Humanoid Robotics

Locomotion 1: Kinematics and Dynamics

Maren Bennewitz, Shahram Khorshidi

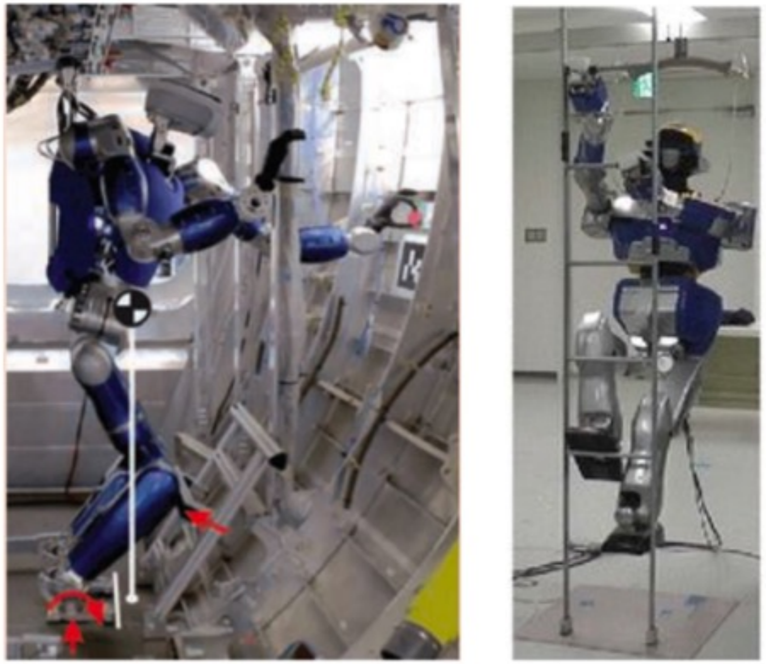
HRL

UNIVERSITÄT BONN

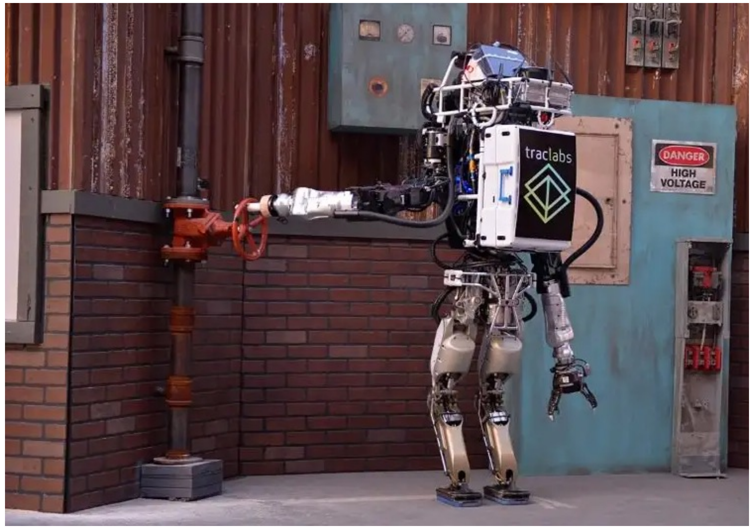
Goals of This Lecture

- Understand **quaternion** representations of 3D orientation
- Know how to compute **forward kinematics** in **legged robots**
- Learn about the kinematics and dynamics of **floating-base robots**
- Know about **efficient algorithms** and software tools for computing robot **kinematics** and **dynamics**

Motivation: Why Legged Robots?

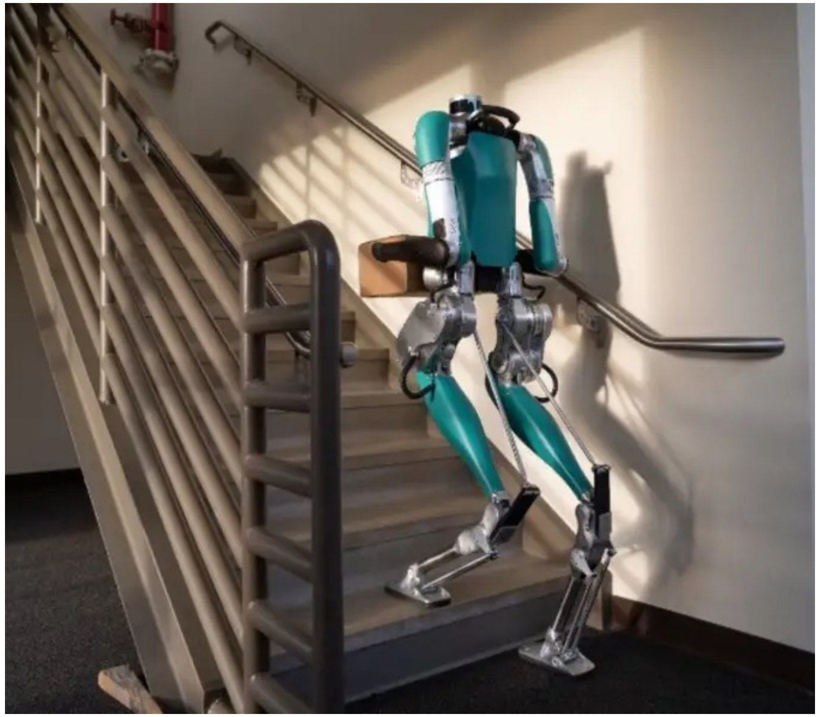


Vaillant et al.,
Autonomous Robots, 2016



DARPA GRAND
Challenge, 2015

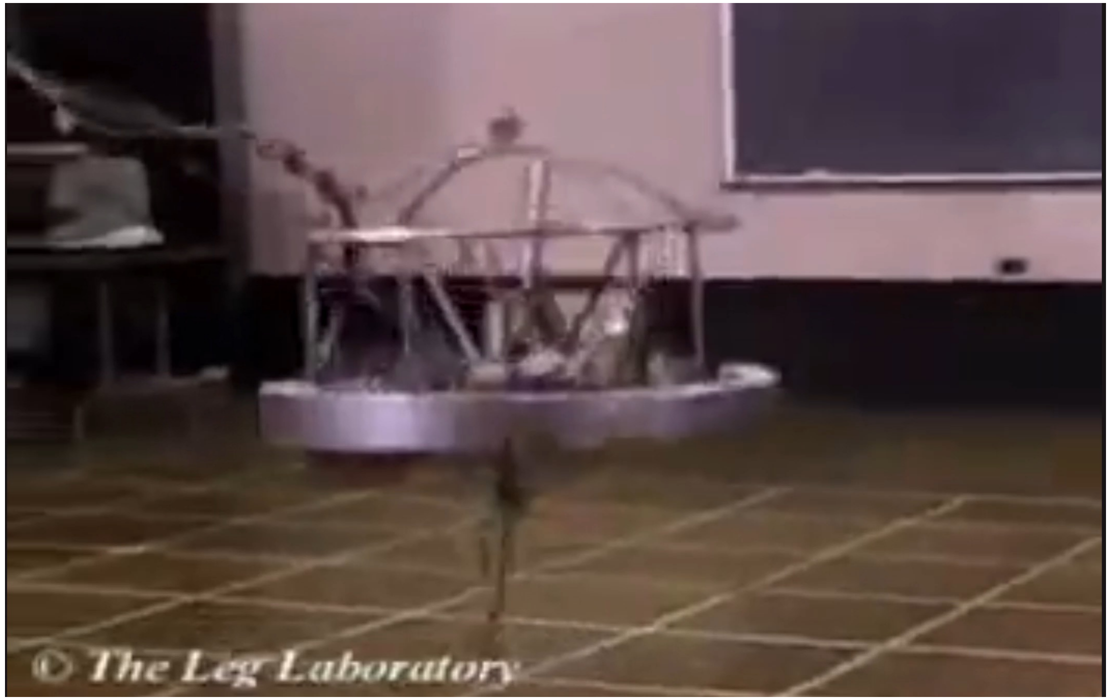
ANYbotics, ANYmal



Agility Robotics

Historical Context (1980s)

- Simple dynamics
- **Closed-form** solutions and **heuristics**
- **Dynamic balance** achieved through active control (torques, contact forces), not just static stability



MIT, Leg Lab Laboratory

Historical Context (2001)

- **Passive** walkers: no actuation, just using gravity
- Study the dynamics of walking (stability, limit cycles)

YouTube: Passive Dynamic Walking Robot, Steve Collins, Cornell University

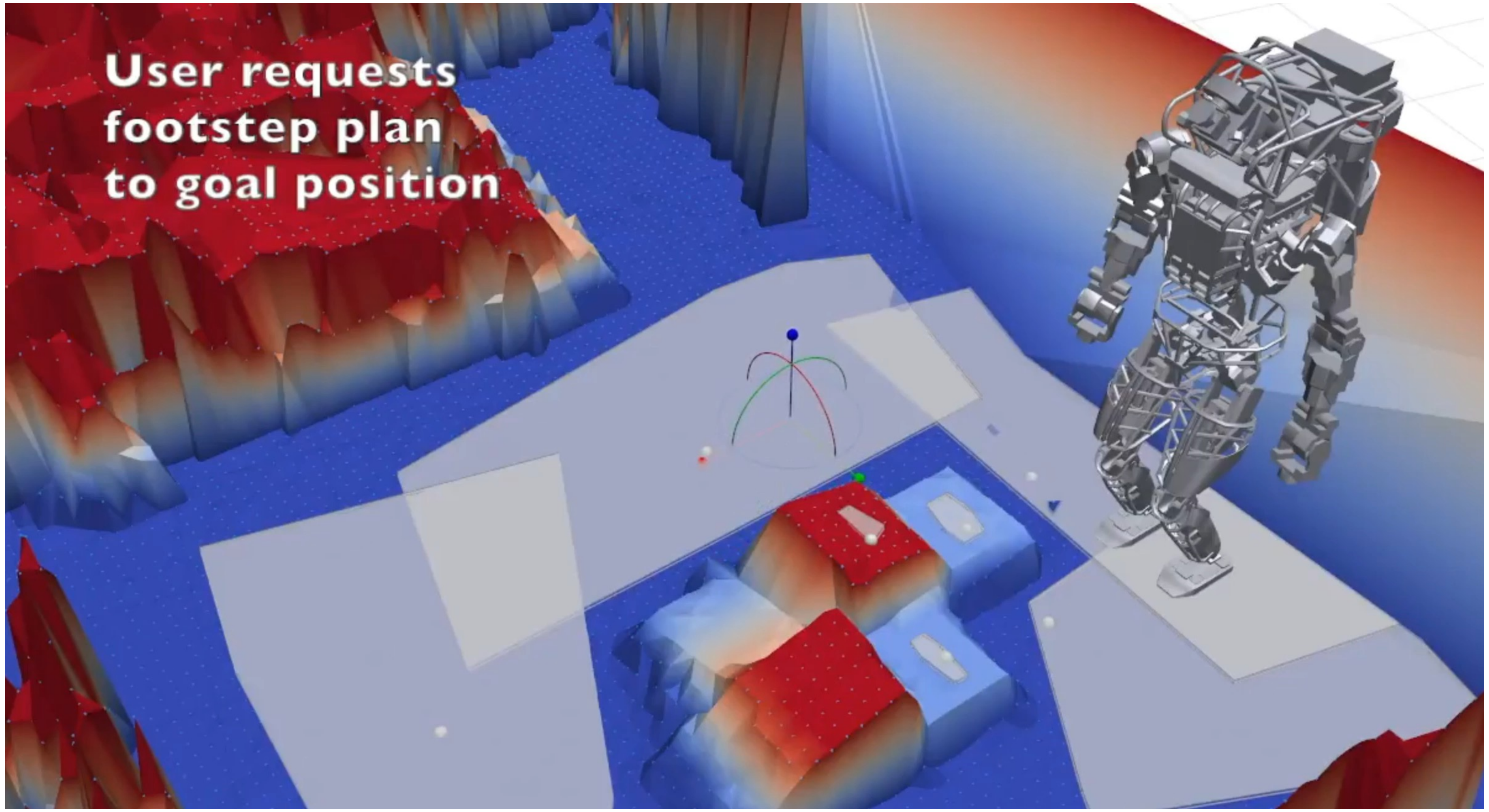
Historical Context (2012)

- **Simplified models** for walking
- Restricted to flat terrain, prioritize **precise trajectory tracking** over dynamic adaptability

Asimo, Honda. YouTube: Honda Unveils All-New ASIMO Humanoid Robot

Historical Context (2014)

- **Convex optimization** for planning
- Footstep and motion planning



Deits&Tedrake, "Footstep Planning on Uneven Terrain with Mixed-Integer Convex Optimization”, Humanoids, 2014

