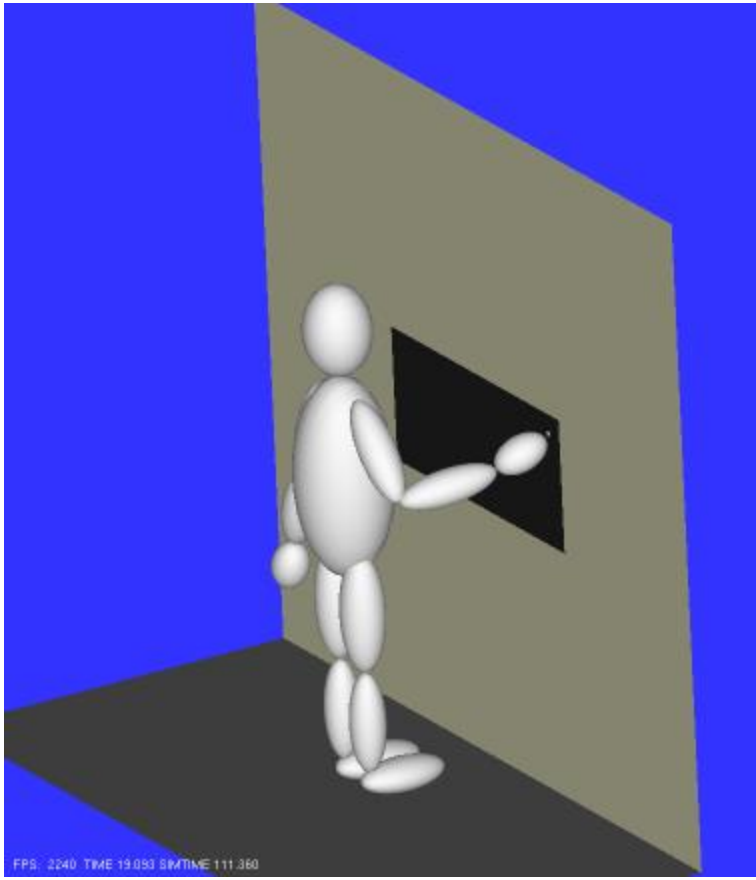
Inverse Kinematics & Motion Capturing

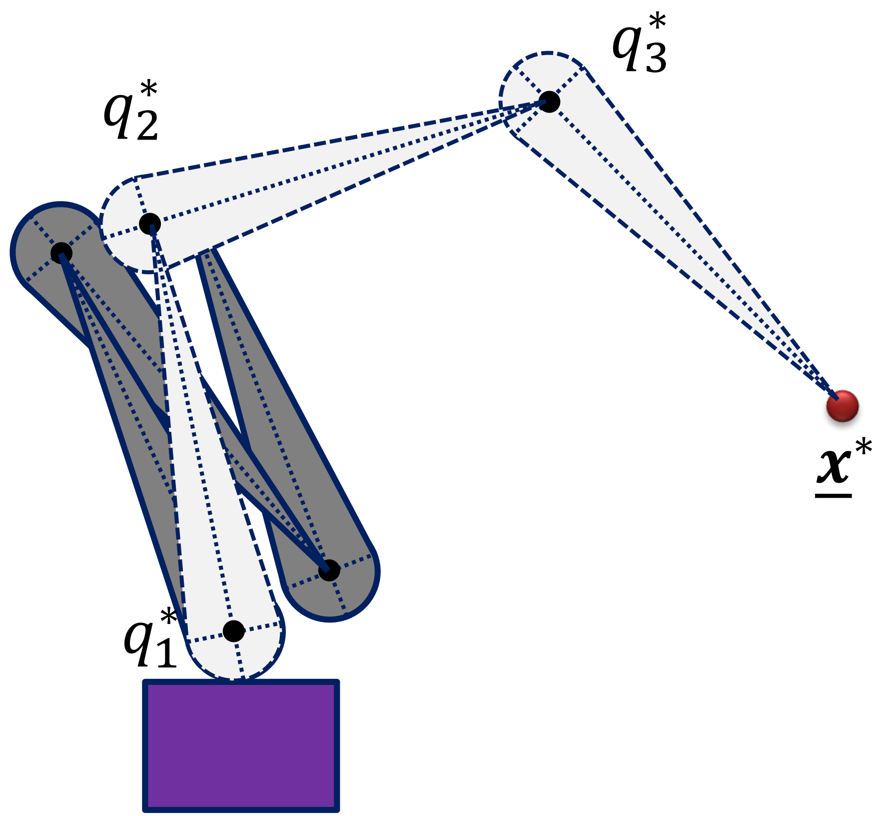
contents



INVERSE KINEMATICS PROBLEM

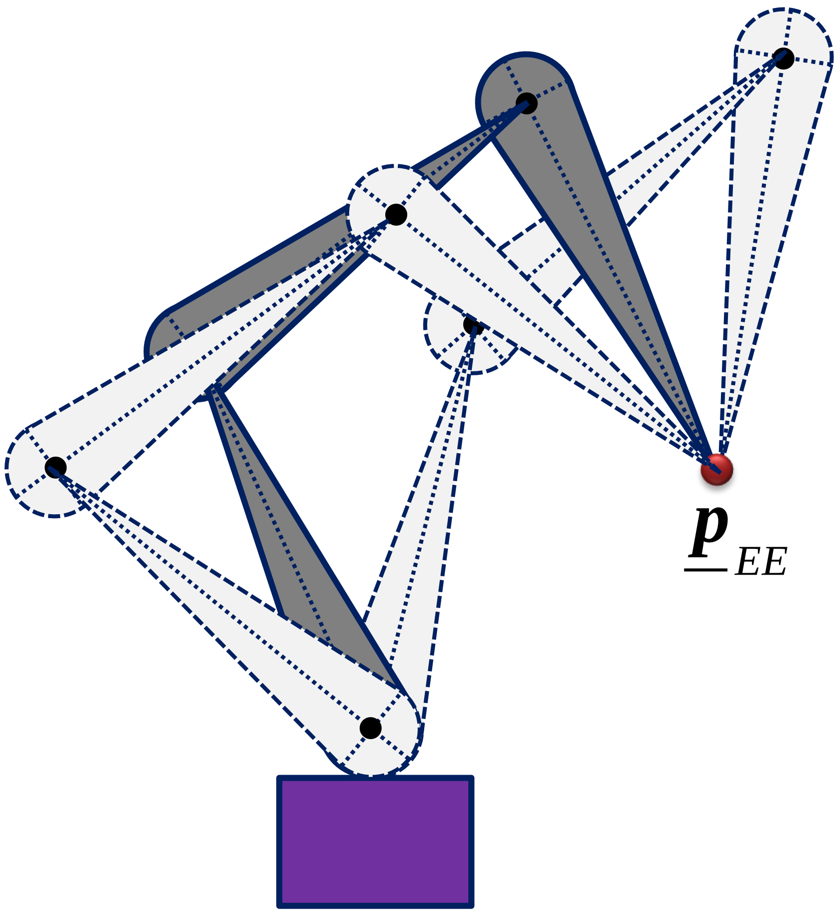
Inverse Kinematics Problem

- Given a kinematic chain with forward kinematics $\underline{\boldsymbol{p}}_{EE} = \underline{\boldsymbol{f}}(q_j), \vec{\boldsymbol{\omega}}_{EE} = \boldsymbol{F}(q_j)$ depending on generalized coordinates q_j and a target pose $[\underline{\boldsymbol{x}}^*, \vec{\boldsymbol{\omega}}^*]$, the inverse kinematics problems solves $\underline{\boldsymbol{p}}^* = \underline{\boldsymbol{f}}(q_j^*) \wedge \vec{\boldsymbol{\omega}}^* = \boldsymbol{F}(q_j^*)$ for the q_j^*
- The parameters q_j are also called state vector.
- The parameters $[\underline{\boldsymbol{p}}, \vec{\boldsymbol{\omega}}]$ of the end effector pose are called the dependent variables



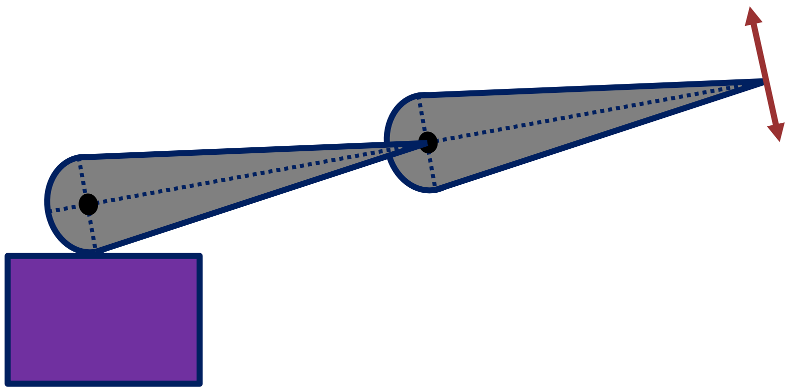
Inverse Kinematics Problem

- The number of generalized coordinates give the degrees of freedom (DOF)
- In most cases the DOFs does not match the number of dependent variables and IK becomes ill-posed (as in example on the right) or unsolvable
- The inverse kinematics problem is often posed in form of a least squares energy minimization as detailed on slides 20 & 21.

