Lecture VI: Constraints and Controllers

Parts Based on Erin Catto's Box2D Tutorial

Motion Constraints

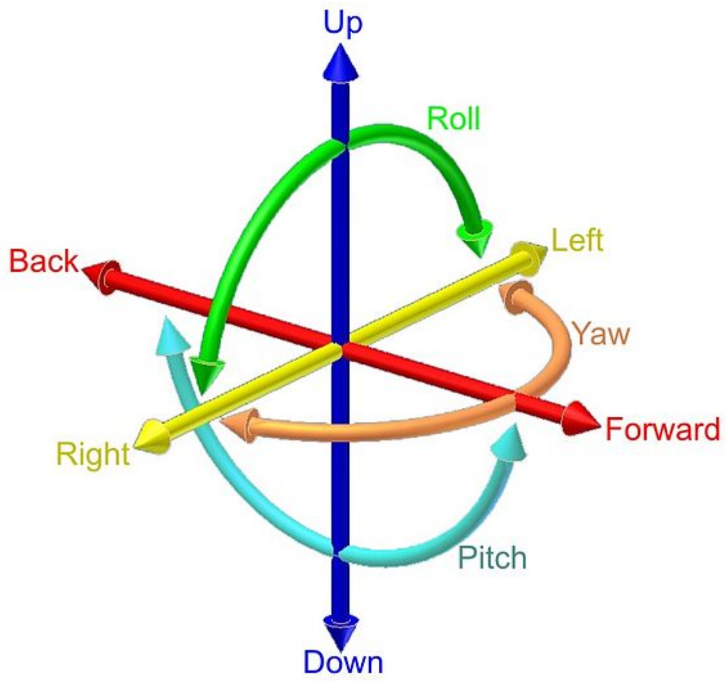
- In practice, no rigid body is free to move around ‘on its own’.
- Movement is constrained:
  - wheels on a chair
  - human body parts
  - trigger of a gun
  - opening door
  - actually almost anything you can think of in a game...

http://i2.cdn.turner.com/cnnnext/dam/assets/140813101135-best-biking-cities-utrecht-horizontal-large-gallery.jpg

http://static4.gamespot.com/uploads/scale_medium/mig/2/4/7/7/2282477-Garrys_13231_screen.jpg

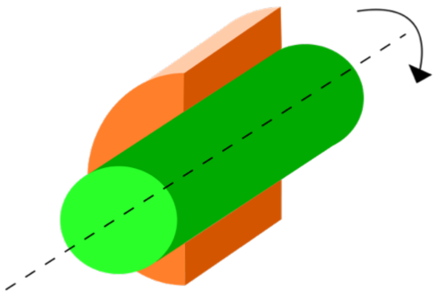
Degrees of Freedom

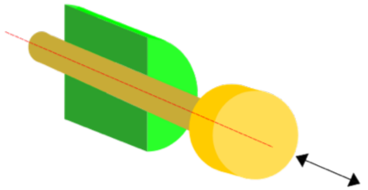
- Description of allowed movement by specifying degrees of freedom (DOF)
  - Translational (3 DOF max)
  - Rotational (3 DOF max)

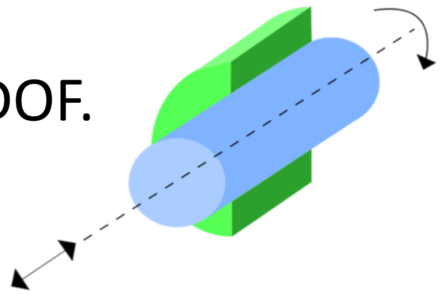


Kinematic Pair

- Kinematic pair: connection between two bodies that imposes constraints on their relative movement.
  - Lower pair: constraint on a point, line or plane
    - Revolute pair, or hinged joint: 1 rotational DOF.
    - Prismatic joint, or slider: 1 translational DOF.
    - Screw pair: 1 coordinated rotation/translation DOF.
    - Cylindrical pair: 1 translational + 1 rotational DOF.
    - Spherical pair, or ball-and-socket joint: 3 rotational DOF.
    - Planar pair: 3 translational DOF.
  - Higher pair: constraint on a curve or surface.





