Manipulation with Friction

15-494 Cognitive Robotics
David S. Touretzky &
Ethan Tira-Thompson

Carnegie Mellon
Spring 2007

Introduction

1. Friction

2. Jacobians

3. Dynamics

4. Control (P, PD, PID)

Friction: Coulomb's Law

Static friction force balances pulling force, up to maximum specified by static friction coefficient

Once object begins moving, frictional force drops to constant value, called sliding friction or kinetic friction
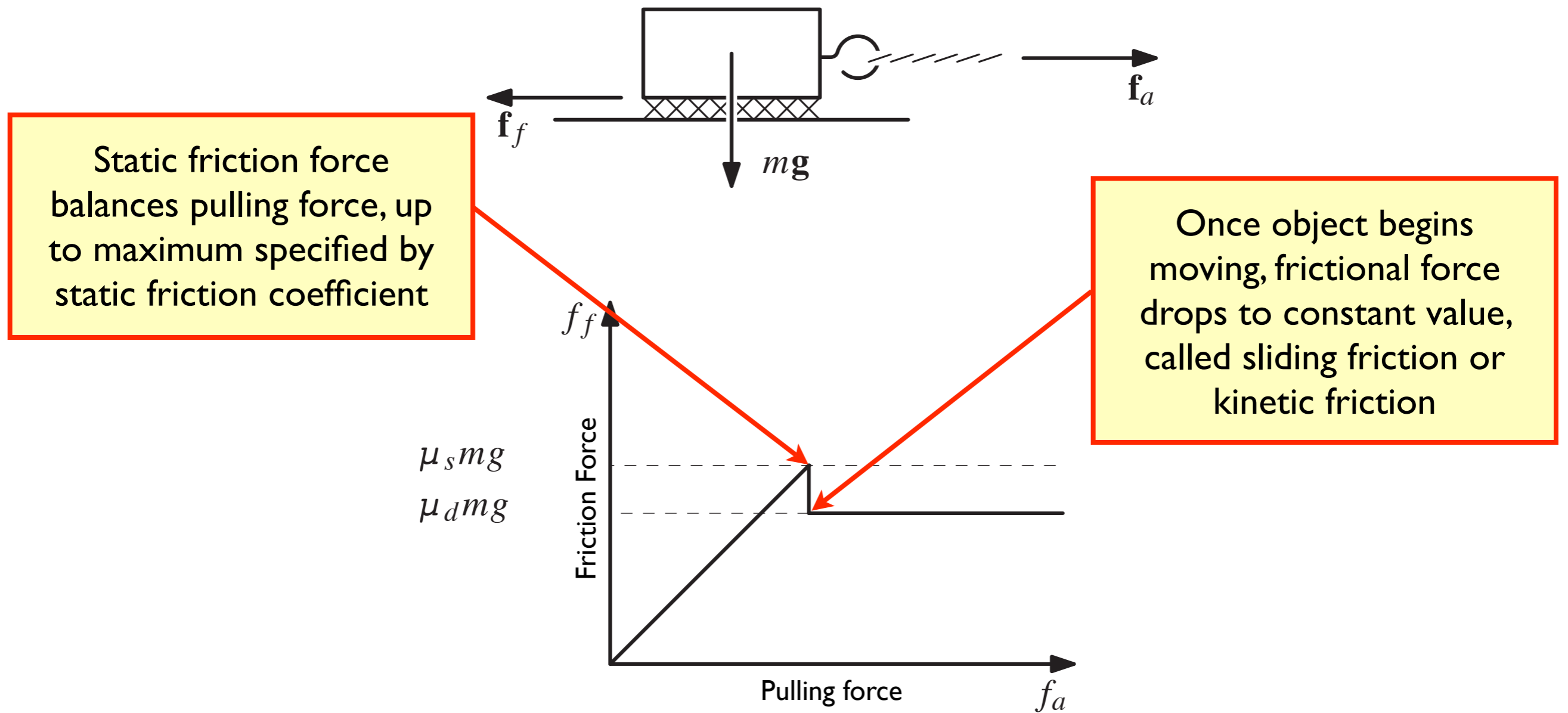
Figure 6.1 - Mason, Mechanics Of Robotic Manipulation

Friction: Coulomb's Law

- For common tasks, independent of velocity and surface area

  - With extreme pressures, coefficient rises

  - With extreme velocities, coefficient drops

- Coefficients of friction are different for every pair of surfaces — table lookup

  - also differ for every change in temperature, humidity, dust/dirt, vibration, celestial alignment, etc. — not terribly accurate

Friction within Joints

- Static friction is a headache for fine motor control

    - motor has to ramp up power to overcome static friction within gears, but as soon as it succeeds in doing so, it's now providing too much power and will "jump" to life.

    - this is the fundamental reason you see the Aibo's joints twitch from time to time

