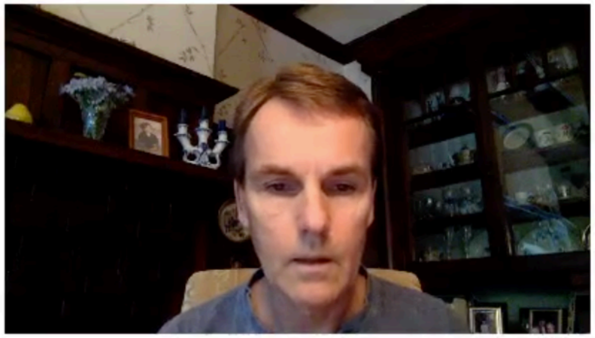
Modeling Contact and Collisions for Robotic Assembly

2020 American Modelica Conference

Scott A. Bortoff

21-September-2020

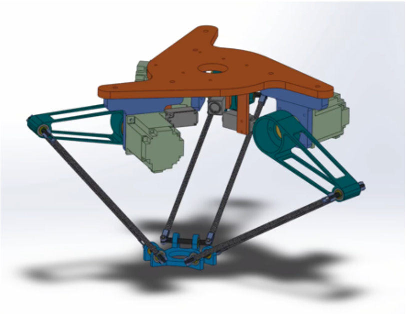
MITSUBISHI ELECTRIC RESEARCH LABORATORIES (MERL)
Cambridge, Massachusetts, USA
http://www.merl.com



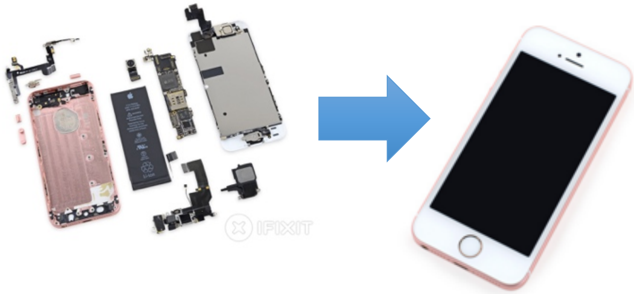
Motivation: Robust Robotic Assembly, Mechanical Contact Problems

- Propose implicit, event-driven, penalty-based model of rigid body contact / collision
- Hybrid: Continuous-time DAE model + finite state machine
- Uses: Not just simulation. Mathematical analysis, model-based control design
- Implementation: Native Modelica, variable-step, stiff (implicit) solver

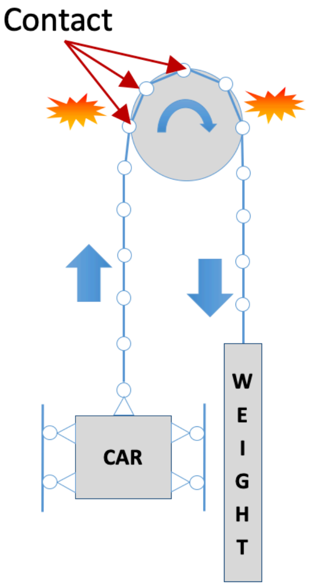
MERL Delta Robot



Mechanical Assembly

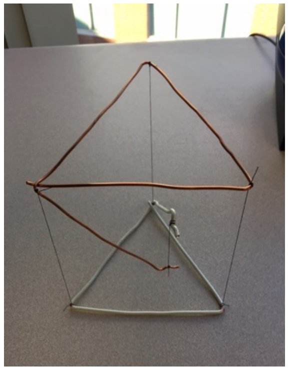


Elevator Motion

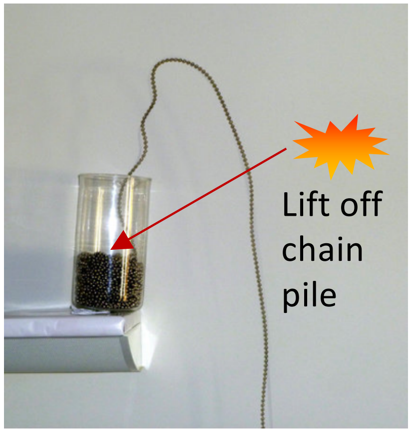


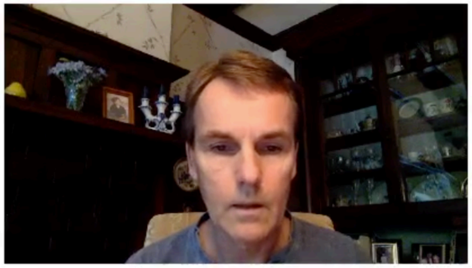
Ball Maze

Tensegrity



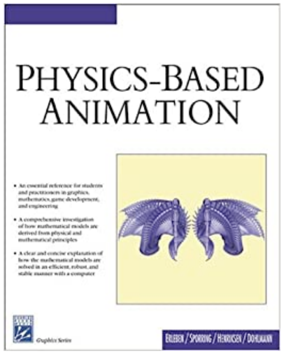
Chain Fountain

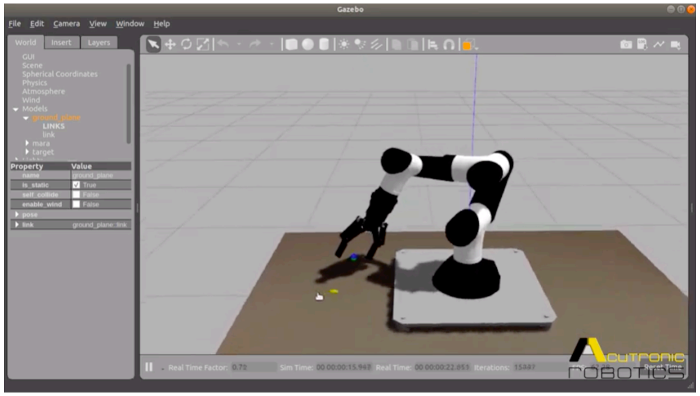


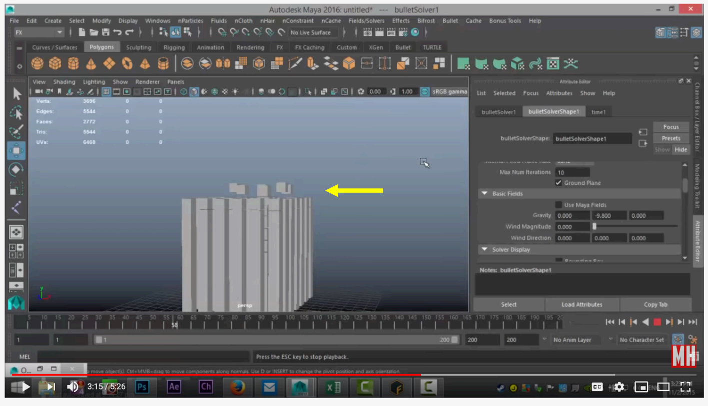


Collisions and Contact: Lots of Previous Work

- Physics-Based Animation: Bullet, PhyX, Gazebo (with different engines such as DART), etc.
- State-of-the-art: Represent as Nonlinear or Linear Complementary Problem (LCP), solve
  - Solve a QP problem at each discrete time step
  - Fixed time step, usually first-order (Euler) symplectic (to approx. conserve energy) integrator
  - Simulation is the only purpose
- Documented limitations and problems…
  - Energy Conservation
  - Difficulty with widely ranging object sizes
  - Collision detection requires non-zero margin
  - Tolerances need tuning
  - Not intended for numerical analysis – only to simulate
  - https://youtu.be/k6nKC_DCh3o?t=188







Using Modelica for advanced Multi-Body modelling in 3D graphical robotic simulators

Gianluca Bardaro[1] Luca Bascetta[1] Francesco Casella[1] Matteo Matteucci[1]

[1]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy,
{luca.bascetta,gianluca.bardaro,francesco.casella,matteo.matteucci}@polimi.it

Abstract

This paper describes a framework to extend the 3D robotic simulation environment Gazebo, and similar ones, with enhanced, tailor-made, multi-body dynamics specified in the Modelica language. The body-to-body interaction models are written in Modelica, but they use the sophisticated collision detection capabilities of the Gazebo engine. This contribution is a first step toward the simulation of complex robotics systems integrating detailed physics modelling and realistic sensors such as lidar and cameras. A proof-of-concept implementation is described in the paper integrating Gazebo collider and the Modelica Multi-Body library, and the results obtained when simulating the interaction of an elastic sphere with a rigid plane are shown.

Keywords: Multi-Body Dynamics, 3D Robotic Simulators, Autonomous Robotics, Autonomous Vehicles.

1 Introduction

The popularity of research on autonomous mobile robots, including autonomous vehicles and mobile manipulators, has been recently increasing due to the huge number of potential applications, ranging from self-driving cars and robots for logistics, to planetary explorations, search and rescue missions, surveillance, humanitarian de-mining, as well as precision agriculture activities such as pruning vines and fruit trees (Paden et al., 2016; Roa et al., 2015; Ko et al., 2015; Chitta et al., 2012).