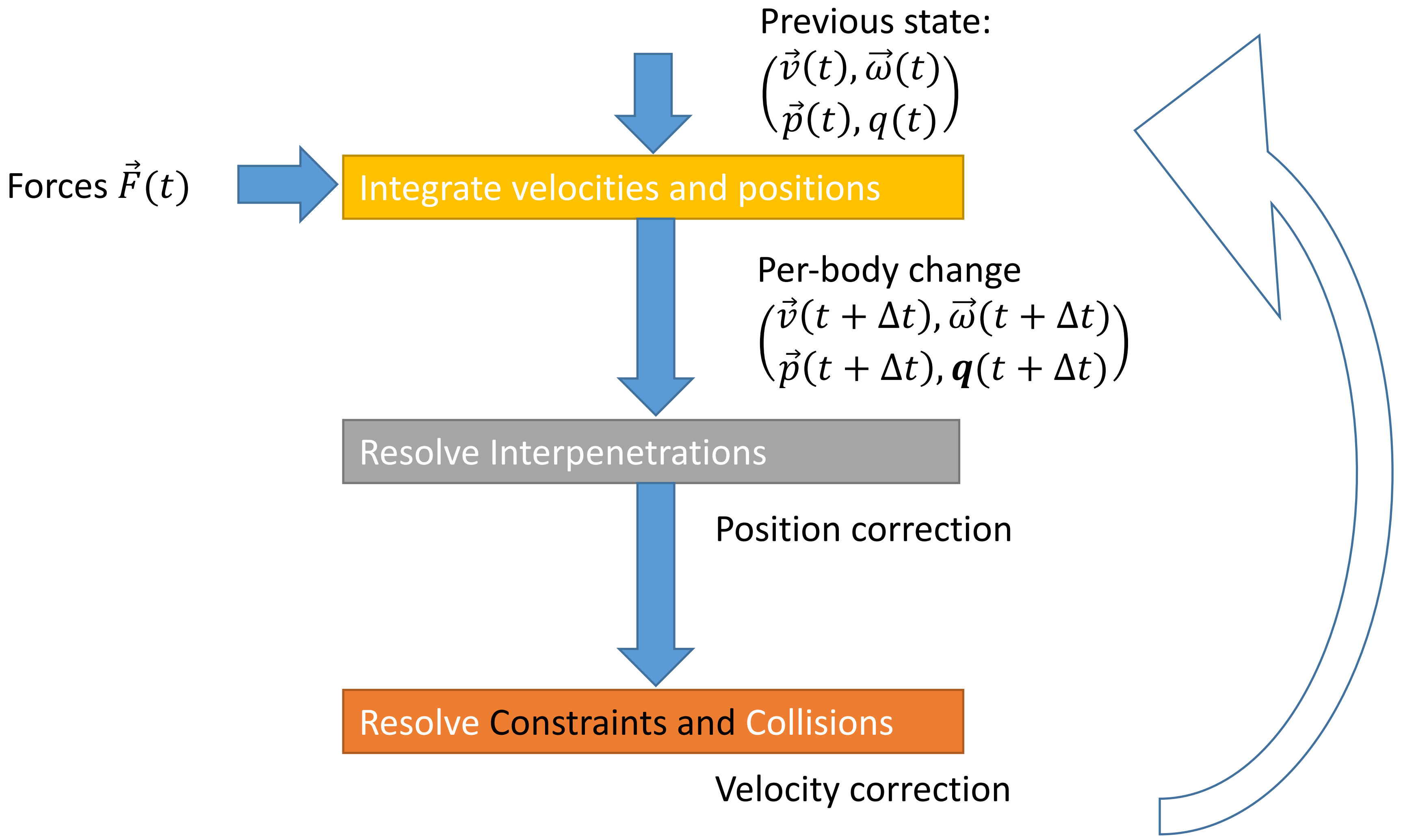


Constraints Resolution

- Energy of components on constrained d.o.f. can be lost.
  - Converted into heat and sound (E_o)
  - As opposed to putting pressure on the joint, causing degradation.
- Project net force on the unconstrained degrees of freedom.
- Not true for soft bodies!
  - Deforming accordingly…
- (Lagrangian) Approximation: no loss of energy.

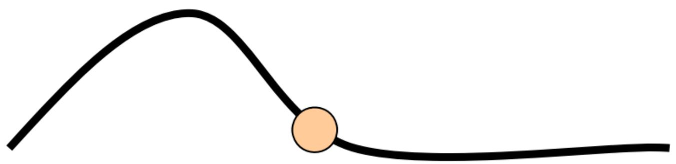
https://www.youtube.com/watch?v=t_ZB0JViW68

Constrained Game-Engine Loop



Constraint Representation

- Often with an implicit function $C(p, p', p'') = 0$.
- Example: position constraint on a curve: $C(x, y) = 0$
  - Constraints dependent only on position are called *holonomic*.



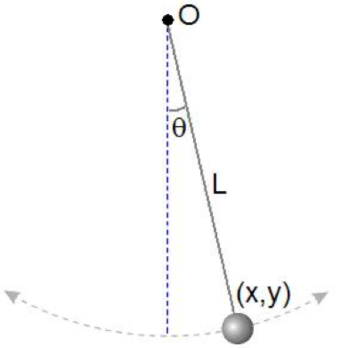
- Equality constraints: $C(p, p', p'') = 0$
- Inequality constraints: $C(p, p', p'') \geq 0$
  - Examples?

Differential Equality Constraints

- Our purpose: Move from $\vec{p}(t)$ to $\vec{p}(t+\Delta t)$ and conserve:

$$C(\vec{p}(t)) = C(\vec{p}(t+\Delta t)) = 0$$

- Algebraic interpretation: the following are conserved:
  - $C(\vec{p}(t)) = 0$ (position)
  - ${dC}/{dt} = 0$ (constraint velocity)
  - ${d^2C}/{dt^2} = 0$ (constraint acceleration)
- Because function should be constant 0!



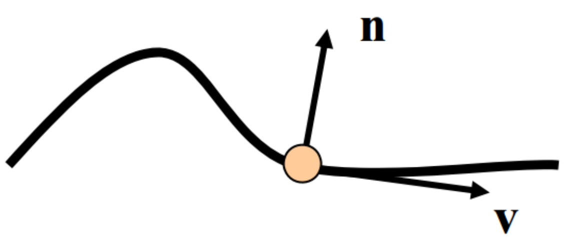
$C(p) = |p| - L$

$C(p, p', p'') = 0$

Velocity Constraint

- Chain rule (single constraint, single bodies):

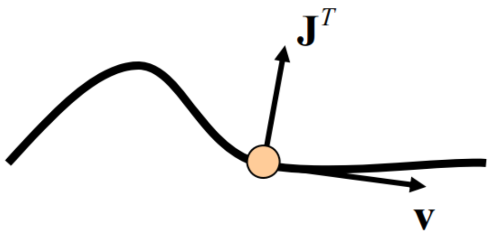
$$\frac{dc}{dt} = \frac{\partial c}{\partial p} \cdot \frac{dp}{dt} = \frac{\partial c}{\partial p} \cdot \vec{v} = 0$$



