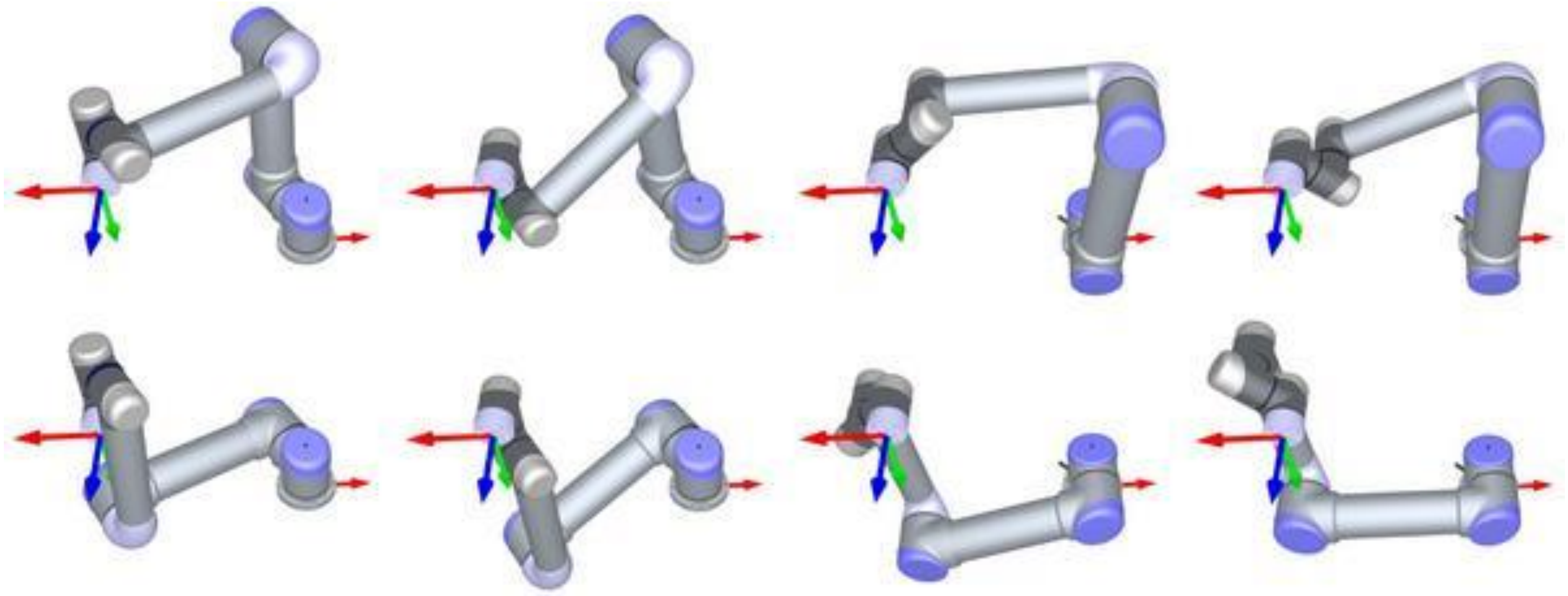
INVERSE KINEMATICS
Numerical Solution



UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Classes online

SAPIENZA
UNIVERSITÀ DI ROMA

Professor
Dipartimento di Ingegneria informatica, automatica e gestionale Antonio Ruberti - DIAG
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Sapienza Università di Roma
Via Ariosto 25, 00185 Roma, Italy

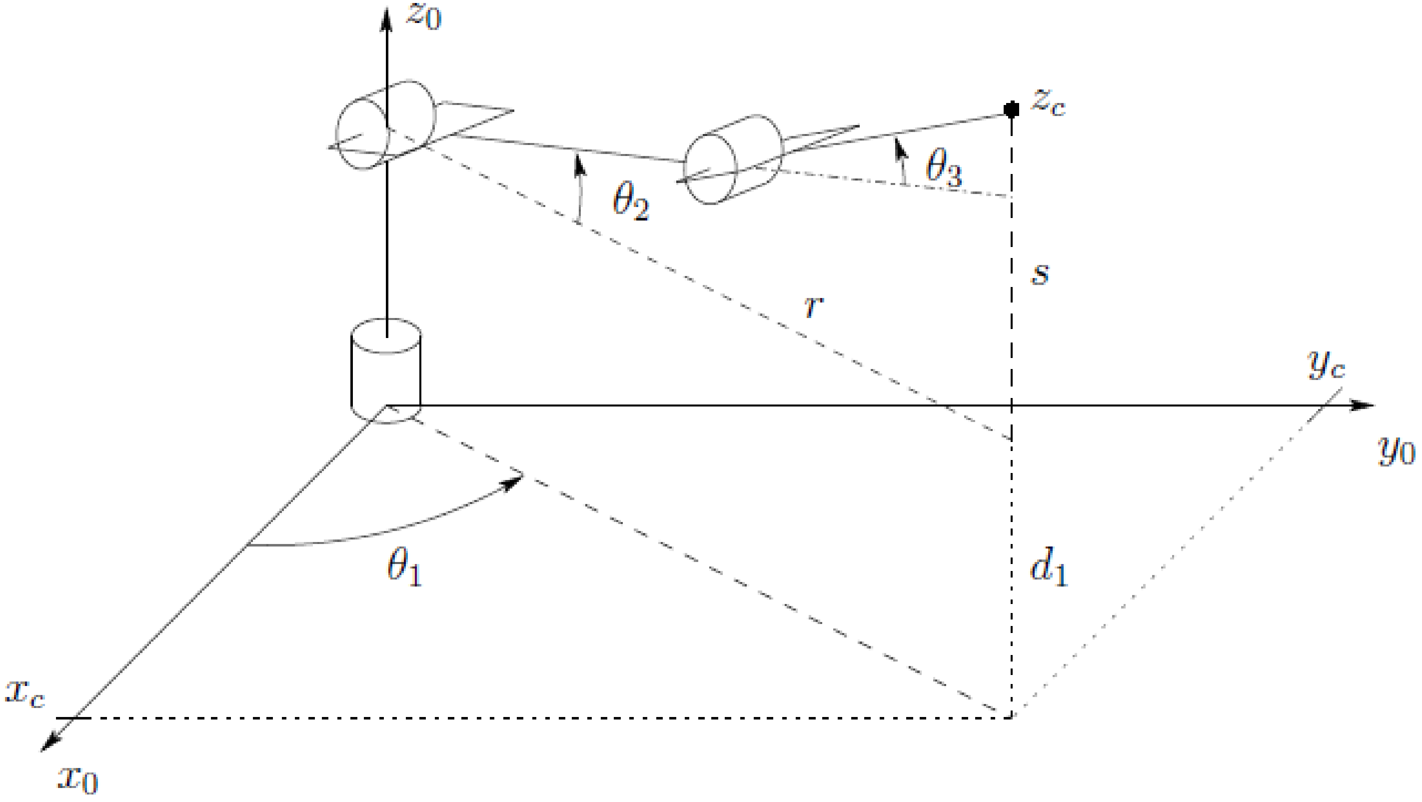
email: deluca [at] diag [dot] uniroma1 [dot] it (or a [dot] deluca [at] uniroma1 [dot] it)
room A-210, tel +39 06 77274 052, fax +39 06 77274 033

Institutional homepage at my department

ISI-WoK ResearcherID: F-3835-2011; Scopus Author ID#: 7201948195; Google Scholar: my citations
ORCID: 0000-0002-0713-5608

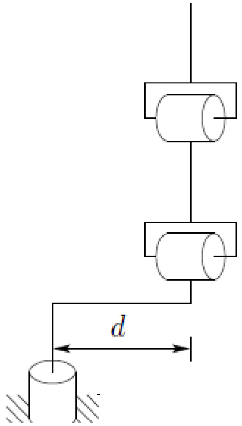
https://www.youtube.com/playlist?list=PLAQopGWlIcyaqDBW1zSKx7lHfVcOmWSWt

Is it the same robot?



Inverse Position: A Geometric Approach EX (offset *d*) : finding θ_1

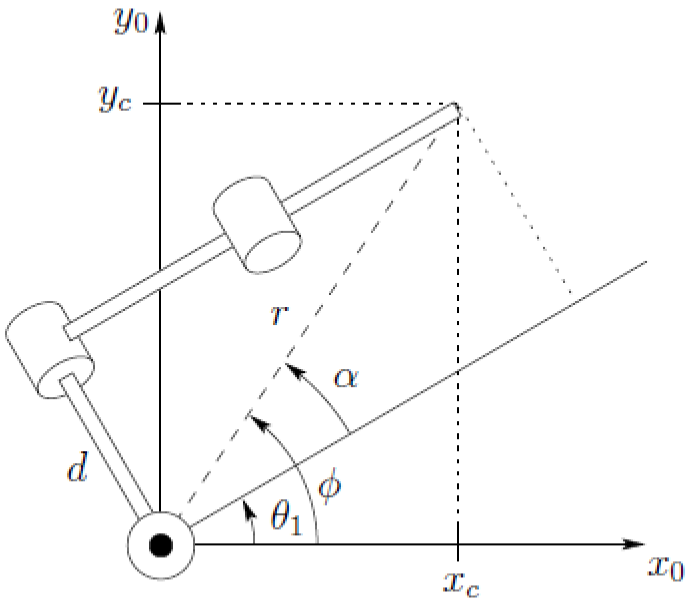
Articulated Configuration

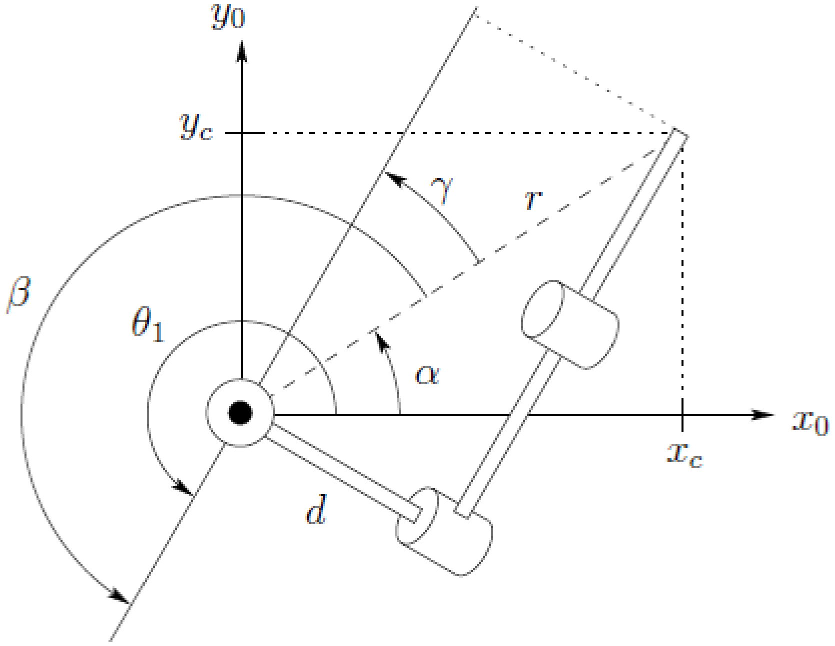


Elbow manipulator with shoulder offset.

