DTU

Software for Autonomous Systems

SFfAS-31391:

Learning ROS Transforms (TF), Robot Visualization (RVIZ) and Simulation (Gazebo)

Lecturer, Course Coordinator: Evangelos Boukas—PhD

Outline

- Transformations
- TF Package
- Universal Robot Description Format
- Simulating Physical Robots in ROS

Outline

- Transformations
- TF Package
- Universal Robot Description Format
- Simulating Physical Robots in ROS
- And ALL of that Hands on!!!

Transformations
Example: Mobile robot in a factory

Transformations
High-level approach

- Attach a separate coordinate system, or frame, for each rigid body
- Relate these frames to each other

- Why? It makes everything so much easier!

- If we have all the coordinate systems, and their relations, we can easily **transform** a **pose** in one **frame** to **any other frame**

Transformations
Example with coordinate systems

Transformations
Example with coordinate systems

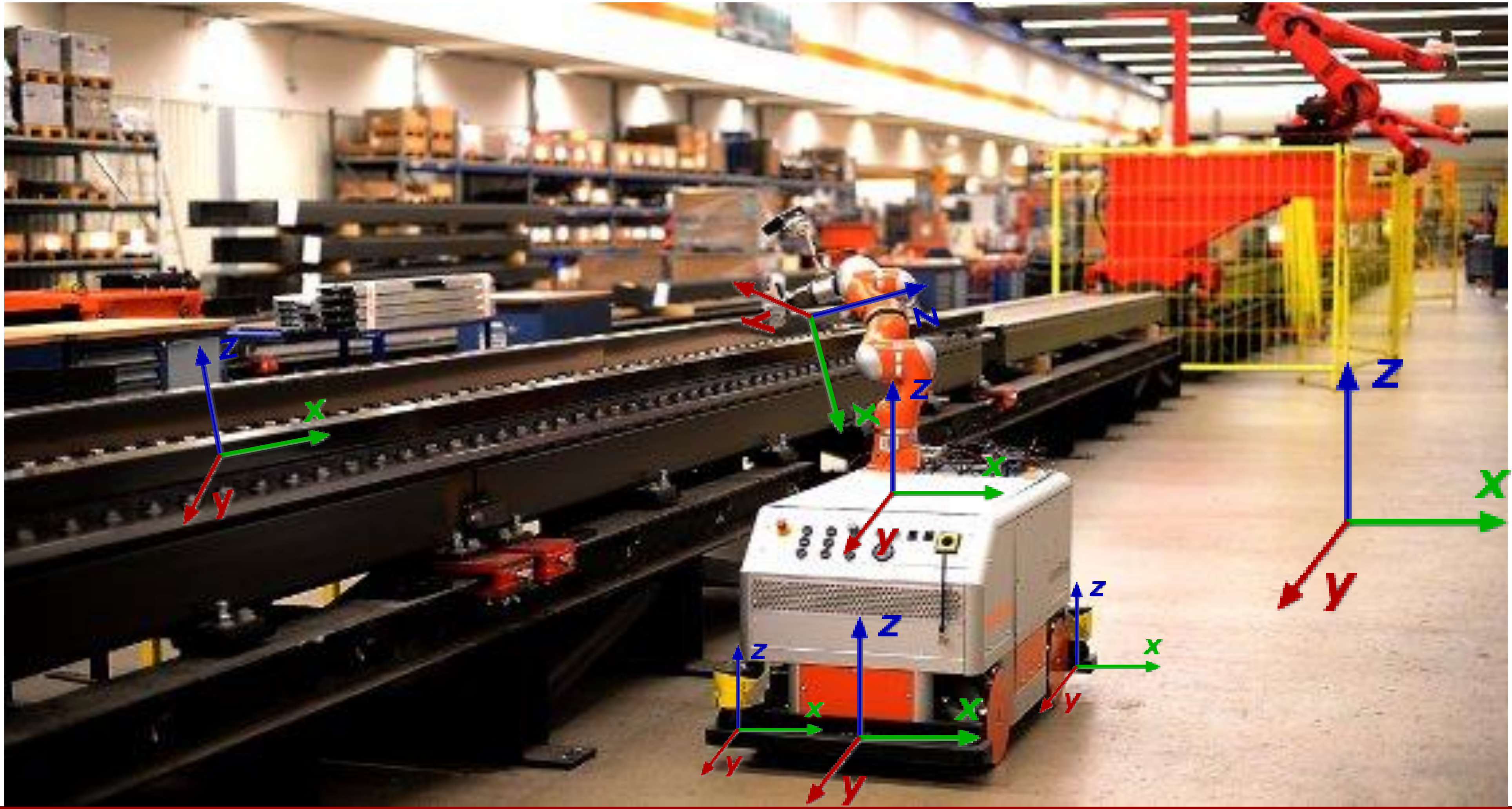
Transformations
Example with coordinate systems