    - the higher the gear ratio, the bigger the problem

Computing with Forces

- Forces are defined by a line through space, and a magnitude

  - usually represented by a vector and a point

  - but the point is not unique — any point along the vector is equally valid ("line of action")
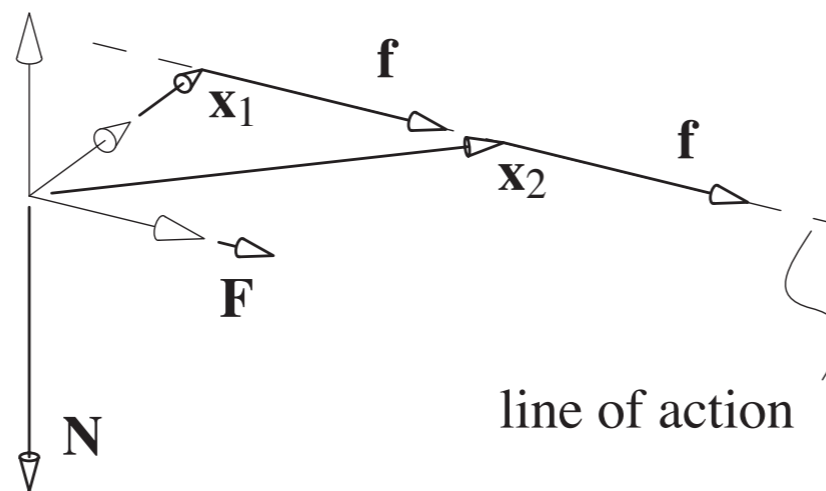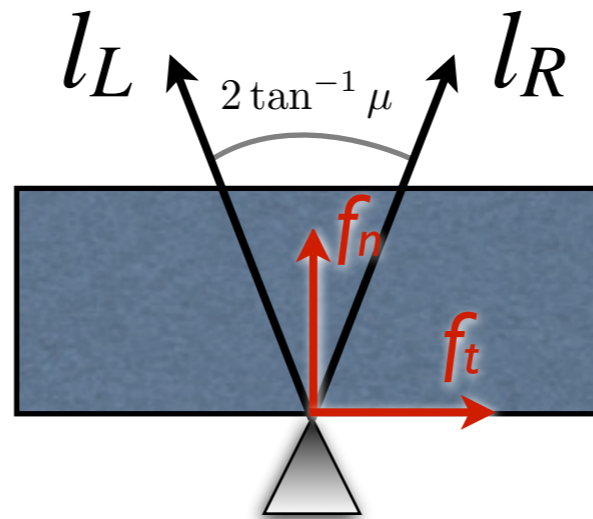

Figure 5.1 - Mason,
*Mechanics Of Robotic Manipulation*

Friction with Objects

- Now we can define a friction cone:

$$l_L \quad {\scriptstyle 2\tan^{-1}\mu} \quad l_R$$

- Edges of the cone define maximum angle allowed for forces without slippage

- If you break applied force into normal force f_n and tangental force f_t, friction cone is defined as $|f_t| \leq \mu|f_n|$, with interior angle $2 \tan^{-1}\mu$
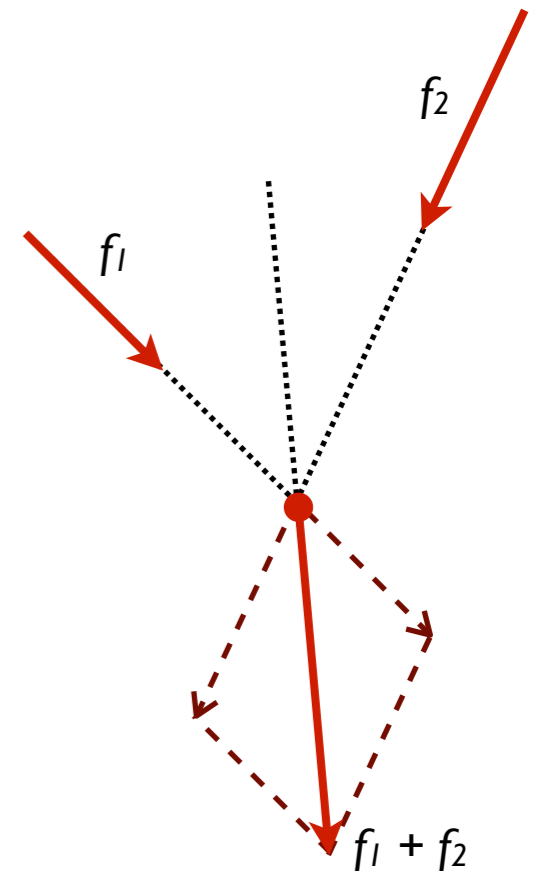
Friction with Objects

