



Universität
Bremen

Institute for Artificial
Intelligence

Faculty 03

Mathematics &

Computer Science

Robot Programming with ROS

2. Coordinates & Transforms

Arthur Niedźwiecki, Stefan Eirich
26th Oct. 2023



Intuition

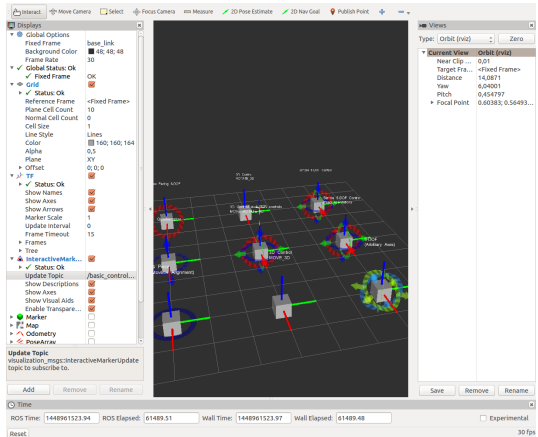
```
$ roscore
```

```
$ rosrn interactive_marker_tutorials basic_controls
```

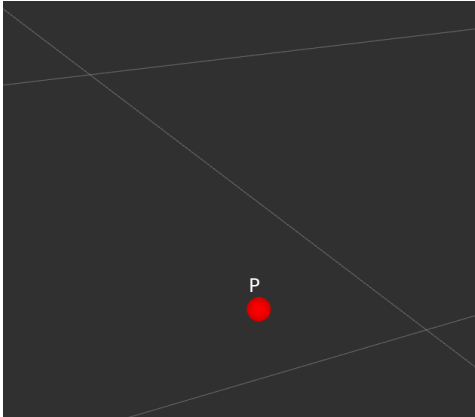
```
$ rosrn rviz rviz
```

Fixed Frame: base_link

Add by topic: InteractiveMarker

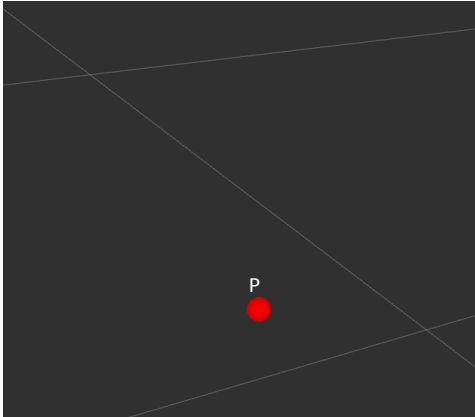


3D Geometry Basics



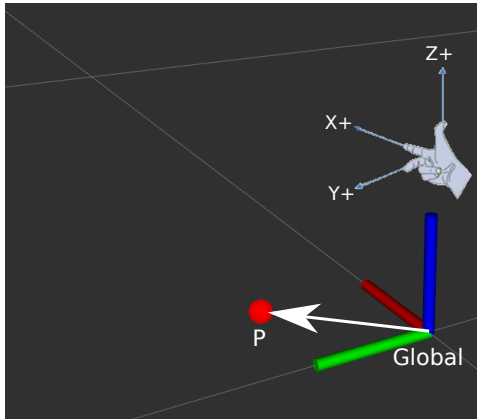
- What is a point in space? How do we represent it?

3D Geometry Basics



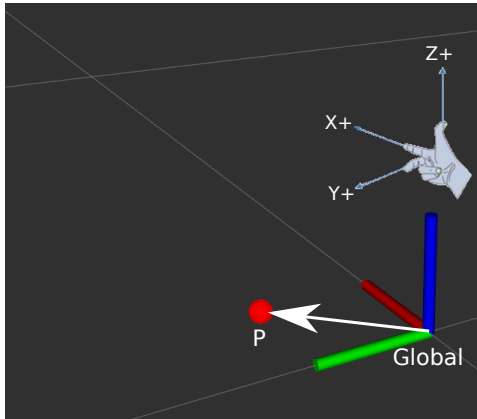
- What is a point in space? How do we represent it?
- Cartesian coordinates (x, y, z)

