Overview

- Basic Kinematics
- Joint Motor Control (theory + API + tutorial)
- Cartesian Control (theory)
- Gaze Control (theory)
- Cartesian Control (API + tutorial)
- Gaze Control (API + tutorial)

- Assignment 1 (together)
- Assignment 2 (on your own)
- Experiment with iCub

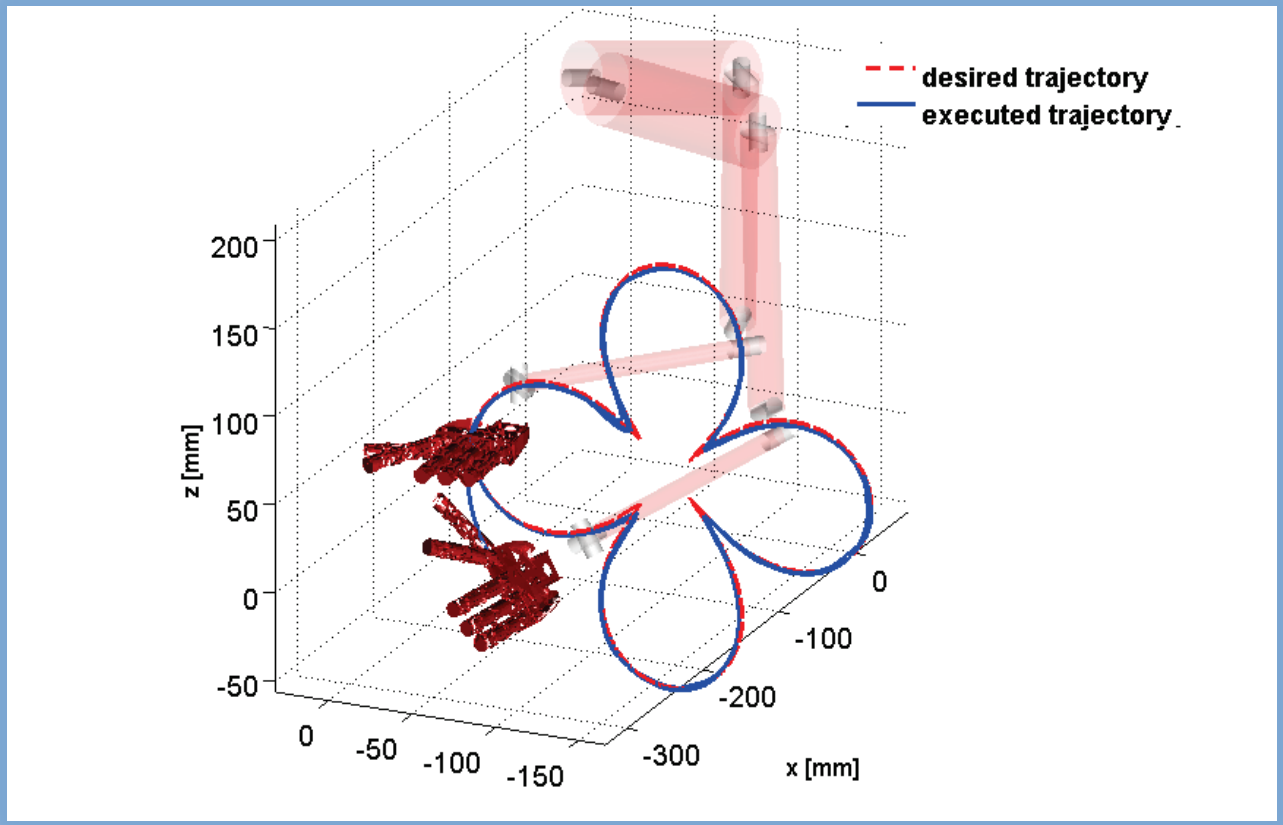
Basic Kinematics (1/3)

Study of properties of motion (position, velocity, acceleration) without considering body inertias and forces

The Problem

$$\begin{cases} \mathbf{x} = \mathbf{f}(\mathbf{q}) \\ \mathbf{q} \in \mathbb{R}^n \\ \mathbf{x} \in \mathbb{R}^6 \end{cases}$$

$$\mathbf{q} \stackrel{?}{=} \mathbf{f}^{-1}(\mathbf{x})$$



Basic Kinematics (2/3)

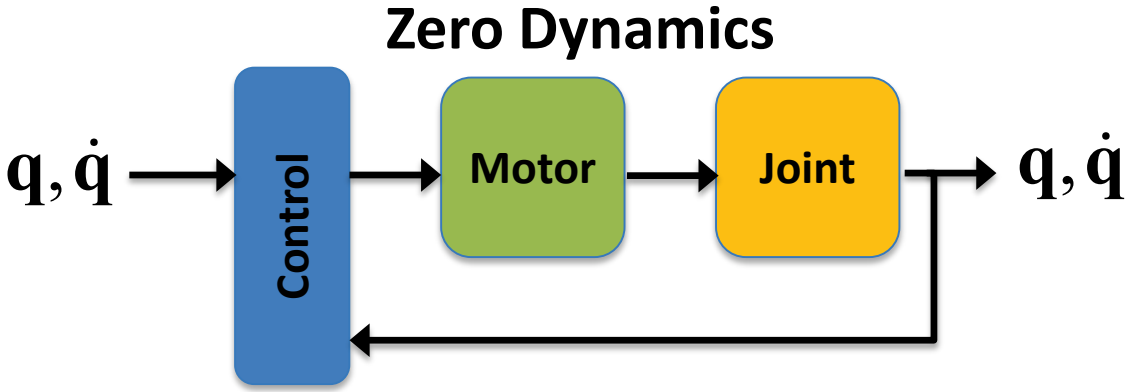
Dynamics – forces, torques, inertias, energy, contact with environment

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{F}_v\dot{\mathbf{q}} + \mathbf{F}_s\,\mathrm{sgn}(\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} - \mathbf{J}^T(\mathbf{q})\mathbf{h}_e$$

VS.

Kinematics – pure motion imposed to the manipulator's joints

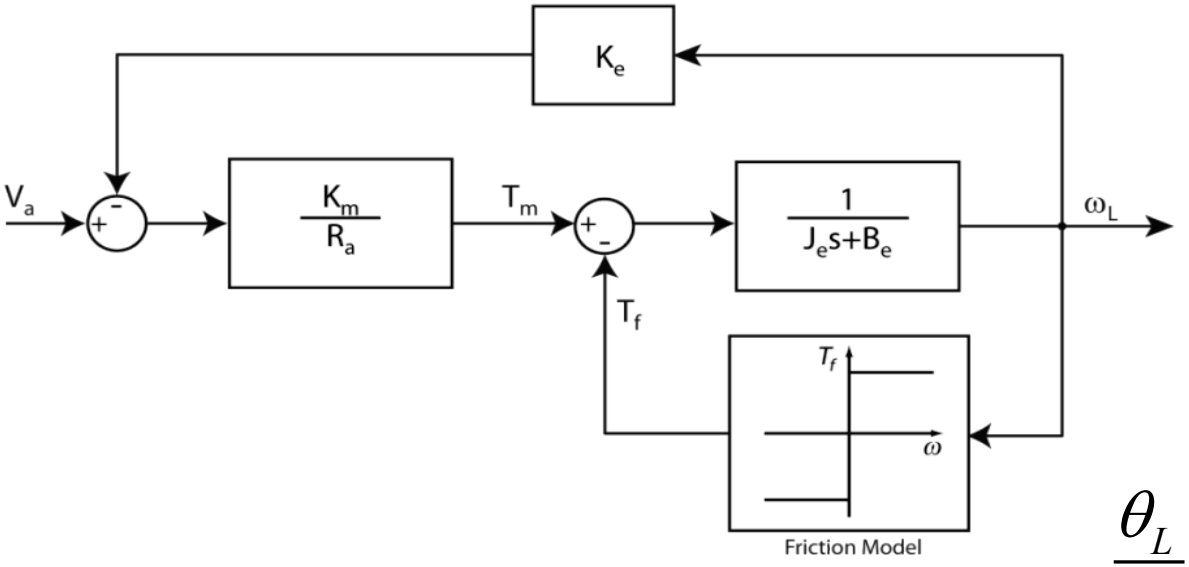
$$\begin{cases} \dot{\mathbf{x}} = \mathbf{J}\dot{\mathbf{q}} \\ \mathbf{J} = \partial\mathbf{f}/\partial\mathbf{q} \end{cases}$$



