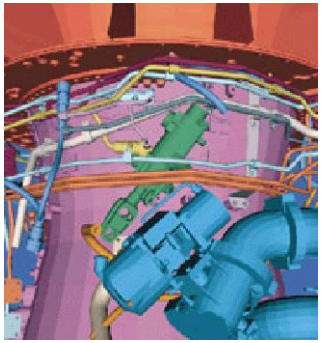
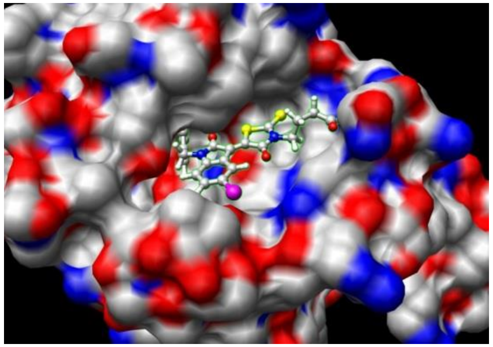
Motion Planning for Articulated Robots 1

Jane Li
Assistant Professor
Mechanical Engineering & Robotics Engineering
http://users.wpi.edu/~zli11

Recap

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





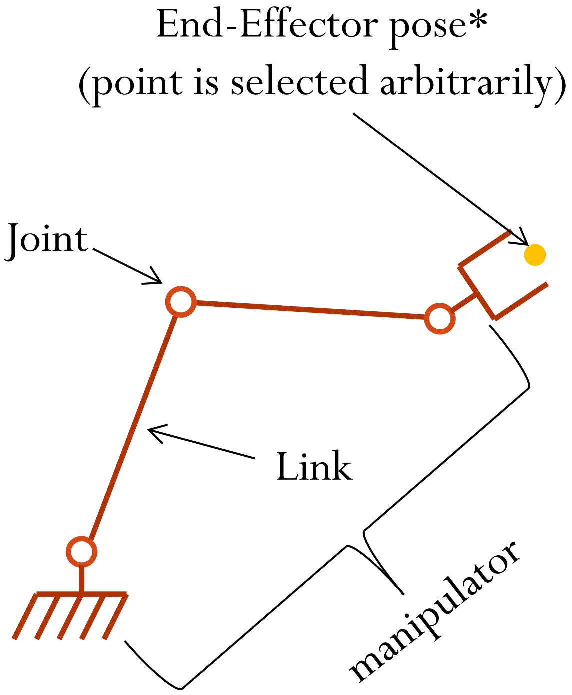
- But many robots are articulated linkages
  - Arms, humanoids, etc.
- Can we take advantage of this structure in motion planning?
  - Yes! But we have to learn how these robots are controlled

Outline

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

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 X S^1
  - In 3D: Task space is SE(3) = R^3 X 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



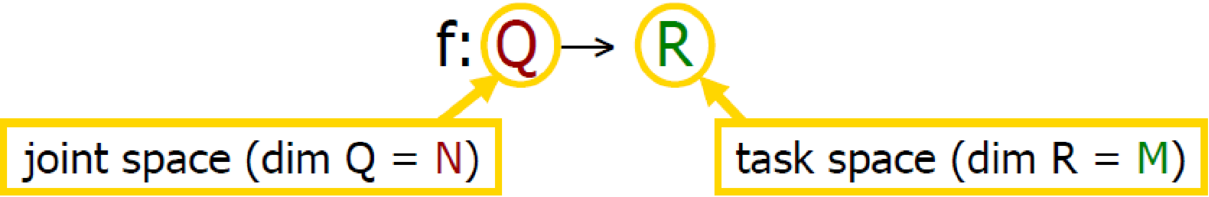
* Some people call this the Tool Center Point (TCP)

Forward Kinematics

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

$$x = FK(q)$$

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

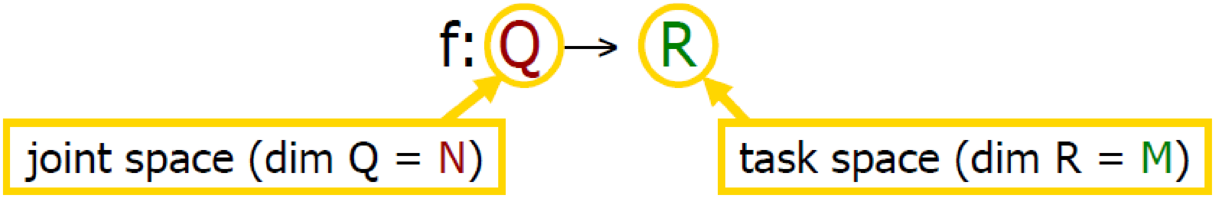


$f: Q \to R$

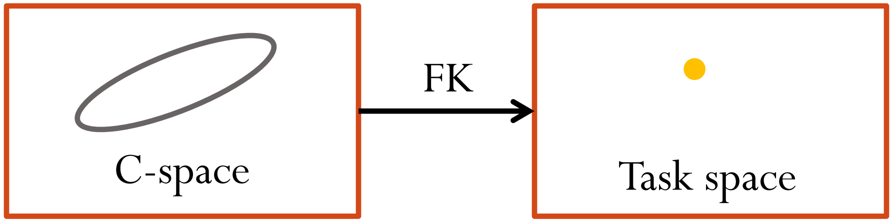
joint space (dim Q = N)

task space (dim R = M)

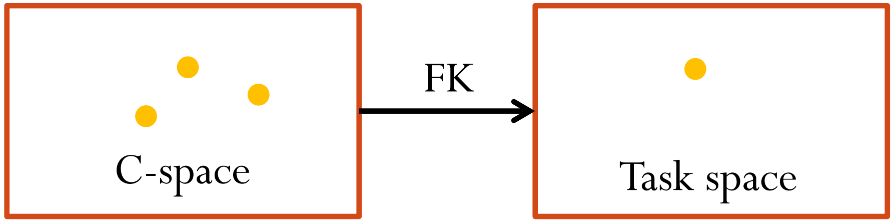
Redundancy



- 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 unique solution if N > M, Why?
        - For N > M, what can we do?

The Jacobian

- The 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:

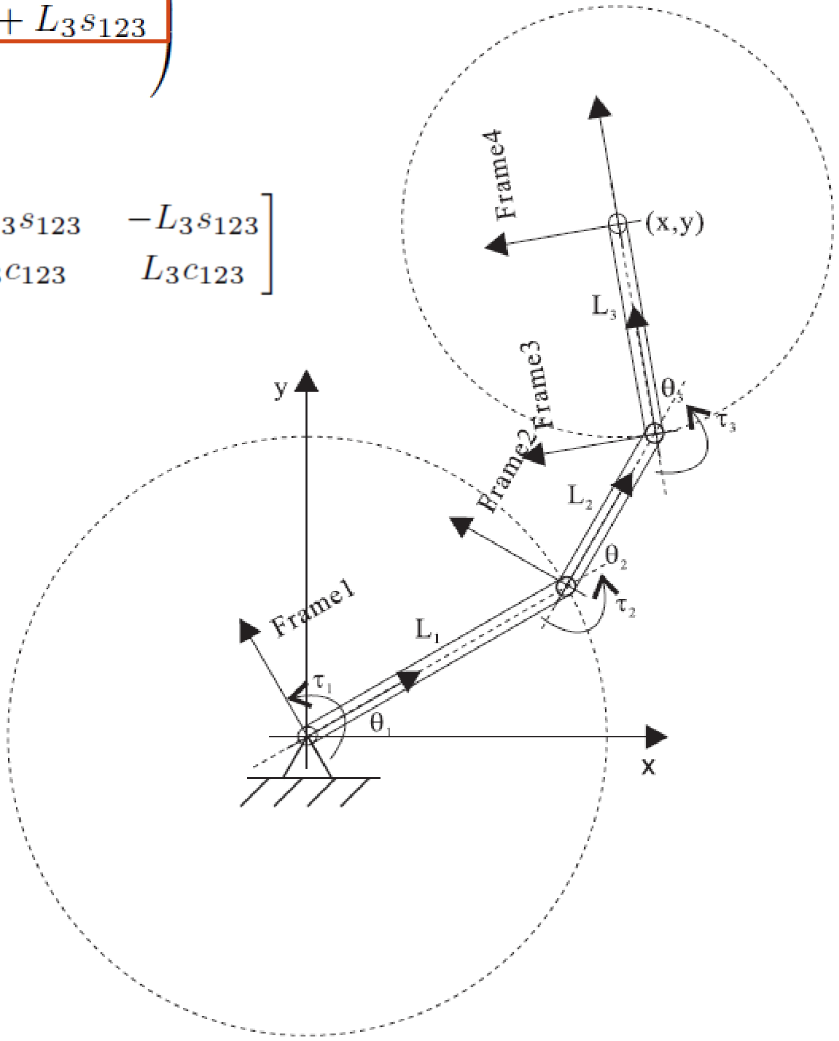
$$\frac{dx}{dt} = J(q)\frac{dq}{dt} \qquad \frac{dFK(q)}{dq} = J(q)$$

Example

$$ {}^4_0\mathrm{T} = {}^4_0\,\mathrm{T}(\theta_1, \theta_2, \theta_3) = \begin{pmatrix} c_{123} & -s_{123} & L_1 c_1 + L_2 c_{12} + L_3 c_{123} \\ s_{123} & c_{123} & L_1 s_1 + L_2 s_{12} + L_3 s_{123} \\ 0 & 0 & 1 \end{pmatrix} $$

$$ \mathbf{J}_{m\times n} = \mathbf{J}_{2\times 3} = \begin{bmatrix} -L_1 s_1 - L_2 s_{12} - L_3 s_{123} & -L_2 s_{12} - L_3 s_{123} & -L_3 s_{123} \\ L_1 s_1 + L_2 c_{12} + L_3 c_{123} & L_2 c_{12} + L_3 c_{123} & L_3 c_{123} \end{bmatrix} $$

$$ \dot{\mathbf{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \mathbf{J}\dot{\mathbf{q}} = \mathbf{J} \begin{pmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{pmatrix} $$



