COMPUTER SCIENCE Capstone PBL
- Character Motion Synthesis and Character Control

4 - Motion Editing, Motion Synthesis

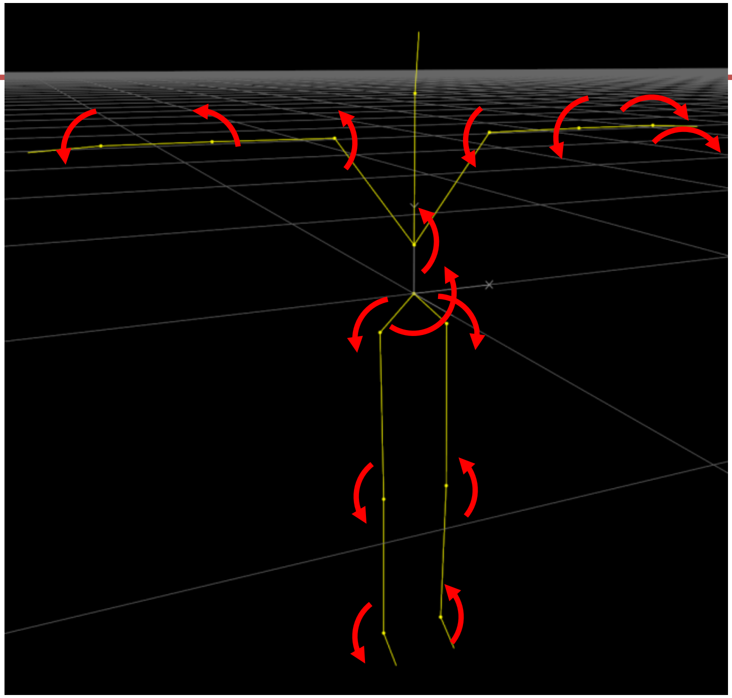
Yoonsang Lee
Fall 2022

Today's Topics

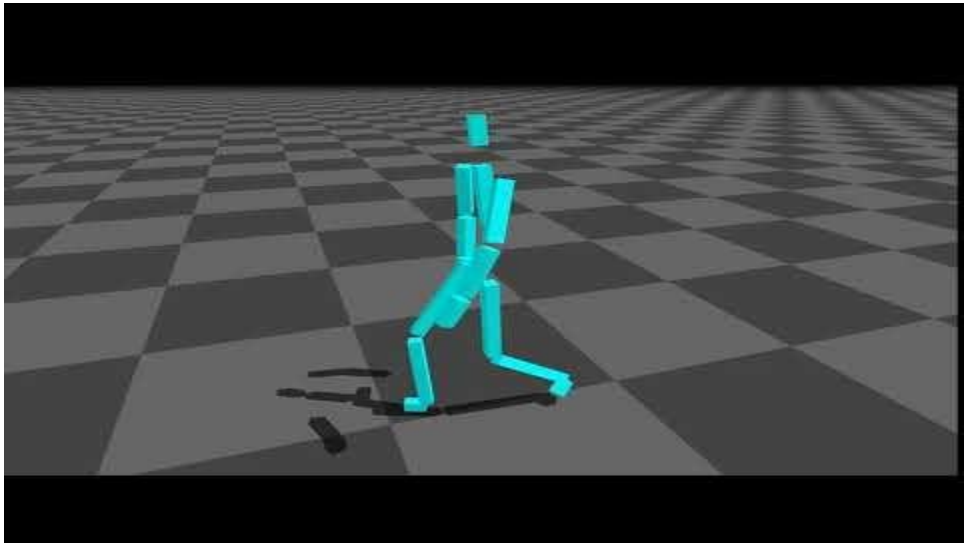
- 3D Orientation & Rotation
- Inverse Kinematics
  - Limb IK
- Motion Editing Techniques
  - Posture / Motion Difference
  - Motion Warping
- Intro to More Motion Editing Techniques
  - Interpolation of Postures
  - Time Warping
  - Motion Stitching
  - Motion Blending
- Intro to Data-Driven Motion Synthesis
  - Motion Graph
  - Motion Matching
- Intro to Deep Learning-Based Motion Synthesis

3D Orientation & Rotation

Recall: Skeletal Motion



- "*Motion*": time-varying data
  - internal joint motion
    - w.r.t. default frame of each joint
      - the frame after applying joint offset to the parent frame
    - usually **rotation**
  - position and **orientation** of skeletal root
    - w.r.t. global frame
    - usually the pelvis part



https://youtu.be/PEhyWI8LGBY

Orientation vs. Rotation and Position vs. Translation

- **Orientation** & Position - *state*
  - Position: The state of being located.
  - **Orientation**: The state of being oriented. (angular position)

- **Rotation** & Translation - *movement*
  - Translation: Linear movement (difference btwn. positions)
  - **Rotation**: Angular movement (difference btwn. orientations)

- This relationship is analogous to *point* vs. *vector* in *coordinate-invariant geometric programming*.
  - point: position
  - vector: difference between two points

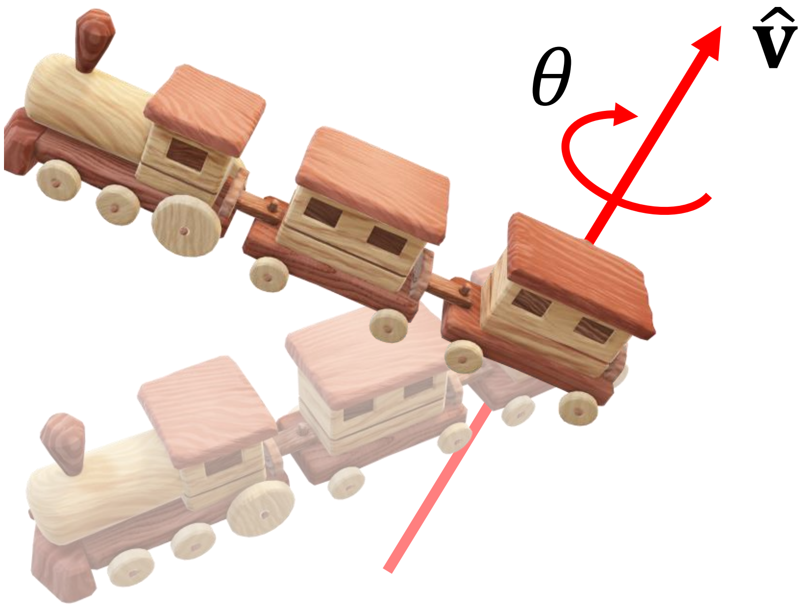
Describing 3D Rotation & Orientation

- Describing 3D rotation & orientation is not as intuitive as the 2D case.

- Several ways to describe 3D rotation and orientation
  - Euler angles
  - Rotation vector (Axis-angle)
  - Rotation matrices
  - Unit quaternions

3D Rotation

- For any 3D rotation, we can always find a fixed *axis* of rotation and an *angle* about the axis.



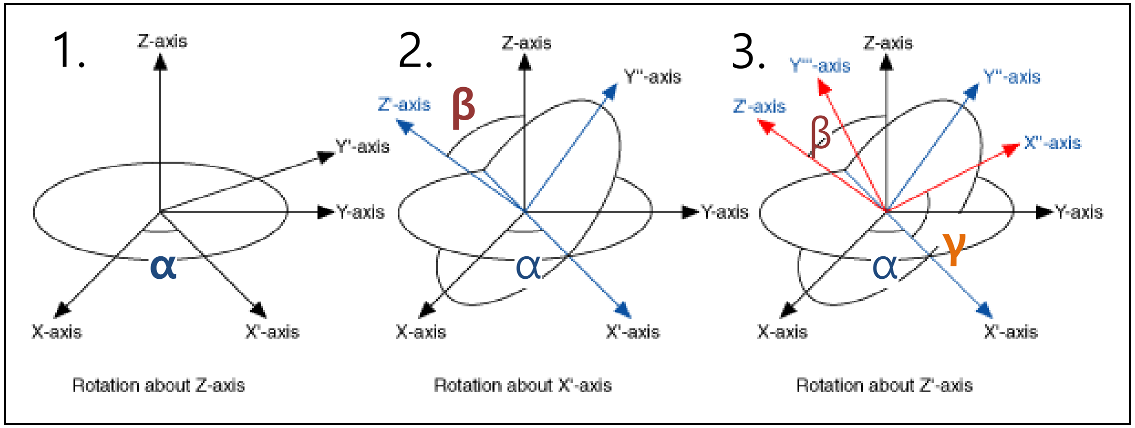
Euler Angles

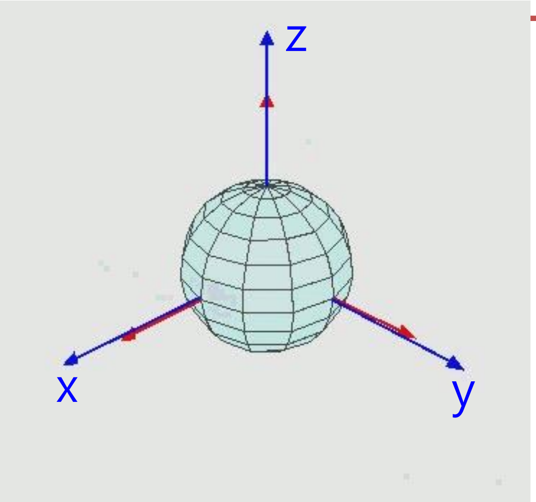
- Express any arbitrary 3D rotation using **three rotation angles about three principle axes.**

- Possible 12 combinations
  - XYZ, XYX, XZY, XZX
  - YZX, YZY, YXZ, YXY
  - ZXY, ZXZ, ZYX, ZYZ

  - (Combination is possible as long as the same axis does not appear consecutively.)

Example: ZXZ Euler Angles





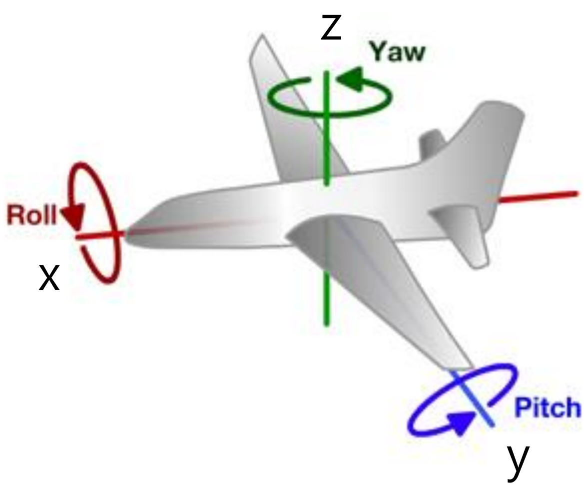
https://commons.wikimedia.org/wiki/File:Euler2a.gif

- 1. Rotate about Z-axis by α
- 2. Rotate about X-axis of the new frame by β
- 3. Rotate about Z-axis of the new frame by γ

$$R= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R= \quad R_z(\alpha) \quad R_x(\beta) \quad R_z(\gamma)$$

Example: Yaw-Pitch-Roll Convention (ZYX Euler Angles)



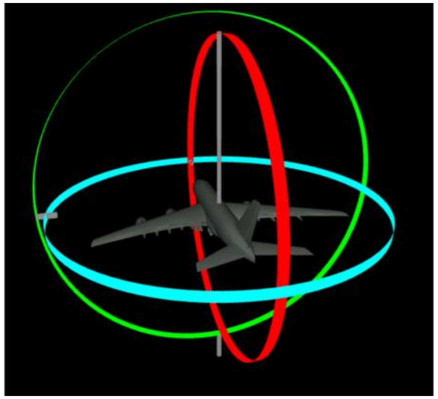
- Common for describing the orientation of aircrafts

- 1. Rotate about Z-axis by yaw angle
- 2. Rotate about Y-axis of the new frame by pitch angle
- 3. Rotate about X-axis of the new frame by roll angle

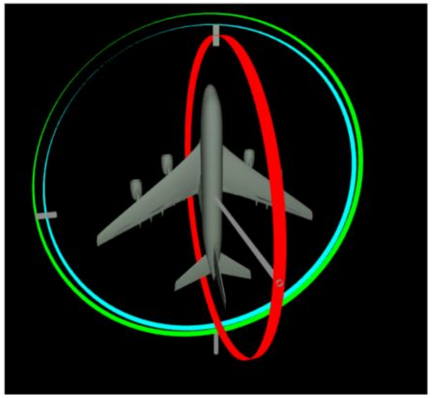
$$R = R_z(\text{yaw})\, R_y(\text{pitch})\, R_x(\text{roll})$$

Gimbal Lock

- Euler angles temporarily lose a DOF when the two axes are aligned.



Normal configuration:
The object can rotate freely.

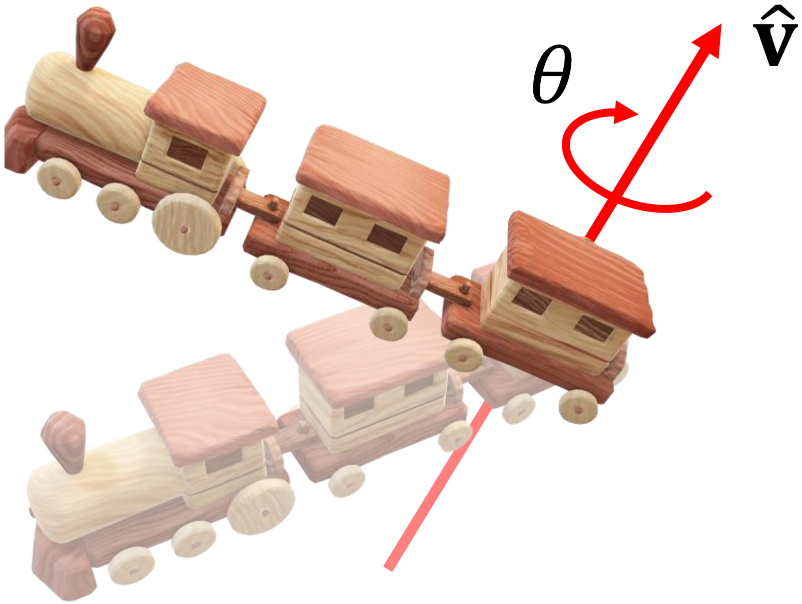


Gimbal lock configuration:
The object can not rotate in one direction.

