Robot Programming with Lisp

8. Coordinate Transformations, TF, ActionLib

Gayane Kazhoyan
(and other members of IAI)

Institute for Artificial Intelligence
University of Bremen

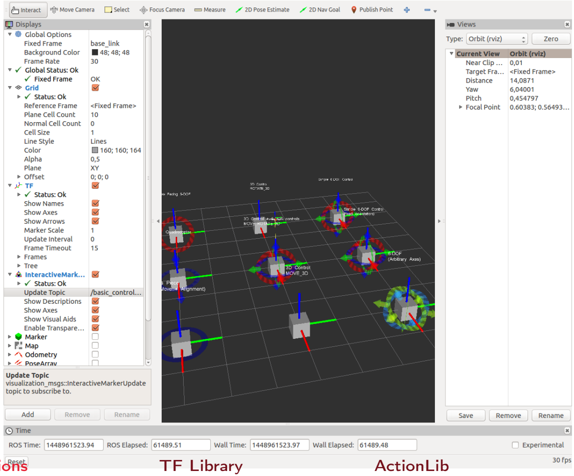
December 6th, 2018

Outline

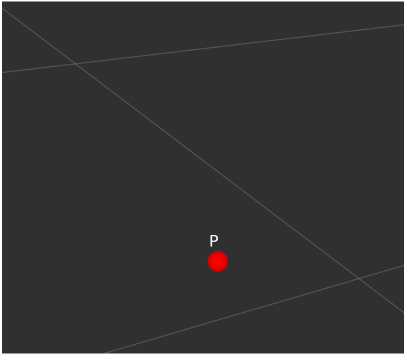
Outline

Intuition

```
$ roscore
$ rosrun interactive_marker_tutorials basic_controls
$ rosrun rviz rviz
```



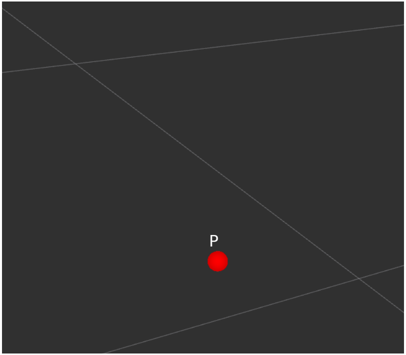
3D Geometry Basics
Coordinates of a point



- What is a point in space? How do we represent it?

3D Geometry Basics

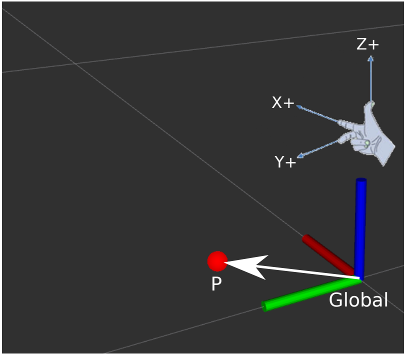
Coordinates of a point



- What is a point in space? How do we represent it?
- Cartesian coordinates (x, y, z)

3D Geometry Basics

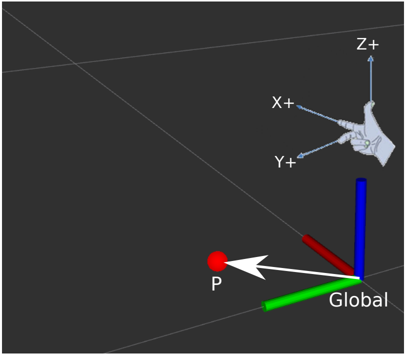
Coordinates of a point



- What is a point in space? How do we represent it?
- Cartesian coordinates (x, y, z)
- Reference frame

$_{global}P = (0.1, 0.1, 0.0)$

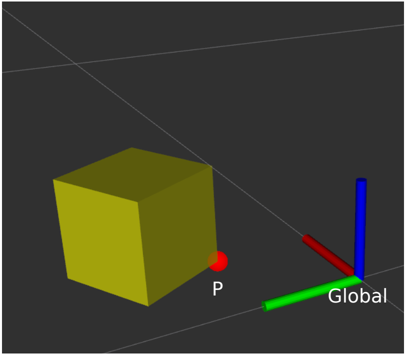
3D Geometry Basics
Coordinates of a point



