TF2: Coordinate Frames & Transforms

Course: Robotic Programming Environments

Michał Drwięga
michal.drwiega@pwr.edu.pl
www.mdrwiega.com/edu/rpe

Department of Cybernetics and Robotics
Wrocław Univeristy of Science and Technology

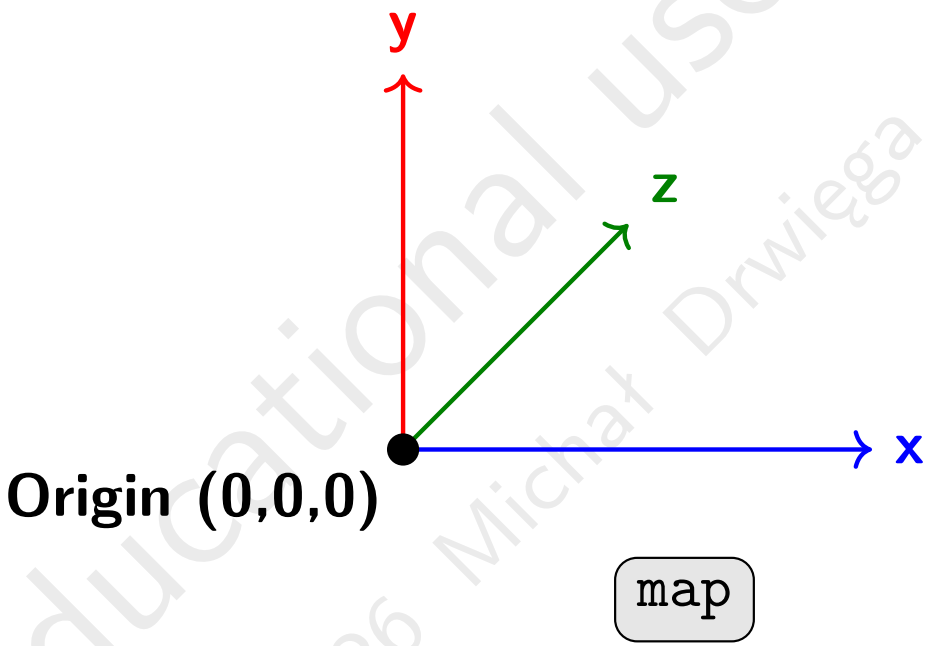
Wrocław University
of Science and Technology

Summary

Prerequisites

What is a Coordinate Frame?

- A reference system for describing positions and orientations in space

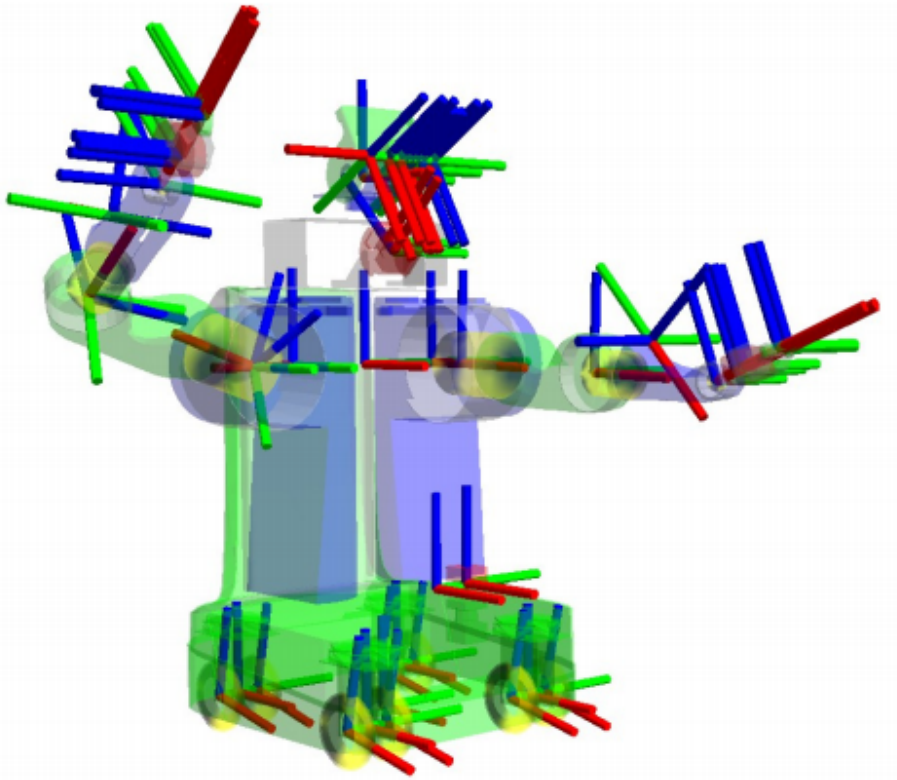


Defined by:

- **Origin**: Position (x, y, z) in space
- **Orientation**: Quaternion e.g. (q_x, q_y, q_z, q_w)

Why do we need coordinate systems?

