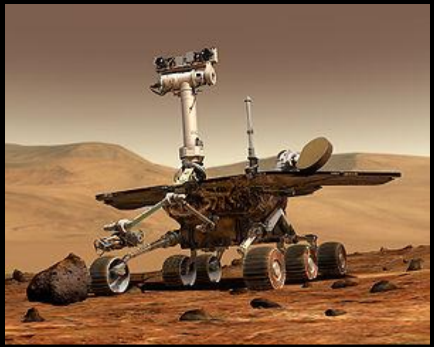
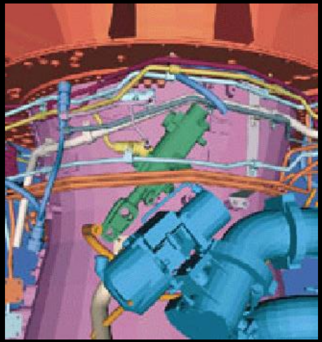
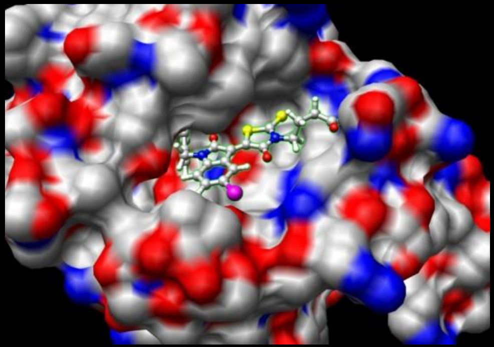
Inverse Kinematics for Articulated Robots

So far…

- We learned about planning algorithms that generalize across many types of robots

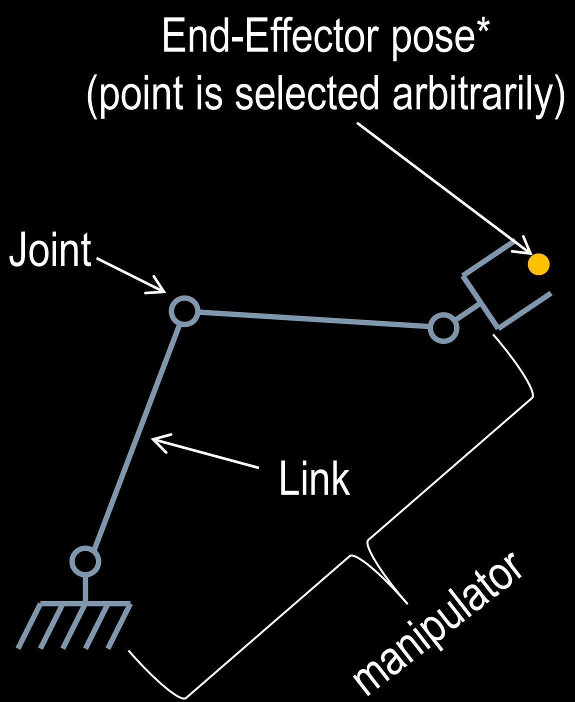
- But many robots are articulated linkages
  - Arms, humanoids, etc.
- Often need to move the *end-effector* (i.e. robot hand) to a goal pose
  - How do we compute the configuration that places the end-effector there?

Outline

- Computing the Manipulator Jacobian
- Using the Manipulator Jacobian for inverse kinematics
- Using the null space to satisfy secondary tasks

Definitions

- C-space is sometimes called **joint space** for articulated robots
  - Let *N* be the number of joints (i.e. the dimension of C-space)
- The end-effector space is called **task space**
  - In 2D: Task space is $SE(2) = R^2 \times S^1$
  - In 3D: Task space is $SE(3) = R^3 \times RP^3$
  - Let *M* be the number of DOF in task space
- A point in task space *x* is called a **pose** of the end-effector



*some people call this the Tool Center Point (TCP)