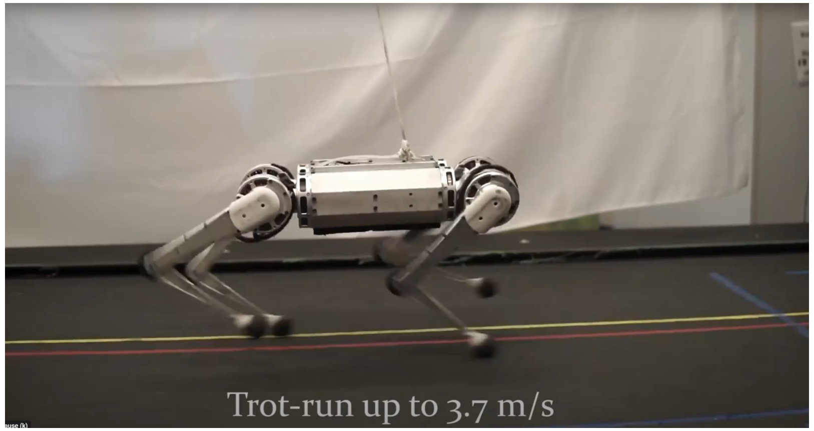
Historical Context (2015)

- DARPA Grand Challenge
- Interaction with the environment **through contact** is a **difficult task** for **legged robots**

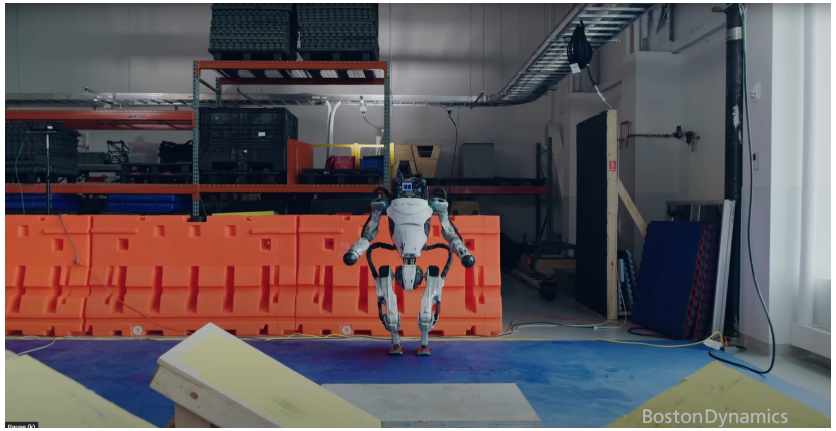
DARPA Robotics Challenge (DRC), 2015

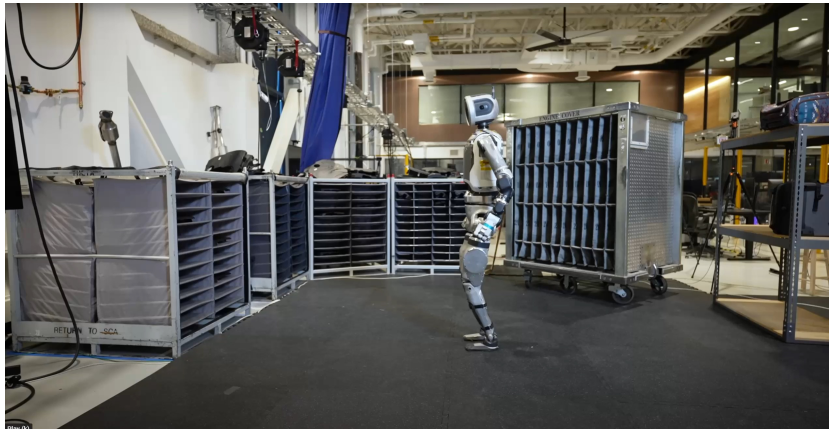
Historical Context (Last Decade)

- Advancements in hardware (high torque-density actuation)
- High performance computers
- Online planning with **model predictive control** (**MPC**)
- Use robot model to predict future behavior and optimize control inputs over a time horizon



Kim et al.m 2019



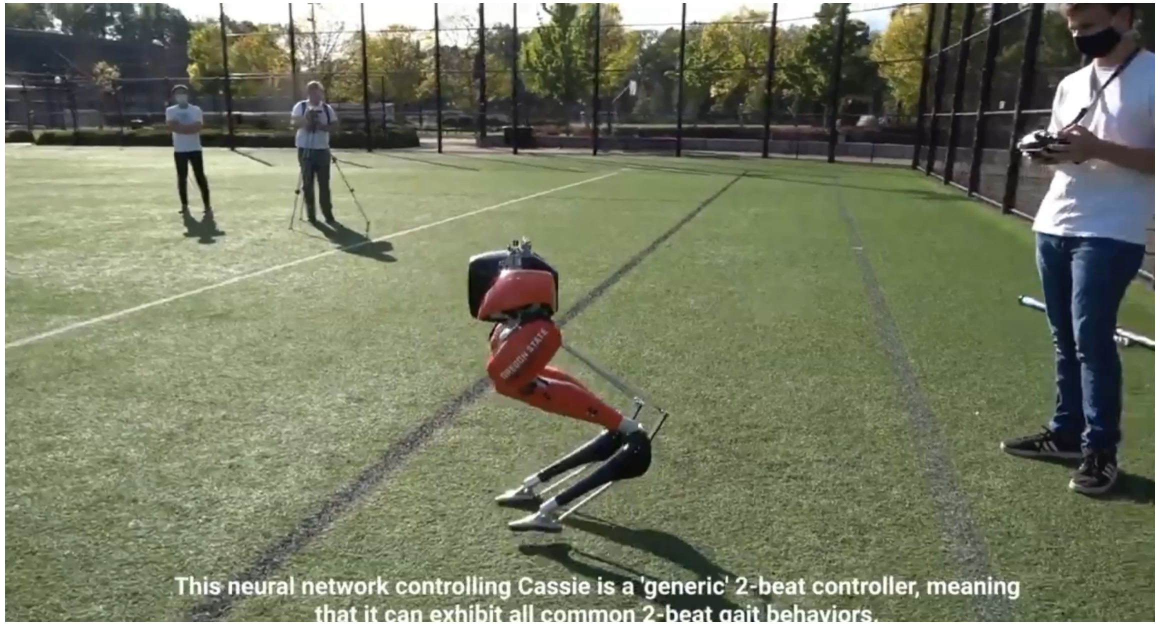


Boston Dynamics, Atlas robots

Historical Context (Last Decade)

- Accurate simulation environments for contact
- **Deep learning** algorithms (reinforcement learning)

Miki et al., Science Robotics, 2022



Dynamic Robotics Laboratory, Oregon State University

Historical Context (Recent works)

- **Retargeting** human motions to the robot via RL policies
- **Motion discovery** using search and optimization

Yang et al., Motion retargeting, ICRA 2026

Taouil et al., Motion discovery, 2026

Quaternions for Rotations

Rotation Matrix

- **Rotation**: We can use a 3x3 matrix to represent the rotation of a rigid body in 3D space

- The **rotation matrix** has two key properties:

$$R^\top = R^{-1}, RR^\top = I \qquad\qquad det(R) = 1$$

Rotation Matrix

- **Euler angles**: one way to represent the general rotation of a rigid body
- Rotation is described by a composition of a **chain of rotations** around different axes
- We obtain the general rotation matrix by using **matrix multiplication**

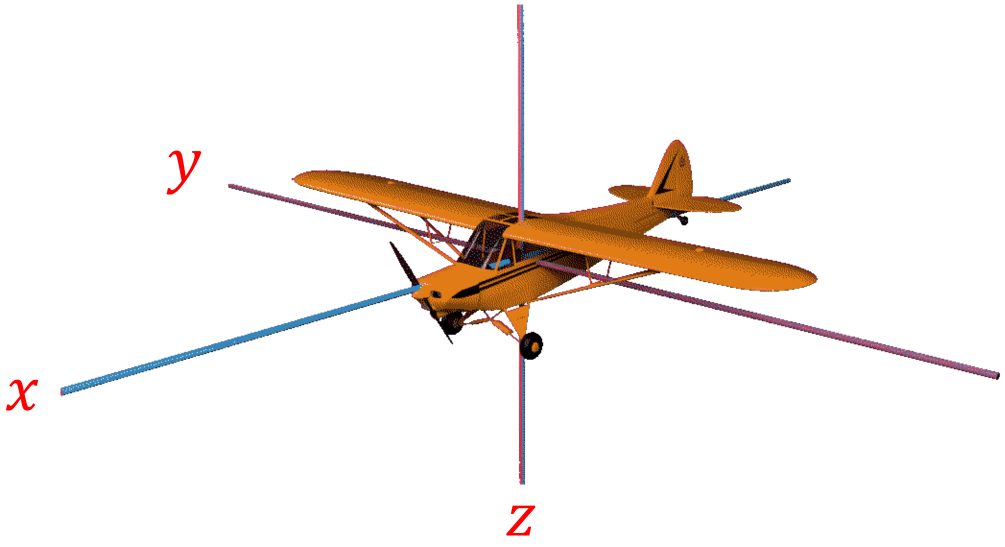
Extrinsic Rotation

- Rotation about **fixed** axes (red axes)
  1. Roll (θ_x)
  2. Pitch (θ_y)
  3. Yaw (θ_z)

- Order is important

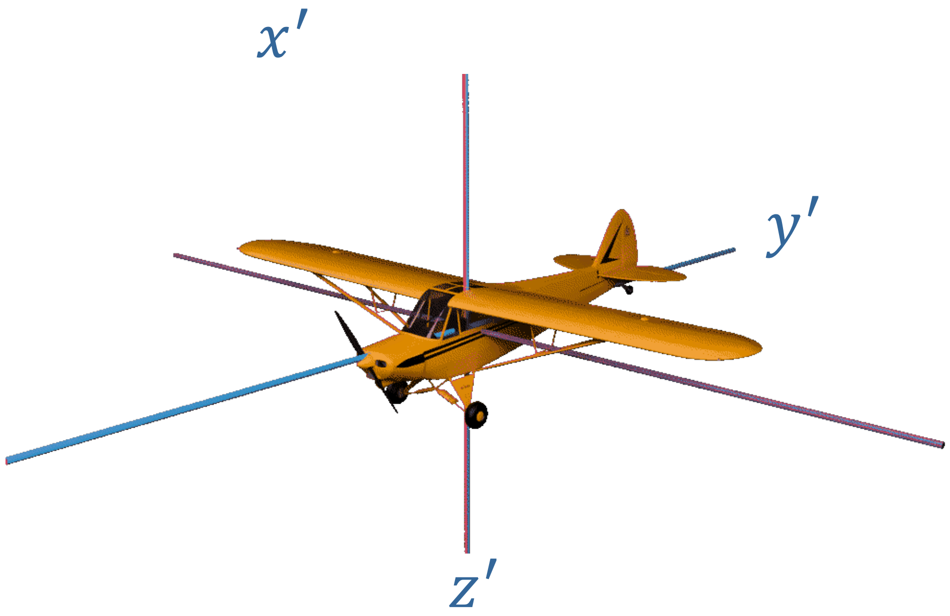
$$R = R_z(\theta_z) R_y(\theta_y) R_x(\theta_x)$$

- Commonly used in **robotics**



Intrinsic Rotation

- Rotation about **rotated** axes (blue axes)
  1. Yaw $(\theta_{z'})$
  2. Pitch $(\theta_{y'})$
  3. Roll $(\theta_{x'})$



