Multi Body Dynamics with Control

#14

Serdar ARITAN

Biomechanics Research Group,
Faculty of Sports Sciences, and
Department of Computer Graphics
Hacettepe University, Ankara, Turkey

Multibody system

- Multibody system is the study of the dynamic behavior of interconnected rigid or flexible bodies, each of which may undergo large translational and rotational displacements
- The simplest bodies or elements of a multibody system were treated by **Newton** (free particle) and **Euler** (rigid body). Euler introduced reaction forces between bodies. Later, a series of formalisms were derived, only to mention Lagrange's formalisms based on minimal coordinates and a second formulation that introduces constraints.

Multibody system

While single bodies or parts of a mechanical system are studied in detail with finite element methods, the behavior of the whole multibody system is usually studied with multibody system methods within the following areas:

- Aerospace engineering
- Biomechanics
- Combustion engine, gears and transmissions, chain drive, belt drive
- Military applications
- Particle simulation (granular media, sand, molecules)
- **Physics engine**
- Robotics

Multibody system

A body is usually considered to be a rigid or flexible part of a mechanical system (not to be confused with the human body). An example of a body is the arm of a robot, a wheel or axle in a car or the human forearm. A link is the connection of two or more bodies, or a body with the ground. The link is defined by certain (kinematical) constraints that restrict the relative motion of the bodies.

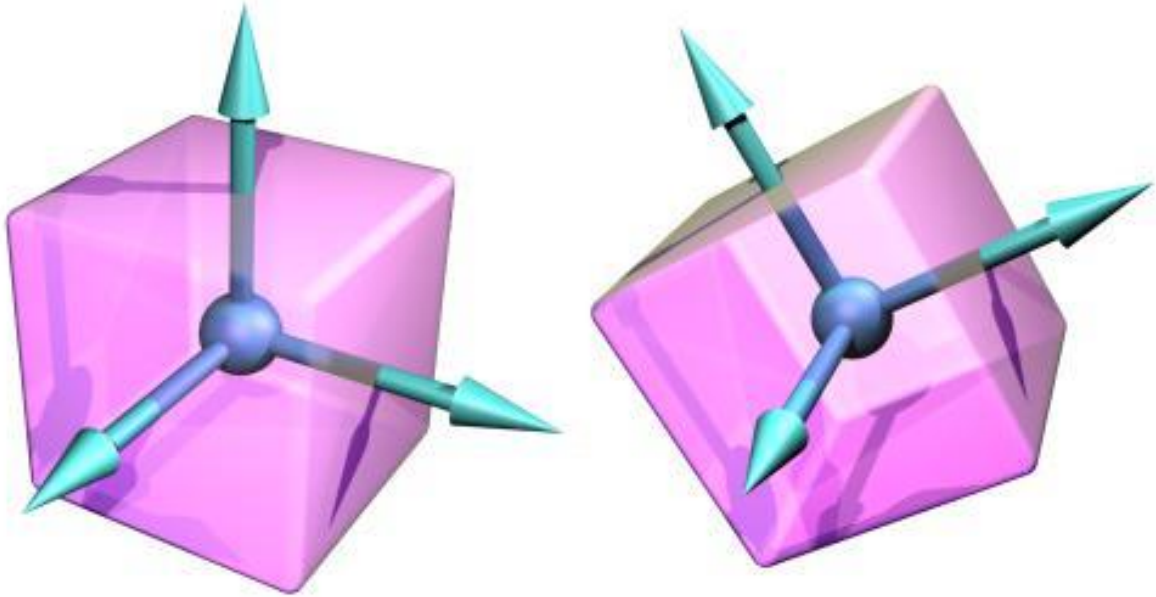
There are two important terms in multibody systems: **degree of freedom** and **constraint condition**.

Multibody system

The **degrees of freedom** denote the number of independent kinematical possibilities to move. In other words, degrees of freedom are the minimum number of parameters required to completely define the position of an entity in space.

A rigid body has six degrees of freedom in the case of general spatial motion, three of them translational degrees of freedom and three rotational degrees of freedom. In the case of planar motion, a body has only three degrees of freedom with only one rotational and two translational degrees of freedom

Rigid bodies



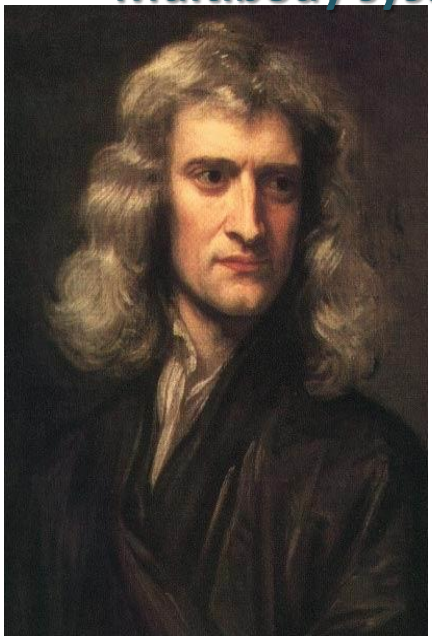
Multibody system

A **constraint condition** implies a restriction in the kinematical degrees of freedom of one or more bodies. The classical constraint is usually an algebraic equation that defines the relative translation or rotation between two bodies. There are furthermore possibilities to constrain the relative velocity between two bodies or a body and the ground. There are furthermore possibilities to constrain the relative velocity between two bodies or a body and the ground. This is for example the case of a rolling disc, where the point of the disc that contacts the ground has always zero relative velocity with respect to the ground. In the case that the velocity constraint condition cannot be integrated in time in order to form a position constraint, it is called ***non-holonomic***. This is the case for the general rolling constraint.

Multibody system

The equations of motion are used to describe the dynamic behavior of a multibody system. Each multibody system formulation may lead to a different mathematical appearance of the equations of motion while the physics behind is the same. The motion of the constrained bodies is described by means of equations that result basically from Newton's second law. The equations are written for general motion of the single bodies with the addition of constraint conditions. Usually the equations of motions are derived from the **Newton-Euler** equations or **Lagrange**'s equations.

Multibody system

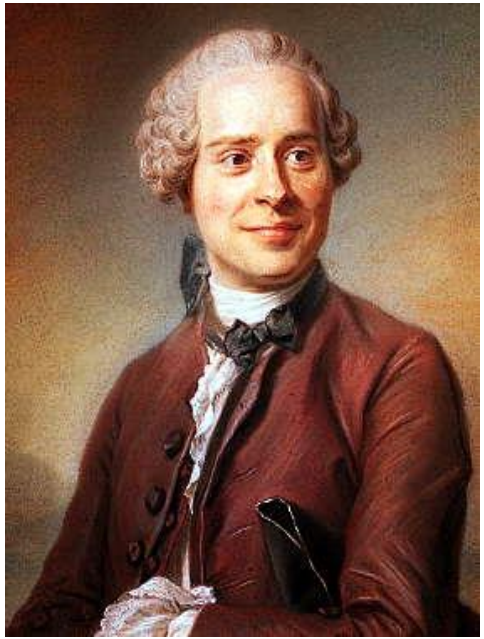


Newton

(1643 -1727)

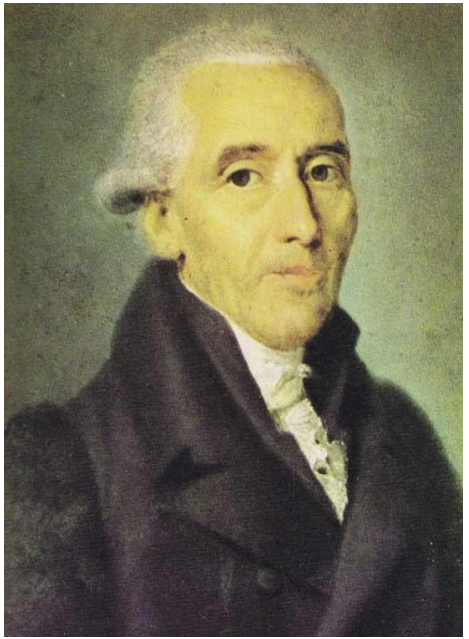
Euler

(1707 -1783)



D'Alembert

(1717 -1783)



Lagrance

(1736 -1813)

Multibody system

Lagrangian Dynamics

Lagrange's equations of motion are specified in terms of the total energy of the body in the kinematic chain.

Newton-Euler Dynamics

The Newton-Euler equations are applied to each body in the model. All forces affecting each body must be considered, which makes this method difficult and tedious for complex systems.

Multibody system

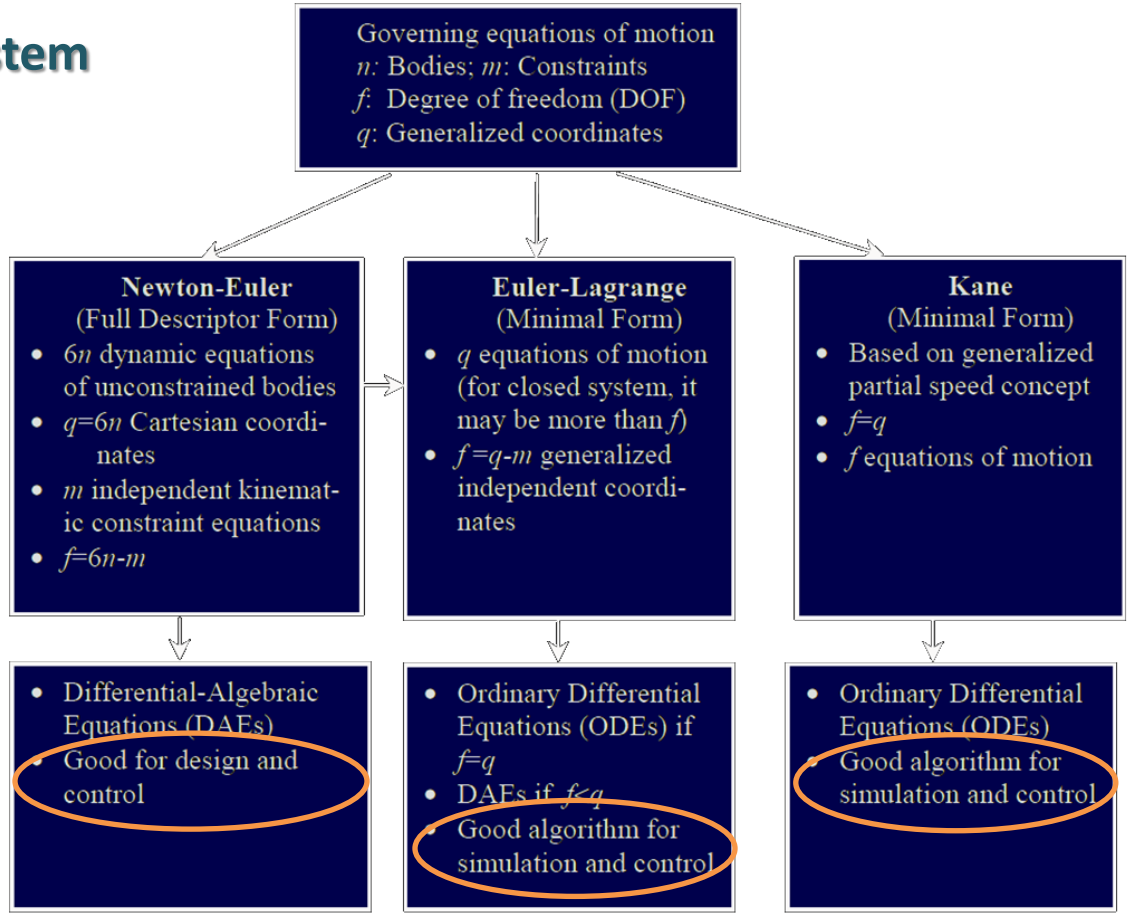
D'Alembert's Principle

Equations of motion are derived by identifying all forces on each body go through an acceleration and writing equilibrium equations. These equilibrium equations are simultaneously solved to obtain the dynamic system response.

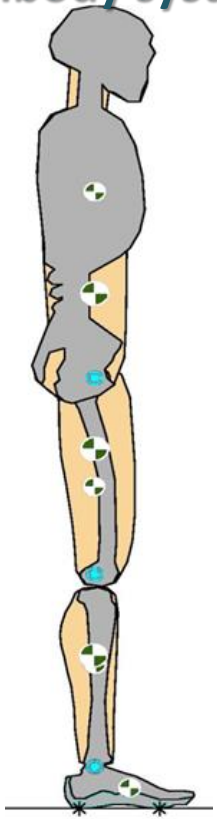
Kane's Dynamics

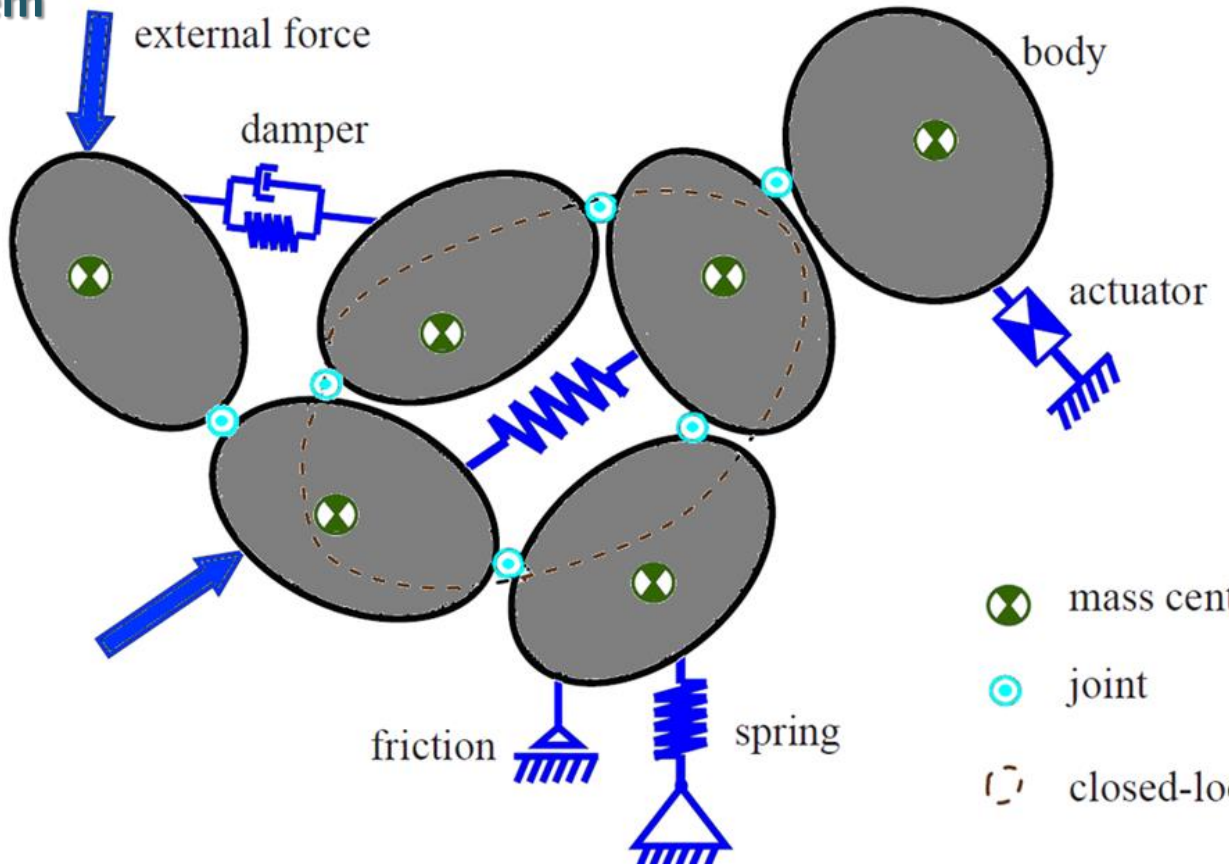
This method is a subset of the group of methods known as "Lagrange's form of D'Alembert's Principle". The Newton-Euler equations are multiplied by 'special vectors' to develop scalar representations of the forces acting on each body.

Multibody system



Multibody system





Rigid bodies

- Position vector of the body's point of reference. Currently the point of reference correspond to the body's center of mass.
- Linear velocity of the point of reference, a vector.
- Orientation of a body, represented by a quaternion or a 3x3 rotation matrix.
- Angular velocity vector which describes how the orientation changes over time.
- Mass of the body.
- Inertia matrix. This is a 3x3 matrix that describes how the body's mass is distributed around the center of mass.

Multibody system- Constraints

From the physical point of view, constraints on a mechanical system are conditions restricting possible geometrical positions of the mechanical system or limiting its motion. We distinguish between geometric and kinematic constraints.

Constraints are called geometric or holonomic if they are expressed by equations of the form;

$$f(q,t) = 0$$

Holonomic constraints are called skleronomic if they do not depend explicitly on time.

$$f(q) = 0$$

Multibody system- Constraints

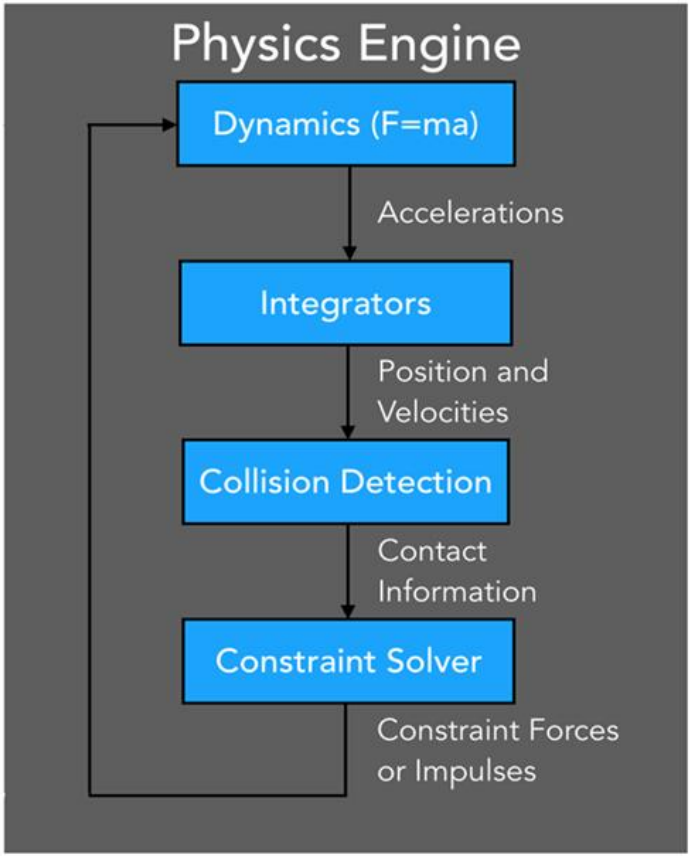
Constraints are called kinematic if they are expressed by

$$f(q, \dot{q}, t) = 0$$

Nonintegrable kinematic constraints, which cannot be reduced to geometric ones are called nonholonomic constraints.

Holonomic or nonholonomic constraints which depend explicitly on time are called rheonomic.

PHYSICS in COMPUTER ANIMATIONS and GAMES



The integrator is responsible for calculating a body's position given the forces acting on it.

Materials Restitution

If constraints are unstable numerical errors can cause constrained bodies (bones) to slowly drift apart

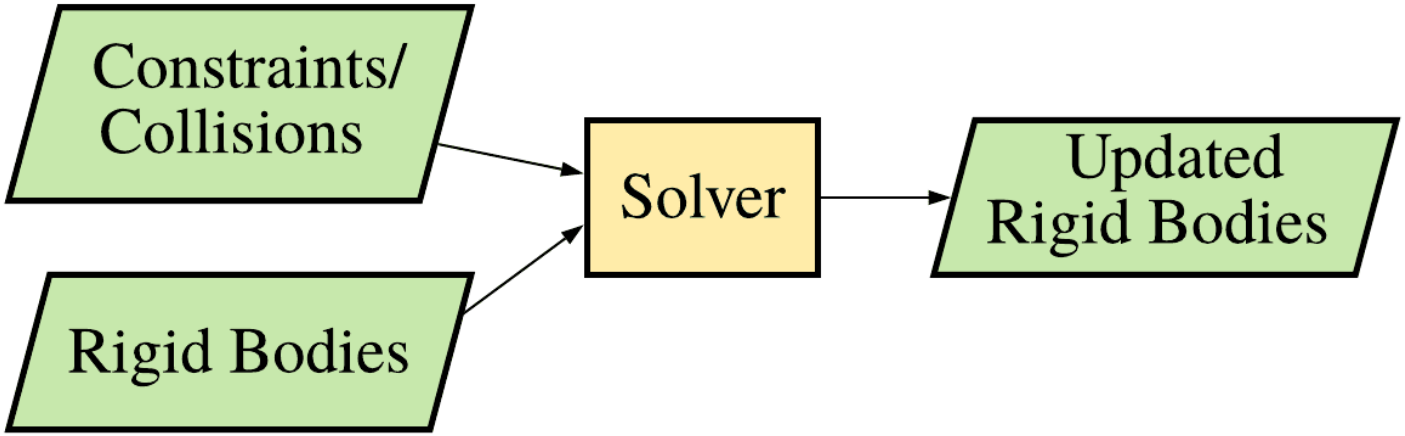
Minimum 60 times in a second

Constraint Solvers

- A physics engine is organized into two phases: collision detection and solving.
- **Collision detection** finds intersections between geometries associated with the rigid bodies, generating appropriate collision information such as collision points, normals and penetration depths.
- Then a solver updates the motion of rigid bodies under the influence of the **collisions that were detected and constraints that were provided by the user**.

Constraint Solvers

- The main objective of a physics engine is to simulate the motion of bodies in a virtual environment.



Constraint Solvers

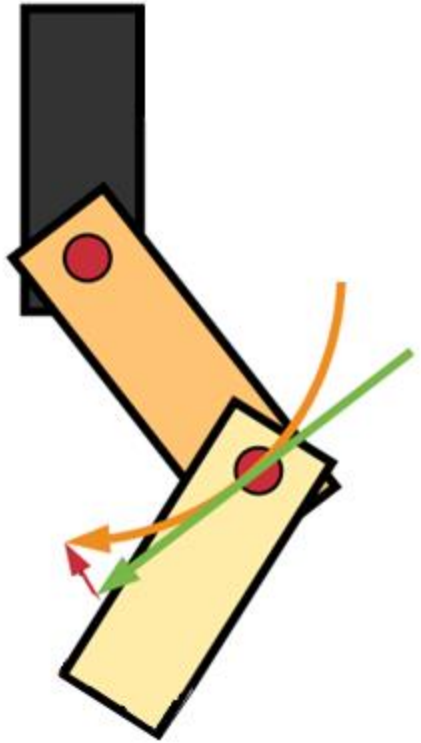
- The motion is the result of the solver interpreting the laws of physics, such as conservation of energy and momentum. But doing this 100% accurately is prohibitively **expensive**, and the trick to simulating it in **real-time** is to approximate to increase performance, as long as the result is physically realistic.

- The main idea of the physics engine is to discretize the motion using time-stepping. The equations of motion of constrained and unconstrained rigid bodies are very difficult to integrate directly and accurately.

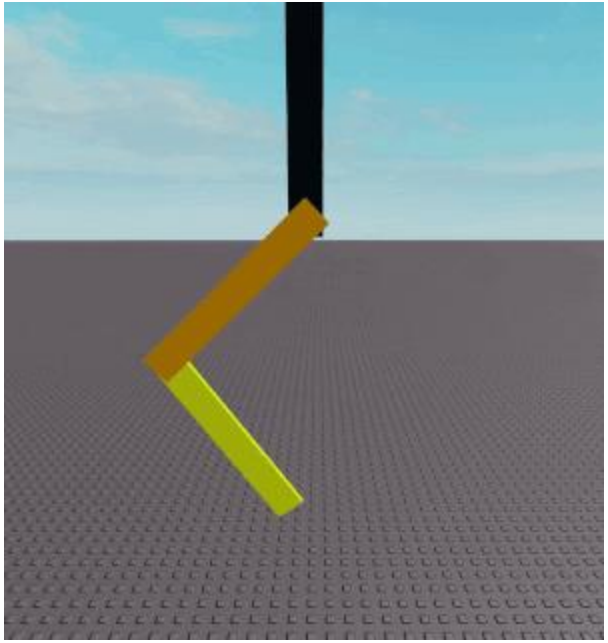
Constraint Solvers

A link is the connection of two or more bodies, or a body with the ground. The link is defined by certain constraints that restrict the relative motion of the bodies.

The discretization **subdivides the motion into small time increments**, where the equations are simplified and linearized making it possible to solve them approximately. This means that during each time step the motion of the relevant parts of rigid bodies that are involved in a constraint is linearly approximated.



