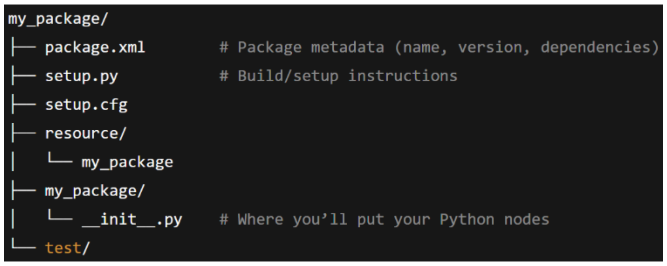
Creating a new workspace

1. mkdir -p ~/ros2_ws/src
2. cd ~/ros2_ws
3. colcon build
4. source install/setup.bash

Creating a new package

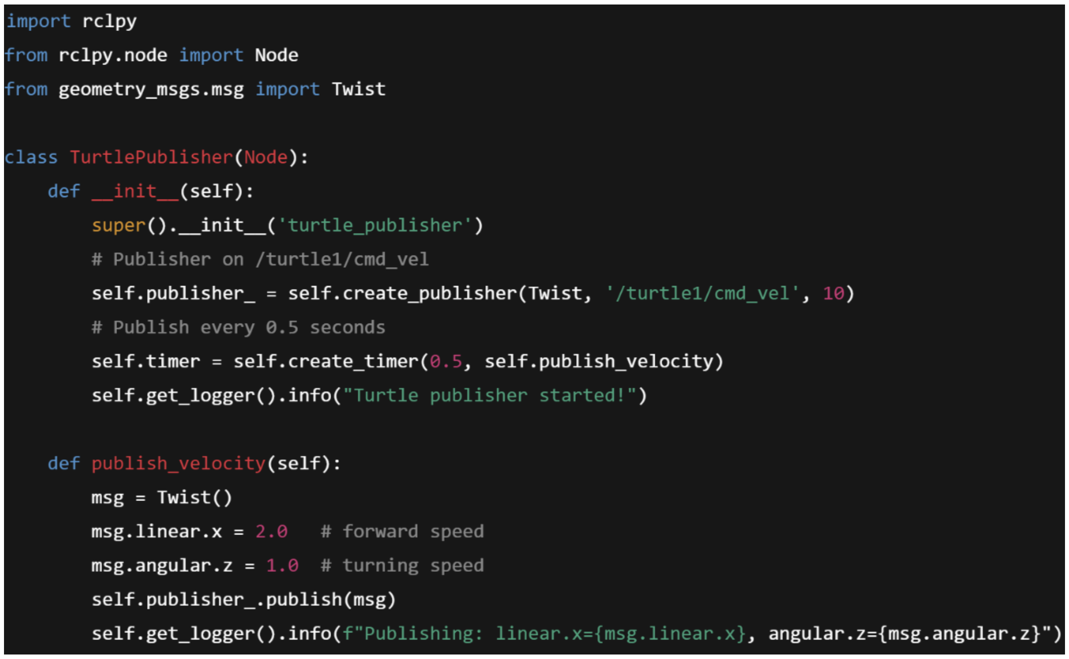
1. cd ~/ros2_ws/src
2. ros2 pkg create --build-type ament_python my_package
   #Replace my_package with our package name
3. In package.xml, Add dependencies
   <depend>rclpy</depend>
   <depend>geometry_msgs</depend>
   <depend>turtlesim</depend>
4. cd ~/ros2_ws
5. colcon build
6. source install/setup.bash



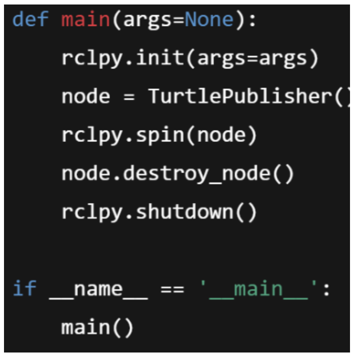
```
my_package/
├── package.xml       # Package metadata (name, version, dependencies)
├── setup.py          # Build/setup instructions
├── setup.cfg
├── resource/
│   └── my_package
├── my_package/
│   └── __init__.py    # Where you'll put your Python nodes
└── test/
```

Creating a publisher

1. cd ros2_ws/src/turtle_controller/turtle_controller
2. nano turtle_publisher.py
3. Write

