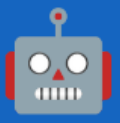

คู่มือการเรียนรู้
ROS 2 Galactic
ด้วย Turtlesim สู่ LotusBot

ระดับ	จำนวน	สถานที่
ม.4-6	**7 บท**	**RAIL Center**
(RAIL)	+ Lab & Quiz	kls.ac.th

บทนำ: การเตรียมความพร้อม

ก่อนเริ่มต้น ผู้เรียนต้องตรวจสอบสภาพแวดล้อมบน Ubuntu 20.04 ให้พร้อมก่อนเสมอ

1. Source Environment

ทุกครั้งที่เปิด Terminal ใหม่ ต้อง source ก่อน:

```
# Source ทุกครั้งที่เปิด Terminal ใหม่
source /opt/ros/galactic/setup.bash

# แนะนำ: เพิ่มใน .bashrc เพื่อให้อัตโนมัติ
echo "source /opt/ros/galactic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

2. ตรวจสอบสถานะ

```
# ตรวจสอบว่า ROS_DISTRO=galactic
printenv | grep ROS

# ควรเห็นผลลัพธ์: ROS_DISTRO=galactic
```

3. ทดสอบการสื่อสาร

```
# Terminal 1: ส่งข้อมูล (Talker)
ros2 run demo_nodes_cpp talker

# Terminal 2: รับข้อมูล (Listener)
ros2 run demo_nodes_py listener
```

> 💡 **Workflow หลัก**
>
> - Source → Build → Launch → Monitor → Debug
> - จำลำดับนี้ไว้เสมอ! ขาดขั้นตอนใดขั้นตอนหนึ่งจะทำให้เกิด error ได้

บทที่ 1

Nodes และโครงสร้างระบบ

ROS 2 Nodes & Graph Architecture

ทฤษฎี: Node คืออะไร?

ROS 2 ทำงานแบบ **Distributed System** โดยมี Node เป็นหน่วยประมวลผลย่อยๆ ที่ทำงานเฉพาะทาง แต่ละ Node เชื่อมต่อกันผ่าน Graph

Node ตัวอย่าง	หน้าที่	ใน LotusBot
`turtlesim_node`	แสดงกราฟิก Turtle	camera_node (ESP32-CAM)
`turtle_teleop_key`	รับ Input จาก Keyboard	teleop_node (รับ cmd จาก PC)
`micro_ros_agent`	เชื่อม ESP32 กับ ROS2	ใน LotusBot จริง! (USB Serial)

คำสั่งพื้นฐาน

```
# รัน Turtlesim
ros2 run turtlesim turtlesim_node

# ดู Node ที่กำลังรันอยู่ทั้งหมด
ros2 node list

# ดูข้อมูลละเอียดของ Node
ros2 node info /turtlesim

# ปิด Node
ros2 node kill /turtlesim
```

Lab 1: ทดลองสร้างและตรวจสอบ Node

วัตถุประสงค์:

1. เปิด Turtlesim node และสังเกตหน้าต่างที่ปรากฏ
2. ใช้คำสั่ง ros2 node list และ ros2 node info
3. เปิด turtle_teleop_key และทดลองควบคุม
4. วาดแผนภาพ Node Graph ด้วยมือ

Quiz ทบทวนบทที่ 1

ข้อ	คำถาม	คำตอบ
1	Node ใน ROS 2 คืออะไร?	หน่วยประมวลผลย่อยที่ทำงานเฉพาะทาง
2	คำสั่งดู Node ที่กำลังรันอยู่คืออะไร?	ros2 node list
3	ใน LotusBot micro_ros_agent ทำหน้าที่อะไร?	เชื่อม ESP32 กับ ROS2 ผ่าน USB Serial

บทที่ 2

Topics และ Messages

Publish / Subscribe Communication

ทฤษฎี: Topics ทำงานอย่างไร?

Topics ใช้รูปแบบ **Publisher/Subscriber** — ผู้ส่ง (Publisher) ส่งข้อมูลเข้า Topic โดยไม่สนใจว่าใครรับ ผู้รับ (Subscriber) รับข้อมูลโดยไม่สนใจว่าใครส่ง เหมาะกับข้อมูลต่อเนื่อง เช่น ค่า Sensor หรือคำสั่งความเร็ว

🔗 **การเชื่อมโยงกับ LotusBot**

- Topic /cmd_vel → ส่งคำสั่งความเร็วไปยัง ESP32 ผ่าน micro-ROS
- Topic /scan → รับข้อมูล LiDAR จาก YDLidar X3 (12Hz)
- Topic /odom → รับค่า Odometry จาก encoder ล้อ

คำสั่งตรวจสอบ Topics

