

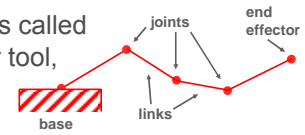
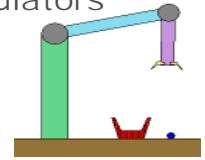
Kinematics

- Kinematics = study of motion, without considering mass or forces
- Uses position, velocity, acceleration
- [Dynamics studies motivating forces]
- Useful for keyframing of articulated structures

35

Kinematics of manipulators

- A *manipulator* is an articulated robotic arm
- Joints may have 1 or more degrees of freedom (DOF, cf. human joints)
- Links may be extensible
- The tip of the arm is called the *end effector* (or tool, hand, gripper)



Joint DOF



Forward kinematics

- *Forward* (or *direct*) kinematics problem: given the joint parameters (angles), compute the end effector position and orientation relative to the base

Forward kinematics

- Example: 2-link manipulator
 - Given unit direction vectors \mathbf{u}_i and link lengths l_i the end effector (\mathbf{P}_2) position is $\mathbf{P}_2 = \mathbf{P}_0 + l_0 \mathbf{u}_0 + l_1 \mathbf{u}_1$
 - The last link angle (relative to the base) is $\theta = \theta_0 + \theta_1 \pmod{2\pi}$
 - The last link direction vector is $\mathbf{u}_1 = (\cos \theta, \sin \theta)$
-

Forward kinematics

- The 2-link results generalise to:
- $$\mathbf{P}_n = \mathbf{P}_0 + \sum_{i=0}^{n-1} l_i \mathbf{u}_i \quad \left| \quad \mathbf{u}_i = \left(\cos \left(\sum_{j=0}^i \theta_j \right), \sin \left(\sum_{j=0}^i \theta_j \right) \right)$$
- $$\mathbf{P}_n = \mathbf{P}_0 + \sum_{i=0}^{n-1} l_i \left(\cos \left(\sum_{j=0}^i \theta_j \right), \sin \left(\sum_{j=0}^i \theta_j \right) \right)$$

Forward kinematics

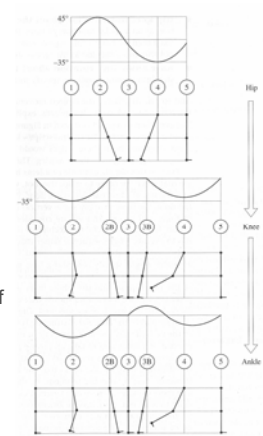
$$\mathbf{P}_n = \mathbf{P}_0 + \sum_{i=0}^{n-1} l_i \left(\cos \left(\sum_{j=0}^i \theta_j \right), \sin \left(\sum_{j=0}^i \theta_j \right) \right)$$

- The forwards kinematics problem is easily solved. \mathbf{P}_n depends explicitly on the given angles θ_i
- Same formula applies if l_i are variable
- Similar formula for joints with more DOFs

41

Forward kinematics animation

- Specify all the joint angles in time, down the articulated model hierarchy
- [Making a mistake and having to go back up the hierarchy almost certainly means having to re-do work...]
- Hard to achieve a natural animation – mocap may help
- Tedious, lots of work. Libraries of joint angles for standard motion (e.g. walking) of standard articulated models (e.g. a human) help.



Inverse kinematics

- Inverse kinematics: given an end effector position, what are the possible joint angles?
- More realistic problem – this is the question that you usually encounter in practice (robotics, games, etc.)
- Hard problem:
 - In general no closed-form solution
 - In general no unique solution

43

Inverse kinematics

- Not unique:

44

Inverse kinematics

- 2-link manipulator

$$\mathbf{P}_n = \mathbf{P}_0 + \sum_{i=0}^{n-1} l_i \left(\cos \left(\sum_{j=0}^i \theta_j \right), \sin \left(\sum_{j=0}^i \theta_j \right) \right)$$

$$\mathbf{P}_2 = \mathbf{P}_0 + l_0 (\cos \theta_0, \sin \theta_0) + l_1 (\cos(\theta_0 + \theta_1), \sin(\theta_0 + \theta_1))$$

- Example

Given

$$\left. \begin{array}{l} \mathbf{P}_0 = (0, 0) \\ \mathbf{P}_2 = (1, 1) \\ l_0 = l_1 = \frac{4}{5} \end{array} \right\} \begin{array}{l} \theta_0 = \cos^{-1} \left(\frac{1}{8} (5 - \sqrt{7}) \right) \\ \theta_1 = -\cos^{-1} \left(\frac{9}{16} \right) \\ \theta_0 = \cos^{-1} \left(\frac{1}{8} (5 + \sqrt{7}) \right) \\ \theta_1 = \cos^{-1} \left(\frac{9}{16} \right) \end{array}$$