- Multiple constraints, multiple bodies:

$$C = \{c_1, \cdots c_n\}, P = \{p_1, \cdots p_m\}$$

- We get the Jacobian $\mathrm{J}_{n \times m} = \left(\frac{\partial c_i}{\partial p_j}\right)$



- Kinematic velocity constraint:

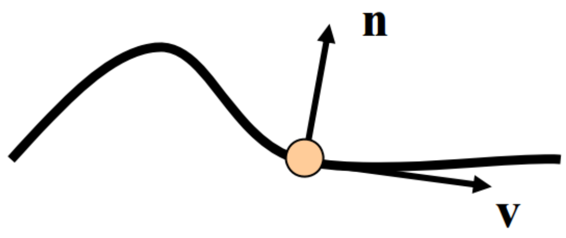
$$\mathrm{J}\vec{v} = 0$$

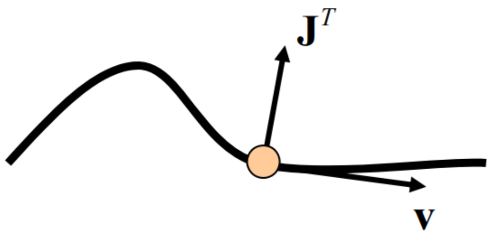
Velocity Constraint

- $\mathrm{J}\vec{v} = 0$ means "$\vec{v}$ is in the right null space of J"

- Dimension of null space ⇔ degrees of valid movement.
- Full rank of J ➔ null space empty ➔ no allowed movement!
  - Physical example: move on two curves at the same time.

- How do we adhere to allowed velocity all the time?



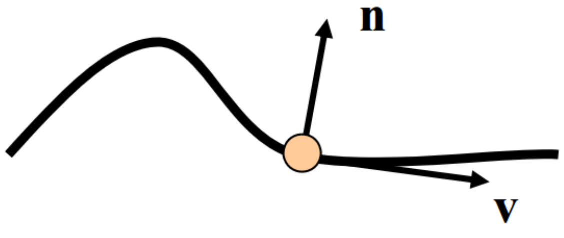


$C(p, p', p'') = 0$

Acceleration Constraint

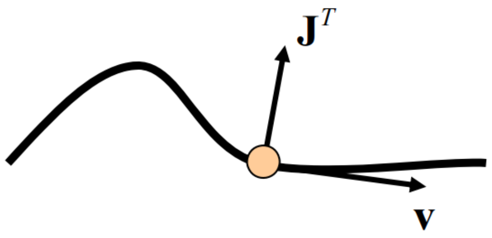
- Acceleration constraint: $\frac{d^2C}{dt^2} = \frac{dJ}{dt}\vec{v} + J\vec{a} = 0$
- Net external force: $\overrightarrow{F_E}$.
- Mass matrix:

$$M_{3mx3m} = diag(M_1, M_1, M_1, \cdots, M_m, M_m, M_m)$$



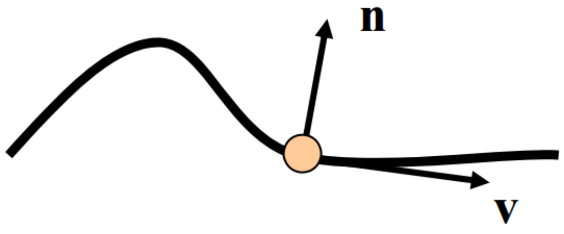
- To obey the constraint, we need to assume a virtual force $\overrightarrow{F_c}$.
- Dynamic force equation:

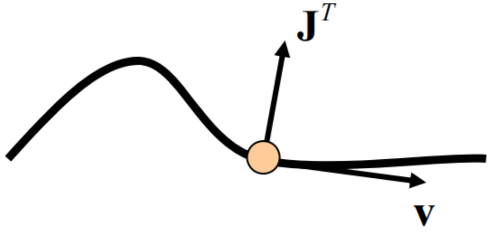
$$\frac{dJ}{dt}\vec{v} + JM^{-1}(\overrightarrow{F_E} + \overrightarrow{F_c}) = 0$$



The Constraint Force

- The constraint force diverts free (unconstrained) movement to valid movement.
- Geometric intuition: constraint force "kills" invalid movement and nothing else.
- Principle of virtual work: $\overrightarrow{F_c}$ does no work on $\vec{v}$.
- Motivation: (energy-wise) lossless constraint.
- Also: since it never causes illegal movement!
  - Where have we seen this?



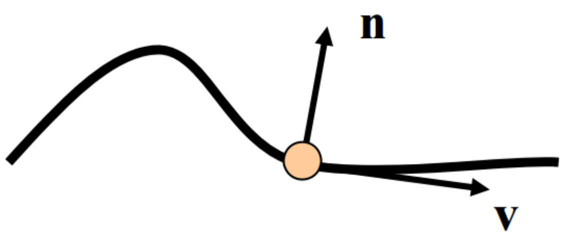


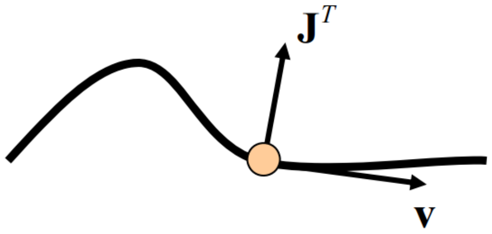
The Constraint Force

- Principle of virtual work: $\overrightarrow{F_c} \cdot \vec{v} = 0.$
- We have that $J\vec{v} = 0.$
  ➔ $\overrightarrow{F_c}$ is spanned by the rows of J:

$$\overrightarrow{F_c} = \sum_i \lambda_i J(I,.) \; = J^T \lambda$$

- λ : Lagrange Multiplier.
- How do we get λ in order to compute $\overrightarrow{F_c}$?





