上海科技大学
ShanghaiTech University

CS289: **Mobile Manipulation Fall 2025**

Sören Schwertfeger

ShanghaiTech University

ShanghaiTech Automation and Robotics Center • 上海科技大学自动化与机器人中心
STAR CENTER

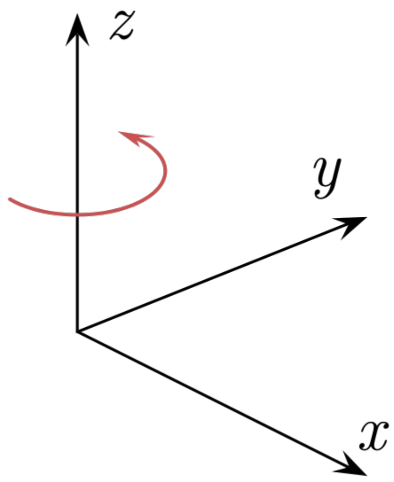
Outline

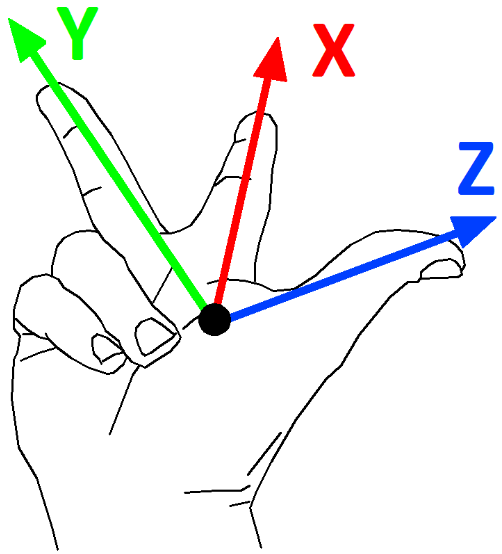
- 3D Geometry
- Kinematics

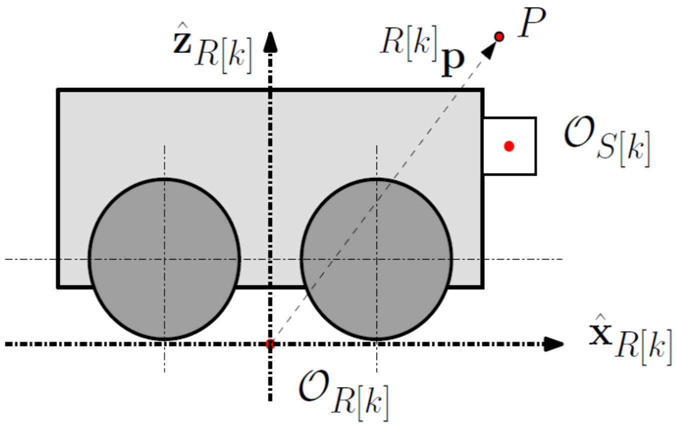
COORDINATE SYSTEM

Right Hand Coordinate System

- Standard in Robotics
- Positive rotation around X is anti-clockwise
- Right-hand rule mnemonic:
  - Thumb: z-axis
  - Index finger: x-axis
  - Second finger: y-axis
  - Rotation: Thumb = rotation axis, positive rotation in finger direction
- Robot Coordinate System:
  - X front
  - Z up (Underwater: Z down)
  - Y ???

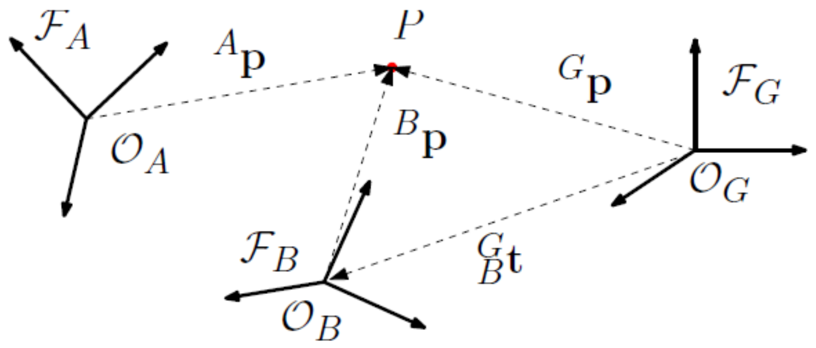






Right Hand Rule http://en.wikipedia.org/wiki/Right-hand_rule

Transform



Notation	Meaning
$\mathcal{F}_{\mathrm{R}[k]}$	Coordinate frame attached to object 'R' (usually the robot) at sample time-instant k.
$\mathcal{O}_{\mathrm{R}[k]}$	Origin of $\mathcal{F}_{\mathrm{R}[k]}$.
${}^{\mathrm{R}[k]}\mathbf{p}$	For any general point P, the position vector $\overrightarrow{\mathcal{O}_{\mathrm{R}[k]}P}$ resolved in $\mathcal{F}_{\mathrm{R}[k]}$.
${}^{\mathrm{H}}\hat{\mathbf{x}}_{\mathrm{R}}$	The x-axis direction of $\mathcal{F}_{\mathrm{R}}$ resolved in $\mathcal{F}_{\mathrm{H}}$. Similarly, ${}^{\mathrm{H}}\hat{\mathbf{y}}_{\mathrm{R}}$, ${}^{\mathrm{H}}\hat{\mathbf{z}}_{\mathrm{R}}$ can be defined. Obviously, ${}^{\mathrm{R}}\hat{\mathbf{x}}_{\mathrm{R}} = \hat{\mathrm{e}}_1$. Time indices can be added to the frames, if necessary.
${}^{\mathrm{R}[k]}_{\mathrm{S}[k']}\mathbf{R}$	The rotation-matrix of $\mathcal{F}_{\mathrm{S}[k']}$ with respect to $\mathcal{F}_{\mathrm{R}[k]}$.
${}^{\mathrm{R}}_{\mathrm{S}}\mathbf{t}$	The translation vector $\overrightarrow{\mathcal{O}_{\mathrm{R}}\mathcal{O}_{\mathrm{S}}}$ resolved in $\mathcal{F}_{\mathrm{R}}$.

Transform between two coordinate frames

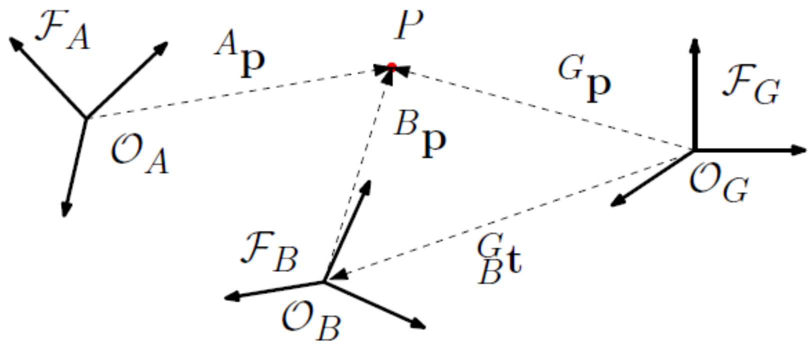
$${}^{\mathrm{G}}_{\mathrm{A}}\mathbf{t} \triangleq \overrightarrow{\mathcal{O}_{\mathrm{G}}\mathcal{O}_{\mathrm{A}}} \text{ resolved in } \mathcal{F}_{\mathrm{G}}$$

$$\begin{pmatrix} {}^{\mathrm{G}}\mathbf{p} \\ 1 \end{pmatrix} \equiv \begin{pmatrix} {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{R} & {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{t} \\ \mathbf{0}_{1\times[2,3]} & 1 \end{pmatrix} \begin{pmatrix} {}^{\mathrm{A}}\mathbf{p} \\ 1 \end{pmatrix} \qquad {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{T} \equiv \left\{ \begin{matrix} {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{t} \\ {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{R} \end{matrix} \right\}$$

$$\begin{aligned} {}^{\mathrm{G}}\mathbf{p} &= {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{R}\ {}^{\mathrm{A}}\mathbf{p} + {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{t} \\ &\triangleq {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{T}({}^{\mathrm{A}}\mathbf{p}). \end{aligned}$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & {}^{\mathrm{G}}_{\mathrm{A}}t_x \\ \sin\theta & \cos\theta & {}^{\mathrm{G}}_{\mathrm{A}}t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Transform: Operations



Transform between two coordinate frames (chaining, compounding):

$$ {}^G_B\mathbf{T} = {}^G_A\mathbf{T}\, {}^A_B\mathbf{T} \equiv \begin{Bmatrix} {}^G_A\mathbf{R}\, {}^A_B\mathbf{t} + {}^G_A\mathbf{t} \\ {}^G_A\mathbf{R}\, {}^A_B\mathbf{R} \end{Bmatrix} $$

Inverse of a Transform :

$$ {}^B_A\mathbf{T} = {}^A_B\mathbf{T}^{-1} \equiv \begin{Bmatrix} -\, {}^A_B\mathbf{R}^\top\, {}^A_B\mathbf{t} \\ {}^A_B\mathbf{R}^\top \end{Bmatrix} $$

Relative (Difference) Transform :

$$ {}^B_A\mathbf{T} = {}^G_B\mathbf{T}^{-1}\, {}^G_A\mathbf{T} $$

See: **Quick Reference to Geometric Transforms in Robotics** by Kaustubh Pathak on the webpage!

