Physics based animation

Lecture 02 - Kinematics

Grégory Leplâtre

g.leplatre@napier.ac.uk, room D32
School of Computing
Edinburgh Napier University

Semester 1 - 2016/2017

Objectives

- Kinematics equations

Objectives

- Kinematics equations
- Forward Kinematics (easy)

Objectives

- Kinematics equations
- Forward Kinematics (easy)
- Inverse Kinematics (more difficult)

Outline

Kinematics vs dynamics

- **Kinematics**: the branch of mechanics that studies the motion of a body or system without consideration for its mass or forces acting on it:
  - **geometry of motion**
  - Omniscient observer that puts all object into motion **instantaneously**
  - **massless** and **forceless**
  - No consideration for **past** history of all other objects in the universe
  - displacement/position, velocity and acceleration

Kinematics vs dynamics

- **Kinematics**: the branch of mechanics that studies the motion of a body or system without consideration for its mass or forces acting on it:
  - **geometry of motion**
  - Omniscient observer that puts all object into motion **instantaneously**
  - **massless** and **forceless**
  - No consideration for **past** history of all other objects in the universe
  - displacement/position, velocity and acceleration
- **Dynamics**: Behaviour of an object/system subjected to forces

Kinematic equations of motion

Kinematic equations of motion

- Variables: (u, v, s, t, a)
  - s=distance
  - v=velocity
  - u=initial velocity
  - a=acceleration
  - t=time
- Four equations:
  - $s = t(u + v)/2$
  - $v = u + at$
  - $s = ut + (1/2)at^2$
  - $v^2 = u^2 + 2as$
- **Important**: valid for constant acceleration

Applications

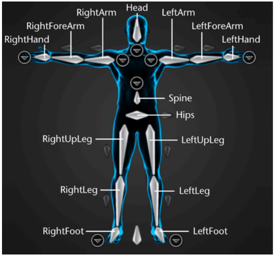
- Particle trajectories (under constant acceleration)
- Robotics
- Character kinematics

Question: what if acceleration isn't constant?

Question: what if acceleration isn't constant?
Answer: **dynamic** equations of motions are considered (forces)

Outline

Hierarchies

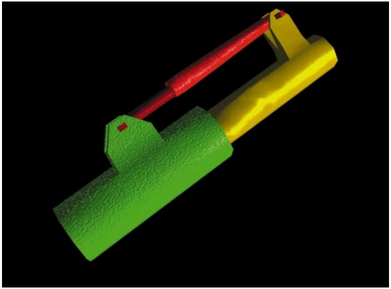


- Common ways of representing *chained* mechanical systems: robots, humans, creatures
- Human representation:
  - bones are **directional**
  - root: usually hips or **pelvis**
- Control: interesting usability question
  - Usable by programmers (API)
  - Usable by Technical Director (higher-level tool)
  - Usable by animator (rig)

Degrees of freedom

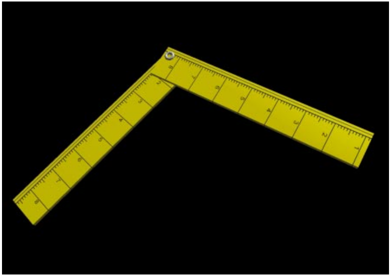
- Number of independent variables required to describe motion:

Degrees of freedom



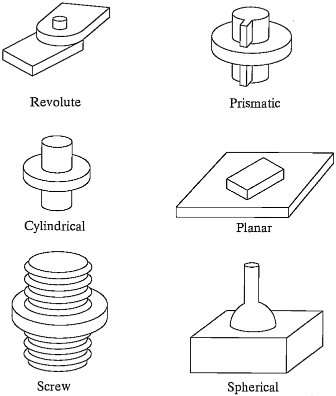
- Number of independent variables required to describe motion:
- up to 3 translations