```
# แสดง Topic ทั้งหมด
ros2 topic list

# Monitor ข้อมูล Real-time
ros2 topic echo /turtle1/pose

# ดูความถี่การส่งข้อมูล (Hz)
ros2 topic hz /turtle1/pose

# ดูชนิดข้อความ (Message Type)
ros2 topic info /turtle1/cmd_vel
```

การส่งคำสั่งด้วย topic pub

```
# ส่งครั้งเดียว (--once)
ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist \
  "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"

# ส่งต่อเนื่องที่ 1 Hz
ros2 topic pub --rate 1 /turtle1/cmd_vel geometry_msgs/msg/Twist \
  "{linear: {x: 1.5, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
```

Lab 2: สังเกตการส่งข้อมูลผ่าน Topic

วัตถุประสงค์:

5. เปิด Turtlesim และ echo /turtle1/pose พร้อมกัน
6. ส่ง cmd_vel ให้เต่าวิ่งวนเป็นวงกลม
7. ปรับค่า linear.x และ angular.z แล้วสังเกตเส้นทาง
8. บันทึกค่าที่ทำให้เต่าวิ่งเป็นรูปสี่เหลี่ยม

Quiz ทบทวนบทที่ 2

ข้อ	คำถาม	คำตอบ
1	รูปแบบการสื่อสารของ Topics คืออะไร?	Publisher/Subscriber
2	Topic ใดที่ใช้ส่งคำสั่งความเร็วให้ Turtle?	/turtle1/cmd_vel
3	ใน LotusBot ข้อมูล LiDAR อยู่ที่ Topic ใด?	/scan (LaserScan message)
4	QoS Reliable ต่างจาก Best Effort อย่างไร?	Reliable เน้นความถูกต้อง, Best Effort เน้นความเร็ว

บทที่ 3

Services และ Parameters

Request/Response & Configuration

Services: Request / Response

Services ใช้รูปแบบ **Request/Response** เหมาะกับงานที่ทำครั้งเดียวจบ เช่น รีเซ็ตหน้าจอ หรือสั่งให้หุ่นยนต์ไปตำแหน่งที่กำหนด

```
# ดู Service ทั้งหมด
ros2 service list

# รีเซ็ต Turtlesim (ล้างหน้าจอ)
ros2 service call /reset std_srvs/srv/Empty

# Spawn เต่าตัวใหม่
ros2 service call /spawn turtlesim/srv/Spawn \
  "{x: 2.0, y: 2.0, theta: 0.0, name: 'turtle2'}"

# ดูชนิดของ Service
ros2 service type /reset
```

Parameters: ค่าปรับแต่งโหนด

```
# ดู Parameters ทั้งหมดของ Node
ros2 param list /turtlesim

# เปลี่ยนสีพื้นหลัง
ros2 param set /turtlesim background_r 150
ros2 param set /turtlesim background_g 50
ros2 param set /turtlesim background_b 200