The design and development of such systems, whose main functionalities are perception, planning, and control, is a multidisciplinary and complex work that has to be supported by virtual prototypes, allowing for a preliminary design and testing of the corresponding algorithms in safe operating conditions. However, due to the huge differences among the three mentioned skills a mobile robot should own, the virtual prototype has to satisfy various requirements. Considering, for example, the development of perception algorithms, the most important characteristics of the virtual prototype are a realistic description, mainly from a geometrical and graphical point of view, of the scene, and the availability of realistic models for the most common commercial sensors, i.e., laser range finders and cameras. On the other hand, testing a control algorithm, e.g., an Advanced Driver Assistance System in a critical situation, requires an accurate physical modelling of the vehicle, including all (and sometimes even only) the phenomena the designer knows to be relevant in the specific application, e.g., cornering stiffness for lateral dynamics control.

Nowadays there are many different, open source and commercial, modelling and simulation environments that are suitable to model vehicles and mobile robots.

A first family is represented by 3D robot simulators, like for example Gazebo[1], V-Rep[2], Webots[3], Morse[4], that are widespread in the robotics community. These simulators allow for an easy development of complex natural/artificial simulation environments, they are already equipped with models of perception devices, and they can be easily integrated with standard robot control middlewares like ROS[5]. For these reasons, they are particularly suitable for the development and testing of planning and perception algorithms, and for the validation of the whole control software before moving to field tests (Bardaro et al., 2014).

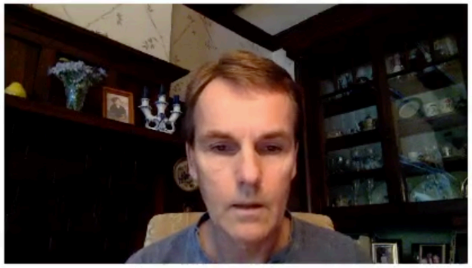
The physical simulation implemented in these tools is targeted at real-time execution and ease of virtual prototype set-up; this is obtained by providing the 3D kinematic models for rotational and translational joints to assemble robots and vehicles, and collision detection primitives with simplified translational and rotational friction models. These building blocks are implemented with low level C++ libraries, such as ODE (Drumwright et al., 2010) or MuJoCo (Erez et al., 2015), and the experimenter is expected to use them in a black box fashion with little, if any, way to alter their physical behaviour. Indeed, the differential equations characterizing the physical behaviour of each building block are hidden in the code, often undocumented, and with no direct tool for altering their behaviour. This makes current 3D robotics physical simulation fidelity and accuracy somehow limited, and requires the coding of external plug-ins, e.g., using C++ custom code, every time the phenomenon we are interested in replicating is more complex that the one which can be obtained assembling the available building blocks.

On the other side of the spectrum, a second family of sim-

---

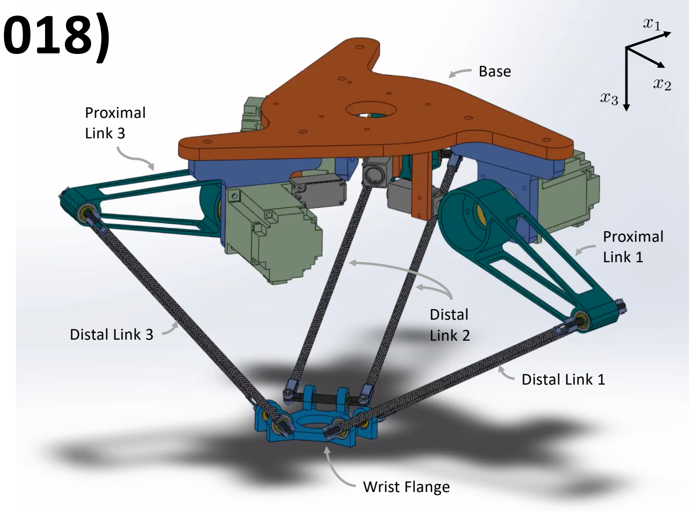
[1] `http://gazebosim.org`
[2] `http://www.coppeliarobotics.com`
[3] `http://www.cyberbotics.com`
[4] `http://www.openrobots.org/wiki/morse`
[5] `http://www.ros.org`

DOI
10.3384/ecp17132887



MITSUBISHI ELECTRIC
Changes for the Better

Delta Robot Differential-Algebraic Model (2018)



- **Forward Kinematics of each arm...**

$$\psi(q_i) = \begin{bmatrix} l_2 \sin(q_{i2}) \sin(q_{i3}) \\ l_1 \cos(q_{i1}) + l_2 \cos(q_{i2}) \\ l_1 \sin(q_{i1}) + l_2 \sin(q_{i2}) \cos(q_{i3}) \end{bmatrix}$$

Arm 1: $q_1 = [q_{11},\, q_{12},\, q_{13}]^T$
Arm 2: $q_2 = [q_{21},\, q_{22},\, q_{23}]^T$
Arm 3: $q_3 = [q_{31},\, q_{32},\, q_{33}]^T$

- **Constraint at wrist flange (6 equations)...**

$$h(q) = \begin{bmatrix} \psi(q_1) - R_z(2\pi/3) \cdot \psi(q_2) \\ \psi(q_1) - R_z(-2\pi/3) \cdot \psi(q_3) \end{bmatrix} \qquad h(q) : \mathcal{R}^9 \to \mathcal{R}^6$$

- **Robot Dynamics...Index 3 DAE** ☹

$$H = \frac{\partial h}{\partial q}$$

$$\begin{aligned} \dot q &= v \\ M(q)\dot v + C(q,v) + G(q) &= \lambda^T H(q) + Bu \\ \boxed{h(q) = 0} \end{aligned}$$

Replace

- **Robot Dynamics ... Index 1 DAE** ☺

$$\begin{aligned} \dot q &= v \\ M(q)\dot v + C(q,v) + G(q) &= \lambda^T H(q) + Bu \end{aligned}$$

$$\boxed{h''(q,v,\lambda) + \alpha_1 h'(q,v) + \alpha_0 h(q) = 0}$$

- **24 equations, 24 states** $(q,\, v,\, \lambda)$

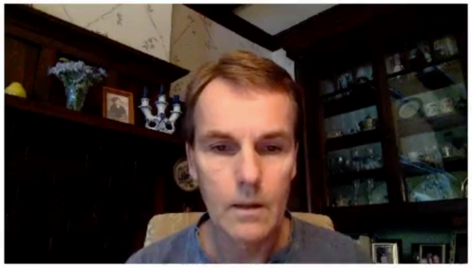
$$\begin{aligned} M(q) &= \mathrm{diag}\,(m(q_1), m(q_2), m(q_3)) \in \mathbb{R}^{9\times 9}, \\ C(q,v) &= \mathrm{diag}(c(q_1,v_1), c(q_2,v_2), c(q_3,v_3)) \in \mathbb{R}^9, \\ G(q) &= \mathrm{diag}(g(q_1), g(q_2), g(q_3)) \in \mathbb{R}^9, \\ B &= \mathrm{diag}\,(b,b,b) \in \mathbb{R}^{9\times 3}. \end{aligned}$$

- **Can be expressed**

$$\begin{aligned} \dot\eta &= f(\eta,\xi,u) & \eta \in \mathcal{R}^6 \\ \dot\xi &= A\xi, & \xi \in \mathcal{R}^{12} \end{aligned}$$

- **Robot dynamics are zero dynamics**

$$\dot\eta = f(\eta, 0, u)$$



Contact and Collision Model Idea

- Dynamics

$\lambda = 0$ Free
$\lambda > 0$ Contact

$$\dot{q} = v$$

$$M(q)\dot{v} + C(q,v) + G(q) = \lambda^T H(q) + Bu$$

$$h''(q,v,\lambda) + \alpha_1 h'(q,v) + \alpha_0 h(q) = 0$$

This can model elastic collision

- Elastic collision... $\ddot{h} + \alpha_1 \dot{h} + \alpha_0 h = 0$

Tune α_1 , α_0

- Properties...
  1. $h \to 0$ after collision transient – enforces penetration constraint
  2. Numerically stiff for large α_0

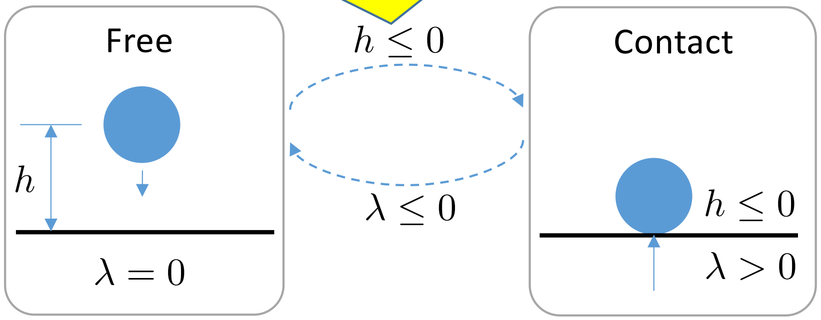
So add nonlinear feedback

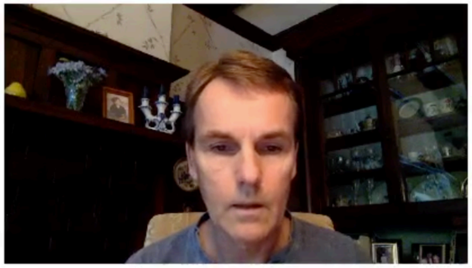
- Replace stabilized constraint with ...