Transformations
Example with coordinate systems

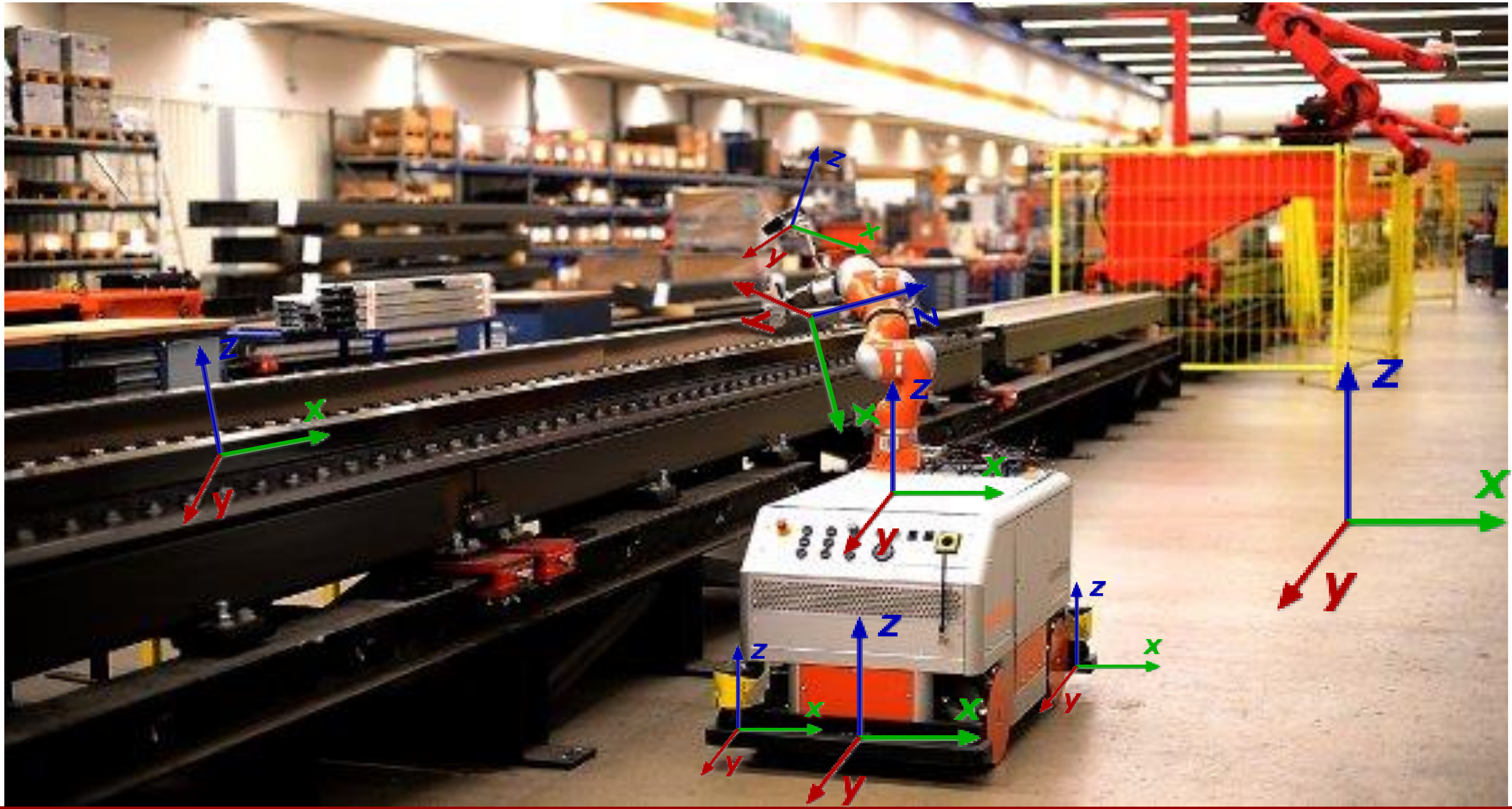
Transformations
Example with coordinate systems

Transformations
Example with coordinate systems

Transformations
Example with coordinate systems

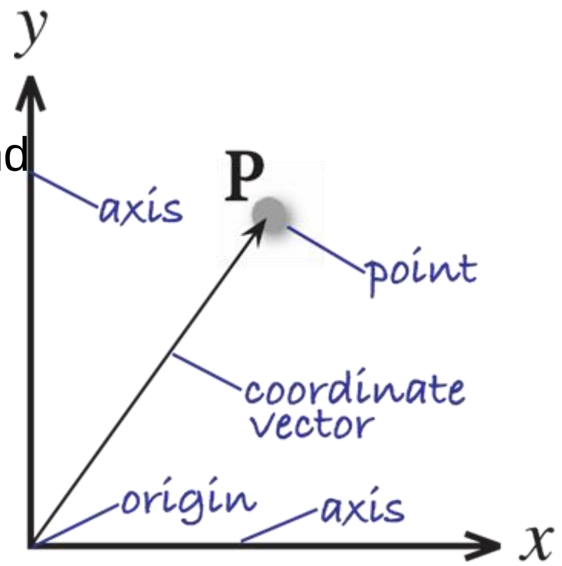


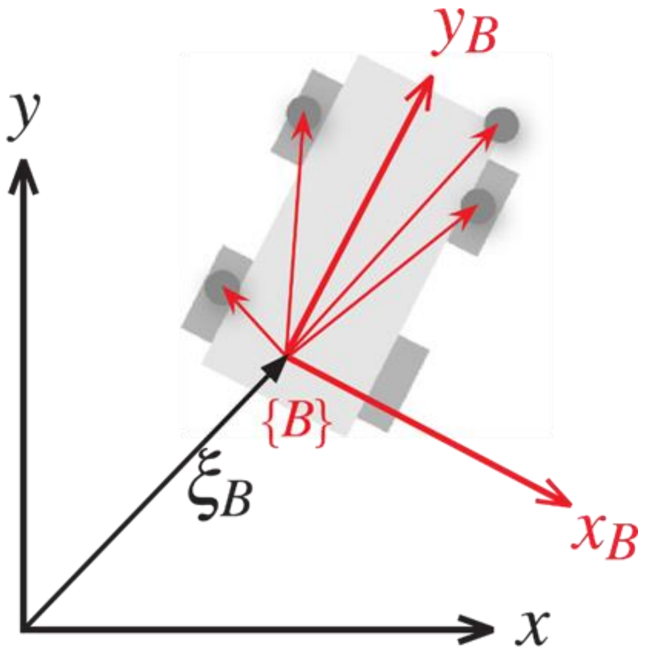
Transformations
Example with coordinate systems



Transformations: Graphical overview

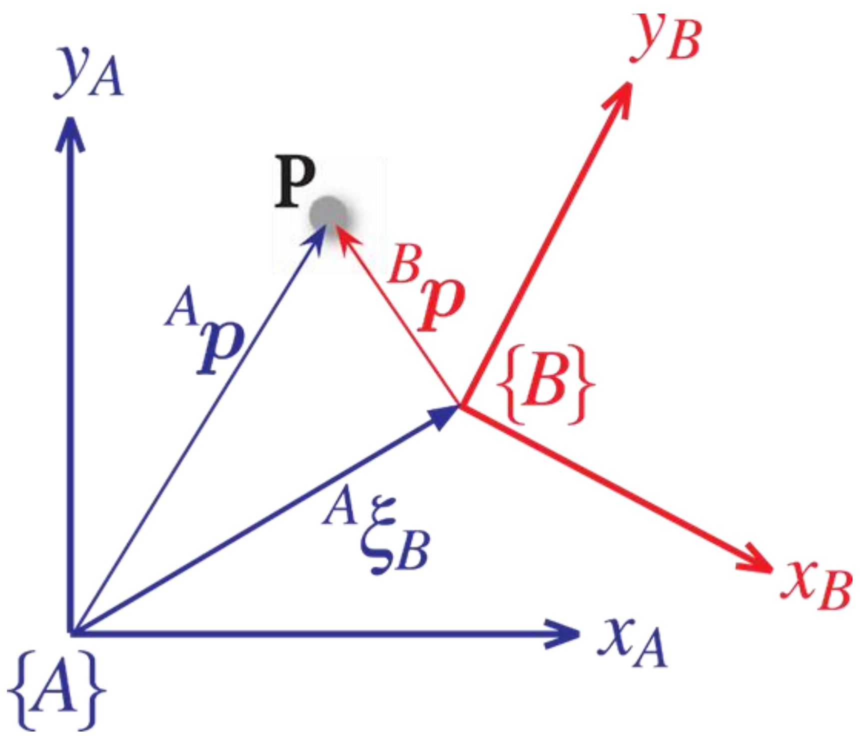
- **Point** is defined by a vector
  - The **frame** matters!
- **Frame** is defined by a changed in **pose ξ**
  - Describes both translation and rotation