# อัปเดตหน้าจอ (ต้องเรียก /clear)
ros2 service call /clear std_srvs/srv/Empty

# บันทึก Parameters ลงไฟล์
ros2 param dump /turtlesim
```

🔗 **การเชื่อมโยงกับ LotusBot**

- Parameter ใน LotusBot: scan_frequency (ความถี่ LiDAR), robot_radius (รัศมีหุ่นยนต์)
- Service: /set_pose ใช้ Reset ตำแหน่งหุ่นยนต์ใน Simulator

- Parameters สามารถโหลดจากไฟล์ .yaml ได้ สะดวกในการตั้งค่า

Lab 3: ทดลอง Services และ Parameters
วัตถุประสงค์:

9. ลอง Spawn เต่า 3 ตัวในตำแหน่งต่างๆ
10. เปลี่ยนสีพื้นหลังเป็นสีที่ชอบและถ่ายภาพหน้าจอ
11. ลบเต่าด้วย Service /kill
12. บันทึก Parameters ด้วย param dump

Quiz ทบทวนบทที่ 3

ข้อ	คำถาม	คำตอบ
1	Services ต่างจาก Topics อย่างไร?	Services เป็น Request/Response ทำครั้งเดียว
2	คำสั่งรีเซ็ต Turtlesim คืออะไร?	ros2 service call /reset std_srvs/srv/Empty
3	ต้องเรียก Service ใดหลังเปลี่ยนสีพื้นหลัง?	/clear

บทที่ 4

การวิเคราะห์ระบบด้วย RQT

Debugging Tools & Visualization

เครื่องมือ GUI สำหรับ Debug

เครื่องมือ	หน้าที่	ใช้เมื่อไร?
`rqt_graph`	แสดงแผนผัง Node/Topic ทั้งหมด	ตอน Debug การเชื่อมต่อ
`rqt_console`	แสดง Log, Warning, Error	ตอน Node มีปัญหา
`rqt_plot`	Plot กราฟค่า Topic Real-time	วิเคราะห์ค่า Sensor
`rqt_reconfigure`	ปรับ Parameters แบบ Real-time	Tune ค่าโดยไม่ต้อง Restart
`rviz2`	Visualize 3D: LiDAR, TF, Map	ดูข้อมูล LotusBot LiDAR

การใช้ rqt_graph

```
# เปิด RQT Graph
rqt_graph

# หรือเปิดผ่าน rqt แล้วเลือก Plugins > Introspection > Node Graph
rqt
```

💡 **การอ่าน rqt_graph**

- วงกลม = Node, กล่องสี่เหลี่ยม = Topic
- ลูกศรชี้ = ทิศทางการส่งข้อมูล (Publisher → Subscriber)
- ใน LotusBot: micro_ros_agent จะปรากฏเชื่อมกับ /cmd_vel และ /scan

Lab 4: Debug ด้วย RQT Tools

วัตถุประสงค์:

13. เปิด rqt_graph พร้อม Turtlesim และ teleop
14. ถ่ายภาพ Node Graph และระบุทิศทางข้อมูล
15. ใช้ rqt_plot วาดกราฟ /turtle1/pose/x และ /y
16. ใช้ rqt_console ดู Log ขณะใช้งาน

Quiz ทบทวนบทที่ 4

ข้อ	คำถาม	คำตอบ
1	rqt_graph แสดงข้อมูลอะไร?	แผนผัง Node และ Topic ทั้งหมดในระบบ
2	เครื่องมือใดใช้ Plot กราฟค่า Topic?	rqt_plot
3	rviz2 ใช้ทำอะไรกับ LotusBot?	Visualize ข้อมูล LiDAR และ TF Frame ใน 3D

บทที่ 5

การเขียนโปรแกรม Python

Python Node Programming

สร้าง Workspace และ Package

```
# สร้าง Workspace
mkdir -p ~/ros2_ws/src && cd ~/ros2_ws/src

# สร้าง Python Package
ros2 pkg create --build-type ament_python my_turtle_controller \
  --dependencies rclpy geometry_msgs turtlesim

# Build
cd ~/ros2_ws
colcon build
source install/setup.bash
```

โค้ดที่ 1: วิ่งวนเป็นวงกลม (Basic Publisher)

```
# ~/ros2_ws/src/my_turtle_controller/my_turtle_controller/circle.py
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist

class TurtleCircle(Node):
    def __init__(self):
        super().__init__('turtle_circle')
        self.publisher = self.create_publisher(Twist, '/turtle1/cmd_vel',
10)
        self.timer = self.create_timer(0.5, self.timer_callback)
        self.get_logger().info('TurtleCircle node started!')

    def timer_callback(self):
        msg = Twist()
        msg.linear.x = 2.0     # ความเร็วเดินหน้า (m/s)
        msg.angular.z = 1.0    # ความเร็วหมุน (rad/s)
        self.publisher.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    node = TurtleCircle()
    rclpy.spin(node)
    rclpy.shutdown()
```

โค้ดที่ 2: ติดตามตำแหน่ง (Publisher + Subscriber)