- Robots use multiple sensors:
  - Camera
  - LiDAR
  - IMU
- Each sensor has its own reference coordinate system

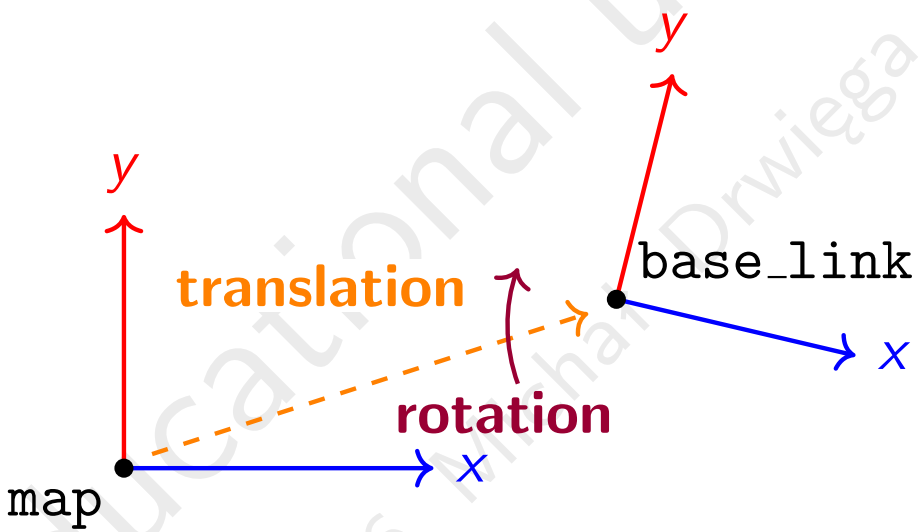


Problem:

- How to relate all these measurements?

Transform Between Coordinate Frames

- A **transform** describes how to move from one frame to another
- Combines translation (position) + rotation (orientation)



Transform $T_{map \rightarrow base}$**:**

Transform direction matters: $T_{A \rightarrow B}$ moves points **from** frame A **to** frame B.

Transform Composition

Chaining Transforms

From frame A to C through intermediate frame B:

$$T_{A \to C} = T_{B \to C} \circ T_{A \to B}$$

Position: $\vec{p}_C = \vec{p}_B + R_{B \to C} \cdot \vec{p}_A$
Orientation: $q_C = q_{B \to C} \otimes q_{A \to B}$

Important

Transforms are NOT commutative: $T_{A \to B} \neq T_{B \to A}$

Rotation vs Orientation

Orientation = State

Rotation = Action / Change

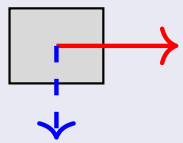
Orientation

- The **current** angular state of a body
- Describes "which way it is facing"
- **Relative to a reference frame**
- Example: "Robot is facing East"

Orientation: 0°

Rotation

- The **process** of changing orientation
- Describes "how it turned"
- **Relative to previous orientation**
- Example: "Robot turned 90° left"

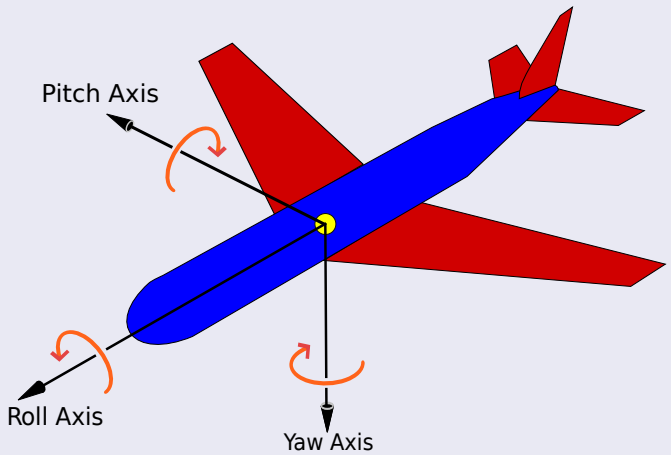


Rotation: -90°

Orientation Representation - Euler Angles / RPY

RPY → Roll, Pitch, Yaw (ZYX Euler angles)

- Three-value orientation representation
- Intuitive for aerospace and mobile robotics
- **Drawback**: Singular configurations (gimbal lock) occur when two axes align, causing loss of one DOF

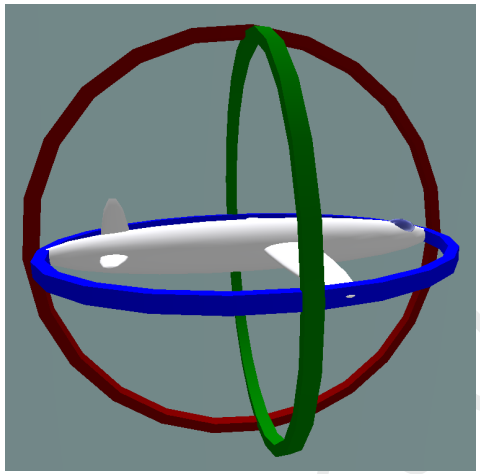


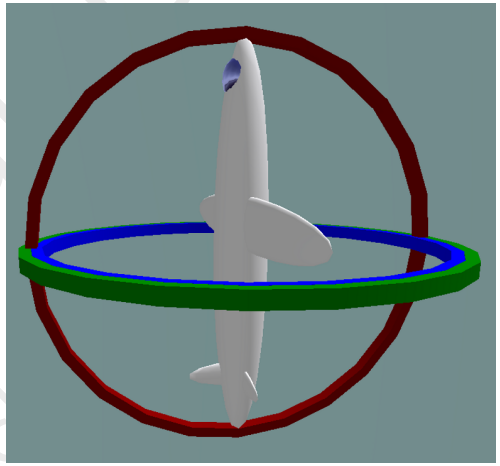
1

[1] https://www.wikiwand.com/en/Aircraft_principal_axes

Gimbal Lock - The Singularity Problem

Gimbal lock – A loss of one degree of freedom in a 3D rotation system when two rotational axes become aligned.





2

When does it happen?

