

# ***Robot Programming Transforms and Sensors in ROS***

Giorgio Grisetti

# Outline

- Robot Devices
  - Overview of Typical sensors and Actuators
  - Operating Devices in ROS
- Describing your Robot
  - Transform Tree
  - Transform Publisher
- Transforms and Time
  - Interpolating Transforms
  - TF library
  - Publishing and reading transforms
- Hands on a robot
- Displaying sensor data (rviz)
- Recording real data with a robot

# How to access a Device in ROS?

- Each device is a node
- The input topics are the commands that the device can output
- The output topics are the feedback given by the device.
- In `sensor_msgs/` many messages for the common sensors are defined.
- Use `rosmmsg show <message_name>` to see the format of a message.
- To start a device it is sufficient to start the corresponding node and to give it the necessary configuration parameters. These include
  - Specific devices parameters (e.g. which serial port/usb device , the resolution of an image, and so on..)
  - The **name** of the **reference frame** in the sensor

# Mobile Base in ROS

- Typical mobile bases are mapped as ROS nodes that
  - Publish messages of type
    - `nav_msgs/Odometry`  
These messages specify the odometry
  - Subscribes to messages of type
    - `geometry_msgs/Twist`  
That specify the desired translational and rotational velocities

All this looks very similar to TurtleSim, but the transforms and the velocities are computed in 3D