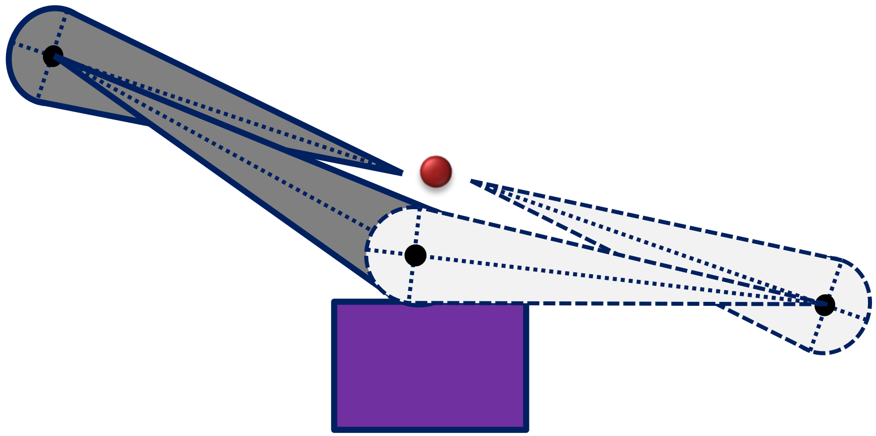


Inverse Kinematics

- In so called degenerate or singular configurations the end effector looses one or several degrees of freedom



- Close to singular positions the distance in the state space can become extremely large compared to the distance in world space leading to oscillations during IK



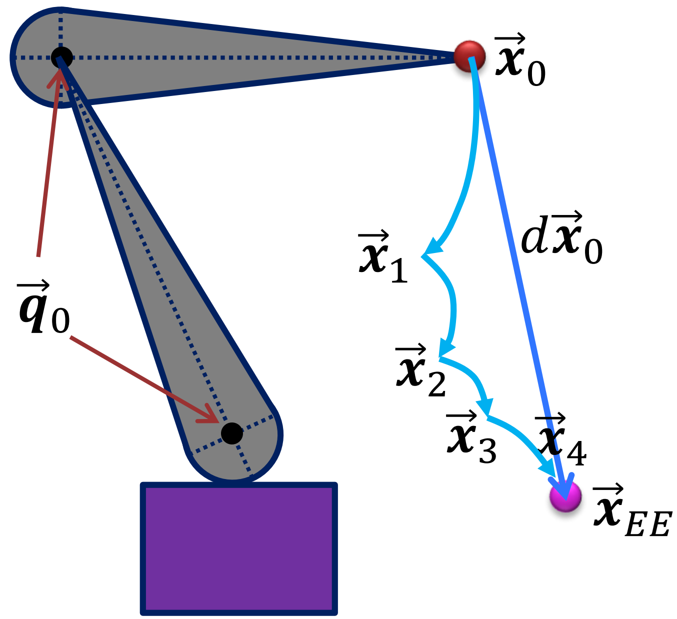
Contraints

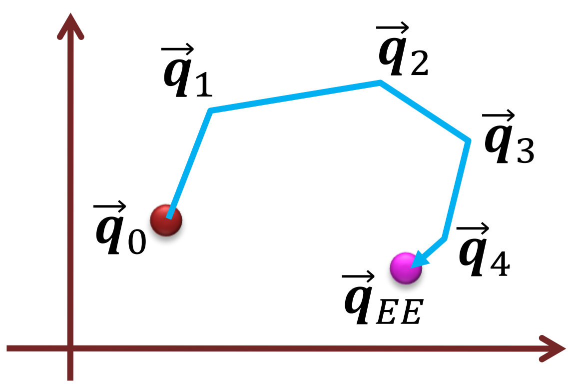
- for realistic behavior it is necessary to consider different constraints:
- collision constraints to avoid self-collisions and collisions with environment. These can only locally be captured in formula.
- joint angle constraints given by the limitations of the joints and written in the form $l_j \leq q_j \leq u_j$
- position constraints to restrict the movement of the end effector. Typically point, line or plane constraints.
- orientation constraints to restrict the orientation of the end effector, for example in case of special demands for grasping.

Inverse Kinematics

Overall strategy

- Most solvers for non linear optimization problems are iterative and start with some initial guess $\vec{\boldsymbol{q}}_0$ for the parameter vector. This can be
  - the default state
  - the state of the previous time step in an animation
  - the result of a previous optimization phase
- At the current guess some descent direction $d\vec{\boldsymbol{q}}_k$ is found in parameter space
- A step size h_k is estimated from the energy function
- Steps $\vec{\boldsymbol{q}}_{k+1} = \vec{\boldsymbol{q}}_k + h_k d\vec{\boldsymbol{q}}_k$ are taken until no further improvement in energy is possible.





Forward Kinematics

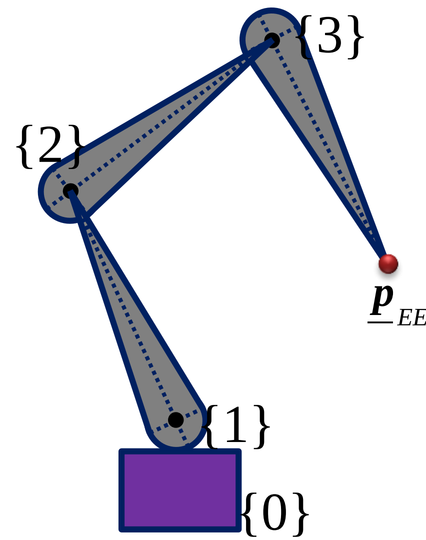
- Each joint in a kinematic chain has a relative transformation ${}^{i-1}\widetilde{\boldsymbol{T}}_i(q_{i1},\dots,q_{in_i})$ with n_i parameters
- Gathering all $n=\sum_i n_i$ parameters in the parameter vector $\vec{\boldsymbol{q}}$ the chain transform is

$$ {}^{0}\widetilde{\boldsymbol{T}}_N(\vec{\boldsymbol{q}}) = {}^{0}\widetilde{\boldsymbol{T}}_1\,{}^{1}\widetilde{\boldsymbol{T}}_2\cdot\ldots\cdot{}^{N-1}\widetilde{\boldsymbol{T}}_N $$

- Transformations are in homogenous notation

$$ {}^{0}\widetilde{\boldsymbol{T}}_N = \begin{pmatrix} {}^{0}\boldsymbol{R}_N & {}^{0}\vec{\boldsymbol{t}}_N \\ \vec{\boldsymbol{0}}^T & 1 \end{pmatrix} $$

  and contain rotation ${}^{0}\boldsymbol{R}_N$ and translation ${}^{0}\vec{\boldsymbol{t}}_N$.
- The columns of ${}^{0}\boldsymbol{R}_N$ define the coordinate axes of the end effector frame with respect to world coordinates



- If the end effector is in the origin of frame N, ${}^{0}\vec{\boldsymbol{t}}_N$ is the location of the end effector in world coordinates, in general we have $\underline{\boldsymbol{p}}^0_{EE} = {}^{0}\boldsymbol{R}_N\underline{\boldsymbol{p}}^N_{EE} + {}^{0}\vec{\boldsymbol{t}}_N$

Euler Angle Formulation

- Representing the orientation as Euler angles, the pose $\vec{\boldsymbol{x}}$ of the end effector is a 6-dimensional vector
$$\vec{\boldsymbol{x}}_{EE} = \left(\underline{\boldsymbol{p}}^0_{EE}, \vec{\boldsymbol{\omega}}^0_{EE}\right) = (x, y, z, \phi, \theta, \psi)^T \in \boldsymbol{R}^6$$
- The forward kinematics is described as a function $\vec{\boldsymbol{f}}$ that maps the n-dimensional parameter vector $\vec{\boldsymbol{q}}$ to a 6D pose:
$$\vec{\boldsymbol{x}} = \vec{\boldsymbol{f}}(\vec{\boldsymbol{q}})$$
- The inverse kinematic problem for a given end effector pose is to find the parameters that best match the pose:
$$\vec{\boldsymbol{q}}^* = \underset{\vec{\boldsymbol{q}}}{\mathrm{minarg}}\, E(\vec{\boldsymbol{q}}; \vec{\boldsymbol{x}}_{EE}) \text{ with } E(\vec{\boldsymbol{q}}; \vec{\boldsymbol{x}}_{EE}) = \frac{1}{2}\left\|\vec{\boldsymbol{f}}(\vec{\boldsymbol{q}}) - \vec{\boldsymbol{x}}_{EE}\right\|^2$$
- This is a non linear least squares problem. One can introduce weights w_i for the residua $r_i(\vec{\boldsymbol{q}}) = f_i(\vec{\boldsymbol{q}}) - x_{EE,i}$:
$$E(\vec{\boldsymbol{q}}; \vec{\boldsymbol{x}}_{EE}) = \frac{1}{2}\sum_i w_i r_i^2(\vec{\boldsymbol{q}})$$

Orientation Matrix Formulation

$$\boldsymbol{o}_x(\vec{\boldsymbol{q}})^T \boldsymbol{o}_{x,EE} = \cos\psi_1$$
$$\boldsymbol{o}_y(\vec{\boldsymbol{q}})^T \boldsymbol{o}_{y,EE} = \cos\psi_2$$
$$\boldsymbol{o}_z(\vec{\boldsymbol{q}})^T \boldsymbol{o}_{z,EE} = \cos\psi_3$$

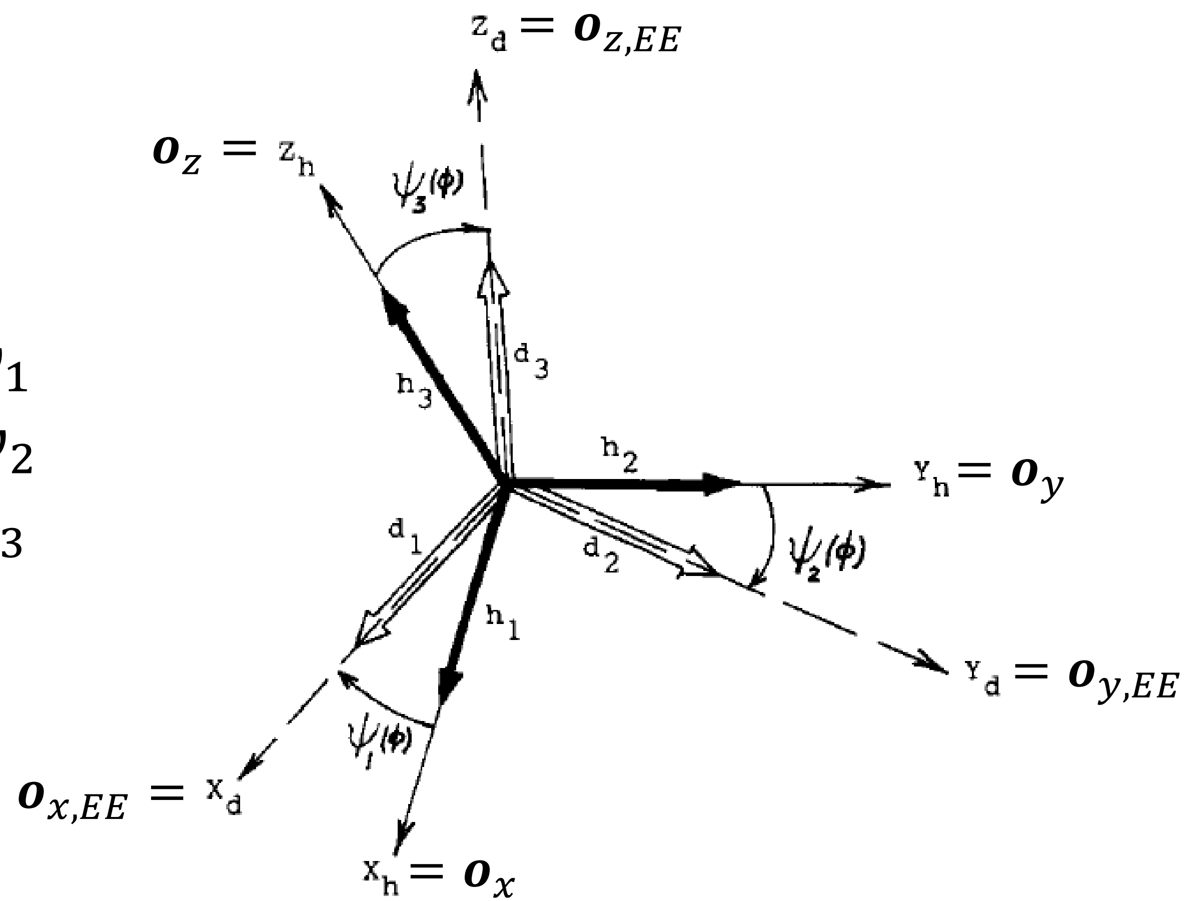


Fig. 5. Definition of the direction angles.