Try:

- https://compsci290-s2016.github.io/CoursePage/Materials/EulerAnglesViz/index.html
- Set pitch to 90°

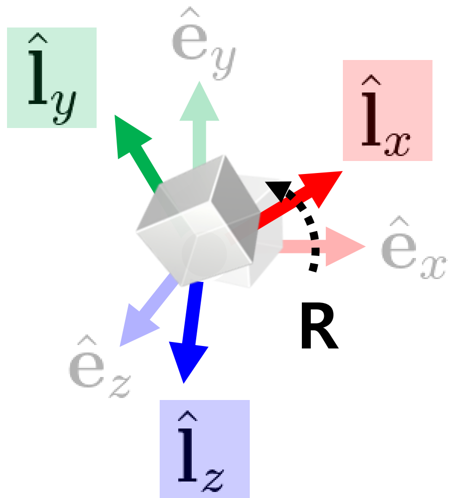
Rotation Vector (Axis-Angle)



$\hat{\mathbf{v}}$: rotation axis (unit vector)
θ: scalar angle

- Rotation vector: $\mathbf{v} = \theta \; \hat{\mathbf{v}} = (x, y, z)$
- Axis-Angle: $(\theta, \; \hat{\mathbf{v}})$

Rotation Matrix



$$\mathbf{R} = \begin{array}{c} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \\ \begin{array}{ccc} \hat{\mathbf{i}}_x & \hat{\mathbf{i}}_y & \hat{\mathbf{i}}_z \end{array} \end{array}$$

- A rotation matrix defines
  - **Orientation** of new rotated frame or,
  - **Rotation** from a global frame to be that rotated frame

Rotation Matrix

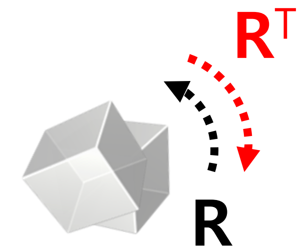
- A square matrix **R** is a rotation matrix if and only if

$$\boxed{1.\ \mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}} \ \&\& \ \boxed{2.\ \det(\mathbf{R}) = 1}$$

- A rotation matrix is an **orthogonal matrix with determinant 1**.
  - Sometimes it is called *special orthogonal matrix*
  - A set of **rotation matrices of size 3** forms a *special orthogonal group*, ***SO(3)***

Geometric Properties of Rotation Matrix

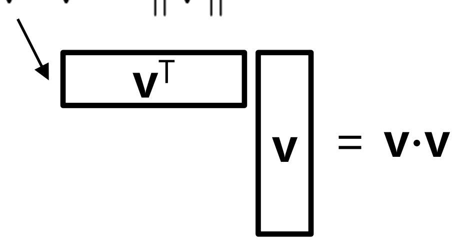
- $\mathbf{R}^\mathrm{T}$ is an inverse rotation of $\mathbf{R}$.
  - Because, $\mathbf{R}\mathbf{R}^T = \mathbf{I} \iff \mathbf{R}^{-1} = \mathbf{R}^T$



- $\mathbf{R}_1\mathbf{R}_2$ is a rotation matrix as well (composite rotation).
  - proof) $(\mathbf{R}_1\mathbf{R}_2)^T(\mathbf{R}_1\mathbf{R}_2) = \mathbf{R}_2^T\mathbf{R}_1^T\mathbf{R}_1\mathbf{R}_2 = \mathbf{R}_2^T\mathbf{R}_2 = \mathbf{I}$

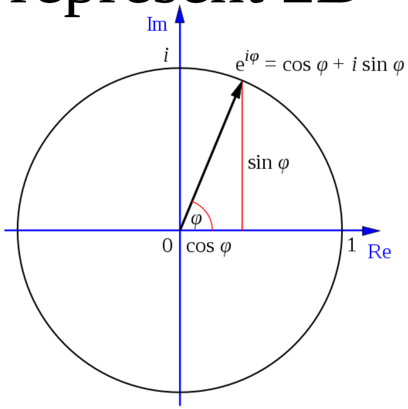
    and $\det(\mathbf{R}_1\mathbf{R}_2) = \det(\mathbf{R}_1)\cdot\det(\mathbf{R}_2) = 1$

- The length of vector $\mathbf{v}$ is not changed after applying a rotation matrix $\mathbf{R}$.
  - proof) $\|\mathbf{R}\mathbf{v}\|^2 = (\mathbf{R}\mathbf{v})^T(\mathbf{R}\mathbf{v}) = \mathbf{v}^T\mathbf{R}^T\mathbf{R}\mathbf{v} = \mathbf{v}^T\mathbf{v} = \|\mathbf{v}\|^2$



Quaternions

- Complex numbers can be used to represent 2D rotations.
- $z = x + yi$, where $i^2 = -1$
  - Euler's formula: $e^{i\varphi} = \cos\varphi + i\sin\varphi$



- Basic idea: Quaternion is its extension to 3D space.
- $q = w + xi + yj + zk$
- , where $i^2 = j^2 = k^2 = ijk = -1$

$$ij = k, \quad jk = i, \quad ki = j$$

$$ji = -k, \; kj = -i, \; ik = -j$$

Quaternions

- $q = w + xi + yj + zk$
  - w is called a *real part* (or *scalar part*).
  - $xi + yj + zk$ is called an *imaginary part* (or *vector part*).

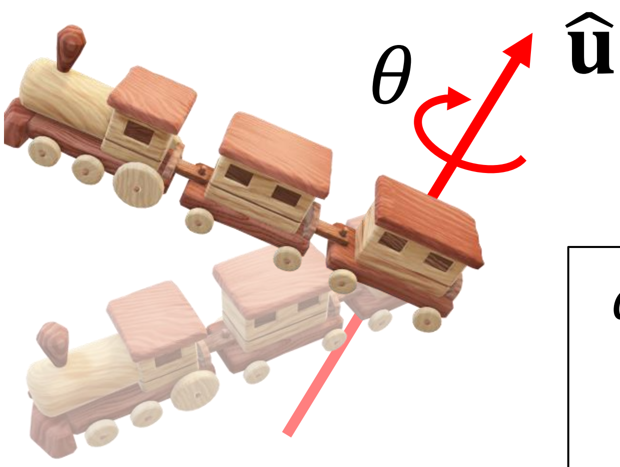
- Notation:

$$\begin{aligned} q &= w + xi + yj + zk \\ &= (w, x, y, z) \\ &= (w, \mathbf{v}) \end{aligned}$$

Unit Quaternions

- *Unit quaternions* represent 3D rotations.
- $q = w + ix + jy + kz,$
- , where $w^2 + x^2 + y^2 + z^2 = 1$

- Rotation about axis $\hat{\mathbf{u}}$ by angle θ:



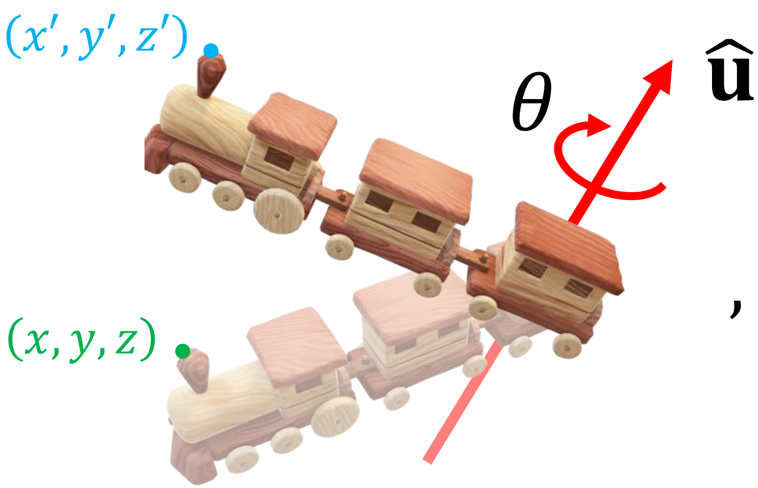
$$\mathbf{q} = \left(\cos\frac{\theta}{2}, \hat{\mathbf{u}}\sin\frac{\theta}{2}\right)$$

$$\boxed{\begin{aligned} q &= w + xi + yj + zk \\ &= (w, x, y, z) \\ &= (w, \mathbf{v}) \end{aligned}}$$

Unit Quaternions

- A 3D position (x, y, z) is represented as a *pure imaginary quaternion* $(0, x, y, z)$.

- If $\mathbf{p} = (0, x, y, z)$ is rotated about axis $\hat{\mathbf{v}}$ by angle θ, then the rotated position $\mathbf{p}' = (0, x', y', z')$ is:



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

, where $\mathbf{q} = \left(\cos\frac{\theta}{2}, \hat{\mathbf{u}}\sin\frac{\theta}{2}\right)$

Unit Quaternions

Identity	$\mathbf{q} = (1,0,0,0)$
Multiplication	$\mathbf{q}_1\mathbf{q}_2 = (w_1, \mathbf{v}_1)(w_2, \mathbf{v}_2)$ $= (w_1 w_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, w_1\mathbf{v}_2 + w_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)$
Inverse	$\mathbf{q}^{-1} = (w, -x, -y, -z)/(w^2 + x^2 + y^2 + z^2)$ $= (-w, x, y, z)/(w^2 + x^2 + y^2 + z^2)$

- $\mathbf{q}_1\mathbf{q}_2$: rotate by $\mathbf{q}_1$ then $\mathbf{q}_2$ w.r.t. local frame

  or rotate by $\mathbf{q}_2$ then $\mathbf{q}_1$ w.r.t. global frame

- $\mathbf{p}' = \mathbf{q}_1\mathbf{q}_2\mathbf{p}(\mathbf{q}_1\mathbf{q}_2)^{-1} = \mathbf{q}_1(\mathbf{q}_2\mathbf{p}\mathbf{q}_2{}^{-1})\mathbf{q}_1{}^{-1}$

Which Representation to Use?

- General recommendation: Use **rotation matrices** or **unit quaternions**.
- Because they provide accurate "addition", "subtraction", and interpolation of rotations.
- In addition, Euler angles have the gimbal lock issue.

"Addition" of Rotations

- Rotation matrix, Unit quaternion:
  - $\mathbf{R}_1\mathbf{R}_2$ (or $\mathbf{q}_1\mathbf{q}_2$)
  - Rotate by $\mathbf{R}_1$(or $\mathbf{q}_1$), then by $\mathbf{R}_2$(or $\mathbf{q}_2$) w.r.t. local frame.
  - (Element-wise addition does NOT even produce a rotation matrix or unit quaternion.)

- Euler angles:
  - $(\alpha_1, \beta_1, \gamma_1) + (\alpha_2, \beta_2, \gamma_2) = (\alpha_1 + \alpha_2, \beta_1 + \beta_2, \gamma_1 + \gamma_2)$?
  - Does **NOT** mean rotate by $(\alpha_1, \beta_1, \gamma_1)$, then by $(\alpha_2, \beta_2, \gamma_2)$!

- Rotation vector:
  - $\mathbf{v}_1 + \mathbf{v}_2$?
  - Does **NOT** mean rotate by $\mathbf{v}_1$, then by $\mathbf{v}_2$!

"Subtraction" of Rotations

- Rotation matrix, Unit quaternion:
  - ${\mathbf{R}_1}^T\mathbf{R}_2$(or ${\mathbf{q}_1}^{-1}\mathbf{q}_2$)
  - Rotational difference: A rotation matrix that rotate a frame $\mathbf{R}_1$(or $\mathbf{q}_1$) to be coincident with the frame $\mathbf{R}_2$(or $\mathbf{q}_2$) when applied w.r.t. the frame $\mathbf{R}_1$(or $\mathbf{q}_1$)
  - Because $\mathbf{R}_1\left({\mathbf{R}_1}^T\mathbf{R}_2\right) = \mathbf{R}_2$
  - (Element-wise subtraction does NOT even produce a rotation matrix or unit quaternion.)

- Euler angles:
  - $(\alpha_2, \beta_2, \gamma_2) - (\alpha_1, \beta_1, \gamma_1) = (\alpha_2 - \alpha_1, \beta_2 - \beta_1, \gamma_2 - \gamma_1)$?
  - Does **NOT** mean difference between rotation $(\alpha_1, \beta_1, \gamma_1)$ and $(\alpha_2, \beta_2, \gamma_2)$!

- Rotation vector:
  - $\mathbf{v}_2 - \mathbf{v}_1$?
  - Does **NOT** mean difference between rotation $\mathbf{v}_1$ and $\mathbf{v}_2$!

Interpolation of Rotations

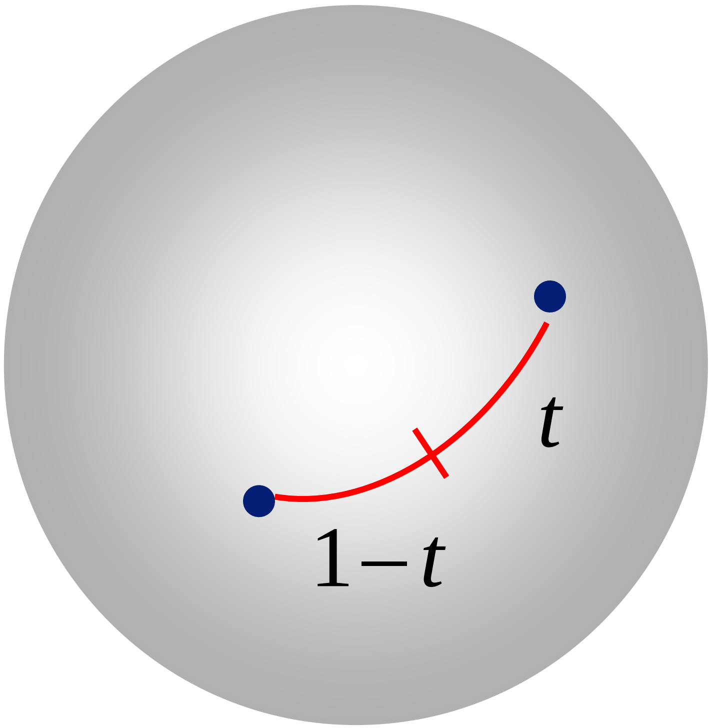
- Can we just linear interpolate each element of
  - Euler angles
  - Rotation vector
  - Rotation matrix
  - Unit quaternion
- ?

- → No!

- The right answer: **slerp**

Slerp

- **Slerp** [Shoemake 1985]
  - **S**pherical **l**inear int**erp**olation
  - Linear interpolation of two orientations



"t" refers power, not transpose

$$\begin{aligned}\text{slerp}(\mathbf{R}_1, \mathbf{R}_2, t) &= \mathbf{R}_1(\mathbf{R}_1^T\mathbf{R}_2)^t \\ &= \mathbf{R}_1\exp(\mathrm{t}\cdot\log(\mathbf{R}_1^T\mathbf{R}_2))\end{aligned}$$

Slerp

- $\mathrm{slerp}(\mathbf{R}_1, \mathbf{R}_2, t) = \mathbf{R}_1(\mathbf{R}_1^T\mathbf{R}_2)^t$
  $= \mathbf{R}_1 \exp(\mathrm{t} \cdot \log(\mathbf{R}_1^T\mathbf{R}_2))$
  - exp(): **rotation vector** to **rotation matrix**
  - log(): **rotation matrix** to **rotation vector**

- Implication
  - $\mathbf{R}_1{}^T\mathbf{R}_2$: difference between orientation $\mathbf{R}_1$ and $\mathbf{R}_2$ ($\mathbf{R}_2$(-)$\mathbf{R}_1$)
  - $\mathbf{R}^t$: scaling rotation (scaling rotation angle)
  - $\mathbf{R}_a\mathbf{R}_b$: add rotation $\mathbf{R}_b$ to orientation $\mathbf{R}_a$ ($\mathbf{R}_a$(+)$\mathbf{R}_b$)

Exp & Log

- **Exp** (exponential): **rotation vector** to **rotation matrix**
  - Given normalized rotation axis $u=(u_x,u_y,u_z)$, rotation angle θ

$$R = \begin{bmatrix} \cos\theta + u_x^2\left(1-\cos\theta\right) & u_x u_y\left(1-\cos\theta\right) - u_z\sin\theta & u_x u_z\left(1-\cos\theta\right) + u_y\sin\theta \\ u_y u_x\left(1-\cos\theta\right) + u_z\sin\theta & \cos\theta + u_y^2\left(1-\cos\theta\right) & u_y u_z\left(1-\cos\theta\right) - u_x\sin\theta \\ u_z u_x\left(1-\cos\theta\right) - u_y\sin\theta & u_z u_y\left(1-\cos\theta\right) + u_x\sin\theta & \cos\theta + u_z^2\left(1-\cos\theta\right) \end{bmatrix}$$

(Rodrigues' rotation formula)

- **Log** (logarithm): **rotation matrix** to **rotation vector**

Given rotation matrix **R**, compute axis **v** and angle θ

$$\theta = \cos^{-1}((R_{11} + R_{22} + R_{33} - 1)/2)$$
$$v_1 = (R_{32} - R_{23})/(2\sin\theta)$$
$$v_2 = (R_{13} - R_{31})/(2\sin\theta)$$
$$v_3 = (R_{21} - R_{12})/(2\sin\theta)$$

→ But this fomula has a singularity at θ=kπ, where k is an integer.