- **Euler angles** representation
- Occurs when pitch $= \pm 90°$ (for XYZ Euler)
- Yaw and roll axes become **aligned**

Consequences

- Cannot distinguish between yaw and roll
- Infinite solutions for the same orientation
- **Singularity** in the math

Orientation Representation - Quaternions

A quaternion is a 4-tuple representation of orientation:

$$q = [x, y, z, w] = [\vec{v}, w]$$

Properties:

- Unit quaternion: $\|q\| = \sqrt{x^2 + y^2 + z^2 + w^2} = 1$
- Conjugate: $q^* = [-x, -y, -z, w]$
- Inverse: $q^{-1} = \frac{q^*}{\|q\|^2}$

Example

Rotation of 90° around Z-axis: $q = [0, 0, 0.7071, 0.7071]$

Orientation Representation - Rotation Matrix

Definition

A rotation matrix performs a rotation in Euclidean space:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

Properties

- **Orthogonality**: $RR^T = R^T R = I_3$
- **Determinant**: $\det(R) = +1$ (preserves orientation)
- **Preserves cross product**: $R(v \times w) = (Rv) \times (Rw)$
- **Can be composed**: $R_{total} = R_2 \cdot R_1$
- **Can be a part of the transformation matrix**

Affine Transformation Matrix

Definition

An affine transformation combines linear transformations (rotation, scaling, shear) and translation in a single matrix form.

Matrix Form (3D)

$$\mathrm{T} = \begin{bmatrix} \mathrm{R} & \mathrm{t} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Properties

- Preserves straight lines and parallelism
- Combines rotation, scaling, shear, and translation
- Enables transformation using matrix multiplication

Affine Transformation Matrix

Application

$$\mathrm{p}' = \mathrm{T}\mathrm{p}, \quad \mathrm{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Widely used in robotics, SLAM, and computer vision
- Allows chaining multiple transformations efficiently

Homogeneous Coordinates

Idea

Homogeneous coordinates extend standard coordinates by adding an extra dimension, enabling translation to be represented as matrix multiplication.

Representation

$$\text{2D: } \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \text{3D: } \begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Why Use Them?

- Allow translation to be written as matrix multiplication
- Unify rotation, scaling, and translation in one framework
- Enable easy chaining of transformations

Rotation Representations - Comparison

Feature	RPY / Euler	Quaternion	Rot. Matrix
Number of parameters	3	4	9
Singularities	Yes	No	No
Intuitive interpretation	Yes	Difficult	Moderate
Computational efficiency	High	Very high	Low
Interpolation	Difficult	Easy (slerp)	Difficult

RPY / Euler

- \+ Min parameters
- \- Singularities
- \+ Intuitive

Quaternions

- \+ No singularities
- \+ Fast interpolation
- \- Hard to interpret

Rotation Matrix

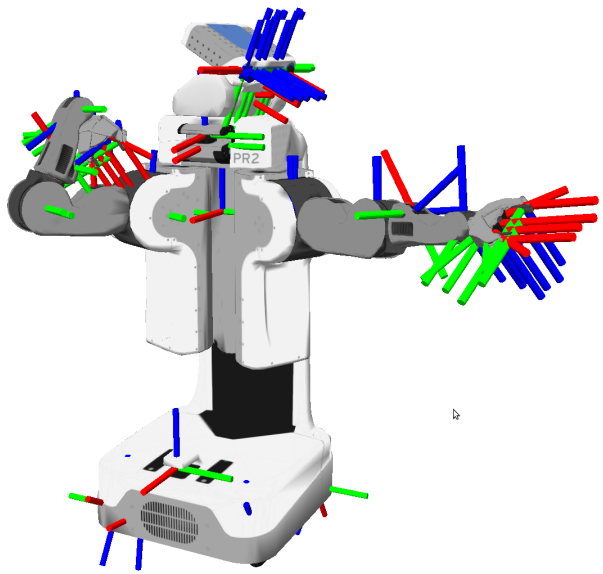
- \+ No singularities
- \+ Works in T matrix
- \- Slow computation

Use **quaternions** for computation/interpolation, **rotation matrices** for transformation composition, **RPY** for human interface.

TF2 Introduction

tf2 (Transform Library) in ROS 2

- Provides a support for managing coordinate systems in a robotic system.
- Enables easy conversion of data between various frames of reference.
- Components: Dynamic Broadcaster, Static Broadcaster, Listener