- Remember Reuleaux's Method?
  - Works with friction cones as well
- Now we're analyzing forces, not displacements, *a different interpretation!* (be careful about trying to mix them...)
- Only forces which agree with the all of the contacts' constraints can be applied by the contact(s):
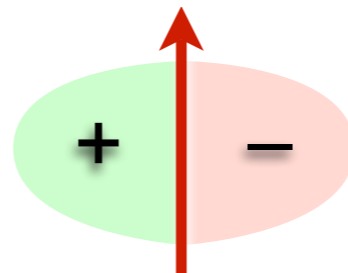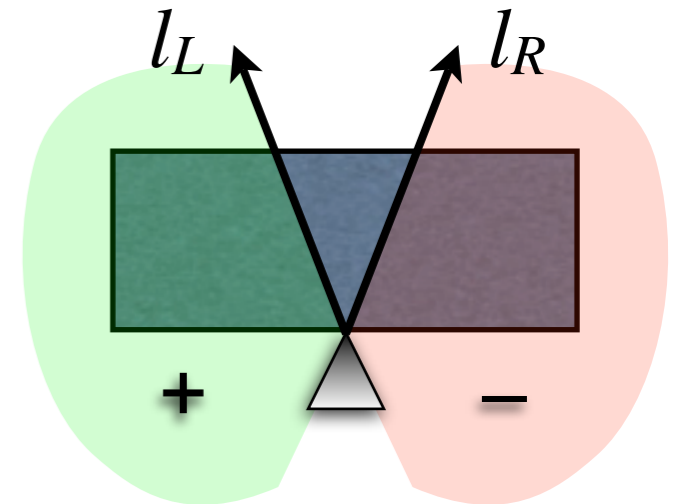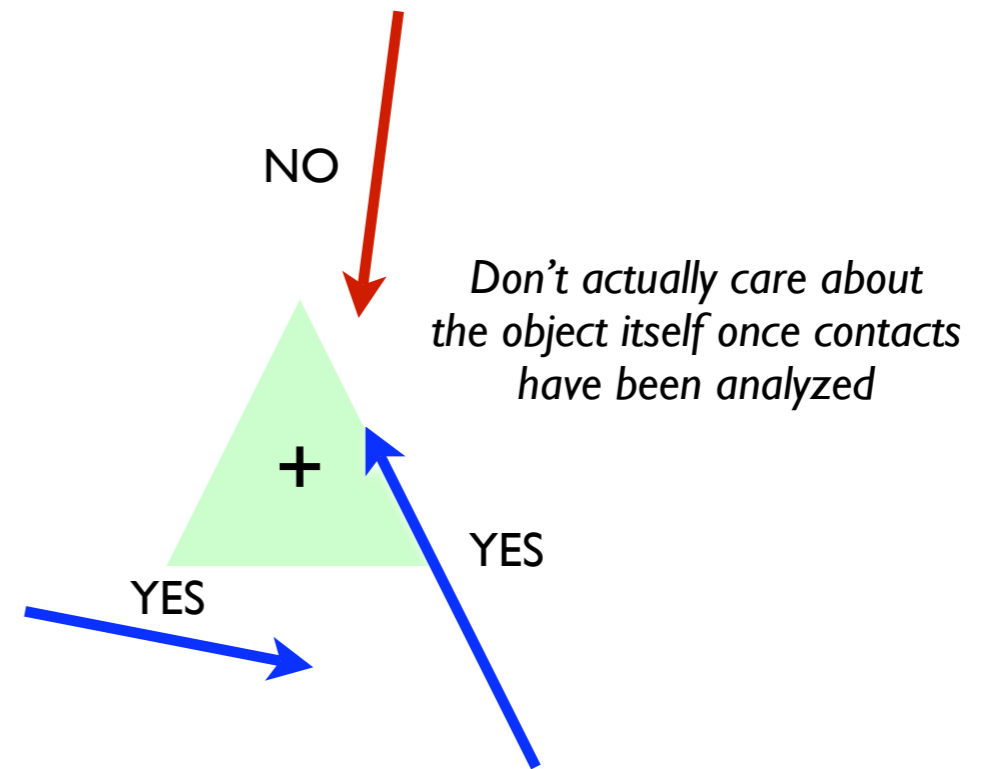
Combining Forces

- Adding multiple contacts allows you to apply any force in the linear span of their friction cones

- Remember that forces act along a line through space

  - slide forces along line of action to intersection

  - Resultant force is the vector sum of the two forces, acting through common intersection

f_2

f_1

$f_1 + f_2$

Friction with Objects: Examples

Don't actually care about the object itself once contacts have been analyzed
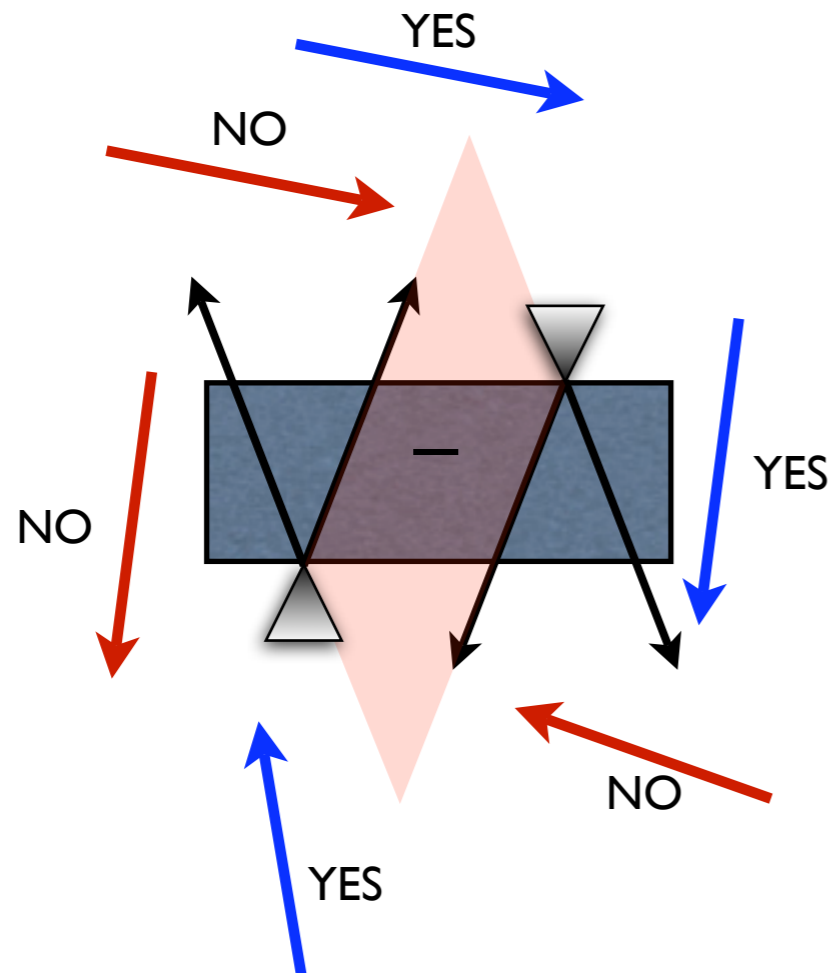
For reference:
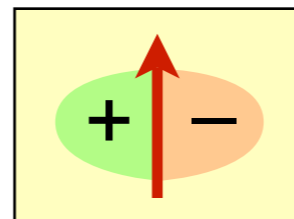
Center of Friction

- Similar to center of mass, center of friction is the integrated pressure over the support region

- Allows you to treat the interaction as a single contact

- Hard to model — with a rigid body, small variances completely throw off pressure distribution

  - Ever play Jenga?

