Robot description & Coordinate Transformations

Dr Gerardo Aragon-Camarasa
gerardo.aragoncamarasa@glasgow.ac.uk

RF – University of Glasgow

Position and Orientation

- Why? Represent the “pose” of objects in the environment.

  - Robotic limbs
  - Tools
  - Obstacles
  - Cameras
  - Sensors
  - Paths
  - …
  - Etc.

- Pose of an object
  - Object’s reference frame with respect to another reference frame

Math Review

- Remember DF (H)?

Matrix $(m \times n)$

$$\mathbf{A} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix}$$

Transpose $(n \times m)$

$$\mathbf{A}^T = \begin{bmatrix} a & d & g & j \\ b & e & h & k \\ c & f & j & l \end{bmatrix}$$

Math Review

- Matrix-Vector product
  - Transforms one vector into another

$$\mathbf{y} = \mathbf{A}\mathbf{x} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} ax_1 + bx_2 + cx_3 \\ dx_1 + ex_2 + fx_3 \\ gx_1 + hx_2 + ix_3 \\ jx_1 + kx_2 + lx_3 \end{bmatrix}$$

$$(n \times 1) = (n \times m)(m \times 1)$$

- Matrix-Matrix product
  - Produces a new matrix

$$\mathbf{A} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}; \quad \mathbf{B} = \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix}; \quad \mathbf{AB} = \begin{bmatrix} (a_1b_1 + a_2b_3) & (a_1b_2 + a_2b_4) \\ (a_3b_1 + a_4b_3) & (a_3b_2 + a_4b_4) \end{bmatrix}$$

$$(n \times n) = (n \times l)(l \times m)$$

Math Review

- Identity matrix
  - No change when it multiplies a vector or matrix

$$\mathbf{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{x} = \mathbf{I}\mathbf{x}$$

- A non-singular square matrix multiplied by its inverse is

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$$

Math Review

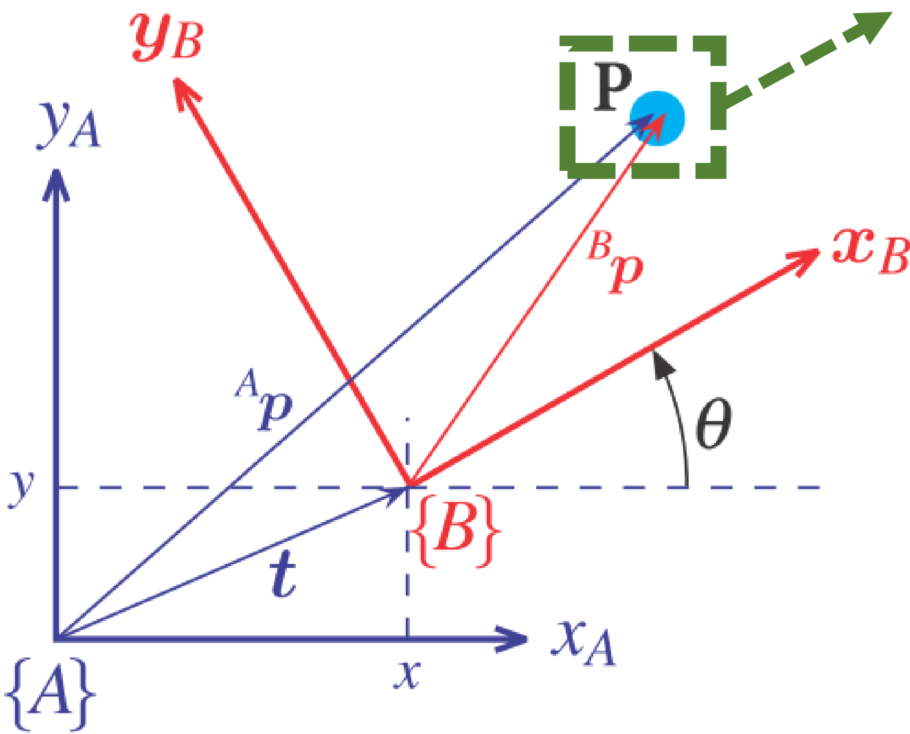
For $\mathbf{x}_2 = \mathbf{A}\mathbf{x}_1$; we get

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_2 = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_1$$

Now, for $\mathbf{x}_1 = \mathbf{A}^{-1}\mathbf{x}_2$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_1 = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_2$$

Position and Orientation → 2D pose



$$\mathbf{P} = \begin{bmatrix} P_{x_A} \\ P_{y_A} \end{bmatrix} = \begin{bmatrix} P_{x_B} \\ P_{y_B} \end{bmatrix}$$

Note: t_x and t_y are relative to the coordinate frame $\{A\}$. **Translation** followed by **rotation** is different than **rotation** followed by **translation**.

That is, knowing the coordinates of a point $[P_{x_B}, P_{y_B}]^T$ in some coordinate frame $\{B\}$, you can find the position of that point relative to the original coordinate frame $\{A\}$.

$$^Ap = \begin{bmatrix} P_{x_A} \\ P_{y_A} \end{bmatrix} = \underbrace{\boxed{\begin{bmatrix} t_x \\ t_y \end{bmatrix}}}_{\text{translation}} + \underbrace{\boxed{\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}}}_{\text{rotation}} \begin{bmatrix} P_{x_B} \\ P_{y_B} \end{bmatrix}$$

Position and Orientation → 3D pose

- Rotation matrices
  - 9 elements → 3 independent variables (angles of each axis)
- Sequence of 3 rotations around independent axes
  - Generally called Roll-Pitch-Yaw (fixed axes) or Euler (moving axes) angles

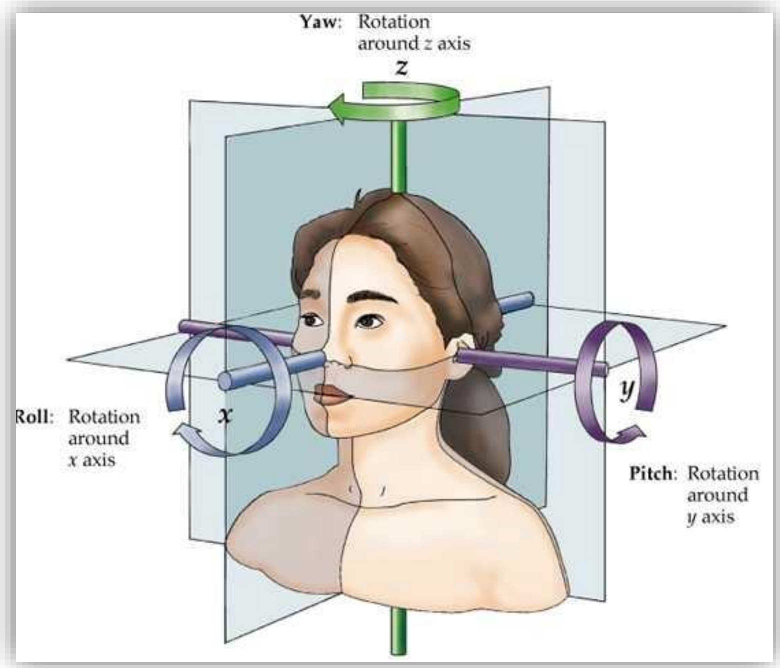


Fig from: http://www.utdallas.edu/~tres/integ/sen5/display9_23.html

Position and Orientation → 3D pose

- Euler angles → 12 different rotation sequences (actually 24!)
  - Proper Euler and Tait-Bryan angles

