Robot Programming with Lisp

7. Coordinate Transformations, TF, ActionLib

Gayane Kazhoyan

Institute for Artificial Intelligence
Universität Bremen

1st December, 2015

Outline

Theory
    Coordinate Transformations
    TF
    ActionLib

Organizational

Outline

Theory
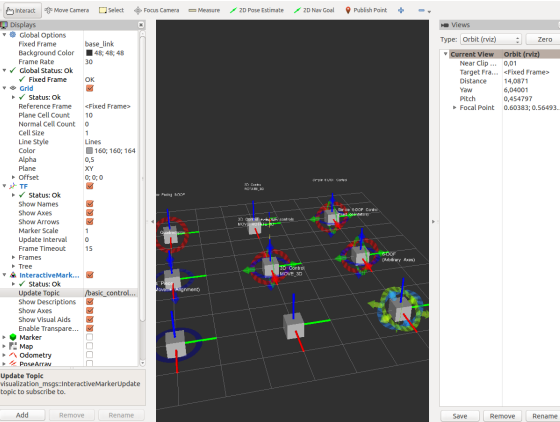   Coordinate Transformations
    TF
    ActionLib

Organizational

Poses in 3D Space

```
$ roscore

$ rosrun interactive_marker_tutorials basic_controls

$ rosrun rviz rviz
```

Representing Poses

Point in 3D: $\{x, y, z\}$

3D-Vector

```
CL-TRANSFORMS> (make-3d-vector 1 2 3)
#<3D-VECTOR (1.0d0 2.0d0 3.0d0)>
CL-TRANSFORMS> (describe *)
#<3D-VECTOR (1.0d0 2.0d0 3.0d0)>
  [standard-object]
Slots with :INSTANCE allocation:
  X  = 1.0d0
  Y  = 2.0d0
  Z  = 3.0d0
CL-TRANSFORMS> (y **)
2.0d0
```

Object in 3D: $\{position, orientation\}$
  Position: $\{x, y, z\}$
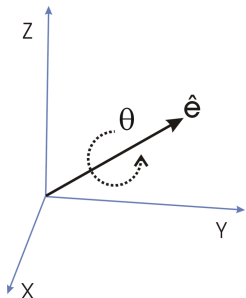 Orientation: axis-angle / rotation matrix / quaternions / ...

Representing Rotations

Axis-Angle representation:

$$< axis, angle > = \left\langle \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}, \theta \right\rangle$$

Axis-Angle → Quaternion:

$$Q = \begin{pmatrix} q_x \\ q_y \\ q_z \\ q_w \end{pmatrix} = \begin{pmatrix} a_x \sin(\theta/2) \\ a_y \sin(\theta/2) \\ a_z \sin(\theta/2) \\ \cos(\theta/2) \end{pmatrix}$$

3D-Vector

```
CL-TRANSFORMS> (make-quaternion 0 0 0 1)
CL-TRANSFORMS> (describe *)
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>
  [standard-object]
Slots with :INSTANCE allocation:
  X = 0.0d0
  Y = 0.0d0
  Z = 0.0d0
  W = 1.0d0
CL-TRANSFORMS> (axis-angle->quaternion
                (make-3d-vector 0 0 1) pi)
```

Poses in Lisp

cl-transforms:pose

```
CL-TRANSFORMS> (setf p (make-pose
                         (make-3d-vector 1 2 0)
                         (make-quaternion 0 0 0 1)))
#<POSE
    #<3D-VECTOR (1.0d0 2.0d0 0.0d0)>
    #<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>>
CL-TRANSFORMS> (origin p)
#<3D-VECTOR (1.0d0 2.0d0 0.0d0)>
CL-TRANSFORMS> (orientation p)
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>
```
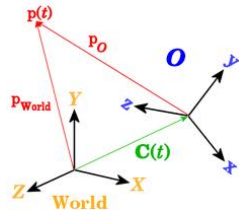
Coordinate Systems