$\overrightarrow{F_c} = J^T \lambda$

Direct Solution

- To get : $\frac{d^2C}{dt^2} = 0$, we need to solve for the Lagrange Multiplier:

$$\frac{dJ}{dt}\vec{v} + JM^{-1}\left(\overrightarrow{F_E} + \overrightarrow{F_c}\right) = \frac{dJ}{dt}\vec{v} + JM^{-1}\left(\overrightarrow{F_E} + J^T\lambda\right)$$

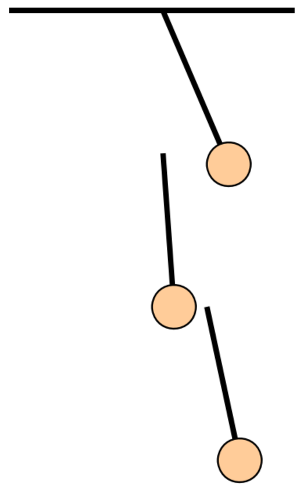
- We get:

$$\boxed{JM^{-1}J^T}\lambda = \frac{dJ}{dt}\vec{v} + JM^{-1}\overrightarrow{F_E}$$

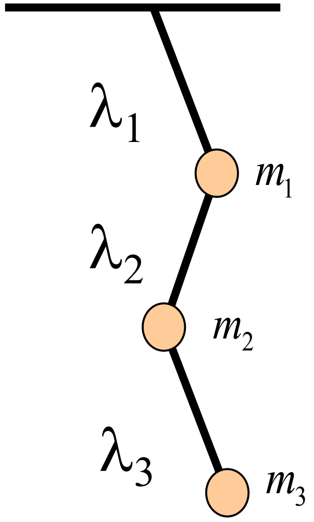
Square $|C|\times|C|$ matrix

- Disadvantages:
  - Expensive linear solve
  - $J, \frac{dJ}{dt}$ change at each point p !
  - Might need complicated derivatives: $\frac{dJ}{dt}$.
  - Discrete integration combined with this will make an inexact solution.

Alternative: Sequential impulses



- Previously: working with forces to get valid acceleration.
- Now: working with impulses to get valid velocity.
- Sequential impulses (SI)
  - Applying impulses at each constraint iteratively to correct velocity.
  - Quite stable and converges to a global solution.
- Pro:
  - Easier friction and collision handling.
  - Velocity rather than acceleration.
  - In a time step, impulse ⇔ force.
- Con: velocity constraints are not precise ➔ might produce position drift ➔ breaking the constraint.



Sequential impulses

Step 1

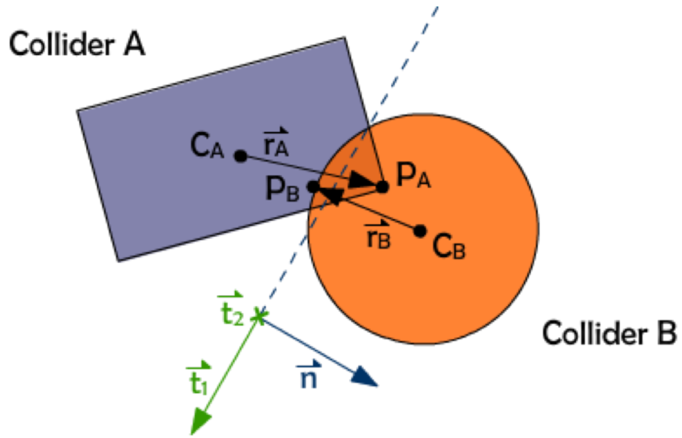
Integrate applied forces, yielding to tentative velocities

Step 2

Apply impulses sequentially for all constraints to correct the velocity errors

Step 3

Use the new velocities to update the positions



http://allenchou.net/wp-content/uploads/2013/12/contacts-figure.png

Computing the Impulses

- A single constraint i currently has $J_i v \neq 0$.
  - J_i - row vector, normal to constraint at closest point.
- Change velocity by Δv to get $J_i(v + \Delta v) = 0$.
- $\Delta P_i = \int_t^{t+\Delta t} \overrightarrow{F_c} dt$. For infinitesimal time step:

$$\Delta P_i \parallel \overrightarrow{F_c} \;\; \Rightarrow \;\; \Delta P_i = {J_i}^T \lambda_i = M \Delta v$$

  - (a different Lagrange multiplier that we need to compute)
- We compute λ_i:

$$J_i\left(v + M^{-1} {J_i}^T \lambda_i\right) = 0$$

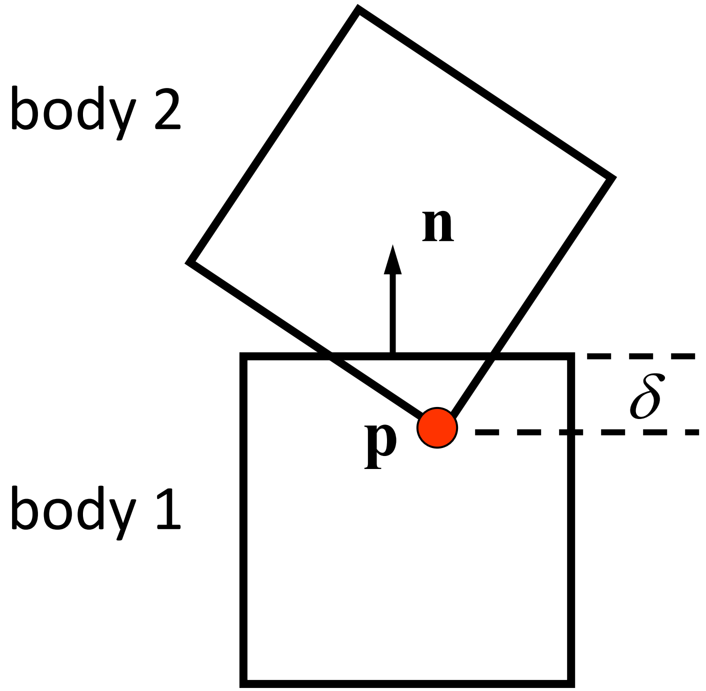
$$\lambda_i = \frac{-J_i v}{J_i M^{-1} {J_i}^T}$$