- What is a point in space? How do we represent it?
- Cartesian coordinates (x, y, z)
- Reference frame
  ${}_{global}P = (0.1, 0.1, 0.0)$
- Right-hand rule:
  $(X, Y, Z) \rightarrow (R, G, B)$

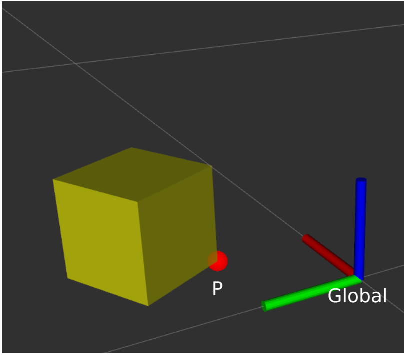
3D Geometry Basics

Coordinates of an object



- How do we represent an object in 3D?

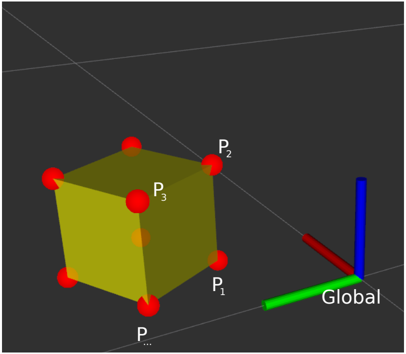
3D Geometry Basics
Coordinates of an object



- How do we represent an object in 3D?
- What is an object?

3D Geometry Basics

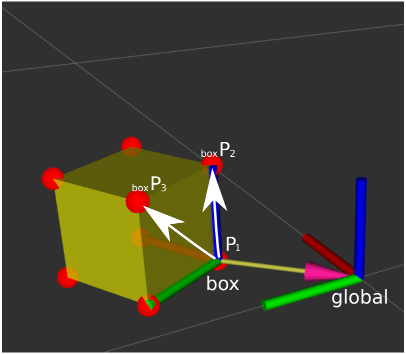
Coordinates of an object



- How do we represent an object in 3D?
- What is an object?
- Problem: all vertices change coordinates during movement

3D Geometry Basics

Coordinates of an object

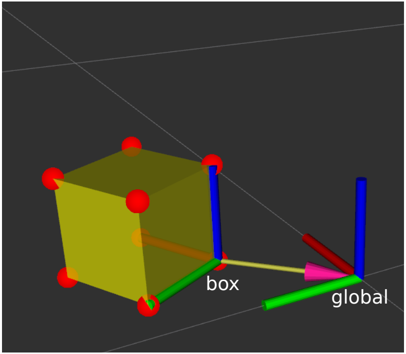


- How do we represent an object in 3D?
- What is an object?
- Problem: all vertices change coordinates during movement
- Solution: describe points on object relative to an object frame

$$_{global}P_1 = (0.1, 0.1, 0.0)$$
$$_{box}P_1 = (0.0, 0.0, 0.0)$$

3D Geometry Basics

Coordinates of an object



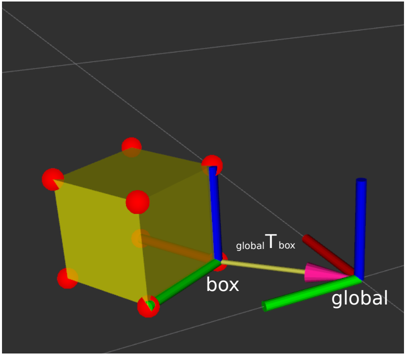
- How do we represent an object in 3D?
- What is an object?
- Problem: all vertices change coordinates during movement
- Solution: describe points on object relative to an object frame

  $_{global}P_1 = (0.1, 0.1, 0.0)$

  $_{box}P_1 = (0.0, 0.0, 0.0)$
- What do we need to describe the object frame?

