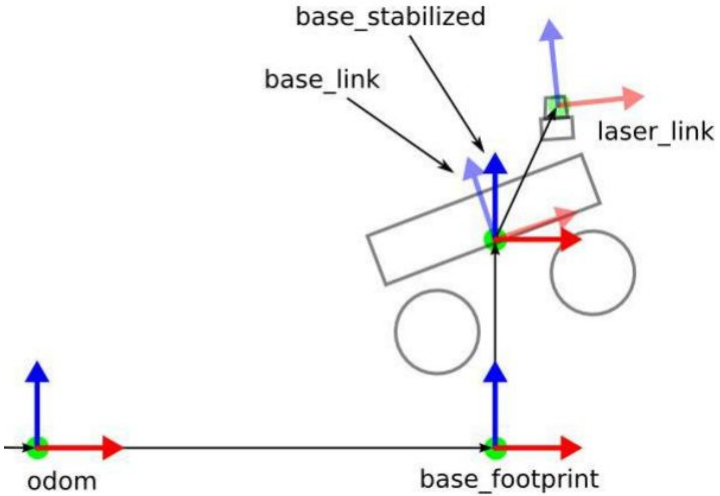
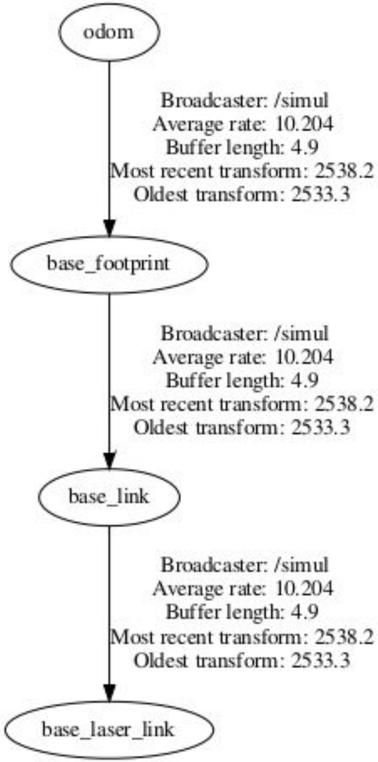
Introduction to TF

Lab 2 - Autonomous Robots

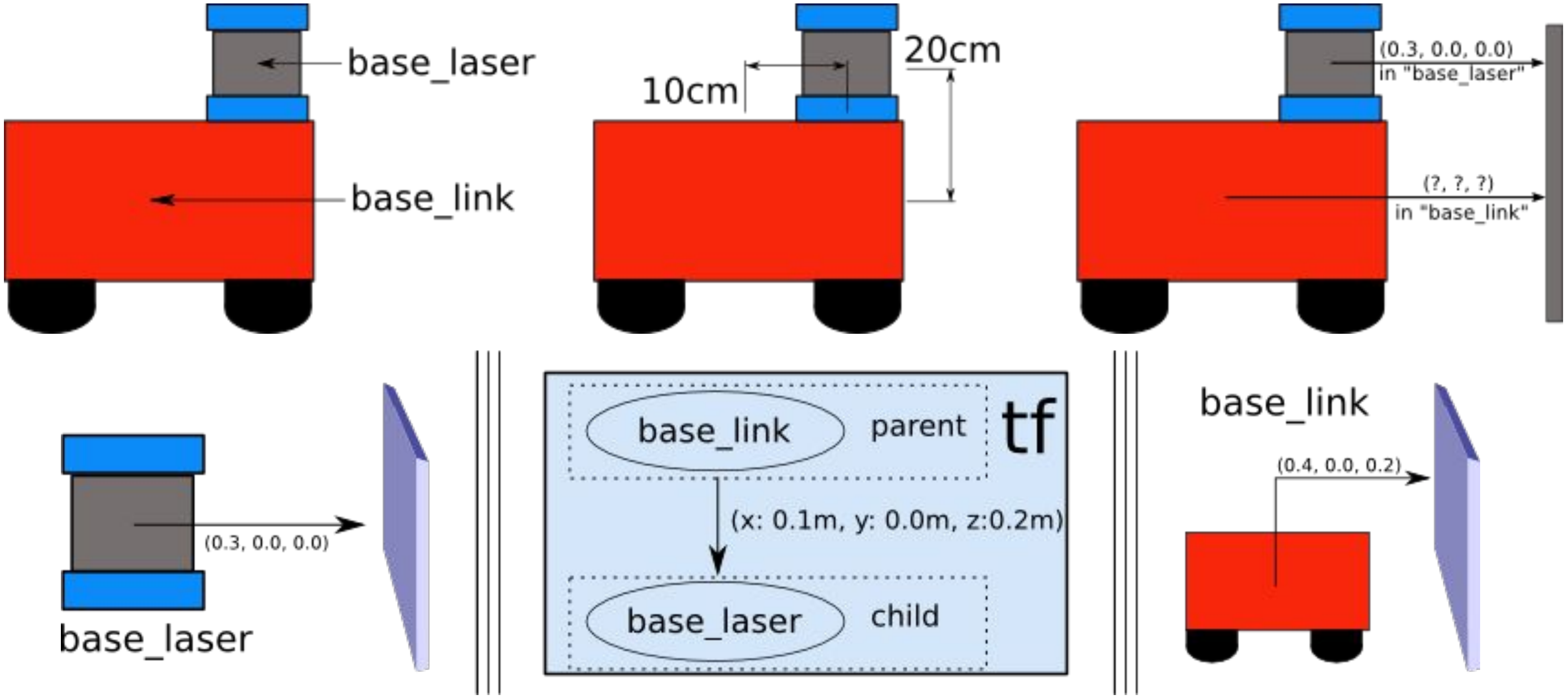
Transformations in ROS (TF)

