

Homework Solution – Automated Navigation Script (Square Path)

Objective

Write a **Python ROS 2 node** that:

- Publishes to `/cmd_vel`
- Moves the robot in a **square path**
- Includes **logging, error checking, and clean shutdown**

Code: `square_path.py`

(save inside your ROS 2 Python package, e.g.
`turtlebot3_square/turtlebot3_square/square_path.py`)

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
import time

class SquarePath(Node):
    def __init__(self):
        super().__init__('square_path_node')
        # Publisher to the /cmd_vel topic
        self.publisher_ = self.create_publisher(Twist, '/cmd_vel',
10)

        # Create a Twist message to reuse
        self.cmd = Twist()

        self.get_logger().info("🚀 TurtleBot3 Square Path Node
Initialized!")

    def move_forward(self, speed=0.2, duration=3.0):
```

```

    """Move straight forward for a fixed duration"""
    self.cmd.linear.x = speed
    self.cmd.angular.z = 0.0
    self.get_logger().info(f"➡️ Moving forward: {speed} m/s for
{duration}s")
    self._publish_for(duration)

def turn(self, angular_speed=1.0, duration=1.57):
    """Rotate robot in place (approx 90 degrees turn)"""
    self.cmd.linear.x = 0.0
    self.cmd.angular.z = angular_speed
    self.get_logger().info(f"🔄 Turning: {angular_speed} rad/s
for {duration}s")
    self._publish_for(duration)

def stop_robot(self):
    """Stop robot motion"""
    self.cmd.linear.x = 0.0
    self.cmd.angular.z = 0.0
    self.publisher_.publish(self.cmd)
    self.get_logger().info("🔴 Robot stopped.")

def _publish_for(self, duration):
    """Helper function to publish velocity commands for a given
time"""
    start_time = time.time()
    while time.time() - start_time < duration:
        self.publisher_.publish(self.cmd)
        time.sleep(0.1) # 10 Hz publishing rate
    self.stop_robot()

def run_square(self):
    """Execute the 4-side square movement"""
    try:
        for i in range(4):
            self.get_logger().info(f"👉 Starting side {i+1}/4")
            self.move_forward(speed=0.2, duration=3.0)
            self.turn(angular_speed=1.0, duration=1.57)
        self.get_logger().info("✅ Finished square path
successfully!")
        self.stop_robot()
    
```

```

        except Exception as e:
            self.get_logger().error(f"❌ Error during movement:
{e} ")
            self.stop_robot()

def main(args=None):
    rclpy.init(args=args)
    node = SquarePath()
    try:
        node.run_square()
    except KeyboardInterrupt:
        node.get_logger().info("✖ Navigation interrupted by user.")
    finally:
        node.stop_robot()
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()

```

Step-by-Step Explanation

1. Node Initialization

```
self.publisher_ = self.create_publisher(Twist, '/cmd_vel', 10)
```

- Creates a publisher that sends **velocity commands** (Twist messages) to control the robot's motion.

2. Movement Commands

a. Move Forward

```
self.cmd.linear.x = 0.2 # forward 0.2 m/s
self.cmd.angular.z = 0.0
```

- This moves the robot in a straight line at 0.2 m/s.

b. Turn 90°

```
self.cmd.linear.x = 0.0  
self.cmd.angular.z = 1.0 # rotate in place
```

- Turns the robot in place at 1 radian/second (1.57 seconds ≈ 90°).

3. Time-Based Motion Control

```
while time.time() - start_time < duration:  
    self.publisher_.publish(self.cmd)  
    time.sleep(0.1)
```

- Keeps sending the command for a set duration, simulating a movement of known length.

4. Stop Command

```
self.cmd.linear.x = 0.0  
self.cmd.angular.z = 0.0  
self.publisher_.publish(self.cmd)
```

- Ensures the robot stops between actions and after finishing.

5. Logging

All steps use:

```
self.get_logger().info("message")
```

- Provides clear logs in your terminal, showing each phase of the motion.

6. Error Handling

The `try/except/finally` blocks ensure:

- If something fails → robot stops safely
- Node shuts down properly → no leftover velocity commands

Running the Node in Simulation

① Launch your **Gazebo simulation**:

```
ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

② Run your navigation script:

```
ros2 run turtlebot3_square square_path
```

③ Observe:

- The robot moves forward, then turns 90°, four times
- Returns to roughly the starting position

Add Feedback and Visualize Path

Overall Goal

Make your `square_path_node.py` smarter:

-  Still publishes velocity commands to `/cmd_vel`
 -  Also subscribes to `/odom` to get position feedback
 -  Logs real-time position and orientation
 -  Visualize everything in **RViz2**
-

Package Structure

```
turtlebot3_square/
└── package.xml
└── setup.py
└── turtlebot3_square/
    ├── __init__.py
    └── square_path_feedback.py  ← NEW Updated node
```