Transformations: Graphical overview

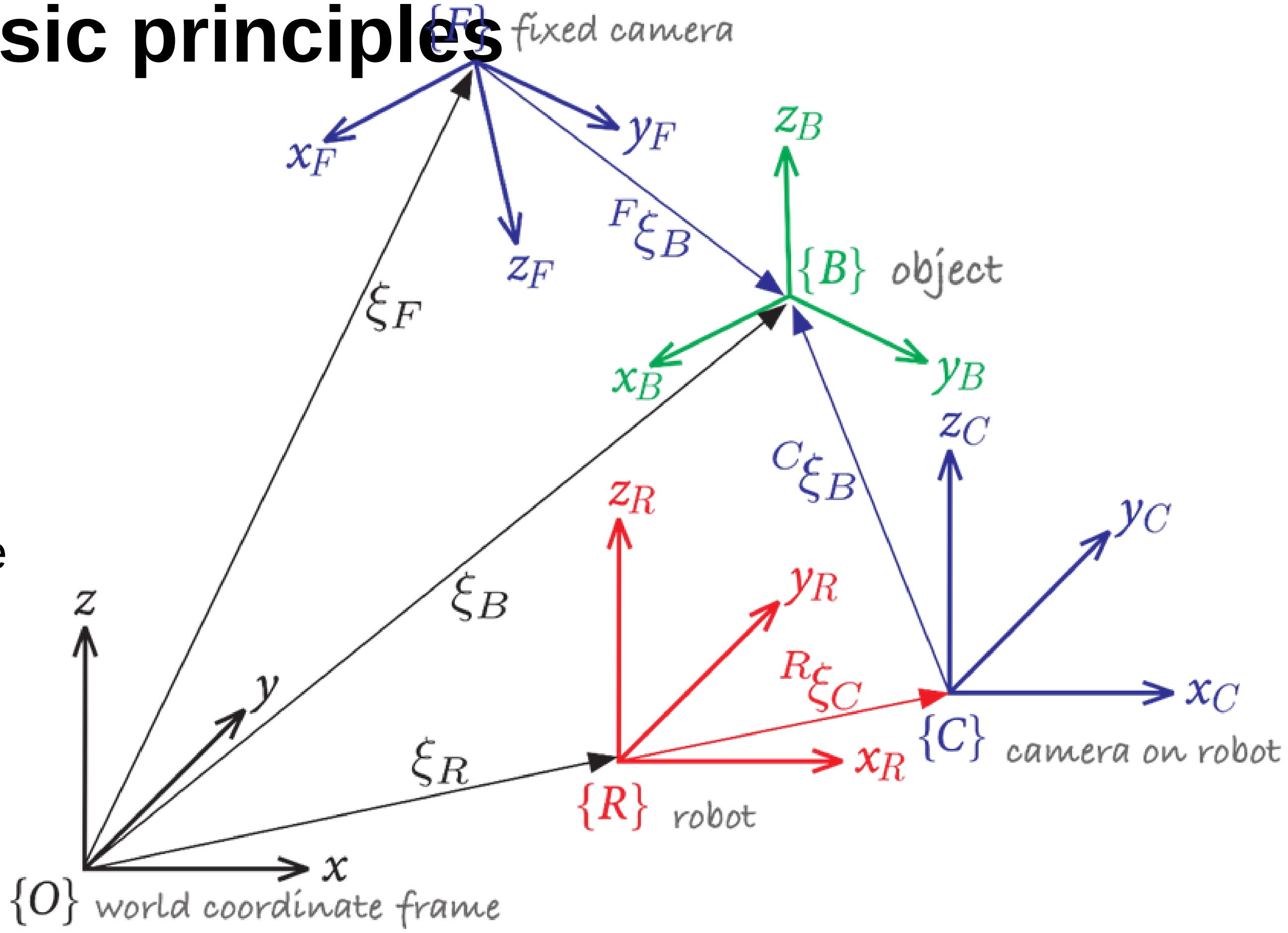
- We use indices to indicate the relevant frames
  - For points, in which frame we have defined the point
  - For transformations, the **pose** of a frame with respect to another
    - Transform: “From A to B”
    - Pose: “B relative to A”



Transformations: Basic principles

- The strategy is to follow the arrows!
- Pose transformations can be applied sequentially ("compounded" in the book)

- They can even be reversed, by taking the inverse $\xi_{\rho} = \xi_{f} \oplus {}^{F}\xi_{f} = \xi_{R} \oplus {}^{R}\xi_{C} \oplus {}^{C}\xi_{B}$



Transformations in a 3D

- We can make rotation matrices from the different rotation representations (See later…)
- We we need to be careful when defining the rotation matrix):

$$\begin{pmatrix} {}^{A}P_x \\ {}^{A}P_y \\ {}^{A}P_z \\ 1 \end{pmatrix} = \begin{pmatrix} {}^{A}R_B & {}^{A}t_B \\ O_{1x3} & 1 \end{pmatrix} * \begin{pmatrix} {}^{B}P_x \\ {}^{B}P_y \\ {}^{B}P_z \\ 1 \end{pmatrix}$$

Transformations: Rotation around one axis

- Rotation around X axis

$$R^x_\phi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\,\phi & sin\,\phi & 0 \\ 0 & -sin\,\phi & cos\,\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotation around X axis

$$R^y_\theta = \begin{bmatrix} cos\,\theta & 0 & -sin\,\theta & 0 \\ 0 & 1 & 0 & 0 \\ sin\,\theta & 0 & cos\,\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotation around X axis

$$R^z_\psi = \begin{bmatrix} cos & sin\,\psi & 0 & 0 \\ -sin\,\psi & cos\,\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Translation X,Y,Z axis