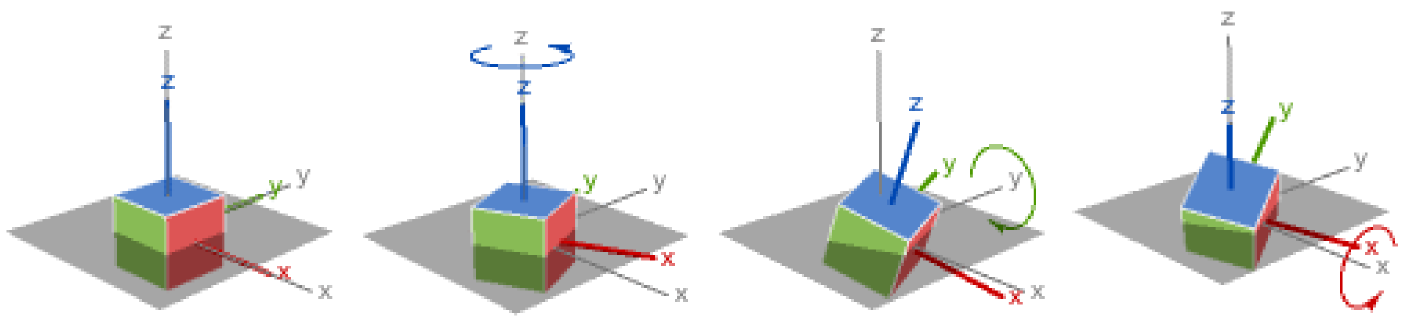
Proper Euler angles	Tait-Bryan angles
$X_1 Z_2 X_3 = \begin{bmatrix} c_2 & -c_3 s_2 & s_2 s_3 \\ c_1 s_2 & c_1 c_2 c_3 - s_1 s_3 & -c_3 s_1 - c_1 c_2 s_3 \\ s_1 s_2 & c_1 s_3 + c_2 c_3 s_1 & c_1 c_3 - c_2 s_1 s_3 \end{bmatrix}$	$X_1 Z_2 Y_3 = \begin{bmatrix} c_2 c_3 & -s_2 & c_2 s_3 \\ s_1 s_3 + c_1 c_3 s_2 & c_1 c_2 & c_1 s_2 s_3 - c_3 s_1 \\ c_3 s_1 s_2 - c_1 s_3 & c_2 s_1 & c_1 c_3 + s_1 s_2 s_3 \end{bmatrix}$
$X_1 Y_2 X_3 = \begin{bmatrix} c_2 & s_2 s_3 & c_3 s_2 \\ s_1 s_2 & c_1 c_3 - c_2 s_1 s_3 & -c_1 s_3 - c_2 c_3 s_1 \\ -c_1 s_2 & c_3 s_1 + c_1 c_2 s_3 & c_1 c_2 c_3 - s_1 s_3 \end{bmatrix}$	$X_1 Y_2 Z_3 = \begin{bmatrix} c_2 c_3 & -c_2 s_3 & s_2 \\ c_1 s_3 + c_3 s_1 s_2 & c_1 c_3 - s_1 s_2 s_3 & -c_2 s_1 \\ s_1 s_3 - c_1 c_3 s_2 & c_3 s_1 + c_1 s_2 s_3 & c_1 c_2 \end{bmatrix}$
$Y_1 X_2 Y_3 = \begin{bmatrix} c_1 c_3 - c_2 s_1 s_3 & s_1 s_2 & c_1 s_3 + c_2 c_3 s_1 \\ s_2 s_3 & c_2 & -c_3 s_2 \\ -c_3 s_1 - c_1 c_2 s_3 & c_1 s_2 & c_1 c_2 c_3 - s_1 s_3 \end{bmatrix}$	$Y_1 X_2 Z_3 = \begin{bmatrix} c_1 c_3 + s_1 s_2 s_3 & c_3 s_1 s_2 - c_1 s_3 & c_2 s_1 \\ c_2 s_3 & c_2 c_3 & -s_2 \\ c_1 s_2 s_3 - c_3 s_1 & c_1 c_3 s_2 + s_1 s_3 & c_1 c_2 \end{bmatrix}$
$Y_1 Z_2 Y_3 = \begin{bmatrix} c_1 c_2 c_3 - s_1 s_3 & -c_1 s_2 & c_3 s_1 + c_1 c_2 s_3 \\ c_3 s_2 & c_2 & s_2 s_3 \\ -c_1 s_3 - c_2 c_3 s_1 & s_1 s_2 & c_1 c_3 - c_2 s_1 s_3 \end{bmatrix}$	$Y_1 Z_2 X_3 = \begin{bmatrix} c_1 c_2 & s_1 s_3 - c_1 c_3 s_2 & c_3 s_1 + c_1 s_2 s_3 \\ s_2 & c_2 c_3 & -c_2 s_3 \\ -c_2 s_1 & c_1 s_3 + c_3 s_1 s_2 & c_1 c_3 - s_1 s_2 s_3 \end{bmatrix}$
$Z_1 Y_2 Z_3 = \begin{bmatrix} c_1 c_2 c_3 - s_1 s_3 & -c_3 s_1 - c_1 c_2 s_3 & c_1 s_2 \\ c_1 s_3 + c_2 c_3 s_1 & c_1 c_3 - c_2 s_1 s_3 & s_1 s_2 \\ -c_3 s_2 & s_2 s_3 & c_2 \end{bmatrix}$	$Z_1 Y_2 X_3 = \begin{bmatrix} c_1 c_2 & c_1 s_2 s_3 - c_3 s_1 & s_1 s_3 + c_1 c_3 s_2 \\ c_2 s_1 & c_1 c_3 + s_1 s_2 s_3 & c_3 s_1 s_2 - c_1 s_3 \\ -s_2 & c_2 s_3 & c_2 c_3 \end{bmatrix}$
$Z_1 X_2 Z_3 = \begin{bmatrix} c_1 c_3 - c_2 s_1 s_3 & -c_1 s_3 - c_2 c_3 s_1 & s_1 s_2 \\ c_3 s_1 + c_1 c_2 s_3 & c_1 c_2 c_3 - s_1 s_3 & -c_1 s_2 \\ s_2 s_3 & c_3 s_2 & c_2 \end{bmatrix}$	$Z_1 X_2 Y_3 = \begin{bmatrix} c_1 c_3 - s_1 s_2 s_3 & -c_2 s_1 & c_1 s_3 + c_3 s_1 s_2 \\ c_3 s_1 + c_1 s_2 s_3 & c_1 c_2 & s_1 s_3 - c_1 c_3 s_2 \\ -c_2 s_3 & s_2 & c_2 c_3 \end{bmatrix}$