Forward Kinematics

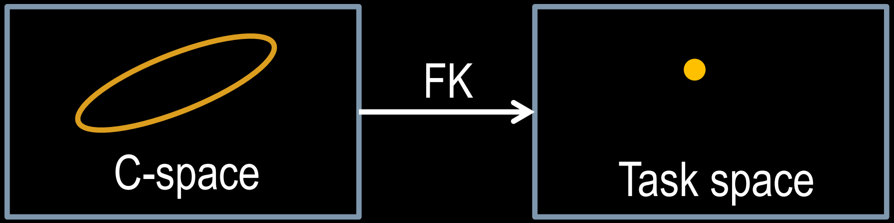
- The **Forward Kinematics** function, given a configuration q, computes the pose of the end-effector x:

$$\boxed{x = FK(q)}$$

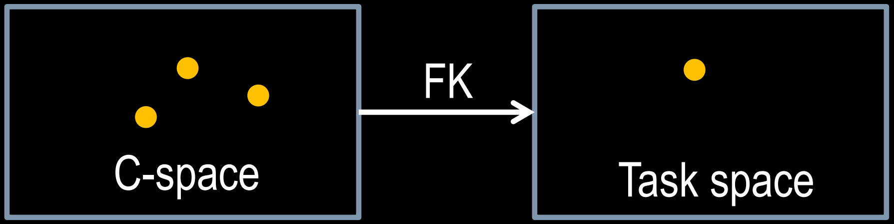
- If *N* (number of joints) is greater than *M* (number of task space DOF), the robot is called **redundant**

Redundancy

- If N > M, FK maps *a continuum* of configurations to *one* end-effector pose:



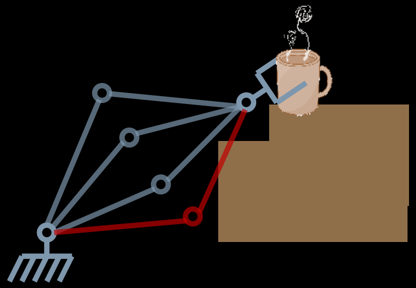
- If N = M, FK maps a *finite number* of configurations to one end-effector pose:



- If N < M, you're in trouble (may not be able to reach a target pose)

C-space and Task Space

- For manipulation, we often don't care about the configuration of the arm (as long as it's feasible), we care about what the end-effector is doing



- Controlling an articulated robot is all about computing a *C-space* motion that does the right thing in *task space*

- Inverse Kinematics (IK) is the problem of computing a configuration that places the end-effector at a given point in task space
  - Analytical solutions exist for some robots if N=M
  - No analytical solution if N > M, Why?
    - For N > M, we can do *Iterative Inverse Kinematics* using the Manipulator Jacobian

The Manipulator Jacobian

- The Manipulator Jacobian converts a velocity in C-space (dq/dt) to a velocity in task space (dx/dt)
- Start with Forward Kinematics function

$$x = FK(q)$$

- Take the derivative with respect to time:

$$\frac{dx}{dt} = \frac{d[FK(q)]}{dt} = \frac{dFK(q)}{dq}\frac{dq}{dt}$$

- Now we get the standard Jacobian equation:

$$\frac{dx}{dt} = J(q)\frac{dq}{dt} \qquad \frac{dFK(q)}{dq} = J(q)$$

Computing the Jacobian

- The Jacobian is a matrix where each column represents the effect of a unit motion of a joint on the end-effector

$$M\left\{\underbrace{\left[\frac{dx}{dq_1}\ \frac{dx}{dq_2}\cdots\right]}_{N}\right. = J(q)$$

Here x is all the end-effector DOF (position and orientation)

- For low-DOF (i.e. up to 3 or 4 joints), you can write the FK function *analytically* and take its derivative to compute $J(q)$
- For higher-DOF robots, it's faster and simpler to compute $J(q)$ *numerically*

The Manipulator Jacobian

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

$$J(q) = \begin{bmatrix} \dfrac{\partial x(q)}{\partial q_1} & \dfrac{\partial x(q)}{\partial q_2} & \cdots & \dfrac{\partial x(q)}{\partial q_n} \\ \xi_1 z_1(q_1) & \xi_2 z_2(q_2) & \cdots & \xi_n z_n(q_n) \end{bmatrix} \begin{array}{l} \leftarrow \text{position} \\ \\ \leftarrow \text{orientation} \end{array}$$