Updated Node: `square_path_feedback.py`

```
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
from math import sqrt, pow

class SquarePathNode(Node):

    def __init__(self):
        super().__init__('square_path_feedback')
        # Publisher to move the robot
        self.publisher_ = self.create_publisher(Twist, '/cmd_vel',
10)
        # Subscriber to get position feedback
        self.subscription = self.create_subscription(
            Odometry, '/odom', self.odom_callback, 10)
        self.subscription

        # Initialize movement parameters
        self.linear_speed = 0.2          # m/s
        self.angular_speed = 0.5         # rad/s
        self.side_length = 1.0           # meters
        self.turn_angle = 1.57          # radians (90°)

        self.current_x = 0.0
        self.current_y = 0.0
        self.start_x = 0.0
        self.start_y = 0.0
        self.step = 0
```

```

        self.turns = 0
        self.in_motion = False

        # Timer loop for publishing commands
        self.timer = self.create_timer(0.1, self.timer_callback)
        self.get_logger().info("Square Path Node with Odometry
Feedback started.")

    def odom_callback(self, msg):
        """Receive odometry updates from Gazebo."""
        self.current_x = msg.pose.pose.position.x
        self.current_y = msg.pose.pose.position.y

    def timer_callback(self):
        """Main control loop."""
        twist = Twist()

        if self.turns >= 4:
            twist.linear.x = 0.0
            twist.angular.z = 0.0
            self.publisher_.publish(twist)
            self.get_logger().info("✅ Completed square path.")
            rclpy.shutdown()
            return

        # Start a new side if not moving
        if not self.in_motion:
            self.start_x = self.current_x
            self.start_y = self.current_y
            self.in_motion = True
            self.get_logger().info(f"🚀 Starting side {self.turns +
1}!")

        # Compute distance moved from starting point
        distance = sqrt(pow(self.current_x - self.start_x, 2) +
pow(self.current_y - self.start_y, 2))

        if distance < self.side_length:
            twist.linear.x = self.linear_speed
            twist.angular.z = 0.0
        else:

```

```

        # Stop, then turn
        twist.linear.x = 0.0
        twist.angular.z = self.angular_speed
        self.in_motion = False
        self.turns += 1
        self.get_logger().info(f"⌚ Turning corner
({self.turns}/4)")

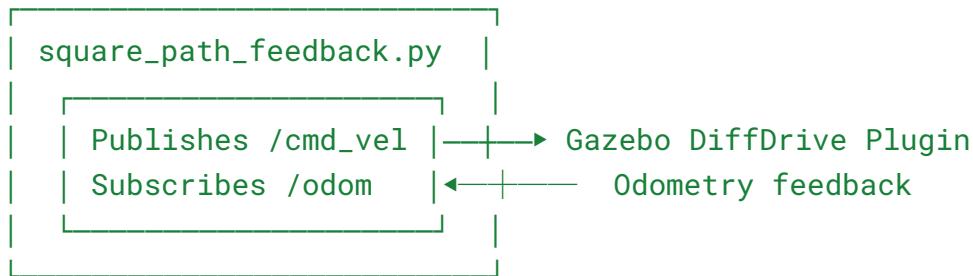
        self.publisher_.publish(twist)
        self.get_logger().info(
            f"Position: x={self.current_x:.2f},
y={self.current_y:.2f}, turns={self.turns}"
        )

def main(args=None):
    rclpy.init(args=args)
    node = SquarePathNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()

```

✳️ Key Data Flow



Message Types:

Topic	Direction	Message Type	Description
/cmd_1	Publish	geometry_msgs/Twist	Send linear & angular velocity commands

```
/odom      Subscribe   nav_msgs/Odometry    Receive position & orientation feedback
```

👀 Visualize in RViz2

Launch RViz2:

```
rviz2
```

Add Displays:

1. **TF** → shows robot frames
 2. **Odometry (/odom)** → shows trajectory path
 3. **RobotModel** → displays the robot model
 4. **LaserScan** (if using sensors) → shows environment readings
-

国旗 What You'll See in RViz2

- The robot moves in a **square trajectory**
- A line (odometry trace) showing the actual path
- You can compare it with the *ideal* square shape to check accuracy

```
^  
|  
|  
|----->  
(Start) ----- Square path in RViz
```

⚙️ How to Run

Build your package

```
colcon build --packages-select turtlebot3_square
```

```
source install/setup.bash
```

1.

Launch Gazebo Simulation

```
ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

2.

Run your node

```
ros2 run turtlebot3_square square_path_feedback
```

3.

Open RViz2 in another terminal

```
rviz2
```

4.

Expected Behavior

Component	Function
Node	Publishes <code>/cmd_vel</code> , reads <code>/odom</code>
Gazebo	Moves robot and publishes odometry
RViz2	Visualizes position and trajectory
Console Logs	Show position (x, y), side count, and motion states