BumperBot Simulation Testing Guide

Hands-On Tutorial: Test the Three AI Agent Packages

Author: RoboShire Team
Version: 1.0
Date: October 23, 2025
Estimated Time: 2-3 hours

BumperBot Simulation Testing Guide

Hands-On Tutorial: Test the Three AI Agent Packages

Author: RoboShire Team Version: 1.0 Date: October 23, 2025 Estimated Time: 2-3 hours Difficulty: Beginner-Friendly

Table of Contents

1. Prerequisites & Setup
2. Test 1: Sarah's Beginner Package
3. Test 2: Marcus's Intermediate Package
4. Test 3: Elena's Enterprise Package
5. Comparison & Results
6. Troubleshooting

1. Prerequisites & Setup

1.1 System Requirements

Hardware: - CPU: Intel i5 or better - RAM: 8GB minimum (16GB recommended) - Disk Space: 10GB free

Software: - Ubuntu 22.04 (or Windows WSL2) - ROS2 Humble Hawksbill - Python 3.10+ - MuJoCo 3.0+ (optional, for physics simulation)

1.2 Install ROS2 Humble

If ROS2 is not installed:

```
# Add ROS2 repository
sudo apt update
sudo apt install software-properties-common
sudo add-apt-repository universe
```

```
# Install ROS2 Humble
sudo apt update
sudo apt install ros-humble-desktop

# Install additional tools
sudo apt install python3-colcon-common-extensions
sudo apt install ros-humble-robot-localization
```

1.3 Verify Installation

```
# Check ROS2
source /opt/ros/humble/setup.bash
ros2 --version

# Should output: ros2 cli version ...
```

1.4 Setup Workspace

```
# Navigate to project
cd d:/ROS2_PROJECT/workspace

# Or if on Linux:
cd /path/to/ROS2_PROJECT/workspace

# Source ROS2
source /opt/ros/humble/setup.bash
```

2. Test 1: Sarah's Beginner Package

2.1 Overview

What You'll Test: - Basic differential drive control - Wheel encoder odometry - IMU sensor integration - EKF sensor fusion - Keyboard teleoperation

Expected Results: - Robot moves in autonomous pattern - Odometry tracks position - 67% accuracy improvement with EKF

2.2 Build the Package

```
# Clean previous builds (optional)
rm -rf build/ install/ log/

# Build Sarah's package
colcon build --packages-select sarah_bumperbot_pkg

# Expected output:
# Starting >>> sarah_bumperbot_pkg
# Finished <<< sarah_bumperbot_pkg [10.5s]
#
# Summary: 1 package finished [10.6s]
```

Success Criteria: - No error messages - install/sarah_bumperbot_pkg/ directory created - Build completes in <30 seconds

If Build Fails: See Troubleshooting Section

2.3 Source the Workspace

```
# On Linux:
source install/setup.bash

# On Windows (WSL):
source install/setup.bash

# Verify package is available
ros2 pkg list | grep sarah_bumperbot_pkg

# Should output: sarah_bumperbot_pkg
```

2.4 Launch the System

Open Terminal 1 (Main System):

```
# Launch all nodes
ros2 launch sarah_bumperbot_pkg bumperbot_bringup.launch.py

# Expected output:
# [INFO] [launch]: All log files can be found below /home/...
# [INFO] [velocity_controller]: Starting velocity_controller node
# [INFO] [wheel_encoder_reader]: Starting wheel_encoder_reader node
# [INFO] [odometry_calculator]: Starting odometry_calculator node
# [INFO] [imu_reader]: Starting imu_reader node
# [INFO] [ekf_localization]: Starting ekf_localization node
# [INFO] [base_to_left_wheel_tf]: Broadcasting TF: base_link -> left_wheel
# [INFO] [base_to_right_wheel_tf]: Broadcasting TF: base_link -> right_wheel
# [INFO] [base_to_imu_tf]: Broadcasting TF: base_link -> imu_link
```

Success Indicators: - All 8 nodes start without errors - Log messages appear continuously - No "[ERROR]" or "[FATAL]" messages

Screenshot Opportunity: Terminal showing all nodes running

2.5 Monitor Topics

Open Terminal 2 (Monitoring):

```
# Source workspace first
source /opt/ros/humble/setup.bash
source install/setup.bash

# List all topics
ros2 topic list

# Expected topics:
# /cmd_vel
# /joint_states
# /imu/data
# /odom
# /odom/filtered
# /tf
# /tf_static
```

Test 1: View Velocity Commands

```
ros2 topic echo /cmd_vel
```

Expected Output:

```
linear:
  x: 0.2      # Moving forward
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0      # Not turning

---

# After 5 seconds, changes to:
linear:
  x: 0.0      # Stopped
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.5      # Turning
```