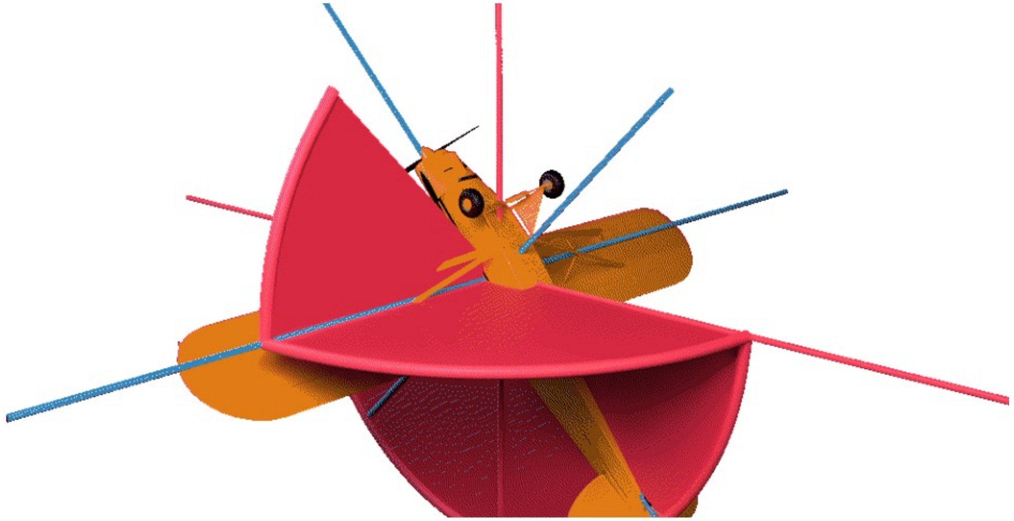
- Order is important

$$R = R_{x'}(\theta_{x'})R_{y'}(\theta_{y'})R_{z'}(\theta_{z'})$$

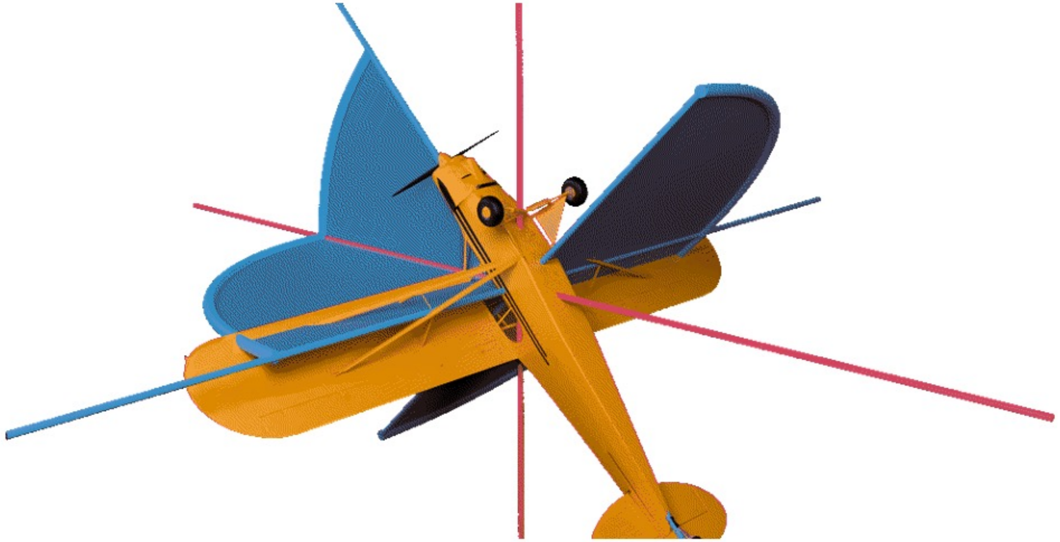
- Commonly used in **aerospace**

Extrinsic and Intrinsic Rotation

- Final orientation is the same
- **Extrinsic** rotation
  1. Roll, $\theta_x = 180$
  2. Pitch, $\theta_{\mathrm{y}} = 45$
  3. Yaw, $\theta_{\mathrm{z}} = 90$

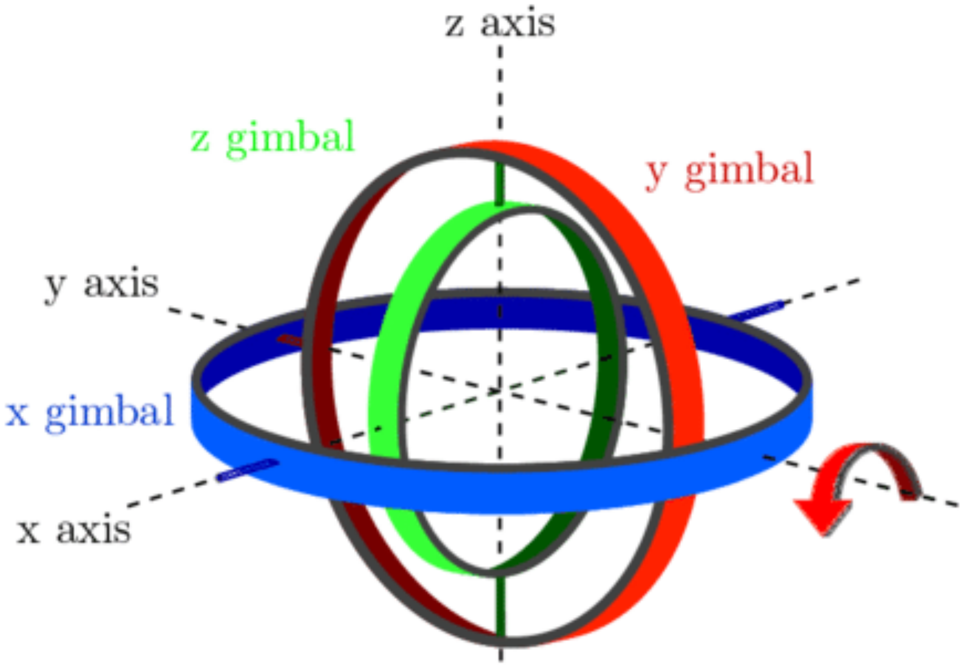


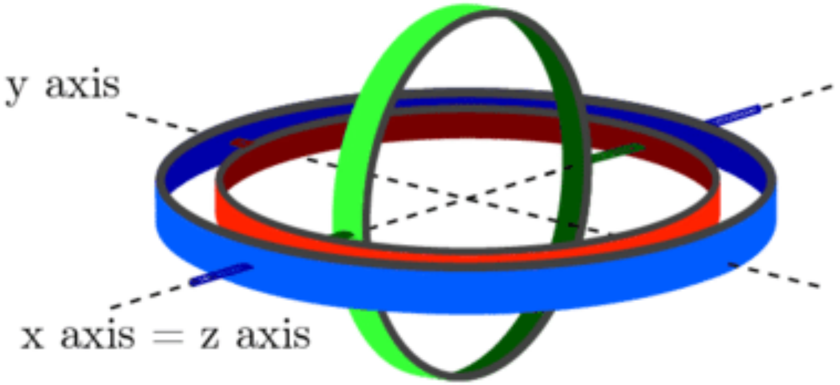
- **Intrinsic** rotation
  1. Yaw, $\theta_{z'} = 90$
  2. Pitch, $\theta_{y'} = 45$
  3. Roll, $\theta_{x'} = 180$



Gimbal Lock

- Euler angles representations suffer from Gimbal lock
- **Loss of one degree of freedom** when two rotation axes align
- When all three align, the whole system can only rotate in one direction from this configuration

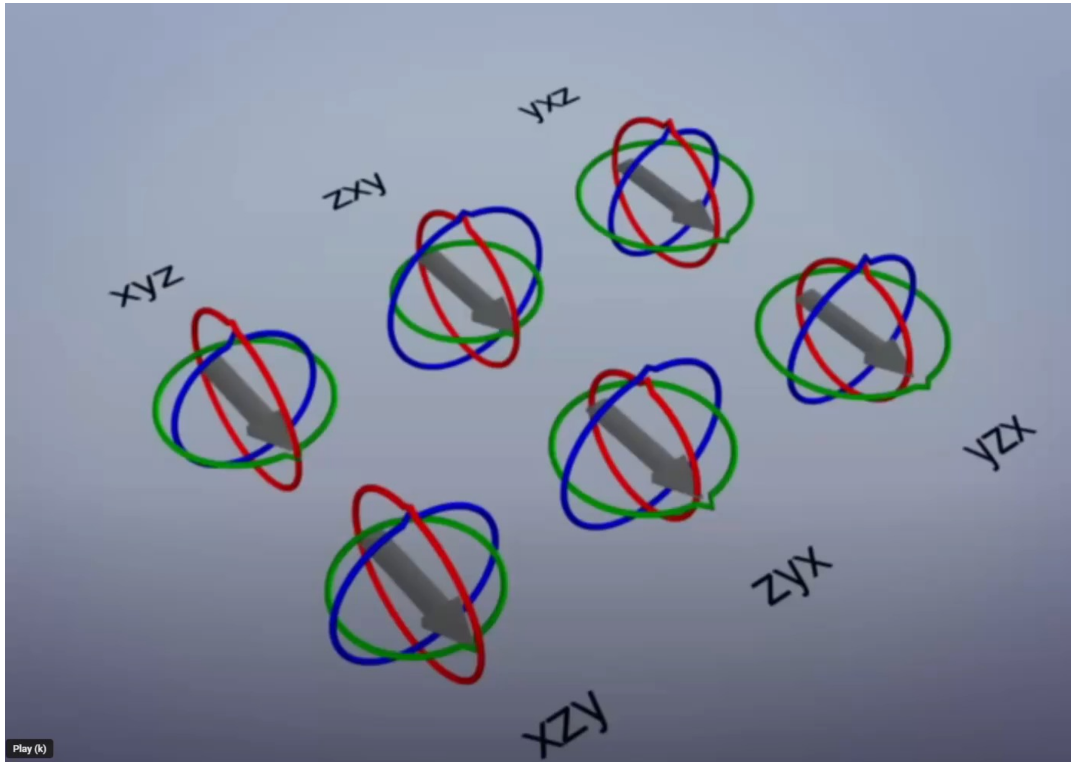




courtesy: Zeithöfler, Thesis, 2019

Gimbal Lock

- Inherent **singularity** with this representation
- Leads to numerical issues and instability
- Usage of **quaternions** avoids such situations



www.youtube.com/watch?v=zc8b2Jo7mno

Quaternions

- 4-parameter representation for modeling 3D rotations
- Can be seen as a complex number with a 3-dimensional complex component
- Not fully intuitive
- But offers very useful properties

Definition Quaternion as Complex Number

- 4D vector:

$$\mathbf{q} = \begin{bmatrix} q \\ \boldsymbol{q} \end{bmatrix} \quad \text{with} \quad \boldsymbol{q} = [q_1, q_2, q_3]^\top$$

- Can be interpreted as a 1D **real number** q and a 3D **complex part** $\boldsymbol{q}$:

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

- With 3 complex units i, j, k
- For which holds:

$$i^2 = j^2 = k^2 = ijk = -1$$

Quaternion as a Tuple

- Written as a **4D vector**:

$$\mathbf{q} = \begin{bmatrix} q \\ \boldsymbol{q} \end{bmatrix} \quad \text{with} \quad \boldsymbol{q} = [q_1, q_2, q_3]^\top$$

- Written as a **tuple**:

$$\mathbf{q} = (q, \boldsymbol{q})$$

- **Unit quaternion**:

$$\boldsymbol{q} \quad \text{with} \quad ||\boldsymbol{q}|| = 1$$

Quaternion Algebra

- **Do not** use regular algebra
- Special quaternion algebra must be used for calculating and combining rotations

Addition

Defined as the element-wise sum (as for regular vectors):

$$\mathbf{p} = \mathbf{q} + \mathbf{r}$$

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} q_0 + r_0 \\ q_1 + r_1 \\ q_2 + r_2 \\ q_3 + r_3 \end{bmatrix} \qquad (p, \boldsymbol{p}) = (q + r, \boldsymbol{q} + \boldsymbol{r})$$

Multiplication

Defined as:

$$\mathbf{p} = \mathbf{qr}$$

$$(p, \boldsymbol{p}) = (qr - \boldsymbol{q} \cdot \boldsymbol{r}, r\boldsymbol{q} + q\boldsymbol{r} + \boldsymbol{q} \times \boldsymbol{r})$$

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} q_0r_0 - q_1r_1 - q_2r_2 - q_3r_3 \\ q_1r_0 + q_0r_1 - q_3r_2 + q_2r_3 \\ q_2r_0 + q_3r_1 + q_0r_2 - q_1r_3 \\ q_3r_0 - q_2r_1 + q_1r_2 + q_0r_3 \end{bmatrix}$$

Multiplication is **not** commutative!