Constraint Solvers

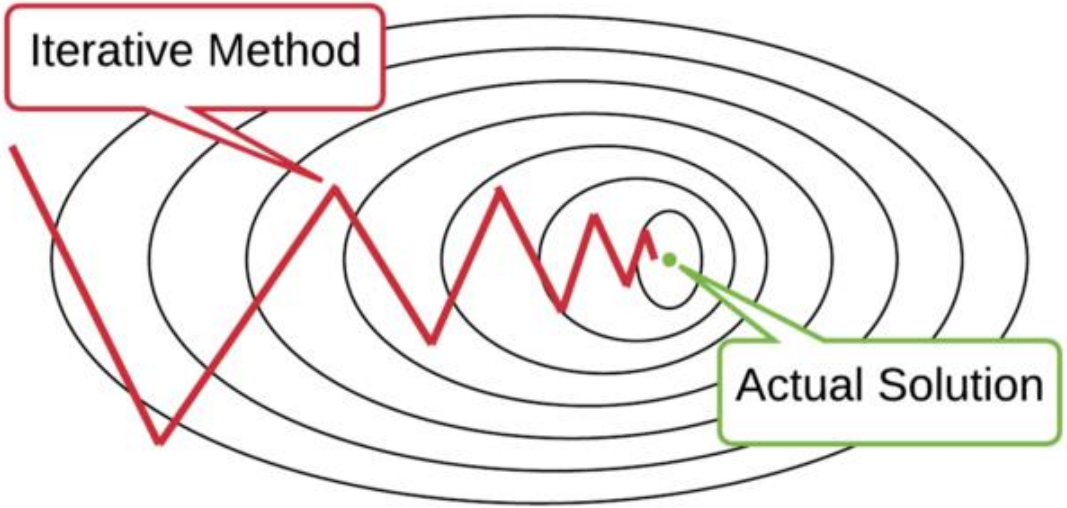


Constraint Solvers

A body is usually considered to be a rigid or flexible part of a mechanical system An example of a body is the arm of a robot, a wheel or axle in a car or the human forearm. Having linearized the equations of motion for a time step, we end up needing to solve a linear system or linear complementarity problem (LCP). These systems can be arbitrarily large and can still be quite expensive to solve exactly. Again the trick is to find an approximate solution using a faster method.

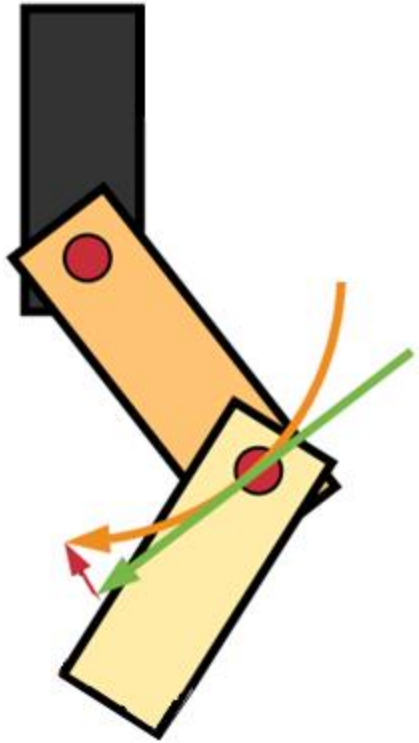
Constraint Solvers

A modern method to approximately solve an LCP with good convergence properties is the **Projected Gauss-Seidel** (PGS). It is an iterative method, meaning that with each iteration the approximate solution is brought closer to the true solution, and its final accuracy depends on the number of iterations.



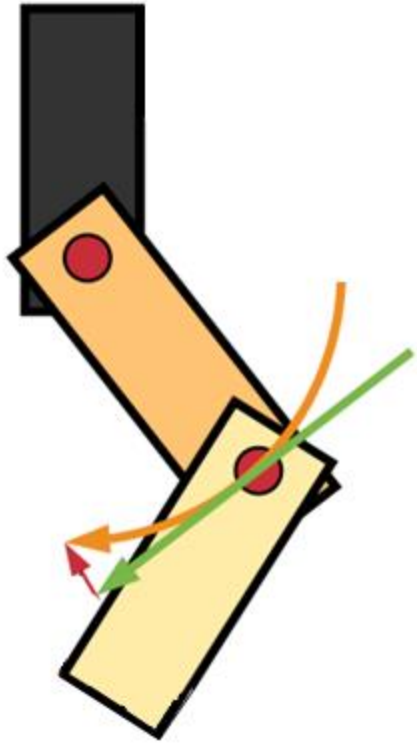
Constraint Solvers

Constraint is defined in terms of a behavior function or constraint function **C**, which takes the state **of a pair of bodies** as parameters (e.g. position and orientation) and outputs **a scalar number**. When the value of this function is in the acceptable range, the constraint is satisfied. Thus, in each step of the simulation, we must apply forces or impulses on the rigid bodies to attempt to keep the value of C in the allowed range.



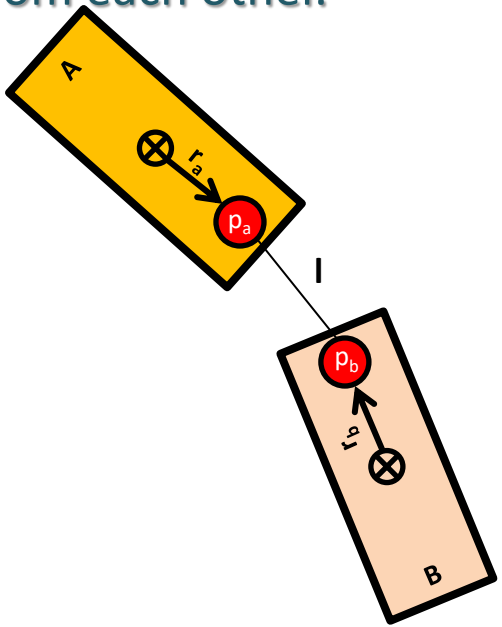
Equality Constraints

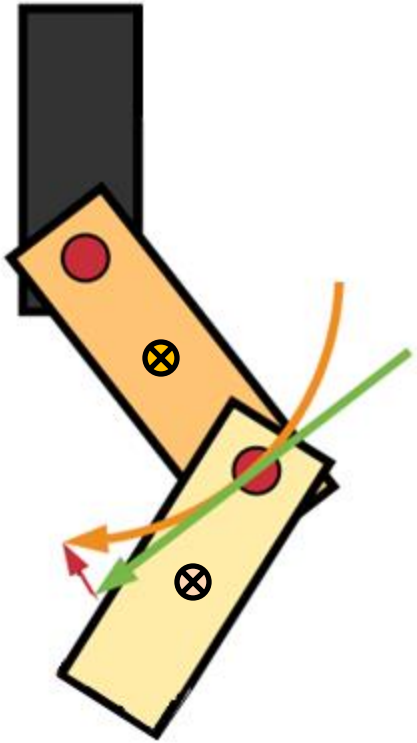
A common class of constraint is known as an equality constraint. An equality constraint is one in which the only acceptable value of **C** is **zero**. Thus, during each step of the simulation, we want to keep **C as close to zero** as possible. In other words, we want to **minimize C.** Equality constraints are used when the position of some point must always exactly match some predefined condition. A good example is a pin joint, where **two rigid bodies must always be connected at the location of the joint**.



Equality Constraints

The distance constraint ensures that point $\mathbf{p_a}$ from rigid body **A** and $\mathbf{p_b}$ from rigid body **B** remain at fixed distance **l** from each other.

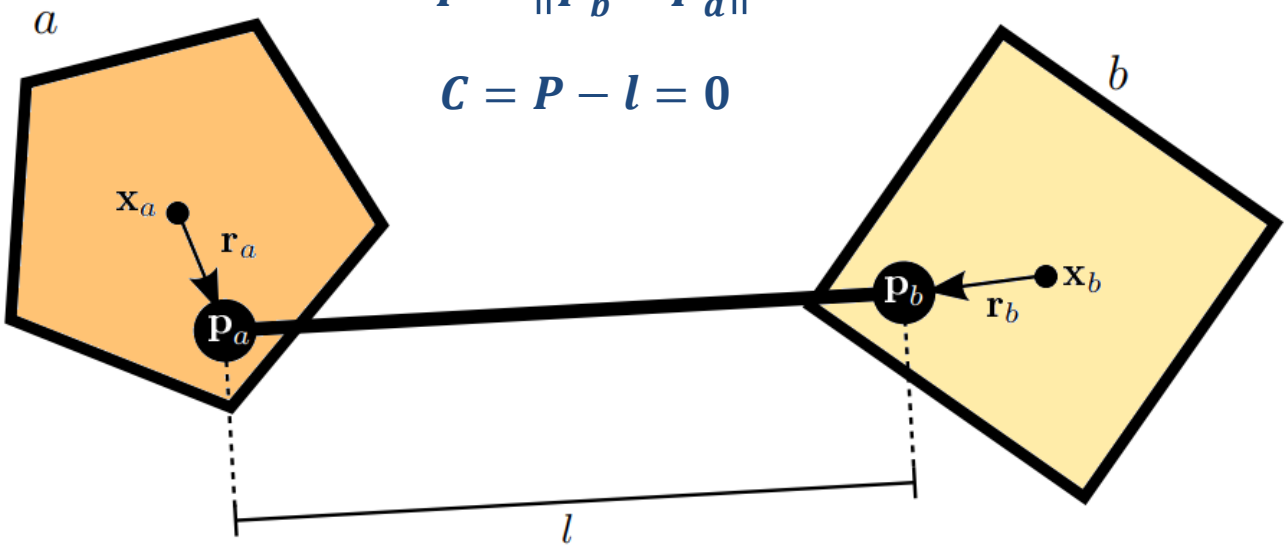




Equality Constraints

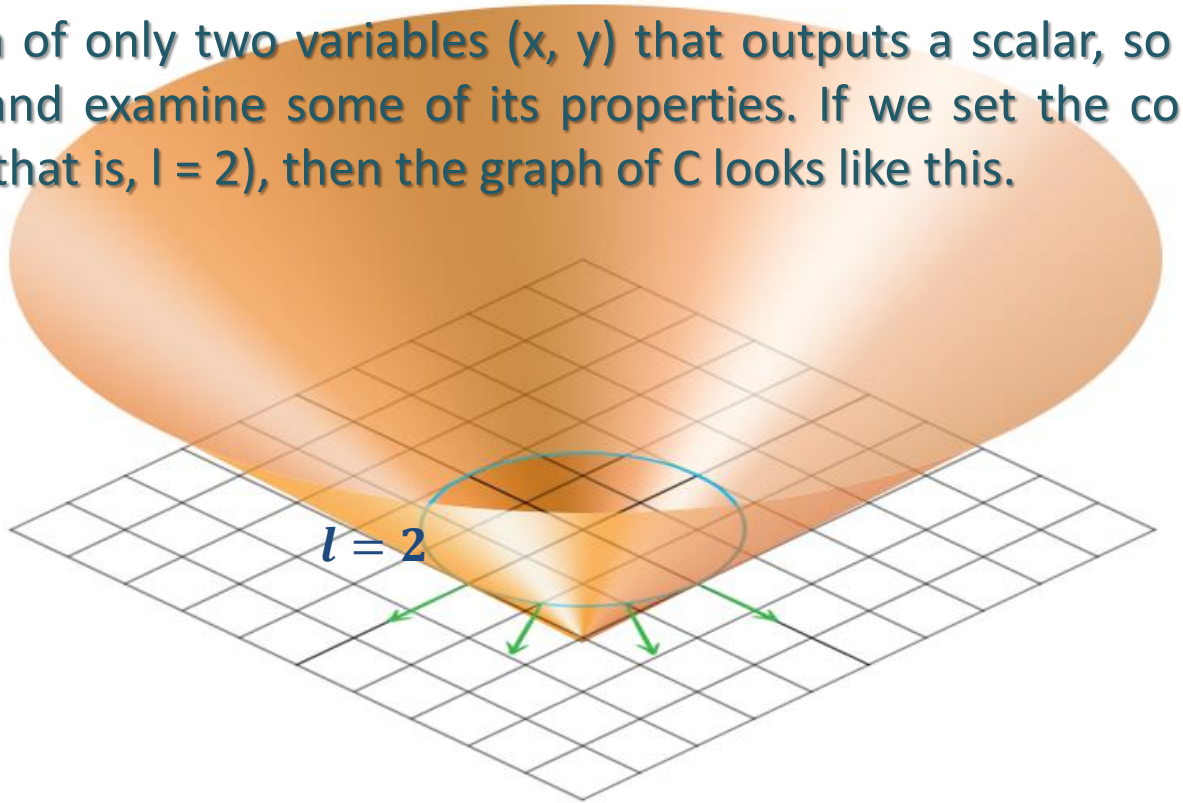
$$P = \|P_b - P_a\|$$

$$C = P - l = 0$$



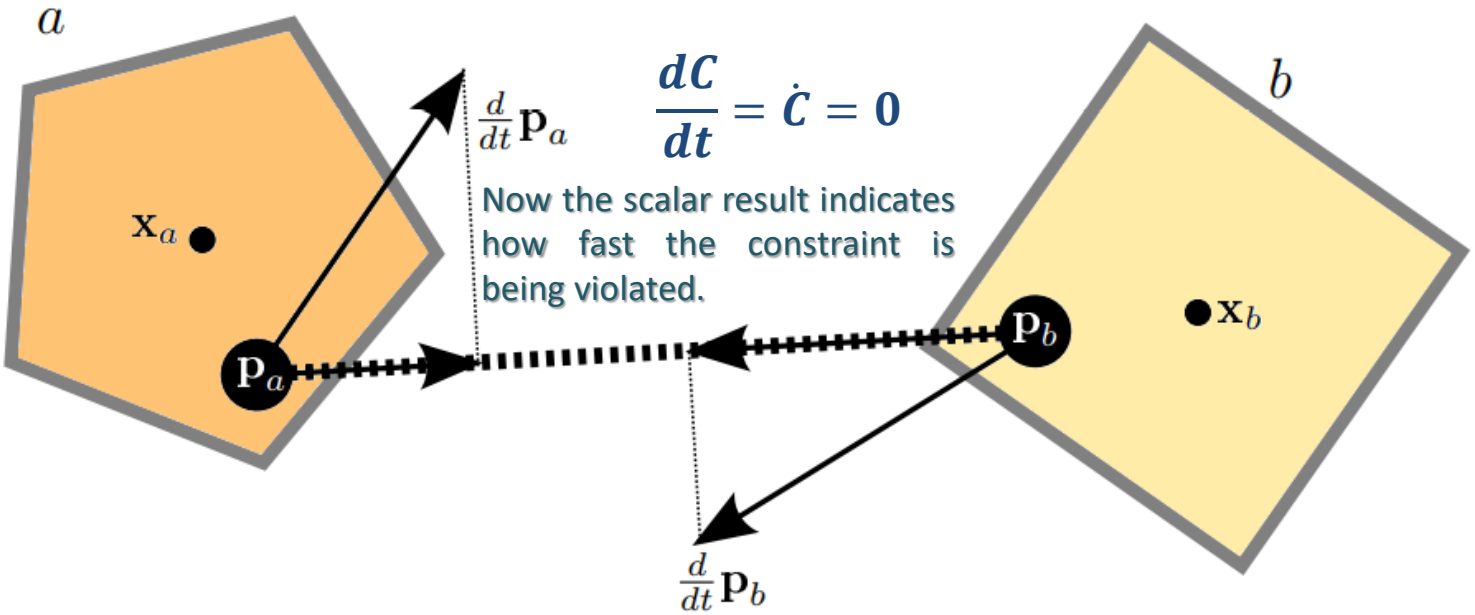
Equality Constraints

C is a function of only two variables (x, y) that outputs a scalar, so we can easily plot it and examine some of its properties. If we set the constraint distance as 2 (that is, l = 2), then the graph of C looks like this.



Equality Constraints

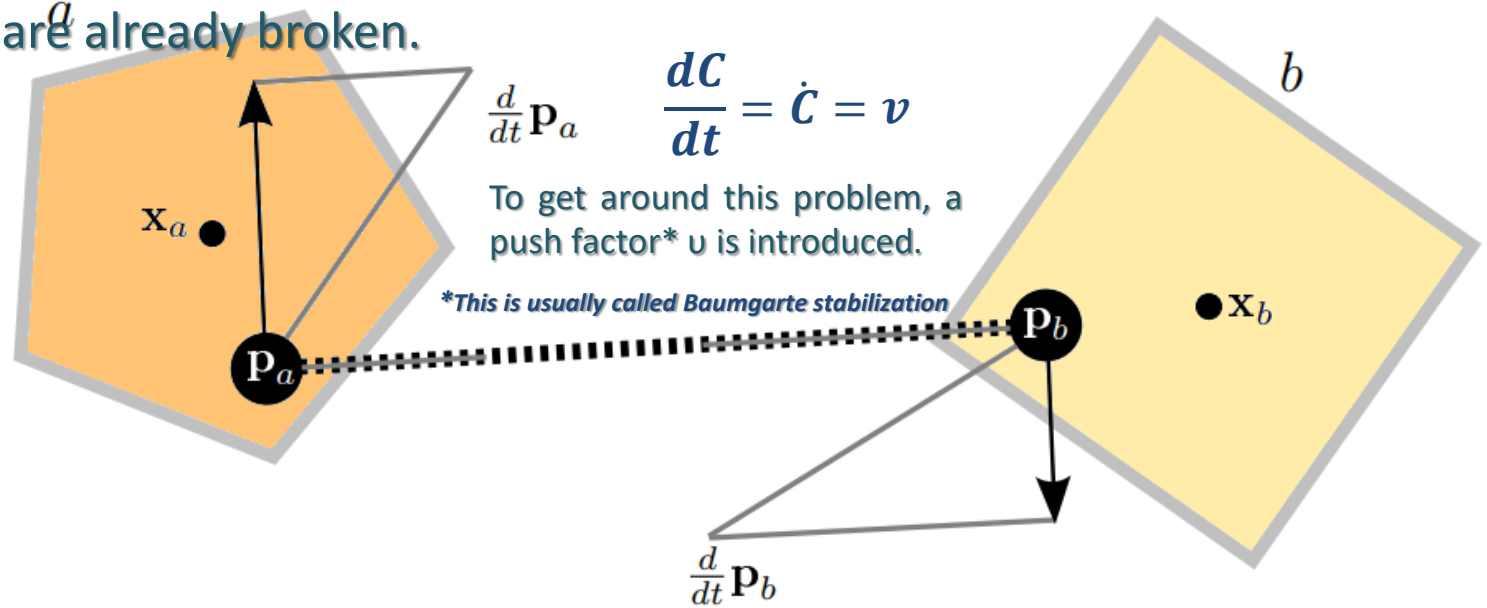
The constraint solver can not solve position constraints, but only velocity constraints. To get the velocity constraint we get the derivative of **C.**



project this velocity on to the constraint axis to see how fast the constraint is becoming broken

Equality Constraints

we lose the information about how much the (position) constraint was violated to begin with. So, it impossible for the solver to correct constraints that are already broken.



solving the constraint this should be 0 as the points may not move away or towards each other along the constraint axis

Equality Constraints

Now v can used to bias $\dot{C}$ so that the constraint can add a velocity to the system in order to fix already broken constraints. The solver expects the constraints to be in the following format. $P_b - P_a$ is the direction in which the constraint can apply force, also called the constraint axis.

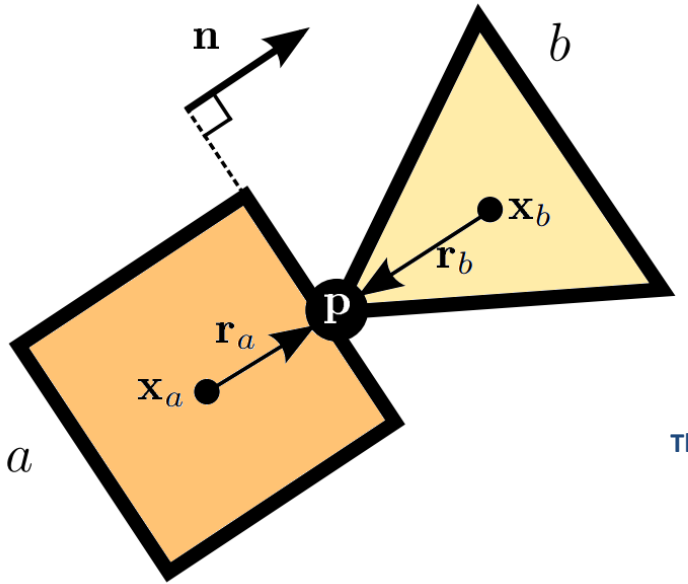
$$\frac{dC}{dt} = \dot{C} = (P_b - P_a)(v_b + \omega_b \times r_b - v_a - \omega_a \times r_a)$$

$$\frac{dC}{dt} = \dot{C} = \begin{bmatrix} -(P_b - P_a) \\ -(r_a \times (P_b - P_a)) \\ (P_b - P_a) \\ (r_b \times (P_b - P_a)) \end{bmatrix}^T \begin{bmatrix} v_a \\ \omega_a \\ v_b \\ \omega_b \end{bmatrix} = J_C v_{ab}$$

Contact Constraint

The contact constraint is setup in the same way as the distance constraint except that P_a and P_b are the same point. This makes it impossible to calculate the constraint axis, so n is used instead.

$$J_C = \begin{bmatrix} -n \\ -(r_a \times n) \\ n \\ (r_b \times n) \end{bmatrix}^T$$