From: https://en.wikipedia.org/wiki/Euler_angles#Rotation_matrix

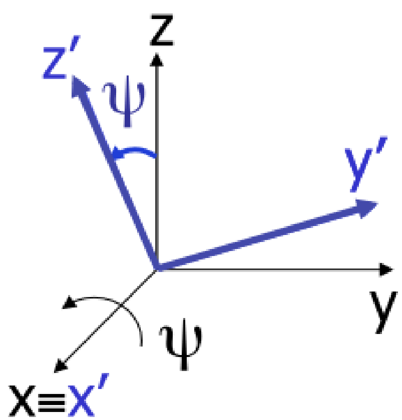
  - Because rotations in 3D do not commute, inverting the sequence of rotations gives a different outcome!

RPY & Euler Angles Convention

- In ROS and RF → we will use ZYX
  - RVC book follows ZYZ and interchanges with ZYX

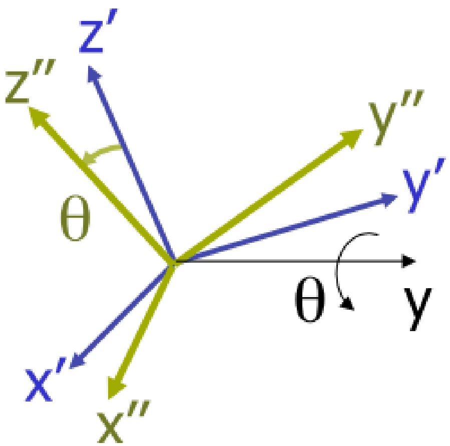


http://reference.wolfram.com/language/ref/RollPitchYawMatrix.html

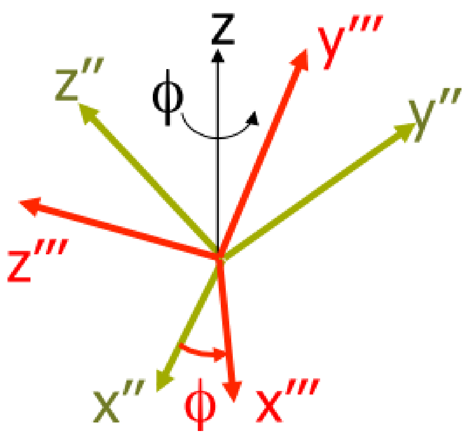


$$R_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix}$$

RPY & Euler Angles Convention



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



$$R_z(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Why this order?

$$\mathbf{R}_{RPY}(\psi,\theta,\phi) = \mathbf{R}_Z(\phi)\mathbf{R}_Y(\theta)\mathbf{R}_X(\psi)$$

order of definition

“reverse” order in the product
(pre-multiplication…)

- Need to refer each rotation in the sequence to one of the original **fixed** axes
  - Use of the angle/axis technique for each rotation in the sequence:

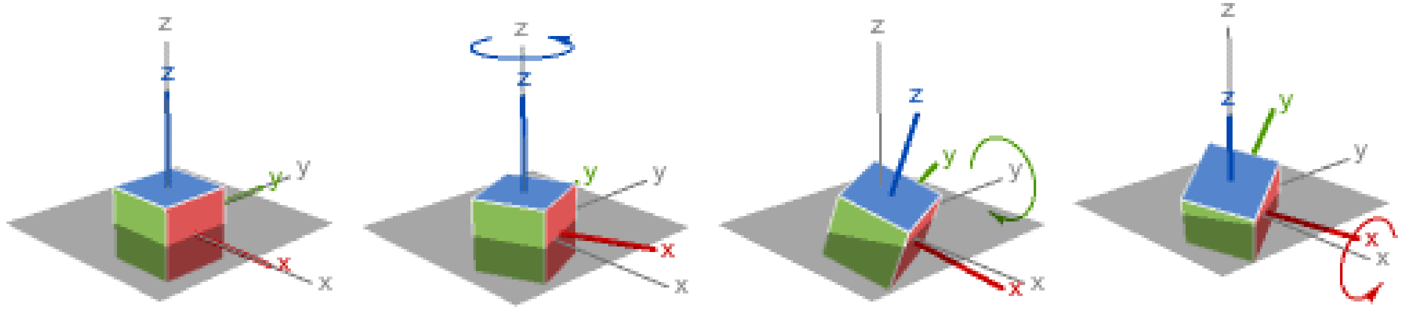
$$\mathbf{C}^T\mathbf{R}(\alpha)\mathbf{C}$$

with **C** being the rotation matrix *reverting* the previous rotation, i.e. go back to the original axes.

Concatenating three rotations (post-multiplication…)

$$\begin{aligned}\mathbf{R}_{RPY}(\psi,\theta,\phi) &= [\mathbf{R}_X(\psi)][\mathbf{R}_X^T(\psi)\mathbf{R}_Y(\theta)\mathbf{R}_X(\psi)] \\ &\quad [\mathbf{R}_X^T(\psi)\mathbf{R}_Y^T(\theta)\mathbf{R}_Z(\phi)\mathbf{R}_Y(\theta)\mathbf{R}_X(\psi)] \\ &= \mathbf{R}_Z(\phi)\mathbf{R}_Y(\theta)\mathbf{R}_X(\psi)\end{aligned}$$

Why this order?