Computing the Jacobian

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

$$M\left\{\underbrace{\begin{bmatrix}\dfrac{dx}{dq_1} & \dfrac{dx}{dq_2} & \cdots\end{bmatrix}}_{N}\right. = J(q)$$

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

- For simple systems (i.e. up to 3 or 4 links), you can write the FK function analytically and take its derivative to compute J(q)

The Manipulator Jacobian

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

$$J(q)=\left[\begin{array}{cccc} \dfrac{\partial x(q)}{\partial q_1} & \dfrac{\partial x(q)}{\partial q_2} & \cdots & \dfrac{\partial x(q)}{\partial q_n} \\ \\ \xi_1 z_0(q) & \xi_2 z_1(q) & \cdots & \xi_n z_{n-1}(q) \end{array}\right] \begin{array}{l} \leftarrow \text{position} \\ \\ \leftarrow \text{rotation} \end{array}$$

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

Computing the Jacobian: Translation

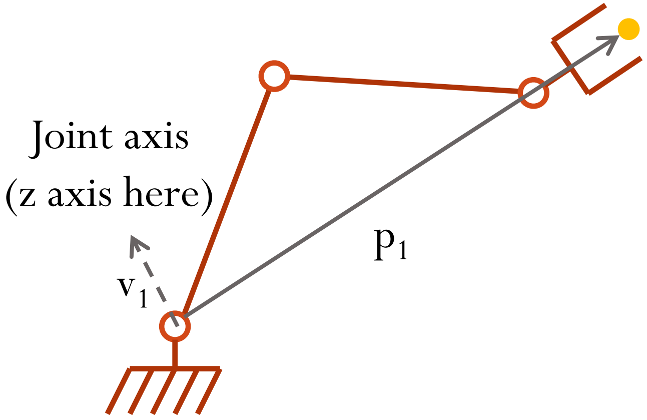
- You can compute the translation part of J(q) numerically:

$$J(q)=\begin{bmatrix} \boxed{\begin{matrix} \frac{\partial x(q)}{\partial q_1} & \frac{\partial x(q)}{\partial q_2} & \cdots & \frac{\partial x(q)}{\partial q_n} \end{matrix}} \\ \begin{matrix} \xi_1 z_0(q) & \xi_2 z_1(q) & \cdots & \xi_n z_{n-1}(q) \end{matrix} \end{bmatrix}$$

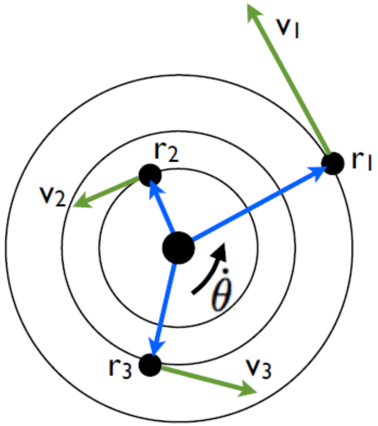
1. Place the robot in configuration q

2. For a translation (prismatic) joint: $\frac{dx}{dq_i}=v_i$

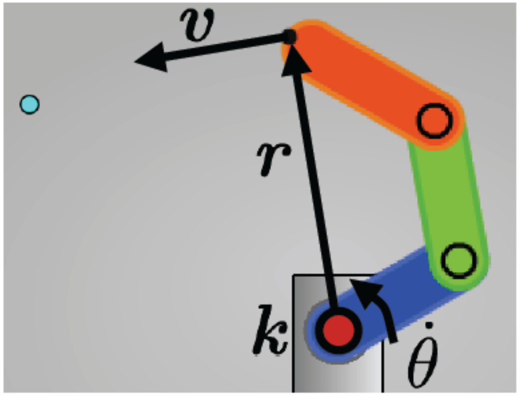
3. For a rotation (hinge) joint: $\frac{dx}{dq_i}=v_i\times p_i$



Computing the Jacobian: Translation



k axis out-of-plane
and passes through frame origin



$$\omega = \dot{\theta}\boldsymbol{k}$$

angular velocity of points in frame wrt. axis k

rotation axis

angular rotation in frame

$$\boldsymbol{v} = \omega \times \boldsymbol{r}$$

linear velocity of points in frame wrt. axis k

vector to point in frame

$$\boldsymbol{v} = \dot{\theta}\boldsymbol{k} \times \boldsymbol{r}$$

vector from joint origin to endeffector

endeffector linear velocity

joint rotation axis

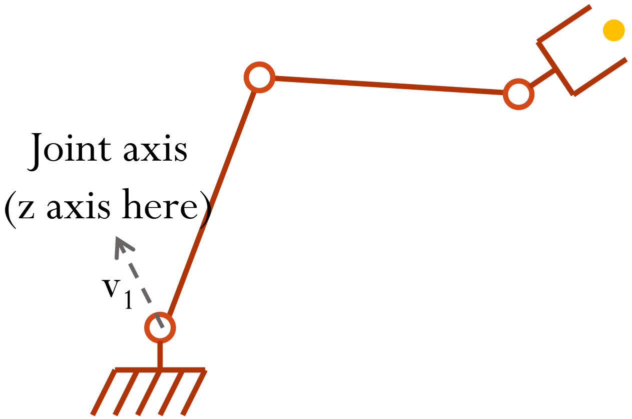
Computing the Jacobian: Rotation

- Represent rotation components with angular velocities

$$J(q)=\begin{bmatrix} \frac{\partial x(q)}{\partial q_1} & \frac{\partial x(q)}{\partial q_2} & \cdots & \frac{\partial x(q)}{\partial q_n} \\ \xi_1 z_0(q) & \xi_2 z_1(q) & \cdots & \xi_n z_{n-1}(q) \end{bmatrix}$$

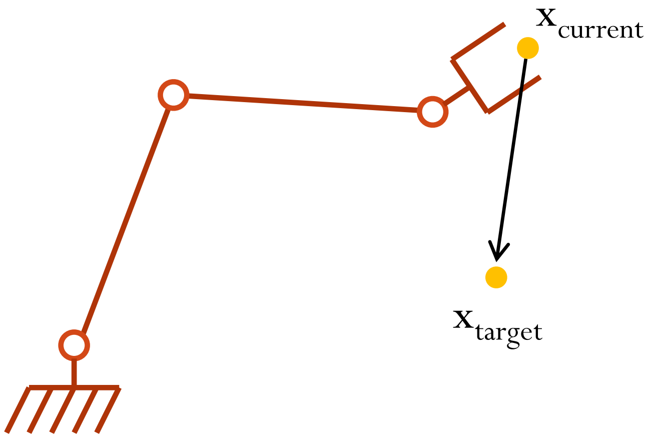
1. Place the robot in configuration q

2. Extract joint axis in world frame $z_i(q)=v_i$



Using the Jacobian for Inverse Kinematics (IK)

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



$$\underbrace{\frac{dx}{dt}}_{(x_{target}-\ x_{current})} = J(q)\underbrace{\frac{dq}{dt}}_{???}$$

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

Inverting the Jacobian

- If N=M,
  - Jacobian is square → Standard matrix inverse
- If N>M ,
  - Pseudo-Inverse
  - Weighted Pseudo-Inverse
  - Damped least squares
  - Iterative Jacobian Pseudo-Inverse

Pseudo-Inverse

$$\dot{q} = J^{\dagger}(q)\dot{x} \qquad \left\{ \begin{aligned} \mathbf{J}\mathbf{J}^{\dagger}\mathbf{J} &= \mathbf{J} \\ \mathbf{J}^{\dagger}\mathbf{J}\mathbf{J}^{\dagger} &= \mathbf{J}^{\dagger} \\ (\mathbf{J}\mathbf{J}^{\dagger})^{T} &= \mathbf{J}\mathbf{J}^{\dagger} \\ (\mathbf{J}^{\dagger}\mathbf{J})^{T} &= \mathbf{J}^{\dagger}\mathbf{J} \end{aligned} \right.$$

- Cases

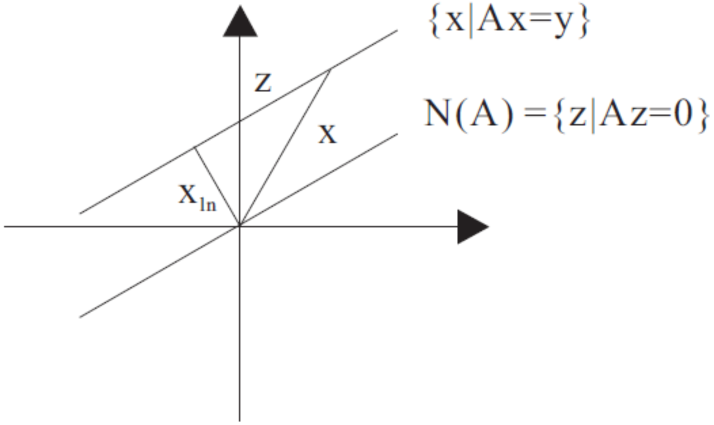
$$\mathbf{J}^{\dagger}_{n \times m} = \begin{cases} \mathbf{J}^{T}(\mathbf{J}\mathbf{J}^{T})^{-1} & m < n \longrightarrow \text{Fat Jacobian, redundant robot} \\ \mathbf{J}^{-1} & m = n \longrightarrow \text{Square Jacobian, standard pseudo inverse} \\ (\mathbf{J}\mathbf{J}^{T})^{-1}\mathbf{J}^{T} & m > n \longrightarrow \text{Tall Jacobian, under-actuated robot} \end{cases}$$

Behind Pseudo-inverse

- What does pseudo-inverse optimize?

