Transforms and Sensors in ROS

Giorgio Grisetti

Outline

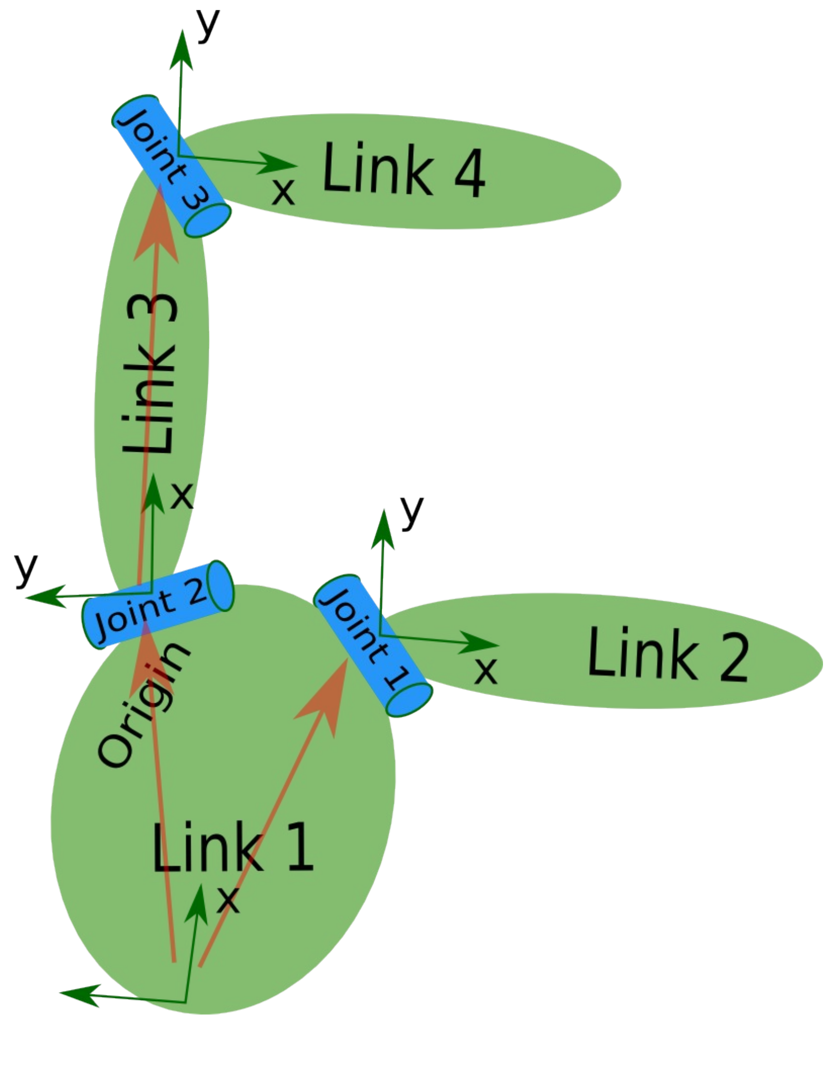
- Robot Devices
  - Overview of Typical sensors and Actuators
  - Operating Devices in ROS
- Describing your Robot
  - Transform Tree
  - Transform Publisher
- Transforms and Time
  - Interpolating Transforms
  - TF library
  - Publishing and reading transforms
- Hands on a robot
- Displaying sensor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