Algorithm for Log

Algorithm for Computing the Logarithm of a Rotation Matrix

Objective: Given $R \in SO(3)$, find $\omega \in \mathbb{R}^3$, $\|\omega\| = 1$, and $\theta \in [0, \pi]$ such that

$$R = e^{[\omega]\theta} = I + \sin\theta\,[\omega] + (1 - \cos\theta)[\omega]^2. \qquad (3.62)$$

(i) If $\operatorname{tr} R = 3$, then set $\omega = 0$, $\theta = 0$.

(ii) If $\operatorname{tr} R = -1$, then set $\theta = \pi$, and ω to any of the three following vectors that is nonzero:

$$\omega = \frac{1}{\sqrt{2(1 + r_{33})}} \begin{bmatrix} r_{13} \\ r_{23} \\ 1 + r_{33} \end{bmatrix} \qquad (3.63)$$

or

$$\omega = \frac{1}{\sqrt{2(1 + r_{22})}} \begin{bmatrix} r_{12} \\ 1 + r_{22} \\ r_{32} \end{bmatrix} \qquad (3.64)$$

or

$$\omega = \frac{1}{\sqrt{2(1 + r_{11})}} \begin{bmatrix} 1 + r_{11} \\ r_{21} \\ r_{31} \end{bmatrix}. \qquad (3.65)$$

(iii) Otherwise set $\theta = \cos^{-1}\left(\frac{\operatorname{tr} R - 1}{2}\right) \in [0, \pi)$ and $[\omega] = \frac{1}{2\sin\theta}(R - R^T)$.

See section 3.2.3.3 of "Modern Robotics" for detail: http://hades.mech.northwestern.edu/images/2/25/MR-v2.pdf

Exp & Log

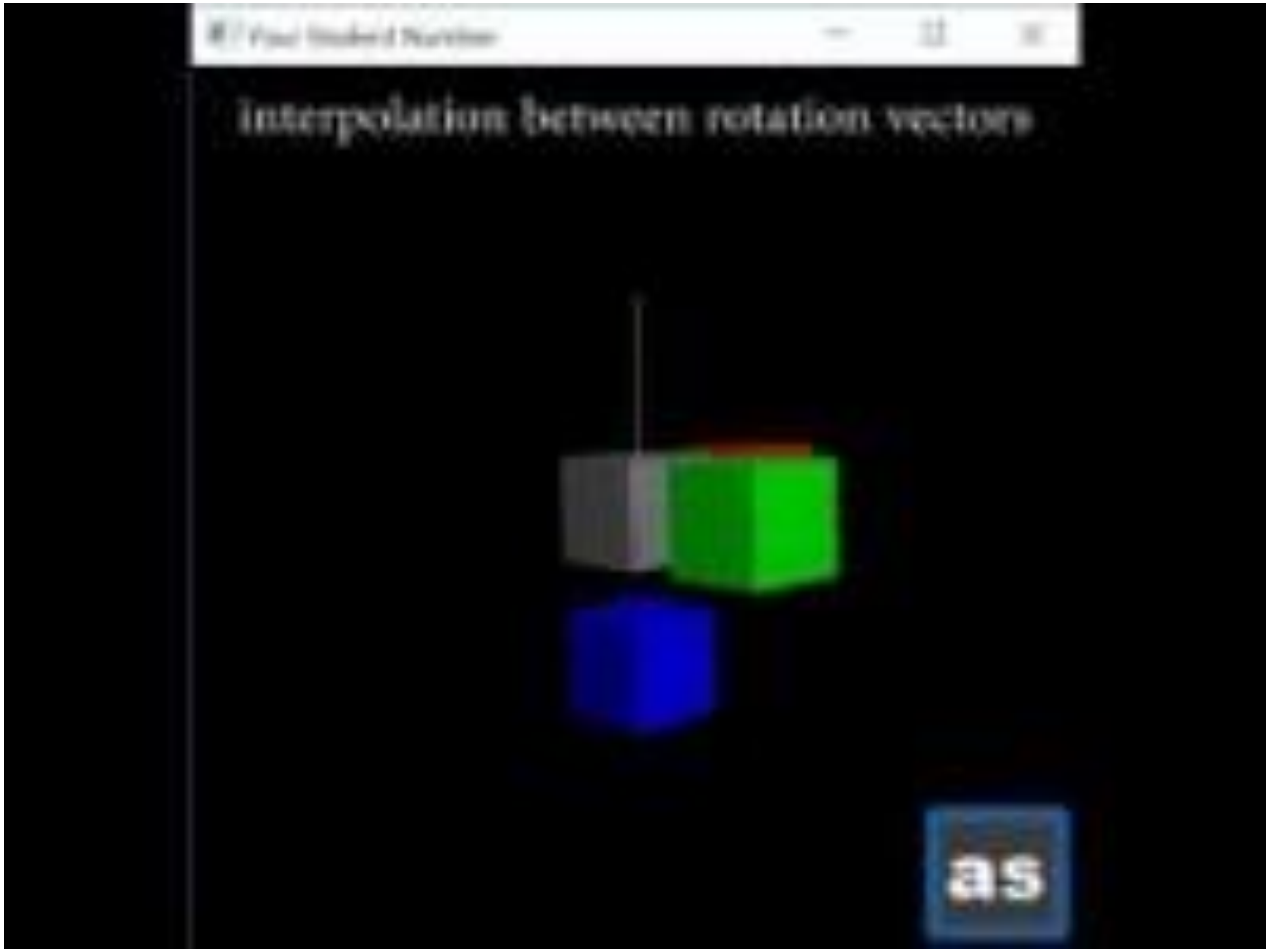
- However, you can just use `scipy.spatial.transform.Rotation` for exp() and log().
  - https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.transform.Rotation.html

Slerp

- Quaternion slerp:
  - $\mathrm{slerp}(\mathbf{q}_1, \mathbf{q}_2, t) = \mathbf{q}_1(\mathbf{q}_1^{-1}\mathbf{q}_2)^t$
- Geometric slerp (equivalent):
  - $\mathrm{slerp}(\mathbf{q}_1, \mathbf{q}_2, t) = \frac{\sin((1-t)\varphi)}{\sin\varphi}\mathbf{q}_1 + \frac{\sin(t\varphi)}{\sin\varphi}\mathbf{q}_2$
    - φ: the angle subtended by the arc ($\cos\varphi = \mathbf{q}_1 \cdot \mathbf{q}_2$)
- No slerp for Euler angles or rotation vector representation!
  - They need to be converted to rotation matrix or unit quaternions to slerp.

Interpolation of Rotations

- Start orientation (ZYX Euler angles): Rz(-90) Ry(90) Rx(0)
- End orientation (ZYX Euler angles): Rz(0) Ry(0) Rx(90)



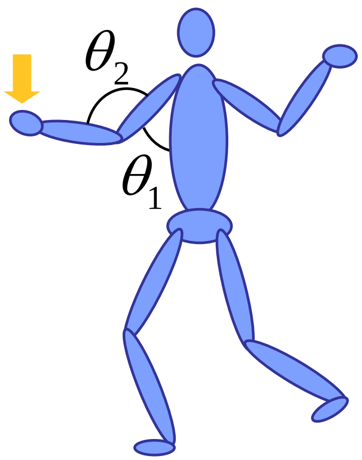
https://youtu.be/Y02MAWKmfGU

Limb IK

Inverse Kinematics

- Given the desired position and possibly orientation of the end effector,

- What is the set of joint orientations for other joints to generate the desired end effector configuration?

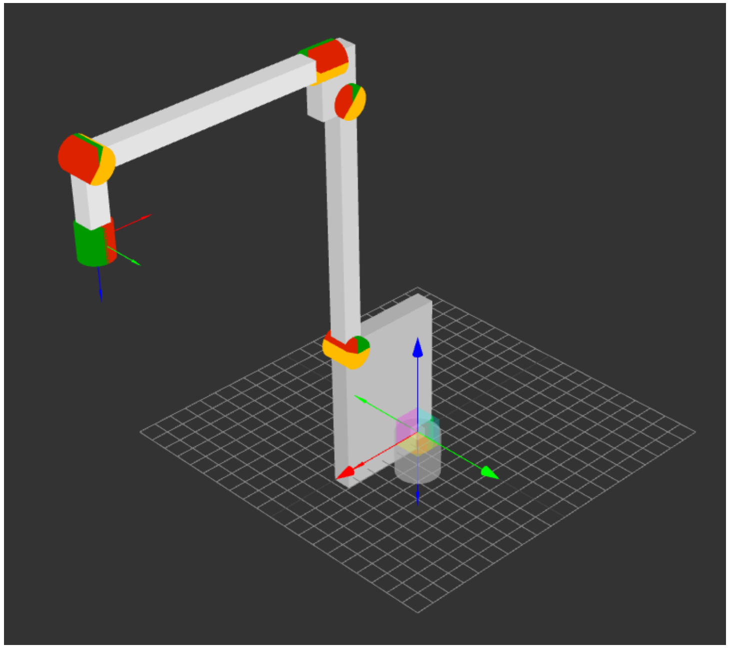


$$\theta_i = \mathrm{F}^{-1}(\mathbf{p}, \mathbf{q})$$

Inverse Kinematics

: Given the position & orientation of end-effector, compute joint angles

[Practice] FK / IK Online Demo



http://robot.glumb.de/

- Forward kinematics : Open “angles” menu and change values
- Inverse kinematics : Move the end-effector position by mouse dragging

Inverse Kinematics

- More challenging than forward kinematics
- An IK problem might have:
  - Multiple solutions
  - Infinitely many solutions
  - No solutions
- IK problems generally do not have analytic solutions.

Inverse Kinematics

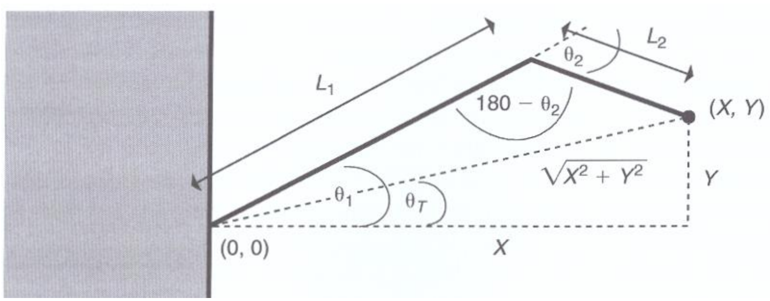
- Analytic solver
  - **Limb IK (Foot IK)**

- Numerical solver
  - Jacobian Transpose IK
  - Pseudo-inverse Jacobian
  - Cyclic coordinate descent (CCD)
  - ...

Simple Analytic Solution

- For sufficiently simple mechanisms (two links in 2D plane), the joint angles can be analytically calculated.

- For example, given L_1, L_2, and (X,Y),
- What is θ_1 and θ_2?



$$\operatorname{acos}(\theta_T) = \frac{X}{\sqrt{X^2 + Y^2}}$$

$$\theta_T = \operatorname{acos}\left(\frac{X}{\sqrt{X^2 + Y^2}}\right)$$

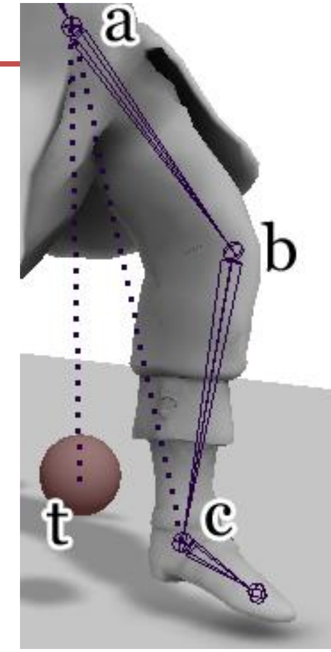
$$\cos(\theta_1 - \theta_T) = \frac{L_1^2 + X^2 + Y^2 - L_2^2}{2L_1\sqrt{X^2 + Y^2}} \qquad \text{(cosine rule)}$$

$$\theta_1 = \operatorname{acos}\left(\frac{L_1^2 + X^2 + Y^2 - L_2^2}{2L_1\sqrt{X^2 + Y^2}}\right) + \theta_T$$

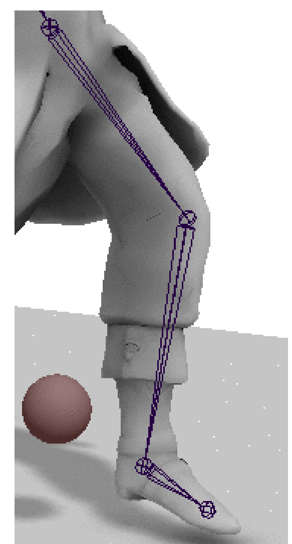
$$\cos(180 - \theta_2) = -\cos(\theta_2) = \frac{L_1^2 + L_2^2 - (X^2 + Y^2)}{2L_1L_2} \qquad \text{(cosine rule)}$$

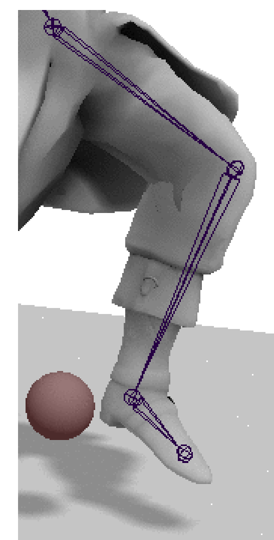
$$\theta_2 = \operatorname{acos}\left(\frac{L_1^2 + L_2^2 - X^2 + Y^2}{2L_1L_2}\right)$$

Limb IK



- Actually, this simple analytic solution alone is not very useful.
- Instead, limb IK (or foot IK) is often used to correct leg joints so that the feet can touch the ground.
- 두 단계로 나누어서 생각 (t: target position)
- 1단계: line ac와 line at의 길이가 같아지도록 joint a와 b의 각도를 조절
- 2단계: line ac와 line at가 일치하도록 joint a의 각도를 조절

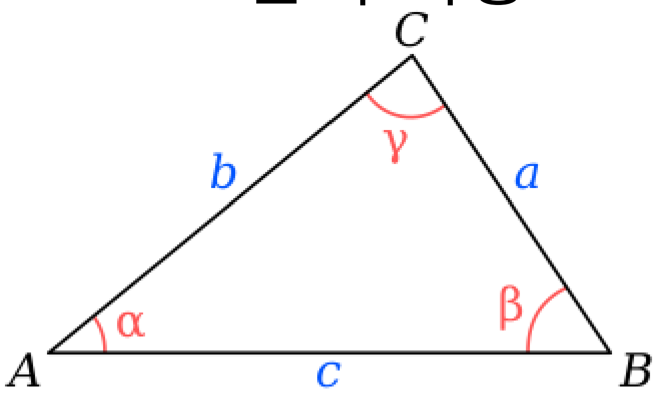




http://theorangeduck.com/page/simple-two-joint

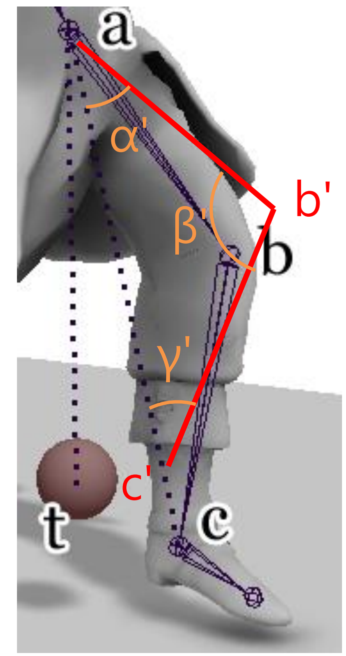
1단계: line ac와 line at의 길이가 같아지도록 joint a와 b의 각도를 조절