Sequential Integration

- As solving a constraint for a body may influence the solving for another one, we need an iterative process
  - ```
    while (!done) {
          for all constraints c do solve c;
      }
    ```
  - Convergence is usually ensured by reaching either a maximal amount of iterations or a minimal change in every constraints
    ```

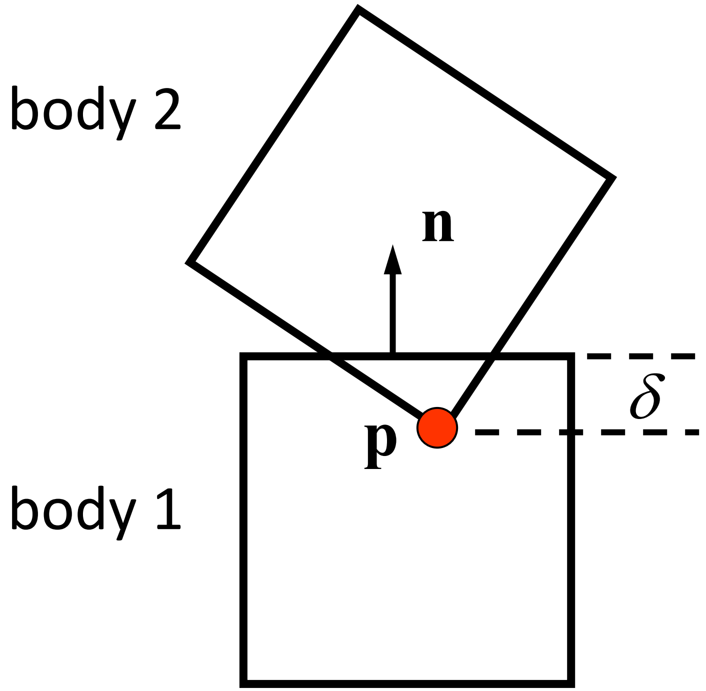
Inequality Constraints

- Constraints are inactive when $C(p) > 0$.
- When they becomes active, solve for them.
  - Otherwise ignore.
- Example: collision and interpenetration.



Collision Constraint

- In previous iteration: x_i is out of the object.
- After position integration: x_{i+1} is penetrating.
- Penetration normal: $\hat{n}$ (pointing outwards)
- Must make sure: the projected point x_{i+1} is non-penetrating.
- Constraint: $C(x_{i+1}) = (x_{i+1} - x_i)\hat{n} \geq 0.$
- Velocity constraint: $\vec{v} \cdot \hat{n} \geq 0$
- Note: Can invalidate other constraints.



Working with rigid bodies

- Constraints depend on chosen center x and orientation q: $\mathrm{c}(\mathrm{x}, \mathrm{q}) = 0$.
- As such, velocity constraints are of the form: $\frac{dC}{dt}(v, \omega) = 0$
- Mass matrix consequently contains tensors of inertia:

$$M = diag\{M_1, M_1, M_1, I_1 \quad, \cdots, M_m, M_m, M_m, I_m \quad\}$$

Collision Resolution as Constraint Solving

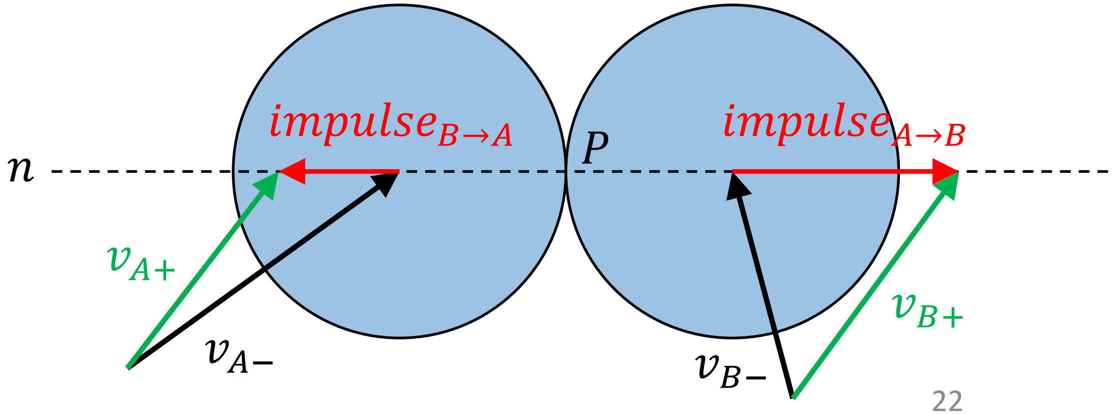
$$\frac{dC}{dt} = (\bar{v}_{A+} - \bar{v}_{B+}) \cdot \hat{n} =$$
$$(\vec{v}_{A+} + \vec{\omega}_{A+} \times \vec{r}_A - \vec{v}_{B+} - \vec{\omega}_{B+} \times \vec{r}_B) \cdot \hat{n} =$$
$$\begin{pmatrix} \hat{n} \\ \vec{r}_A \times \hat{n} \\ -\hat{n} \\ -\vec{r}_A \times \hat{n} \end{pmatrix} \begin{pmatrix} \vec{v}_{A+} \\ \vec{\omega}_{A+} \\ \vec{v}_{B+} \\ \vec{\omega}_{B+} \end{pmatrix} = Jv$$

Handy Identities

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

$$M = \begin{pmatrix} M_A & & & \\ & I_A & & \\ & & M_A & \\ & & & I_B \end{pmatrix}$$

$$\lambda_i = \frac{-Jv}{JM^{-1}J^T}$$



Collision Resolution as Constraint Solving