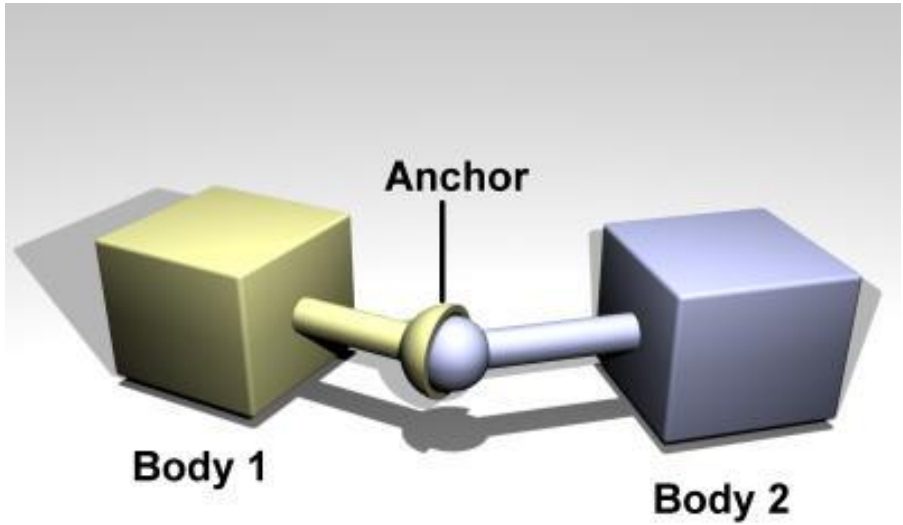


$$J_C = \begin{bmatrix} -n \\ -(r_a \times n) \\ 0 \\ 0 \end{bmatrix}^T$$

The constraint can be simplified if rigid body b is static

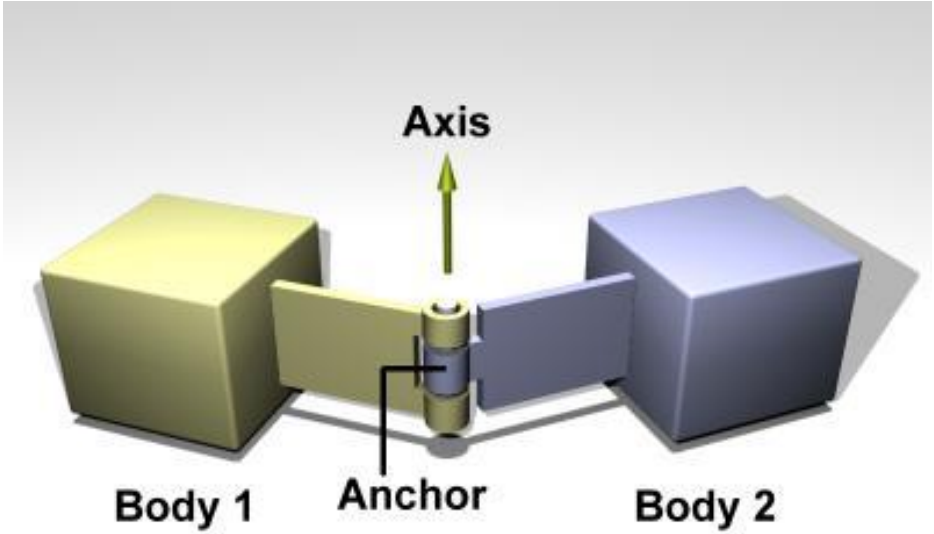
Joints and constraints

- Ball and Socket



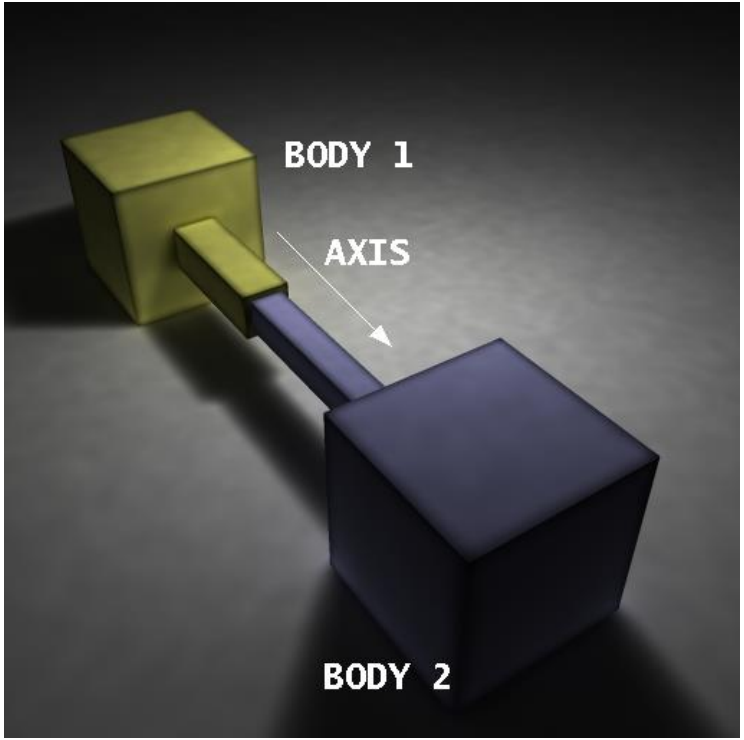
Joints and constraints

- Hinge



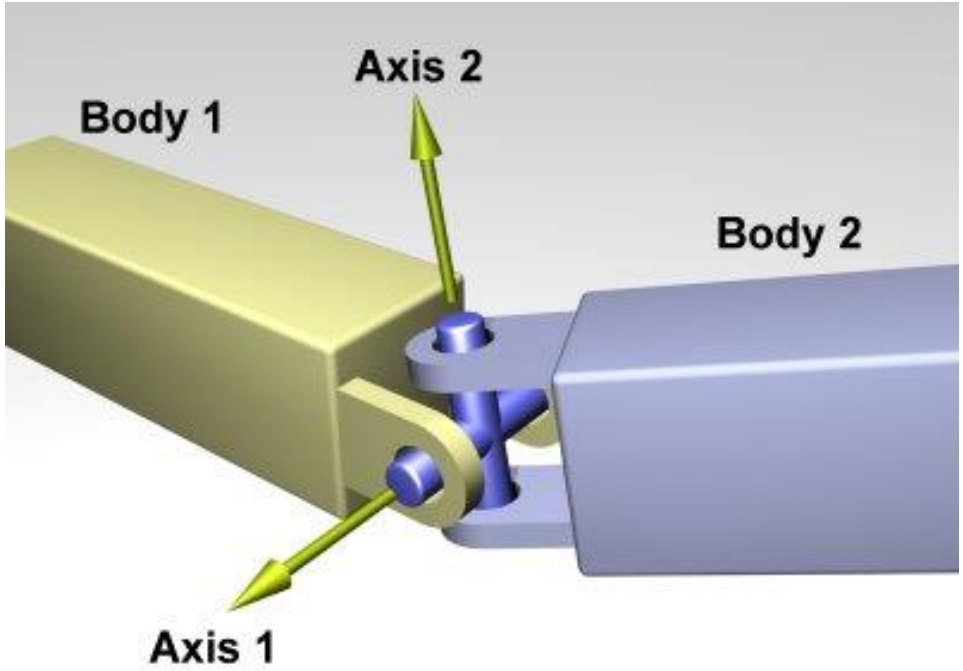
Joints and constraints

- Slider or Prismatic



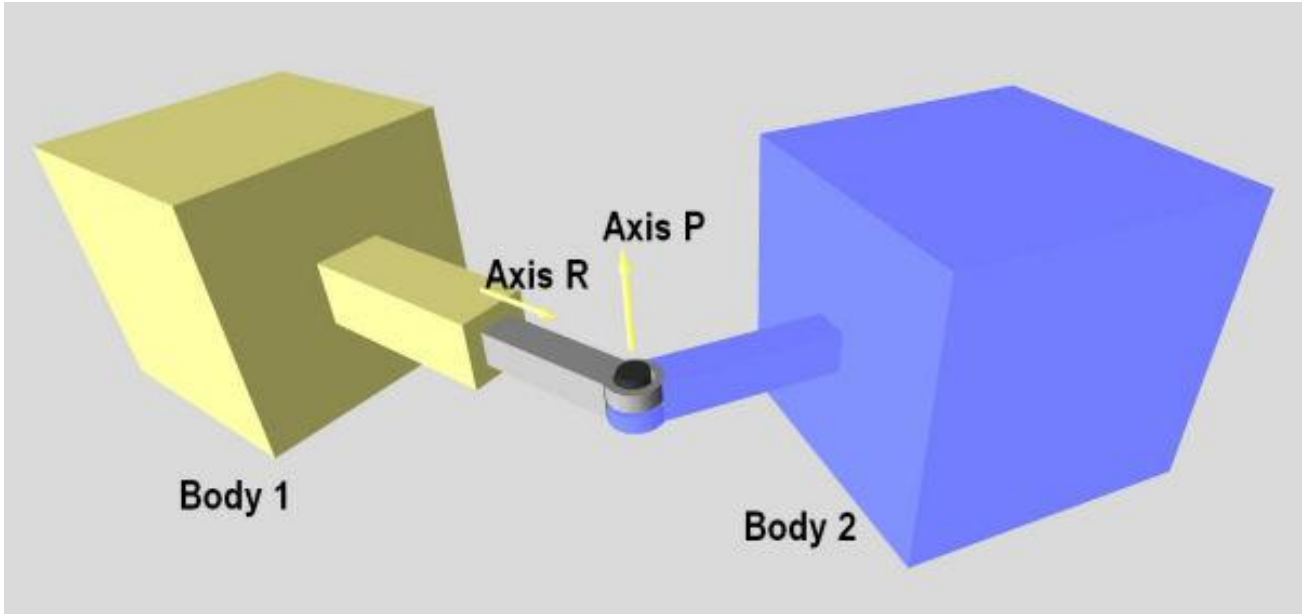
Joints and constraints

- Universal



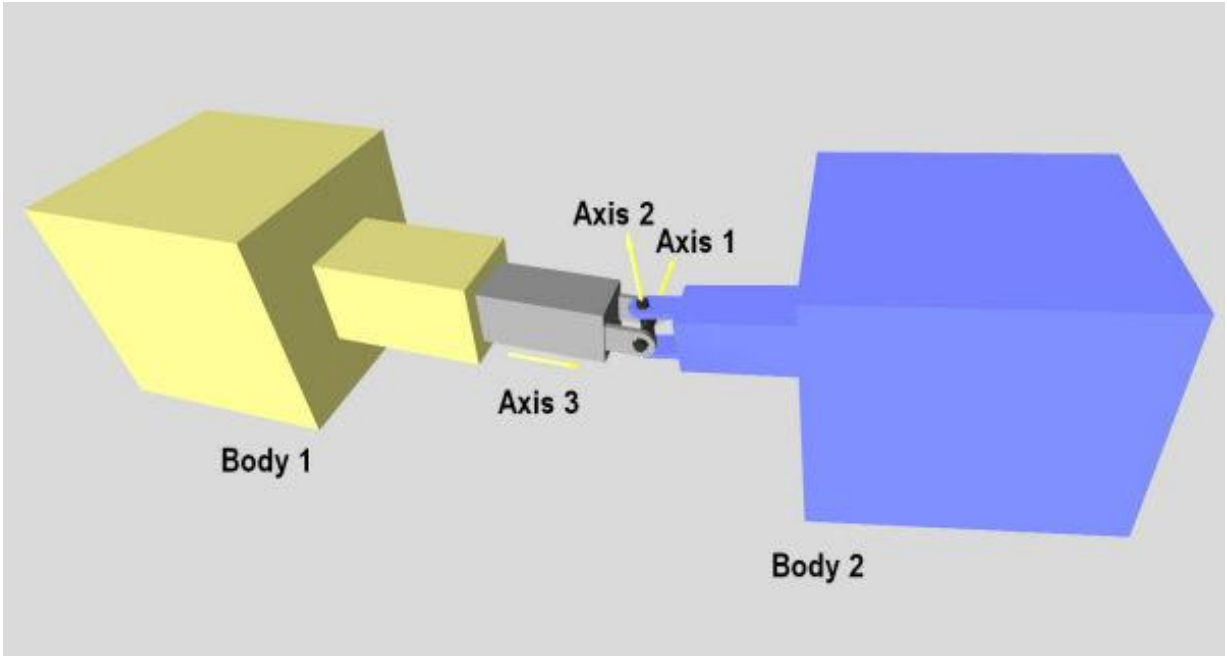
Joints and constraints

- Prismatic and Rotoide



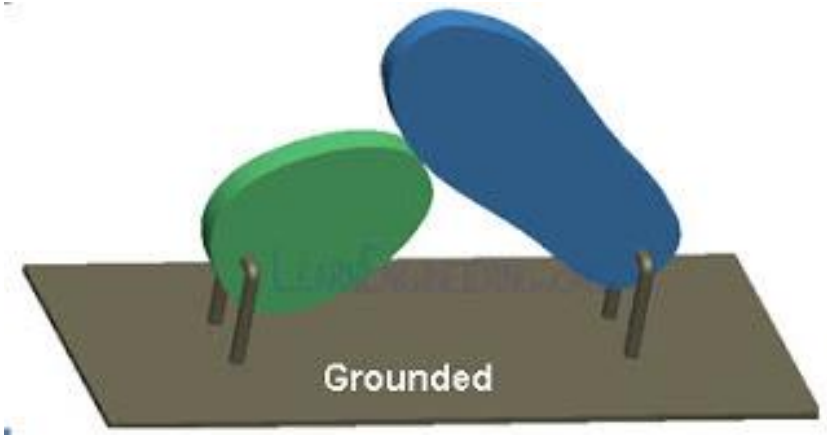
Joints and constraints

- Prismatic - Universal



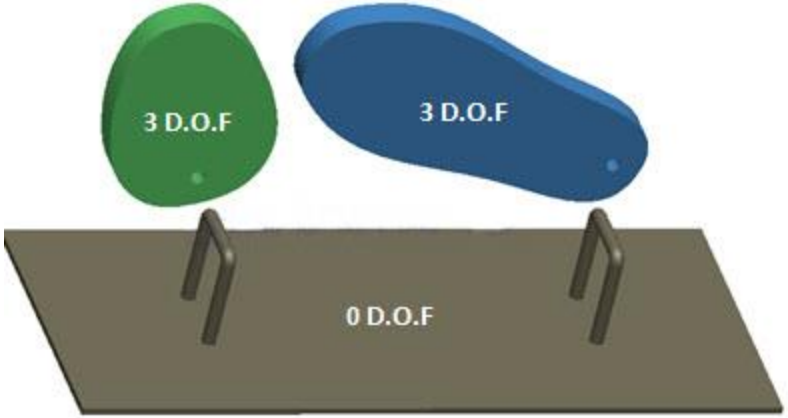
Degrees of Freedom of a Mechanism in 2D

- A mechanism is a collection of rigid bodies or links, connected through pairs, provided one link is grounded.



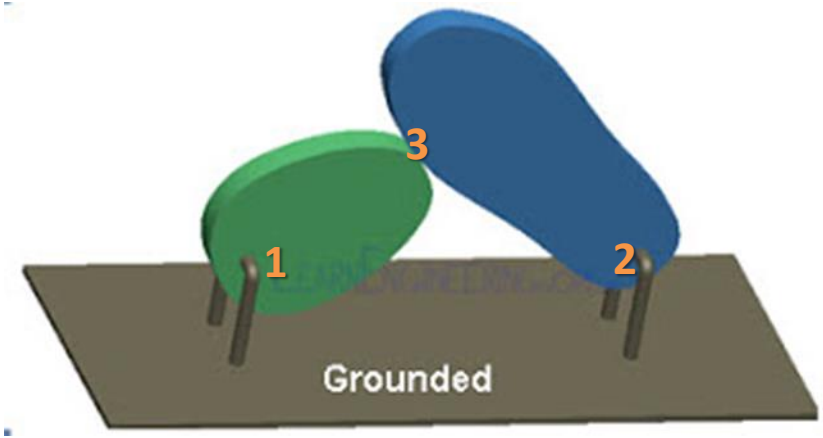
Degrees of Freedom of a Mechanism in 2D

- If this system were not connected like this, then each link except the ground would have 3 degrees of freedom.



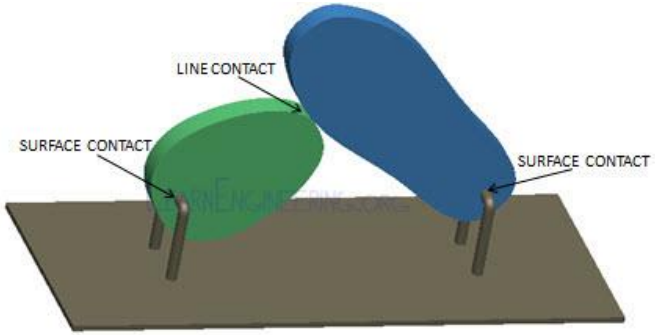
Degrees of Freedom of a Mechanism in 2D

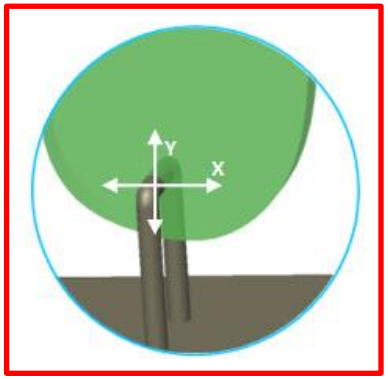
- So total degrees of freedom, or mobility is **3(N-1)**. N represents total number of links. In this case N is 3. But when we connect it together through pairs, links will not have the same 3 degrees of freedom.

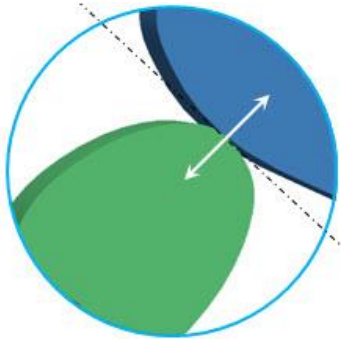


Degrees of Freedom of a Mechanism in 2D

- If joint between 2 links is having surface contact as shown below, then both the links will have same **translatonary** motion, in X and Y directions. So for each such pairs, there will be a deduction of 2 mobility from total mobility. Where L_p represents number of pairs with surface contacts. Such pairs are called lower pairs. In this case we have 2 lower pairs.

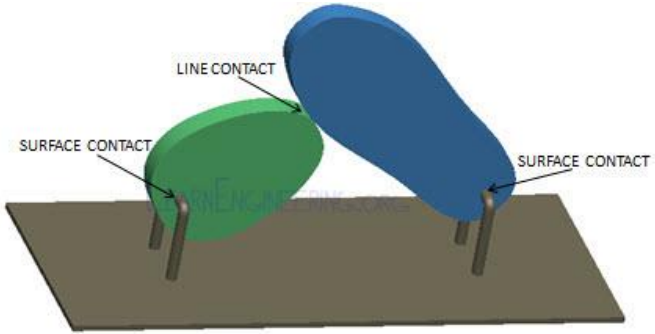


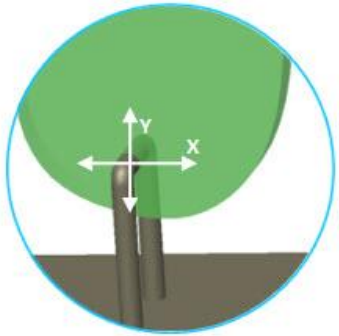


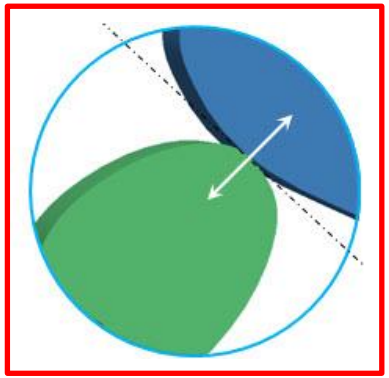


Degrees of Freedom of a Mechanism in 2D

- Now consider the joint which is having a line contact. If joint between 2 links is having **line** or **point contact**, both the link should have same translational motion along the common normal. However it could have different motion, in tangential direction. So for each such pairs, there will be deduction of 1 mobility from total mobility. This kind of pair is called higher pair (H_P). Here we have got **1 higher pair**.







Degrees of Freedom of a Mechanism in 2D

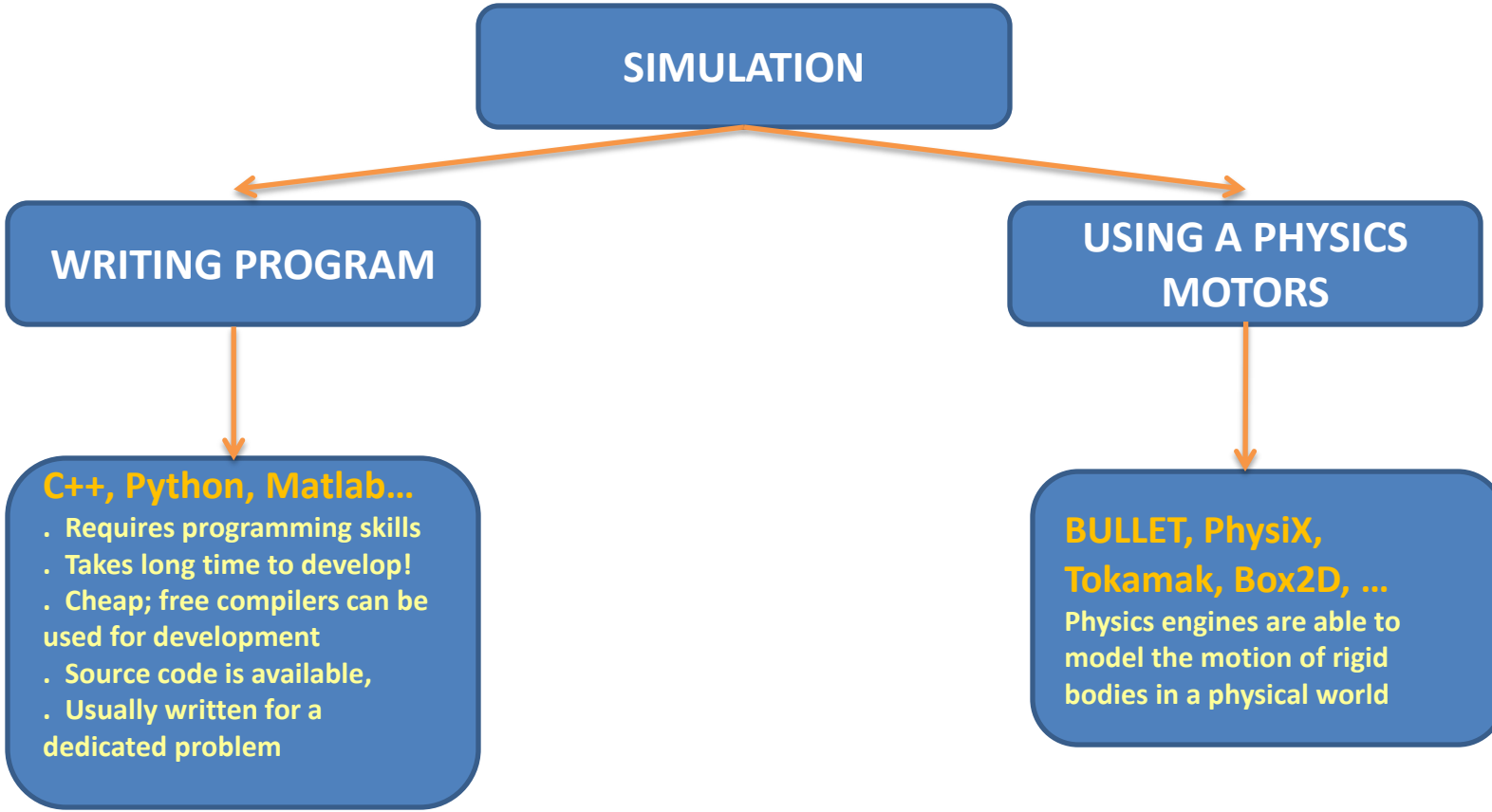
- The general equation to find out degrees of freedom of a planar mechanism is given below. This equation is also known as Kuthbach equation.

- Here N represent total number of links in the mechanism. LP and HP represent number of lower pairs and higher pairs respectively.

$$DOF = 3(N-1) - 2L_p - H_p = 3(3-1) - 2.2 - 1 = 9 - 3 - 4 - 1 = 1$$

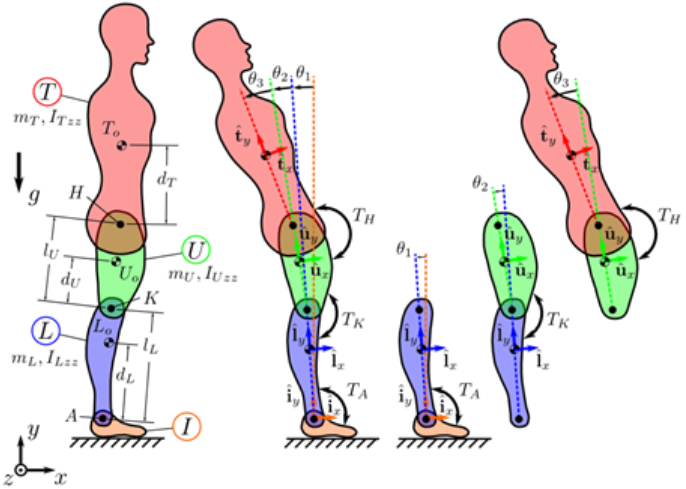
- So this mechanism has got 1 degree of freedom. Means, by knowing position of only one cam, we can completely determine this mechanism.

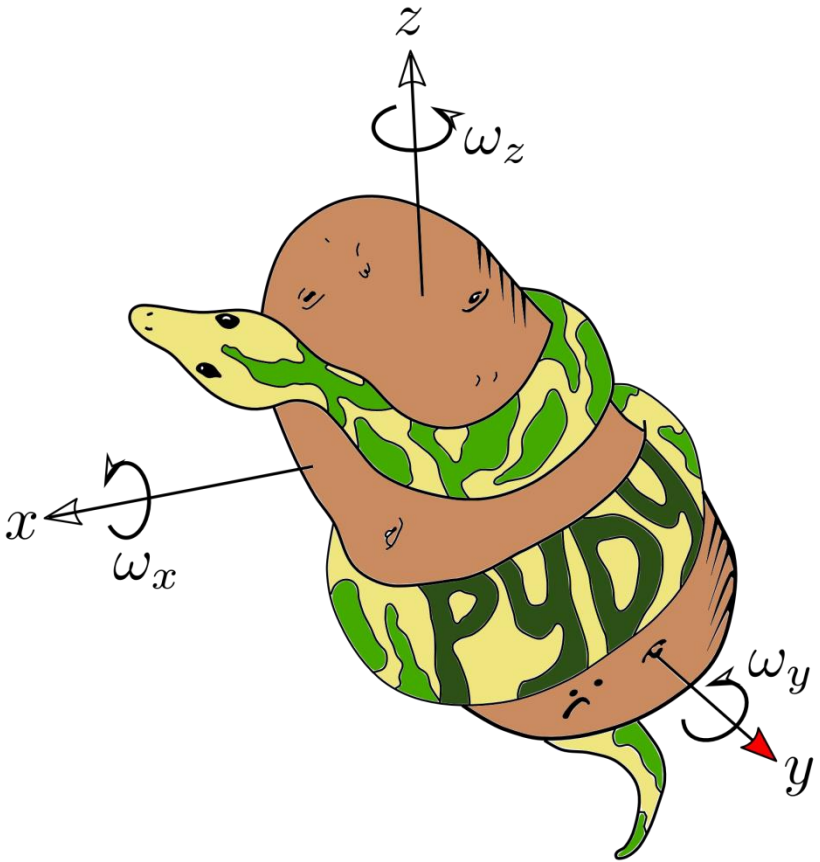
PHYSICS in COMPUTER ANIMATIONS and GAMES



PyDy

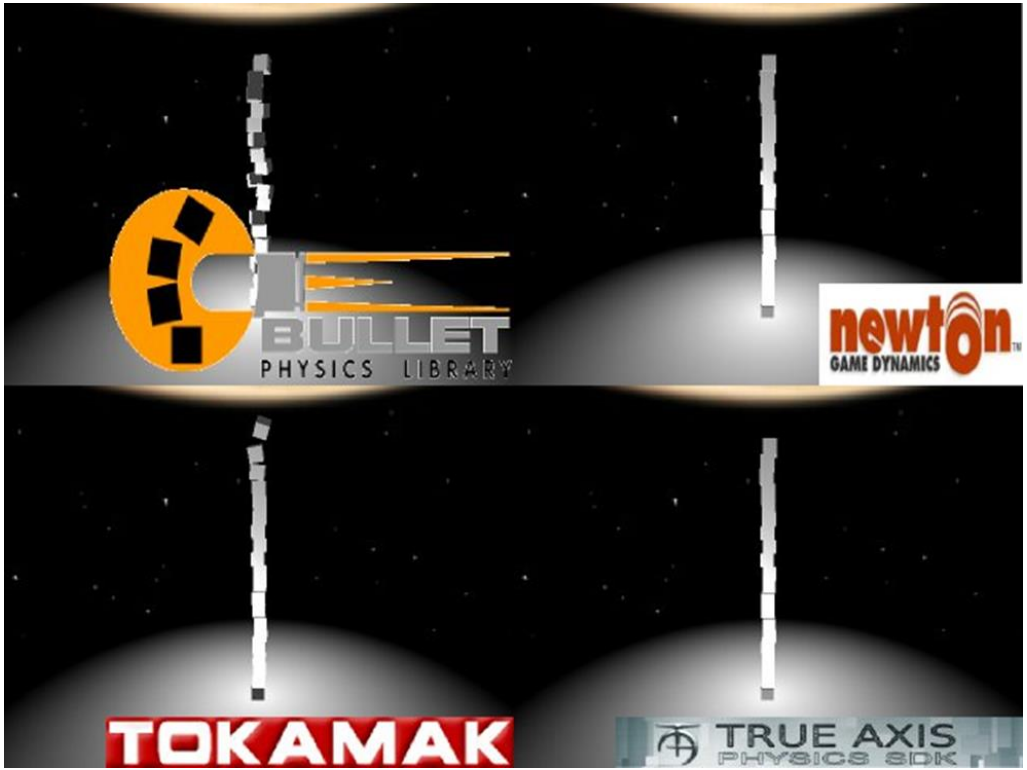
PyDy is a general tool for mutlibody dynamic analysis written in Python





PHYSICS in COMPUTER ANIMATIONS and GAMES

Multibody system Physics engines are able to model the motion of rigid bodies in a physical world



Multibody system Physics engines are able to model the motion of rigid bodies in a physical world

The integrator is responsible for calculating a body's position given the forces acting on it.