Applying Friction & Forces

- Use weight to flip brick

- Use wall to direct ball (extra arm)

- Get ball away from wall

- Use wall to align/direct brick

- Stand bone upright

- Insert objects without jamming or wedging

The Jacobian Matrix

- One of the most important tools in analyzing and controlling robot motion!

- Provides the instantaneous velocity in each of the 6 freedoms (translation and orientation along/around each of *x*, *y*, and *z*) as a function of each of the robot's links

$$\mathbf{J}(q)\dot{q} = \dot{x}$$

$\mathbf{J}(q)$ = Jacobian (6×*n*) — a function of current joint positions (*q*)

$\dot{q}$ = joint velocity vector (length *n*)

$\dot{x}$ = workspace velocity vector (length 6)

The Jacobian Matrix: Usage

- Find current workspace velocity/force

- Determine contribution of individual joints

- Analyze rank to detect singularities (for better or worse)

  - a singularity occurs when joints become aligned, causing a loss in effector mobility (but increased strength along that axis!)

  - under-actuated robots always have incomplete rank
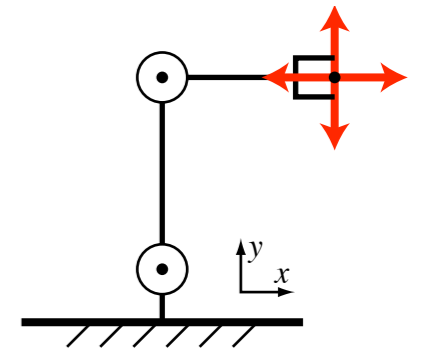
Full (planar) mobility
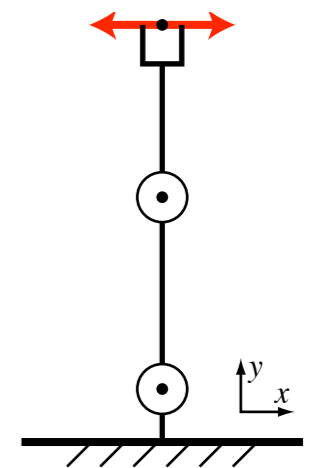
*Singularity:
cannot move along y axis, but
also don't have to do any
work to resist forces along it*

The Jacobian Matrix: Usage

- Things to watch out for at/near singularities:

  - Small workspace movements/forces may require instantaneous joint motion (infinite motor torque!)

  - Usually occur at workspace limits

  - May have infinite inverse kinematic solutions

- Test for configuration "quality":

M(q) *becomes zero at singularities*	$M(q) = \sqrt{\det(\mathbf{J}(q)\mathbf{J}^{\mathsf{T}}(q))}$	*Swap $\mathbf{J}$(q) and $\mathbf{J}^{\mathsf{T}}$(q) if under-actuated (i.e. $\mathbf{J}$(q) is less than full rank)*

The Jacobian Matrix: Composition

$$\mathbf{J}(q)\dot{q} = \dot{x}$$

Jacobian is split into two components:

$\mathbf{J}_p(q)$ = Position component (sometimes $\mathbf{J}_v(q)$)

$\mathbf{J}_o(q)$ = Orientation component

$$\begin{bmatrix} \mathbf{J}_p(q) \\ \mathbf{J}_o(q) \end{bmatrix} \dot{q} = \begin{bmatrix} \dot{p} \\ \omega \end{bmatrix}$$

$\dot{p}$ = Linear velocity vector of end effector

ω = Angular velocity vector of end effector

The Jacobian Matrix: Position Component

$$\mathbf{J}_p(q) = \begin{bmatrix} \dfrac{\partial p_x}{\partial q_1} & \dfrac{\partial p_x}{\partial q_2} & \cdots & \dfrac{\partial p_x}{\partial q_n} \\[2em] \dfrac{\partial p_y}{\partial q_1} & \dfrac{\partial p_y}{\partial q_2} & \cdots & \dfrac{\partial p_y}{\partial q_n} \\[2em] \dfrac{\partial p_z}{\partial q_1} & \dfrac{\partial p_z}{\partial q_2} & \cdots & \dfrac{\partial p_z}{\partial q_n} \end{bmatrix}$$

$\mathbf{J}_p(q)_1 \qquad \mathbf{J}_p(q)_2 \qquad \cdots \qquad \mathbf{J}_p(q)_n$

Remember that a joint's z axis is always defined to point along its axis of motion

If Joint *i* is prismatic: $\mathbf{J}_p(q)_i = z_i$

If Joint *i* is revolute: $\mathbf{J}_p(q)_i = z_i \times (p - p_i)$

Where:

z_i = z axis of joint *i* p_i = position of joint *i*'s origin

p = position of the end effector   (all relative to base frame)
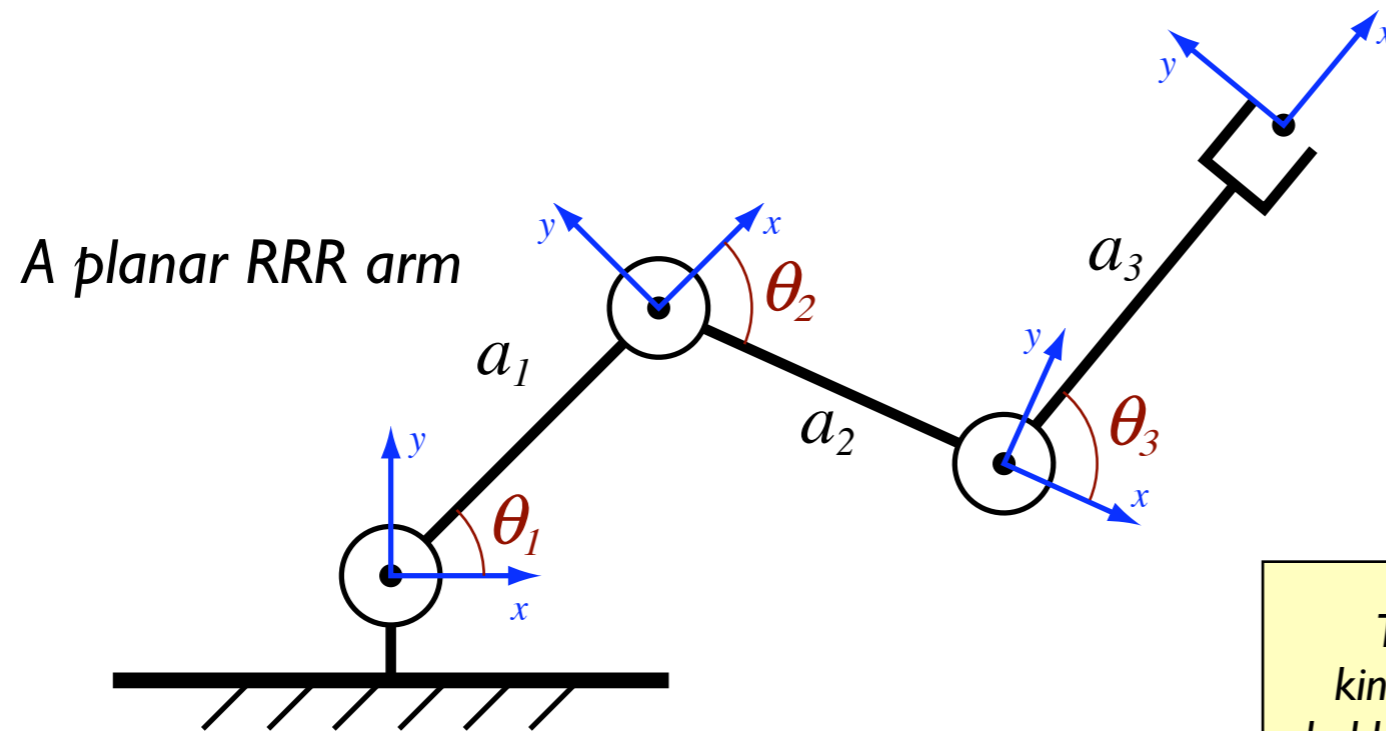
The Jacobian Matrix: Orientation Component

$$\mathbf{J}_o(q) = \begin{bmatrix} \dfrac{\partial \omega_x}{\partial q_1} & \dfrac{\partial \omega_x}{\partial q_2} & \cdots & \dfrac{\partial \omega_x}{\partial q_n} \\[2em] \dfrac{\partial \omega_y}{\partial q_1} & \dfrac{\partial \omega_y}{\partial q_2} & \cdots & \dfrac{\partial \omega_y}{\partial q_n} \\[2em] \dfrac{\partial \omega_z}{\partial q_1} & \dfrac{\partial \omega_z}{\partial q_2} & \cdots & \dfrac{\partial \omega_z}{\partial q_n} \end{bmatrix}$$

$$\mathbf{J}_o(q)_1 \qquad \mathbf{J}_o(q)_2 \qquad \cdots \qquad \mathbf{J}_o(q)_n$$