3D Geometry Basics



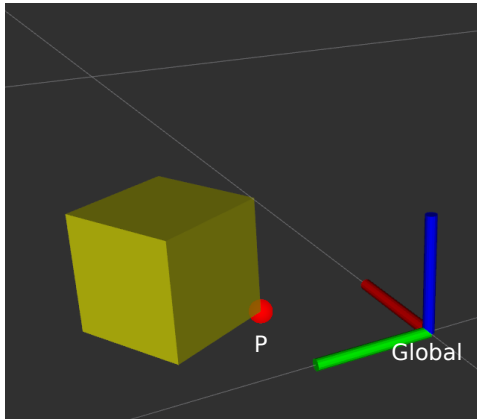
- What is a point in space? How do we represent it?
- Cartesian coordinates (x, y, z)
- Reference frame
 $global P = (0.1, 0.1, 0.0)$

3D Geometry Basics



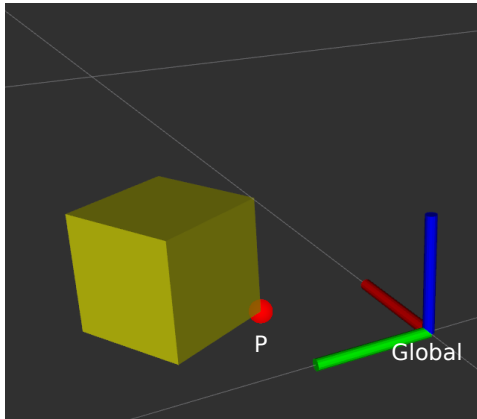
- What is a point in space? How do we represent it?
- Cartesian coordinates (x, y, z)
- Reference frame
 $_{global}P = (0.1, 0.1, 0.0)$
- Right-hand rule:
 $(X, Y, Z) \rightarrow (R, G, B)$

3D Geometry Basics



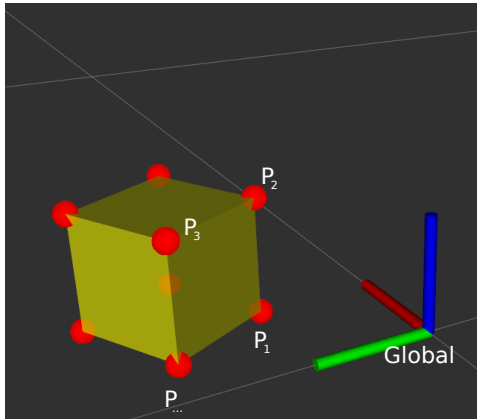
- How do we represent an object in 3D?

3D Geometry Basics



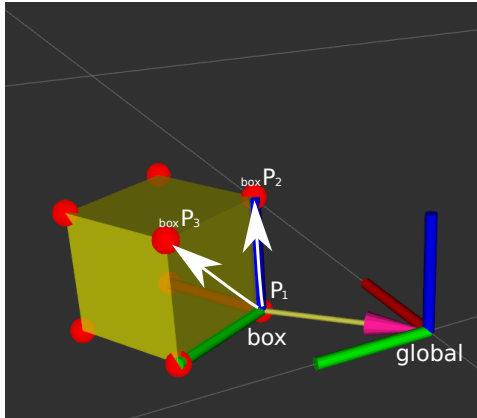
- How do we represent an object in 3D?
- What is an object?

