ROS2 Foundations
Day 3

Transformations using tf2

May 29, 2025

TF2 – Coordinate Frame Transforms

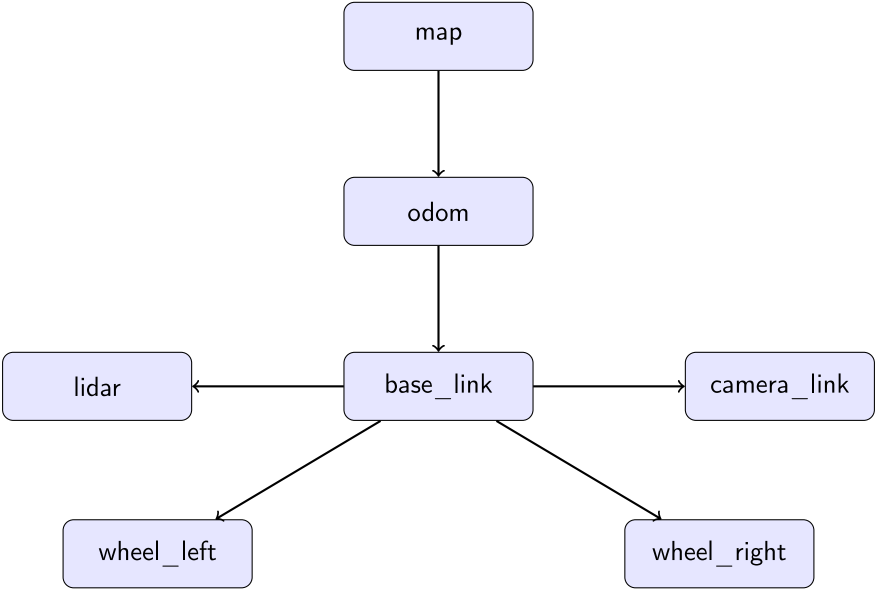
What is TF2?

- TF2 manages a dynamic **tree of coordinate frames** over time.
- Each frame defines a spatial transform: position + orientation relative to a parent.
- Used to transform data (e.g., sensor readings) between frames.

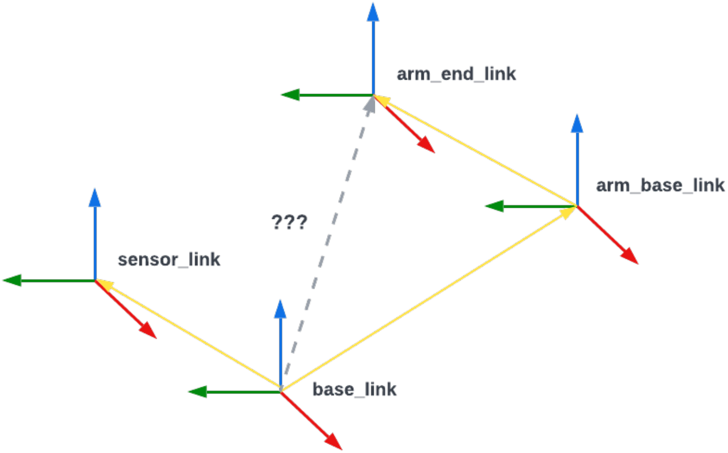
Core Concepts

- Tree structure: `map` → `odom` → `base_link` → `camera_link`, etc.
- All transforms are **timestamped**, enabling interpolation over time.
- Transforms are published by nodes (e.g., robot state publishers, sensor drivers).

TF2 Frame Tree – Example Robot



TF2 Frame Tree – Example Robot



Source: Understanding ROS Transforms

TF2 - What do we need?

The *tf2* package is used for transformations. This package uses a special message type: *geometry_msgs/TransformStamped*.
This type includes:

- **Header header**: current timestamp and the parent frame id
- **string child_frame_id**: child frame id
- **Transform transform**: transformation from coordinate frame header.frame_id to the coordinate frame child_frame_id

TF2 - Static and Dynamic

TF2 supports two types of transformations:

Static Transformations

- Used when the relationship between two frames does not change over time.
- Published once and stored in the TF2 tree.
- Example: A sensor rigidly attached to a robot.
- Use: `static_transform_publisher` or via launch files.

Dynamic Transformations

- Used when the relationship between two frames changes over time.
- Continuously updated by a broadcaster (e.g. robot movement).
- Example: A moving robot base with respect to the map frame.
- Implemented using a `TransformBroadcaster` in code.

TF2 Frame – Broadcaster 1/2

class Broadcaster(Node):

```
class FrameBroadcaster(Node):
    def __init__(self):
        super().__init__('frame_broadcaster')
        self.br = TransformBroadcaster(self)
        self.timer =
            self.create_timer(0.5, self.broadcast_frame)
```

TF2 Frame – Broadcaster 2/2

class Broadcaster(Node):

```
    def broadcast_frame(self):
        t = TransformStamped()
        t.header.stamp = self.get_clock().now().to_msg()
        t.header.frame_id = 'base_link'
        t.child_frame_id = 'camera_link'
        t.transform.translation.x = 0.1
        t.transform.translation.y = 0.2
        t.transform.translation.z = 0.3
        t.transform.rotation.x = 0.0
        t.transform.rotation.y = 0.0
        t.transform.rotation.z = 0.0
        t.transform.rotation.w = 1.0
        self.br.sendTransform(t)
```