3D Geometry Basics

Coordinates of a frame



- *box* has a position and orientation relative to *global*
- *position* & *orientation* together are called *pose*
- ${}_{global}T_{box}$ is a transformation that transforms poses from *box* to *global*
- How do we represent position and orientation?

Outline

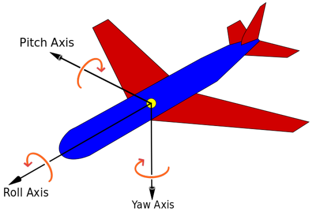
Rotation Representations

There are 4 common ways to describe rotations:

- euler angles
- rotation matrix
- axis-angle
- quaternion

Euler Angles

- Describes orientation using 3 angles:
  roll (x-rotation), pitch (y-rotation), yaw (z-rotation)
- Rotations are applied in sequence.
  What is the sequence is defined through a convention.
  There are many conventions, most common are z-y-x, x-y-z and z-x-z



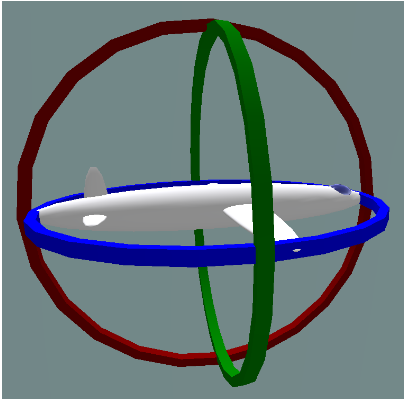
Euler Angles
Pros/Cons

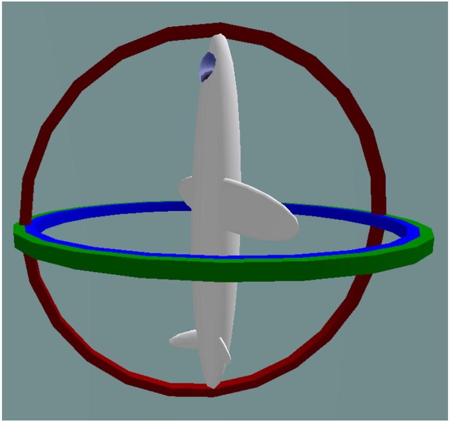
+ easy to interpret
– has a Gimbal lock problem
– not suited for interpolation
– there are many possible conventions, always make sure you know which one is used!

→ only useful for user interaction

Euler Angles
Gimbal lock

Loss of one degree of freedom, e.g. after 90° pitch (in this case red axis).





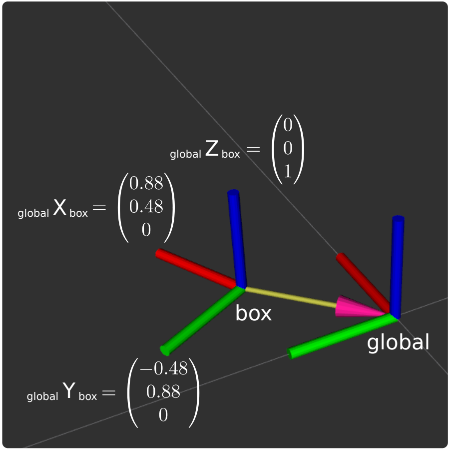
Rotation Matrix

- 3 x 3 matrix R
- is an orthogonal matrix, i.e. $det(R) = 1$ and $R^{-1} = R^T$
- this means, all raw (and correspondingly column) vectors are unit vectors, orthogonal to each other
- example: $R = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$ rotates about z-axis by θ

Rotation Matrix

Interpretation

- example: $R = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$ rotates about z-axis by θ
- $_{global}R_{box} = \begin{pmatrix} 0.88 & -0.48 & 0 \\ 0.48 & 0.88 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
- columns are axis of `box` in the `global` coordinate frame