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

Where J is full (row)-rank matrix



- Optimization

Minimize $\frac{1}{2}\dot{\theta}^T\dot{\theta}$ given that $\dot{x} = J(\theta)\dot{\theta}$

Behind Pseudo-inverse

- Minimize $||\mathbf{x}||^2 = \mathbf{x}^T\mathbf{x}$ given $A\mathbf{x} = \mathbf{y}$
- Derive the optimization problem using **Lagrange multipliers**

$$L(x,\lambda) = \mathbf{x}^T\mathbf{x} + \lambda^T(A\mathbf{x} - \mathbf{y})$$

- Optimal condition

$$\left.\begin{array}{lll} \dfrac{\partial L}{\partial \mathbf{x}} = 2\mathbf{x} + A^T\lambda = 0 \Longrightarrow \mathbf{x} = -\dfrac{A^T\lambda}{2} & & \\ \dfrac{\partial L}{\partial \lambda} = A\mathbf{x} - \mathbf{y} = 0 \Longrightarrow \mathbf{y} = A\mathbf{x} = \dfrac{-AA^T\lambda}{2} & \Longrightarrow & \lambda = -2(AA^T)^{-1}\mathbf{y} \end{array}\right\}$$

$$\mathbf{x}_{ln} = \frac{-A^T(-2(AA^T)^{-1}\mathbf{y})}{2} = \boxed{A^T(AA^T)^{-1}\mathbf{y}}$$

Weighted Pseudo-Inverse

$$\dot{q} = J_w^{\dagger}(q)\dot{x}$$

- Weighted Pseudo-Inverse

$$J_w^{\dagger}(q) = W^{-1}J^T(JW^{-1}J^T)^{-1}$$

- What to optimize?

Minimize $\frac{1}{2}\|\dot{q}\|_w^2 = \frac{1}{2}\dot{q}^T W \dot{q}$ given that $\dot{x} = J(\theta)\dot{\theta}$

- Significance of the weight
  - W>0 and symmetric
  - Large weight → small joint velocity
  - Weight can be chosen proportional to the inverse of the joint angle range

Singular Value Decomposition (SVD)

$$J = U\Sigma V^T$$

the SVD routine of Matlab applied to J provides two orthonormal matrices $U_{M\times M}$ and $V_{N\times N}$, and a matrix $\Sigma_{M\times N}$ of the form

$$\Sigma = \left(\begin{array}{cccc|c} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \ddots & & 0_{Mx(N-M)} \\ & & & \sigma_M & \end{array}\right) \qquad \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_\rho > 0, \quad \sigma_{\rho+1} = \ldots = \sigma_M = 0$$

singular values of J

- The columns of U → eigenvectors of JJ^T
- The columns of V → eigenvectors of J^TJ

Singular Value Decomposition (SVD)

$$J = U\Sigma V^T$$

$$\begin{aligned}
\mathbf{J}_{m\times n} &= \mathbf{U}_{m\times m}\,\mathbf{\Sigma}_{m\times n}\,\mathbf{V^T}_{n\times n} \\
&= \begin{bmatrix} \mathbf{U}_{1_{m\times r}} & \mathbf{U}_{2_{m\times(m-r)}} \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma_1}_{r\times r} & \mathbf{0}_{r\times(n-r)} \\ \mathbf{0}_{(m-r)\times r} & \mathbf{0}_{(m-r)\times(n-r)} \end{bmatrix} \begin{bmatrix} \mathbf{V^T}_{1_{r\times n}} \\ \mathbf{V^T}_{2_{(n-r)\times n}} \end{bmatrix}
\end{aligned}$$

$\mathbf{V^T}_{1_{r\times n}}$ The orthogonal basis for the subspace of **joint velocity** that generate **non-zero** task space velocities

$\mathbf{V^T}_{2_{(n-r)\times n}}$. The orthogonal basis for the subspace of **joint velocity** that gives **zero** task space velocity →Null space of J

Singular Value Decomposition (SVD)

$$J = U\Sigma V^T$$

$$\begin{aligned}
\mathbf{J}_{m\times n} &= \mathbf{U}_{m\times m}\,\mathbf{\Sigma}_{m\times n}\,\mathbf{V^T}_{n\times n} \\
&= \begin{bmatrix} \mathbf{U}1_{m\times r} & \mathbf{U}2_{m\times(m-r)} \end{bmatrix}
\begin{bmatrix} \mathbf{\Sigma_1}_{r\times r} & \mathbf{0}_{r\times(n-r)} \\ \mathbf{0}_{(m-r)\times r} & \mathbf{0}_{(m-r)\times(n-r)} \end{bmatrix}
\begin{bmatrix} \mathbf{V^T}1_{r\times n} \\ \mathbf{V^T}2_{(n-r)\times n} \end{bmatrix}
\end{aligned}$$

$\mathbf{U}1_{m\times r}$ The orthogonal basis for the subspace of achievable **task space** velocity $\dot{\mathrm{X}}$

$\mathbf{U}2_{m\times(m-r)}$ The orthogonal basis for the subspace of **task space** velocities $\dot{\mathrm{X}}$ that can not be generated by the robots

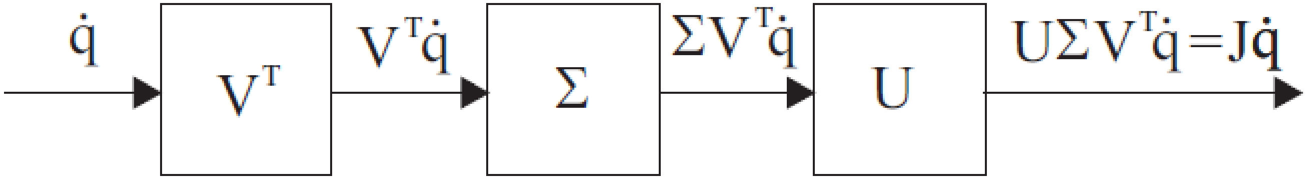
Singular Value Decomposition (SVD)

$$J = U\Sigma V^T$$

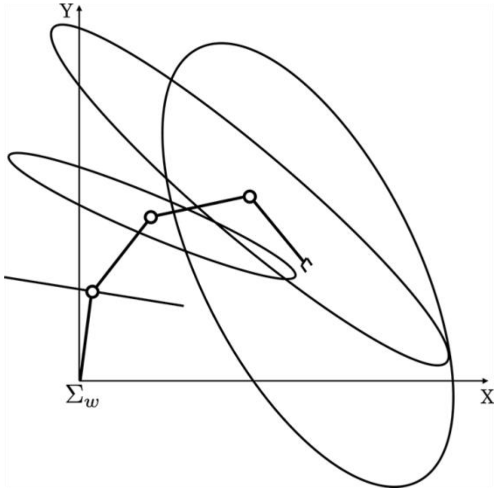
$$\begin{aligned}\mathbf{J}_{m\times n} &= \mathbf{U}_{m\times m}\boldsymbol{\Sigma}_{m\times n}\mathbf{V^T}_{n\times n}\\ &= \begin{bmatrix}\mathbf{U}1_{m\times r} & \mathbf{U}2_{m\times(m-r)}\end{bmatrix}\begin{bmatrix}\boldsymbol{\Sigma}_{\mathbf{1}_{r\times r}} & \mathbf{0}_{r\times(n-r)}\\ \mathbf{0}_{(m-r)\times r} & \mathbf{0}_{(m-r)\times(n-r)}\end{bmatrix}\begin{bmatrix}\mathbf{V^T}1_{r\times n}\\ \mathbf{V^T}2_{(n-r)\times n}\end{bmatrix}\end{aligned}$$

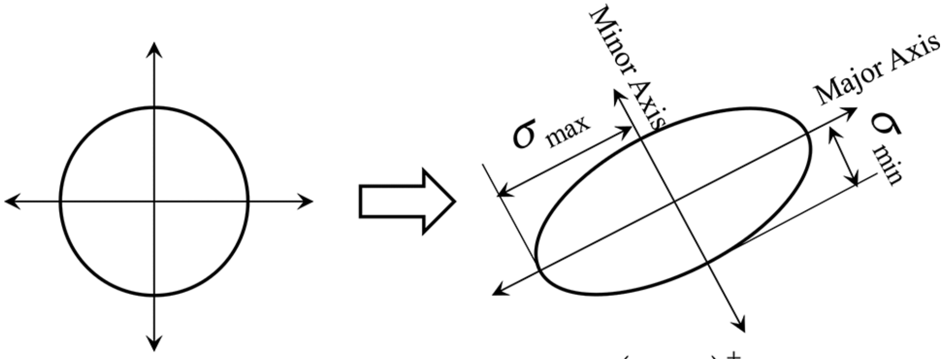
$\boldsymbol{\Sigma}1_{r\times r}$ The velocity **transmission ratio** from the joint space to the task space

Singular Value Decomposition (SVD)



$\dot{q}$ → V^T → $V^T\dot{q}$ → Σ → $\Sigma V^T\dot{q}$ → U → $U\Sigma V^T\dot{q} = J\dot{q}$