$$\ddot{h} + \alpha_1 \dot{h} + \alpha_0 h + \alpha_3 h^3 = 0$$

- This is stiff during collision, but soft in steady-state
- Replace switching condition with FSM
  - Because $h \to 0$

Not quite right ... because h → 0





Contact and Collision Model Idea

- Dynamics

$$\dot{q} = v$$

$$M(q)\dot{v} + C(q,v) + G(q) = \lambda^T H(q) + Bu$$

$$h''(q,v,\lambda) + \alpha_1 h'(q,v) + \alpha_0 h(q) = 0$$

$\lambda = 0$ Free
$\lambda > 0$ Contact

This can model elastic collision

- Elastic collision… $\ddot{h} + \alpha_1 \dot{h} + \alpha_0 h = 0$

Tune α_1 , α_0

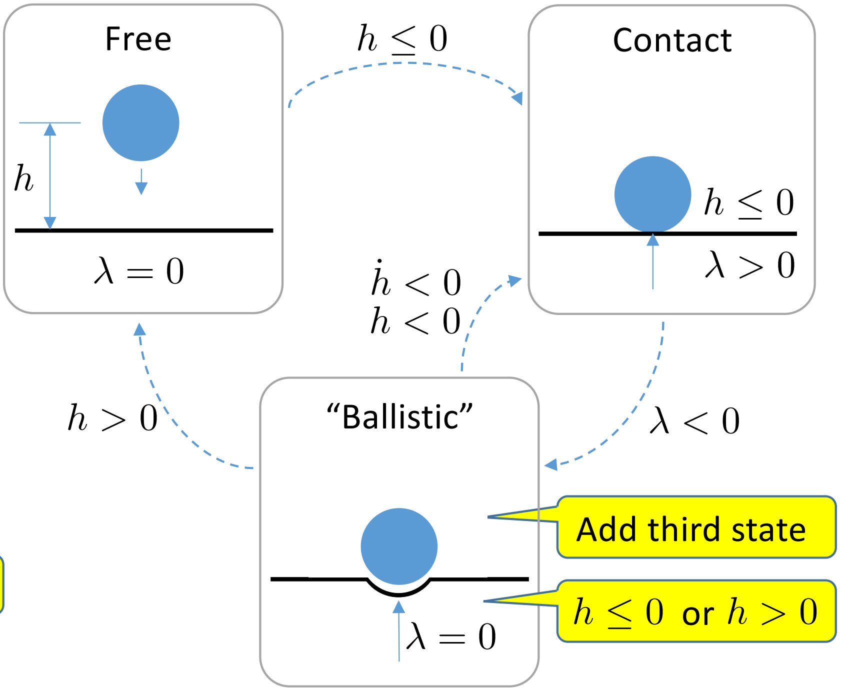
- Properties…
  1. $h \to 0$ after collision transient – enforces penetration constraint
  2. Numerically stiff for large α_0

So add nonlinear feedback

- Replace stabilized constraint with …

$$\ddot{h} + \alpha_1 \dot{h} + \alpha_0 h + \alpha_3 h^3 = 0$$

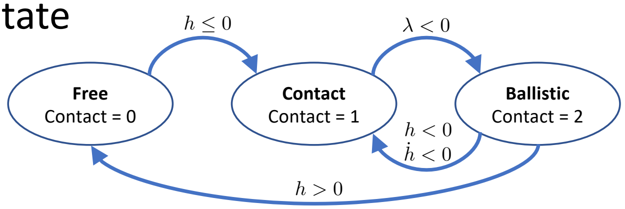
- This is stiff during collision, but soft in steady-state
- Replace switching condition with FSM
  - Because $h \to 0$

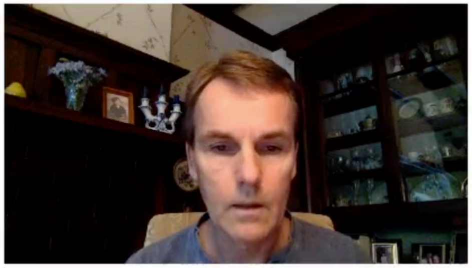


Add third state

$h \leq 0$ or $h > 0$

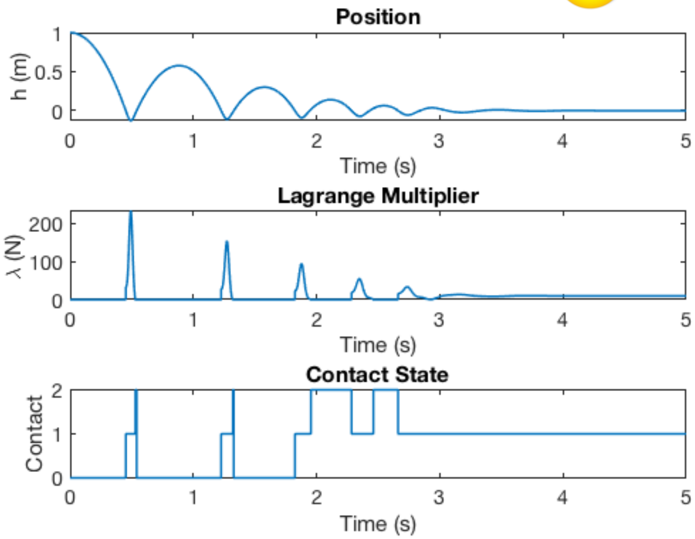
Finite State Machine

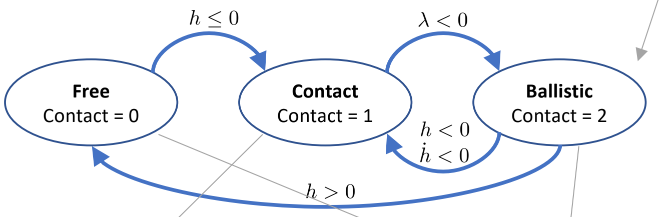




Bouncing Ball Example

Example 1: Positive Damping





$h \leq 0$

$\lambda < 0$

Free
Contact = 0

Contact
Contact = 1

Ballistic
Contact = 2

$h < 0$
$\dot{h} < 0$

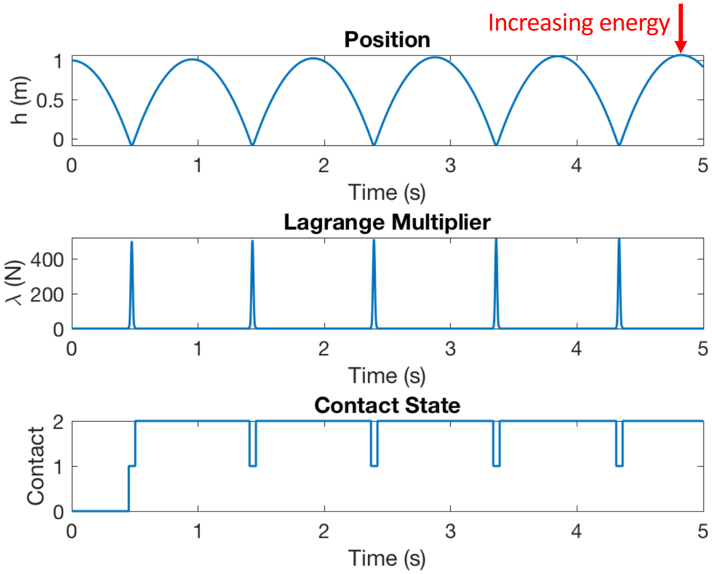
$h > 0$

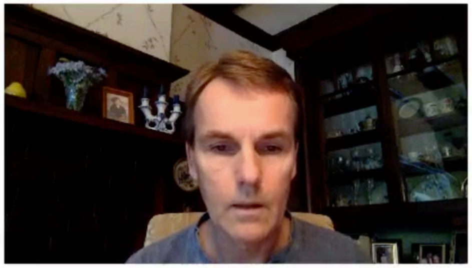
$$\dot{q} = v$$
$$m\dot{v} = -mg + \lambda$$
$$0 = \ddot{h} + \alpha_1 \dot{h} + \alpha_0 h + \alpha_3 h^3$$

$$\dot{q} = v$$
$$m\dot{v} = -mg$$
$$\lambda = 0$$

```
model myBouncingBall

Real q(start=1.0), v(start=0.0), f;
Real h, hDot, hDotDot, lambda(start=0);
discrete Integer contact(start=0);
Boolean b1, b2, b3, b4;
parameter Real g=9.81, m=1.0;
parameter Real a0=100, a1=20, a2=1e6;
parameter Boolean linFlag = false;

algorithm
b1 := h <= 0;
b2 := lambda < 0.0;
b3 := h > 0.0;
b4 := h <= 0 and hDot < 0;
if edge(b1) and contact == 0 then
  contact := 1;
end if;
if contact == 1 and edge(b2) then
  contact := 2;
end if;
if contact == 2 and edge(b3) then
  contact := 0;
end if;
if contact == 2 and edge(b4) then
  contact := 1;
end if;

equation
if contact == 1 or linFlag then
  0 = hDotDot + a1*hDot + a0*h + a2*h^3;
else
  lambda = 0.0;
end if;

f = if linFlag then lambda
     else contact*lambda;

der(q) = v;
m * der(v) = -m * g + f;

h = q;
hDot = der(h);
hDotDot = der(hDot);

end myBouncingBall;
```

