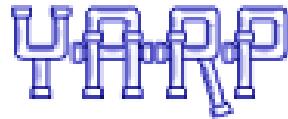


# **Robotran** : A Fast Symbolic, Dynamic Simulator

interfaced with



Timothée Habra (Université catholique de Louvain)  
Houman Dallali (Instituto Italiano di Technologia)



# Introduction

- Robots model complexity (#dof, soft actuation, ...)
- Model-based control (model predictive control, optimal feedback control, ...)

# Introduction

- Robots model complexity (#dof, soft actuation, ...)
- Model-based control (model predictive control, optimal feedback control, ...)

**Need :** Fast and accurate simulator

# Introduction

- Robots model complexity (#dof, soft actuation, ...)
- Model-based control (model predictive control, optimal feedback control, ...)

**Need :** Fast and accurate simulator

## Speed-accuracy tradeoff

“ODE should not be used for quantitative engineering”. (ODE Manual)  
Simbody slower than real-time for CoMan

# Introduction

- Robots model complexity (#dof, soft actuation, ...)
- Model-based control (model predictive control, optimal feedback control, ...)

**Need** : Fast and accurate simulator

Speed-accuracy tradeoff

“ODE should not be used for quantitative engineering”. (ODE Manual)  
Simbody slower than real-time for CoMan

**Solution** : Robotran symbolic simulator

# Outline

- **Robotran Principles**  
Generation of symbolic routines for dynamics
- **Robotics Simulator**  
Available features for robot simulation
- **Robotran-Yarp interface**  
Running a Yarp module with Robotran

# Outline

- **Robotran Principles**  
Generation of symbolic routines for dynamics
- **Robotics Simulator**  
Available features for robot simulation
- **Robotran-Yarp interface**  
Running a Yarp module with Robotran

# **Robotran**

- **Symbolic** software to model and analyze **Multibody Systems**
- Developed since **1990** at ***Université catholique de Louvain*** (UCL)
- Used in a **wide range of applications** :

# **Robotran**

- **Symbolic** software to model and analyze **Multibody Systems**
- Developed since **1990** at ***Université catholique de Louvain*** (UCL)
- Used in a **wide range of applications** :

Redundant actuators



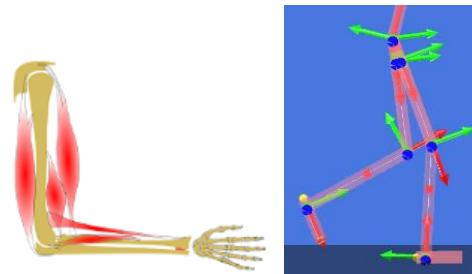
# **Robotran**

- **Symbolic** software to model and analyze **Multibody Systems**
- Developed since **1990** at ***Université catholique de Louvain*** (UCL)
- Used in a **wide range of applications** :

Redundant actuators



Biomechanics



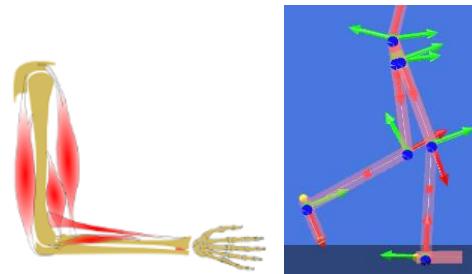
# *Robotran*

- **Symbolic** software to model and analyze **Multibody Systems**
- Developed since **1990** at ***Université catholique de Louvain*** (UCL)
- Used in a **wide range of applications** :

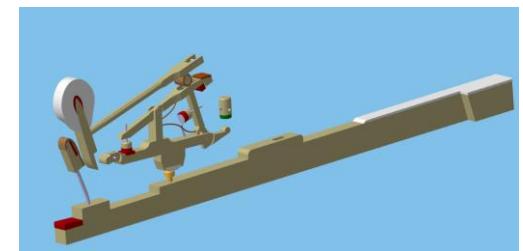
Redundant actuators



Biomechanics



Piano action



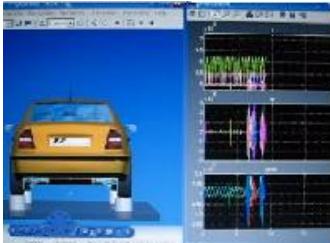
# *Robotran*

- **Symbolic** software to model and analyze **Multibody Systems**
- Developed since **1990** at ***Université catholique de Louvain*** (UCL)
- Used in a **wide range of applications** :

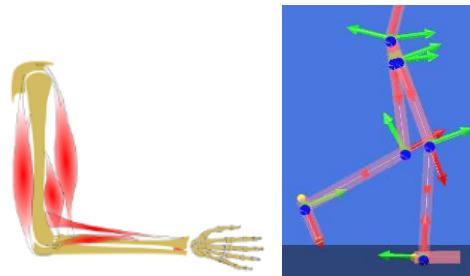
Redundant actuators



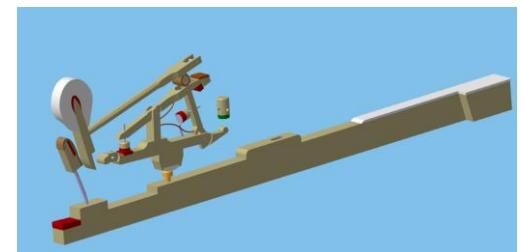
Road vehicles



Biomechanics



Piano action



# *Robotran*

- **Symbolic** software to model and analyze **Multibody Systems**
- Developed since **1990** at ***Université catholique de Louvain*** (UCL)
- Used in a **wide range of applications** :

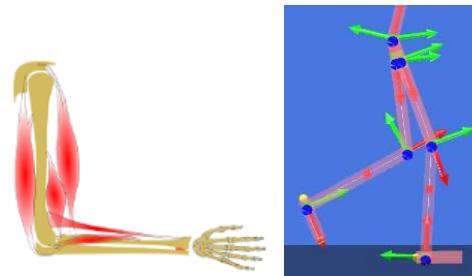
Redundant actuators



Road vehicles



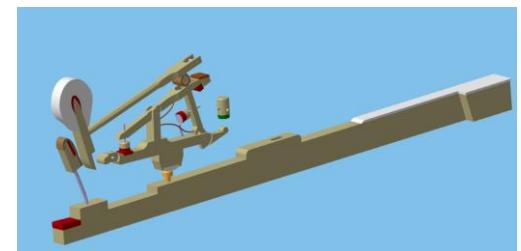
Biomechanics



Railway dynamics



Piano action



# *Robotran*

- **Symbolic** software to model and analyze **Multibody Systems**
- Developed since **1990** at ***Université catholique de Louvain*** (UCL)
- Used in a **wide range of applications** :

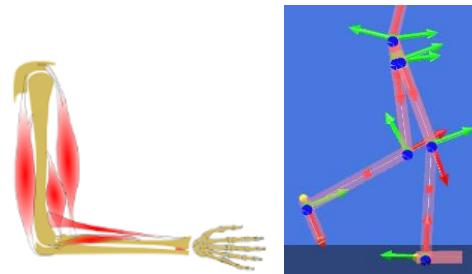
Redundant actuators



Road vehicles



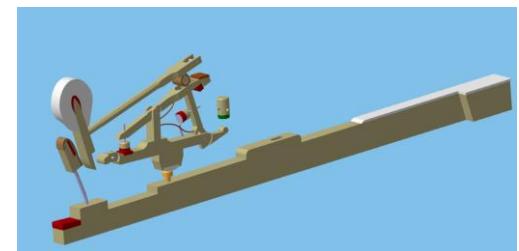
Biomechanics



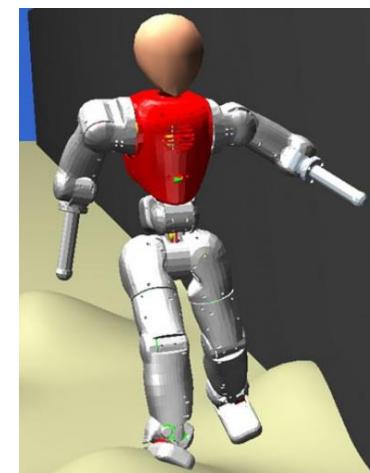
Railway dynamics



Piano action



Robotics

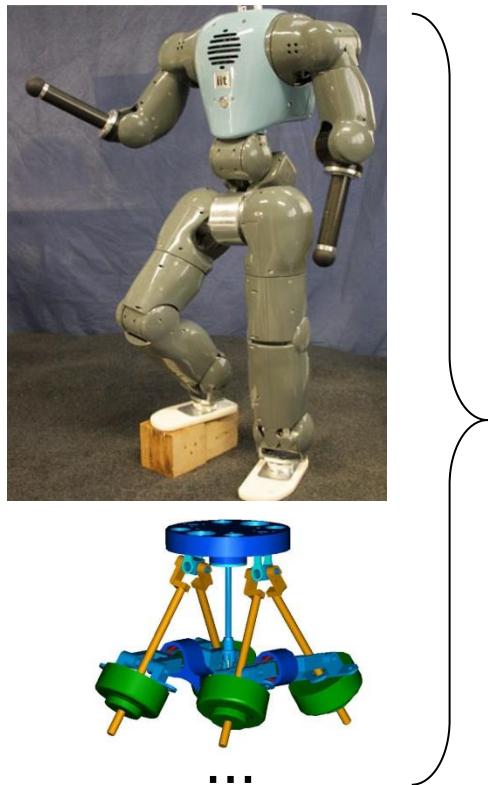


# Robotran formalism

## Abstraction

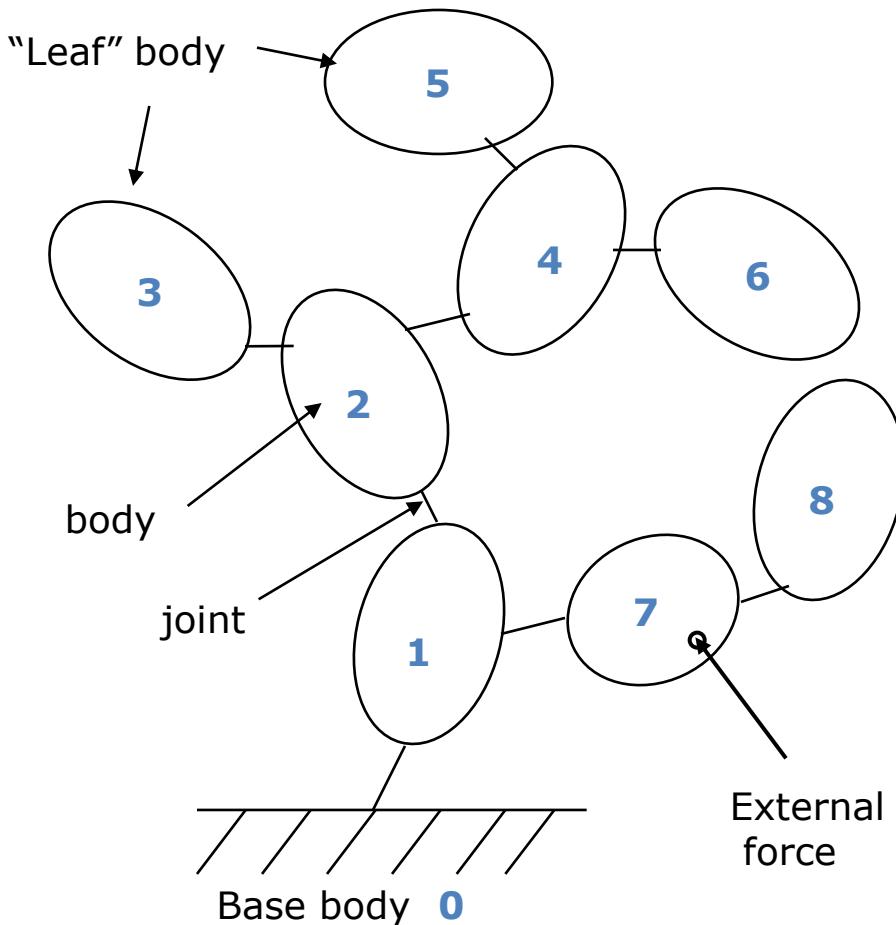
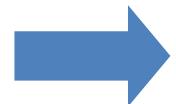
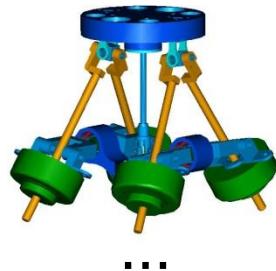
# Robotran formalism