Degrees of freedom



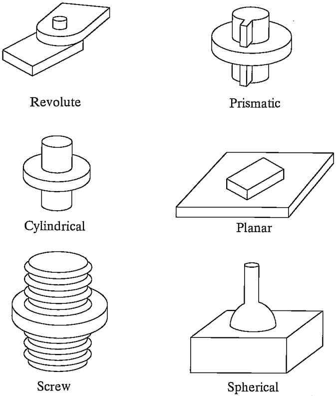
- Number of independent variables required to describe motion:
- up to 3 translations
- up to 3 rotations

Degrees of freedom



- Number of independent variables required to describe motion:
- up to 3 translations
- up to 3 rotations

Degrees of freedom

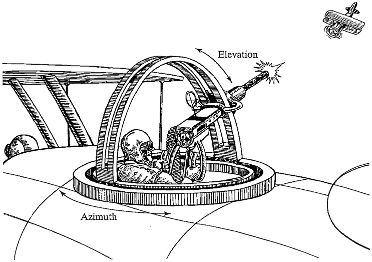


- Number of independent variables required to describe motion:
- up to 3 translations
- up to 3 rotations
- For most mechanical systems:

Degrees of freedom

- Number of independent variables required to describe motion:
- up to 3 translations
- up to 3 rotations
- For most mechanical systems:
  - Limited number of DoF
  - limited range

Degrees of freedom



- Number of independent variables required to describe motion:
- up to 3 translations
- up to 3 rotations
- For most mechanical systems:
  - Limited number of DoF
  - limited range
  - singularities!

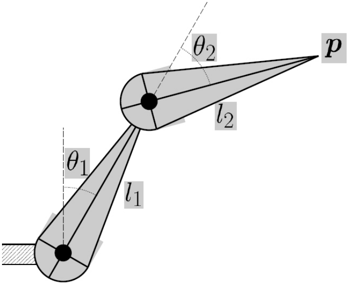
Forward Kinematics and Inverse Kinematics

Forward Kinematics

Inverse Kinematics

Outline

Forward Kinematics: General problem

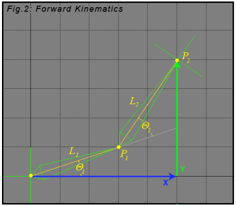


Find f such that:

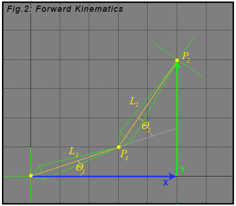
$$\boxed{p = f(\theta)}$$

where:
p is the position of the end effector.
$\theta = (\theta_1, \theta_2, ..., \theta_n)$

Two joint chain

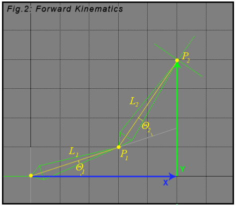


Two joint chain



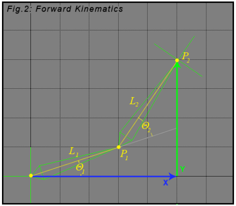
- P1:
  $$P_{x_1} = L_1.cos(\theta_1)$$
  $$P_{y_1} = L_1.sin(\theta_1)$$

Two joint chain



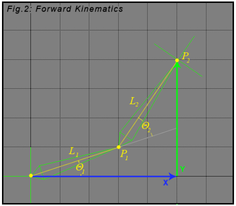
- P1:
$$P_{x_1} = L_1.cos(\theta_1)$$
$$P_{y_1} = L_1.sin(\theta_1)$$
- P2:
$$P_{x_2} = P_{x_1} + L_2.cos(\theta_1 + \theta_2)$$
$$P_{y_2} = P_{y_1} + L_2.sin(\theta_1 + \theta_2)$$