Basic Kinematics (3/3)

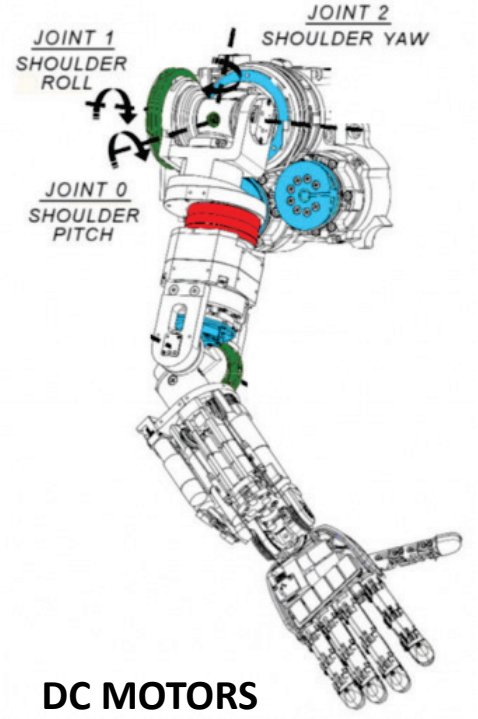
Configuration (Joint) Space Control

You know **q** (joints set-points), you can control directly **motors+joints** to achieve **Zero Dynamics**



$$\frac{\theta_L}{V_a} = \frac{1}{s} \cdot \frac{K}{1+\tau s}$$

Joint Motor Control (1/11)



DC MOTORS

USER CODE @ 100 Hz

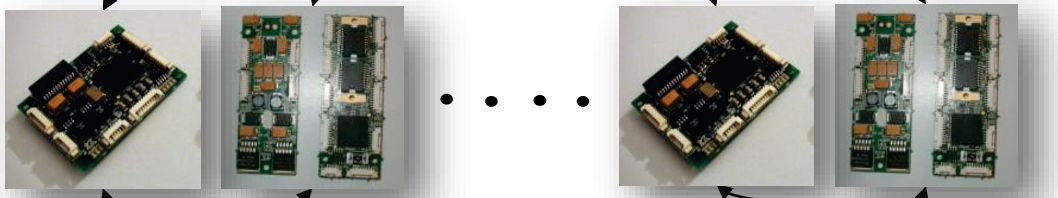
```
ipos->positionMove(poss.data());
while (norm(poss-encs)>1.0) {
   Time::delay(0.1);
   ienc->getEncoders(encs.data());
   cout<<encs.toString()<<endl;
}
```

Gbit Ethernet

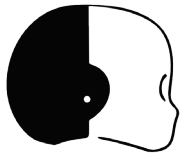
PC104

FIRMWARE @ 1 KHz

CONTROLLERS

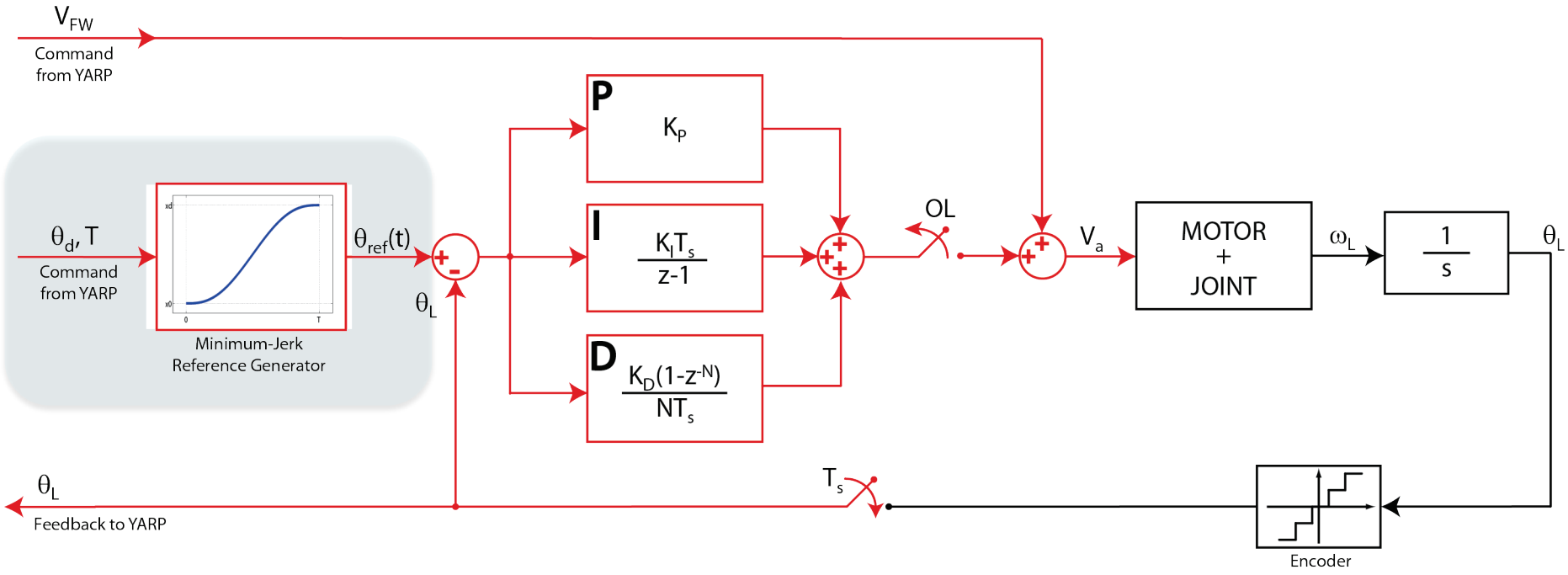


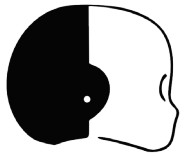
Sensors & Actuators



Joint Motor Control (2/11)