Inverse

- Defined as:

$$\mathbf{q}^* = \begin{bmatrix} q \\ -\boldsymbol{q} \end{bmatrix}$$

conjugate

$$\boldsymbol{q}^{-1} = \frac{\boldsymbol{q}^*}{\|\boldsymbol{q}\|^2}$$

$$\|\boldsymbol{q}\|^2 = q_0^2 + q_1^2 + q_2^2 + q_3^2$$

- For unit quaternion: $\boldsymbol{q}^{-1} = \boldsymbol{q}^*$

- Identity: $\boldsymbol{q} = [1, 0, 0, 0]^\top$

Quaternions to Represent Rotations

- Rotation of θ about $\boldsymbol{r}$ is represented by the quaternion:

$$\mathbf{q} = \begin{bmatrix} q \\ \boldsymbol{q} \end{bmatrix} = \begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2)\boldsymbol{r} \end{bmatrix}$$

- Normalized rotation axis:

$$\boldsymbol{r} = [r_1, r_2, r_3]^\top \quad \text{with} \quad \|\boldsymbol{r}\| = 1$$

- Note: $\|\boldsymbol{r}\| = 1$ always yields a unit quaternion

Example: Quaternion Rotation

- Rotation of θ about $\boldsymbol{r}$ with:

$$\theta = 30^\circ \qquad \boldsymbol{r} = \begin{bmatrix} 0.6 \\ 0.8 \\ 0.0 \end{bmatrix}$$

- Equation:

$$\mathbf{q} = \begin{bmatrix} q \\ \boldsymbol{q} \end{bmatrix} = \begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2)\boldsymbol{r} \end{bmatrix}$$

- Different notations:

$$\mathbf{q} = \begin{bmatrix} \cos(15^\circ) \\ \sin(15^\circ) \begin{bmatrix} 0.6 \\ 0.8 \\ 0.0 \end{bmatrix} \end{bmatrix}$$

$$\mathbf{q} = \cos(15^\circ) + \sin(15^\circ)(0.6\,i + 0.8\,j + 0.0\,k)$$

Executing a Rotation

- By left and right multiplication, we can perform a rotation of the complex part $\boldsymbol{p}$ of $\mathbf{p} = (0, \boldsymbol{p})$ by $\mathbf{q}$:

$$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1}$$

Executing a Rotation

- By left and right multiplication, we can perform a rotation of the complex part $\boldsymbol{p}$ of $\mathbf{p} = (0, \boldsymbol{p})$ by $\mathbf{q}$:

$$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1}$$

Example:

- 3D point to rotate: $\boldsymbol{p}$
- Define: $\mathbf{p} = (0, \boldsymbol{p})$
- Definition rotation: $\mathbf{q} = (\cos(\theta/2), \sin(\theta/2)\boldsymbol{r})$
- Rotate point: $\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1}$

Compositions of Rotations

- Rotations can be easily composed by multiplication
- Combined rotation $\mathbf{q}$ is obtained by **multiplying the rotation quaternions** $\mathbf{q}'', \mathbf{q}'$:

$$\mathbf{q} = \mathbf{q}''\mathbf{q}'$$

- And executing the rotation by

$$\mathbf{p}'' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1}$$

Rigid Body Transformation w/ Quaternions

- Rotations with quaternions can be written similarly to those with rotation matrices
- Transforming a point $\boldsymbol{p}$ by rotation $\mathbf{q}$ and translation $\boldsymbol{t}$:
  1. Turn 3D point into a quaternion
     $$\mathbf{p} = (0, \boldsymbol{p})$$
  2. Perform the rotation with
     $$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1}$$
  3. Perform the translation with $\boldsymbol{t}$ (add complex parts)
     $$\boldsymbol{p}'' = \boldsymbol{p}' + \boldsymbol{t}$$

Advantages of Quaternions for Robotic Applications

- **No singularities** and **no discontinuities** in contrast to Euler angles
- **More compact and efficient** than rotation matrices (4 vs. 9 parameters)
- **Only 1 constraint** needs to be fulfilled in contrast to rotation matrices
- **Allow for interpolation** in contrast to the two other approaches

Comparison with Rotation Matrix

	Rotation matrix, $\mathbf{R}$	Quaternion, $\mathbf{q}$
Parameters	$3 \times 3 = 9$	$1 + 3 = 4$
Degrees of freedom	3	3
Constraints	$9 - 3 = 6$	$4 - 3 = 1$
Constraints	$\mathbf{R}\mathbf{R}^\top = \mathbf{I} \;\; ; \;\; \det(\mathbf{R}) = +1$	$\mathbf{q} \otimes \mathbf{q}^* = 1$

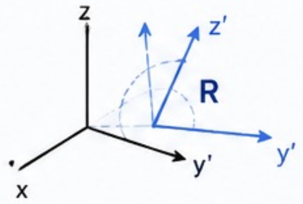
Identity	$\mathbf{I}$	1
Inverse	$\mathbf{R}^\top$	$\mathbf{q}^*$
Composition	$\mathbf{R}_1\,\mathbf{R}_2$	$\mathbf{q}_1 \otimes \mathbf{q}_2$
Rotation operator	$\mathbf{R} = \mathbf{I} + \sin\phi\,[\mathbf{u}]_\times + (1 - \cos\phi)\,[\mathbf{u}]_\times^2$	$\mathbf{q} = \cos\phi/2 + \mathbf{u}\sin\phi/2$
Rotation action	$\mathbf{R}\,\mathbf{x}$	$\mathbf{q} \otimes \mathbf{x} \otimes \mathbf{q}^*$

Quaternions vs. Rotation Matrices

Two representations of 3D orientation – used for different strengths

ROTATION MATRICES ($R \in SO(3)$)

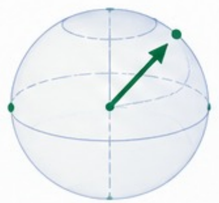
$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$



VS.

QUATERNIONS ($q = [q_x \quad q_y \quad q_z \quad q_w]$)

$$q = \begin{bmatrix} q_x \\ q_y \\ q_z \\ q_w \end{bmatrix}$$



Unit quaternion:
$||q|| = 1$

Rotation Matrices

WHAT IT IS $3{\times}3$ orthonormal matrix with $\det(R) = +1$

ADVANTAGES

- Intuitive geometric meaning (axes / frames)
- Directly used in coordinate transformations
- Easy to compose: $R_3 = R_1 R_2$
- Natural for rigid body kinematics: $\dot{R} = R[\omega]_\times$
- Standard in robotics, vision geometry, and PBVS

BEST USED WHEN

- Working with coordinate frames and poses ($T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}$)
- Deriving control laws in visual servoing (SO(3) / se(3))
- Computing projections, Jacobians, and geometric relations
- Pose estimation from vision / calibration

KEY PROS

- ✓ Very clear geometric interpretation
- ✓ Directly compatible with spatial algebra
- ✓ Widely used in robotics and computer vision

Quaternions

WHAT IT IS 4D unit vector (q_x, q_y, q_z, q_w) with $||q|| = 1$

ADVANTAGES

- Compact (4 parameters vs. 9 for matrices)
- No singularities (unlike Euler angles)
- Smooth interpolation with SLERP
- Efficient and numerically stable
- Easy to optimize: simple constraint $||q|| = 1$

BEST USED WHEN

- Interpolating rotations / generating smooth motion
- Optimization and filtering (e.g., EKF, bundle adjustment)
- Learning / neural networks (compact, continuous)
- Representing orientation in robot policies and logs

KEY PROS

- ✓ Smooth, singularity-free interpolation
- ✓ Compact and efficient
- ✓ Well-suited for optimization and learning

SUMMARY

- Use rotation matrices for geometric reasoning, frame transformations, and visual servoing math.
- Use quaternions for smooth motion, interpolation, optimization, and learning-based methods.
- In practice: estimate in matrices (R, p), control in $\mathbb{R}^3$ (position + axis-angle), and command in quaternion when needed.

RELATION

They represent the same rotation.

$\mathbf{R} \leftrightarrow \mathbf{q}$

Use the best tool for each stage!

Takeaway: Matrices excel at geometry. Quaternions excel at smoothness and efficiency.
Both are essential in modern robotics and visual servoing pipelines.

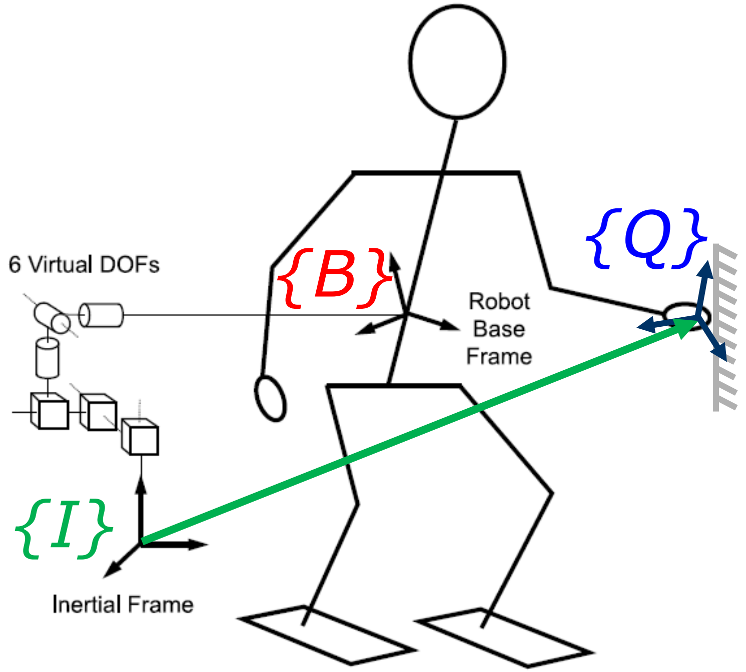
generated with
GPT-5.3-mini

Robot Kinematics and Motion

Floating Base Forward Kinematics

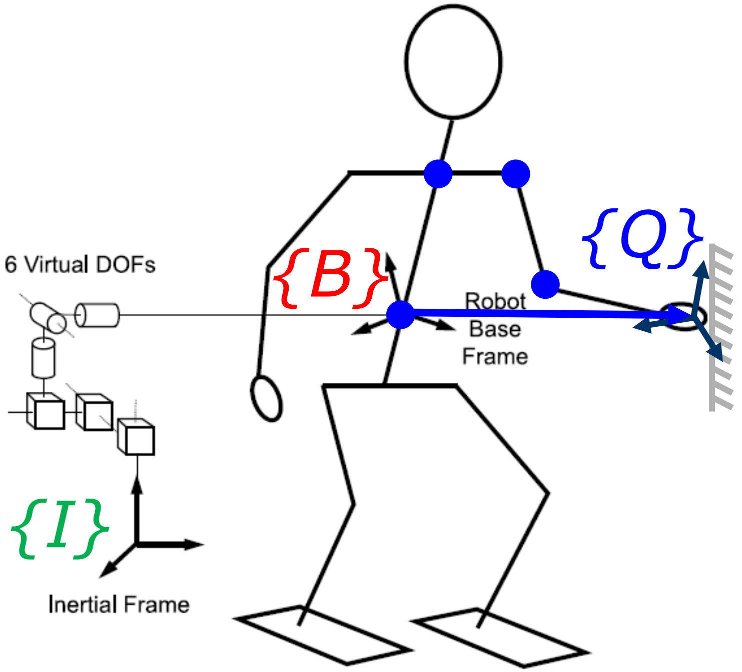
- How to find the end-effector (EE) position w.r.t. the inertial frame {I}?

$$\boxed{{}_{I}\boldsymbol{r}_{IQ}(\boldsymbol{q}) = ?}$$



Floating Base Forward Kinematics

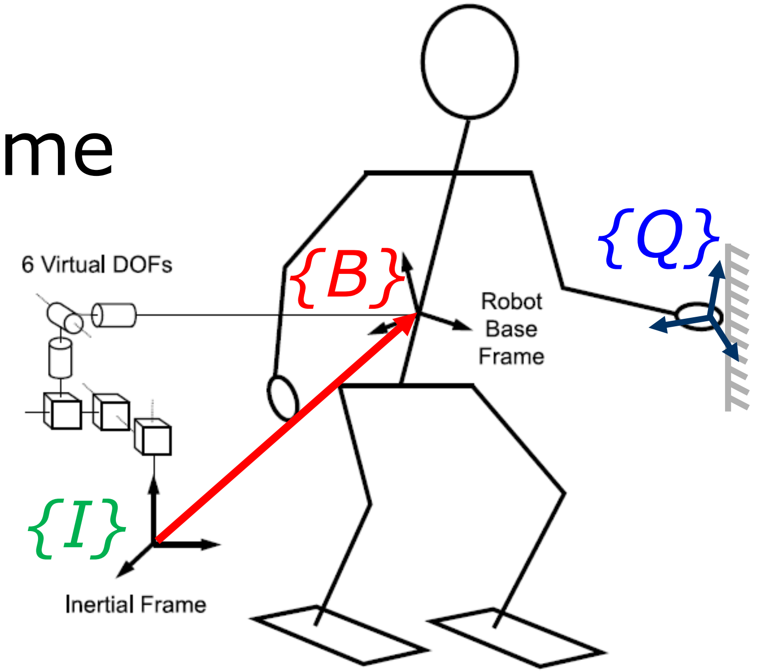
- Forward kinematics from the robot base to the end-effector's frame to calculate the EE position w.r.t. the base frame {B}



${}^{B}\boldsymbol{r}_{BQ}(\boldsymbol{q})$ = *EE position using forward kinematics from B to Q calculated from joint encoders*

Floating Base Forward Kinematics

- Assume we know the robot base position and orientation with respect to inertial frame



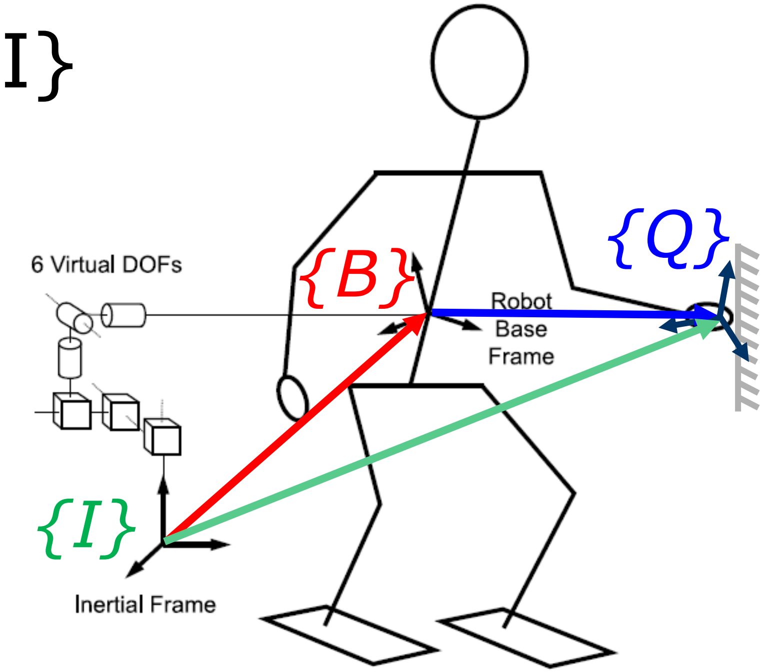
${}^{I}\boldsymbol{r}_{IB}(\boldsymbol{q})$: *Base position*