Success: Velocity alternates between forward (0.2 m/s) and turning (0.5 rad/s)

Screenshot Opportunity: Terminal showing /cmd_vel output

Test 2: View Raw Odometry

```
ros2 topic echo /odom
```

Expected Output:

```
header:
  stamp:
    sec: 1698123456
    nanosec: 789000000
  frame_id: odom
child_frame_id: base_link
pose:
  pose:
    position:
      x: 0.523    # Changes over time
      y: 0.087
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.123
      w: 0.992
  covariance: [0.05, 0.0, ...]
```

Success: Position (x, y) changes as robot moves

Test 3: View Filtered Odometry

```
ros2 topic echo /odom/filtered
```

Expected Output: Similar to /odom but more stable (smoother values)

Test 4: Compare Raw vs Filtered

Open Terminal 3:

```
# Monitor both simultaneously
ros2 topic echo /odom | grep "x:" &
ros2 topic echo /odom/filtered | grep "x:"
```

Observation: Filtered odometry has less noise (smoother progression)

2.6 Test Keyboard Control

Stop the autonomous controller first:

In Terminal 1 (where launch is running): - Press Ctrl+C to stop all nodes

Launch with keyboard control:

```
# Edit launch file to enable keyboard teleop
# OR launch individual nodes:

# Terminal 1: Launch sensors and odometry
ros2 run sarah_bumperbot_pkg wheel_encoder_reader &
ros2 run sarah_bumperbot_pkg odometry_calculator &
ros2 run sarah_bumperbot_pkg imu_reader &

# Terminal 2: Launch keyboard teleop
ros2 run sarah_bumperbot_pkg keyboard_teleop
```

Expected Output:

```
Keyboard Teleop Controller
==========================
Use the following keys:

    W - Forward
    S - Backward
    A - Turn Left
    D - Turn Right
    SPACE - Emergency Stop

Current speed: 0.2 m/s
```

Interactive Test: 1. Press W Robot should move forward 2. Press A Robot should turn left 3. Press D Robot should turn right 4. Press S Robot should move backward 5. Press SPACE Robot should stop immediately

Success: All keys control the robot correctly

Screenshot Opportunity: Teleop terminal showing controls

2.7 Visualize TF Tree

Open Terminal 4:

```
# Install TF tools if needed
sudo apt install ros-humble-tf2-tools

# Generate TF tree PDF
ros2 run tf2_tools view_frames

# Wait 5 seconds, then:
# Output: Listening to tf data... Finished.
# Outputted frames.pdf

# View the PDF
evince frames.pdf  # Or use your PDF viewer
```

Expected TF Tree:

```
odom
  base_link
       left_wheel
       right_wheel
       imu_link
```

Success: All frames present, proper hierarchy

Include in PDF: Screenshot of frames.pdf showing tree structure

2.8 Measure Odometry Accuracy

Test: Drive in a Square

This test measures how accurate the odometry is by driving in a 2m x 2m square and checking if the robot returns to the origin.

Setup Script (create test_square_path.py):

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

class SquarePathTest(Node):
    def __init__(self):
        super().__init__('square_path_test')

        self.cmd_pub = self.create_publisher(Twist, '/cmd_vel', 10)
        self.odom_sub = self.create_subscription(
            Odometry, '/odom', self.odom_callback, 10)
        self.filtered_sub = self.create_subscription(
            Odometry, '/odom/filtered', self.filtered_callback, 10)

        self.raw_odom = None
        self.filtered_odom = None

        self.get_logger().info("Square path test ready")

    def odom_callback(self, msg):
        self.raw_odom = msg

    def filtered_callback(self, msg):
        self.filtered_odom = msg

    def drive_forward(self, distance):
        """Drive forward by distance meters"""
        start_x = self.raw_odom.pose.pose.position.x if self.raw_odom else 0

        twist = Twist()
        twist.linear.x = 0.2  # 0.2 m/s

        # Drive until distance covered
        while True:
            self.cmd_pub.publish(twist)
            rclpy.spin_once(self, timeout_sec=0.1)

            if self.raw_odom:
                current_x = self.raw_odom.pose.pose.position.x
                if abs(current_x - start_x) >= distance:
                    break

        # Stop
        twist.linear.x = 0.0
        self.cmd_pub.publish(twist)
        time.sleep(1)

    def turn_90_degrees(self):
        """Turn 90 degrees (/2 radians)"""
        twist = Twist()
        twist.angular.z = 0.5  # 0.5 rad/s