3D Geometry Basics



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

3D Geometry Basics

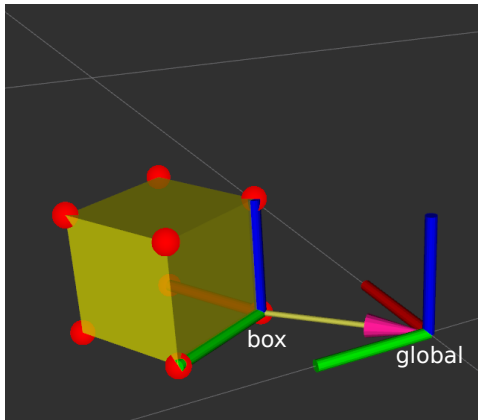


- How do we represent an object in 3D?
- What is an object?
- Problem: all vertices change coordinates during movement
- Solution: describe points on object relative to an object frame

$$global P_1 = (0.1, 0.1, 0.0)$$

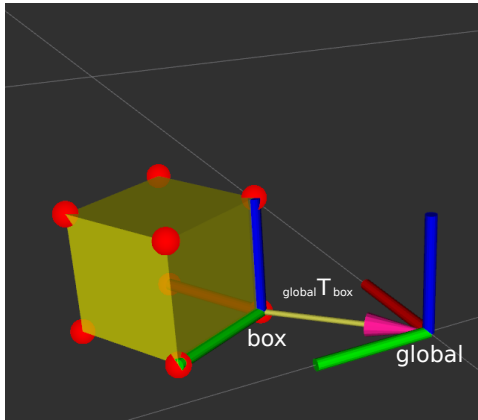
$$box P_1 = (0.0, 0.0, 0.0)$$

3D Geometry Basics



- 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



- *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?

Rotation Representations

There are 4 common ways to describe rotations:

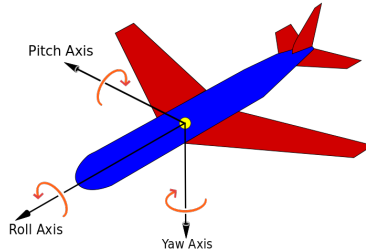
- euler angles
- rotation matrix
- axis-angle
- quaternion

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

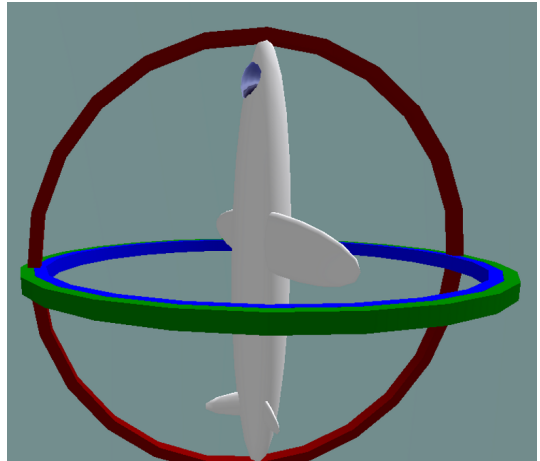
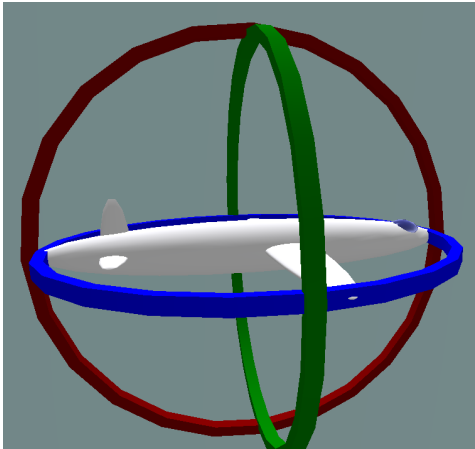


Euler Angles

- + 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

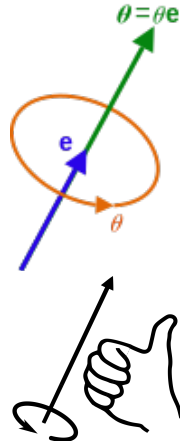
Loss of one degree of freedom, e.g. after 90° pitch (in this case red axis).