3 4

[3] http://wiki.ros.org/tf2

[4] https://docs.ros.org/en/humble/Concepts/Intermediate/About-Tf2.html

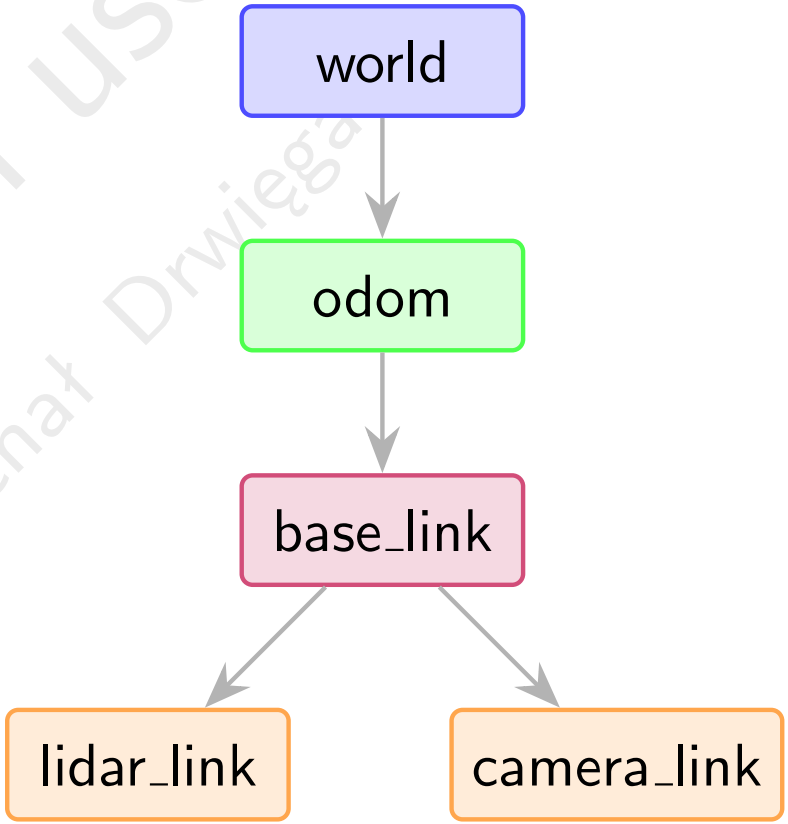
TF2 Core Concepts

- **Frame**: A coordinate system (e.g. base_link)
- **Transform**: Position + orientation between two frames
- **Time**: Transforms have temporal validity
- **Tree Structure**: Frames organized in a directed tree

TF2 Transform Tree Structure

Frames in TF2 form a **tree** (not a graph or mesh)

- Each frame has exactly **one parent**
- A frame can have **multiple children**
- Transform direction: Parent → Child
- No cycles or loops allowed!

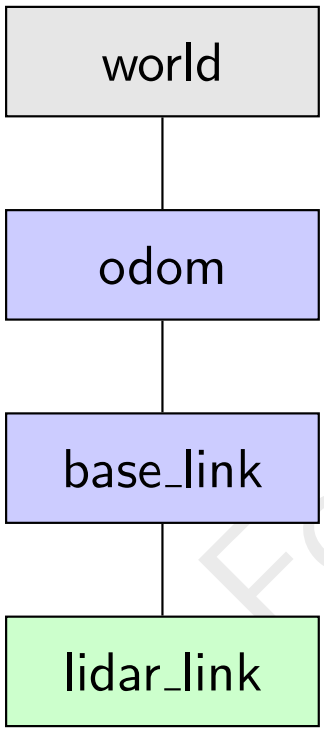


Transform Tree: Static vs Dynamic Frames

Tree Organization Principle

Frames are organized by **transform type**:

- Static transforms: Fixed relationships (sensor mounting)
- Dynamic transforms: Change over time (robot movement)



Transform	Type	Update Rate
world → odom	Dynamic	1-10 Hz
odom → base_link	Dynamic	10-100 Hz
base_link → lidar_link	Static	Once (on startup)

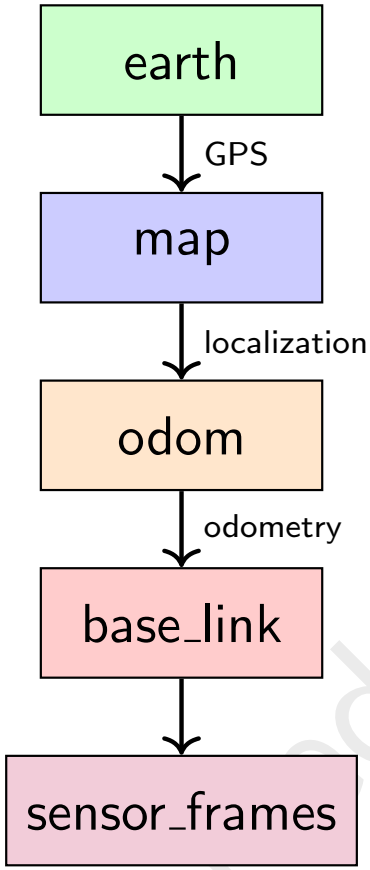
REP: ROS Enhancement Proposals for TF2

What are REPs?

ROS Enhancement Proposals are design documents that define **standards, conventions, and best practices** for the ROS ecosystem.

REP	**Title**	**Key TF2 Content**
REP 105	Coordinate Frames for Mobile Platforms	map, odom, base_link, earth frames

REP 105: Coordinate Frames for Mobile Robots



earth

- Global fixed frame

map

- World-aligned frame

odom

- Continuous frame
- Dead reckoning

base_link

- Robot centroid

sensor_frames

- `laser_frame`, `camera_frame`

Core TF2 Packages

- `tf2`[5]
  - Core library for handling coordinate transforms
  - Data structures, buffering, and time handling
- `tf2_ros`
  - ROS 2 interface for tf2 (broadcasters and listeners)
- `tf2_msgs`
  - TF2 messages definitions, e.g. TFMessage.msg
- `tf2_tools`
  - Debugging and visualization tools: e.g. `view_frames`
- `tf2_geometry_msgs`
  - Transform support for geometry messages (e.g. `TransformStamped`)

Additional integrations:

- `tf2_sensor_msgs` (PointCloud, LaserScan)
- `tf2_eigen`, `tf2_kdl` (math libraries)

[5] https://github.com/ros2/geometry2

TransformStamped Message

Represents a time-stamped transform between two coordinate frames.

```
std_msgs/Header header
  stamp          # time of transform
  frame_id       # parent frame

string child_frame_id

geometry_msgs/Transform transform
  geometry_msgs/Vector3 translation
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion rotation
    float64 x
    float64 y
    float64 z
    float64 w
```

Dynamic TF Broadcaster in ROS2

```python
class DynamicBroadcaster(Node):
  def __init__(self):
    super().__init__('dynamic_broadcaster')
    self.tfb_ = TransformBroadcaster(self)

  def broadcast_pose(self, pose):
    tfs = TransformStamped()
    tfs.header.stamp = self.get_clock().now().to_msg()
    tfs.header.frame_id="world"
    tfs._child_frame_id = "base_link"
    tfs.transform.translation.x = pose.position.x
    ...
    r = R.from_euler('xyz',[0,0,pose.orient.theta])
    tfs.transform.rotation.x = r.as_quat()[0]
    ...

    self.tfb_.sendTransform(tfs)
```

Static TF Broadcasting in ROS2

Overview

Static TF broadcasting in ROS 2 is used to establish fixed coordinate transforms between frames that remain constant throughout the robot's operation.