Transformations

```
CL-TRANSFORMS> (setf W (make-identity-pose))
#<POSE
   #<3D-VECTOR (0.0d0 0.0d0 0.0d0)>
   #<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>>
CL-TRANSFORMS> (setf O (make-pose
                        (make-3d-vector 2 0 0)
                        (make-quaternion 0 0 0 1)))
#<POSE
   #<3D-VECTOR (2.0d0 0.0d0 0.0d0)>
   #<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>>
CL-TRANSFORMS> (transform
                  (transform-inv (pose->transform O))
                  p)
#<POSE
   #<3D-VECTOR (-1.0d0 2.0d0 0.0d0)>
   #<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>>
```
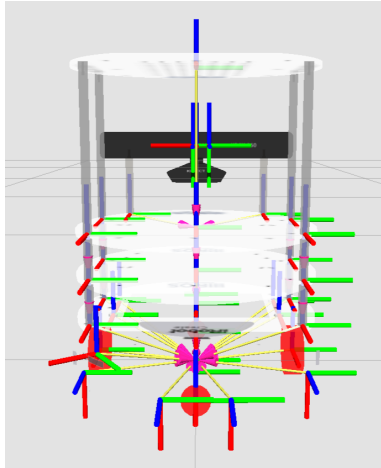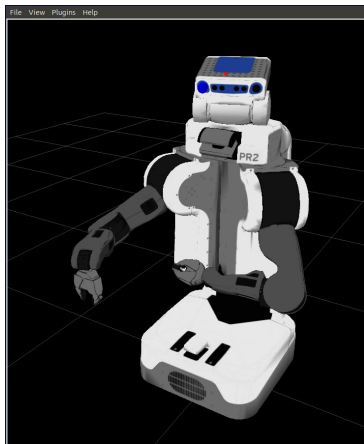
TurtleBot Coordinate Frames

Theory

Image courtesy: Yujin Robot

Organizational

Gayane Kazhoyan

Robot Programming with Lisp

1st December, 2015

9

Outline

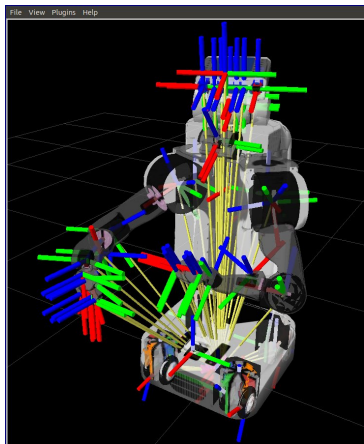Theory
  Coordinate Transformations
  TF
  ActionLib

Organizational

Motivation

- Robots consist of many *links*
- Every link describes its own *coordinate system*
- Sensor measurements are local to the corresponding link
- Links change their position over time (including the robot base)

Motivation

- Robots consist of many *links*
- Every link describes its own *coordinate system*
- Sensor measurements are local to the corresponding link
- Links change their position over time (including the robot base)

Implementation

- Transforms are produced by different nodes:
  - Localization in map (AMCL, gmapping)
  - Odometry (base controller)
  - Joint positions (robot controllers and robot_state_publisher)
- Many publishers, many consumers
- Distributed system, redundancy issues, ...
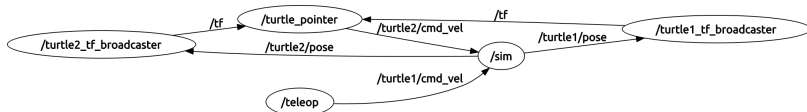
$$\downarrow$$

- **TF**: a coordinate frame tracking system
  - Publishing transforms to tf listeners
  - Looking up and calculating transforms by asking tf listeners
- Transformation data is cached over time
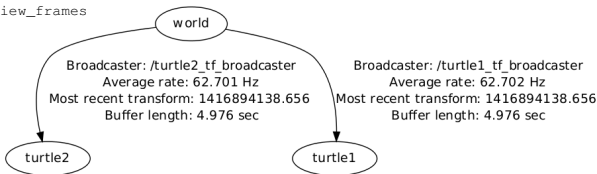- All the transforms together build a TF tree

TurtleSim TF

- Launch the turtlesim TF demo:

```
$ roslaunch turtle_tf turtle_tf_demo.launch
```


- Generate a TF tree graph:

```
$ rosrun tf view_frames
```
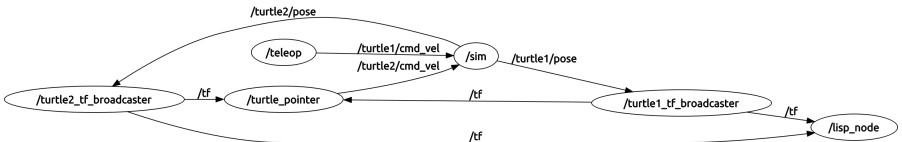

