Robot operációs rendszerek és fejlesztői ökoszisztémák

Robot modellezés

Pepó Tamás

2025. szeptember 29.

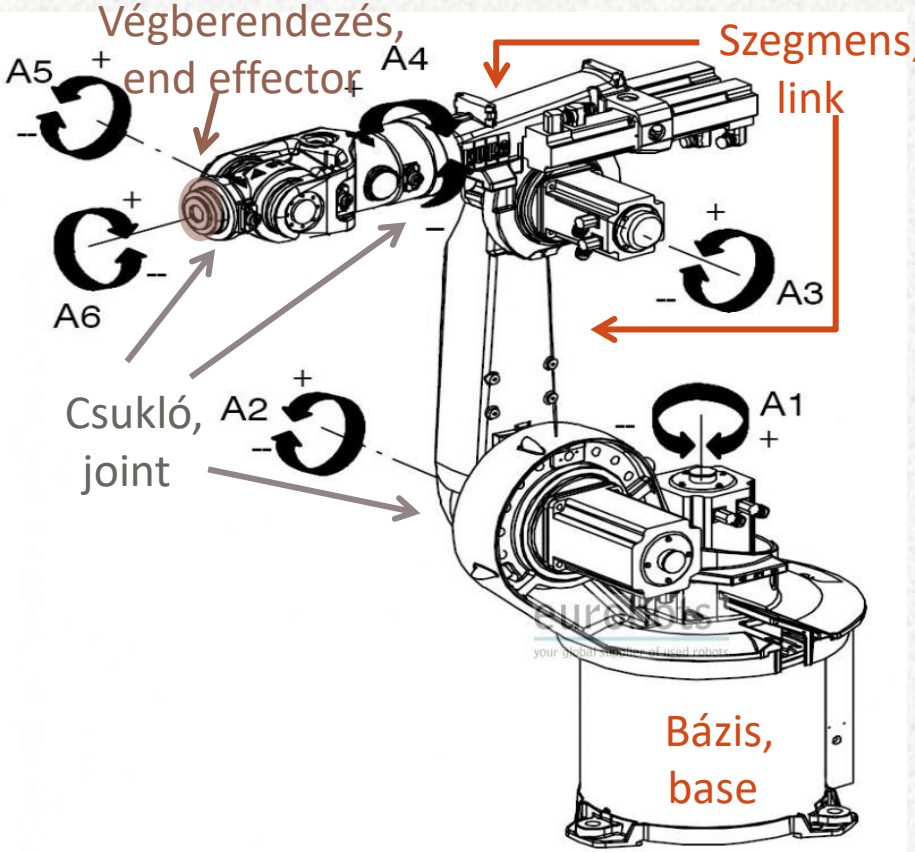
KUKA

MŰEGYETEM 1782

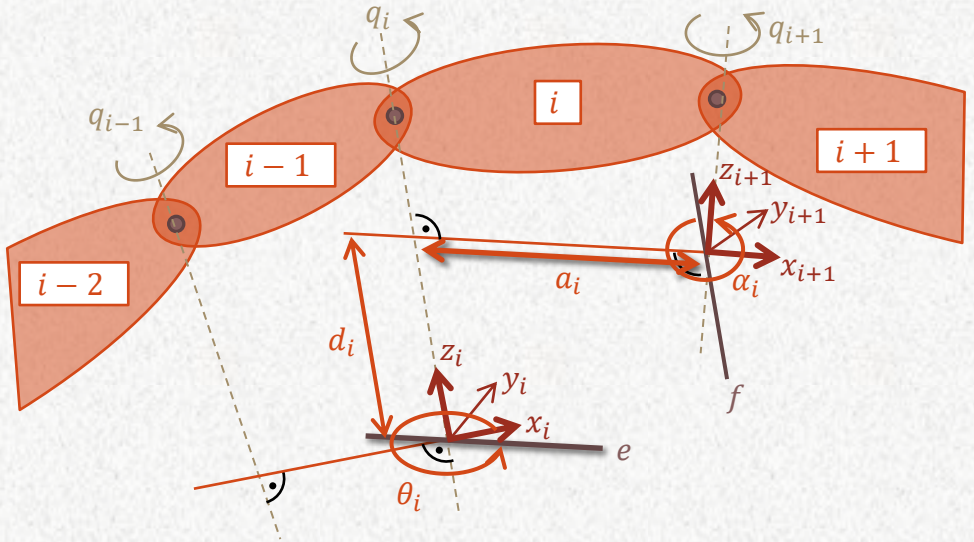
iit

Visszaemlékezés…

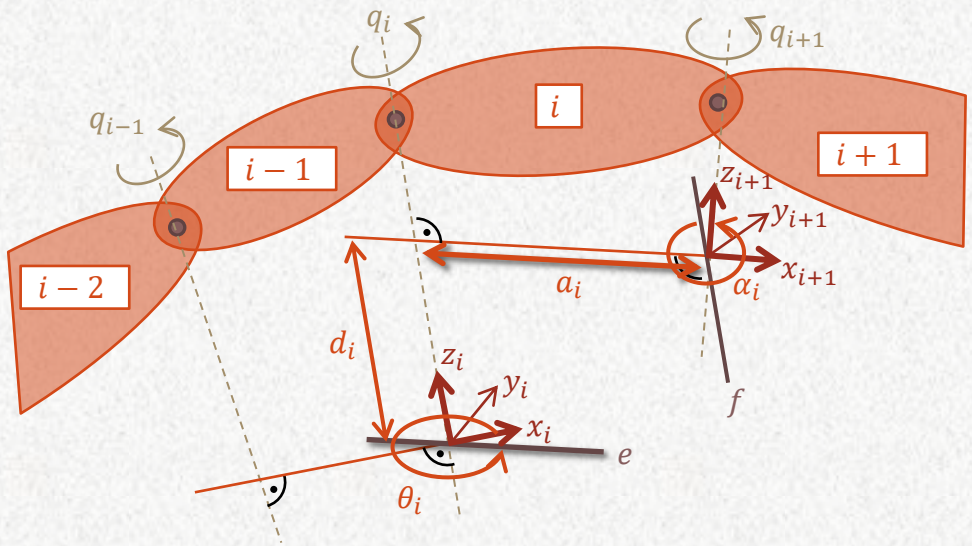
Robotikai alapok



Robotkar geometriai modellezése DH paraméterekkel



Ipari robotkarok – geometriai modell



Denavit-Hartenberg (DH) alak:

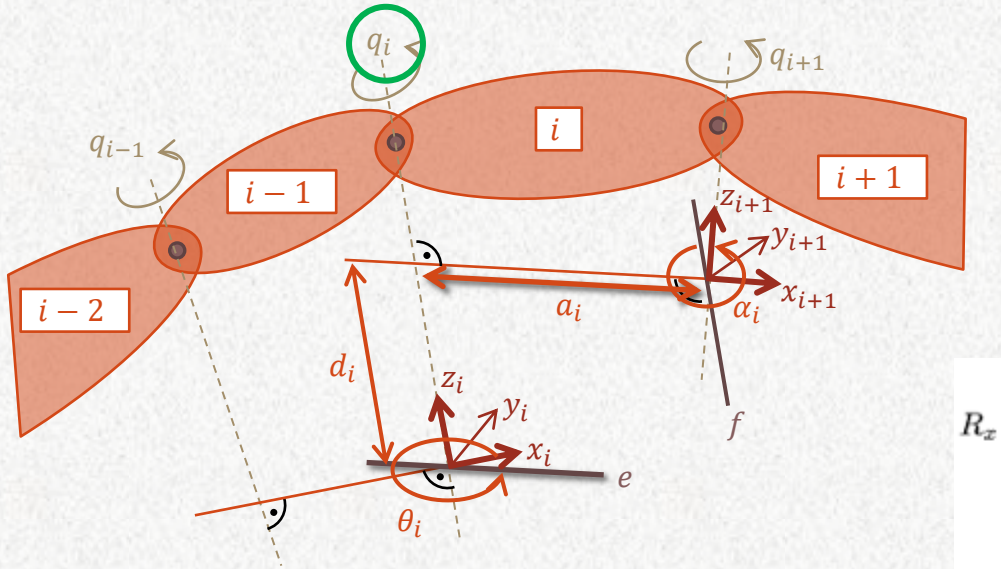
- megadja a csuklótengelyekhez rendelt koordináta-rendszerek egymáshoz képesti helyzetét

Csuklónként négy lépés:

- z_i körül forgatás θ_i szöggel (x_i-t e-be)
- z_i mentén eltolás d_i távolsággal (e-t x_{i+1}-be)
- x_{i+1} mentén eltolás a_i távolsággal (z_i-t f-be)
- x_{i+1} körüli forgatás α_i szöggel (f-et z_{i+1}-be)

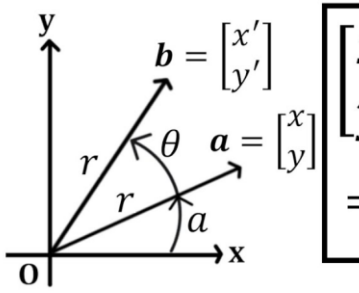
Csuklónként négy DH parameter: $\theta_i, d_i, a_i, \alpha_i$

Ipari robotkarok – geometriai modell



$(q_1,\ldots,q_m) \rightarrow (x,y,z,\phi,\theta,\gamma)$

Rotation Matrix



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R(\theta) \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

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

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

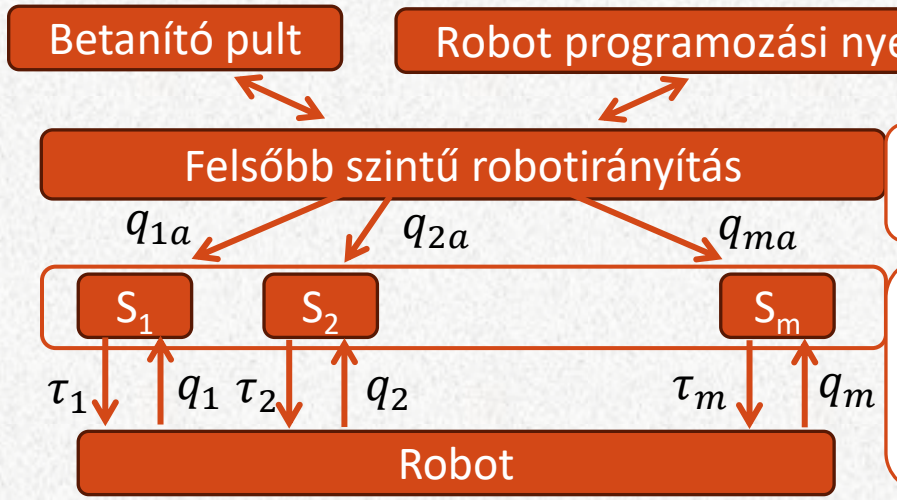
$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}_{i-1,i} = \mathbf{Rot}(z,\vartheta_i)\cdot\mathbf{Trans}(z,d_i)\cdot\mathbf{Trans}(x,a_i)\cdot\mathbf{Rot}(x,\alpha_i) =$$

$$= \begin{bmatrix} C_{\vartheta_i} & -S_{\vartheta_i} & 0 & 0 \\ S_{\vartheta_i} & C_{\vartheta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C_{\alpha_i} & -S_{\alpha_i} & 0 \\ 0 & S_{\alpha_i} & C_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} C_{\vartheta_i} & -S_{\vartheta_i}C_{\alpha_i} & S_{\vartheta_i}S_{\alpha_i} & a_iC_{\vartheta_i} \\ S_{\vartheta_i} & C_{\vartheta_i}C_{\alpha_i} & -C_{\vartheta_i}S_{\alpha_i} & a_iS_{\vartheta_i} \\ 0 & S_{\alpha_i} & C_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