$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformations: Euler

- 12 possible representations

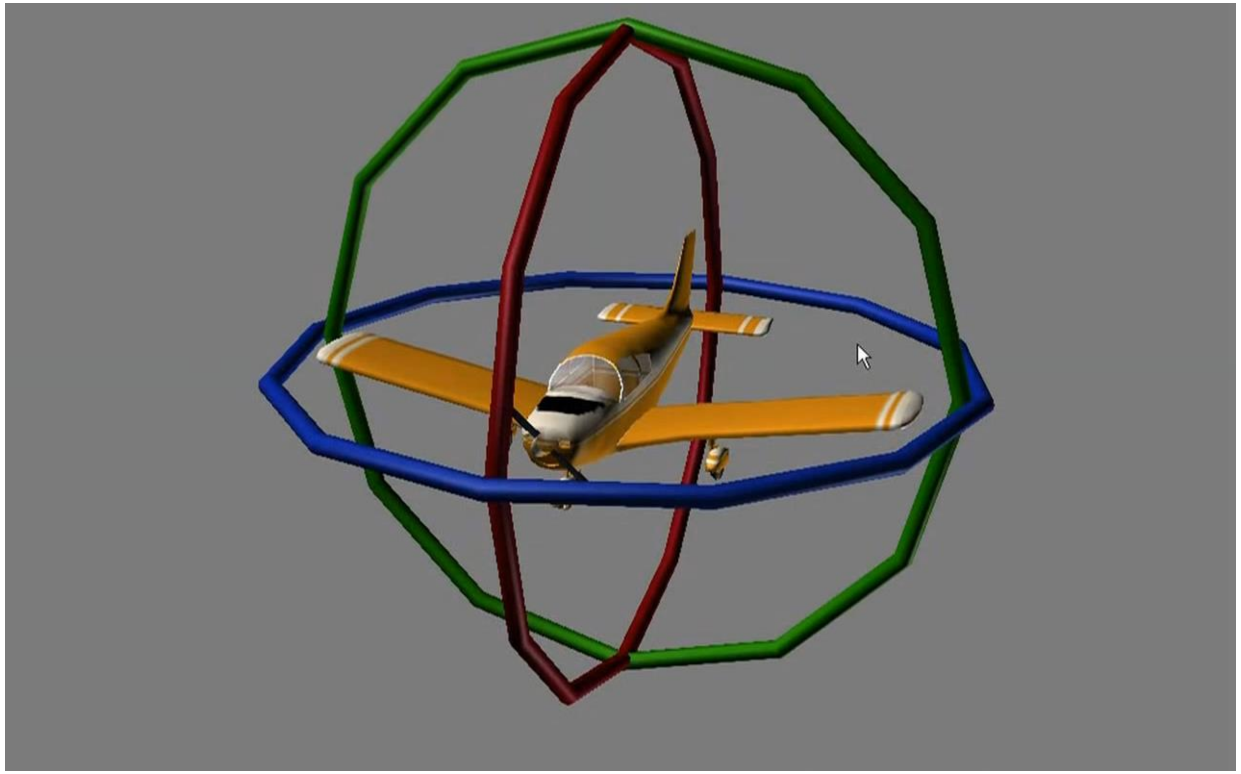
xyz	yzx	zxy
xzy	yxz	zyx
xyx	yzy	zxz
xzx	yxy	zyz

- The most popular is the roll, pitch, yaw one:

$$
\begin{aligned}
R &= R^x_\phi R^y_\theta R^z_\psi \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,\phi & sin\,\phi \\ 0 & -sin\,\phi & cos\,\phi \end{bmatrix} \begin{bmatrix} cos\,\theta & 0 & -sin\,\theta \\ 0 & 1 & 0 \\ sin\,\theta & 0 & cos\,\theta \end{bmatrix} \begin{bmatrix} cos\,\psi & sin\,\psi & 0 \\ -sin\,\psi & cos\,\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

Transformations: Euler

- A major problem is the Gimbal lock:



Transformations: Euler

- A major problem is the Gimbal lock:

```
     [ 1,      0,       0]          [ 0.0000        0    1.0000]        [ cos(g), -sin(g), 0]
Rx = [ 0, cos(a), -sin(a)]     Ry = [      0   1.0000         0]   Rz = [ sin(g),  cos(g), 0]
     [ 0, sin(a),  cos(a)]          [-1.0000        0    0.0000]        [      0,       0, 1]
```

```
           [                         cos(b)*cos(g),                        -cos(b)*sin(g),         sin(b)]
R=Rx*Ry*Rz = [ cos(a)*sin(g) + cos(g)*sin(a)*sin(b), cos(a)*cos(g) - sin(a)*sin(b)*sin(g), -cos(b)*sin(a)]
           [ sin(a)*sin(g) - cos(a)*cos(g)*sin(b), cos(g)*sin(a) + cos(a)*sin(b)*sin(g),  cos(a)*cos(b)]
```

```
b =  pi/2
```

```
     [ 0.0000        0    1.0000]
Ry = [      0   1.0000         0]
     [-1.0000        0    0.0000]
```

```
             [                             0,                             0, 1]
R=Rx*Ry*Rz = [ cos(a)*sin(g) + cos(g)*sin(a), cos(a)*cos(g) - sin(a)*sin(g), 0]
             [ sin(a)*sin(g) - cos(a)*cos(g), cos(a)*sin(g) + cos(g)*sin(a), 0]
```

```
              [           0,          0, 1]
simplify(R) = [  sin(a + g), cos(a + g), 0]
              [ -cos(a + g), sin(a + g), 0]
```

Transformations: Quaternions

- Provide Orientations
- Invented by Hamilton in 1843

Transformations: Quaternions

- Provide Orientations
- Invented by Hamilton in 1843
- The governing rule is:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{i}\,\mathbf{j}\,\mathbf{k} = -1$$

Transformations: Quaternions

- Provide Orientations
- Invented by Hamilton in 1843
- The governing rule is:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{i}\,\mathbf{j}\,\mathbf{k} = -1$$

- A quaternion is defined as:

$$q = q_0 + \mathbf{q} \quad = \quad q_0 + \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3$$

,where $q0$ is the scalar and q is called the vector part.

i,j,z is the common orthonormal bases of R^3

Transformations: Quaternions

- Provide Orientations
- Invented by Hamilton in 1843
- The governing rule is:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{i}\,\mathbf{j}\,\mathbf{k} = -1$$

- A quaternion is defined as:

$$q = q_0 + \mathbf{q} \quad = \quad q_0 + \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3$$

,where $q0$ is the scalar and q is called the vector part.
i,j,z is the common orthonormal bases of R^3