If Joint i is prismatic: $\mathbf{J}_o(q)_i = 0$

If Joint i is revolute: $\mathbf{J}_o(q)_i = z_i$

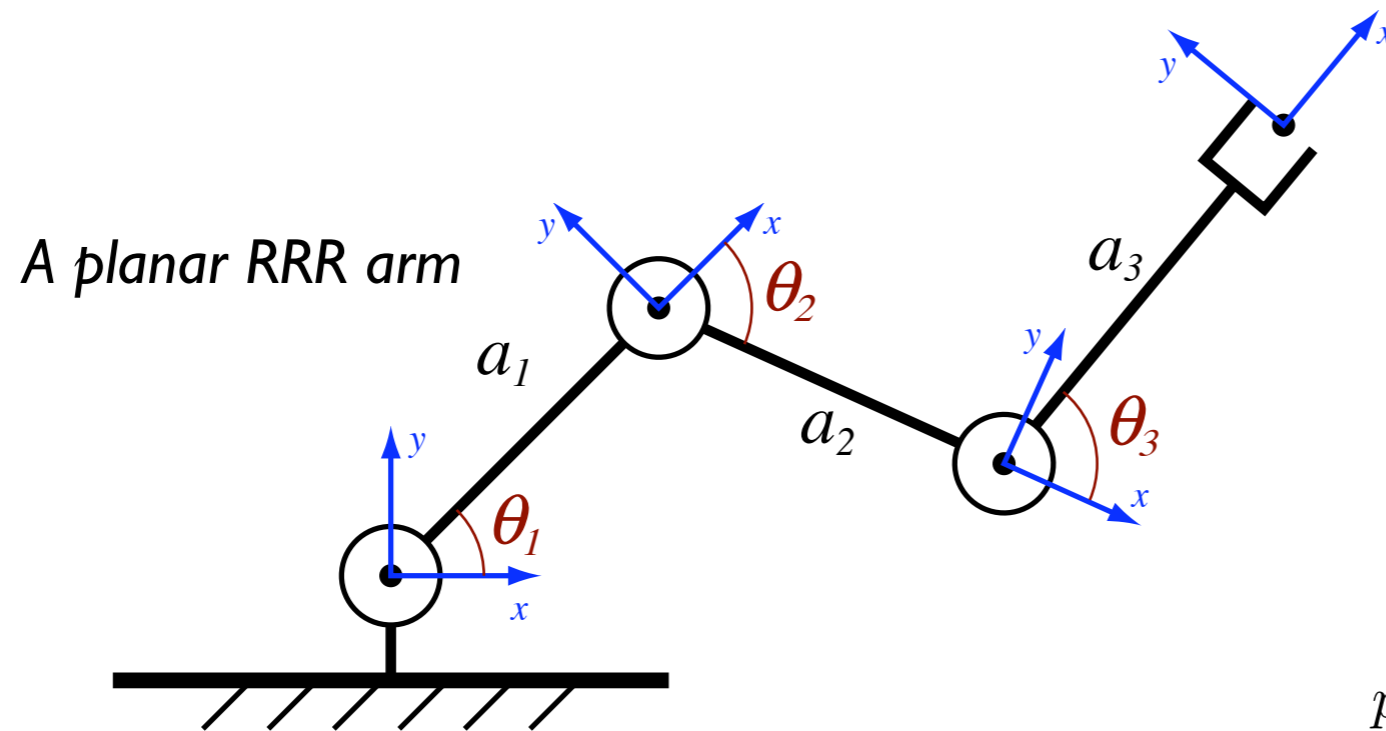# The Jacobian Matrix: Example

A planar RRR arm

These are all given to you by the forward kinematics: each joint's transformation matrix holds the current z vector in the 3rd column and the current position in the 4th column

$$\mathbf{J}(q) = \begin{bmatrix} z_0 \times (p_e - p_0) & z_1 \times (p_e - p_1) & z_2 \times (p_e - p_2) \\ z_0 & z_1 & z_2 \end{bmatrix}$$

15-494 Cognitive Robotics

The Jacobian Matrix: Example

Notation:
$s_1 = \sin(\theta_1)$
$c_{123} = \cos(\theta_1 + \theta_2 + \theta_3)$

$$p_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$p_1 = \begin{bmatrix} a_1 c_1 \\ a_1 s_1 \\ 0 \end{bmatrix}$$

A planar RRR arm

$$p_2 = \begin{bmatrix} a_1 c_1 + a_2 c_{12} \\ a_1 s_1 + a_2 s_{12} \\ 0 \end{bmatrix}$$
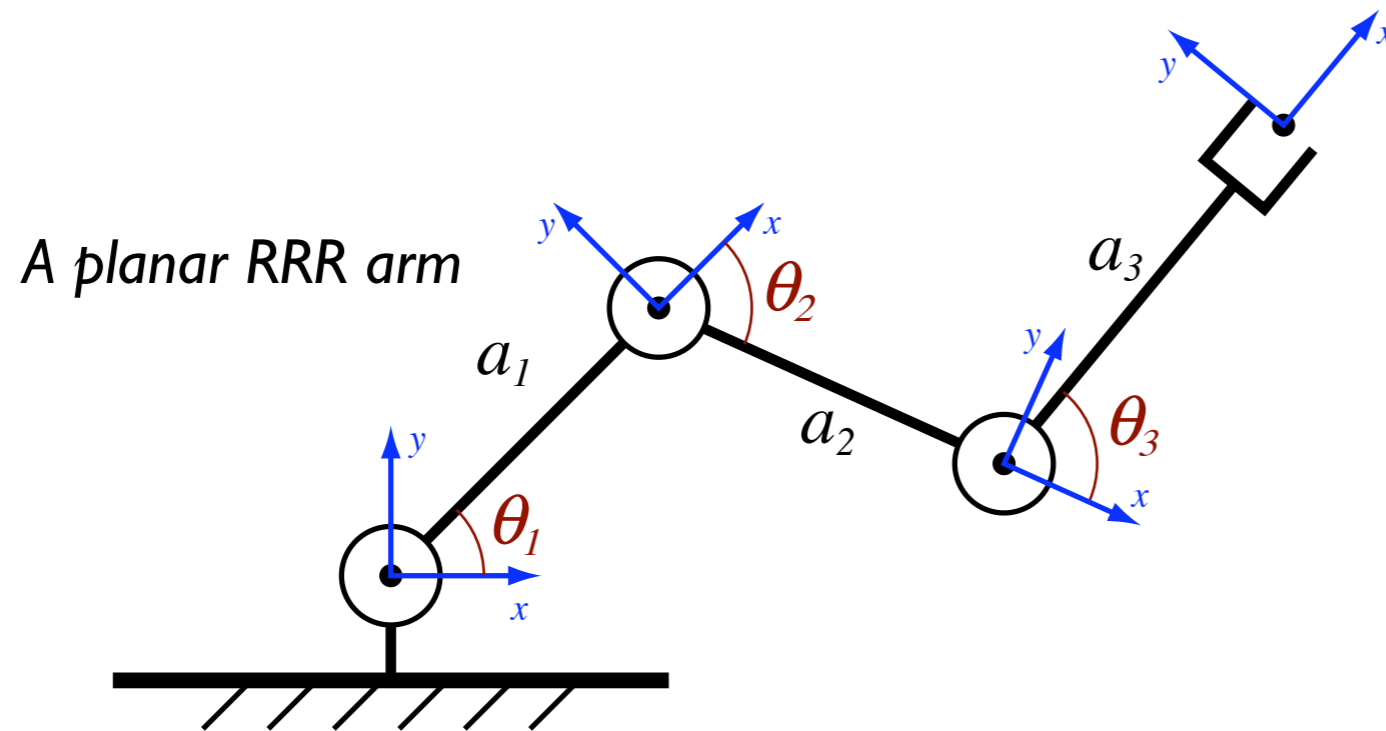
$$p_e = \begin{bmatrix} a_1 c_1 + a_2 c_{12} + a_3 c_{123} \\ a_1 s_1 + a_2 s_{12} + a_3 s_{123} \\ 0 \end{bmatrix}$$

$$z_0 = z_1 = z_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\mathbf{J}(q) = \begin{bmatrix} z_0 \times (p_e - p_0) & z_1 \times (p_e - p_1) & z_2 \times (p_e - p_2) \\ z_0 & z_1 & z_2 \end{bmatrix}$$