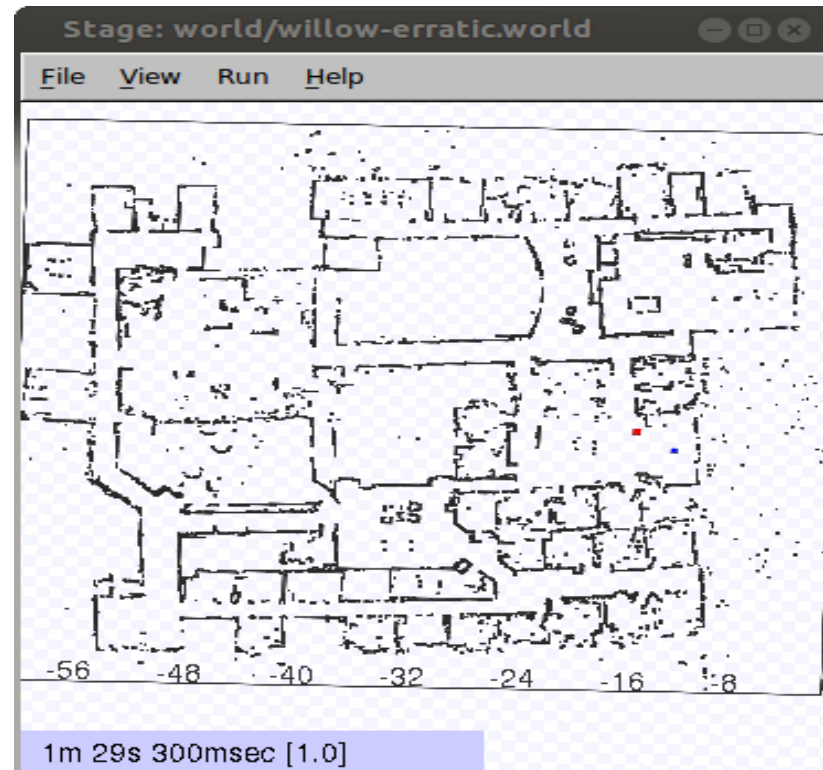
# Stage

## Unsupported in ROS2

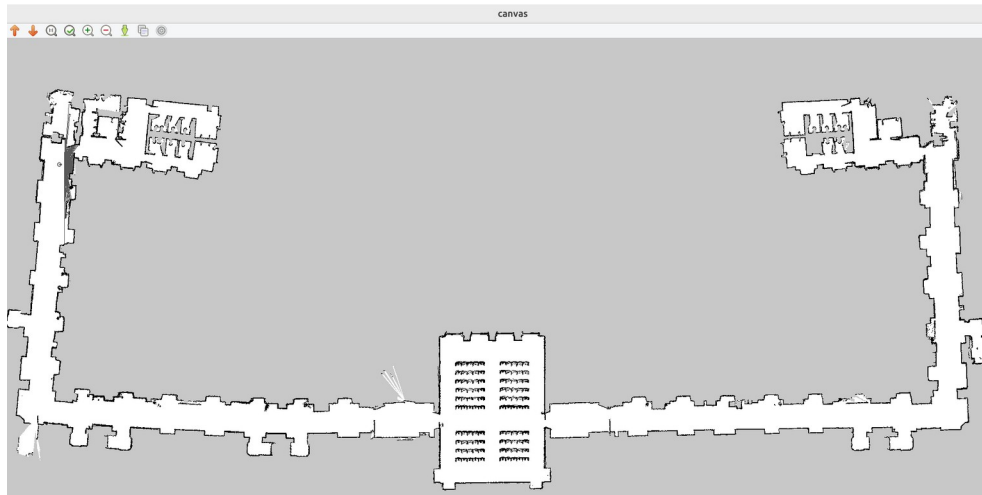
- ~~To launch stage~~

- ~~\$> roscore~~
- ~~\$> roscd stage~~
- ~~\$> rosrun stage stageros~~  
~~words/willow\_erratic word~~

- We have our own in rp\_nav



# RP Simulator



```
$> ros2 run rp_simulator simulator --ros-args -p config_file:=assets/diag_single_robot.yaml
```

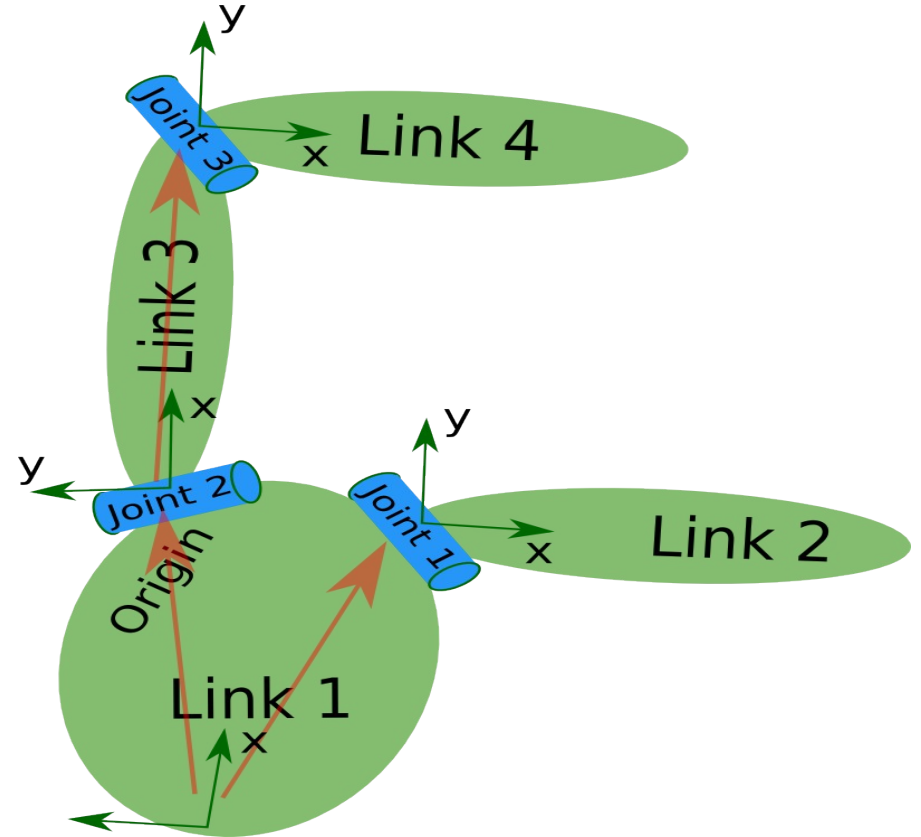
- With ros2 topic you will see that there is a `/cmd_vel` argument. Publishing on this topic allows you to set the the robot speed
- The robot sends you the odometry feedback by the `/odom` topic, and potentially some additional state packet.