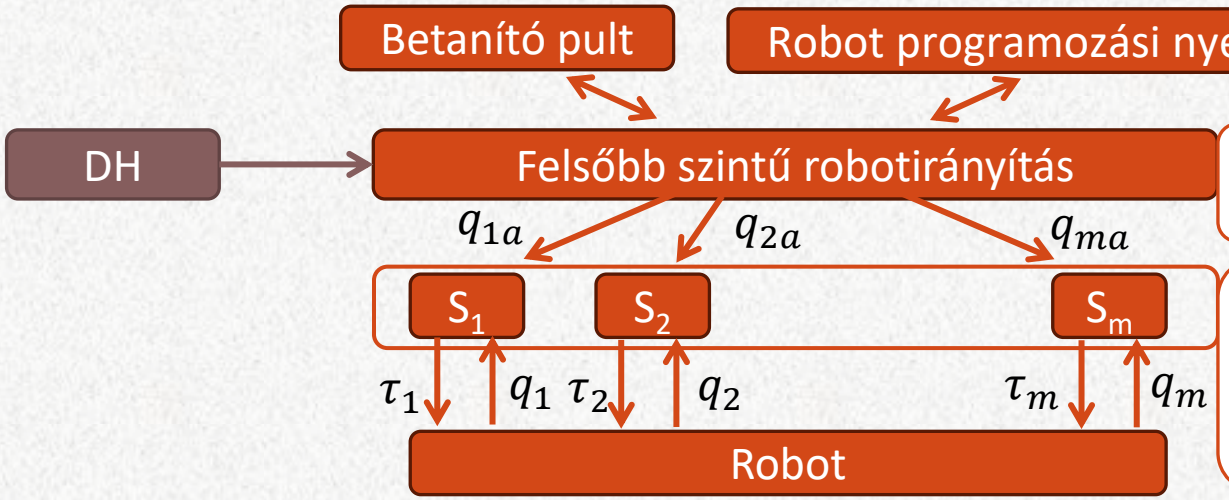
Robotikai alapok



Felső szintű robotirányítás feladata:
- Csuklónkénti alapjel meghatározása (q_{ia})

Alsó szintű robotirányítás (S_i) elemei:
- Motor + áttétel
- Érzékelő (q_i mérése)
- Hiba számítása ($q_{ia} - q_i$)
- Szabályozó + teljesítmény elektronika (τ_i nyomaték)

Robotikai alapok



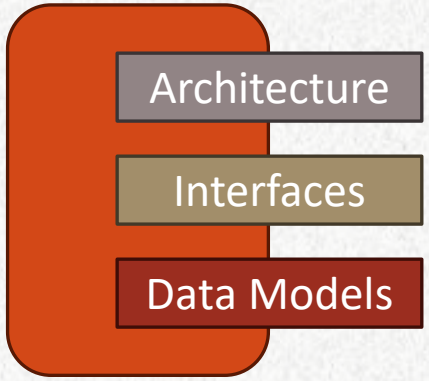
Felső szintű robotirányítás feladata:

- Csuklónkénti alapjel meghatározása (q_{ia})

Alsó szintű robotirányítás (S_i) elemei:

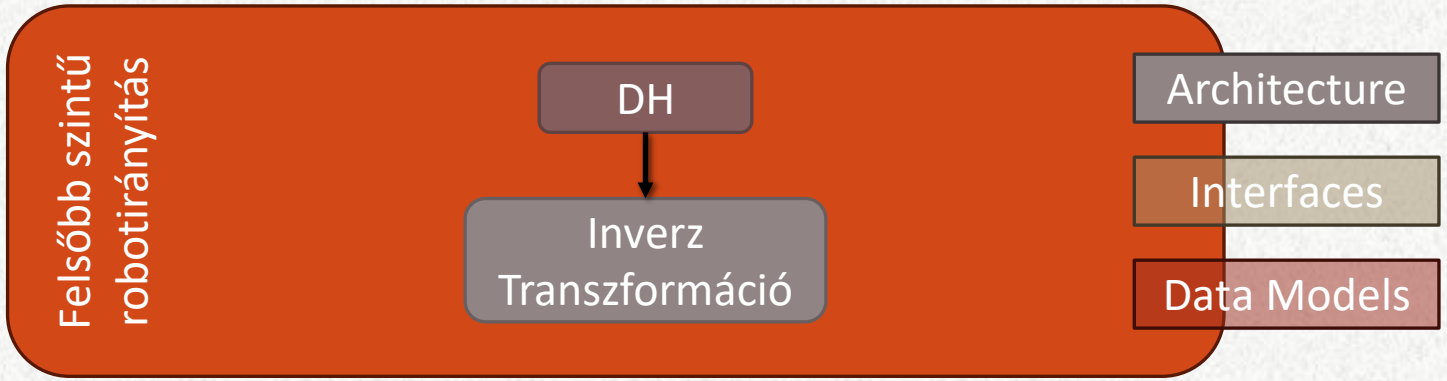
- Motor + áttétel
- Érzékelő (q_i mérése)
- Hiba számítása ($q_{ia} - q_i$)
- Szabályozó + teljesítmény elektronika (τ_i nyomaték)