Rotation Matrix

Pros/Cons

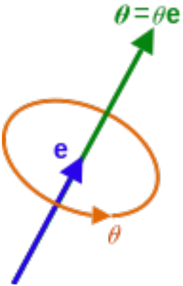
- \+ easiest to do math with
  - – rotate a vector with rotation matrix using matrix multiplication
  - – rotation matrices can be combined using matrix multiplication
- \+ easy to construct rotation matrix from 3 vectors
- \+ can be extended to include translation in 4x4 matrix
- – uses 9 numbers to describe 3 degrees of freedom
- – matrix operations result in buildup of rounding error, you might have to normalize often
- – not suitable for interpolation

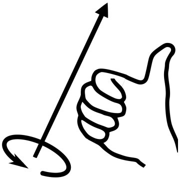
Axis-Angle

- any rotation can be represented as right hand rotation by θ degree about a unit vector e
- angle can be encoded in length of the vector

$$\begin{pmatrix} e_x \\ e_y \\ e_z \end{pmatrix}, \theta \rightarrow \begin{pmatrix} \theta e_x \\ \theta e_y \\ \theta e_z \end{pmatrix}$$

- can be rotated by rotation matrices using matrix multiplication





Axis-Angle
Pros/Cons

- – math can get unstable when θ is close to 0 or π, because there are infinitively many possible axis
- – represents rotation by θ differently from $\theta + 2\pi$, but it is the same rotation
- + easy interpolation, just scale the angle, but take into account that $\theta = \theta + 2\pi$
- → more useful when describing rotation differences/changes instead of orientations, found in ROS messages like Twist or Accel.

Quaternion

- $q = (x, y, z, w)$
- number system introduced by Hamilton as an extension of complex numbers, only use case is representation of rotations
- only unit quaternions are used to represent rotations
- can be interpreted as an improved version of axis-angle
- $\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}, \alpha \rightarrow \begin{pmatrix} a_x \cdot \sin(\alpha/2) \\ a_y \cdot \sin(\alpha/2) \\ a_z \cdot \sin(\alpha/2) \\ \cos(\alpha/2) \end{pmatrix}$

Quaternion
Pros/Cons

- + in contrast to axis-angle, stable when angle is close to zero and π
- + removes the $\theta = \theta + 2\pi$ problem from axis-angle
- + more compact representation than rotation matrices
- + best for interpolation (slerp algorithm)
- – difficult to interpret

$\rightarrow$ most useful for interpolation and describing orientations
ROS standard for representing poses

Rotations representations

Conclusion

- use euler angles only on an interface level
- use axis-angle or quaternion for rigid body dynamics
- use quaternions when storing/sending orientation information or for interpolation
- else use rotation matrices for easy mathematical operations

Outline

Homogeneous Transformations

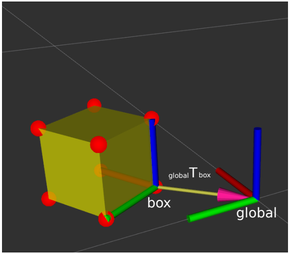
- 4 x 4 matrix to represent pose transformations
- ${}_aT_b$ means transform from frame b to a, i.e.: ${}_aT_b \cdot {}_bP = {}_aP$
- ${}_aT_b$ is the same as ${}_aP_b$, i.e. pose of origin of b in a
- combined transformation: ${}_cT_b \cdot {}_bT_a = {}_cT_a$
- invertible: ${}_bT_a^{-1} = {}_aT_b$
- but ${}_bT_a^{-1} \neq {}_bT_a^T$

Rotation Matrix

$$\begin{pmatrix} r_{0,0} & r_{0,1} & r_{0,2} & x \\ r_{1,0} & r_{1,1} & r_{1,2} & y \\ r_{2,0} & r_{1,2} & r_{2,2} & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Translation