Two joint chain



- P1:
  $$P_{x_1} = L_1.cos(\theta_1)$$
  $$P_{y_1} = L_1.sin(\theta_1)$$
- P2:
  $$P_{x_2} = P_{x_1} + L_2.cos(\theta_1 + \theta_2)$$
  $$P_{y_2} = P_{y_1} + L_2.sin(\theta_1 + \theta_2)$$
- P2 (expanded):
  $$P_{x_2} = L_1.cos(\theta_1) + L_2.cos(\theta_1 + \theta_2)$$
  $$P_{y_2} = L_1.sin(\theta_1) + L_2.sin(\theta_1 + \theta_2)$$

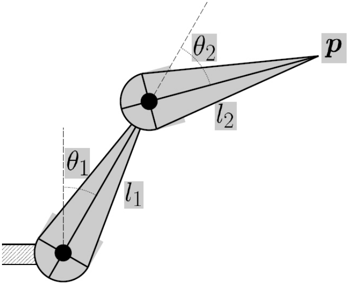
Two joint chain



- P1:
  $$P_{x_1} = L_1.cos(\theta_1)$$
  $$P_{y_1} = L_1.sin(\theta_1)$$
- P2:
  $$P_{x_2} = P_{x_1} + L_2.cos(\theta_1 + \theta_2)$$
  $$P_{y_2} = P_{y_1} + L_2.sin(\theta_1 + \theta_2)$$
- P2 (expanded):
  $$P_{x_2} = L_1.cos(\theta_1) + L_2.cos(\theta_1 + \theta_2)$$
  $$P_{y_2} = L_1.sin(\theta_1) + L_2.sin(\theta_1 + \theta_2)$$
- Easily generalisable to *n* joints

Outline

Inverse Kinematics: General problem



Resolve:

$$\boxed{\theta = f^{-1}(p)}$$

where:
p is the position of the end effector.
$\theta = (\theta_1, \theta_2, ..., \theta_n)$

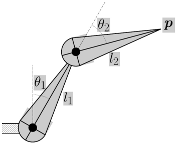
IK solving techniques

- **Closed form solutions**: can be expressed analytically i.e., algebraically or geometrically.
  - only possible/practical in simple scenarios
  - systems with revolute and prismatic joints with up to 6 DoF solvable.
  - there isn't always one and one unique solution

IK solving techniques

- **Closed form solutions**: can be expressed analytically i.e., algebraically or geometrically.
  - only possible/practical in simple scenarios
  - systems with revolute and prismatic joints with up to 6 DoF solvable.
  - there isn't always one and one unique solution
- **Numerical solutions**: finds the solution iteratively by minimising the distance to a target.
  - Cyclic Coordinate Descent (CCD)
  - Jacobian Transpose
  - Levenberg-Marquardt Damped Least Squares (DLS)
  - Neural net and AI

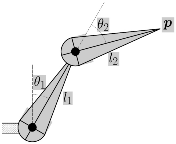
Analytical solution (geometric)



Two joint chain, one DoF

$p : (x, y)$

Analytical solution (geometric)

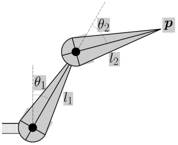


Two joint chain, one DoF

$p : (x, y)$

$$\theta_2 = acos\left(\frac{l_1^2 + l_2^2 - (x^2 + y^2)}{2 l_1 l_2}\right)$$

Analytical solution (geometric)



Two joint chain, one DoF

$p : (x, y)$

$$\theta_2 = acos\left(\frac{l_1^2 + l_2^2 - (x^2 + y^2)}{2 l_1 l_2}\right)$$

$$\theta_1 = atan\left(\frac{x(l_2 cos(\theta_2) + l_1) - y l_2 sin(\theta_2)}{y(l_2 cos(\theta_2) + l_1) - x l_2 sin(\theta_2)}\right)$$

Intuitive algorithm?

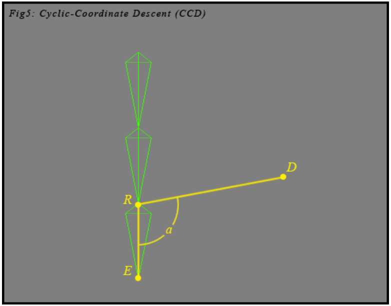
Intuitive algorithm?