TF2 Frame – Listener 1/2

Transform Broadcaster Node

```
class FrameListener(Node):
    def __init__(self):
        super().__init__('frame_listener')
        self.tf_buffer = Buffer()
        self.listener =
            tf2_ros.TransformListener(self.tf_buffer, self)
        self.timer = self.create_timer(1.0, self.lookup)
```

TF2 Frame – Listener 2/2

Transform Broadcaster Node

```
def lookup(self):
    try:
        now = rclpy.time.Time()
        t = self.tf_buffer.lookup_transform(
            'base_link', 'camera_link', now)
        self.get_logger().info(
        f'Transform: {t.transform.translation}'
        )
    except Exception as e:
        self.get_logger().warn(f'Could not transform: {e}')
```

Launching TF2 Broadcaster and Listener

Terminal Commands:

Start both nodes in separate terminals

```
# Terminal 1 - Broadcast a transform
ros2 run my_package broadcaster_node

# Terminal 2 - Listen and print the transform
ros2 run my_package listener_node
```

TF2 - Coordinate Frame Transform Tools

Helpful Tools

- `rviz2`: visualize TF tree
- `ros2 run tf2_tools view_frames.py`: generate TF graph
- `ros2 run tf2_ros tf2_echo frame1 frame2`: live transform info

Important: TF must be a directed tree. **No loops allowed!**

....

TF_Demo - Create Package

We create the new package tf_demo:

Run in terminal

```
ros2 pkg create –build-type ament_python tf_demo
```

In this package we will implement following things:

- Dynamic Transform:
  A transform from the frame 'odom' to 'base_link' with the transformation being a circle.
- Static Transform:
  A static transform from the frame 'base_link' to 'camera_link'.
- Launch file:
  Start the two transform publisher nodes together with rviz.

TF_Demo - Node Creation

For our first static node we need to create a new file in tf_demo/tf_demo with the name static_tf_pub.py.

Write the class StaticTFPublisher(Node):

```
def __init__(self):
    super().__init__('static_tf_pub')
    broadcaster = StaticTransformBroadcaster(self)
    t = TransformStamped()
    t.header.stamp = self.get_clock().now().to_msg()
    t.header.frame_id = 'base_link'
    t.child_frame_id = 'camera_link'
    t.transform.translation.x = 0.5
    t.transform.translation.y = 0.0
    t.transform.translation.z = 0.2
    t.transform.rotation.w = 1.0  # no rotation
    broadcaster.sendTransform(t)
```

TF_Demo - Node Creation

For our dynamic node we need to create a new file at the same place as before, namly, tf_demo/tf_demo with the name dynamic_tf_pub.py.

Write the class DynamicTFPublisher(Node):

```
def __init__(self):
    super().__init__('dynamic_tf_pub')
    self.broadcaster = TransformBroadcaster(self)
    # Let's run our timer at 20Hz
    self.timer = self.create_timer(0.05, self.timer_callback)
    self.start_time = time.time()
```

TF_Demo - Node Creation

Write the class DynamicTFPublisher(Node):

```
def timer_callback(self):
    now = self.get_clock().now().to_msg()
    elapsed = time.time() - self.start_time

    t = TransformStamped()
    t.header.stamp = now
    t.header.frame_id = 'odom'
    t.child_frame_id = 'base_link'

    # To be continued...
```

TF_Demo - Node Creation

Write the class DynamicTFPublisher(Node):

```
    # Continuation ...

    # Move in a circle
    radius = 1.0
    t.transform.translation.x = radius * math.cos(elapsed)
    t.transform.translation.y = radius * math.sin(elapsed)
    t.transform.translation.z = 0.0
    t.transform.rotation.z = math.sin(elapsed / 2)
    t.transform.rotation.w = math.cos(elapsed / 2)

    self.broadcaster.sendTransform(t)
```

TF_Demo - Node Creation

Important: Do not forget to add the main() function for both nodes!

Main function in both Nodes:

```
def main():
    rclpy.init()
    node = <NodeClassName>()
    rclpy.spin(node)
    rclpy.shutdown()


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

Replace <> with either StaticTFPublisher or DynamicTFPublisher.

Practice: Build, Simulate, Teleop

Project Goal:

- TODO

TF & Simulation Debugging Tips

- `ros2 run tf2_tools view_frames` → generates PDF frame tree.
- `ros2 topic echo /tf` to verify publishing.
- `ros2 param list` + get to check sim config.
- `gazebo` –verbose for plugin load errors.
- `ros2 bag record -a` to capture all data.

TF & Simulation Debugging Tips

- `ros2 run tf2_tools view_frames` → generates PDF frame tree.
- `ros2 topic echo /tf` to verify publishing.
- `ros2 param list` + `get` to check sim config.
- `gazebo` −verbose for plugin load errors.
- `ros2 bag record -a` to capture all data.

Meshes, Visuals and Collisions

- **Visual:** high-res meshes (.dae, .stl) used in RViz or Gazebo for rendering.
- **Collision:** simplified shapes (boxes, cylinders) used for physics.
- **Material:** defines color/texture, only visible in simulation tools.
- Use in and .