Chaining : $_{R[X+1]}^{G}\mathbf{T} = {}_{R[X]}^{G}\mathbf{T}\ {}_{R[X+1]}^{R[X]}\mathbf{T} \equiv \begin{Bmatrix} {}_{R[X]}^{G}\mathrm{R}\ {}_{R[X+1]}^{R[X]}t + {}_{R[X]}^{G}t \\ {}_{R[X]}^{G}\mathrm{R}\ {}_{R[X+1]}^{R[X]}\mathrm{R} \end{Bmatrix} = \begin{Bmatrix} {}_{R[X+1]}^{G}t \\ {}_{R[X+1]}^{G}\mathrm{R} \end{Bmatrix}$

In 2D Translation:

$$\begin{bmatrix} {}_{R[X+1]}^{G}t_x \\ {}_{R[X+1]}^{G}t_y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos {}_{R[X]}^{G}\theta & -\sin {}_{R[X]}^{G}\theta & {}_{R[X]}^{G}\mathrm{t}_\mathrm{x} \\ \sin {}_{R[X]}^{G}\theta & \cos {}_{R[X]}^{G}\theta & {}_{R[X]}^{G}\mathrm{t}_\mathrm{y} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}_{R[X+1]}^{R[X]}\mathrm{t}_\mathrm{x} \\ {}_{R[X+1]}^{R[X]}t_y \\ 1 \end{bmatrix}$$

In 2D Rotation:

$${}_{R[X+1]}^{G}\mathrm{R} = \begin{bmatrix} \cos {}_{R[X+1]}^{G}\theta & -\sin {}_{R[X+1]}^{G}\theta \\ \sin {}_{R[X+1]}^{G}\theta & \cos {}_{R[X+1]}^{G}\theta \end{bmatrix} = \begin{bmatrix} \cos {}_{R[X]}^{G}\theta & -\sin {}_{R[X]}^{G}\theta \\ \sin {}_{R[X]}^{G}\theta & \cos {}_{R[X]}^{G}\theta \end{bmatrix} \begin{bmatrix} \cos {}_{R[X+1]}^{R[X]}\theta & -\sin {}_{R[X+1]}^{R[X]}\theta \\ \sin {}_{R[X+1]}^{R[X]}\theta & \cos {}_{R[X+1]}^{R[X]}\theta \end{bmatrix}$$

In 2D Rotation (simple): ${}_{R[X+1]}^{G}\theta = {}_{R[X]}^{G}\theta + {}_{R[X+1]}^{R[X]}\theta$

In ROS: nav_2d_msgs/Pose2DStamped

- First Message at time 97 : G
- Message at time 103 : X
- Next Message at time 107 : X+1

$$ {}^{G}_{R[X+1]}\mathbf{T} = {}^{G}_{R[X]}\mathbf{T} \; {}^{R[X]}_{R[X+1]}\mathbf{T} $$

$$ {}^{R[X]}_{R[X+1]}t_x $$

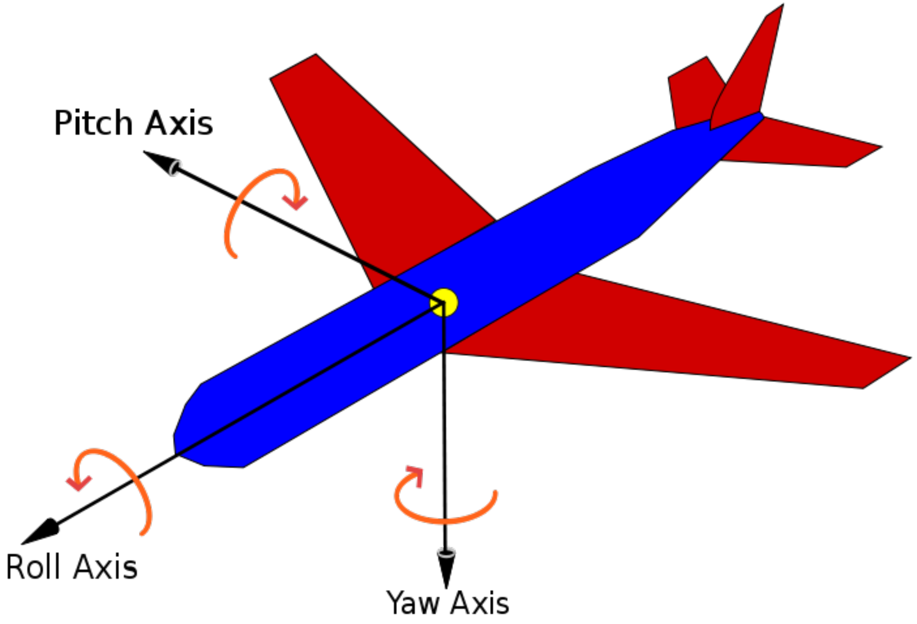
$$ {}^{R[X]}_{R[X+1]}t_y $$

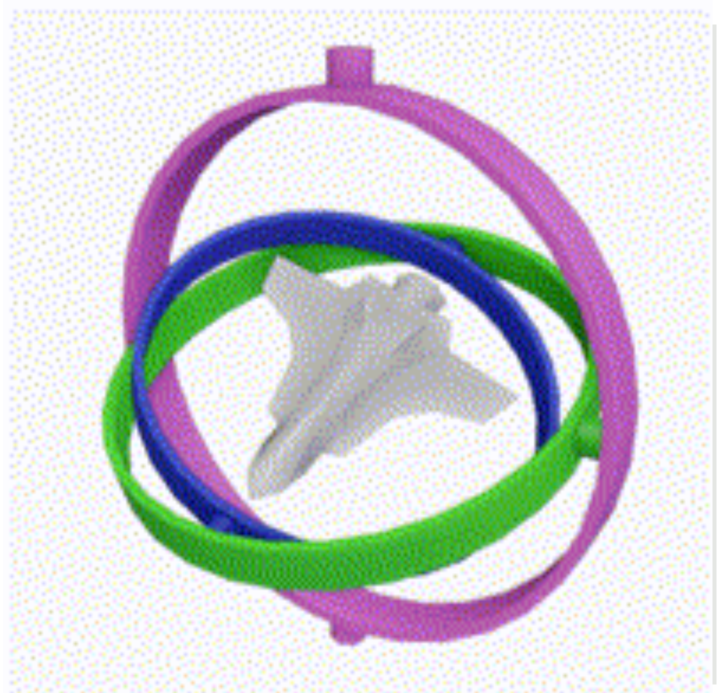
$$ {}^{R[X]}_{R[X+1]}\Theta $$

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose2D pose2D
  float64 x
  float64 y
  float64 theta
```

3D Rotation

- Many 3D rotation representations:
https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
- Euler angles:
  - Roll: rotation around x-axis
  - Pitch: rotation around y-axis
  - Yaw: rotation around z-axis
  - Apply rotations one after the other…
    - => Order important! E.g.:
      - x-z-x; x-y-z; z-y-x; …
  - ☹ Singularities
  - Gimbal lock in Engineering
    - "a condition caused by the collinear alignment of two or more robot axes resulting in unpredictable robot motion and velocities"





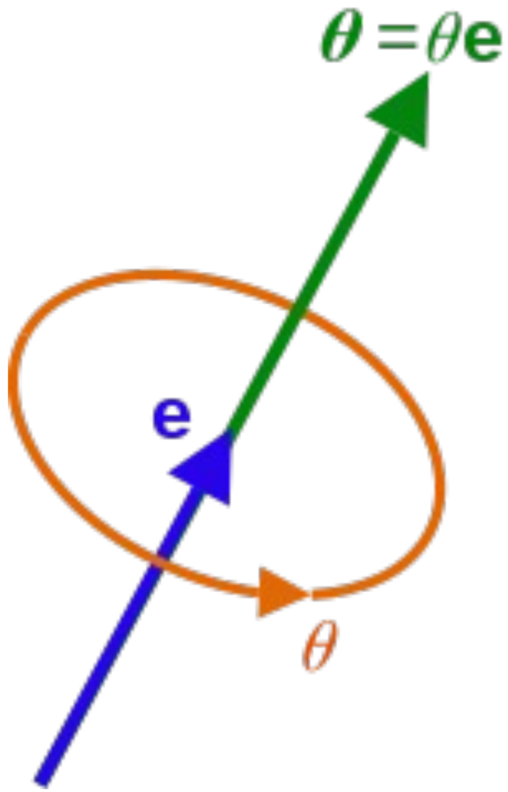
3D Rotation

- Axis Angle
  - Angle θ and
  - Axis unit vector **e** (3D vector with length 1)
    - Can be represented with 2 numbers (e.g. elevation and azimuth angles)

- Euler Angles: sequence of 3 rotations around coordinate axes

equivalent to:

- Axis Angle: pure rotation around a single fixed axis



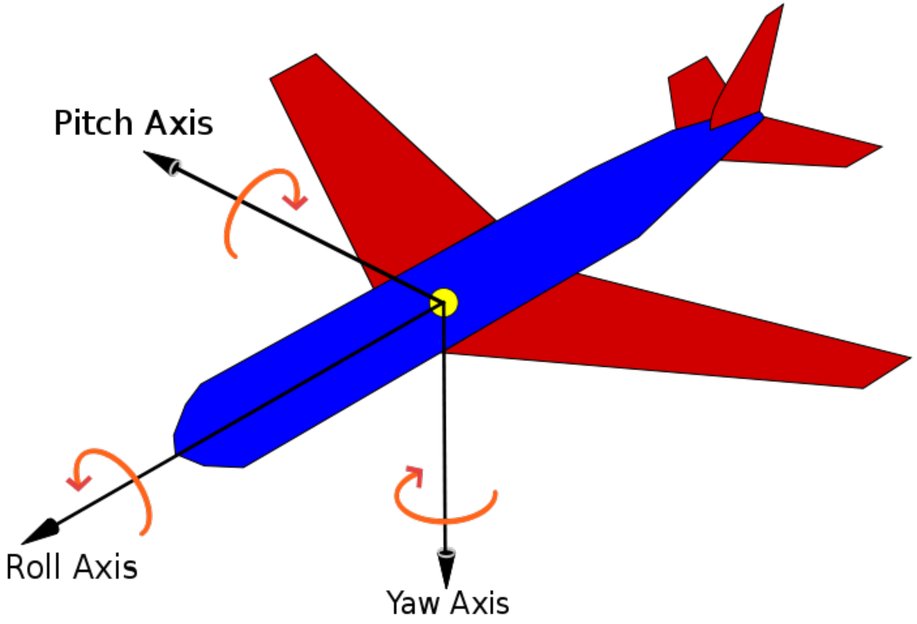
3D Rotation

- Quaternions:
  - Concatenating rotations is computationally faster and numerically more stable
  - Extracting the angle and axis of rotation is simpler
  - Interpolation is more straightforward
  - Unit Quaternion: norm = 1
    - Versor: https://en.wikipedia.org/wiki/Versor

https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation

  - Scalar (real) part: q_0 , sometimes q_w
  - Vector (imaginary) part: q
  - Over determined: 4 variables for 3 DoF (but: unit!)

- Check out: https://eater.net/quaternions ! Excellent interactive video…



$$\check{\mathbf{p}} \equiv p_0 + p_x \mathbf{i} + p_y \mathbf{j} + p_z \mathbf{k}$$

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

$$\check{\mathbf{q}} = \begin{pmatrix} q_0 & q_x & q_y & q_z \end{pmatrix}^\mathsf{T} \equiv \begin{pmatrix} q_0 \\ \mathbf{q} \end{pmatrix}$$

Transform in 3D

Matrix Euler Quaternion

$$ {}^{G}_{A}\mathbf{T} = \begin{bmatrix} {}^{G}_{A}\mathrm{R} & {}^{G}_{A}\mathrm{t} \\ 0_{1x3} & 1 \end{bmatrix} = \begin{pmatrix} {}^{G}_{A}\mathrm{t} \\ {}^{G}_{A}\Theta \end{pmatrix} = \begin{pmatrix} {}^{G}_{A}\mathrm{t} \\ {}^{G}_{A}\check{\mathrm{q}} \end{pmatrix} $$

$$ {}^{G}_{A}\Theta \triangleq (\theta_r, \theta_p, \theta_y)^T $$

In ROS: Quaternions! (w, x, y, z)
Uses Eigen library for Transforms

Rotation Matrix 3x3

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

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

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

$$ R = R_z(\alpha)\, R_y(\beta)\, R_x(\gamma) $$

yaw = α, pitch = β, roll = γ

Eigen

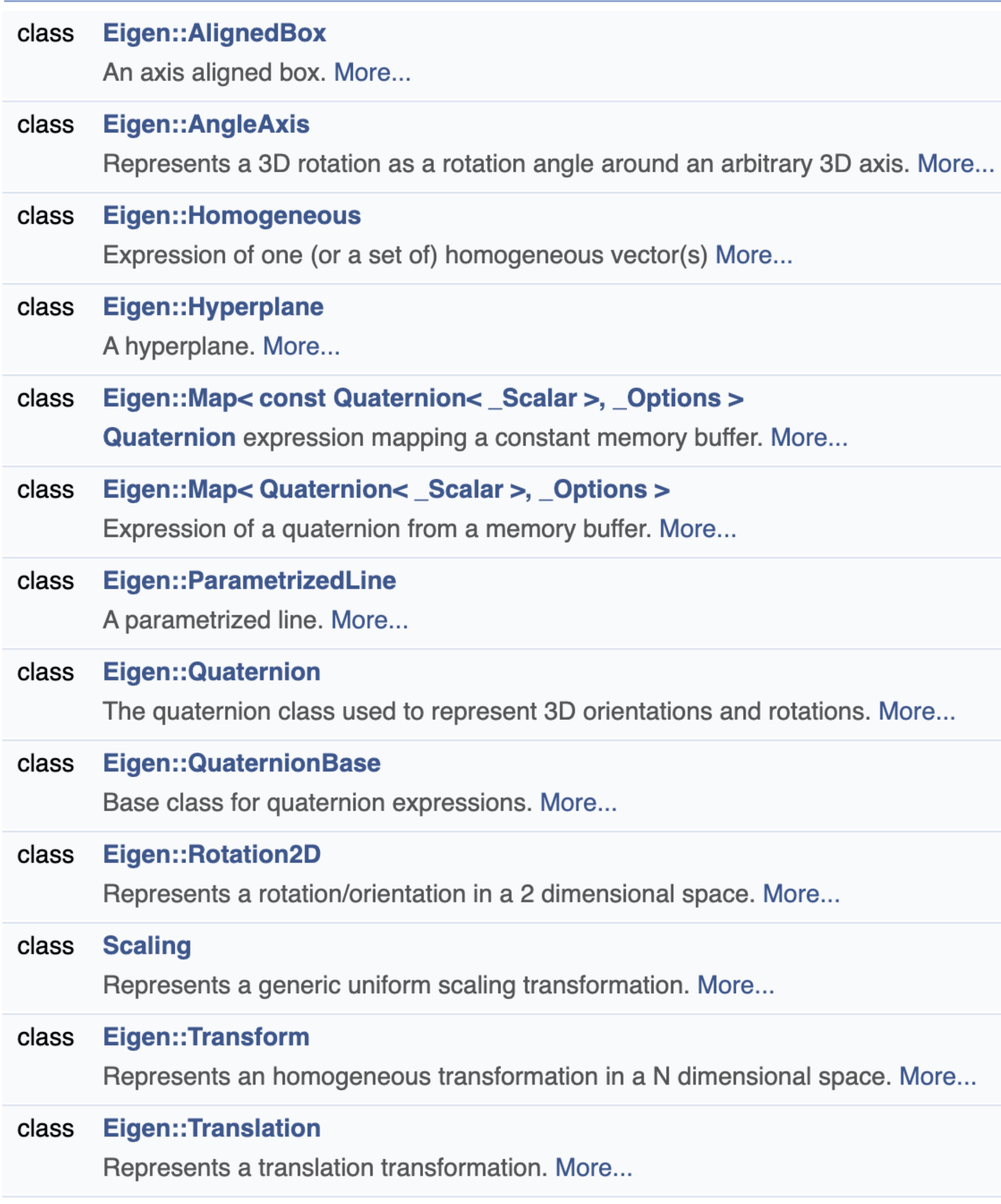
- Don't have to deal with the details of transforms too much ☺

- Conversions between ROS and Eigen:

http://docs.ros.org/noetic/api/eigen_conversions/html/namespacetf.html