Cyclic Coordinate Descent

Cyclic Coordinate Descent

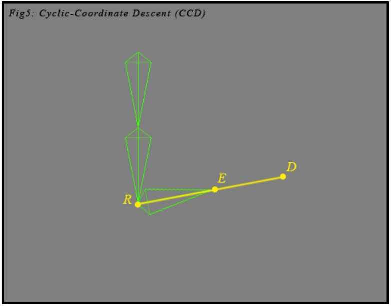
- solve 1 DoF problem recursively.

Cyclic Coordinate Descent



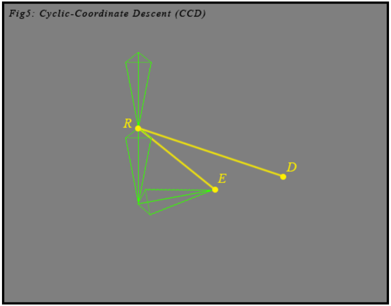
- solve 1 DoF problem recursively.
- **Example 1**: 3 link chain resolution

Cyclic Coordinate Descent



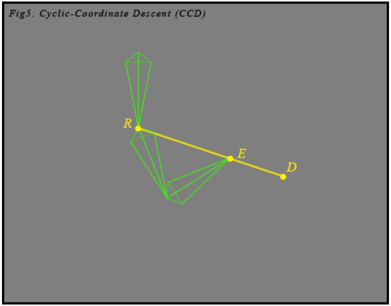
- solve 1 DoF problem recursively.
- **Example 1**: 3 link chain resolution

Cyclic Coordinate Descent



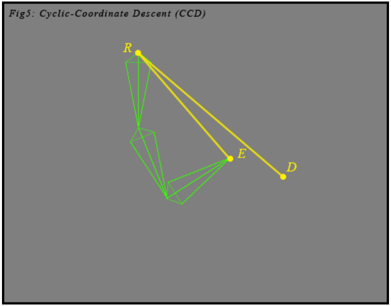
- solve 1 DoF problem recursively.
- **Example 1**: 3 link chain resolution

Cyclic Coordinate Descent



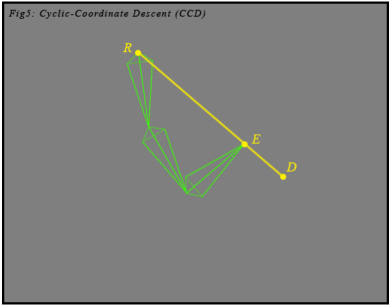
- solve 1 DoF problem recursively.
- **Example 1**: 3 link chain resolution

Cyclic Coordinate Descent



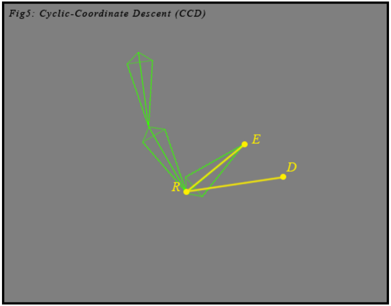
- solve 1 DoF problem recursively.
- **Example 1**: 3 link chain resolution

Cyclic Coordinate Descent



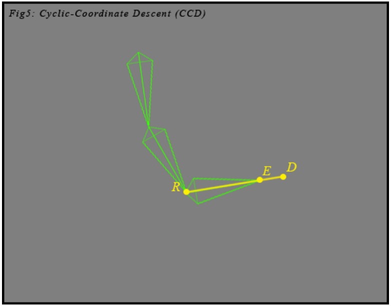
- solve 1 DoF problem recursively.
- **Example 1**: 3 link chain resolution