45

Inverse kinematics

- Finding an explicit solution is already difficult for 2 links
- The problem becomes increasingly difficult with more links, DOFs, and motion constraints
- For larger models numerical approaches (e.g. minimizing motion energy) are necessary
- Animation (or robotic trajectory planning) also requires finding a feasible path through the 'reachable space' of the end effector, not just a single end configuration
- ...and must ensure this path can be traveled by smooth variation of the joint angles
- ...and must ensure this path looks natural (e.g. walking) → very hard to model as a mathematical constraint
- Becomes infeasible for large animated models

46

Kinematics for animation: the lowdown

- FK is a lot of work. Realistic look depends mostly on the skill of the animator. Mocap helps, but restricts motion to the captured scripts.
- IK is a more realistic problem statement requiring little input, but
 - hard to solve for large models
 - hard to achieve a natural look
- Mocap + FK seems to be the current preferred system for large-scale animations in the industry, possibly because much of the technical detail can be hidden, and the artist's knowledge included: a natural look can be achieved by capturing the 'hands-on' animations of animation artists (see also *Jurassic Park* example in Watt, p.399)


Shrek

- Kinematics of articulated structures, done right and wrong

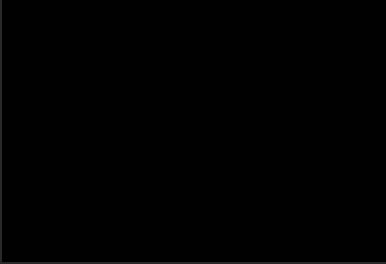


Motion capture

- Records real motion using body markers and trackers
- Marked points are mapped to an animation model, usually an articulated model



Motion capture




Motion capture

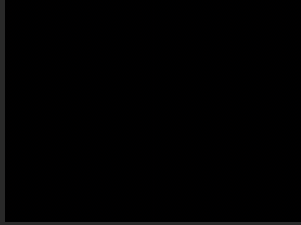
- System
 - Current systems use magnetic or optical tracking of markers
 - Markerless system is sometimes possible: direct matching of model to video footage. Slower and less precise
- Strengths
 - Natural look of captured data
 - Allows free motion expression, technical stuff of animation can mostly be hidden
- Weaknesses
 - System: noise, occlusion of markers, skin suits, marker shift
 - Mapping of markers to model often not perfect
 - Not retrospective, (mostly) limited to recorded scripts
 - Poor scalability / portability of recorded data to different objects

Example: *I, robot*

- Motion captured data to articulated model



Horse doubles



53

Horse doubles - movie footage



54

Procedural animation

- Animation is produced by a program computing position (or other object attributes) over time, subject to a set of rules
- The rule set is often physical; the animation is a *physically based simulation*
- Little input is needed to enable a large physically plausible animation
- For complex systems (e.g. a human walking), hard to make look natural
- Special case: *dynamics*: consider *forces* that generate motion.

55

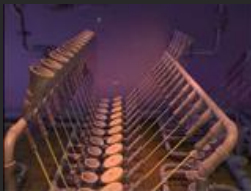
Procedural animation

- Simulations can be *active* or *passive*
- Passive systems
 - have no internal energy source; move only because of external forces
 - An animation requires only a set of rules and initial conditions to run
 - Examples include most particle systems we've seen, dropping a ball, dropping a leaf, etc.
 - Disadvantage: little control once the animation has started
- Active systems
 - Have an internal energy source and can move of their own volition
 - E.g. people, animals, robots, adaptive systems
 - Harder to animate: needs physical laws *and* a control system

56

Passive system example

- Completely driven by external conditions (music driven)



57

Procedural vs. keyframe

- Easy to generate lots of motion
 - Easy to generate interactive behavior
 - Poor low-level control
 - Natural movement and expression of characters poor
 - CPU-intensive
-
- Lots of motion needs lots of work
 - Interactivity limited to scripted actions
 - Full control

58

Dynamics

- Most common systems studied (in the context of animation):
 - Rigid bodies
 - Unconstrained (e.g. a rocket)
 - Constrained (e.g. a skeleton)
 - Deformable bodies
 - Mass-spring models
 - Particle systems
- Solving / approximating a solution requires knowledge of
 - physical systems (e.g. Newtonian mechanics for rigid bodies)
 - Differential equations
 - Numerical methods

59

Rigid body dynamics

- Outline:
 - Particles – Newton's 2nd law
 - Types of forces
 - Rigid bodies:
 - Moments
 - Velocities
 - Forces and torques