Fixed

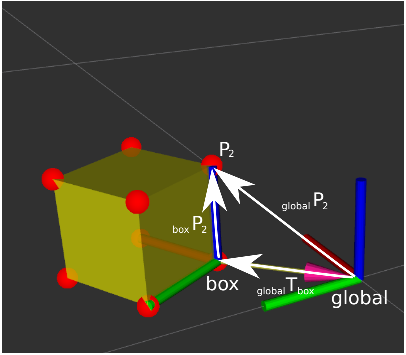


Homogeneous Transformation

- How do we do ${}_cT_b \cdot {}_bP = {}_cP$?
- Append 1 to point P, before matrix multiplication:

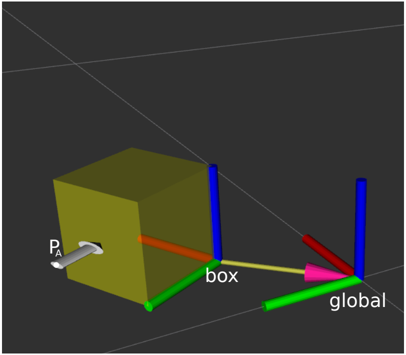
$$\begin{pmatrix} r_{0,0} & r_{0,1} & r_{0,2} & x \\ r_{1,0} & r_{1,1} & r_{1,2} & y \\ r_{2,0} & r_{2,1} & r_{2,2} & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} r_{0,0}p_x + r_{0,1}p_y + r_{0,2}p_z + x \cdot 1 \\ r_{1,0}p_x + r_{1,1}p_y + r_{1,2}p_z + y \cdot 1 \\ r_{2,0}p_x + r_{2,1}p_y + r_{2,2}p_z + z \cdot 1 \\ 0p_x + 0p_y + 0p_z + 1 \cdot 1 \end{pmatrix}$$

Homogeneous Transformation



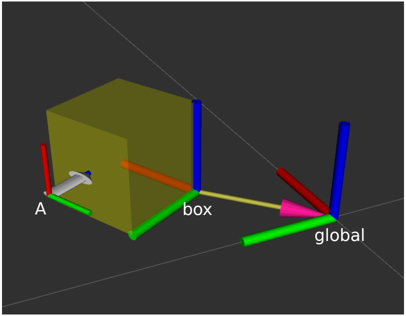
- to transform ${}_{box}P_2$ into the global frame ${}_{global}P_2$, multiply with ${}_{global}T_{box}$
- ${}_{global}P_2 = {}_{global}T_{box} \cdot {}_{box}P_2$

Homogeneous Transformation



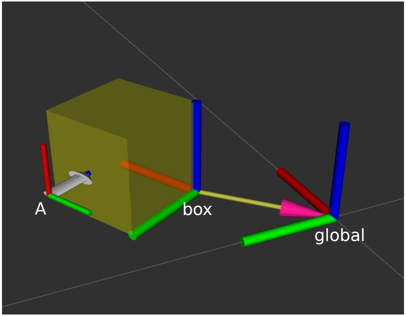
- what is the pose of P_A in global coordinate frame: ${}_{global}P_A$?
- choose frame where it is the easiest to express a pose
- ${}_{box}P_A = (0.05, 0.15, 0.05, 1.0)$
- ${}_{global}P_A = {}_{global}T_{box} \cdot {}_{box}P_A$

Homogeneous Transformation



$$_{box}T_A = \begin{pmatrix} & & & 0.05 \\ & & & 0.15 \\ & & & 0.05 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Homogeneous Transformation



$$
{}_{box}T_A = \begin{pmatrix} 0 & -1 & 0 & 0.05 \\ 0 & 0 & -1 & 0.15 \\ 1 & 0 & 0 & 0.05 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

Points in ROS Lisp

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 / ...