In this case, there will, in general, be only two solutions for θ_1.

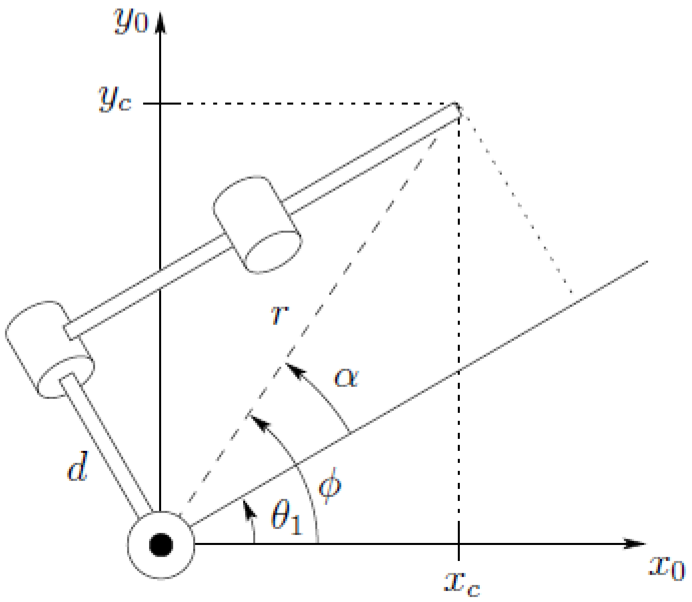
the so-called **left arm** and **right arm** configurations





Inverse Position: A Geometric Approach
EX 3 (<u>the robotic arm has offset *d*</u>) : finding θ_1

Articulated Configuration



Right Arm Configuration

$\theta_1 =$
$\phi =$
$\alpha =$

$$\theta_1 = \phi - \alpha$$

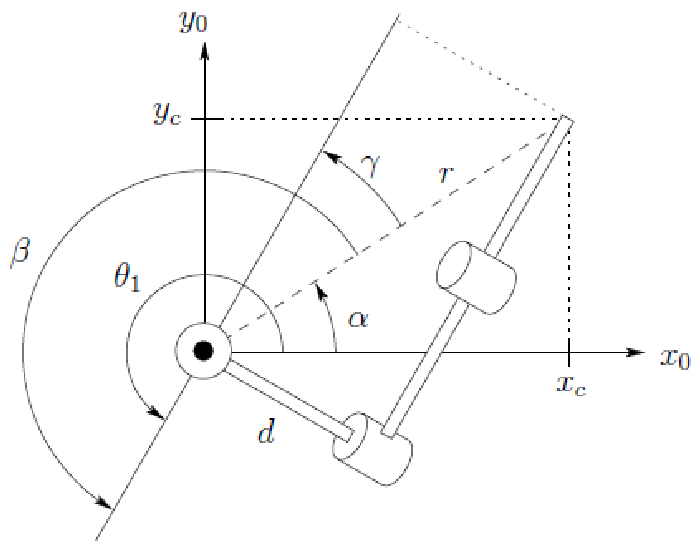
$$\phi = A\tan(x_c, y_c)$$

$$\alpha = A\tan\left(\sqrt{r^2 - d^2}, d\right) = A\tan\left(\sqrt{x_c^2 + y_c^2 - d^2}, d\right)$$

Refer to the same procedure of previous slides (articulated elbow) for the missing angles θ_2 *and* θ_3

Inverse Position: A Geometric Approach
EX 3 (offset *d*) : finding θ_1

Articulated Configuration



Left Arm Configuration

$$\theta_1 = A\tan(x_c, y_c) + A\tan\left(-\sqrt{r^2 - d^2}, -d\right).$$

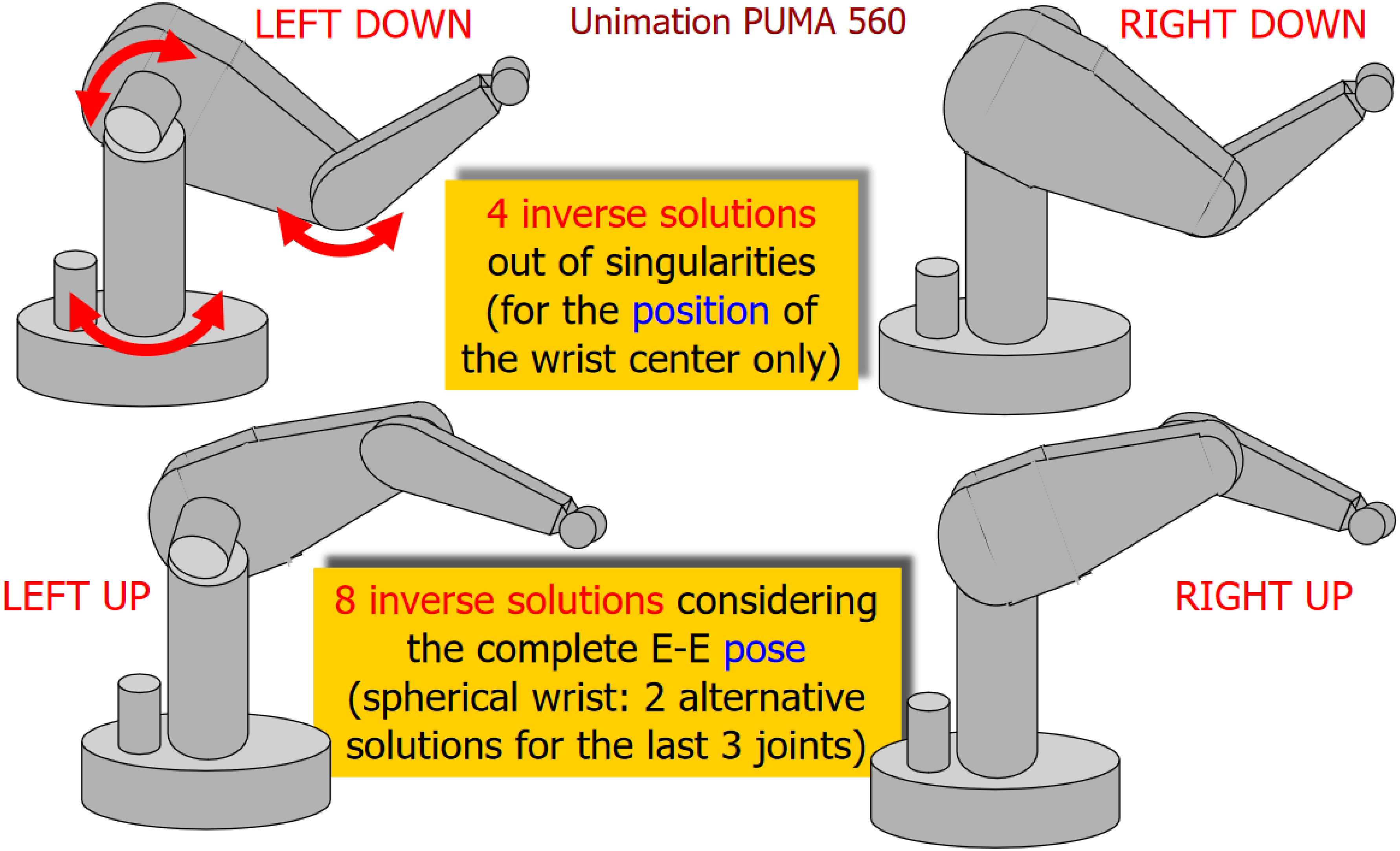
To see this, note that

$$\begin{aligned}
\theta_1 &= \alpha + \beta \\
\alpha &= A\tan(x_c, y_c) \\
\beta &= \gamma + \pi \\
\gamma &= A\tan(\sqrt{r^2 - d^2}, d)
\end{aligned}$$

which together imply that $\quad \beta = A\tan\left(-\sqrt{r^2 - d^2}, -d\right)$

since $\cos(\theta + \pi) = -\cos(\theta)$ and $\sin(\theta + \pi) = -\sin(\theta)$.

inverse solutions for an articulated 6R robot

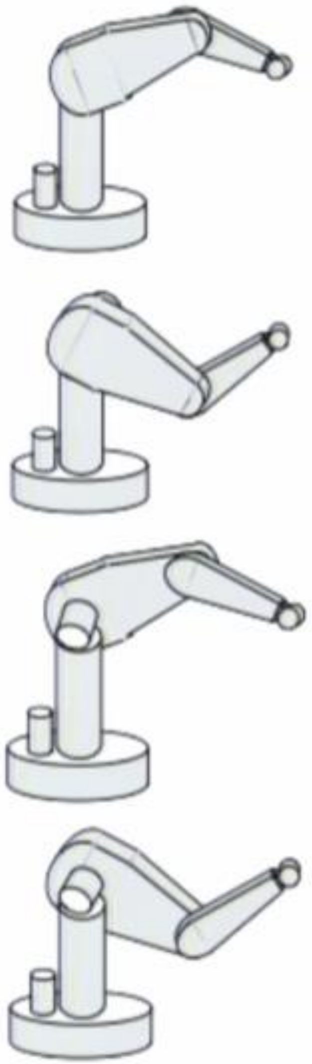


4 inverse solutions
out of singularities
(for the position of
the wrist center only)