- Quaternions solve all the problems with euler angles

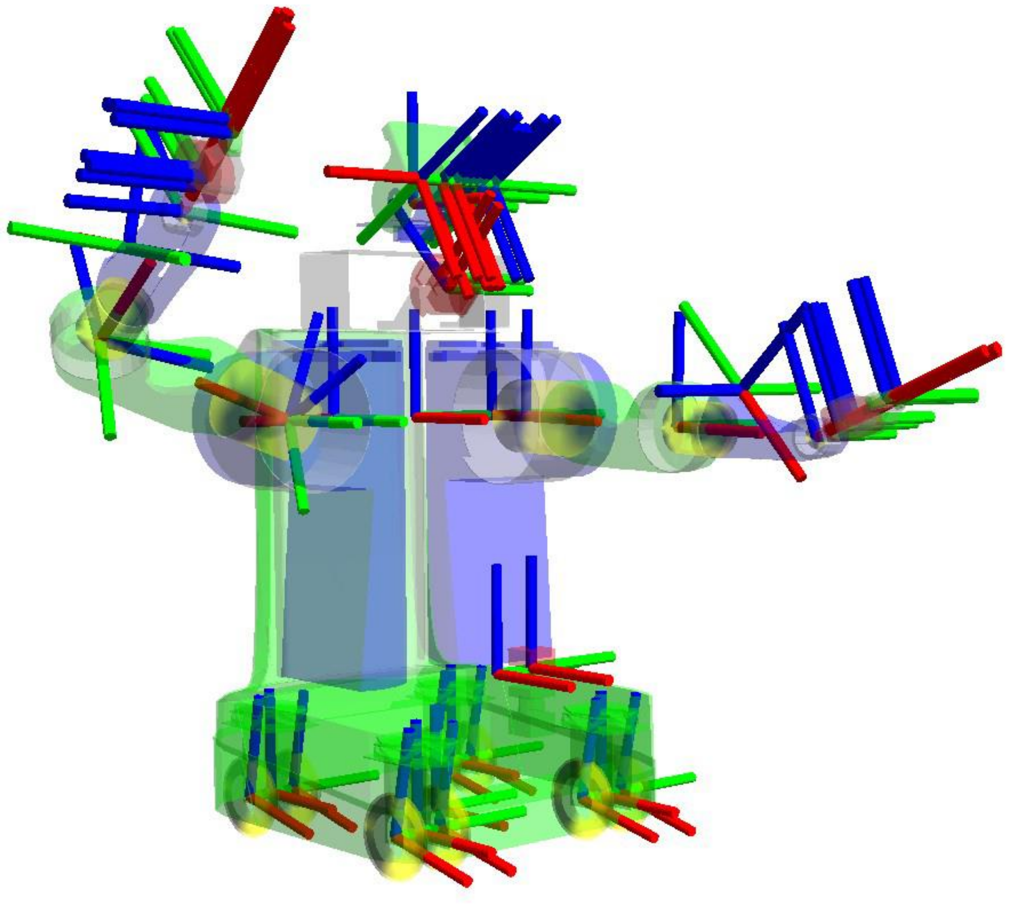
Transformations (Rotations): Overall

Task/Property	Matrix	Euler Angles	Quaternion
Rotating points between coordinate spaces (object and internal)	Possible	Impossible (must convert to matrix)	Impossible (must convert to matrix)
Concatenation or incremental rotation	Possible but usually slower than quaternion form	Impossible	Possible, and usually faster than matrix form
Interpolation	Basically impossible	Possible, but aliasing causes Gimbal lock and other problems	Provides smooth interpolation
Human interpretation	Difficult	Easy	Difficult
Storing in memory	Nine numbers	Three numbers	Four numbers
Representation is unique for a given orientation	Yes	No - an infinite number of Euler angle triples alias to the same orientation	Exactly two distinct representations for any orientation
Possible to become invalid	Can be invalid	Any three numbers form a valid orientation	Can be invalid

Outline

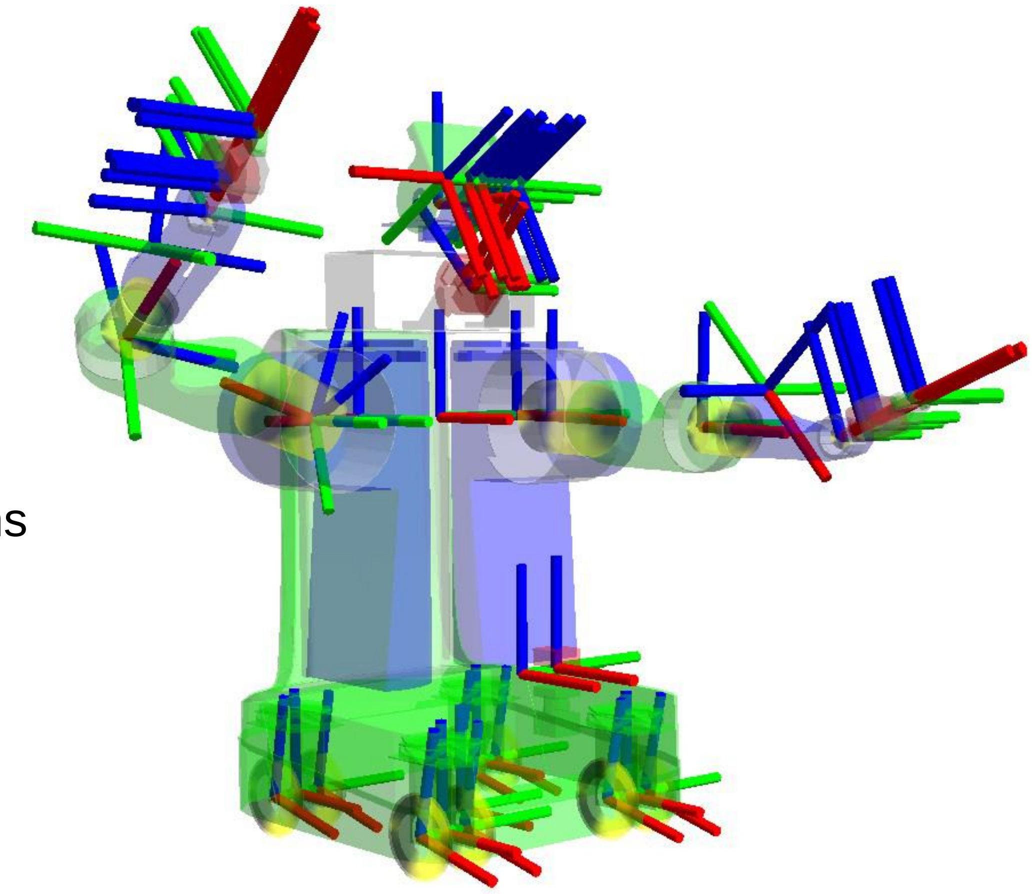
- Transformations
- <u>TF Package</u>
- Universal Robot Description Format
- Simulating Physical Robots in ROS
- And ALL of that Hands on!!!