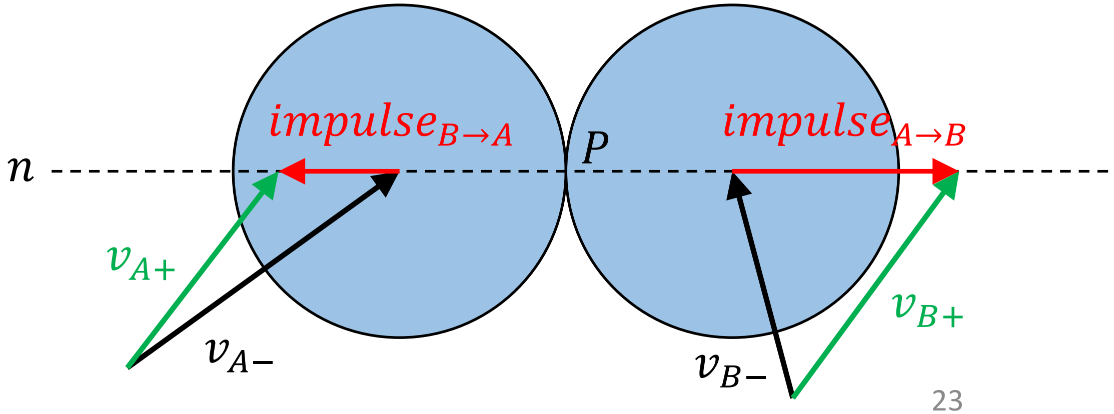
- We get the "well-known":

$$\lambda = \frac{-(1 + C_R)[(\bar{v}_{A-} - \bar{v}_{B-}) \cdot \hat{n}]}{\left(\frac{1}{m_A} + \frac{1}{m_B}\right) + [(\vec{r}_A \times \hat{n})^T I_A^{-1} (\vec{r}_A \times \hat{n}) + (\vec{r}_B \times \hat{n})^T I_B^{-1} (\vec{r}_B \times \hat{n})]}$$

- ...For $C_R = 0$.
  - Explanation: we solved for "minimum velocity to resolve".
- Elastic restitution is an extra velocity bias.

Handy Identities

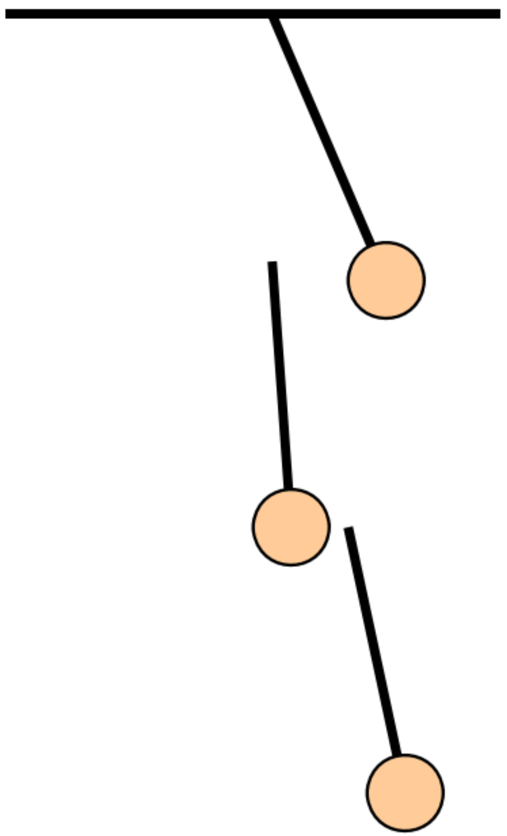
$$\mathbf{A} \cdot (\mathbf{B} \times \mathbf{C}) =$$
$$\mathbf{C} \cdot (\mathbf{A} \times \mathbf{B}) =$$
$$\mathbf{B} \cdot (\mathbf{C} \times \mathbf{A})$$



$$\lambda_i = \frac{-Jv}{JM^{-1}J^T}$$

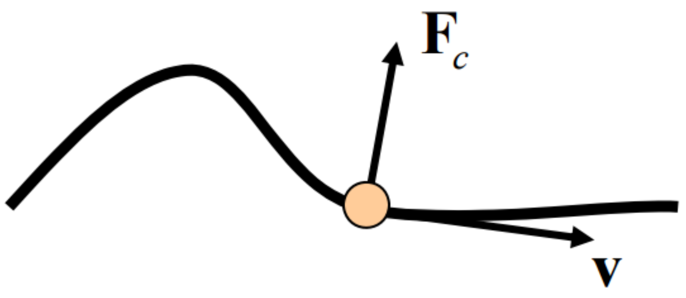
Position Drift

- Even if velocities are fixed, the position integration might drift
  - Since discrete time step.
- Solution: nothing simple
  - Like interpenetration resolution
- Possible easy fix: project positions as well!
  - More on that in position-based dynamics…



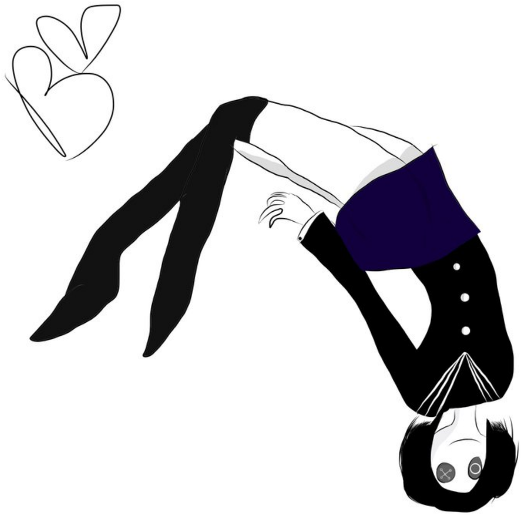
Alternative: Reduced Coordinate Simulation

- Instead of breaking & correcting => only produce force / torque along a specific DOF.
  - Reducing possible movement.
- impulse-based is a full coordinate simulation.
- Pro: faster simulation.
  - less calculations.
  - less tuning.
- Con: more vulnerable to numerical instability.
  - might be difficult to parameterize the DOF system.
- For instance: $Jv = 0$ => use only null space of J!



Motion control

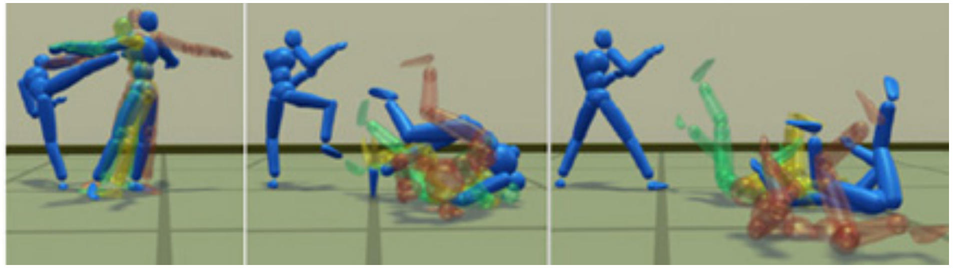
- Imagine you have rigid bodies moving and constrained correctly according to the forces you apply.
- Letting them 'live' on their own is fine for passive objects.
  - projectile, furniture, environmental objects *etc*.
- Living beings produce motion!
  - Otherwise they will just fall onto the ground at the beginning of the game



http//:img13.deviantart.net/8e3c/i/2015/333/d/7/a_falling_puppet_by_xxxblakexxx-d9ii9wq.jpg

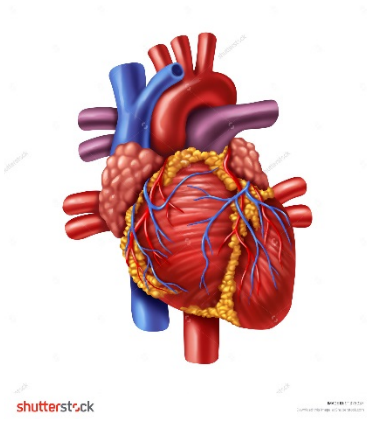
Motion Control

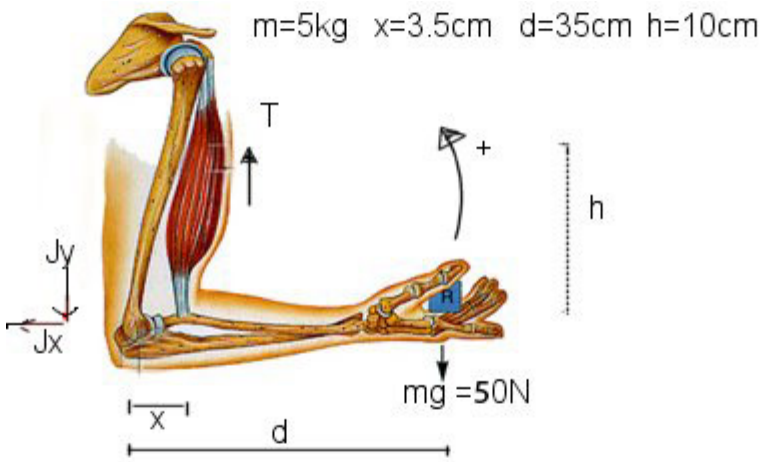