http://reference.wolfram.com/language/ref/RollPitchYawMatrix.html

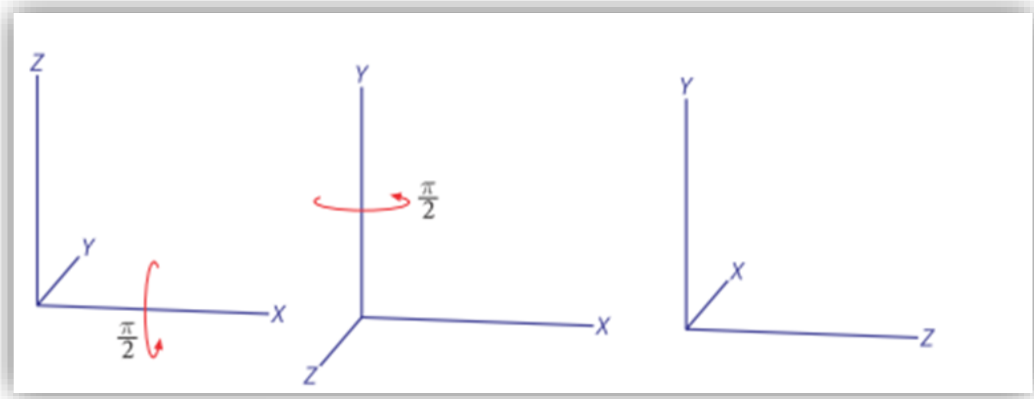
$$\begin{aligned}\mathbf{R}_{RPY}(\psi,\theta,\phi) &= [\mathbf{R}_X(\psi)][\mathbf{R}_X^T(\psi)\mathbf{R}_Y(\theta)\mathbf{R}_X(\psi)] \\ &\quad [\mathbf{R}_X^T(\psi)\mathbf{R}_Y^T(\theta)\mathbf{R}_Z(\phi)\mathbf{R}_Y(\theta)\mathbf{R}_X(\psi)] \\ &= \mathbf{R}_Z(\phi)\mathbf{R}_Y(\theta)\mathbf{R}_X(\psi)\end{aligned}$$

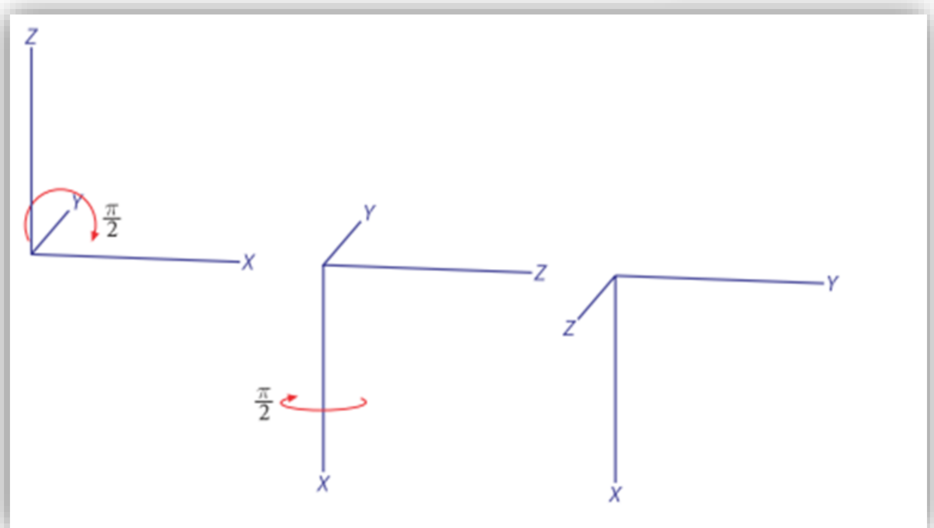
- Right-hand rule!
- Order of rotation matters
  - Rotations are non-commutative!

Example

What is the sequence of rotations for the plots below?

$$R_x\left(\frac{\pi}{2}\right)R_y\left(\frac{\pi}{2}\right) =$$





$$= R_y\left(\frac{\pi}{2}\right)R_x\left(\frac{\pi}{2}\right)$$

Figure 2.12 in RVC

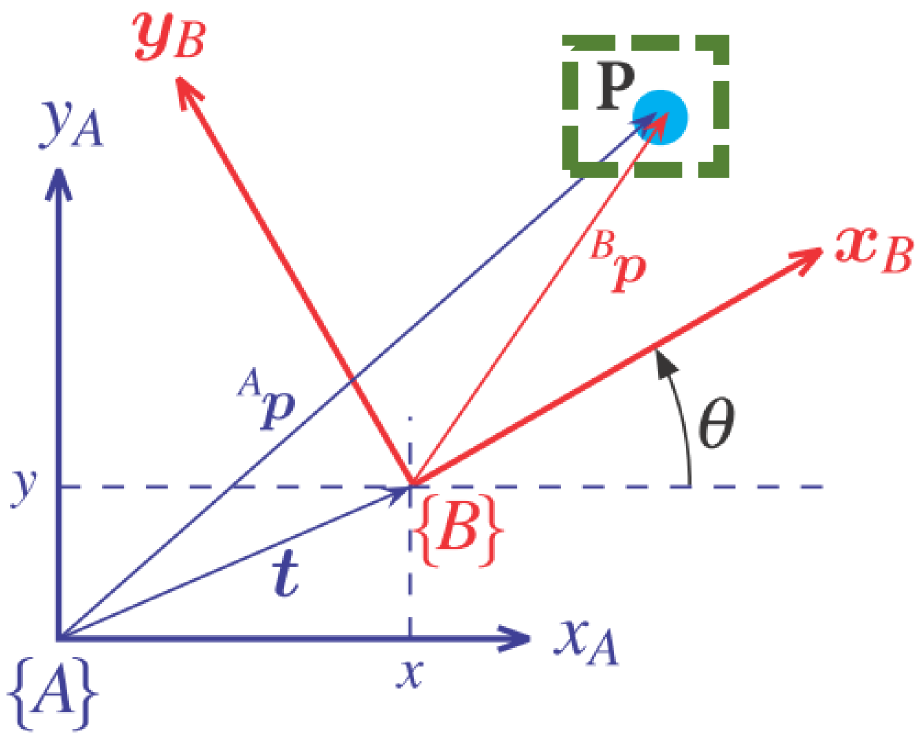
Homogenous Transformations

- Introduced in mathematics:
  - For projections and drawings
  - Artillery, architecture
  - They were top-secret in the 1850s!

https://en.wikipedia.org/wiki/Coincidence_rangefinder#/media/File:11-27-30_-_90mm_Height_finder_at_new_4th_Btry_position.jpg

- In 2D, add a 3rd coordinate, e.g. w
  - A 3D point is then a 3 coordinate vector: $\begin{bmatrix} x \\ y \\ w \end{bmatrix}$
- In 3D, add a 4th coordinate, e.g. $\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$
- w is usually 1! (always 1 in robotics!)