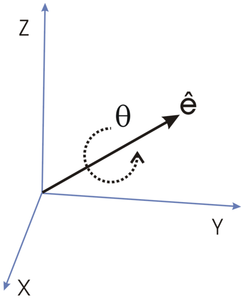
Rotations in ROS Lisp

Axis-Angle representation:

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

Axis-Angle $\rightarrow$ 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

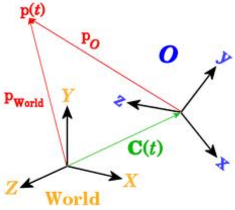
```
CL-TRANSFORMS> (make-quaternion 0 0 0 1)
CL-TRANSFORMS> (describe *)
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>
  [standard-object]
Slots with :INSTANCE allocation:
  X   = 0.0d0
  Y   = 0.0d0
  Z   = 0.0d0
  W   = 1.0d0
CL-TRANSFORMS> (axis-angle->quaternion
                 (make-3d-vector 0 0 1)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

Poses in ROS 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)>
```

Transformations in ROS Lisp

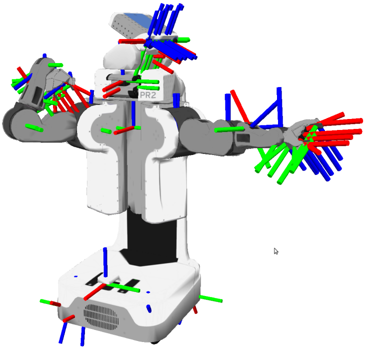


Transformations

```
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)>>
```

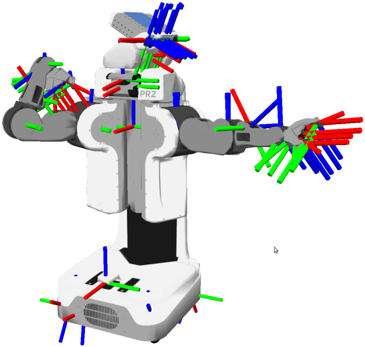
Outline

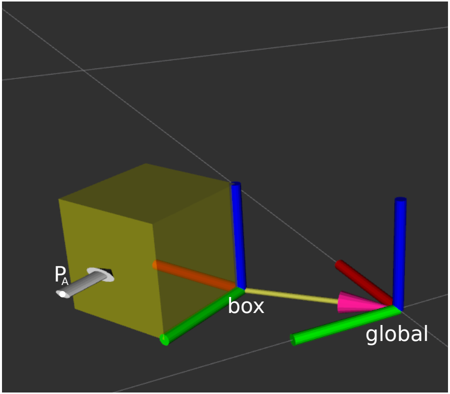
Motivation



- Robots consist of many *parts* aka *links*
- Each link has its own *coordinate frame*
- Links change their position over time (including the robot base)
- Sensors measurements are defined in their own frame
- Example: transformations from camera to hand coordinates are needed for grasping objects

Motivation





TurtleBot Coordinate Frames

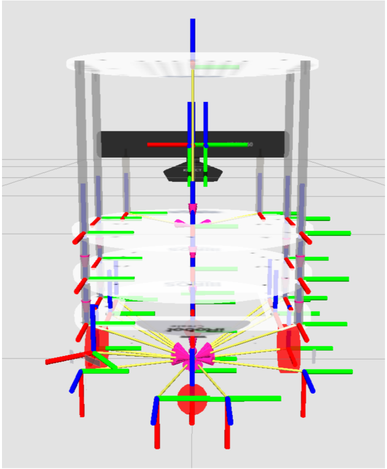


Image courtesy: Yujin Robot

Tracking Coordinate Frame Changes

- Transforms are produced by different nodes:
  - Localization node (AMCL, gmapping) for finding robot's pose in map
  - Odometry node (base driver) for tracking movement since initial pose
  - Joint positions (robot controllers and robot_state_publisher)
- Many publishers, many consumers
- Distributed system, redundancy issues, ...