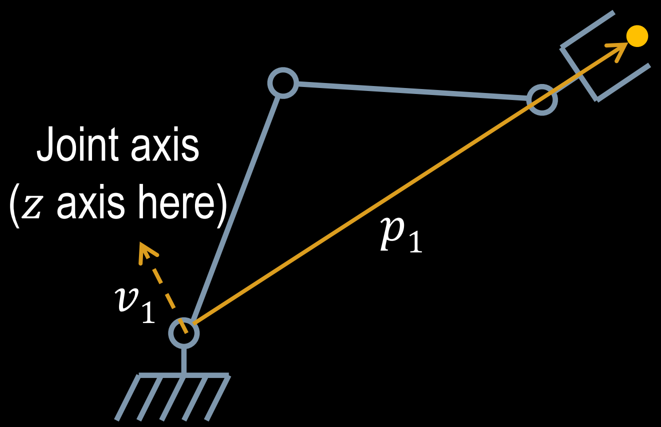
$$\xi_k = \begin{cases} 0 & \text{Prismatic Joint k} \\ 1 & \text{Revolute Joint k} \end{cases}$$

Computing the Manipulator Jacobian: Translation

- You can compute the translation part of J(q) numerically:

$$J(q) = \begin{bmatrix} \boxed{\frac{\partial x(q)}{\partial q_1} \quad \frac{\partial x(q)}{\partial q_2} \quad \cdots \quad \frac{\partial x(q)}{\partial q_n}} \\ \xi_1 z_1(q_1) \quad \xi_2 z_2(q_2) \quad \cdots \quad \xi_n z_n(q_n) \end{bmatrix}$$

1. Place the robot in configuration q (i.e. do FK)
2. For a translation (prismatic) joint: $\frac{dx}{dq_i} = v_i$
3. For a rotation (revolute) joint: $\frac{dx}{dq_i} = v_i \times p_i$



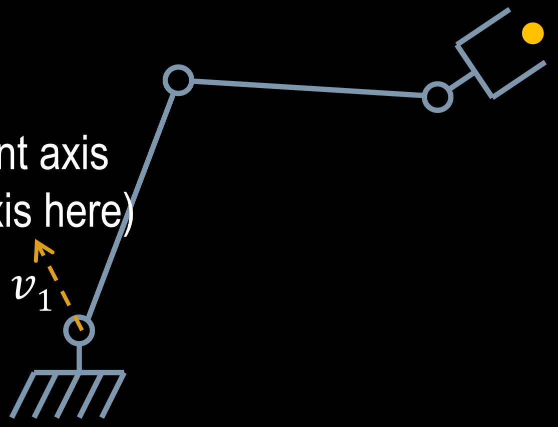
Computing the Manipulator Jacobian: Rotation

- Represent rotation components with angular velocities

$$J(q) = \begin{bmatrix} \frac{\partial x(q)}{\partial q_1} & \frac{\partial x(q)}{\partial q_2} & \cdots & \frac{\partial x(q)}{\partial q_n} \\ \xi_1 z_1(q_1) & \xi_2 z_2(q_2) & \cdots & \xi_n z_n(q_n) \end{bmatrix}$$

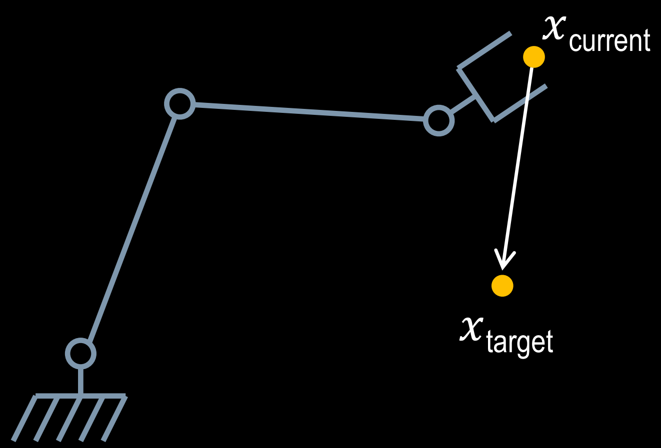
1. Place the robot in configuration q (i.e. do FK)