${}^{I}R_{B}(\boldsymbol{q})$: *Base orientation*

Floating Base Forward Kinematics

- End-effector frame position w.r.t. frame {I}

$$ {}^{I}\boldsymbol{r}_{IQ}(\boldsymbol{q}) = {}^{I}\boldsymbol{r}_{IB}(\boldsymbol{q}) + {}^{I}R_{B}(\boldsymbol{q}) \ {}^{B}\boldsymbol{r}_{BQ}(\boldsymbol{q}) $$

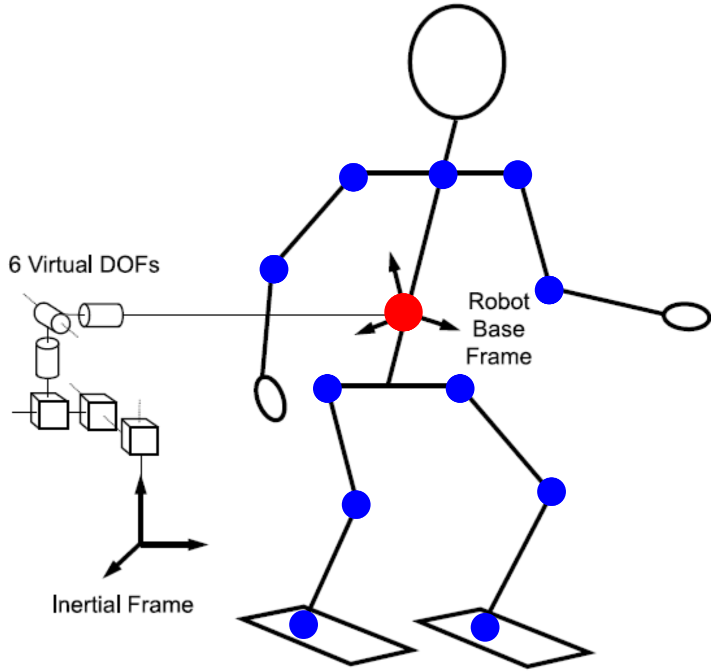


Floating Base Kinematics

- n_b unactuated base pose: no actuator on the robot's base to control its general motion

- n_j actuated joint coordinates

- **Configuration** vector



$$\boldsymbol{p}_B \equiv {}^I\mathbf{r}_{IB}(\mathbf{q})$$

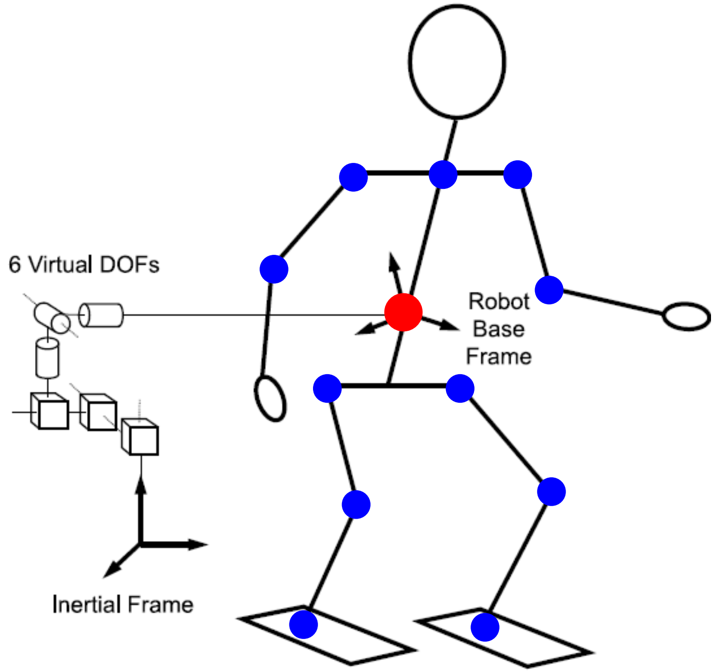
$$quat_B \equiv {}^I\mathbf{R}_B(\mathbf{q})$$

$$\boldsymbol{q} = \begin{pmatrix} \boldsymbol{p}_B \\ quat_B \\ \theta_1 \\ \vdots \\ \theta_{n_j} \end{pmatrix} \quad \begin{matrix} \begin{pmatrix} \boldsymbol{p}_B \\ quat_B \end{pmatrix} \in SE(3) \\ \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_{n_j} \end{pmatrix} \in \mathbb{R}^{n_j} \end{matrix}$$

Floating Base Kinematics

- Generalized **velocity**

$$\boldsymbol{v} = \begin{pmatrix} {}_B\mathbf{v}_B \\ {}_B\boldsymbol{\omega}_{IB} \\ \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_{n_j} \end{pmatrix} \in \mathbb{R}^{6+n_j}$$



- Generalized **acceleration**

$$\dot{\boldsymbol{v}} = \begin{pmatrix} {}_B\boldsymbol{a}_B \\ {}_B\boldsymbol{\alpha}_{IB} \\ \ddot{\theta}_1 \\ \vdots \\ \ddot{\theta}_{n_j} \end{pmatrix} \in \mathbb{R}^{6+n_j}$$

State Estimation

- Sensor modalities on a robot: **IMU**, **joint encoders**

$$\boldsymbol{q} = \begin{pmatrix} \boldsymbol{p}_b \\ quat_b \\ \theta_1 \\ \vdots \\ \theta_{n_j} \end{pmatrix} \qquad \boldsymbol{v} = \begin{pmatrix} {}^{I}\mathbf{V}_B \\ {}^{B}\boldsymbol{\omega}_{IB} \\ \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_{n_j} \end{pmatrix} \in \mathbb{R}^{6+n_j} \qquad \dot{\boldsymbol{v}} = \begin{pmatrix} {}^{I}\boldsymbol{a}_B \\ {}^{B}\boldsymbol{\alpha}_{IB} \\ \ddot{\theta}_1 \\ \vdots \\ \ddot{\theta}_{n_j} \end{pmatrix} \in \mathbb{R}^{6+n_j}$$

- Base states = base position, orientation, and velocity
- Now, how to find the robot's base pose and linear velocity?
- Use **state estimation** to estimate robot's base states

State Estimation

- Base **state estimation** with Kalman Filtering (KF)

- **KF** is a **recursive algorithm** used to estimate the robot's state when the system is nonlinear

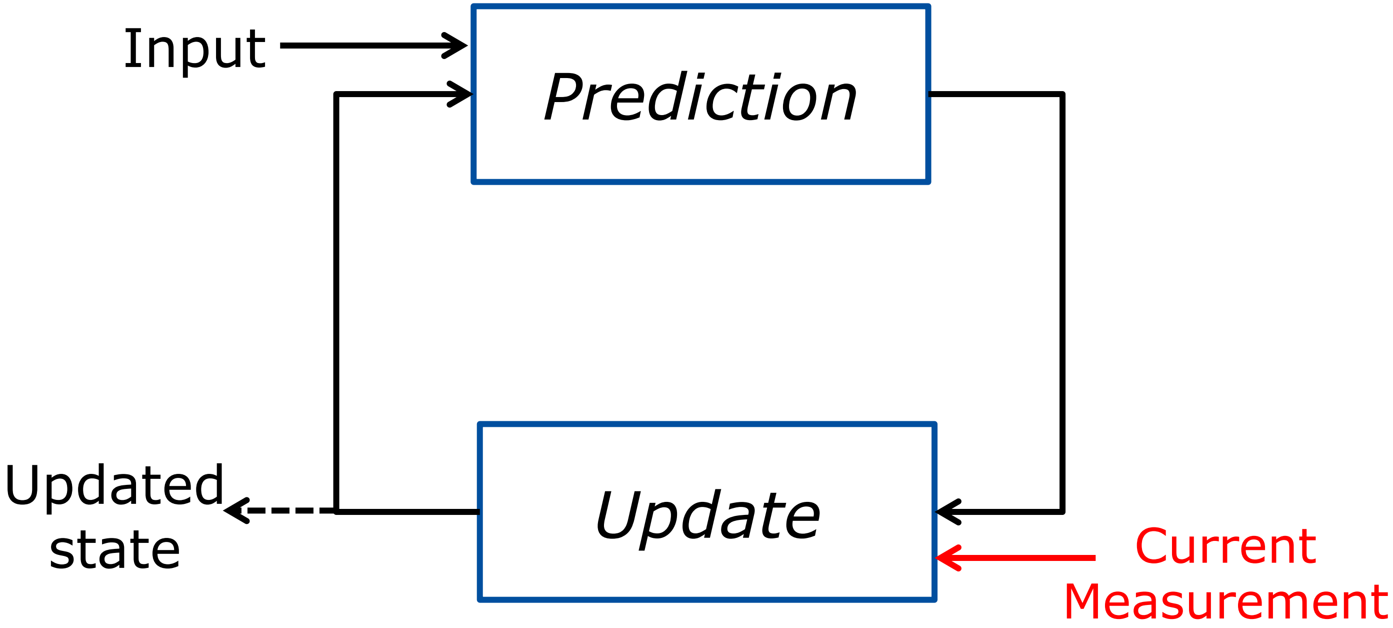
- **Process** model, describes how the system state evolves over time

- **Measurement** model, describes how sensor measurements relate to the current state

State Estimation

KF runs in **two steps**

1. **Prediction**: Predict the next state by integrating a process model (input sensors such as IMU)
2. **Update**: Correct the predicted state using external measurements like leg odometry (from forward kinematics)



[Cognitive Robotics/
Intro to Mobile Robotics]

Kalman Filter: State Estimation

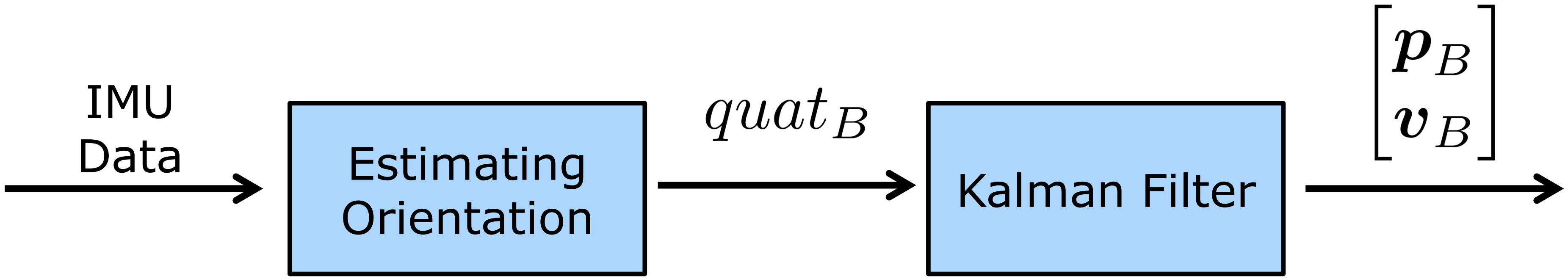
- **State:** Base position, orientation, and linear velocity (expressed in the world frame)

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p}_B \\ quat_B \\ \boldsymbol{v}_B \end{bmatrix}$$

- **Decomposing approach:** Key idea is to decompose the problem into two separate estimations

Kalman Filter: State Estimation

- **First:** Estimate the **orientation** from IMU data
- Under **decomposition assumption**, estimating position and velocity becomes a linear problem, so we can use standard KF



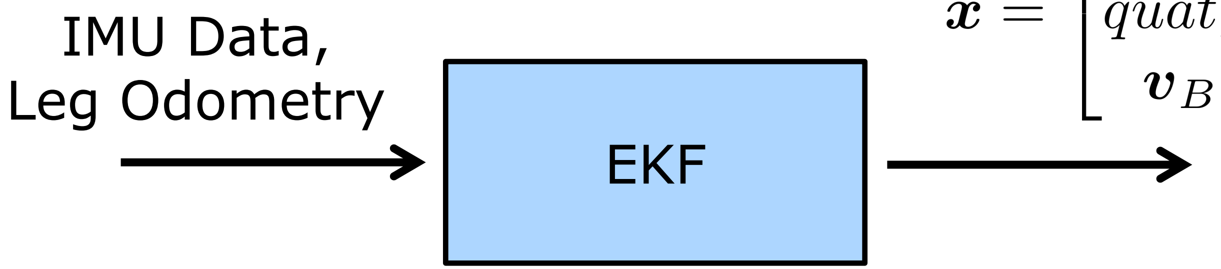
- How to implement KF? Next exercise sheet*.

