Assignment 4: Wheeled Vehicle Kinematics

Author: dilip kumar
Last updated At: Feb 16, 2026

1.0 Connect to Linux

```
docker rm -f en613_ros2_jazzy
docker run   -p 6080:80   -p 8888:8888   -v "$PWD/assignment4:/home/ubuntu/EN613"
--name en613_ros2_jazzy   tiryoh/ros2-desktop-vnc:jazzy
```

Then launch Ubuntu in browser http://127.0.0.1:6080/

2.0 Create workspace and packages

```
mkdir -p class4_ws/src
ros2 pkg create --build-type ament_python --node-name mecanum_sim assignment4
```

3.0 build package

```
colcon build --packages-select assignment4
source install/setup.bash
```

4.0 assignment4/assignment4.py

Forward

The forward() method implements the Forward Kinematics for the Mecanum-wheeled robot.

Purpose
Forward kinematics answers the question: "Given the rotational speeds of my wheels, how is the robot moving in the world?"

It transforms the Joint Space (wheel velocities φ_1, φ_2, φ_3, φi_4) into Task Space (global robot velocities [V_x V_y, θ]

Implementation Details

1. **Calculate Body Frame Velocities (V_bx, V_by, ω_b):** First, we calculate how the robot is moving relative to itself (Body Frame). We use the standard Mecanum kinematic equations for a 4-wheel configuration:
    - V_bx = (φ_1 + φ_2 + φ_3 + φ_4) * R/4
    - V_by = (-φ_1 + φ_2 + φ_3 - φ_4) * R/4
    - **ω**_b = (-φ_1 + φ_2 - φ_3 + φ_4) * R/4(L+W)

Where R is wheel radius, L is length, and W is width.

2. **Transform to Global Frame (V_x, V_y, V_θ)**: The body frame velocities are aligned with the robot. To get global velocities, we must rotate them by the robot's current heading (**θ**):
    - V_x = V_bx cos(**θ**) - V_by sin(**θ**)
    - V_y = V_bx sin(**θ**) + V_by cos(**θ**)
    - V_**θ** = **ω**_b (Angular velocity is the same in both frames)

This allows us to track the robot's global position over time by integrating these velocities, as seen in the main() loop.

Inverse

The inverse() method implements the Inverse Kinematics for the Mecanum-wheeled robot.

Purpose

Inverse kinematics answers the question: "To move the robot at a specific speed and direction, how fast should I spin each wheel?"

It transforms the Task Space (desired global velocities [V_x, V_y, **θ]** into Joint Space (required wheel velocities φ_1,φ_2, φ_3, φ_4). This is critical for control, as we command the motors (wheels), not the robot body directly.