- cosine 법칙 이용



$$c^2 = a^2 + b^2 - 2ab\cos C$$

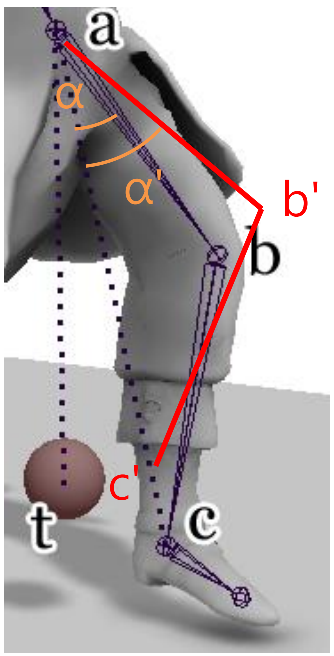
$$\cos C = \frac{a^2 + b^2 - c^2}{2ab}$$



- c'을 ac'길이==at길이 가 되도록 하는 ac 상의 위치, b'을 그때의 관절 b 위치라 하면,
- (모든 변의 길이를 구할 수 있기 때문에) cosine 법칙을 이용해서 α', β', γ'을 계산 가능

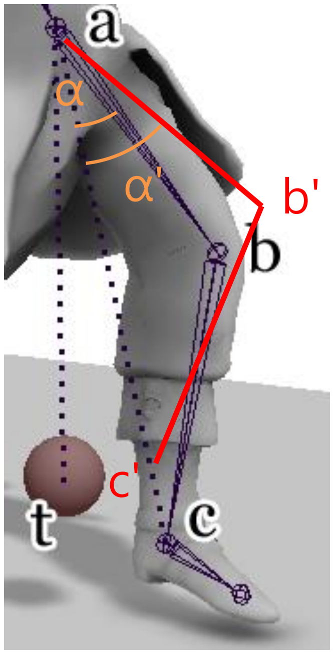
1단계: line ac와 line at의 길이가 같아지도록 joint a와 b의 각도를 조절

- Joint a: (ab × ac의 unit vector)인 $axis^{\{g\}}$에 대해 (α - α')만큼 회전하는 rotation $R_{diff}^{\{g\}}$ 을 현재 joint a의 orientation 에 더하면 (rotation matrix의 곱) 됨.
  - $R_{diff}^{\{g\}}$: a, b, c의 global position으로 구한 $axis^{\{g\}}$에 대한 회전을 뜻하는 rotation matrix는 global frame {g}에서 적용되는 회전을 의미하게 된다.
  - 하지만 human motion에서 joint a의 orientation은 일반적으로 joint a's default frame {ad} (parent joint frame에서 link transform만 적용된 후의 frame) 대해서 표현되므로,
  - 모션으로부터 joint a의 orientation을 바로 구하면 {ad}에 대해 표현된 $R_a^{\{ad\}}$ 을 얻게 되며, 새로운 orientation을 설정할 때도 {ad}에 대해 표현된 new $R_a^{\{ad\}}$ 를 설정해야 한다.
  - $R_{diff}^{\{g\}}$와 $R_a^{\{ad\}}$를 어떻게 더해서 new $R_a^{\{ad\}}$를 구할 수 있을까?



1단계: line ac와 line at의 길이가 같아지도록 joint a와 b의 각도를 조절

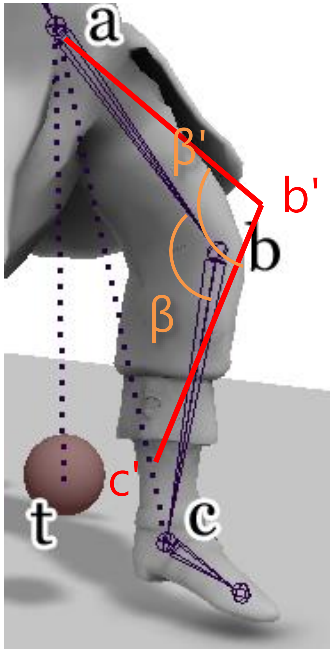
- 방법 1: local frame에 대한 rotation을 오른쪽에 곱함
  - forward kinematics로 joint a의 global orientation $R_a^{\{g\}}$를 구해서
  - $R_a^{\{g\}\,-1}$를 $axis^{\{g\}}$ 에 곱해서 joint a's frame {a}에 대해 표현된 $axis^{\{a\}}$로 만들고 (joint a's default frame {ad}가 아님에 주의),
  - 이것에 대해 (α - α') 만큼 회전하는 $R_{diff}^{\{a\}}$ 을 구해서
  - new $R_a^{\{ad\}} = R_a^{\{ad\}} R_{diff}^{\{a\}}$
  - new $R_a^{\{ad\}}$ 를 joint a의 orientation으로 설정하면 됨.



- 방법 2: global frame에 대한 rotation을 왼쪽에 곱함
  - new $R_a^{\{g\}} = R_{diff}^{\{g\}} R_a^{\{g\}}$
  - new $R_a^{\{g\}}$로부터 new $R_a^{\{ad\}}$를 계산하여 이를 joint a의 orientation으로 설정하면 됨.

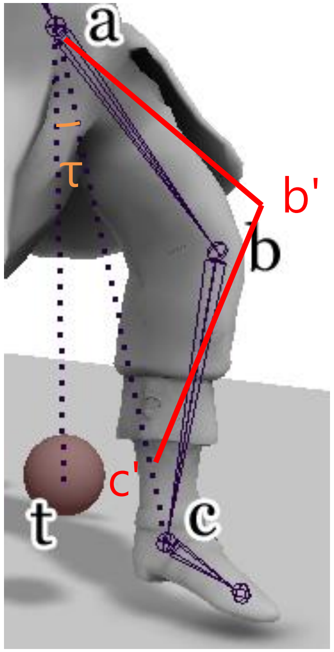
1단계: line ac와 line at의 길이가 같아지도록 joint a와 b의 각도를 조절

- Joint b: 마찬가지 방법으로 β가 β'로 바뀌도록 하는 R_{diff} 를 구해서 $R_b^{\{bd\}}$ 를 업데이트 할 수 있음.



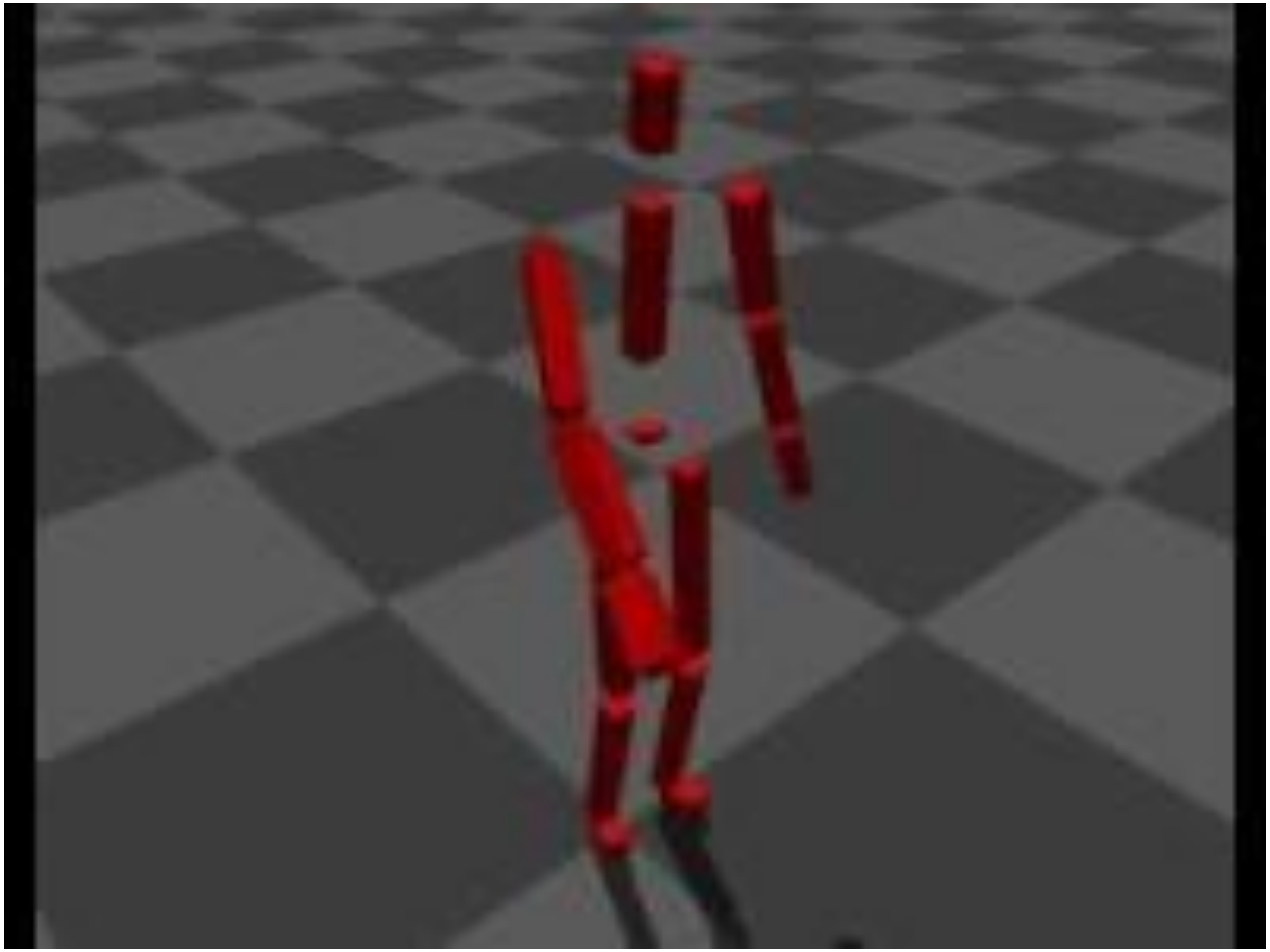
2단계: line ac와 line at가 일치하도록 joint a의 각도를 조절

- Joint a: (ac' × at의 unit vector)인 axis$^{\{g\}}$에 대해 τ 만큼 회전하는 rotation $R_{diff}^{\{g\}}$ 을 현재 joint a의 orientation 에 앞과 마찬가지 방식으로 더하면 됨.
  - τ 는 cosine 법칙을 이용해서 구할 수도 있지만, 벡터 내적 (inner product)을 이용해 구할 수도 있다.



- 참고: http://theorangeduck.com/page/simple-two-joint
  - (unit quaternion을 이용해 기술되어 있음)

Example - Human figure



https://youtu.be/bt3hTDwiTi0

Posture / Motion Difference

Posture Difference

- "Motion": time-varying data
  - internal joint orientation: $\mathbf{R}_1(t), ..., \mathbf{R}_n(t)$
  - position and orientation of skeletal root: $\mathbf{p}_0(t), \mathbf{R}_0(t)$

- Posture (pose): "motion" at a single frame

Posture Difference

- Let's think about "posture difference" **d**.
- Valid operations (**o**: posture, **d**: difference):

$\mathbf{o}_1 + \mathbf{d} = \mathbf{o}_2$	
$\mathbf{o}_2 - \mathbf{o}_1 = \mathbf{d}$	
$c * \mathbf{d}_1 = \mathbf{d}_2$ (scalar c)	
$\mathbf{d}_1 + \mathbf{d}_2 = \mathbf{d}_3$	

Posture Difference

- Let's think about "posture difference" **d**.
- Valid operations (**o**: posture, **d**: difference):

$\mathbf{o}_1 + \mathbf{d} = \mathbf{o}_2$	$\mathbf{p}_0^{o1} + \mathbf{p}_0^{d} = \mathbf{p}_0^{o2}$ $\mathbf{R}_i^{o1}\,\mathbf{R}_i^{d} = \mathbf{R}_i^{o2}$
$\mathbf{o}_2 - \mathbf{o}_1 = \mathbf{d}$	$\mathbf{p}_0^{o2} - \mathbf{p}_0^{o1} = \mathbf{p}_0^{d}$ $(\mathbf{R}_i^{o1})^T\,\mathbf{R}_i^{o2} = \mathbf{R}_i^{d}$
$c * \mathbf{d}_1 = \mathbf{d}_2$ (scalar c)	$c * \mathbf{p}_0^{d1} = \mathbf{p}_0^{d2}$ $\exp(\ c * \log(\mathbf{R}_i^{d1})\ ) = \mathbf{R}_i^{d2}$
$\mathbf{d}_1 + \mathbf{d}_2 = \mathbf{d}_3$	$\mathbf{p}_0^{d1} + \mathbf{p}_0^{d2} = \mathbf{p}_0^{d3}$ $\mathbf{R}_i^{d1}\,\mathbf{R}_i^{d2} = \mathbf{R}_i^{d3}$

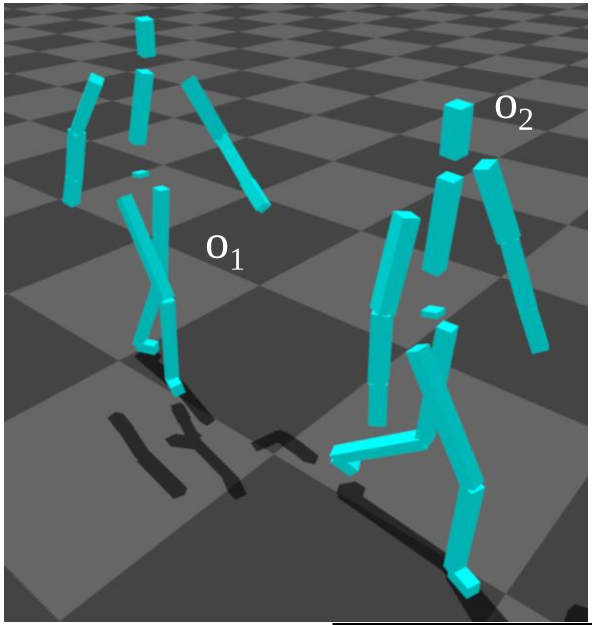
- Invalid operations:
  - $o_1 + o_2 = ?$
  - $c * o_1 = ?$
  - $d + o_1 = ?$ (not commutative)

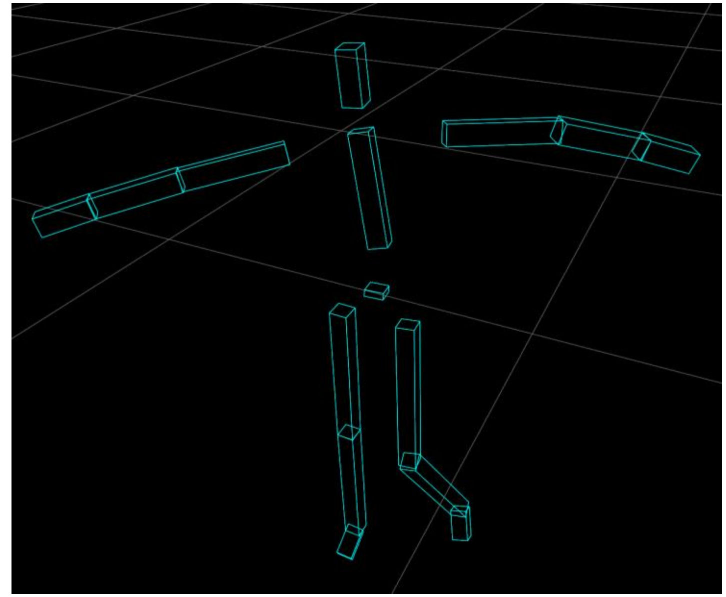
Motion Difference

- Just compute posture differences between postures at the same frame for every frame of both motions.
  - for every t,
  - $\mathbf{m}_1(t) + \mathbf{d}(t) = \mathbf{m}_2(t)$
  - $\mathbf{m}_2(t) - \mathbf{m}_1(t) = \mathbf{d}(t)$
  - $c * \mathbf{d}_1(t) = \mathbf{d}_2(t)$ (scalar c)
  - $\mathbf{d}_1(t) + \mathbf{d}_2(t) = \mathbf{d}_3(t)$