The Jacobian Matrix: Example

A planar RRR arm

$$\mathbf{J}(q) = \begin{bmatrix} -a_1s_1 - a_2s_{12} - a_3s_{123} & -a_2s_{12} - a_3s_{123} & -a_3s_{123} \\ a_1c_1 + a_2c_{12} + a_3c_{123} & a_2c_{12} + a_3c_{123} & a_3c_{123} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Dynamics

- How will joints move as power is applied?
  - Ideally, the robot manufacturer tells you:
    - Inertia Tensor ($\mathbf{I}$, $\mathbf{3} \times \mathbf{3}$ matrix) for each link: angular momentum can then be found: $L = \mathbf{I}\omega$
    - Motor properties for each joint: rotor inertia (I_m), gear ratio, viscous and coulomb friction
  - Sony isn't ideal – we don't have these parameters
  - Aibo doesn't give direct control over torque anyway (we specify position, it computes power)
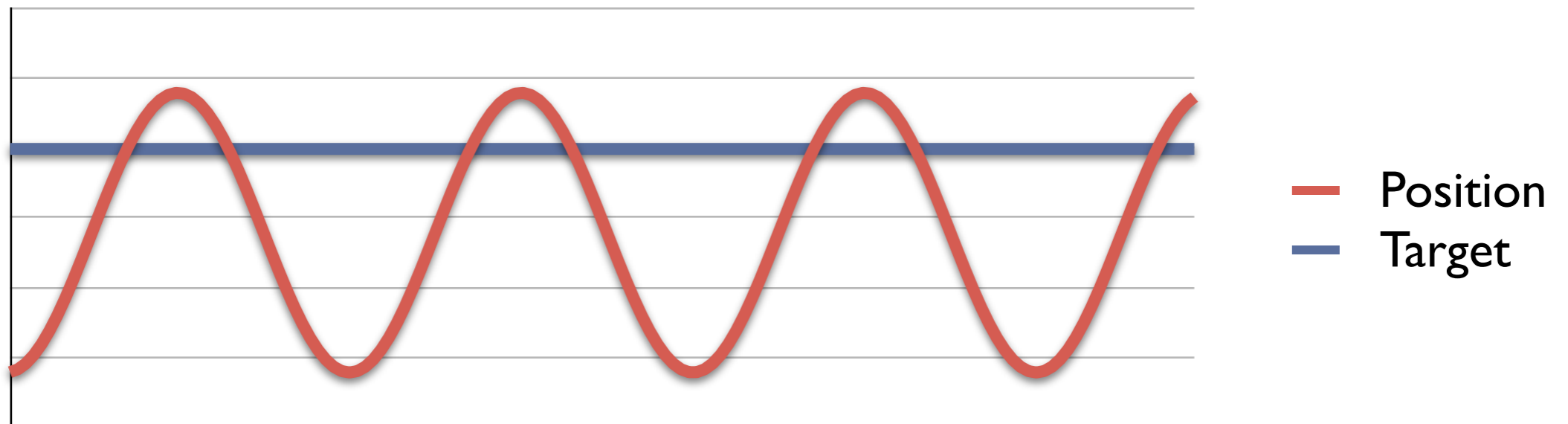
Control

- So then, how does it compute the power for each joint?

- We want to move the joint to a specified position, and hold it there

  - Sounds easy, right? Harder than it sounds:

    - there may be other forces acting on the joint (e.g. gravity, inertia, etc.)

    - you're controlling acceleration, two derivatives away from position — go fast, but don't oscillate

Proportional Control

- Here's an idea:

  - take the current position error ($e(t) = x(t) - x_{tgt}$)

  - multiply $e(t)$ by some parameter k_p

  - use this value as the new power output
  $$output = -k_p \cdot e(t)$$

- Should work, right? Farther away means more power. As we get closer, reduce power.

Proportional Control

- Here's the resulting graph of position over time:

Position
Target

- Whoa, look at that oscillation, and it isn't even oscillating around the right value!

  - One thing at a time buckaroo – oscillation first

PD Control

- The oscillation is caused because there's nothing to cause it to slow down as it's approaching the target — inertia will keep the link moving and blow right past the target

  - if you have significant friction or little inertia (with a speed limit), this may not be a problem

- Add a braking factor k_d, multiplied by the current error derivative $\partial e(t)$

$$output = -(k_p \cdot e(t)) - (k_d \cdot \partial e(t))$$

PD Control

- Here's a new graph of position over time:

Position
Target

- Closer! Now, let's take care of that offset.

PID Control

- That offset is caused by external forces, like gravity. We need another term to handle its constant input to our system.