60

Particles

- Movement of particles is controlled by Newton's 2nd law:

$$\mathbf{F} = m\mathbf{a}$$

\mathbf{F} = force vector
 \mathbf{a} = acceleration vector
 m = mass

- Relation to velocity \mathbf{v} and position \mathbf{x} is given by the time derivatives:

$$\mathbf{F} = m\dot{\mathbf{v}} = m\ddot{\mathbf{x}}$$

- If mass is not constant (e.g. a rocket burning fuel):

$$\mathbf{F} = \frac{d(m\mathbf{v})}{dt}$$

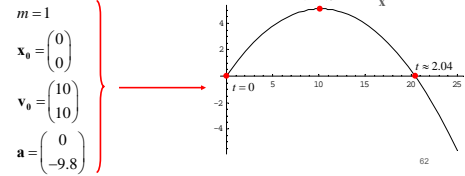
61

Particles $\mathbf{F} = m\mathbf{a} = m\dot{\mathbf{v}} = m\ddot{\mathbf{x}}$

- Simple example: throwing a ball (2D), gravity only.
- \mathbf{a} is constant, so \mathbf{x} and \mathbf{v} can be solved by direct integration:

$$\mathbf{v} = \mathbf{a}t + \mathbf{v}_0$$

$$\mathbf{x} = \frac{1}{2}\mathbf{a}t^2 + \mathbf{v}_0t + \mathbf{x}_0$$



62

Types of forces

- Common forces used in animation:
 - Gravity $\mathbf{F} = m\mathbf{g} = m\begin{pmatrix} 0 \\ -9.8 \end{pmatrix}$
 - Spring forces $\mathbf{F} = -c\Delta\mathbf{U}$
 - Friction $\mathbf{F} = -c\frac{\mathbf{v}}{|\mathbf{v}|}$
 - Viscosity $\mathbf{F} = -c\mathbf{v}$
 - Applied forces
 - Force fields

63

Rigid body moments

- Moments are important concepts because without applying forces *moments are preserved*

Linear moment $\rightarrow \mathbf{M} = m\mathbf{v}$

$$\dot{\mathbf{M}} = m\dot{\mathbf{v}} = \mathbf{F}$$

Angular moment $\rightarrow \mathbf{L} = \mathbf{I}\boldsymbol{\omega}$

Radial velocity

$$\dot{\mathbf{L}} = \mathbf{I}\dot{\boldsymbol{\omega}} = \boldsymbol{\tau} \rightarrow \text{Torque}$$

64

Rigid bodies

- Useful to maintain two coordinate frames:
 - Fixed world coordinate frame
 - Body frame centered at centre of mass, rotates with the body



65

Rigid bodies

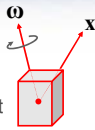
- Orientation is specified by a rotation matrix \mathbf{R} relating the body axes to world axes
- Position is specified by a translation \mathbf{x} of the body centre of mass
- World coordinates of a body frame point \mathbf{r}_{body} are

$$\mathbf{r} = \mathbf{R}\mathbf{r}_{\text{body}} + \mathbf{x}$$

66

Angular velocity

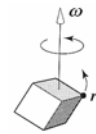
- Movement of a rigid body can be decomposed into a vector movement (translation) and a rotational part
- The rotation is specified by the velocity vector $\boldsymbol{\omega}$
- $\boldsymbol{\omega}$ passes through the centre of mass, is oriented along the body rotation axis and its magnitude determines the rotational speed



67

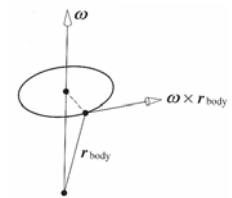
Angular velocity

- The velocity due to rotation of a point \mathbf{r} is determined by the angular velocity $\boldsymbol{\omega}$ and its distance to the rotation axis (circle radius)



- The speed vector equals

$$\dot{\mathbf{r}}_{\text{body}}(t) = \boldsymbol{\omega}(t) \times \mathbf{r}_{\text{body}}(t)$$



68

Angular velocity

$$\dot{\mathbf{r}}_{\text{body}}(t) = \boldsymbol{\omega}(t) \times \mathbf{r}_{\text{body}}(t)$$

- The angular speed vector formula holds for any point \mathbf{r} relative to the body.
- The orientation of the body is specified by the rotation matrix \mathbf{R}
- The columns of \mathbf{R} represent the three axes of the body coordinate system, so can be regarded as points $\mathbf{r} \rightarrow$ we can apply the speed vector formula to the columns of \mathbf{R} :

$$\dot{\mathbf{R}}(t) = (\boldsymbol{\omega}(t) \times \mathbf{R}[1], \boldsymbol{\omega}(t) \times \mathbf{R}[2], \boldsymbol{\omega}(t) \times \mathbf{R}[3]) = \boldsymbol{\omega}(t) * \mathbf{R}(t)$$

where $\mathbf{R}[i]$ represents column i of \mathbf{R}

69

Total velocity

$$\mathbf{r}(t) = \mathbf{x}(t) + \mathbf{R}(t)\mathbf{r}_{\text{body}} \quad \text{position}$$