        # Turn for ~3 seconds (0.5 * 3 = 1.5 rad  86 degrees)
        for _ in range(30):  # 30 iterations * 0.1s = 3s
            self.cmd_pub.publish(twist)
            time.sleep(0.1)

        # Stop
        twist.angular.z = 0.0
        self.cmd_pub.publish(twist)
        time.sleep(1)

    def run_square_test(self):
        """Drive in a 2m x 2m square"""
        self.get_logger().info("Starting square path test...")
        time.sleep(2)  # Wait for odometry to initialize
        # Record start position
        if not self.raw_odom:
            self.get_logger().error("No odometry data!")
            return

        start_x = self.raw_odom.pose.pose.position.x
        start_y = self.raw_odom.pose.pose.position.y

        self.get_logger().info(f"Start position: ({start_x:.3f}, {start_y:.3f})")

        # Drive square: forward, turn, forward, turn, forward, turn, forward, turn
        for i in range(4):
            self.get_logger().info(f"Side {i+1}: Driving forward 2m")
            self.drive_forward(2.0)

            if i < 3:  # Don't turn after last side
                self.get_logger().info(f"Turning 90 degrees")
                self.turn_90_degrees()

        time.sleep(2)  # Settle

        # Record end position
        end_x = self.raw_odom.pose.pose.position.x
        end_y = self.raw_odom.pose.pose.position.y

        filtered_x = self.filtered_odom.pose.pose.position.x if self.filtered_odom else 0
        filtered_y = self.filtered_odom.pose.pose.position.y if self.filtered_odom else 0

        # Calculate errors
        raw_error = math.sqrt((end_x - start_x)**2 + (end_y - start_y)**2)
        filtered_error = math.sqrt((filtered_x - start_x)**2 + (filtered_y - start_y)**2)

        self.get_logger().info("\\n=== Test Results ===")
        self.get_logger().info(f"Start: ({start_x:.3f}, {start_y:.3f})")
        self.get_logger().info(f"End (raw): ({end_x:.3f}, {end_y:.3f})")
        self.get_logger().info(f"End (filtered): ({filtered_x:.3f}, {filtered_y:.3f})")
        self.get_logger().info(f"Raw error: {raw_error*100:.1f} cm")
        self.get_logger().info(f"Filtered error: {filtered_error*100:.1f} cm")

        improvement = (1 - filtered_error/raw_error) * 100 if raw_error > 0 else 0
        self.get_logger().info(f"Improvement: {improvement:.1f}%")

def main():
    rclpy.init()
    node = SquarePathTest()

    try:
        node.run_square_test()
    except KeyboardInterrupt:
        pass

    node.destroy_node()
    rclpy.shutdown()

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

Run the Test:

```
# Make executable
chmod +x test_square_path.py

# Run test
python3 test_square_path.py
```

Expected Results:

```
[INFO] Starting square path test...
[INFO] Start position: (0.000, 0.000)
[INFO] Side 1: Driving forward 2m
[INFO] Turning 90 degrees
[INFO] Side 2: Driving forward 2m
[INFO] Turning 90 degrees
[INFO] Side 3: Driving forward 2m
[INFO] Turning 90 degrees
[INFO] Side 4: Driving forward 2m

=== Test Results ===
[INFO] Start: (0.000, 0.000)
[INFO] End (raw): (0.142, -0.085)
```

```
[INFO] End (filtered): (0.034, -0.021)
[INFO] Raw error: 16.6 cm
[INFO] Filtered error: 4.0 cm
[INFO] Improvement: 75.9%
```

Success Criteria: - Raw error: 10-20 cm (acceptable) - Filtered error: 3-5 cm (excellent!) - Improvement: 65-75% (matches Sarah's 67% result)

Record Your Results:

Test	Your Raw Error	Your Filtered Error	Improvement %
Square Path	_ cm	_ cm	_ %

Screenshot Opportunity: Terminal showing test results

2.9 Stop the System

```
# In Terminal 1 (where launch is running)
Ctrl+C

# Wait for clean shutdown:
# [INFO] Shutting down velocity_controller
# [INFO] Shutting down odometry_calculator
# ...
# [INFO] All nodes stopped cleanly
```

2.10 Sarah's Package Summary

What You Tested: - [x] Build and installation - [x] All 8 nodes launch successfully - [x] Autonomous movement pattern (forward turn) - [x] Raw odometry tracking - [x] EKF filtered odometry - [x] Keyboard teleoperation - [x] TF tree structure - [x] Square path accuracy test

Your Results:

Metric	Expected	Your Result
Build Time	<30s	_____ s
Nodes Running	8/8	_____ / 8
Velocity Command	alternating	_____
Raw Error	10-20cm	_____ cm
Filtered Error	3-5cm	_____ cm
Improvement	65-75%	_____ %