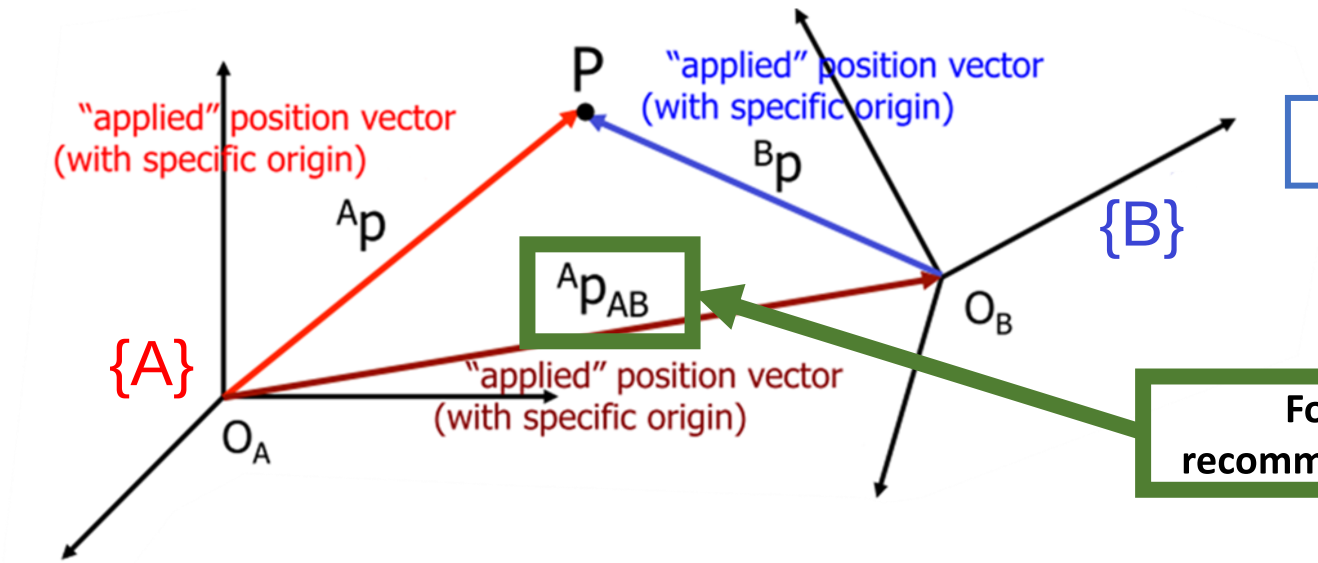
Homogenous Transformations in 2D



$\boxed{{}^{A}\mathbf{T}_{B} = \mathbf{R}(\theta) + t}$ where $t = [x \quad y \quad 1]^T$

$${}^{A}\boldsymbol{p} = {}^{A}\mathbf{T}_{B} \; [0 \quad 0 \quad 1]^T$$

Homogenous Transformations in 3D



$$\boxed{{}^{A}p_{AB} = t}$$

For clarity, it is recommended to use this!

$$
{}^{A}p = \left[\begin{array}{c} {}^{A}p \\ \hline 1 \end{array}\right] = \left[\begin{array}{c:c} {}^{A}R_{B} & {}^{A}p_{AB} \\ \hline 0 \;\; 0 \;\; 0 & 1 \end{array}\right] \left[\begin{array}{c} {}^{B}p \\ \hline 1 \end{array}\right] = {}^{A}T_{B}\, {}^{B}p
$$

linear relationship

vector in homogeneous coordinates

4x4 matrix of homogeneous transformation

Properties of Homogenous Transformations

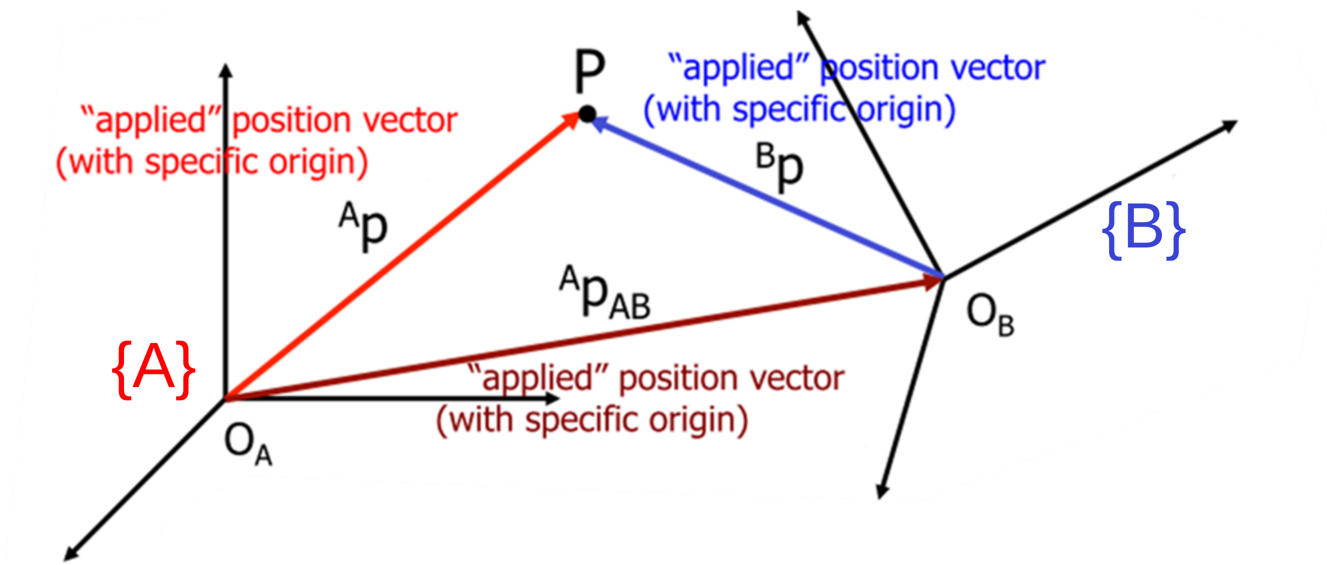
- Describe the relation between reference frames
  - Relative **pose** = position & orientation

- Transform the representation of a position vector from a given frame to another frame
  - The vector starts from the origin of the reference frame

- It is a concatenated operator (rotation and translation) for vectors in the 3D space
- It can be inverted → ${}^{A}T_{B} = \left({}^{B}T_{A}\right)^{-1}$
- It can be composed → ${}^{A}T_{C} = {}^{A}T_{B}\,{}^{B}T_{C}$