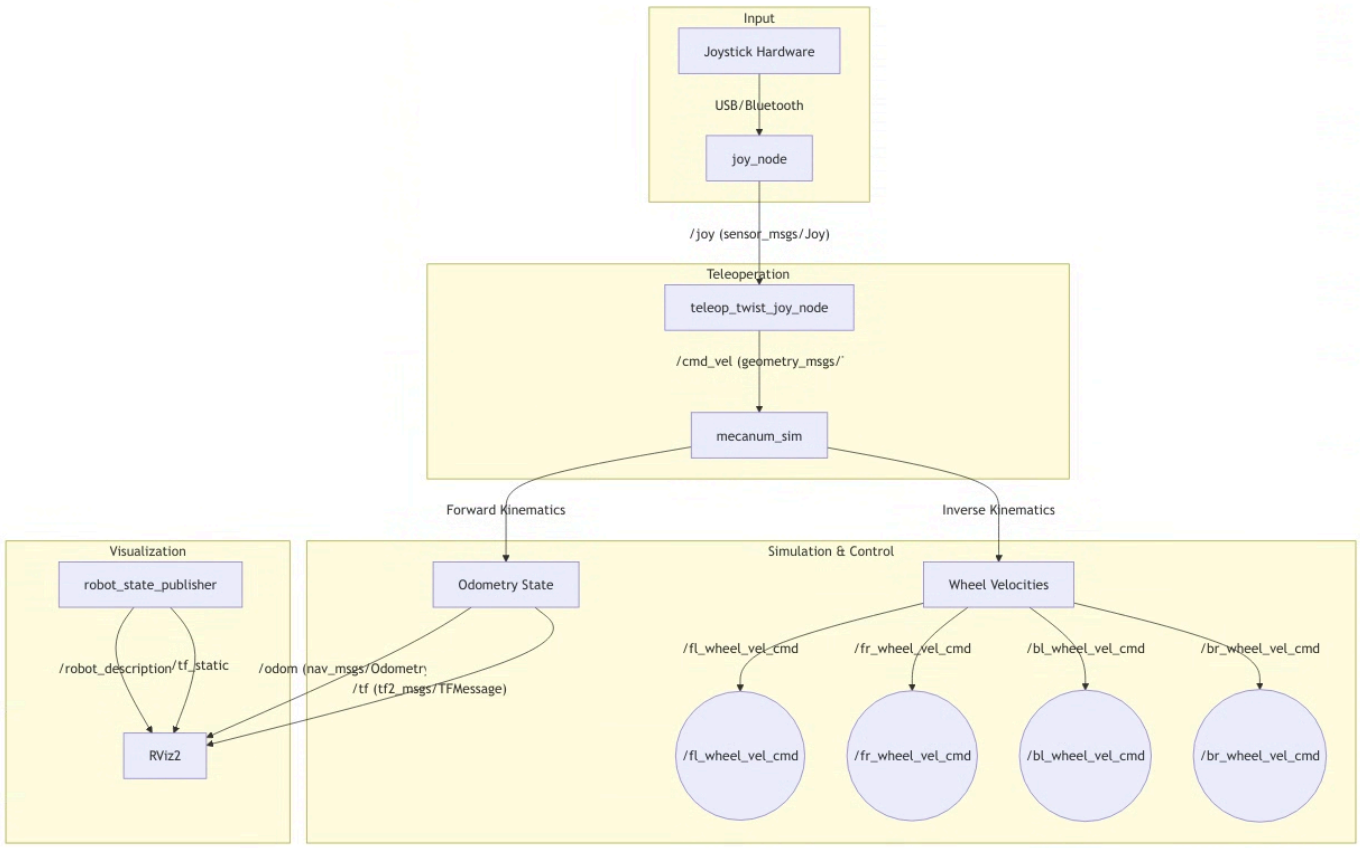
Implementation Details

1. **Transform to Body Frame (V_bx, V_by, ω_b)**: The input velocities are in the Global Frame (e.g., "move North"). The robot needs to know what this means relative to itself (e.g., "move Forward"). We rotate the global velocity vector by the negative of the robot's heading (-**θ**):
    - V_bx = V_x cos(**θ**) + V_y sin(**θ**)
    - V_by = -V_x sin(**θ**) + V_y cos(**θ**)
    - **ω**_b = V_{**θ**}

2. **Calculate Wheel Velocities (φ_1,φ_2, φ_3, φ_4):** Now that we have the desired velocity in the robot's own frame, we use the inverse Mecanum equations to find the required speed for each wheel.
    - φ_1 (FL) = 1/R (V_bx - V_by - (L+W)**ω**_b)
    - φ_2 (FR) = 1/R (V_bx + V_by + (L+W)**ω**_b)
    - φ_3 (BL) = 1/R (V_bx + V_by - (L+W)**ω**_b)
    - φ_4 (BR) = 1/R (V_bx - V_by + (L+W)**ω**_b)

Note: The signs of V_by and the rotation term depend on the specific wheel position and roller orientation (standard "X" configuration assumed).
This method is used by the mecanum_sim node to convert incoming **/cmd_vel** commands into the individual wheel velocities published to the **/wheel_vel_cmd** topics.