↓

- **TF**: a coordinate frame tracking system

What is tf?

transform Library – a distributed coordinate frame tracking system

- Standardized protocol for publishing transforms to tf listeners
- Looking up and calculating transforms by asking tf listeners
- tf listener can be either local Lisp program or global tf buffer
- default global tf buffer is TF2's buffer_server
- ROS API for looking up, calculating and sending transforms
- Transforms are published on `/tf` and `/tf_static` topics:

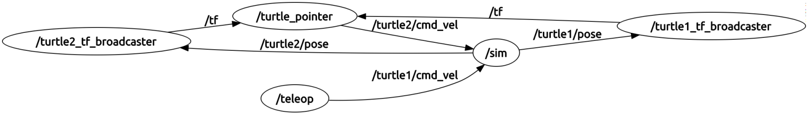
  `/tf`
  - for all transforms that change over time
  - publish with a fixed rate, even if transform didn't change

  `/tf_static`
  - assumed to be static, thus never outdated
  - useful for reducing redundancy
  - only publish once with latched flag

TurtleSim TF

Launch the turtlesim TF demo:

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



Utilities

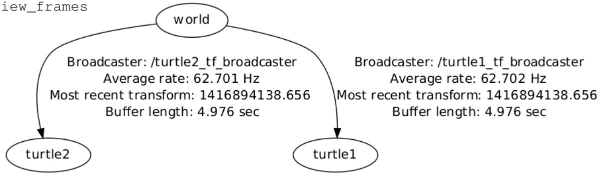
- `view_frames`
- `tf_echo`
- `tf_monitor`
- `static_transform_publisher`
- RViz

Utilities

rosrun tf view_frames

Generate a TF tree graph:

```
$ rosrun tf view_frames
```



- TF tree consists of frames (links) and the transforms between them.
- Each transform is cached (10 secs default caching time)
- Transforms must form a proper tree (no cycles)
- Can have disconnected trees, but you can only ask for transforms inside of the same tree

Utilities

tf_echo

```
$ rosrun tf tf_echo <source_frame> <target_frame>
```

tf_echo

```
$ rosrun tf tf_echo turtle1 turtle2
At time 0.000
- Translation: [0.100, 0.100, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.247, 0.969]
            in RPY (radian) [0.000, -0.000, 0.500]
            in RPY (degree) [0.000, -0.000, 28.648]
```

Utilities

static_transform_publisher

- `rosrun tf2_ros static_transform_publisher x y z yaw pitch roll frame_id child_frame_id`
  or
  `rosrun tf2_ros static_transform_publisher x y z qx qy qz qw frame_id child_frame_id`
- publishes ${}_{global}T_{box}$

static_transform_publisher

```
$ rosrun tf2_ros static_transform_publisher 0.1 0.1 0 3.14 0 0 global box
```

Utilities

tf_monitor

- rosrun tf tf_monitor

tf_monitor

```
$ rosrun tf tf_monitor
RESULTS: for all Frames

Frames:
Frame: turtle1 published by /turtle1_tf_broadcaster Average Delay: 0.000382455 Max Delay: 0...
Frame: turtle2 published by /turtle2_tf_broadcaster Average Delay: 0.000267847 Max Delay: 0...

All Broadcasters:
Node: /turtle1_tf_broadcaster 64.6996 Hz, Average Delay: 0.000382455 Max Delay: 0.000991178
Node: /turtle2_tf_broadcaster 64.7127 Hz, Average Delay: 0.000267847 Max Delay: 0.00133464
```

TF data types

- `frame_id`: name of the published frame
- `child_frame_id` has to be an existing frame
- `stamp`: time when this transform is valid
- ${}^{child\_frame\_id}T_{frame\_id}$

tf2_msgs/TFMessage

```
geometry_msgs/TransformStamped[]
transforms
 std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
 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
```

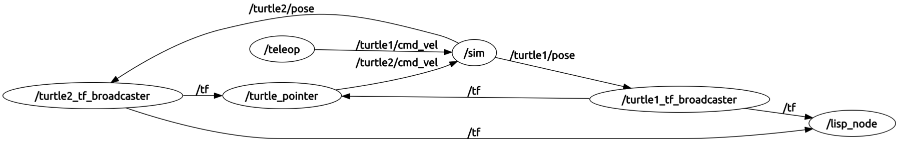
TF and time

- tf buffers transforms for X seconds
- possible to lookup transforms from the past
- tf interpolates frames
- tf does not extrapolate! it can't see into the future