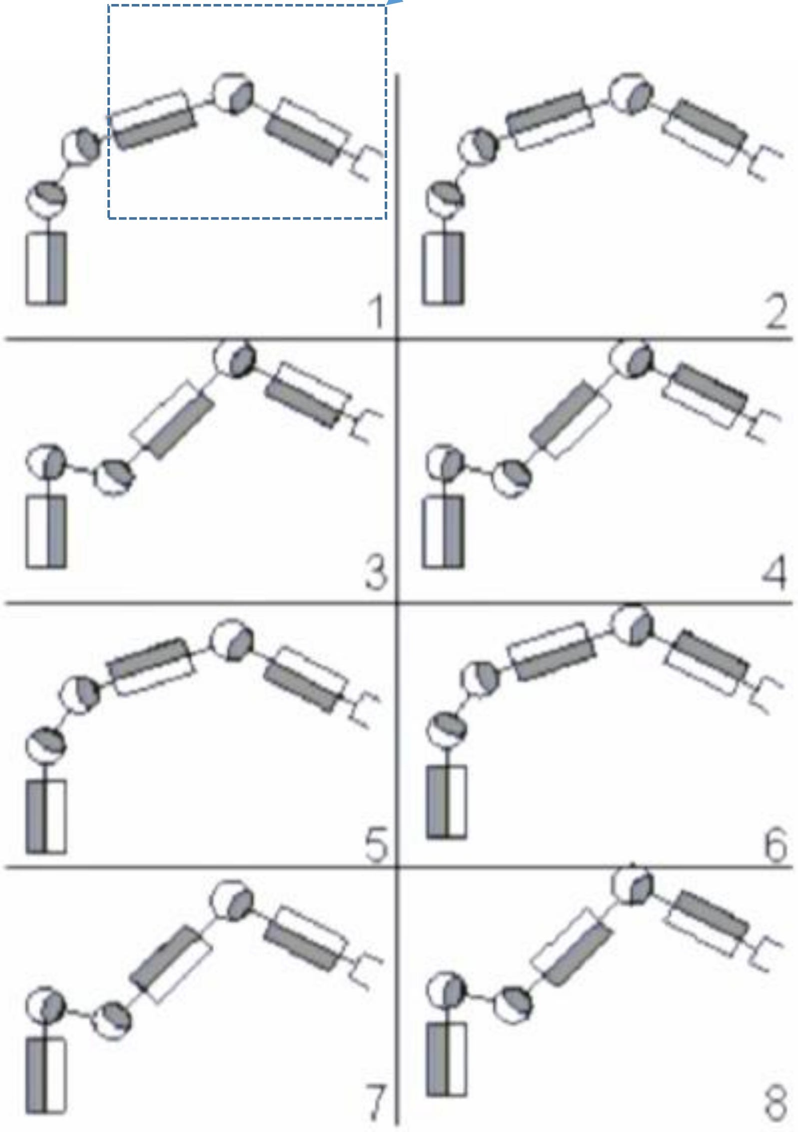
8 inverse solutions considering
the complete E-E pose
(spherical wrist: 2 alternative
solutions for the last 3 joints)

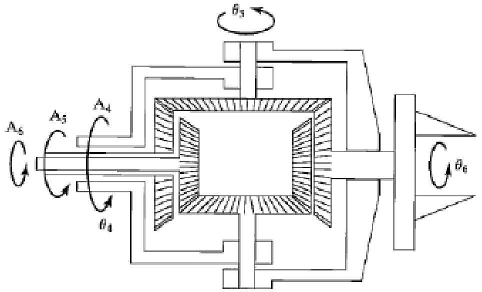
UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

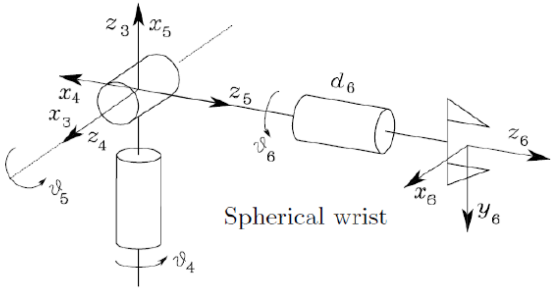
Why 8 solutions?



Spherical Wrist







shoulderRight
wristDown
elbowUp

$$q = \begin{pmatrix} 1.0472 \\ -1.2833 \\ -0.7376 \\ -2.6915 \\ -1.5708 \\ 3.1416 \end{pmatrix}$$

shoulderRight
wristDown
elbowDown

$$q = \begin{pmatrix} 1.0472 \\ -1.9941 \\ 0.7376 \\ 2.8273 \\ -1.5708 \\ 3.1416 \end{pmatrix}$$

shoulderRight
wristUp
elbowUp

$$q = \begin{pmatrix} 1.0472 \\ -1.5894 \\ -0.5236 \\ 0.5422 \\ 1.5708 \\ 0 \end{pmatrix}$$

shoulderRight
wristUp
elbowDown

$$q = \begin{pmatrix} 1.0472 \\ -2.0944 \\ 0.5236 \\ 0 \\ 1.5708 \\ 0 \end{pmatrix}$$

shoulderLeft
wristDown
elbowDown

$$q = \begin{pmatrix} 2.7686 \\ -1.0472 \\ -0.5236 \\ 3.1416 \\ -1.5708 \\ 1.4202 \end{pmatrix}$$

shoulderLeft
wristDown
elbowUp

$$q = \begin{pmatrix} 2.7686 \\ -1.5522 \\ 0.5236 \\ 2.5994 \\ -1.5708 \\ 1.4202 \end{pmatrix}$$

shoulderLeft
wristUp
elbowDown

$$q = \begin{pmatrix} 2.7686 \\ -1.1475 \\ -0.7376 \\ 0.3143 \\ 1.5708 \\ -1.7214 \end{pmatrix}$$

shoulderLeft
wristUp
elbowUp

$$q = \begin{pmatrix} 2.7686 \\ -1.8583 \\ 0.7376 \\ -0.4501 \\ 1.5708 \\ -1.7214 \end{pmatrix}$$

Solution methods

ANALYTICAL solution (in closed form) ⇄ **NUMERICAL solution** (in iterative form)

ANALYTICAL solution
(in closed form)

- preferred, if it can be found*
- use ad-hoc geometric inspection
- algebraic methods (solution of polynomial equations)
- systematic ways for generating a reduced set of equations to be solved

* sufficient conditions for 6-dof arms
- 3 consecutive rotational joint axes are incident (e.g., spherical wrist), or
- 3 consecutive rotational joint axes are parallel

D. Pieper, PhD thesis, Stanford University, 1968

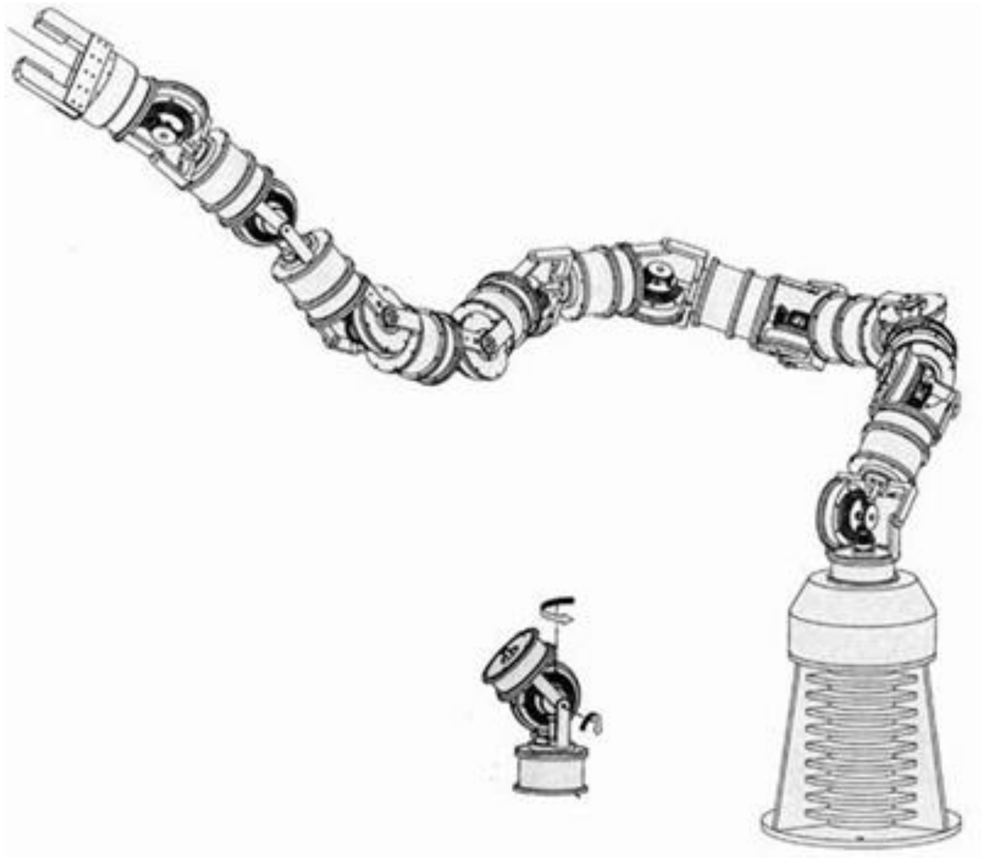
NUMERICAL solution
(in iterative form)

- certainly needed if n>m (redundant case), or at/close to singularities
- slower, but easier to be set up
- in its basic form, it uses the (analytical) Jacobian matrix of the direct kinematics map

$$J_r(q) = \frac{\partial f_r(q)}{\partial q}$$

- Newton method, Gradient method, and so on…

Numerical Methods. When using it?



NUMERICAL solution
(in iterative form)

- certainly needed if n>m (redundant case), or at/close to singularities
- slower, but easier to be set up
- in its basic form, it uses the (analytical) Jacobian matrix of the direct kinematics map

$$J_r(q) = \frac{\partial f_r(q)}{\partial q}$$

- Newton method, Gradient method, and so on...

Numerical approach

- use when a closed-form solution q to $r_d = f_r(q)$ does not exist or is "too hard" to be found
- $J_r(q) = \frac{\partial f_r}{\partial q}$ (analytical Jacobian)

Definition

[Jacobian matrix]

The Jacobian matrix of the forward kinematics mapping at a given configuration $\boldsymbol{q}_0$ is defined by:

$$\boldsymbol{J}(\boldsymbol{q}_0) := \left.\frac{\partial \mathrm{FK}(\boldsymbol{q})}{\partial \boldsymbol{q}}\right|_{\boldsymbol{q}=\boldsymbol{q}_0}$$

In the case of the planar 2-DOF manipulator, one has

$$\boldsymbol{J}(\theta_1, \theta_2) = \begin{pmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} \\ \frac{\partial \theta}{\partial \theta_1} & \frac{\partial \theta}{\partial \theta_2} \end{pmatrix} = \begin{pmatrix} -d_1 \sin(\theta_1) - d_2 \sin(\theta_1 + \theta_2) & -d_2 \sin(\theta_1 + \theta_2) \\ d_1 \cos(\theta_1) + d_2 \cos(\theta_1 + \theta_2) & d_2 \cos(\theta_1 + \theta_2) \\ 1 & 1 \end{pmatrix}$$

Numerical method 1: Newton

- $r = \begin{bmatrix} p \\ \phi \end{bmatrix} = f_r(q)$, or for any other task vector $\qquad r_d = f_r(q)$

It is a linearization using a Taylor series expansion.