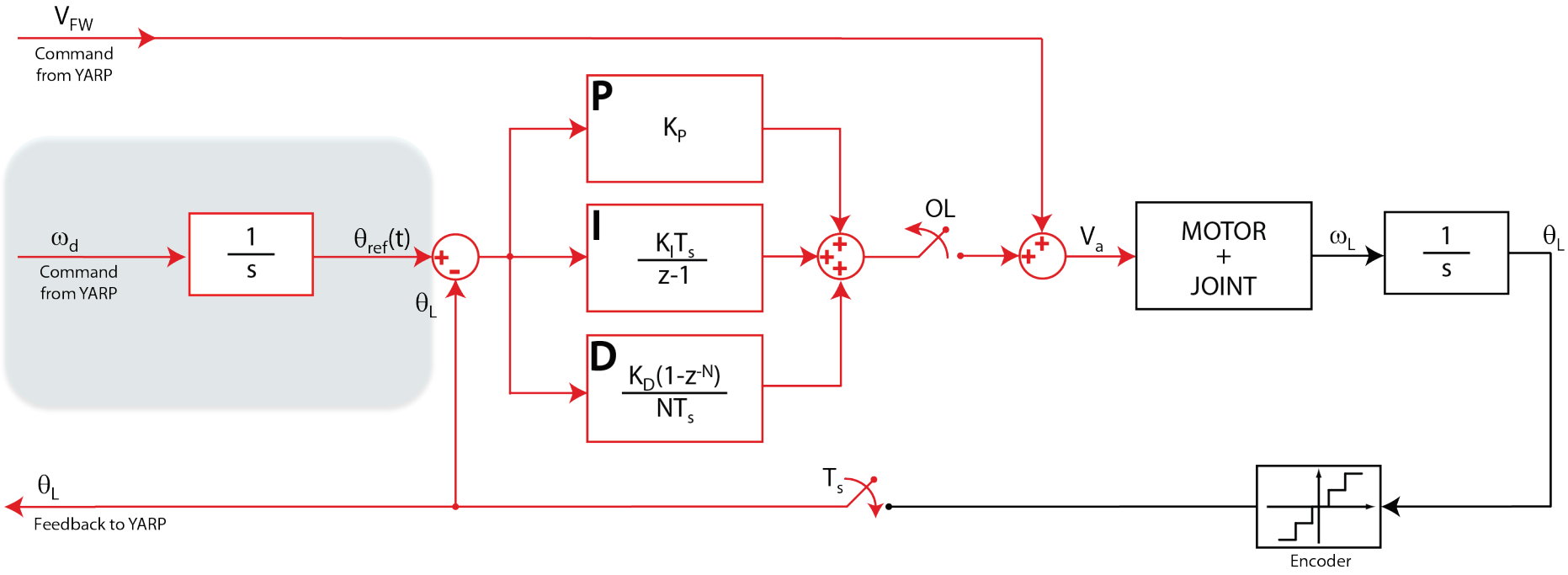
Position Control





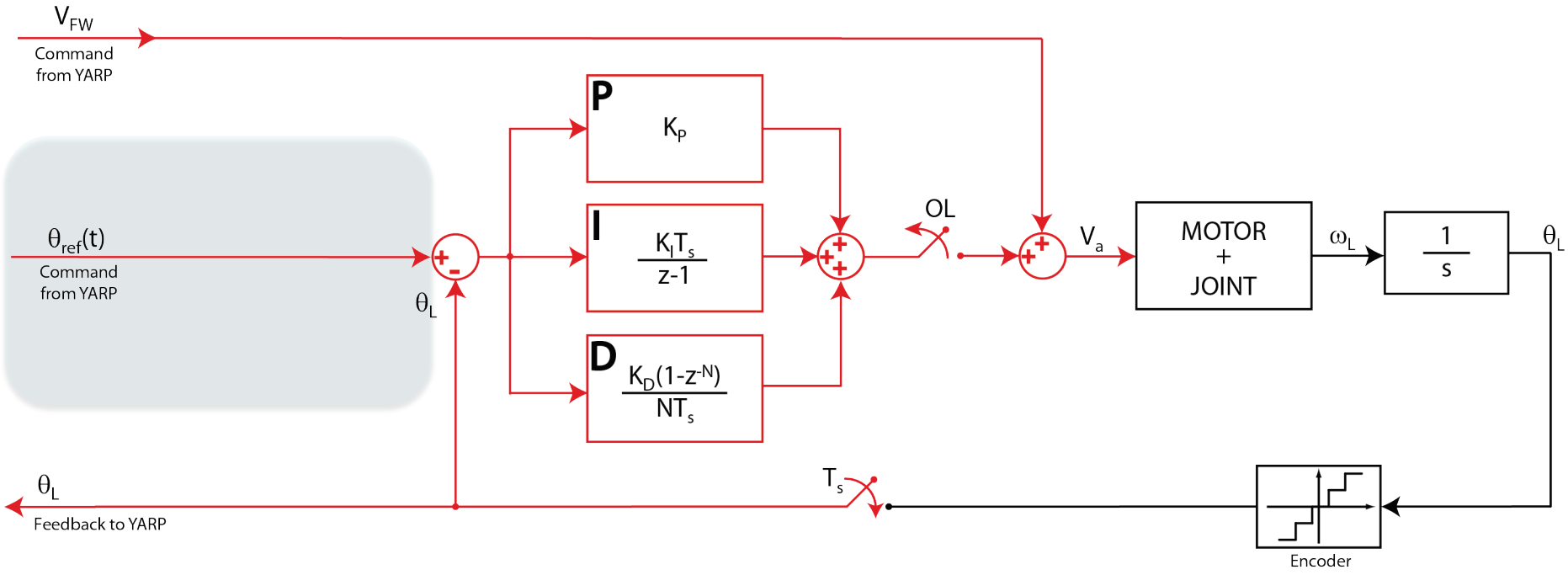
Joint Motor Control (3/11)

Velocity Control



Joint Motor Control (4/11)

Position-Direct Control



Joint Motor Control (5/11)

Motor control interface to the robot is through three ports for each part:

head, **torso**, **legs** and **arms**

/icub/head/rpc:i

/icub/head/state:o

/icub/head/command:i

Joint Motor Control (6/11)

Simplest way to control the robot:

$ yarp rpc /icub/right_arm/rpc:i

>> set acc 1 100.0

>> set vel 1 30.0

>> set pos 1 10.0

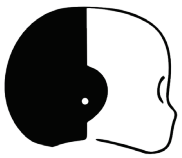
Read encoders:

$ yarp read ... /icub/right_arm/state:o

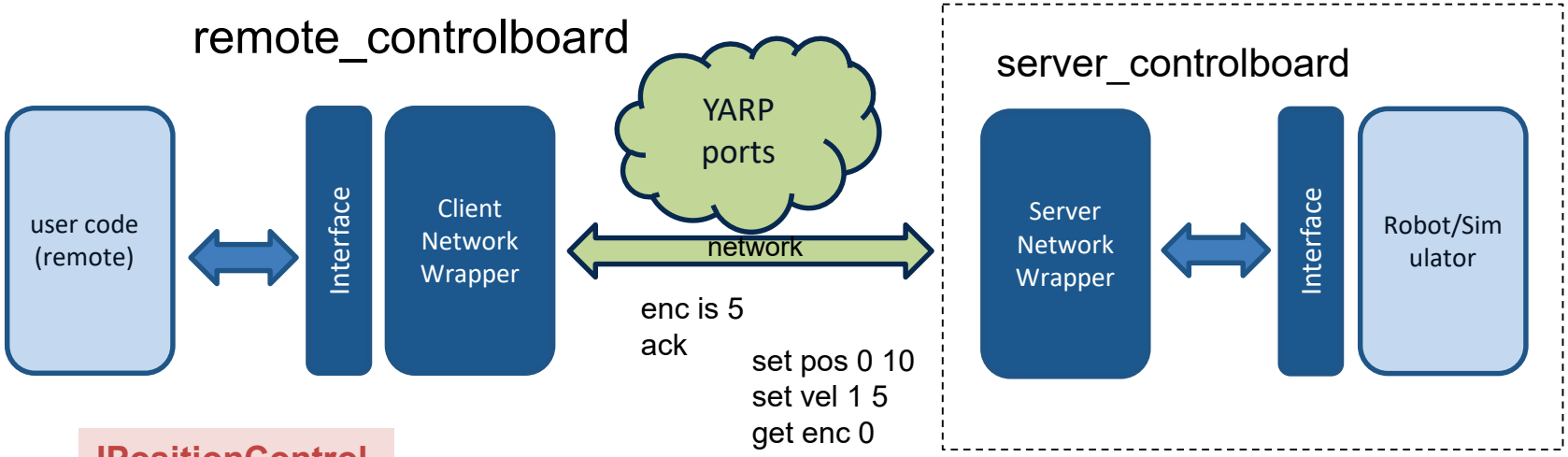
Joint Motor Control (7/11)

SETTING/GETTING JOINT POSITION via YARP PORTS

```
Bottle cmd;
cmd.addVocab(Vocab::encode("set"));
cmd.addVocab(Vocab::encode("pos"));
cmd.addInt(1);
cmd.addDouble(-10.0);
commands.write(cmd);

cmd.clear();
cmd.addVocab(Vocab::encode("get"));
cmd.addVocab(Vocab::encode("enc"));
cmd.addInt(1);
Bottle rep;
commands.write(cmd,rep);
cout<<rep.toString();
```



Joint Motor Control (8/11)



IPositionControl
IVelocityControl
IEncoders
…

/client/right_arm/rpc:i
/client/right_arm/state:i
/client/right_arm/command:o

/icub/right_arm/rpc:i
/icub/right_arm/state:i
/icub/right_arm/command:o

Joint Motor Control (9/11)

OPENING THE JOINT INTERFACE

```
#include <yarp/dev/all.h>
Property option;

option.put("device",“remote_controlboard");
option.put("remote","/icub/right_arm");
option.put("local","/client/right_arm");

PolyDriver clientJointCtrl(option);

IEncoders *ienc=NULL;
IControlMode *imod=NULL;
IPositionControl *ipos=NULL;
if (clientJointCtrl.isValid()) {
   clientJointCtrl.view(ienc);
   clientJointCtrl.view(imod);
   clientJointCtrl.view(ipos);
}
```

Joint Motor Control (10/11)

This will create the following ports:

/client/right_arm/state:i
/client/right_arm/command:o
/client/right_arm/rpc:o

```
#include <yarp/dev/all.h>
Property option;

option.put("device",“remote_controlboard");
option.put("remote","/icub/right_arm");
option.put("local","/client/right_arm");

PolyDriver clientJointCtrl(option);

IEncoders *ienc=NULL;
IControlMode *imod=NULL;
IPositionControl *ipos=NULL;
if (clientJointCtrl.isValid()) {
   clientJointCtrl.view(ienc);
   clientJointCtrl.view(imod);
   clientJointCtrl.view(ipos);
}
```

And will automatically connect them to the server, on the following (remote):

/icub/right_arm/state:o
/icub/right_arm/command:i
/icub/right_arm/rpc:i

Joint Motor Control (11/11)

ALMOST WORKING EXAMPLE

```
int nj; pos->getAxes(&nj);
VectorOf<int> modes(nj);
Vector accs(nj),vels(nj),poss(nj),encs(nj);

// prepare here our goals in terms of:
// accelerations [deg/s^2], speeds [deg/s]
// and set-points [deg]

imod->setControlModes(modes.data());
ipos->setRefAccelerations(accs.data());
ipos->setRefSpeeds(vels.data());
ipos->positionMove(poss.data());

while (norm(poss-encs)>1.0) {
   Time::delay(0.1);
   ienc->getEncoders(encs.data());
   cout<<encs.toString()<<endl;
}
```

The Cartesian Controller (1/11)

Operational (Cartesian) Space Control

You know **x** (3D/6D points), you cannot control directly the motors, you have to solve the **Inverse Kinematics (IK) problem** beforehand.

Jacobian Transpose

$$\dot{\mathbf{q}} = \mathbf{J}^T \mathbf{K} \mathbf{e}, \quad \mathbf{e} = \mathbf{x}_d - \mathbf{x}_e$$

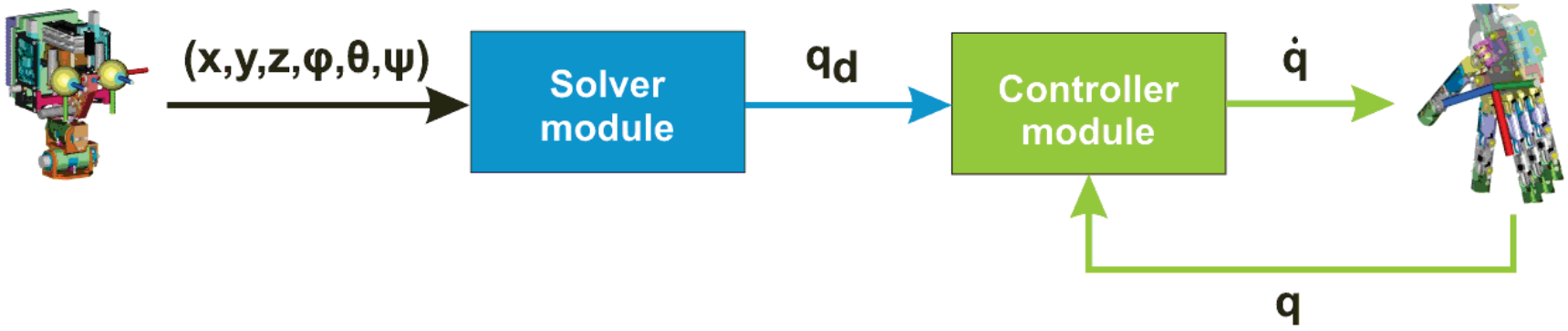
Jacobian Pseudoinverse

$$\dot{\mathbf{q}} = \mathbf{J}^\dagger \mathbf{K} \mathbf{e} + \left(\mathbf{I} - \mathbf{J}^\dagger \mathbf{J} \right) \dot{\mathbf{q}}_0, \quad \mathbf{J}^\dagger = \mathbf{J}^T \left(\mathbf{J} \mathbf{J}^T \right)^{-1}$$

Damped Least-Squares

$$\dot{\mathbf{q}} = \mathbf{J}^* \mathbf{K} \mathbf{e}, \quad \mathbf{J}^* = \mathbf{J}^T \left(\mathbf{J} \mathbf{J}^T + \lambda^2 \mathbf{I} \right)^{-1}$$

The Cartesian Controller (2/11)



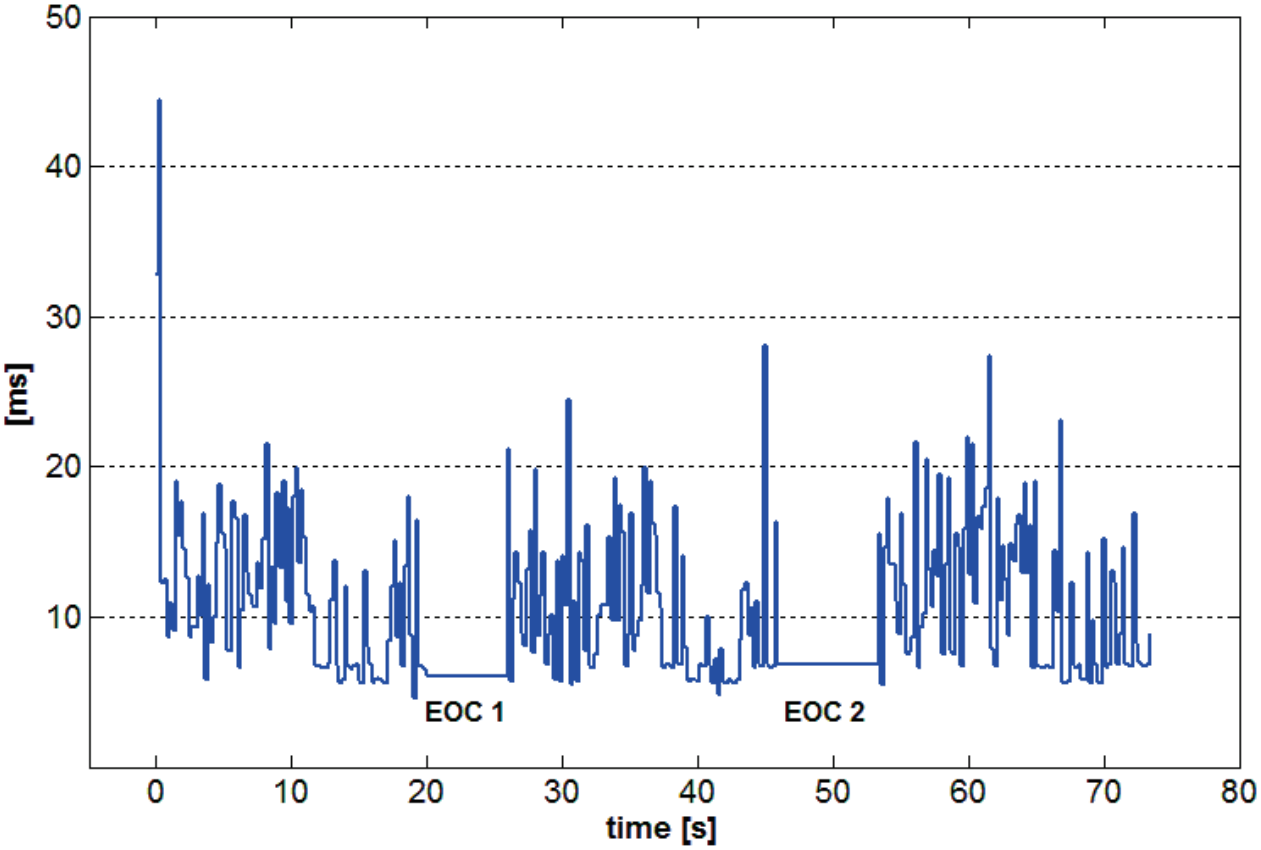
The Cartesian Controller (3/11)

$$\tilde{q}_d = \arg\min_{q \in \mathbb{R}^n} \left(\left\| \alpha_d - K_\alpha(q) \right\|^2 + \lambda \cdot (q_{\text{rest}} - q)^T W (q_{\text{rest}} - q) \right)$$

$$\text{s.t.} \begin{cases} \left\| x_d - K_x(q) \right\|^2 < \varepsilon \\ q_L < q < q_U \\ \text{other obstacles ...} \end{cases}$$