## Abstraction



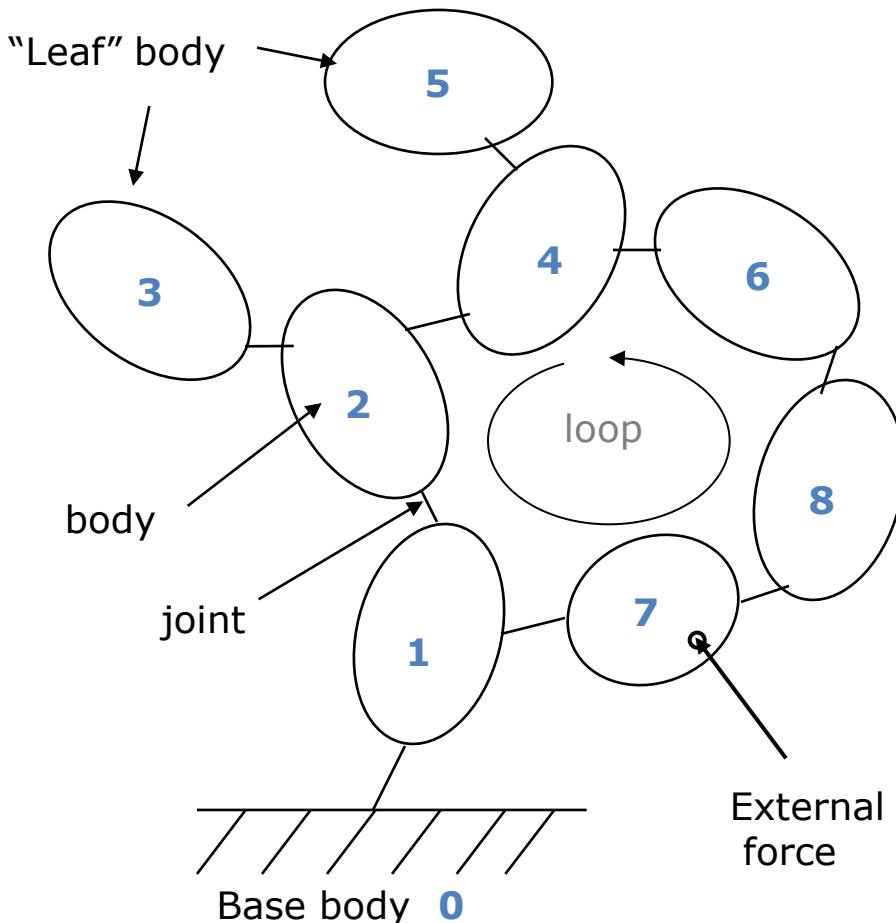
# Robotran formalism

## Abstraction



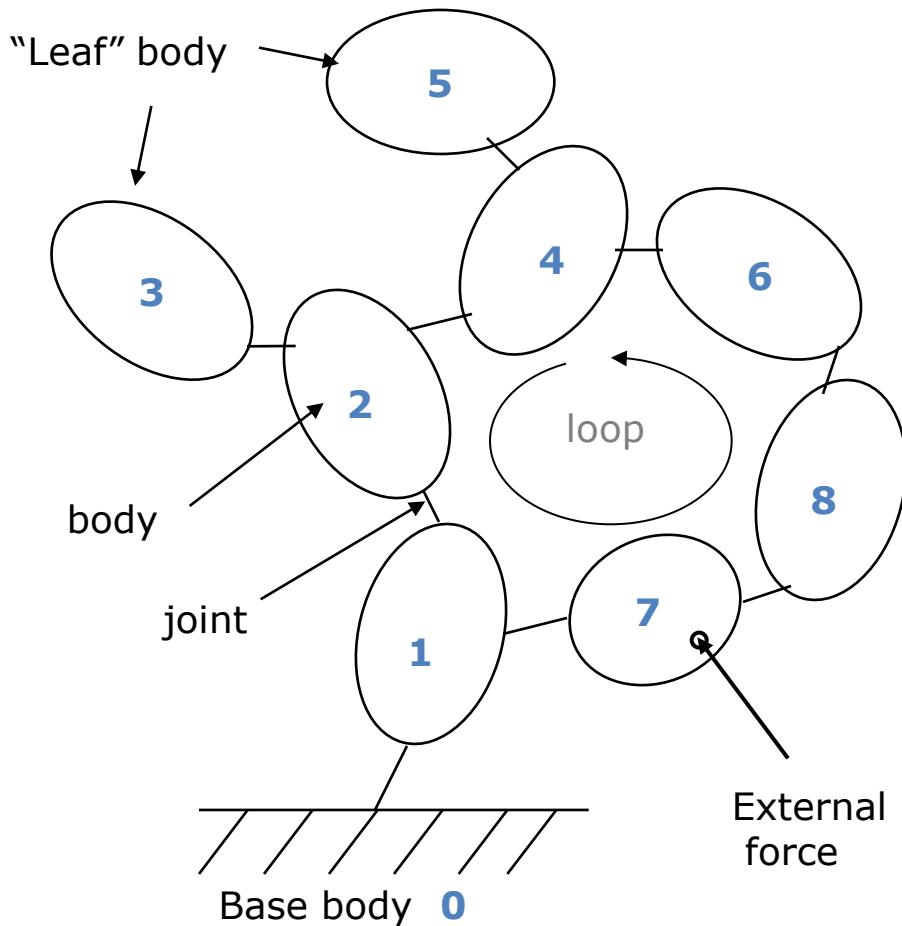
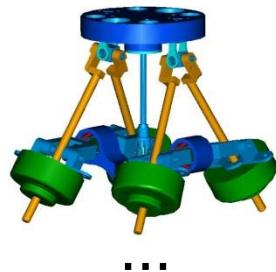
# Robotran formalism

## Abstraction



# Robotran formalism

## Abstraction



## Equations

Each body must satisfy :

$$\left\{ \begin{array}{l} \text{Newton: } m(C) \ddot{\mathbf{x}}^G = \mathbf{F} \\ \text{Euler: } \mathbf{I}^G \cdot \dot{\boldsymbol{\omega}} + \tilde{\boldsymbol{\omega}} \cdot \mathbf{I}^G \cdot \boldsymbol{\omega} = \mathbf{L}^G \end{array} \right.$$

# Robotran formalism

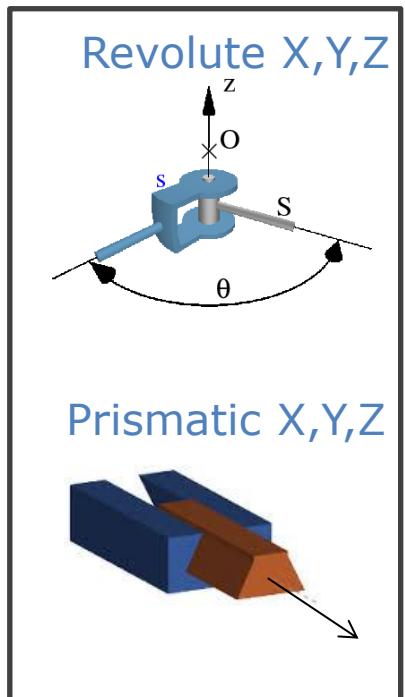
## Body (Rigid)

**10 parameters** : mass (1), center of mass (3), inertia matrix (6)

## Joint

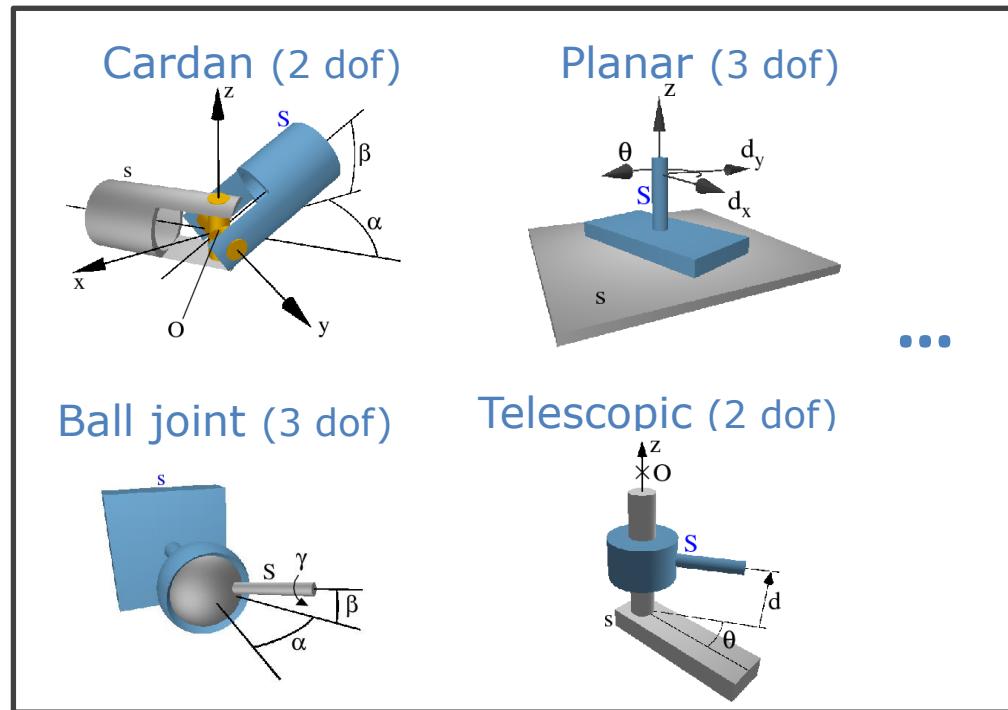
### Relative coordinates

6 basic joints (1dof)



combination  
→

Any joints



# Symbolic generation

ways to generate multibody equations:

# Symbolic generation

ways to generate multibody equations:

## 1) Manually

Generating by hand the equations of motion (E.O.M) specific to the model.

+ : simplification of math

- : small system only

$$\begin{aligned} a &= \ddot{s} = l\ddot{\theta} \\ l\ddot{\theta} &= -g \sin \theta \\ \ddot{\theta} &= -\frac{g}{l} \sin \theta \end{aligned}$$

# Symbolic generation

ways to generate multibody equations:

## 1) Manually

Generating by hand the equations of motion (E.O.M) specific to the model.

+ : simplification of math

- : small system only

$$\begin{aligned} a &= \ddot{s} = l\ddot{\theta} \\ l\ddot{\theta} &= -g \sin \theta \\ \ddot{\theta} &= -\frac{g}{l} \sin \theta \end{aligned}$$

## 2) Numerically

Solving the E.O.M with generic code.

++ : handle very large system

```
If(next_joint == revolute)  
then F = matrix_mult(A,B)
```

- : code not optimized for the system  
- : rebuilt model at each time step

# Symbolic generation

ways to generate multibody equations:

## 1) Manually

Generating by hand the equations of motion (E.O.M) specific to the model.