Rating: _____ / 10

3. Test 2: Marcus's Intermediate Package

3.1 Overview

What You'll Test: - Optimized node performance - Unified TF broadcaster (1 node vs 3) - System monitoring - Performance profiling - Multiple launch configurations - QoS profile optimization

Expected Results: - 25% better CPU usage vs Sarah's - Real-time system health monitoring - Performance metrics tracking

3.2 Build the Package

```
# Clean build (optional)
rm -rf build/marcus_bumperbot_pkg install/marcus_bumperbot_pkg

# Build
colcon build --packages-select marcus_bumperbot_pkg

# Expected: Faster build due to fewer nodes
# Summary: 1 package finished [8.2s]
```

3.3 Launch Full System

Open Terminal 1:

```
source install/setup.bash

# Launch with all features
ros2 launch marcus_bumperbot_pkg bumperbot_full.launch.py

# Expected output:
# [INFO] [velocity_controller]: Optimized controller starting
# [INFO] [odometry_calculator]: Using exact circular arc integration
# [INFO] [imu_reader]: Low-pass filter enabled (cutoff: 5Hz)
# [INFO] [tf_broadcaster]: Broadcasting 3 transforms (unified)
# [INFO] [system_monitor]: Monitoring system health
# [INFO] [performance_profiler]: Tracking metrics
```

Key Difference from Sarah's: Only 7 nodes (vs 8), unified TF broadcaster

3.4 Monitor System Health

Open Terminal 2:

```
source install/setup.bash

# View system diagnostics
ros2 topic echo /diagnostics
```

Expected Output:

```
header:
  stamp: ...
status:
  - name: "velocity_controller"
    level: 0  # 0 = OK
    message: "Running normally"
    hardware_id: ""
    values:
      - key: "CPU Usage"
        value: "0.5%"
      - key: "Memory"
        value: "8.2 MB"
```

```
          - key: "Update Rate"
            value: "10.0 Hz"

      - name: "odometry_calculator"
        level: 0
        message: "Running normally"
        values:
          - key: "CPU Usage"
            value: "2.1%"
          - key: "Update Rate"
            value: "50.0 Hz"
```

Success: All nodes report status: OK (level: 0)

Screenshot Opportunity: Diagnostics showing all healthy nodes

3.5 View Performance Metrics

Open Terminal 3:

```
# View performance data
ros2 topic echo /performance_metrics
```

Expected Output:

```
timestamp: ...
metrics:
  - node_name: "velocity_controller"
    cpu_percent: 0.5
    memory_mb: 8.2
    loop_frequency_hz: 10.0
    callback_duration_ms: 0.8

  - node_name: "odometry_calculator"
    cpu_percent: 2.1
    memory_mb: 12.4
    loop_frequency_hz: 50.0
    callback_duration_ms: 1.2

  - node_name: "system_monitor"
    cpu_percent: 1.5
    memory_mb: 10.1
```

Record Performance Data:

Node	CPU %	Memory (MB)	Frequency (Hz)
velocity_controller	______	______	______
odometry_calculator	______	______	______
tf_broadcaster	______	______	______
system_monitor	______	______	______
TOTAL	______	______	-

Compare with Sarah's Package:

Metric	Sarah's	Marcus's	Improvement
Total CPU	~20%	~15%	25% better
Total Memory	~125 MB	~100 MB	20% better
Node Count	8	7	1 fewer

3.6 Test Modular Launch Files

Test A: Sensors Only

```
# Stop full system (Ctrl+C in Terminal 1)

# Launch only sensor nodes
ros2 launch marcus_bumperbot_pkg sensors_only.launch.py
```

Expected: Only sensor-related nodes start (no controllers)

```
# Check running nodes
ros2 node list

# Should see:
# /wheel_encoder_reader
# /imu_reader
# /odometry_calculator
# (No velocity_controller, no keyboard_teleop)
```

Use Case: Testing sensor pipeline without robot movement

Test B: Controllers Only

```
# Stop sensors (Ctrl+C)

# Launch only control nodes
ros2 launch marcus_bumperbot_pkg controllers_only.launch.py
```

Expected: Only control nodes start

```
ros2 node list

# Should see:
# /velocity_controller
# /keyboard_teleop
# (No sensors, for development/testing)
```

Use Case: Development and testing of control algorithms

3.7 Test Multi-Robot Capability

Launch Robot 1:

```
# Terminal 1
ros2 launch marcus_bumperbot_pkg bumperbot_full.launch.py namespace:=robot1
```