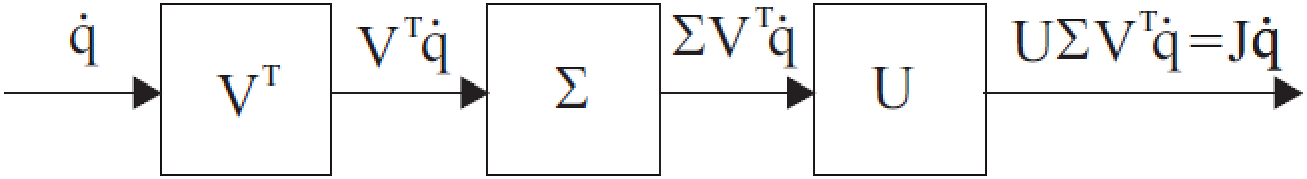


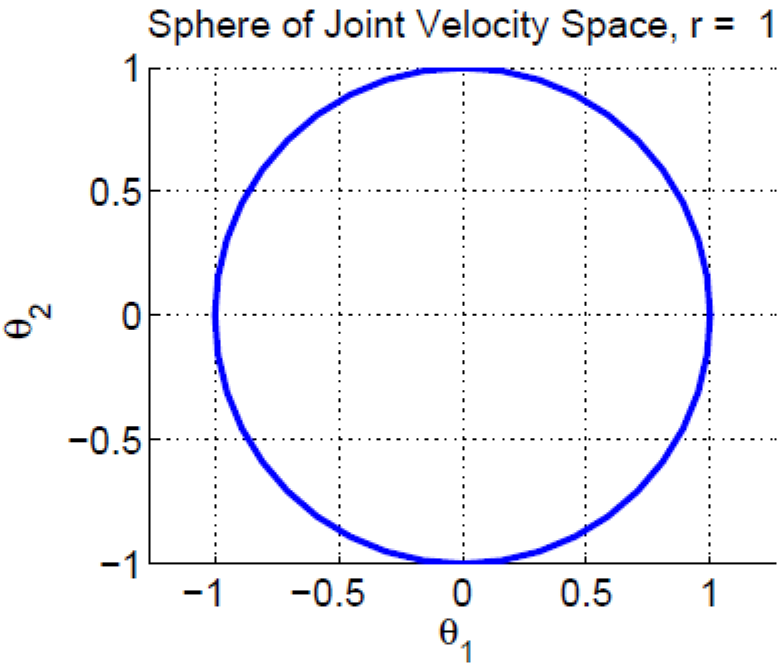


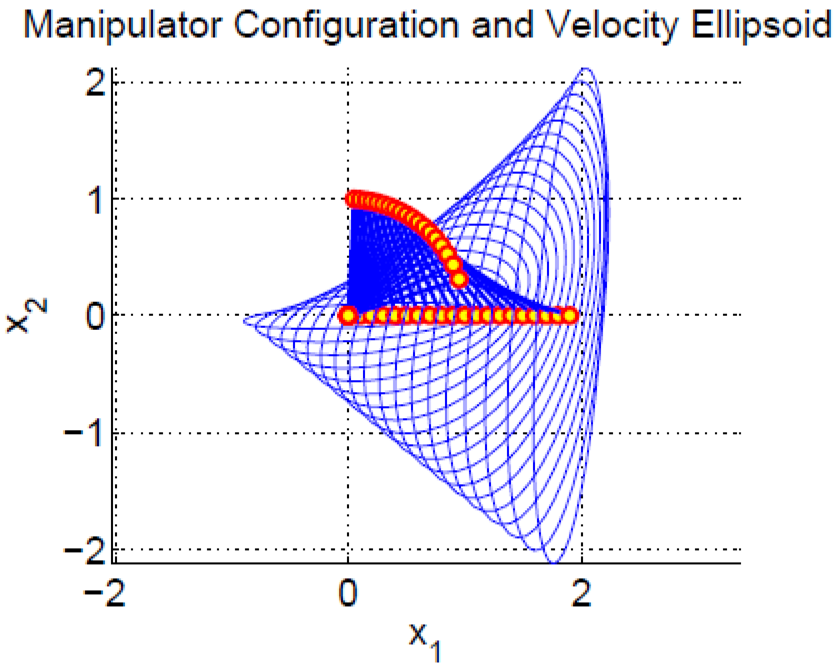
$\boldsymbol{q}^T\boldsymbol{q} \le 1$

$\boldsymbol{V}^T\left(\boldsymbol{J}\boldsymbol{J}^T\right)^+\boldsymbol{V} \le 1$

Singular Value Decomposition (SVD)







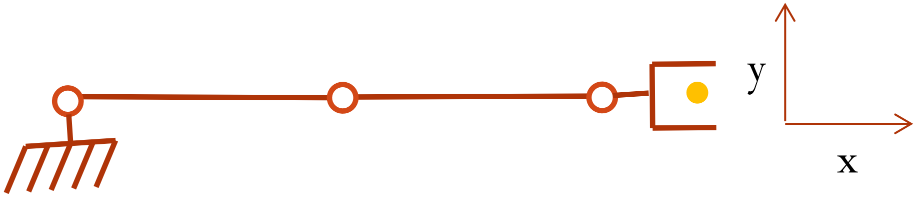
Singular Value Decomposition (SVD)

$$J = U\Sigma V^T \quad \text{where} \quad \Sigma = \left(\begin{array}{cccc|c} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \ddots & & 0_{\mathrm{Mx(N-M)}} \\ & & & \sigma_{\mathrm{M}} & \end{array}\right)$$

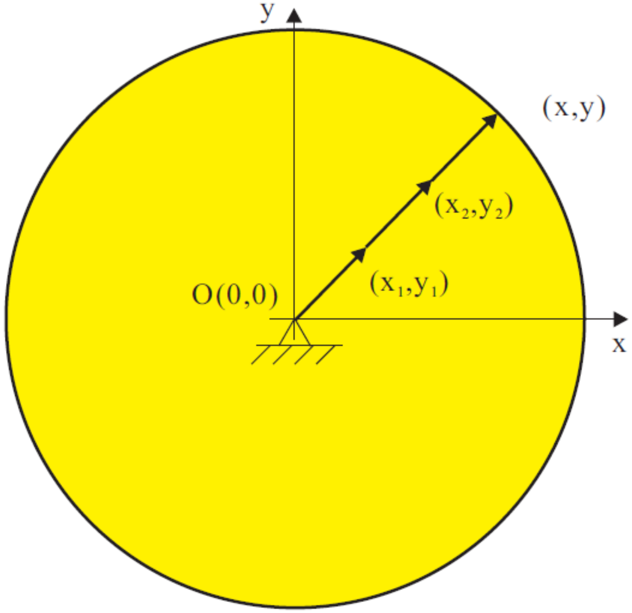
$$J^\dagger = V\Sigma^\dagger U^T \quad \text{where} \quad \Sigma^\dagger = \left(\begin{array}{ccc} \frac{1}{\sigma_1} & & \\ & \ddots & \\ & & \frac{1}{\sigma_\rho} \\ & & 0 \\ \hline & 0_{\mathrm{(N-M)xM}} & \end{array}\right)$$

Singularities

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



A singular configuration: no way to move in x!



Damped Least Squares

unconstrained minimization of a **suitable** objective function

$$\min_{\dot{q}} \frac{\mu^2}{2}\|\dot{q}\|^2 + \frac{1}{2}\|\dot{x} - J\dot{q}\|^2 = H(\dot{q})$$

compromise between large joint velocity and task accuracy

SOLUTION

$$\boxed{\dot{q} = J_{DLS}(q)\dot{X} = J^T\left(JJ^T + \mu^2 I_M\right)^{-1}\dot{X}}$$

- Significance
  - To render robust behavior when crossing the singularity, we can add a small constant along the diagonal of $(J(q)^T J(q))$ to make it invertible when it is singular → "**damped least-squares**"
  - The matrix will be invertible but this technique introduces a small inaccuracy → **error**?

Damped Least Squares

- Induced error by DLS

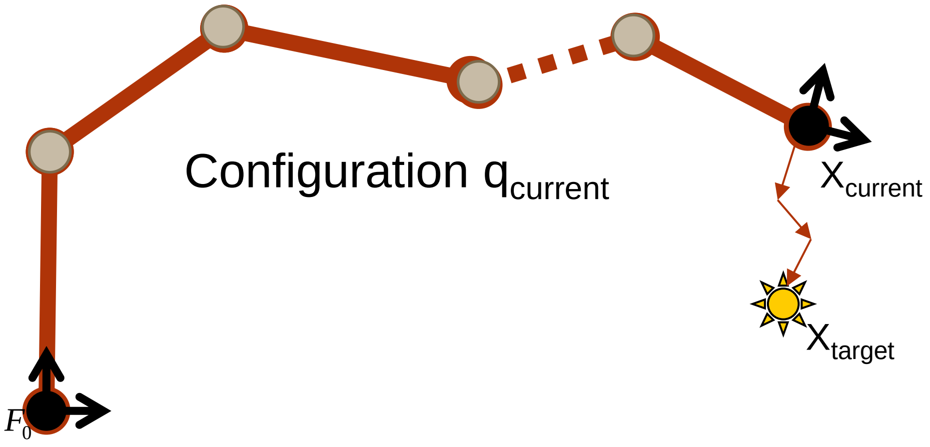
$$\dot{e} = \mu^2 \left(JJ^T + \mu^2 I_M\right)^{-1} \dot{x} \quad \text{(as in N=M case)}$$

$$\text{using SVD of } J=U\Sigma V^T \Rightarrow J_{DLS} = V\Sigma_{DLS}U^T \text{ with } \Sigma_{DLS} = \left(\begin{array}{cc} \boxed{\mathrm{diag}\{\frac{\sigma_i}{\sigma_i^2+\mu^2}\}} & \\ \rho\times\rho & \mathrm{diag}\{\frac{1}{\mu^2}\} \\ \hline 0_{(N-M)x\rho} & 0_{(N-M)x(N-\rho)} \end{array}\right)$$