$$\dot{\mathbf{r}}(t) = \dot{\mathbf{x}}(t) + \dot{\mathbf{R}}(t)\mathbf{r}_{\text{body}} \quad \text{speed by definition}$$

$$= \mathbf{v}(t) + \boldsymbol{\omega}(t) * \mathbf{R}(t)\mathbf{r}_{\text{body}}$$

$$= \mathbf{v}(t) + \boldsymbol{\omega}(t) * (\mathbf{R}(t)\mathbf{r}_{\text{body}} + \mathbf{x}(t) - \mathbf{x}(t))$$

$$= \mathbf{v}(t) + \boldsymbol{\omega}(t) * (\mathbf{r}(t) - \mathbf{x}(t))$$

$$= \mathbf{v}(t) + \boldsymbol{\omega}(t) \times (\mathbf{r}(t) - \mathbf{x}(t))$$

- The total velocity of a body point by linear and angular movement

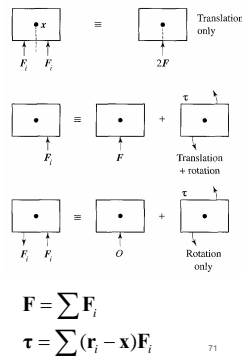
70

Forces and torques

- A force applied to a rigid body can induce both a translation and a rotation
- The 'force' part that causes rotation is known as **torque τ**

$$\boldsymbol{\tau} = (\mathbf{r} - \mathbf{x}) \times \mathbf{F}$$

$\mathbf{r} - \mathbf{x}$: point of force application
 \mathbf{F} : applied force
 \mathbf{x} : body centre of mass



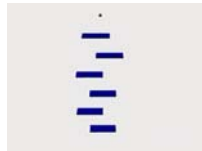
Torque

$$\begin{cases} \mathbf{M} = m\mathbf{v} \\ \dot{\mathbf{M}} = m\dot{\mathbf{v}} = \mathbf{F} \\ \mathbf{L} = \mathbf{I}\boldsymbol{\omega} \\ \dot{\mathbf{L}} = \mathbf{I}\dot{\boldsymbol{\omega}} = \boldsymbol{\tau} \end{cases}$$

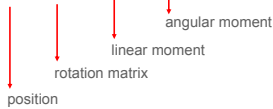
- Similar as with 'linear' velocity (and position) which can be determined from force, angular velocity (and position) can be determined from torque
- We need only the *inertial tensor* \mathbf{I} , a 3x3 matrix which is a measure for mass distribution (see Watt 14.6.6. for more details).

Practical dynamic simulation

- We need modules for:
 - Input: forces and torques
 - Dynamics handling
 - Numerical integration
- Dynamics module:
 - For each object, track a *state vector*



$$\mathbf{S}(t) = (\mathbf{x}(t), \mathbf{R}(t), \mathbf{M}(t), \mathbf{L}(t))$$



Practical dynamic simulation

$$\mathbf{S}(t) = (\mathbf{x}(t), \mathbf{R}(t), \mathbf{M}(t), \mathbf{L}(t))$$

- Derivative ('update' vector):

$$\dot{\mathbf{S}}(t) = (\mathbf{v}(t), \boldsymbol{\omega} * \mathbf{R}(t), \mathbf{F}(t), \boldsymbol{\tau}(t))$$
- Integrate the update vector (over a certain time step), and add to the state vector

Space-time constraints

- Dynamic simulation gives the animator little control: he can only set initial conditions and apply forces, the object movements are then bound by physical laws
- Setting 'space-time constraints' gives more control, because the motion is specified as a *boundary value* problem. E.g.: begin and end position are specified (cf. keyframes!) The simulation is still carried out bound by physical laws.

Space-time constraints

- Animator specifies:
 - *What* the character has to do ('jump the hurdle')
 - *How* he should do it ('spend minimal energy')
 - Object *structure* (geometry, mass, connectivity, etc.)
 - Object *resources* (muscles, a handy floor to push off from...)
- Gives animator more control and is more high-level

Space-time constraints: Example

- 'Particle' rocket ship, with jet force $\mathbf{F}(t)$
- Equation of motion: $m\ddot{\mathbf{x}} = \mathbf{F} + m\mathbf{g}$

'Classical' approach

- Specify initial conditions (start position and start velocity) and \mathbf{F}
- Run dynamic simulation

Space-time constraints

- Specify constraints (e.g.) start and end position
- Specify to travel expending minimal fuel, e.g. $\min \int_{t_0}^{t_1} |\mathbf{F}|^2 dt$
- Find minimal \mathbf{F} subject to constraints

77

The ultimate example: *Final Fantasy* (2001)

Particle systems, flocking, dynamic simulations, advanced facial animation, advanced hair animation, ...



78

Bonus: Shrek – 3D



5 min
12 min

79