- Defines transforms that do not change during runtime.
- Ideal for representing fixed relationships between frames in a robotic system, e.g. transforms between sensors
- Improves efficiency by eliminating the need for continuous updates.

CLI

```
ros2 run tf2_ros static_transform_publisher 0.1 0 0 -1.57
0.0 0.0 parent_frame child_frame
```

TF Listening in ROS 2

Overview

TF listening in ROS 2 allows a node to receive and use real-time coordinate transforms between frames, providing up-to-date spatial information in a robotic system.

- Listens to TF messages to retrieve transforms dynamically
- Essential for tasks requiring knowledge of spatial relationships in real-time
- Enables adaptability in response to changes in the robot's environment

TF2 LookupTransform Function

Function Signature

```
lookup_transform(
    target_frame: str,
    source_frame: str,
    time: rospy.Time,
    timeout: rospy.Duration
) -> TransformStamped
```

What it returns?

Transform of **source_frame** expressed in **target_frame** coordinates

Usage

```
from tf2_ros import Buffer,
    TransformListener
import rospy

tf_buffer = Buffer()
tf_listener =
    TransformListener(
    tf_buffer)

transform = tf_buffer.
    lookup_transform(
    target_frame="odom",
    source_frame="base_link"
        ,
    time=rospy.Time(0)
)
```

TF Listener in ROS2 - code example (Python)

```python
class TfListener(Node):
  def __init__(self):
    super().__init__('tf_listener')
    self.parent_frame = 'world'
    self.child_frame = 'base_link'

    self._tf_buffer = Buffer()
    self._tf_listener = TransformListener(self._tf_buffer,
        self)

    self.timer = self.create_timer(0.1, self.timer_callback)
```

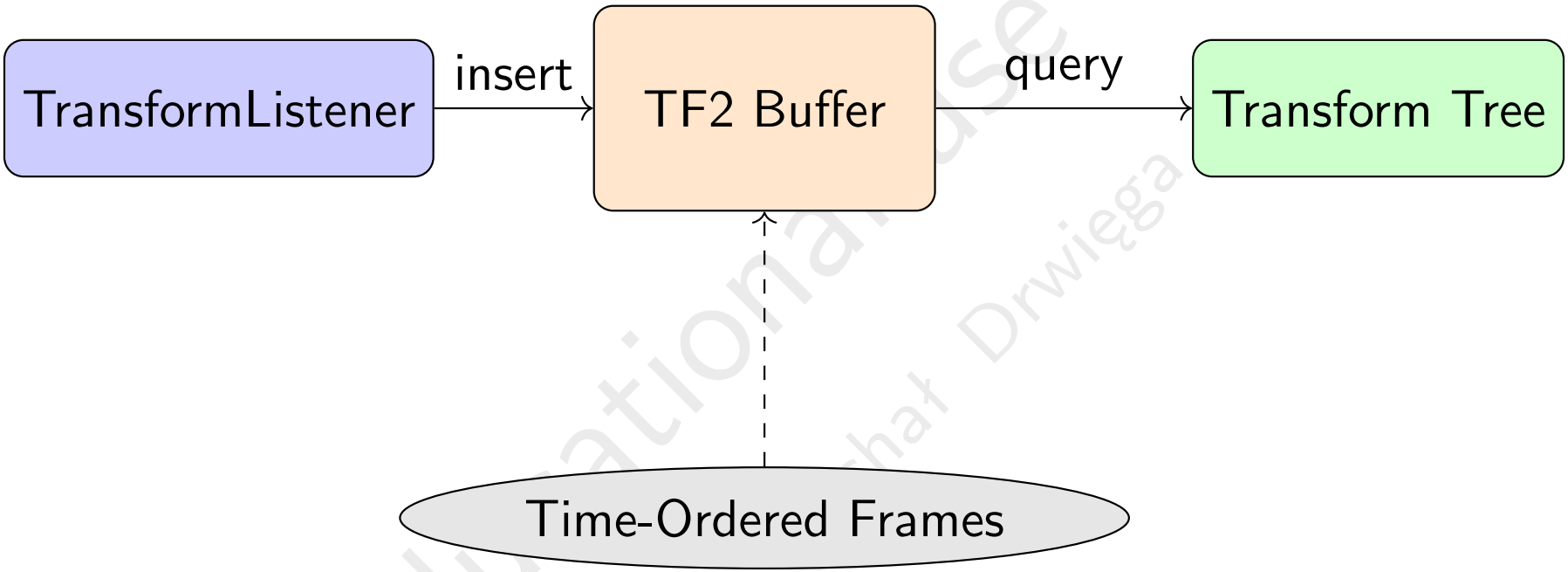
TF Listener in ROS2 - code example (Python)

```python
def timer_callback(self):
    try:
        transform_stamped =
            self._tf_buffer.lookup_transform(
                self.parent_frame,
                self.child_frame,
                rclpy.time.Time()
            )
        print(f'Received transform: {transform_stamped}')

    except LookupException as e:
        self.get_logger().error('failed to get transform {} \n'.
            format(repr(e)))
```