Describing a service

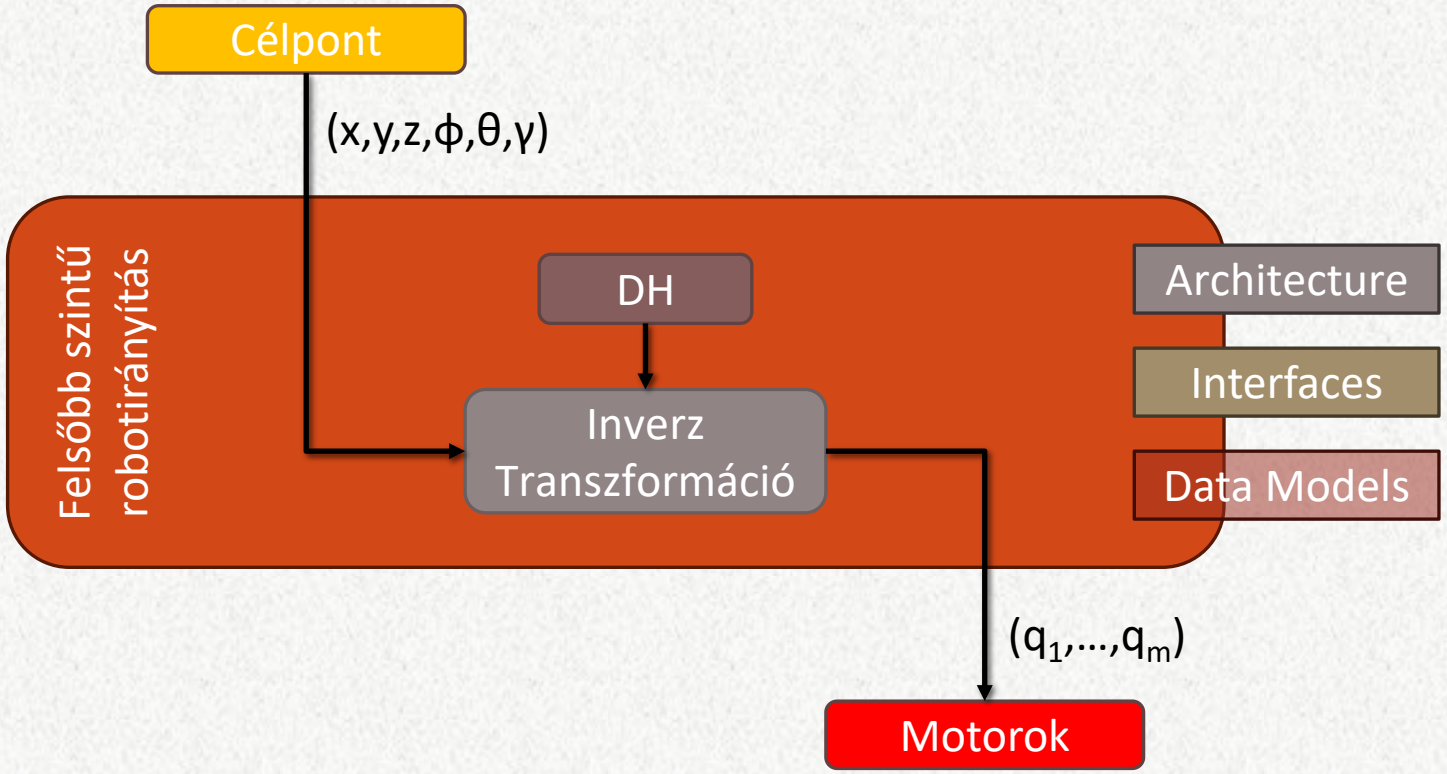


Describing a service

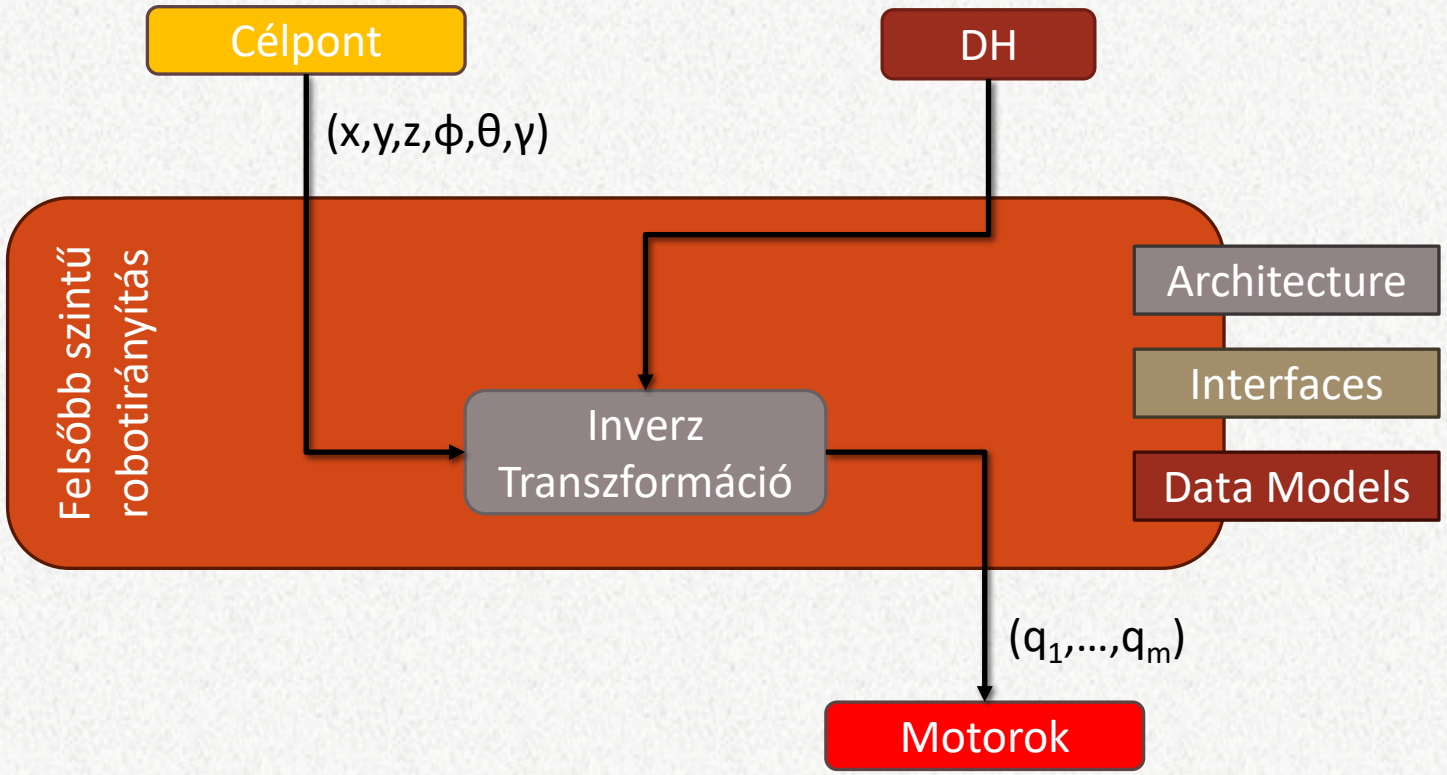
Describing a service



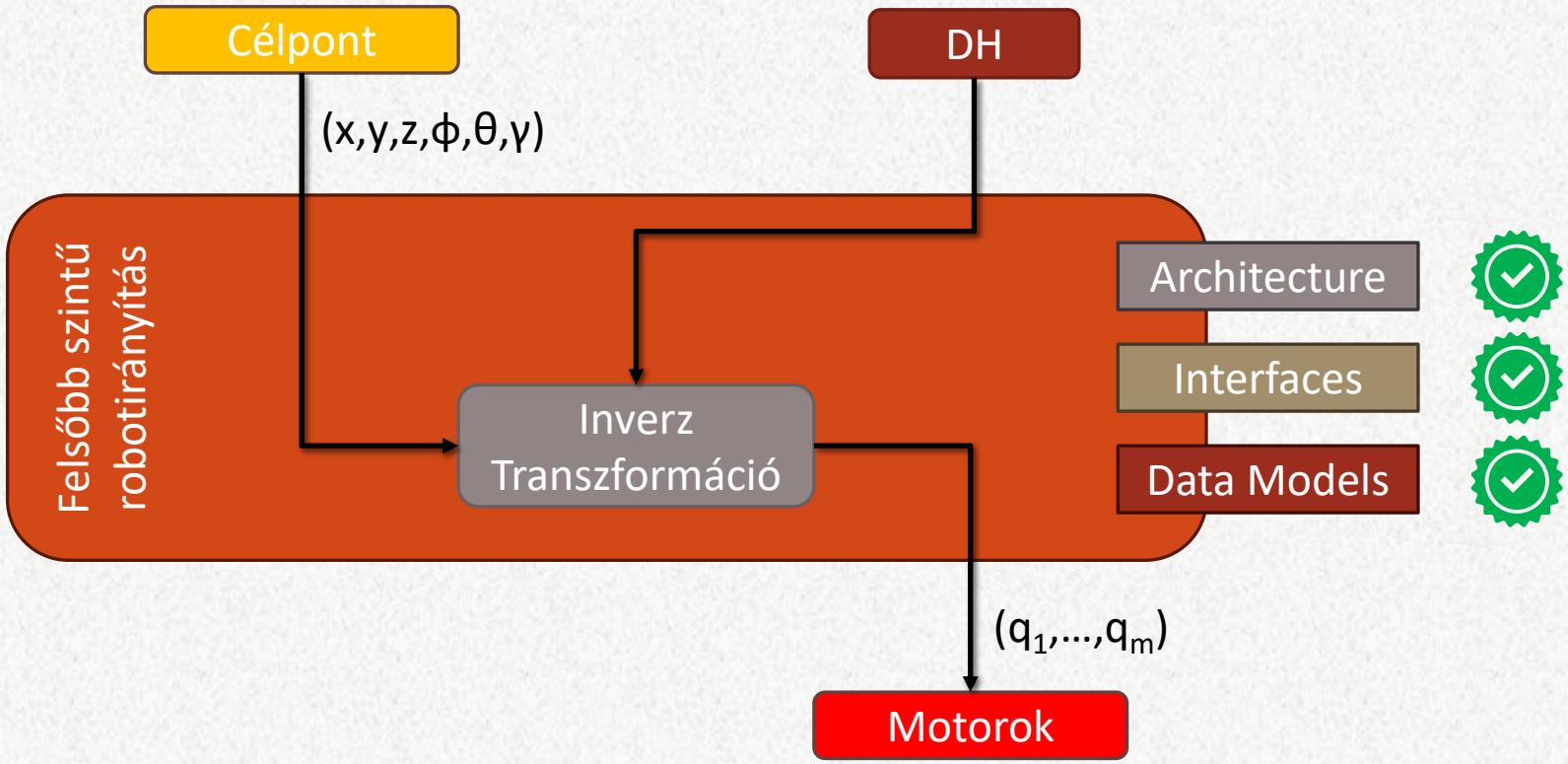
Describing a service



Describing a service



Describing a service

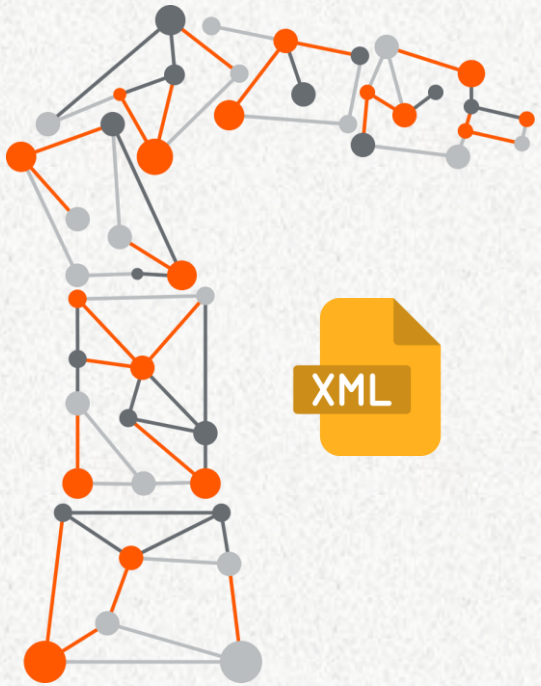


Describing a service

Data Models

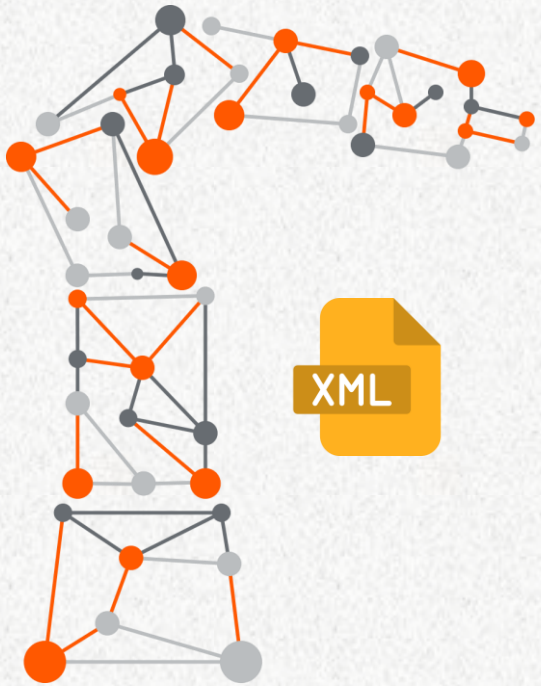
URDF

Unified Robot Description Format



- XML-based
  - On top of a defined schema
- Used for multibody system modeling
- Not part of ROS
  - Used in several other robotics applications
  - Or a converter exists to proprietary formats

Unified Robot Description Format

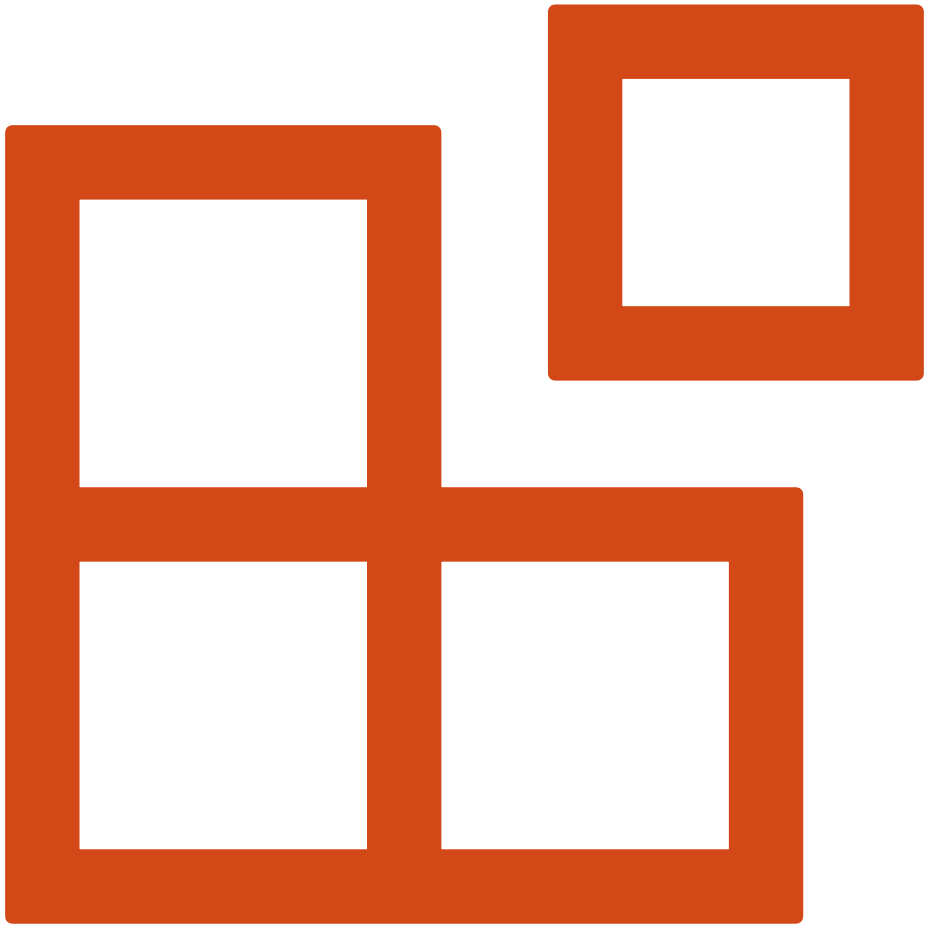


- Main parts are: <link>, <joint>
  - Sounds familiar, right?
- As URDF is a standard, plenty of documentation is available