+ : simplification of math

- : small system only

$$\begin{aligned} a &= \ddot{s} = l\ddot{\theta} \\ l\ddot{\theta} &= -g \sin \theta \\ \ddot{\theta} &= -\frac{g}{l} \sin \theta \end{aligned}$$

## 2) Numerically

Solving the E.O.M with generic code.

++ : handle very large system

If (next\_joint == revolute)  
then  $F = \text{matrix\_mult}(A, B)$

- : code not optimized for the system  
- : rebuilt model at each time step

## 3) Symbolically

Automatic generation of E.O.M by a software specific to the model



+ : simplification of math

- : fixed topology

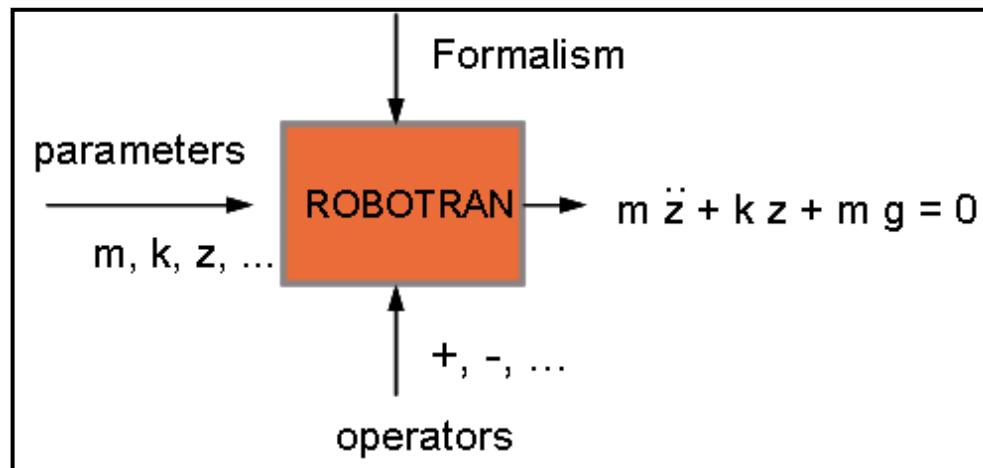
+ : handle large systems

# Symbolic generation

Robotran : a symbolic generator

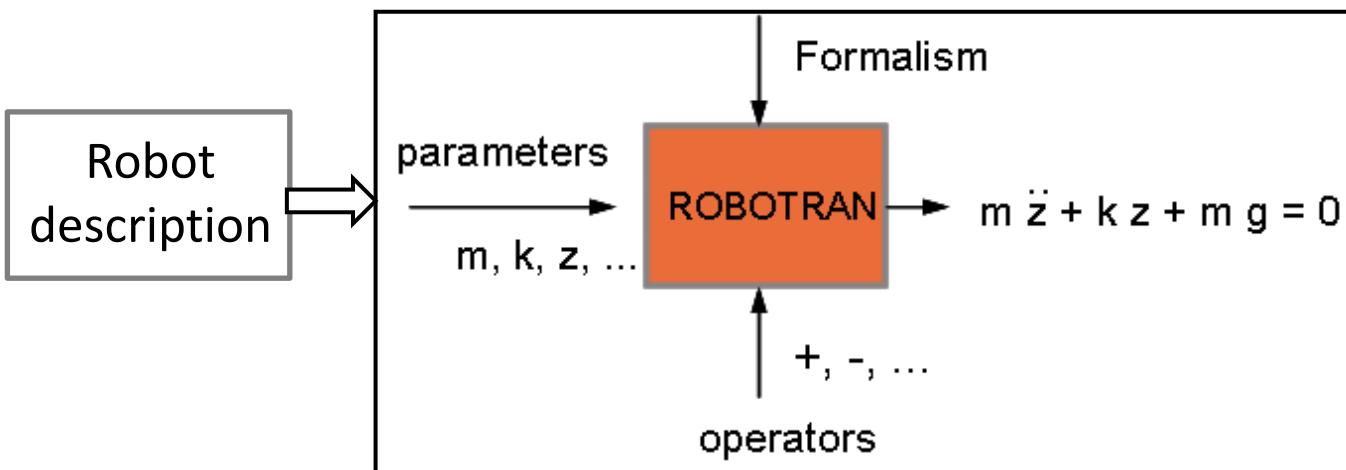
# Symbolic generation

Robotran : a symbolic generator



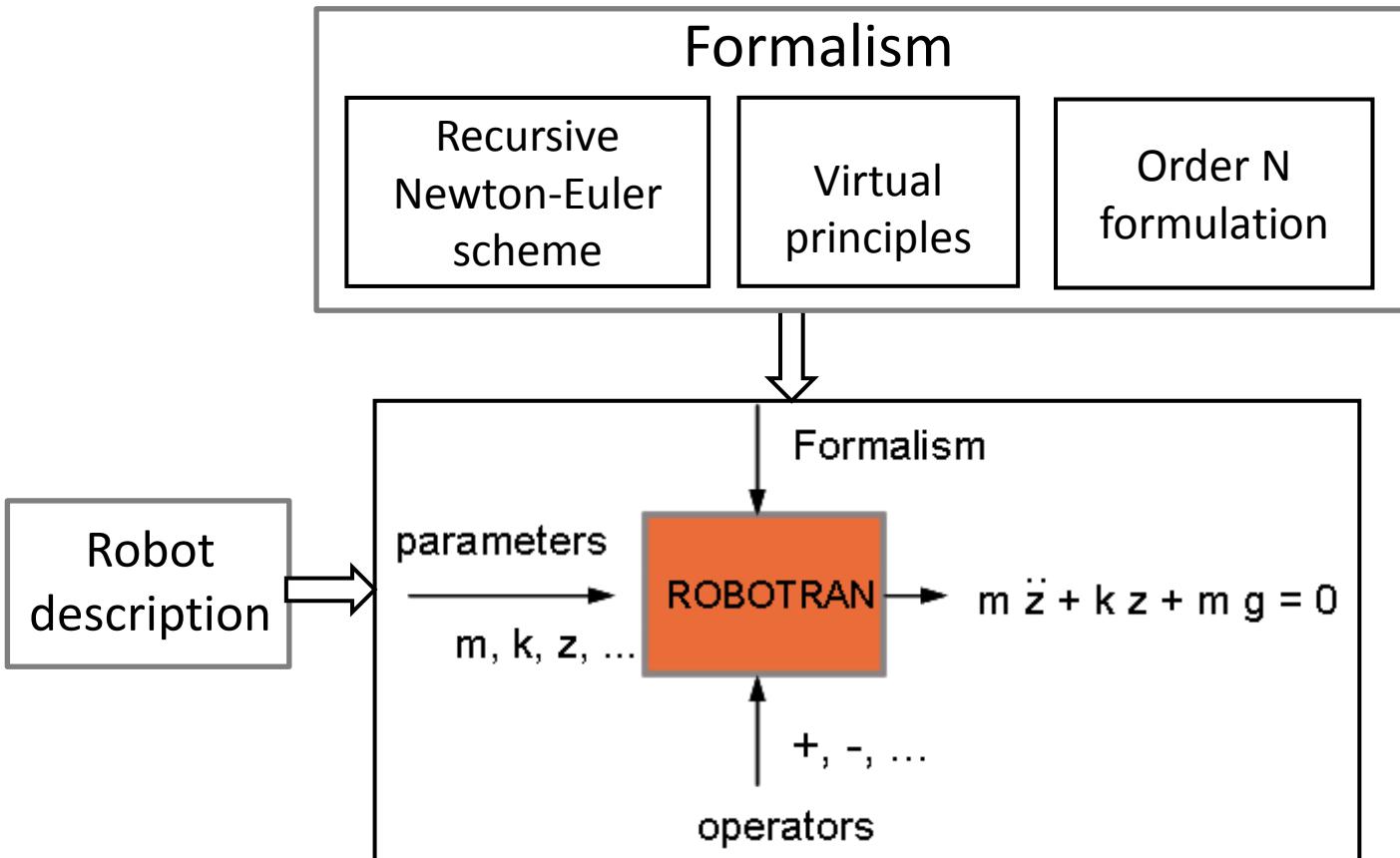
# Symbolic generation

Robotran : a symbolic generator



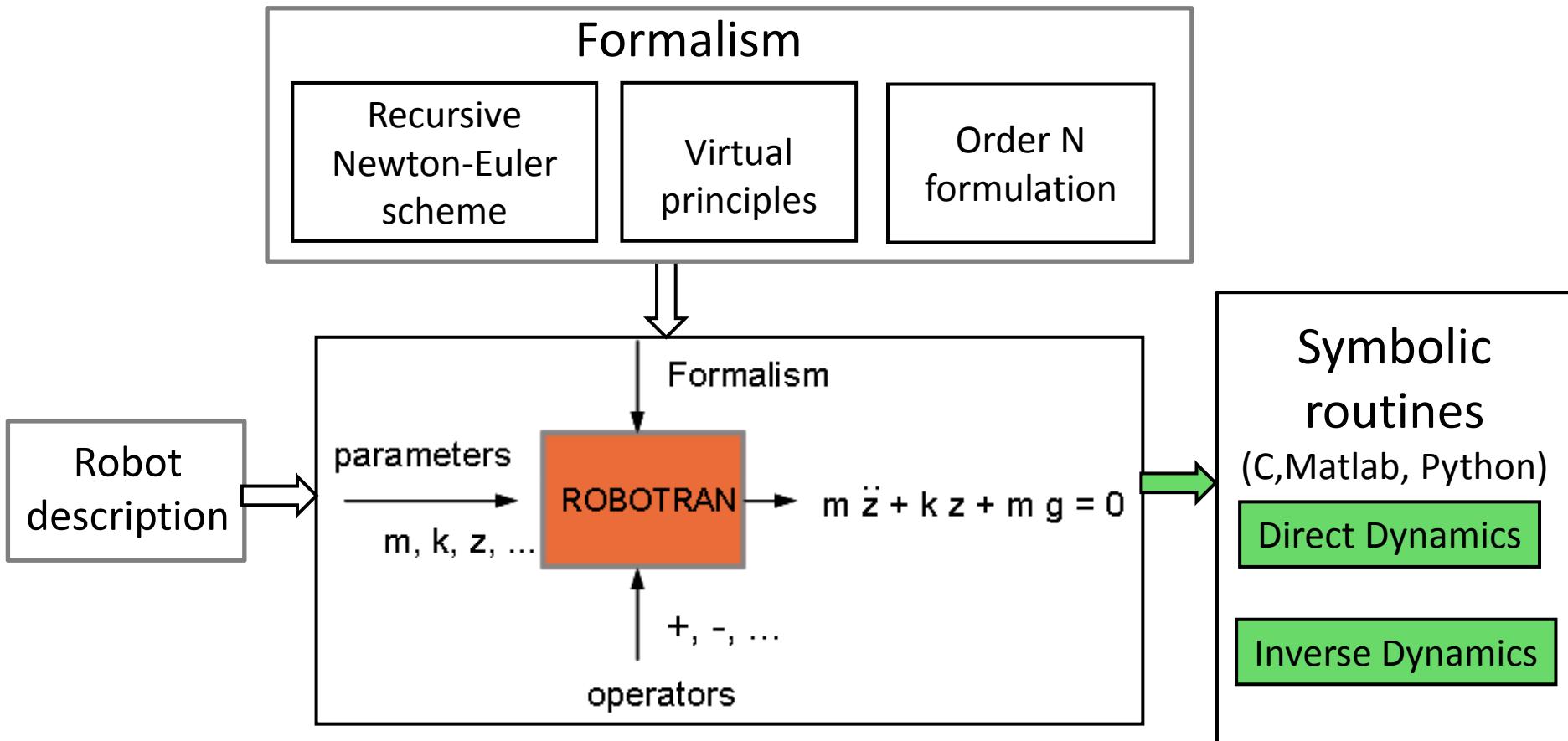
# Symbolic generation

## Robotran : a symbolic generator



# Symbolic generation

## Robotran : a symbolic generator



**certified up to 300 dof.**

# Symbolic generation

Symbolic routine :  
example

```
void dirdyna(double **M,double *c,MBSdataStruct *s, double tsim){\n\n    ... \n\n    // Trigonometric Variables \n\n    C7 = cos(q[7]);\n    S7 = sin(q[7]);\n    C8 = cos(q[8]);\n    S8 = sin(q[8]);\n\n    ... \n\n    // Forward Kinematics \n\n    OM113 = OM16*C13-OM36*S13;\n    OM313 = OM16*S13+OM36*C13;\n    AlF113 = C13*(AlF16+BS16*s->dpt[1][4]+BeF26*s->dpt[2][4]+BeF36*s->dpt[3][4])-S13*(AlF35+BS96*s->dpt[3][4]+BeF76*\n    s->dpt[1][4]+BeF86*s->dpt[2][4]);\n\n    ... \n\n    // Backward Dynamics \n\n    FA18 = -(s->frc[1][8]-s->m[8]*(AlF17*C8-AlF37*S8+s->l[3][8]*(OpF26+OM18*OM38)));\n    FA28 = -(s->frc[2][8]-s->m[8]*(AlF26+BS56*s->dpt[2][1]+BeF46*s->dpt[1][1]+BeF66*s->dpt[3][1]+s->l[3][8]*(OM28*OM38+C8*\n    (qd[8]*OM37-C7*(OpF16-qd[7]*OM36)+S7*(OpF35+qd[7]*OM16))+S8*(qd[8]*OM17+C7*(OpF35+qd[7]*OM16)+S7*(OpF16-qd[7]*OM36))));\n    FA38 = -(s->frc[3][8]-s->m[8]*(AlF17*S8+AlF37*C8-s->l[3][8]*(OM18*OM18+OM28*OM28)));\n\n    ... \n\n    // Symbolic Outputs \n\n    M[1][1] = FM41_15;\n    M[1][2] = FM31_24;\n    M[1][3] = FM31_34;\n    M[1][4] = CM41_15;\n\n    ... \n    c[1] = FF4_15;\n    c[2] = FF3_24;\n    c[3] = FF3_34;\n\n}\n\n}
```

# Symbolic generation

Symbolic routine :  
example

```
void dirdyna(double **M,double *c,MBSdataStruct *s, double tsim){  
...  
// Trigonometric Variables  
  
C7 = cos(q[7]);  
S7 = sin(q[7]);  
C8 = cos(q[8]);|  
S8 = sin(q[8]);  
...  
  
// Forward Kinematics  
  
OM113 = OM16*C13-OM36*S13;  
OM313 = OM16*S13+OM36*C13;  
AlF113 = C13*(AlF16+BS16*s->dpt[1][4]+BeF26*s->dpt[2][4]+BeF36*s->dpt[3][4])-S13*(AlF35+BS96*s->dpt[3][4]+BeF76*s->dpt[1][4]+BeF86*s->dpt[2][4]);  
...  
  
// Backward Dynamics  
  
FA18 = -(s->frc[1][8]-s->m[8]*(AlF17*C8-AlF37*S8+s->l[3][8]*(OpF26+OM18*OM38)));  
FA28 = -(s->frc[2][8]-s->m[8]*(AlF26+BS56*s->dpt[2][1]+BeF46*s->dpt[1][1]+BeF66*s->dpt[3][1]+s->l[3][8]*(OM28*OM38+C8*(qd[8]*OM37-C7*(OpF16-qd[7]*OM36)+S7*(OpF35+qd[7]*OM16))+S8*(qd[8]*OM17+C7*(OpF35+qd[7]*OM16)+S7*(OpF16-qd[7]*OM36))));  
FA38 = -(s->frc[3][8]-s->m[8]*(AlF17*S8+AlF37*C8-s->l[3][8]*(OM18*OM18+OM28*OM28)));  
...  
  
// Symbolic Outputs  
  
M[1][1] = FM41_15;  
M[1][2] = FM31_24;  
M[1][3] = FM31_34;  
M[1][4] = CM41_15;  
...  
c[1] = FF4_15;  
c[2] = FF3_24;  
c[3] = FF3_34;  
}  
}
```

# Symbolic generation

Symbolic routine :  
example

```
void dirdyna(double **M,double *c,MBSdataStruct *s, double tsim){  
...  
// Trigonometric Variables  
  
C7 = cos(q[7]);  
S7 = sin(q[7]);  
C8 = cos(q[8]);|  
S8 = sin(q[8]);  
...  
  
// Forward Kinematics  
  
OM113 = OM16*C13-OM36*S13;  
OM313 = OM16*S13+OM36*C13;  
