UC San Diego

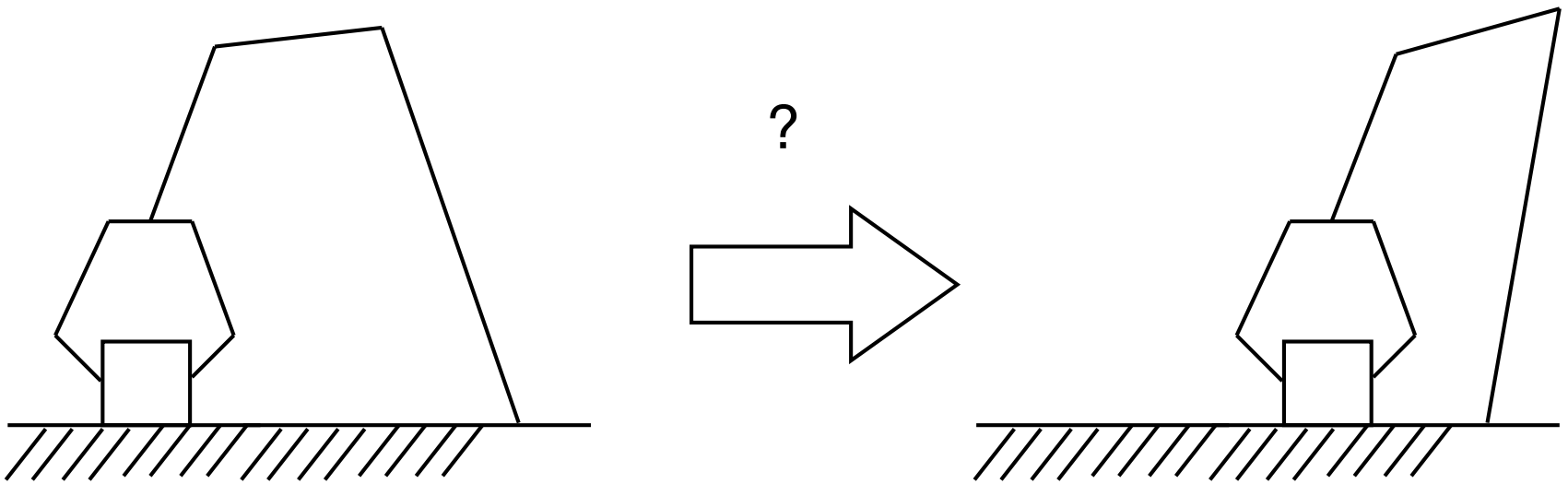
Robot Control and Physical Simulation

Fanbo Xiang

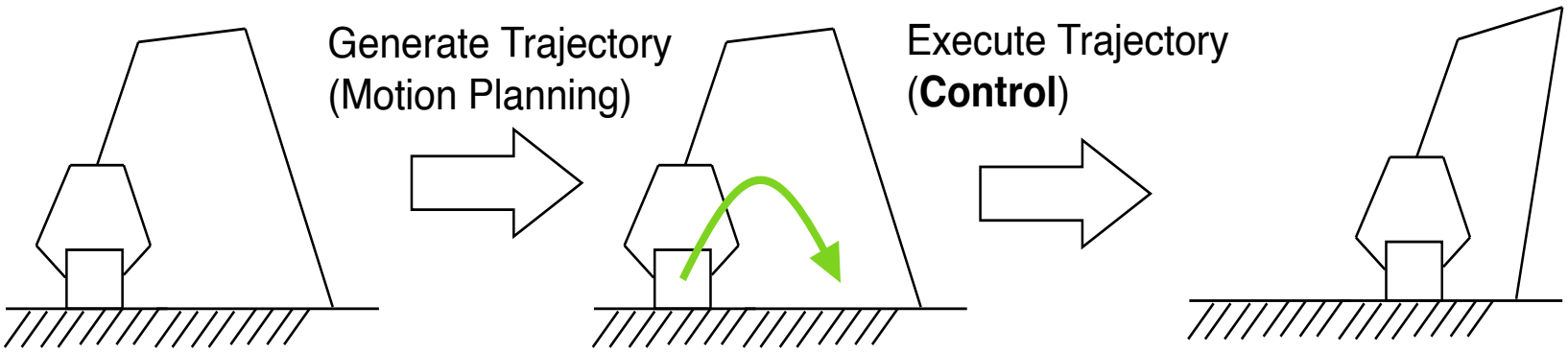
CSE291-G00 - Spring 2020

Start from a Problem

Suppose we have a robot, how do we use it to move objects?



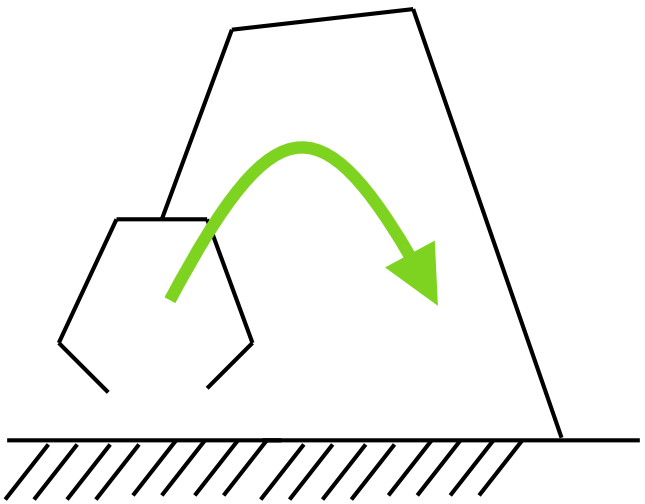
Plan and Control



Motion planning algorithms can generate a trajectory, specifying the position, velocity, and acceleration of the robot at any point.

Now we need to know how to control the robot to follow such trajectory.

What we have learned



Lagrangian dynamics of the robot

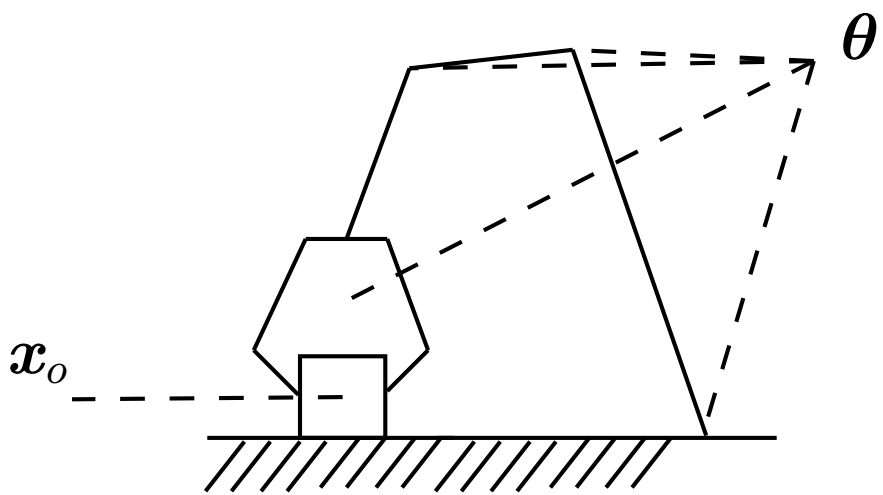
$$\boldsymbol{M}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} + \boldsymbol{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\dot{\boldsymbol{\theta}} + \boldsymbol{N}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \boldsymbol{\Upsilon}$$

We know how to compute the actuation needed to achieve acceleration given by the trajectory.

What is still missing

The previous equation does not account for the grasped object. So let's add the DOFs from the object into our system. Let $\boldsymbol{x}_o$ be the 6 degree pose of the grasped object. And the state of the whole system becomes

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{x}_o \end{bmatrix}$$



Poll 1

Can we directly solve

$$\boldsymbol{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \Upsilon$$

of the new system and find out how to achieve certain accelerations $\ddot{\boldsymbol{q}}$ with external force. Can we directly use it to control the system? Why and why not?

A. We are able to control the system.

B. We know how to achieve certain acceleration, but some forces are out of our control.

C. We cannot find the acceleration from forces so we cannot control the system.

D. The whole system cannot be formulated into a unified Lagrangian view.

Poll 1

Can we directly solve

$$L(q, \dot{q}) = \Upsilon$$

of the new system and find out how to achieve certain accelerations $\ddot{q}$ with external force. Can we directly use it to control the system? Why and why not?