- Newton method (here for m=n)
  - $r_d = f_r(q) = f_r(q^k) + J_r(q^k)\,(q - q^k) + o(\|q - q^k\|^2)$ ← neglected

The Jacobian is evaluated in the $\mathbf{q}^k$ which is the iteration number **k** and not the **k**th component of the joint variable vector.
What we obtained is a linear function in **q**.

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

How do we solve? Next q^{k+1}

$r_d = f_r(q) = f_r(q^k) + J_r(q^k)\,(q - q^k)$ We extract q and we have to invert the jacobian to find it.

$q^{k+1} = q^k + J_r^{-1}(q^k)\,[r_d - f_r(q^k)]$ We call the solution q^{k+1}

This term is the **Error**: the difference between $\mathbf{r_d}$ which is where the robot should be and the
actual position evaluated by the direct kinematics $\mathbf{f_r(q)}$ in $\mathbf{q^k}$

If the Error is zero we have solved our problem and because $q^{k+1}= q^k$

Otherwise we continue the iteration until and whenever a convergence is found

When it works and does not?

- convergence if q^0 (initial guess) is close enough to some q^*: $f_r(q^*) = r_d$

Convergence is assured if my initial set of joint coordinates (guess) $\mathbf{q}^{\mathbf{0}}$ is close to a possible solution or of or the possible solutions for the desired task $\mathbf{r}_{\mathbf{d}}$

- problems near singularities of the Jacobian matrix $J_r(q)$

$$q^{k+1} = q^k + J_r^{-1}(q^k)\,[r_d - f_r(q^k)]$$

$$M^{-1} = \frac{1}{\det(M)} \times \mathrm{Adj}(M)$$

If a work with the algorithm in a $\mathbf{q}^{k}$ which is close to a singular configuration, when my jacobian is loosing rank (det(J)) close to zero), then the algorithm becomes unstable because I am multiplying the error for a large number.

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

When it works and does not?

- in case of robot redundancy (m<n), use the pseudo-inverse $J_r^{\#}(q)$

Introduce the pseudo inverse

Overactuated arm

- Consider a robot with 100 joints

6x100

$$\begin{pmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = \begin{pmatrix} J_{1,1} & J_{1,2} & \cdots & J_{1,100} \\ J_{2,1} & J_{2,2} & \cdots & J_{2,100} \\ \vdots & \vdots & \ddots & \cdots \\ J_{6,1} & J_{6,2} & \cdots & J_{6,100} \end{pmatrix} \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_{100} \end{pmatrix}$$

$N > 6$

- It cannot be inverted

How it works this method?

- in the scalar case, also known as "method of the tangent"
- for a differentiable function f(x), find a root of $f(x^*)=0$ by iterating as

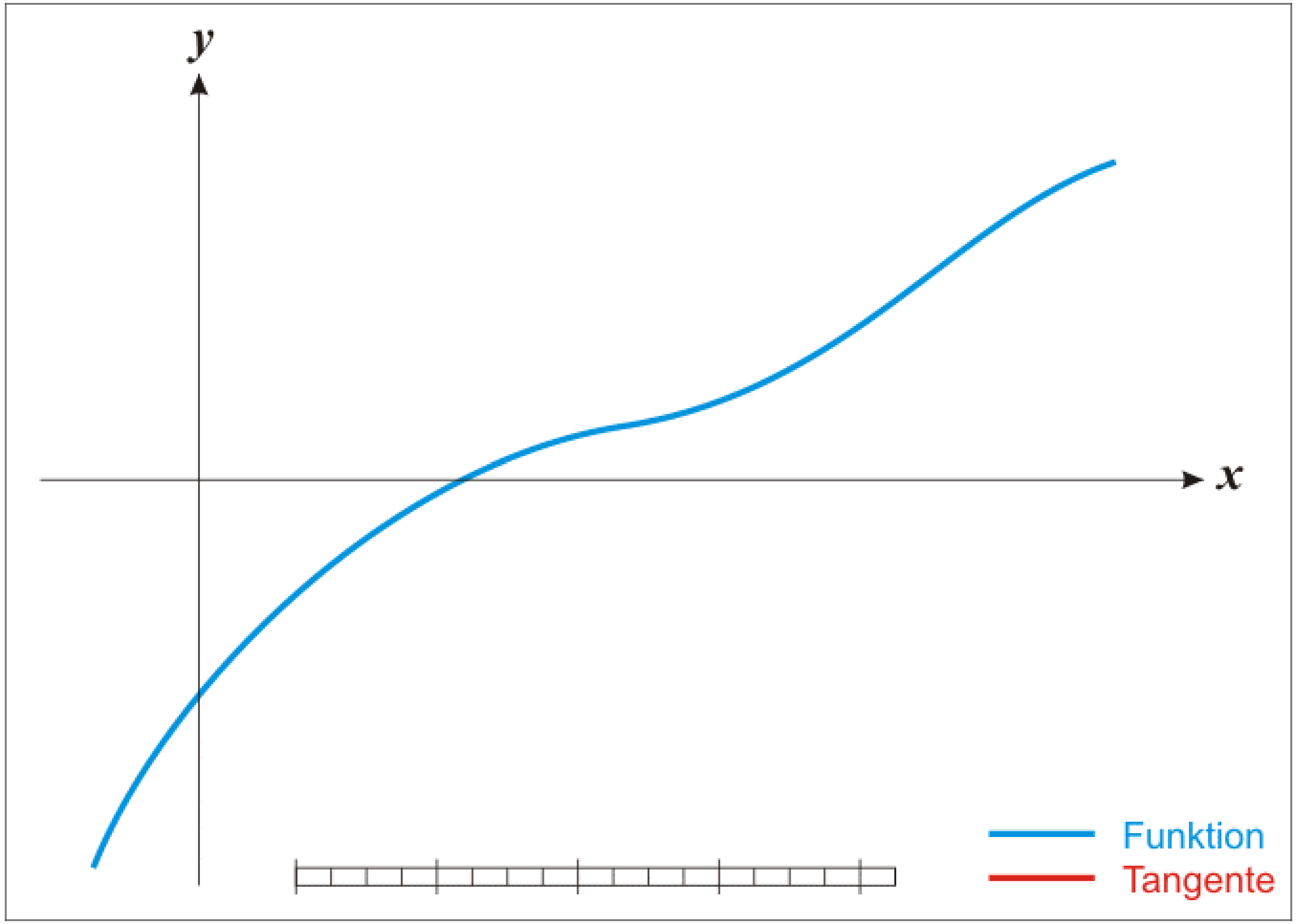
$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

an approximating sequence

$$\{x_1, x_2, x_3, x_4, x_5, \ldots\} \rightarrow x^*$$

In case of multidimensional calculus

$$q^{k+1} = q^k + J_r^{-1}(q^k)\,[r_d - f_r(q^k)]$$



https://en.wikipedia.org/wiki/File:NewtonIteration_Ani.gif

Numerical method 2: Gradient

- Gradient method (max descent)
  - minimize the error function

$$H(q) = \frac{1}{2} \left\| r_d - f_r(q) \right\|^2 = \frac{1}{2} [r_d - f_r(q)]^T [r_d - f_r(q)]$$

In multidimensional problem we want to find the gradient which is zero and there fore we are in a point of minimum, where the difference between the $\mathbf{f_r(q)}$ and $\mathbf{r_d}$ is close to zero.

We choose the norm of *H(q)*: it always positive or zero when we have solution.

The method is the following: $q^{k+1} = q^k - \alpha \nabla_q H(q^k)$

$$\nabla_q H(q) = - J_r^T(q) [r_d - f_r(q)]$$

$$q^{k+1} = q^k + \alpha J_r^T(q^k) [r_d - f_r(q^k)]$$

The iteration to be set up is this with α which is the iteration step.

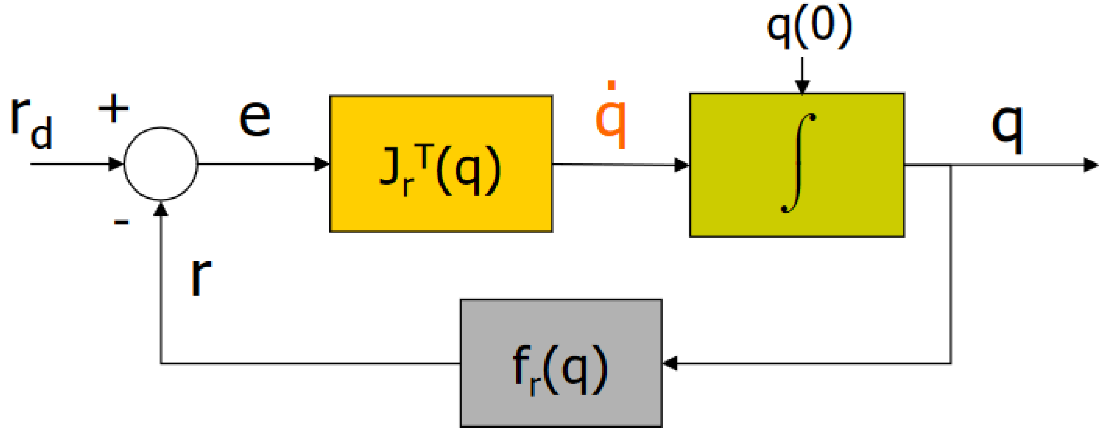
Numerical method 2: Gradient

$$q^{k+1} = q^k + \alpha\, J_r^T(q^k)\, [r_d - f_r(q^k)]$$

- the scalar step size $\alpha > 0$ should be chosen so as to guarantee a decrease of the error function at each iteration (too large values for α may lead the method to "miss" the minimum)
- when the step size α is too small, convergence is extremely slow

The good advantage respect to the Newton method is that we don t use an inverse of the Jacobian and therefore there is no danger of unstable computations.