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

- 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

- $\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}, \alpha \rightarrow \begin{pmatrix} a_x \cdot \sin(\alpha/2) \\ a_y \cdot \sin(\alpha/2) \\ a_z \cdot \sin(\alpha/2) \\ \cos(\alpha/2) \end{pmatrix}$

Rotation Matrix

- 3 x 3 matrix R
- is an orthogonal matrix, i.e. $\det(R) = 1$ and $R^{-1} = R^T$
- this means, all row (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

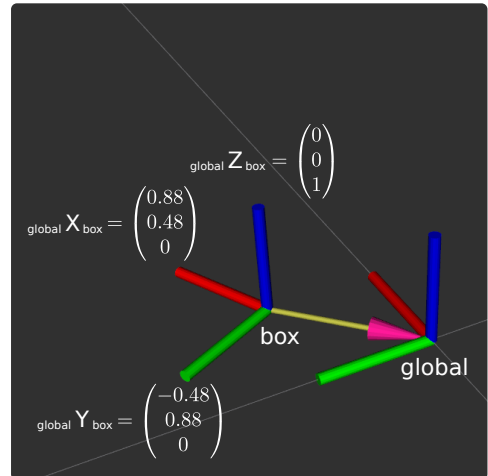
- example:

$$R = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

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

- + 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

Quaternion

- + 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
- most useful for interpolation and describing orientations
ROS standard for representing poses

Rotations representations

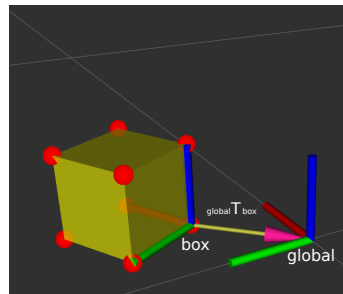
- use euler angles only on an interface level
- use axis-angle or quaternion for rigid body dynamics
- use quaternions when storing/sending orientation information or for interpolation
- else use rotation matrices for easy mathematical operations

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 ${}_bT_a^{-1} \neq {}_bT_a^T$

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

Rotation Matrix Translation
Fixed

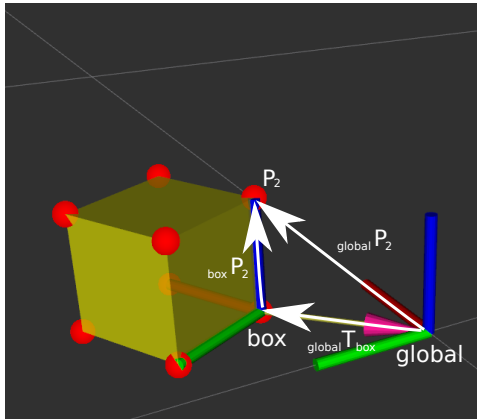


Homogeneous Transformation

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

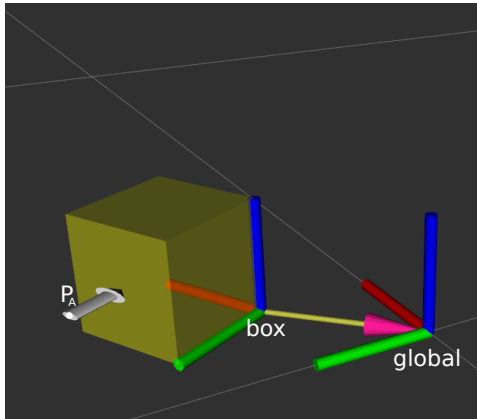
$$\begin{pmatrix} r_{0,0} & r_{0,1} & r_{0,2} & x \\ r_{1,0} & r_{1,1} & r_{1,2} & y \\ r_{2,0} & r_{2,1} & r_{2,2} & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} r_{0,0}p_x + r_{0,1}p_y + r_{0,2}p_z + x \cdot 1 \\ r_{1,0}p_x + r_{1,1}p_y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