* [Flayols et al., "Experimental Evaluation of Simple Estimators for Humanoid Robots," Humanoids, 2017]

Kalman Filter: State Estimation

- **State:** Base position, orientation, and linear velocity (expressed in the world frame)

- **Second approach**: **Jointly** estimate all states
- Now the process and measurement models become **nonlinear**



$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p}_B \\ quat_B \\ \boldsymbol{v}_B \end{bmatrix}$$

Extended Kalman Filter: State Estimation

- **Extended Kalman Filter** (EKF) follows the same basic idea as the Kalman Filter:

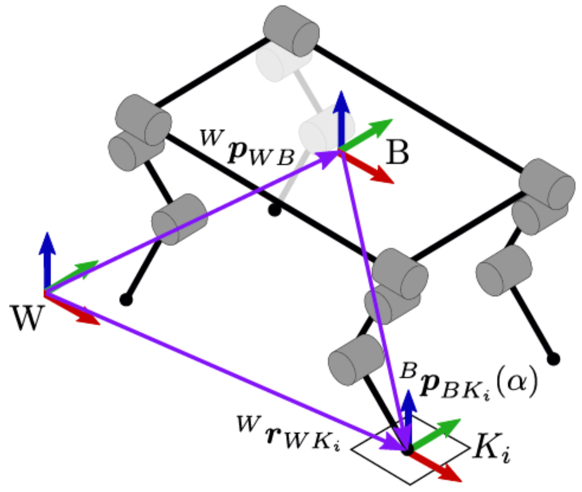
1. **Prediction:** predict the next state using a process model
2. **Update:** correct the prediction using sensor measurements

- However, the process and measurement models are **nonlinear**
- So EKF linearizes these models at every iteration using Jacobians

Extended Kalman Filter: State Estimation

- We have known **forward kinematics** (FK), for any foot at contact

$$\boxed{{}^{W}\boldsymbol{p}_{WB} + {}^{W}\boldsymbol{R}_{B}\left({}^{B}\boldsymbol{p}_{BK_i}(\alpha)\right) = {}^{W}\boldsymbol{r}_{WK_i}}$$



- **Measurement model:** Derivative of FK w.r.t. time, which is a nonlinear model*

$$\boxed{{}^{W}\dot{\boldsymbol{r}}_{WK_i} = {}^{B}\boldsymbol{v}_{WB} + {}^{B}\boldsymbol{v}_{BK_i} + {}^{B}\boldsymbol{\omega}_{WB} \times {}^{B}\boldsymbol{p}_{BK_i} = 0}$$

contact-point velocity must be zero

* [Camurri et al., "Pronto: A Multi-Sensor State Estimator for Legged Robots in Real-World Scenarios," Front. in Robotics&AI, 2020]

State Estimation - Comparison

Kalman Filter

- Orientation estimated **separately**
- Position and velocity estimated using KF
- Easier to understand, implement, and tune in practice
- **Modular structure**: Each part of the estimator can be adjusted separately
- **Limitation**: Errors propagate through the cascade
- If orientation estimation deteriorates, position and velocity estimation are also affected

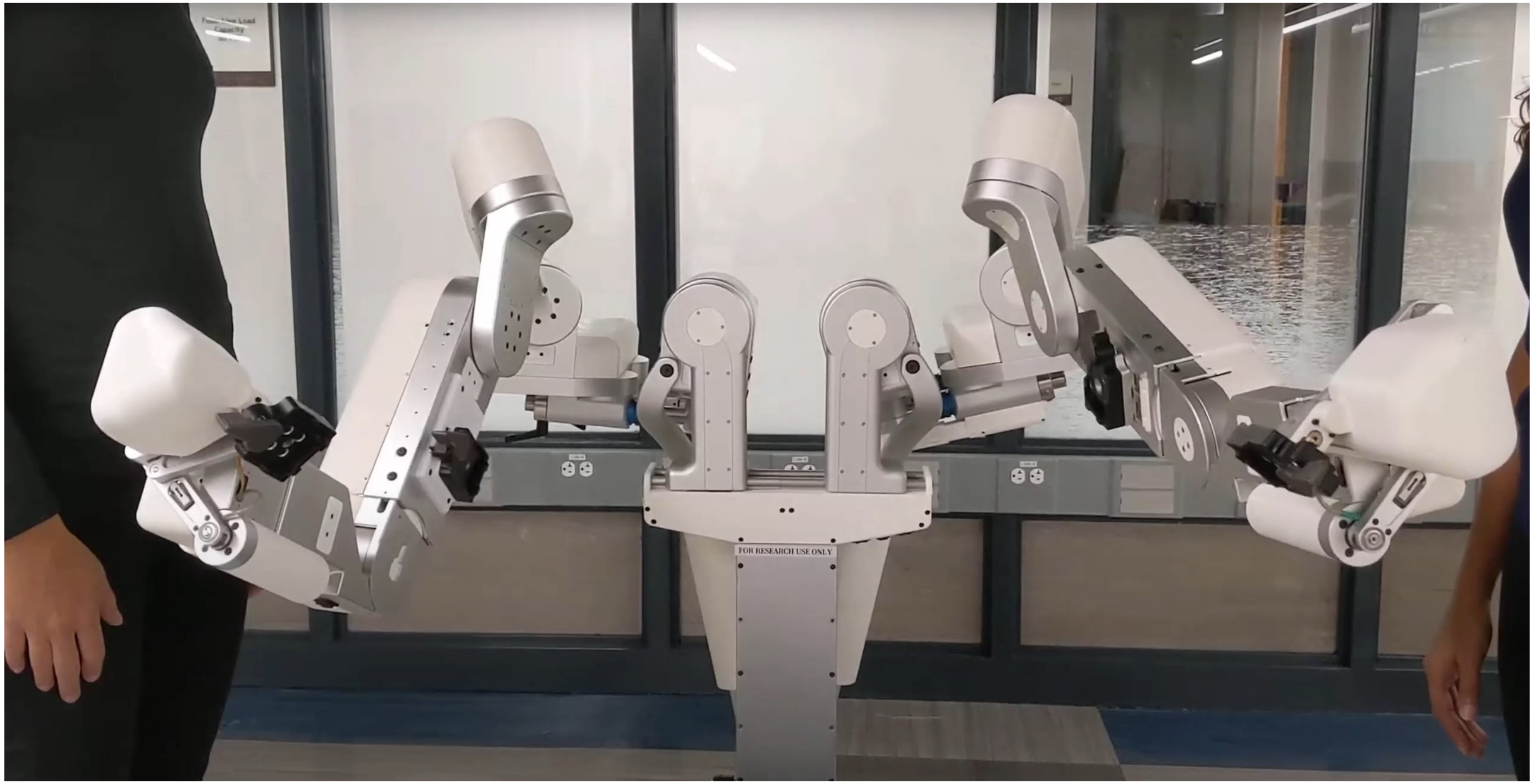
Extended Kalman Filter

- **Jointly** estimates base orientation, position, velocity, and possibly sensor biases
- Can **fuse multiple sensor modalities** in a single probabilistic framework
- Captures **coupling** between different parts of the state
- Often **performs robustly** in real-world legged robot applications
- **Limitation**: Nonlinear models require linearization at every step
- Can be difficult to tune and does not generally guarantee stable error dynamics

Robot Dynamics

Motivation

- Why do we need robot dynamics?



Force control vs. Position control

Motivation

Force control

- Needs robot's dynamics
- Needs torque sensing
- Inverse dynamics
- Compliant behavior

Position control

- Needs robot's kinematics
- Joint positions are enough
- High PID gains
- Stiff behavior

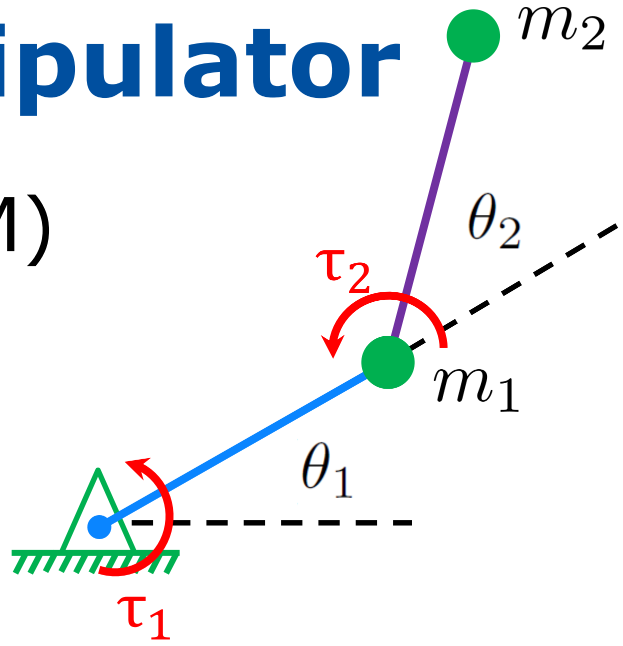
Dynamics of a Chain of Rigid Bodies

- To control a legged robot, position control using kinematics is not enough

- We need to know the dynamics of the robot

- Need to consider inertia, mass, gravity, and force

Dynamics of 2-DoF Robotic Manipulator



- General Form of **equations of Motion** (EoM)

$$\boxed{M(\boldsymbol{q})\dot{\boldsymbol{v}} + \boldsymbol{c}(\boldsymbol{q}, \boldsymbol{v}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau}}$$

- Goal: Find required torque (τ_1, τ_2)

$M(\boldsymbol{q})$: matrix, represents forces due to **inertia**

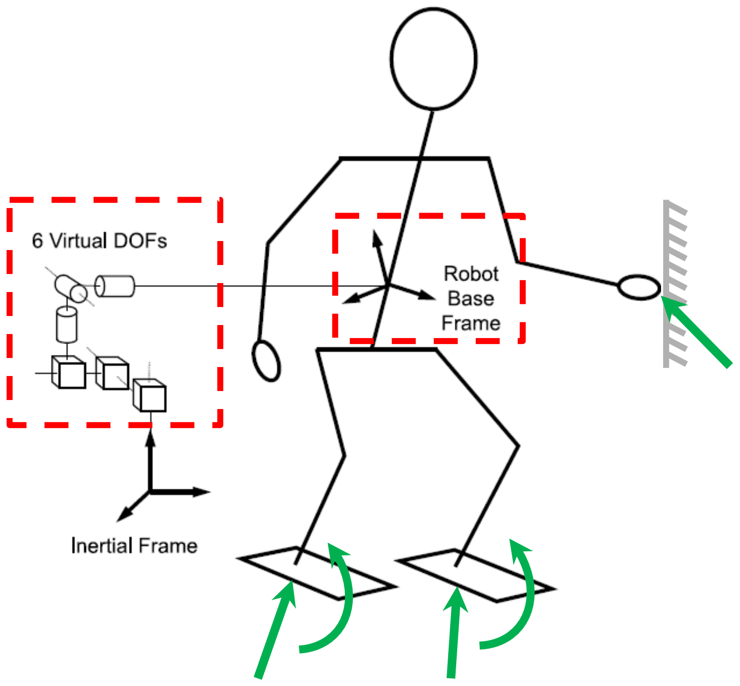
$\boldsymbol{c}(\boldsymbol{q}, \boldsymbol{v})$: vector, forces which appear due to **rotation**

$\boldsymbol{g}(\boldsymbol{q})$: vector, represents forces due to **gravity**

Dynamics of Legged Robots

- Legged robots are generally **underactuated**
- They have more degrees of freedom than number of actuators

- For the dynamics, we need to consider:
  1. Unactuated base
  2. Contacts forces/torques through interaction with the environment

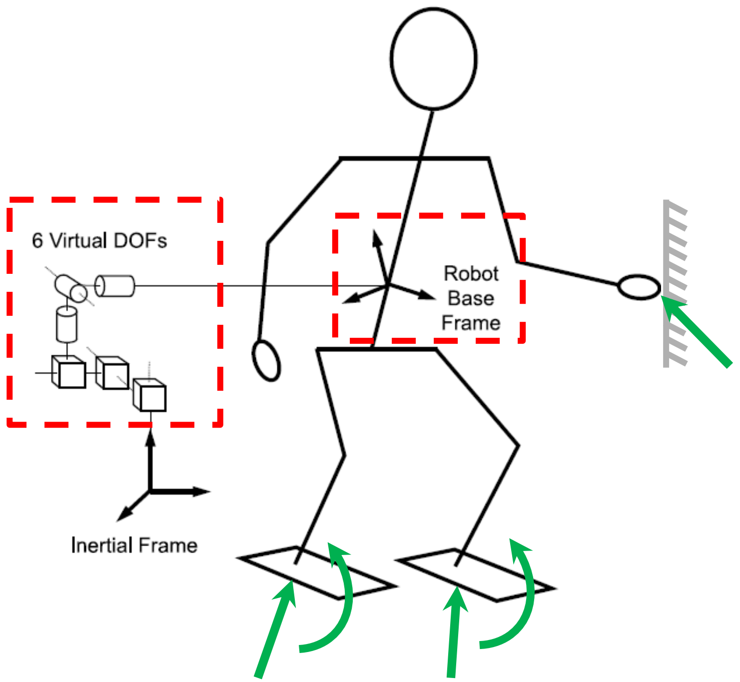


Dynamics of Legged Robots

- General form of EoM

$$S = \begin{bmatrix} 0_{n_j \times 6} & I_{n_j \times n_j} \end{bmatrix}$$

$$M(\boldsymbol{q})\dot{\boldsymbol{v}} + \boldsymbol{c}(\boldsymbol{q}, \boldsymbol{v}) + \boldsymbol{g}(\boldsymbol{q}) = \boxed{S^\top}\boldsymbol{\tau} + \sum_i J_i^T \boxed{\boldsymbol{f}_i}$$