Orientation Matrix Formulation

- Using an orthonormal matrix $\boldsymbol{O} = (\boldsymbol{o}_x\ \boldsymbol{o}_y\ \boldsymbol{o}_z)$ to represent orientation, the pose $\vec{\boldsymbol{x}}$ of the end effector is 12D and defined as concatenation of position columns of $\boldsymbol{O}$:

$$\vec{\boldsymbol{x}}_{EE} = \left(\underline{\boldsymbol{p}}_{EE}^{0}{}^{T}\ \boldsymbol{o}_{x,EE}^{T}\ \boldsymbol{o}_{y,EE}^{T}\ \boldsymbol{o}_{z,EE}^{T}\right)^{T} \in \boldsymbol{R}^{12}$$

- The forward kinematics is described as a function $\vec{\boldsymbol{f}}$ that maps the n-dimensional parameter vector $\vec{\boldsymbol{q}}$ to a pose:

$$\vec{\boldsymbol{x}} = \vec{\boldsymbol{f}}(\vec{\boldsymbol{q}}), \qquad \text{with}\ \ \vec{\boldsymbol{f}} = \left(\underline{\boldsymbol{p}}^{T}\ \boldsymbol{o}_x^{T}\ \boldsymbol{o}_y^{T}\ \boldsymbol{o}_z^{T}\right)^{T}$$

- The inverse kinematic problem is again based on an energy function: $\vec{\boldsymbol{q}}^{*} = \underset{\vec{\boldsymbol{q}}}{\operatorname{minarg}} E(\vec{\boldsymbol{q}}; \vec{\boldsymbol{x}}_{EE})$ with weights

$$E(\vec{\boldsymbol{q}}; \vec{\boldsymbol{x}}_{EE}) = \frac{1}{2}\left\|\underline{\boldsymbol{p}}(\vec{\boldsymbol{q}}) - \underline{\boldsymbol{p}}_{EE}^{0}\right\|^{2} + \frac{1}{2}\sum_{\alpha=x}^{z} w_\alpha \left(1 - \boldsymbol{o}_\alpha(\vec{\boldsymbol{q}})^{T}\boldsymbol{o}_{\alpha,EE}\right)^{2}$$

- The weights w_α can be used to blend out orientation constraints for individual axes

Inverse Kinematics

Computer Graphics and Visualization

Quaternion Formulation

- With a normalized quaternion $\hat{q} = (s\ x\ y\ z)$ the pose is 7D:

$$\vec{x}_{EE} = \left(\underline{\boldsymbol{p}}_{EE}^{0}{}^{T}\ \hat{q}_{EE}^{0}{}^{T}\right)^{T} \in \boldsymbol{R}^{7}, \qquad \vec{\boldsymbol{f}}(\vec{\boldsymbol{q}}) = \left(\underline{\boldsymbol{p}}^{T}\ \hat{q}^{T}\right)^{T}$$

- Care needs to be taken with the least squares energy

$$E(\vec{\boldsymbol{q}}; \vec{\boldsymbol{x}}_{EE}) = \frac{1}{2}\left\|\underline{\boldsymbol{p}}(\vec{\boldsymbol{q}}) - \underline{\boldsymbol{p}}_{EE}^{0}\right\|^{2} + \frac{1}{2} w_q \left\|\hat{q}(\vec{\boldsymbol{q}}) - \hat{q}_{EE}^{0}\right\|^{2}$$

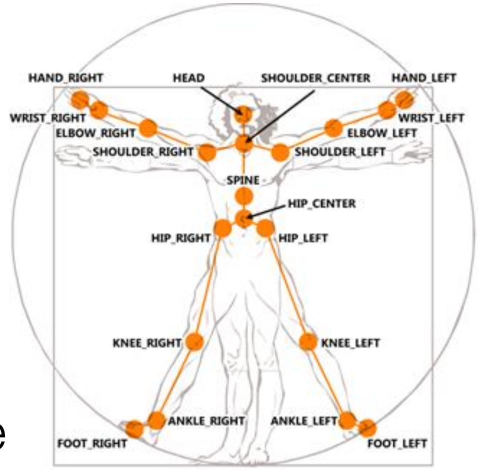
as $\hat{q}$ and $-\hat{q}$ represent the same rotation.

- Solution 1:
  - during iterative optimization compute $\hat{q}(\vec{\boldsymbol{q}}_i)$ for current $\vec{\boldsymbol{q}}_i$.
  - test if $\langle \hat{q}(\vec{\boldsymbol{q}}_i), \hat{q}_{EE}^{0} \rangle < 0$
  - if yes replace $\hat{q}_{EE}^{0}$ with $-\hat{q}_{EE}^{0}$ in energy
  - compute the next $\vec{\boldsymbol{q}}_{i+1}$.
- Solution 2: use energy function that ignores sign:

$$E(\vec{\boldsymbol{q}}; \vec{\boldsymbol{x}}_{EE}) = \frac{1}{2}\left\|\underline{\boldsymbol{p}}(\vec{\boldsymbol{q}}) - \underline{\boldsymbol{p}}_{EE}^{0}\right\|^{2} - w_q \langle \hat{q}(\vec{\boldsymbol{q}}_i), \hat{q}_{EE}^{0} \rangle^{2}$$

Skeleton IK

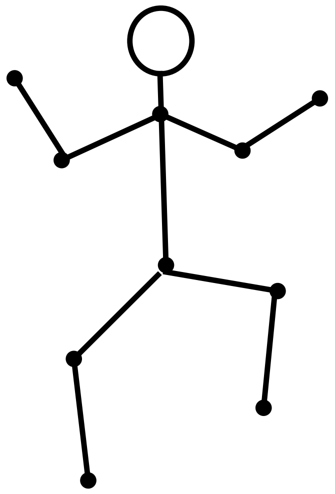
- multiple target locations are important in applications with skeletons like transforming point based motion capture data to skeleton parameters or when working with multiple end effectors
- a fixed base joint can only be used in the very special case of two constraints – one at a base node and one at an endeffector node
- otherwise we can alternatingly optimize
  - the rigid body transform of the root node by minimizing the squared sum of the end effector constraints with the Kabsch algorithm
  - the joint parameters by a joined IK problem minimizing an energy that sums over the squared endeffector-constraint distances where the endeffector locations are computed along kinematic chains from the fixed root node



Discussion

Computer Graphics and Visualization

- In an interactive editor one has to define kinematic chains based on user input. This can be done by adding fixation points (in the simplest case one at the root node)
- For a climbing figure we need several end effector constraints. This is discussed by Chris Hecker on his inverse kinematics page
- Dual quaternion IK was presented by Ben Kenwright in 2013: Inverse Kinematics with Dual-Quaternions, Exponential-Maps, and Joint Limits



CYCLIC COORDINATE DESCENT

Iterative Methods

- Iterative algorithms need an initial guess $\boldsymbol{x}_0$
  $\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \Rightarrow f_k = f(\boldsymbol{x}_k), \boldsymbol{\nabla} f_k = \boldsymbol{\nabla} f(\boldsymbol{x}_k), \boldsymbol{\nabla}^2 f_k = \boldsymbol{\nabla}^2 f(\boldsymbol{x}_k),$
- sequence chosen monoton with termination criterion:

$$f_0 \geq f_1 \geq f_2 \geq \cdots \quad \|\boldsymbol{\nabla} f_k\| < \varepsilon$$

Line Search Approach

choose search direction $\boldsymbol{h}_k$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{h}_k$$

- choose step width α_k to minimize f along line:
  $\phi_k(\alpha) = f(\boldsymbol{x}_k + \alpha \boldsymbol{h}_k),$
  $\alpha_k = \underset{\alpha>0}{\operatorname{minarg}}\, \phi_k(\alpha)$

Trust Region Approach

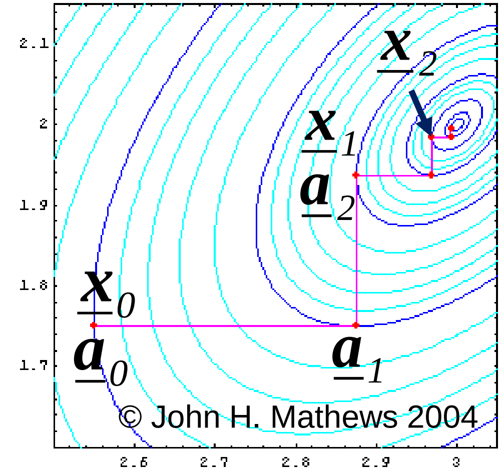
- choose model m_k to approximate f inside trust region T around $\boldsymbol{x}_k$
  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{h}_k,$
  $\boldsymbol{h}_k = \underset{\boldsymbol{h}\in T}{\operatorname{minarg}}\, m_k(\boldsymbol{x}_k + \boldsymbol{h})$
- example:

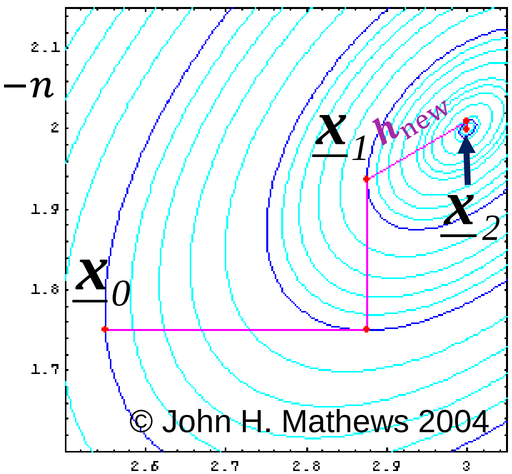
approximates Hessian

$$m_k(\boldsymbol{x}_k + \boldsymbol{h}) = f_k + \boldsymbol{\nabla} f_k^T \boldsymbol{h} + \frac{1}{2} \boldsymbol{h}^T \boldsymbol{B}_k \boldsymbol{h}$$