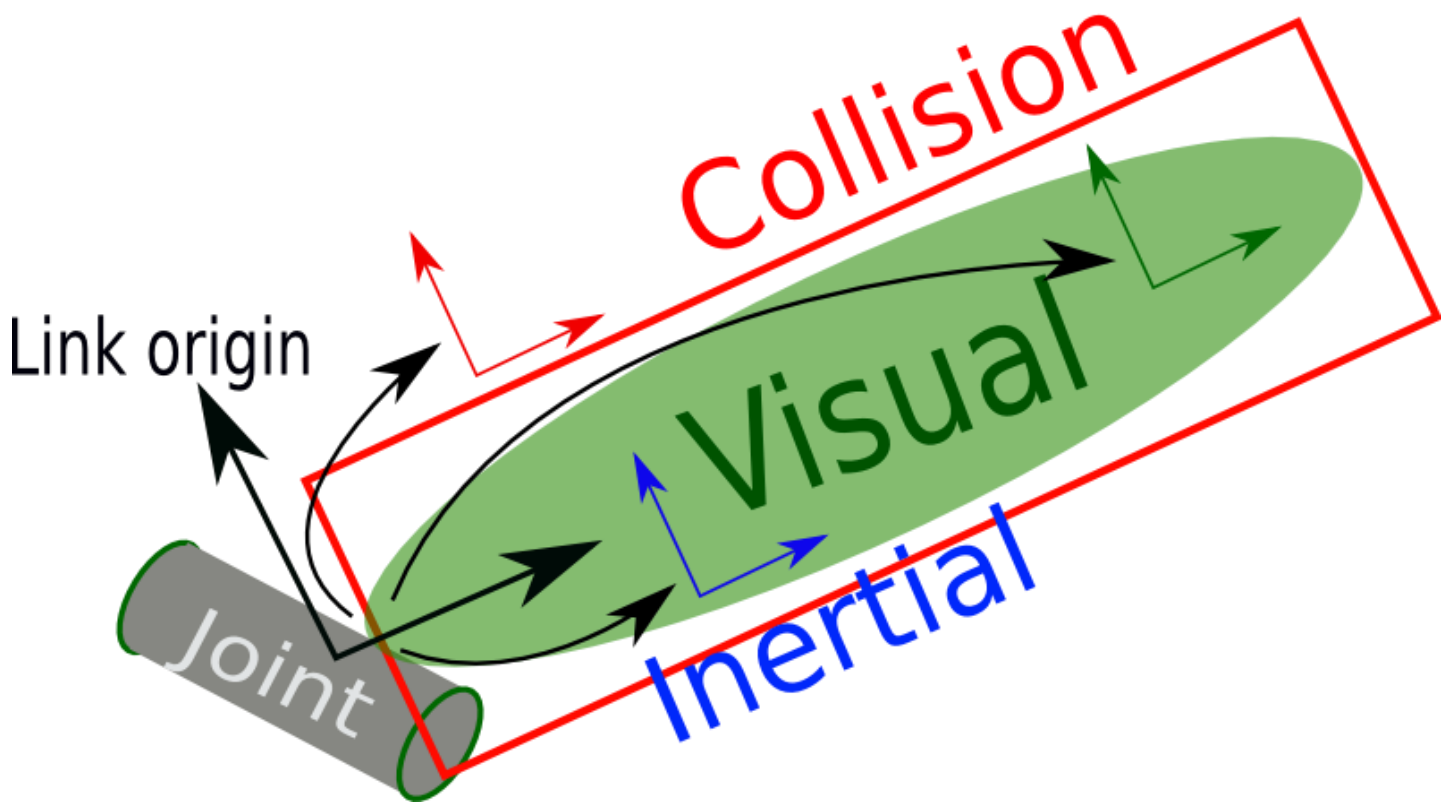
Why do we want to model?

- To solve the…
  - Direct & Inverse geometric problem
  - Direct & Inverse kinematic problem
  - Direct & Inverse dynamic problem
- To visualize our robot
- To have collision constraints
- In general: To depict our hardware in the scope of the target ecosystem.

URDF Elements



<link>



Link attribute(s)

- **name** *(required)*
  - The name of the link itself.

Link elements

<inertial> *(optional: defaults to a zero mass and zero inertia if not specified)*

- The link's mass, position of its center of mass, and its central inertia properties.
- **<origin>** *(optional: defaults to identity if not specified)*
  - This pose (translation, rotation) describes the position and orientation of the link's center of mass frame C relative to the link-frame L.
  - **xyz** *(optional: defaults to zero vector)*
    - Represents the position vector from Lo (the link-frame origin) to Co (the link's center of mass) as **x L̂x + y L̂y + z L̂z**, where **L̂x, L̂y, L̂z** are link-frame L's orthogonal unit vectors.
  - **rpy** *(optional: defaults to identity if not specified)*
    - Represents the orientation of C's unit vectors **Ĉx, Ĉy, Ĉz** relative to link-frame L as a sequence of Euler rotations (r p y) in radians. Note: **Ĉx, Ĉy, Ĉz** do not need to be aligned with the link's principal axes of inertia.
- **<mass>**
  - The mass of the link is represented by the **value** attribute of this element

Link elements

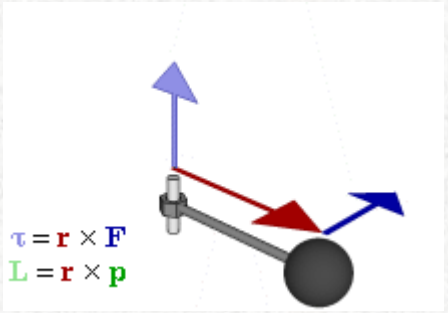
<inertial> *(optional: defaults to a zero mass and zero inertia if not specified)*

- The link's mass, position of its center of mass, and its central inertia properties.
- **<inertia>**
  - This link's moments of inertia **ixx, iyy, izz** and products of inertia **ixy, ixz, iyz** about Co (the link's center of mass) for the unit vectors **Ĉx, Ĉy, Ĉz** fixed in the center-of-mass frame C.
    - Note: the orientation of **Ĉx, Ĉy, Ĉz** relative to **L̂x, L̂y, L̂z** is specified by the rpy values in the <origin> tag.
  - The attributes **ixx, ixy, ixz, iyy, iyz, izz** for some primitive shapes are here. URDF assumes a negative product of inertia convention (for more info, see these MathWorks docs for working with CAD tools).

Inertia?

Angular Momentum

Analogous to linear momentum (P)
Usual notation: L or N



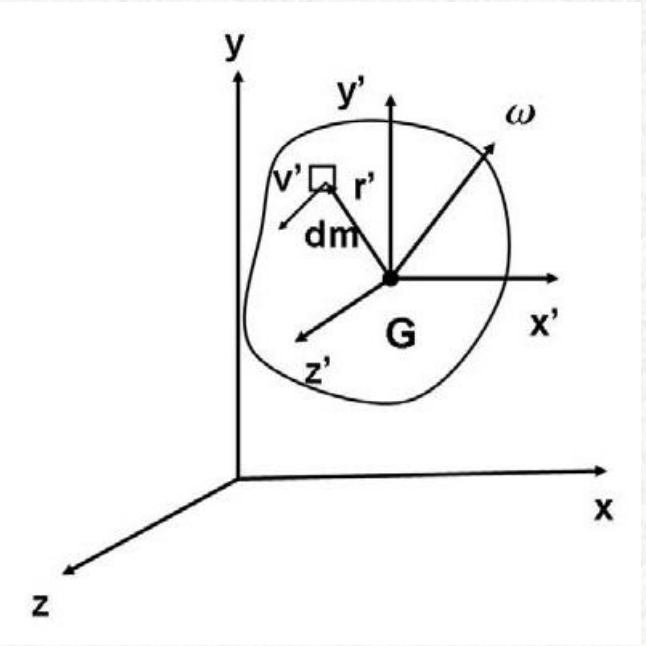
Inertia

Angular momentum of a system of particles about the center of mass

$$\boldsymbol{H}_G = \sum_{i=1}^{n} (\boldsymbol{r}'_i \times m_i(\boldsymbol{\omega} \times \boldsymbol{r}'_i)) = \sum_{i=1}^{n} m_i r'^2_i \boldsymbol{\omega}$$

$$\boldsymbol{H}_G = \sum_{i=1}^{n} (\boldsymbol{r}'_i \times m_i(\boldsymbol{\omega} \times \boldsymbol{r}'_i)) = \sum_{i=1}^{n} m_i r'^2_i \boldsymbol{\omega} = \int_m \boldsymbol{r}' \times \boldsymbol{v}' \, dm$$

$$\boldsymbol{H}_G = \int_m \boldsymbol{r}' \times \boldsymbol{v}' \, dm \, .$$



Inertia

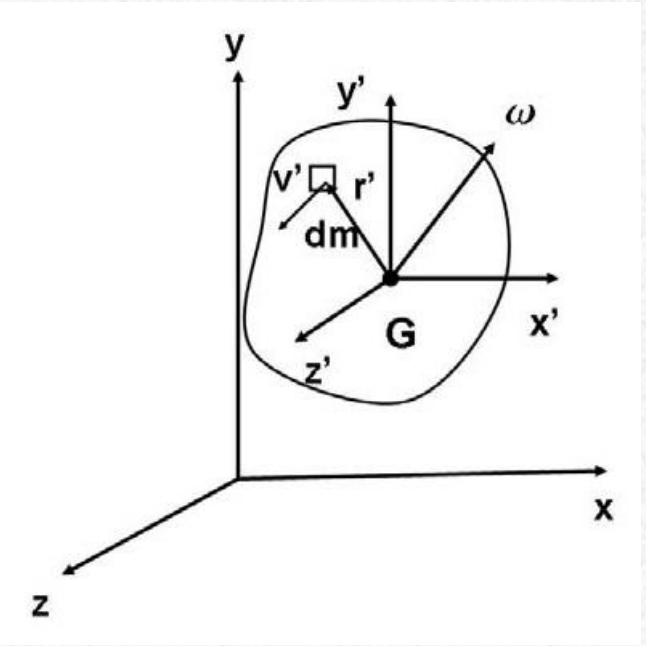
$$\boldsymbol{H}_G = \sum_{i=1}^{n} (\boldsymbol{r}'_i \times m_i(\boldsymbol{\omega} \times \boldsymbol{r}'_i)) = \sum_{i=1}^{n} m_i r'^2_i \boldsymbol{\omega}$$

$$\boldsymbol{H}_G = \sum_{i=1}^{n} (\boldsymbol{r}'_i \times m_i(\boldsymbol{\omega} \times \boldsymbol{r}'_i)) = \sum_{i=1}^{n} m_i r'^2_i \boldsymbol{\omega} = \int_m \boldsymbol{r}' \times \boldsymbol{v}' \, dm$$

$$\boldsymbol{H}_G = \int_m \boldsymbol{r}' \times \boldsymbol{v}' \, dm \, .$$

$$\boldsymbol{H}_G = \int_m \boldsymbol{r}' \times (\boldsymbol{\omega} \times \boldsymbol{r}') \, dm = \int_m [(\boldsymbol{r}' \cdot \boldsymbol{r}')\boldsymbol{\omega} - (\boldsymbol{r}' \cdot \boldsymbol{\omega})\boldsymbol{r}'] \, dm$$

$$\boldsymbol{A} \times (\boldsymbol{B} \times \boldsymbol{C}) = (\boldsymbol{A} \cdot \boldsymbol{C})\boldsymbol{B} - (\boldsymbol{A} \cdot \boldsymbol{B})\boldsymbol{C}$$

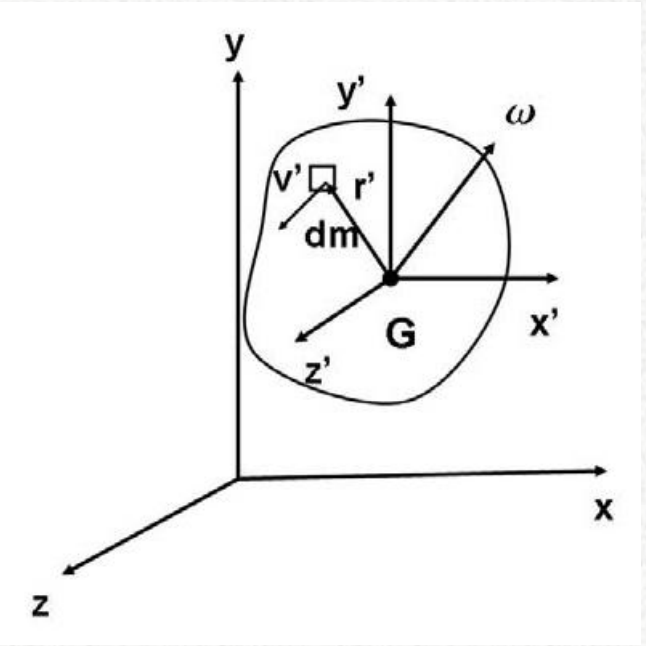


Inertia

$$\boldsymbol{H}_G = \int_m \boldsymbol{r}' \times (\boldsymbol{\omega} \times \boldsymbol{r}')\, dm = \int_m [(\boldsymbol{r}' \cdot \boldsymbol{r}')\boldsymbol{\omega} - (\boldsymbol{r}' \cdot \boldsymbol{\omega})\boldsymbol{r}']\, dm$$

$$\boldsymbol{A} \times (\boldsymbol{B} \times \boldsymbol{C}) = (\boldsymbol{A} \cdot \boldsymbol{C})\boldsymbol{B} - (\boldsymbol{A} \cdot \boldsymbol{B})\boldsymbol{C}$$