- Dimensions: $\begin{cases} M(\boldsymbol{q}) \in \mathbb{R}^{(6+n_j)\times(6+n_j)} \\ \boldsymbol{c}(\boldsymbol{q}, \boldsymbol{v}), \boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^{6+n_j} \\ \boldsymbol{\tau} \in \mathbb{R}^{n_j} \end{cases}$

- $J_i \in \mathbb{R}^{6\times(6+n_j)}$: Jacobian of the contact

Recursive Newton Euler Algorithm

- Very fast and **efficient algorithm** to compute robot inverse dynamics, i.e., joint torques required for a given motion
- Use **spatial algebra** to represent motion and forces compactly in 6D (linear + angular)
- **Computationally efficient**, linear in the number of links O(n), making it suitable for real-time control

Recursive Newton Euler Algorithm

Two recursive passes through the robot's kinematic chain:

1. *Forward pass*: computes link **velocities** and **accelerations** from base to end-effector
2. *Backward pass*: computes **forces** and **joint torques** from end-effector back to base

Software Tool for Forward and Inverse Dynamics

- **Pinocchio**: suitable for robotics applications, computer graphics, vision, etc.
- Examples on how to use Pinocchio with the Python bindings
- https://github.com/stack-of-tasks/pinocchio/tree/master/examples

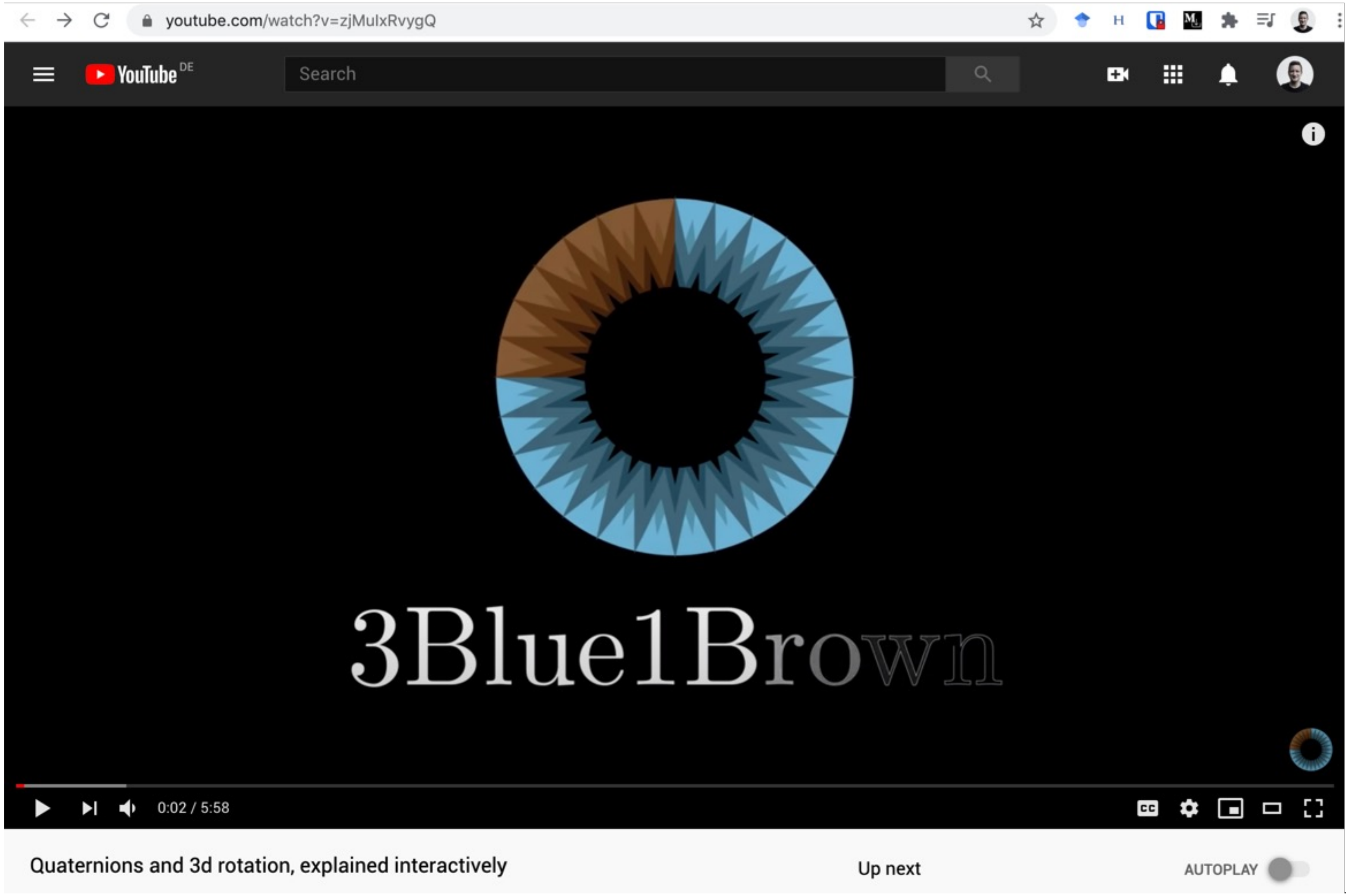
Summary

- **History** of humanoids walking
- **Quaternions**: 4-parameter presentation for modeling 3D rotations, offer some advantages over rotation matrices
- Extension of **kinematics to legged robots**
- Difference between **position** and **torque control**
- **Dynamics** of a chain of rigid bodies and extension to humanoid robots
- **Efficient** algorithm and software tools to calculate robots' **kinematics** and **dynamics**

Literature

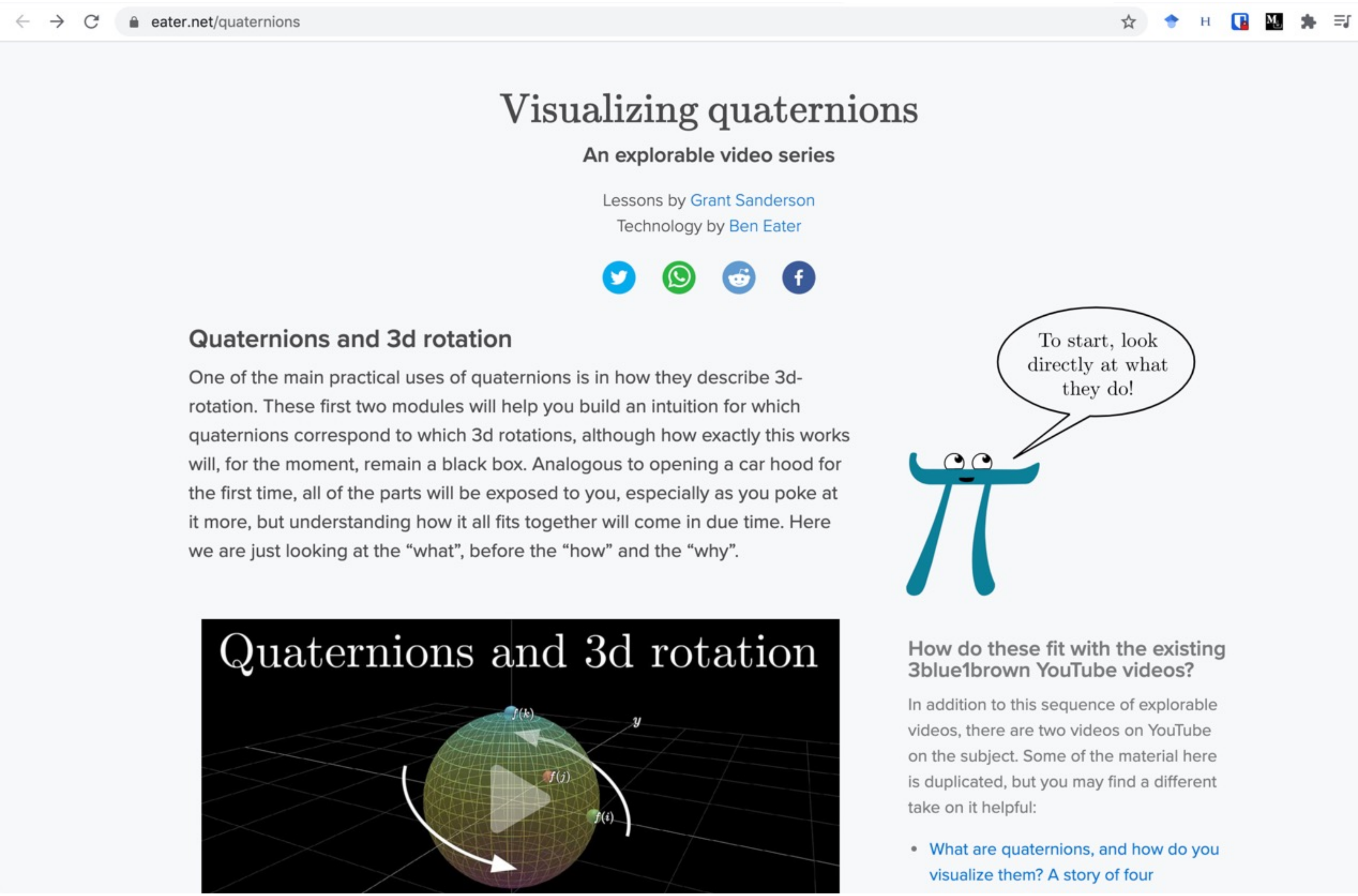
- Wensing, Posa, Hu, Escande, Mansard, and Prete, "Optimization-Based Control for Dynamic Legged Robots," IEEE Transactions on Robotics, 2024
- Ha, Lee, van de Panne, Xie, Yu, Khadiv, "Learning-Based Legged Locomotion: State of the Art and Future Perspectives," International Journal of Robotics Research (IJRR), 2025
- Joan Solà, "Quaternion kinematics for the error-state Kalman filter," 2015
- Agarwal et al., "State Estimation for Legged Robots: Consistent Fusion of Leg Kinematics and IMU", Robotics: Science and Systems (RSS), 2013
- Park and Lynch, "Modern Robotics," Cambridge University Press, 2017
- Roy Featherstone, "Rigid Body Dynamics Algorithms," 2008
- Carpentier andMansard, "Analytical Derivatives of Rigid Body Dynamics Algorithms," Robotics: Science and Systems (RSS) 2018
- Carpentier et al., "Pinocchio: Fast Forward and Inverse Dynamics for Poly-Articulated Systems," https://stack-of-tasks.github.io/Pinocchio

Quaternions: Video Explanation



https://www.youtube.com/watch?v=zjMuIxRvygQ

Quaternions: Interactive Tutorial



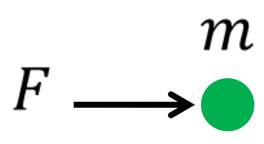
https://eater.net/quaternions

Additional Material
(Derivation of Equations of Motion for Robot Dynamics)