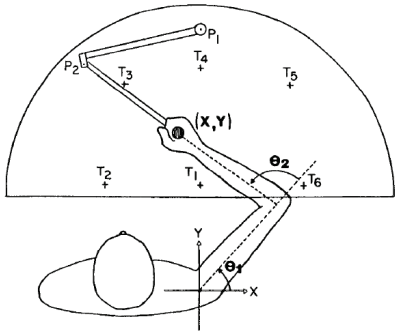
- **Quick convergence**: real-time compliant, < **20 ms**
- **Scalability**: *n* can be high and set on the fly
- **Singularities handling**: no Jacobian inversion
- **Joints bound handling**: no explicit boundary functions
- **Tasks hierarchy**: no use of null space
- **Complex constraints**: intrinsically nonlinear

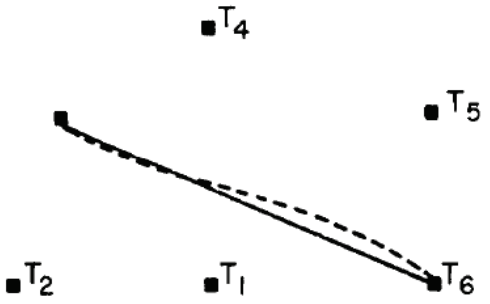
The Cartesian Controller (4/11)

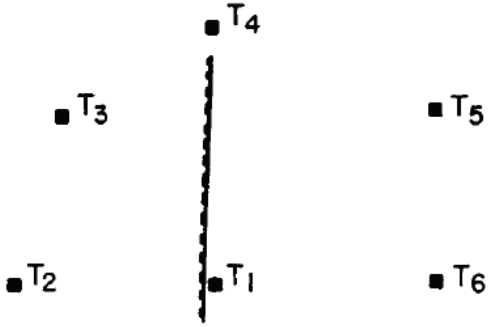


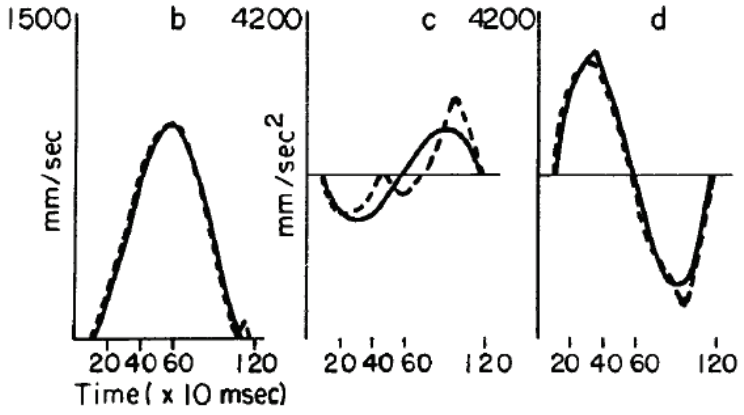
Solver running @ 33 Hz on multicore Intel (R) Xeon 2.27 GHz

The Cartesian Controller (5/11)

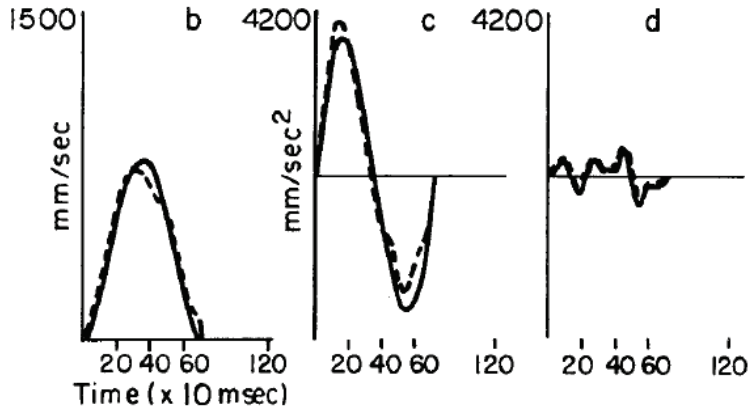






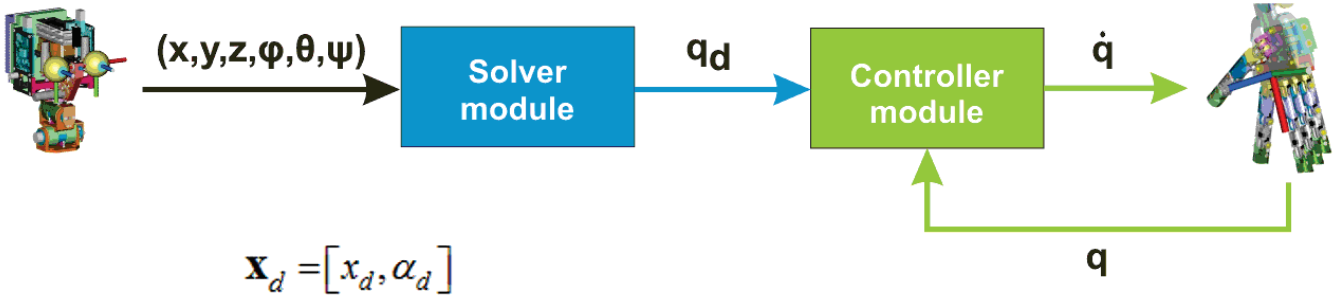


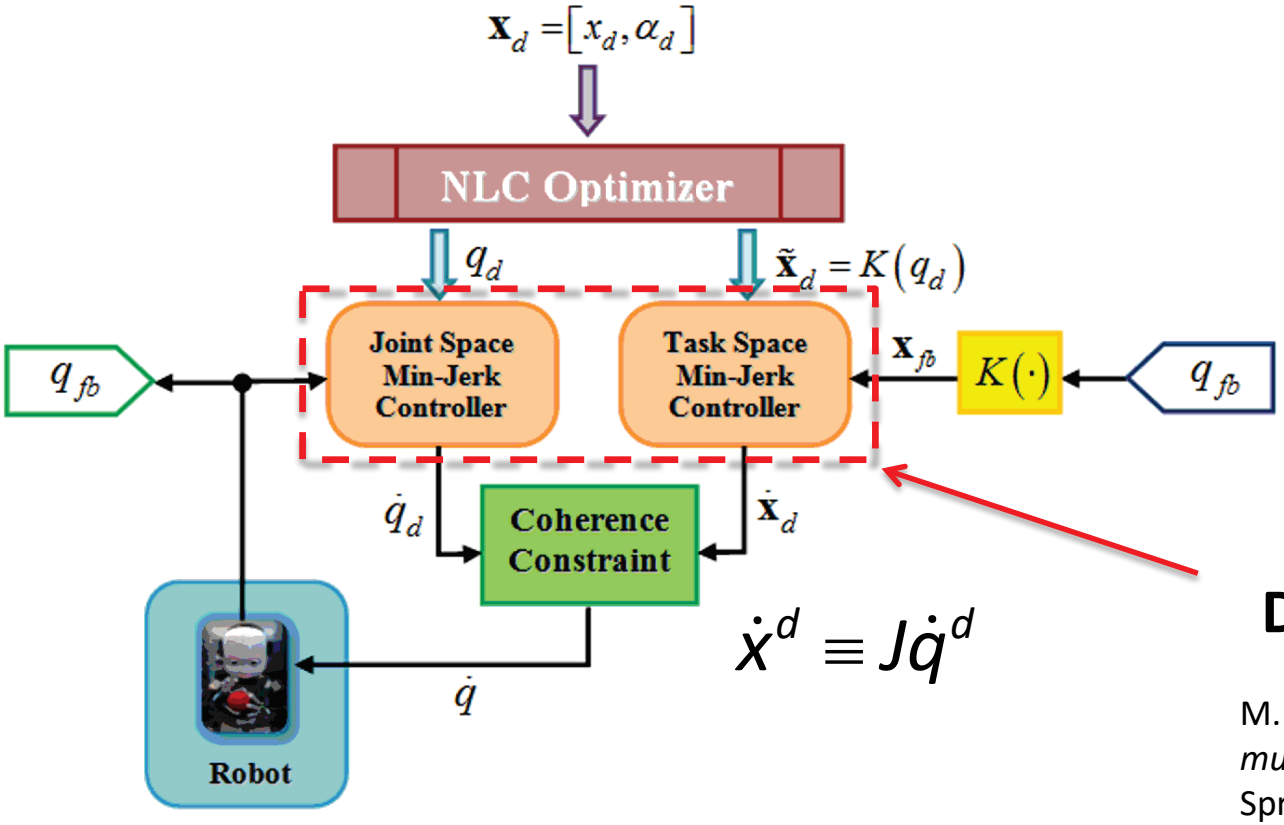
A



B

The Cartesian Controller (6/11)