AlF113 = C13*(AlF16+BS16*s->dpt[1][4]+BeF26*s->dpt[2][4]+BeF36*s->dpt[3][4])-S13*(AlF35+BS96*s->dpt[3][4]+BeF76*s->dpt[1][4]+BeF86*s->dpt[2][4]);  
...  
  
// Backward Dynamics  
  
FA18 = -(s->frc[1][8]-s->m[8]*(AlF17*C8-AlF37*S8+s->l[3][8]*(OpF26+OM18*OM38)));  
FA28 = -(s->frc[2][8]-s->m[8]*(AlF26+BS56*s->dpt[2][1]+BeF46*s->dpt[1][1]+BeF66*s->dpt[3][1]+s->l[3][8]*(OM28*OM38+C8*(qd[8]*OM37-C7*(OpF16-qd[7]*OM36)+S7*(OpF35+qd[7]*OM16))+S8*(qd[8]*OM17+C7*(OpF35+qd[7]*OM16)+S7*(OpF16-qd[7]*OM36))));  
FA38 = -(s->frc[3][8]-s->m[8]*(AlF17*S8+AlF37*C8-s->l[3][8]*(OM18*OM18+OM28*OM28)));  
...  
  
// Symbolic Outputs  
  
M[1][1] = FM41_15;  
M[1][2] = FM31_24;  
M[1][3] = FM31_34;  
M[1][4] = CM41_15;  
...  
c[1] = FF4_15;  
c[2] = FF3_24;  
c[3] = FF3_34;  
}  
}
```

Trigo of joint angles

Joint location



# Symbolic generation

Symbolic routine :  
example

```
void dirdyna(double **M,double *c,MBSdataStruct *s, double tsim){  
...  
// Trigonometric Variables  
  
C7 = cos(q[7]);  
S7 = sin(q[7]);  
C8 = cos(q[8]);|  
S8 = sin(q[8]);  
...  
  
// Forward Kinematics  
  
OM113 = OM16*C13-OM36*S13;  
OM313 = OM16*S13+OM36*C13;  
AlF113 = C13*(AlF16+BS16*s->dpt[1][4]+BeF26*s->dpt[2][4]+BeF36*s->dpt[3][4])-S13*(AlF35+BS96*s->dpt[3][4]+BeF76*s->dpt[1][4]+BeF86*s->dpt[2][4]);  
...  
  
// Backward Dynamics  
  
FA18 = -(s->frc[1][8]-s->m[8]*(AlF17*C8-AlF37*S8+s->l[3][8]*(OpF26+OM18*OM38)));  
FA28 = -(s->frc[2][8]-s->m[8]*(AlF26+BS56*s->dpt[2][1]+BeF46*s->dpt[1][1]+BeF66*s->dpt[3][1]+s->l[3][8]*(OM28*OM38+C8*(qd[8]*OM37-C7*(OpF16-qd[7]*OM36)+S7*(OpF35+qd[7]*OM16))+S8*(qd[8]*OM17+C7*(OpF35+qd[7]*OM16)+S7*(OpF16-qd[7]*OM36))));  
FA38 = -(s->frc[3][8]-s->m[8]*(AlF17*S8+AlF37*C8-s->l[3][8]*(OM18*OM18+OM28*OM28)));  
...  
  
// Symbolic Outputs  
  
M[1][1] = FM41_15;  
M[1][2] = FM31_24;  
M[1][3] = FM31_34;  
M[1][4] = CM41_15;  
...  
c[1] = FF4_15;  
c[2] = FF3_24;  
c[3] = FF3_34;  
}  
}
```

Trigo of joint angles

Joint location

Body mass

# Symbolic generation

Symbolic routine :  
example

```
void dirdyna(double **M, double *c, MBSdataStruct *s, double tsim){  
...  
// Trigonometric Variables  
  
C7 = cos(q[7]);  
S7 = sin(q[7]);  
C8 = cos(q[8]);|  
S8 = sin(q[8]);  
...  
}
```

Trigo of joint angles

```
// Forward Kinematics
```

```
OM113 = OM16*C13-OM36*S13;  
OM313 = OM16*S13+OM36*C13;  
AlF113 = C13*(AlF16+BS16*s->dpt[1][4]+BeF26*s->dpt[2][4]+BeF36*s->dpt[3][4])-S13*(AlF35+BS96*s->dpt[3][4]+BeF76*s->dpt[1][4]+BeF86*s->dpt[2][4]);  
...  
}
```

Joint location

```
// Backward Dynamics
```

```
FA18 = -(s->frc[1][8]-s->m[8]*(AlF17*C8-AlF37*S8+s->l[3][8]*(OpF26+OM18*OM38)));  
FA28 = -(s->frc[2][8]-s->m[8]*(AlF26+BS56*s->dpt[2][1]+BeF46*s->dpt[1][1]+BeF66*s->dpt[3][1]+s->l[3][8]*(OM28*OM38+C8*(qd[8]*OM37-C7*(OpF16-qd[7]*OM36)+S7*(OpF35+qd[7]*OM16))+S8*(qd[8]*OM17+C7*(OpF35+qd[7]*OM16)+S7*(OpF16-qd[7]*OM36))));  
FA38 = -(s->frc[3][8]-s->m[8]*(AlF17*S8+AlF37*C8-s->l[3][8]*(OM18*OM18+OM28*OM28)));  
...  
}
```

Body mass

```
// Symbolic Outputs
```

```
M[1][1] = FM41_15;  
M[1][2] = FM31_24; ← Mass matrix  
M[1][3] = FM31_34;  
M[1][4] = CM41_15;  
...  
}
```

```
c[1] = FF4_15;  
c[2] = FF3_24;  
c[3] = FF3_34;
```

# Symbolic generation

Symbolic routine :  
example

```
void dirdyna(double **M, double *c, MBSdataStruct *s, double tsim){  
...  
// Trigonometric Variables  
  
C7 = cos(q[7]);  
S7 = sin(q[7]);  
C8 = cos(q[8]);|  
S8 = sin(q[8]);  
...  
}
```

Trigo of joint angles

```
// Forward Kinematics
```

```
OM113 = OM16*C13-OM36*S13;  
OM313 = OM16*S13+OM36*C13;  
AlF113 = C13*(AlF16+BS16*s->dpt[1][4]+BeF26*s->dpt[2][4]+BeF36*s->dpt[3][4])-S13*(AlF35+BS96*s->dpt[3][4]+BeF76*s->dpt[1][4]+BeF86*s->dpt[2][4]);  
...  
}
```