```
# ~/ros2_ws/src/my_turtle_controller/my_turtle_controller/pose_tracker.py
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
from turtlesim.msg import Pose

class PoseTracker(Node):
    def __init__(self):
        super().__init__('pose_tracker')
        self.publisher = self.create_publisher(Twist, '/turtle1/cmd_vel',
10)
        self.subscriber = self.create_subscription(
            Pose, '/turtle1/pose', self.pose_callback, 10)
        self.current_pose = None

    def pose_callback(self, msg):
        self.current_pose = msg
        self.get_logger().info(
            f'Position: x={msg.x:.2f}, y={msg.y:.2f},
theta={msg.theta:.2f}')

        # หยุดถ้าออกนอกขอบเขต
        if msg.x > 9.0 or msg.x < 1.0 or msg.y > 9.0 or msg.y < 1.0:
            stop_msg = Twist()
            self.publisher.publish(stop_msg)
            self.get_logger().warn('Boundary reached! Stopping.')

def main(args=None):
    rclpy.init(args=args)
    node = PoseTracker()
    rclpy.spin(node)
    rclpy.shutdown()
```

โค้ดที่ 3: นำทางหลายจุด (Waypoint Navigator) — เชื่อมกับ LotusBot

```
# ~/ros2_ws/src/my_turtle_controller/my_turtle_controller/waypoint.py
import rclpy, math
from rclpy.node import Node
from geometry_msgs.msg import Twist
from turtlesim.msg import Pose

class WaypointNavigator(Node):
    def __init__(self):
        super().__init__('waypoint_navigator')
```

```
        self.pub = self.create_publisher(Twist, '/turtle1/cmd_vel', 10)
        self.sub = self.create_subscription(Pose, '/turtle1/pose',
                                            self.pose_cb, 10)
        # กำหนด Waypoints (x, y)
        self.waypoints = [(2.0,2.0),(8.0,2.0),(8.0,8.0),(2.0,8.0),(5.0,5.0)]
        self.current_wp = 0
        self.pose = None
        self.timer = self.create_timer(0.1, self.control_loop)

    def pose_cb(self, msg): self.pose = msg

    def control_loop(self):
        if self.pose is None or self.current_wp >= len(self.waypoints):
return
        tx, ty = self.waypoints[self.current_wp]
        dx = tx - self.pose.x; dy = ty - self.pose.y
        dist = math.sqrt(dx**2 + dy**2)
        if dist < 0.3:  # ถึง Waypoint แล้ว
            self.get_logger().info(f'Reached waypoint {self.current_wp}!')
            self.current_wp += 1; return
        angle_to_goal = math.atan2(dy, dx)
        angle_err = angle_to_goal - self.pose.theta
        msg = Twist()
        msg.linear.x  = min(1.5, 0.5 * dist)
        msg.angular.z = 2.0 * angle_err
        self.pub.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    node = WaypointNavigator()
    rclpy.spin(node)
    rclpy.shutdown()
```

🔗 เชื่อมกับ LotusBot จริง

- โค้ด Waypoint Navigator นี้ใช้ได้กับ LotusBot โดยเปลี่ยน Topic จาก /turtle1/cmd_vel เป็น /cmd_vel
- เปลี่ยน Pose จาก turtlesim.msg.Pose เป็น nav_msgs.msg.Odometry
- Logic การควบคุมเหมือนกันทุกอย่าง! นี่คือพื้นฐานของ Navigation Stack

Lab 5: เขียนโปรแกรม Python Node

วัตถุประสงค์:

17. สร้าง Package และ Node วิ่งวนวงกลม Build และรัน
18. แก้โค้ดให้เต่าวิ่งรูปสี่เหลี่ยม (4 waypoints)
19. เพิ่ม Log แสดงค่า Pose ทุก 2 วินาที
20. Challenge: ทำให้เต่าหยุดเมื่อถึง Waypoint สุดท้าย

Quiz ทบทวนบทที่ 5

ข้อ	คำถาม	คำตอบ
1	create_publisher() ต้องระบุอะไรบ้าง?	Message Type, Topic Name, Queue Size
2	create_timer(0.5, callback) หมายความว่าอะไร?	เรียก callback ทุก 0.5 วินาที (2 Hz)
3	ต้องเปลี่ยน Topic ใดเพื่อใช้กับ LotusBot?	/turtle1/cmd_vel → /cmd_vel
4	math.atan2(dy, dx) ใช้คำนวณอะไร?	มุมจากตำแหน่งปัจจุบันไปยัง Goal

บทที่ 6

Actions และระบบพิกัด TF

Actions & Transform Frames

Actions: ภารกิจระยะยาว

Actions เหมาะกับงานที่ใช้เวลานาน เช่น Navigation ไปยังเป้าหมาย โดยมี 3 ส่วนหลัก:

ส่วนประกอบ	คำอธิบาย	ตัวอย่าง LotusBot
Goal	เป้าหมายที่ต้องการ	ไปที่พิกัด x=3.0, y=2.0
Feedback	ความคืบหน้าระหว่างทาง	ระยะทางที่เหลือ: 1.5m
Result	ผลลัพธ์เมื่อทำเสร็จ	Reached goal: True