  - **But… it doesn't commute, the order matters!**

Inverse of a Homogenous Transform



$$ {}^{A}p = {}^{A}p_{AB} + {}^{A}R_{B}\,{}^{B}p \qquad {}^{B}p = {}^{B}p_{BA} + {}^{B}R_{A}\,{}^{A}p = -\,{}^{A}R_{B}^{T}\,{}^{A}p_{AB} + {}^{A}R_{B}^{T}\,{}^{A}p $$

$$ \left[\begin{array}{ccc:c} \multicolumn{3}{c:}{{}^{A}R_{B}} & {}^{A}p_{AB} \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] \qquad \left[\begin{array}{ccc:c} \multicolumn{3}{c:}{{}^{B}R_{A}} & {}^{B}p_{BA} \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc:c} \multicolumn{3}{c:}{{}^{A}R_{B}^{T}} & -\,{}^{A}R_{B}^{T}\,{}^{A}p_{AB} \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] $$

$$ {}^{A}T_{B} \qquad\qquad {}^{B}T_{A} \qquad\qquad ({}^{A}T_{B})^{-1} $$

Summary Homogenous Transforms

- Main mathematical tool for computing the **forward and inverse kinematics** of robots
- They are used in many application areas (in robotics, computer vision, and beyond)
  - Positioning/orienting a vision camera in terms of its “extrinsic parameters” (more about this in coming lectures and lab 5)
- They are the basis for Affine and Projective geometries
  - Mapping a 3D volume into a 2D display

$$ {}^{A}T_{B} = \left[\begin{array}{ccc|c} & {}^{A}R_{B} & & {}^{A}p_{AB} \\ \hline \alpha_x & \alpha_y & \alpha_z & \sigma \end{array}\right] $$

all zero in robotics

coefficients of perspective deformation

scaling coefficient

always unitary in robotics

Quaternions

- Homogenous transformations are great but…
  - ROS uses quaternions to represent orientation!
- Quaternions…
  - avoid gimbal lock (section 2.2.13 in RVC) watch this! https://www.youtube.com/watch?v=OmCzZ-D8Wdk (not assessed in the exam!)
  - compact and practical representation is the vector and unit quaternion pair, is easy to compound, and singularity free.
  - represents pose using just 7 numbers; $(x,\ y,\ z)$ for position; and

    $(scalar,\ q_x,\ q_y,\ q_z)$ for orientation.
- In ROS, you can use RPY rotations plus translation; and,
  - **let ROS handle quaternions for you!**
  - Some Python libraries don't follow the same convention as ROS, e.g.

    $(\ q_x,\ q_y,\ q_z,\ scalar)$
- <u>Beyond homogenous transformations and quaternions:</u> Geometric algebras → a framework for classical geometries!
  - *Geometric Algebra For Computer Science* http://www.geometricalgebra.net/

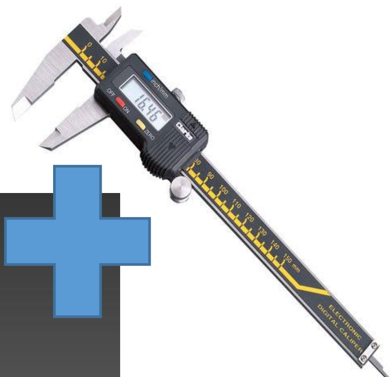
The need for transformations – Example

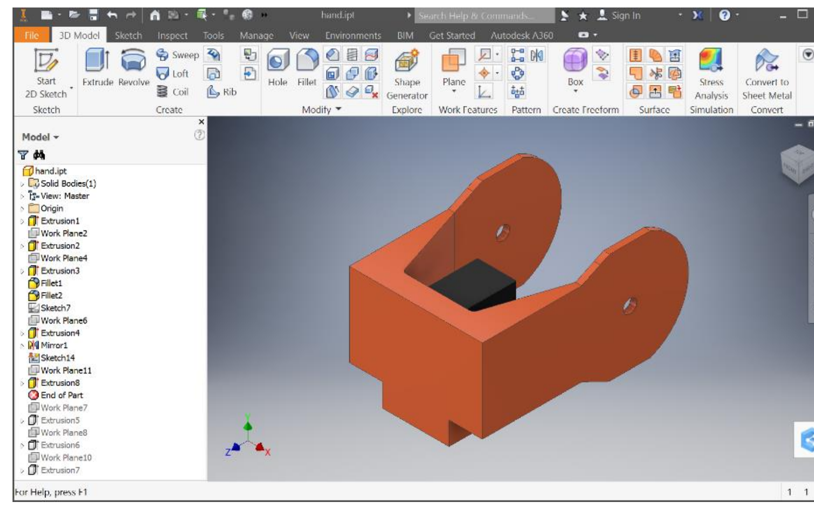
- **Arduino Braccio Arm**
- Low cost robot
- 5 Degrees of Freedom (DoF)
- Versatile
  - Multiple configurations
- Ready to use embedded hardware

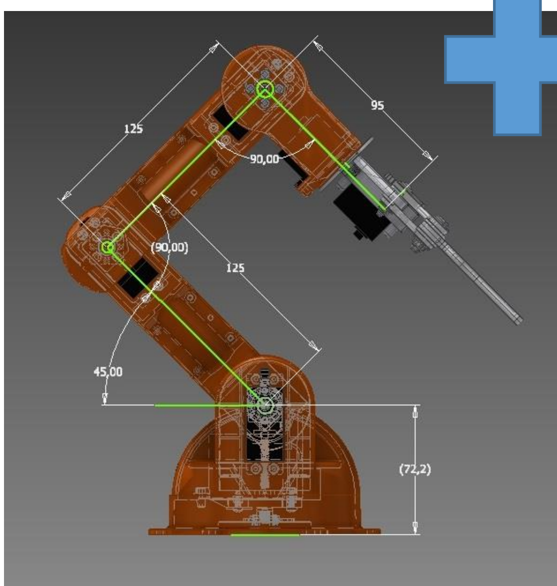
The need for transformations – Example

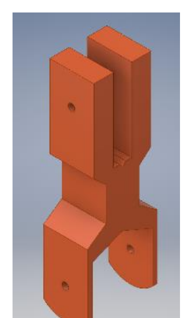
- **First step – Mechanics!**

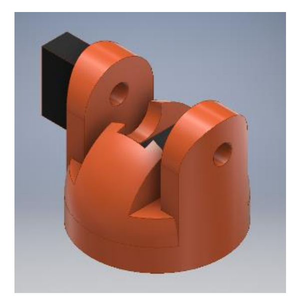
- 3D model of the robot – Autodesk Inventor
- Export 3D model parts to STL

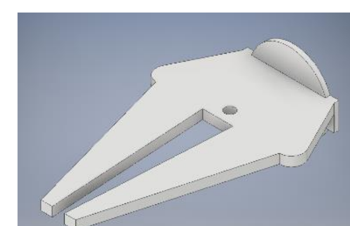












The need for transformations – Example

- **Second step – Forward kinematics**

- URDF and XACRO files
  - Next lecture and lab 2!