- Animated bodies are controlled using a mix of kinematics and dynamics.
- Kinematics: replay and slightly adapt pre-recorded motions.
- Dynamics: passively animate objects reacting to external forces (*e.g.* human ragdoll).
- A real-time controller switches from one to the other according to events, forces, poses *etc.*



Motion Control

- Imagine that you want to actively actuate rigid bodies using forces and torques.
  - not yet in games but will probably in a near future…
- Actuators generate motion from within the bodies
  - Joint torques
  - External forces
  - Virtual forces
  - Muscle forces
  - *contractile elements*



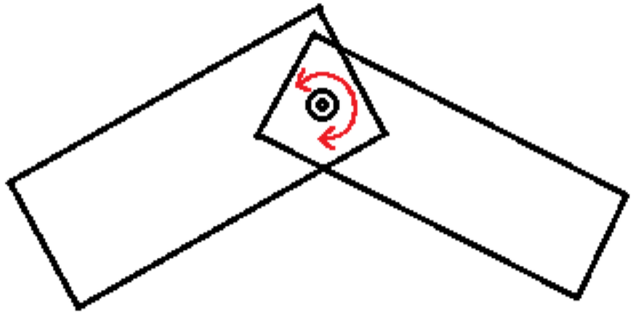


continuummechanics.org/cm/index.html

https://www.youtube.com/watch?v=wuJwwHdAD9Q

Joint Torques

- Most straightforward actuation model.
- Joint torques directly generate torques for each actuated DOF.
- Assuming that there is a ‘fake’ motor at the location of the joint that can produce torque.
- e.g. increase a joint angle ⇔ positive torque.
- Very useful for tracking pre-recorded motions or any other error-based poses (*e.g.* balance pose)
  - Amount of applied torque depends directly on the error



http://files.maartenbaert.be/extremephysics/illustration-hinge-joint.png https://www.youtube.com/watch?v=nONBRvKjEUQ

External Forces

- Applying external forces on the right object, local position, and for the right amount of time.
  - Difficult to produce realistic motions.
  - Does not really fit natural motions.
    - Motion originates from inside!
- Similar to a puppetry technique.
  - System control must be mastered.
- But very useful for the control of the global orientation and position of a complex system (root node).

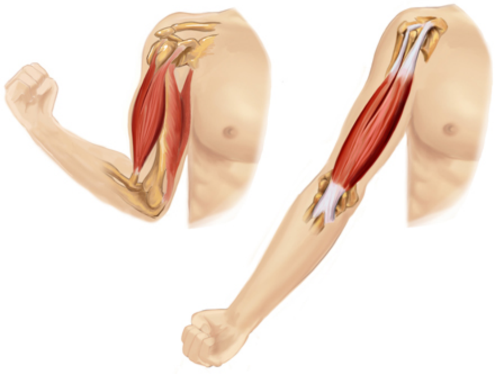
http://previews.123rf.com/images/pkdinkar/pkdinkar1109/pkdinkar110900022/10551395-employee-as-a-puppet-on-strings-Stock-Vector-puppet-man-marionette.jpg

Virtual Forces

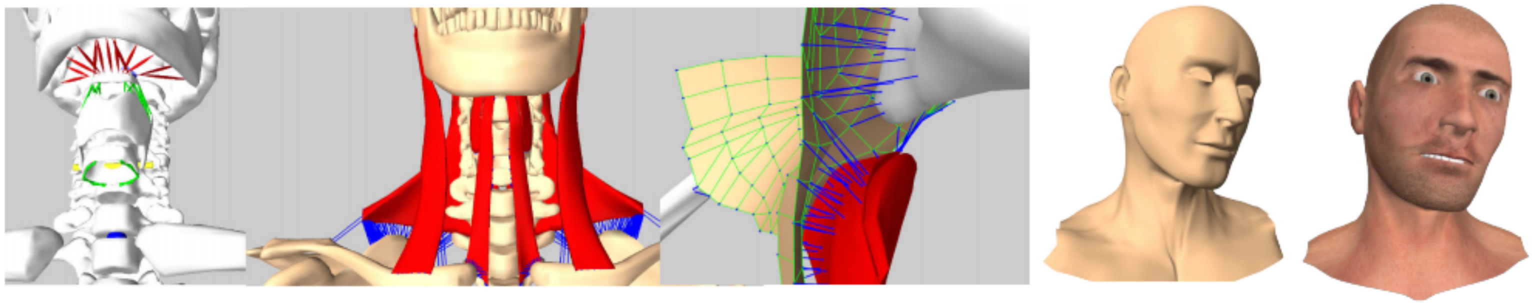
- Not really an actuation method.