Coordinate Descent & Powell's Method

- Coordinate Descent:
  - Search directions are standard basis vectors $\hat{\boldsymbol{e}}_i$
  - line search along search directions
  - Stop if no improvement in one complete cycle
- Powell's method:
  - start with standard basis as set of search directions and do line search
  - after one cycle add new direction $\boldsymbol{h}_{\text{new}} = \boldsymbol{x}_i - \boldsymbol{x}_{i-n}$
  - remove search direction with largest improvement (closest to new search direction $\boldsymbol{h}_{\text{new}}$) to avoid degenerated set of search directions
- Both are derivative free methods!
- Another mentionable derivative-free alternative is the Downhill-Simplex method of Nelder and Mead





Cyclic Coordinate Descent (CCD)

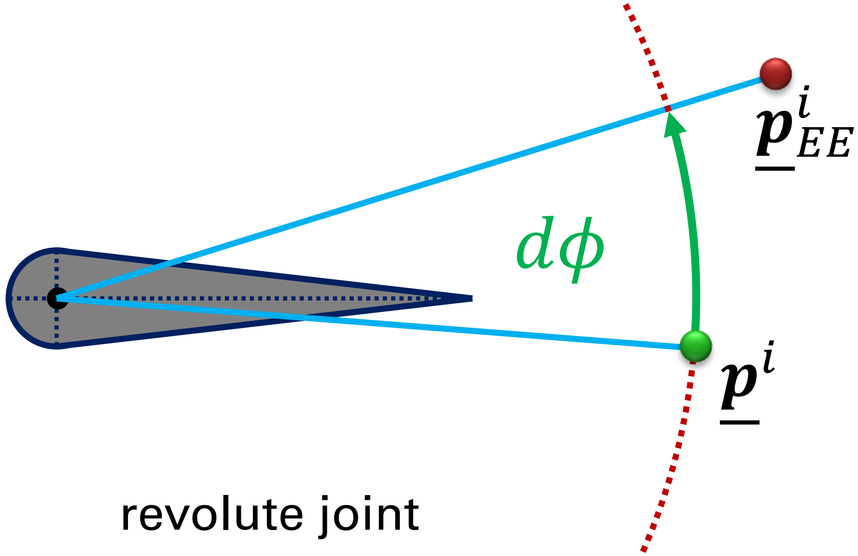
- In IK coordinate descent is often used to generate initial guess for secondary method with fast convergence rate close to optimum
- coordinate descent can be implemented efficiently as the line search problem can be solved analytically

Cyclic Coordinate Descent Algorithm

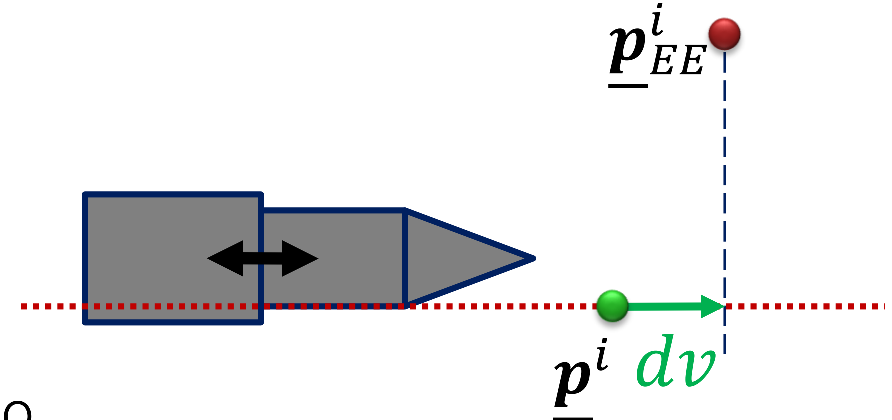
- start with initial guess $\vec{\boldsymbol{q}}^0$ for joint parameters
- for increasing $j = 1 \ldots$ do
  - for each joint parameter i (typically from end effector to base)
    - reduce IK to parameter q_i^j
    - solve line search IK analytically to get q_i^{j+1}
- until convergence, i.e. $\|\vec{\boldsymbol{f}}(\vec{\boldsymbol{q}}^{j+1}) - \vec{\boldsymbol{f}}(\vec{\boldsymbol{q}}^{j})\| < \varepsilon$

CCD – Line Search – Simplified

- First approach: only optimize end effector position $\underline{\boldsymbol{p}} \rightarrow \underline{\boldsymbol{p}}_{EE}$
- For given joint i transform positions into joint coordinate system $\Rightarrow \underline{\boldsymbol{p}}^i, \underline{\boldsymbol{p}}^i_{EE}$
- Update joint parameter q_i^j such that end effector comes as close to target position as possible
- **Revolute joint:** $q_i^j \equiv \phi$ rotation $d\phi$ makes position vector $\underline{\boldsymbol{p}}^i$ parallel to $\underline{\boldsymbol{p}}^i_{EE}$
- **Prismatic joint:** $q_i^j \equiv v$ translation dv moves $\underline{\boldsymbol{p}}^i$ to the orthogonal projection of $\underline{\boldsymbol{p}}^i_{EE}$onto z-direction.



revolute joint



prismatic joint

CCD – Line Search – Full Approach (1)

- **Prismatic joints** do not affect orientation and optimization is done as in the case without orientation
- For a **revolute joint** i they reformulate energy minimization in local coordinates according to Orientation Matrix Formulation

$$E(d\phi) = \underbrace{\frac{1}{2}\left\|\underline{\boldsymbol{p}}^i(\phi_i + d\phi) - \underline{\boldsymbol{p}}_{EE}^i\right\|^2}_{E_1(d\phi)} + \underbrace{\frac{1}{2}\sum_{\alpha=x}^{z} w_\alpha\left(1 - \boldsymbol{o}_\alpha^i(\phi_i + d\phi)^T \boldsymbol{o}_{\alpha,EE}^i\right)^2}_{E_2(d\phi)}$$

into the maximization of a derived function $g = g_1 + g_2$

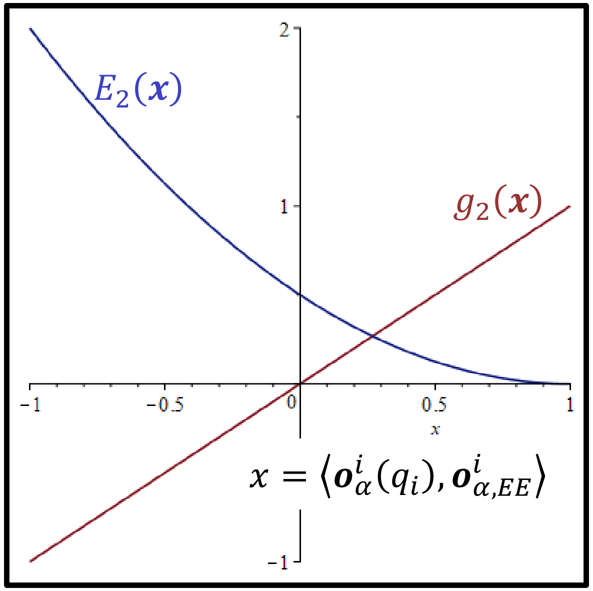
- **Position term:** $2E_1(d\phi) = \left\|\underline{\boldsymbol{p}}^i(\phi_i + d\phi) - \underline{\boldsymbol{p}}_{EE}^i\right\|^2$

$$= \left\|\underline{\boldsymbol{p}}^i(\phi_i + d\phi)\right\|^2 + \left\|\underline{\boldsymbol{p}}_{EE}^i\right\|^2 - 2\left\langle\underline{\boldsymbol{p}}^i(\phi_i + d\phi), \underline{\boldsymbol{p}}_{EE}^i\right\rangle$$

The first two terms on the right side are not affected by the joint rotation, such that we can choose $g_1(d\phi) := \left\langle\underline{\boldsymbol{p}}^i(\phi_i + d\phi), \underline{\boldsymbol{p}}_{EE}^i\right\rangle$ and it holds: $\mathrm{minarg}_{d\phi}\, E_1(d\phi) = \mathrm{maxarg}_{d\phi}\, g_1(d\phi)$

CCD – Line Search – Full Approach (1)

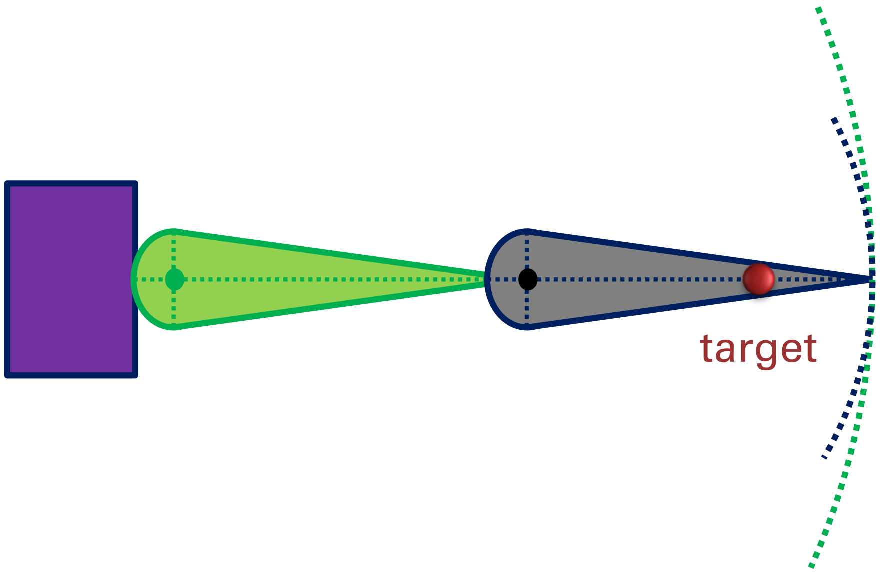
- **Revolute Joint** transform $E = E_1 + E_2$ into $g = g_1 + g_2$
- **Position term:** $g_1(d\phi) = \left\langle \underline{\boldsymbol{p}}^i(\phi_i + d\phi), \underline{\boldsymbol{p}}^i_{EE} \right\rangle$
- **Orientation term:**
  $$2E_{2,\alpha}(d\phi) = w_\alpha\left(1 - \boldsymbol{o}^i_\alpha(\phi_i + d\phi)^T \boldsymbol{o}^i_{\alpha,EE}\right)^2$$
  $$= w_\alpha \left(1 - 2\langle \boldsymbol{o}^i_\alpha(\phi_i + d\phi), \boldsymbol{o}^i_{\alpha,EE}\rangle + \left(\boldsymbol{o}^i_\alpha(\phi_i + d\phi)^T \boldsymbol{o}^i_{\alpha,EE}\right)^2\right)$$
- Ignoring $\left(\boldsymbol{o}^i_\alpha(\phi_i + d\phi)^T \boldsymbol{o}^i_{\alpha,EE}\right)^2$ one can set $g_2(d\phi) := \langle \boldsymbol{o}^i_\alpha(\phi_i + d\phi), \boldsymbol{o}^i_{\alpha,EE}\rangle$
- **Careful:** Only in case the IK problem has a solution, it can be shown that $\mathrm{minarg}_{d\phi}\, E_2(d\phi) = \mathrm{maxarg}_{d\phi}\, g_2(d\phi)$