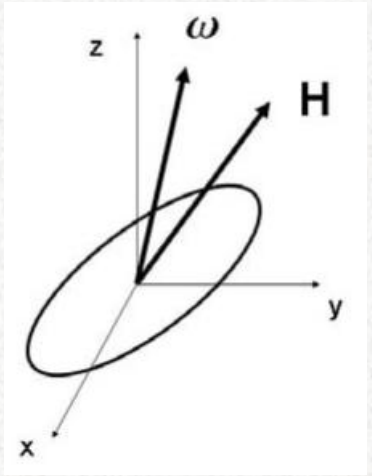
$$\begin{aligned}
\boldsymbol{H}_G &= \left(\omega_x \int_m (x'^2 + y'^2 + z'^2)\, dm - \int_m (\omega_x x' + \omega_y y' + \omega_z z')x'\, dm\right) \boldsymbol{i} \\
&+ \left(\omega_y \int_m (x'^2 + y'^2 + z'^2)\, dm - \int_m (\omega_x x' + \omega_y y' + \omega_z z')y'\, dm\right) \boldsymbol{j} \\
&+ \left(\omega_z \int_m (x'^2 + y'^2 + z'^2)\, dm - \int_m (\omega_x x' + \omega_y y' + \omega_z z')z'\, dm\right) \boldsymbol{k} \\
&= (\ I_{xx}\omega_x - I_{xy}\omega_y - I_{xz}\omega_z)\, \boldsymbol{i} \\
&+ (-I_{yx}\omega_x + I_{yy}\omega_y - I_{yz}\omega_z)\, \boldsymbol{j} \\
&+ (-I_{zx}\omega_x - I_{zy}\omega_y + I_{zz}\omega_z)\, \boldsymbol{k}\ .
\end{aligned}$$



Inertia

Angular momentum with respect to a fixed point O

$$
\begin{aligned}
\boldsymbol{H}_O &= \left((I_{xx})_O\, \omega_x - (I_{xy})_O\, \omega_y - (I_{xz})_O\, \omega_z \right) \boldsymbol{i} \\
&+ \left(-(I_{yx})_O\, \omega_x + (I_{yy})_O\, \omega_y - (I_{yz})_O\, \omega_z \right) \boldsymbol{j} \\
&+ \left(-(I_{zx})_O\, \omega_x - (I_{zy})_O\, \omega_y + (I_{zz})_O\, \omega_z \right) \boldsymbol{k}
\end{aligned}
$$



Moments > 0 Products symmetrical

$$
\begin{pmatrix} H_{Gx} \\ H_{Gy} \\ H_{Gz} \end{pmatrix} = \begin{pmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}
$$

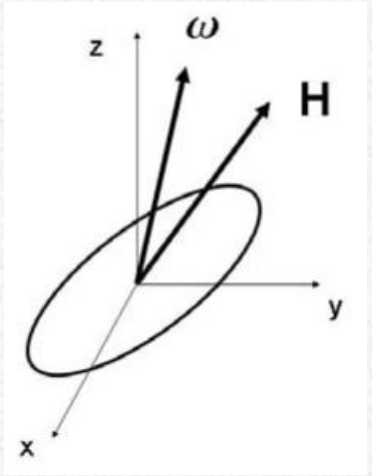
$$
\boldsymbol{H}_G = [I_G]\boldsymbol{\omega},
$$

$$
\boldsymbol{H}_O = [I_O]\,\boldsymbol{\omega}\ ,
$$

Inertia

Angular momentum with respect to a fixed point O

$$
\begin{aligned}
\boldsymbol{H}_O \;=\; & \left((I_{xx})_O\,\omega_x - (I_{xy})_O\,\omega_y - (I_{xz})_O\,\omega_z \right) \boldsymbol{i} \\
+ \; & \left(-(I_{yx})_O\,\omega_x + (I_{yy})_O\,\omega_y - (I_{yz})_O\,\omega_z \right) \boldsymbol{j} \\
+ \; & \left(-(I_{zx})_O\,\omega_x - (I_{zy})_O\,\omega_y + (I_{zz})_O\,\omega_z \right) \boldsymbol{k}
\end{aligned}
$$

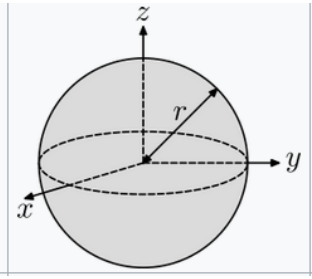


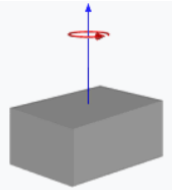
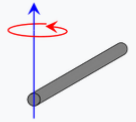
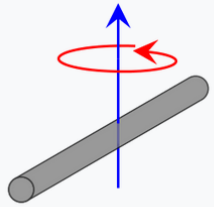
Inertia tensor

$$
\begin{pmatrix} H_{Gx} \\ H_{Gy} \\ H_{Gz} \end{pmatrix} = \begin{pmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}
$$

$$
\boldsymbol{H}_G = [I_G]\boldsymbol{\omega},
$$

$$
\boldsymbol{H}_O = [I_O]\,\boldsymbol{\omega}\ ,
$$

Solid sphere of radius *r* and mass *m*		$I = \begin{bmatrix} \frac{2}{5}mr^2 & 0 & 0 \\ 0 & \frac{2}{5}mr^2 & 0 \\ 0 & 0 & \frac{2}{5}mr^2 \end{bmatrix}$

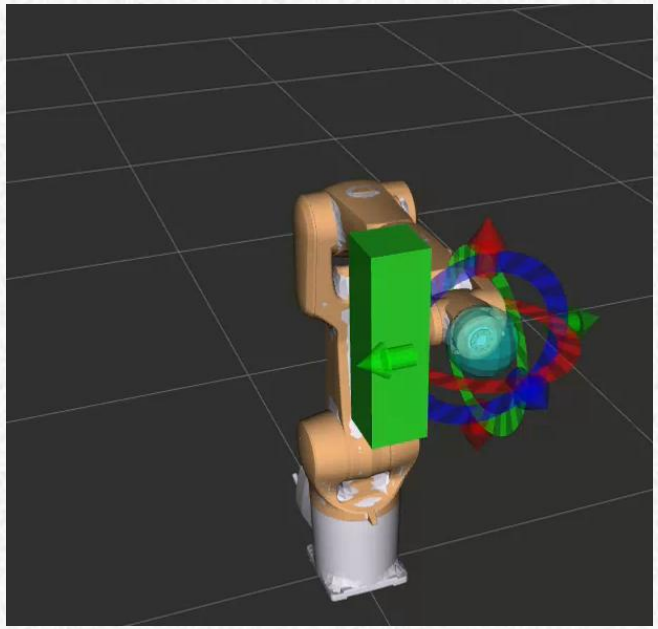
Solid cuboid of width *w*, height *h*, depth *d*, and mass *m*		$I = \begin{bmatrix} \frac{1}{12}m(h^2+d^2) & 0 & 0 \\ 0 & \frac{1}{12}m(w^2+d^2) & 0 \\ 0 & 0 & \frac{1}{12}m(w^2+h^2) \end{bmatrix}$
Slender rod along *y*-axis of length *l* and mass *m* about end		$I = \begin{bmatrix} \frac{1}{3}ml^2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3}ml^2 \end{bmatrix}$
Slender rod along *y*-axis of length *l* and mass *m* about center		$I = \begin{bmatrix} \frac{1}{12}ml^2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{12}ml^2 \end{bmatrix}$

Example Intertia tensors

Link elements

<visual> *(optional)*

- The visual properties of the link. This element specifies the shape of the object (box, cylinder, etc.) for visualization purposes.
  - **Note:** multiple instances of <visual> tags can exist for the same link. The union of the geometry they define forms the visual representation of the link.
- **name** *(optional)*
  - Specifies a name for a part of a link's geometry. This is useful to be able to refer to specific bits of the geometry of a link.
- **<origin>** *(optional: defaults to identity if not specified)*
  - The reference frame of the visual element with respect to the reference frame of the link.
  - **xyz** *(optional: defaults to zero vector)*
    - Represents the **x**, **y**, **z** offset.
  - **rpy** *(optional: defaults to identity if not specified)*
    - Represents the fixed axis roll, pitch and yaw angles in radians.



Link elements

<visual> *(optional)*

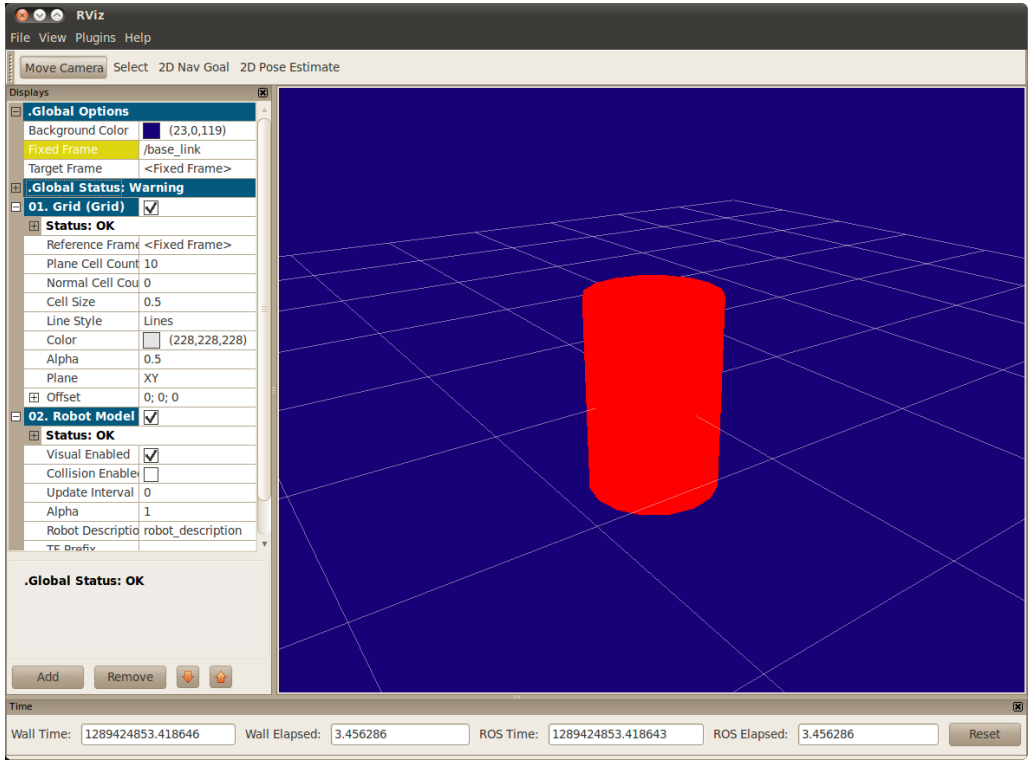
- <geometry> (required)
- The shape of the visual object. This can be one of the following:
  - <box>
  - <cylinder>
  - <sphere>
  - <mesh>
- A trimesh element specified by a filename, and an optional scale that scales the mesh's axis-aligned-bounding-box. Any geometry format is acceptable, but specific application compatibility is dependent on implementation. The recommended format for best texture and color support is Collada .dae files.
- The mesh file is not transferred between machines referencing the same model. It must be a local file. Prefix the filename with package://<packagename>/<path> to make the path to the mesh file relative to the package <packagename>.