```
Matrix3f m;
m = AngleAxisf(angle1, Vector3f::UnitZ())
    * AngleAxisf(angle2, Vector3f::UnitY())
    * AngleAxisf(angle3, Vector3f::UnitZ());
```

https://eigen.tuxfamily.org/dox/group__Geometry__Module.html



Examples of Transforms

- Transform between global coordinate frame and robot frame at time X
- Transform between robot frame at time X and robot frame at time X+1

- Transform between robot camera frame and robot base frame (mounted fixed – not dependend on time! => static transform)
- Transform between map origin and door pose in map (not time dependend)

- Transform between robot camera frame and fingers (end-effector) of a robot arm at time X
- Transform between robot camera frame and map frame at time X
- Transform between robot 1 camera at time X and robot 2 camera at time X+n

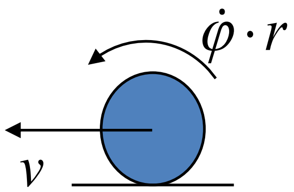
ROS Standards:

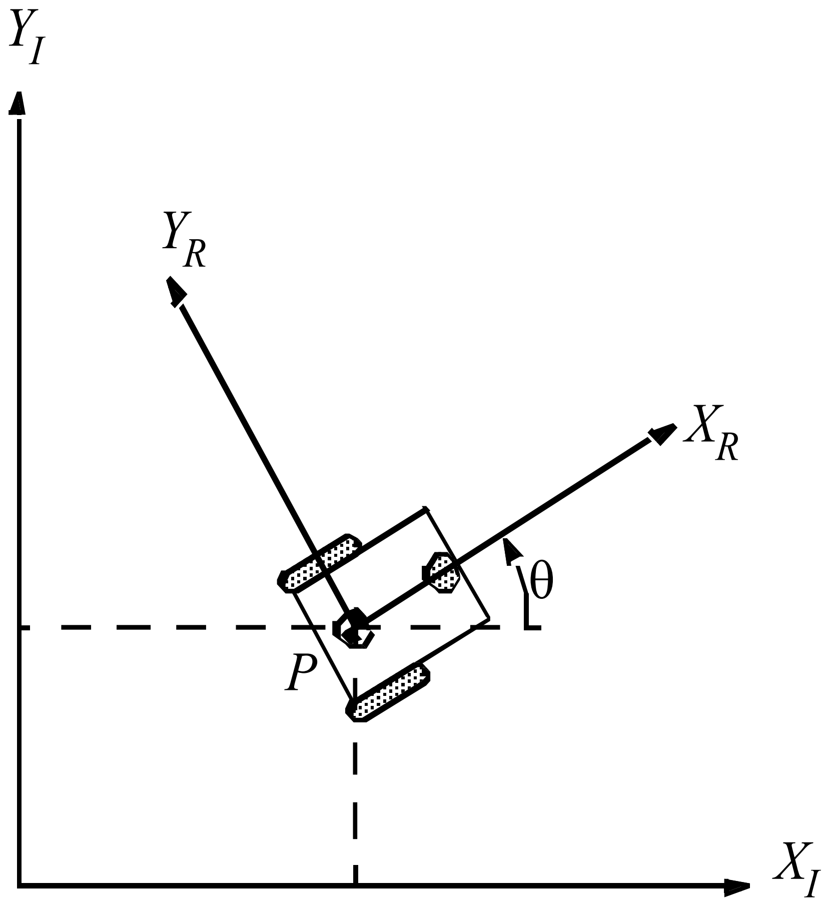
- Standard Units of Measure and Coordinate Conventions
  - http://www.ros.org/reps/rep-0103.html

- Coordinate Frames for Mobile Platforms:
  - http://www.ros.org/reps/rep-0105.html

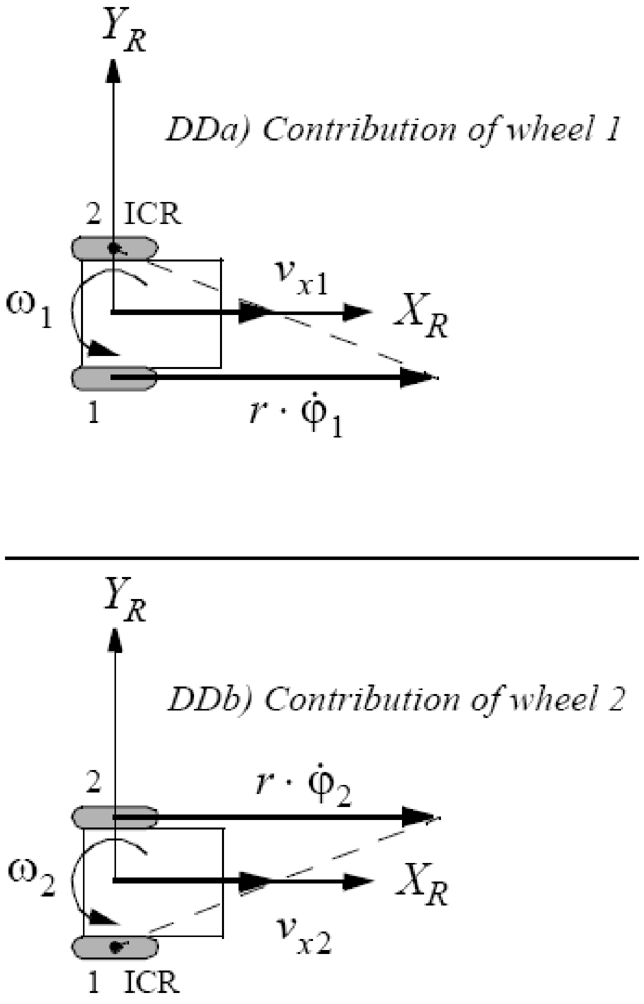
Wheel Kinematic Constraints: Assumptions

- Movement on a horizontal plane
- Point contact of the wheels
- Wheels not deformable
- Pure rolling
  - $v_c = 0$ at contact point
- No slipping, skidding or sliding
- No friction for rotation around contact point
- Steering axes orthogonal to the surface
- Wheels connected by rigid frame (chassis)





Forward Kinematic Model: Geometric Approach



Differential-Drive:

DDa) $v_{x1} = \frac{1}{2} r\dot{\varphi}_1 \quad ; \quad v_{y1} = 0 \quad ; \quad \omega_1 = \frac{1}{2l} r\dot{\varphi}_1$