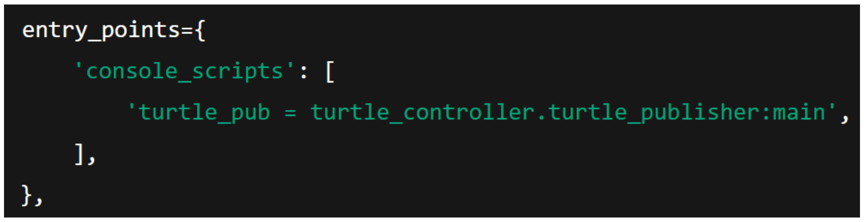


```python
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist

class TurtlePublisher(Node):
    def __init__(self):
        super().__init__('turtle_publisher')
        # Publisher on /turtle1/cmd_vel
        self.publisher_ = self.create_publisher(Twist, '/turtle1/cmd_vel', 10)
        # Publish every 0.5 seconds
        self.timer = self.create_timer(0.5, self.publish_velocity)
        self.get_logger().info("Turtle publisher started!")

    def publish_velocity(self):
        msg = Twist()
        msg.linear.x = 2.0   # forward speed
        msg.angular.z = 1.0  # turning speed
        self.publisher_.publish(msg)
        self.get_logger().info(f"Publishing: linear.x={msg.linear.x}, angular.z={msg.angular.z}")
```



```python
def main(args=None):
    rclpy.init(args=args)
    node = TurtlePublisher()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

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

4. cd ~/ros2_ws/src/turtle_controller
5. nano setup.py
6. Add

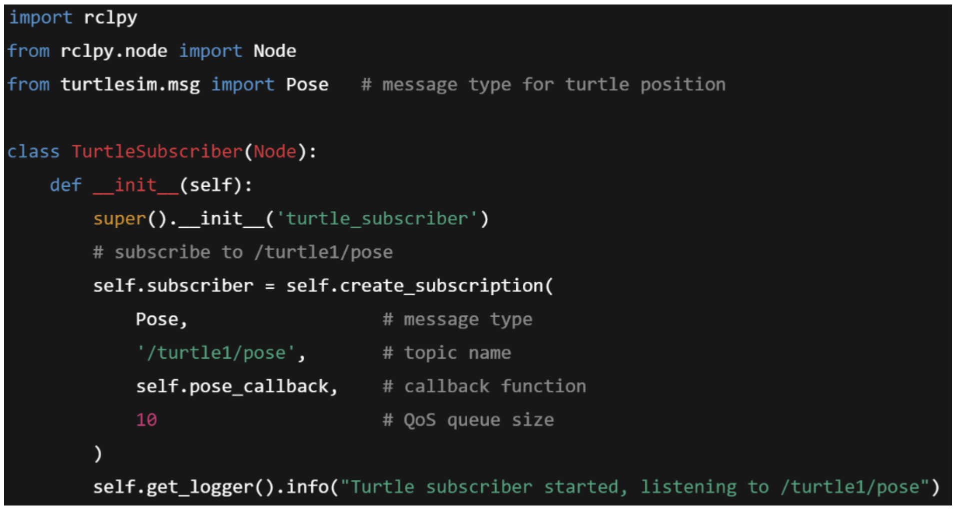


```python
entry_points={
    'console_scripts': [
        'turtle_pub = turtle_controller.turtle_publisher:main',
    ],
},
```

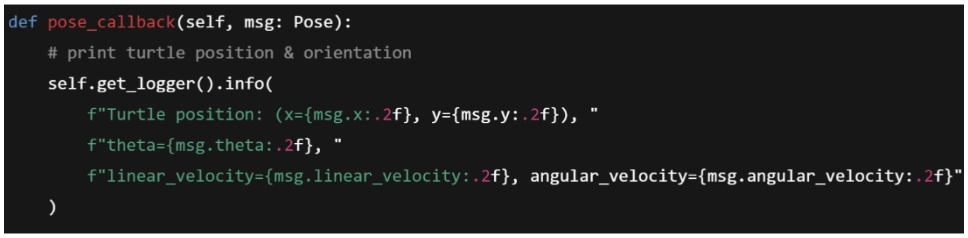
7. cd ~/ros2_ws
8. colcon build

Creating a subscriber

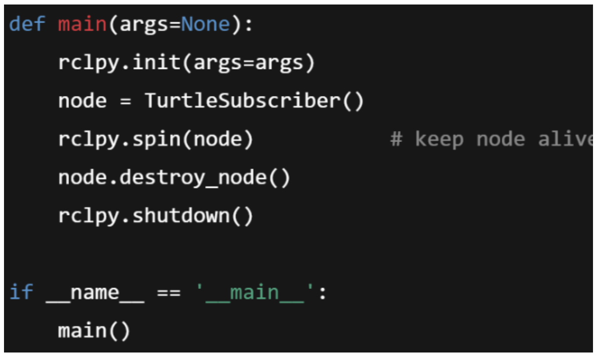
1. cd ros2_ws/src/turtle_controller/turtle_controller
2. nano turtle_subscriber.py
3. Write