# Specifying the Arrangement of Devices

- All these devices are mounted on a robot in an articulated way.
- Some devices are mounted on other devices that can move.
- In order to use all the sensors/actuators together we need to describe this configuration.
  - For each “device” specify one or more frames of interest
  - Describe how these frames are located w.r.t each other

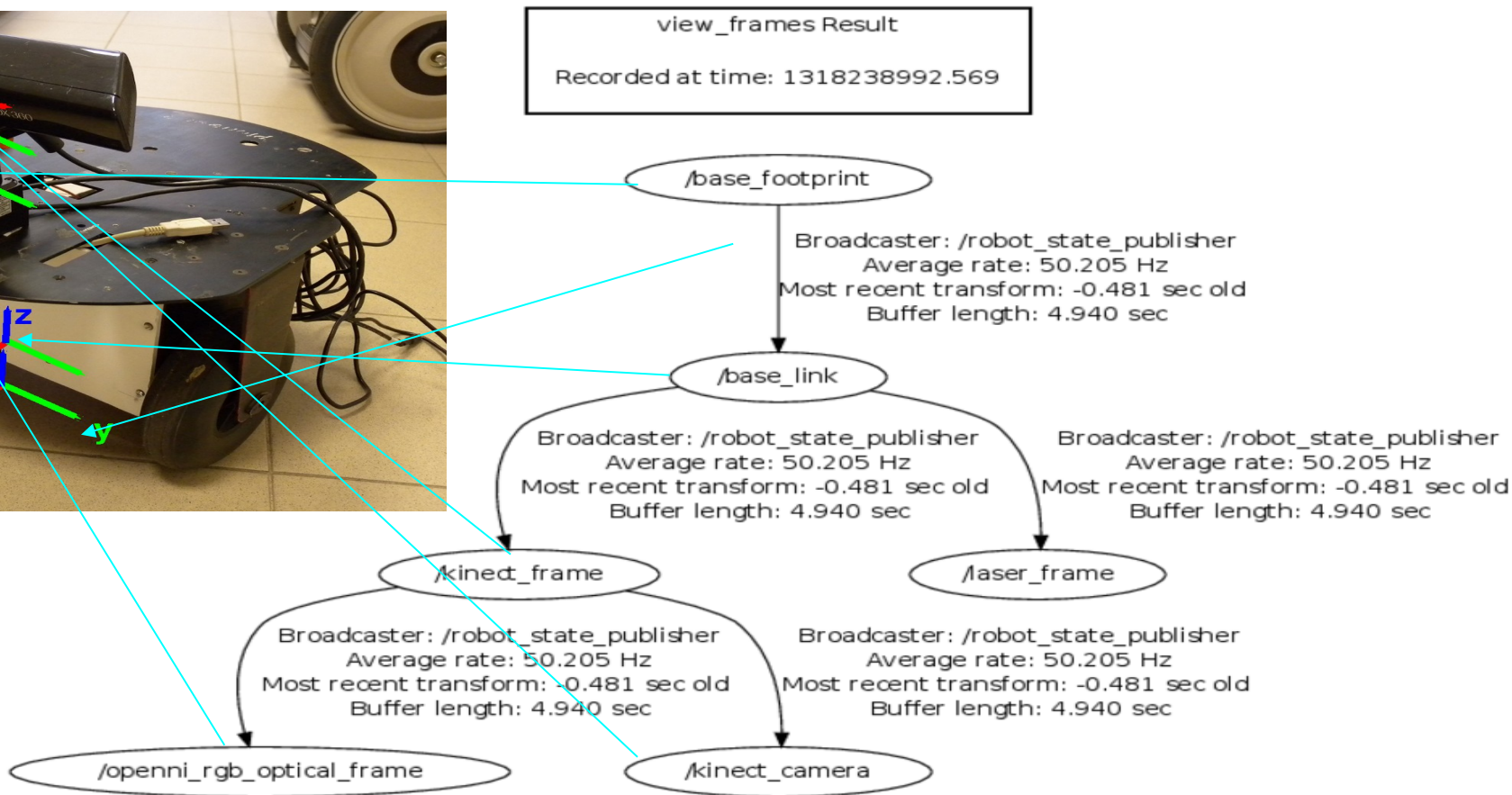
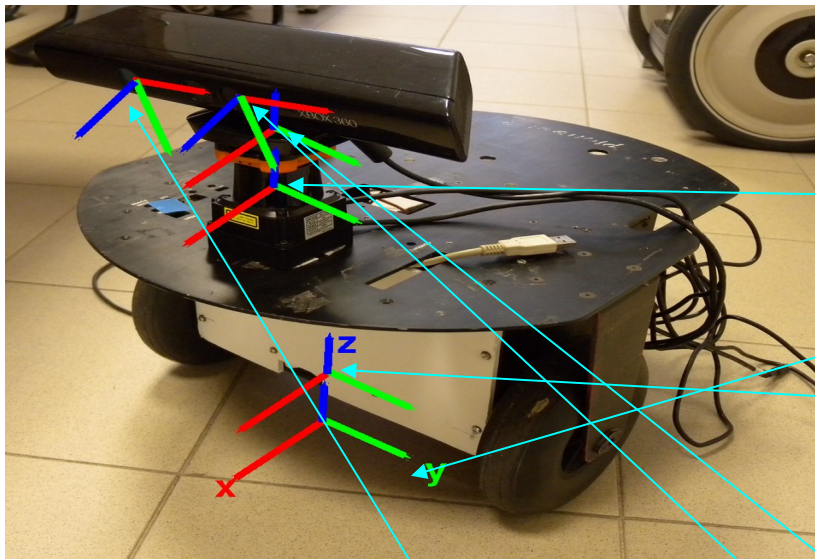
# Defining the Structure