Cyclic Coordinate Descent



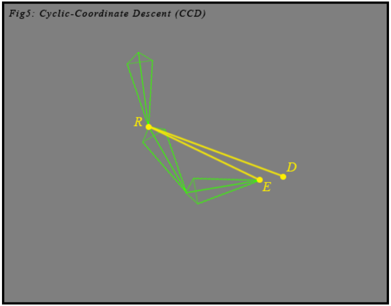
- solve 1 DoF problem recursively.
- **Example 1**: 3 link chain resolution

Cyclic Coordinate Descent



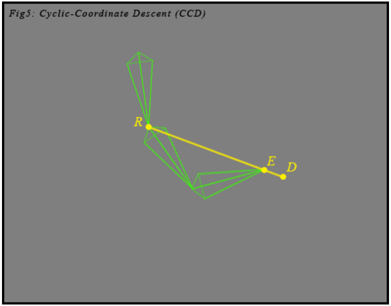
- solve 1 DoF problem recursively.
- **Example 1**: 3 link chain resolution

Cyclic Coordinate Descent



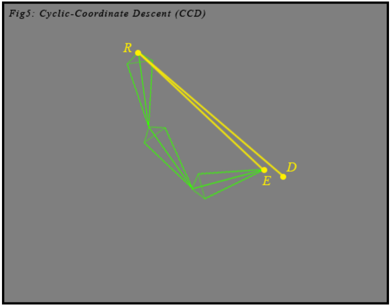
- solve 1 DoF problem recursively.
- **Example 1**: 3 link chain resolution

Cyclic Coordinate Descent



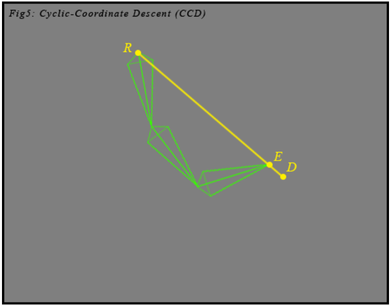
- solve 1 DoF problem recursively.
- **Example 1**: 3 link chain resolution

Cyclic Coordinate Descent



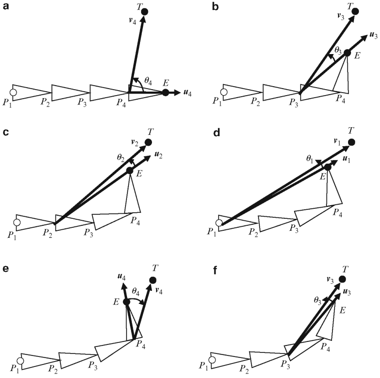
- solve 1 DoF problem recursively.
- **Example 1**: 3 link chain resolution

Cyclic Coordinate Descent



- solve 1 DoF problem recursively.
- **Example 1**: 3 link chain resolution

Cyclic Coordinate Descent



- solve 1 DoF problem recursively.
- **Example 1**: 3 link chain resolution
- **Example 2**: 4 link chain resolution

CCD - Strengths and weaknesses

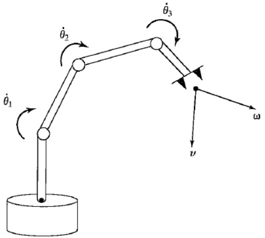
- + simple to implement
- + computationally cheap
- + scalable
- - biased towards end of chain rotations
- - may lead to odd solutions

CCD - Optimisation/modification

- Optimisation ideas:
  - Restricting DoF
  - Damping (limit the range of rotation of joints)
  - Overshooting the target
- Extension ideas: later in the lecture ...

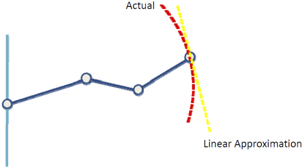
Jacobian technique

What is a Jacobian?



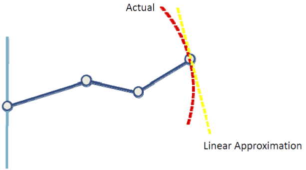
- Mapping from velocities in **joint space** to velocities in **Cartesian space**.