Link elements

<visual> *(optional)*

- The visual properties of the link. This element specifies the shape of the object (box, cylinder, etc.) for visualization purposes.
  - **Note:** multiple instances of <visual> tags can exist for the same link. The union of the geometry they define forms the visual representation of the link.
- **<material>** *(optional)*
  - The material of the visual element. It is allowed to specify a material element outside of the 'link' object, in the top level 'robot' element. From within a link element you can then reference the material by name.
  - **name** name of the material
  - **<color>** *(optional)*
    - **rgba** The color of a material specified by set of four numbers representing red/green/blue/alpha, each in the range of [0,1].
  - **<texture>** *(optional)*
    - The texture of a material is specified by a **filename**

```
<?xml version="1.0"?>
<robot name="myfirst">
  <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
    </visual>
  </link>
</robot>
```



- The fixed frame is the transform frame where the center of the grid is located. Here, it's a frame defined by our one link, base_link.
- The visual element (the cylinder) has its origin at the center of its geometry as a default. Hence, half the cylinder is below the grid.

Link elements

<collision> *(optional)*

- The collision properties of a link. Note that this can be different from the visual properties of a link, for example, simpler collision models are often used to reduce computation time. **Note:** multiple instances of <collision> tags can exist for the same link. The union of the geometry they define forms the collision representation of the link.

- **name** *(optional)*
  - Specifies a name for a part of a link's geometry. This is useful to be able to refer to specific bits of the geometry of a link.

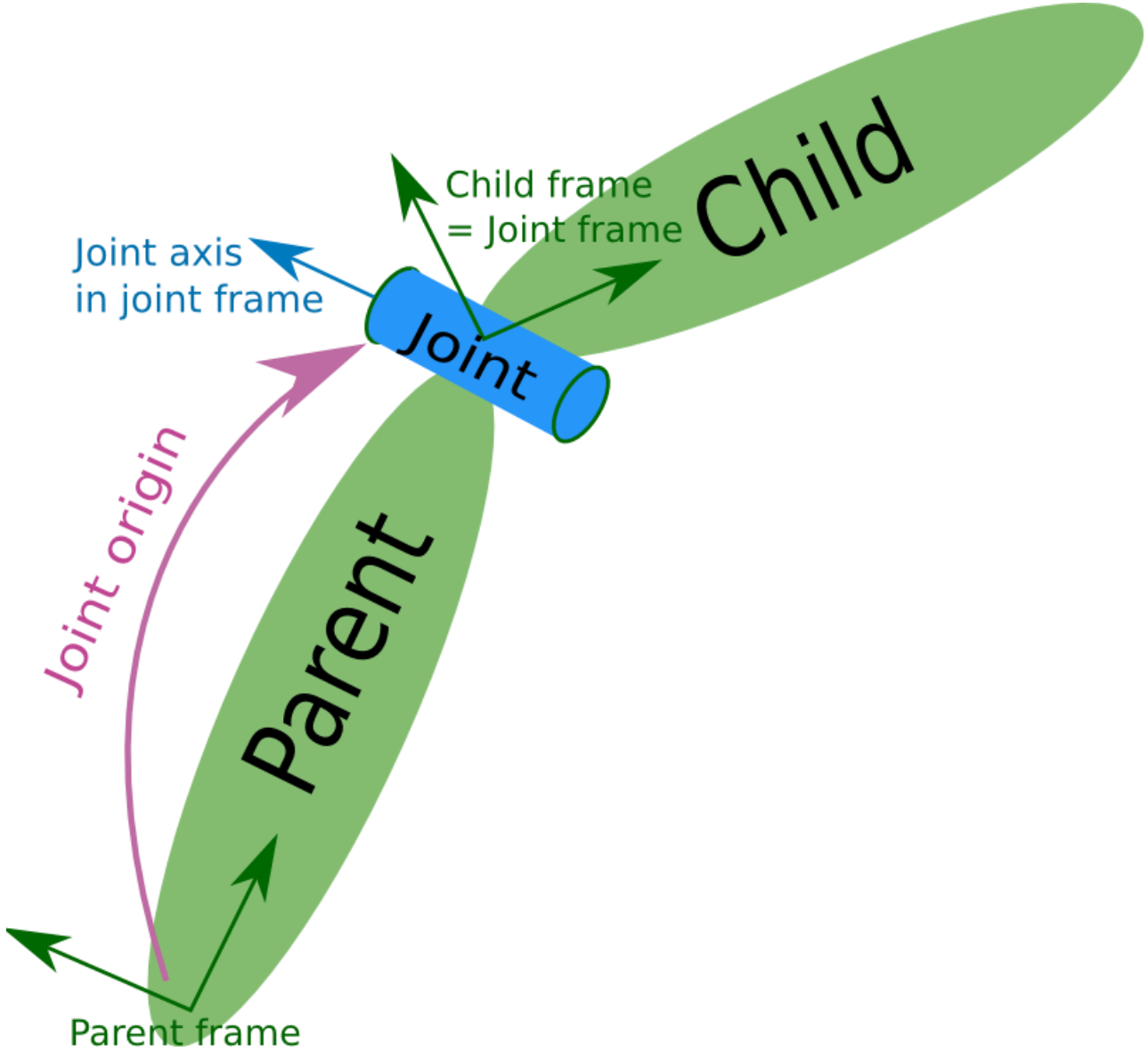
- **<origin>** *(optional: defaults to identity if not specified)*
  - The reference frame of the collision element, relative to the reference frame of the link.
  - **xyz** *(optional: defaults to zero vector)*
    - Represents the **x**, **y**, **z** offset.
  - **rpy** *(optional: defaults to identity if not specified)*
    - Represents the fixed axis roll, pitch and yaw angles in radians.

- **<geometry>**
  - See the geometry description in the visual element.

Example link

```
<link name="my_link">
  <inertial>
    <origin xyz="0 0 0.5" rpy="0 0 0"/>
    <mass value="1"/>
    <inertia ixx="100"  ixy="0"  ixz="0" iyy="100" iyz="0" izz="100" />
  </inertial>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <box size="1 1 1" />
    </geometry>
    <material name="Cyan">
      <color rgba="0 1.0 1.0 1.0"/>
    </material>
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder radius="1" length="0.5"/>
    </geometry>
  </collision>
</link>
```

<joint>


Child
Child frame
= Joint frame
Joint axis
in joint frame
Joint
Joint origin
Parent
Parent frame

Joint attributes

- **name** *(required)*
  - Specifies a unique name of the joint
- **type** *(required)*
  - Specifies the type of joint, where type can be one of the following:
    - **revolute** — a hinge joint that rotates along the axis and has a limited range specified by the upper and lower limits.
    - **continuous** — a continuous hinge joint that rotates around the axis and has no upper and lower limits.
    - **prismatic** — a sliding joint that slides along the axis, and has a limited range specified by the upper and lower limits.
    - **fixed** — this is not really a joint because it cannot move. All degrees of freedom are locked. This type of joint does not require the **<axis>**, **<calibration>**, **<dynamics>**, **<limits>** or **<safety_controller>**.
    - **floating** — this joint allows motion for all 6 degrees of freedom.
    - **planar** — this joint allows motion in a plane perpendicular to the axis.

Joint elements

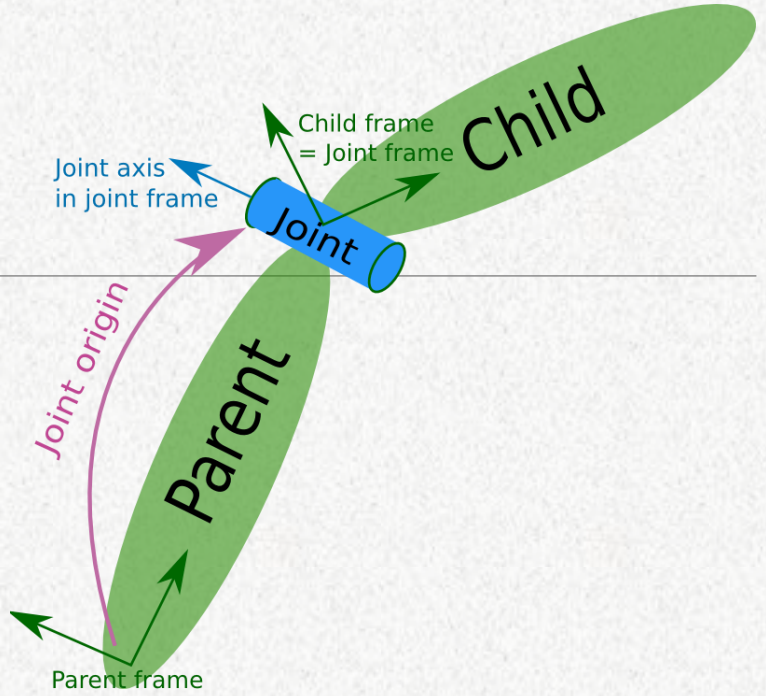
<parent> *(required)*

- Parent link name with mandatory attribute: **link**
  - The name of the link that is the parent of this link in the robot tree structure.

<child> *(required)*

- Child link name with mandatory attribute: **link**
  - The name of the link that is the child link.

Joint elements



<origin> *(optional: defaults to identity if not specified)*

- This is the transform from the parent link to the child link. The joint is located at the origin of the child link, as shown in the figure above.
- **xyz** *(optional: defaults to zero vector)*
  - Represents the *x*, *y*, *z* offset. All positions are specified in *metres*.
- **rpy** *(optional: defaults to zero vector)*
  - Represents the rotation around fixed axis: first *roll* around *x*, then *pitch* around *y* and finally *yaw* around *z*. All angles are specified in *radians*.

Joint elements

<axis> *(optional: defaults to (1,0,0))*

- The joint axis specified in the joint frame. This is the axis of rotation for revolute joints, the axis of translation for prismatic joints, and the surface normal for planar joints. The axis is specified in the joint frame of reference. Fixed and floating joints do not use the axis field.
  - **xyz** *(required)*
    - Represents the (*x*, *y*, *z*) components of a vector. The vector should be normalized.

<calibration> *(optional)*

- The reference positions of the joint, used to calibrate the absolute position of the joint.
- **rising** *(optional)*
  - When the joint moves in a positive direction, this reference position will trigger a rising edge.
- **falling** *(optional)*
  - When the joint moves in a positive direction, this reference position will trigger a falling edge.

Joint elements

<dynamics> *(optional)*

- An element specifying physical properties of the joint. These values are used to specify modeling properties of the joint, particularly useful for simulation.
- **damping** *(optional, defaults to 0)*
  - The physical damping value of the joint (in *newton-seconds per metre* [$N \cdot s/m$] for prismatic joints, in *newton-metre-seconds per radian* [$N \cdot m \cdot s/rad$] for revolute joints).
- **friction** *(optional, defaults to 0)*
  - The physical static friction value of the joint (in *newtons* [N] for prismatic joints, in *newton-metres* [$N \cdot m$] for revolute joints).