A. We are able to control the system.

B. We know how to achieve certain acceleration, but some forces are out of our control.

C. We cannot find the acceleration from forces so we cannot control the system.

D. The whole system cannot be formulated into a unified Lagrangian view.

Contact Point Twist (Robot)

When robot grasps an object, there should be no relative motion at each contact point between robot finger and the point on the object.
We know the pose of all contact point on the finger is a function of $\boldsymbol{\theta}$, $\boldsymbol{f}_h(\boldsymbol{\theta})$. Let

$$\boldsymbol{J}_h = \frac{d\boldsymbol{f}_h}{d\boldsymbol{\theta}}$$

$\boldsymbol{J}_h$ is called the **hand Jacobian**. Let the contact point velocity be $\dot{\boldsymbol{x}}_c$

$$\dot{\boldsymbol{x}}_c = \boldsymbol{J}_h \dot{\boldsymbol{\theta}}$$

Contact Point Twist (Object)

Recap: **Grasp Map** G maps contact wrench f_c to net wrench F_o.
Fact: G^T maps object twist x_o to contact twist $\dot{x}_c$.

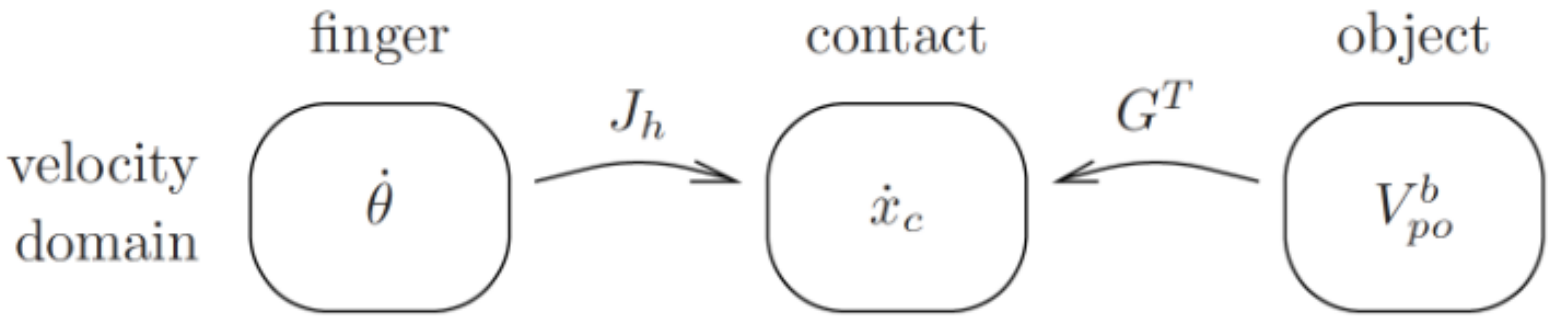
$$\dot{x}_c = G^T \dot{x}_o$$

Since contact point on robot and object should move at the same velocity

$$G^T \dot{x}_o = \dot{x}_c = J_h \dot{\theta}$$

This equation is called **Grasp Constraints**.

Diagram

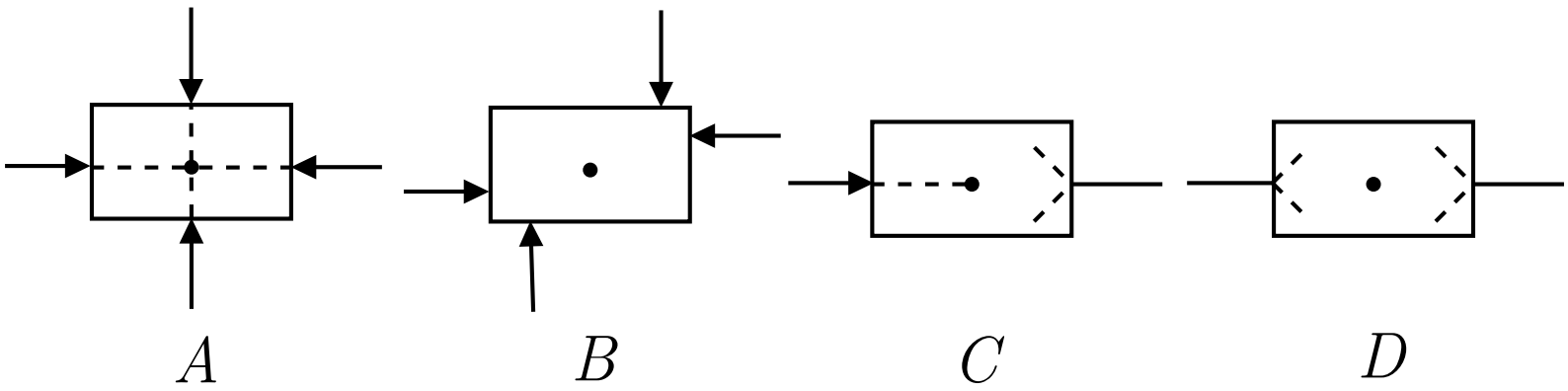

finger
contact
object
J_h
G^T
velocity domain
$\dot{\theta}$
$\dot{x}_c$
V_{po}^b

Manipulable

- A multi-fingered grasp is **manipulable** at a configuration $(\boldsymbol{\theta}, \boldsymbol{x}_o)$ if for any $\dot{\boldsymbol{x}}_o$ there exists $\dot{\boldsymbol{\theta}}$ that satisfies grasp constraints.
- A grasp is **manipulable** iff $R(\boldsymbol{G}^T) \subset R(\boldsymbol{J}_h)$.
- Note: manipulable does not involve force/wrench.

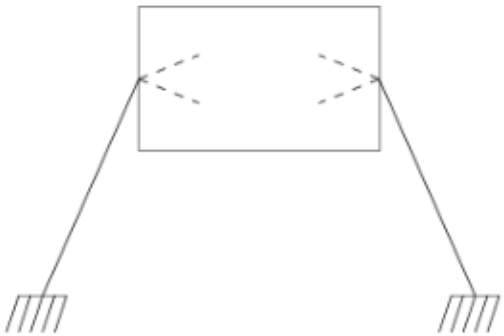
Poll 2: Recap Force Closure

Consider in 2D space (2 translation direction and 1 rotation direction). Which of the following grasps are in force closure.

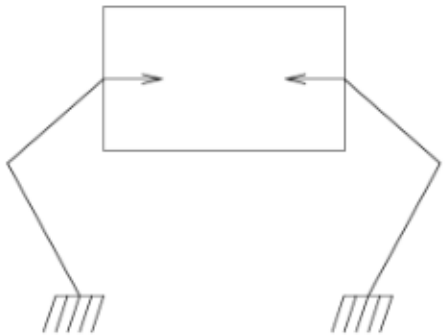


Manipulable and Force Closure

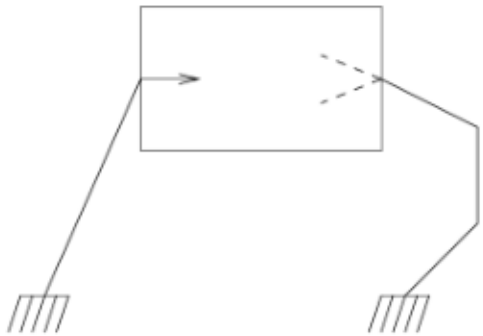
Manipulable and force closure are “independent”.



force closure
not manipulable



not force-closure
manipulable



not force-closure
not manipulable

Back to Grasp Constraints

$$\boldsymbol{G}^T \dot{\boldsymbol{x}}_o = \boldsymbol{J}_h \dot{\boldsymbol{\theta}}$$

Rearrange

$$\begin{bmatrix} \boldsymbol{J}_h & -\boldsymbol{G}^T \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{\theta}} \\ \dot{\boldsymbol{x}}_o \end{bmatrix} = 0$$

Let

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{x}_o \end{bmatrix}, \boldsymbol{A}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{J}_h & -\boldsymbol{G}^T \end{bmatrix}$$

Grasp constraints becomes

$$\boldsymbol{A}(\boldsymbol{q}) \dot{\boldsymbol{q}} = 0$$

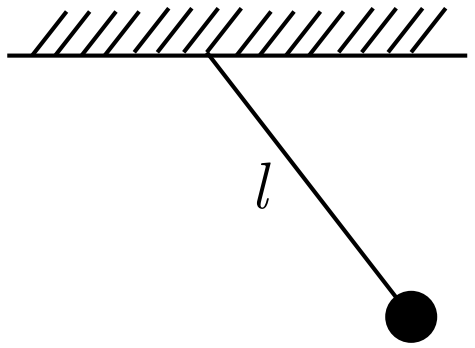
General Constraints

In fact, constraints with the form

$$\boldsymbol{A}(\boldsymbol{q})\dot{\boldsymbol{q}} = \mathbf{0}$$

is very common. Consider a pendulum.
It satisfies

$$\|\boldsymbol{q}\|_2^2 = l^2$$



Take the time derivative

$$\begin{bmatrix} 2x & 0 \\ 0 & 2y \end{bmatrix} \dot{\boldsymbol{q}} = \mathbf{0}$$

$$\boldsymbol{q} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Note: $\boldsymbol{A}(\boldsymbol{q})\dot{\boldsymbol{q}} = 0$ is so common that it gets a name: **Pfaffian constraints**