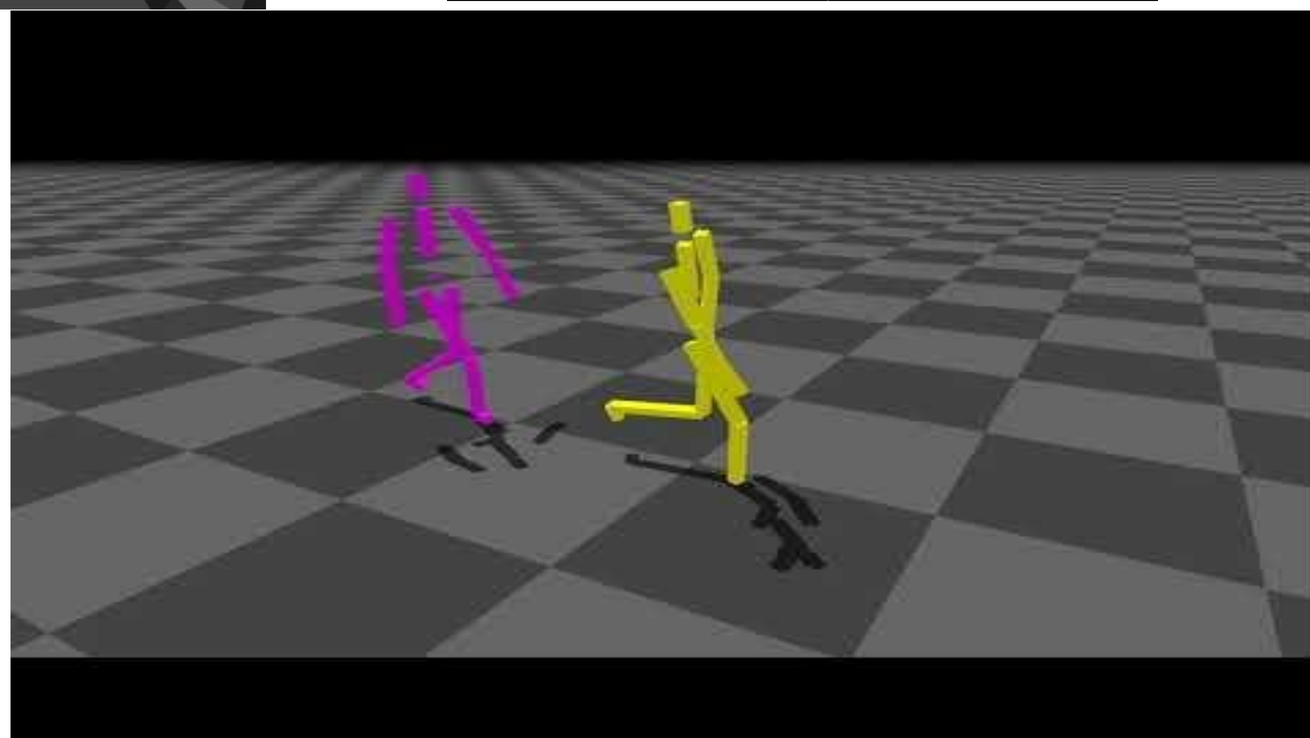
- Motion difference is sometimes called *motion displacement mapping*.

$\mathbf{d} = o_2 - o_1$





$\mathbf{m}_1(t) + \mathbf{d} = \mathbf{m}_2(t)$

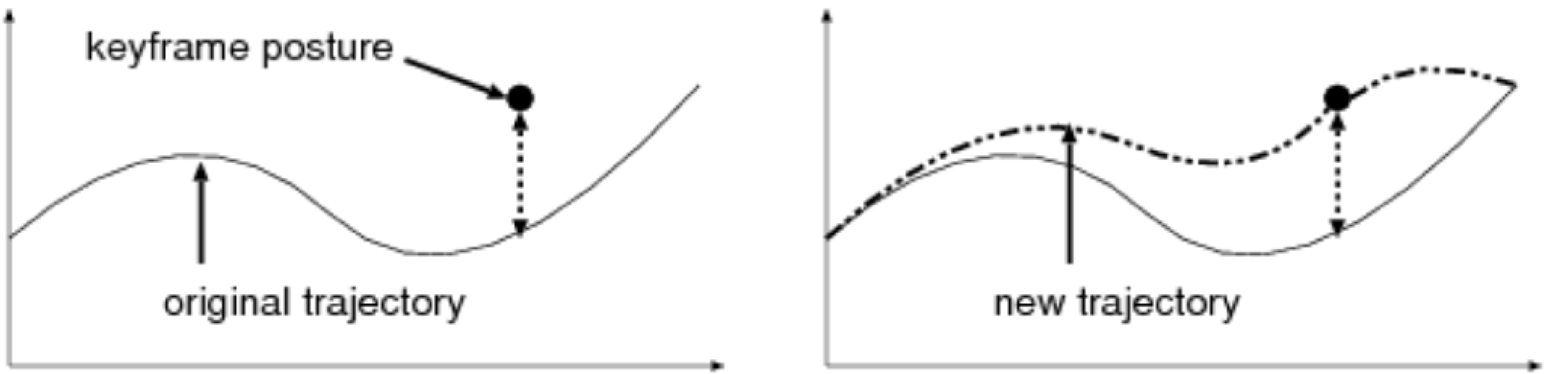


https://youtu.be/skmepI3ZnR4

Motion Warping

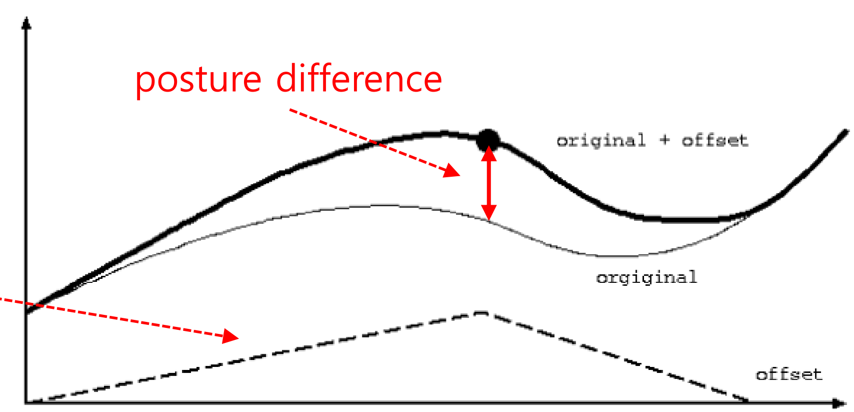
Motion Warping

- Adding offset to the data so the constraint is satisfied



Warped Motion = original motion + offset

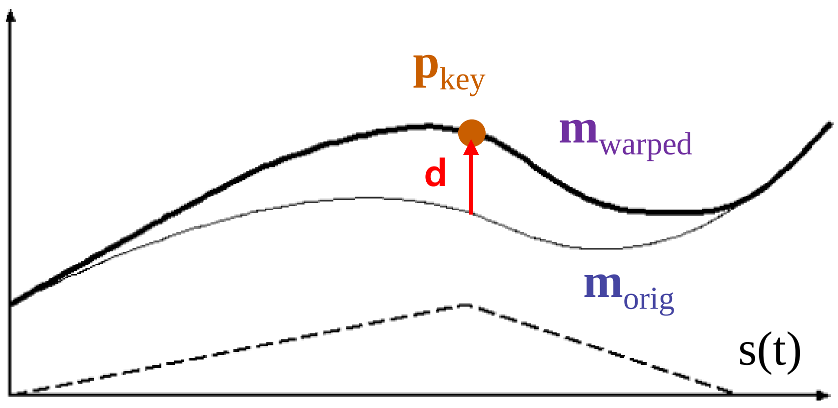
Offset can be a simple 1D motion



http://homepages.inf.ed.ac.uk/tkomura/cav/presentation6.pdf

Motion Warping

- Set the keyframe posture so that the constraint is satisfied.



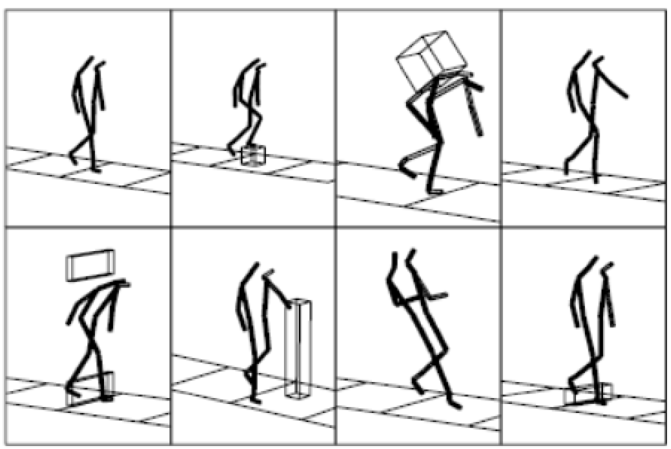
$$\mathbf{d} = \mathbf{p}_{key}(T_{key}) - \mathbf{m}_{orig}(T_{key})$$

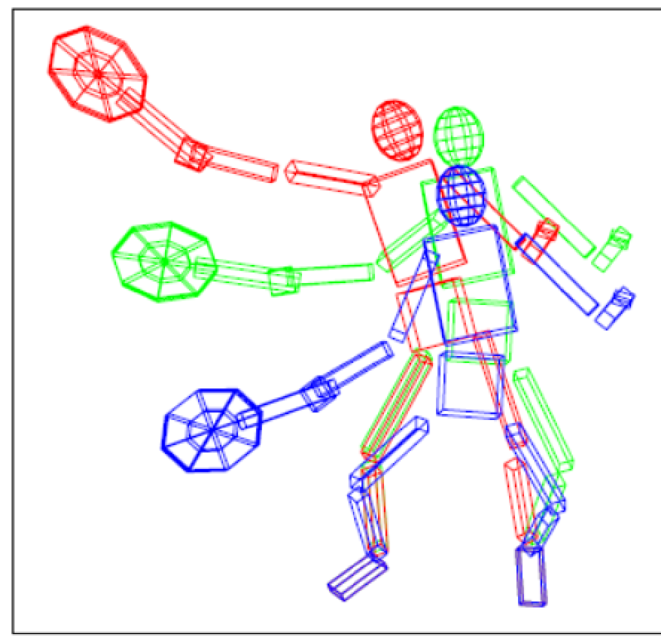
($\mathbf{p}_{key}$: key frame posture at time T_{key})

$$\mathbf{m}_{warped}(t) = \mathbf{m}_{orig}(t) + s(t) \cdot \mathbf{d}$$

Motion Warping

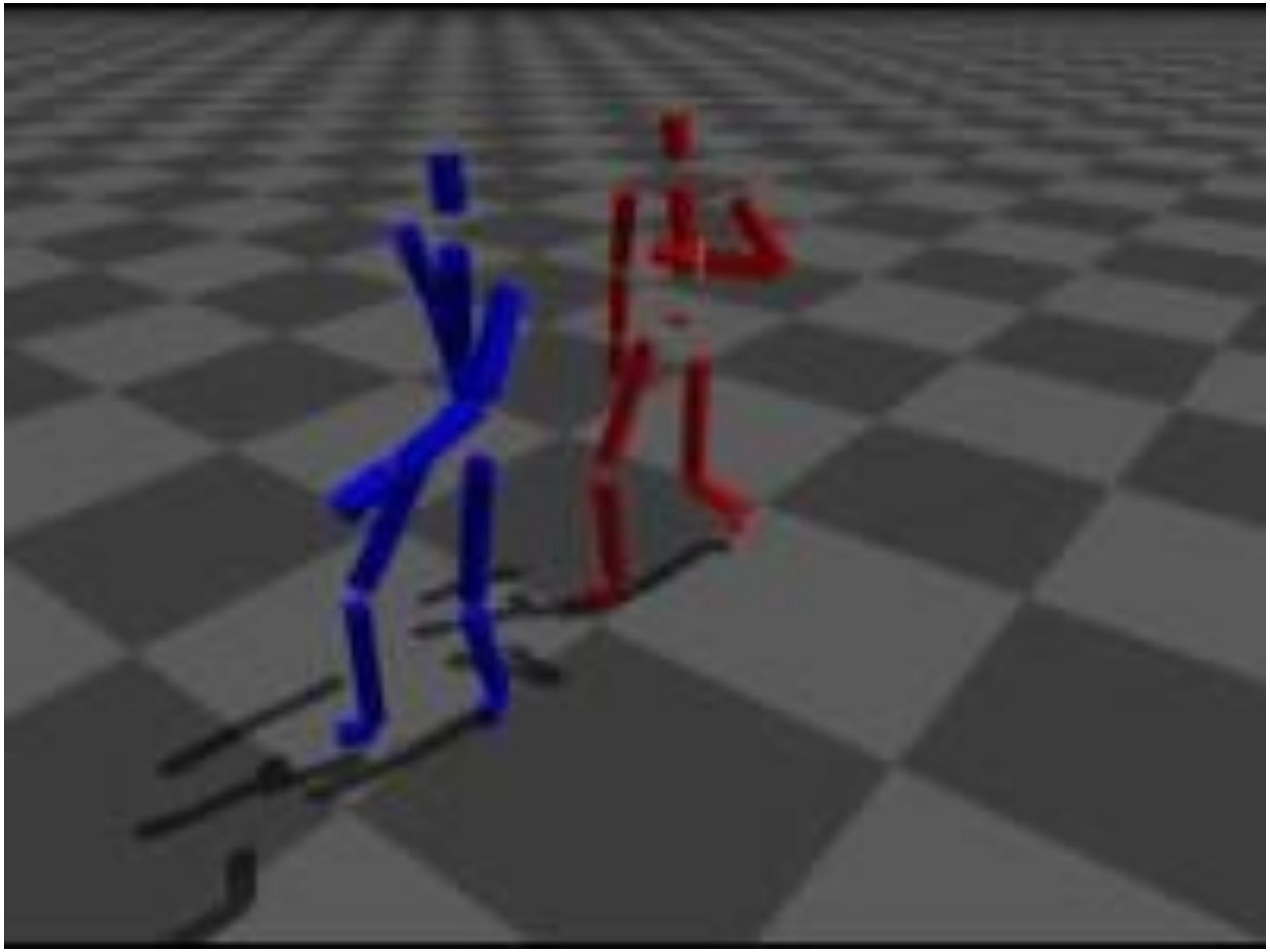
- Edit the captured motion a little bit so that it satisfies the requirements
  - Effective for changing the location the hand or the foot passes