Revisited as a “feedback” scheme



r_d = cost

(α = 1)

$e = r_d - f_r(q) \rightarrow 0 \;\Leftrightarrow$ closed-loop equilibrium e=0 is asymptotically stable

$V = ½\, e^T e \geq 0$ Lyapunov candidate function

$$\dot{V} = e^T \dot{e} = e^T \frac{d}{dt}(r_d - f_r(q)) = -e^T J_r\, \dot{q} = -e^T J_r J_r^T e \leq 0$$

$\dot{V} = 0 \;\Leftrightarrow\; e \in Ker(J_r^T)$ in particular $e = 0$

asymptotic stability

Properties of Gradient method

- computationally simpler: Jacobian transpose, rather than its (pseudo)-inverse
- direct use also for robots that are redundant for the task
- may not converge to a solution, but it never diverges

- the discrete-time evolution of the continuous scheme

$$q^{k+1} = q^k + \Delta T\, J_r^T(q^k)\, [r_d - f(q^k)] \qquad (\alpha = \Delta T)$$

is equivalent to an iteration of the Gradient method

- scheme can be accelerated by using a gain matrix K>0

$$\dot{q} = J_r^T(q)\, K\, e$$

note: K can be used also to "escape" from being stuck in a stationary point, by rotating the error e out of the kernel of J_r^T (if a singularity is encountered)

Case Study Newton Vs Gradient

- 2R robot with $l_1 = l_2 = 1$, desired end-effector position $r_d = p_d = (1,1)$
- direct kinematic function and error

$$f_r(q) = \begin{pmatrix} c_1 + c_{12} \\ s_1 + s_{12} \end{pmatrix} \qquad e = p_d - f_r(q) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} - f_r(q)$$

- Jacobian matrix

$$J_r(q) = \frac{\partial f_r(q)}{\partial q} = \begin{pmatrix} -(s_1 + s_{12}) & -s_{12} \\ c_1 + c_{12} & c_{12} \end{pmatrix}$$

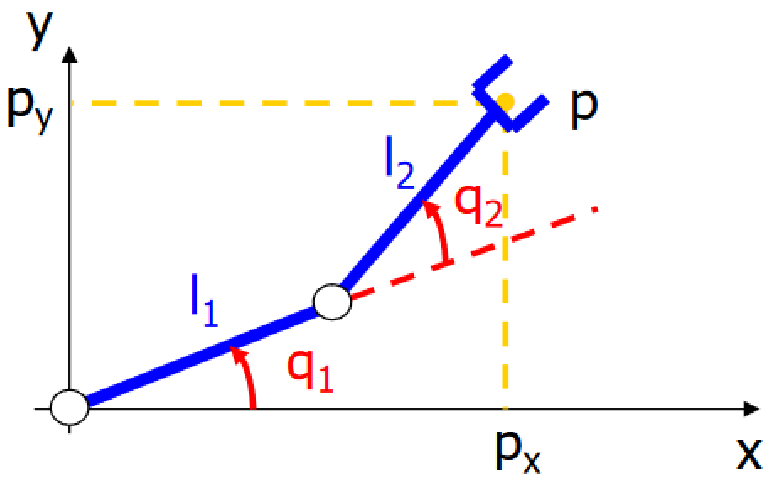
direct kinematics

$$p_x = l_1\, c_1 + l_2\, c_{12}$$

$$p_y = l_1\, s_1 + l_2\, s_{12}$$

data

q_1, q_2 unknowns

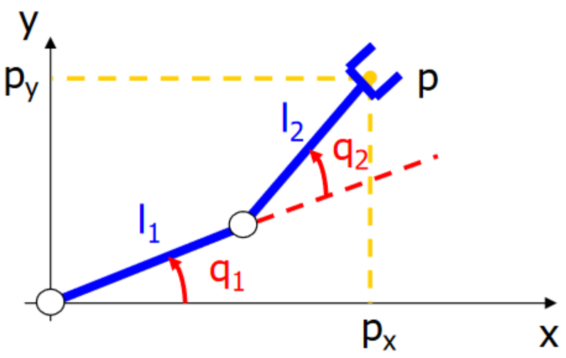


UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Case Study Newton Vs Gradient

$$q^{k+1} = q^k + J_r^{-1}(q^k)\,[r_d - f_r(q^k)]$$

$$q^{k+1} = q^k + \alpha\, J_r^T(q^k)\,[r_d - f_r(q^k)]$$



direct kinematics

$p_x = l_1 c_1 + l_2 c_{12}$

$p_y = l_1 s_1 + l_2 s_{12}$

data — q_1, q_2 unknowns

- Newton versus Gradient iteration

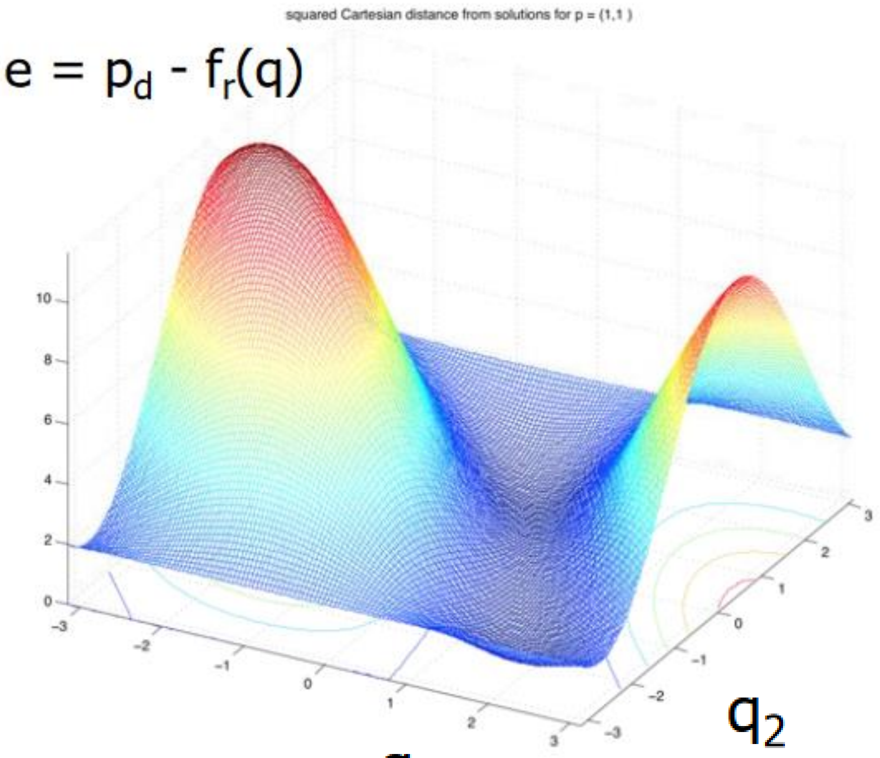
$J_r^{-1}(q^k)$ — det $J_r(q)$ — e_k — $J_r^T(q^k)$ — $r_d - f_r(q)$

$$q^{k+1} = q^k + \left\{ \begin{array}{l} \dfrac{1}{s_2}\begin{pmatrix} c_{12} & s_{12} \\ -(c_1 + c_{12}) & -(s_1 + s_{12}) \end{pmatrix}\Bigg|_{q=q^k} \\ \alpha \begin{pmatrix} -(s_1 + s_{12}) & c_1 + c_{12} \\ -s_{12} & c_{12} \end{pmatrix}\Bigg|_{q=q^k} \end{array} \right\} \bullet \begin{pmatrix} 1 - (c_1 + c_{12}) \\ 1 - (s_1 + s_{12}) \end{pmatrix}\Bigg|_{q=q^k}$$

Error function

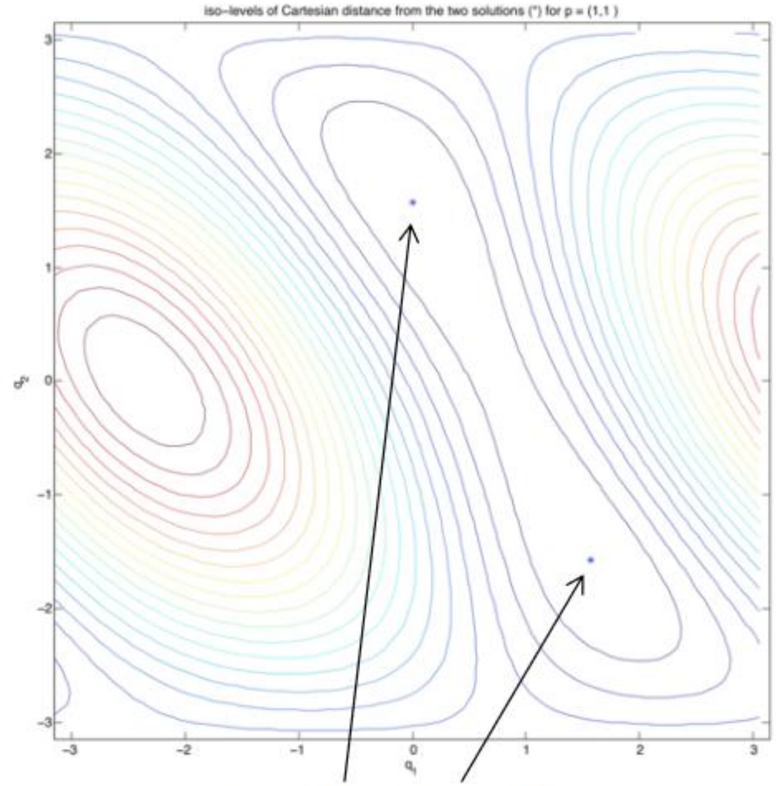
- 2R robot with $l_1=l_2=1$, desired end-effector position $p_d = (1,1)$