Example 2: No Damping


Increasing energy



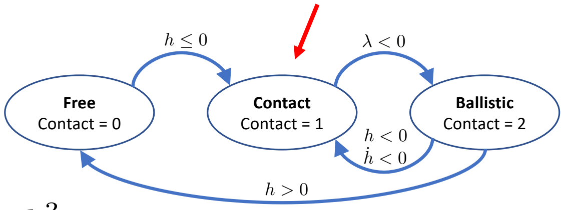
What’s Going On? A Failure to Conserve Energy. D’ Oh!

- Contact State

$$\dot{q} = v$$

$$m\dot{v} = -mg + \lambda$$

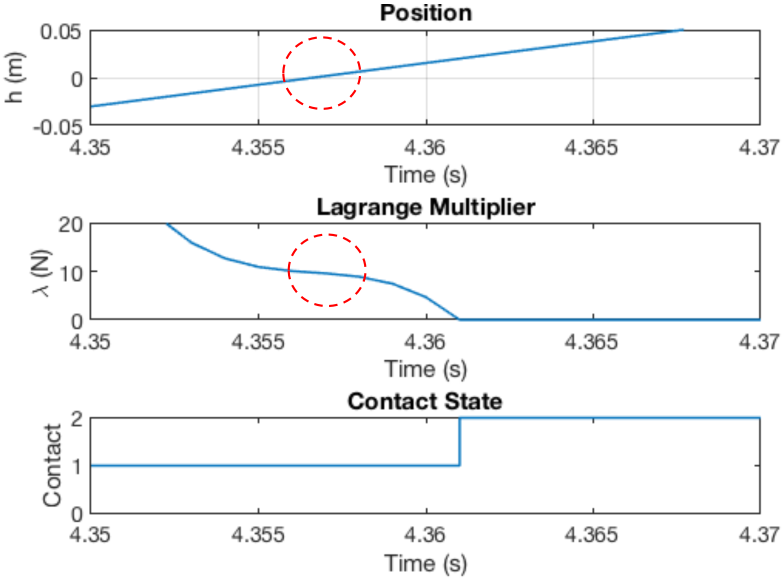
$$0 = \ddot{h} + \alpha_1\dot{h} + \alpha_0 h + \alpha_3 h^3$$

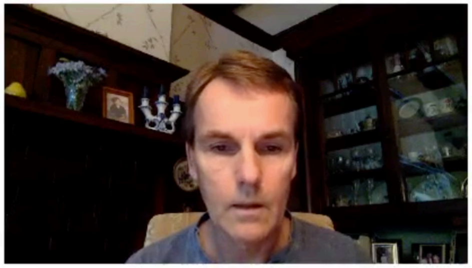


- Solve for Lagrange Multiplier...

$$\lambda(t) = (\alpha_0 q(t) + \alpha_2 q^3(t) + g)m.$$

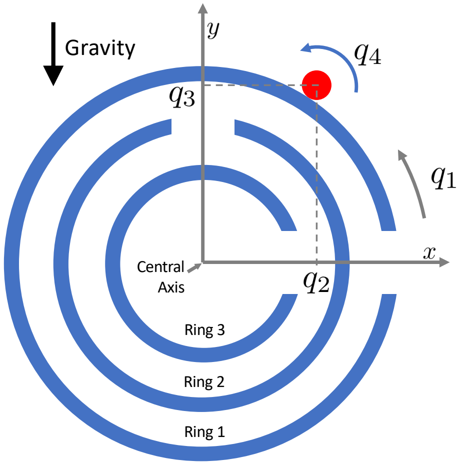
- If this were a “spring force,” the g would not appear
- This force adds energy to the ball when $\lambda > 0$ and $h > 0$
- This force is responsible for
  - $h \to 0$ (no penetration in steady state) ... Good
  - Failure to conserve energy ... Bad
- But, its not *that* bad
  - Energy added only when $\lambda > 0$ and $\dot{h} \neq 0$
  - In typical applications, $\alpha_1 > 0$ (and $\alpha_1 = 0$ isn’t stable)
  - Many “physics engines” also fail to conserve energy!





Ball Maze Game

- Strategy: Tip maze to constant angle, rotate maze about axis
- Open-loop unstable against rings. Stabilize with feedback.



Dynamics

$$M\ddot{q} + G = H^T(q,i)\lambda + Bu$$

$$\ddot{h}(q,\dot{q},\lambda,i) + \alpha_1 \dot{h}(q,\dot{q},i) + \alpha_0 h(q,i) = 0$$

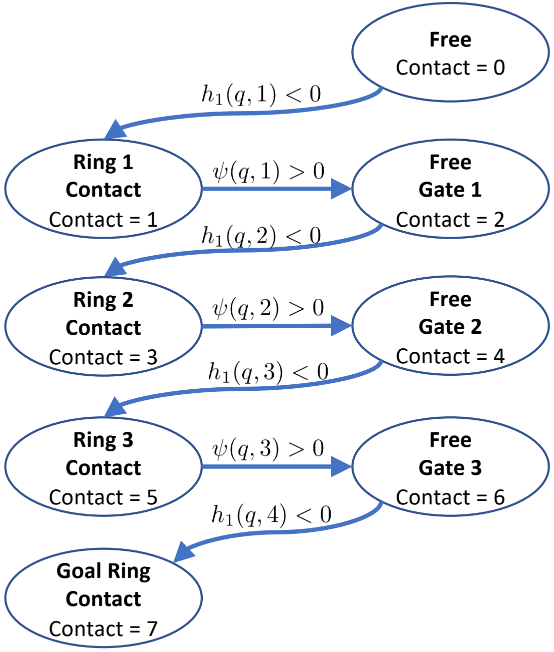
$$\begin{aligned} M &= \mathrm{diag}(J_m, m_b, m_b, J_b)\,, \\ G &= [0\;0\;gm_b\;0]^T, \\ B &= [1\;0\;0\;0]^T, \\ H(q,i) &= \frac{\partial h(q,i)}{\partial q}\,, \end{aligned}$$

Constraints

$$h_1(q,i) = q_1^2 + q_2^2 + (r_{mi} + r_b)^2 \qquad \longleftarrow \textbf{Ring}$$

$$\begin{aligned} h_2(q,i) = {} & r_b(q_4 - q_{40}) + r_{mi}(q_1 - q_{10}) \qquad \longleftarrow \textbf{Rolling} \\ & + r_{mi}\left(\mathrm{atan}(q_2/q_3) - \mathrm{atan}(q_{20}/q_{30})\right) \end{aligned}$$