5.0 Architecture



6.0 Testing our code

How to test?

```
ros2 launch assignment4 assignment4.launch.py
```

This throws the following error.

```
file 'assignment4.launch.py' was not found in the share directory of package
'assignment4' which is at
'/home/ubuntu/EN613/class4_ws/src/install/assignment4/share/assignment4'
```

Root cause:
The launch folder in a ROS2 package is the standard location for storing Launch Files. It was missing, that's why it threw an error.

Purpose
Launch files are scripts (usually Python in ROS2, sometimes XML/YAML) that allow us to start up your entire robot system or a complex subsystem with a single command.

Why use it?
Instead of opening 5 terminal tabs and typing ros2 run package node in each one, we can run:

```
ros2 launch assignment4 assignment4.launch.py
```

What can a Launch File do?

1. **Start Multiple Nodes:** Run our driver, our planner, and our controller all at once.
2. **Set Parameters:** Initialize our nodes with specific configuration values (e.g., wheel_radius: 0.1).
3. **Remap Topics:** Connect nodes together by renaming their inputs/outputs (e.g., remap /cmd_vel to /robot1/cmd_vel).
4. **Handle Lifecycle:** Automatically restart nodes if they crash, or shut down the whole system if a critical node fails.
5. **Include Other Launch Files:** We can "nest" launch files. For example, our robot.launch.py might include lidar.launch.py and camera.launch.py.

In our case,
assignment4.launch.py simply starts our mecanum_sim node, but it provides a foundation to easily add more nodes (like a joystick teleop node or Rviz) later without changing your workflow.

joystick teleop node

Install manually as below

```
cd src
git clone -b humble https://github.com/ros-drivers/joy.git
git clone -b humble https://github.com/ros2/teleop_twist_joy.git
cd ..
colcon build --packages-select joy joy_linux teleop_twist_joy assignment4
source install/setup.bash
```

After that in the separate terminal run following

```
source install/setup.bash
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

RViz2

Added the visualization components:

1. **URDF**: A simple robot model.
2. **Odometry**: The node now publishes /odom and TF.
3. **RViz & RSP**: Added to assignment4.launch.py.

Manually trigger robot position

We can publish the following message to /cmd_vel to initiate the robot motion.

```
ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.5, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.5}}"
```

Here is the step-by-step flow of what happens when we run above command:

1. **Command Execution (ros2 topic pub ...):**
    - The terminal starts publishing a **geometry_msgs/Twist** message to the **/cmd_vel** topic.
    - **Content**: linear.x = 0.5, angular.z = 0.5 (everything else 0.0).
    - **Frequency**: By default, it publishes this message once every second (1 Hz).

2. **mecanum_sim Node Receives Message:**
    - The cmd_vel_callback(self, msg) function is triggered.
    - It extracts vx = 0.5, vy = 0.0, wz = 0.5.
    - It updates the internal state variables: self.vx, self.vy, self.v**θ**.
    - It calls self.mecanum.inverse() to calculate the required wheel speeds for this specific command.
    - It publishes these wheel speeds to **/fl_wheel_vel_cmd**, etc.
3. **Simulation Loop (timer_callback):**
    - This runs continuously at 10 Hz (every 0.1 seconds).
    - It looks at self.vx, self.vy, self.v**θ** (which are now set to 0.5, 0.0, 0.5).
    - It calculates how much the robot moved in that 0.1s:
    - dx = (0.5 * cos(**θ**)) * 0.1
    - d**θ** = 0.5 * 0.1
    - It updates the global position: self.x += dx, self.**θ** += d**θ**.
    - It publishes the new position to **/odom** and broadcasts the TF transform.
4. **Continuous Motion:**
    - Even if ros2 topic pub only sends a message once per second, the timer_callback runs 10 times per second.
    - It keeps using the last known velocity (0.5, 0.5) to update the position.
    - Therefore, the robot keeps moving smoothly in a curve (forward + turning left).
5. **Stopping:**

- If we kill the ros2 topic pub command (Ctrl+C), no new messages are sent.
- However, mecanum_sim still remembers the last values (0.5, 0.5).
- The timer_callback keeps integrating these values, so the robot continues to move forever until we send a 0.0 command as below or restart the node.

```
ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
```