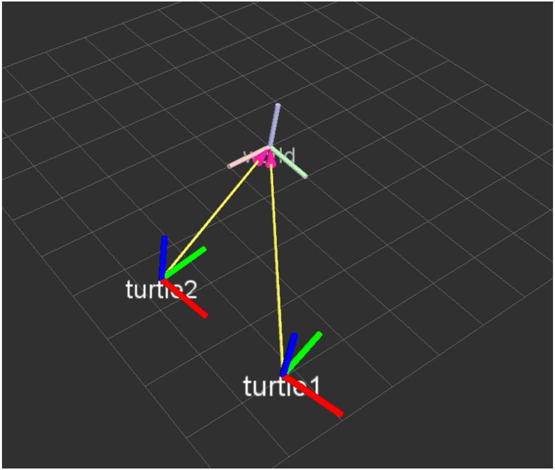
Lisp TF

cl_tf

```
TF> (roslisp:start-ros-node "lisp_node")
TF> (defparameter *transform-listener*
       (make-instance 'transform-listener))
TF> (lookup-transform *transform-listener* :source-frame "turtle1"
                                           :target-frame "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

ROS Actions

Interface to define and execute goals:

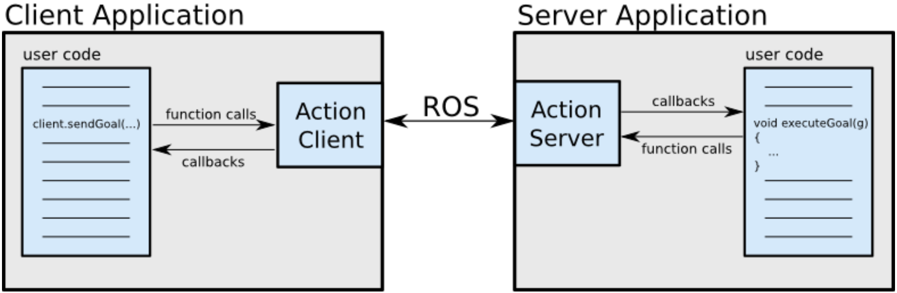


Illustration source: ROS actionlib wiki

Action Protocol

Relies on ROS topics to transport messages.

Action Interface

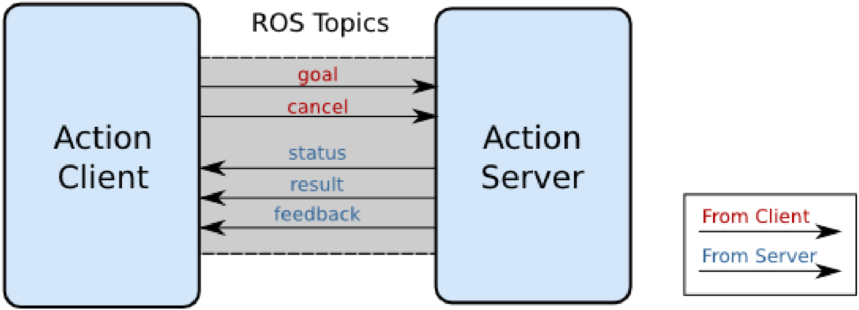


Illustration source: ROS actionlib wiki

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

Info

- Assignment points: 10 points
- TF Lisp tutorial:
  http://wiki.ros.org/cl_tf/Tutorials/clTfBasicUsage
- ActionLib Lisp tutorial (Section 1 and 2, not 3):
  http://wiki.ros.org/actionlib_lisp/Tutorials/actionlibBasicUsage
- Next class: 13.12, 14:00!, bring your laptops!

Q & A

Thanks for your attention!