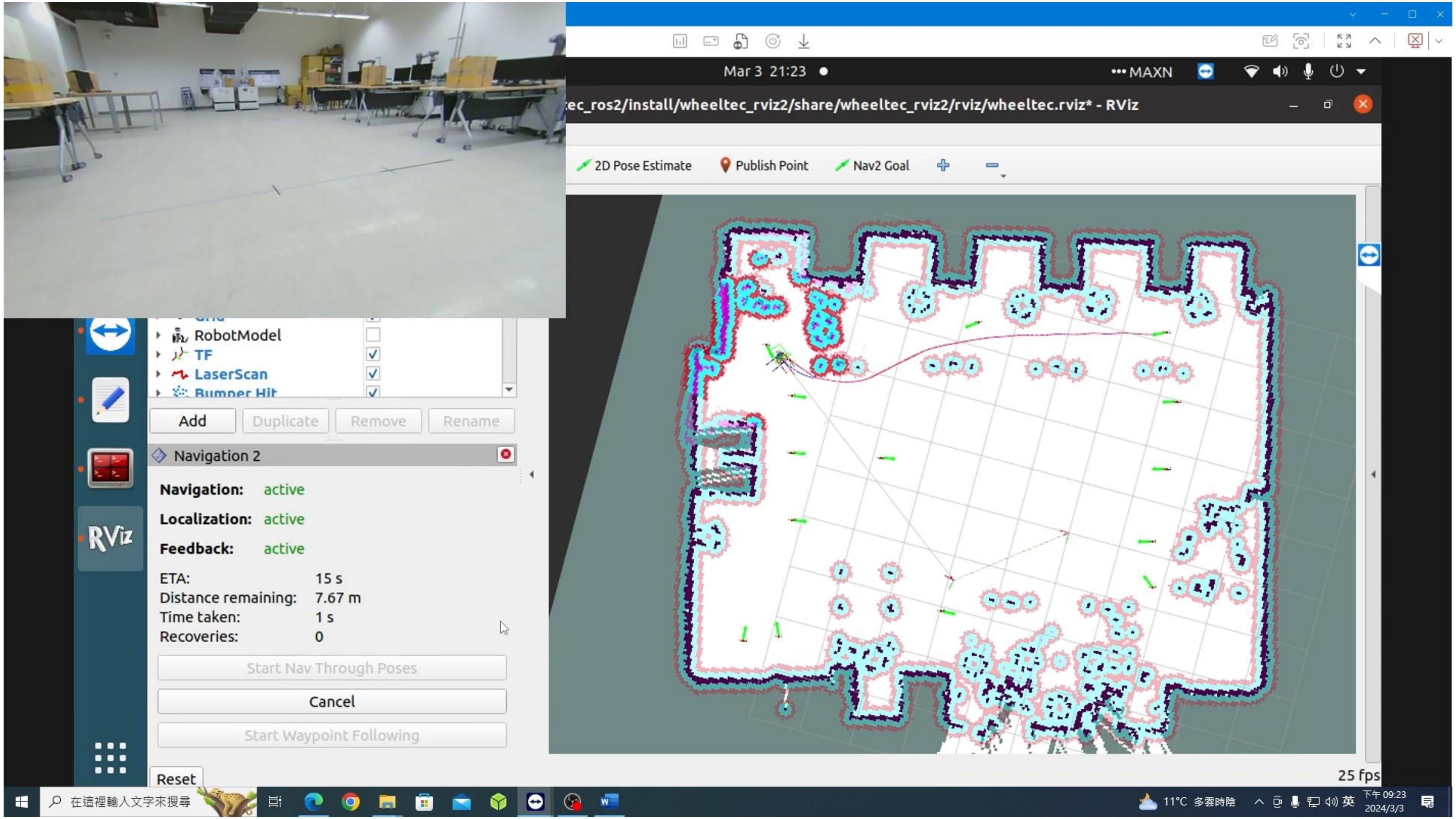
- **Step 3: Open RViz to visualize the SLAM process.**
  - Keyboard shortcut Ctrl+Shift+T: Create a new terminal. [Terminal D]
  - Enter “rviz2” to open rviz.
  - Using keyboard to control the robot.
- When building the map, it is necessary to slow down the AMR speed to achieve optimal results.
- **Step 4: Save the map**
  - Enter “ ros2 launch wheeltec_nav2 save_map.launch.py” :After mapping is completed, use this command to save the map for future use.
  - You can find your map at following path:

    /home/wheeltec/wheeltec_ros2/src/wheeltec_robot_nav2/map

Practice 2 - SLAM



Future



Mar 3 21:23
MAXN
ec_ros2/install/wheeltec_rviz2/share/wheeltec_rviz2/rviz/wheeltec.rviz* - RViz
2D Pose Estimate
Publish Point
Nav2 Goal
RobotModel
TF
LaserScan
Bumper Hit
Add
Duplicate
Remove
Rename
Navigation 2
Navigation: active
Localization: active
Feedback: active
ETA: 15 s
Distance remaining: 7.67 m
Time taken: 1 s
Recoveries: 0
Start Nav Through Poses
Cancel
Start Waypoint Following
Reset
25 fps
在這裡輸入文字來搜尋
11°C 多雲時陰
下午 09:23
2024/3/3

Thank you