What is a Jacobian?



- Mapping from velocities in **joint space** to velocities in **Cartesian space.**
- A linear approximation of $f()$

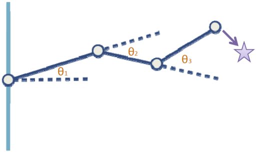
What is a Jacobian?



- Mapping from velocities in **joint space** to velocities in **Cartesian space**.
- A linear approximation of $f()$
- Defines how the end effector p changes relative to instantaneous changes in the system with regards to θ.

What is a Jacobian?

Mathematically:



$$\boxed{J = \frac{dp}{d\theta}}$$

$$J = \begin{pmatrix} \frac{\partial p_x}{\partial \theta_1} & \frac{\partial p_x}{\partial \theta_2} & \cdots & \frac{\partial p_x}{\partial \theta_m} \\ \frac{\partial p_y}{\partial \theta_1} & \frac{\partial p_y}{\partial \theta_2} & \cdots & \frac{\partial p_y}{\partial \theta_m} \\ \frac{\partial p_z}{\partial \theta_1} & \frac{\partial p_z}{\partial \theta_2} & \cdots & \frac{\partial p_z}{\partial \theta_m} \end{pmatrix}$$

What is a Jacobian?

We can therefore approximate small variations $\Delta\theta$ with:

$$J = \frac{dp}{d\theta} \approx \frac{\Delta p}{\Delta\theta}$$

What is a Jacobian?

We can therefore approximate small variations $\Delta\theta$ with:

$$J = \frac{dp}{d\theta} \approx \frac{\Delta p}{\Delta\theta}$$

$$J.\Delta\theta = \Delta p$$

What is a Jacobian?

We can therefore approximate small variations $\Delta\theta$ with:

$$J = \frac{dp}{d\theta} \approx \frac{\Delta p}{\Delta\theta}$$

$$J.\Delta\theta = \Delta p$$

$$\boxed{\Delta\theta = J^{-1}\Delta p}$$

Next steps

1. Jacobian must be computed
   - Analytically
   - Numerically

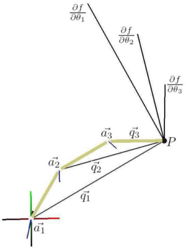
Next steps

1. Jacobian must be computed
   - Analytically
   - Numerically
2. Jacobian must be inverted
   - Not easy!

Next steps

1. Jacobian must be computed
   - Analytically
   - Numerically
2. Jacobian must be inverted
   - Not easy!
3. Iterate

1. Computing the Jacobian (analytically)



- For a rotational joint, the linear change in the end effector is the cross product of the axis of revolution and a vector from the joint to the end effector

$$\frac{\partial p}{\partial \theta_i} = \left[\frac{\partial p_x}{\partial \theta_i}, \frac{\partial p_y}{\partial \theta_i}, \frac{\partial p_z}{\partial \theta_i} \right]^T = \vec{a}_i \times \vec{q}_i$$

2. Inverting the Jacobian

- J may not be square and/or invertible

2. Inverting the Jacobian

- J may not be square and/or invertible
- Classic solutions:

2. Inverting the Jacobian

- J may not be square and/or invertible
- Classic solutions:
  - Jacobian Transpose
    Faster, but less efficient

2. Inverting the Jacobian

- J may not be square and/or invertible
- Classic solutions:
  - Jacobian Transpose
    Faster, but less efficient
  - Pseudo-inverse (Moore-Penrose Inverse)

2. Inverting the Jacobian

- J may not be square and/or invertible
- Classic solutions:
  - Jacobian Transpose
    Faster, but less efficient
  - Pseudo-inverse (Moore-Penrose Inverse)
  - Singular Value Decomposition (SDV)