CCD – Line Search – Full Approach (1)

- **Revolute Joint** transform $E = E_1 + E_2$ into $g = g_1 + g_2$
- applying joint rotation: $\boldsymbol{R}(d\phi) = \begin{pmatrix} \cos d\phi & -\sin d\phi & 0 \\ \sin d\phi & \cos d\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$
  gives $\underline{\boldsymbol{p}}^i(\phi_i + d\phi) = \boldsymbol{R}(d\phi)\underline{\boldsymbol{p}}^i(\phi_i)$ and
  $$\begin{aligned} g_1(d\phi) &= \left\langle \underline{\boldsymbol{p}}^i(\phi_i + d\phi), \underline{\boldsymbol{p}}^i_{EE} \right\rangle \\ &= a_p \cdot \cos d\phi + b_p \cdot \sin d\phi + c_p \end{aligned}$$
  with $a_p = p^i_x p^i_{EE,x} + p^i_y p^i_{EE,y} \quad b_p = p^i_x p^i_{EE,y} - p^i_y p^i_{EE,x} \quad c_p = p^i_z p^i_{EE,z}$
- similarly we get
  $$\begin{aligned} g_2(d\phi) &= \left\langle \boldsymbol{o}^i_\alpha(\phi_i + d\phi), \boldsymbol{o}^i_{\alpha,EE} \right\rangle \\ &= a_{o,\alpha} \cdot \cos d\phi + b_{o,\alpha} \cdot \sin d\phi + c_{o,\alpha} \end{aligned}$$
- gathering all we get
  $$g(d\phi) = g_1(d\phi) + g_2(d\phi) = a \cdot \cos d\phi + b \cdot \sin d\phi + c$$
- we get optimal $d\phi$ from $\partial_{d\phi} g(d\phi) = 0$ and $\partial^2_{d\phi} g(d\phi) < 0$

CCD – Degenerate Case

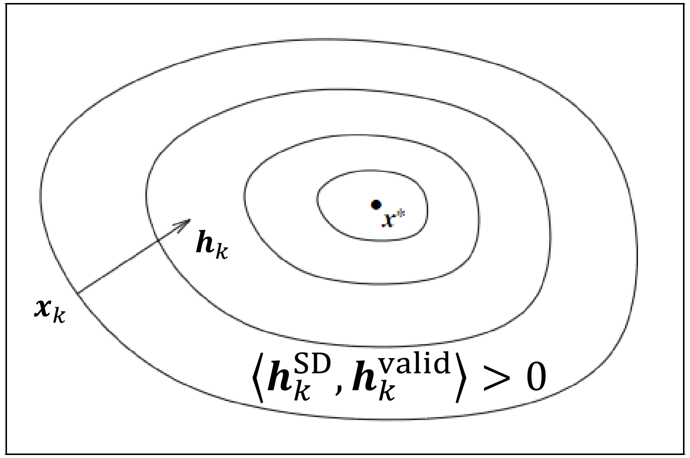


- the CCD can get stuck in degenerate situations
- solution strategy:
  - per iteration add a random offset to each joint parameter
  - decrease (i.e. divide by 2) amplitude of random offset after each iteration

UNCONSTRAINED IK

Steepest Descent

- In the steepest descent method the search direction is the negative gradient which points in direction of steepest descent of the function
$$\boldsymbol{h}_k^{\mathrm{SD}} = -\boldsymbol{\nabla} f_k$$
- In case of a linear least squares problem $f(\boldsymbol{p}) = \boldsymbol{r}^T \boldsymbol{W} \boldsymbol{r}$, with $\boldsymbol{r} = \boldsymbol{A}\boldsymbol{p} - \boldsymbol{b}$ gradient is $\boldsymbol{\nabla} f_k = 2\boldsymbol{A}^T \boldsymbol{W} \boldsymbol{r}_k$ and the optimal step width can be computed exactly to
$$\alpha_k = -\frac{\boldsymbol{r}^T \boldsymbol{W} \boldsymbol{A} \boldsymbol{\nabla} f_k}{\boldsymbol{\nabla} f_k{}^T \boldsymbol{A}^T \boldsymbol{W} \boldsymbol{A} \boldsymbol{\nabla} f_k}$$
- otherwise line search needed.



All directions that have a positive scalar product with negative gradient are descent and therefore valid search directions

Newton direction

- The Newton direction follows from the Taylor expansion up to second order
- The direction is computed from setting the gradient of the second order approximation to zero
- optionally one can solve linear system for h_k^N
- natural step length: $\alpha_k = 1$
- h_k^N is descent direction if Hessian is positive definite

$$m_k(\boldsymbol{x}_k + \boldsymbol{h}) = f_k + \nabla f_k^T \boldsymbol{h} + \frac{1}{2}\boldsymbol{h}^T \nabla^2 f_k \boldsymbol{h}$$

$$\begin{aligned} \mathbf{0} &= \nabla_{\boldsymbol{h}} m_k(\boldsymbol{x}_k + \boldsymbol{h}_k^N) \\ &= \nabla f_k + \nabla^2 f_k \boldsymbol{h}_k^N \end{aligned}$$

$$\boldsymbol{h}_k^N = -(\nabla^2 f_k)^{-1} \nabla f_k$$

$$\nabla^2 f_k \boldsymbol{h}_k^N = -\nabla f_k$$

- convergence rate near local minimum quadratic

Quasi Newton methods

- Quasi Newton methods track approximation of Hessian or its inverse
- Taylor expansion of gradient gives secant equation as constraint
- Furthermore, $\boldsymbol{B}_{k+1}$ should be symmetric, positive def. and minimize difference to $\boldsymbol{B}_k$ in Frob. norm yielding Davidon, Fletcher, Powell (DFP) update
- same idea for $\boldsymbol{H}_k$ yields the better Broyden, Fletcher, Goldfarb, Shanno (BFGS) update
- Low memory implementation represents $\boldsymbol{H}_k$ through few vectors ➔ L-BFGS

$$\boldsymbol{B}_k \approx \nabla^2 f_k \quad \text{or} \quad \boldsymbol{H}_k \approx \nabla^2 f_k^{-1}$$

$$\underbrace{\nabla f_{k+1} - \nabla f_k}_{\boldsymbol{y}_k} \approx \nabla^2 f_k \cdot \underbrace{\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right)}_{\boldsymbol{s}_k}$$

$$\Rightarrow \boldsymbol{B}_{k+1}\boldsymbol{s}_k = \boldsymbol{y}_k$$

(DFP)

$$\boldsymbol{B}_{k+1} = \left(\boldsymbol{I} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^T\right)\boldsymbol{B}_k\left(\boldsymbol{I} - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^T\right) + \rho_k \boldsymbol{y}_k \boldsymbol{y}_k^T$$

$$\text{with} \quad \rho_k = \frac{1}{\boldsymbol{y}_k^T \boldsymbol{s}_k}$$

(BFGS)

$$\boldsymbol{H}_{k+1} = \left(\boldsymbol{I} - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^T\right)\boldsymbol{H}_k\left(\boldsymbol{I} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^T\right) + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T$$

Weighted Non Linear Least Squares WNLLS

- n-dim. parameter space $P = R^n$: $q_{j=1\ldots n}$ or $\vec{\boldsymbol{q}} \in P$
- m-dim. target space $T = R^m$: $x_{i=1\ldots m}$ or $\vec{\boldsymbol{x}} \in T$
- forward kinematic function $\vec{\boldsymbol{f}}: \vec{\boldsymbol{q}} \mapsto f_i(\vec{\boldsymbol{q}})$ or $\vec{\boldsymbol{f}}(\vec{\boldsymbol{q}}) \in T$
- gradient is Jacobian matrix: $\boldsymbol{J}(\vec{\boldsymbol{q}}) = \left(\frac{\partial f_i(\vec{\boldsymbol{q}})}{\partial q_j}\right)_{ij} \in R^{m\times n}$
- Taylor series: $\vec{\boldsymbol{f}}(\vec{\boldsymbol{q}}_0 + d\vec{\boldsymbol{q}}) = \vec{\boldsymbol{f}}(\vec{\boldsymbol{q}}_0) + \boldsymbol{J}(\vec{\boldsymbol{q}})d\vec{\boldsymbol{q}} + \boldsymbol{O}(d\vec{\boldsymbol{q}}^2)$
- Residua: $r_i(\vec{\boldsymbol{q}}) = f_i(\vec{\boldsymbol{q}}) - x_i$ or $\vec{\boldsymbol{r}}(\vec{\boldsymbol{q}}) = \vec{\boldsymbol{f}}(\vec{\boldsymbol{q}}) - \vec{\boldsymbol{x}}$
- objective function: $E(\vec{\boldsymbol{q}}) = \frac{1}{2}\sum_{i=1}^{n} w_i r_i^2$ or $E(\vec{\boldsymbol{q}}) = \frac{1}{2}\vec{\boldsymbol{r}}^T\boldsymbol{W}\vec{\boldsymbol{r}}$
  with $\boldsymbol{W} = \mathrm{diag}(w_i)$
  being the weight matrix

Computing the Jacobian

- In matrix representation:

$$\underline{\boldsymbol{p}}_{EE}^{0} = {}^{0}\boldsymbol{R}_N \underline{\boldsymbol{p}}_{EE}^{N} + {}^{0}\vec{\boldsymbol{t}}_N$$

$${}^{0}\boldsymbol{T}_N(\vec{\boldsymbol{q}}) = {}^{0}\boldsymbol{T}_1 \, {}^{1}\boldsymbol{T}_2 \cdot \ldots \cdot {}^{N-1}\boldsymbol{T}_N = \begin{pmatrix} {}^{0}\boldsymbol{R}_N & {}^{0}\vec{\boldsymbol{t}}_N \\ \vec{\boldsymbol{0}}^T & 1 \end{pmatrix}$$

- We exploit the fact that only ${}^{j-1}\boldsymbol{T}_j(q_j)$ depends on q_j. Therefore we have

$$\frac{\partial\, {}^{0}\boldsymbol{T}_N(\vec{\boldsymbol{q}})}{\partial q_j} = {}^{0}\boldsymbol{T}_{j-1} \frac{\partial\, {}^{j-1}\boldsymbol{T}_j(q_j)}{\partial q_j} \, {}^{j}\boldsymbol{T}_N$$

- For a rotation around an axis we get in 3D:

$$\frac{\partial}{\partial\varphi}\mathrm{Rot}_{\boldsymbol{z}}(\varphi) = \frac{\partial}{\partial\varphi}\begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -\sin\varphi & -\cos\varphi & 0 \\ \cos\varphi & -\sin\varphi & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