Joint location

```
// Backward Dynamics
```

```
FA18 = -(s->frc[1][8]-s->m[8]*(AlF17*C8-AlF37*S8+s->l[3][8]*(OpF26+OM18*OM38)));  
FA28 = -(s->frc[2][8]-s->m[8]*(AlF26+BS56*s->dpt[2][1]+BeF46*s->dpt[1][1]+BeF66*s->dpt[3][1]+s->l[3][8]*(OM28*OM38+C8*(qd[8]*OM37-C7*(OpF16-qd[7]*OM36)+S7*(OpF35+qd[7]*OM16))+S8*(qd[8]*OM17+C7*(OpF35+qd[7]*OM16)+S7*(OpF16-qd[7]*OM36))));  
FA38 = -(s->frc[3][8]-s->m[8]*(AlF17*S8+AlF37*C8-s->l[3][8]*(OM18*OM18+OM28*OM28)));  
...  
}
```

Body mass

```
// Symbolic Outputs
```

```
M[1][1] = FM41_15;  
M[1][2] = FM31_24;  
M[1][3] = FM31_34;  
M[1][4] = CM41_15;  
...  
c[1] = FF4_15;  
c[2] = FF3_24;  
c[3] = FF3_34;
```

Mass matrix

$$M(q)\ddot{q} + C(q, \dot{q}) = Q_q$$

# Symbolic generation

Symbolic simplifications :

# Symbolic generation

## Symbolic simplifications :

### Arithmetic

$$a + b + c - b$$

$$0*a$$

# Symbolic generation

## Symbolic simplifications :

### Arithmetic

$$\begin{array}{l} \cancel{a + b + c - b} \rightarrow a + \\ \cancel{0 * a} \rightarrow 0 \end{array}$$

e.g. Gravity = [0,0,-9.81]

# Symbolic generation

## Symbolic simplifications :

### Arithmetic

$$\begin{array}{l} \cancel{a + b + c - b} \\ \cancel{0 * a} \end{array} \quad \Rightarrow a + \quad \Rightarrow 0$$

e.g. Gravity = [0,0,-9.81]

### Trigonometric

$$\cos^2(\alpha) + \sin^2(\alpha)$$

$$2\cos(\alpha)\sin(\alpha)$$

# Symbolic generation

## Symbolic simplifications :

### Arithmetic

$$\begin{array}{l} \cancel{a + b + c - b} \rightarrow a + \\ \cancel{0 * a} \rightarrow 0 \end{array}$$

e.g. Gravity = [0,0,-9.81]

### Trigonometric

$$\begin{array}{l} \cos^2(\alpha) + \sin^2(\alpha) \rightarrow 1 \\ 2\cos(\alpha)\sin(\alpha) \rightarrow \sin(2\alpha) \end{array}$$

# Symbolic generation

## Symbolic simplifications :

### Arithmetic

$$\begin{array}{ll} \cancel{a + b + c - b} & \rightarrow a + \\ \cancel{0 * a} & \rightarrow 0 \end{array}$$

e.g. Gravity = [0,0,-9.81]

### Trigonometric

$$\begin{array}{ll} \cos^2(\alpha) + \sin^2(\alpha) & \rightarrow 1 \\ 2\cos(\alpha)\sin(\alpha) & \rightarrow \sin(2\alpha) \end{array}$$

### Recursive

```
// Data  
A  
B  
C  
D
```

### // Equations

```
A1 = A + B  
A2 = 2 * C  
B1 = A1 + C  
B2 = A2 * A2  
B3 = C * A2  
C1 = A1 + B2  
C2 = B + B2  
D1 = C2 - C1  
D2 = C1 + C2
```

### // Results

M = D2

# Symbolic generation

## Symbolic simplifications :

### Arithmetic

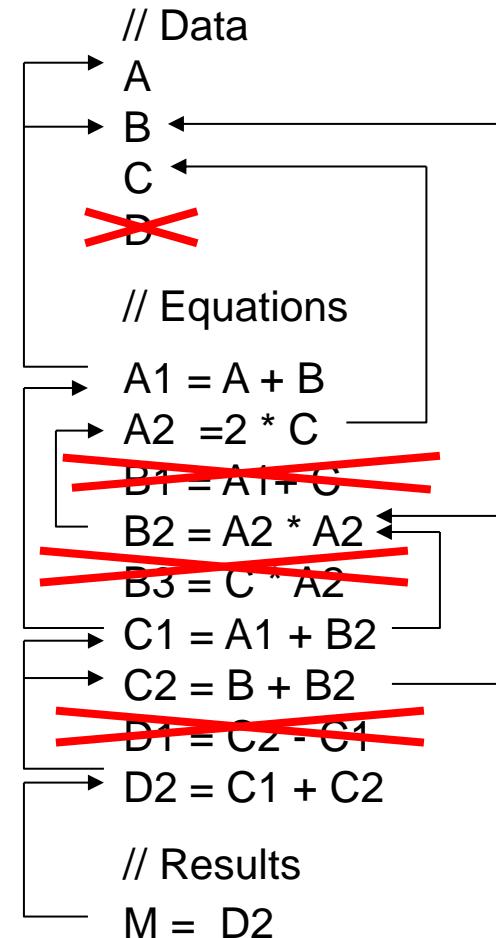
$$\begin{array}{lcl} \cancel{a + b + c - b} & \rightarrow & a + \\ \cancel{0 * a} & \rightarrow & 0 \end{array}$$

e.g. Gravity = [0,0,-9.81]

### Trigonometric

$$\begin{array}{lcl} \cos^2(\alpha) + \sin^2(\alpha) & \rightarrow & 1 \\ 2\cos(\alpha)\sin(\alpha) & \rightarrow & \sin(2\alpha) \end{array}$$

### Recursive



# Symbolic generation

## Symbolic simplifications :

### Arithmetic

$$\begin{array}{lcl} \cancel{a + b + c - b} & \rightarrow & a + \\ \cancel{0 * a} & \rightarrow & 0 \end{array}$$

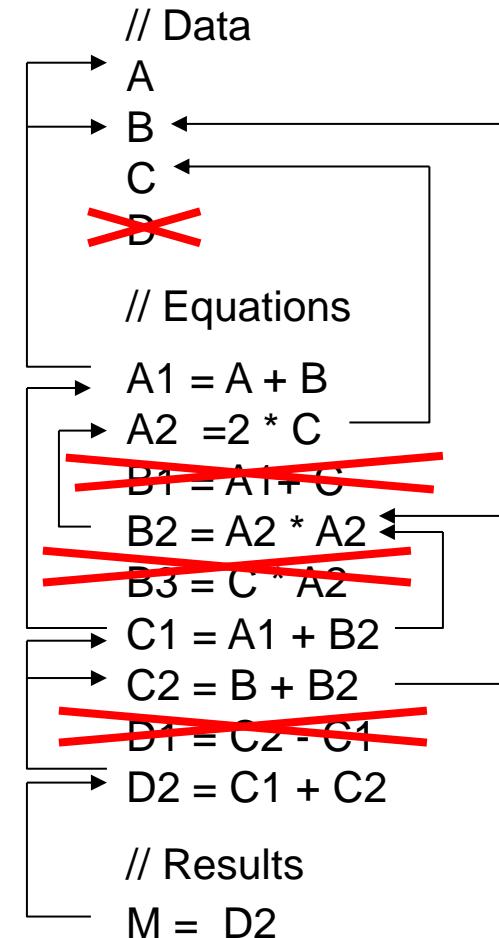
e.g. Gravity = [0,0,-9.81]

### Trigonometric

$$\begin{array}{lcl} \cos^2(\alpha) + \sin^2(\alpha) & \rightarrow & 1 \\ 2\cos(\alpha)\sin(\alpha) & \rightarrow & \sin(2\alpha) \end{array}$$

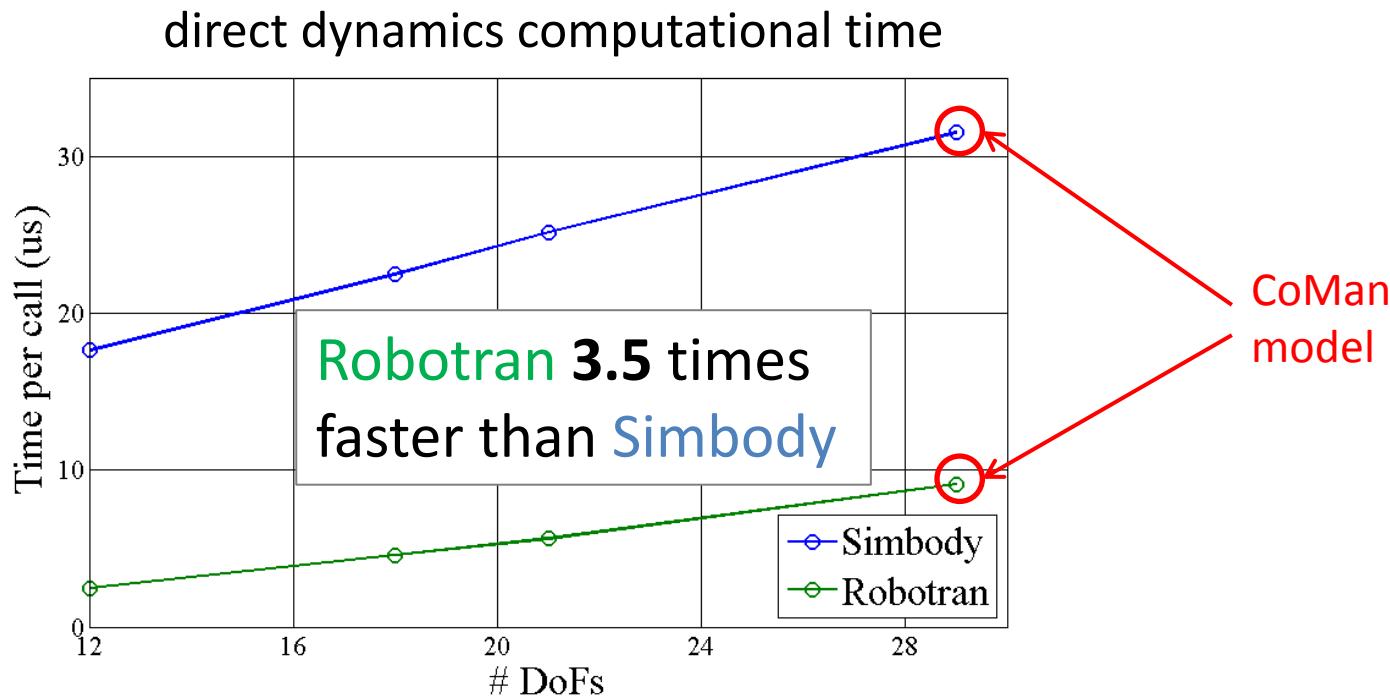
**30 to 90% of the operations are typically removed [1]**

### Recursive



# Symbolic generation

Symbolic (**Robotran**) is much faster than numeric (**Simbody**)



Note : Both Robotran and Simbody use relative coordinates.  
Their results were similar (diff =  $10^{-11}$ )

# Robotran in Practice

**Step 1 : *Draw your multibody system***



robot description  
file (xml)



**Step 2 : *Generate your multibody equations***



Symbolic routines  
(C, Matlab, python)