```
import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose   # message type for turtle position

class TurtleSubscriber(Node):
    def __init__(self):
        super().__init__('turtle_subscriber')
        # subscribe to /turtle1/pose
        self.subscriber = self.create_subscription(
            Pose,                 # message type
            '/turtle1/pose',      # topic name
            self.pose_callback,   # callback function
            10                    # QoS queue size
        )
        self.get_logger().info("Turtle subscriber started, listening to /turtle1/pose")
```

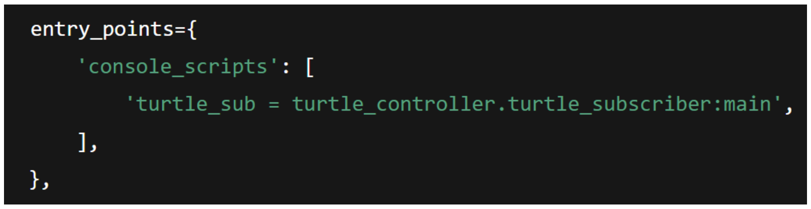


```
def pose_callback(self, msg: Pose):
    # print turtle position & orientation
    self.get_logger().info(
        f"Turtle position: (x={msg.x:.2f}, y={msg.y:.2f}), "
        f"theta={msg.theta:.2f}, "
        f"linear_velocity={msg.linear_velocity:.2f}, angular_velocity={msg.angular_velocity:.2f}"
    )
```



```
def main(args=None):
    rclpy.init(args=args)
    node = TurtleSubscriber()
    rclpy.spin(node)           # keep node alive
    node.destroy_node()
    rclpy.shutdown()

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

4. cd ~/ros2_ws/src/turtle_controller
5. nano setup.py
6. Add



```
entry_points={
    'console_scripts': [
        'turtle_sub = turtle_controller.turtle_subscriber:main',
    ],
},
```

7. cd ~/ros2_ws
8. colcon build

Running all nodes

1. Opening 3 terminals
2. For each terminal, run
   ```
   cd ros2_ws
   source install/setup.bash
   ```
3. In first terminal, ros2 run turtlesim turtlesim_node
4. In second terminal, ros2 run turtle_controller turtle_pub
5. In third terminal, ros2 run turtle_controller turtle_sub

Constructing Twist() msg

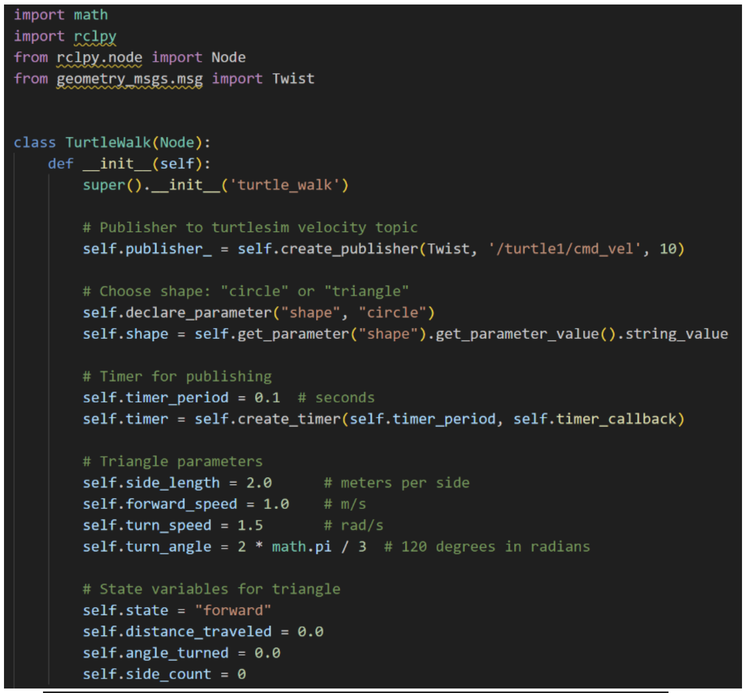
```
msg = Twist()
msg.linear.x =2.0
msg.angular.z =1.0
print(msg)
```

```
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
```