2. Get joint axis in world frame $\boxed{z_i(q_i) = v_i}$



Using the Manipulator Jacobian for Inverse Kinematics (IK)

- Process: Starting at some configuration, iteratively move closer to x_{target}



$$\frac{dx}{dt} = J(q)\frac{dq}{dt}$$

$(x_{target} - x_{current})$

???

- We need to invert the Jacobian to get the joint velocity dq/dt

$$\frac{dq}{dt} = J(q)^{-1}\frac{dx}{dt}$$

Is it always possible to compute the inverse?

Inverting the Jacobian

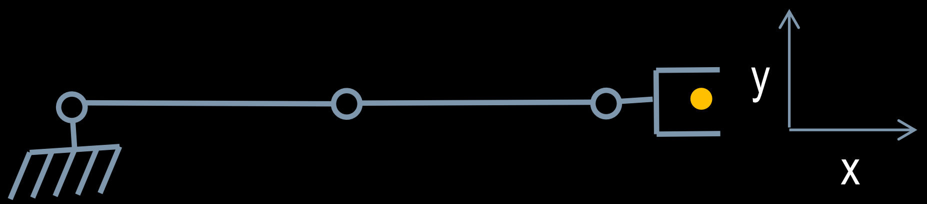
- If N=M, Jacobian is square, so can use the standard matrix inverse
- If N > M, use the Moore-Penrose Right Pseudo-Inverse

$$J(q)^+ = J(q)^T (J(q)J(q)^T)^{-1}$$

- This is the least-squares solution to computing dq/dt

Singularities

- $(J(q)J(q)^T)^{-1}$ is square, but what if $(J(q)J(q)^T)^{-1}$ is singular, i.e. we have lost a degree of freedom?



A singular configuration: no way to move in x!

Inverting the Jacobian

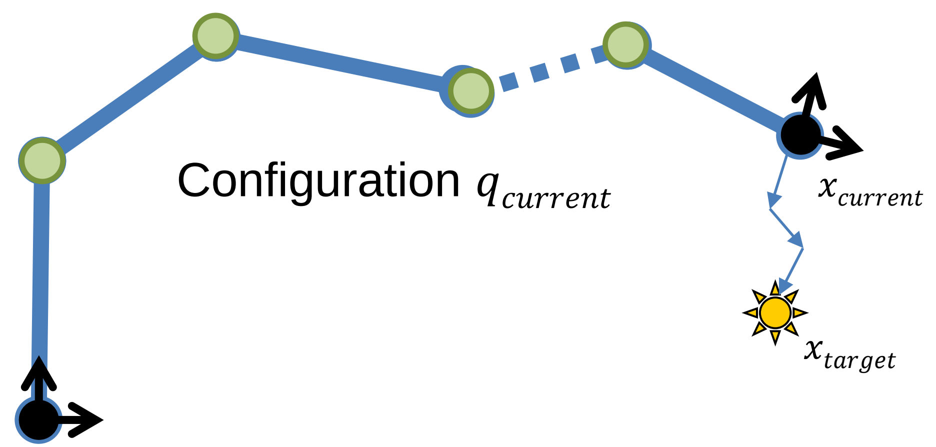
- We can add a small constant along the diagonal of $J(q)J(q)^T$ to make it invertible when it is singular

$$J(q)^+ \approx J(q)^T(J(q)J(q)^T + \boxed{\lambda^2\mathbf{I}})^{-1}$$

- This is called “damped least-squares”
- The matrix will be invertible but this technique introduces a small inaccuracy