Launch Robot 2:

```
# Terminal 2
ros2 launch marcus_bumperbot_pkg bumperbot_full.launch.py namespace:=robot2
```

Verify Namespaces:

```
# Terminal 3
ros2 topic list | grep cmd_vel

# Should see:
# /robot1/cmd_vel
# /robot2/cmd_vel
# (No conflicts!)
```

Control Each Robot Independently:

```
# Control Robot 1
ros2 topic pub /robot1/cmd_vel geometry_msgs/Twist "linear: {x: 0.2}" -1

# Control Robot 2
ros2 topic pub /robot2/cmd_vel geometry_msgs/Twist "linear: {x: -0.1}" -1
```

Success: Both robots run simultaneously without topic conflicts

Screenshot Opportunity: Two terminals showing separate robot namespaces

3.8 Accuracy Test (Same as Sarah's)

Run the square path test again:

```
python3 test_square_path.py
```

Expected Results: Similar to Sarah's (73% improvement)

Compare:

Package	Raw Error	Filtered Error	Improvement
Sarah's	16.6 cm	4.5 cm	73%
Marcus's	_____ cm	_____ cm	_____ %

Analysis: Accuracy should be nearly identical (same odometry algorithm)

3.9 Marcus's Package Summary

What You Tested: - [x] Optimized node performance - [x] System health monitoring - [x] Performance profiling - [x] Modular launch files (3 options) - [x] Multi-robot namespaces - [x] CPU/memory optimization

Your Results:

Metric	Expected	Your Result
CPU Usage	~15%	_____ %
Memory Usage	~100 MB	_____ MB
Node Count	7	_____
Multi-Robot	works	_____
System Monitor	running	_____

Advantages over Sarah's: - [ ] Lower CPU usage (_ %) - [ ] Lower memory usage (___ MB) - [ ] Unified TF broadcaster - [ ] System health monitoring - [ ] Performance profiling - [ ] Multi-robot support

Rating: _____ / 10

4. Test 3: Elena's Enterprise Package

4.1 Overview

What You'll Test: - Lifecycle-managed nodes - Watchdog monitoring system - Multi-layer safety features - Docker deployment - Production configuration - Unit tests

Expected Results: - Professional lifecycle management - Automatic fault detection - Enterprise-grade safety - Docker containerization working

4.2 Build the Package

```
colcon build --packages-select elena_bumperbot_pkg

# Expected: Longer build time (more features)
# Summary: 1 package finished [15.3s]
```

4.3 Run Unit Tests

Before launching, test the package:

```
# Run all tests
cd src/elena_bumperbot_pkg
./scripts/run_tests.sh

# Or manually:
pytest test/unit/ -v

# Expected output:
# test_kinematics.py::test_forward_kinematics PASSED
# test_kinematics.py::test_inverse_kinematics PASSED
# test_kinematics.py::test_odometry_integration PASSED
# test_safety.py::test_velocity_limiting PASSED
# test_safety.py::test_acceleration_limiting PASSED
# test_safety.py::test_timeout_detection PASSED
# test_filters.py::test_low_pass_filter PASSED
# test_filters.py::test_kalman_filter PASSED
# ...
# ======================== 40 passed in 2.3s ========================
```

Success: All tests pass

Test Results:

Test Suite	Tests Run	Passed	Failed
Kinematics	______	______	______
Safety	______	______	______
Filters	______	______	______
TOTAL	40	______	______

4.4 Launch Production System

Open Terminal 1:

```
source install/setup.bash

# Launch production configuration
ros2 launch elena_bumperbot_pkg production.launch.py
```

Expected Output:

```
[INFO] [lifecycle_manager]: Starting lifecycle nodes
[INFO] [velocity_controller]: State: UNCONFIGURED  CONFIGURING
[INFO] [velocity_controller]: Safety monitor initialized
[INFO] [velocity_controller]: State: CONFIGURING  INACTIVE
```

```
[INFO] [lifecycle_manager]: Configuring velocity_controller... OK
[INFO] [lifecycle_manager]: Activating velocity_controller... OK
[INFO] [velocity_controller]: State: INACTIVE  ACTIVE
[INFO] [velocity_controller]: Now publishing to /cmd_vel

[INFO] [odometry_calculator]: State: UNCONFIGURED  CONFIGURING
[INFO] [odometry_calculator]: Differential drive parameters loaded
[INFO] [odometry_calculator]: State: INACTIVE  ACTIVE

[INFO] [sensor_fusion]: State: UNCONFIGURED  CONFIGURING
[INFO] [sensor_fusion]: Complementary filter ready (alpha=0.98)
[INFO] [sensor_fusion]: State: INACTIVE  ACTIVE

[INFO] [watchdog]: Monitoring 3 lifecycle nodes
[INFO] [watchdog]: Heartbeat timeout: 3.0 seconds
[INFO] [watchdog]: All nodes healthy
```

Key Difference: Lifecycle state transitions visible!