**Step 3 : *Simulate and analyse your multibody model***

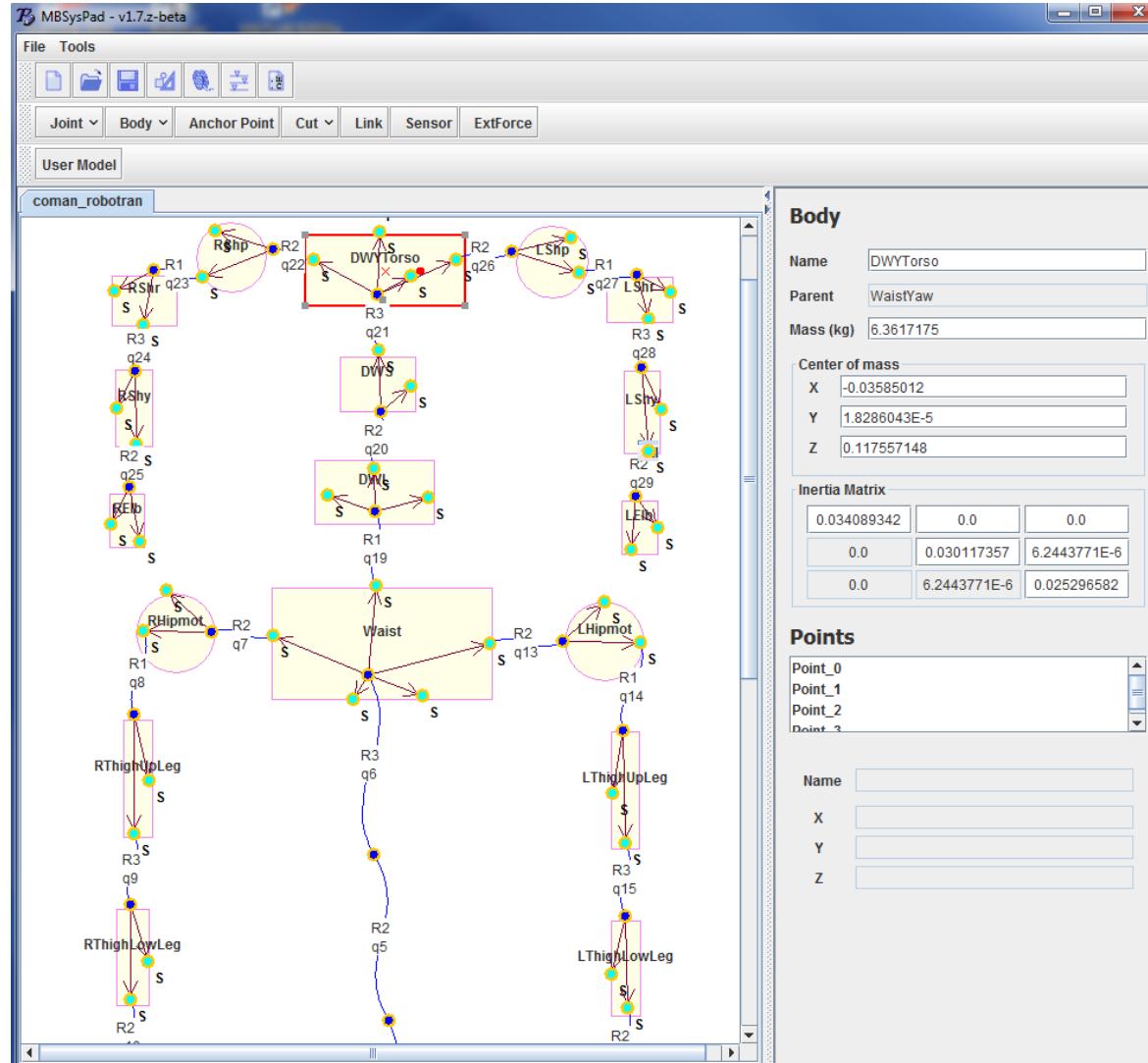


Results, video ...

# Robotran in Practice

Step 1 : *Draw your multibody system* using

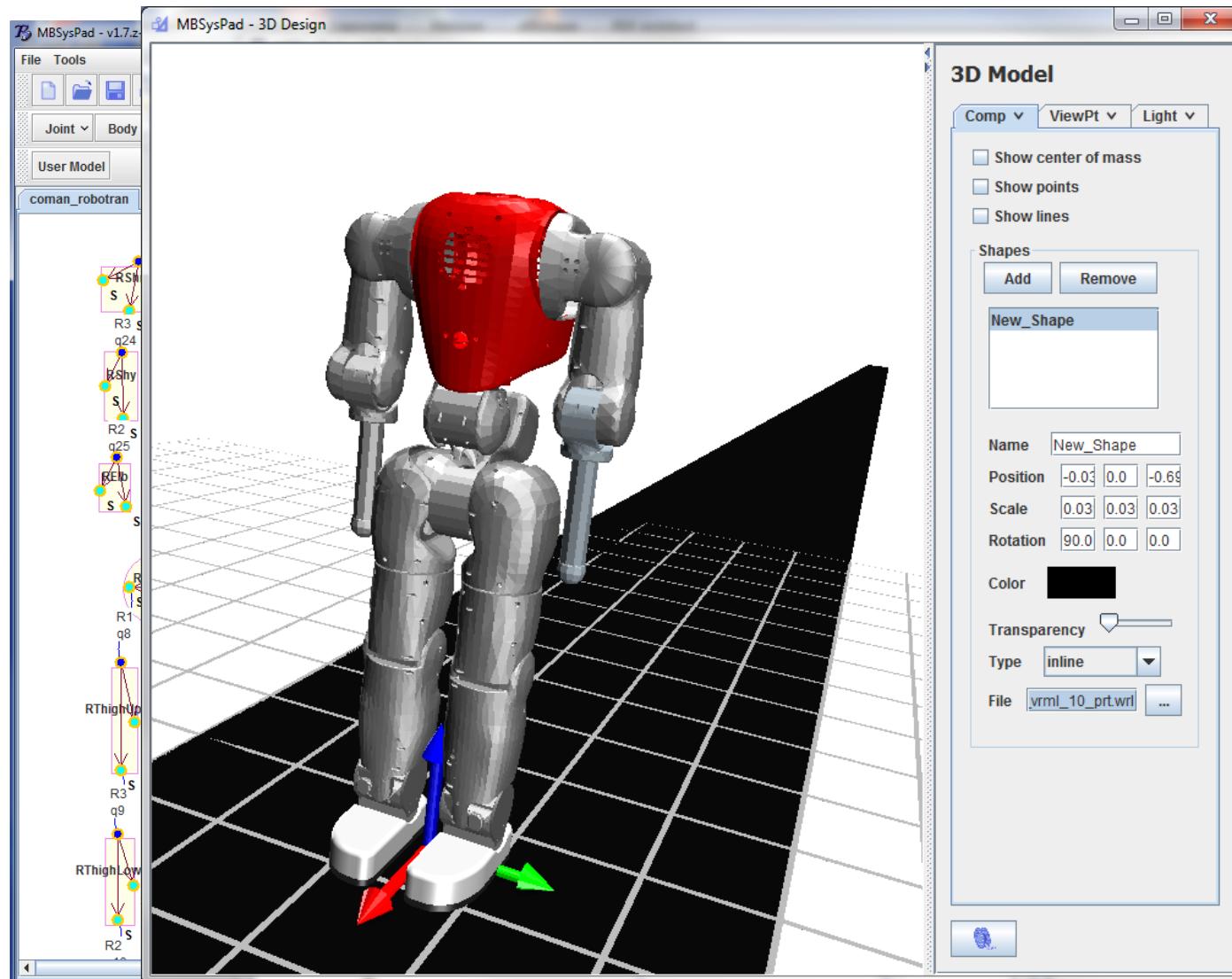
MBSysPad  
MBS Graphical  
Editor



# Robotran in Practice

Step 1 : *Draw your multibody system* using

MBSysPad  
MBS Graphical  
Editor



# Robotran in Practice

Step 2 : Generate your multibody equations using



Your model is sent to a sever, which sends back the model symbolic function

PRM - Robotran client properties

user name	habra	password	
server	robotran-soft.meca.ucl.ac.be	port	1099
languages	Matlab, C	serv	
functions	dirdyna, invdyna		
ssl trust store	password: *****	provider	SUN
upload folder	C:/Users/habra/.robotran/MBsysPad/upload		
template folder	C:/Users/habra/.robotran/MBsysPad/template		
project folder	C:/Users/habra/Documents/MBProjects/coman_robotran		
xml file	C:/Users/habra/Documents/MBProjects/coman_robotran\dataR\coman_robotran.mbs		
full symbolic	<input type="checkbox"/>		

Save To File    Reset    Stop    Generate

# Robotran in Practice

Step 3 : *Simulate and analyze your multibody model using*



- **Cross-language**

Collection of routines in 3 different environments :

Matlab : fast code prototyping + Matlab toolbox

Python : like Matlab but free

C/C++ : Fast execution + embedded code

Available in pure C (Cmake) or with Simulink

- **Cross-platform**



- **Open source**

→ easy to interface with user code

Note : User interact with symbolic routines through user functions  
(e.g. external forces, joint force, constraints, ...).

# Outline

- **Robotran Principles**  
Generation of symbolic routines for dynamics
- **Robotics Simulator**  
Available features for robot simulation
- **Robotran-Yarp interface**  
Running a Yarp module with Robotran

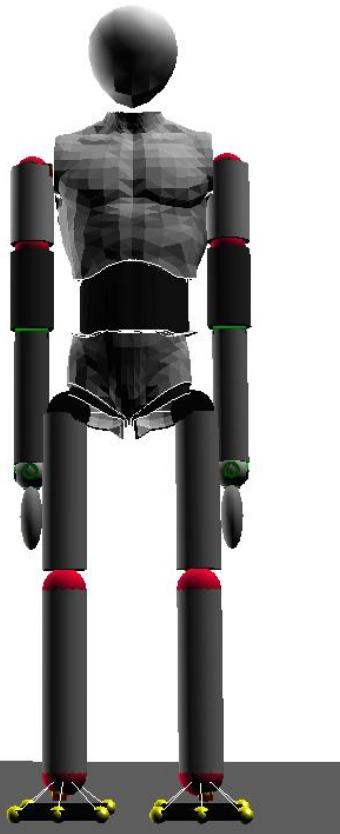
# Models

- Data from mechanical design
- Series Elastic Actuators
- Ground contact model

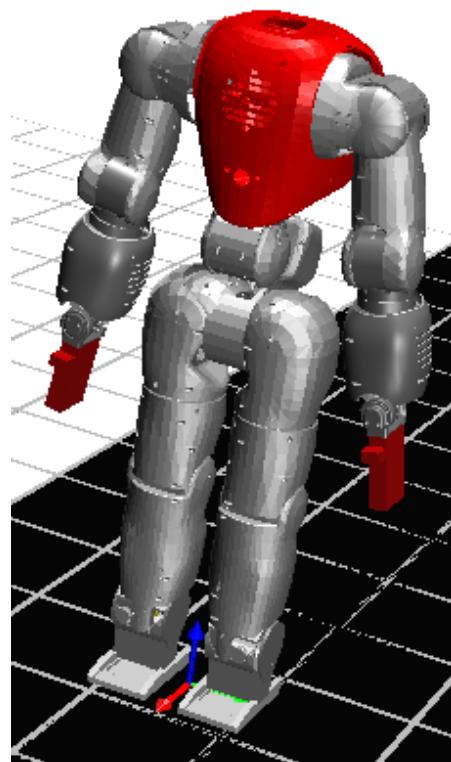
# Models

- Data from mechanical design
- Series Elastic Actuators
- Ground contact model

**Walkman**



**CoMan**

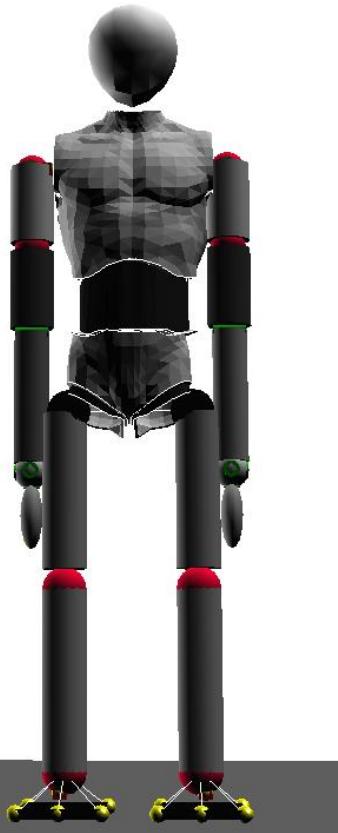


# Models

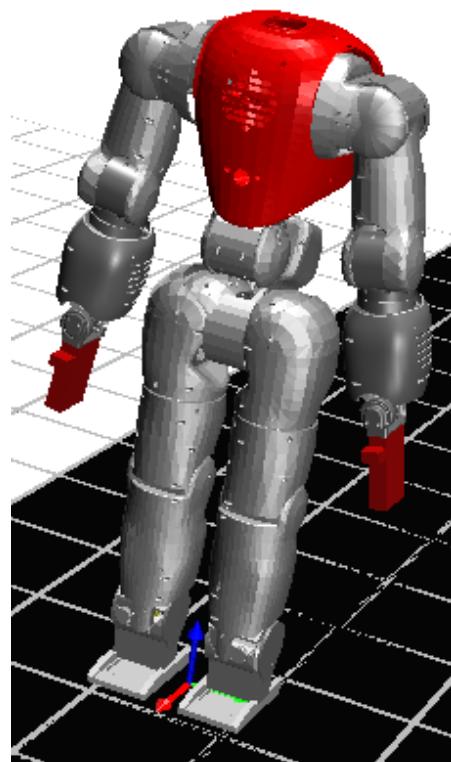
- Data from mechanical design
- Series Elastic Actuators
- Ground contact model

Available Soon ...