```
# ส่ง Goal ให้ Turtle หมุน
ros2 action send_goal /turtle1/rotate_absolute \
  turtlesim/action/RotateAbsolute "{theta: 1.57}"

# ดู Action Server ทั้งหมด
ros2 action list

# ดูรายละเอียด Action
ros2 action info /turtle1/rotate_absolute
```

TF: ระบบพิกัด Transform

TF จัดการความสัมพันธ์ระหว่างพิกัดของชิ้นส่วนต่างๆ ของหุ่นยนต์ เช่น **world → base_link → laser_frame**

```
# ดู TF Tree ทั้งหมด
ros2 run tf2_tools view_frames

# Monitor TF Real-time
ros2 run tf2_ros tf2_monitor

# แปลงพิกัดระหว่าง Frame
ros2 run tf2_ros tf2_echo world turtle1
```

🔗 **TF ใน LotusBot**

- Frame หลัก: map → odom → base_link → laser_frame
- YDLidar X3 ติดอยู่ที่ laser_frame ห่างจาก base_link ตามที่กำหนดใน URDF
- SLAM จะสร้าง Transform ระหว่าง map และ odom โดยอัตโนมัติ

Lab 6: ทดลอง Actions และ TF

วัตถุประสงค์:

21. ส่ง Goal ให้ Turtle หมุนไป 90 องศา (1.57 rad)
22. ดู TF Tree ด้วย view_frames และ echo
23. ส่ง Goal หลายครั้งติดกันแล้วสังเกต Feedback
24. วาดแผนภาพ TF Frame ของ LotusBot

Quiz ทบทวนบทที่ 6

ข้อ	คำถาม	คำตอบ
1	Actions ต่างจาก Services อย่างไร?	Actions มี Feedback ระหว่างทำงาน, Services ไม่มี
2	ใน LotusBot มี TF Frame อะไรบ้าง?	map, odom, base_link, laser_frame
3	คำสั่งดู TF Tree คืออะไร?	ros2 run tf2_tools view_frames

บทที่ 7

SLAM และการนำทางอัตโนมัติ

SLAM & Autonomous Navigation with LotusBot

บทนี้เชื่อมโยงทุกสิ่งที่เรียนมาเข้ากับ **LotusBot** จริงๆ บน Jetson Nano เพื่อทำ SLAM และ Navigation อัตโนมัติ

การเริ่มต้น LotusBot

```
# 1. SSH เข้า Jetson Nano
ssh lotusbot@192.168.x.x

# 2. Source Environment
source /opt/ros/galactic/setup.bash
source ~/lotusbot_ws/install/setup.bash

# 3. รัน micro-ROS Agent (USB Serial)
ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyUSB0

# 4. รัน YDLidar X3 (Terminal แยก)
ros2 launch ydlidar_ros2_driver ydlidar_launch.py

# 5. ตรวจสอบ Topic
ros2 topic list  # ควรเห็น /scan, /cmd_vel, /odom
```

การทำ SLAM ด้วย slam_toolbox

```
# Terminal บน Host PC: เปิด RViz2
rviz2

# บน Jetson Nano: รัน SLAM
ros2 launch slam_toolbox online_async_launch.py \
  slam_params_file:=~/lotusbot_ws/config/slam_params.yaml