DDb) $v_{x2} = \frac{1}{2} r\dot{\varphi}_2 \quad ; \quad v_{y2} = 0 \quad ; \quad \omega_2 = -\frac{1}{2l} r\dot{\varphi}_2$

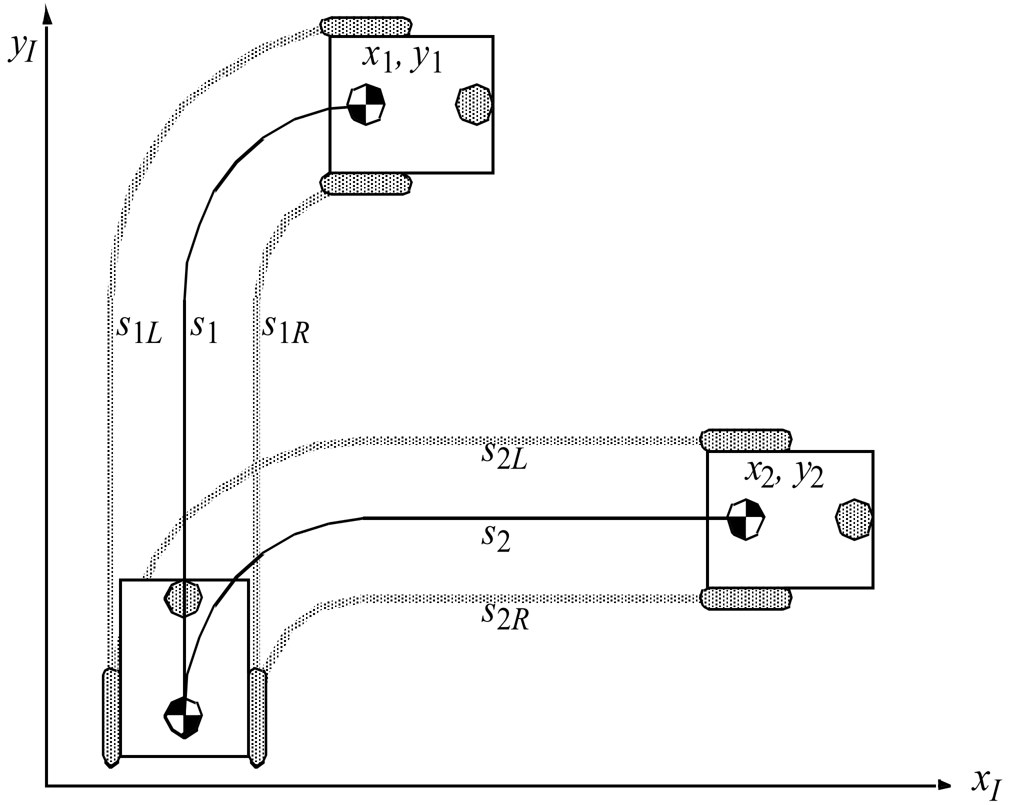
$$-> \dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_I = R(\theta)^{-1} \begin{bmatrix} v_{x1} + v_{x2} \\ v_{y1} + v_{y2} \\ \omega_1 + \omega_2 \end{bmatrix} = R(\theta)^{-1} \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ \frac{r}{2l} & -\frac{r}{2l} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_1 \\ \dot{\varphi}_2 \end{bmatrix}$$

Inverse of R => Active and Passive Transform:
http://en.wikipedia.org/wiki/Active_and_passive_transformation

Mobile Robot Kinematics: Non-Holonomic Systems

$s_1 = s_2$; $s_{1R} = s_{2R}$; $s_{1L} = s_{2L}$

but: $x_1 \neq x_2$; $y_1 \neq y_2$



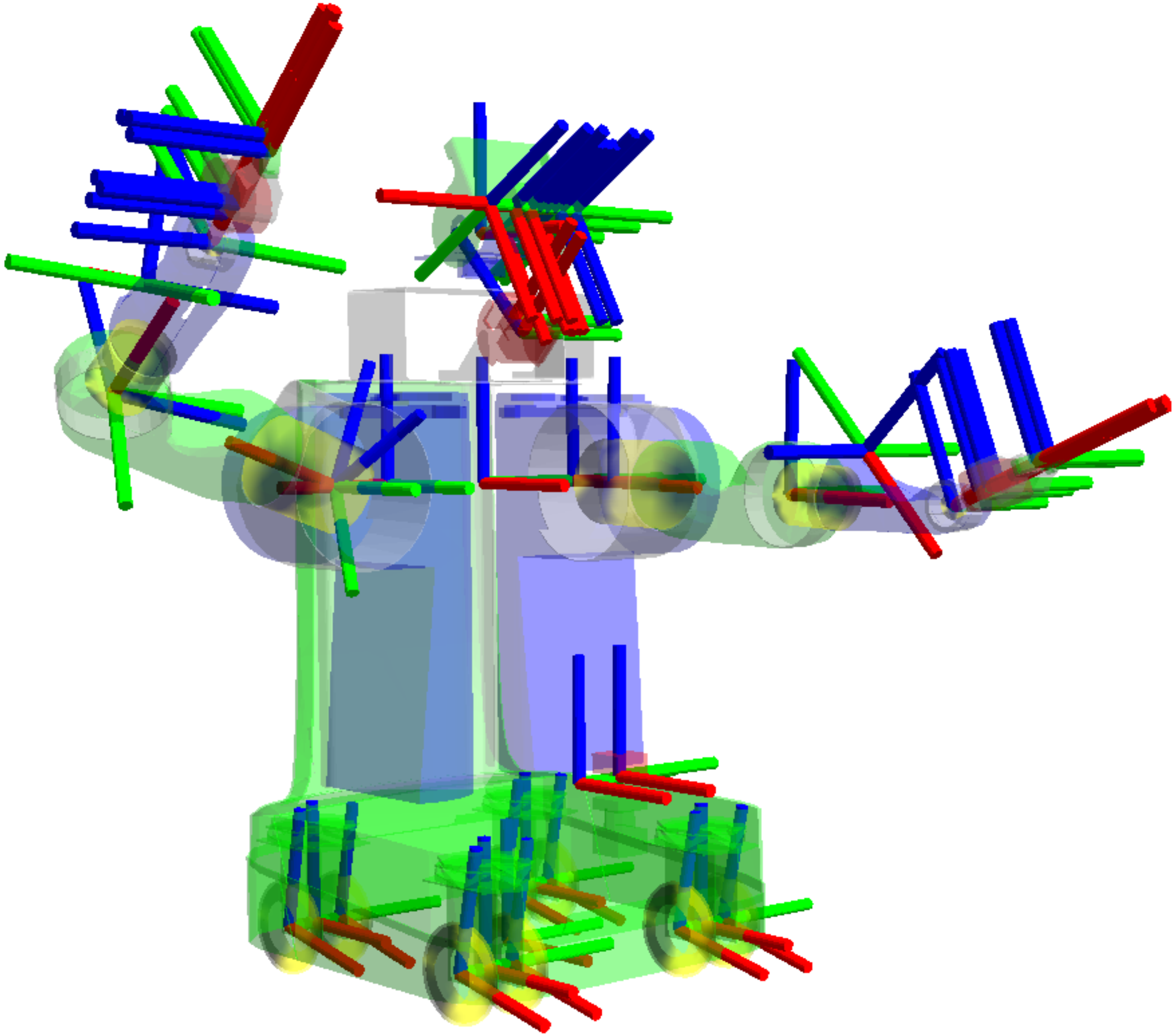
- Non-holonomic systems
  - differential equations are not integrable to the final pose.
  - the measure of the traveled distance of each wheel is not sufficient to calculate the final position of the robot. One has also to know how this movement was executed as a function of time.

Holonomic examples

Uranus, CMU

ROS: 3D Transforms : TF

- http://wiki.ros.org/tf
- http://wiki.ros.org/tf/Tutorials



ROS geometry_msgs/TransformStamped

- ${}^{\text{header.frame\_id[header.stamp]}}\mathbf{T}_{\text{child\_frame\_id[header.stamp]}}$

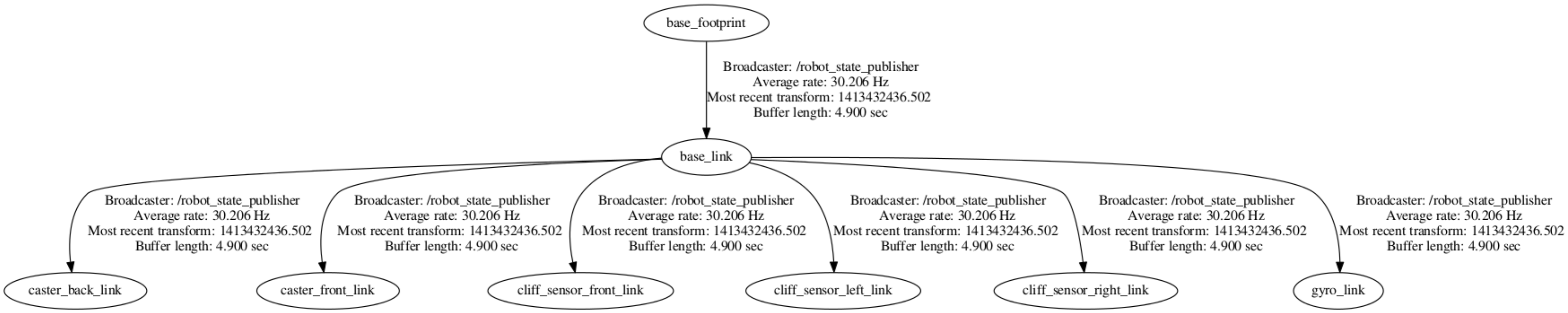
- Transform between header (time and reference frame) and child_frame
- 3D Transform representation:
  - geometry_msgs/Transform:
    - Vector3 for translation (position)
    - Quaternion for rotation (orientation)

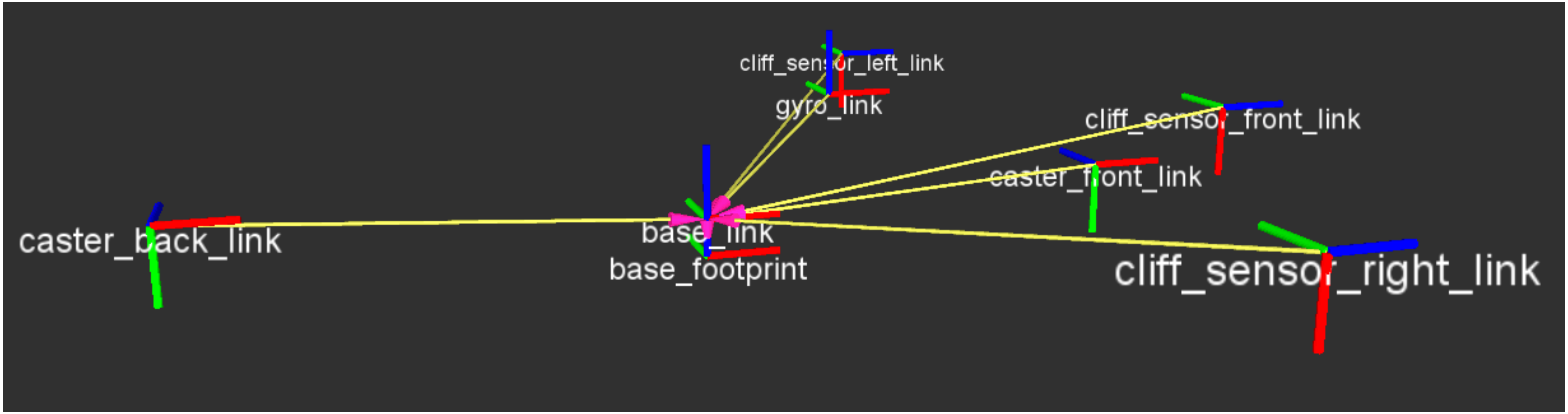
```
rosmsg show geometry_msgs/TransformStamped