Specifying the Arrangement of Devices

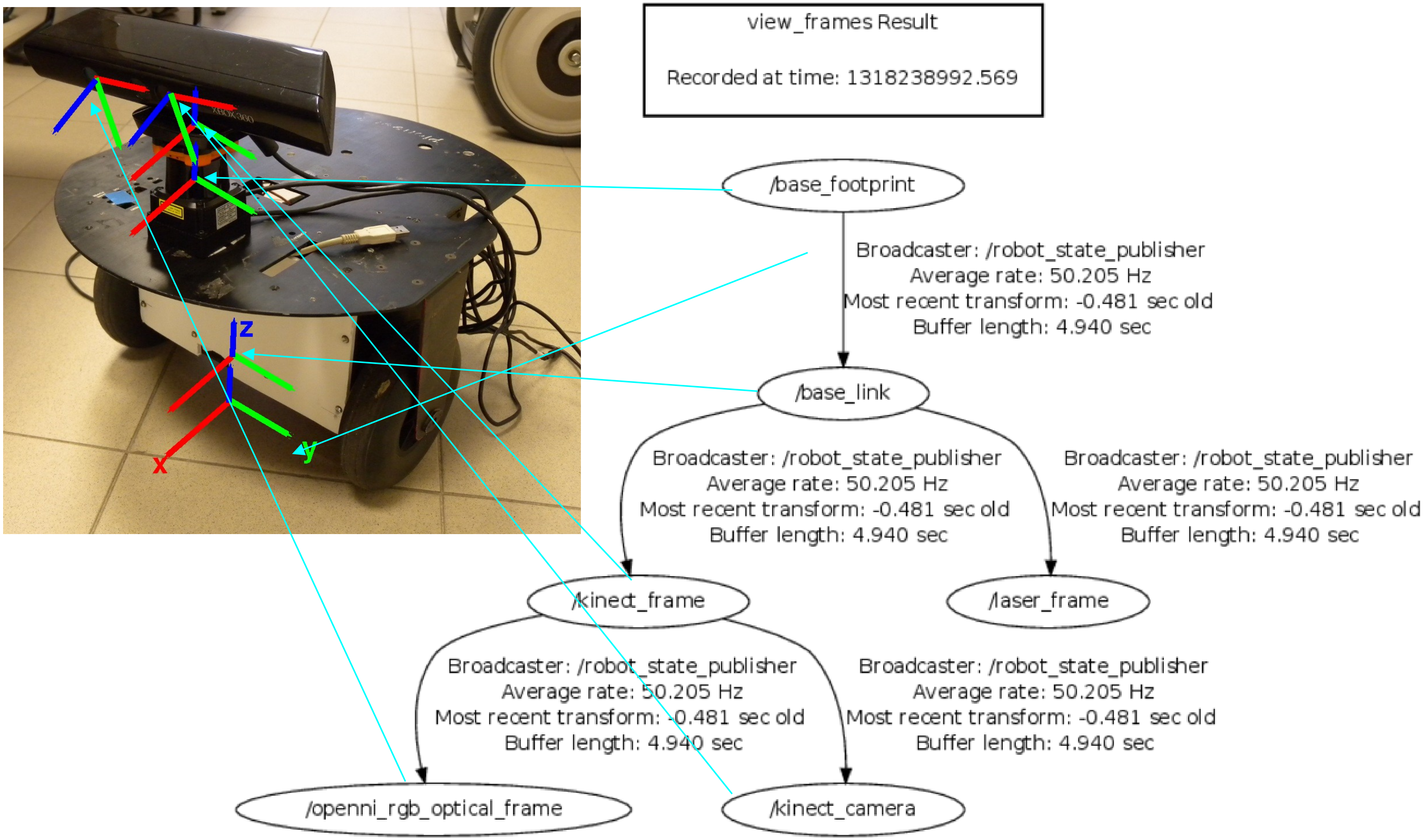
- All these devices are mounted on a robot in an articulated way.
- Some devices are mounted on other devices that can move.
- In order to use all the sensors/actuators together we need to describe this configuration.
  - For each “device” specify one or more frames of interest
  - Describe how these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

Defining the Structure

- You have to specify the kinematics of the robot, according to what you learned in the Robotics course.
- 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

  $> **`rosrun tf static_transform_publisher fromFrame toFrame x y z roll pitch yaw hz`**

  e.g.

  $> **`rosrun tf static_transform_publisher baseFrame cameraFrame 0 0 0.3 0 0 3.14 10`**

  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

    ```
    $> rosrun tf view_frames
    ```
  - **static_transform_publisher**: is a node that streams a specific transform given as argument.