- Listen to transforms:

```
$ rosrun tf tf_echo turtle1 turtle2
```
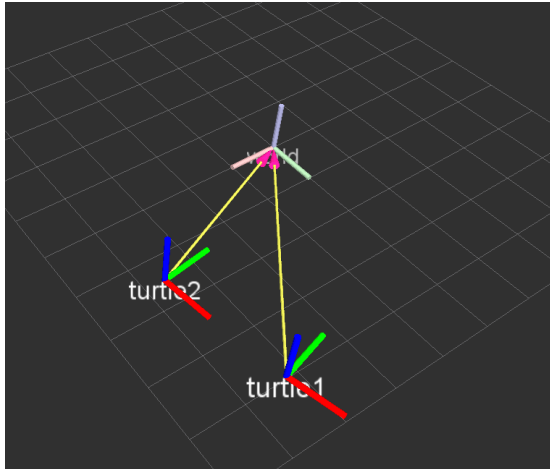
Lisp TF

cl_tf

```
TF> (roslisp:start-ros-node "lisp_node")
TF> (defparameter *transform-listener*
        (make-instance 'transform-listener))
TF> (lookup-transform *transform-listener* "turtle1" "turtle2")
#<STAMPED-TRANSFORM
   FRAME-ID: "turtle1", CHILD-FRAME-ID: "turtle2", STAMP: 1.4169d9
   #<3D-VECTOR (0.0d0 0.0d0 0.0d0)>
   #<QUATERNION (0.0d0 0.0d0 -0.5401331068059835d0 0.8415796022552d0)>>
```

$ rosrun rviz rviz

Outline

Theory
  ~~Coordinate Transformations~~
  ~~TF~~
  ActionLib

~~Organizational~~

ROS Actions

Interface to define and execute goals:
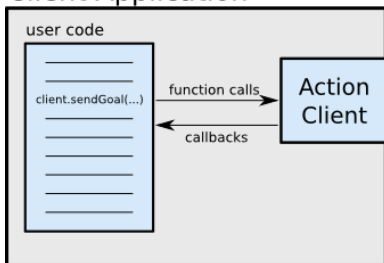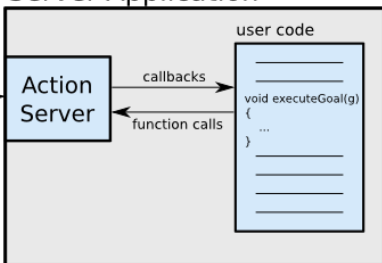
Illustration source: ROS actionlib wiki

Action Protocol

Relies on ROS topics to transport messages.
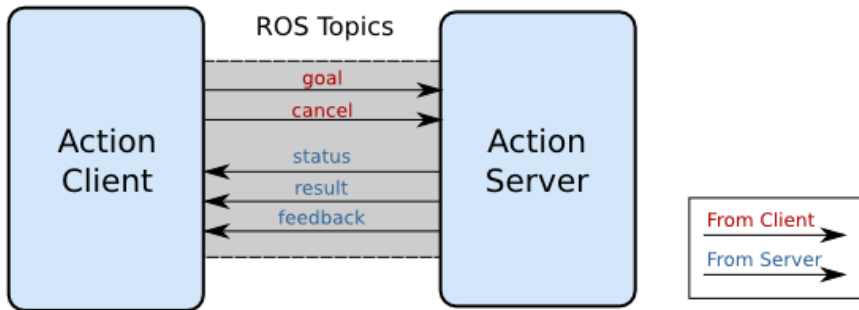
Action Interface

Illustration source: ROS actionlib wiki

Action Definitions

- Similar to messages and services.
- Definition: request + result + feedback
- Defined in *your_package/action/*.action*
- Example: *actionlib_tutorials/Fibonacci.action*

```
# goal definition
int32 order
---
# result definition
int32[] sequence
---
# feedback
int32[] sequence
```

Outline

Organizational

Links

- roslisp_common repo:

  https://github.com/cram2/roslisp_common

- ActionLib Lisp tutorials:

  http://wiki.ros.org/actionlib_lisp/Tutorials

Info

- Last assignment this week (5 out of 50 points)
- Assignment code: REPO/assignment_7_README.txt
- Next class: 08.12, 14:15, TAB 1.58, bring your laptops!

Thanks for your attention!