- Emulate the effect of applying an external force by computing the equivalent joint torque.
- Use the relation between joint rotation and position where a force is applied (by the Jacobian of the system).

Muscle Forces

- Motion actually comes from the contraction of muscles.
  - they also produce torques at joints, but not in a 'fake motor' way.
- Emulating physical behavior to the model => "realistic" actuation of rigid bodies.
  - commonly used in biomechanics / motion analysis
  - the muscle model is usually a combination of non-linear springs and dampers.



https://jonbarron.org/sites/default/files/arm-muscle.jpg



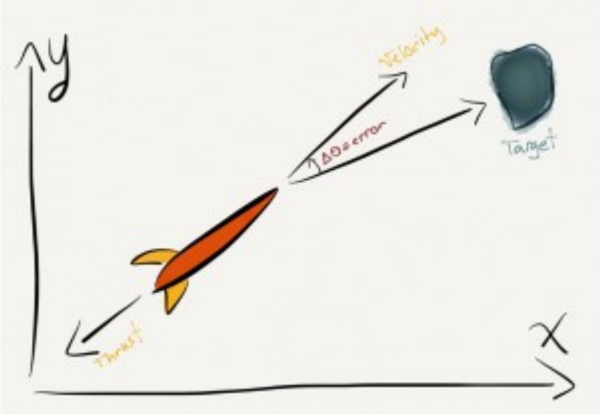
[Lou *et al.* 2013] "Physics-based Human Neck Simulation"

Controller design

- Joint-space motion control.
  - defines and tracks kinematics target.
- Stimulus-response network control
  - genetically evolves controllers according to objectives.
- Constrained dynamics optimization control.
  - finds optimal torques through online optimization.

PD Controller

- Proportional Derivative (PD) controller
  - Used to compute joint torques linearly proportional to the difference between the current state and the target state.
  - Based on joint orientation and angular velocity.



http://www.gamedev.net/uploads/monthly_10_2014/ccs-224713-0-48143500-1414694390.jpg

PD Controller

$$\tau = k_p(\theta_d - \theta) + k_v(\dot{\theta}_d - \dot{\theta})$$

Where:

- τ is the generated joint torque.
- θ_d and θ the desired and current joint angles.
- $\dot{\theta}_d$ and $\dot{\theta}$ the desired and current joint angular velocity.
- k_p and k_v are the controller gain.
  - Regulating how responsive the controller is to deviation.

PD Controller

- Con: The motion-controller designer needs to carefully define the gain values.
  - too high k_p: producing stiff unresponsive motions.
  - too low k_p: not track target correctly.
  - too high k_v: converging too slowly to the target.
  - too low k_v: producing oscillations.
- …Expect to spend time fine tuning gains!

$$\tau = k_p(\theta_d - \theta) + k_v(\dot{\theta}_d - \dot{\theta})$$