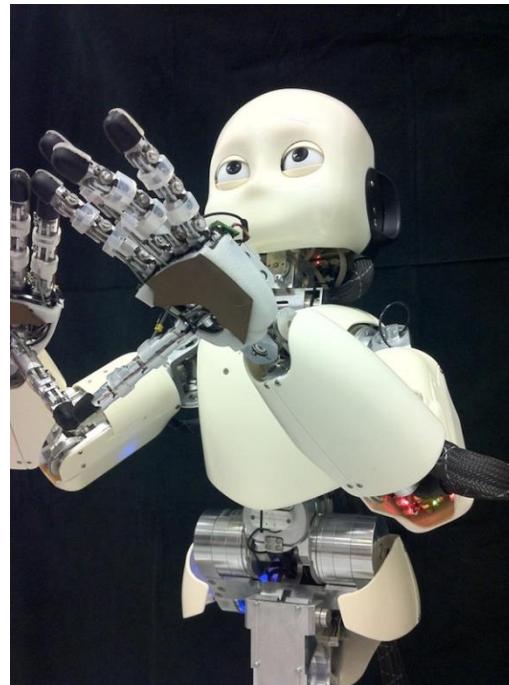
**Walkman**



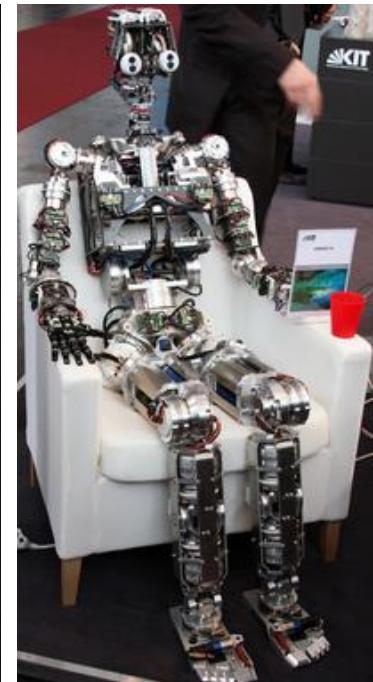
**CoMan**



**iCub**



**Armar IV (KIT)**



# Actuator model

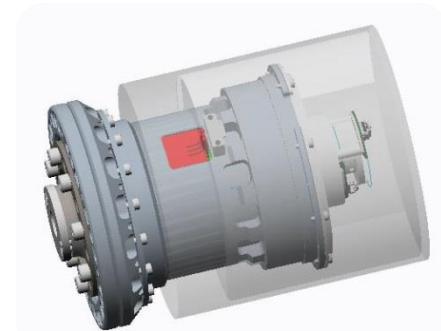
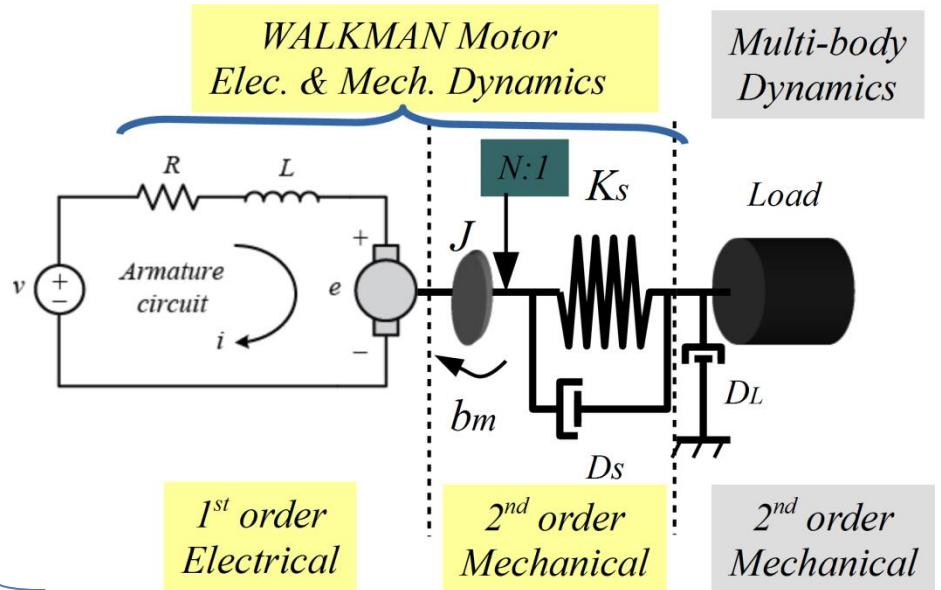
Actuation Model Equation

$$M(\theta) \frac{d^2\theta}{dt^2} + C(\theta, \dot{\theta}, f, \tau) = \tau_L$$

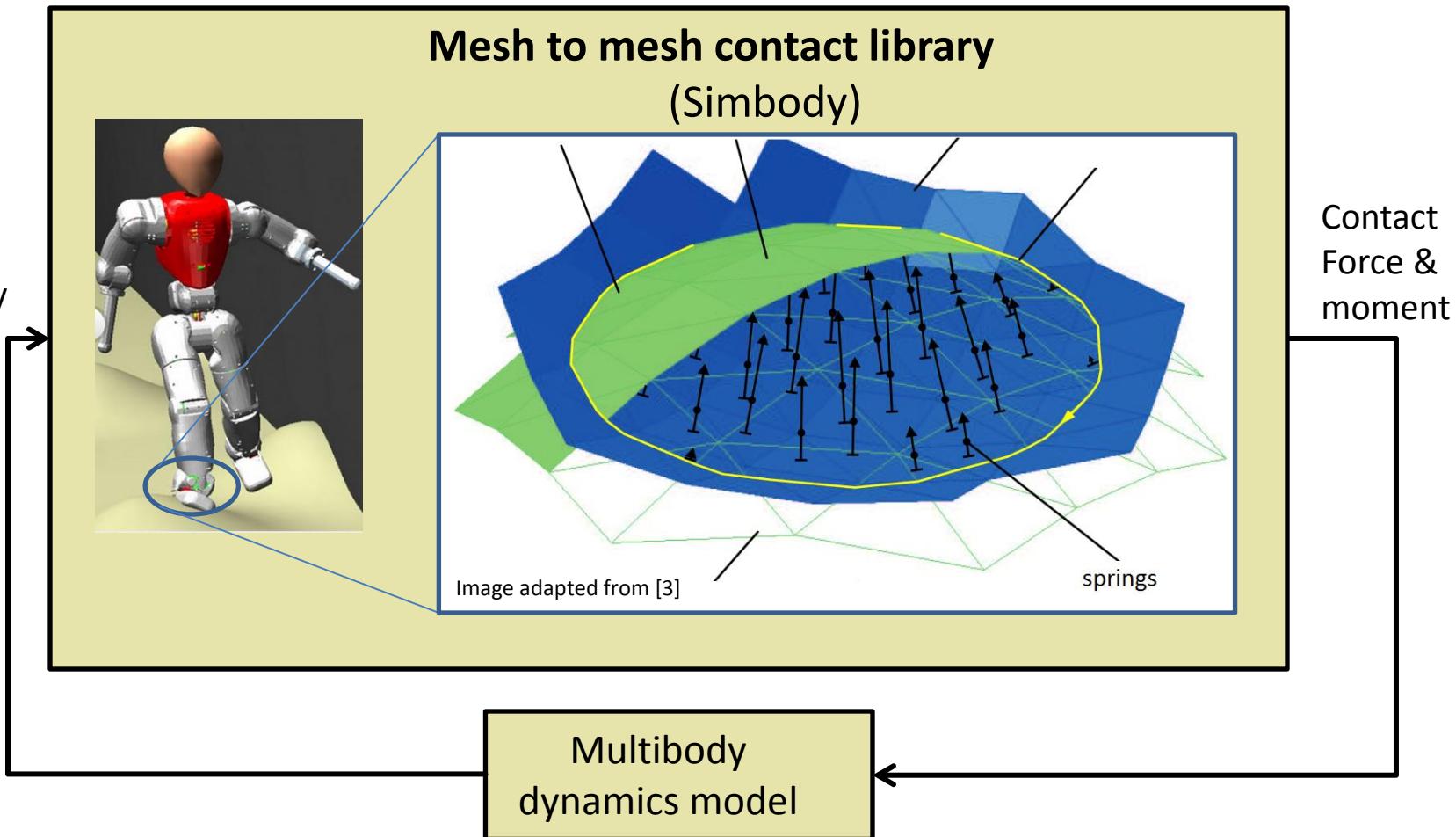
$$J \frac{d^2\theta_m}{dt^2} + b_m \frac{d\theta_m}{dt} = K_t i - \tau_L$$

$$L \frac{di}{dt} + Ri + K_w \theta_m = v(t)$$

$$\tau_L = K_s (\theta_m - \theta) + D_s \left( \frac{d\theta_m}{dt} - \frac{d\theta}{dt} \right)$$



# Contact handling



[3] G. Hippmann, "An algorithm for compliant contact between complexly shaped surfaces in multibody dynamics," *Multibody Dyn.* Jorge AC Ambrosio Ed IDMECIST Lisbon Port. July, vol. 14, 2003