FSM



Free
Contact = 0

$h_1(q,1) < 0$

Ring 1 Contact
Contact = 1

$\psi(q,1) > 0$

Free Gate 1
Contact = 2

$h_1(q,2) < 0$

Ring 2 Contact
Contact = 3

$\psi(q,2) > 0$

Free Gate 2
Contact = 4

$h_1(q,3) < 0$

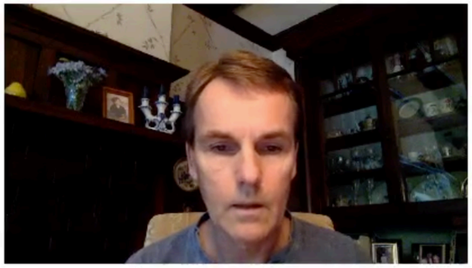
Ring 3 Contact
Contact = 5

$\psi(q,3) > 0$

Free Gate 3
Contact = 6

$h_1(q,4) < 0$

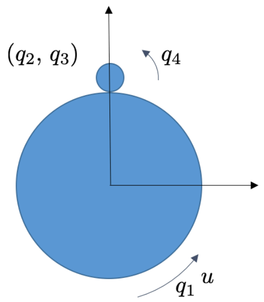
Goal Ring Contact
Contact = 7



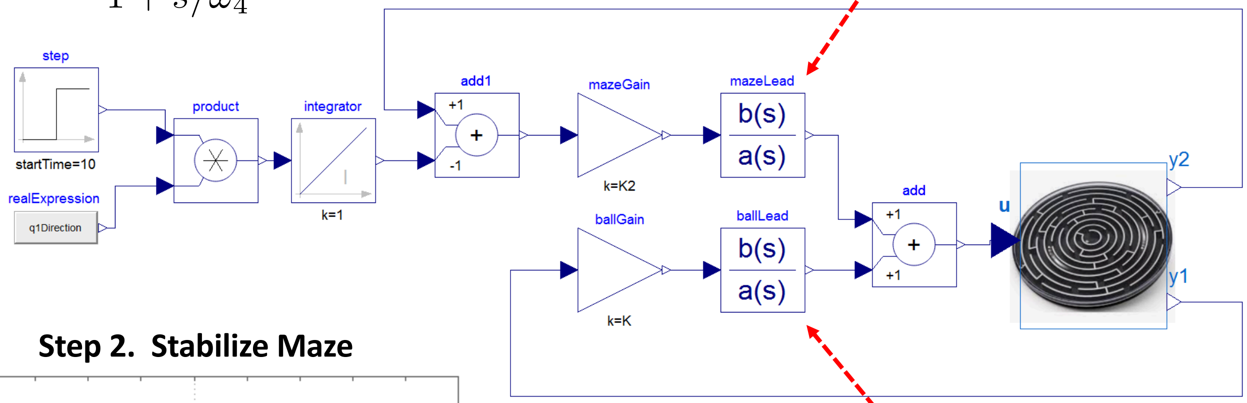
Ball Maze Lead-Compensator Control Design

Note Fundamental Limits: $\left\{ \begin{array}{l} \text{Lower bound on } k_b \\ \text{Upper bound on } k_m \end{array} \right.$

1. Stabilize ball with lead compensator $k_b \dfrac{1+s/\omega_1}{1+s/\omega_2}$
2. Stabilize maze with lead compensator $k_m \dfrac{1+s/\omega_3}{1+s/\omega_4}$

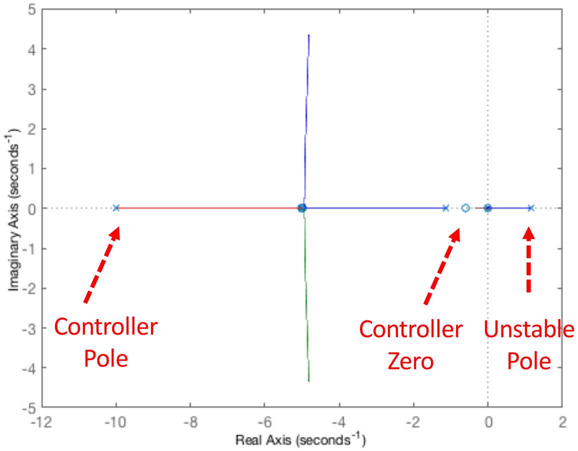


2) Maze lead compensator



1) Ball lead compensator

Step 1. Stabilize Ball

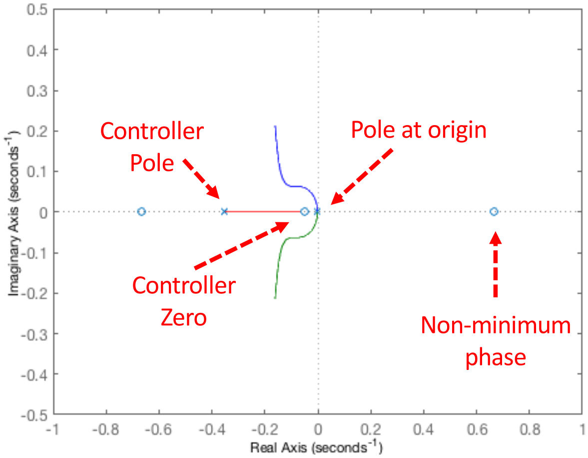


Controller Pole

Controller Zero

Unstable Pole

Step 2. Stabilize Maze

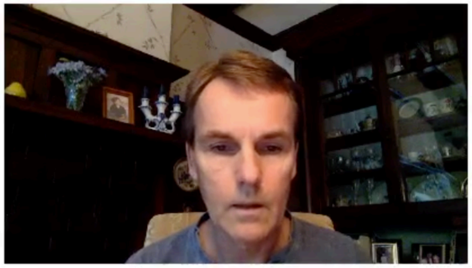


Controller Pole

Pole at origin

Controller Zero

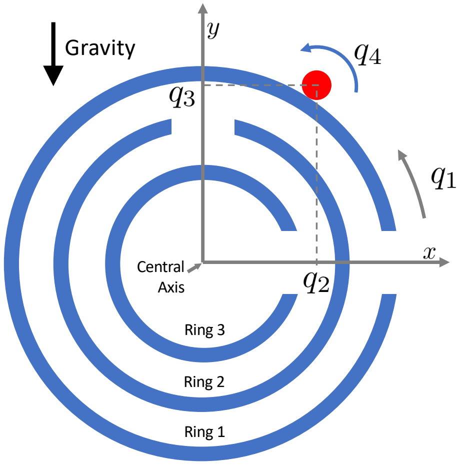
Non-minimum phase

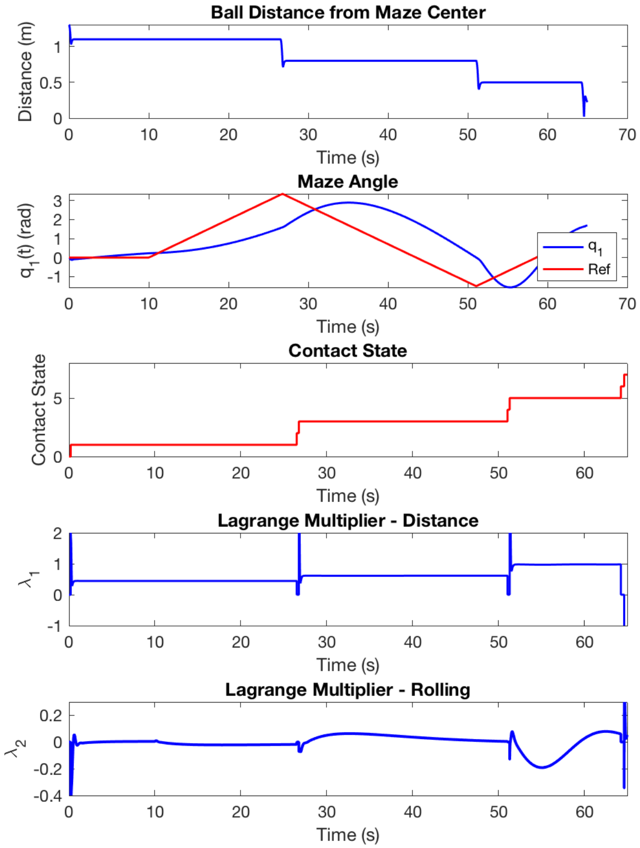


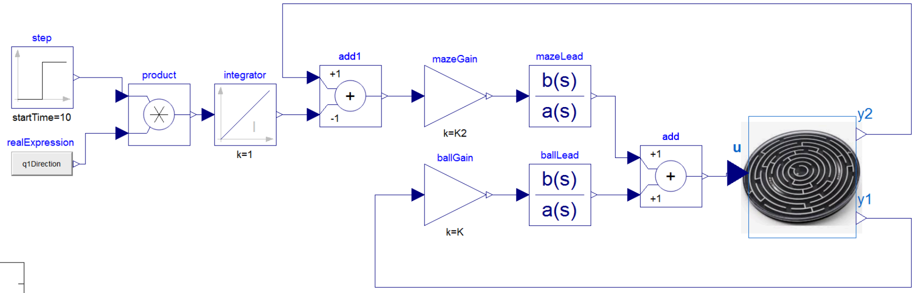
MITSUBISHI ELECTRIC
Changes for the Better

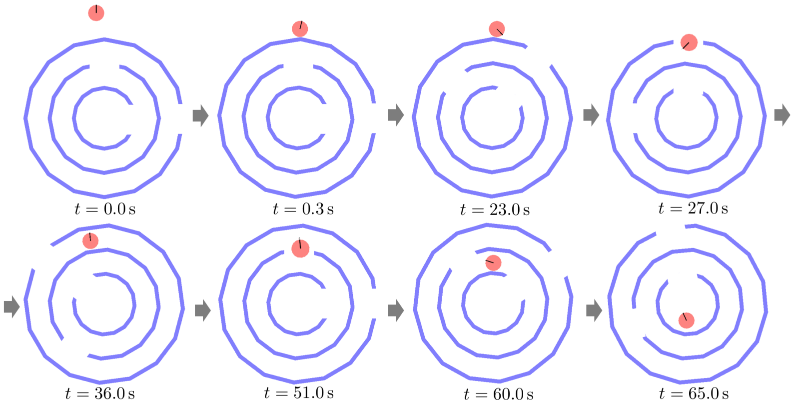
Ball – Maze Simulation Results

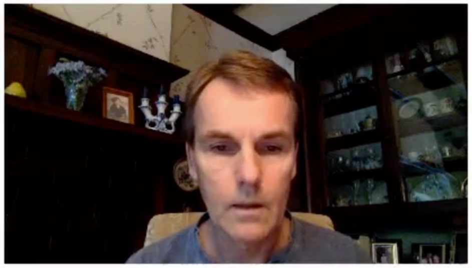
- Feedback control to stabilize ball
- Rotate maze to bring gate sequence under ball