- Choice of the damping factor $\mu^2(q) \geq 0,$
  - As a function the minimum singular value → measure of distance to singularity
  - Induce the damping only/mostly in the non-feasible direction of the task

Iterative Jacobian Pseudo-Inverse Inverse Kinematics

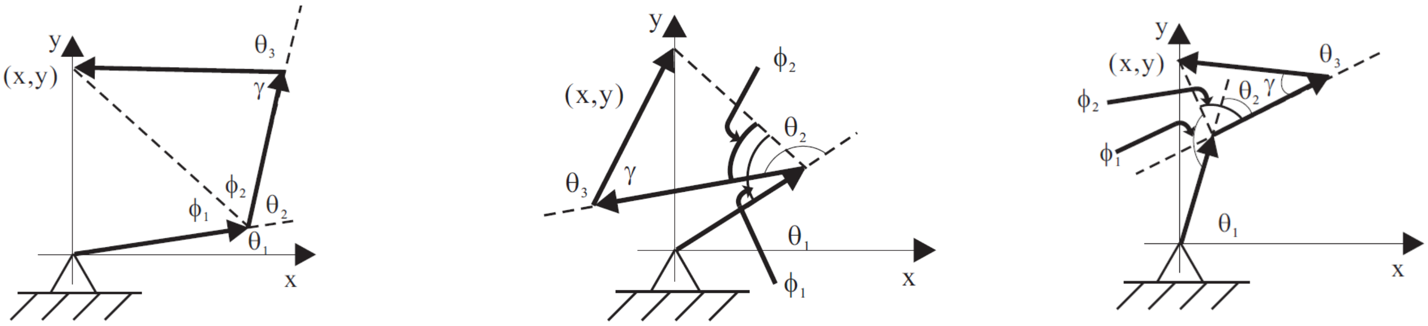
```
While true
    $X_{current} = FK(q_{current})$
    $\dot{x} = (x_{target} - x_{current})$
    error = $|| \dot{x} ||$
    If error < threshold
        return Success
    $\dot{q} = J(q)^{+} \dot{x}$
    If($||\dot{q}|| > \alpha$)
        $\dot{q} = \alpha(\dot{q} / ||\dot{q}||)$
    $q_{current}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



- This is a local method, it will get stuck in local minima (i.e. joint limits)!!!
- α is the step size
- 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)

Null Space of Jacobian

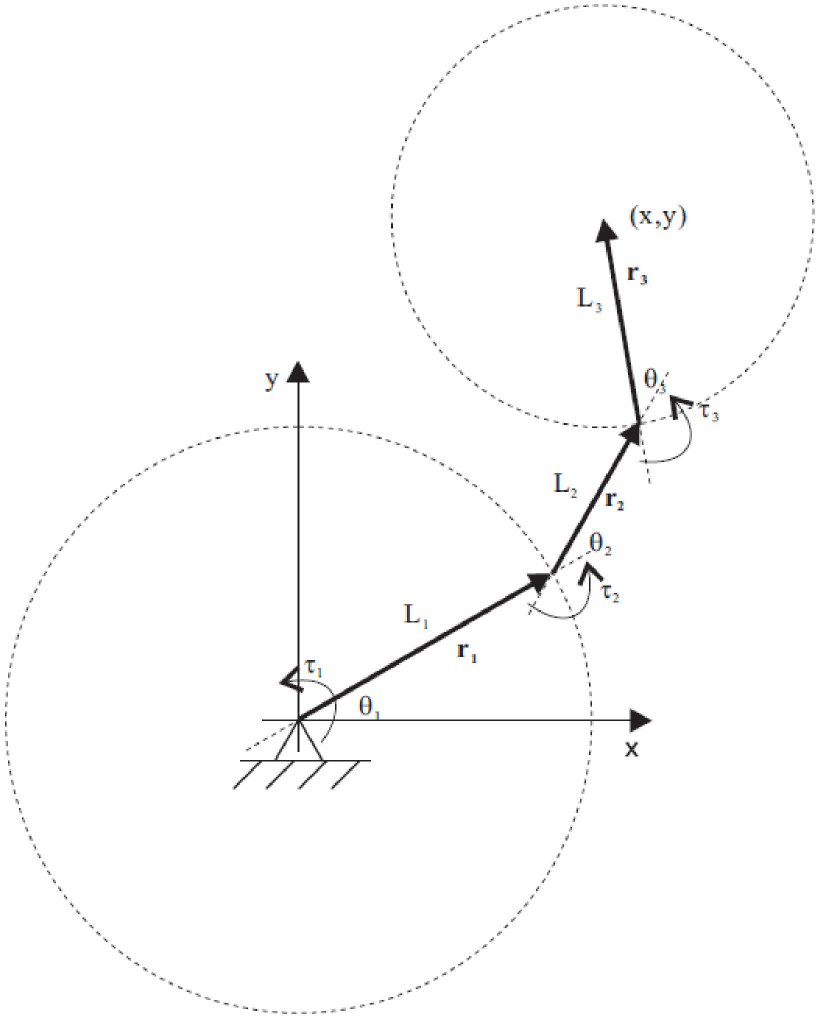
Families of IK Solutions

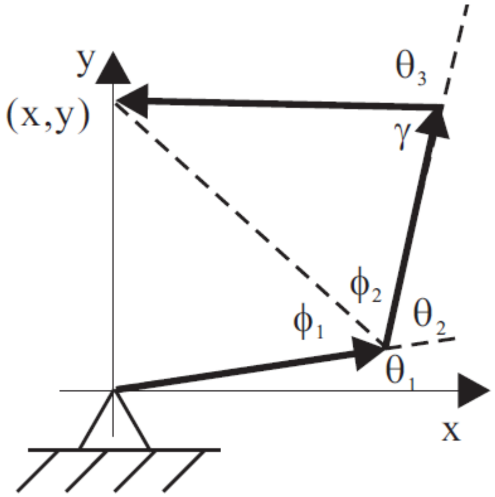


Families	Range of joint angles	Reachable range (along y axis)
Family 1 (Fig. 10.2.A)	$\theta_1 > 0,\ \theta_2 > 0,\ \theta_3 > 0$	(0,1)
Family 2 (Fig. 10.2.B)	$\theta_1 > 0,\ \theta_2 > 0,\ \theta_3 < 0$	(0.3,1)
Family 3 (Fig. 10.2.C)	$\theta_1 > 0,\ \theta_2 < 0,\ \theta_3 > 0$	(0.3,1)

- Consider $\theta_1 > 0$.