Iterative Jacobian Pseudo-Inverse Inverse Kinematics

While true
$\quad x_{current} = FK(q_{current})$
$\quad \dot{x} = (x_{target} - x_{current})$
$\quad$ error = $||\dot{x}||$
$\quad$ If error < threshold
$\quad\quad$ return $q_{current}$
$\quad \dot{q} = J(q)^{+}\,\dot{x}$
$\quad$ If($||\dot{q}|| > \alpha$)
$\quad\quad \dot{q} = \alpha(\dot{q} / ||\dot{q}||)$
$\quad q_{current} = q_{current} + \dot{q}$
end



- This is a local method, it will get stuck in local minima (i.e. joint limits)!!!
- α is the step size
- Numerical error handling not shown
- A correction matrix has to be applied to the angular velocity components to map them into the target frame (not shown)

Break

Secondary Tasks

- So far, we only considered how to get the end-effector to a given pose

- What if we also want to avoid
  - Joint limits
  - Obstacles

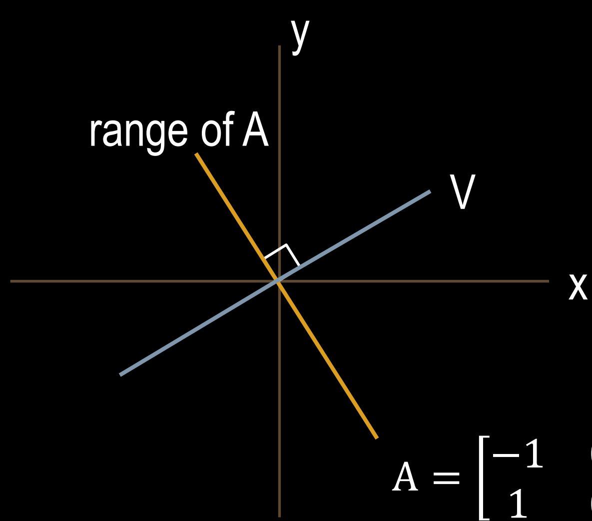
- How do we account for these secondary tasks when doing Jacobian-based Iterative IK?

The Left Null-space

- We can try to satisfy secondary tasks in the *null-space* of the Jacobian pseudo-inverse
- In linear algebra, the *left null-space* of a matrix A is the set of vectors *V*:

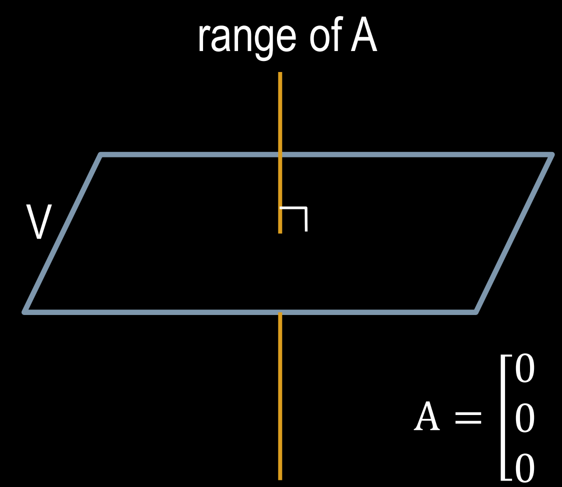
$$V = \{v \in \boldsymbol{R}^m \mid \mathrm{A}^T v = 0\}, \quad \mathrm{A} \in \boldsymbol{R}^{m \times n}$$