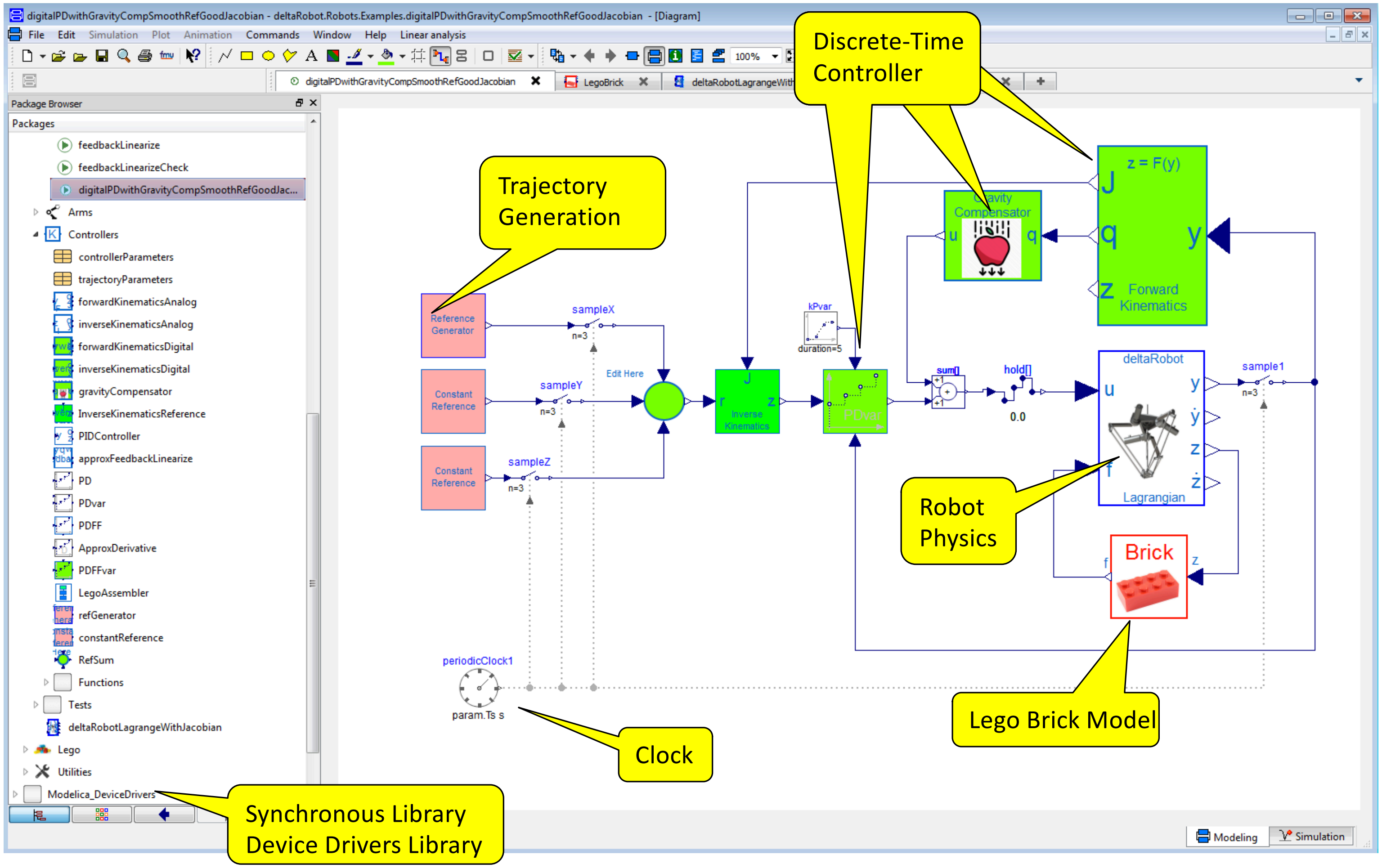








Soft-Touch Robotic Control



Discrete-Time Controller

Trajectory Generation

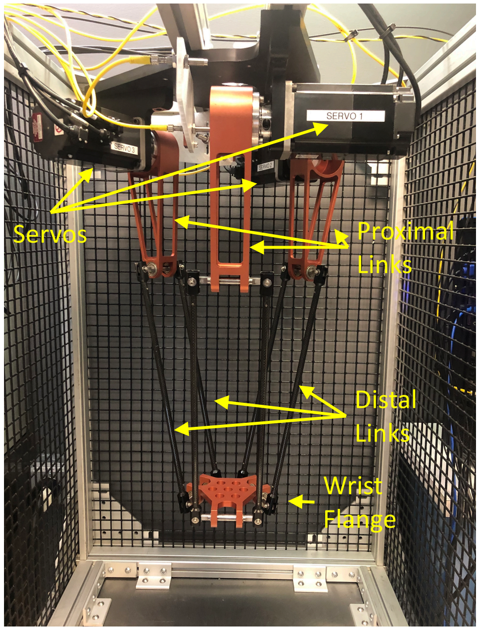
Robot Physics

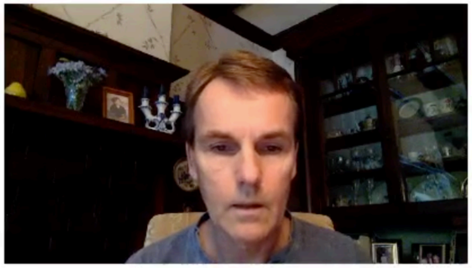
Lego Brick Model

Clock

Synchronous Library
Device Drivers Library

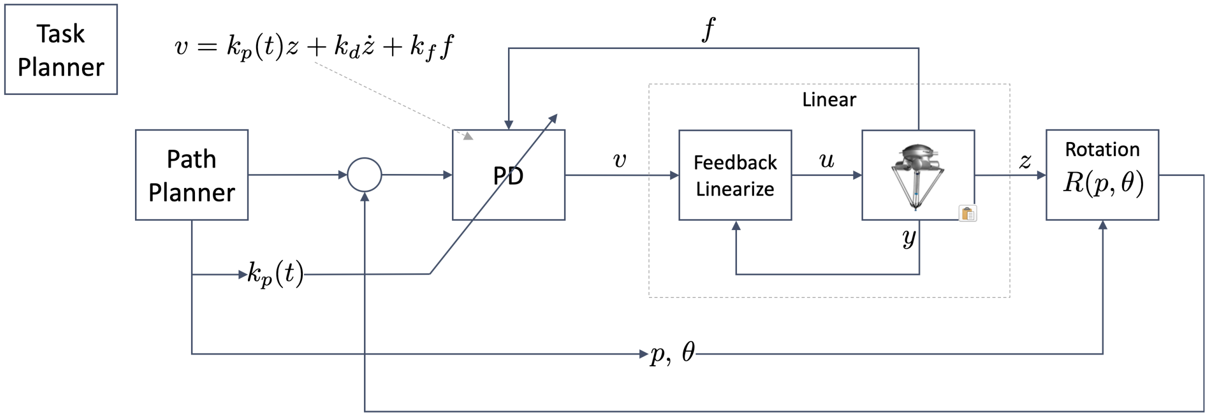
MERL's Kamaji

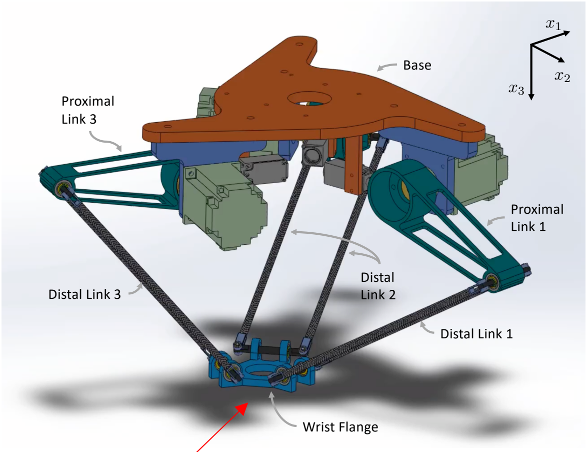


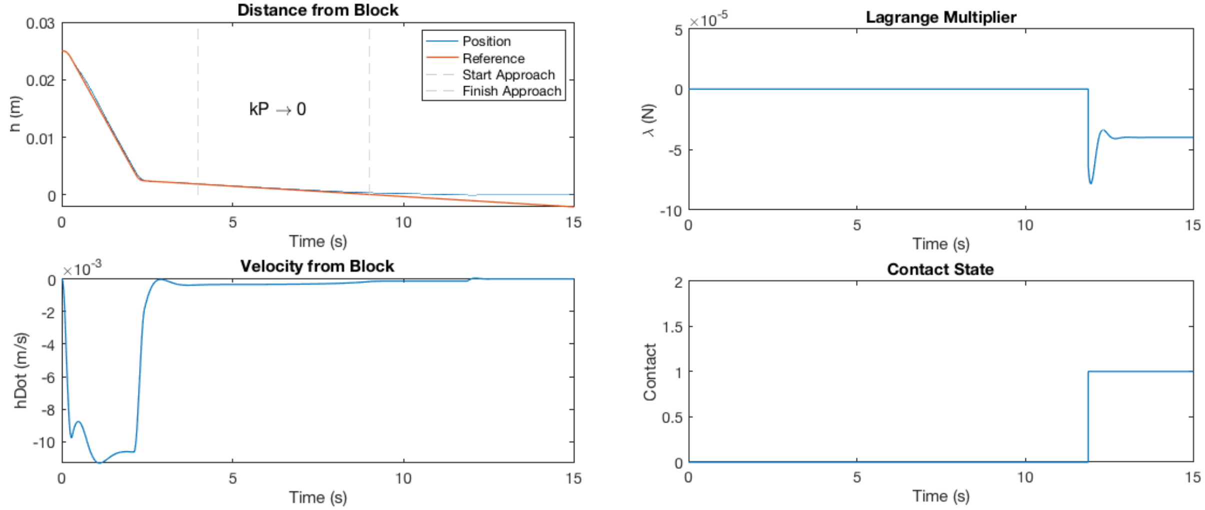


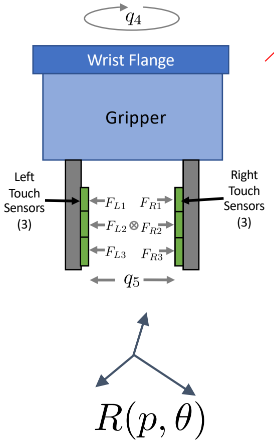
Changes for the Better

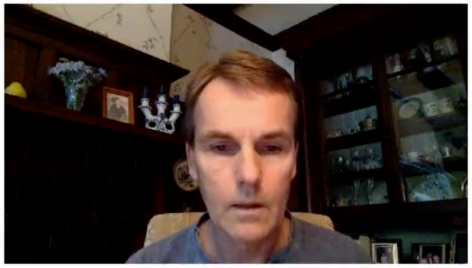
Simulation Example – Soft-Contact









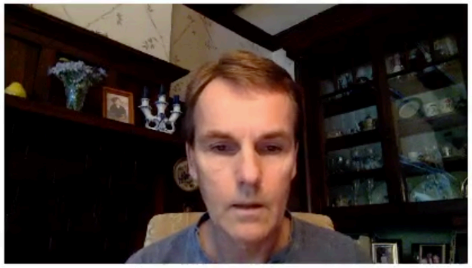


Summary & Conclusions

- Hybrid DAE – FSM model of rigid body contact and collision
- Consistent mathematics – modeling with separation of concerns e.g. solver
- Native Modelica enables analysis beyond simulation. But can FSM be improved?
- Useful for some types of contact
    - Low dimensions
    - Positive damping, but not too stiff
- Caveat: Event based method – critically dependent on event detection. (Can fail.)
- Code for Ball Maze, bouncing ball available. Email: bortoff@merl.com

MITSUBISHI
ELECTRIC
Changes for the Better

Thank – you!

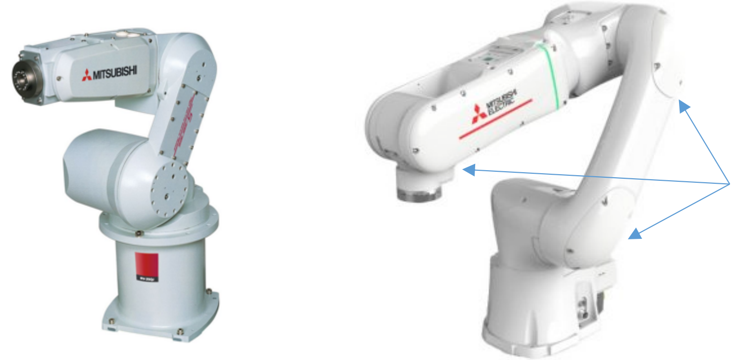


MITSUBISHI
ELECTRIC

Changes for the Better

Serial-Link Vs. Delta Robots – Mechanical Design

Not So Good for Assembly (Contact) Applications

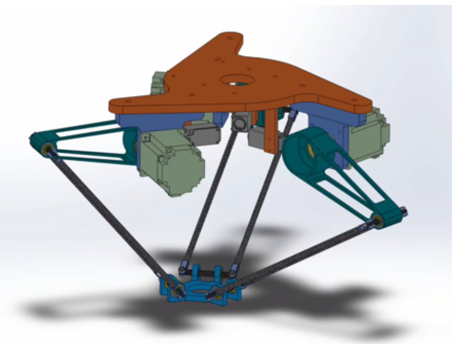


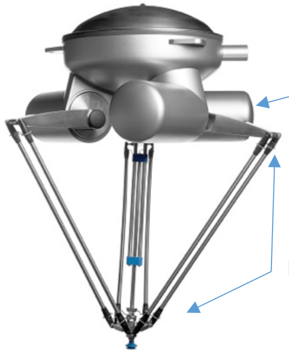
All Joints Actuated

MELCO Co-bot (1Q2020)

Open Chain
High Mass, High Inertia → Slow
High Ratio Gears
Small Servos, High Friction, Not Back-Drivable, although...
High Impedance – Stiff
Low payload to robot mass ratio ~ O(0.1)
Coupled Force / Torque
High Precision Mechanical Position Control
High Joint & Link Bending
Larger work volume
More expensive
Pick and Place, Assembly Applications