- Family 4 – Flip Family 1 to left plane

IK Solutions for Redundant Manipulators

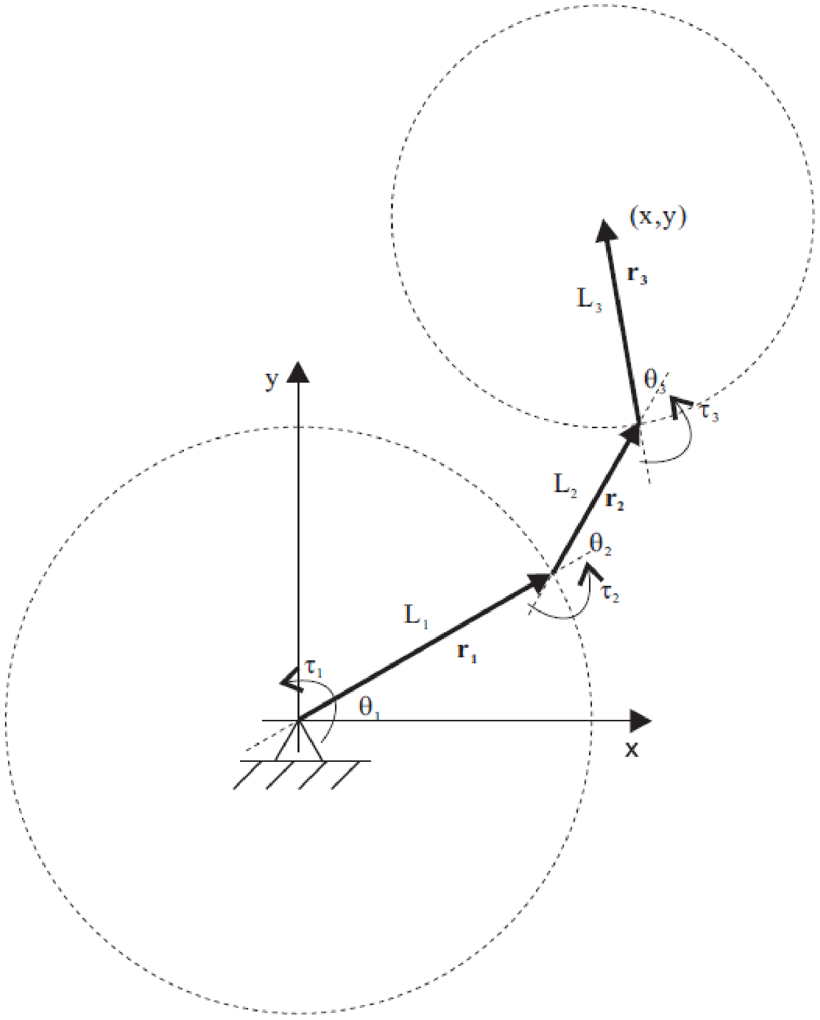


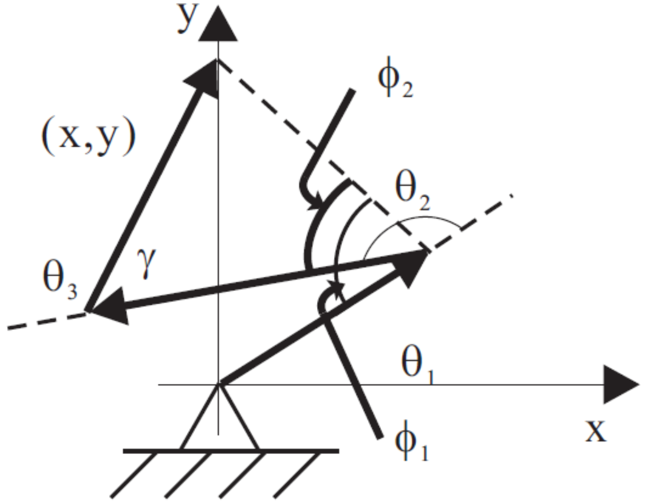


$$\theta_1 > 0, \theta_2 > 0, \theta_3 > 0$$

IK Solution – Family 1

IK Solutions for Redundant Manipulators

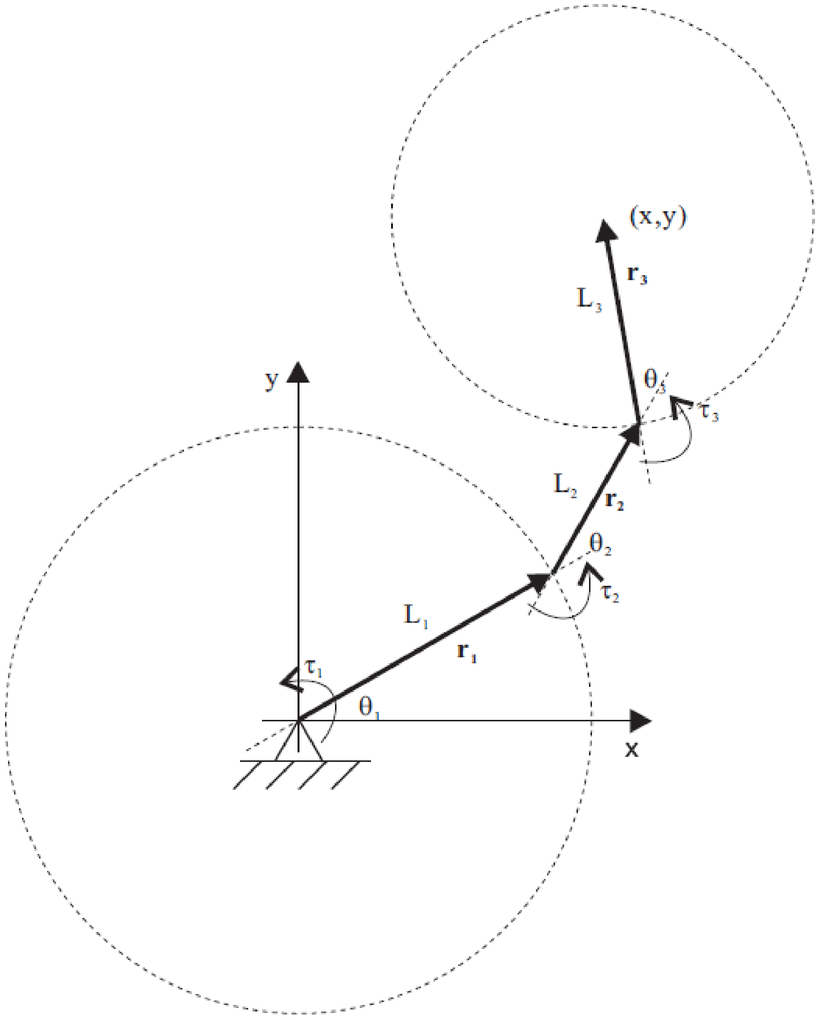


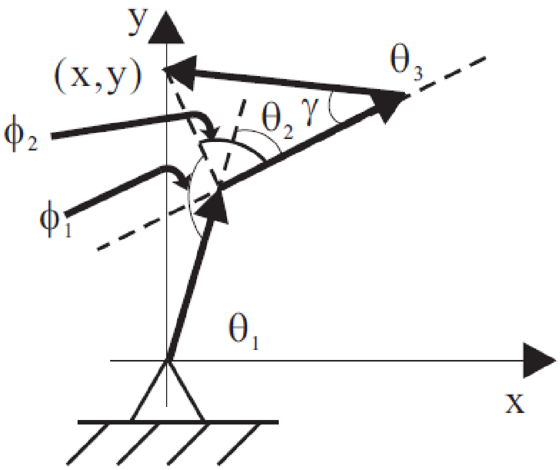


$$\theta_1 > 0, \theta_2 < 0, \theta_3 > 0$$

IK Solution – Family 2

IK Solutions for Redundant Manipulators

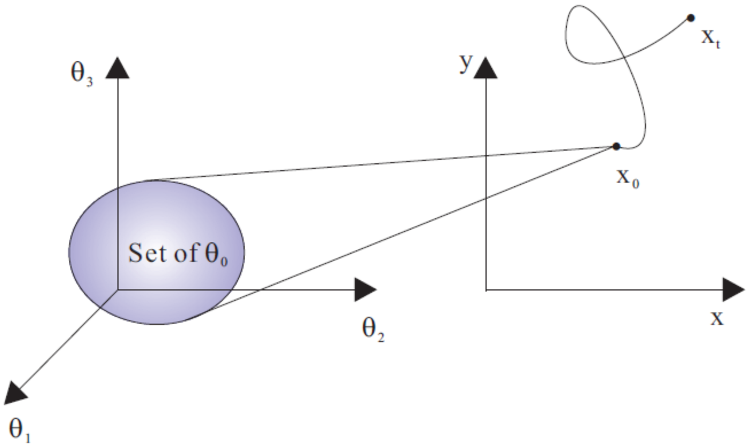


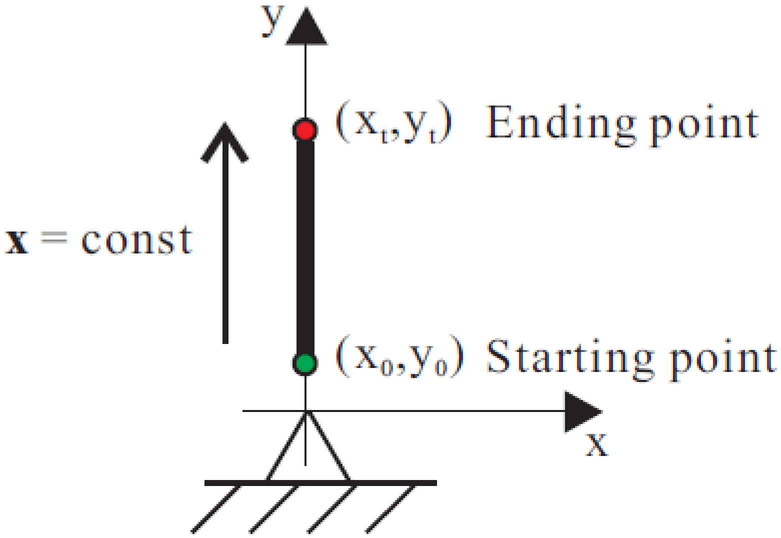


$$\theta_1 > 0, \theta_2 > 0, \theta_3 < 0$$

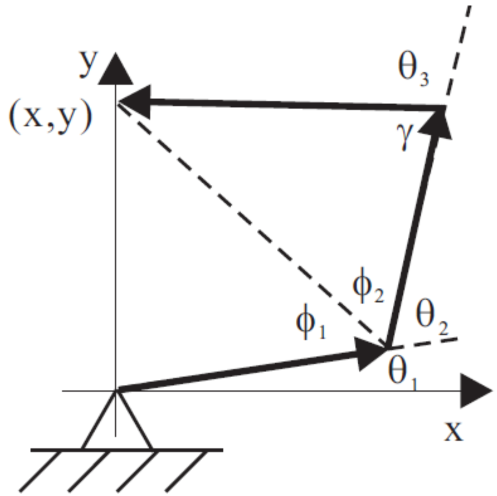
IK Solution – Family 3

IK Solutions for Redundant Manipulators



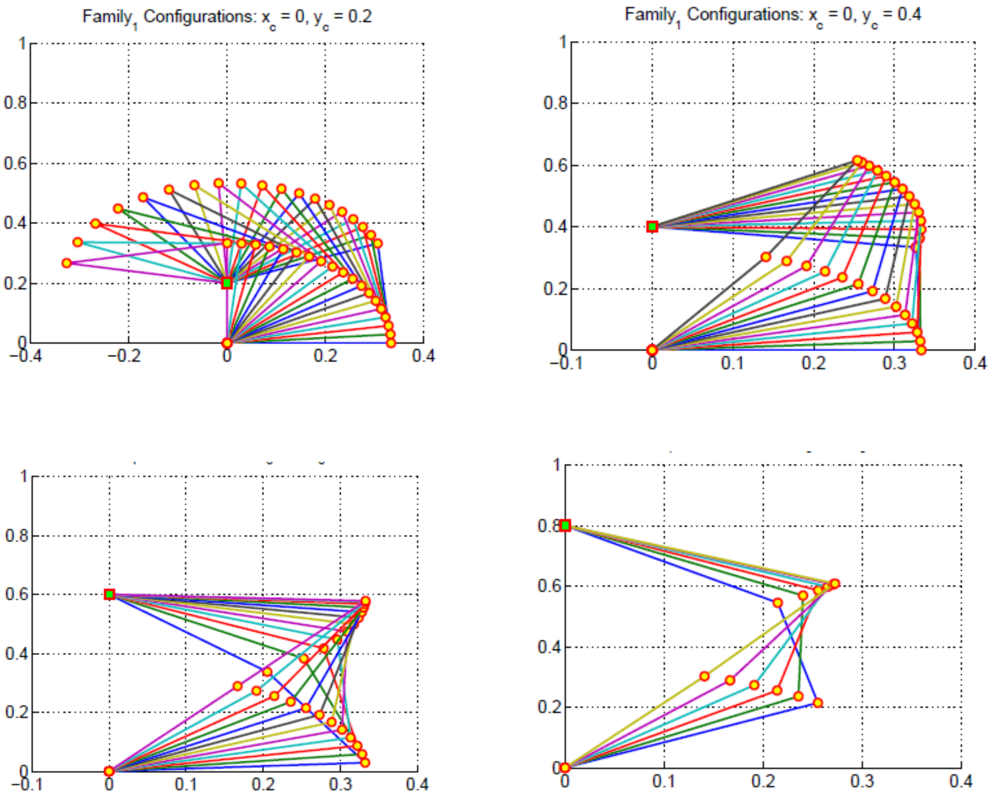


Families of IK Solutions

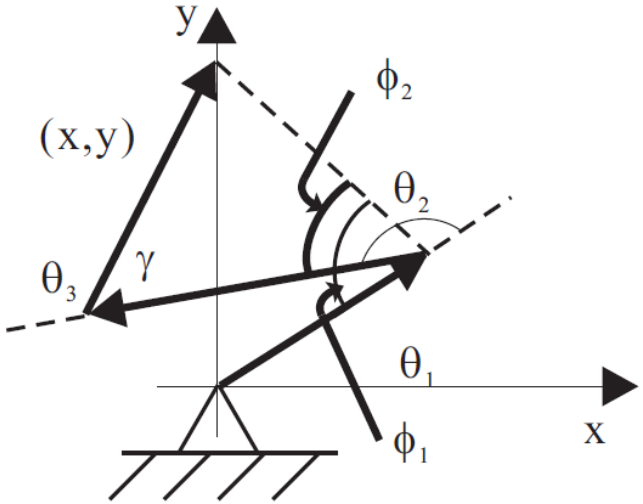


$$\theta_1 > 0, \theta_2 > 0, \theta_3 > 0$$

IK Solution – Family 1

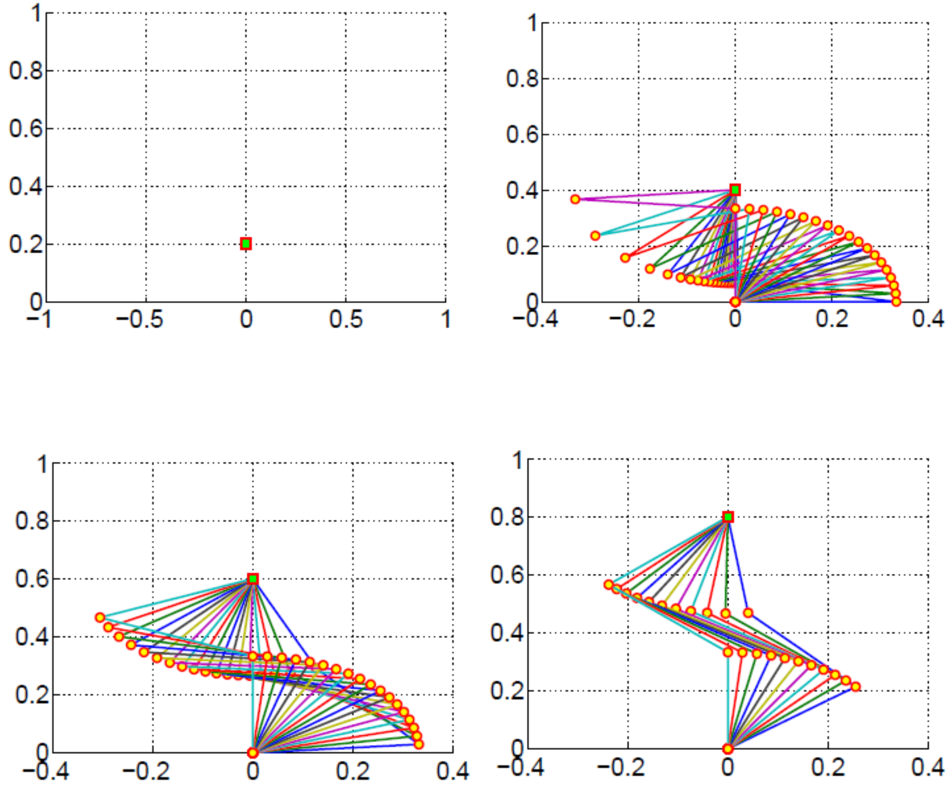


Families of IK Solutions

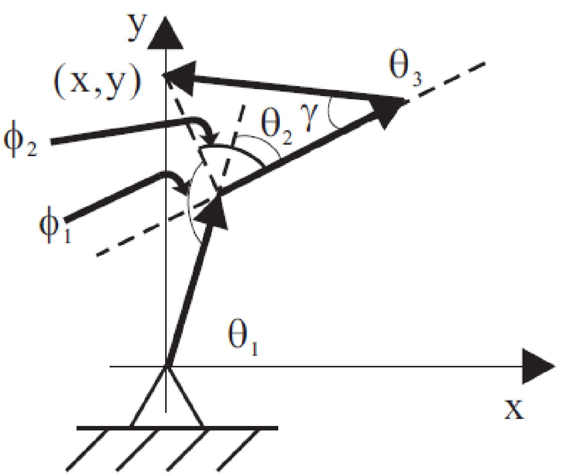


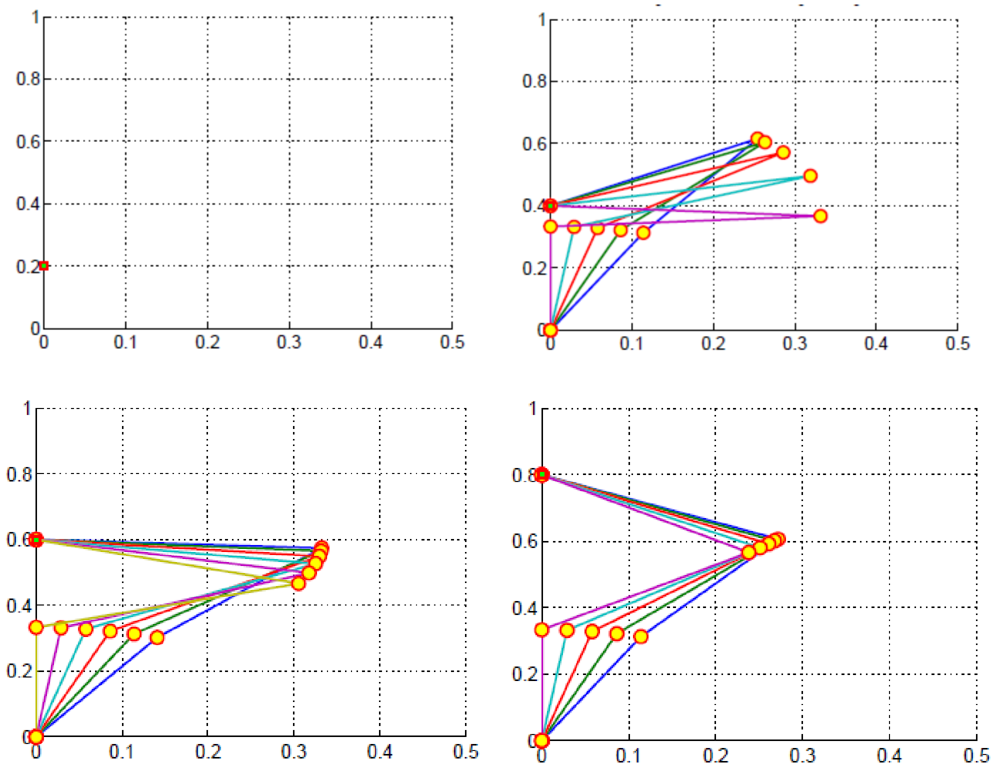
$$\theta_1 > 0, \theta_2 < 0, \theta_3 > 0$$

IK Solution – Family 2



Families of IK Solutions

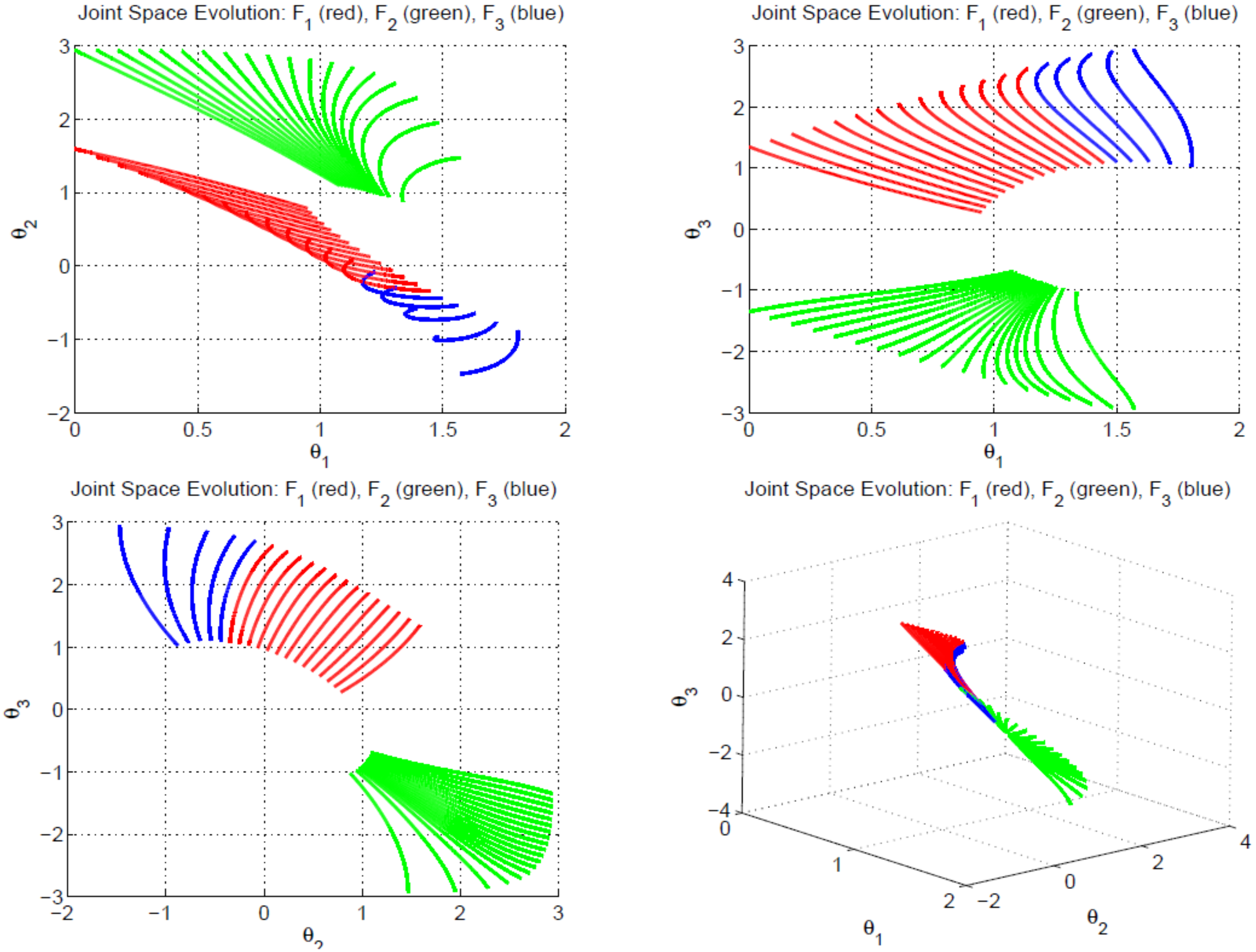




$$\theta_1 > 0, \theta_2 > 0, \theta_3 < 0$$

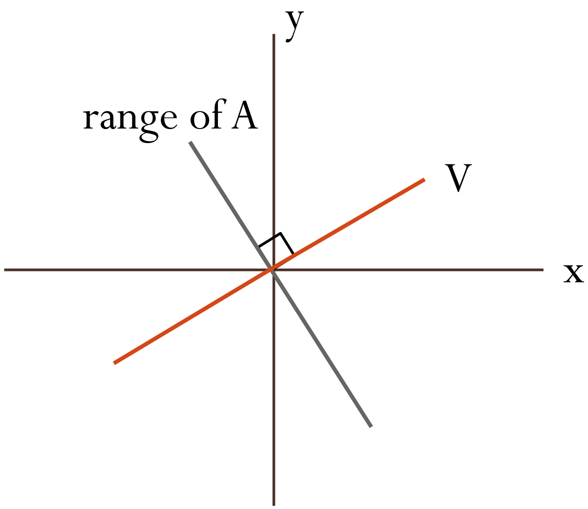
IK Solution – Family 3