- You have to specify the kinematics of the robot.
- Each “Link” is a reference frame of a sensor
- Each “joint” defines the transformation that maps the child link in the parent link.
- ROS does not handle closed kinematic chains, thus only a “tree” structure is allowed
- The root of the tree is usually some convenient point on the mobile base (or on its footprint)





# Practical Example



# Transform Publishers

- **A transform can be published by any ros node.**
- The local configuration of a robot (e.g. the position of the sensors/actuators w.r.t a frame on the robot platform) is usually published by a convenience node: the **robot\_state\_publisher**.
- **The robot state publisher:**
  - takes a description of the robot (the kinematics), that specifies for each frame:
    - the parent frame
    - the type of joint
  - Listens the state of the joints
  - Computes the transforms for all the frames.
- If the robot has no movable devices (except the base) one can use the **static\_transform\_publisher**.
- The **static transform publisher** is a node that can be invoked like that  
`$> ros2 run tf2_ros static_transform_publisher fromFrame toFrame x y z roll pitch yaw`  
  
e.g. `$> ros2 run tf2_ros static_transform_publisher baseFrame cameraFrame 0 0 0.3 0 0 3.14`  
will start a node that publishes a transform between the baseFrame and the camera, telling that the camera is mounted at 30 cm above the mobile base and is looking backwards (yaw =  $M\_PI$ ).(\*)

(\*) check the online documentation for an updated command line

# Visualizing The Data

- Once all sensors are started and the robot description is correctly done, we can visualize the data.
- To this end, we will use the RVIZ ros tool.
- I will give a practical example, you can look at the ros wiki, for rviz.