http://homepages.inf.ed.ac.uk/tkomura/cav/presentation6.pdf

Example - Motion Warping with Limb IK



https://youtu.be/VH4QuV2mFcg

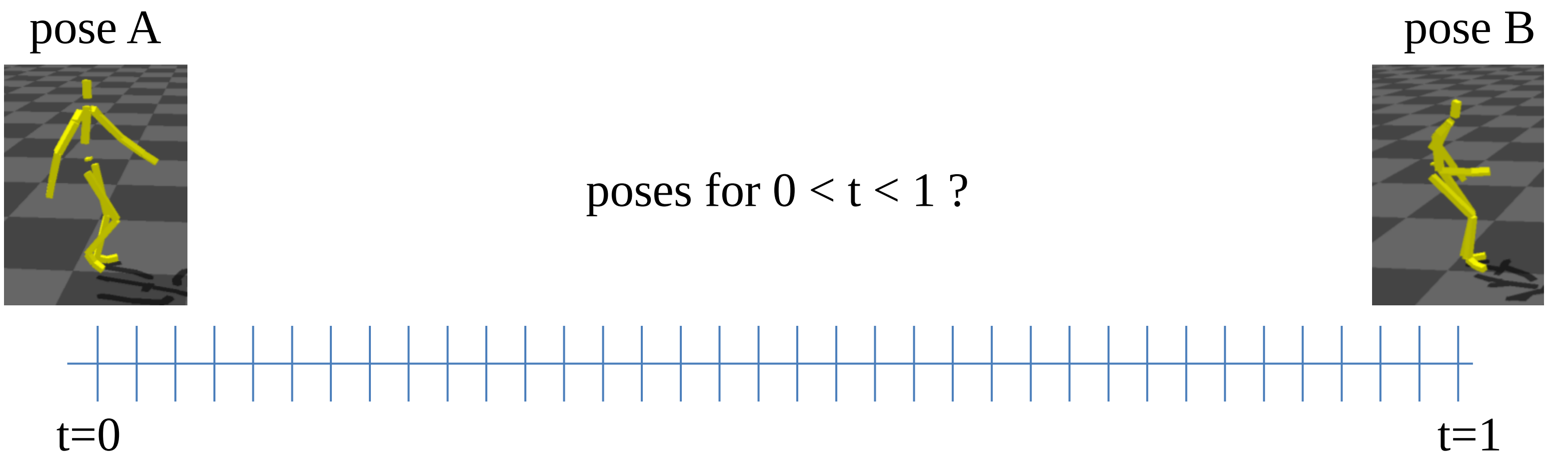
Discussion #1

- Go to https://www.slido.com/
- Join #**pbl-ys**
- Click "Polls"

- 아래의 형식으로 적어서 제출할 것.
  - **이름**: 자신의 의견 blah blah ...

Intro to More Motion Editing Techniques

Interpolation of Postures

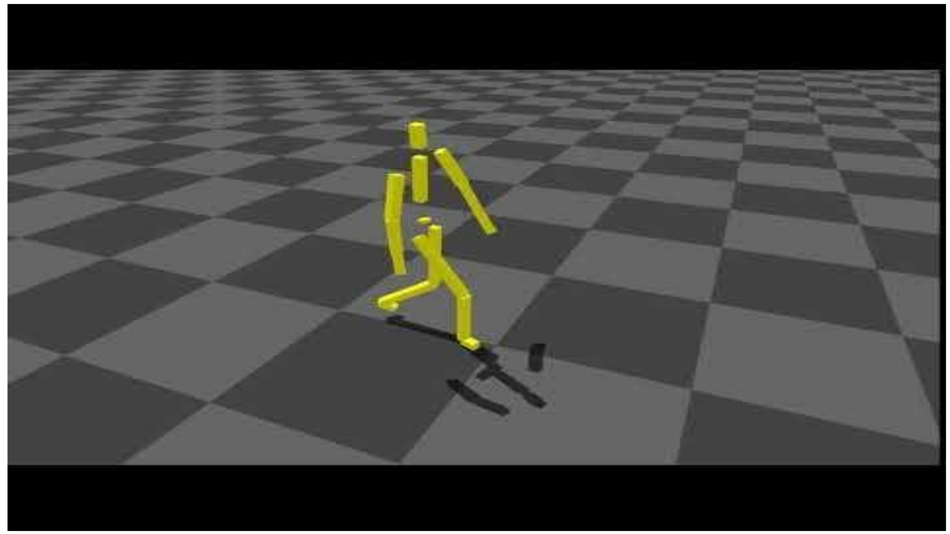


- Simple solution: linear interpolation
- Linear interpolation of two poses by:
  - Linear interpolation of root positions
  - Slerp of root & joint orientations

Time Warping

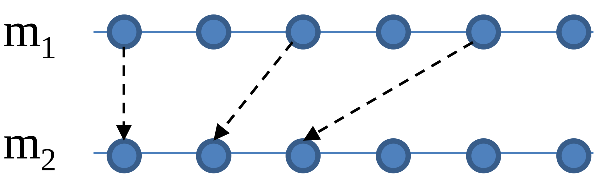
$$\boxed{\mathbf{m}_2(t) = \mathbf{m}_1(s(t))}$$

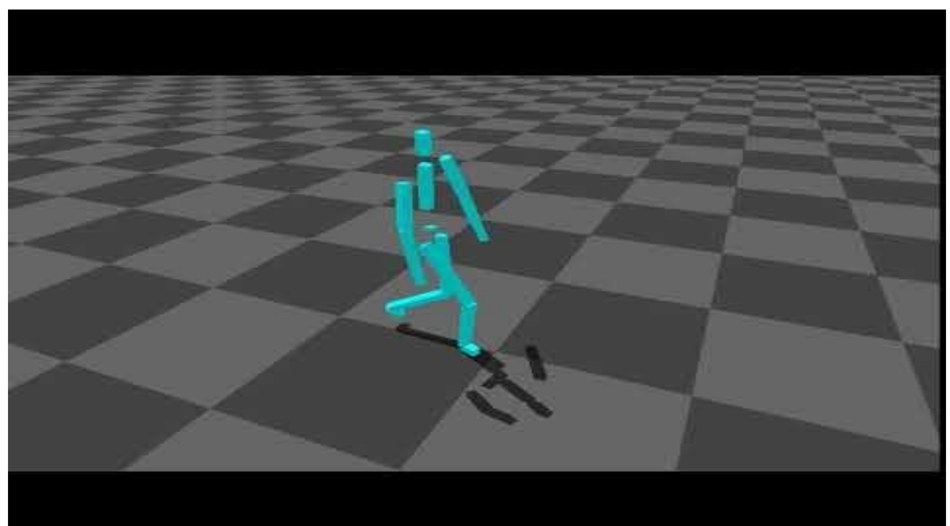
$\mathbf{m}_1$ (original motion)



https://youtu.be/ViYTecPLrho

$\mathbf{m}_2(t) = \mathbf{m}_1(2^*t)$

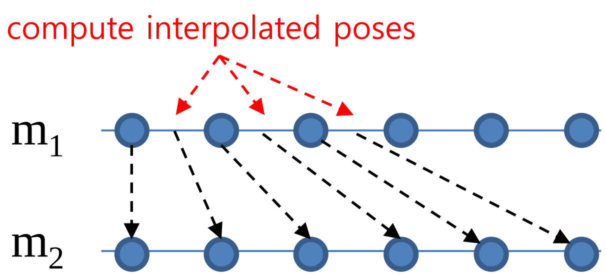




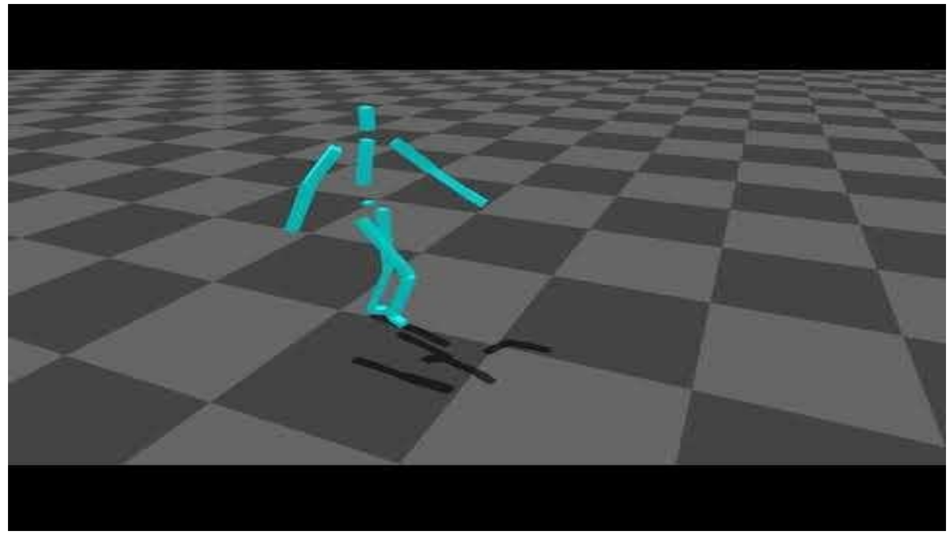
https://youtu.be/dStMgTRR5iw

Time Warping

$\mathbf{m}_2(t) = \mathbf{m}_1(0.5{*}t)$

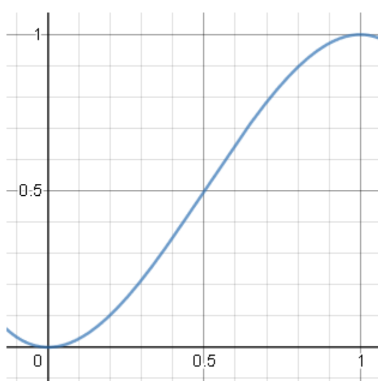


https://youtu.be/c3ZI7vMqQMM

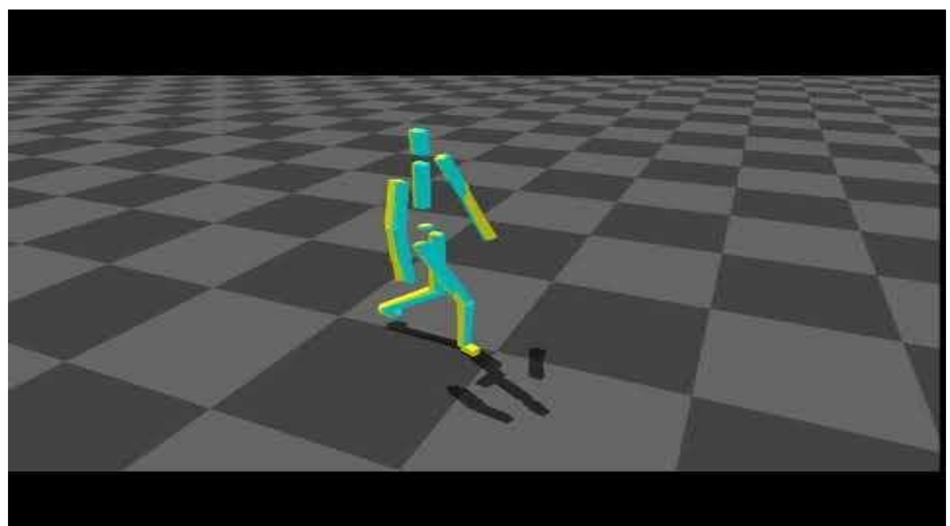


$\mathbf{m}_2(t) = \mathbf{m}_1(H_1(t))$

$H_1(t)$:

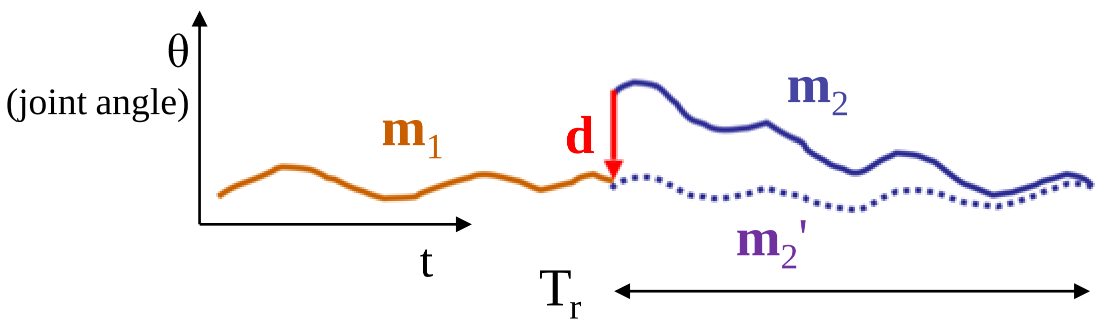


https://youtu.be/nAJ-rGNBXSc



Motion Stitching

- Editing the second motion ($\mathbf{m}_2$) so that it connects seamlessly after the first motion ($\mathbf{m}_1$).

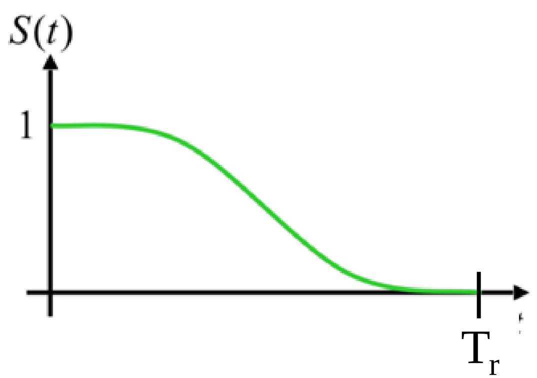


$$\mathbf{d} = \mathbf{m}_1(T_1) - \mathbf{m}_2(0)$$

(T_1: last frame time of $\mathbf{m}_1$, 0: first frame time)

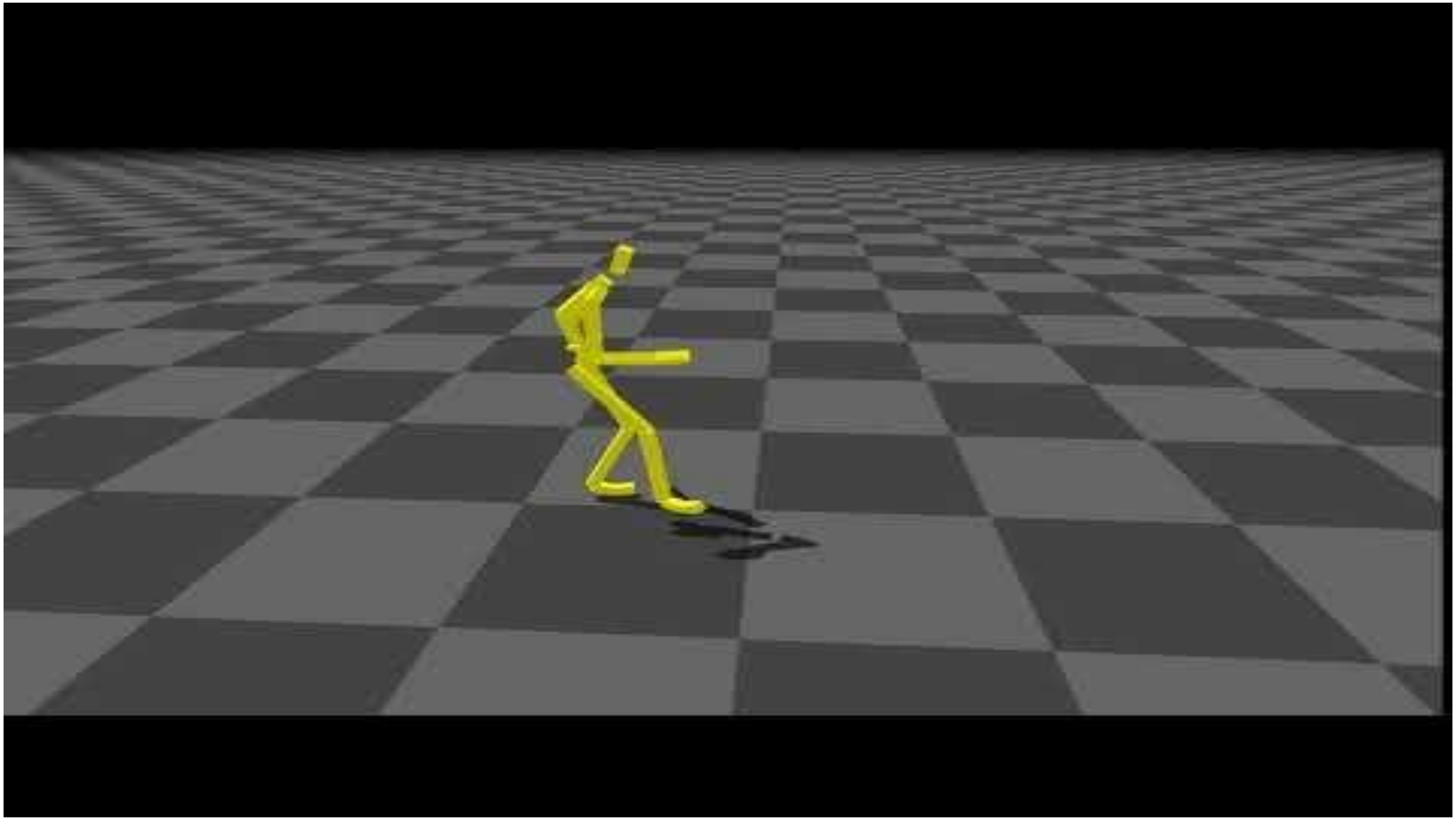
$$\mathbf{m}_2'(t) = \mathbf{m}_2(t) + s(t) \cdot \mathbf{d}$$

s(t): transition function



T_r : transition duration
(transition length)