Lagrangian with Constraints

$$\frac{d}{dt}\frac{\partial \boldsymbol{L}}{\partial \dot{\boldsymbol{q}}} - \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{q}} = \boldsymbol{\Upsilon}$$

Now it is time to fix our Lagrangian equation when there are constraints.

$$\frac{d}{dt}\frac{\partial \boldsymbol{L}}{\partial \dot{\boldsymbol{q}}} - \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{q}} + \underbrace{\boldsymbol{A}^T(\boldsymbol{q})\boldsymbol{\lambda}}_{\text{constraint force}} = \boldsymbol{\Upsilon}'$$

$$\boldsymbol{A}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0}$$

where $\boldsymbol{\lambda}$ is the Lagrange multiplier, and we need to compute $\boldsymbol{\lambda}$ from $\boldsymbol{A}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0}$

Constraint Force

Why do we assume constraint forces satisfy $\boldsymbol{A}^T(\boldsymbol{q})\boldsymbol{\lambda}$?

Look at $\boldsymbol{A}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0}$. The system cannot move in the direction of any row of $\boldsymbol{A}$, otherwise the constraint is violated.

In fact, the constraints apply forces in such directions to resist motion. So the constraint force is a linear combination of the rows of $\boldsymbol{A}$, or $\boldsymbol{A}^T\boldsymbol{\lambda}$.

This assumption is known as **d'Alembert's principle.**

Lagrange with Constraints

$$\frac{d}{dt}\frac{\partial \boldsymbol{L}}{\partial \dot{\boldsymbol{q}}} - \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{q}} + \boldsymbol{A}^T(\boldsymbol{q})\boldsymbol{\lambda} = \boldsymbol{\Upsilon}'$$

$$\boldsymbol{A}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0}$$

Substitute $\boldsymbol{L}$ with $\frac{1}{2}\dot{\boldsymbol{q}}\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}} - V(\boldsymbol{q})$, we get

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{N}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{A}^T(\boldsymbol{q})\boldsymbol{\lambda} = \boldsymbol{\tau}$$

We also have

$$\boldsymbol{A}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0} \implies \boldsymbol{A}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{A}}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0}$$

Forward/Inverse Dynamics

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{N}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{A}^T(\boldsymbol{q})\boldsymbol{\lambda} = \boldsymbol{\tau}$$

$$\boldsymbol{A}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{A}}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0}$$

Forward dynamics: given $\boldsymbol{\tau}$ compute $\ddot{\boldsymbol{q}}$

Inverse dynamics: given (part of) $\ddot{\boldsymbol{q}}$ compute $\boldsymbol{\tau}$

Control

Now let's look at what we have.

- A trajectory $(\boldsymbol{\theta}_d, \dot{\boldsymbol{\theta}}_d, \ddot{\boldsymbol{\theta}}_d)$
- Forward dynamics $\ddot{\boldsymbol{\theta}} = \text{FD}(\boldsymbol{\tau})$
- Inverse dynamics $\boldsymbol{\tau} = \text{ID}(\ddot{\boldsymbol{\theta}})$

If we use inverse dynamics to match $\ddot{\boldsymbol{\theta}}$ all the time, we are done!

However, real world is not perfect. There will be some error.

$$\boldsymbol{e} = \boldsymbol{\theta} - \boldsymbol{\theta}_d$$

where $\boldsymbol{\theta}$ is the real joint values and θ_d is the desired joint values.

Computed Torque Control Law

$$\ddot{\boldsymbol{\theta}} \leftarrow \ddot{\boldsymbol{\theta}}_d - \boldsymbol{K}_v \dot{\boldsymbol{e}} - \boldsymbol{K}_p \boldsymbol{e}$$

$$\boldsymbol{\tau} \leftarrow \mathrm{ID}(\ddot{\boldsymbol{\theta}})$$

where $\boldsymbol{K}_v, \boldsymbol{K}_p$ are constant matrices.
$\boldsymbol{e}$ term: if we are falling behind, $-\boldsymbol{K}_p \boldsymbol{e}$ should be positive, so we accelerate more to catch up (and vice versa).

$\dot{\boldsymbol{e}}$ term: if error value is decreasing (we will fall behind if we let it go this way), we also accelerate more (and vice versa).

Convergence

How should we set $\boldsymbol{K}_p, \boldsymbol{K}_v$?

- If $\boldsymbol{K}_p, \boldsymbol{K}_v \in \mathbb{R}^{n \times n}$ are positive definite symmetric matrices, then control law applied to the system results in exponential tracking (error decreases exponentially).
- Personal suggestion: increase $\boldsymbol{K}_p$ until it overshoots a little; increase $\boldsymbol{K}_v$ until it does not overshoot.

PD Control

PD Control law as the form

$$\boldsymbol{\tau} = -\boldsymbol{K}_v \dot{\boldsymbol{e}} - \boldsymbol{K}_p \boldsymbol{e}$$

where $\boldsymbol{K}_p, \boldsymbol{K}_e$ are positive definite matrices and $\boldsymbol{e} = \boldsymbol{\theta} - \boldsymbol{\theta}_d$.

- Does not compute inverse dynamics at all.
- No theoretical guarantee in general.
- Not practical in general.
- Even if $\boldsymbol{e}$ converges, it usually does not converge to $\boldsymbol{0}$.

PID Control

PD Control law as the form

$$\boldsymbol{\tau} = -\boldsymbol{K}_v \dot{\boldsymbol{e}} - \boldsymbol{K}_p \boldsymbol{e} - \boldsymbol{K}_i \int_0^t \boldsymbol{e}(t')\, dt'$$

where $\boldsymbol{K}_p, \boldsymbol{K}_e, \boldsymbol{K}_i$ are positive definite matrices and $\boldsymbol{e} = \boldsymbol{\theta} - \boldsymbol{\theta}_d$.

- $\boldsymbol{K}_i$ term: accumulate over all past time steps.
- Inherits all the problems from PD, except
- When e converges, it usually can converge to $\mathbf{0}$.
- Personal suggestion: clip $\int_0^t \boldsymbol{e}(t')\, dt'$.

Augmented PD Control

$$\boldsymbol{\tau} = \mathrm{ID}(\boldsymbol{\theta}_d) - \boldsymbol{K}_v \dot{\boldsymbol{e}} - \boldsymbol{K}_p \boldsymbol{e}$$

- Uses inverse dynamics.
- Compared to computed torque control, $\boldsymbol{K}$s may be harder to tune.
- Does result in exponential tracking if $\boldsymbol{K}_v, \boldsymbol{K}_p$ are positive definite.
- Now you can imagine augmented PID control.
  - Augmented PID control should be able to solve the homework with robot inverse dynamics alone, so you may be able to skip computing grasp constraints with some parameter tuning.

Tuning PID

https://en.wikipedia.org/wiki/PID_controller#Manual_tuning

Effects of *increasing* a parameter independently[22][23]

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor change	Decrease	Decrease	No effect in theory	Improve if K_d small

In general, tune PID in order $\boldsymbol{K}_p$, $\boldsymbol{K}_v$, $\boldsymbol{K}_i$.

Control Wrap-up

- All material above requires thorough understanding.
- Questions?
  - Manipulable
  - Force Closure
  - Pfaffian constraints($\boldsymbol{A}\dot{\boldsymbol{q}} = \boldsymbol{0}$)
  - d'Alembert's principle (constraint force = $\boldsymbol{A}^T\boldsymbol{\lambda}$)
  - Computed Torque, PD, PID, Augmented PD.

Physical Simulation Cont.

Recall the constraint

$$\boldsymbol{A}(\boldsymbol{q})\dot{\boldsymbol{q}} = \mathbf{0}$$

Solving such constraints is actually a core module in physical simulation.

Topics

- **Rigidbody Constraints**
  - Constraint solver for joints
  - Contact, friction, and impact
- Numerical Integration
- Collision Detection
  - Primitive shape collision
  - Broad and narrow phases
  - Convex shape collision

Constraints in Physical Simulation

If we treat a robot as a whole, we can derive generalized coordinates θ for it, however, physical simulation may model the robot joints as constraints.

After all, robot joints are not that different from contact points!