- You can prove that V is orthogonal to the range of A



$$\mathrm{A} = \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}$$

2D example



$$\mathrm{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

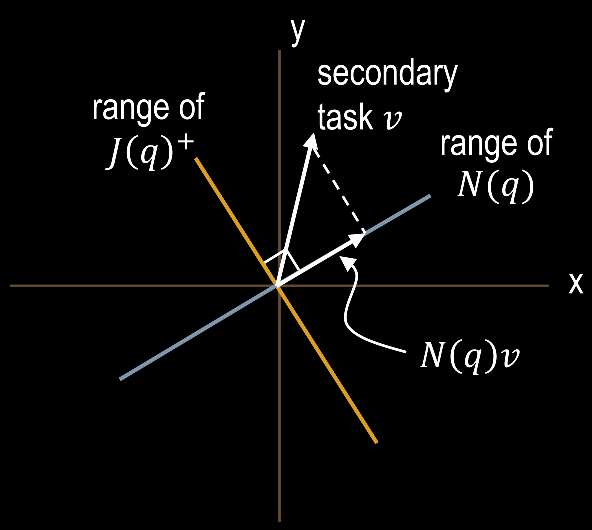
3D example

Left Null-space of the Jacobian Pseudo-inverse

- For our purposes, this means that the secondary task will not disturb the primary task
- The left null-space projection matrix for the Jacobian pseudo-inverse is:

$$N(q) = (I - J(q)^{+}J(q))$$

- To project a vector into the left null-space, just multiply it by the above matrix



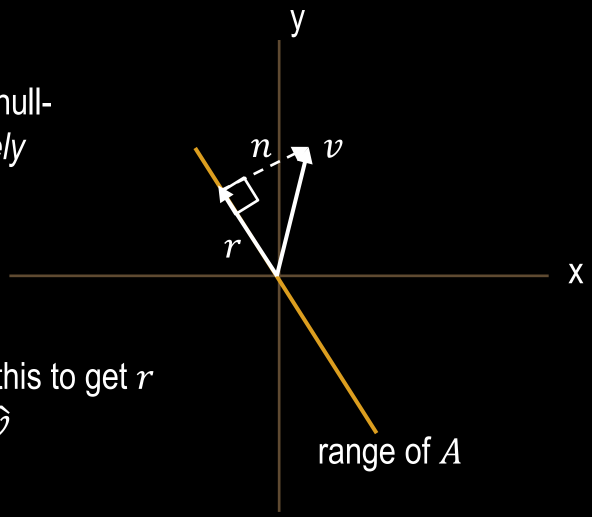
Why does this work?

- First, decompose v into two orthogonal parts:

Part of v that is in the range of A

$$v = r + n$$

Part of v that is in the left null-space of A *(this is ultimately what we want to get)*



$$n = v - r = v - A\hat{v}$$

Need to compute this to get r

$$r = A\hat{v}$$

$$A^T n = 0$$

$$A^T(v - A\hat{v}) = 0$$

$$A^T v = A^T A\hat{v}$$

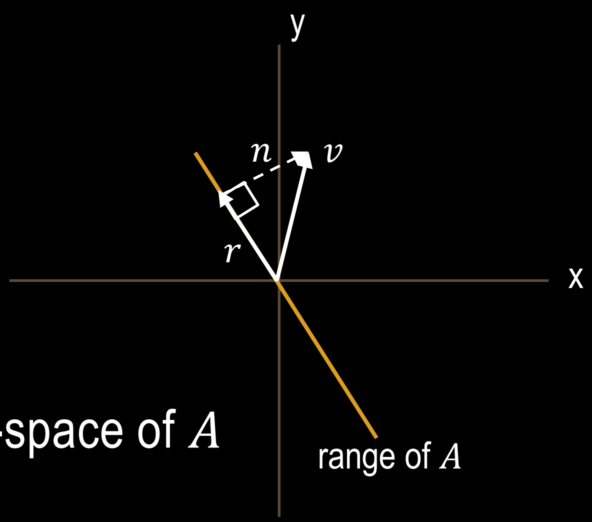
$$\hat{v} = (A^T A)^{-1} A^T v$$

Why does this work?