Joint elements

<limit> *(required only for revolute and prismatic joint)*

- An element can contain the following attributes:
- **lower** *(optional, defaults to 0)*
  - An attribute specifying the lower joint limit (in *radians* for revolute joints, in *metres* for prismatic joints). Omit if joint is continuous.
- **upper** *(optional, defaults to 0)*
  - An attribute specifying the upper joint limit (in *radians* for revolute joints, in *metres* for prismatic joints). Omit if joint is continuous.
- **effort** *(required)*
  - An attribute for enforcing the maximum joint effort (|*applied effort*| < |*effort*|).
- **velocity** *(required)*
  - An attribute for enforcing the maximum joint velocity (in *radians per second* [*rad/s*] for revolute joints, in *metres per second* [*m/s*] for prismatic joints).

Joint elements

<mimic> *(optional)*

- This tag is used to specify that the defined joint mimics another existing joint. The value of this joint can be computed as *value = multiplier * other_joint_value + offset*.
- **joint** *(required)*
  - This specifies the name of the joint to mimic.
- **multiplier** *(optional)*
  - Specifies the multiplicative factor in the formula above.
- **offset** *(optional)*
  - Specifies the offset to add in the formula above. Defaults to 0 (radians for revolute joints, meters for prismatic joints)

Joint elements

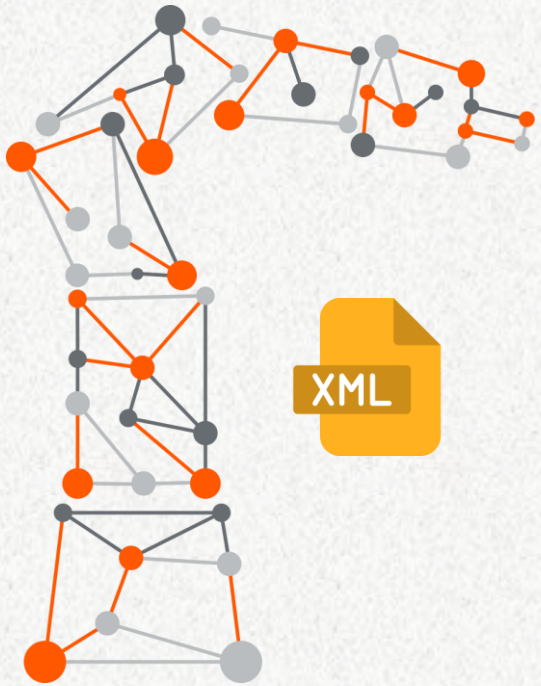
<safety_controller> *(optional)*

- An element can contain the following attributes:
- **soft_lower_limit** *(optional, defaults to 0)*
  - An attribute specifying the lower joint boundary where the safety controller starts limiting the position of the joint. This limit needs to be larger than the lower joint limit (see above). See See safety limits for more details.
- **soft_upper_limit** *(optional, defaults to 0)*
  - An attribute specifying the upper joint boundary where the safety controller starts limiting the position of the joint. This limit needs to be smaller than the upper joint limit (see above). See See safety limits for more details.
- **k_position** *(optional, defaults to 0)*
  - An attribute specifying the relation between position and velocity limits. See See safety limits for more details.
- **k_velocity** *(required)*
  - An attribute specifying the relation between effort and velocity limits. See See safety limits for more details.

Example joint

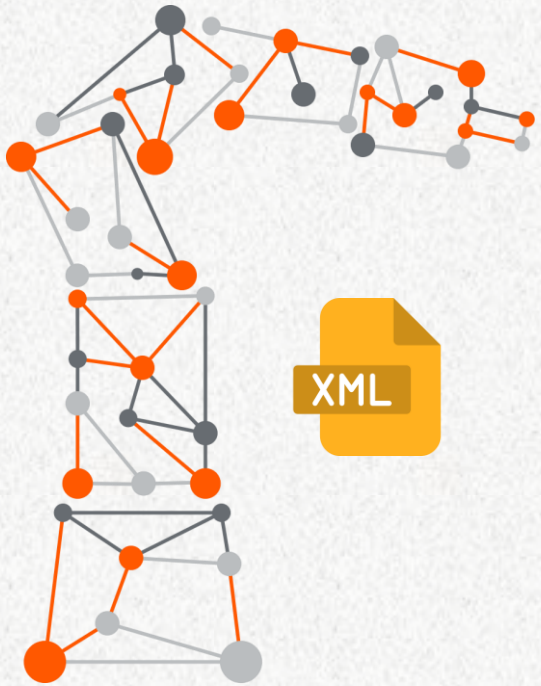
```
<joint name="my_joint" type="floating">
   <origin xyz="0 0 1" rpy="0 0 3.1416"/>
   <parent link="link1"/>
   <child link="link2"/>

   <calibration rising="0.0"/>
   <dynamics damping="0.0" friction="0.0"/>
   <limit effort="30" velocity="1.0" lower="-2.2" upper="0.7" />
   <safety_controller k_velocity="10" k_position="15" soft_lower_limit="-2.0" soft_upper_limit="0.5" />
</joint>
```

Unified Robot Description Format



- Is it really just <links> and <joints>?

Unified Robot Description Format



- Is it really just <links> and <joints>?
  - Okay, not much more.
- What happens if I put multiple robots of the same type, to my scene?
  - Name clash!
  - Robot's name is one problem…
  - Link, Joint names impose a bigger issue…

Xacro

- As its name implies, xacro is a macro language for XML.
- The xacro program runs all of the macros and outputs the result.
- Can be triggered from roslaunch.
- Typical usage looks something like this:
  - xacro model.xacro > model.urdf

Xacro

```
<xacro:property name="width" value="0.2" />
<xacro:property name="bodylen" value="0.6" />
<link name="base_link">
    <visual>
        <geometry>
            <cylinder radius="${width}" length="${bodylen}"/>
        </geometry>
        <material name="blue"/>
    </visual>
    <collision>
        <geometry>
            <cylinder radius="${width}" length="${bodylen}"/>
        </geometry>
    </collision>
</link>
```

Textual/Constant placeholders

```
<cylinder radius="${wheeldiam/2}" length="0.1"/>
<origin xyz="${reflect*(width+.02)} 0 0.25" />
```

Math

Xacro

```
1    <xacro:macro name="default_inertial" params="mass">
2        <inertial>
3                <mass value="${mass}" />
4                <inertia ixx="1.0" ixy="0.0" ixz="0.0"
5                     iyy="1.0" iyz="0.0"
6                     izz="1.0" />
7        </inertial>
8    </xacro:macro>
```

…or parameterized macro blocks.

```
1 <xacro:default_inertial mass="10"/>
```

KROSHU macros

kuka_robot_descriptions / kuka_fortec_support / urdf / kr560_r3100

naslevente Refactor Xacro structure (#86)

Code Blame 208 lines (208 loc) · 9.12 KB

```
1   <?xml version="1.0"?>
2   <robot xmlns:xacro="http://wiki.ros.org/xacro">
3     <xacro:macro name="kuka_kr560_r3100_2_robot" params="pre
```

Kristóf Mátyás Pásztor

Intern

Platform Robot OS

Idén először GYAKORLAT! ☺

kuka_robot_descriptions / kuka_fortec_support / urdf / kr560_r3100_2.urdf.xacro

pasztork Add robot model verification argument to Xacro files (#89)

Code Blame 26 lines (26 loc) · 1.58 KB

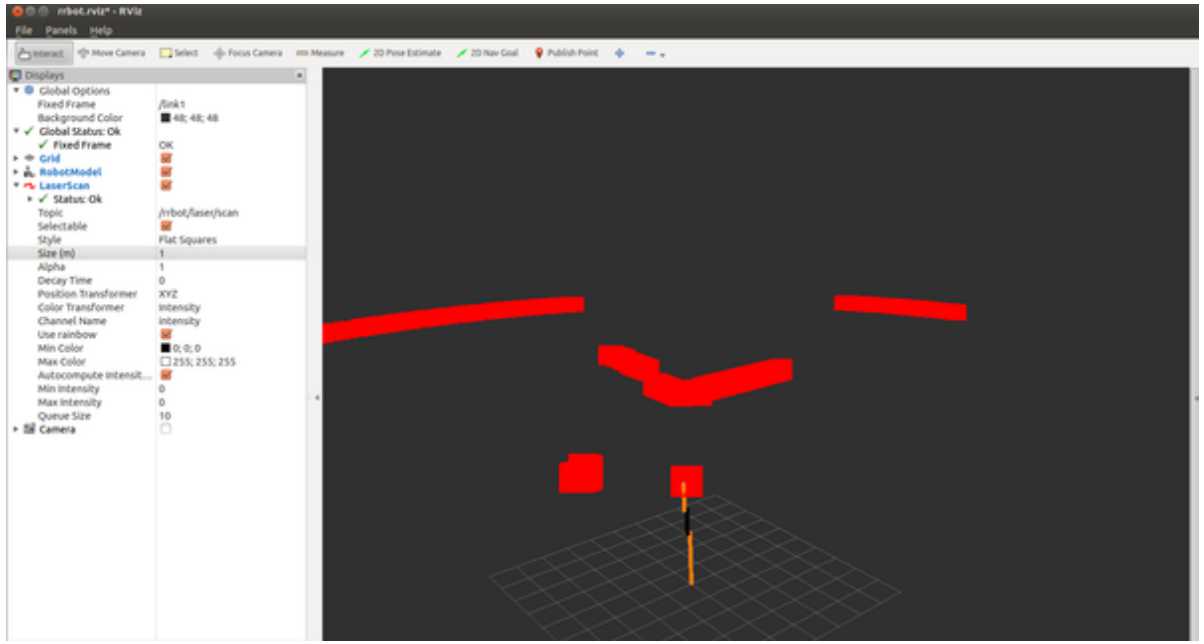
```
1   <?xml version="1.0"?>
2   <robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="kr560_r3100_2">
3     <!-- Import macro files  -->
4     <xacro:include filename="$(find kuka_fortec_support)/urdf/kr560_r3100_2_macro.xacro"/>
5     <xacro:include filename="$(find kuka_fortec_support)/urdf/kr_fortec_ros2_control_macro.xacr
6     <!-- Read additional arguments  -->
7     <xacro:arg name="driver_version" default="rsi_only"/>
8     <xacro:arg name="mode" default="hardware"/>
9     <xacro:arg name="use_gpio" default="false"/>
10    <xacro:arg name="roundtrip_time" default="0"/>
11    <xacro:arg name="client_ip" default="0.0.0.0"/>
12    <xacro:arg name="client_port" default="59152"/>
13    <xacro:arg name="controller_ip" default="0.0.0.0"/>
14    <xacro:arg name="verify_robot_model" default="true"/>
15    <xacro:arg name="prefix" default=""/>
16    <xacro:arg name="x" default="0"/>
17    <xacro:arg name="y" default="0"/>
18    <xacro:arg name="z" default="0"/>
19    <xacro:arg name="roll" default="0"/>
20    <xacro:arg name="pitch" default="0"/>
21    <xacro:arg name="yaw" default="0"/>
22    <xacro:kuka_fortec_ros2_control name="$(arg prefix)kr560_r3100_2" driver_version="$(arg dr
23    <xacro:kuka_kr560_r3100_2_robot prefix="$(arg prefix)" package_name="kuka_fortec_support">
24      <origin xyz="$(arg x) $(arg y) $(arg z)" rpy="$(arg roll) $(arg pitch) $(arg yaw)"/>
25    </xacro:kuka_kr560_r3100_2_robot>
26  </robot>
```