Maximal Coordinate Simulation

Concatenate the position and rotation of **all** objects

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{\alpha}_1 \\ \vdots \\ \boldsymbol{x}_n \\ \boldsymbol{\alpha}_n \end{bmatrix}$$

Let $\boldsymbol{F}$ be the wrench, $\boldsymbol{M}$ be the generalized mass matrix, which combines all mass and inertia.

$$\boldsymbol{F}_C + \boldsymbol{F}_{ext} = \boldsymbol{M} \cdot \ddot{\boldsymbol{q}}$$

https://www.cs.ubc.ca/grads/resources/thesis/Nov02/Michael_Cline.pdf

Lagrange Multiplier Method

Suppose we have m equality constraints.

$$\boldsymbol{C}(\boldsymbol{q}) = \begin{bmatrix} C_1(\boldsymbol{q}) & \cdots & C_m(\boldsymbol{q}) \end{bmatrix}^T = \boldsymbol{0}.$$

There should not be relative velocity or relative acceleration.

$$\dot{\boldsymbol{C}} = \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}} = \boldsymbol{0}$$

Define the constraint's Jacobian matrix

$$\boldsymbol{A} = \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{q}}$$

$$\boldsymbol{A}\dot{\boldsymbol{q}} = \boldsymbol{0}$$

Lagrange Multiplier Method

Remember

$$\boldsymbol{F}_C + \boldsymbol{F}_{ext} = \boldsymbol{M}\ddot{\boldsymbol{q}}, \boldsymbol{F}_C = \boldsymbol{A}^T\boldsymbol{\lambda}, \dot{\boldsymbol{A}}\dot{\boldsymbol{q}} + \boldsymbol{A}\ddot{\boldsymbol{q}} = 0$$

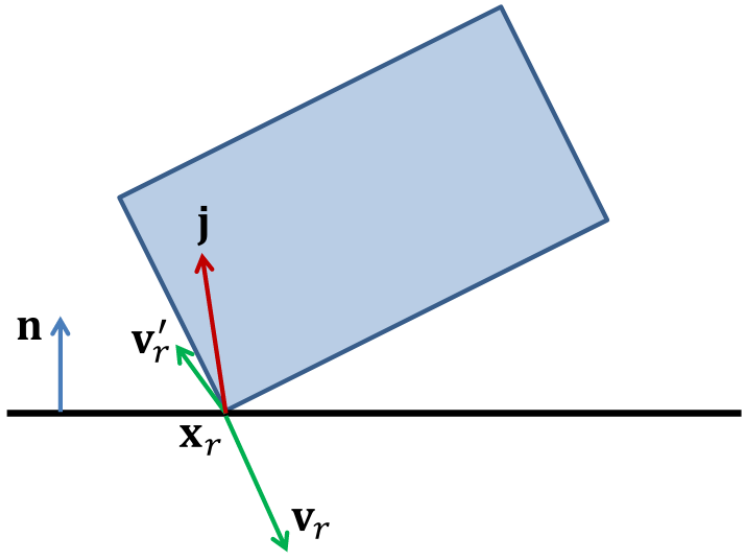
Now we only need to solve a linear system. Let $\boldsymbol{k} \triangleq \dot{\boldsymbol{A}}\dot{\boldsymbol{q}}$

$$\begin{bmatrix} \boldsymbol{M} & -\boldsymbol{A}^T \\ \boldsymbol{A} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{F}_{ext} \\ -\boldsymbol{k} \end{bmatrix}$$

This is exactly the **forward dynamics** we have seen!

Contact Simulation

(Note: **contact** and **impact** are treated differently.)



Contact Simulation

In physical simulation, we model contact as inequality constraint.

$$\boldsymbol{C}_e(\boldsymbol{q}) = \boldsymbol{0}, \dot{\boldsymbol{C}}_e = \boldsymbol{0}, \ddot{\boldsymbol{C}}_e = \boldsymbol{0}$$
$$\boldsymbol{C}_c(\boldsymbol{q}) \geqslant \boldsymbol{0}, \dot{\boldsymbol{C}}_c \geqslant \boldsymbol{0}, \ddot{\boldsymbol{C}}_c \geqslant \boldsymbol{0}$$

Contact also needs to satisfy

- For each contact, acceleration must be non-negative, so $\boldsymbol{A}_c\ddot{\boldsymbol{q}} + \boldsymbol{k}_c \geqslant \boldsymbol{0}$
- All contact forces should be non-negative, so $\boldsymbol{\lambda}_c \geqslant \boldsymbol{0}$
- For each contact, when there is force, there is no acceleration, so $\lambda_i q_i = 0$, meaning $\boldsymbol{\lambda}^T\boldsymbol{q} = \boldsymbol{0}$.

Contact Simulation

$$\begin{bmatrix} \boldsymbol{M} & -\boldsymbol{A}_e^T & -\boldsymbol{A}_c^T \\ \boldsymbol{A}_e & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{A}_c & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda}_e \\ \boldsymbol{\lambda}_c \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ a \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_{ext} \\ -\boldsymbol{k}_e \\ -\boldsymbol{k}_c \end{bmatrix}$$

$$\boldsymbol{a} \geqslant \boldsymbol{0}, \boldsymbol{\lambda}_c \geqslant \boldsymbol{0}, \boldsymbol{a}^T \boldsymbol{\lambda}_c = 0, \ddot{\boldsymbol{q}} \text{ free}$$

Inequality constraints make the linear system more complicated. It becomes a **mixed linear complementary problem** (mixed LCP). Physical simulators typically use **projected Gauss-Seidel (PGS)** to solve these problems.

But in Reality...

Do we really use the Lagrangian Multiplier method above to simulate robot?
No, since numerically solve LCP can be inaccurate, but we do use it in contact simulation.

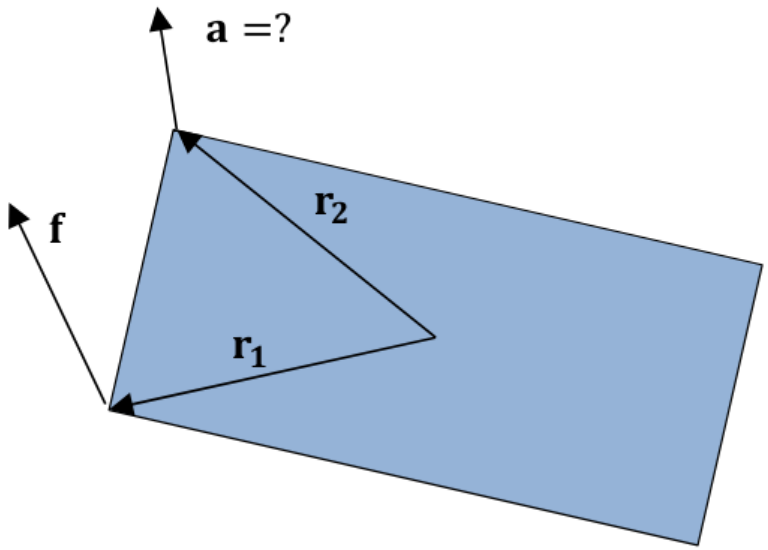
So do we use the original Lagrangian method?
No, since we have more efficient methods.

Constraint Solver

- Lagrange multiplier method
  - Maximal coordinate (All DOFs are considered)
  - Easy to implement and extend
- **Featherstone's method**
  - Newton-Euler equations
  - Reduced coordinate
  - Fast and accurate
  - Very hard to implement it right

Offset Forces

If we apply force at r_1, what is the acceleration of r_2?



Offset Forces

offset acceleration formula: calculate acceleration at r.

$$\boldsymbol{a}_r = \boldsymbol{a} + \dot{\boldsymbol{\omega}} \times \boldsymbol{r} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{r})$$

inverse mass matrix: calculate acceleration at r_2 from force at r_1.

$$\boldsymbol{M}_{12}^{-1} = \begin{bmatrix} 1/m & 0 & 0 \\ 0 & 1/m & 0 \\ 0 & 0 & 1/m \end{bmatrix} - \hat{\boldsymbol{r}}_2 \boldsymbol{I}^{-1} \hat{\boldsymbol{r}}_1$$

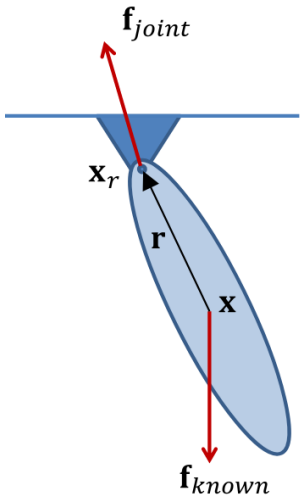
$$\boldsymbol{a}_2 = \boldsymbol{M}_{12} \boldsymbol{f}_1$$

(Note: it is okay to have M_{11} and M_{22} and they are very useful)

Ball Joint

We first consider a 3 DOF ball joint. This is the simplest joint since the rotational DOFs are free and the joint does not provide torque.