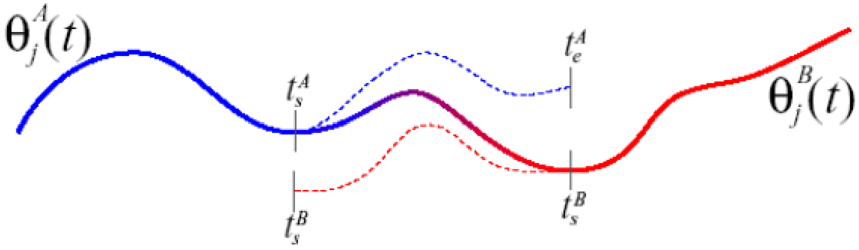
Motion Stitching



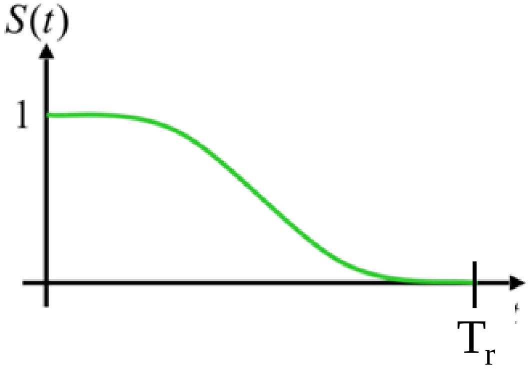
https://youtu.be/3UBO_CDFjOI

Motion Blending

- Interpolation between motions

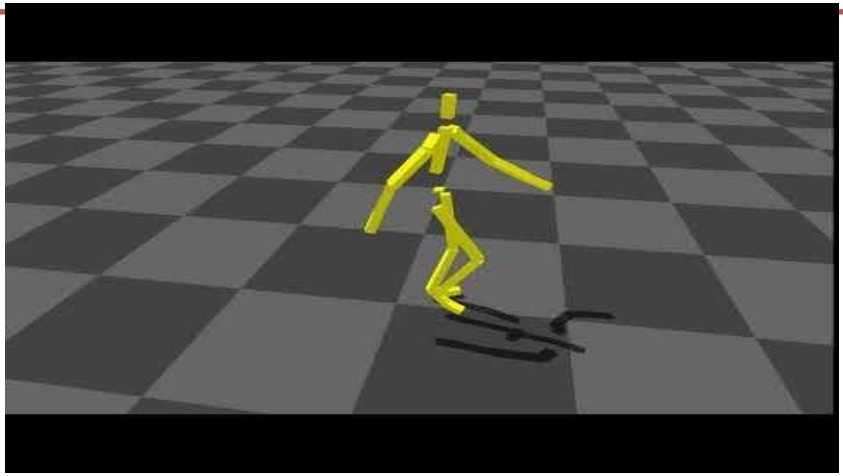


s(t): transition function

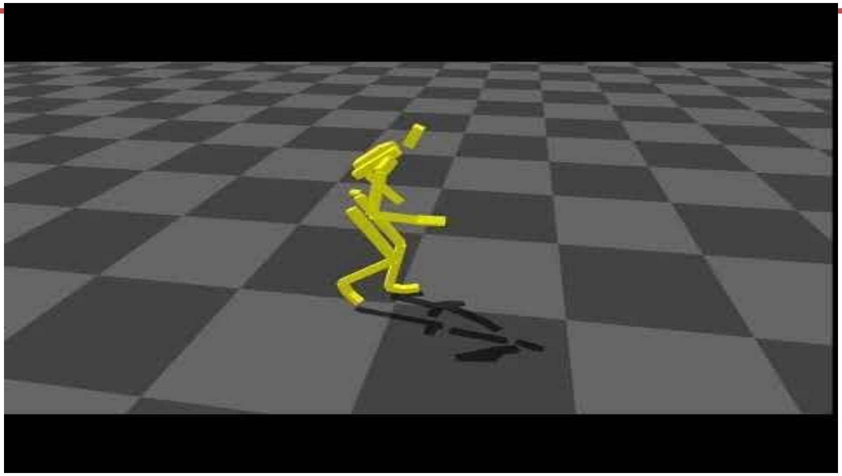


$$\mathbf{m}(t) = (1-s(t)) \cdot \mathbf{m}_1(t) + s(t) \cdot \mathbf{m}_2(t)$$

Motion Blending

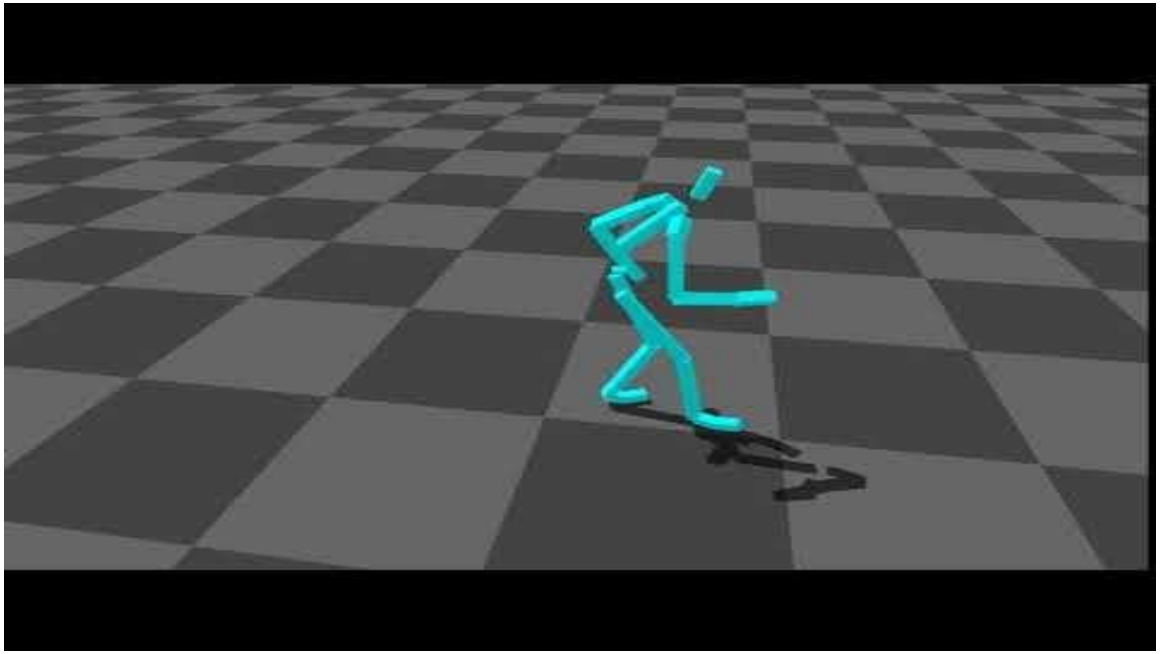


cycle0 https://youtu.be/T0xRiNzQ2To



cycle1 https://youtu.be/fKCgIgOKl-E

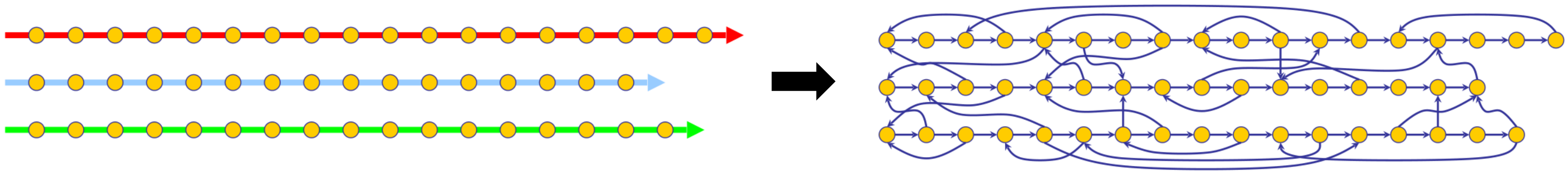
blended motion
(motion0 + blended cycle + motion1)



https://youtu.be/_y7BtHh6Yvw

Intro to Data-Driven Motion Synthesis

Motion Graph [Lee et al. 2002] [Kovar et al. 2002] [Arikan&Forsyth 2002]



- Consideration for creating transitions:
  - Contact states, pose similarity, avoiding dead-ends

- Once a motion graph is constructed, you can find a series of transitions passing through...
  - Specified poses
  - Specified locations / continuous path
  - Specified poses and times
  - ...
- by using graph search algorithms (such as Dijkstra, A*, ...) or dynamic programming.

- Motion editing techniques facilitates smooth transitions.

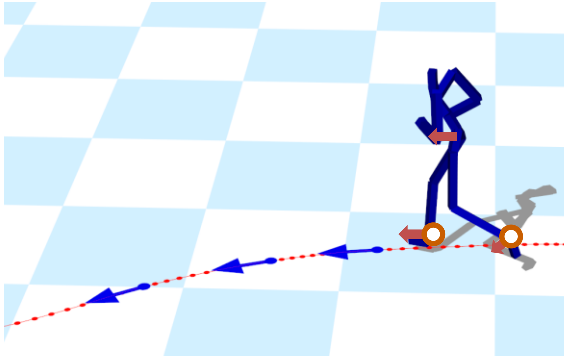
Motion Graph

[Lee et al. 2002]

http://graphics.cs.cmu.edu/projects/Avatar/

Motion Matching [Büttner and Clavet 2015]

- *Motion DB* stores the pose for each frame of motion data.
- *Feature DB* stores extracted "features" for each frame of motion data.
  - Feature: (current state, future information)



- Matching (performed periodically):
  - Query q: (current character state, future information created by user input)
  - Search for the frame j^* that corresponds to the feature closest to the query q.
- , then motions are played sequentially from the j^* frame in the *motion DB*.

- Motion editing techniques facilitates smooth transitions.

Motion Matching

https://youtu.be/qBbCjuJpE9o

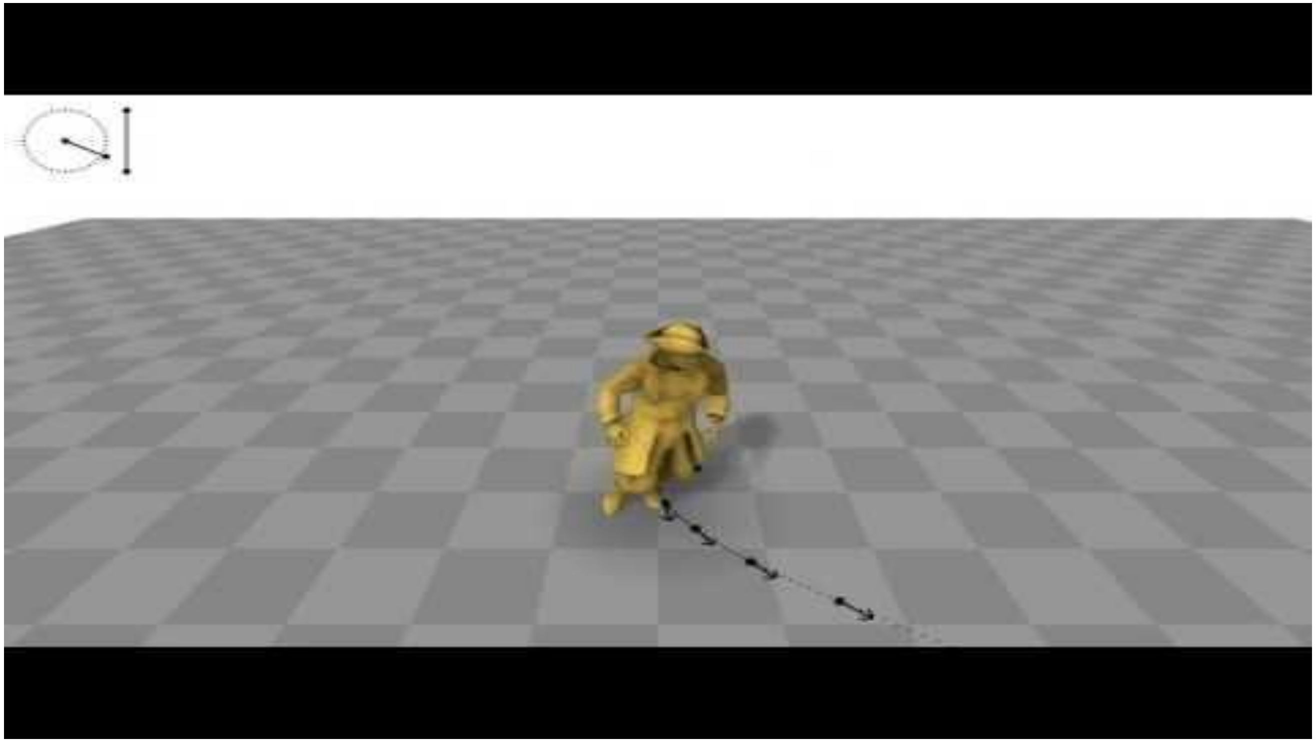
(Submitted Paper) Interactive Character Path-Following using Motion Matching

Intro to Data-Driven Motion Synthesis

Deep Motion Synthesis - Problems

- Character control
  - [Holden 2017], [Lee 2018], [Zhang 2018], [Starke 2019], [Henter 2020], [Ling 2020], [Starke 2020], [Starke 2021], [Lee 2021], [Cho 2021]

https://youtu.be/Ul0Gilv5wvY



[Holden 2017]

Problems

- Motion prediction
  - [Holden 2015], [Holden 2016], [Harvey 2020]

https://youtu.be/fTV7sXqO6ig

[Harvey 2020]

Problems

- Style transfer
  - [Holden 2016], [Aberman 2020b]

https://youtu.be/m04zuBSdGrc

[Aberman 2020b]

Problems

- Sound to motion
  - [Yoon 2020], [Valle-Pérez 2021]

https://youtu.be/uBnCePehA-Y

[Valle-Pérez 2021]

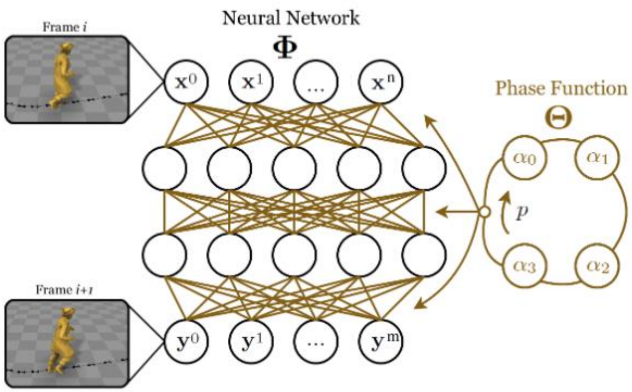
Learning Approaches

- Learns a model generating new motions
  - Discriminative models with FFNN, CNN, or RNN structures
  - Generative models with GAN, VAE, or Flow
  - Manifold learning with Autoencoders
  - Reinforcement learning to learn a policy
  - ...