- Use an integral of the error term, and multiply it by a new coefficient k_i:
  $$output = -(k_p \cdot e(t)) - (k_i \cdot \textstyle\int e(t)dt) - (k_d \cdot \partial e(t))$$

- Actual implementations vary in the parameterization, many use:
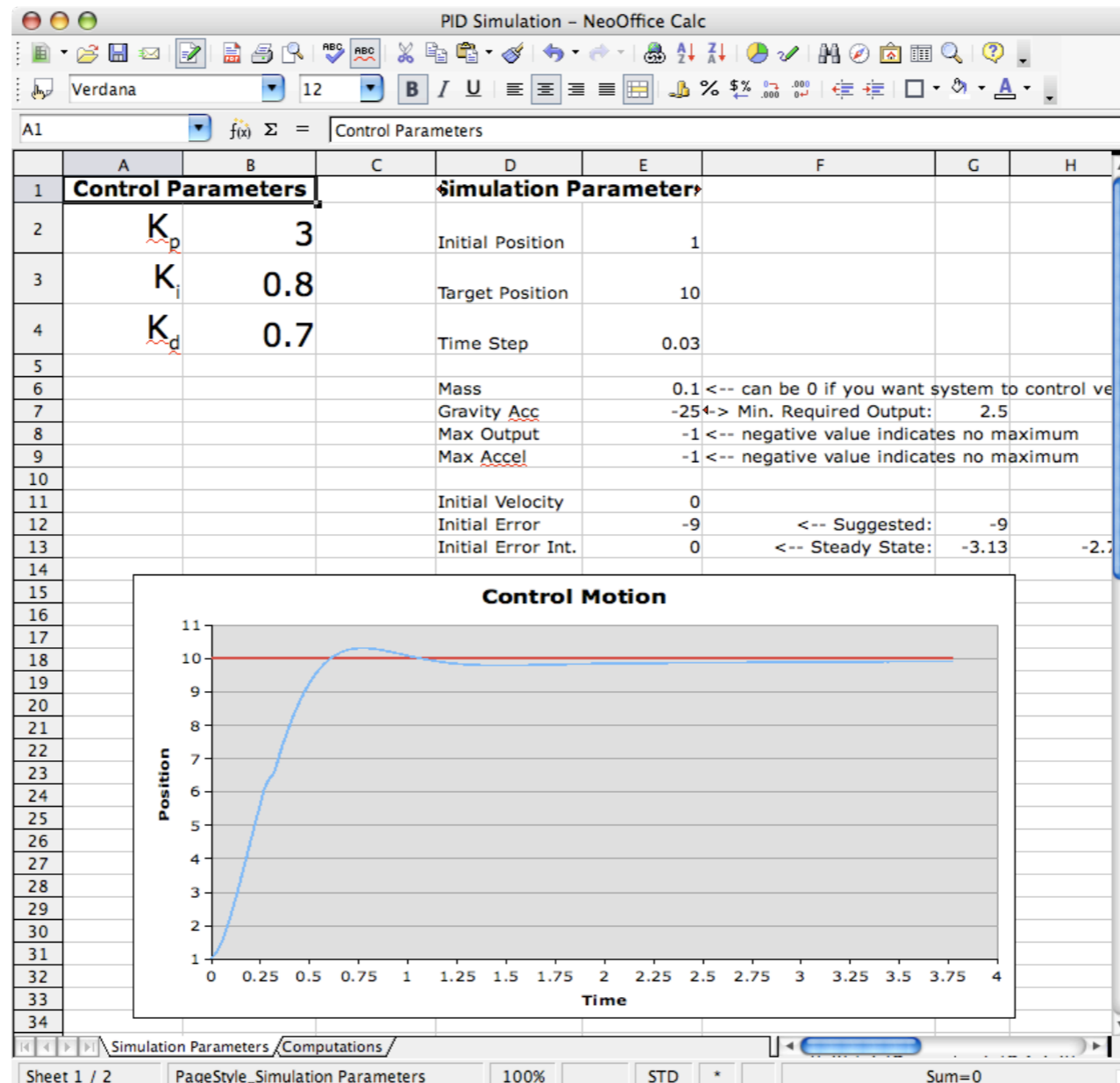 $$output = -k_p \cdot (e(t) + (k_i \cdot \textstyle\int e(t)dt) + (k_d \cdot \partial e(t)))$$

PID Control

- Now look at the graph:

	Position
	Target

- Ta-da!

PID Control

- We've put an Excel spreadsheet for this simulation online so you can play with it

Qualifications

- The graphs shown previously were based on a system with inertia

- If the system you are controlling does not have inertia, or equivalently, you are <u>controlling velocity</u> directly (not force), proportional control may be all you need!

  - Proportional control often used as a potential field function for steering mobile robots...

Downside of the I Parameter

- If grasping an object with several manipulators, any error in the manipulator's position will cause gradually increasing internal strain

- This is why the Aibo will sometimes shutdown with a joint overload error, simply from standing idly on the ground

The Dirty Little Secret

- How do we pick the *P*, *I*, and *D* parameters?

  - Hard way: lots of math (a lecture unto itself)

    - Read up on: Laplace Transforms, characteristic equations, pole placement, bode plots

  - Easy way: play with them until you get something you like

    - be careful not to make big changes at a time — don't want to get into unstable feedback loops

  - Smart way: adaptive self-tuning

Intuitive PID Tuning Advice

- In our notation (which I *believe* the AIBO uses)

    - Scaling all the parameters together will scale maximum power output without changing control style (very much) — in the alternative formulation, only *P* can be scaled this way.

    - *P* tends to have the biggest impact — higher P means more power , but more oscillation

    - *D* balances oscillation, but reduces top speed

    - *I* balances final errors (remember joint twitching?)

Pulse-Width Modulation (PWM)

- Finally, one last trick: servos are not controlled with analog power levels

- Instead, power is "pulsed" on and off at high frequency

- The portion of the period during which the power is turned on is called the duty cycle

- Generally, this is a transparent effect, but knowing this allows you to interpret the "duty cycle" feedback given by each joint

Getting Power From Position

- So, now that we have some understanding of how power is computed from desired position, we should be able to invert it to compute a target position which will result in a desired force!

- Make life easy for yourself: set k_i and k_d to 0, and specify an offset from current joint position as the target — force will be directly proportional to your offset (and k_p)