$e = p_d - f_r(q)$



q_1 q_2

plot of $\|e\|^2$ as a function of $q = (q_1,q_2)$

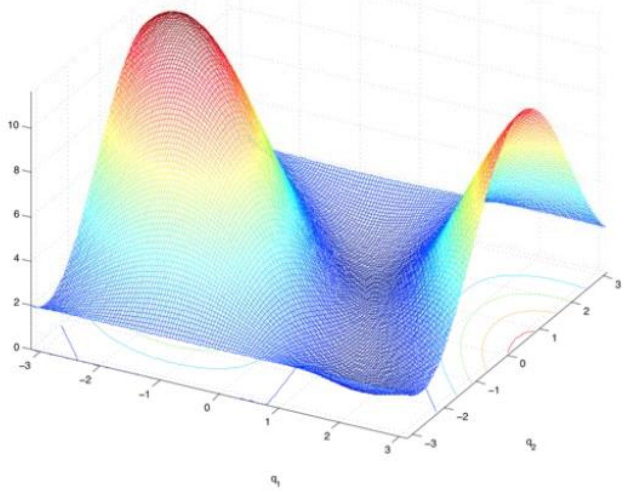


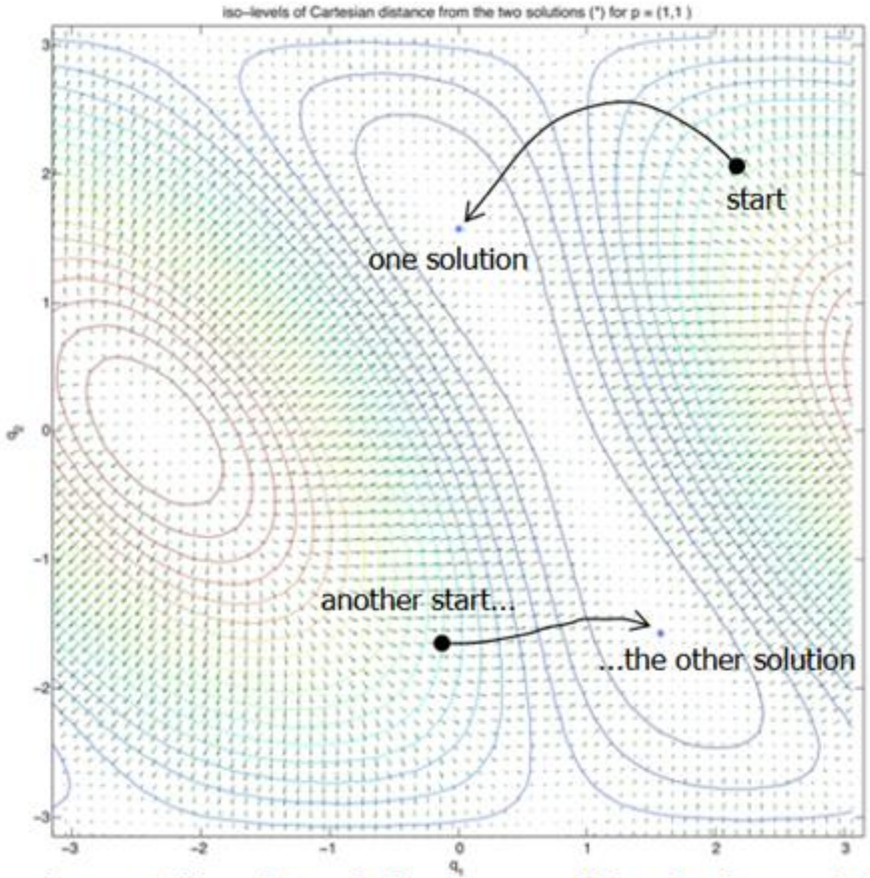
two local minima
(inverse kinematic solutions)

Error reduction by Gradient method

- flow of iterations along the negative (or anti-) gradient

$e = p_d - f_r(q)$

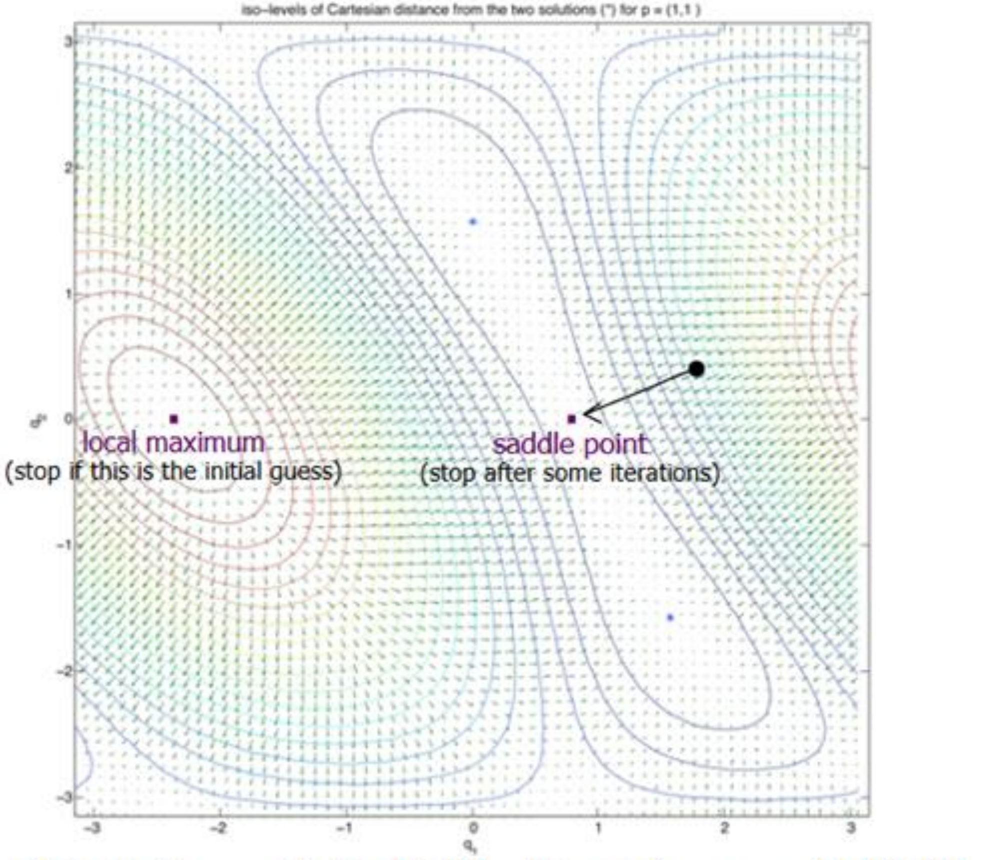




$(q_1,q_2)' = (0,\pi/2)$ $(q_1,q_2)'' = (\pi/2,-\pi/2)$

Error reduction by Gradient method

- two possible cases: convergence or stuck (at zero gradient)



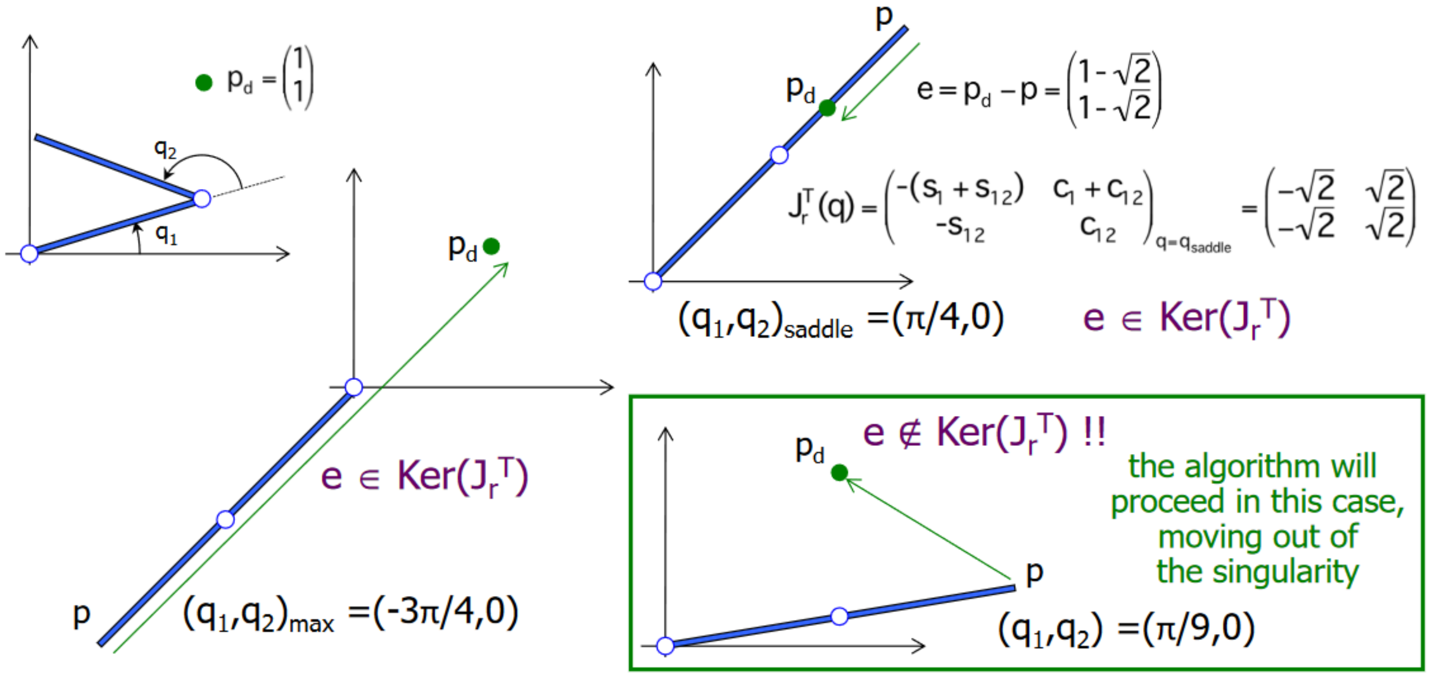
$(q_1,q_2)_{max} = (-3\pi/4,0)$ $(q_1,q_2)_{saddle} = (\pi/4,0)$

$e \in Ker(J_r^T)$!

Convergence analysis

when does the gradient method get stuck?

- lack of convergence occurs when
  - the Jacobian matrix $J_r(q)$ is singular (the robot is in a "singular configuration")
  - **AND** the error is in the "null space" of $J_r^T(q)$



Issues in implementation

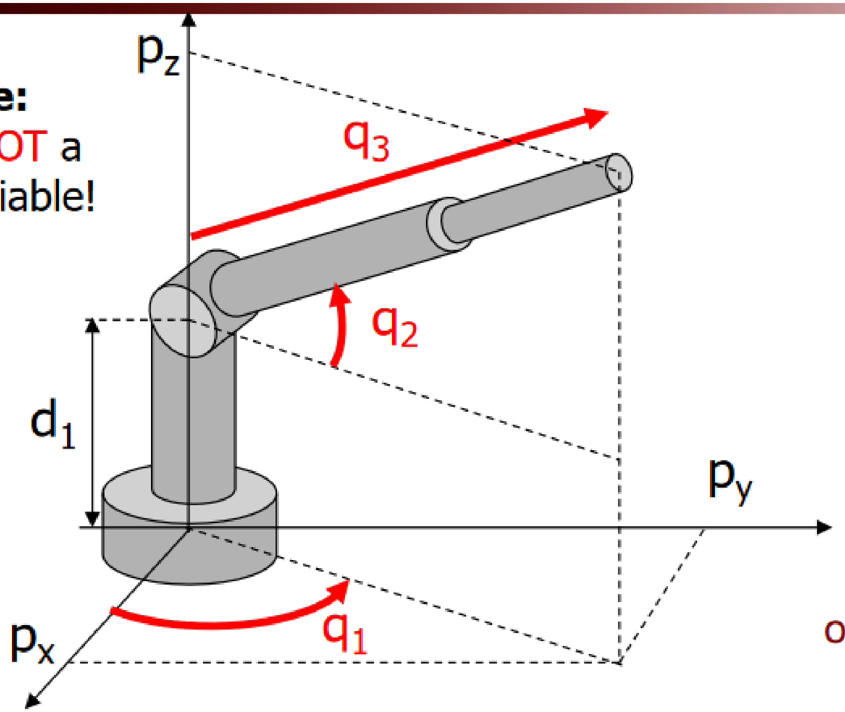
- initial guess q^0
  - only one inverse solution is generated for each guess
  - multiple initializations for obtaining other solutions
- optimal step size α in Gradient method
  - a constant step may work good initially, but not close to the solution (or vice versa)
  - an adaptive one-dimensional line search (e.g., Armijo's rule) could be used to choose the best α at each iteration
- stopping criteria