2. Inverting the Jacobian

- J may not be square and/or invertible
- Classic solutions:
  - Jacobian Transpose
    Faster, but less efficient
  - Pseudo-inverse (Moore-Penrose Inverse)
  - Singular Value Decomposition (SDV)
  - Damped Least Squares (DLS)

2. Inverting the Jacobian

- J may not be square and/or invertible
- Classic solutions:
  - Jacobian Transpose
    Faster, but less efficient
  - Pseudo-inverse (Moore-Penrose Inverse)
  - Singular Value Decomposition (SDV)
  - Damped Least Squares (DLS)
  - Pseudoinverse DLS

2. Inverting the Jacobian - Transpose

$$\Delta\theta = J^{-1}\Delta p$$

approximated by

$$\boxed{\Delta\theta = \alpha J^T \Delta\vec{e}}$$

Where:
$\vec{e}$ desired change of effector

$$alpha = \frac{\vec{e} \cdot J J^T \vec{e}}{J J^T \vec{e} \cdot J J^T \vec{e}}$$

2. Inverting the Jacobian - Pseudo-inverse

$$\Delta\theta = J^{-1}\Delta p$$

approximated by

$$\boxed{\Delta\theta = J^{\dagger}\vec{e}}$$

Where:
$\vec{e}$ desired change of effector

$$\boxed{J^{\dagger} = (J^T J)^{-1} J^T}$$

3. Iteration

```
while dp is larger than tolerance{
    dp = p - pTarget
    compute J
    compute J* = approximation of inv(J)
    compute dAngle = J* dX
    Angle = Angle + dAngle // update joint angle
    compute new p
}
```

Outline

Summary

- To do this week:
  - Work through tutorial
  - Practical implementation of IK solver (lab)

Summary

- To do this week:
  - Work through tutorial
  - Practical implementation of IK solver (lab)
- Looking forward (potential project)

Summary

- To do this week:
  - Work through tutorial
  - Practical implementation of IK solver (lab)
- Looking forward (potential project)
  - Real-time giant snake arm

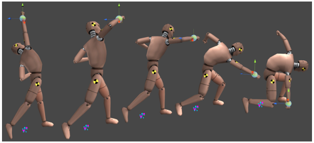
Summary

- To do this week:
  - Work through tutorial
  - Practical implementation of IK solver (lab)
- Looking forward (potential project)
  - Real-time giant snake arm
  - Dealing with obstacles

Summary

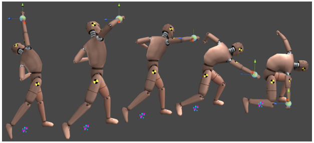
- To do this week:
  - Work through tutorial
  - Practical implementation of IK solver (lab)
- Looking forward (potential project)
  - Real-time giant snake arm
  - Dealing with obstacles
  - Adding DoF

Summary



- To do this week:
  - Work through tutorial
  - Practical implementation of IK solver (lab)
- Looking forward (potential project)
  - Real-time giant snake arm
  - Dealing with obstacles
  - Adding DoF
  - Human IK / full-body IK (illustration from root-motion.com)

Summary



- To do this week:
  - Work through tutorial
  - Practical implementation of IK solver (lab)
- Looking forward (potential project)
  - Real-time giant snake arm
  - Dealing with obstacles
  - Adding DoF
  - Human IK / full-body IK (illustration from root-motion.com)
  - Performance/optimisation
  - etc (see literature)

References

- Aristidou, A. and Lasenby, J. (2009), Inverse Kinematics: a review of existing techniques and introduction of a new fast iterative solver. tech. rep., Department of engineering,Cambridge University, 2009
- Buss, S. (2009) Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods
- Lander, J. (1998), Oh My God, I inverted Kine! *Game Developer*, Sept 1998
- Lander, J. (1998), Making Kine More Flexible *Game Developer*, Nov 1998
- Meredith, M. and Maddock (2004), S., Real-Time Inverse Kinematics: The Return of the Jacobian. tech. rep., Department of Computer Science, University of Sheffield