Multi Referential Dynamical Systems

M. Hersch, A.G. Billard, "*Reaching with multi-referential dynamical systems*", Springer-Verlag.

The Cartesian Controller (7/11)

MJ closed-loop:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\frac{60}{(T-t)^3} & -\frac{36}{(T-t)^2} & -\frac{9}{T-t} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{60}{(T-t)^3} \end{bmatrix} x_d$$

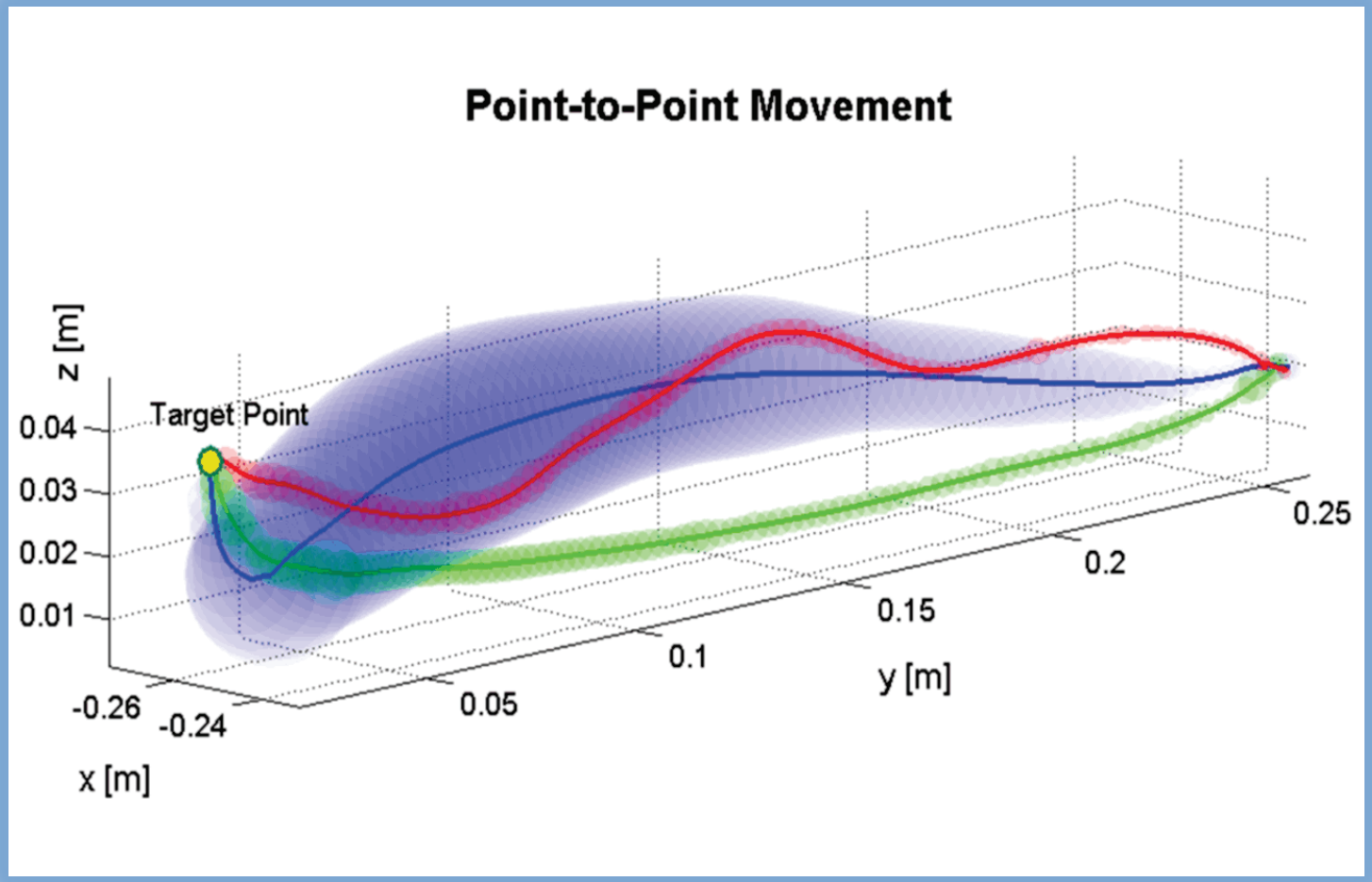
$$x(t) = \left(C_1 + C_2 t + C_3 t^2\right) \cdot \exp(\lambda \cdot t) + x_d$$

$$\lambda^* = \arg\min_{\lambda \in \mathbb{R}} \left(\int_0^\infty \dddot{x}^2(\tau)\, d\tau \right) \quad \text{s.t.} \begin{cases} \lambda < 0 \\ x(1) \geq 1 - \varepsilon \end{cases}$$

LTI min-jerk approximation:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dddot{x} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ \frac{a(\lambda)}{T^3} & \frac{b(\lambda)}{T^2} & \frac{c(\lambda)}{T} \end{bmatrix}}_{A} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ -\frac{a(\lambda)}{T^3} \end{bmatrix}}_{B} x_d$$

The Cartesian Controller (8/11)