# Interpolation

- A robot is a complex system consisting in a potentially large set of devices
- These devices typically run in an asynchronous fashion. Each of them outputs the data when available.
- In many tasks, we are interested in knowing the position of the robot when a specific information is gathered by the sensor
- At this time, however there might not be a valid transformation, thus we have to determine the sensor position by interpolation.

## Interpolation (II)

- To interpolate the position of a joint at time  $t$  we need to know
  - The position at time  $t_m < t$
  - The position at time  $t_M > t$
  - The velocities and
  - The kinematic constraints
- All these informations are available in the tf messages
- ROS provides a **tf** client library to interpolate and publish transforms.

# TF Main Facts

- To perform interpolation it installs a set of transform buffers, one for each frame.
- It allows to send/receive transform messages
- One can obtain the interpolated position between any pair of frames.
- The ***tf*** package contains several useful programs to debug the system
  - **view\_frames**: generates a pdf file by listening all transforms  
`$> ros2 run tf2_tools view_frames`
  - **static\_transform\_publisher**: is a node that streams a specific transform given as argument.

# Using TF

- Listener:
  - Creates a buffer
  - Creates a listener hooked to the buffer
  - When needed uses interpolation to query between two instants in the buffer
  - Potential exceptions
    - No path between two frames
    - Extrapolation

<https://docs.ros.org/en/jazzy/Tutorials/Intermediate/Tf2/Writing-A-Tf2-Listener-Cpp.html>

- Broadcaster:
  - Just publishes a tf message

<https://docs.ros.org/en/jazzy/Tutorials/Intermediate/Tf2/Writing-A-Tf2-Broadcaster-Cpp.html>

# Recording a Dataset

- With **ros2 bag** you can record in a bag all the messages about a specific topic
- We will now record a bag of a moving robot
- This bag will be made available to you



# Launch Files

- A system running on ROS may consist in a large number of nodes, each with its parameters
- To start these nodes, one might use the .launch files (See roslaunch).
- Launch files are xml scripts used to start and configure a large number of nodes
- They need to reside in the /launch directory of a package
- They can be started with  
**roslaunch <package\_name>  
<launch\_file>**

```
<launch>

  <node name="map_server" pkg="map_server" type="map_server" args="$
(find dis_navigation)/maps/dis-B1-2011-09-27.yaml"/>

    <group ns="erratic1">
      <param name="tf_prefix" value="erratic1" />

      <include file="$(find
dis_robots)/launch/erratic_hokuyo.launch" />
      <param name="hokuyo/frame_id" type="str"
value="/erratic1/laser_frame"/>

      <include file="$(find
dis_navigation)/config/localization/glocalizer_node.xml" />
      <include file="$(find
dis_navigation)/config/navigation/move_base.xml" />
      <node pkg="tf" type="static_transform_publisher"
name="link_broadcaster_0" args="0 0 0 0 0 /map /erratic1/map
100" />
    </group>

    <group ns="erratic1">
      <param name="glocalizer/initial_pose_x" value="0" />
      <param name="glocalizer/initial_pose_y" value="1.8" />
      <param name="glocalizer/initial_pose_a" value="0" />
    </group>

</launch>
```

## Exercise 1

- Write a ros node that writes in a text format the 2D location of the laser (x,y,theta) when laser messages arrive, and the timestamp
- FORMAT:
  - One line per message
  - **LASER <timestamp.sec>.<timestamp.usec> <laser pose w.r.t. odom frame (x,y,theta)>**

# Exercise Hints

- 1. create a transform listener
- 2. create a laser callback
  - In the laser callback query if you can obtain a transform from /odom to the laser frame specified in the laser message header (use canTransform)
  - If this is true, retrieve the transform (use lookup\_transform)
  - Extract x,y,and yaw from the pair <translation, quaternion> stored in the transform filled by lookup\_transform.
  - Append it to a file.

## Exercise 2

- Extend the previous exercise to generate a file containing the laser endpoints in the /odom reference frame, at robot pose, and dumping them on a file.
- One point per line <x> <y>