We know all external forces f_{known}. By **offset acceleration** formula, we can calculate acceleration at x_r.



Knowing a_r, we can compute joint forces.

$$f_{joint} = -M_{rr} a_r$$

(Note: M is the mass matrix from x_r to x_r)

Simulate A Single Ball Joint

Algorithm

- Apply external forces to get $\boldsymbol{f}_{ext}$ and $\boldsymbol{\tau}_{ext}$
- Compute $\boldsymbol{a}$ and $\dot{\boldsymbol{\omega}}$
- Compute $\boldsymbol{a}_r$ at joint
- Compute $\boldsymbol{M}^{-1}$ and then $\boldsymbol{f}_{joint}$
- Apply $\boldsymbol{f}_{joint}$
- Simulate the motion

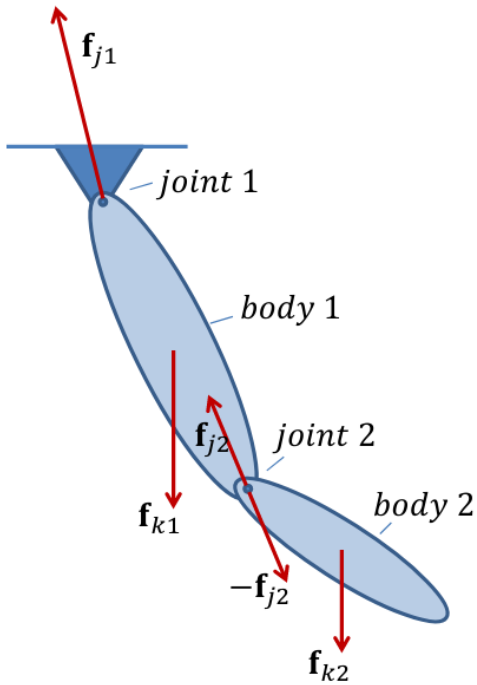
Issue

- Discrete time integration will accumulate errors.
- Solution: project back to correct position.

Simulate Kinematic Chain

Similar to previous example, $\boldsymbol{a}_1^{(1)}$, $\boldsymbol{a}_2^{(1)}$, $\boldsymbol{a}_2^{(2)}$, can be computed from external forces.

We want to make sure $\boldsymbol{a}_{j_1}^{(1)} = 0$ and $\boldsymbol{a}_{j_2}^{(2)} - \boldsymbol{a}_{j_2}^{(1)} = 0$. "No relative acceleration"



$$\boldsymbol{a}_{j_1}^{(1)} = \boldsymbol{M}_{12}^{(1)-1}(-\boldsymbol{f}_{j_2}) + \boldsymbol{M}_{11}^{(1)-1}\boldsymbol{f}_{j_1} + \boldsymbol{a}_1^{(1)}$$

$$\boldsymbol{a}_{j_2}^{(1)} = \boldsymbol{M}_{22}^{(1)-1}(-\boldsymbol{f}_{j_2}) + \boldsymbol{M}_{21}^{(1)-1}\boldsymbol{f}_{j_1} + \boldsymbol{a}_2^{(1)}$$

$$\boldsymbol{a}_{j_2}^{(2)} = \boldsymbol{M}_{22}^{(2)}\boldsymbol{f}_{j_2} + \boldsymbol{a}_2^{(2)}$$

Eventually it simplifies to the form $\boldsymbol{a} = \boldsymbol{M}^{-1}\boldsymbol{f}$ where $\boldsymbol{M}$ is 6×6.

Simulate Kinematic Chain

Algorithms in practice

- Featherstone's algorithm
  - Similar to the previous method.
  - Linear time.
  - May be modified to handle kinematic loops at a cost.

Impulse Based Simulation

In fact, physical simulator does not compute constraint forces, but constraint impulses that directly change velocities.

$$\boldsymbol{j} = \int \boldsymbol{f} dt \approx \boldsymbol{f} \Delta t$$

$$\Delta \boldsymbol{p} = \boldsymbol{j}$$

$$\Delta \boldsymbol{L} = \boldsymbol{r} \times \boldsymbol{j}$$

Take away: when you see impulse, think of it as $\boldsymbol{f} \Delta t$.

Notes on SAPIEN

The simulation backend for SAPIEN, PhysX, simulates with the following methods.

Robot simulation (Articulation): Featherstone's method.

Contact Simulation: Projected Gauss-Seidel.

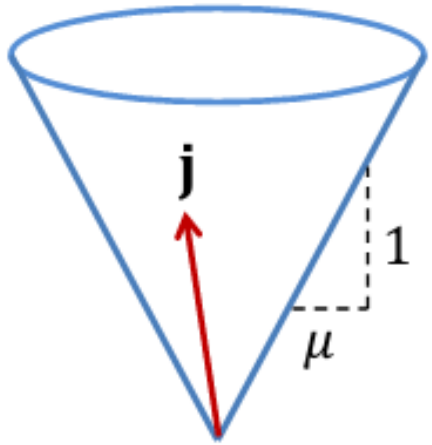
Forward/inverse dynamics computation: Lagrangian view.

Contact Point Friction

(Point Finger Model)

Assuming the Coulomb friction model. $f = \mu F_N$.

If the impulse is within the friction cone, we have static friction. If not, we have kinetic friction. Including friction as a constraint is quite non-trivial.

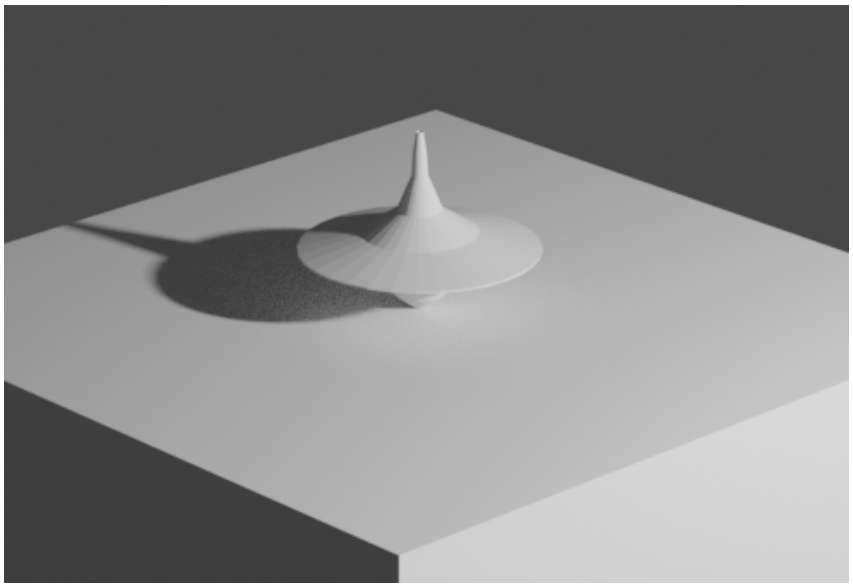


Contact Patch Friction

(Modified Soft Finger Model)

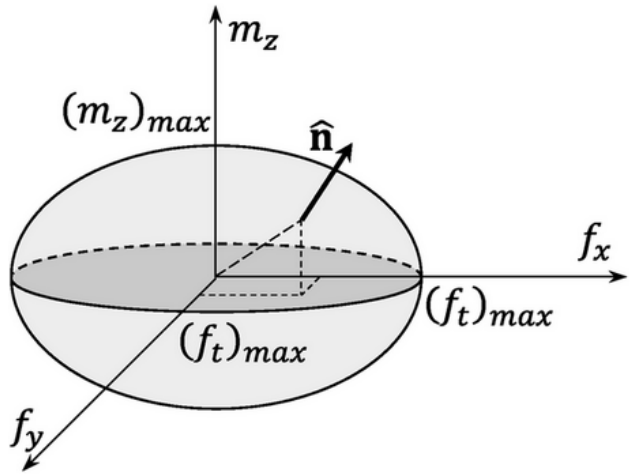
Point friction cannot provide **torsional friction**, but patch friction can.

Some simulators provide a torsional friction constant, but it is inaccurate it can actually be (roughly) computed from contact geometry.



Contact Patch Friction

Uniform 2D surface friction can be modeled as an ellipsoid. m_z is the friction torque, f_x, f_y are x, y direction friction force. Static friction is a point inside the ellipsoid. Kinetic friction is a point on the ellipsoid.