The performance of the integrator effects the accuracy of the simulation.

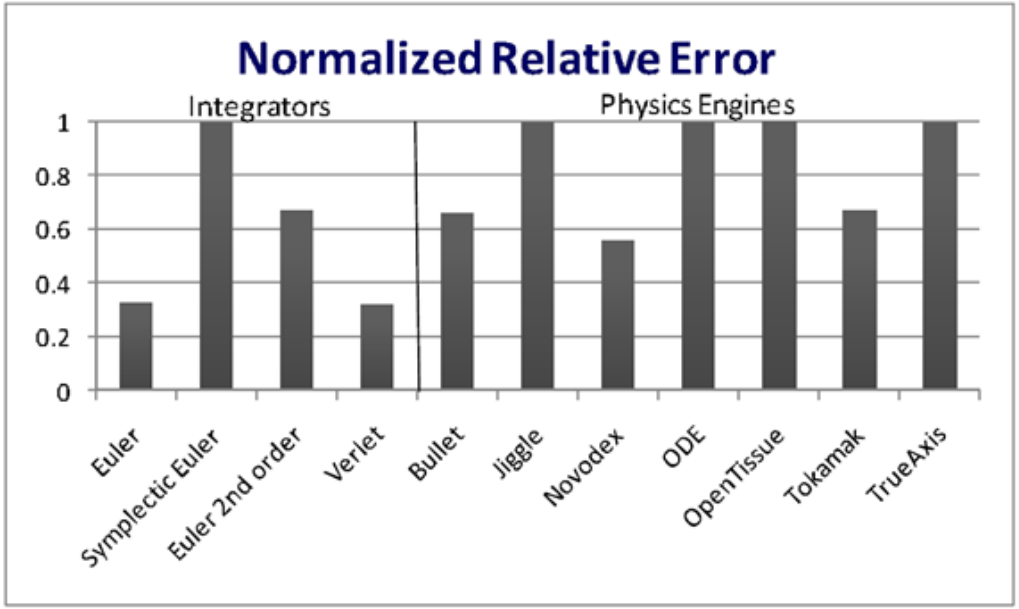


Figure 1 – Positional error from cumulative numerical integrators relative to the ideal case normalized to the Symplectic Euler integrator error

Boeing,A. Bräunl,T. (2007) **Evaluation of real-time physics simulation systems,** Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia, Pages 281-288

Multibody system Physics engines are able to model the motion of rigid bodies in a physical world

The materials restitution properties were tested by colliding a box with a sphere. The box is placed on the ground and the sphere is placed one meter above.

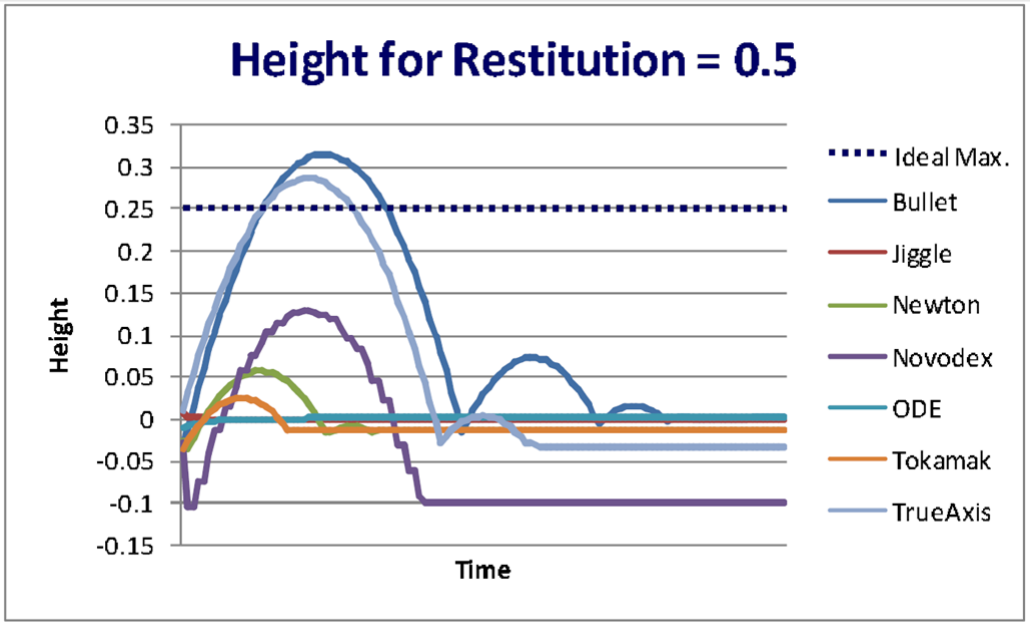


Figure 4 - Bounce height for a coefficient of restitution of 0.5

Multibody system Physics engines are able to model the motion of rigid bodies in a physical world

Constraint stability is one of the areas in rigid body calculations.

If constraints are unstable numerical errors can cause constrained bodies (bones) to slowly drift apart.

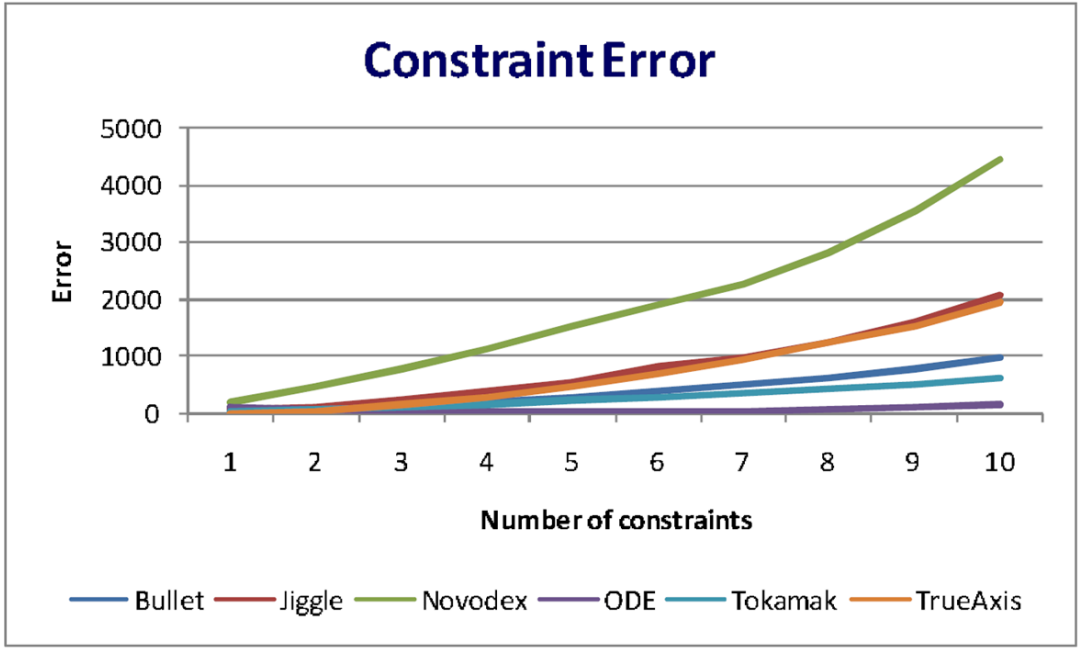


Figure 9 - Constraint error

Multibody system Physics engines are able to model the motion of rigid bodies in a physical world

Box2D

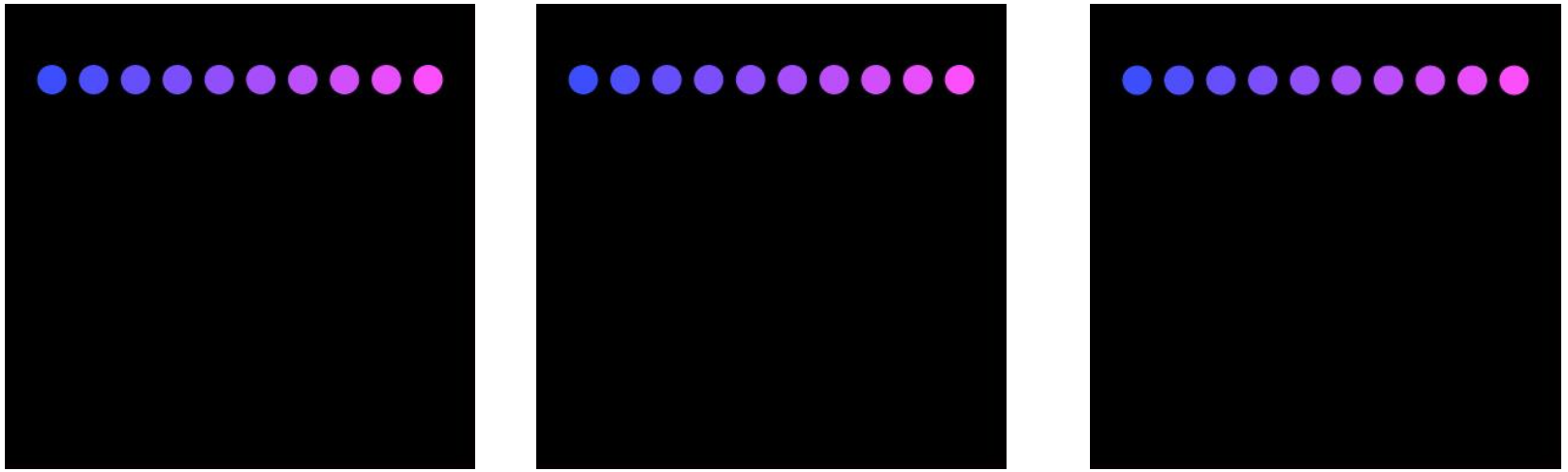
CHIPMUNK
Game Dynamics

Multibody system Physics engines are able to model the motion of rigid bodies in a physical world

Box2D

Coefficient of restitution s of balls

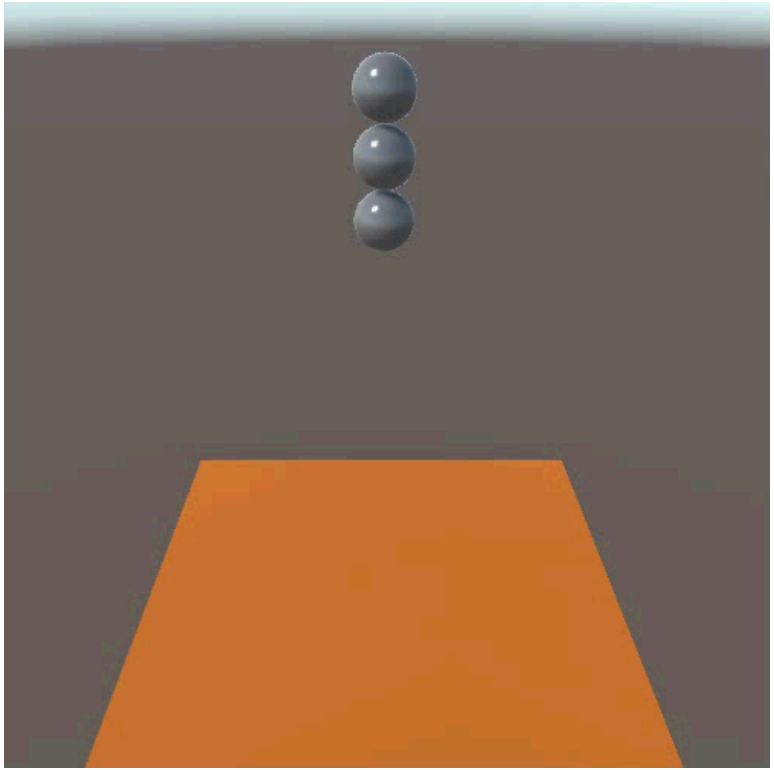
0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1



0 **0.5** **1**

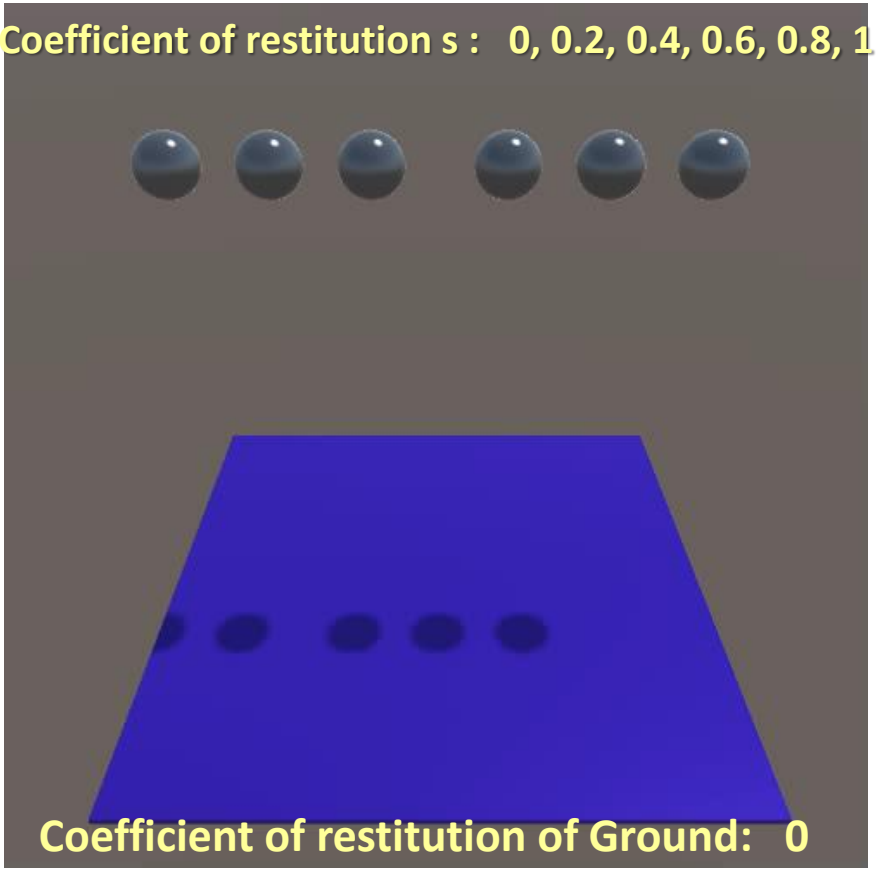
Coefficient of restitution of Ground

Multibody system Physics engines are able to model the motion of rigid bodies in a physical world



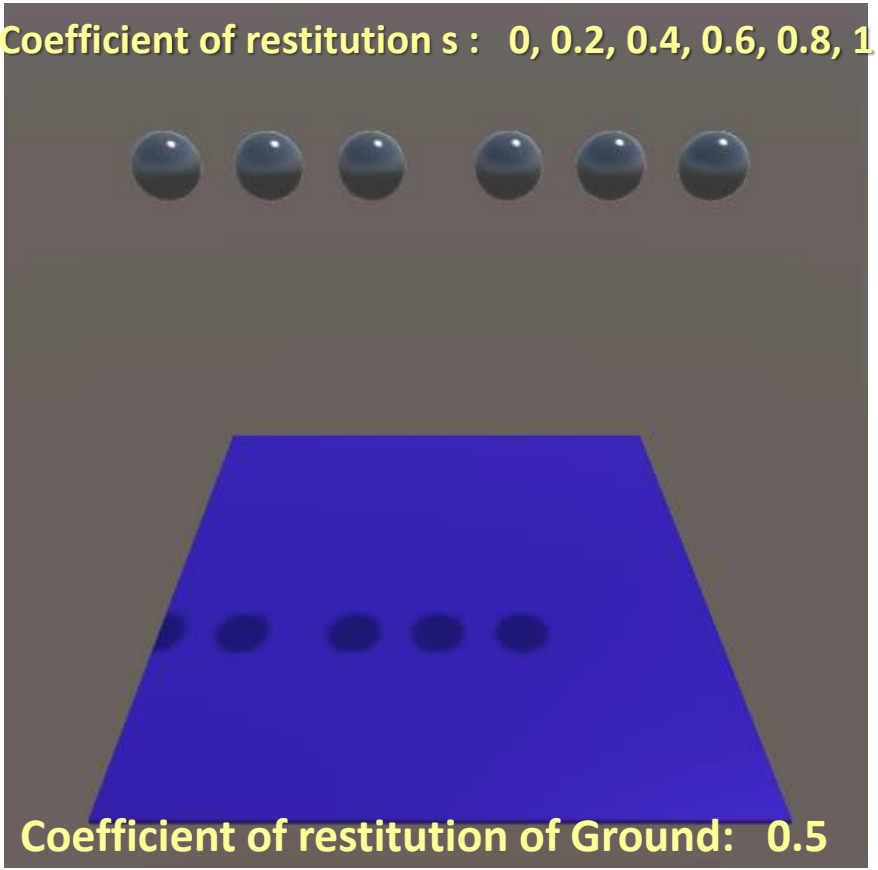
PhysX™ by NVIDIA

Multibody system Physics engines are able to model the motion of rigid bodies in a physical world



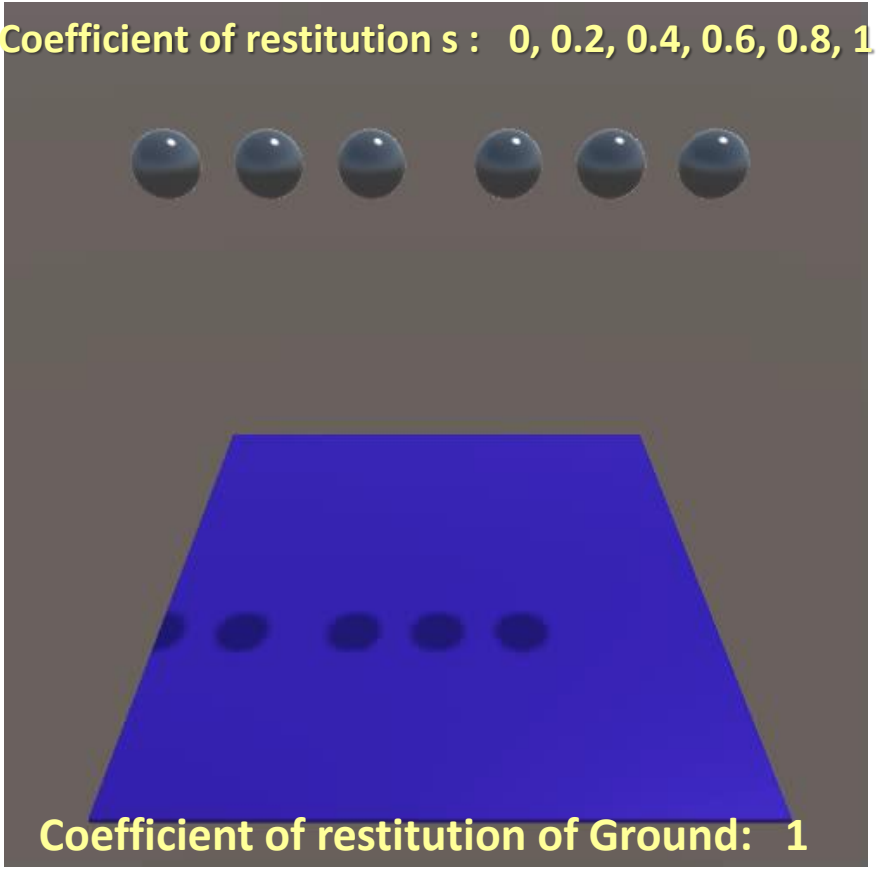
PhysX by NVIDIA

Multibody system Physics engines are able to model the motion of rigid bodies in a physical world



PhysX by NVIDIA

Multibody system Physics engines are able to model the motion of rigid bodies in a physical world

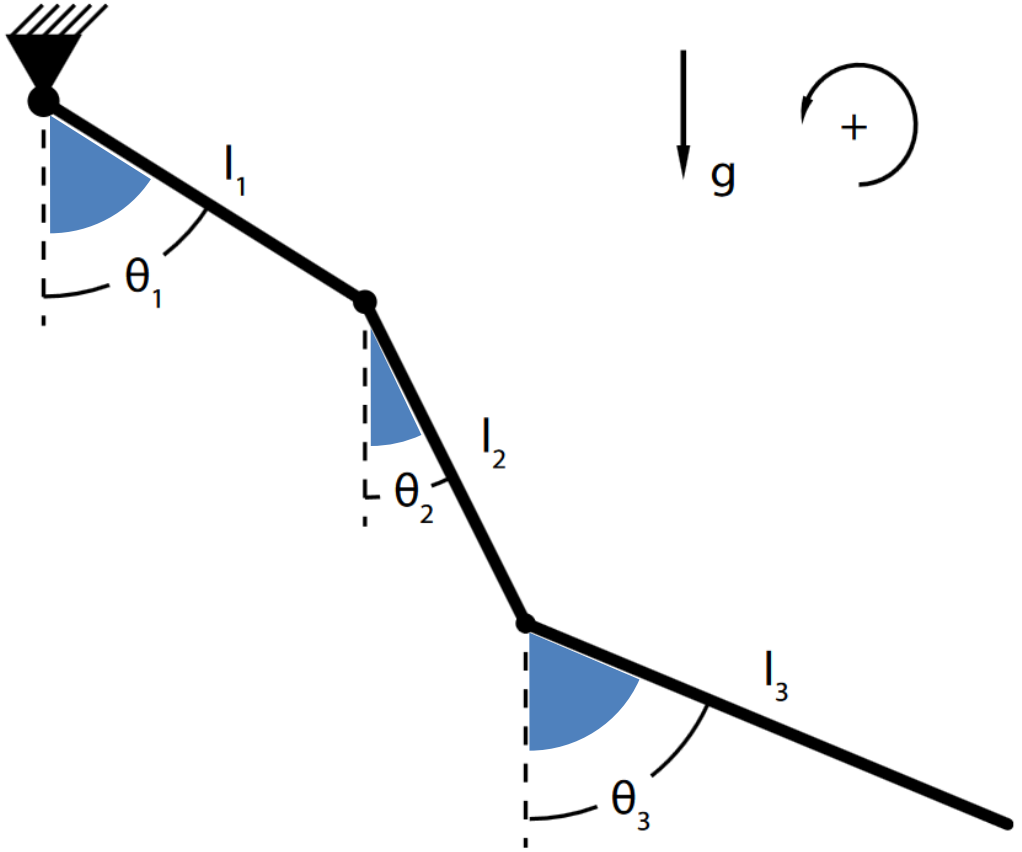


PhysX by NVIDIA

it is natural that most of a game company's efforts will be spent on 'how things look' rather than 'how things move'

n-Link Pendulum

While the double pendulum equations of motion can be solved relatively straightforwardly, the equations for **a triple pendulum** are much more involved.