Homogeneous Transformation



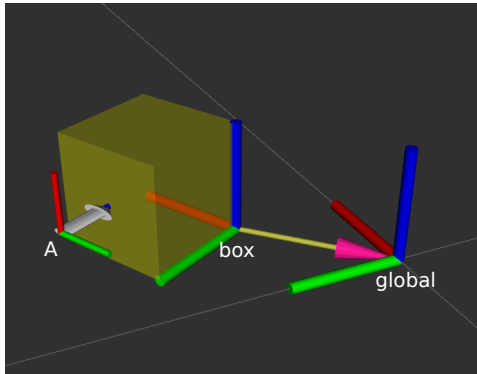
- to transform ${}_{box}P_2$ into the global frame ${}_{global}P_2$, multiply with ${}_{global}T_{box}$
- ${}_{global}P_2 = {}_{global}T_{box} \cdot {}_{box}P_2$

Homogeneous Transformation



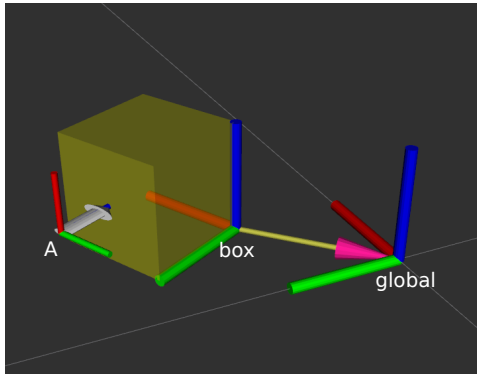
- what is the pose of P_A in global coordinate frame: $global P_A$?
- choose frame where it is the easiest to express a pose
- $box P_A = (0.05, 0.15, 0.05, 1.0)$
- $global P_A = global T_{box} \cdot box P_A$

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

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

Point

```
1 from geometry_msgs.msg import Vector3
2
3 point = Vector3(1, 2, 0)
4 print(point)    # x: 1, y: 2, z: 0
5 print(point.y)  # 2
```

Object in 3D: $\{position, orientation\}$

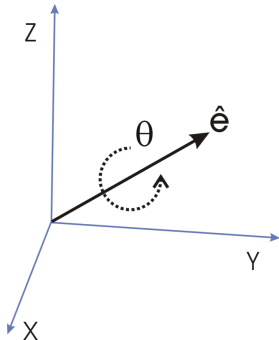
Position: $\{x, y, z\}$

Orientation: axis-angle / rotation matrix / quaternions / ...

Rotations in ROS

Axis-Angle representation:

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



Axis-Angle $\rightarrow$ Quaternion:

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

Quaternion

```
1 from geometry_msgs.msg import Quaternion
2
3 quaternion = Quaternion(0, 0, 0, 1)
4 print(quaternion) # x: 0, y: 0, z: 0, w: 1
```

Poses in ROS

Pose Matrix

```
1 from geometry_msgs.msg import Pose
2
3 pose = Pose(point, quaternion)
4 print(pose)
5 # position:
6 # x: 1
7 # y: 2
8 # z: 0
9 # orientation:
10 # x: 0
11 # y: 0
12 # z: 0
13 # w: 1
```

Transformations in ROS

Identity Matrix

```
1 import tf.transformations as tr
2 I = tr.identity_matrix()
3 #[[1. 0. 0. 0.]
4 # [0. 1. 0. 0.]
5 # [0. 0. 1. 0.]
6 # [0. 0. 0. 1.]]
```

Docs: <https://docs.ros.org/en/jade/api/tf/html/python/transformations.html>

Transformations in ROS

Translation + Rotation

```
1 import numpy
2 Rz = tr.rotation_matrix(numpy.pi*0.5, (0, 0, 1))
3 Vx = tr.translation_matrix([2, 0, 0])
4 Vx_Rz = tr.concatenate_matrices(Vx, Rz)
5 print(Vx_Rz)
6 #[[0. -1.  0.  2.]
7  # [1.  0.  0.  0.]
8  # [0.  0.  1.  0.]
9  # [0.  0.  0.  1.]]
```