TF2 Buffering, Interpolation & Synchronization

The TF2 Buffer Architecture



- **Buffer**: Core data structure storing transforms over time
- Each transform stored with: `frame_id`, `child_frame_id`, `transform`, `timestamp`

How Buffering Works

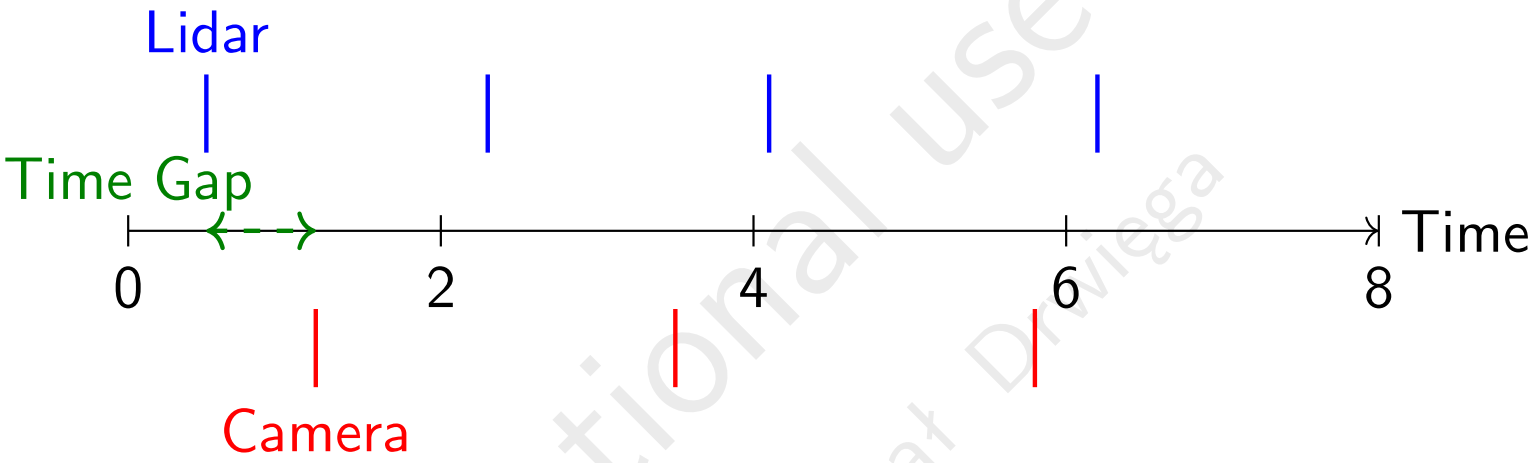
Storage

- Each frame pair has a circular buffer of transforms over time
- Newer transforms replace older ones based on time
- Default buffer length: 10 seconds

Buffer Lookup

```
buffer.lookupTransform("world", "robot", time_point);
```

The Time Synchronization Problem



Common Issues

1. **Message latency**: Sensor messages arrive at different times
2. **Clock skew**: Different computers have slightly different clocks
3. **Jitter**: Network/processing delays vary

TF2 Synchronization Strategies

1. Time Travel

- Look up transform at the exact timestamp of your data
- `lookupTransform(target, source, data_timestamp)`
- *"What was the transform when this sensor reading happened?"*

2. Latest Available

- Use the most recent transform (ros::Time(0))
- Risk: Temporal misalignment
- Only for very high-rate, low-latency systems

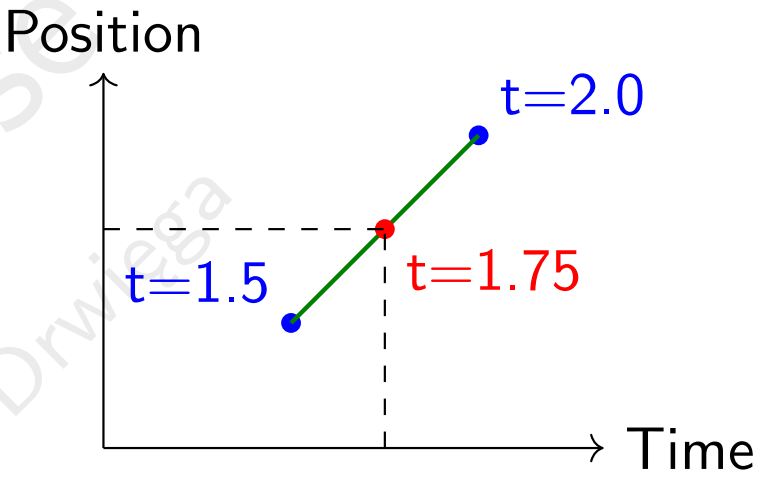
3. Interpolation

- Buffer automatically interpolates between stored transforms
- Requires at least 2 transforms around target time
- Linear interpolation for position, SLERP for orientation

TF2 Transform Interpolation

What is Interpolation?

When querying a transform at a timestamp **between** two stored transforms, TF2 automatically interpolates the pose.



Example Query

```
lookupTransform("world", "robot", t=1.75)
```

Interpolation Methods

- **Linear Interpolation**: Used for position (x, y, z)
- **Spherical Linear Interpolation (SLERP)**: Used for orientation (quaternions)
- **Constant Velocity Assumption**: Default behavior between stored transforms

TF2 Limitations

Covariance Matrix in Robotics

What is a Covariance Matrix?