tf Package



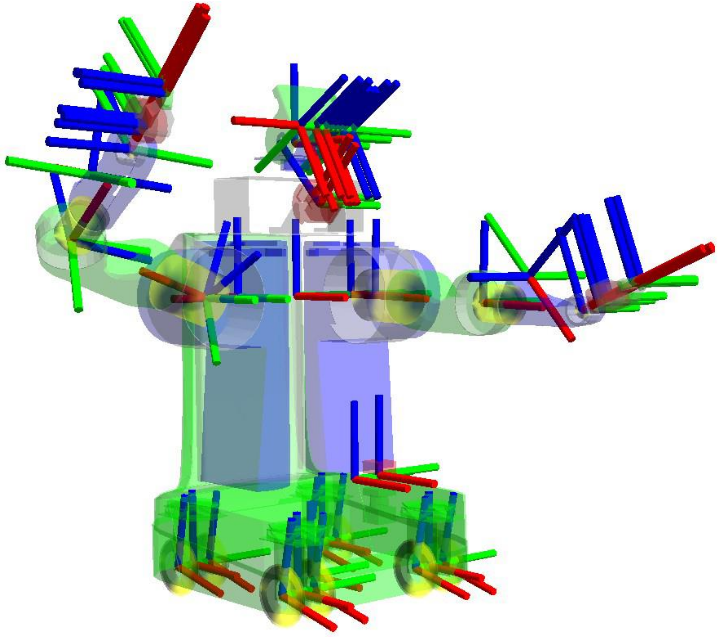
tf Package

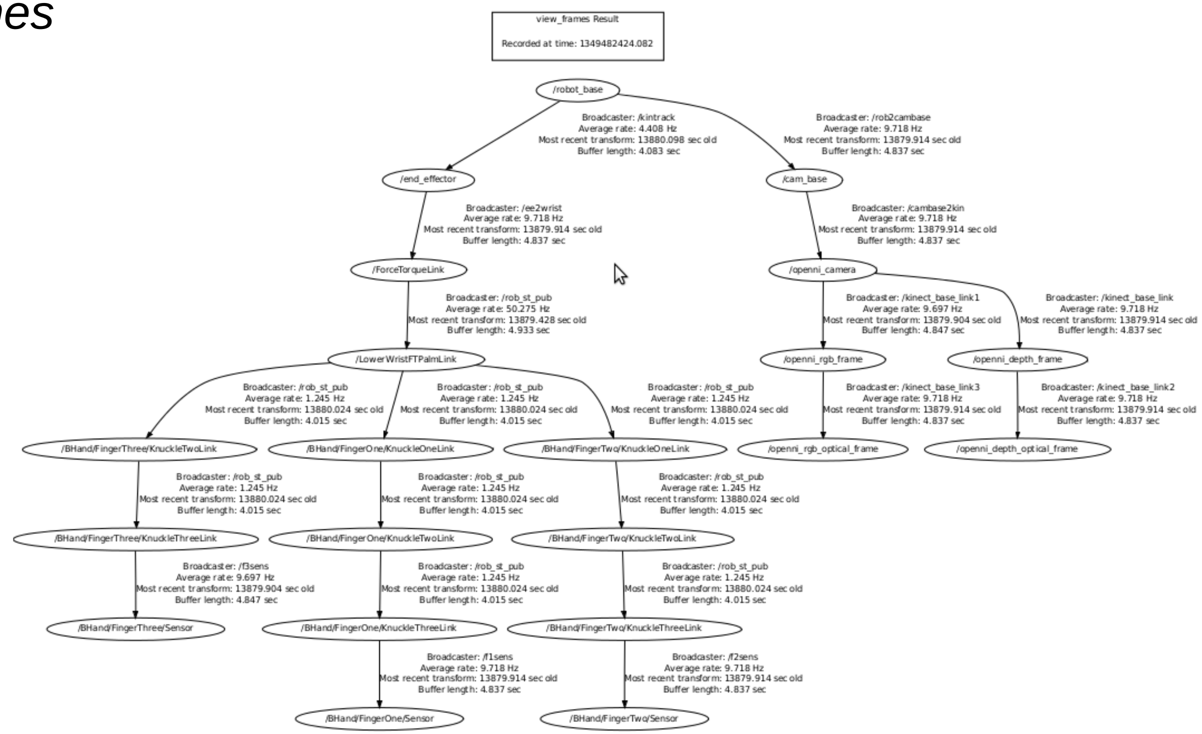
- The *tf* package allows the tracking over time of coordinate systems tree(s)
- Allows the easily creation of new frames (static or dynamic)
- Eases the process of transforming points, vectors, etc.
- Distributed system – no centralized storage
- Caches the past information on the transforms



The *tf* coordinate frame tree

- A tree of the current coordinate frame can be generated using the command: *rosrun tf view_frames*
- Outputs a pdf of current tree





tf Package | Terminal commands

- rosrun tf tf_echo
- rosrun tf tf_monitor
- rosrun tf static_transform_publisher:
  - Usage: rosrun tf static_transform_publisher x y z yaw pitch roll frame_id child_frame_id period(milliseconds)

    OR
  - Usage: rosrun tf static_transform_publisher x y z qx qy qz qw frame_id child_frame_id period(milliseconds)

tf Package | Python code

- Transform Broadcasting:

```
br = tf.TransformBroadcaster()
br.sendTransform(x,y,z,rot,Time,” frame1” , “ frame2” )
```

- Listening a transform:

```
listener = tf.TransformListener()
(trans,rot)=listener.lookupTransform('/frame1','/frame2’, rospy.Time(0))
```

tf Package | time

- Check whether the transform is up..
- Get a transform in the past

```
try:
    now = rospy.Time.now()
    past = now - rospy.Duration(5.0)
    listener.waitForTransformFull("/frame2", now, “/frame1", past, "/transform", rospy.Duration(1.0))
    (trans, rot) = listener.lookupTransformFull("/ frame2", now, "/ frame1", past, "/transform")
```

tf Package

- Let me show you with some hands on…

You’ll do something similar during Lab time

Outline

- Transformations
- TF Package
- Universal Robot Description Format
- Simulating Physical Robots in ROS
- And ALL of that Hands on!!!