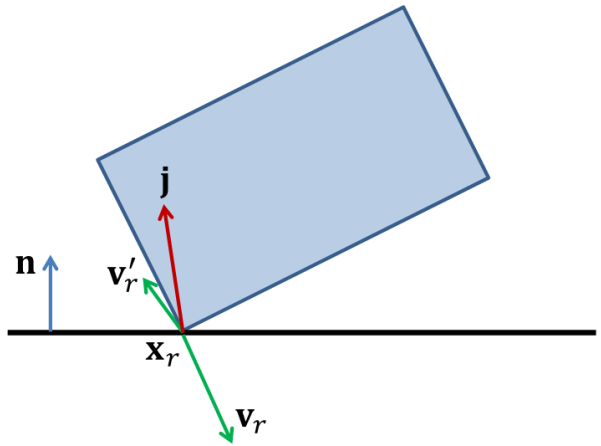
Torsional motion will cause a reduced linear friction. (Spinning objects can be pushed easier)

Determine Friction Coefficient

How do we determine pairwise friction coefficient?
Short answer: you will have to measure it for each pair.

What we end up doing: give a number to each material, and then add, multiply, or take the average. There is no "correct" way to do it.

Impact



When objects collide, they can bounce back, and we call this situation an impact. In this case, we compute an impact impulse j.
Perfectly inelastic collision: object stick to each other.
Perfectly elastic collision: kinetic energy is conserved.

Impact

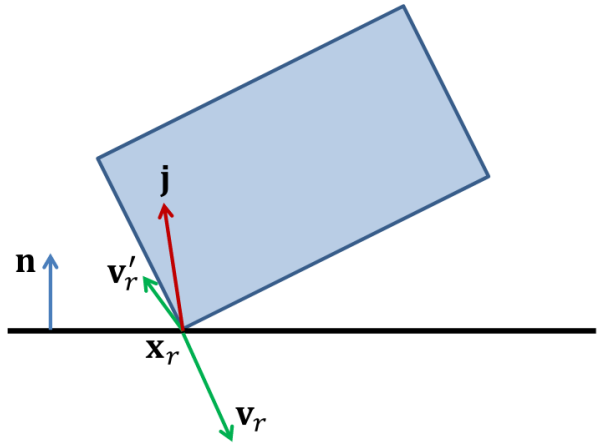
To simulate an impact, we first consider perfectly inelastic collision without sliding. Using conservation of (angular) momentum, it is very easy to compute a $\boldsymbol{j}'$. However, If $\boldsymbol{j}'$ does not lie in the friction cone, the impact has sliding, and we project $\boldsymbol{j}'$ back into the friction cone.

Next, we use the fact that the impulse of perfect elastic collision is twice the impulse of perfect inelastic collision. So the impulse will be between $\boldsymbol{j}'$ and $2\boldsymbol{j}'$. We use restitution coefficient ϵ to describe the elasticity.

$$\boldsymbol{j} = (1 + \epsilon)\boldsymbol{j}'$$

Note: different people may define restitution coefficient differently.

Impact



Specifically, the impulse in a non-sliding impact with a static surface is

$$\boldsymbol{j} = -(1+\epsilon)\boldsymbol{M}_{rr} \cdot \boldsymbol{v}_r$$

Topics

- Rigidbody Constraints
  - Constraint solver for joints
  - Contact, friction, and impact
- **Numerical Integration**
- Collision Detection
  - Primitive shape collision
  - Broad and narrow phases
  - Convex shape collision

From Physics to Simulation

Newton-Euler allows us to compute acceleration

$$\boldsymbol{f} = m\boldsymbol{a} \quad \text{(Linear motion)}$$
$$\boldsymbol{\tau} = \boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega} + \boldsymbol{I}\dot{\boldsymbol{\omega}} \quad \text{(angular motion)}$$

How do we get position and velocity from acceleration? Through integration.

$$\boldsymbol{x} = \int \dot{\boldsymbol{x}} dt, \dot{\boldsymbol{x}} = \int \ddot{\boldsymbol{x}} dt$$

However, in reality, integration has to be computed numerically.

Rigidbody Simulation: integration

Forward Euler: $\boldsymbol{v}_{t+1} \leftarrow \boldsymbol{v}_t + \boldsymbol{a}\Delta t, \boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t + \boldsymbol{v}_t\Delta t$
Observe: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{v}_t\Delta t$

Semi-implicit Euler: $\boldsymbol{v}_{t+1} \leftarrow \boldsymbol{v}_t + \boldsymbol{a}\Delta t, \boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t + \boldsymbol{v}_{t+1}\Delta t$
Observe: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{v}_t\Delta t + \boldsymbol{a}\Delta t^2$

However, we know $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{v}_t\Delta t + \frac{1}{2}\boldsymbol{a}\Delta t^2$. How do we fix it?

Rigidbody Simulation: integration

Runge-Kutta method (RK4) https://en.wikipedia.org/wiki/Runge-Kutta_methods

$$\boldsymbol{k}_1 = \boldsymbol{v}_t$$

$$\boldsymbol{k}_2 = \boldsymbol{k}_3 = \boldsymbol{v}_t + \boldsymbol{a}\frac{1}{2}\Delta t$$

$$\boldsymbol{k}_4 = \boldsymbol{k}_3 = \boldsymbol{v}_t + \boldsymbol{a}\Delta t$$

$$\boldsymbol{x}_{t+1} = \frac{1}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4)\Delta t = \boldsymbol{x}_t + \boldsymbol{v}_t\Delta t + \frac{1}{2}\boldsymbol{a}\Delta t^2$$

RK4 is accurate for constant acceleration. However, most game engines uses semi-implicit Euler. Why?

- 4 times faster.
- Semi-implicit Euler preserves energy better in a lot of problems (such as orbit motion, spring-damping)

Takeaway

- One major problem that physical simulation solves is the numerical integration given by Newton-Euler.
- If simulation results do not match your expectation by a small margin, check integration technique used.

Topics

- Rigidbody Constraints
  - Constraint solver for joints
  - Contact, friction, and impact
- Numerical Integration
- **Collision Detection**
  - Primitive shape collision
  - Broad and narrow phases
  - Convex shape collision

Collision Detection

- Collision detection is the geometric problem of finding object intersection.
- It is required for contact simulation.
- Contact simulation only depends on contact point and normal, so collision detection module can be totally separated from contact solver.

Collision Primitives

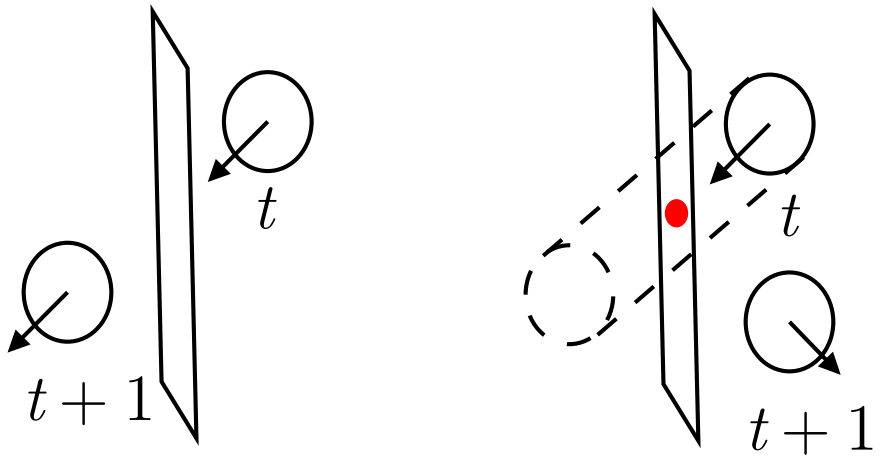
There are various primitives that we may want collision test:

- Point
- Line
- Triangle
- Sphere
- Cylinder
- Capsule
- Axis-aligned box
- Box
- Convex mesh
- Height map
- Triangle mesh

For the chosen primitives, collision detection procedures for each pair are required. So there will be $n(n-1)/2$ different procedures.

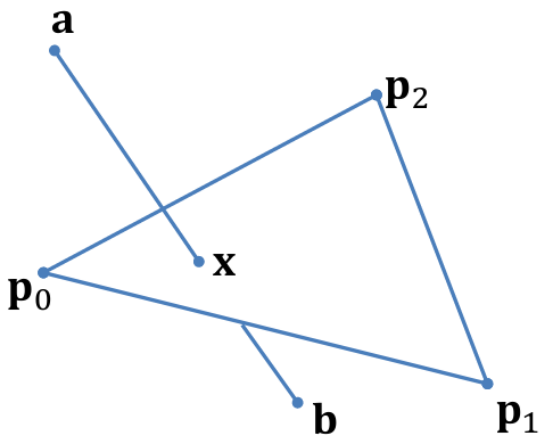
Discrete and Continuous Collision

Discrete collision detection only test intersection at time points. High speed objects may penetrate thin walls without collision. (left)
Continuous collision detection build sweeping volume of an object relative to the other object. It is more computationally expensive.