Families of IK Solutions

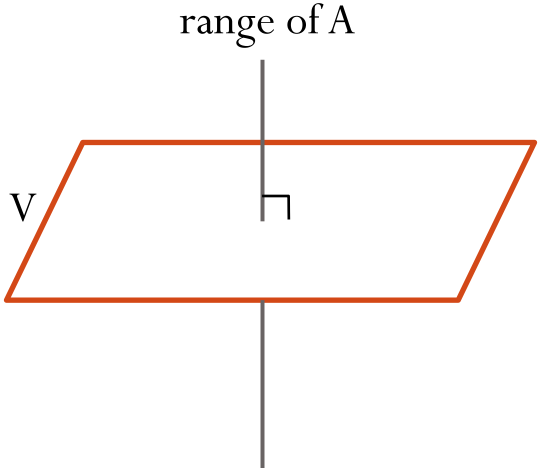


The Null-space of Jacobian

- We can try to satisfy secondary tasks in the ***null-space*** of the Jacobian pseudo-inverse
- In linear algebra, the ***null-space*** of a matrix A is the set of vectors V such that, for any v in **V**, $0 = A^T v$.
- You can prove that **V** is orthogonal to the range of A



2D example



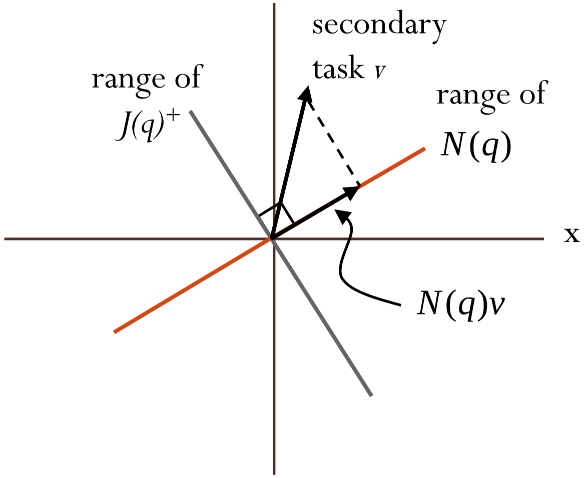
3D example

The Null-space of Jacobian

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

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

- To project a vector into the 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*

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

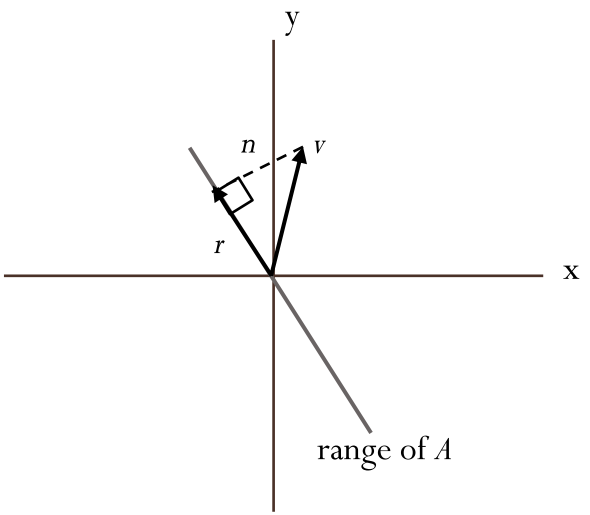
Need to find this

$$A^T n = 0$$

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

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

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



Why does this work?

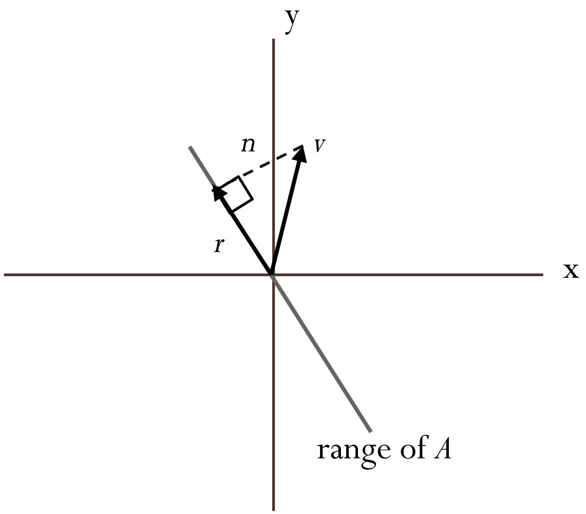
- Use this relationship to get *r*

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

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

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

$$r = AA^+ v$$



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