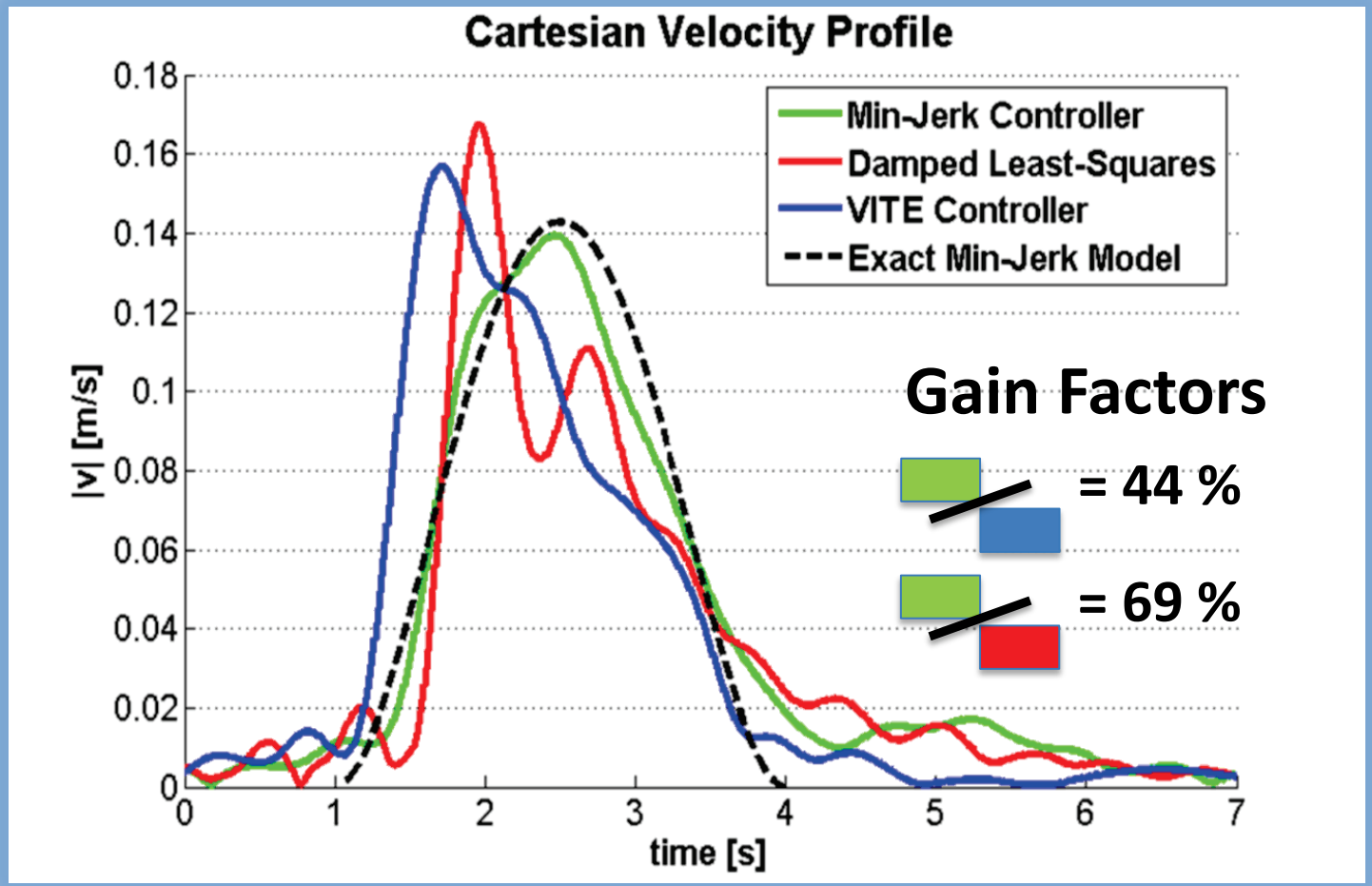


Min-Jerk

DLS

VITE

The Cartesian Controller (9/11)

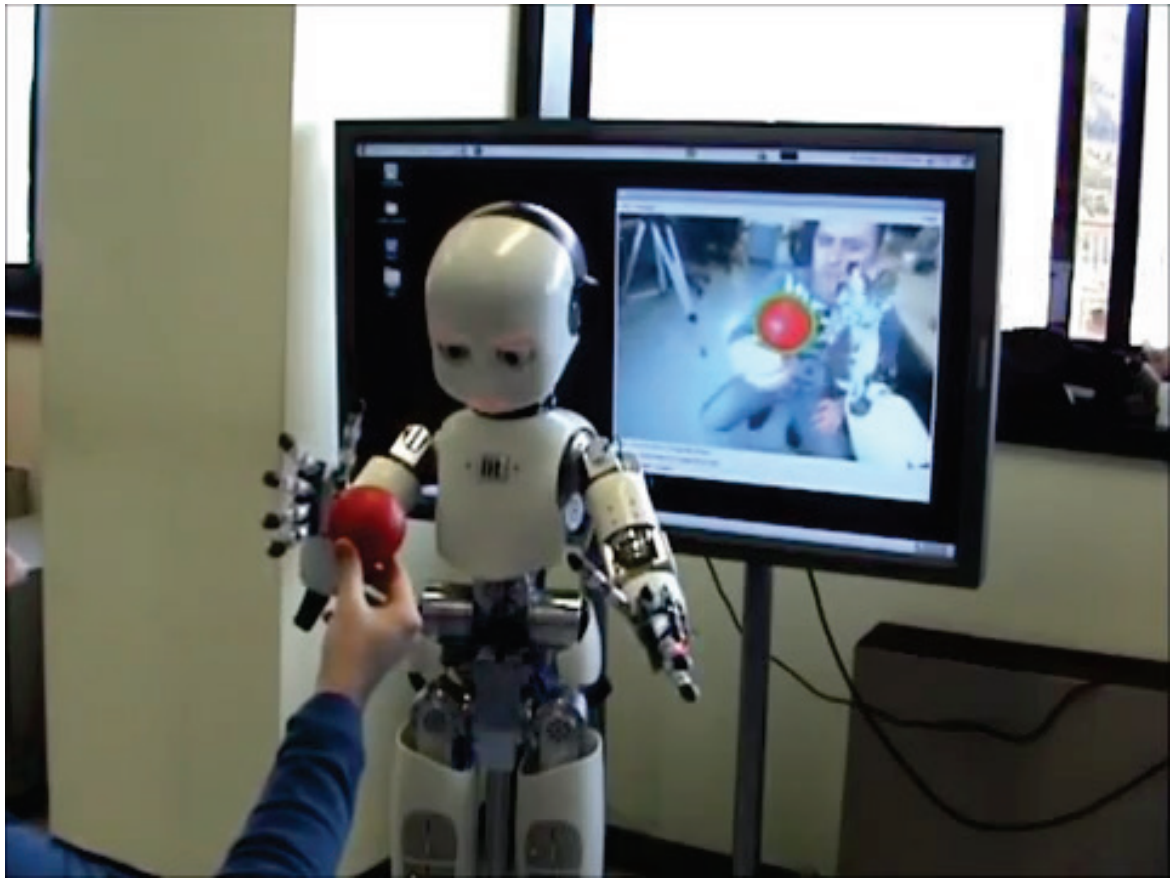


Min-Jerk

DLS

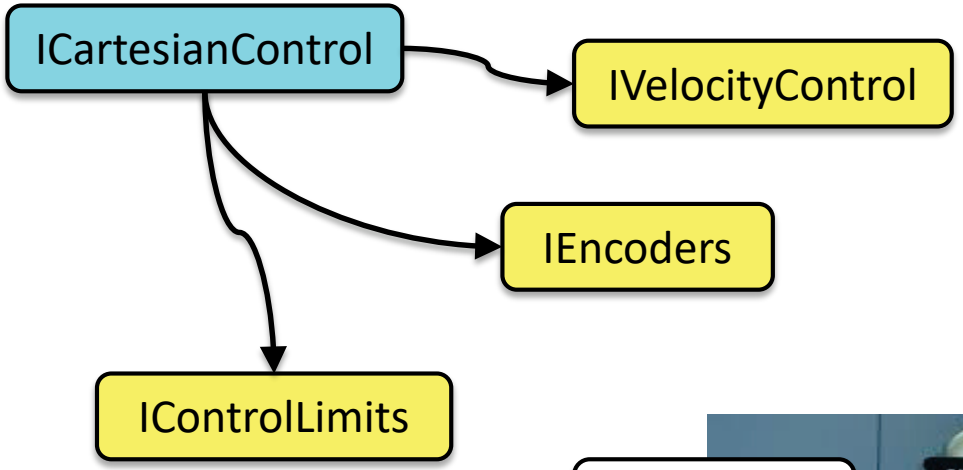
VITE

The Cartesian Controller (10/11)

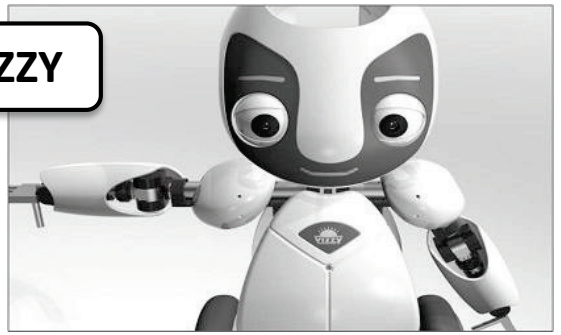


http://y2u.be/LMGSok5tN4A

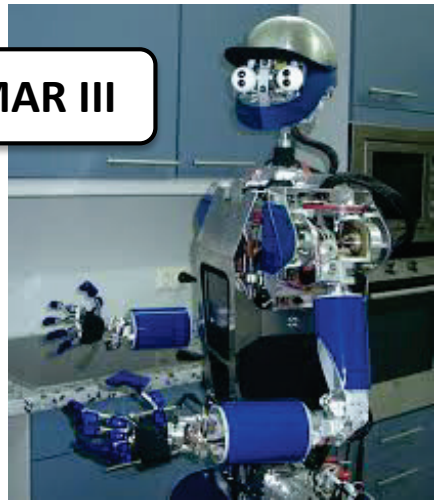
The Cartesian Controller (11/11)