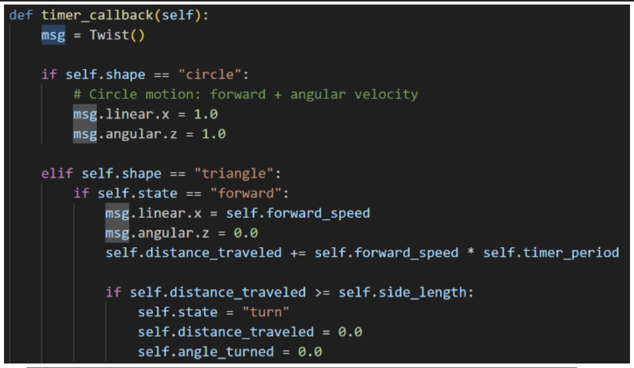
Constructing Pose() msg

```
pose = Pose()
pose.x = 5.5
pose.y = 5.5
pose.theta = 1.57
pose.linear_velocity = 2.0
pose.angular_velocity = 1.0
```

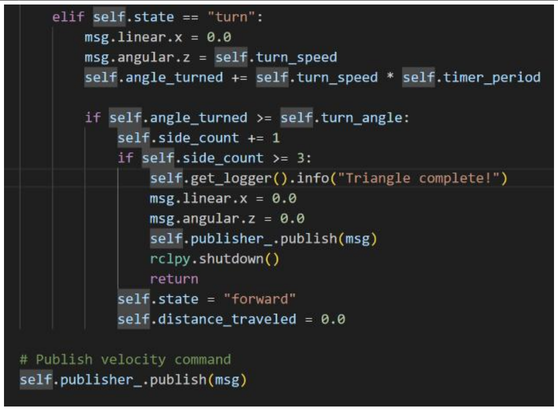
```
x: 5.5
y: 5.5
theta: 1.57
linear_velocity: 2.0
angular_velocity: 1.0
```



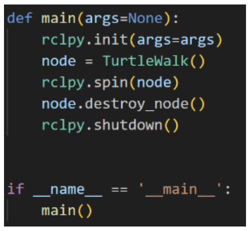
```python
import math
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist


class TurtleWalk(Node):
    def __init__(self):
        super().__init__('turtle_walk')

        # Publisher to turtlesim velocity topic
        self.publisher_ = self.create_publisher(Twist, '/turtle1/cmd_vel', 10)

        # Choose shape: "circle" or "triangle"
        self.declare_parameter("shape", "circle")
        self.shape = self.get_parameter("shape").get_parameter_value().string_value

        # Timer for publishing
        self.timer_period = 0.1  # seconds
        self.timer = self.create_timer(self.timer_period, self.timer_callback)

        # Triangle parameters
        self.side_length = 2.0      # meters per side
        self.forward_speed = 1.0    # m/s
        self.turn_speed = 1.5       # rad/s
        self.turn_angle = 2 * math.pi / 3  # 120 degrees in radians

        # State variables for triangle
        self.state = "forward"
        self.distance_traveled = 0.0
        self.angle_turned = 0.0
        self.side_count = 0
```