$$n = v - AA^+ v = \underbrace{(I - AA^+)}_{\text{This is the left null-space projection matrix}} v$$

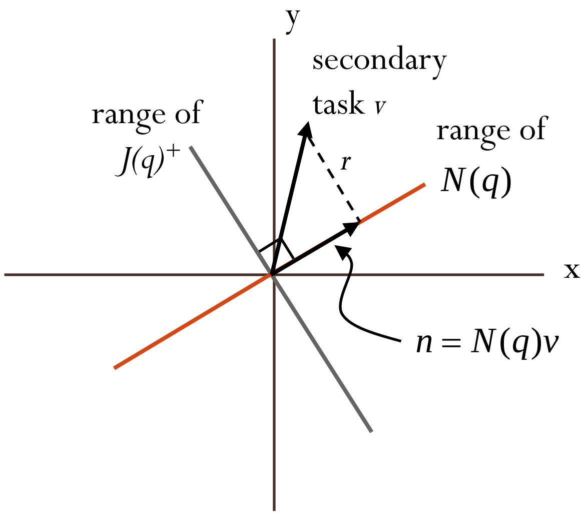
Why does this work?

- 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 :

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

- Guaranteeing 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?

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

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?

What guarantees do we have about accomplishing the secondary task?

Recursive Null-space Projection

- What if you have three or more tasks?
- The *i*th task is:

$$T_i = J_i(q)^+ \frac{dx_i}{dt}$$

- The *i*th null-space is:

$$N_i = \beta_i (I - J_i(q)^+ J_i(q))$$

- The recursive null-space formula is then:

$$\frac{dq}{dt} = T_1 + N_1(T_2 + N_2(T_3 + N_3(T_4 + \cdots N_{(n-1)} T_n)))$$

Recursive Null-space Projection

- You can do as many tasks as you want, right?

$$\frac{dq}{dt} = T_1 + N_1(T_2 + N_2(T_3 + N_3(T_4 + \cdots N_{(n-1)}T_n)))$$

- Sadly, no. Every time you go down a level, you loose degrees of freedom.

- For example, let's say we have a 6DOF manipulator. It's primary task is to place its end-effector at some 6D pose. What is the dimensionality of the null-space of this task?