4.5 Test Lifecycle Management

Open Terminal 2:

```
source install/setup.bash

# Check lifecycle state
ros2 lifecycle get /velocity_controller

# Output: active [3]
```

Manual Lifecycle Control:

```
# Deactivate node (stop publishing)
ros2 lifecycle set /velocity_controller deactivate

# Check state
ros2 lifecycle get /velocity_controller
# Output: inactive [2]

# Verify no messages published
ros2 topic hz /cmd_vel
# Output: no new messages received
```

Reactivate:

```
ros2 lifecycle set /velocity_controller activate

# Check state
ros2 lifecycle get /velocity_controller
# Output: active [3]

# Verify messages resume
ros2 topic hz /cmd_vel
# Output: average rate: 10.0 Hz
```

Success: Lifecycle transitions work correctly

Screenshot Opportunity: Lifecycle state changes

4.6 Test Watchdog Monitoring

The watchdog monitors all nodes and detects failures.

Simulate Node Failure:

```
# Terminal 2: Kill a node
ros2 lifecycle set /odometry_calculator shutdown

# Watch watchdog response in Terminal 1:
# [WARN] [watchdog]: odometry_calculator missed 1 heartbeat
# [WARN] [watchdog]: odometry_calculator missed 2 heartbeats
# [ERROR] [watchdog]: odometry_calculator missed 3 heartbeats - FAILED
# [WARN] [watchdog]: Emergency stop triggered
# [INFO] [velocity_controller]: Received emergency stop command
# [INFO] [velocity_controller]: Publishing zero velocity
```

Success: Watchdog detects failure and triggers emergency stop

4.7 Test Safety Features

Open Terminal 3:

```
# Monitor safety status
ros2 topic echo /safety_status
```

Test 1: Velocity Limiting

```
# Try to command excessive velocity
ros2 topic pub /cmd_vel_raw geometry_msgs/Twist "linear: {x: 10.0}" -1

# Check actual command sent
ros2 topic echo /cmd_vel
```

Expected: Velocity clamped to max (0.22 m/s in production config)

```
linear:
  x: 0.22  # Limited! (not 10.0)
```

Test 2: Acceleration Limiting

```
# Command sudden change
ros2 topic pub /cmd_vel_raw geometry_msgs/Twist "linear: {x: 0.5}" -1

# Watch actual velocity ramp up gradually
ros2 topic echo /cmd_vel --once
```

Expected: Smooth acceleration (not instant jump)

Test 3: Timeout Detection

```
# Stop publishing commands
# (Just wait 2 seconds)

# Watch logs in Terminal 1:
# [WARN] [safety_monitor]: cmd_vel timeout (0.5s exceeded)
# [INFO] [velocity_controller]: Timeout detected, stopping robot
# [INFO] [velocity_controller]: Publishing zero velocity
```

Success: Safety limits enforced

4.8 Test Docker Deployment

Build Docker Image:

```
cd src/elena_bumperbot_pkg/docker

# Build production image
docker build -t bumperbot:production --target production .

# Expected: Multi-stage build completes
# Successfully built abc123def456
# Successfully tagged bumperbot:production
```

Run in Docker:

```
# Start container
docker-compose up bumperbot_production

# Expected output in container logs:
# bumperbot_production | [INFO] [lifecycle_manager]: Starting...
# bumperbot_production | [INFO] [velocity_controller]: State: ACTIVE
# ...
```

Verify from Host:

```
# Check container is running
docker ps

# Should see:
# CONTAINER ID   IMAGE                      STATUS
# abc123def456   bumperbot:production       Up 30 seconds
```

Stop Docker:

```
docker-compose down
```

Success: Docker deployment works

4.9 Test Production vs Development Configs

Production Config (conservative): - Max velocity: 0.22 m/s - Timeout: 0.5s - Safety: Strict

Development Config (permissive): - Max velocity: 0.35 m/s - Timeout: 2.0s - Safety: Lenient

Test:

```
# Launch with development config
ros2 launch elena_bumperbot_pkg development.launch.py

# Try higher velocity
ros2 topic pub /cmd_vel geometry_msgs/Twist "linear: {x: 0.3}" -1

# Check actual command
ros2 topic echo /cmd_vel

# Expected: 0.3 m/s allowed (vs 0.22 in production)
```

Compare Configs:

Parameter	Production	Development
Max Linear Vel	0.22 m/s	0.35 m/s
Max Angular Vel	2.84 rad/s	4.0 rad/s
Cmd Timeout	0.5s	2.0s
Auto-Recovery	Disabled	Enabled

4.10 Elena's Package Summary

What You Tested: - [x] Unit tests (40+ tests) - [x] Lifecycle management - [x] Watchdog monitoring - [x] Safety features (3 types) - [x] Docker deployment - [x] Production vs development configs

Your Results:

Metric	Expected	Your Result
Tests Passed	40/40	_____ / 40
Lifecycle States	working	_____
Watchdog Detection	detects failures	_____

Velocity Limiting	enforced	_____
Docker Build	success	_____

Enterprise Features: - [ ] Lifecycle management - [ ] Watchdog monitoring - [ ] Multi-layer safety - [ ] Unit test coverage (80%+) - [ ] Docker deployment - [ ] Production/dev configs - [ ] Structured logging

Rating: _____ / 10

5. Comparison & Results

5.1 Side-by-Side Comparison

Feature	Sarah (Beginner)	Marcus (Intermediate)	Elena (Expert)
Nodes	8	7	4
Build Time	_____ s	_____ s	_____ s
CPU Usage	_____ %	_____ %	_____ %
Memory	_____ MB	_____ MB	_____ MB
Accuracy	_____ %	_____ %	_____ %
Safety	Basic	Moderate	Multi-layer
Monitoring	None	System + Perf	Watchdog
Lifecycle	No	No	Yes
Multi-Robot	No	Yes	Yes
Docker	No	No	Yes
Tests	None	10+	40+
Your Rating	_____ /10	_____ /10	_____ /10

5.2 Performance Summary

CPU Usage (lower is better):

```
Sarah:   ~20%
Marcus:  ~15%
Elena:   ~14%
```

Memory Usage (lower is better):

```
Sarah:   ~125 MB
Marcus:  ~100 MB
Elena:   ~110 MB
```

Feature Count (more is better):

```
Sarah:   8 features
```

```
Marcus:  12 features
Elena:   20+ features
```

5.3 Use Case Recommendations

Choose Sarah's Package If: - You're learning ROS2 for the first time - You want simple, understandable code - You're teaching beginners - You need quick results

Your Experience:________

Choose Marcus's Package If: - You have ROS2 experience - You need optimized performance - You're building prototypes - You need multi-robot support

Your Experience:________

Choose Elena's Package If: - You're deploying to production - Safety is critical - You need enterprise features - You want Docker deployment

Your Experience:________

6. Troubleshooting

6.1 Build Errors

Error:package 'robot_localization' not found

Solution:

```
sudo apt install ros-humble-robot-localization
```

Error:colcon: command not found

Solution:

```
sudo apt install python3-colcon-common-extensions
```

Error:ModuleNotFoundError: No module named 'rclpy'

Solution:

```
source /opt/ros/humble/setup.bash
```

6.2 Runtime Errors

Error: Nodes not starting

Solution:

```
# Re-source workspace
source install/setup.bash

# Check if nodes are executable
ls -la install/*/lib/*/
```

Error:/cmd_vel not being received

Solution:

```
# Check if topic exists
ros2 topic list | grep cmd_vel

# Check if anyone is publishing
ros2 topic info /cmd_vel

# Manually publish to test
ros2 topic pub /cmd_vel geometry_msgs/Twist "linear: {x: 0.1}" -1
```

Error: EKF node not starting

Solution:

```
# Install robot_localization
sudo apt install ros-humble-robot-localization

# Check config file exists
ls install/*/share/*/config/ekf_config.yaml
```

Error: Docker build fails

Solution:

```
# Update Docker
sudo apt update
sudo apt install docker.io docker-compose

# Check Docker service
sudo systemctl status docker

# Build with verbose output
docker build -t bumperbot:production --target production --progress=plain .
```

6.3 Performance Issues

Issue: High CPU usage

Solution: - Reduce update rates in config files - Check for infinite loops in custom code - Monitor with htop or top

Issue: High latency

Solution: - Use BEST_EFFORT QoS for high-frequency topics - Reduce queue sizes - Check network congestion (if using remote ROS)

6.4 Getting Help

Documentation: - Main README: d:/ROS2_PROJECT/README.md - Tutorial: docs/tutorials/ROBOSHIRE_SELF_DRIVING_TUTORIAL.md - Package READMEs: workspace/src/*/README.md

Community: - RoboShire Discord: [community chat] - GitHub Issues: https://github.com/yourusername/roboshire/issues - ROS Answers: https://answers.ros.org/