- This can be used in DH-notation:

$$\frac{\partial}{\partial\varphi}\, {}^{j-1}\boldsymbol{T}_j = \underbrace{\mathrm{Rot}_{\boldsymbol{x}}(\alpha_{j-1}) \cdot \mathrm{Trans}_{\boldsymbol{x}}(a_{j-1}) \cdot \mathrm{Trans}_{\boldsymbol{z}}(d_j)}_{{}^{i-1}\boldsymbol{C}_i} \cdot \frac{\partial}{\partial\varphi_j}\mathrm{Rot}_{\boldsymbol{z}}(\varphi_j)$$

WNLLS – Descent Directions

- energy: $E(\vec{\boldsymbol{q}}) = \frac{1}{2}\sum_{i=1}^{n} w_i r_i^2$, with $r_i(\vec{\boldsymbol{q}}) = f_i(\vec{\boldsymbol{q}}) - x_i$
- gradient: $\frac{\partial}{\partial q_j} E = \sum_i w_i r_i \frac{\partial f_i}{\partial q_j}$ or $\vec{\boldsymbol{\nabla}}_{\vec{\boldsymbol{q}}} E = \vec{\boldsymbol{r}}^T \boldsymbol{W}\boldsymbol{J}$ (row vector)

Descent directions:

- steepest descent direction:

$$\boldsymbol{h}_k^{\mathrm{SD}} = -\vec{\boldsymbol{\nabla}}_{\vec{\boldsymbol{q}}}^T E_k = -\boldsymbol{J}_k^T \boldsymbol{W} \vec{\boldsymbol{r}}_k$$

- approximate Hessian $\widetilde{\boldsymbol{H}}$ (assume $\boldsymbol{J}$ in $\vec{\boldsymbol{\nabla}}_{\vec{\boldsymbol{q}}} E$ to be constant):

$$\boldsymbol{H}_k := \vec{\boldsymbol{\nabla}}_{\vec{\boldsymbol{q}}} \vec{\boldsymbol{\nabla}}_{\vec{\boldsymbol{q}}}^T E_k \approx \boldsymbol{J}_k^T \boldsymbol{W} \boldsymbol{J}_k =: \widetilde{\boldsymbol{H}}_k$$

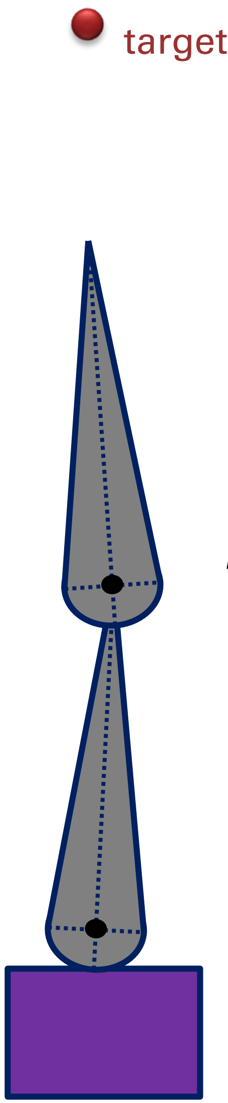
- approximate also Newton direction (this direction is then called Gauss-Newton method):

$$\widetilde{\boldsymbol{H}}_k \boldsymbol{h}_k^{\mathrm{GN}} = -\vec{\boldsymbol{\nabla}}_{\vec{\boldsymbol{q}}}^T E_k \Rightarrow \boldsymbol{J}_k^T \boldsymbol{W} \boldsymbol{J}_k \boldsymbol{h}_k^{\mathrm{GN}} = -\boldsymbol{J}_k^T \boldsymbol{W} \vec{\boldsymbol{r}}_k$$

- as $\boldsymbol{J}$ can become singular at singular configurations, solve for Gauss-Newton direction with truncated SVD.

WNLLS – Drawback with Gauss-Newton

- One problem with the Gauss-Newton method is that the Pseudo inverse becomes unstable in degenerate positions
- These arise in IK for example when the robot arm is fully extended
- This problem does not arise in the steepest descent method where only the transposed of the Jacobian is used
- Idea: combine both methods optimally



$$J = \begin{pmatrix} \frac{\partial \vec{f}}{\partial q_1} & \frac{\partial \vec{f}}{\partial q_2} \end{pmatrix} = \begin{pmatrix} -0.95 & -0.97 \\ 0.08 & -0.06 \end{pmatrix}$$

$$J^{-1} = \begin{pmatrix} -0.45 & 7.2 \\ -0.59 & -7.1 \end{pmatrix}$$

$$d\vec{x} = \begin{pmatrix} 0 \\ 0.5 \end{pmatrix}$$

$$d\vec{q} = \begin{pmatrix} 3.6 \\ -3.5 \end{pmatrix}$$

Levenberg Marquardt

Computer Graphics and Visualization

- Idea: combine steepest descent $\boldsymbol{h}_k^{\mathrm{SD}} = -\boldsymbol{J}_k^T \boldsymbol{W} \vec{\boldsymbol{r}}_k$ with Gauss-Newton approach $\boldsymbol{J}_k^T \boldsymbol{W} \boldsymbol{J}_k \boldsymbol{h}_k^{\mathrm{GN}} = -\boldsymbol{J}_k^T \boldsymbol{W} \vec{\boldsymbol{r}}_k$ with a weighting factor λ:
$$\left(\boldsymbol{J}_k^T \boldsymbol{W} \boldsymbol{J}_k + \lambda \boldsymbol{I}\right) \boldsymbol{h}_k^{\mathrm{LM}} = -\boldsymbol{J}_k^T \boldsymbol{W} \vec{\boldsymbol{r}}_k \qquad [1]$$
- Large λ result in steepest descent update and small λ in Gauss-Newton update. λ is initialized to large values and decreased close to the optimum, which leads to fast convergence (quadratic in best case).
- The simplest approach with quadratic convergence close to solution is to set $\lambda = \|\vec{\boldsymbol{r}}_k\|$
- Levenberg Marquardt suggest to adapt λ-control to the directional curvature by the generalization
$$\left(\boldsymbol{J}_k^T \boldsymbol{W} \boldsymbol{J}_k + \lambda \operatorname{diag}\left(\boldsymbol{J}_k^T \boldsymbol{W} \boldsymbol{J}_k\right)\right) \boldsymbol{h}_k^{\mathrm{LM}} = -\boldsymbol{J}_k^T \boldsymbol{W} \boldsymbol{r}_k \qquad [2]$$

Self-adapting Levenberg Marquardt

- An important criterion to control λ is the fraction between the real improvement and the improvement predicted by the quadratic approximation with $\boldsymbol{J}$ and $\widetilde{\boldsymbol{H}}$:

$$\rho_k(\vec{\boldsymbol{q}}) = \frac{E(\vec{\boldsymbol{q}}) - E\left(\vec{\boldsymbol{q}} + \boldsymbol{h}_k^{\mathrm{LM}}\right)}{2\boldsymbol{h}_k^{\mathrm{LM}^{\mathrm{T}}}\left(\lambda_k \boldsymbol{h}_k^{\mathrm{LM}} - \boldsymbol{J}_k^T \boldsymbol{W} \boldsymbol{r}_k\right)} \qquad [3]$$

- One can update a factor α_k in $\lambda_k = \sqrt{\alpha_k}\|\vec{\boldsymbol{r}}_k\|$ over the optimization resulting in the self-adaptive LM method:
- input from user: $\vec{\boldsymbol{q}}_0, \alpha_0$

1. compute $\|\vec{\boldsymbol{r}}_k\|$ and check for convergence
2. set $\lambda_k = \sqrt{\alpha_k}\|\vec{\boldsymbol{r}}_k\|$
3. solve [1] or [2] for $\boldsymbol{h}_k^{\mathrm{LM}}$ and set $\vec{\boldsymbol{q}}_{k+1} = \vec{\boldsymbol{q}}_k + \boldsymbol{h}_k^{\mathrm{LM}}$
4. compute ρ_k from [3]
5. set $\alpha_{k+1} = \alpha_k \cdot \max\left\{\frac{1}{4}, 1 - 2(2\rho_k - 1)^3\right\}$ and goto 1.

IK Methods

- Descent with steepest descend direction is also called transposed Jacobian method in IK literature. It needs to be combined with a line search method. As no SVD is necessary, a single iteration is fast. The method is therefore often used in interactive IK approaches, e.g. in character pose editors.
- Descent with the Newton direction computed from the approximate Hessian is called Gauss-Newton method or non linear least squares in math and inverse Jacobian method in IK. The natural step width of 1 simplifies line search. One can simply check for decrease in energy and half the step width until an energy decrease is achieved.
- The taxi cab method corresponds to iteratively optimize only one joint parameter at the time. This is called cyclic coordinate descent method in IK and often implemented as the optimal step width can be computed analytically.

IK Methods

- Levenberg-Marquardt and BFGS are better than inverse Jacobian and often used for IK. Furthermore, one can use the non linear conjugate gradient method discussed in CG1.
- Another strategy is to start with cyclic coordinate descent and continue with Levenberg-Marquardt or BFGS.

CONSTRAINED IK

Parameter Constraints

- Most joints have constraints on their parameters which can be defined with a lower and upper bound:
$$l_j \leq q_j \leq u_j$$
- such constraints are often called simple constraints and the optimization problem is called bound constrained.
- In all descent approaches – including the cyclic coordinate descent – these constraints can be easily incorporated by the gradient projection method:
  - First one defines the projection operation Π on the parameter space P:
  $$\Pi\left(\vec{\boldsymbol{q}},\vec{\boldsymbol{l}},\vec{\boldsymbol{u}}\right)_j = \mathrm{median}\{l_j, q_j, u_j\}$$
  - And uses this to project descent directions to the feasible region:
  $$\vec{\bar{\boldsymbol{h}}} = \Pi\left(\vec{\boldsymbol{q}} + \vec{\boldsymbol{h}},\vec{\boldsymbol{l}},\vec{\boldsymbol{u}}\right) - \vec{\boldsymbol{q}}$$
  - The rest of the algorithms stays the same
- Bound constrained versions exists for L-BFGS-B, (C++)

General Constraints

- Positional constraints like that a foot should stay on the floor are non linear in the parameters.
- Such general constraints can be written in the form of equalities or inequalities:

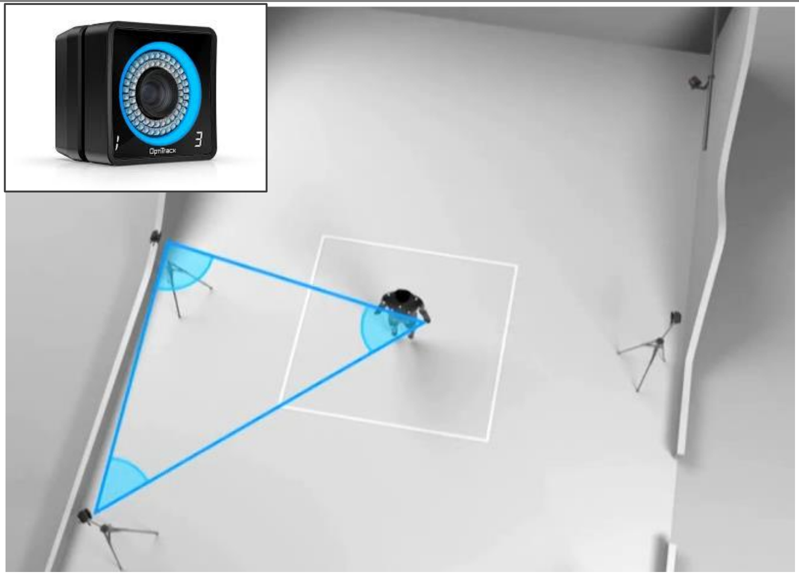
$$c_l(\vec{\boldsymbol{q}}) = 0 \text{ or } h_o(\vec{\boldsymbol{q}}) \geq 0$$

- approaches to incorporate equality constraints:
  - Lagrangian multiplier method incorporates constraints into objective function and adds multipliers as additional parameters
  - constraint forces can be derived from constraints and also be used to reinforce constraints in case of numerical deviations or initialization that violates constraints
- approaches to incorporate inequality constraints
  - Linear complementary problems (LCP)

MOTION CAPTURING

Standard Motion Capturing

- Standard MoCap Approach
  - add markers at joints
  - illuminate markers from all directions
  - acquire views from several synchronized cameras
  - detect markers in each acquired view
  - match markers between views
  - reconstruct 3D positions
  - track points over time
  - per frame match points to skeleton and fit skeleton
- extension: use light emitters of different frequency as markers with id (no matching necessary)



http://www.youtube.com/watch?v=DaOAZA0xSMY

Style-Based Inverse Kinematics

Computer Graphics and Visualization

Keith Grochow, Steven L. Martin, Aaron Hertzmann, Zoran Popovic, Siggraph 2004

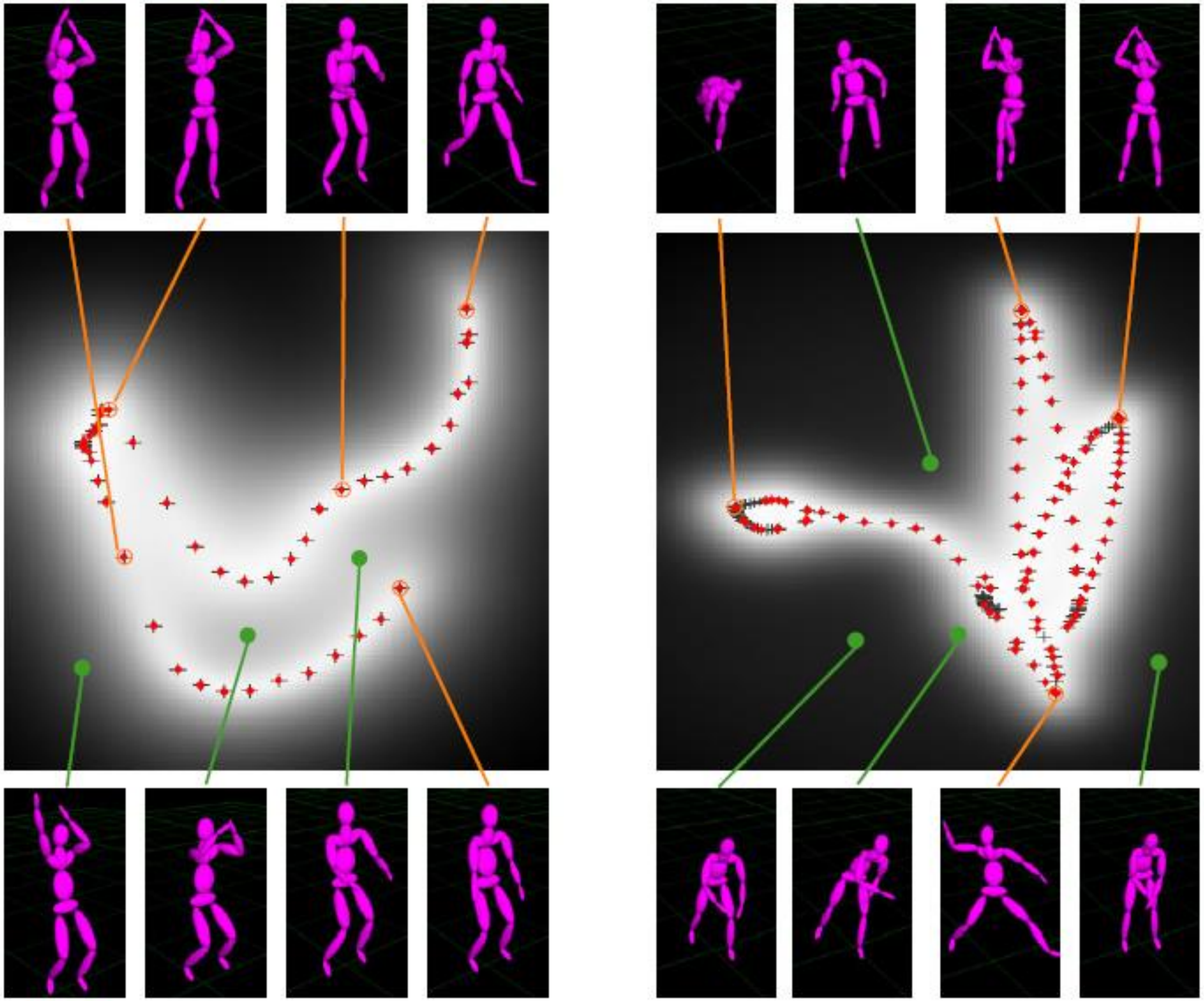
Idea

- use motion capture data to disambiguate IK

Overview

- capture different motion sequences
- map each pose to a feature space
- learn distribution of poses in reduced feature space
- solve IK problem by maximizing pose probability

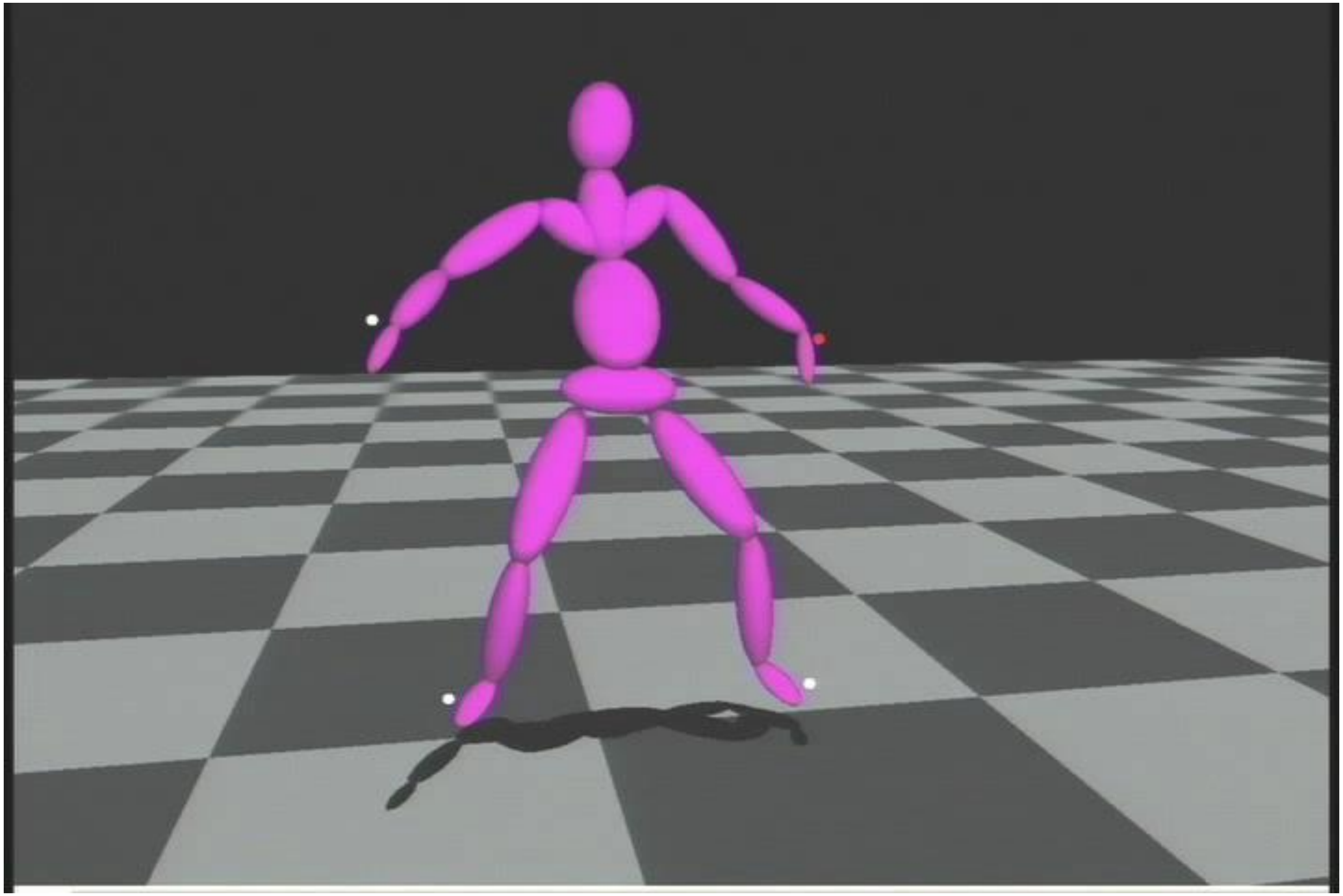
Style-Based Inverse Kinematics



Example for learned pose spaces of jump shot (left) and baseball pitch (right) sequences and their pose probability distributions.