# ควบคุมด้วย Keyboard (Host PC)
ros2 run teleop_twist_keyboard teleop_twist_keyboard

# บันทึกแผนที่
ros2 run nav2_map_server map_saver_cli -f ~/map/my_map
```

Python Waypoint Navigator สำหรับ LotusBot

```
# ~/lotusbot_ws/src/lotusbot_nav/lotusbot_nav/multi_waypoint.py
import rclpy, math
from rclpy.node import Node
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry

class LotusWaypointNav(Node):
    def __init__(self):
        super().__init__('lotus_waypoint_nav')
        self.pub = self.create_publisher(Twist, '/cmd_vel', 10)
        self.sub = self.create_subscription(
            Odometry, '/odom', self.odom_cb, 10)
        # Waypoints สำหรับ LotusBot (เมตร)
        self.waypoints = [(1.0,0.0),(1.0,1.0),(0.0,1.0),(0.0,0.0)]
        self.current_wp = 0
        self.x = self.y = self.yaw = 0.0
        self.timer = self.create_timer(0.1, self.control_loop)

    def odom_cb(self, msg):
        self.x = msg.pose.pose.position.x
        self.y = msg.pose.pose.position.y
        # แปลง Quaternion → Euler
        q = msg.pose.pose.orientation
        self.yaw = math.atan2(
            2*(q.w*q.z + q.x*q.y), 1-2*(q.y**2 + q.z**2))

    def control_loop(self):
        if self.current_wp >= len(self.waypoints):
            self.pub.publish(Twist())  # หยุด
            self.get_logger().info('All waypoints reached!')
            return
        tx, ty = self.waypoints[self.current_wp]
        dx = tx - self.x; dy = ty - self.y
        dist = math.sqrt(dx**2 + dy**2)
        if dist < 0.1:
            self.current_wp += 1; return
        angle_err = math.atan2(dy, dx) - self.yaw
        # Normalize angle [-pi, pi]
        angle_err = math.atan2(math.sin(angle_err), math.cos(angle_err))
        msg = Twist()
        msg.linear.x  = min(0.2, 0.3 * dist)
        msg.angular.z = 1.5 * angle_err
        self.pub.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    node = LotusWaypointNav()
    rclpy.spin(node)
    rclpy.shutdown()
```

Lab 7: SLAM และ Navigation จริงด้วย LotusBot

วัตถุประสงค์:

25. เชื่อมต่อ LotusBot และตรวจสอบ Topic /scan และ /odom
26. รัน slam_toolbox และขับ LotusBot สร้างแผนที่ห้อง
27. บันทึกแผนที่และโหลดใช้งานใน Nav2
28. Challenge: เขียน Python Node นำทาง 3 Waypoints อัตโนมัติ

Quiz ทบทวนบทที่ 7

ข้อ	คำถาม	คำตอบ
1	SLAM ย่อมาจากอะไร?	Simultaneous Localization and Mapping
2	LotusBot ใช้ LiDAR รุ่นใด?	YDLidar X3 (12Hz, USB Serial)
3	micro-ROS Agent ทำงานผ่าน Transport ใด?	USB Serial (ttyUSB0)
4	ต้องแปลงข้อมูลใดจาก Odometry เพื่อหามุม?	Quaternion → Euler (ใช้ atan2)

บทที่ 🔧

การแก้ไขปัญหาที่พบบ่อย

Troubleshooting Guide

ปัญหา	สาเหตุ	วิธีแก้ไข
`command not found`	ยังไม่ source setup.bash	`source /opt/ros/galactic/setup.bash`
มองไม่เห็น Node กัน	ROS_DOMAIN_ID ต่างกัน	`export ROS_DOMAIN_ID=0`
Build แล้วโค้ดไม่เปลี่ยน	ยังไม่ source install/	`source install/setup.bash`
Permission Denied	สิทธิ์ workspace ผิด	`sudo chown -R $USER ~/ros2_ws`
LiDAR ไม่ตอบสนอง	USB Device Permission	`sudo chmod 666 /dev/ttyUSB0`
micro-ROS ไม่ Connect	Baudrate หรือ Port ผิด	ตรวจสอบ --dev และ Baudrate 115200

📋 **Checklist ก่อนเริ่มใช้ LotusBot ทุกครั้ง**

- source /opt/ros/galactic/setup.bash
- source ~/lotusbot_ws/install/setup.bash
- ตรวจสอบ USB: ls /dev/ttyUSB*
- รัน micro_ros_agent ก่อนเสมอ
- ตรวจสอบ Topic: ros2 topic list | grep scan

สรุป: Workflow หลักของ ROS 2

1. Source	2. Build	3. Launch	4. Monitor	5. Debug
`source setup.bash`	`colcon build`	`ros2 run / launch`	`topic echo / hz`	`rqt_graph / console`