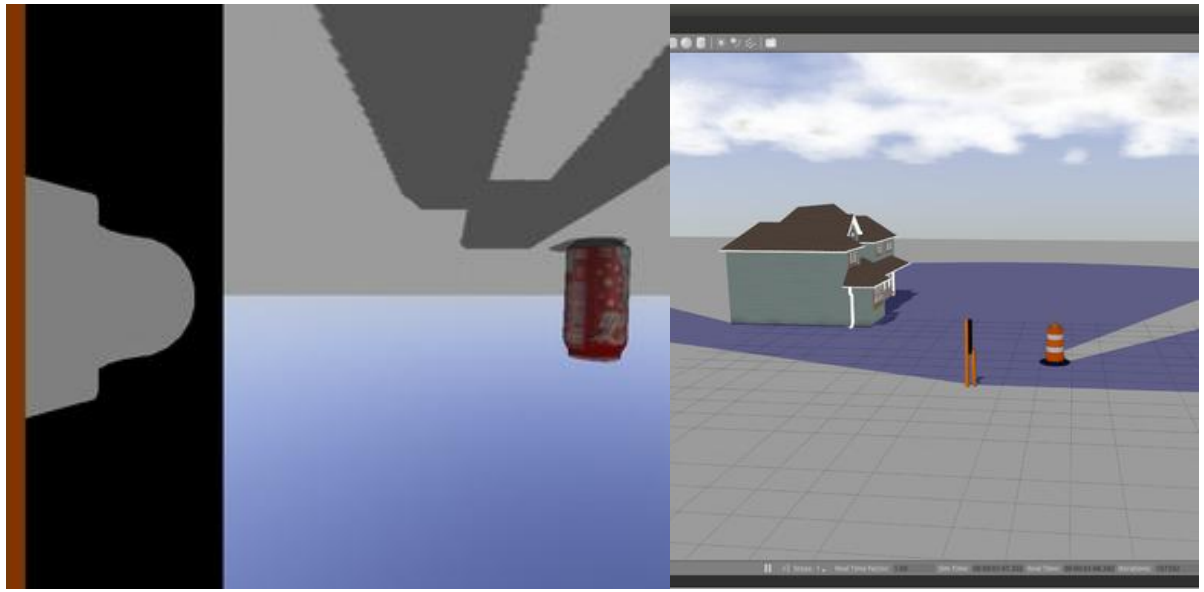
Simulation with Gazebo

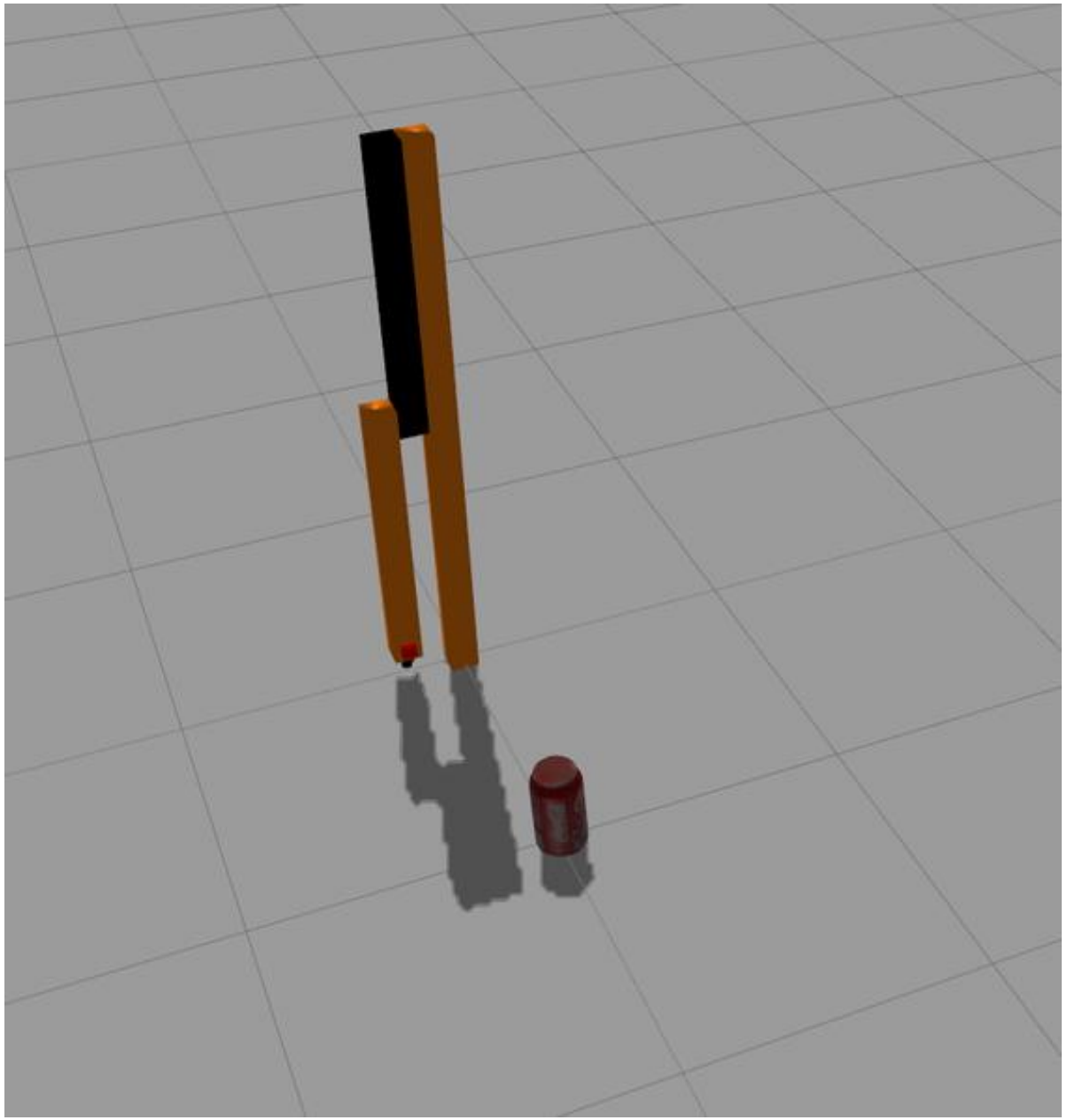
- Not all parameters are in URDF as of now, to enable Gazebo usage
  - Think of URDF as a generic configuration of the robot
  - In which you can add parameters, parts (like sensor physic geometry or so…) into the scene
  - Which then can be referenced by Gazebo

- Lot of parameters should be configured to use Gazebo properly
  - List can be found here:
    - http://classic.gazebosim.org/tutorials?tut=ros_urdf&cat=connect_ros

- Motor drive, sensors, actuators… all can be referenced from and used by Gazebo


rrbot.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
Global Options
Fixed Frame /link1
Background Color 48; 48; 48
Global Status: Ok
Fixed Frame OK
Grid
RobotModel
LaserScan
Status: Ok
Topic /rrbot/laser/scan
Selectable
Style Flat Squares
Size (m) 1
Alpha 1
Decay Time 0
Position Transformer XYZ
Color Transformer Intensity
Channel Name intensity
Use rainbow
Min Color 0; 0; 0
Max Color 255; 255; 255
Autocompute Intensit...
Min Intensity 0
Max Intensity 0
Queue Size 10
Camera

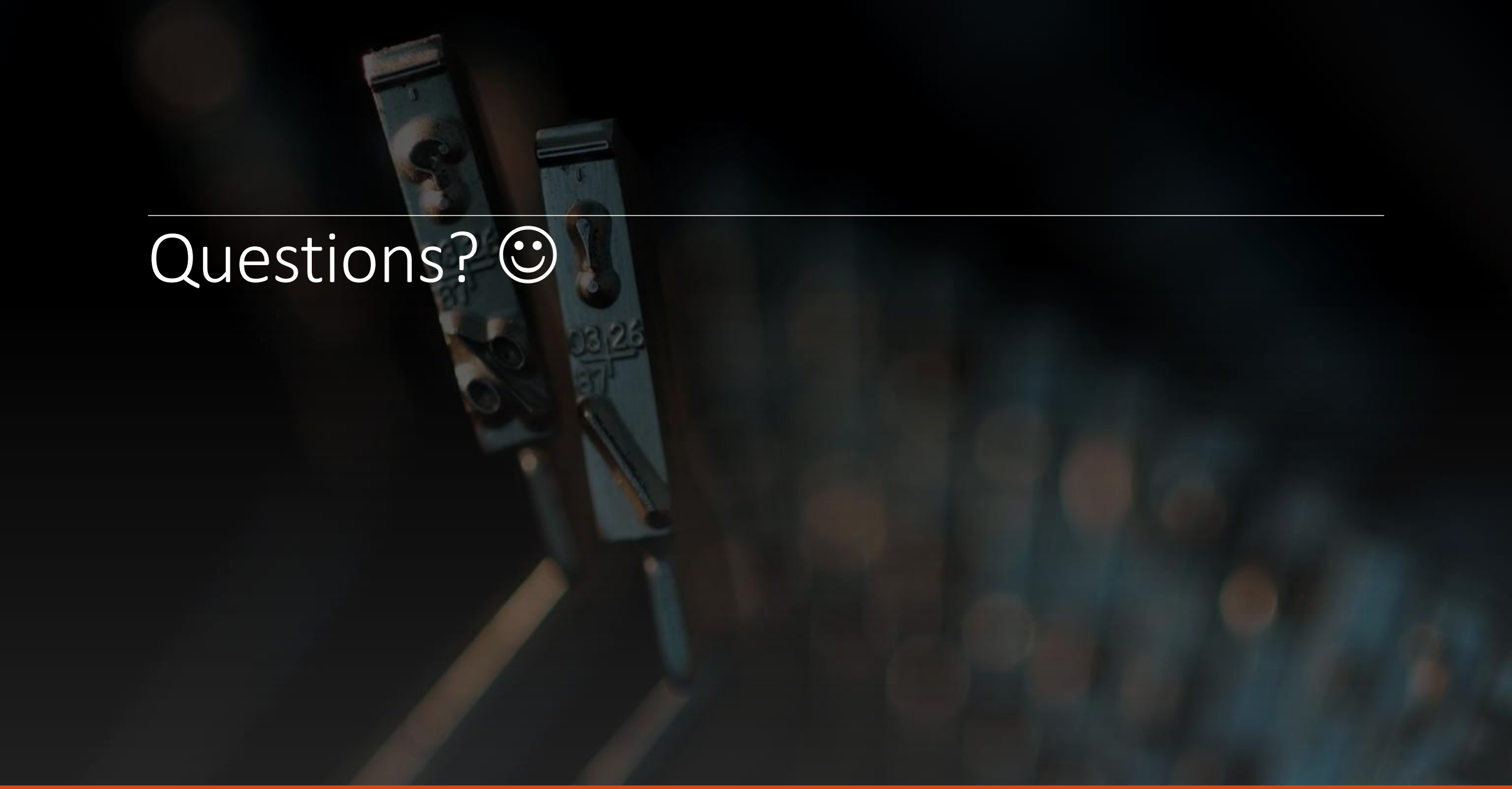




Summary

What can we do from today?

- Create a robot model
  - With kinematics and dynamics
    - Understand the data requirements for both
    - Understand inertial parameters
  - With visuals and collison data
- Create a URDF file for our robots
  - Understand the format specifics of URDF
  - Understand the conventions of URDF (Semantics)
- Try to move a robot in ROS / ROS 2
- Try to integrate to Gazebo simulation



Questions? ☺