Dynamics Fundamentals

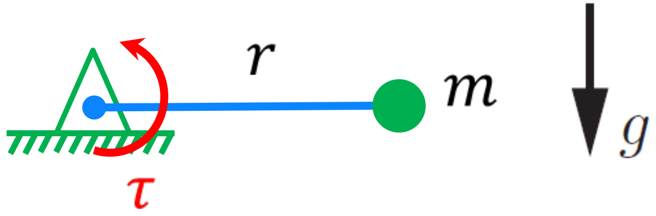
- **Newton's law**: point mass

$$\boldsymbol{F} = m\,\boldsymbol{a}$$



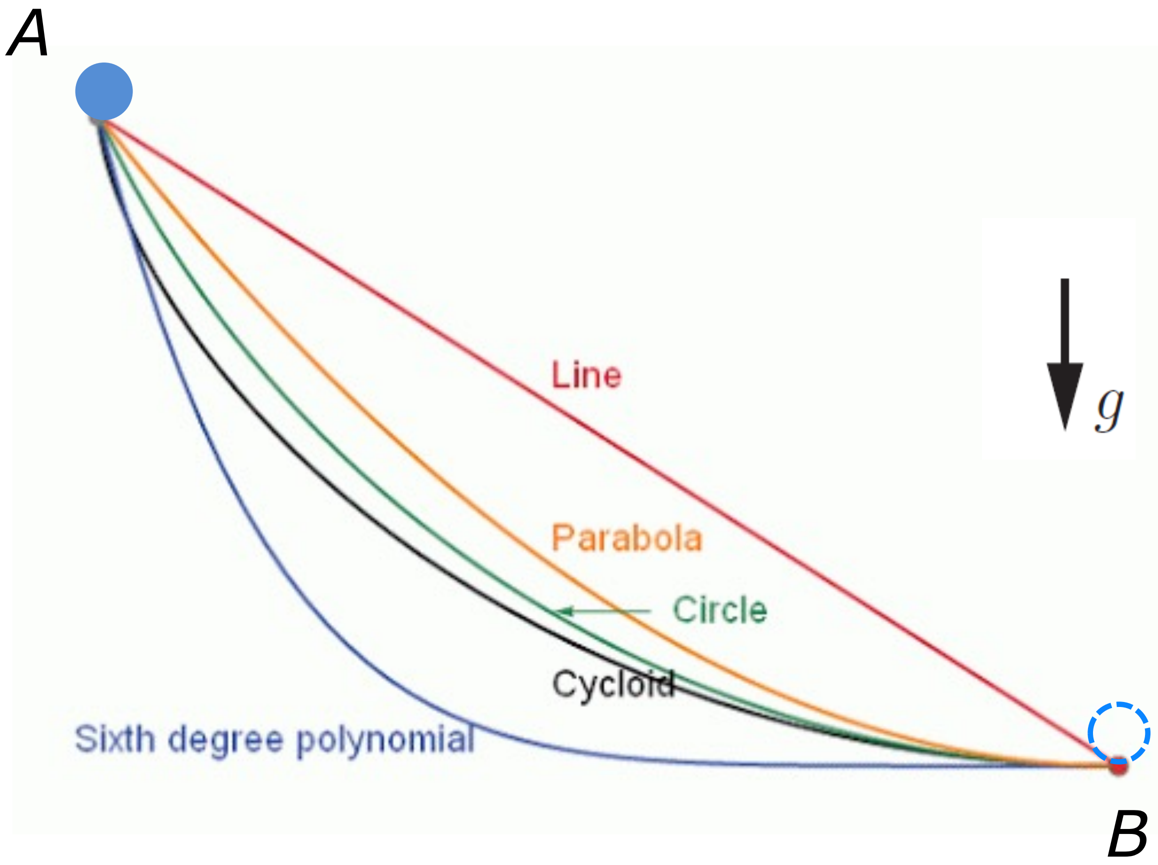
- Rotation with massless links

$$\boldsymbol{\tau} = \boldsymbol{r} \times m\boldsymbol{g}$$



Calculus of Variations

- **Motivation**
- The ball is falling under gravity
- Assume no friction
- What is the path from A to B in terms of **shortest time**? (**Brachistochrone Problem**)



Calculus of Variations

- Euler-Lagrange equation

$$f = \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}} - \frac{\partial \mathcal{L}}{\partial q}$$

- q : Generalized coordinate
- $\dot{q}$: Generalized velocity
- $\mathcal{L}$: Lagrangian

Equations of Motion

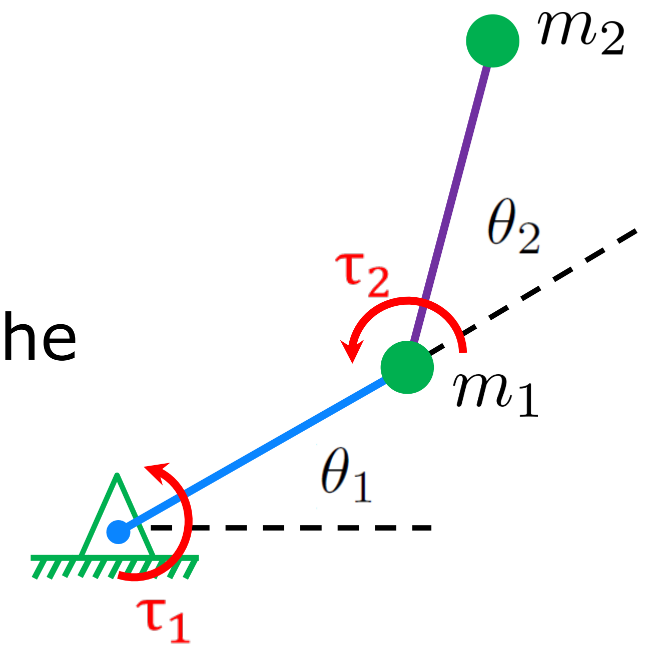
- Lagrangian function: Kinetic energy minus potential energy

$$\mathcal{L}(q,\dot{q}) = \mathcal{K}(q,\dot{q}) - \mathcal{P}(q)$$

- Use Euler-Lagrange equation to find equations of motion

$$f = \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}} - \frac{\partial \mathcal{L}}{\partial q}$$

2 DoF Robotic Manipulator



- **Generalized coordinates**

 (a set of independent variables that fully describes the system's configuration)

$$\boldsymbol{q} = \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}, \quad \dot{\boldsymbol{q}} = \begin{pmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{pmatrix}, \quad \ddot{\boldsymbol{q}} = \begin{pmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{pmatrix}$$

- Control input: torque

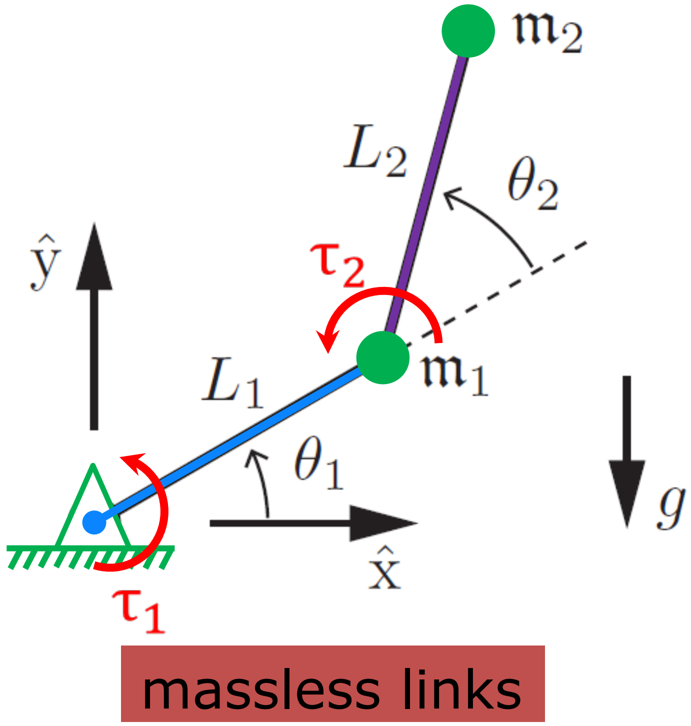
$$\boldsymbol{\tau} = \begin{pmatrix} \tau_1 \\ \tau_2 \end{pmatrix}$$

2 DoF Robotic Manipulator

- Position and velocity of mass-1 in the Cartesian coordinate

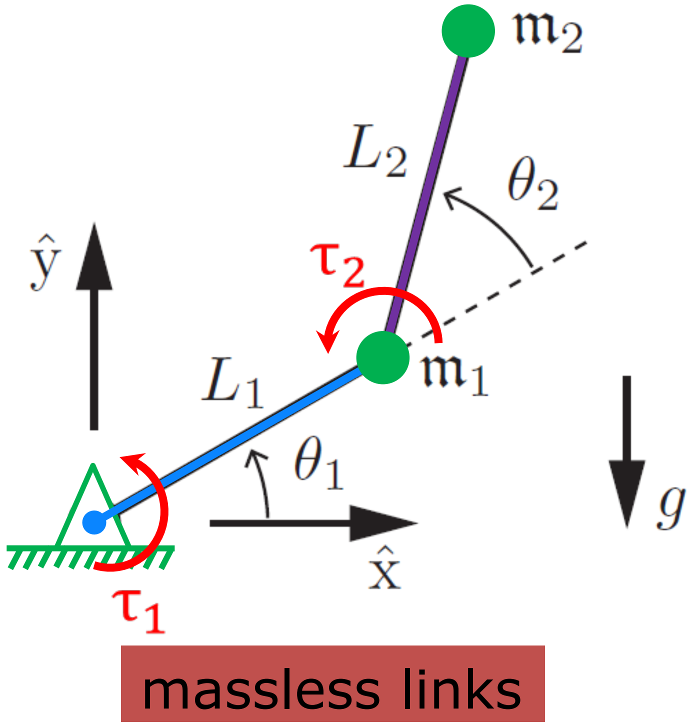
$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} L_1 \cos\theta_1 \\ L_1 \sin\theta_1 \end{bmatrix}$$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \end{bmatrix} = \begin{bmatrix} -L_1 \sin\theta_1 \\ L_1 \cos\theta_1 \end{bmatrix} \dot{\theta}_1$$



2 DoF Robotic Manipulator

- Position and velocity of mass-2 in the Cartesian coordinate



$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) \\ L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2) \end{bmatrix}$$

$$\begin{bmatrix} \dot{x}_2 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} -L_1 \sin\theta_1 - L_2 \sin(\theta_1 + \theta_2) & -L_2 \sin(\theta_1 + \theta_2) \\ L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) & L_2 \cos(\theta_1 + \theta_2) \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

2 DoF Robotic Manipulator

- **Kinetic energy** of the system

$$\begin{aligned}
\mathcal{K}_1 &= \frac{1}{2}\mathfrak{m}_1(\dot{x}_1^2 + \dot{y}_1^2) = \frac{1}{2}\mathfrak{m}_1 L_1^2\dot{\theta}_1^2 \\
\mathcal{K}_2 &= \frac{1}{2}\mathfrak{m}_2(\dot{x}_2^2 + \dot{y}_2^2) \\
&= \frac{1}{2}\mathfrak{m}_2\left((L_1^2 + 2L_1L_2\cos\theta_2 + L_2^2)\dot{\theta}_1^2 + 2(L_2^2 + L_1L_2\cos\theta_2)\dot{\theta}_1\dot{\theta}_2 + L_2^2\dot{\theta}_2^2\right)
\end{aligned}$$

- **Potential energy** of the system

$$\begin{aligned}
\mathcal{P}_1 &= \mathfrak{m}_1 g y_1 = \mathfrak{m}_1 g L_1 \sin\theta_1, \\
\mathcal{P}_2 &= \mathfrak{m}_2 g y_2 = \mathfrak{m}_2 g(L_1\sin\theta_1 + L_2\sin(\theta_1 + \theta_2))
\end{aligned}$$

2 DoF Robotic Manipulator

- Form the Lagrangian and use Euler-Lagrange equation

$$\mathcal{L}(\theta, \dot{\theta}) = \sum_{i=1}^{2} (\mathcal{K}_i - \mathcal{P}_i) \longrightarrow \tau_i = \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\theta}_i} - \frac{\partial \mathcal{L}}{\partial \theta_i}, \qquad i = 1, 2$$

- Take the derivatives and substitute in the equation above

2 DoF Robotic Manipulator

- Gather terms together, write equations in general form

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau}$$

- $M(\boldsymbol{q})$: (2x2) **mass** Matrix (symmetric and positive-definite)

$$M(\boldsymbol{q}) = \left[\begin{array}{c|c} \boxed{\mathfrak{m}_1} L_1^2 + \boxed{\mathfrak{m}_2}(L_1^2 + 2L_1 L_2 \cos\theta_2 + L_2^2) & \boxed{\mathfrak{m}_2}(L_1 L_2 \cos\theta_2 + L_2^2) \\ \hline \boxed{\mathfrak{m}_2}(L_1 L_2 \cos\theta_2 + L_2^2) & \boxed{\mathfrak{m}_2} L_2^2 \end{array}\right]$$

2 DoF Robotic Manipulator

- $\boldsymbol{c}(\boldsymbol{q}, \boldsymbol{v})$: (2x1) vector, containing **Coriolis** and **centrifugal** forces

$$\boldsymbol{c}(\boldsymbol{q}, \boldsymbol{v}) = \begin{bmatrix} -\mathfrak{m}_2 L_1 L_2 \sin\theta_2 (2\boxed{\dot{\theta}_1 \dot{\theta}_2} + \boxed{\dot{\theta}_2^2}) \\ \mathfrak{m}_2 L_1 L_2 \boxed{\dot{\theta}_1^2} \sin\theta_2 \end{bmatrix}$$

- $\boldsymbol{g}(\boldsymbol{q})$: (2x1) vector, contacting **gravitational** force

$$\boldsymbol{g}(\boldsymbol{q}) = \begin{bmatrix} (\mathfrak{m}_1 + \mathfrak{m}_2) L_1 \boxed{g} \cos\theta_1 + \mathfrak{m}_2 \boxed{g} L_2 \cos(\theta_1 + \theta_2) \\ \mathfrak{m}_2 \boxed{g} L_2 \cos(\theta_1 + \theta_2) \end{bmatrix}$$