Basic Intersection Testing

Line-triangle intersection (ray-tracing)

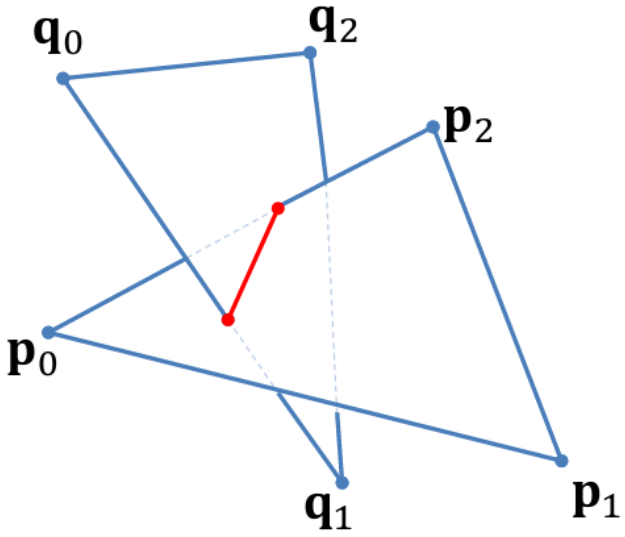


Let $\boldsymbol{v}_1 = (p_0p_1), \boldsymbol{v}_2 = (p_0p_2)$, any point in the triangle can be expressed as $\boldsymbol{p}_0 + t_1\boldsymbol{v}_1 + t_2\boldsymbol{v}_2, t_1, t_2, t_1 + t_2 \in [0, 1]$. Let $\boldsymbol{d} = (ab)$ Any point on the line can be expressed as $\boldsymbol{a} + t\boldsymbol{d}$. We only need to solve $\boldsymbol{p}_0 + t_1\boldsymbol{v}_1 + t_2\boldsymbol{v}_2 = \boldsymbol{a} + t\boldsymbol{d}$ for t_1, t_2 and check $t_1, t_2, t_1 + t_2$.

Basic Intersection Testing

Triangle-triangle intersection

- If all vertices of one triangle is on the same side of the other triangle plane, we are done.
- If not, we use the previous segment-triangle intersection test and try to find the intersection segment as shown in the picture.



Basic Intersection Testing

Sphere-sphere intersection

- Center separation is greater than sum of radii.

$$\|c_1 - c_2\|^2 \leqslant (r_1 + r_2)^2$$

Line-sphere intersection

- Find the point on the line that is closest to the sphere center.
- Test distance and radius.

Basic Intersection Testing

Triangle-sphere intersection

- Test distance of the point on the triangle plain that is closest to sphere center, test if it is in the triangle.
- Test whether the sphere hits any edge of the triangle.
- Test whether any of the triangle vertices is inside the sphere.

Collision Phases

It would be too expensive to check collision between each pair of objects. So collision checking is divided into phases and data structures are used to accelerate checking.

Broad Phase: high-level coarse collision checking to reduce number of pairs we need to check.

Narrow Phase: geometric checking between individual shapes.

Sometimes, people also mention a mid-level phase similar to the broad phase.

Broad Phase

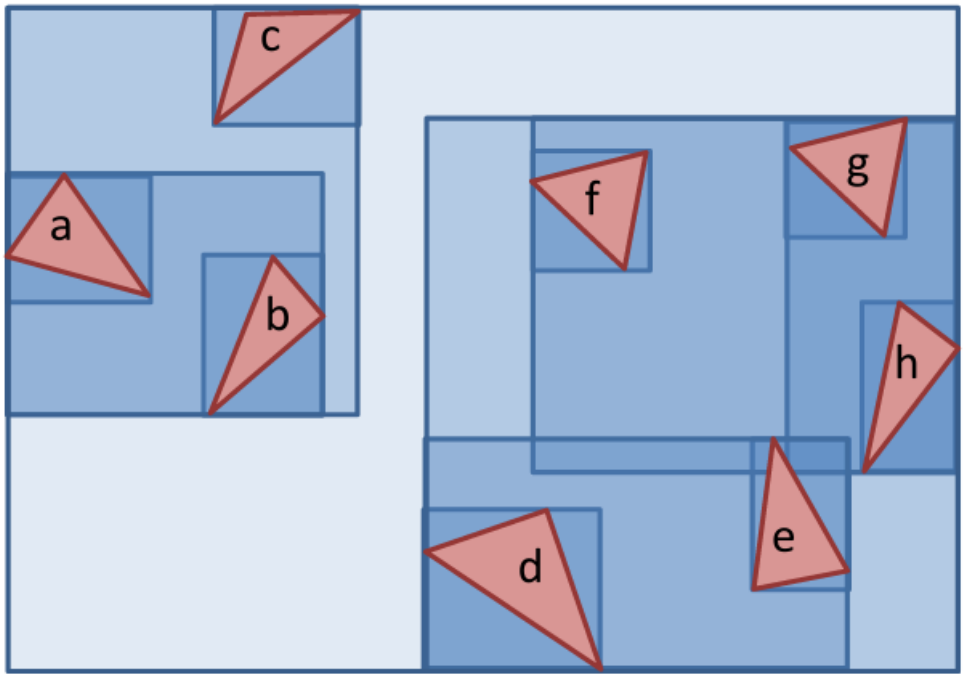
- Spatial data structures
  - Bounding volume hierarchies
  - Spatial partitions
- Sweep and prune algorithm

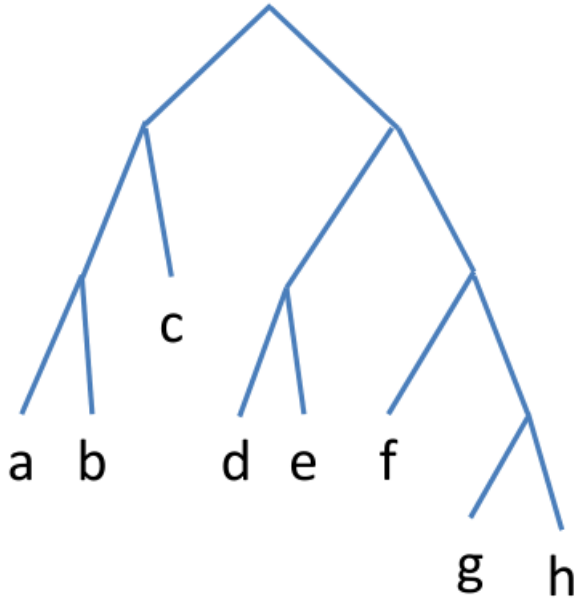
Bounding Volume Hierarchy

Bounding Volume Hierarchies (BVH) are trees that groups bounding volumes. Common data structures include