Good for Assembly (Contact) Applications





Some Joints Actuated

Unactuated

Codian - MELCO Delta Robot* (3Q2019)

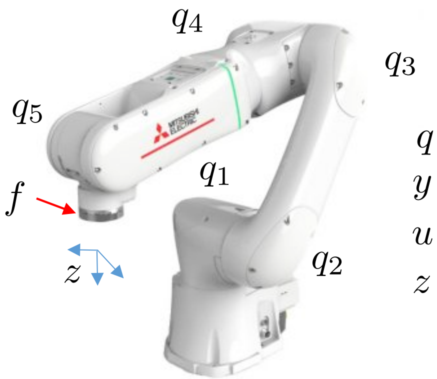
Complex Closed Chain
Low Mass, Inertia → Fast
Direct Drive, although...
Big Servos, Low Friction, Back-Drivable
Low Impedance – Soft
High payload to robot mass ratio ~ O(10)
Decoupled Force / Torque
High Precision Mechanical Position Control
Low Joint & Link Bending
Smaller work volume, although...
Less expensive
Pick and Place Applications

- Volume: 1.5 m x 0.4 m
- Payload < 100kg
- Max. 250 ops / min

* http://www.pandct.com/media/shownews.asp?ID=54275

Serial-Link Vs. Delta Robot – Kinematics, Dynamics

Open Chain – Conventional Calculations



$q \in \mathcal{R}^5$ Joint Angles
$y = q$ Measurements
$u \in \mathcal{R}^5$ Inputs (Co-located)
$z \in \mathcal{R}^3$ End effector

Forward Kinematics – Explicit (analytic, closed-form)

$$z = F(q)$$

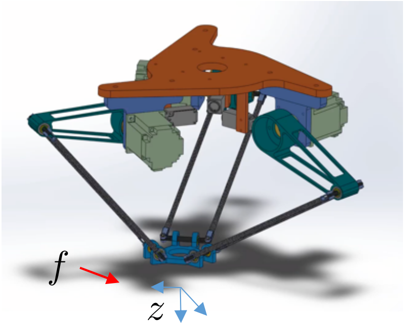
Jacobian – Explicit (analytic closed form)

$$J(q) = \frac{\partial F(q)}{\partial q} \quad \Rightarrow \quad \tau = J^T(q) f$$

Dynamics – ODE (all analytic, closed-form)

$$M(q)\dot{v} + C(q,v) + D(v) + G(q) = u + \tau$$

Kinematic Loop – Conventional Becomes Difficult
Unless we are unconventional !



$q \in \mathcal{R}^9$ Joint Angles
$y \in \mathcal{R}^3$ Measurements
$u \in \mathcal{R}^3$ Inputs

Forward Kinematics – Implicit (no Closed Form)

$$F(q,z) = 0$$ (6 equations, 6 unknowns, solved numerically)

Conventional Jacobian – Implicit if expressed in y ...

$$J(y) = \frac{\partial f(y)}{\partial y}$$ (computed by Implicit Function Theorem, evaluated numerically. Used for control.)

BUT, Jacobian is <u>explicit</u> if expressed in q

$$J(q) = \frac{\partial f}{\partial q}$$ (maps forces at end effector to *virtual* torques at all 9 joints. Used for simulation.)