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
```

ROS tf2_msgs/TFMessage

- An array of TransformStamped
- Transforms form a tree
- Transform listener: traverse the tree
  - tf::TransformListener listener;
- Get transform:
  - tf::StampedTransform transform;
  - listener.lookupTransform("/base_link", "/camera1", ros::Time(0), transform);
  - ros::Time(0): get the latest transform
  - Will calculate transform by chaining intermediate transforms, if needed

```
rosmsg show 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
```


base_footprint
Broadcaster: /robot_state_publisher
Average rate: 30.206 Hz
Most recent transform: 1413432436.502
Buffer length: 4.900 sec
base_link
Broadcaster: /robot_state_publisher
Average rate: 30.206 Hz
Most recent transform: 1413432436.502
Buffer length: 4.900 sec
caster_back_link
caster_front_link
cliff_sensor_front_link
cliff_sensor_left_link
cliff_sensor_right_link
gyro_link


cliff_sensor_left_link
gyro_link
cliff_sensor_front_link
caster_front_link
caster_back_link
base_link
base_footprint
cliff_sensor_right_link

Transforms in ROS

- Imagine: Object recognition took 3 seconds – it found an object with:
  - tf::Transform object_transform_camera; // ${}^{\mathrm{Cam}[X]}_{Obj}\mathbf{T}$ (has tf::Vector3 and tf::Quaternion)
  - and header with: ros::Time stamp; // Timestamp of the camera image (== X)
    - and std::string frame_id; // Name of the frame (“Cam”)
- Where is the object in the global frame (= odom frame) “odom” ${}^{\mathrm{G}}_{Obj}\mathbf{T}$?
  - tf::StampedTransform object_transform_global; // the resulting frame
  - listener.lookupTransform(child_frame_id, “/odom", header.stamp, object_transform_global);
- tf::TransformListener keeps a history of transforms – by default 10 seconds

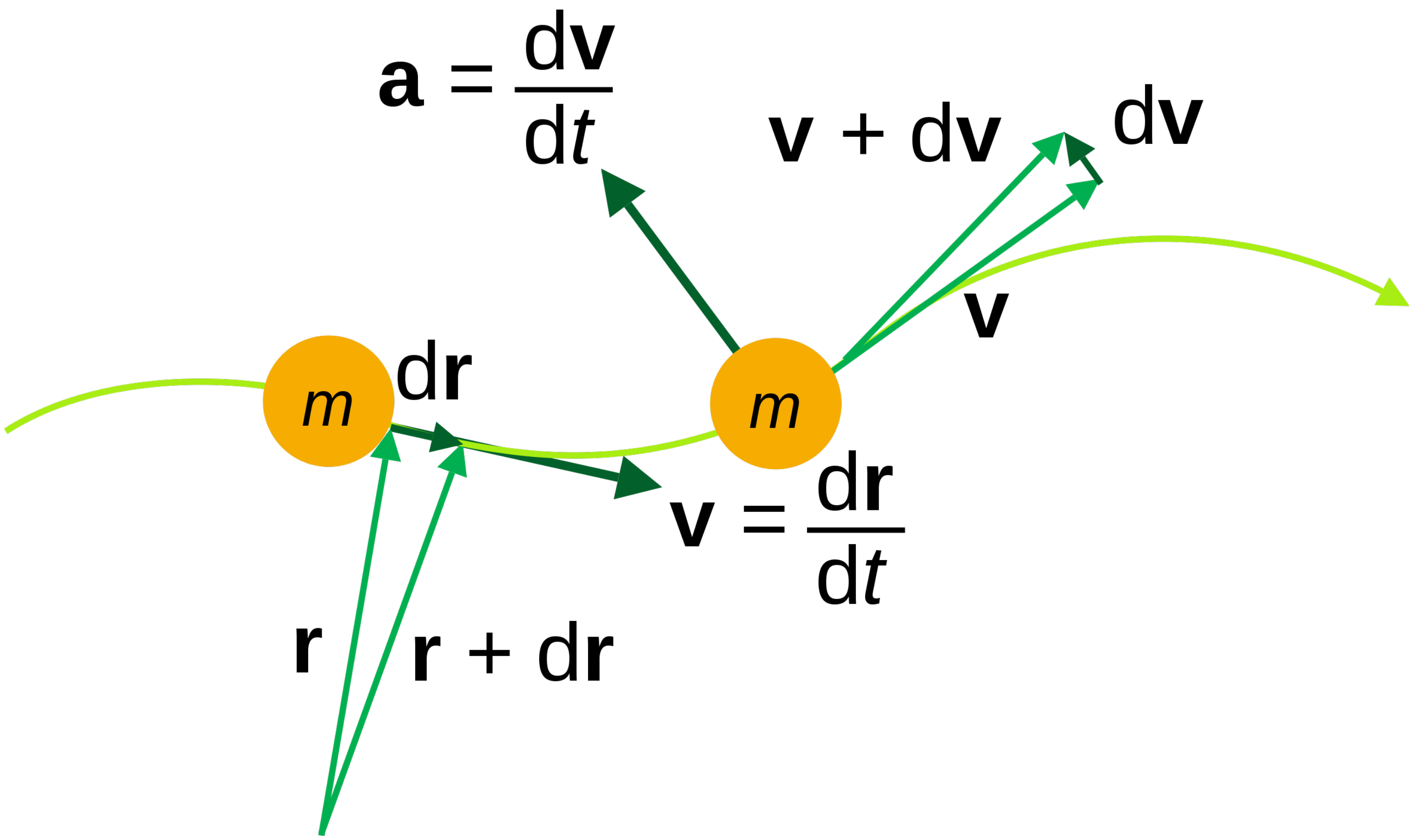
KINEMATICS

What are kinematics?

- Describes the motion of points, bodies (objects), and systems of objects
  - Does not consider the forces that cause them (that would be kinetics)
  - Also known as “the geometry of motion”

- For manipulators
  - Describes the motion of the arm