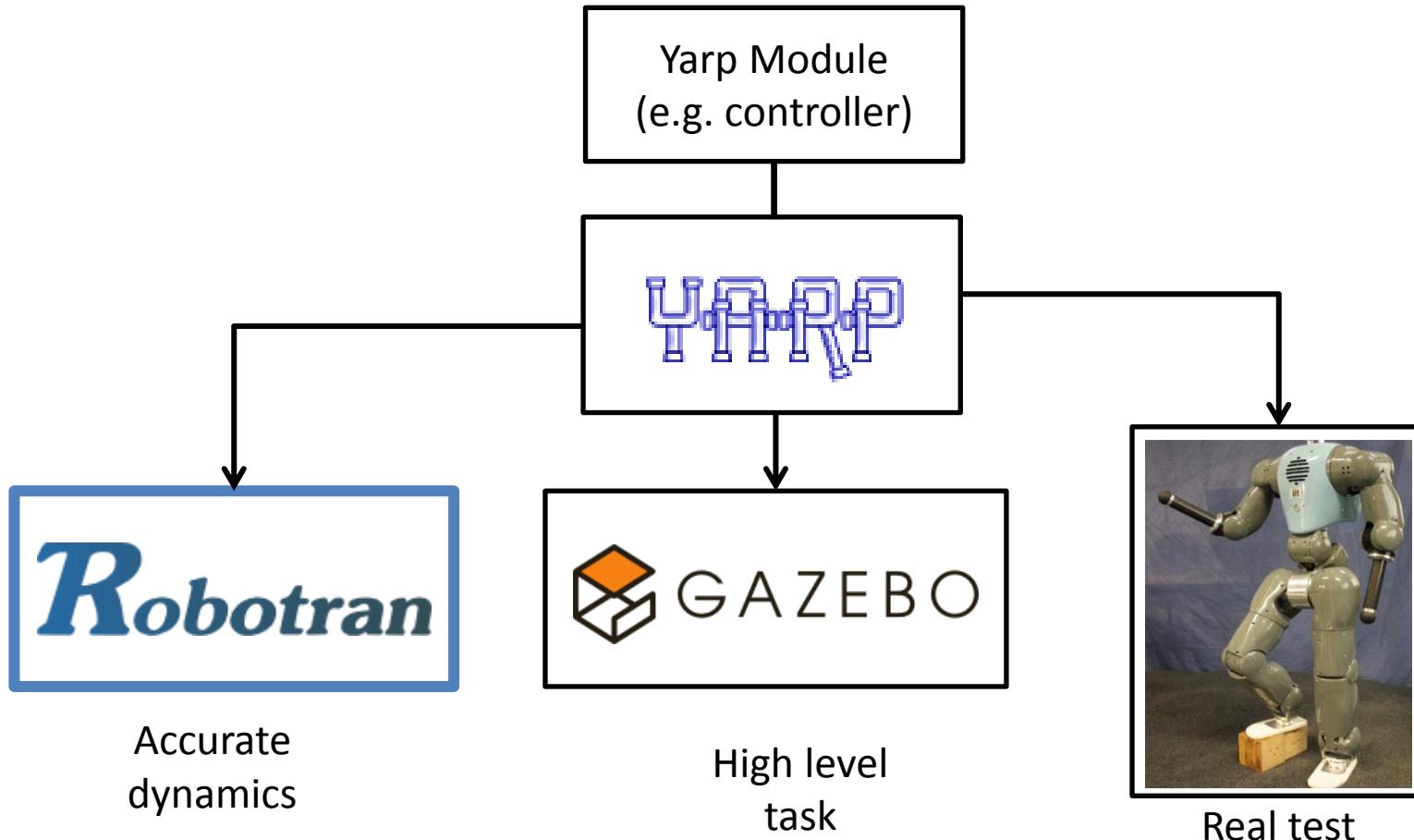
# Realtime features

# Outline

- **Robotran Principles**  
Generation of symbolic routines for dynamics
- **Robotics Simulator**  
Available features for robot simulation
- **Robotran-Yarp interface**  
Running a Yarp module with Robotran

# Robotran-Yarp plugin

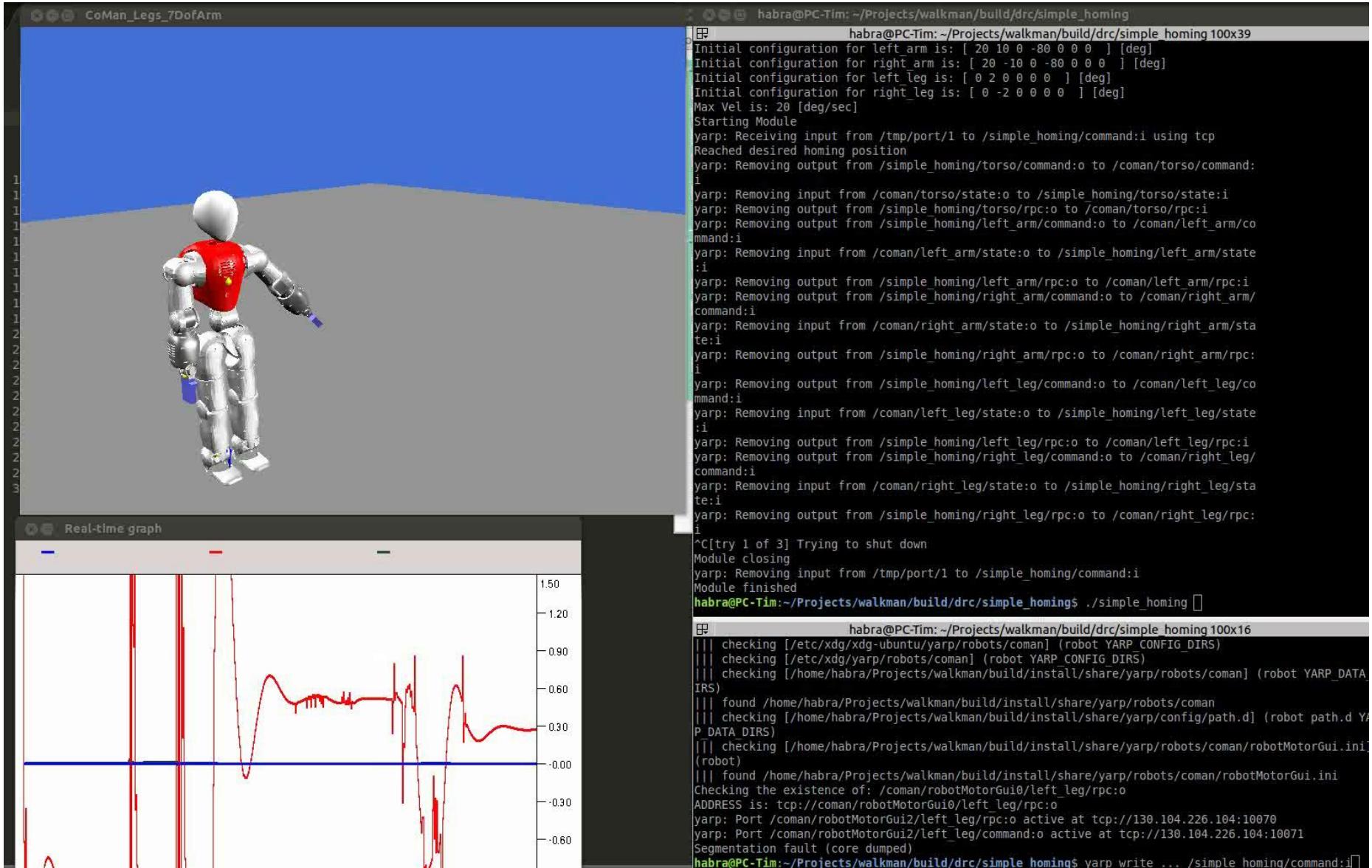
Controller **decoupled** from the robot. Controllers are “robot-platform agnostic”.



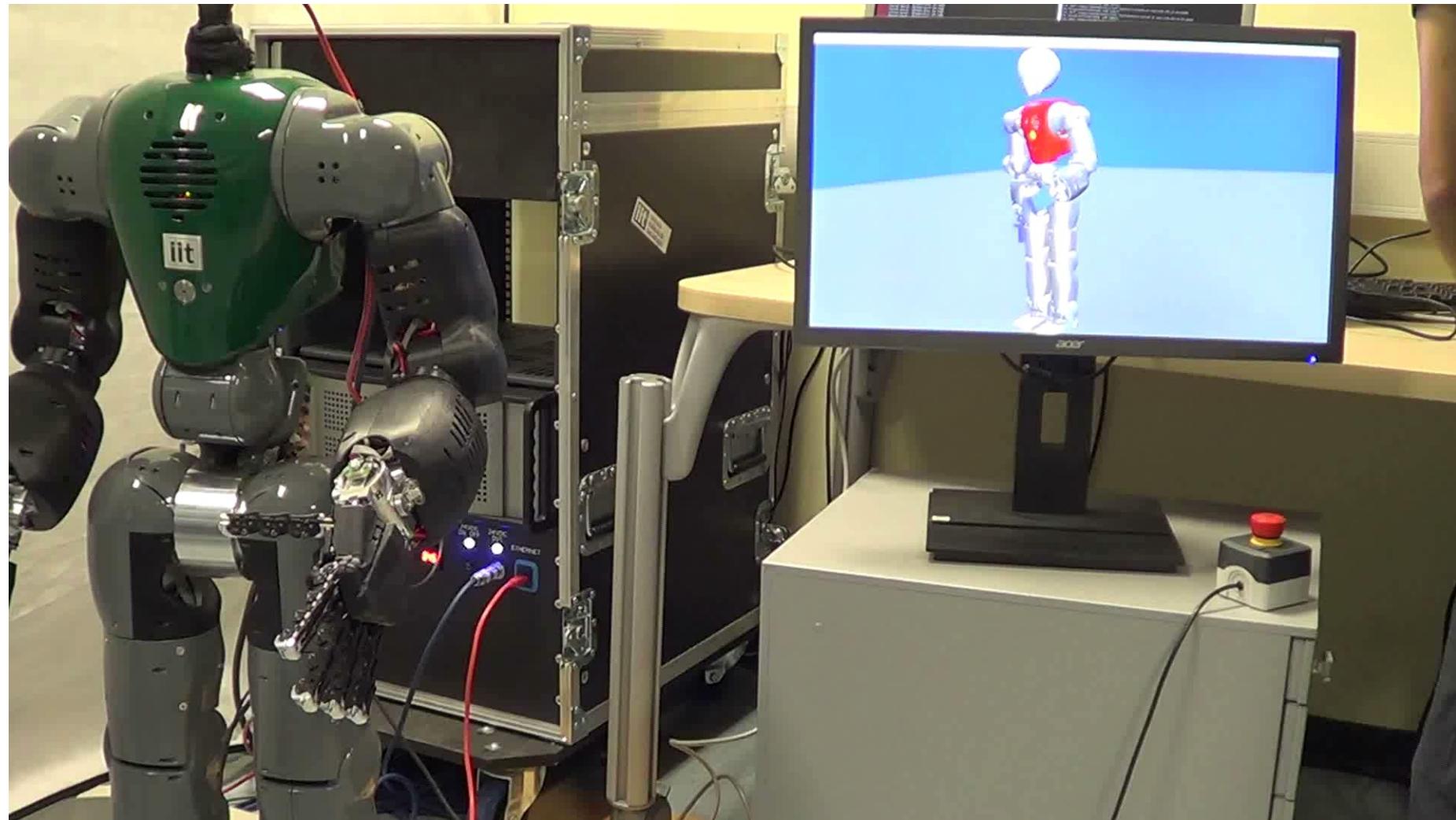
Run module on simulator :

- before robot : fast software development + parallel work
- within robot : internal model (prediction, teleoperation,...)

# Robotran interfaced with Yarp



# Robotran and real CoMan



# Conclusion

A new simulator is available for running Yarp modules

- fast
- accurate dynamics
- open source [https://gitlab.robotran.be/walkman/coman\\_robotran.git](https://gitlab.robotran.be/walkman/coman_robotran.git)
- lightweight
- see also [www.robotran.be](http://www.robotran.be)

## Acknowledgment

IIT : Alberto Cardellino, Lorenzo Natale, Nikos Tsagarakis

UCL : Nicolas Van der Noot, Aubain Verle, Nicolas Docquier, Paul Fisette, Renaud Ronsse

Moscow State Univ : Alexandra Zobova