Supervised Learning in Deep Motion Synthesis

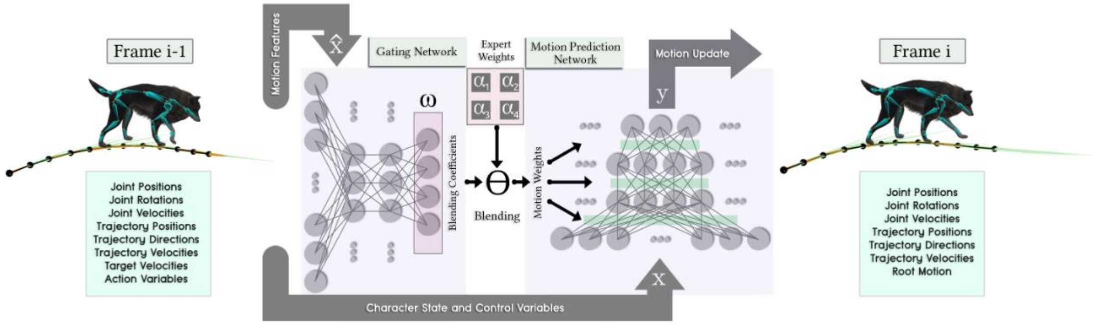
- *Feed-forward neural networks* are widely used to output the next pose.
  - Training data: motion capture data set
  - Input: pose at frame i, control input, ...
  - Output: pose at frame i+1, ...

https://youtu.be/Ul0Gilv5wvY



[Holden 2017]

https://youtu.be/uFJvRYtjQ4c

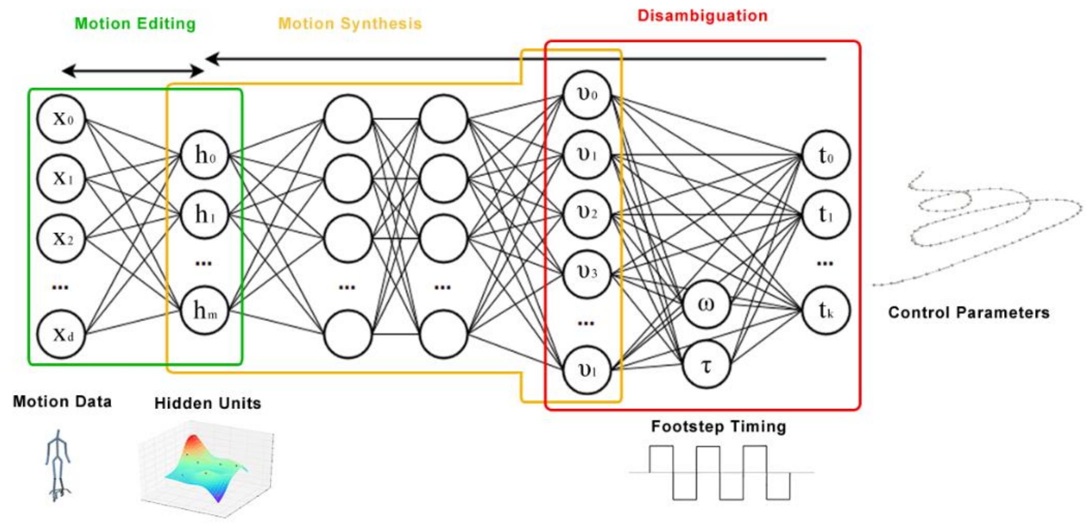


[Zhang 2018]

Supervised Learning in Deep Motion Synthesis

- *Convolutional neural networks* are used to output the new motion.
  - Input: some input sequence
  - Output: output motion (pose sequence)

https://youtu.be/urf-AAIwNYk

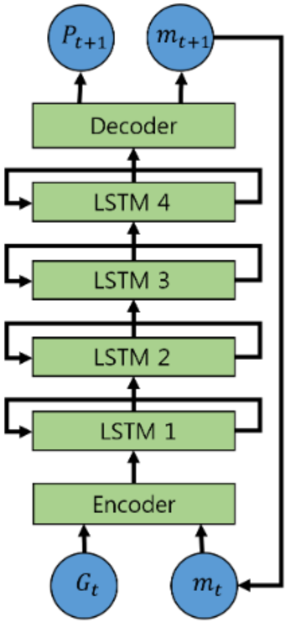


[Holden 2016]

Supervised Learning in Deep Motion Synthesis

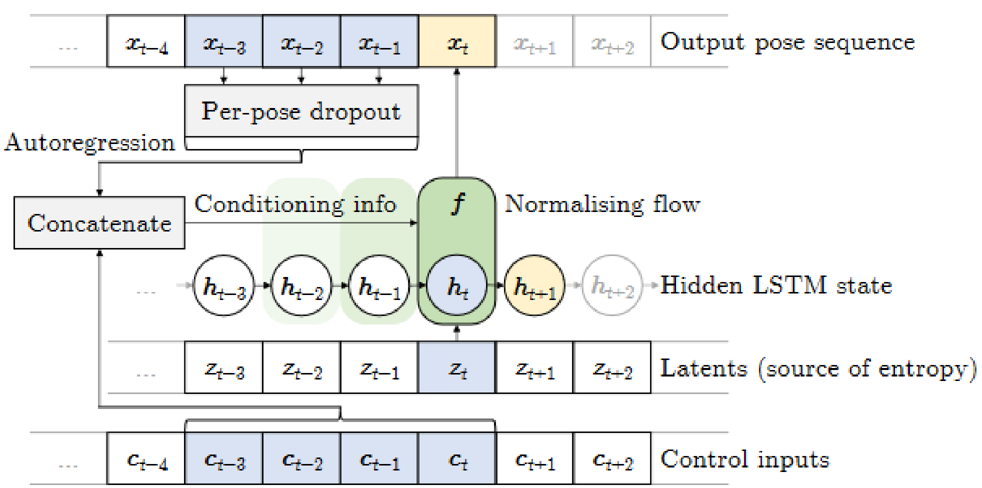
- *Recurrent neural networks* are used to output the next pose.
  - Input: pose at frame i, *hidden state* (representing past information), ...
  - Output: pose at frame i+1, ...

https://youtu.be/pe-YTvavbtA



[Lee 2018]

https://youtu.be/pe-YTvavbtA



[Henter 2020]

Unsupervised Learning in Deep Motion Synthesis

- AutoEncoder

https://youtu.be/dLopOB6D9co

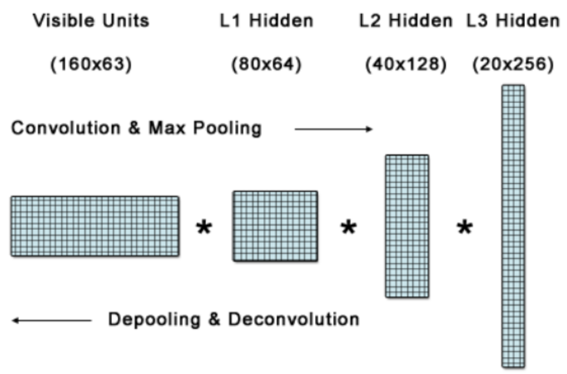
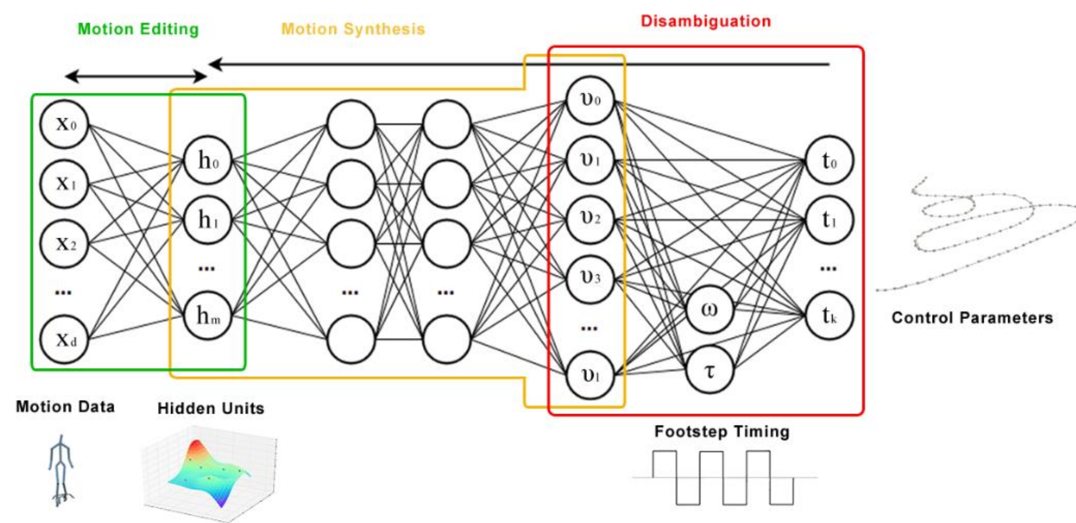


Figure 3: *Units of the Convolutional Autoencoder. The input to layer 1 is a window of 160 frames of 63 degrees of freedom. After the first convolution and max pooling this becomes a window of 80 with 64 degrees of freedom. After layer 2 it becomes 40 by 128, and after layer 3 it becomes 20 by 256.*

[Holden 2015]

https://youtu.be/urf-AAIwNYk

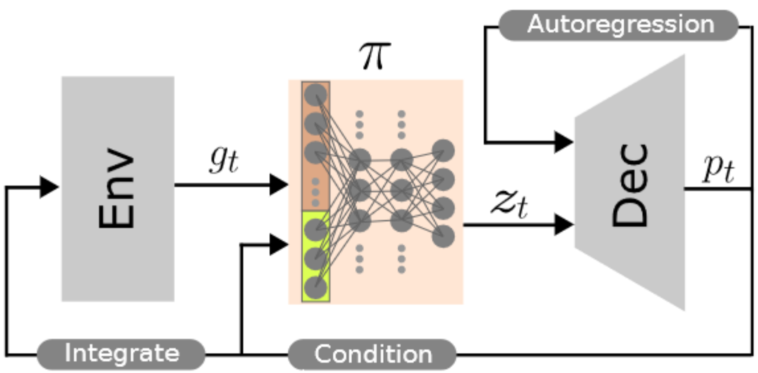


[Holden 2016]

Reinforcement Learning in Deep Motion Synthesis

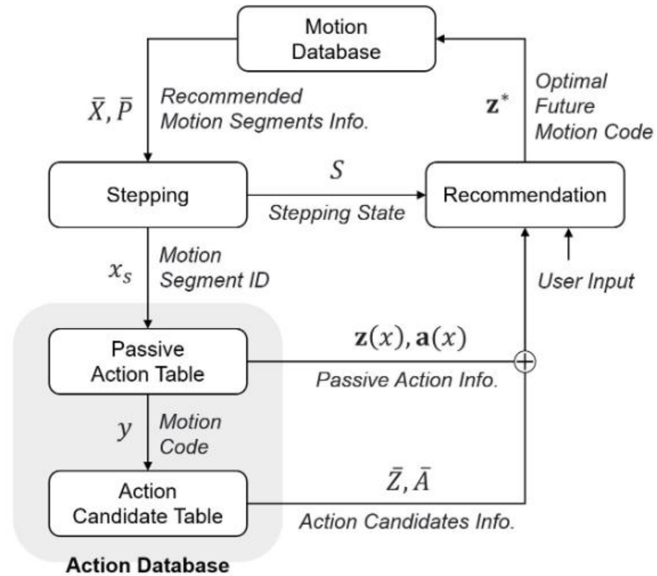
- Deep reinforcement learning (DRL) has been mainly used for physically-simulated character control.
- But recently it has been gradually applied to motion synthesis tasks.

https://youtu.be/Zm3G9oqmQ4Y



[Ling 2020]

https://youtu.be/ZVVI1VOMdoQ

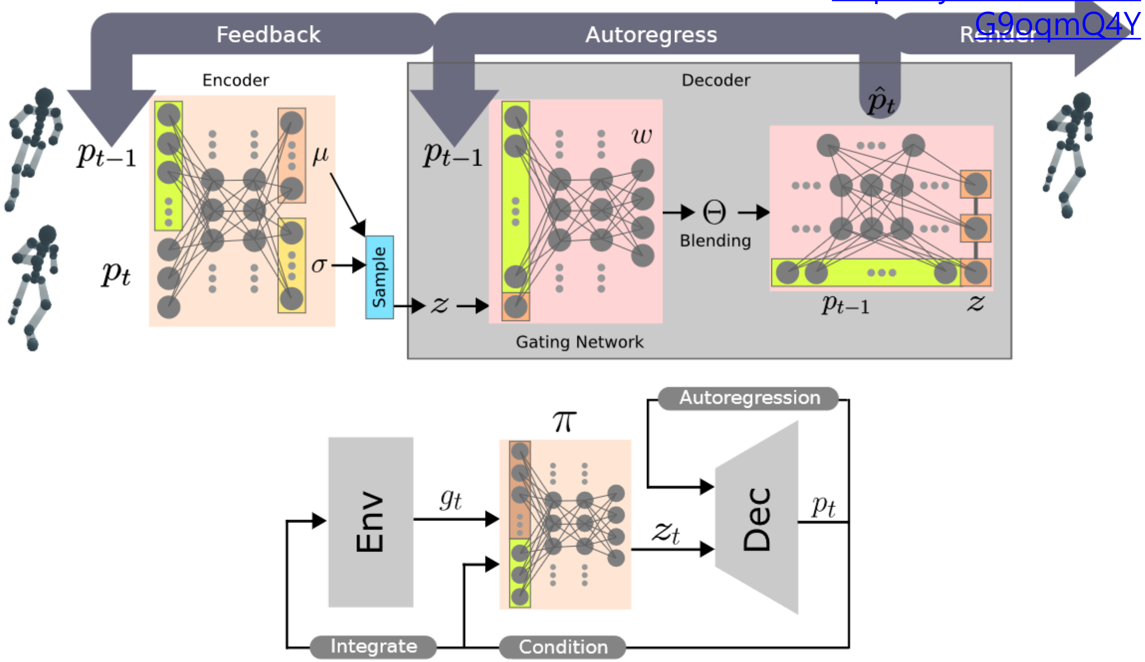


[Cho 2021]

Generative Model in Deep Motion Synthesis

- Generative models are used to generate different motions for the same control signal
  - because human motion is not completely deterministic

https://youtu.be/Zm3G9oqmQ4Y



[Ling 2020]

Next Time

- Next week:
  - Preparing Project 3 (No class)

- The week after next:
  - Presentation: Project 3