..
cfg
launch
meshes
urdf
CMakeLists.txt
package.xml

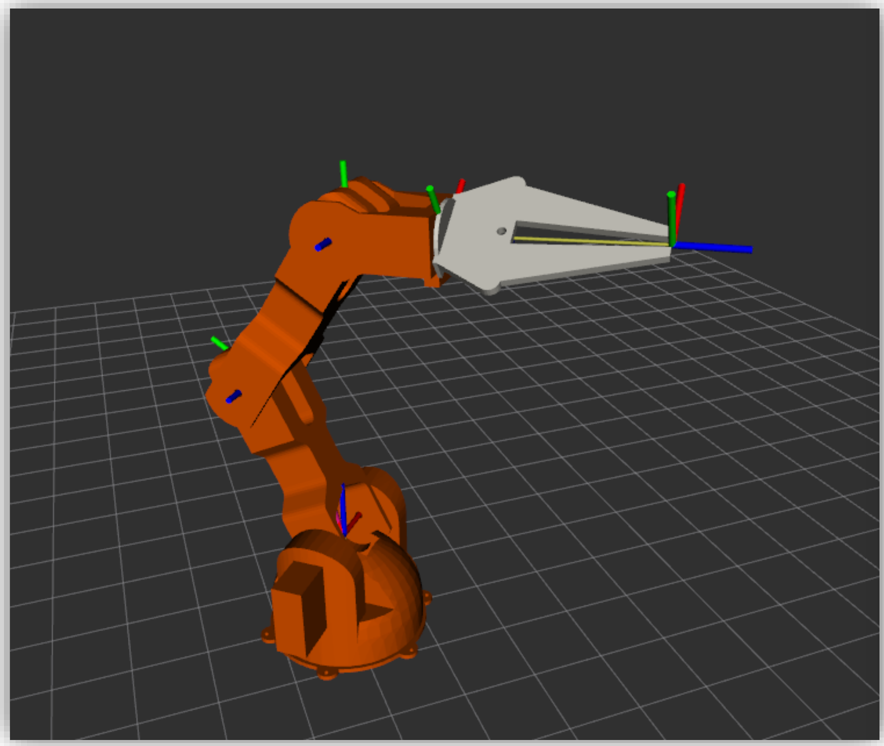
..
braccio.urdf
braccio.xacro
braccio_macro.xacro
cmd.txt
colours.xacro
constants.xacro
materials.xacro

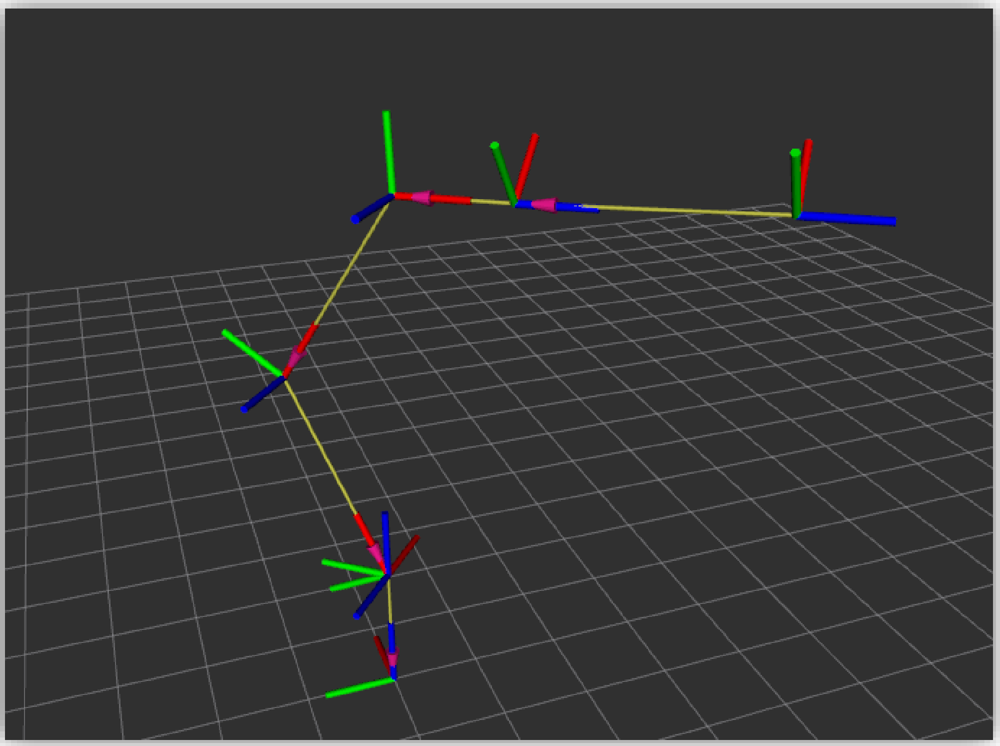
```
<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:include filename="$(find braccio_support)/urdf/materials.xacro"/>
  <xacro:include filename="$(find braccio_support)/urdf/constants.xacro"/>

  <xacro:macro name="braccio_arm" params="prefix">
    <!-- links -->
    <link name="${prefix}_base">
      <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://braccio_support/meshes/visual/base_plate.stl"/>
        </geometry>
        <xacro:material_body />
      </visual>
      <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://braccio_support/meshes/collision/base_plate.stl"/>
        </geometry>
      </collision>
    </link>

    <link name="${prefix}_link_1">
      <visual>
        <origin xyz="0 0 ${-7.22+0.01}" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://braccio_support/meshes/visual/base_m1.stl"/>
        </geometry>
        <xacro:material_body />
      </visual>
      <collision>
        <origin xyz="0 0 ${-7.22+0.01}" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://braccio_support/meshes/collision/base_m1.stl"/>
        </geometry>
      </collision>
    </link>

    <link name="${prefix}_link_2">
```