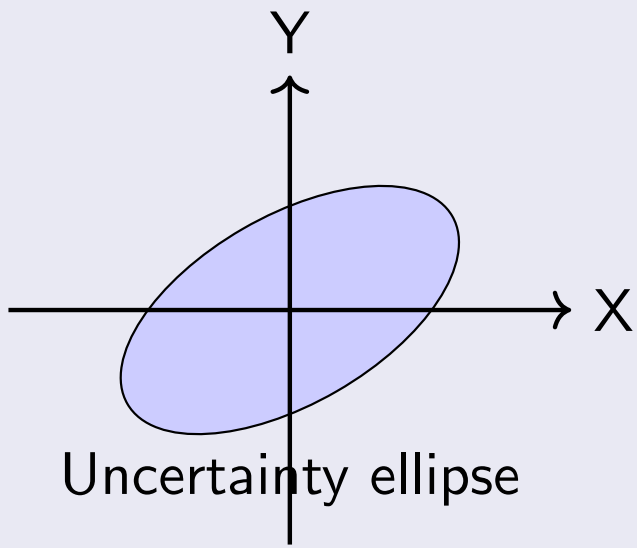
- Measures **uncertainty** in state estimates
- Diagonal = variance of each variable
- Off-diagonal = correlations between variables
- Size: $n \times n$ for n state variables

3D Position Covariance

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_z^2 \end{bmatrix}$$

Where σ_x^2 = variance in X, σ_{xy} = correlation between X and Y

Visual Interpretation



Larger ellipse = **more uncertainty**

TF2 Limitations - Transform Covariance

What is Transform Covariance?

Covariance represents **uncertainty** in a transform:

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{xz} & \cdots \\ \sigma_{yx} & \sigma_y^2 & \sigma_{yz} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

- 6x6 matrix for position (3) + orientation (3)
- Describes how confident we are in the transform
- Essential for sensor fusion (Kalman filters, EKF)

TF2's Problem

TF2 does NOT store or propagate covariance information!

TF2 Limitations - Why Covariance Matters

Real World vs TF2

- Odometry drifts $\rightarrow$ uncertainty grows over time
- Sensor noise $\rightarrow$ measurements have confidence
- TF2 treats all transforms as **perfectly certain**

TF2 Limitations - Workarounds for Covariance

Approach	**Description**
`geometry_msgs::PoseWithCovariance`	Store covariance separately, not in TF2
`nav_msgs::Odometry`	Store covariance separately, not in TF2
Custom TF2 buffer	Extend TF2 to store covariance

Major TF2 Limitations

Cycles / Loops

- Tree structure only
- Child cannot be parent's ancestor

No Covariance

- No uncertainty representation
- TF2 = **What** the transform is (deterministic)
- Covariance = **How certain** we are (probabilistic)

Time Limitations

- Can't query future transforms (no extrapolation)
- Limited history (cache size)
- `ExtrapolationException` common