$$\ddot{\theta_1} = -(2((l_3^2 m_3^2 \sin(2\theta_1 - 2\theta_3)(4I_2 - l_2^2 m_2) + l_2^2 \sin(2\theta_1 - 2\theta_2)(m_2 + 2m_3)(m_2 m_3 l_3^2 + 4I_3(m_2 + 2m_3)))l_1^2 \dot{\theta_1}^2 + (l_2(\sin(\theta_1 - \theta_2)((m_2 m_3(m_2 + 3m_3)l_3^2 + 4I_3(m_2^2 + 6m_2 m_3 + 8m_3^2))l_2^2 + 4I_2(m_3(m_2 + m_3)l_3^2 + 4I_3(m_2 + 2m_3))) + l_3^2 m_3^2 \sin(\theta_1 + \theta_2 - 2\theta_3)(4I_2 - l_2^2 m_2))\dot{\theta_2}^2 - 4k_2 l_2(cos(\theta_1 - \theta_2)(m_3(m_2 + m_3)l_3^2 + 4I_3(m_2 + 2m_3)) - l_3^2 m_3^2 cos(\theta_1 + \theta_2 - 2\theta_3))\dot{\theta_2} + l_3 m_3(\sin(\theta_1 - \theta_3)(8I_3 m_3 l_2^2 + 4I_2 m_3 l_3^2 + 16I_2 I_3) + l_2^2 \sin(\theta_1 - 2\theta_2 + \theta_3)(m_2 m_3 l_3^2 + 4I_3(m_2 + 2m_3)))\dot{\theta_3}^2 - 4k_3 l_3 m_3(cos(\theta_1 - \theta_3)(2m_3 l_2^2 + 4I_2) - l_2^2 cos(\theta_1 - 2\theta_2 + \theta_3)(m_2 + 2m_3))\dot{\theta_3} - g(\sin(\theta_1)((m_3(m_1 m_2 + 2m_1 m_3 + 3m_2 m_3 + m_2^2)l_3^2 + 4I_3(m_2^2 + 6m_2 m_3 + m_1 m_2 + 4m_3^2 + 4m_1 m_3))l_2^2 + 4I_2(m_3(m_1 + 2m_2 + m_3)l_3^2 + 4I_3(m_1 + 2m_2 + 2m_3))) + l_3^2 m_3^2(\sin(\theta_1 - 2\theta_3)(4I_2 - l_2^2 m_2) - 2l_2^2 cos(2\theta_2 - 2\theta_3)\sin(\theta_1)(m_1 + m_2)) + l_2^2 \sin(\theta_1 - 2\theta_2)(m_2 + 2m_3)(m_2 m_3 l_3^2 + 4I_3(m_2 + 2m_3)))l_1 + 2k_1(4I_2(m_3 l_3^2 + 4I_3) + l_2^2(m_3(m_2 + 2m_3)l_3^2 + 4I_3(m_2 + 4m_3)) - 2l_2^2 l_3^2 m_3^2 cos(2\theta_2 - 2\theta_3))\dot{\theta_1}))/(64I_1 I_2 I_3 + 8I_3 l_1^2 l_2^2 m_2^2 + 8I_1 l_2^2 l_3^2 m_3^2 + 8I_2 l_1^2 l_3^2 m_3^2 + 32I_3 l_1^2 l_2^2 m_3^2 + 16I_2 I_3 l_1^2 m_1 + 16I_1 I_3 l_2^2 m_2 + 64I_2 I_3 l_1^2 m_2 + 16I_1 I_2 l_3^2 m_3 + 64I_1 I_3 l_2^2 m_3 + 64I_2 I_3 l_1^2 m_3 + 4I_3 l_1^2 l_2^2 m_1 m_2 + 4I_2 l_1^2 l_3^2 m_1 m_3 + 16I_3 l_1^2 l_2^2 m_1 m_3 + 4I_1 l_2^2 l_3^2 m_2 m_3 + 16I_2 l_1^2 l_3^2 m_2 m_3 + 48I_3 l_1^2 l_2^2 m_2 m_3 - 8I_1 l_2^2 l_3^2 m_3^2 cos(2\theta_2 - 2\theta_3) - 2l_1^2 l_2^2 cos(2\theta_1 - 2\theta_2)(m_2 + 2m_3)(m_2 m_3 l_3^2 + 4I_3(m_2 + 2m_3)) - 2l_1^2 l_3^2 m_3^2 cos(2\theta_1 - 2\theta_3)(-m_2 l_2^2 + 4I_2) + 2l_1^2 l_2^2 l_3^2 m_1 m_3^2 + 6l_1^2 l_2^2 l_3^2 m_2 m_3^2 + 2l_1^2 l_2^2 l_3^2 m_2^2 m_3 + l_1^2 l_2^2 l_3^2 m_1 m_2 m_3 - 2l_1^2 l_2^2 l_3^2 m_1 m_3^2 cos(2\theta_2 - 2\theta_3) - 4l_1^2 l_2^2 l_3^2 m_2 m_3^2 cos(2\theta_2 - 2\theta_3))$$

$$\ddot{\theta_2} = (2((l_1^2\sin(2\theta_1 - 2\theta_2)(m_2 + 2m_3)(m_2m_3l_3^2 + 4I_3(m_2 + 2m_3)) - l_3^2m_3^2\sin(2\theta_2 - 2\theta_3)((m_1 + 2m_2)l_1^2 + 4I_1))l_2^2\dot{\theta_2}^2 + l_1(\sin(\theta_1 - \theta_2)((m_3(m_1(m_2 + m_3) + 2m_2(2m_2 + 3m_3))l_3^2 + 4I_3(m_2 + 2m_3)(m_1 + 4m_2 + 4m_3))l_1^2 + 4I_1(m_3(m_2 + m_3)l_3^2 + 4I_3(m_2 + 2m_3))) - l_3^2m_3^2\sin(\theta_1 + \theta_2 - 2\theta_3)((m_1 + 2m_2)l_1^2 + 4I_1))l_2\dot{\theta_1}^2 + 4k_1l_1(cos(\theta_1 - \theta_2)(m_3(m_2 + m_3)l_3^2 + 4I_3(m_2 + 2m_3)) - l_3^2m_3^2cos(\theta_1 + \theta_2 - 2\theta_3))l_2\dot{\theta_1} + (-l_3m_3(\sin(\theta_2 - \theta_3)((m_3(m_1 + 3m_2)l_3^2 + 4I_3(m_1 + 3m_2 + 2m_3))l_1^2 + 4I_1(m_3l_3^2 + 4I_3)) - l_1^2\sin(2\theta_1 - \theta_2 - \theta_3)(m_2m_3l_3^2 + 4I_3(m_2 + 2m_3)))\dot{\theta_3}^2 + 4k_3l_3m_3(cos(\theta_2 - \theta_3)((m_1 + 3m_2 + 2m_3)l_1^2 + 4I_1) - l_1^2cos(2\theta_1 - \theta_2 - \theta_3)(m_2 + 2m_3))\dot{\theta_3} + g(\sin(\theta_2)((m_2m_3(2m_2 + 3m_3)l_3^2 + 8I_3(m_2^2 + 3m_2m_3 + 2m_3^2))l_1^2 + 4I_1(m_3(m_2 + m_3)l_3^2 + 4I_3(m_2 + 2m_3))) - l_1^2\sin(2\theta_1 - \theta_2)(m_3(m_1(m_2 + m_3) + m_2(2m_2 + 3m_3))l_3^2 + 4I_3(m_2 + 2m_3)(m_1 + 2m_2 + 2m_3)) + l_3^2m_3^2(\sin(\theta_2 - 2\theta_3)(m_2l_1^2 + 4I_1) + l_1^2\sin(2\theta_1 + \theta_2 - 2\theta_3)(m_1 + m_2))))l_2 - 2k_2(4I_1(m_3l_3^2 + 4I_3) + l_1^2(m_3(m_1 + 4m_2 + 2m_3)l_3^2 + 4I_3(m_1 + 4m_2 + 4m_3)) - 2l_1^2l_3^2m_3^2cos(2\theta_1 - 2\theta_3))\dot{\theta_2}))/(64I_1I_2I_3 + 8I_3l_1^2l_2^2m_2^2 + 8I_1l_2^2l_3^2m_3^2 + 8I_2l_1^2l_3^2m_3^2 + 32I_3l_1^2l_2^2m_3^2 + 16I_2I_3l_1^2m_1 + 16I_1I_3l_2^2m_2 + 64I_2I_3l_1^2m_2 + 16I_1I_2l_3^2m_3 + 64I_1I_3l_2^2m_3 + 64I_2I_3l_1^2m_3 + 4I_3l_1^2l_2^2m_1m_2 + 4I_2l_1^2l_3^2m_1m_3 + 16I_3l_1^2l_2^2m_1m_3 + 4I_1l_2^2l_3^2m_2m_3 + 16I_2l_1^2l_3^2m_2m_3 + 48I_3l_1^2l_2^2m_2m_3 - 8I_1l_2^2l_3^2m_3^2cos(2\theta_2 - 2\theta_3) - 2l_1^2l_2^2cos(2\theta_1 - 2\theta_2)(m_2 + 2m_3)(m_2m_3l_3^2 + 4I_3(m_2 + 2m_3)) - 2l_1^2l_3^2m_3^2cos(2\theta_1 - 2\theta_3)(-m_2l_2^2 + 4I_2) + 2l_1^2l_2^2l_3^2m_1m_3^2 + 6l_1^2l_2^2l_3^2m_2m_3^2 + 2l_1^2l_2^2l_3^2m_2^2m_3 + l_1^2l_2^2l_3^2m_1m_2m_3 - 2l_1^2l_2^2l_3^2m_1m_3^2cos(2\theta_2 - 2\theta_3) - 4l_1^2l_2^2l_3^2m_2m_3^2cos(2\theta_2 - 2\theta_3))$$

PHYSICS in COMPUTER ANIMATIONS and GAMES

$$\ddot{\theta_3} = -(2(32I_1I_2k_3\dot{\theta_3} - l_2l_3m_3\dot{\theta_2}^2(\sin(\theta_2-\theta_3)((l_2^2(m_1m_2+4m_1m_3+6m_2m_3+m_2^2)+4I_2(m_1+3m_2+2m_3))l_1^2+4I_1(4I_2+l_2^2(m_2+4m_3)))+l_1^2\sin(2\theta_1-\theta_2-\theta_3)(m_2+2m_3)(4I_2-m_2l_2^2))-l_1l_3m_3\dot{\theta_1}^2(\sin(\theta_1-\theta_3)(8I_1(m_3l_2^2+2I_2)+2l_1^2((m_1m_3-m_2^2)l_2^2+2I_2(m_1+4m_2+4m_3)))-l_2^2\sin(\theta_1-2\theta_2+\theta_3)(m_2+2m_3)((m_1+2m_2)l_1^2+4I_1))+4k_3l_1^2l_2^2m_2^2\dot{\theta_3}+16k_3l_1^2l_2^2m_3^2\dot{\theta_3}+8I_2k_3l_1^2m_1\dot{\theta_3}+8I_1k_3l_2^2m_2\dot{\theta_3}+32I_2k_3l_1^2m_2\dot{\theta_3}+32I_1k_3l_2^2m_3\dot{\theta_3}+32I_2k_3l_1^2m_3\theta_3-4k_1l_1l_3m_3\dot{\theta_1}(cos(\theta_1-\theta_3)(2m_3l_2^2+4I_2)-l_2^2cos(\theta_1-2\theta_2+\theta_3)(m_2+2m_3))-16I_1I_2gl_3m_3\sin(\theta_3)-4I_2l_1^2l_3^2m_3^2\dot{\theta_3}^2\sin(2\theta_1-2\theta_3)-4I_1l_2^2l_3^2m_3^2\dot{\theta_3}^2\sin(2\theta_2-2\theta_3)+8I_2gl_1^2l_3m_3^2\sin(2\theta_1-\theta_3)+8I_1gl_2^2l_3m_3^2\sin(2\theta_2-\theta_3)+2k_3l_1^2l_2^2m_1m_2\dot{\theta_3}+8k_3l_1^2l_2^2m_1m_3\dot{\theta_3}+24k_3l_1^2l_2^2m_2m_3\dot{\theta_3}-4k_2l_2l_3m_3\dot{\theta_2}(cos(\theta_2-\theta_3)((m_1+3m_2+2m_3)l_1^2+4I_1)-l_1^2cos(2\theta_1-\theta_2-\theta_3)(m_2+2m_3))-8I_1gl_2^2l_3m_3^2\sin(\theta_3)-8I_2gl_1^2l_3m_3^2\sin(\theta_3)-4k_3l_1^2l_2^2m_2^2\dot{\theta_3}cos(2\theta_1-2\theta_2)-16k_3l_1^2l_2^2m_3^2\dot{\theta_3}cos(2\theta_1-2\theta_2)-8I_2gl_1^2l_3m_2m_3\sin(\theta_3)-16k_3l_1^2l_2^2m_2m_3\dot{\theta_3}cos(2\theta_1-2\theta_2)-l_1^2l_2^2l_3^2m_1m_3^2\dot{\theta_3}^2\sin(2\theta_2-2\theta_3)+l_1^2l_2^2l_3^2m_2m_3^2\dot{\theta_3}^2\sin(2\theta_1-2\theta_3)-2l_1^2l_2^2l_3^2m_2m_3^2\dot{\theta_3}^2\sin(2\theta_2-2\theta_3)+2gl_1^2l_2^2l_3m_1m_3^2\sin(2\theta_1-\theta_3)-gl_1^2l_2^2l_3m_2^2m_3\sin(2\theta_1-\theta_3)+2gl_1^2l_2^2l_3m_2m_3^2\sin(2\theta_2-\theta_3)+gl_1^2l_2^2l_3m_2^2m_3\sin(2\theta_2-\theta_3)+4I_2gl_1^2l_3m_1m_3\sin(2\theta_1-\theta_3)+8I_2gl_1^2l_3m_2m_3\sin(2\theta_1-\theta_3)+4I_1gl_2^2l_3m_2m_3\sin(2\theta_2-\theta_3)-2gl_1^2l_2^2l_3m_1m_3^2\sin(2\theta_1-2\theta_2+\theta_3)-2gl_1^2l_2^2l_3m_2m_3^2\sin(2\theta_1-2\theta_2+\theta_3)-gl_1^2l_2^2l_3m_2^2m_3\sin(2\theta_1-2\theta_2+\theta_3)+gl_1^2l_2^2l_3m_2^2m_3\sin(\theta_3)-gl_1^2l_2^2l_3m_1m_2m_3\sin(2\theta_1-2\theta_2+\theta_3)))/(64I_1I_2I_3+8I_3l_1^2l_2^2m_2^2+8I_1l_2^2l_3^2m_3^2+8I_2l_1^2l_3^2m_3^2+32I_3l_1^2l_2^2m_3^2+16I_2I_3l_1^2m_1+16I_1I_3l_2^2m_2+64I_2I_3l_1^2m_2+16I_1I_2l_3^2m_3+64I_1I_3l_2^2m_3+64I_2I_3l_1^2m_3+4I_3l_1^2l_2^2m_1m_2+4I_2l_1^2l_3^2m_1m_3+16I_3l_1^2l_2^2m_1m_3+4I_1l_2^2l_3^2m_2m_3+16I_2l_1^2l_3^2m_2m_3+48I_3l_1^2l_2^2m_2m_3-8I_1l_2^2l_3^2m_3^2cos(2\theta_2-2\theta_3)-2l_1^2l_2^2cos(2\theta_1-2\theta_2)(m_2+2m_3)(m_2m_3l_3^2+4I_3(m_2+2m_3))-2l_1^2l_3^2m_3^2cos(2\theta_1-2\theta_3)(4I_2-m_2l_2^2)+2l_1^2l_2^2l_3^2m_1m_3^2+6l_1^2l_2^2l_3^2m_2m_3^2+2l_1^2l_2^2l_3^2m_2^2m_3+l_1^2l_2^2l_3^2m_1m_2m_3-2l_1^2l_2^2l_3^2m_1m_3^2cos(2\theta_2-2\theta_3)-4l_1^2l_2^2l_3^2m_2m_3^2cos(2\theta_2-2\theta_3))$$

Sympy

Fortunately, there are easier approaches than **brute-force algebra**, that rely on higher abstractions: one such approach is known as **Kane's Method**. This method still involves a significant amount of book-keeping for any but the most trivial problems, but the **Sympy** package has a nice implementation that handles the details for you.

Kane's Method in Physics/Mechanics

`mechanics` provides functionality for deriving equations of motion using Kane's method [Kane1985]. This document will describe Kane's method as used in this module, but not how the equations are actually derived.

Structure of Equations

In `mechanics` we are assuming there are 5 basic sets of equations needed to describe a system. They are: holonomic constraints, non-holonomic constraints, kinematic differential equations, dynamic equations, and differentiated non-holonomic equations.

$$\begin{aligned}
\mathbf{f_h}(q,t) &= 0 \\
\mathbf{k_{nh}}(q,t)u + \mathbf{f_{nh}}(q,t) &= 0 \\
\mathbf{k_{k\dot{q}}}(q,t)\dot{q} + \mathbf{k_{ku}}(q,t)u + \mathbf{f_k}(q,t) &= 0 \\
\mathbf{k_d}(q,t)\dot{u} + \mathbf{f_d}(q,\dot{q},u,t) &= 0 \\
\mathbf{k_{dnh}}(q,t)\dot{u} + \mathbf{f_{dnh}}(q,\dot{q},u,t) &= 0
\end{aligned}$$

In `mechanics` holonomic constraints are only used for the linearization process; it is assumed that they will be too complicated to solve for the dependent coordinate(s). If you are able to easily solve a holonomic constraint, you should consider redefining your problem in terms of a smaller set of coordinates. Alternatively, the time-differentiated holonomic constraints can be supplied.

Kane's method forms two expressions, F_r and F_r^*, whose sum is zero. In this module, these expressions are rearranged into the following form:

$$\mathbf{M}(q,t)\dot{u} = \mathbf{f}(q,\dot{q},u,t)$$

Kane's Method in Physics/Mechanics

`Sympy.physics.mechanics` provides functionality for deriving equations of motion using Kane's method [Kane1985]. With all of the necessary point velocities and particle masses defined, the **`KanesMethod`** class can be used to derive the equations of motion of the system automatically.

This method is a subset of the group of methods known as "**Lagrange's form of D'Alembert's Principle**". The Newton-Euler equations are multiplied by 'special vectors' to develop scalar representations of the forces acting on each body.

Thomas R. **Kane** taught mechanics and computation for 45 years, has published 10 textbooks and 172 technical papers, and is the preeminent expert and author of modern dynamics theory (known as "**Kane Dynamics**").

Kanisms - sayings by Thomas Kane

Kane's 1st theorem - "***Nothing is equal to anything***."
Kane's 2nd theorem - "***Everything is equal to everything else***."
"When you're not sure whether you know or not - **you don't know**."
"**Lets go slow - we do not have time to go fast**."
"**Differential equations come in two kinds, good and bad - and there are no good ones**."
"Linear algebra is simply a method of bookkeeping."
"Always keep an extra negative sign in your pocket."
"Avoid differentiation, especially vectors."
Newton's 1st law: "***An object moves in a straight line with a constant speed, unless it doesn't***."
"UHT: useful half-truth."
"Your boss doesn't know anything - by definition."
"Always be scared to say anything."
"When in doubt - cheat."

Integrate the pendulum

```
# Step 1: construct the pendulum model

# Generalized coordinates and velocities
# (in this case, angular positions & velocities of each mass)
q = mechanics.dynamicsymbols('q:{0}'.format(n))
u = mechanics.dynamicsymbols('u:{0}'.format(n))

# mass and length
m = symbols('m:{0}'.format(n))
l = symbols('l:{0}'.format(n))

# gravity and time symbols
g, t = symbols('g,t')
```

…

Integrate the pendulum

```
# Step 2: build the model using Kane's Method

# Create pivot point reference frame
A = mechanics.ReferenceFrame('A')
P = mechanics.Point('P')
P.set_vel(A, 0)
# lists to hold particles, forces, and kinetic ODEs
# for each pendulum in the chain
particles = []
forces = []
kinetic_odes = []
```

...

```
# Generate equations of motion
KM = mechanics.KanesMethod(A, q_ind=q, u_ind=u,
                           kd_eqs=kinetic_odes)
fr, fr_star = KM.kanes_equations(forces, particles)
```

Integrate the pendulum

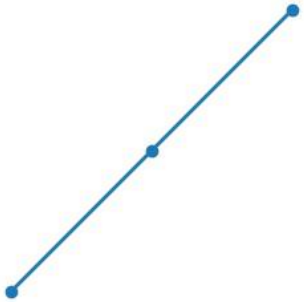
```
# Step 3: numerically evaluate equations and integrate

# initial positions and velocities - assumed to be given in degrees
y0 = np.deg2rad(np.concatenate([np.broadcast_to(initial_positions, n),
                                np.broadcast_to(initial_velocities, n)]))

# lengths and masses
if lengths is None:
    lengths = np.ones(n) / n
lengths = np.broadcast_to(lengths, n)
masses = np.broadcast_to(masses, n)
...
# ODE integration
return odeint(gradient, y0, times, args=(parameter_vals,))
```

Integrating the pendulum returns generalized coordinates, which in this case are the angular position and velocity of each pendulum segment, relative to vertical.

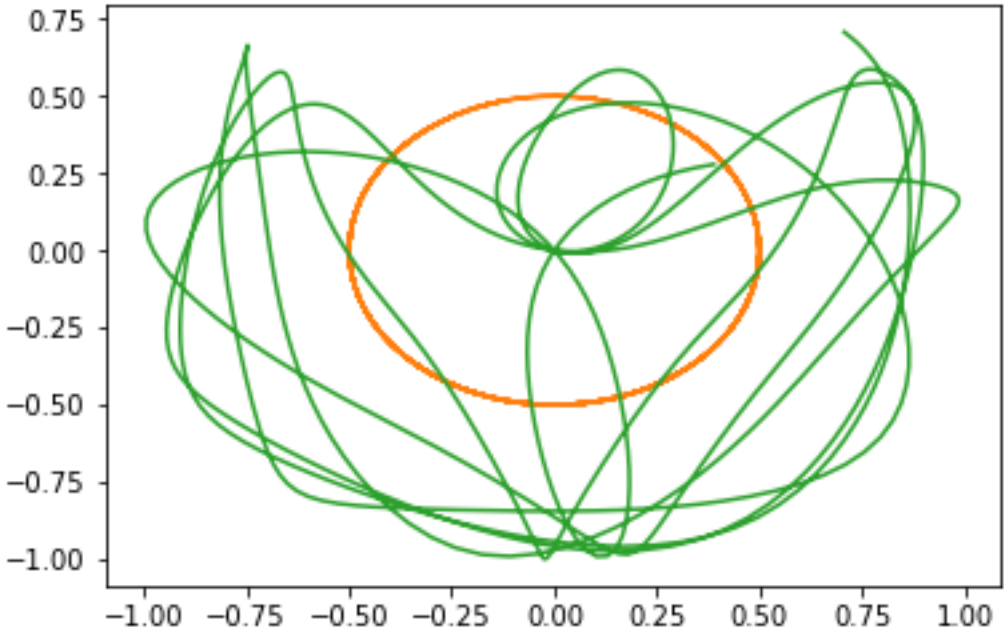
In order to visualize the pendulum, we need a utility to extract x and y coordinates from these angular coordinates

```
def get_xy_coords(p, lengths=None):
    """Get (x, y) coordinates from generalized coordinates p"""
```

Double Pendulum



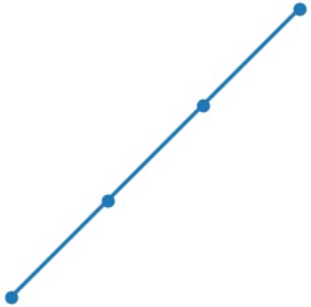
Double Pendulum



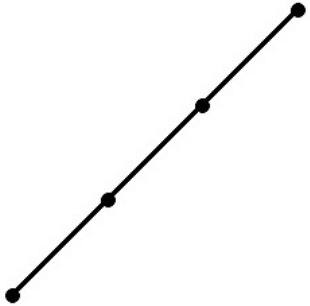
Double Pendulum

```
####################
Number of pendulum 2
####################
Matrix([
[-g*l0*m0*sin(q0(t)) - g*l0*m1*sin(q0(t))],
[                     -g*l1*m1*sin(q1(t))]])

Matrix([
[-l0*l1*m1*(sin(q0(t))*sin(q1(t)) + cos(q0(t))*cos(q1(t)))*Derivative(u1(t), t) + l0*l1*m1*(-
sin(q0(t))*cos(q1(t)) + sin(q1(t))*cos(q0(t)))*u1(t)**2 - (l0**2*m0 +
l0**2*m1)*Derivative(u0(t), t)],
[              -l0*l1*m1*(sin(q0(t))*sin(q1(t)) + cos(q0(t))*cos(q1(t)))*Derivative(u0(t), t) +
l0*l1*m1*(sin(q0(t))*cos(q1(t)) - sin(q1(t))*cos(q0(t)))*u0(t)**2 - l1**2*m1*Derivative(u1(t),
t)]])
```