159 lines final URDF file

Practical Example

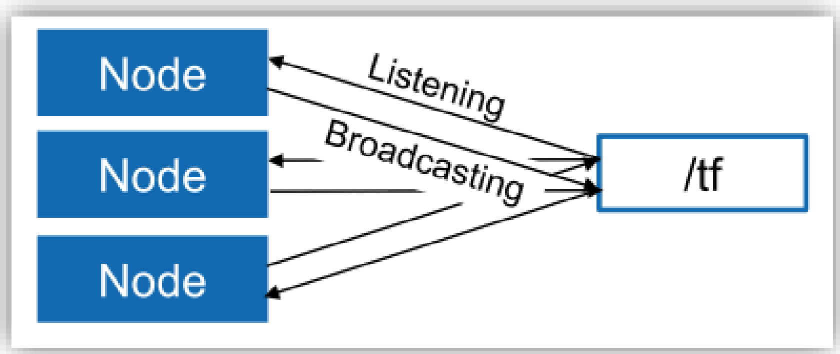
- **Second step – Forward kinematics**





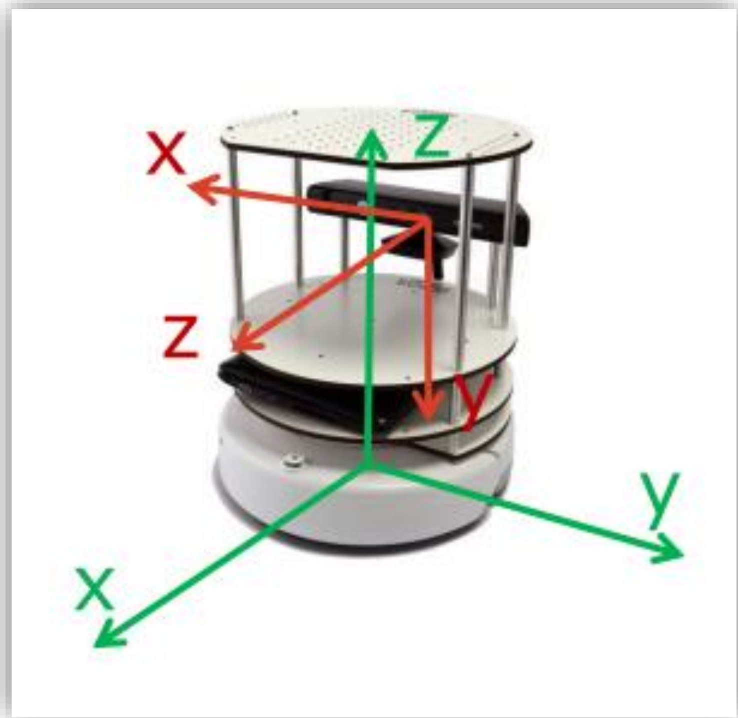
Transformations in ROS

- Named: TF (Transformation System)
  - http://wiki.ros.org/tf2
- API and tool for keeping track of coordinate frames over time
- Maintains relationships between coordinate frames in a tree structure buffered in time
- Lets the user retrieve transformations in order to transform:
  - Points
  - Vectors, etc
- Implemented as a publisher / subscriber model on the topics
  - /tf and /tf_static



TF: Conventions

- Orientation of the robot or object axes:
  - Right-handed coordinate frame:
    - x: forward
    - y: left
    - z: up
- Orientation of camera axes
  - Left-handed coordinate frame:
    - x: right
    - Y: down
    - z: forward
- Rotation representations
  - **Quaternions**
  - Rotation matrix
  - Euler angles

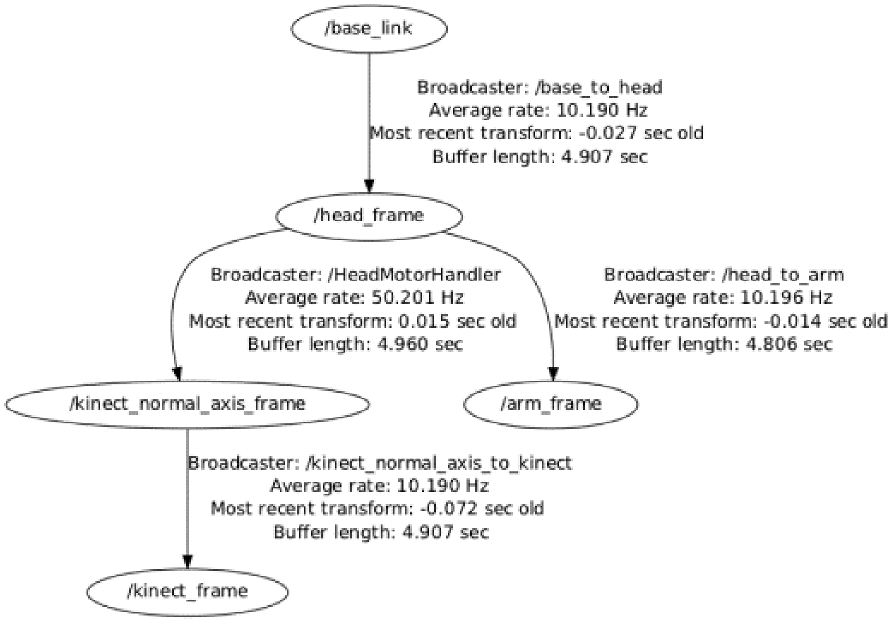


TF: Transform Tree

- TF listeners use a buffer to listen to all broadcasted transforms
- Query transforms from the transform tree between arbitrary frames

tf2_msgs/TFMessage.msg

```
geometry_msgs/TransformStamped[] transforms
  std_msgs/Header header
    uint32 seqtime stamp
    string frame_id
  string child_frame_id
  geometry_msgs/Transform transform
    geometry_msgs/Vector3 translation
    geometry_msgs/Quaternion rotation
```



Transformations in ROS

Command line

Print information about the current tranform tree

```
> rosrun tf tf_monitor
```

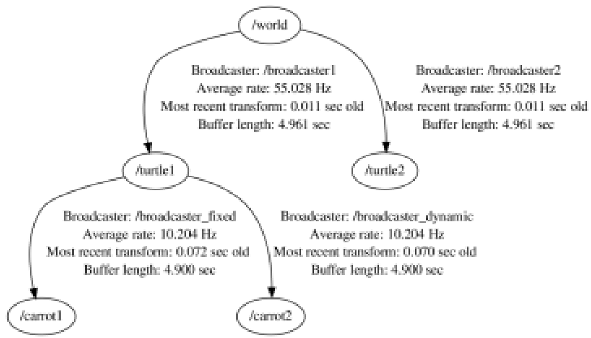
Print information about the transform between two frames

```
> rosrun tf tf_echo
    source_frame target_frame
```

View Frames

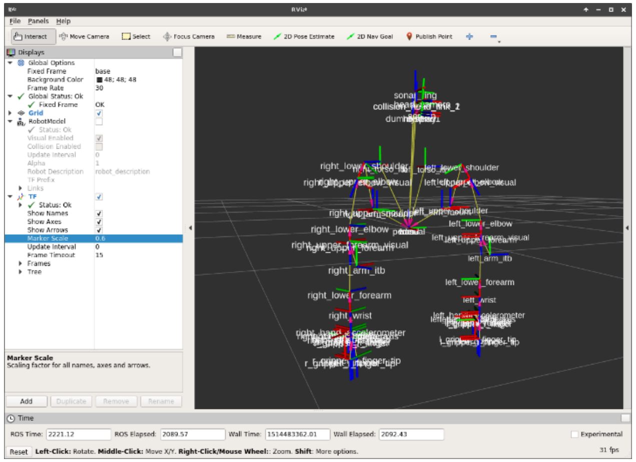
Creates a visual graph (PDF) of the transform tree

```
> rosrun tf view_frames
```



RViz

3D visualization of the transforms



Next Lecture!

- **Forward and Inverse kinematics**