Coordinate Frames Debugging

Debugging TF2 Synchronization Issues

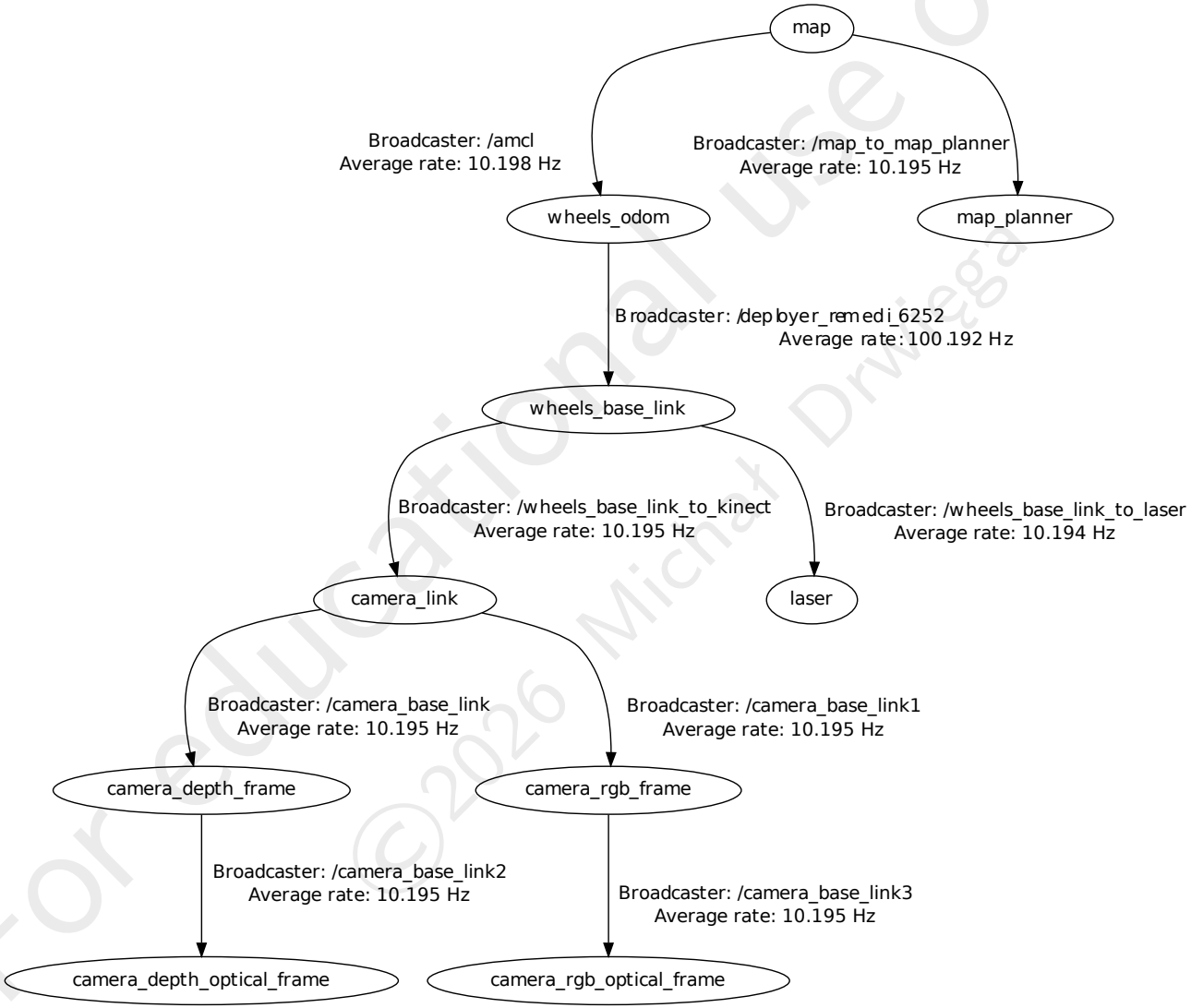
Common Issues

- *"Lookup would require extrapolation"* → Need more transforms in buffer
- *"Frame not found"* → Transform not published yet or frame name typo
- Large pose jumps → Transform interpolation failure or wrong timestamp

Debugging Tools

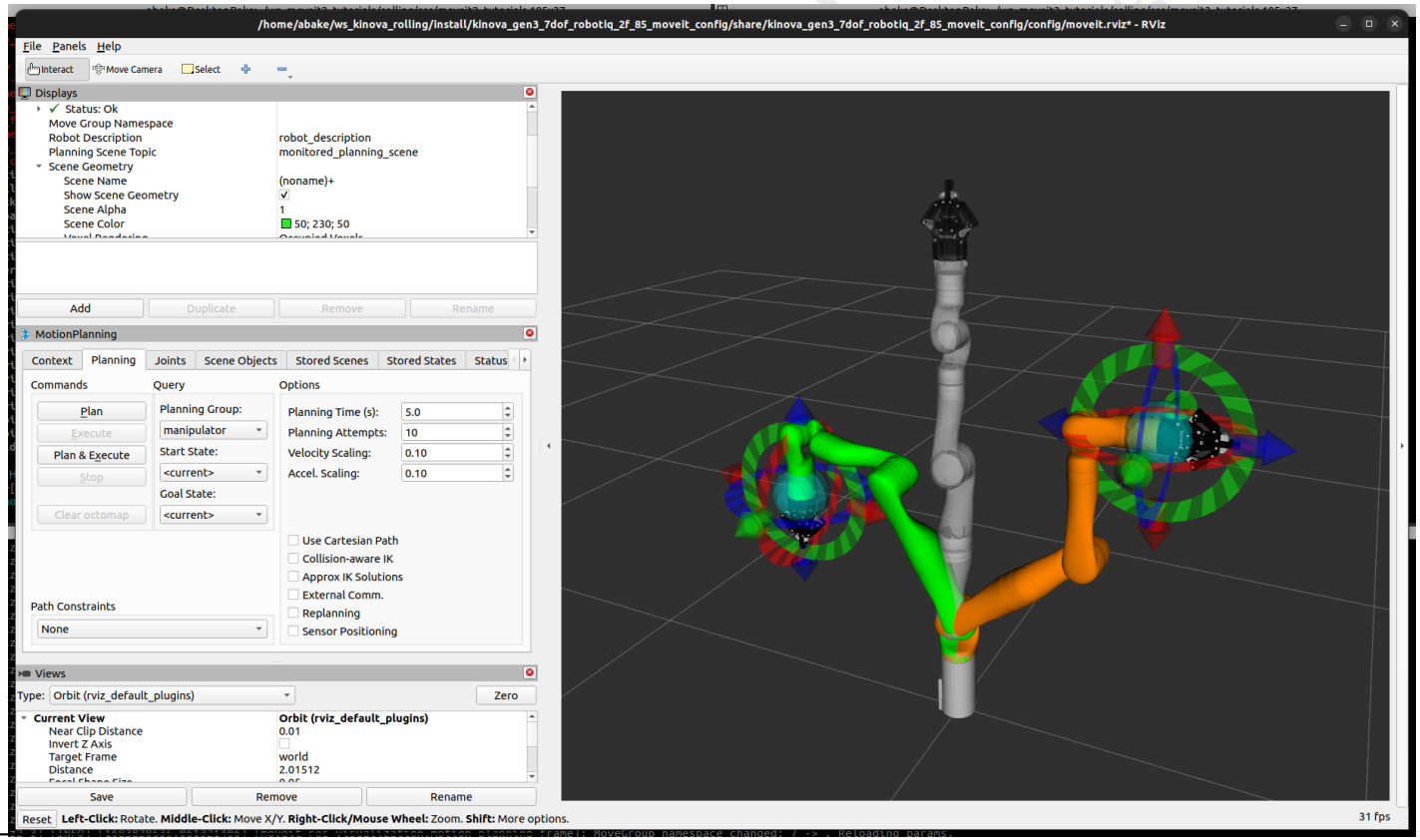
- `view_frames`: Generates PDF of transform tree
- `tf2_echo frame1 frame2`: Live transform streaming
- `ros2 run tf2_ros tf2_monitor`: Monitor transform latency

view_frames - Example Output



Rviz: Robot Visualization Tool

- `Rviz` is a 3D visualization tool for ROS, used to display robot states, sensor data, and environmental models.
- Allows real-time visualization of data from ROS topics such as laser scans, point clouds, transformations, and robot models.



[6]

[6] https://moveit.picknik.ai/main/_images/rviz_plugin_head.png

Summary

The end

Thank you for your attention!