Debugging:

```
# View logs
ros2 run rqt_console rqt_console

# Node diagnostics
ros2 node info /node_name

# Check TF tree
ros2 run tf2_tools view_frames
evince frames.pdf

# Topic debugging
ros2 topic echo /topic_name
ros2 topic hz /topic_name
ros2 topic bw /topic_name
```

7. Conclusion

7.1 Summary

You have successfully tested three complete ROS2 packages created by AI agents Sarah, Marcus, and Elena. Each package demonstrates different skill levels and use cases:

- Sarah's Package: Educational, beginner-friendly
- Marcus's Package: Optimized, production-prototype
- Elena's Package: Enterprise-grade, safety-critical

7.2 Key Learnings

Technical Skills: - Building ROS2 packages with colcon - Launching multi-node systems - Monitoring topics and nodes - Testing odometry accuracy - Lifecycle management - Docker deployment

Robotics Concepts: - Differential drive kinematics - Wheel encoder odometry - IMU sensor fusion - Extended Kalman Filter - Transform frames (TF2) - Safety monitoring

7.3 Next Steps

Continue Learning: 1. Modify the packages for your own robot 2. Add new sensors (LiDAR, camera) 3. Integrate with Nav2 for navigation 4. Implement behavior trees 5. Deploy to real hardware

Advanced Topics: - SLAM (Simultaneous Localization and Mapping) - Path planning and obstacle avoidance - Computer vision integration - Multi-robot coordination - Cloud robotics

7.4 Certificate of Completion

Fill this out after completing all tests:

RoboShire Simulation Testing Certificate

I, ___, have successfully completed the BumperBot Simulation Testing Guide on ____ (date).

Packages Tested: - [ ] Sarah's Beginner Package (Rating: _ /10) - [ ] Marcus's Intermediate Package (Rating: /10) - [ ] Elena's Enterprise Package (Rating: __ /10)

Skills Acquired: - [ ] ROS2 package building - [ ] Multi-node system launching - [ ] Odometry accuracy testing - [ ] Lifecycle management - [ ] Docker deployment - [ ] Performance profiling

Odometry Accuracy Achieved: _____ % improvement

Signature:_____

7.5 Feedback

Please provide feedback to improve this tutorial:

What worked well:

What was confusing:

Suggestions:

Would you recommend this tutorial? (1-10): _____

Appendix A: Quick Reference Commands

Build Commands

```
colcon build --packages-select sarah_bumperbot_pkg
colcon build --packages-select marcus_bumperbot_pkg
colcon build --packages-select elena_bumperbot_pkg
source install/setup.bash
```

Launch Commands

```
# Sarah's
ros2 launch sarah_bumperbot_pkg bumperbot_bringup.launch.py

# Marcus's (3 options)
ros2 launch marcus_bumperbot_pkg bumperbot_full.launch.py
ros2 launch marcus_bumperbot_pkg sensors_only.launch.py
ros2 launch marcus_bumperbot_pkg controllers_only.launch.py

# Elena's (3 configs)
ros2 launch elena_bumperbot_pkg production.launch.py
ros2 launch elena_bumperbot_pkg development.launch.py
ros2 launch elena_bumperbot_pkg simulation.launch.py
```

Monitoring Commands

```
ros2 topic list
ros2 topic echo /odom
ros2 topic echo /odom/filtered
ros2 topic hz /cmd_vel
ros2 node list
ros2 node info /velocity_controller
ros2 run tf2_tools view_frames
```

Testing Commands

```
python3 test_square_path.py
pytest test/unit/ -v
./scripts/run_tests.sh
```

Docker Commands

```
docker build -t bumperbot:production --target production .
docker-compose up bumperbot_production
docker ps
docker logs bumperbot_production
docker-compose down
```

Appendix B: Resources

Documentation

- ROS2 Humble: https://docs.ros.org/en/humble/
- robot_localization: http://docs.ros.org/en/humble/p/robot_localization/
- MuJoCo: https://mujoco.readthedocs.io/

Tutorials

- RoboShire Self-Driving Tutorial: docs/tutorials/ROBOSHIRE_SELF_DRIVING_TUTORIAL.md
- Quick Reference: docs/tutorials/QUICK_REFERENCE_ODOMETRY_CONTROL.md

Example Code

- Sarah's package: workspace/src/sarah_bumperbot_pkg/
- Marcus's package: workspace/src/marcus_bumperbot_pkg/
- Elena's package: workspace/src/elena_bumperbot_pkg/

End of Tutorial

Total Pages: ~35 pages (when converted to PDF) Estimated Completion Time: 2-3 hours Difficulty: Beginner-Friendly with Advanced Options

Thank you for testing RoboShire!

Version 1.0 | October 23, 2025