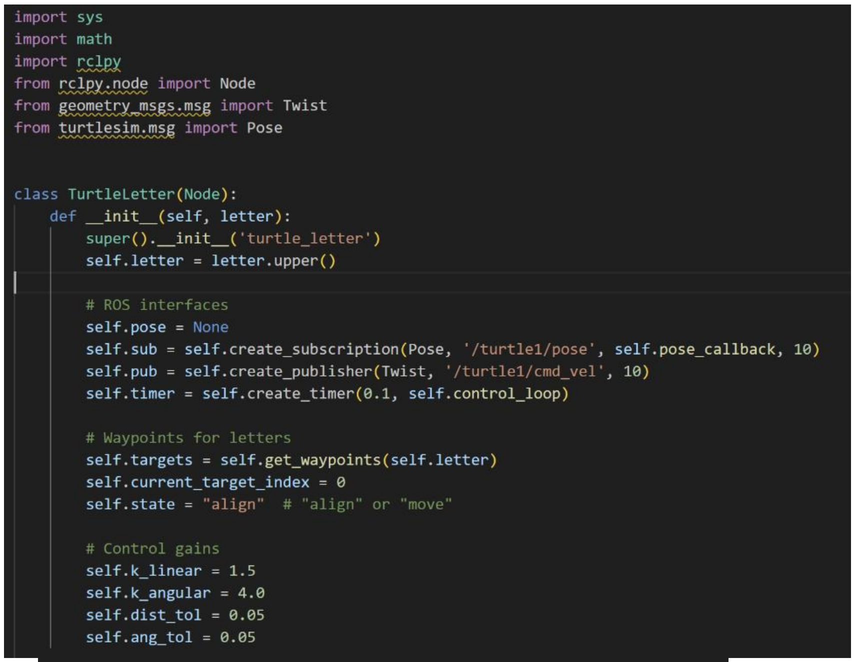
```python
    def timer_callback(self):
        msg = Twist()

        if self.shape == "circle":
            # Circle motion: forward + angular velocity
            msg.linear.x = 1.0
            msg.angular.z = 1.0

        elif self.shape == "triangle":
            if self.state == "forward":
                msg.linear.x = self.forward_speed
                msg.angular.z = 0.0
                self.distance_traveled += self.forward_speed * self.timer_period

                if self.distance_traveled >= self.side_length:
                    self.state = "turn"
                    self.distance_traveled = 0.0
                    self.angle_turned = 0.0
```



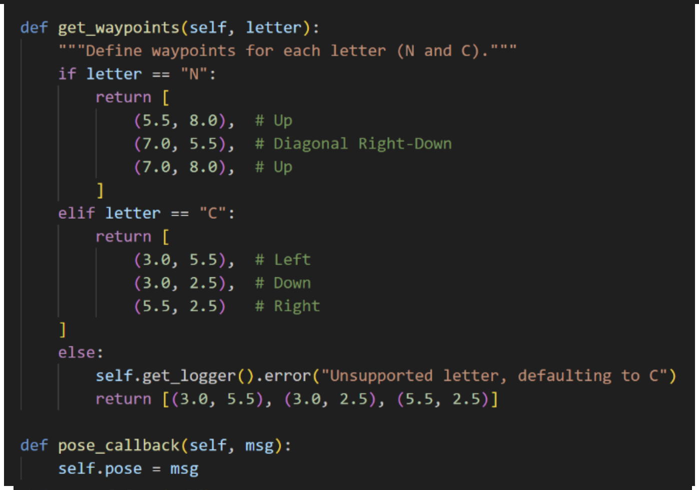
```python
            elif self.state == "turn":
                msg.linear.x = 0.0
                msg.angular.z = self.turn_speed
                self.angle_turned += self.turn_speed * self.timer_period

                if self.angle_turned >= self.turn_angle:
                    self.side_count += 1
                    if self.side_count >= 3:
                        self.get_logger().info("Triangle complete!")
                        msg.linear.x = 0.0
                        msg.angular.z = 0.0
                        self.publisher_.publish(msg)
                        rclpy.shutdown()
                        return
                    self.state = "forward"
                    self.distance_traveled = 0.0

        # Publish velocity command
        self.publisher_.publish(msg)
```