Docs: <https://docs.ros.org/en/jade/api/tf/html/python/transformations.html>

Transformations in ROS

Inverse

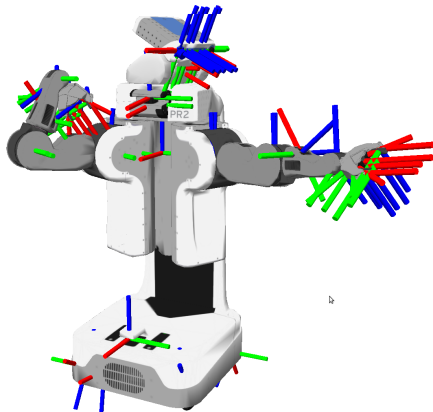
```
1 tr.inverse_matrix(Vx_Rz)
2 # [ [ 0.  1.  0.  0.]
3 # [-1.  0.  0.  2.]
4 # [ 0.  0.  1.  0.]
5 # [ 0.  0.  0.  1.]]
```

Docs: <https://docs.ros.org/en/jade/api/tf/html/python/transformations.html>

Plan

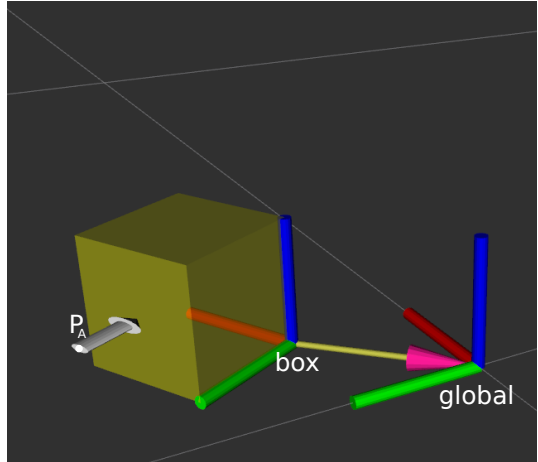
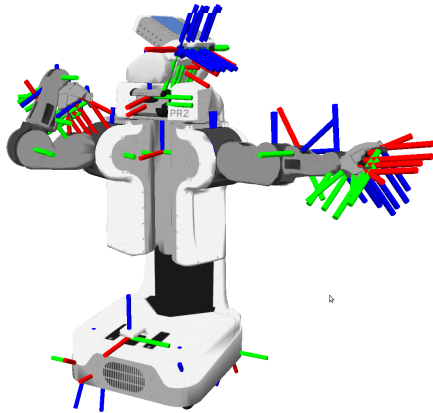
- 1 Coordinate Transformations
 - 3D Geometry Basics
 - Rotation Representations
 - Homogeneous Transformations
 - Transform Messages in ROS
- 2 TF Library
- 3 Organizational