  - Puts position/ angle and their rate of change (speed) of joints in relation with 3D pose of points on the arm, especially tool center point (tcp, end effector)

What are kinematics?

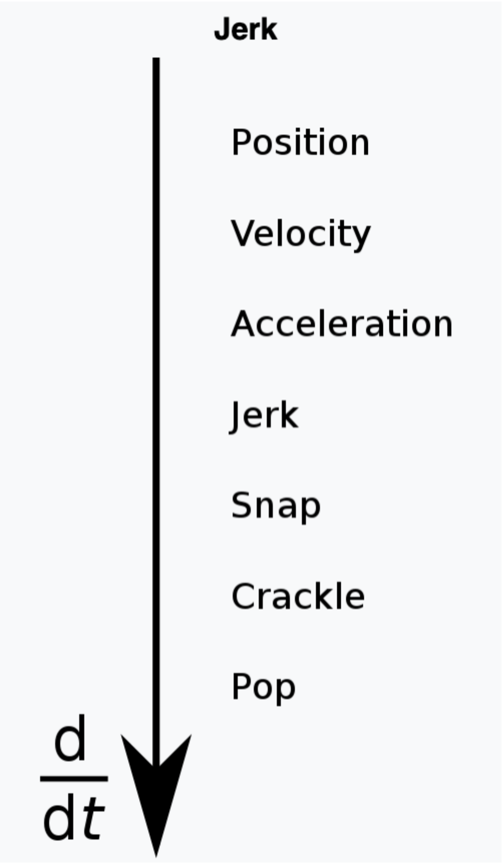


What are kinematics?

- It does not stop at acceleration, but theory involves an arbitrarily high number of derivatives:

$$\mathbf{x}(t) = \mathbf{x}_0 + \mathbf{v}_0 t + \frac{1}{2}\mathbf{a}_0 t^2 + \frac{1}{3}\mathbf{j} t^3 + \ldots$$

$$\boldsymbol{\theta}(t) = \boldsymbol{\theta}_0 + \boldsymbol{\omega}_0 t + \frac{1}{2}\boldsymbol{\alpha}_0 t^2 + \frac{1}{3}\zeta t^3 + \ldots$$



Jerk equations: minimal setting for solutions showing chaotic behavior!

In practice

- Often we use finite models to simplify/smoothify the system
  - Locally constant acceleration

$$\mathbf{x}(t) = \mathbf{x}_0 + \mathbf{v}_0 t + \frac{1}{2}\mathbf{a}_0 t^2$$

  - Locally constant velocity

$$\mathbf{x}(t) = \mathbf{x}_0 + \mathbf{v}_0 t$$

Why do we want to introduce kinematic models

- For control
  - E.g.: Knowledge of how the system is moving is beneficial for reaching the goal pose
- For prediction
  - E.g.: If we have an initial estimate, we can use a kinematic model to generate a prior pose at a later point
- For smoothness
  - E.g.: If we estimate poses, we may constrain their difference to be consistent with some prior or measured velocity
- To impose constraints
  - E.g.: The motion may be more specific and include kinematic constraints

Two types of kinematic constraints

- Holonomic
  - The total number of degrees of freedom is equal to the controllable number of degrees of freedom
  - E.g.: slider on rail (1 DoF)

- Non-holonomic
  - The total number of degrees of freedom is higher than the controllable number of degrees of freedom
  - E.g: A passenger vehicle (3 DoF, while controllable DoF is only 2)

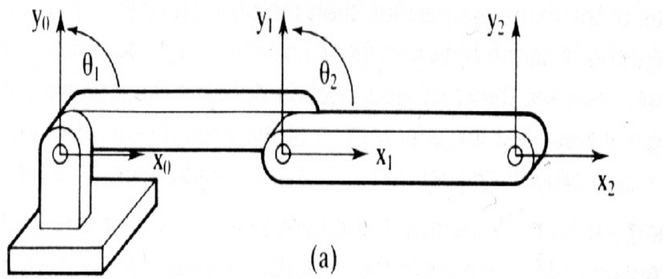
KINEMATIC REPRESENTATIONS

Robot Arm

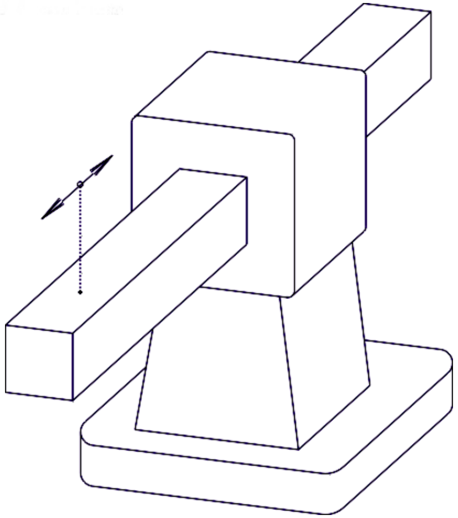
- Consists of Joints and Links …
- and a Base and a Tool (or End-Effector or Tip)

Joints

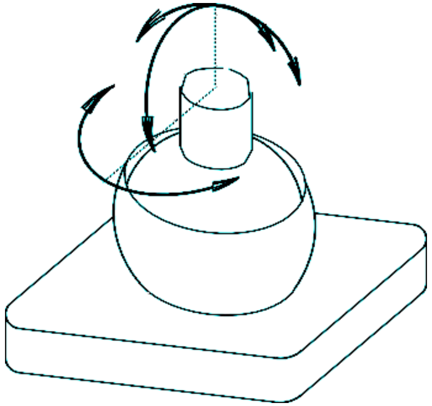