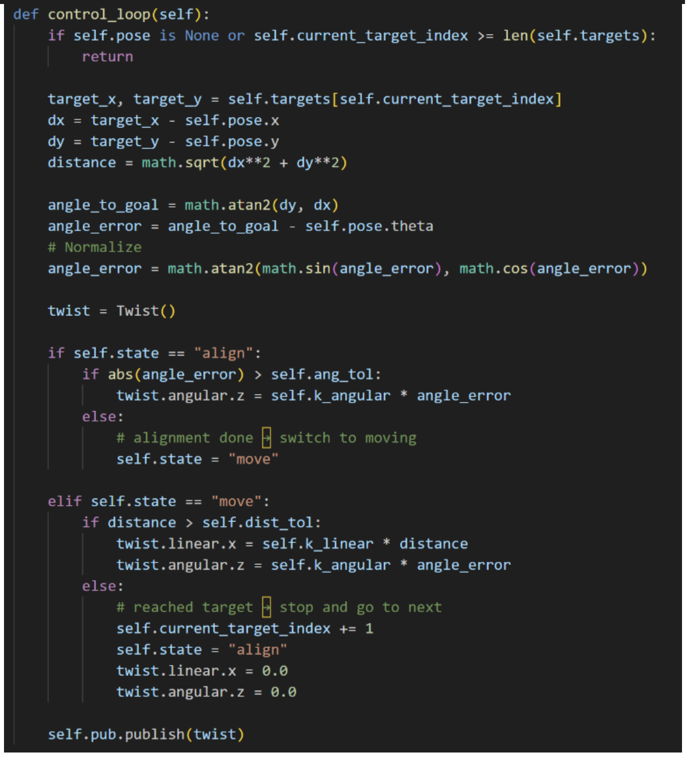
```python
def main(args=None):
    rclpy.init(args=args)
    node = TurtleWalk()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()


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



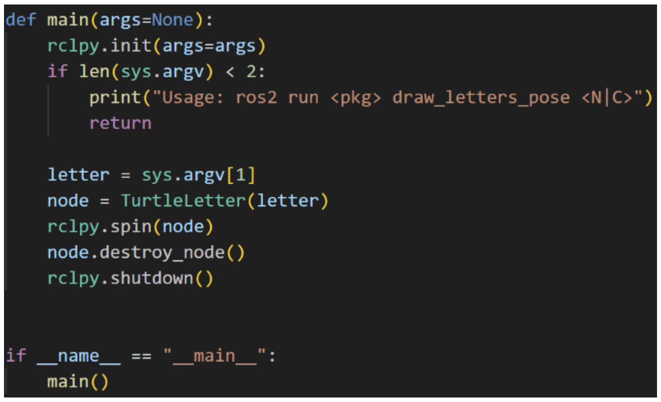
```python
import sys
import math
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
from turtlesim.msg import Pose


class TurtleLetter(Node):
    def __init__(self, letter):
        super().__init__('turtle_letter')
        self.letter = letter.upper()

        # ROS interfaces
        self.pose = None
        self.sub = self.create_subscription(Pose, '/turtle1/pose', self.pose_callback, 10)
        self.pub = self.create_publisher(Twist, '/turtle1/cmd_vel', 10)
        self.timer = self.create_timer(0.1, self.control_loop)

        # Waypoints for letters
        self.targets = self.get_waypoints(self.letter)
        self.current_target_index = 0
        self.state = "align"  # "align" or "move"

        # Control gains
        self.k_linear = 1.5
        self.k_angular = 4.0
        self.dist_tol = 0.05
        self.ang_tol = 0.05
```



```python
    def get_waypoints(self, letter):
        """Define waypoints for each letter (N and C)."""
        if letter == "N":
            return [
                (5.5, 8.0),  # Up
                (7.0, 5.5),  # Diagonal Right-Down
                (7.0, 8.0),  # Up
            ]
        elif letter == "C":
            return [
                (3.0, 5.5),  # Left
                (3.0, 2.5),  # Down
                (5.5, 2.5)   # Right
        ]
        else:
            self.get_logger().error("Unsupported letter, defaulting to C")
            return [(3.0, 5.5), (3.0, 2.5), (5.5, 2.5)]

    def pose_callback(self, msg):
        self.pose = msg
```



```python
    def control_loop(self):
        if self.pose is None or self.current_target_index >= len(self.targets):
            return

        target_x, target_y = self.targets[self.current_target_index]
        dx = target_x - self.pose.x
        dy = target_y - self.pose.y
        distance = math.sqrt(dx**2 + dy**2)

        angle_to_goal = math.atan2(dy, dx)
        angle_error = angle_to_goal - self.pose.theta
        # Normalize
        angle_error = math.atan2(math.sin(angle_error), math.cos(angle_error))

        twist = Twist()

        if self.state == "align":
            if abs(angle_error) > self.ang_tol:
                twist.angular.z = self.k_angular * angle_error
            else:
                # alignment done → switch to moving
                self.state = "move"

        elif self.state == "move":
            if distance > self.dist_tol:
                twist.linear.x = self.k_linear * distance
                twist.angular.z = self.k_angular * angle_error
            else:
                # reached target → stop and go to next
                self.current_target_index += 1
                self.state = "align"
                twist.linear.x = 0.0
                twist.angular.z = 0.0

        self.pub.publish(twist)
```



```python
def main(args=None):
    rclpy.init(args=args)
    if len(sys.argv) < 2:
        print("Usage: ros2 run <pkg> draw_letters_pose <N|C>")
        return

    letter = sys.argv[1]
    node = TurtleLetter(letter)
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()


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