So Dynamics – Loops in Loops - are NOT a problem

$$M(y)\ddot{y} + C(y,\dot{y}) + D(y) + G(y) = u + \tau \leftarrow \tau = J^T(q) f$$

Modelica Realization of the DAE Model

Declare 3 arms

$$\dot{q}_i = v_i$$

$$m(q_i)\dot{v}_i + c(q_i, v_i) + g(q_i) = b\tau$$

Declare Lagrange Multiplier

Declare Constraint

Constant Rotation Matrices

$$\dot{q} = v$$

$$M(q)\dot{v} + C(q,v) + D(v) + G(q) = \boxed{H^T(q)\lambda + B(u + \tau_u) + \tau_v}$$

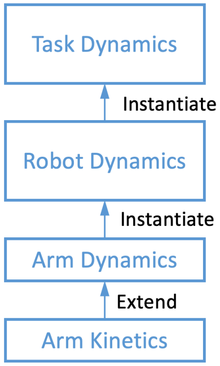
$$\boxed{\ddot{h}(q,v,\dot{v}) + \alpha_1 \dot{h}(q,v) + \alpha_0 h(q) = 0}$$

Servo Angle Measurements $y,\ \dot{y}$

Virtual Jacobian J_v

Wrist Flange location $z,\ \dot{z}$

```
model deltaRobotLagrange

  Arms.deltaRobotArmLagrange arm1, arm2, arm3;
  Real lambda[6];      // Lagrange multiplier
  Real h0[6], h1[6], h2[6];
  Input Real u[3], f[3];      // torque inputs, force inputs
  parameter Real POLE = 5.0;

  constant Real Rot2[3,3] = Utilities.RotZ(2.0*PI/3.0);
  constant Real Rot3[3,3] = Utilities.RotZ(-2.0*PI/3.0);
  constant Real B[3] = {1, 0, 0};  // input torque vector

equation

  arm1.tau =  transpose(arm1.dh) * lambda[1:3] + transpose(arm1.dh) * lambda[4:6] + transpose(dz1) * f + B * u[1];
  arm2.tau = -transpose(Rot2 * arm2.dh) * lambda[1:3] + B * u[2] + transpose(dz2) * f;
  arm3.tau = -transpose(Rot3 * arm3.dh) * lambda[4:6] + B * u[3] + transpose(dz3) * f;

  h0 = cat(1,arm1.psi-Rot2*arm2.psi,arm1.ps -Rot3*arm3.psi);
  h1 = der(h0);
  h2 = der(h1);
  zeros(6) = h2 + 2.0 * POLE * h1 + POLEˆ2 * h0;

  y = cat(1, vector(arm1.q[1]), vector(arm2.q[1]), vector(arm3.q[1]));
  yDot = cat(1, vector(arm1.v[1]), vector(arm2.v[1]), vector(arm3.v[1]));

  dz1 = Controllers.Functions.armJacobian(arm1.q) / 3.0;
  dz2 = R2*Controllers.Functions.armJacobian(arm2.q) / 3.0;
  dz3 = R3*Controllers.Functions.armJacobian(arm3.q) / 3.0;
  Jv = cat(2, dz1, dz2, dz3);

  z = arm1.h;
  zDot = Jv * cat(1, vector(arm1.v), vector(arm2.v), vector(arm3.v));

end deltaRobotLagrange;
```


Task Dynamics

Instantiate

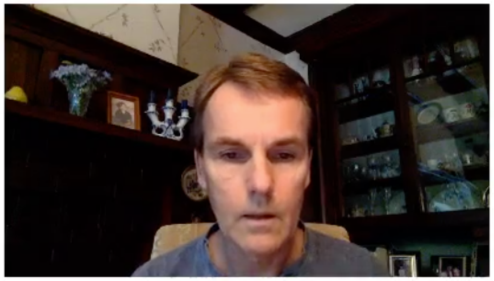
Robot Dynamics

Instantiate

Arm Dynamics

Extend

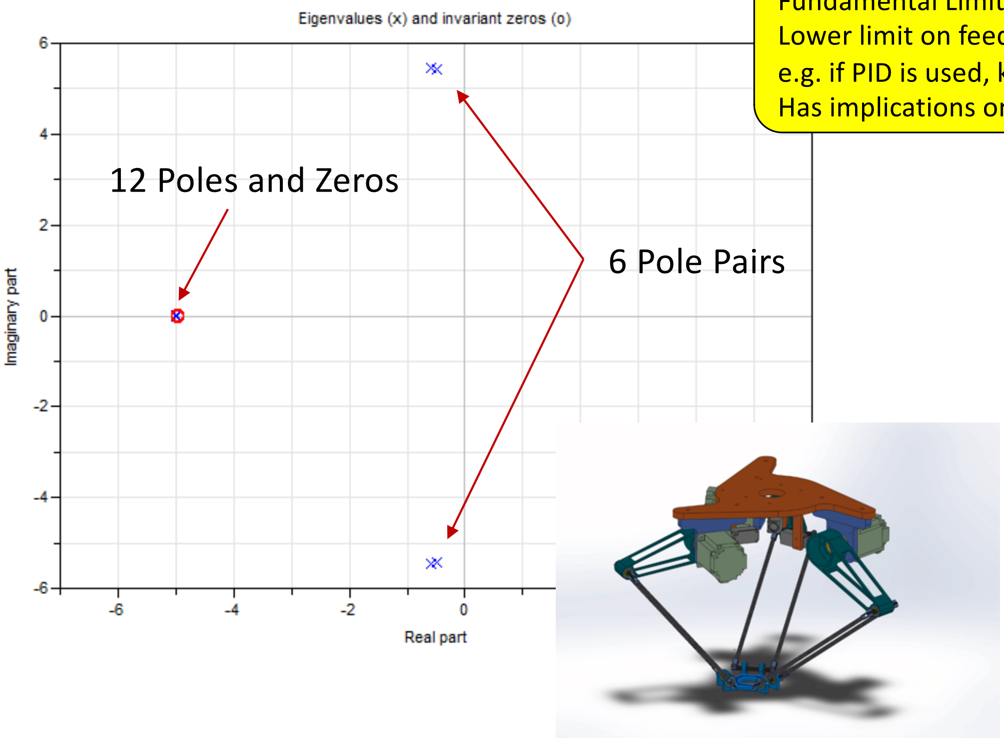
Arm Kinetics



Simple Example of Dynamic Analysis (Robot is Open-Loop Unstable)

Stable Configuration

$$q_{i1} = \frac{\pi}{4}\,\mathrm{rad} = 45^\circ$$



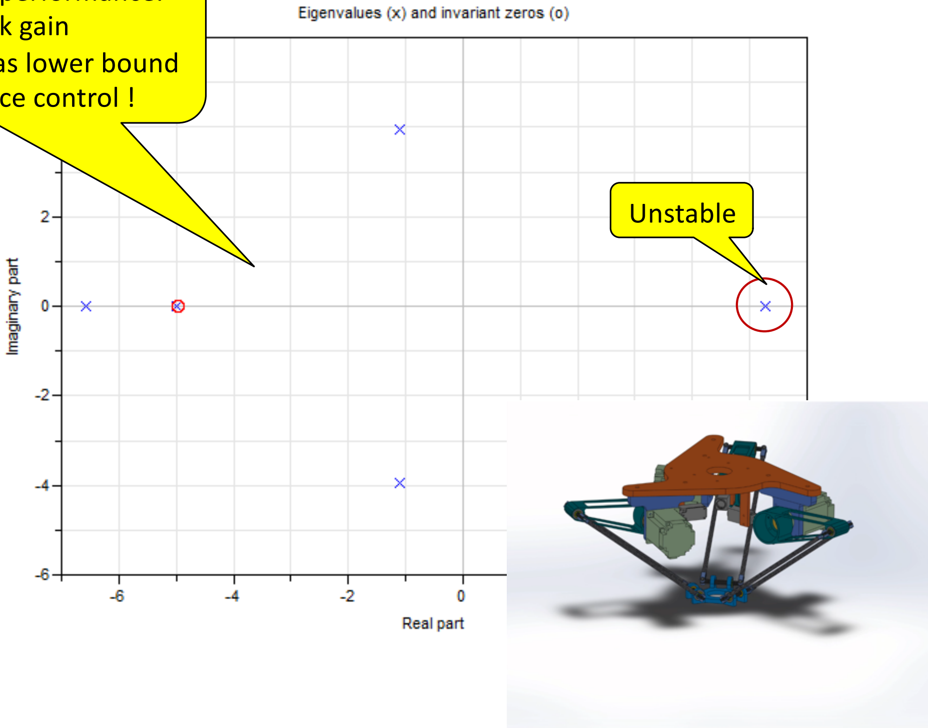
12 Poles and Zeros

6 Pole Pairs

Fundamental Limits of performance.
Lower limit on feedback gain
e.g. if PID is used, kP has lower bound
Has implications on force control !

Unstable Configuration

$$q_{i1} = 0\,\mathrm{rad} = 0^\circ$$



Unstable