- Revolute Joint: 1DOF



- Prismatic Joint/ Linear Joint: 1DOF



- Spherical Joint: 3DOF

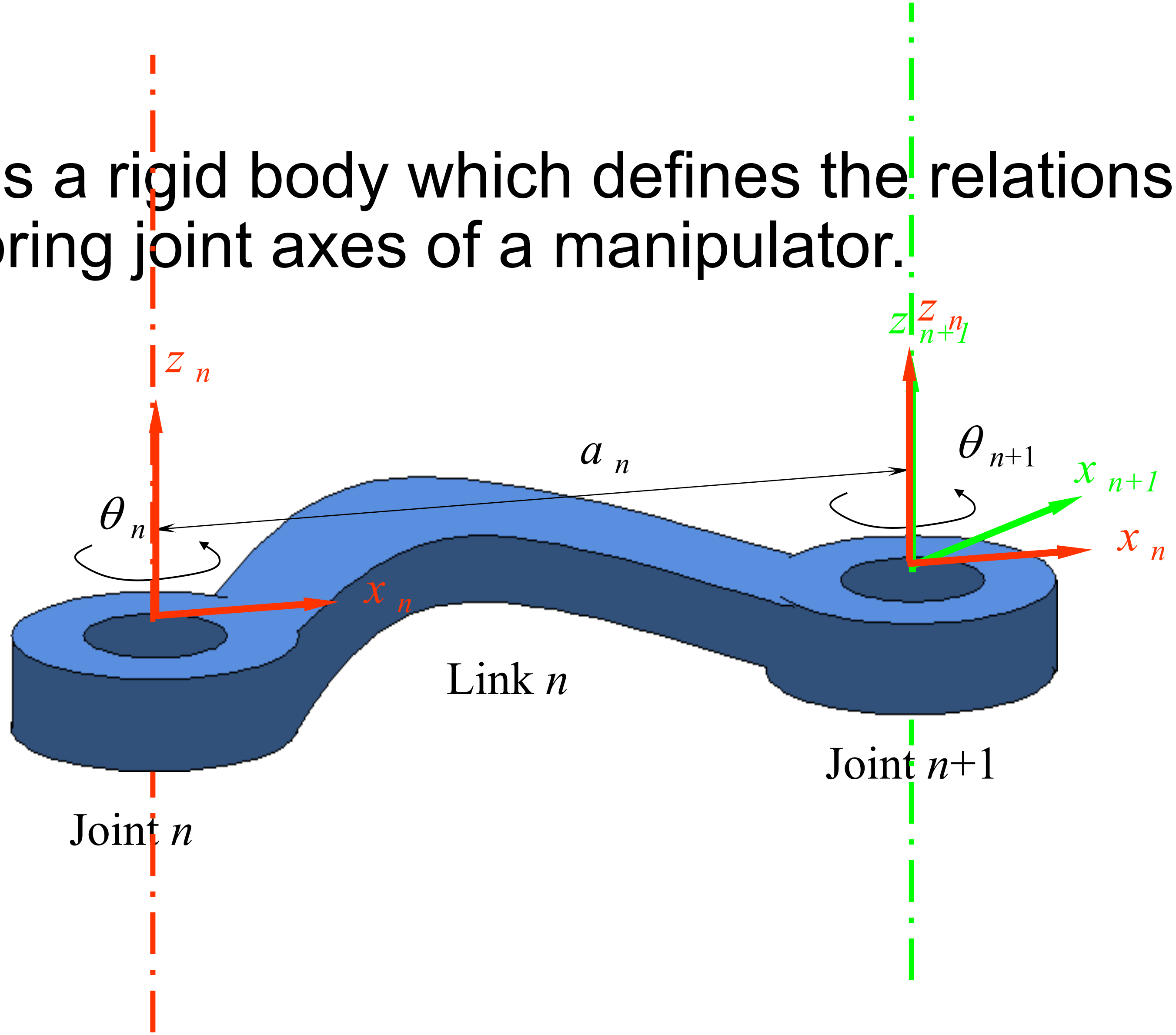


Note on Joints

- Without loss of generality, we will consider only manipulators which have joints with a single degree of freedom.
- A joint having n degrees of freedom can be modeled as n joints of one degree of freedom connected with n-1 links of zero length.
- We could use 6-DoF Transforms to describe their 3D relation – but we can do better (fewer variables):

Link

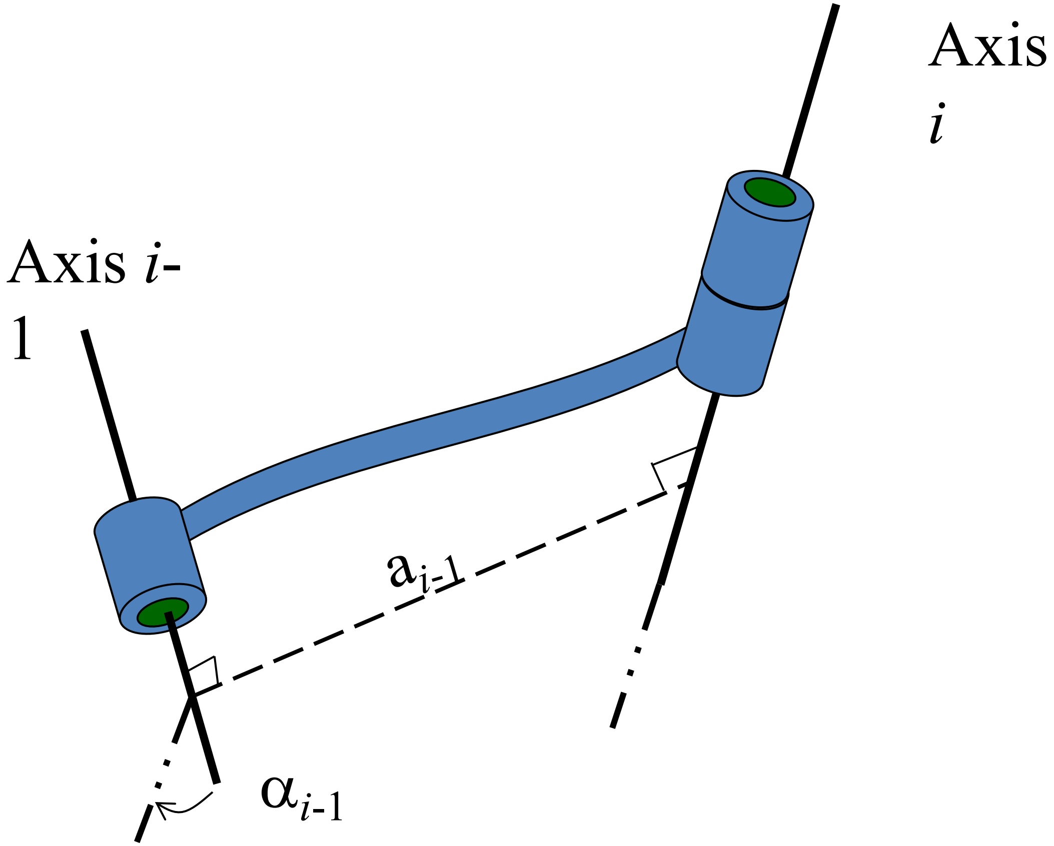
- A link is considered as a rigid body which defines the relationship between two neighboring joint axes of a manipulator.



The Kinematics Function of a Link

- The kinematics function of a link is to maintain a fixed relationship between the two joint axes it supports.

- This relationship can be described with two parameters: the link length a, the link twist a



Link Length

- Is measured along a line which is **mutually perpendicular** to **both** axes.

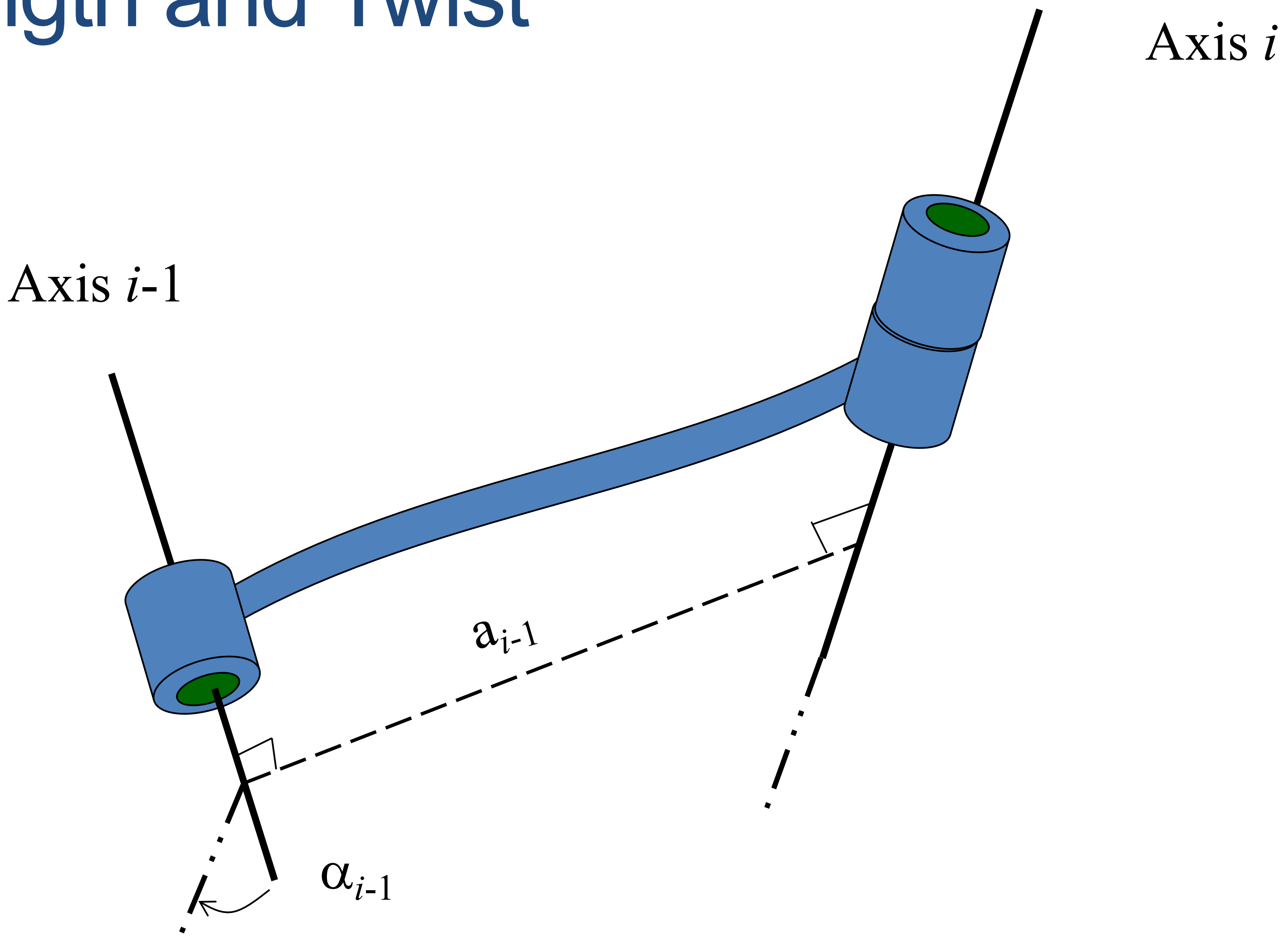
- The mutually perpendicular always exists and is unique except when both axes are parallel.

Link Twist

- Project both axes i-1 and i onto the plane whose normal is the mutually perpendicular line, and measure the angle between them

- Right-hand coordinate system

Link Length and Twist

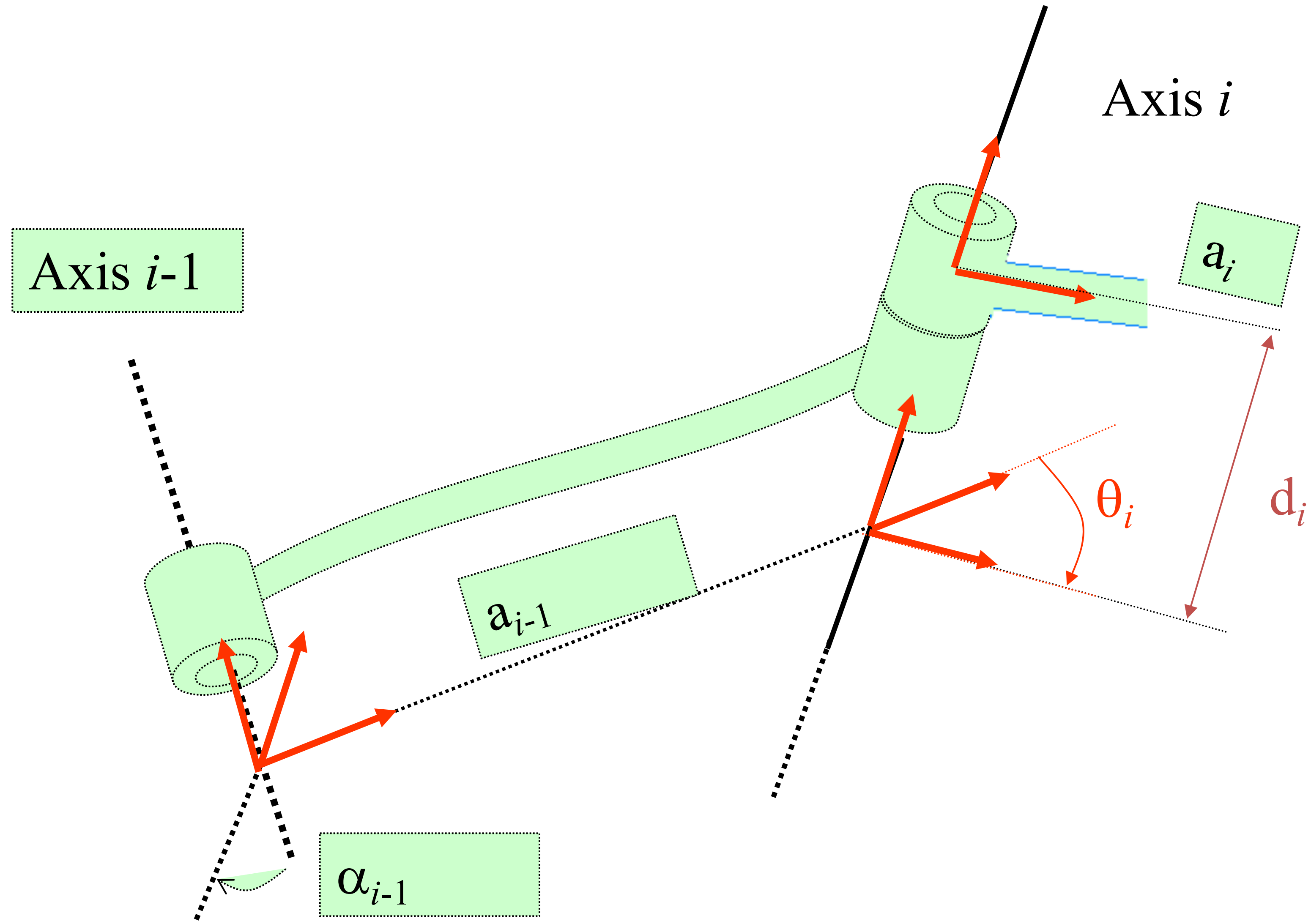


Joint Parameters (the Denavit-Hartenberg (DH) Parameters)

A joint axis is established at the connection of two links. This joint will have two normals connected to it - one for each of the links.

- The relative position of two links is called *link offset* d_n which is the distance between the links (the displacement, along the joint axes between the links).
- The *joint angle* θ_n between the normals is measured in a plane normal to the joint axis.

Link and Joint Parameters



Link and Joint Parameters

4 parameters are associated with each link. You can align the two axis using these parameters.

- Link parameters:

a_n the length of the link.

α_n the twist angle between the joint axes.