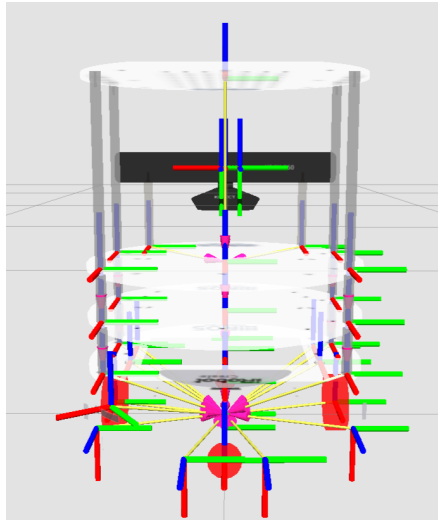
Motivation



- 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





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

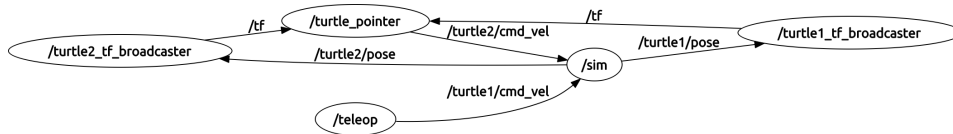
`/tf_static`

- 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
```



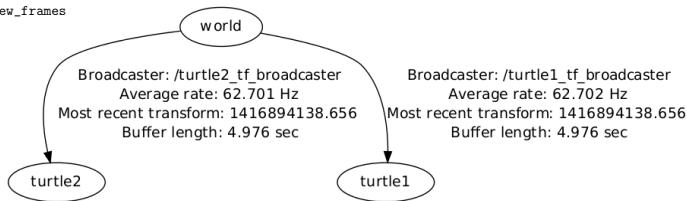
Utilities

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

Utilities

Generate a TF tree graph:

```
$ rosrn 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

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

tf_echo

```
1 $ rosrn tf tf_echo turtle1 turtle2
2 At time 0.000
3 - Translation: [0.100, 0.100, 0.000]
4 - Rotation: in Quaternion [0.000, 0.000, 0.247, 0.969]
5             in RPY (radian) [0.000, -0.000, 0.500]
6             in RPY (degree) [0.000, -0.000, 28.648]
```

Utilities

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

static_transform_publisher

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

Utilities

- `roslaunch tf tf_monitor`

tf_monitor

```
$ roslaunch 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

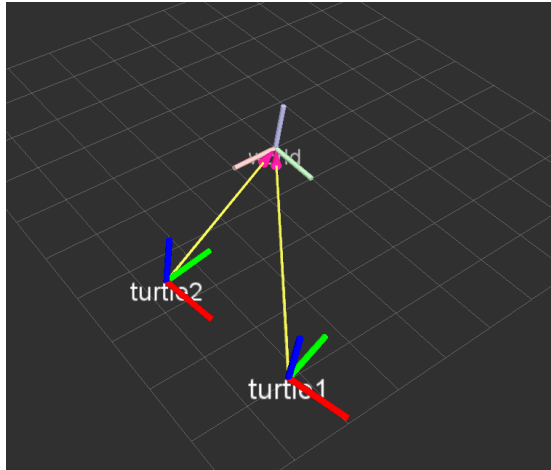
- tf buffers transforms for X seconds
- possible to lookup transforms from the past
- tf interpolates frames
- tf does not extrapolate! it can't see into the future

Python TF

tf

```
1 import rospy
2 import tf
3 rospy.init_node('turtle_tf_listener')
4 listener = tf.TransformListener()
5 rospy.wait_for_service('spawn')
6 rate = rospy.Rate(10.0)
7 while not rospy.is_shutdown():
8     try:
9         (trans, rot) = listener.lookupTransform('/turtle2', '/turtle1', rospy.Time(0))
10        print(trans, rot)
11    except (tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException):
12        print("No frame found")
13        pass
14    rate.sleep()
```

\$ rosrun rviz rviz



Links

- Gilbert Strang's MIT course on linear algebra (free access):

<https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/>

- 3Blue1Brown's essence of linear algebra (Playlist):

https://www.youtube.com/watch?v=fNk_zzaMoSs&list=PLZHQ0b0WTQDPD3MizzM2xVFitgF8hE_ab

- Writing a TF Listener:

<https://wiki.ros.org/tf/Tutorials/Writing%20a%20tf%20listener%20%28Python%29>

- Documentation of the tf.transformations package:

<https://docs.ros.org/en/jade/api/tf/html/python/transformations.html>

Plan

- 1 Coordinate Transformations
 - 3D Geometry Basics
 - Rotation Representations
 - Homogeneous Transformations
 - Transform Messages in ROS
- 2 TF Library
- 3 Organizational

Assignment 2

- Update repository:

```
$ git pull origin
```

- Check the new assignment:

```
https://github.com/artnie/rpwr-assignments
```

- Solve in the Binder:

```
https://binder.intel4coro.de/v2/gh/artnie/pycram/binder
```

Info

- Assignment points: 8 points
- Due: 01.11.23
- Next class: 02.11.23, 14:00

Q & A

Thanks for your attention!