- Sphere tree
- AABB tree
- OBB tree
- k-DOP tree

AABB Tree



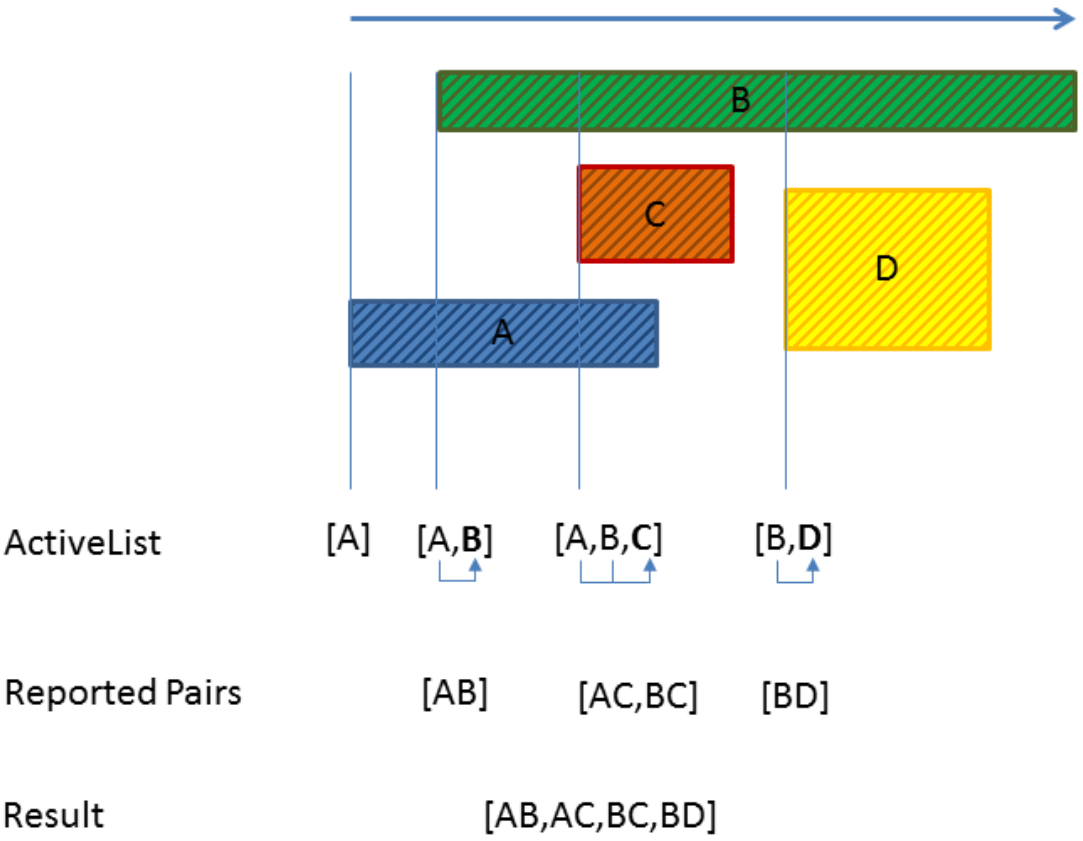


Spatial Partitions

Instead of dividing the bounding volumes, spatial partitions divide the space. Common data structures include

- Octree
- KD-tree
- BSP-tree
- Uniform grid
- Spatial hash table

Sweep and Prune



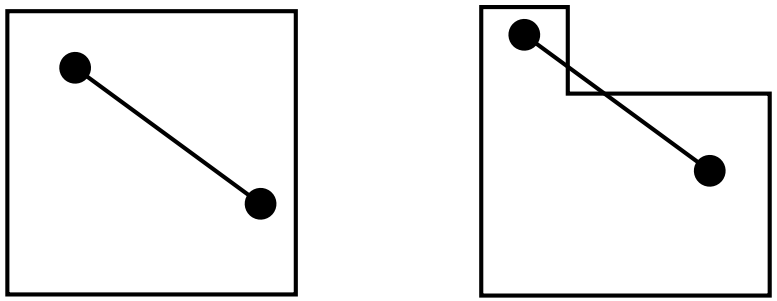
https://github.com/mattleibow/jitterphysics/wiki/Sweep-and-Prune

Narrow Phase

- Convex and concave shapes
- Separating axis theorem
- Separation/penetration distance
- Convex hull algorithms
- Convex decomposition

Convex and Concave

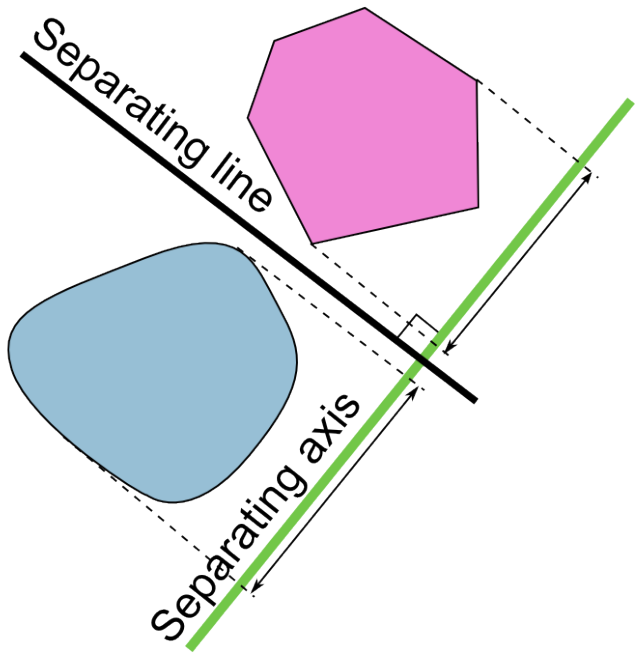
In a convex shape, a line segment between any two internal points will be entirely inside the shape.
We have efficient distance and intersection testing algorithms for convex shapes, so convex shapes are preferred in most simulation.



Convex Concave

Separating Axis Theorem

- Also called hyperplane separation theorem.
- two convex bodies do not intersect iff there is a hyperplane in between them.
- An axis which is orthogonal to a separating hyperplane is a **separating axis**, because the orthogonal projections of the convex bodies onto the axis are disjoint



Separating Axis Theorem

- For convex polyhedra, we only need to test finite number of axis.
- The algorithm typically runs in $O(e_1 e_2)$ time, but there may be improvements for specific cases.
- For example, we need 15 axis tests for oriented boxes. (much better than $e_1 e_2$)

Separation Distance

the Gilbert-Johnson-Keerthi (GJK) algorithm computes distance between arbitrary convex objects. It can also be modified to compute penetration distance.
Given two convex shapes A, B, their Minkowski difference, defined as

$$A \ominus B = \{\boldsymbol{a} - \boldsymbol{b} : \boldsymbol{a} \in A, \boldsymbol{b} \in B\},$$

is also convex.
GJK algorithm solves the convex optimization problem of

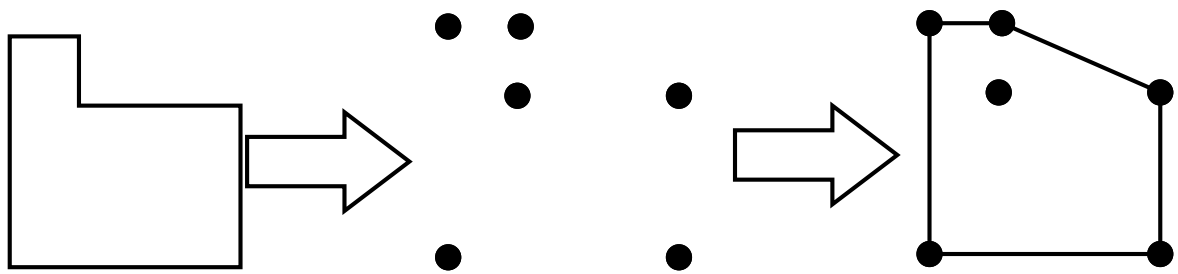
$$\left\{ \|\boldsymbol{a} - \boldsymbol{b}\|_2^2 : (\boldsymbol{a} - \boldsymbol{b}) \in A \ominus B \right\}$$

with standard optimization technique.

Convex Hull Algorithms

Not all shapes come as convex, so we need to first compute a convex approximation: finding the smallest convex shape that contains the given shape. The containing shape is known as a **Convex Hull**.

In 2D, we have the simple gift-wrapping algorithm.

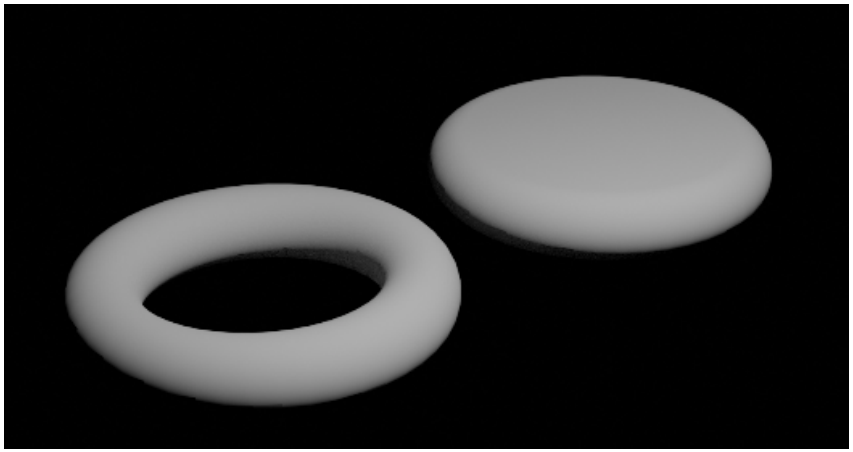


Gift wrapping algorithm

In 3D, we have quickhull algorithm. It runs in $O(n \log n)$.

Convex Decomposition

Sometimes, we cannot sacrifice certain concavity. Flat donuts are unacceptable!



HACD(left) and v-HACD(right).

Resources

This lecture is adapted from the following sources:

A fantastic tutorial on the topic
https://www.toptal.com/game/video-game-physics-part-i-an-introduction-to-rigid-body-dynamics

Professor Steve Rotenberg's course
https://cseweb.ucsd.edu/classes/sp19/cse291-d/