- Use this relationship to get r:

$$\hat{v} = (A^T A)^{-1} A^T v$$

$$r = A\hat{v}$$
$$r = A(A^T A)^{-1} A^T v$$
$$r = AA^+ v$$



- Now we can find n, the part of v that is in the left null-space of A

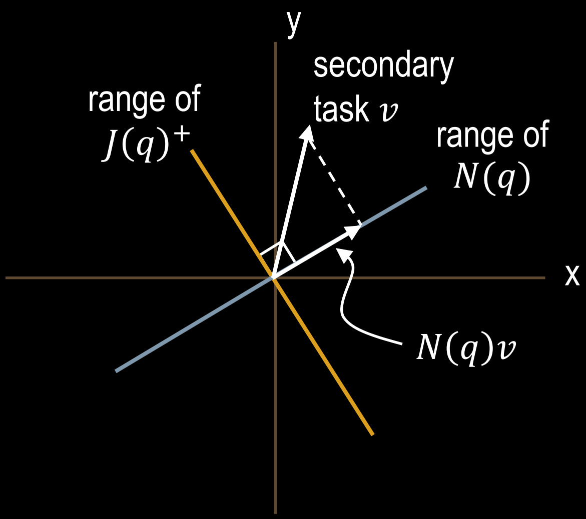
$$n = v - AA^+ v = (I - AA^+)v$$

This is the left null-space projection matrix

Why does this work?

- Now we plug in the Jacobian pseudo-inverse

$$n = (I - AA^+)v$$
$$A = J(q)^+$$
$$n = \underbrace{(I - J(q)^+ J(q))}_{\text{This is } N(q)} v$$



Combining tasks using the null-space

- Combining the primary task dx_1/dt and the secondary task dq_2/dt :

$$\frac{dq}{dt} = \overbrace{J(q)^+ \frac{dx_1}{dt}}^{\text{Motion for Primary Task}} + \overbrace{\beta(I - J(q)^+ J(q)) \frac{dq_2}{dt}}^{\text{Motion for Secondary Task}}$$

- This guarantees that the projection of q_2 is orthogonal to $J(q)^+(dx_1/dt)$
  - Assuming the system is linear

Using the Null-space

- The null-space is often used to “push” IK solvers away from
  - Joint limits
  - Obstacles

- How do we define the secondary task for the two constraints above?

$$\frac{dq}{dt} = J(q)^{+}\frac{dx_1}{dt} + \beta\left(I - J(q)^{+}J(q)\right)\frac{dq_2}{dt}$$

???

$$\frac{dq}{dt} = J(q)^+ \frac{dx_1}{dt} + \beta(I - J(q)^+ J(q)) \frac{dq_2}{dt}$$

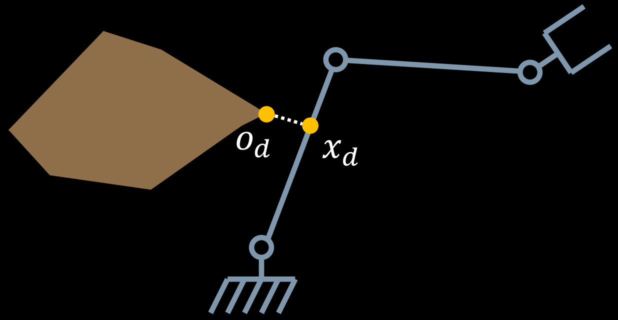
???

- Joint limits

$q_{max,i}$ is upper limit for joint i
$q_{min,i}$ is lower limit for joint i

$$\frac{dq_2}{dt} =$$

- Obstacles



$$\frac{dq_2}{dt} =$$

Using the Null-space

$$\frac{dq}{dt} = J(q)^+ \frac{dx_1}{dt} + \beta(I - J(q)^+ J(q)) \frac{dq_2}{dt}$$

Why do we need this scalar?

- What guarantees do we have about accomplishing the secondary task?

- Let's say you have a 6 DOF arm reaching for a 3D pose (6 DOF). Assume the arm is not at a singularity. What will $(I - J(q)^+ J(q))$ be?

Summary

- We saw how to compute the Jacobian numerically
- The Jacobian can be used to solve IK problems, but we have to be careful about numerical issues
  - It is a local method, not a substitute for path planning
- The null-space of the Jacobian pseudo-inverse can be used to accomplish secondary tasks but we lose degrees of freedom in the null-space projection
  - Really only useful when you have a redundant robot

Homework

- Read Grasping Foundations