Cartesian error
(possibly, separate for position and orientation) $\|r_d - f(q^k)\| \leq \varepsilon$

algorithm increment $\|q^{k+1} - q^k\| \leq \varepsilon_q$

Inverse kinematics of polar (RRP) arm

Note:
q_2 is NOT a DH variable!



$$p_x = q_3\, c_2\, c_1$$

$$p_y = q_3\, c_2\, s_1$$

$$p_z = d_1 + q_3\, s_2$$

$$p_x^2 + p_y^2 + (p_z - d_1)^2 = q_3^2$$

$$q_3 = +\sqrt{p_x^2 + p_y^2 + (p_z - d_1)^2}$$

our choice: take here only the positive value...

if $q_3 = 0$, then q_1 and q_2 remain both undefined (stop); else

$$q_2 = \mathrm{ATAN2}\{(p_z - d_1)/q_3,\ \pm\sqrt{(p_x^2 + p_y^2)/q_3^2}\}$$

(if it stops, a singular case: ∞^2 or ∞^1 solutions)

if $p_x^2+p_y^2 = 0$, then q_1 remains undefined (stop); else

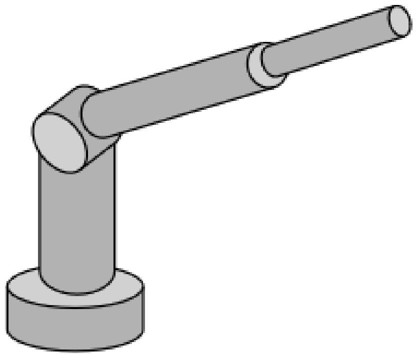
$$q_1 = \mathrm{ATAN2}\{p_y/c_2,\ p_x/c_2\}$$

(2 regular solutions $\{q_1,q_2,q_3\}$)

we have eliminated $q_3>0$ from both arguments!

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Numerical tests on RRP robot



- RRP/polar robot: desired E-E position $r_d = p_d = (1, 1, 1)$ with $d_1=0.5$
- the two (known) analytical solutions, with $q_3 \geq 0$, are:

$q^* = (0.7854, 0.3398, 1.5)$

$q^{**} = (q_1^* - \pi, \pi - q_2^*, q_3^*) = (-2.3562, 2.8018, 1.5)$

- norms $\varepsilon = 10^{-5}$ (max Cartesian error), $\varepsilon_q = 10^{-6}$ (min joint increment)
- $k_{max}=15$ (max # iterations), $|\det(J_r)| \leq 10^{-4}$ (closeness to singularity)
- numerical performance of Gradient (with different steps α) vs. Newton
- test 1: $q^0 = (0, 0, 1)$ as initial guess
- test 2: $q^0 = (-\pi/4, \pi/2, 1)$ —"singular" start, since $c_2=0$
- test 3: $q^0 = (0, \pi/2, 0)$ —"double singular" start, since also $q_3=0$
- solution and plots with Matlab code

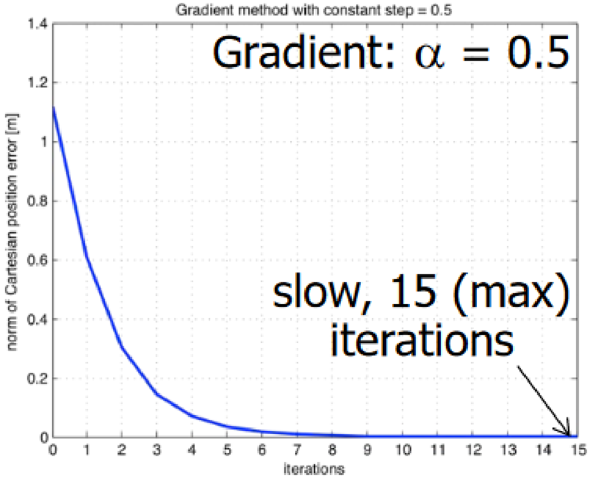
UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

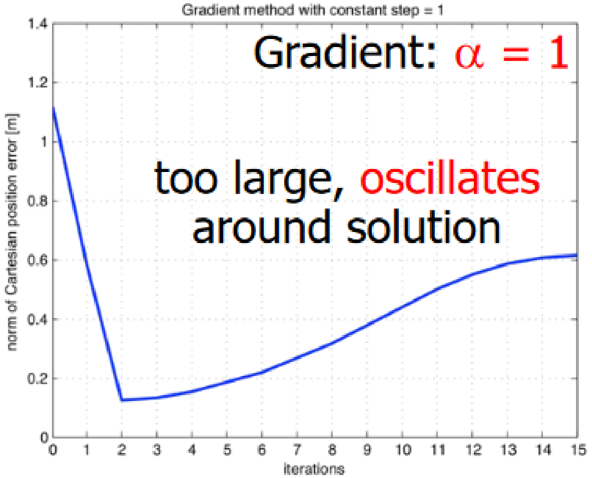
$$q^{k+1} = q^k + J_r^{-1}(q^k)\,[r_d - f_r(q^k)]$$

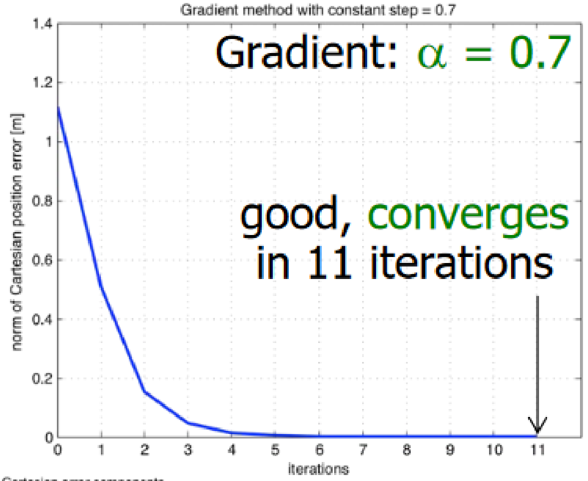
$$q^{k+1} = q^k + \alpha\, J_r^{T}(q^k)\,[r_d - f_r(q^k)]$$

Numerical test - 1

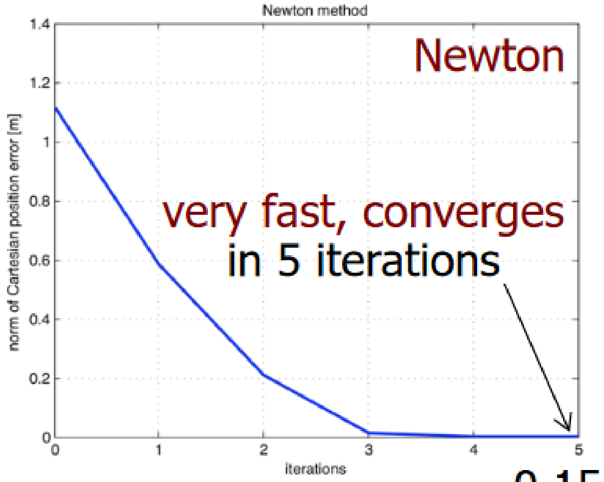
- test 1: $q^0 = (0, 0, 1)$ as initial guess; evolution of error norm





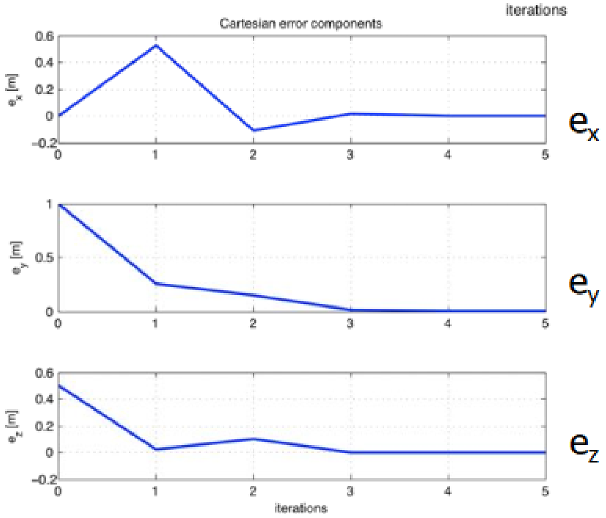


$0.57 \cdot 10^{-5}$



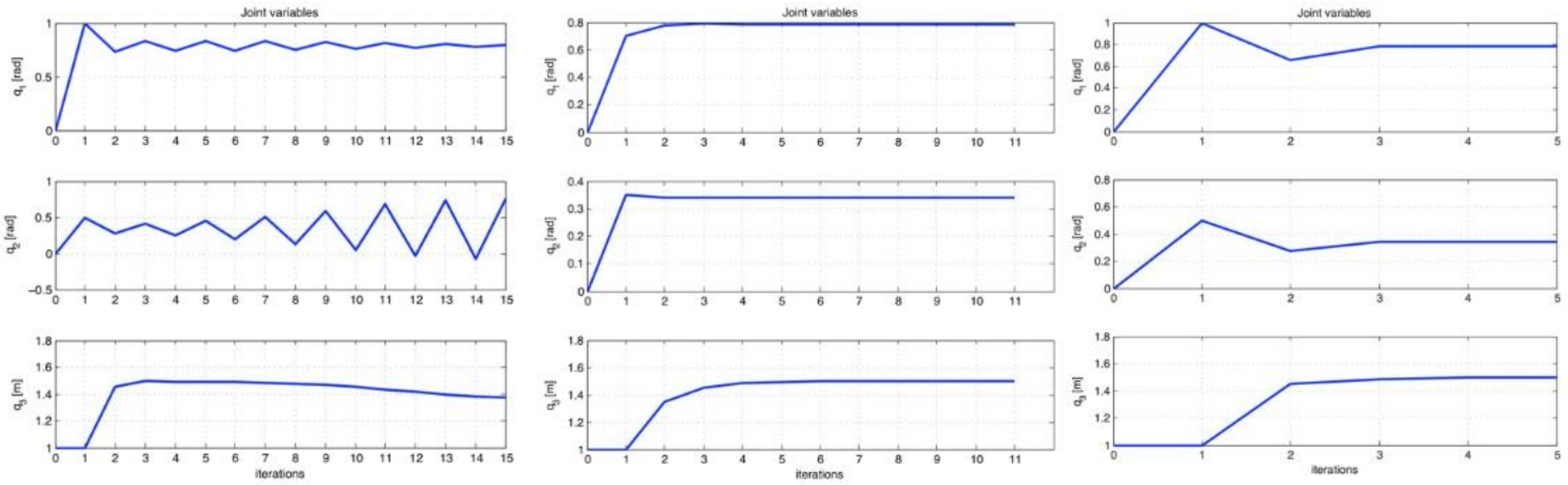
$0.15 \cdot 10^{-8}$

Cartesian errors component-wise



Numerical test - 1

- test 1: $q^0 = (0, 0, 1)$ as initial guess; evolution of joint variables



Gradient: $\alpha = 1$	Gradient: $\alpha = 0.7$	Newton
not converging to a solution	converges in 11 iterations	converges in 5 iterations