Using TF

- TF has an own Listener that sets up the buffers

```
TransformListener(
      ros::Duration max_cache_time=ros::Duration(DEFAULT_CACHE_TIME),
      bool spin_thread=true)
```

- To see if you can compute the position of a frame w.r.t. another one you should first check that the buffers are consistend with the query

```
bool tf::TransformListener::canTransform (
      const std::string &target_frame,
      const std::string &source_frame,
      const ros::Time &time,
      std::string *error_msg=NULL) const
```

- To compute a transform between to frames use the following function

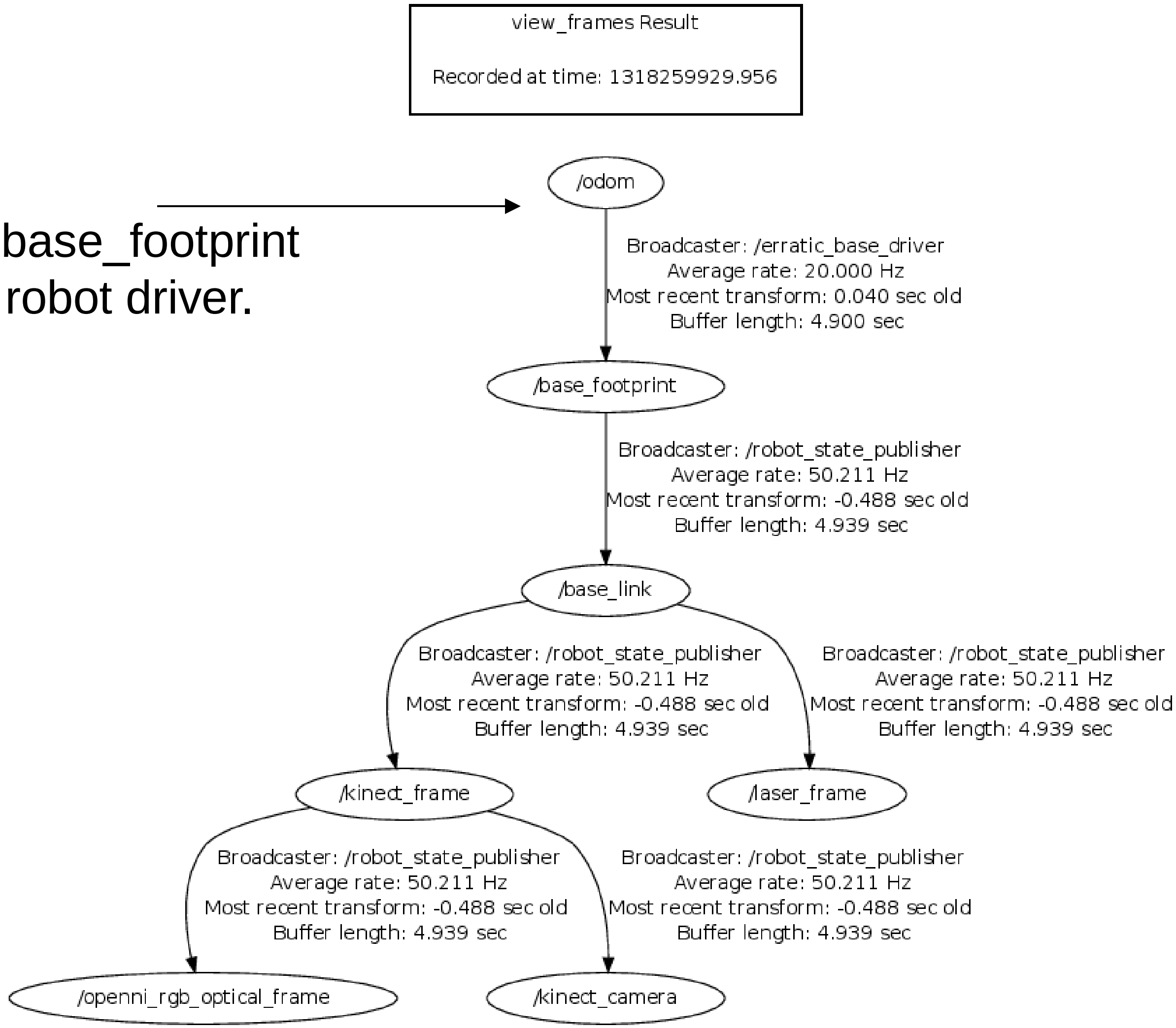
```
void tf::TransformListener::lookupTransform (
      const std::string &target_frame,
      const ros::Time &target_time,
      const std::string &source_frame,
      const ros::Time &source_time,
      const std::string &fixed_frame,
      StampedTransform &transform) const
```

Recording a Dataset

- With **rosbag** 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

Transform Tree in the Bag

New base frame.
Transform from odom to base_footprint
Is published by the base robot driver.



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 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>
```

Homework (2)

- 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)>`**