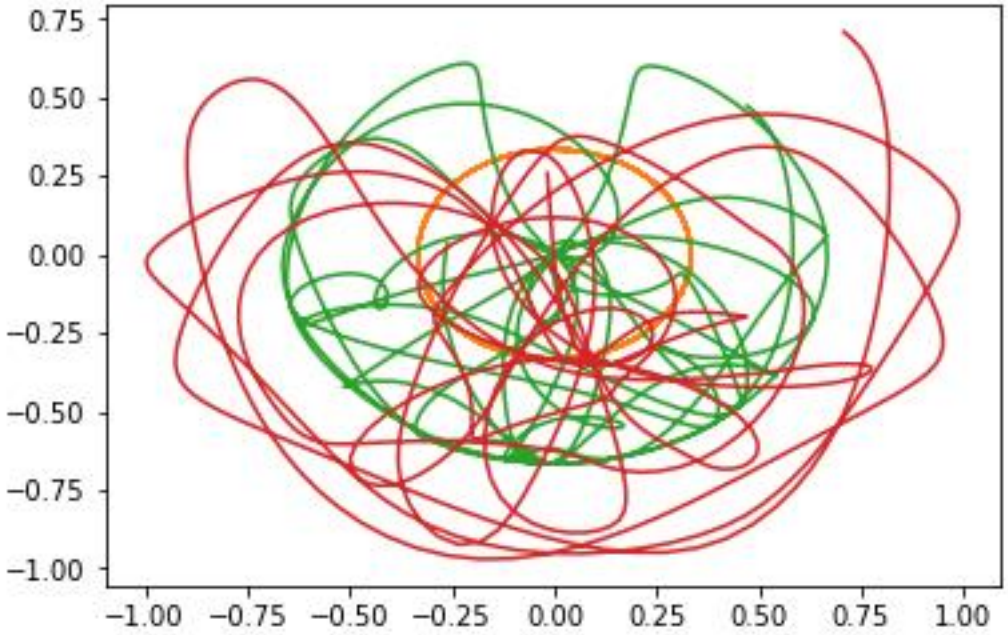
Triple Pendulum



Triple Pendulum

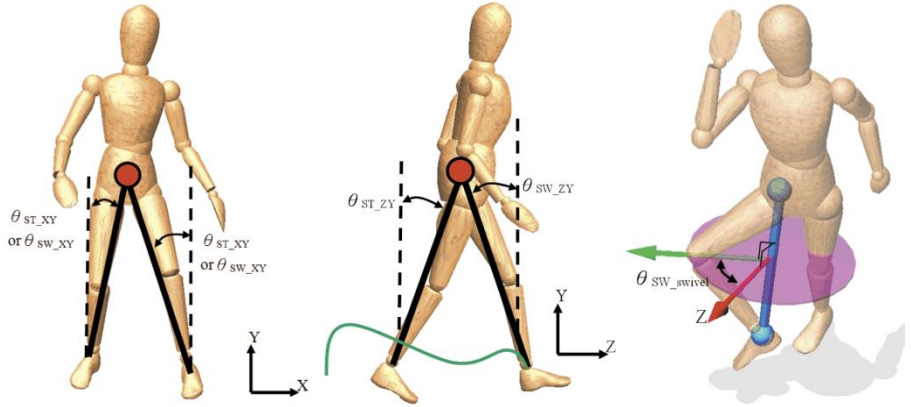


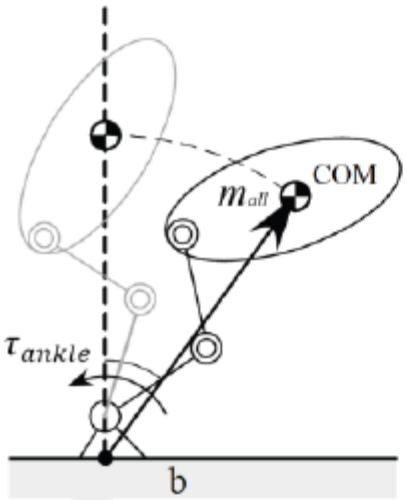
Triple Pendulum

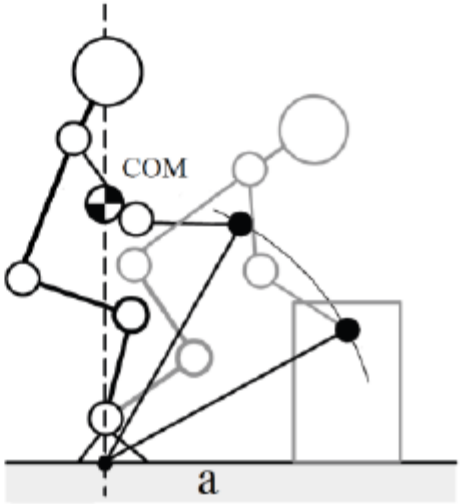


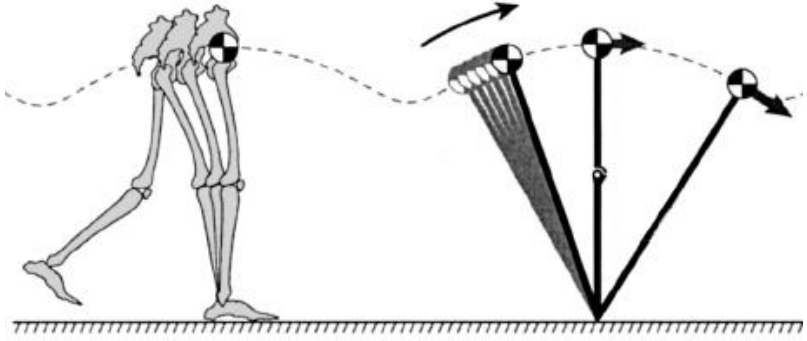
```
####################
Number of pendulum 3
####################
Matrix([
[-g*l0*m0*sin(q0(t)) - g*l0*m1*sin(q0(t)) - g*l0*m2*sin(q0(t))],
[                     -g*l1*m1*sin(q1(t)) - g*l1*m2*sin(q1(t))],
[                                          -g*l2*m2*sin(q2(t))]])
Matrix([
[l0*l1*m1*(-sin(q0(t))*cos(q1(t)) + sin(q1(t))*cos(q0(t)))*u1(t)**2 + l0*l1*m2*(-
sin(q0(t))*cos(q1(t)) + sin(q1(t))*cos(q0(t)))*u1(t)**2 - l0*l2*m2*(sin(q0(t))*sin(q2(t)) +
cos(q0(t))*cos(q2(t)))*Derivative(u2(t), t) + l0*l2*m2*(-sin(q0(t))*cos(q2(t)) +
sin(q2(t))*cos(q0(t)))*u2(t)**2 - (l0*l1*m1*(sin(q0(t))*sin(q1(t)) + cos(q0(t))*cos(q1(t))) +
l0*l1*m2*(sin(q0(t))*sin(q1(t)) + cos(q0(t))*cos(q1(t))))*Derivative(u1(t), t) - (l0**2*m0 +
l0**2*m1 + l0**2*m2)*Derivative(u0(t), t)],
[             l0*l1*m1*(sin(q0(t))*cos(q1(t)) - sin(q1(t))*cos(q0(t)))*u0(t)**2 +
l0*l1*m2*(sin(q0(t))*cos(q1(t)) - sin(q1(t))*cos(q0(t)))*u0(t)**2 - l1*l2*m2*(sin(q1(t))*sin(q2(t))
+ cos(q1(t))*cos(q2(t)))*Derivative(u2(t), t) + l1*l2*m2*(-sin(q1(t))*cos(q2(t)) +
sin(q2(t))*cos(q1(t)))*u2(t)**2 - (l1**2*m1 + l1**2*m2)*Derivative(u1(t), t) -
(l0*l1*m1*(sin(q0(t))*sin(q1(t)) + cos(q0(t))*cos(q1(t))) + l0*l1*m2*(sin(q0(t))*sin(q1(t)) +
cos(q0(t))*cos(q1(t))))*Derivative(u0(t), t)],
[
-l0*l2*m2*(sin(q0(t))*sin(q2(t)) + cos(q0(t))*cos(q2(t)))*Derivative(u0(t), t) +
l0*l2*m2*(sin(q0(t))*cos(q2(t)) - sin(q2(t))*cos(q0(t)))*u0(t)**2 - l1*l2*m2*(sin(q1(t))*sin(q2(t))
+ cos(q1(t))*cos(q2(t)))*Derivative(u1(t), t) + l1*l2*m2*(sin(q1(t))*cos(q2(t)) -
sin(q2(t))*cos(q1(t)))*u1(t)**2 - l2**2*m2*Derivative(u2(t), t)]])
```

Why Pendulum is so important ?

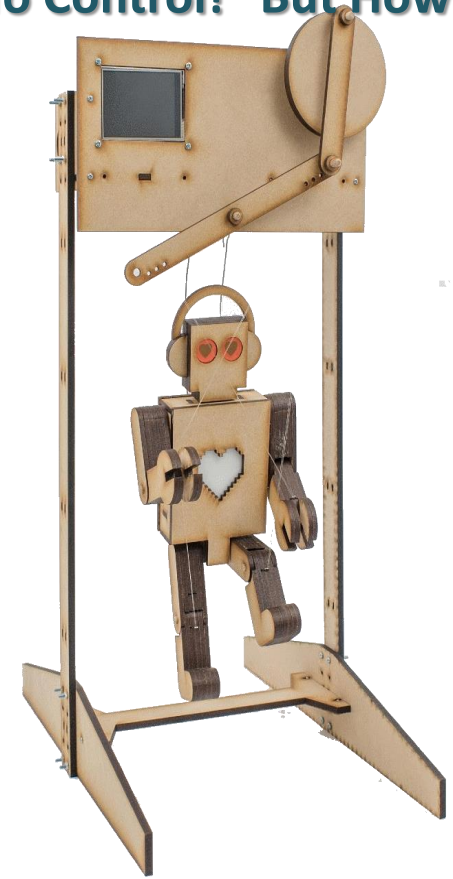


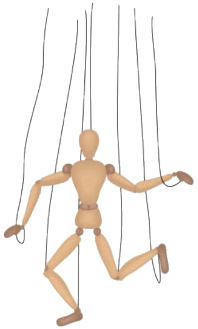




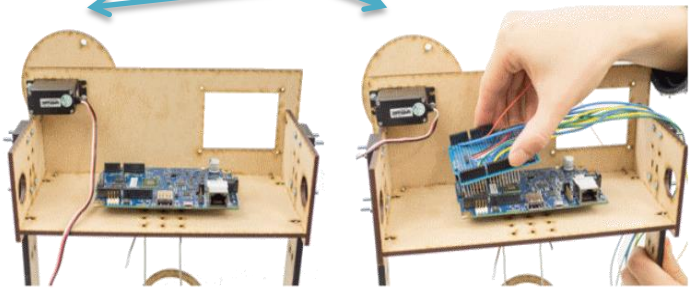


So We Need To Control! But How





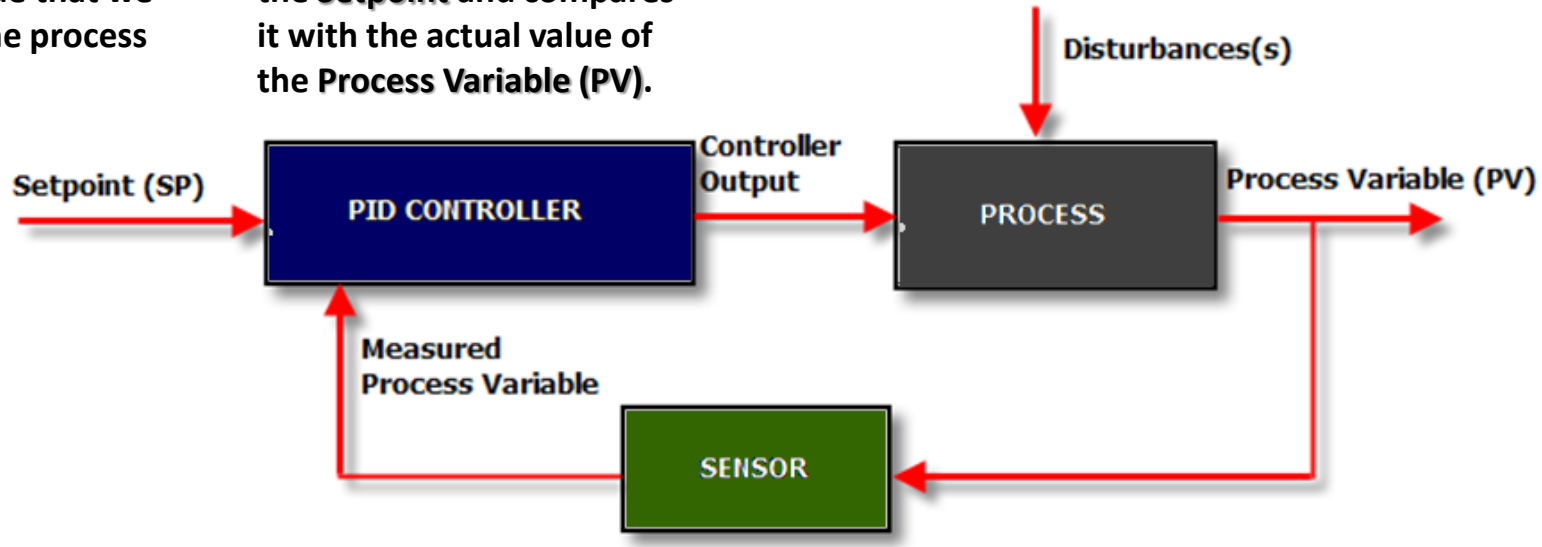
SERVO MOTOR



What Is Pid Control?

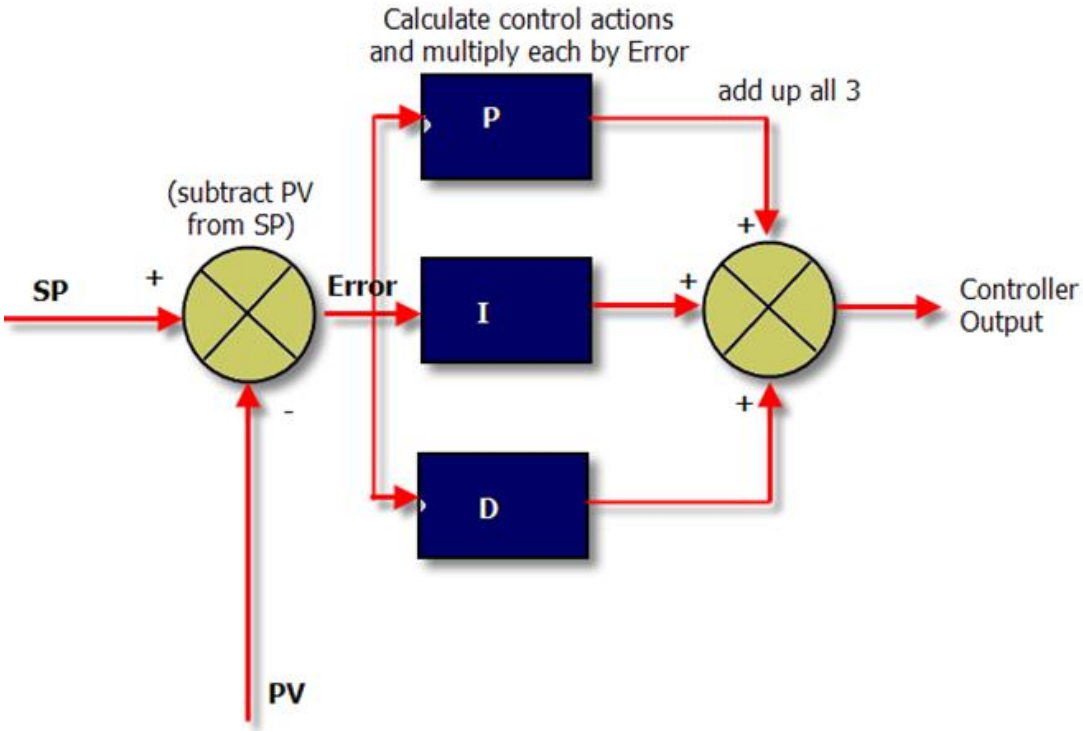
The **Setpoint (SP)** is the value that we want the process to be

The PID controller looks at the **Setpoint** and compares it with the actual value of the **Process Variable (PV)**.



If the **SP** and the **PV** are the same – then the controller is a very happy little box. It doesn't have to do anything, it will set its output to zero. However, if there is a disparity between the **SP** and the **PV** we have an error and corrective action is needed.

Understanding The Controller?



The **PV** is subtracted from the **SP** to create the Error. The error is simply multiplied by one, two or all of the calculated P, I and D actions (depending which ones are turned on). Then the resulting "error x control actions" are added together and sent to the controller output.

These 3 modes are used in different combinations:

P – Sometimes used

PI - Most often used

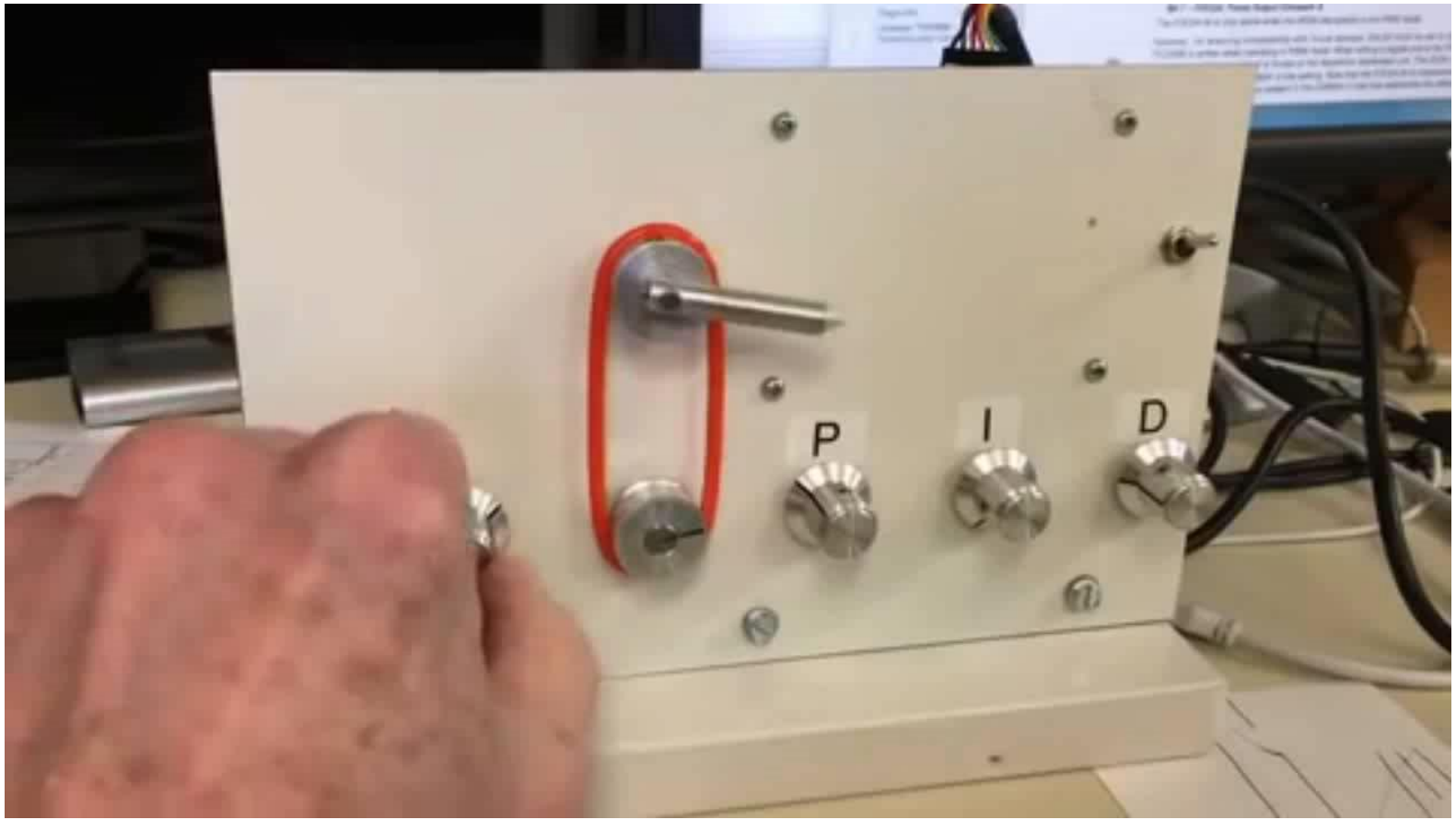
PID – Sometimes used

PD – rare, but it can be useful for controlling servomotors ???

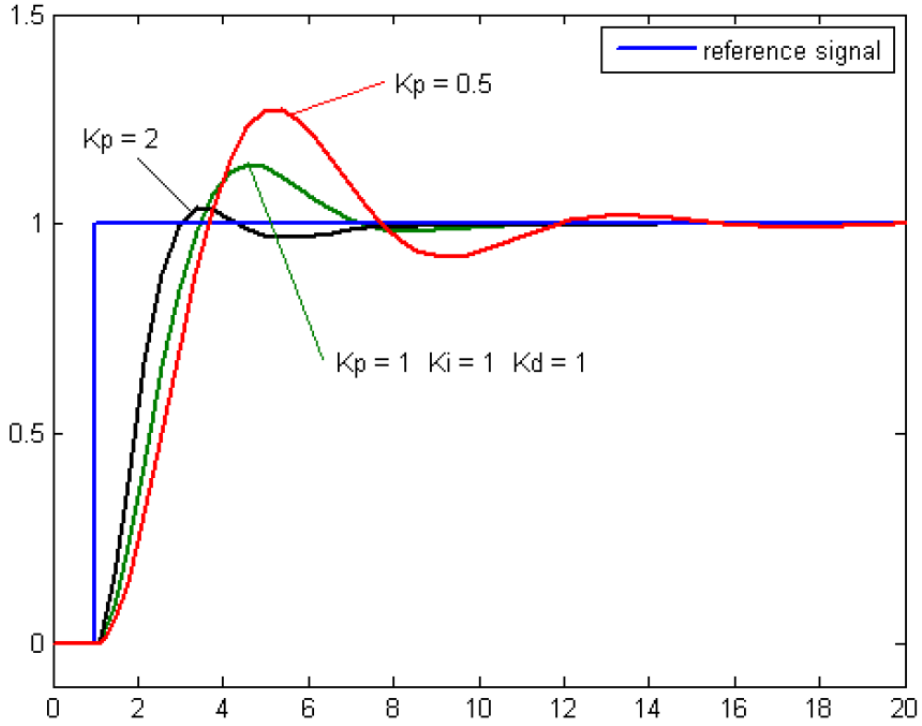
Forward Dynamics Without Feedback Control

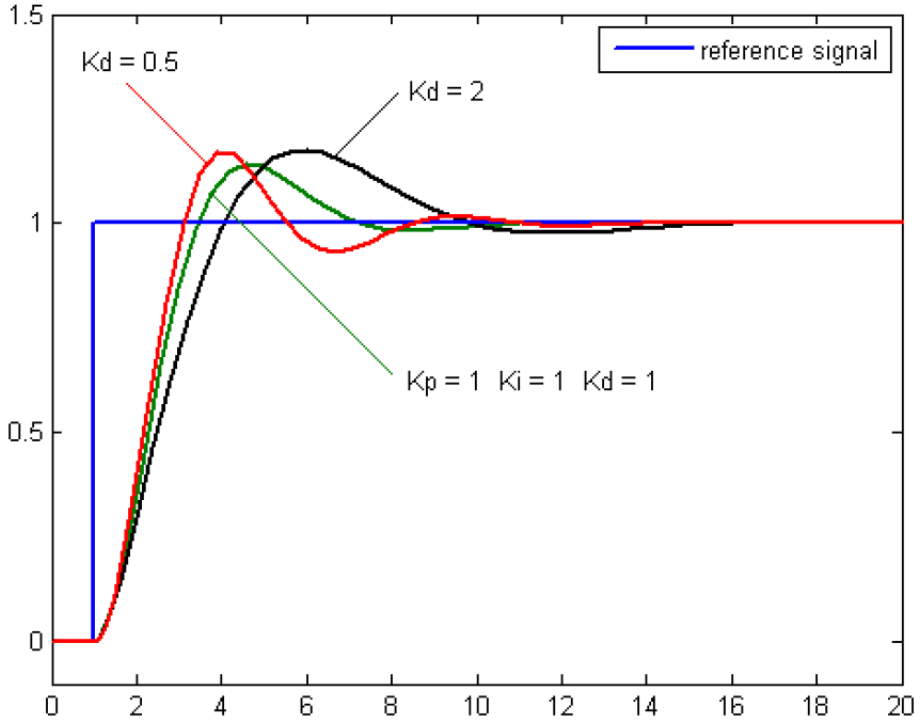
E.Kavafoğlu, S.Aritan (2017) Real-time Physics-based Motion Control With An Efficient Inverse Dynamics Method. .MSc Thesis.
Institute of Informatics of Hacettepe University