both to the same solution $q^* = (0.7854, 0.3398, 1.5)$

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

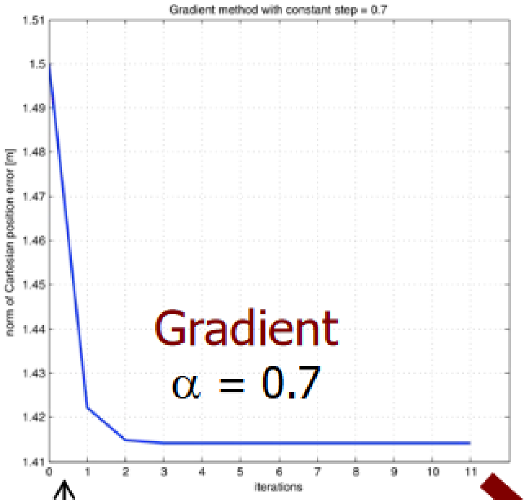
Numerical test - 2

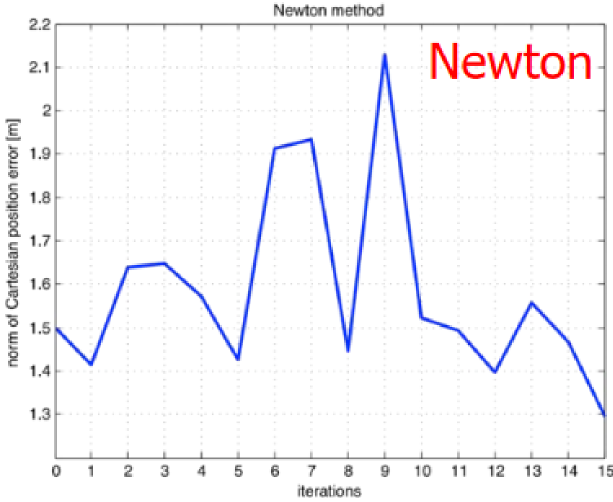
$q^{k+1} = q^k + \alpha\, J_r^T(q^k)\,[r_d - f_r(q^k)]$

$q^{k+1} = q^k + J_r^{-1}(q^k)\,[r_d - f_r(q^k)]$

- test 2: $q^0 = (-\pi/4, \pi/2, 1)$: singular start

error norms



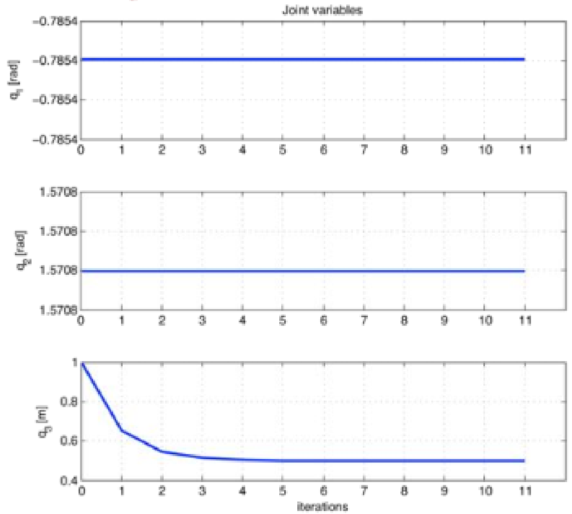


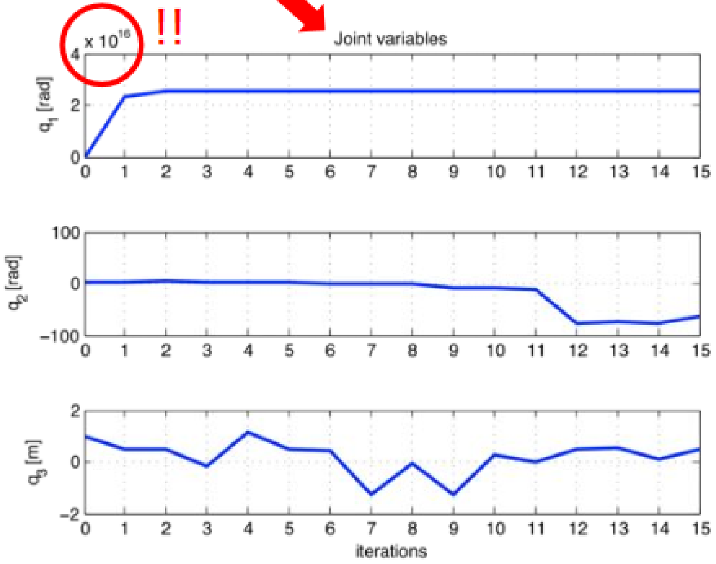
with check of singularity: blocked at start

without check: it diverges!

starts toward solution, but slowly stops (in singularity): when Cartesian error vector $e \in Ker(J_r^T)$

joint variables

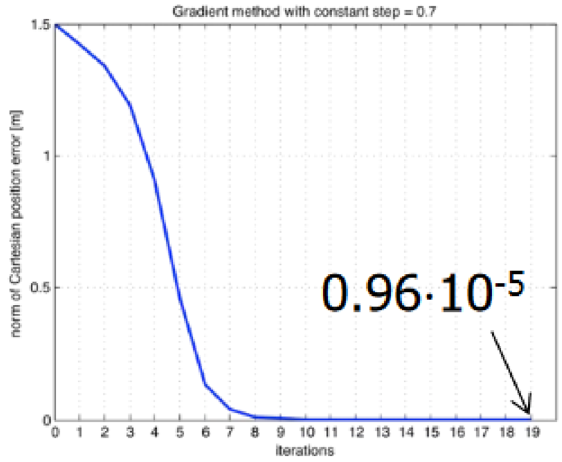




Numerical test - 3

- test 3: $q^0 = (0, \pi/2, 0)$: "double" singular start

error norm



Gradient (with $\alpha = 0.7$)

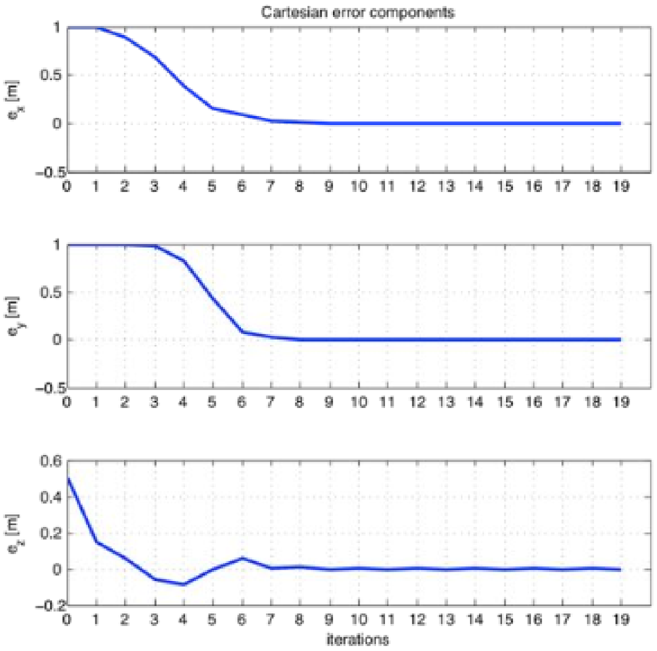
① starts toward solution

② exits the double singularity

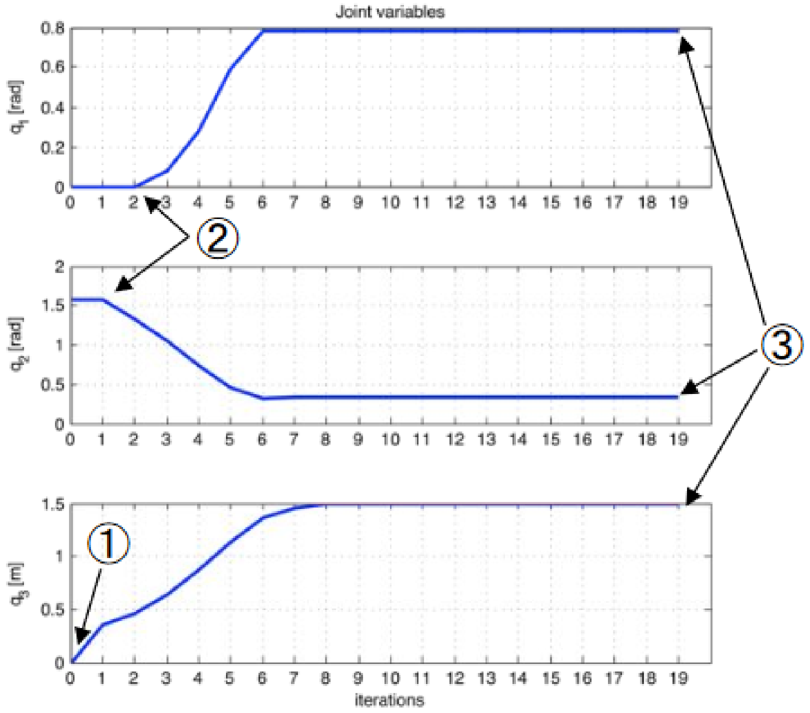
③ slowly converges in 19 iterations to the solution $q^* = (0.7854, 0.3398, 1.5)$

Newton
is either
blocked at start
or (w/o check)
explodes!
→ "NaN" in Matlab

Cartesian errors



joint variables



Final remarks

- an efficient iterative scheme can be devised by combining
  - initial iterations using Gradient ("sure but slow", linear convergence rate)
  - switch then to Newton method (quadratic terminal convergence rate)
- joint range limits are considered only at the end
  - check if the solution found is feasible, as for analytical methods
- in alternative, an optimization criterion can be included in the search
  - driving iterations toward an inverse kinematic solution with nicer properties
- if the problem has to be solved on-line
  - execute iterations and associate an actual robot motion: repeat steps at times t_0, $t_1=t_0+T$, ..., $t_k=t_{k-1}+T$ (e.g., every T=40 ms)
  - the "good" choice for the initial guess q^0 at t_k is the solution of the previous problem at t_{k-1} (provides continuity, needs only 1-2 Newton iterations)
  - crossing of singularities/handling of joint range limits need special care
- Jacobian-based inversion schemes are used also for kinematic control, along a continuous task trajectory $r_d(t)$

The end!