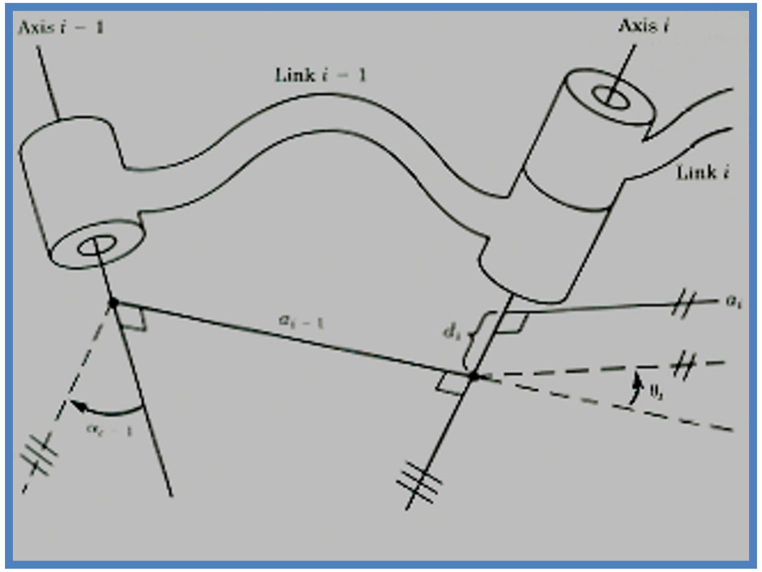
- Joint parameters:

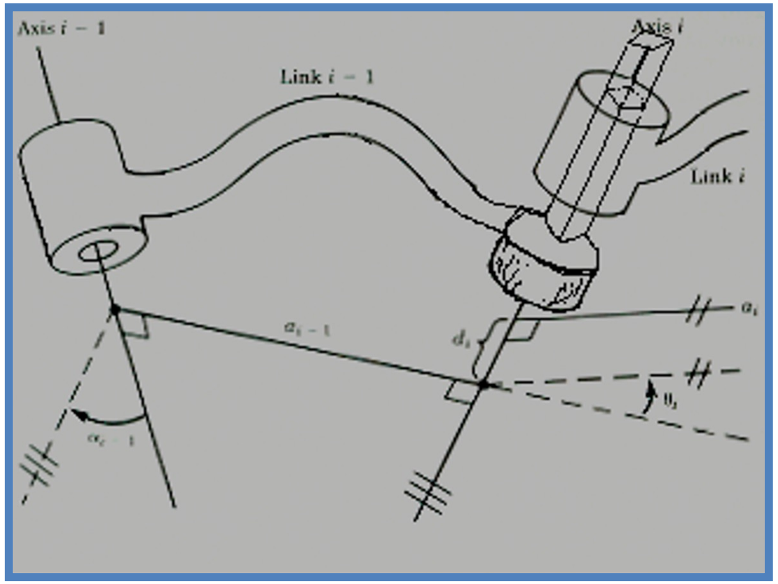
θ_n the angle between the links.

d_n the distance between the links

Link Connection Description:

For Revolute Joints: a, α, and d. are all fixed, then "θ_i" is the. Joint Variable.





For Prismatic Joints: a, α, and θ. are all fixed, then "d_i" is the. Joint Variable.

These four parameters: (Link-Length a_{i-1}), (Link-Twist α_{i-1}), (Link-Offset d_i), (Joint-Angle θ_i) are known as the *Denavit-Hartenberg Link Parameters*.

Transforms vs. DH Parameters

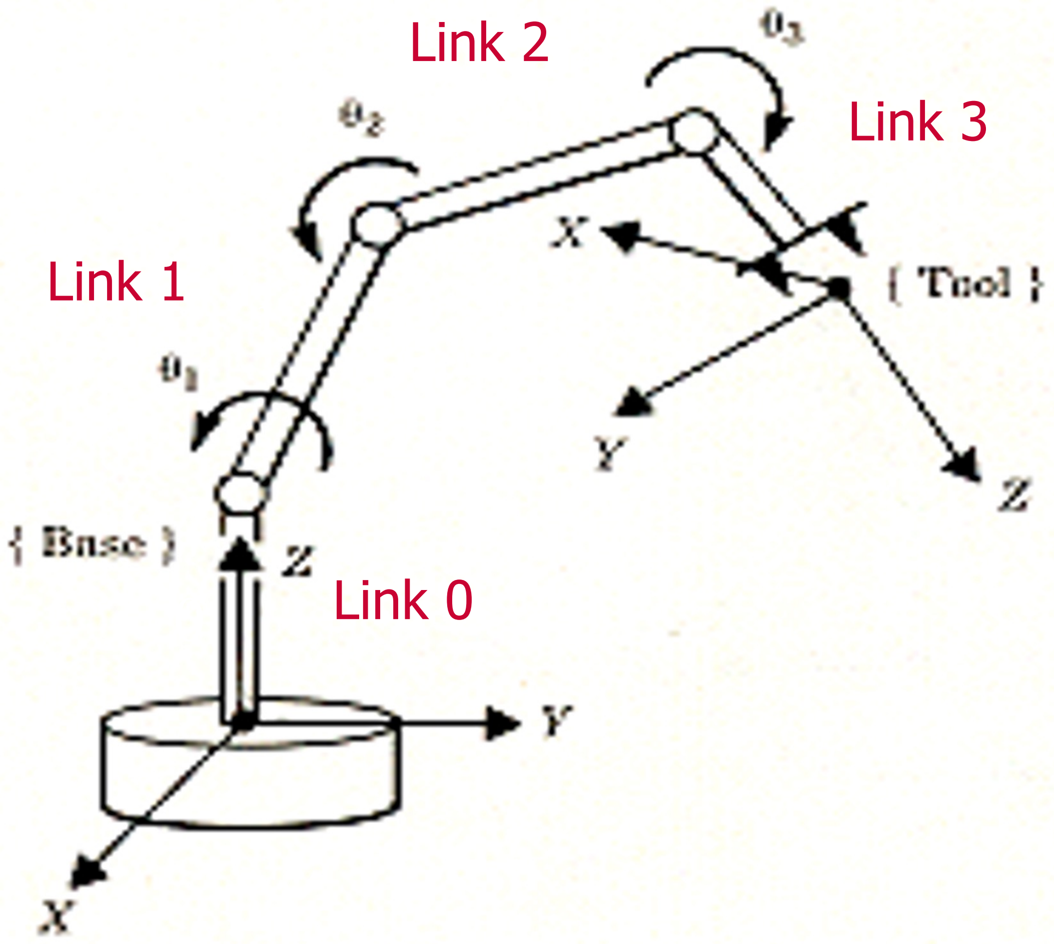
- We could represent an arm using 6 DoF Transforms (ROS MoveIt! is doing this…)

- But:
  General Transform: 6 DoF => DH 4: DoF more efficient reperesentation. E.g. Jacobians for IK will have lots of 0's in DH

- The 2 missing DOFs don't disappear — they're *eliminated by convention*: the DH frame assignment rules fix them, leaving only 4 parameters per link to describe everything you need.
  - E.g. revolute joint: can only rotate around one axis => 2 rotations are not needed!

Links Numbering Convention

Base of the arm:	**Link-0**
1^{st} moving link:	**Link-1**
.	**.**
.	**.**
.	**.**
Last moving link:	**Link-n**



A 3-DOF Manipulator Arm

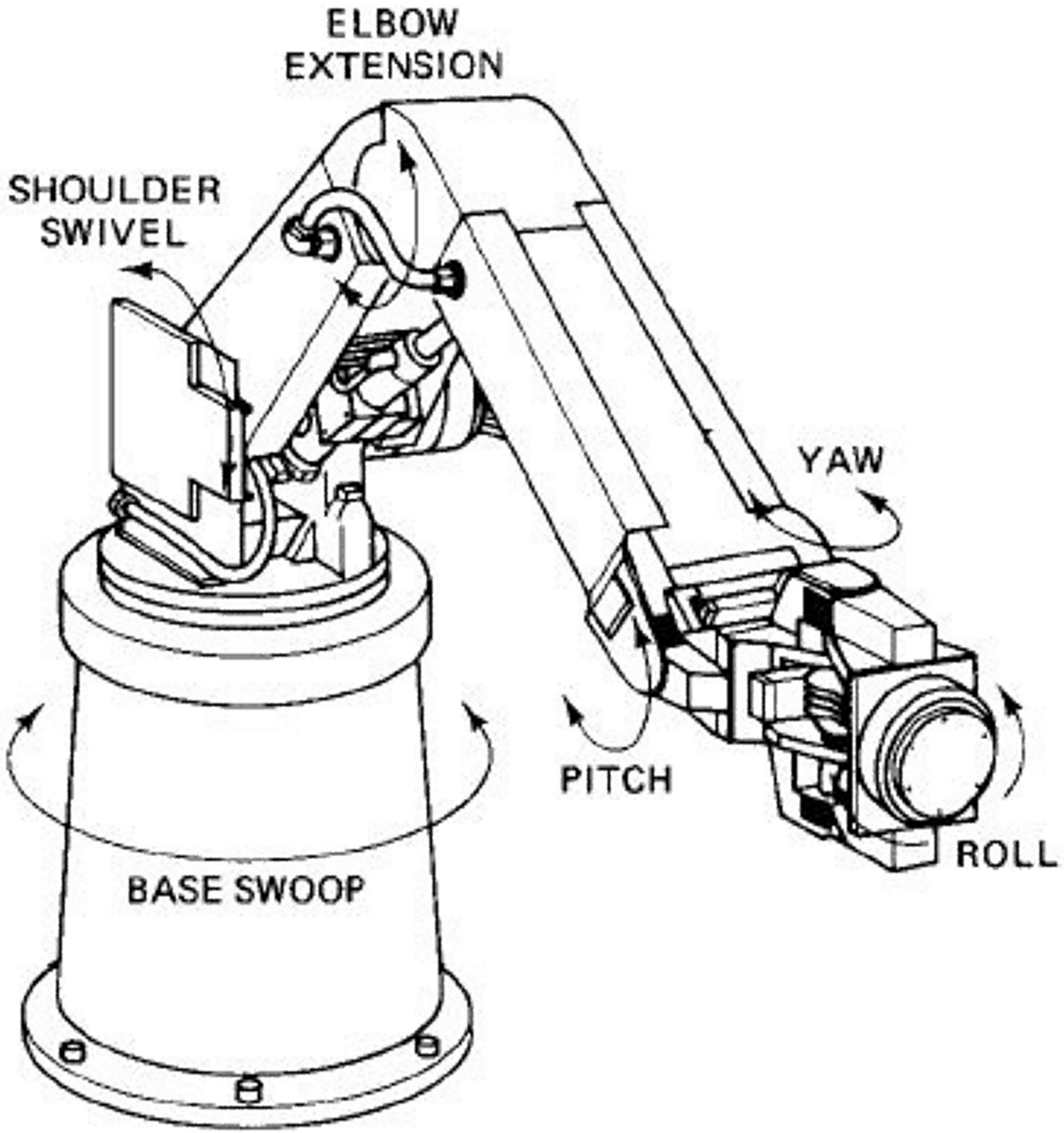
First and Last Links in the Chain

- $a_0 = \alpha_{n.} = 0$
- $\alpha_0 = \alpha_{n.} = 0$
- If joint 1 is revolute: d_0 is fixed and θ_1 is arbitrary
- If joint 1 is prismatic: d_0= arbitrary and θ_1 is fixed

Robot Specifications

Number of axes

- Major axes, (1-3) => position the wrist
- Minor axes, (4-6) => orient the tool
- Redundant, (7-n) => reaching around obstacles, avoiding undesirable configuration



Example: Puma 500

Unimate Puma 500
Oussama Khatib | Used with permission

θ_j	d_j	a_j	α_j
q1	0.0000	0.0000	$\pi/2$
q2	0.0000	0.4318	0
q3	0.1500	0.0203	$-\pi/2$
q4	0.4318	0.0000	$\pi/2$
q5	0.0000	0.0000	$-\pi/2$
q6	0.0000	0.0000	0

Frames

- Choose the base and tool coordinate frame
  - Make your life easy!
- Several conventions
  - Denavit Hartenberg (DH), modified DH, Hayati, etc.