URDF

Robot Modeling

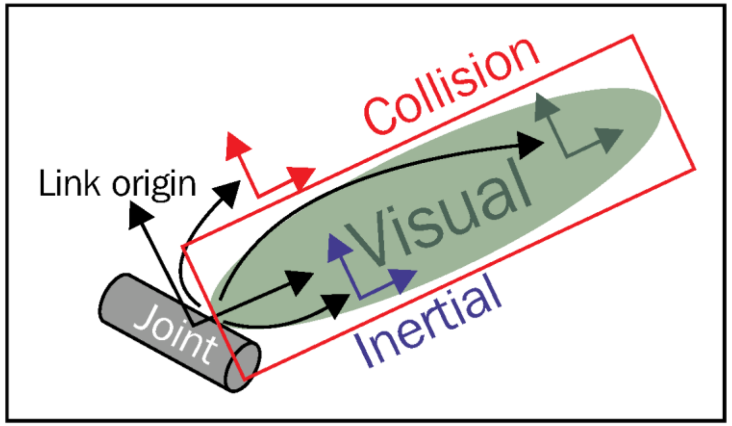
- Mechanical design of Robot parts in CAD
  - AutoCAD, Blender
- Virtual Robot Model
  - Universal Robot Description Format
    - XML

URDF

1/4 main components (tags) in URDF:

- Links:
  - Represents a link of a robot and includes the properties:
    - Size, Shape, Color or maybe include the 3D Mesh
    - Inertial Matrix, Collision info
  - The syntax is as follows:

```
<link name="<name of the link>">
<inertial>...........</inertial>
  <visual> ............</visual>
  <collision>..........</collision>
</link>
```



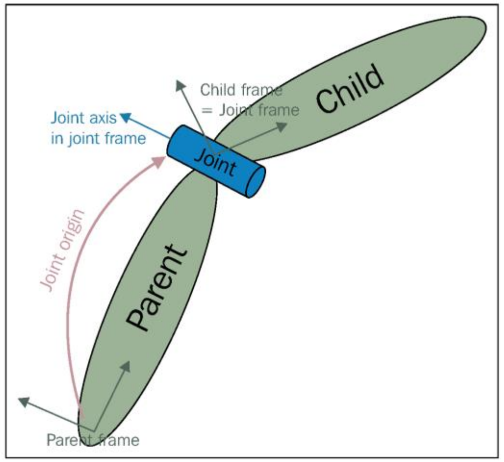
URDF

2/4 main components (tags) in URDF:

- Joints:
  - Represents a joint of a robot and includes the properties:
    - Kinematics, Dynamics, limits of the joints
    - Different type: “revolute, continuous, prismatic, fixed, floating, planar”
  - The syntax is as follows:

```
<joint name="<name of the joint>">
  <parent link="link1"/>
  <child link="link2"/>

  <calibration .... />
  <dynamics damping ..../>
  <limit effort .... />
</joint>
```



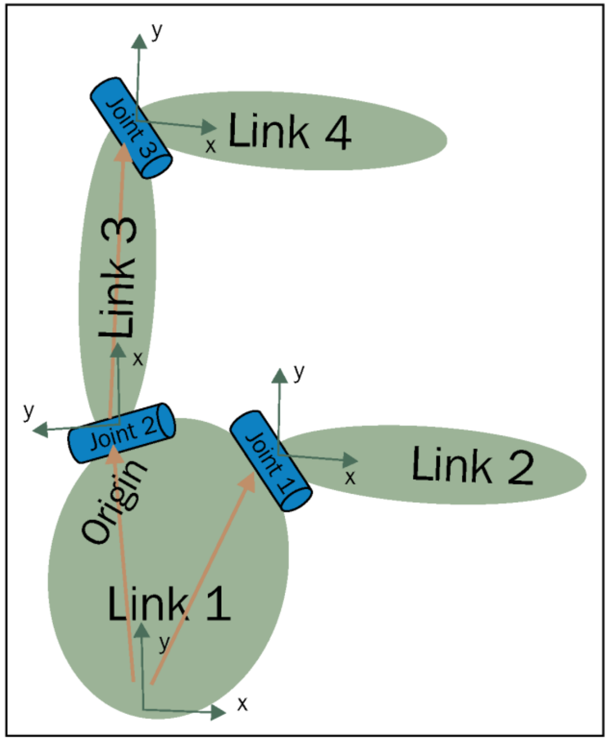
URDF

3/4 main components (tags) in URDF:

- Robot:
  - This includes the whole model (and other tags):
    - Name, links, joints
  - The syntax is as follows:

```
<robot name="<name of the robot>"
  <link>  ..... </link>
  <link> ...... </link>

  <joint> ....... </joint>
  <joint> ........</joint>
</robot>
```



URDF

4/4 main components (tags) in URDF:

- Gazebo:
  - This includes the simulation specific parameters:
    - Gazebo plugins, Gazebo materials, etc..
  - The syntax is as follows:

```
<gazebo reference="link_1">
  <material>Gazebo/Black</material>
</gazebo>
```

URDF main functions

- Check URDF:

  check_urdf pan_tilt.urdf

- In launch File:
  - Loading the description and main parameters:

```
<arg name="model" />
<param name="robot_description" textfile="urdf/pan_tilt.urdf" />
<param name="use_gui" value="true"/>
```

URDF main functions

- Check URDF:

  check_urdf pan_tilt.urdf

- In launch File:
  - Loading the description and main parameters:

```
<arg name="model" />
<param name="robot_description" textfile="urdf/pan_tilt.urdf" />
<param name="use_gui" value="true"/>
```

  - Defining the joint states "**joint_state_publisher**":

```
<node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />
```

URDF main functions

- Check URDF:

  check_urdf pan_tilt.urdf

- In launch File:
  - Loading the description and main parameters:

```
<arg name="model" />
<param name="robot_description" textfile="urdf/pan_tilt.urdf" />
<param name="use_gui" value="true"/>
```

  - Defining the joint states “**joint_state_publisher**”:

```
<node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />
```

  - Loading the TF of the robot “**robot_state_publisher”:**

```
<node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher" />
```

URDF

- Let me show you with some hands on…

You'll do something similar during Lab time

Outline

- Transformations
- TF Package
- Universal Robot Description Format
- Simulating Physical Robots in ROS
- And ALL of that Hands on!!!

Gazebo Implementation

Robot Modelling is cool and all but…

…robots are much more sophisticated than this!

Gazebo Implementation

Robot Modelling is cool and all but…

…robots are much more sophisticated than this!

- Where is the mass?
- Where is the actuation power?
- Where is the inertia?
- Where is the collision?

Gazebo Implementation

Gazebo is able to provide all this and more using a Physics engine!