PHYSICS in COMPUTER ANIMATIONS and GAMES



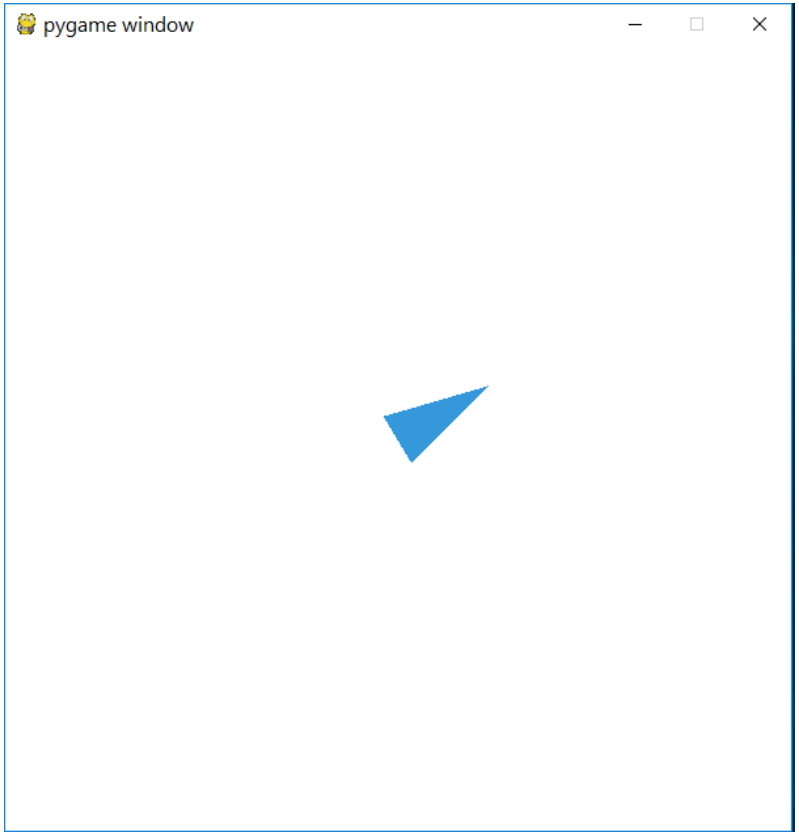
PD Controllers

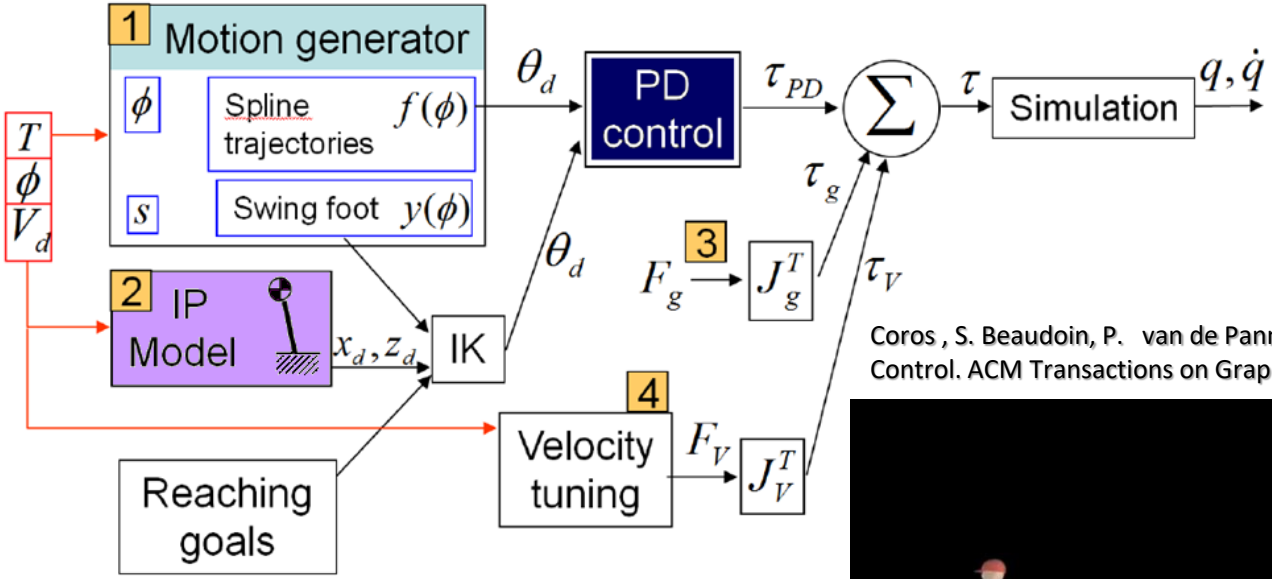




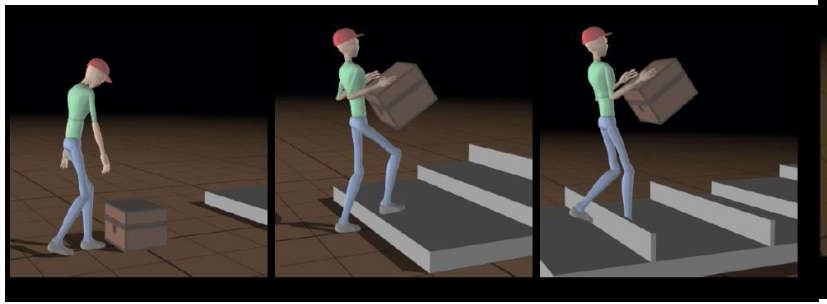
PD Controllers

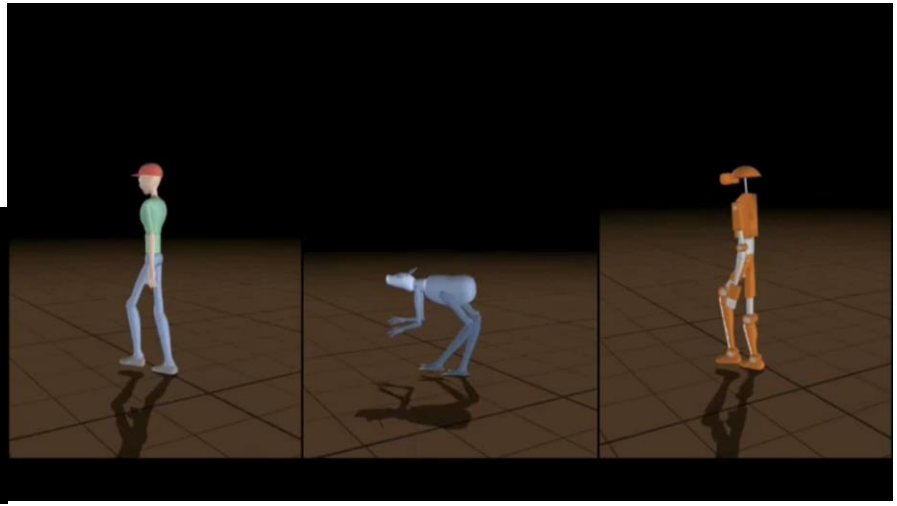
`pymunkPD.py`



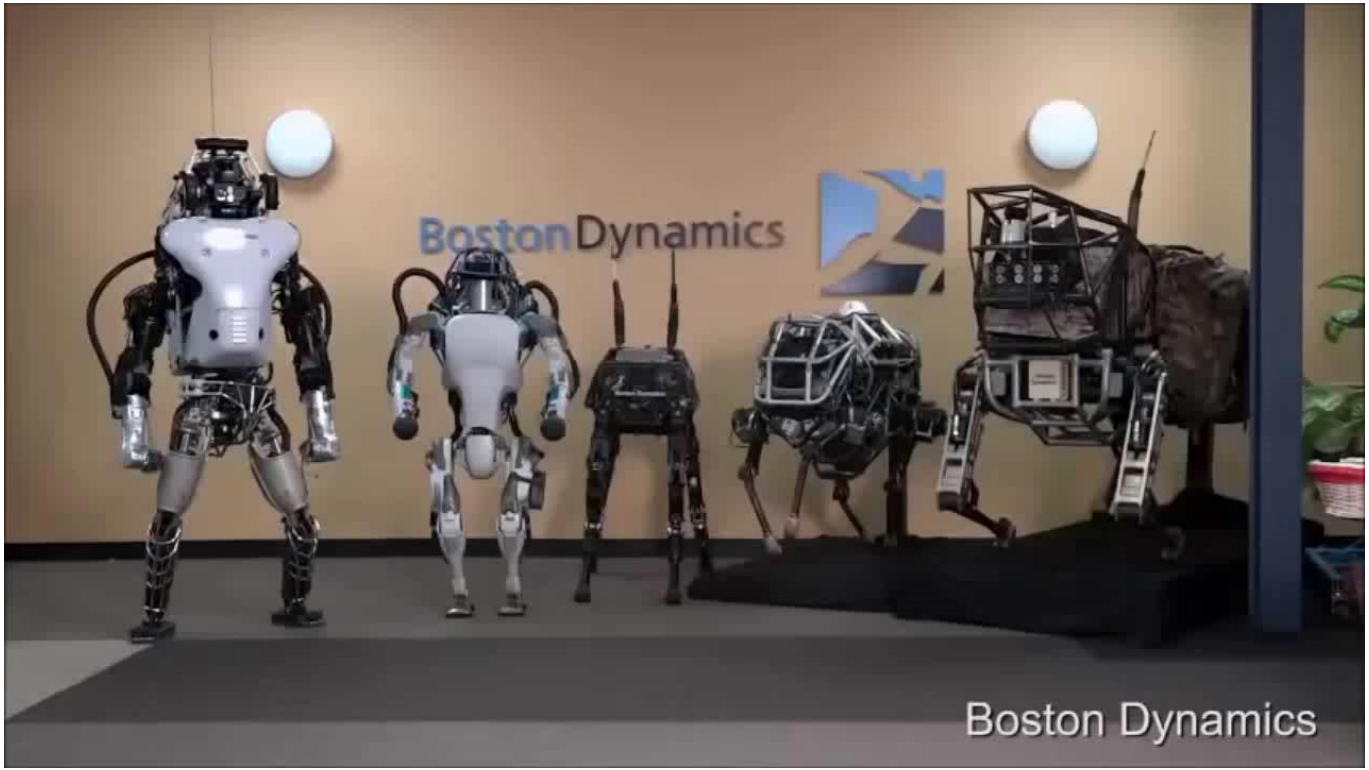


Coros , S. Beaudoin, P. van de Panne, M. (2010) Generalized Biped Walking Control. ACM Transactions on Graphics, Vol. 29, No. 4, Article 130,



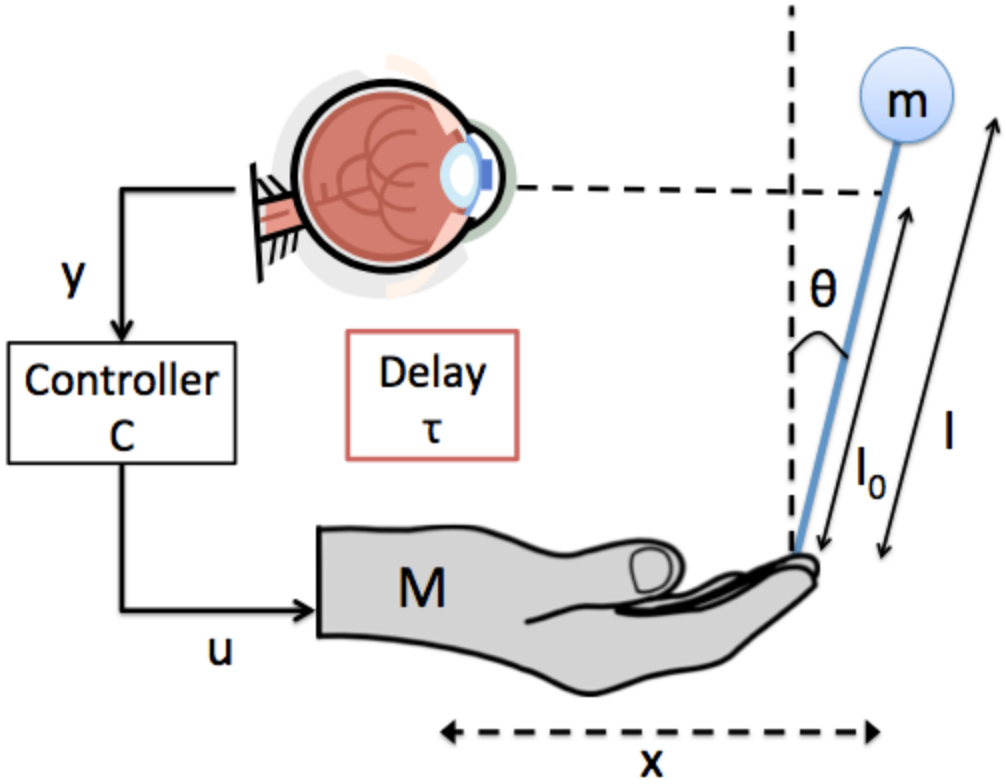


PHYSICS in COMPUTER ANIMATIONS and GAMES



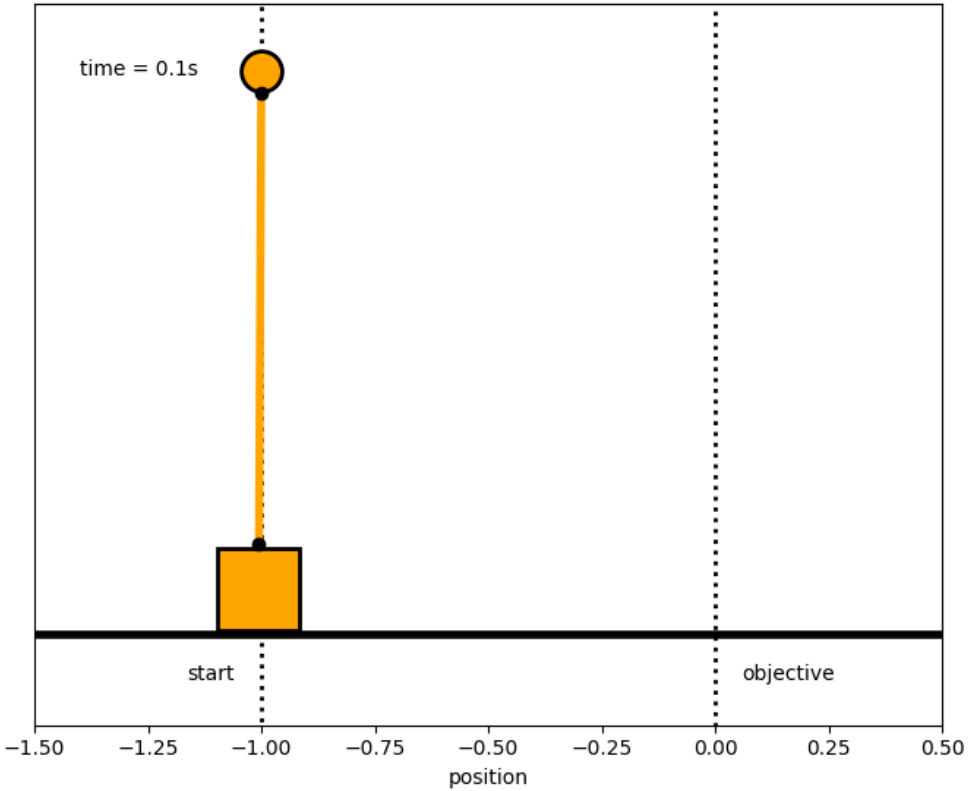
Generalized Biped Walking Control algorithms by Coros , et.al. used for the Agile Anthropomorphic Robot "Atlas" .

Controllers in Human



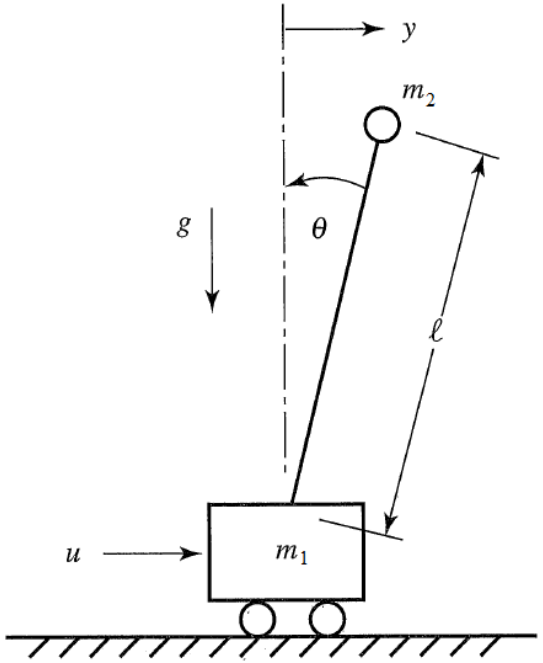
Controllers in Human

The cart can perform a sequence of moves to maneuver from position **y=-1.0** to **y=0.0** within **6.2 seconds**.



Inverted Pendulum Optimal Control

Verify that v, θ, and q are zero before and after the maneuver.



The inverted pendulum is described by the following dynamic equations:

$$\begin{bmatrix} \dot{y} \\ \dot{v} \\ \dot{\theta} \\ \dot{q} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\epsilon & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y \\ v \\ \theta \\ q \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \end{bmatrix} u$$

where **u** is the force applied to the cart, **ε** is **m2/(m1+m2)**, **y** is the position of the cart, **v** is the velocity of the cart, **θ** is the angle of the pendulum relative to the cart, **m1=10**, **m2=1**, and **q** is the rate of angle change. Tune the controller to minimize the use of force applied to the cart either in the forward or reverse direction

Inverted Pendulum Optimal Control

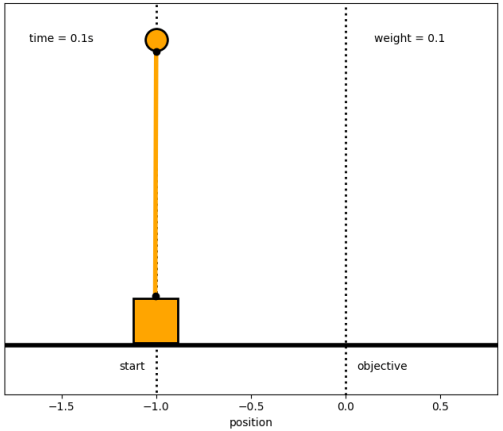
There are many methods to implement control including basic strategies such as a proportional-integral-derivative (**PID**) controller or more advanced methods such as model predictive techniques.

Model predictive control (**MPC**) is a type of control algorithm that is used to control systems with dynamics. It is a model-based method that uses a predictive model of the system to compute control actions that optimize a performance criterion over a finite horizon. The basic idea of **MPC** is to optimize a control policy that minimizes the difference between the desired reference signal and the output of the system, subject to constraints on the control inputs and the states of the system.

Inverted Pendulum Optimal Control

GEKKO DYNAMIC OPTIMIZATION

GEKKO is a Python package for machine learning and optimization of mixed-integer and differential algebraic equations. It is coupled with large-scale solvers for linear, quadratic, nonlinear, and mixed integer programming (LP, QP, NLP, MILP, MINLP). Modes of operation include parameter regression, data reconciliation, real-time optimization, dynamic simulation, and nonlinear predictive control.



https://gekko.readthedocs.io/en/latest/

This collection of 188 nonlinear programing test examples is a supplement of the test problem collection published by Hock and Schittkowski.

$$\min\ x_1 x_4 (x_1 + x_2 + x_3) + x_3$$

$$x_1 x_2 x_3 x_4 \geq 25$$

$$x_1{}^2 + x_2{}^2 + x_3{}^2 + x_4{}^2 = 40$$

$$1 \leq x_1, x_2, x_3, x_4 \leq 5$$

$$x_0 = (1, 5, 5, 1)$$

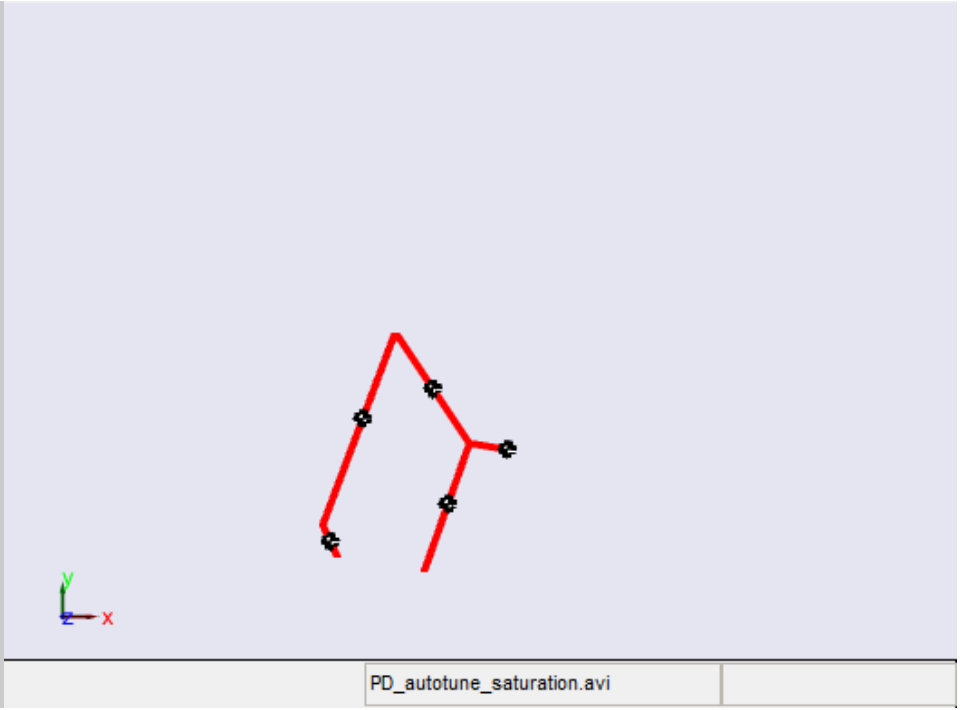
PHYSICS in COMPUTER ANIMATIONS and GAMES

```
from gekko import GEKKO

#Initialize Model
m = GEKKO()
#define parameter
eq = m.Param(value=40)
#initialize variables
x1,x2,x3,x4 = [m.Var(lb=1, ub=5) for i in range(4)]
#initial values
x1.value = 1
x2.value = 5
x3.value = 5
x4.value = 1
#Equations
m.Equation(x1*x2*x3*x4>=25)
m.Equation(x1**2+x2**2+x3**2+x4**2==eq)
#Objective
m.Minimize(x1*x4*(x1+x2+x3)+x3)
```

```
#Set global options
m.options.IMODE = 3 #steady state optimization

#Solve simulation
m.solve()

#Results
print('')
print('Results')
print('x1: ' + str(x1.value))
print('x2: ' + str(x2.value))
print('x3: ' + str(x3.value))
print('x4: ' + str(x4.value))
```

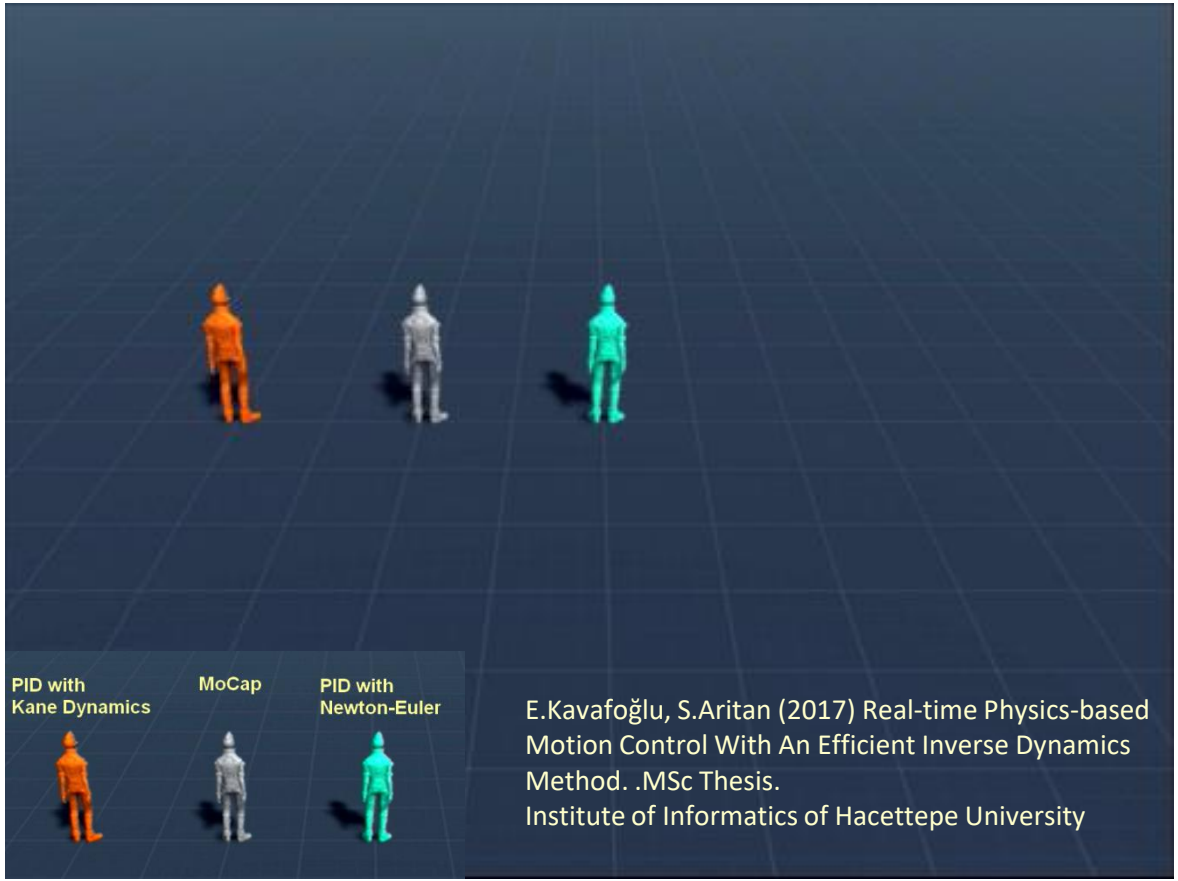
```
EXIT: Optimal Solution Found.

 The solution was found.

 The final value of the objective function is
17.0140171270735

 ---------------------------------------------------
 Solver          :  IPOPT (v3.12)
 Solution time   :   1.310000000012224E-002 sec
 Objective       :    17.0140171270735
 Successful solution
 ---------------------------------------------------

Results
x1: [1.000000057]
x2: [4.74299963]
x3: [3.8211500283]
x4: [1.3794081795]
```

```
from gekko import GEKKO
import numpy as np
m = GEKKO()
x = m.Array(m.Var,4,value=1,lb=1,ub=5)
x1,x2,x3,x4 = x                    # rename variables
x2.value = 5; x3.value = 5         # change guess
m.Equation(np.prod(x)>=25)         # prod>=25
m.Equation(m.sum([xi**2 for xi in x])==40) # sum=40
m.Minimize(x1*x4*(x1+x2+x3)+x3) # objective
m.solve()
print(x)
```