Red points are training poses, orange connections mark some training poses and green connections show extrapolated poses

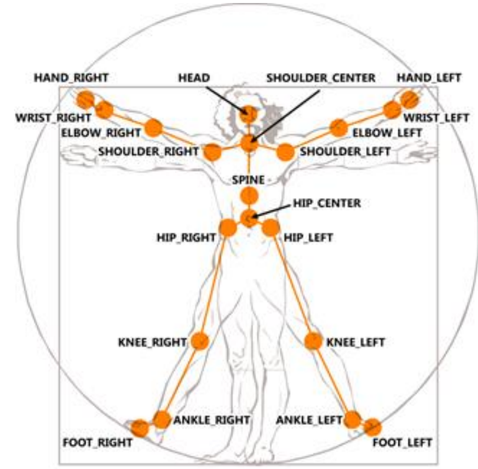
Style-Based Inverse Kinematics



www.youtube.com/watch?v=X5Z7ZJ39zAA

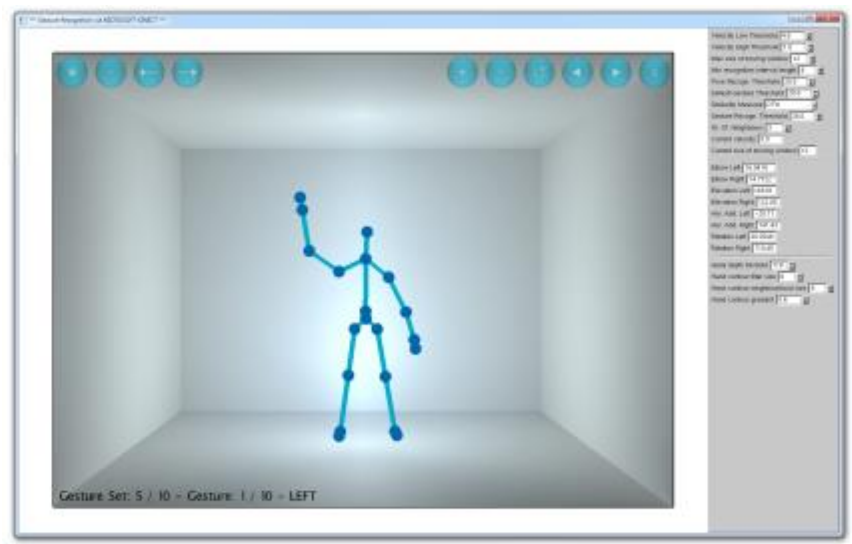
RGBD Motion Capturing

- uses depth camera with a machine learning approach
- learning
  - 2D parameterized template human mesh (vitruvian manifold)
  - generate a huge number of depth images from different poses of vitruvian manifold (render depth & texture coords.)
  - learn local image features with decision tree to map each depth image pixel to texture coordinates
- recognition
  - use decision tree to estimate per pixel texcoords
  - fit pose to depth and texcoord image



kinect skeleton

[vitruvian manifold](#)
Taylor, Shotton, Sharp, Fitzgibbon
CVPR 2012



© Ludwig Schmutzler

Interactive Motion Mapping

- Helge Rhodin, James Tompkin, Kwang In Kim, Kiran Varanasi, Hans-Peter Seidel, Christian Theobalt, Eurographics 2014 (http)

Idea

- use Kinect skeleton tracking to steer character with different skeleton

Overview

- Given sparse pose mapping from source to target, learn pose mapping without rigging and skinning
- Allows interactive control of virtual character

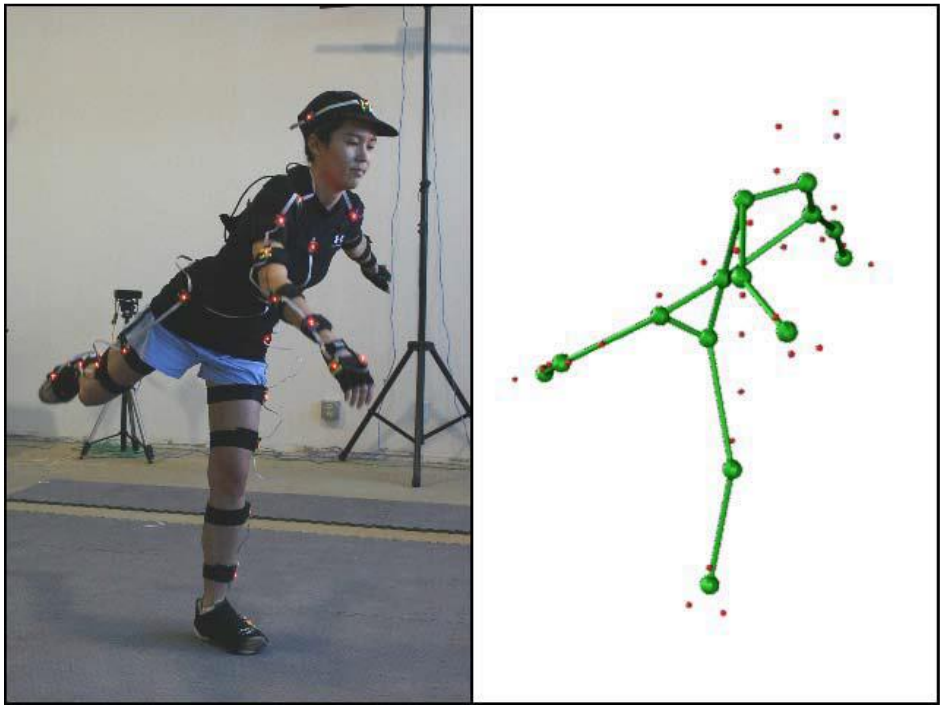
Interactive Motion Mapping

www.youtube.com/watch?v=SG5D12tBBAk

SKELETON FITTING

Problem Statement

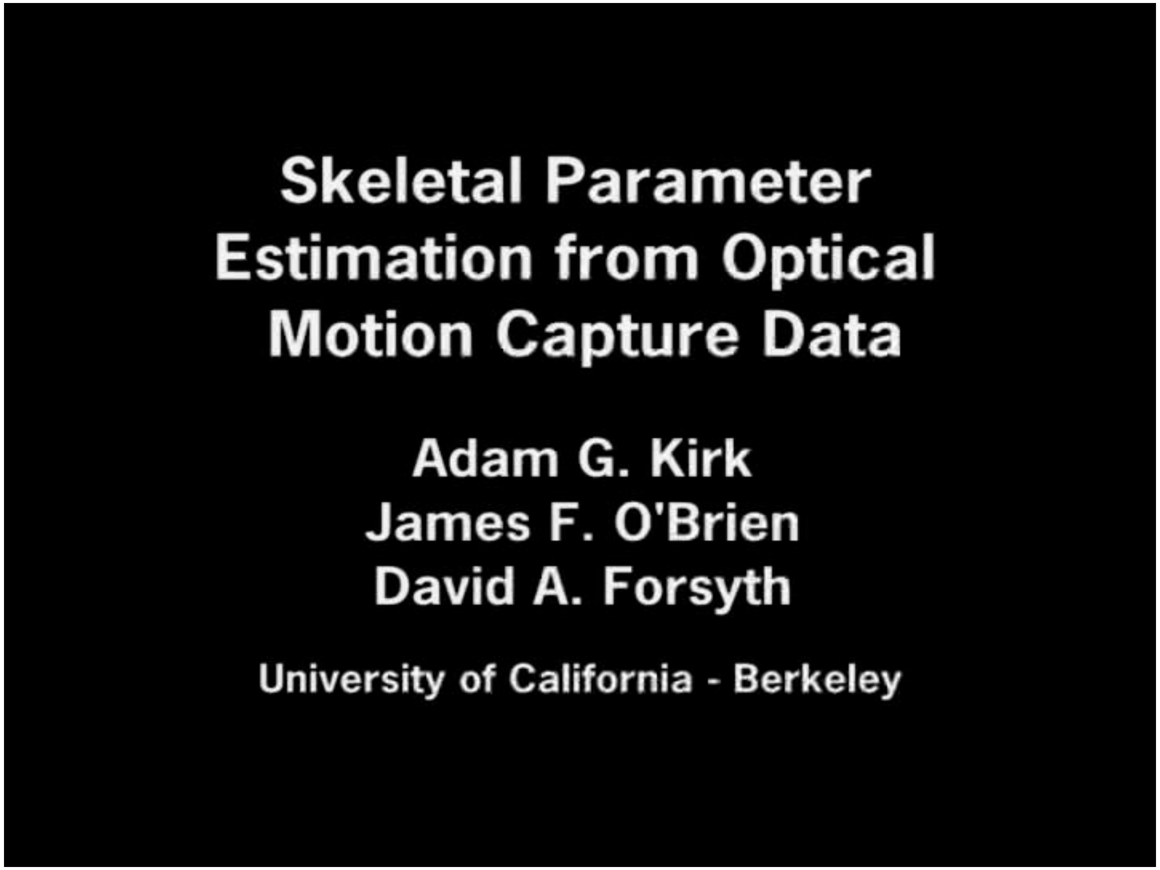
- Input: set of 3D marker points tracked over time where markers can be occluded and re-appearing markers are not identified but receive a new label
- Output: skeleton topology, bone lengths, joint locations in rotation centers (markers are placed on surface)



Reference: Adam G. Kirk James F. O'Brien David A. Forsyth, *Skeletal Parameter Estimation from Optical Motion Capture Data*, CVPR 2015

Solution approach

- filter erroneous marker information before vanishing and after their re-appearance
- build matrix of marker pair distances and measure pair distance variance over time
- segment markers into rigid components by spectral clustering
- build all segment pairs and fit joint locations to minimize variance of distance to both segment markers over time
- extract topology by minimum spanning tree
- match marker time slabs
- use IK to extract joint parameters



http://graphics.berkeley.edu/papers/Kirk-SPE-2005-06

References

(1) Wang, L-CT, and Chih-Cheng Chen. "A combined optimization method for solving the inverse kinematics problems of mechanical manipulators." *Robotics and Automation, IEEE Transactions on* 7.4 (1991): 489-499.

(2) Welman, Chris. *Inverse kinematics and geometric constraints for articulated figure manipulation*. Diss. Simon Fraser University, 1993.

(3) Meredith, Michael, and Steve Maddock. *Real-time inverse kinematics: The return of the Jacobian*. Technical Report No. CS-04-06, Department of Computer Science, University of Sheffield, 2004.

(4) Hess, Peter, et al. "Style-Based Inverse Kinematics." (2005).

(5) Shan, Shidong. *A Levenberg-Marquardt method for large-scale bound-constrained nonlinear least-squares*. Diss. The University of British Columbia (Vancouver), 2008.

(6) Taylor, Jonathan, et al. "The Vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation." *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012.

(7) Kenwright, Ben. "Inverse Kinematics with Dual-Quaternions, Exponential-Maps, and Joint Limits." *International Journal On Advances in Intelligent Systems* 6.1 and 2 (2013): 53-65.

(8) Rhodin, Helge, et al. "Interactive motion mapping for real-time character control." *Computer Graphics Forum*. Vol. 33. No. 2. 2014.

(9) Adam G. Kirk James F. O’Brien David A. Forsyth, *Skeletal Parameter Estimation from Optical Motion Capture Data*, CVPR 2015

(10) http://chrishecker.com/Inverse_kinematics

(11) https://github.com/PatWie/CppNumericalSolvers (L-BFGS-B Solver in C++)