However, we need to provide this information:

- Collision
- Inertia
- Transmission:

```xml
<transmission name="tran0">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="hip">
    <hardwareInterface>PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="motor0">
    <hardwareInterface>PositionJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

- Control plugin:

```xml
<gazebo>
  <plugin name="control" filename="libgazebo_ros_control.so"/>
</gazebo>
```

Gazebo

- Let me show you with some hands on…

You'll do something similar during Lab time

Robot Planning

Ok now we're more realistic…

… however, still autonomous robots are more complex than this

Robot Planning

Ok now we’re more realistic…

… however, still autonomous robots are more complex than this

- The robots have to autonomously find their way through..

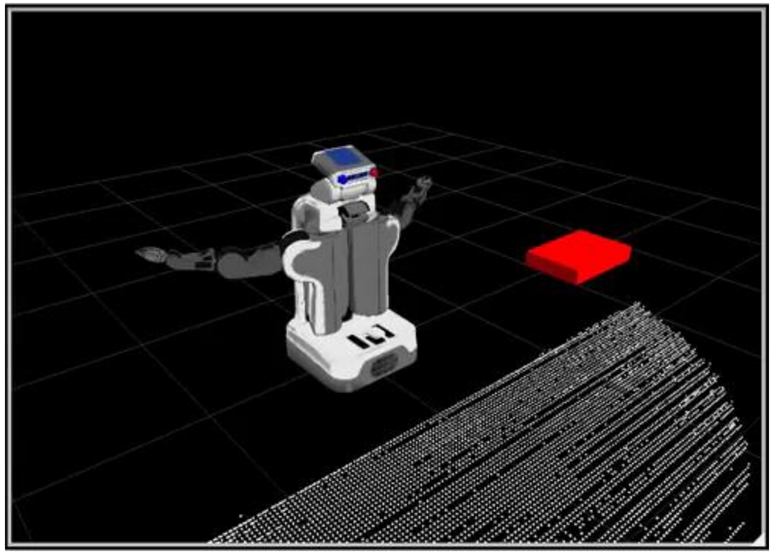
Robot Planning

Ok now we're more realistic…

… however, still autonomous robots are more complex than this

- The robots have to autonomously find their way through..
  - Robotic Manipulators need:
    - To solve the inverse kinematics and dynamics problems
    - Find the correct trajectories to avoid obstacles
    - …

Robot Planning

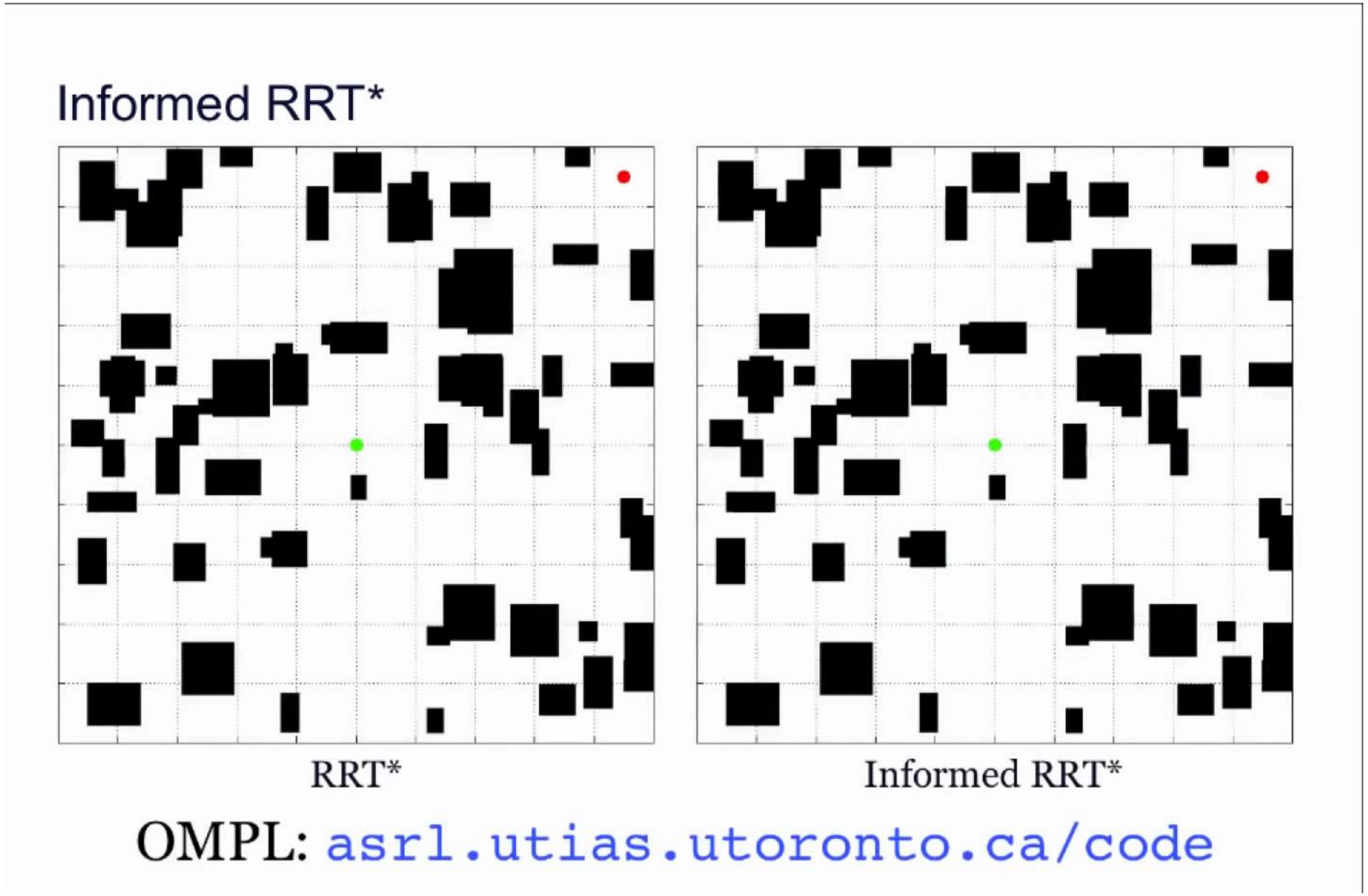


Ok now we’re more realistic…

… however, still autonomous robots are more complex than this

- The robots have to autonomously find their way through..
  - Robotic Manipulators need:
    - To solve the inverse kinematics and dynamics problems
    - Find the correct trajectories to avoid obstacles
    - …
  - Mobile Robots need to use path planning to navigate the environment

Robot Planning examples



SumUP

- We learned about Transformations and their meaning, as well as how to implement it in ROS
- We learned how to model, implement and control a robot from scratch!
  – Modeling - URDF
  – Simulation – Gazebo
  – Intro to Robot Planning!

Software for Autonomous Systems

SFfAS-31391:

Learning ROS Transforms (TF), Robot Visualization (RVIZ) and Simulation (Gazebo)

Lecturer, Course Coordinator: Evangelos Boukas—PhD