- The tf package is a tool to **track** multiple 3D coordinate frames over time
- It maintains the **relationship between coordinate frames** in a tree structure buffered in time.
- It allows the user to **transform points and vectors** between the coordinate frames at desired time





Transform Example (Laser)



Source: http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF

Terminal commands

- **$ rosrun tf2_tools view_frames.py**
- **$ evince frames.pdf**

- **$ roslaunch arob_lab2 start.launch**
- **$ rosrun tf tf_monitor**
- **$ rosrun tf tf_echo source_frame target_frame**
- **$ rosrun tf tf_echo odom base_link**

```
- Translation: [-6.000, -6.000, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.000, 1.000]
            in RPY (radian) [0.000, -0.000, 0.000]
            in RPY (degree) [0.000, -0.000, 0.000]
```

Exercise 1

For this exercise, you have to download or clone in your ~/catkin_ws/src folder from the following URL:

https://github.com/luisriazuelo/pXX_arob_lab2.git

- **Change pXX prefix**, where pXX corresponds to the assigned pair number (e.g. p00, p01 ......) for this course.
- Execute ***start.launch*** file
- Get the current **transform tree** and print **information** about the **transformation** between two frames.
- **Create a node** called robot_location **for obtaining** the **information** about **transformations.**

Defining the world and the objects in stage

- **Definition of a model is done using the following format:**

```
define model_name model (
  # parameters
 )
```

- **Stage simulation window**

```
window (
  size [ 635.000 666.000 ] # size of the window in pixels
  scale 36.995   # pixels per meter
  center [ -0.040  -0.274 ] # location of the center of the window in world coordinates
  rotate [ 0.000  0.000 ] # angle of rotation relative to straight up (degrees)
  show_data 1  # 1=on 0=off : show the laser view of the robot
)
```

Exercise 2

For this exercise, you have to update the world file simple.world from pXX_arob_lab2 package:

- **Add a new model** of size 0,15 x 0,15 x 0,15 and yellow color **defining a target point**.
- **Add such objects** at the following poses: [2 -5 0 0], [4 -1 0 0], [-2 0 0 0] and [-1 5 0 0].
- **Move** the **robot** with the **keyboard.**
- **Check** that the robot is **not colliding** with this **new model** introduced.
- **Check** that the robot is **colliding** with the dynamic **obstacle in the original** worldfile.

Typical messages in ROS

- **`geometry_msgs/PoseStamped.msg`** can be used to define the state of the robot. In this case, the variable `Goal` has the following structure:
  - `Goal.pose.point.x; Goal.pose.point.y; Goal.pose.point.z`
  - `Goal.pose.orientation.x; Goal.pose.orientation.y; Goal.pose.orientation.z; Goal.pose.orientation.w`

- **`geometry_msgs/Twist.msg`** can be used to send linear and angular velocities to the robot. In this case, the variable input has the following structure:
  - `input.linear.x`- is the linear velocity;
  - `input.angular.z`- is the angular velocity.

Typical messages in ROS

- **nav_msgs/Odometry.msg** can be used to get the actual position of the robot in the environment. The msg constant has the following structure:
  - msg.pose.pose.position.x,
  - msg.pose.pose.position.y are the x and y position of the robot;

- **In order to publish a command velocity message:**

  ```
  $ rostopic pub /cmd_vel geometry_msgs/Twist -r 100 '[10, 0, 0]' '[0, 0, 4]'
  ```

Exercise 3

Complete the node lowcontrol:

- **Subscribe** to the **robot's position** */base_pose_ground_truth*
- **Subscribe** to a topic called */goal* to receive the **target point**.
- **Publish** the required **velocities** to control the robot to the goal */cmd_vel* topic.
- Test the node by given sequentially the poses of the target objects defined in the worldfile in Exercise 1.

For publishing manually a goal, you can use the following command,

```
$ rostopic pub /goal geometry_msgs/PoseStamped '{header: {stamp: now, frame_id: "odom"},
pose: {position: {x: 2, y: -5, z: 0.0}, orientation: {w: 0}}}'
```

Exercise 4

Testing the code implemented in the previous exercise on a real platform:

- **Send** the package "pXX_arob_lab2" **to one of the robots** available for real-world testing.

  ```
  $ scp -r pXX_arob_lab2 arob@ip_robot:arob_ws/src/
  ```

- To launch the code on the robot, you should use the launch file real_robot.launch.

ip address:

- turtlebot1: 10.1.31.215
- turtlebot2: 10.1.31.214

robot user account:

- user: arob
- password: unizar

Exercise 5

Complete the followTargets node:

- **Read** from the text file **targets.txt** the list of targets
- **Publish** them **one by one** to the topic */goal* created in Exercise 3.
- The new goal is **published** when the **robot** is "**sufficiently close**" to the previous target.

Laboratory 2 evaluation

- **Submit** the **code** for all exercises. Send the complete *pxx_arob_lab2* package before the beginning of the next session.
- **Run** Exercise 4 on the **real platform**.
- **Multiple-choice test** through Moodle at the **beginning** of the **next session**. Test will be conducted **individually**, **without** any **extra material.**