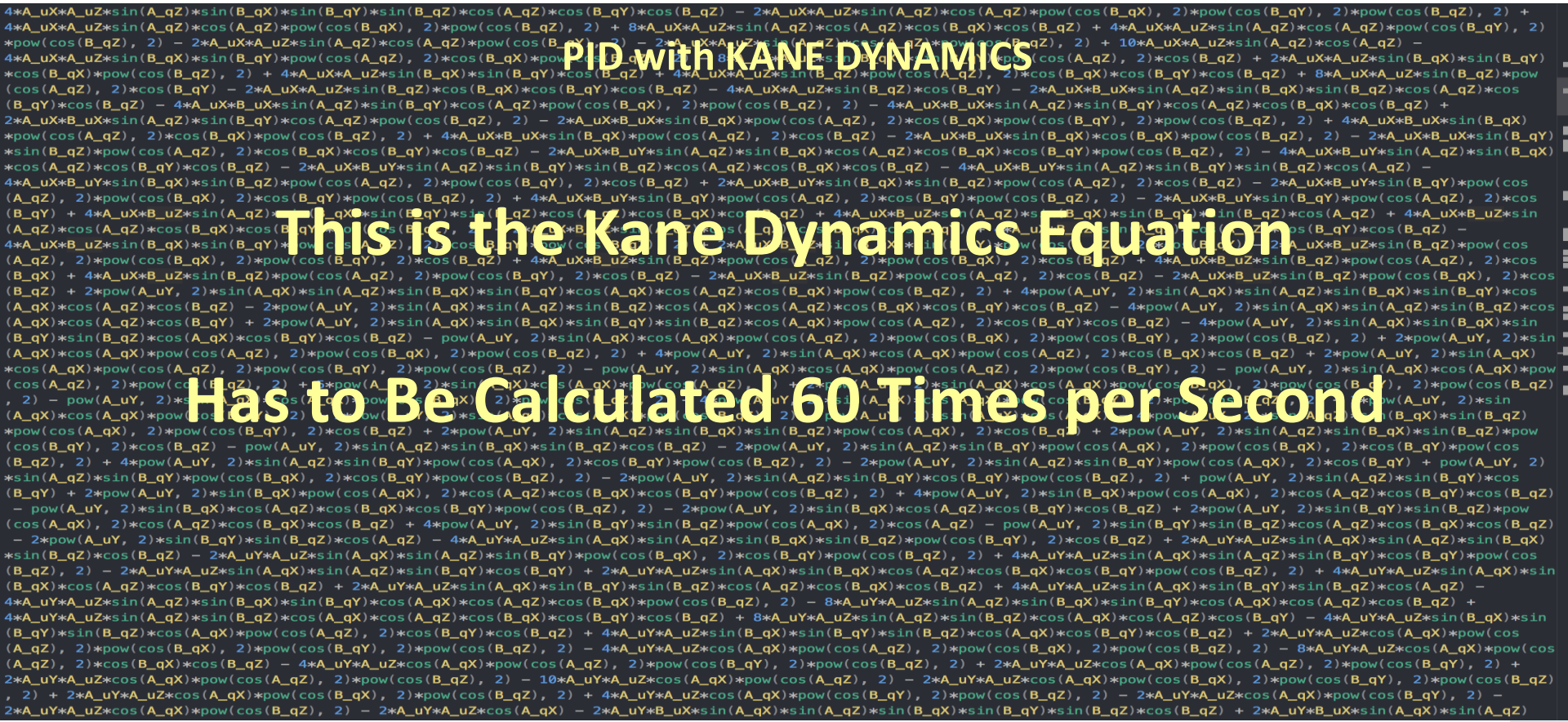


Özgören, NŞ. Arıtan, S. (2016) Evaluating responses of a feedback control system for a multi - link biomechanical model. 8th National Biomechanics Congress. Ankara.

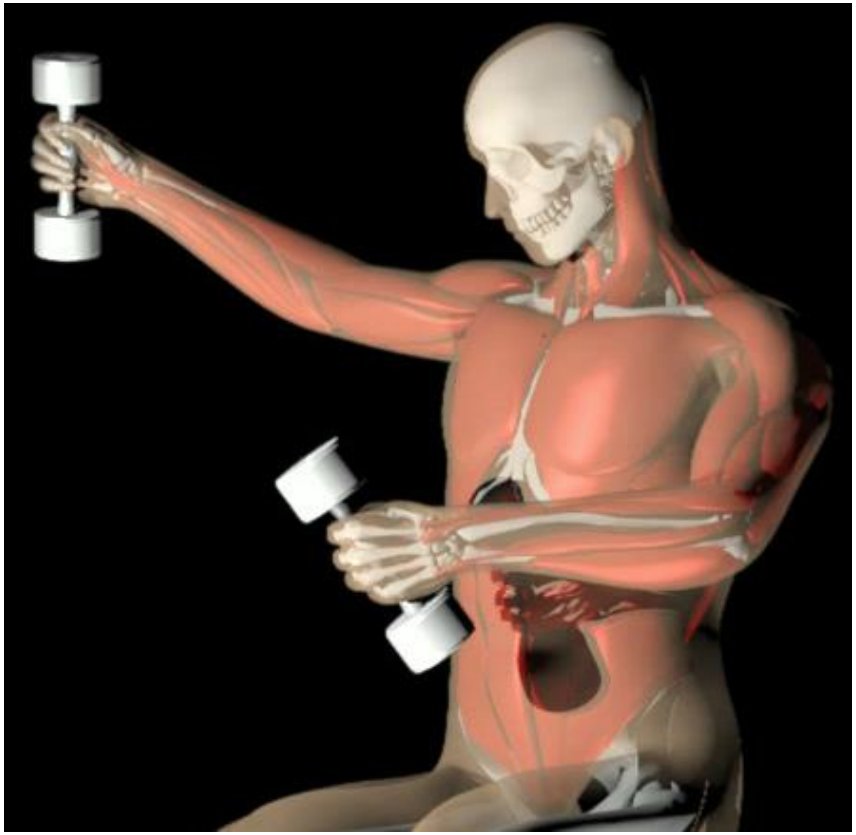
PHYSICS in COMPUTER ANIMATIONS and GAMES



E.Kavafoğlu, S.Aritan (2017) Real-time Physics-based Motion Control With An Efficient Inverse Dynamics Method. .MSc Thesis.
Institute of Informatics of Hacettepe University


PID with KANE DYNAMICS
This is the Kane Dynamics Equation
Has to Be Calculated 60 Times per Second

HOW TO INCLUDE MUSCLES INTO THE MODEL?



Optimizing Locomotion Controllers Using Biologically-Based Actuators and Objectives

Wang, JM., Hamner, SR., Delp, SL., Koltun V. from Stanford University

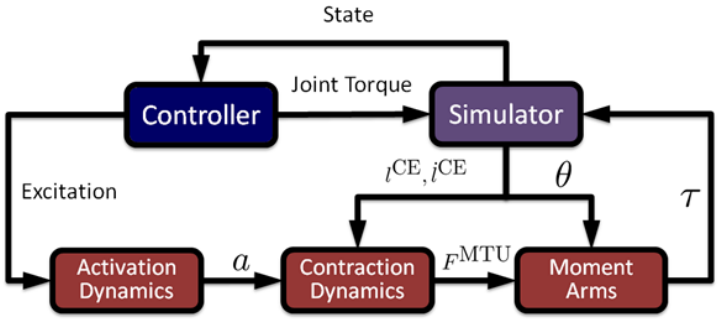
A more *biologically faithful force generation mechanism* is needed to generate more human-like motions

Graphics bipeds:

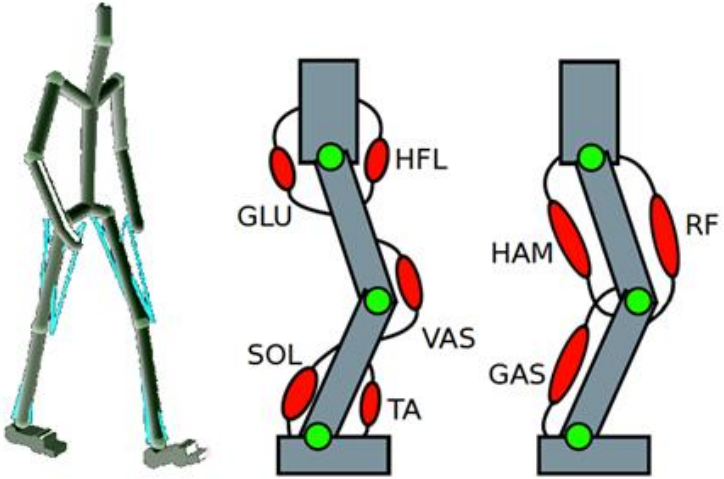
- Controller directly output joint torques

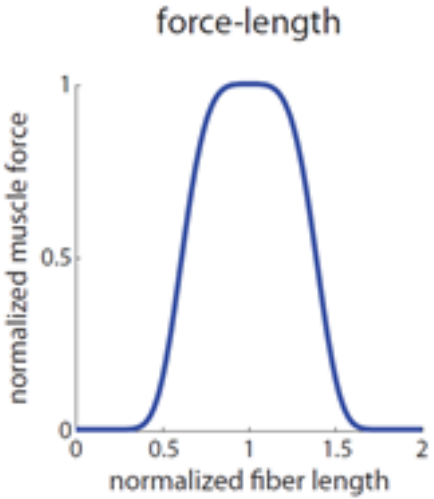
Humans:

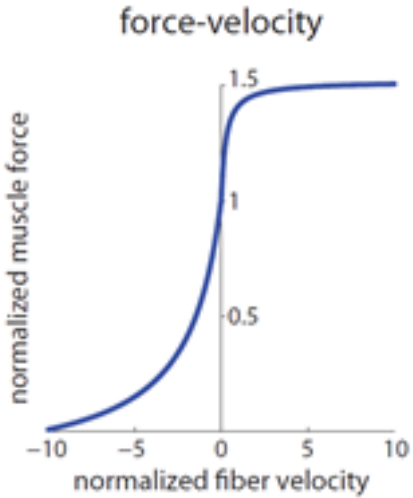
- Controller output neural excitation levels to musculotendon units (MTUs)
- MTUs generate forces constrained by muscle physiology



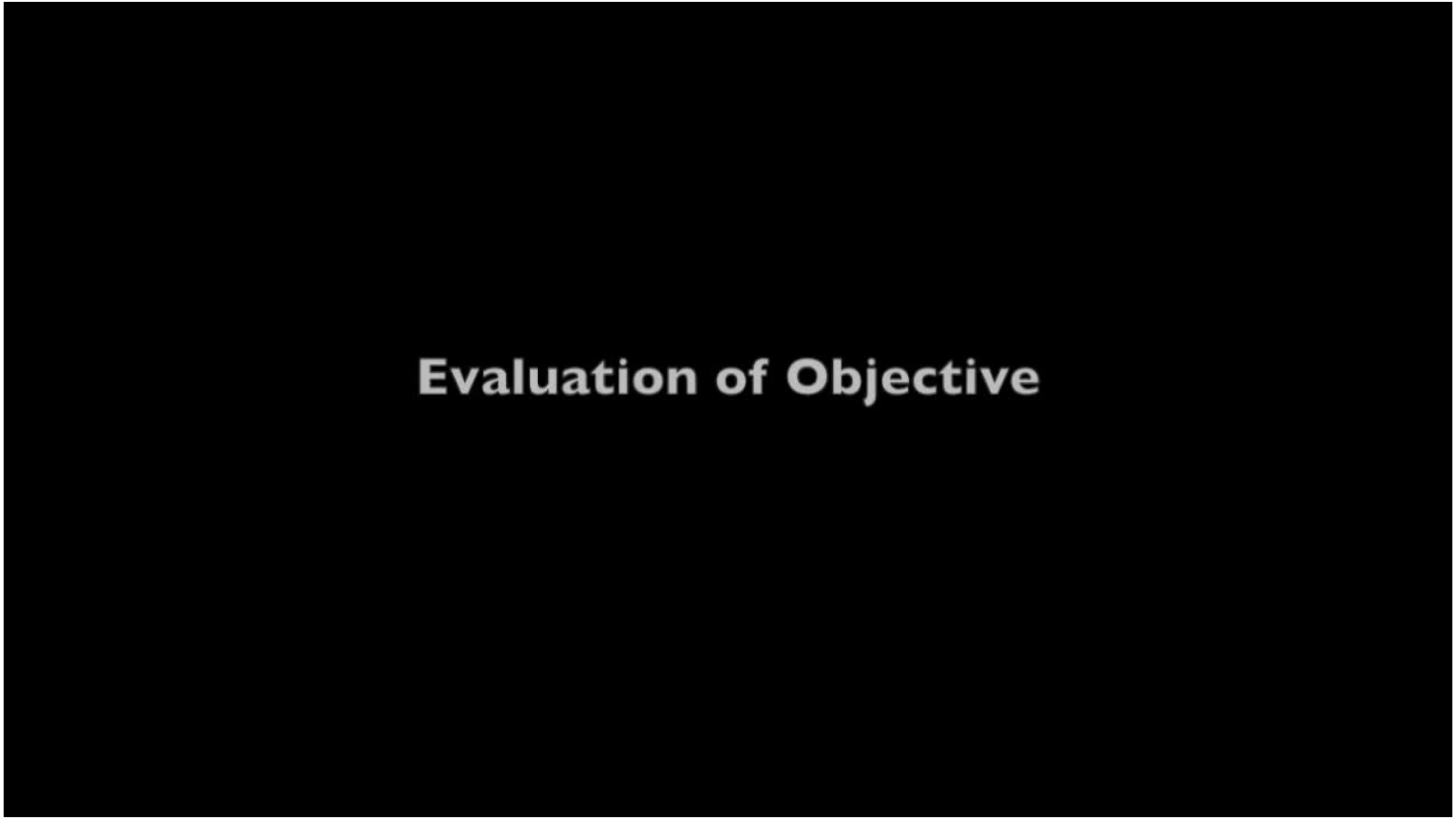
Optimizing Locomotion Controllers Using Biologically-Based Actuators and Objectives





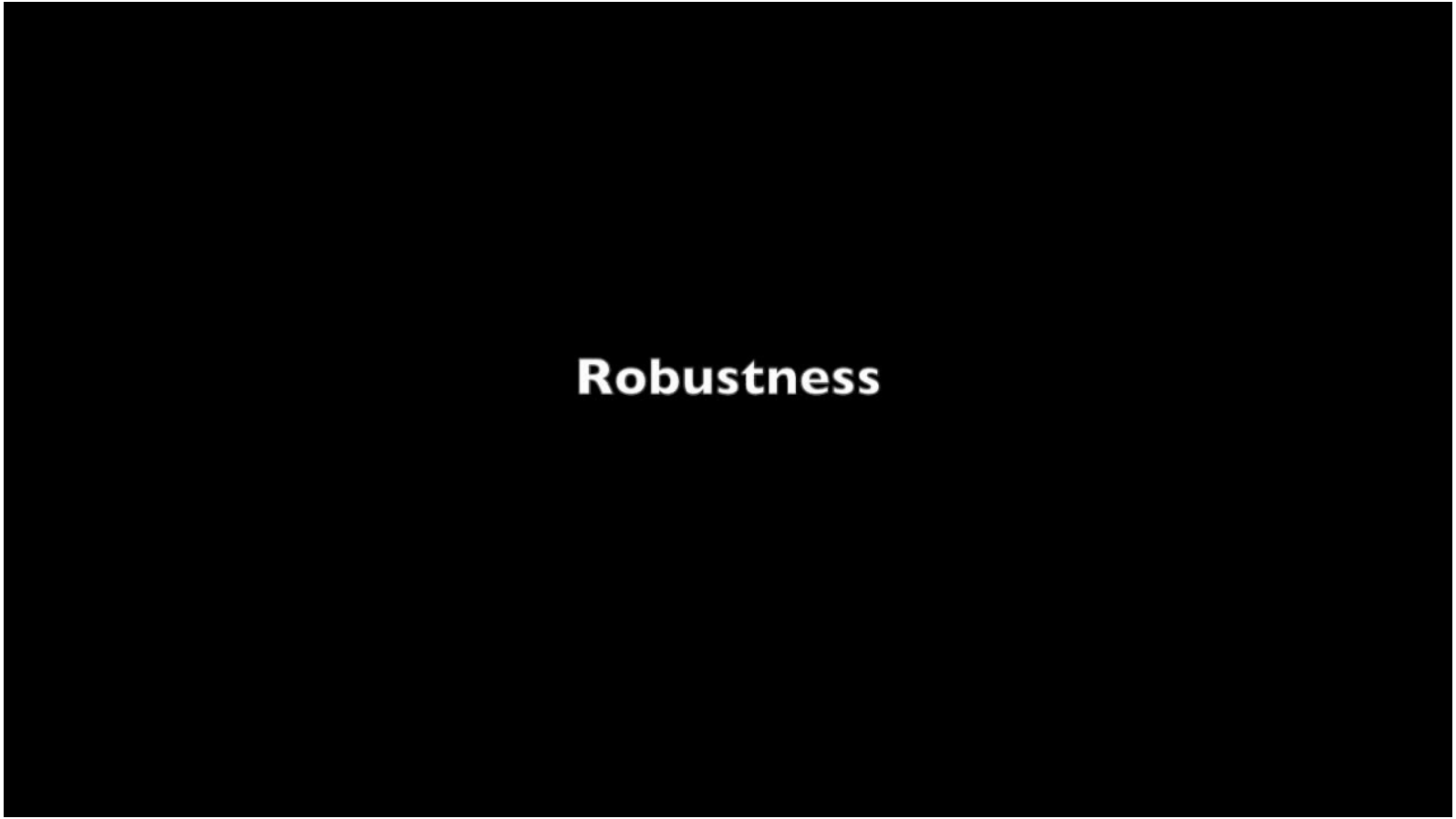



Optimization

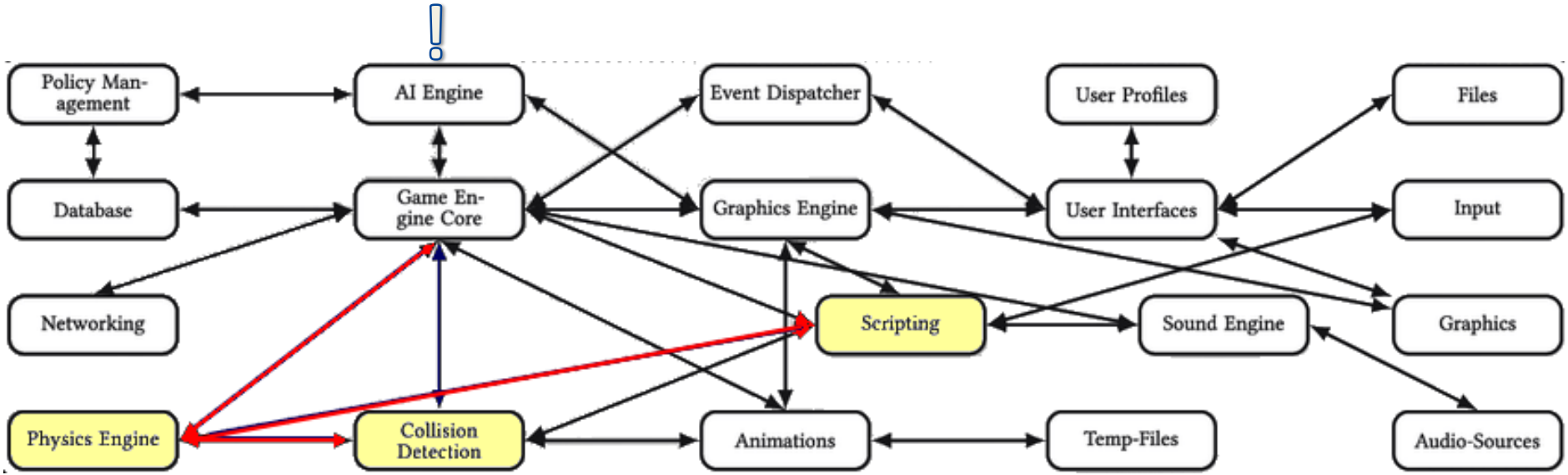

Evaluation of Objective


Changing Muscle Properties


Walking and Running at a Range of Speeds


Robustness

Game Engine Architecture



Learning to Simulate Dynamic Environments with GameGAN

Seung Wook Kim
Yuhao Zhou
Jonah Philion
Antonio Torralba
Sanja Fidler

Learning to Simulate Dynamic Environments with GameGAN

Seung Wook Kim[1,2,3*] Yuhao Zhou[2†] Jonah Philion[1,2,3] Antonio Torralba[4] Sanja Fidler[1,2,3*]
[1]NVIDIA [2]University of Toronto [3]Vector Institute [4]MIT
{seungwookk, jphilion, sfidler}@nvidia.com henryzhou@cs.toronto.edu torralba@mit.edu
https://nv-tlabs.github.io/gameGAN

Abstract

Simulation is a crucial component of any robotic system. In order to simulate correctly, we need to write complex rules of the environment: how dynamic agents behave, and how the actions of each of the agents affect the behavior of others. In this paper, we aim to learn a simulator by simply watching an agent interact with an environment. We focus on graphics games as a proxy of the real environment. We introduce GameGAN, a generative model that learns to visually imitate a desired game by ingesting screenplay and keyboard actions during training. Given a key pressed by the agent, GameGAN "renders" the next screen using a carefully designed generative adversarial network. Our approach offers key advantages over existing work: we design a memory module that builds an internal map of the environment, allowing for the agent to return to previously visited locations with high visual consistency. In addition, GameGAN is able to disentangle static and dynamic components within an image making the behavior of the model more interpretable, and relevant for downstream tasks that require explicit reasoning over dynamic elements. This enables many interesting applications such as swapping different components of the game to build new games that do not exist.

1. Introduction

Before deployment to the real world, an artificial agent needs to undergo extensive testing in challenging simulated environments. Designing good simulators is thus extremely important. This is traditionally done by writing procedural models to generate valid and diverse scenes, and complex behavior trees that specify how each actor in the scene behaves and reacts to actions made by other actors, including the ego agent. However, writing simulators that encompass a large number of diverse scenarios is extremely time consuming and requires highly skilled graphics experts. Learning to simulate by simply observing the dynamics of the real world is the most scaleable way going forward.

A plethora of existing work aims at learning behavior models [2, 33, 19, 4]. However, these typically assume a significant amount of supervision such as access to agents' ground-truth trajectories. We aim to learn a simulator by simply watching an agent interact with an environment. To simplify the problem, we frame this as a 2D image generation problem. Given sequences of observed image frames and the corresponding actions the agent took, we wish to emulate image creation as if "rendered" from a real dynamic environment that is reacting to the agent's actions.

Towards this goal, we focus on graphics games, which represent simpler and more controlled environments, as a proxy of the real environment. Our goal is to replace the graphics engine at test time, by visually imitating the game using a learned model. This is a challenging problem: different games have different number of components as well as different physical dynamics. Furthermore, many games require long-term consistency in the environment. For example, imagine a game where an agent navigates through a

Figure 1: If you look at the person on the top-left picture, you might think she is playing Pacman of Toru Iwatani, but she is not! She is actually playing with a GAN generated version of Pacman. In this paper, we introduce GameGAN that learns to reproduce games by just observing lots of playing rounds. Moreover, our model can disentangle background from dynamic objects, allowing us to create new games by swapping components as shown in the center and right images of the bottom row.

*Correspondence to {seungwookk, sfidler}@nvidia.com
†YZ worked on this project during his internship at NVIDIA.

https://research.nvidia.com/labs/toronto-ai/gameGAN/

PHYSICS in COMPUTER ANIMATIONS and GAMES

https://www.latent-technology.com/technology

"Instead of loading the characters with thousands of hand-crafted animations, we allow them to decide their movements in real time."

Our tech:

GENERATIVE
PHYSICS
ANIMATION

- Characters are **physics-based**.
- Movements are generated by **neural networks** in real-time.
- We train them using **deep reinforcement learning** and **generative modelling** techniques.

On starting Latent Technology

Updated: Feb 7, 2023

I'm very excited to announce that we closed a $2.1M pre-seed round led by Root Ventures and Spark Capital, with participation from gaming fund Bitkraft. We are excited to grow the team and release the first version of our product, which will allow game developers to build worlds with unprecedented interactivity and immersiveness.

This milestone calls for a reflection on our company and our vision.