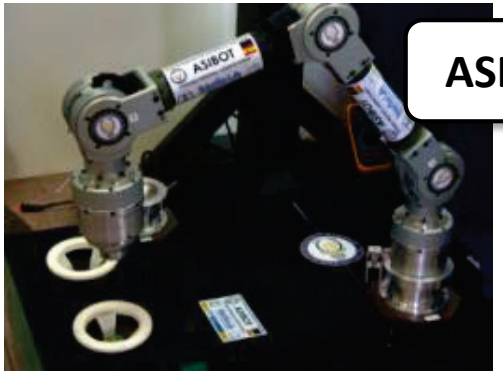


VIZZY



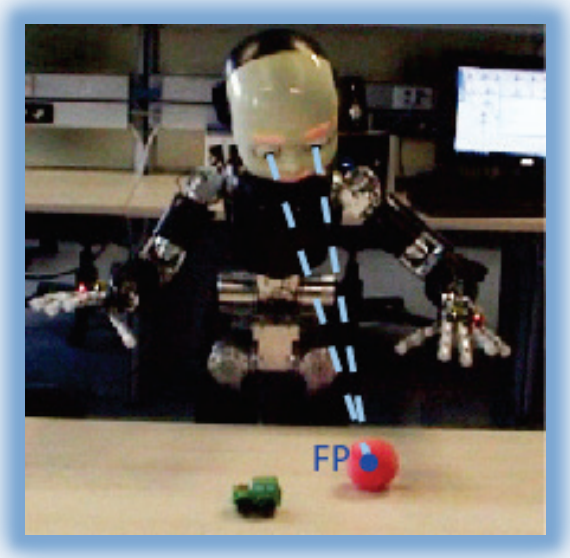
ARMAR III

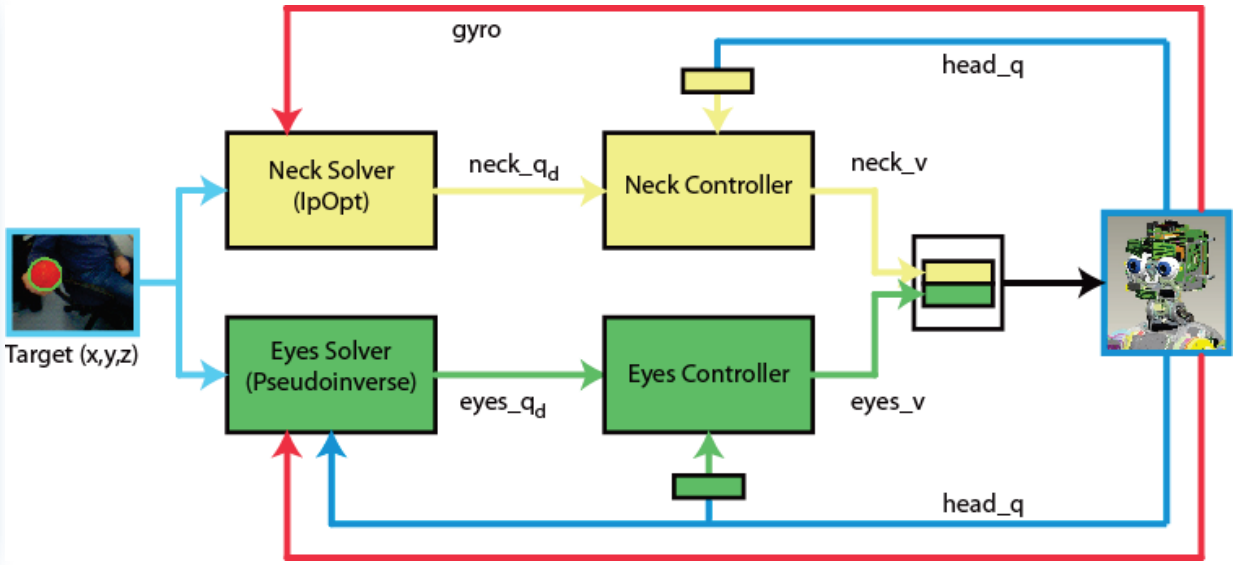




ASIBOT

The Gaze Controller (1/7)

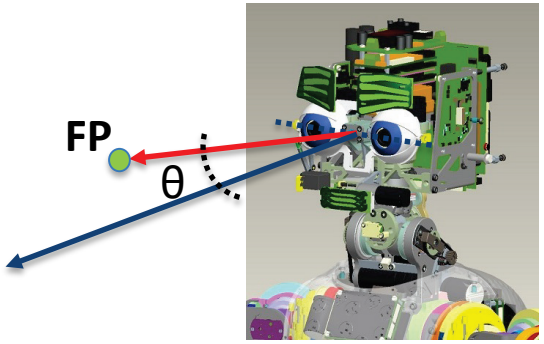




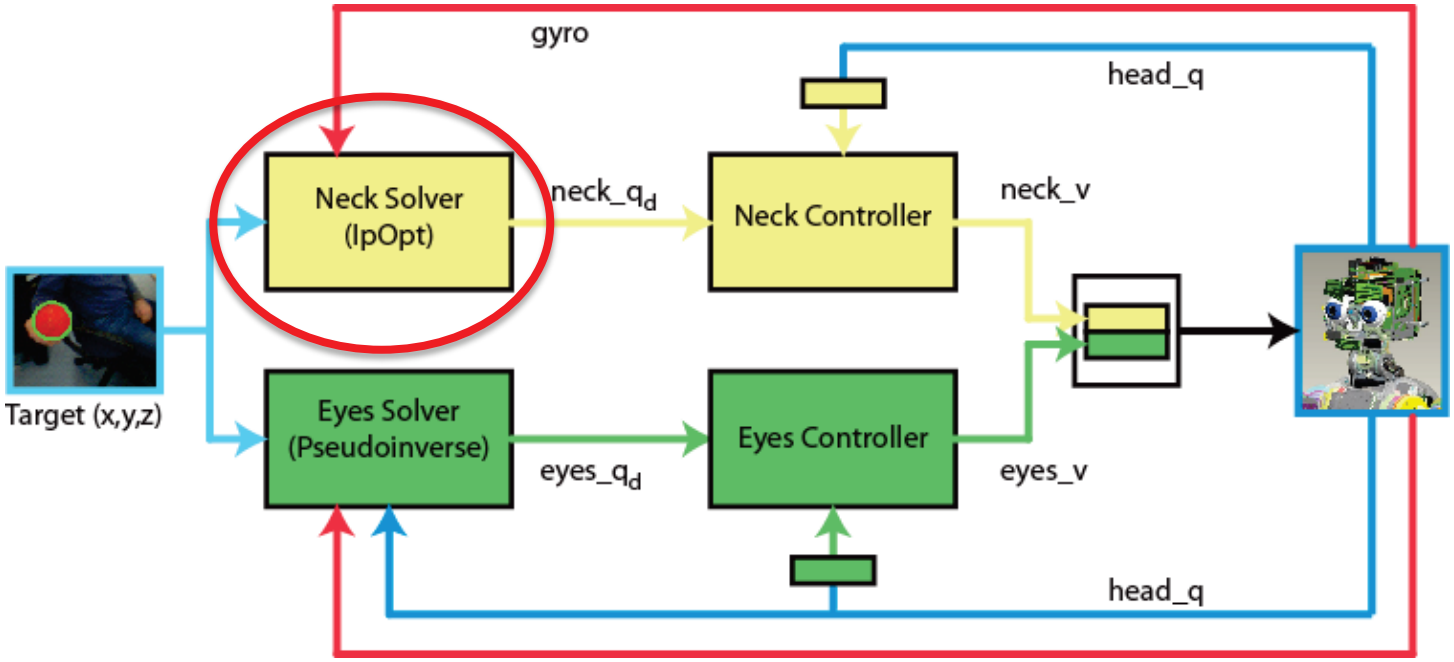
Yet another Cartesian Controller: **reuse ideas** …

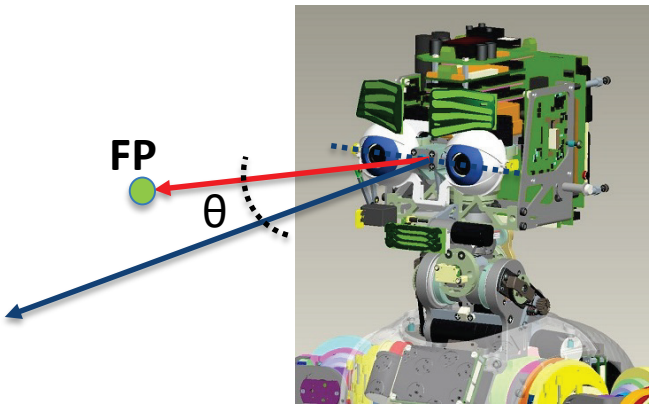
Then, apply easy transformations from Cartesian to …

1. Egocentric angular space
2. Image planes (mono and stereo)



The Gaze Controller (2/7)

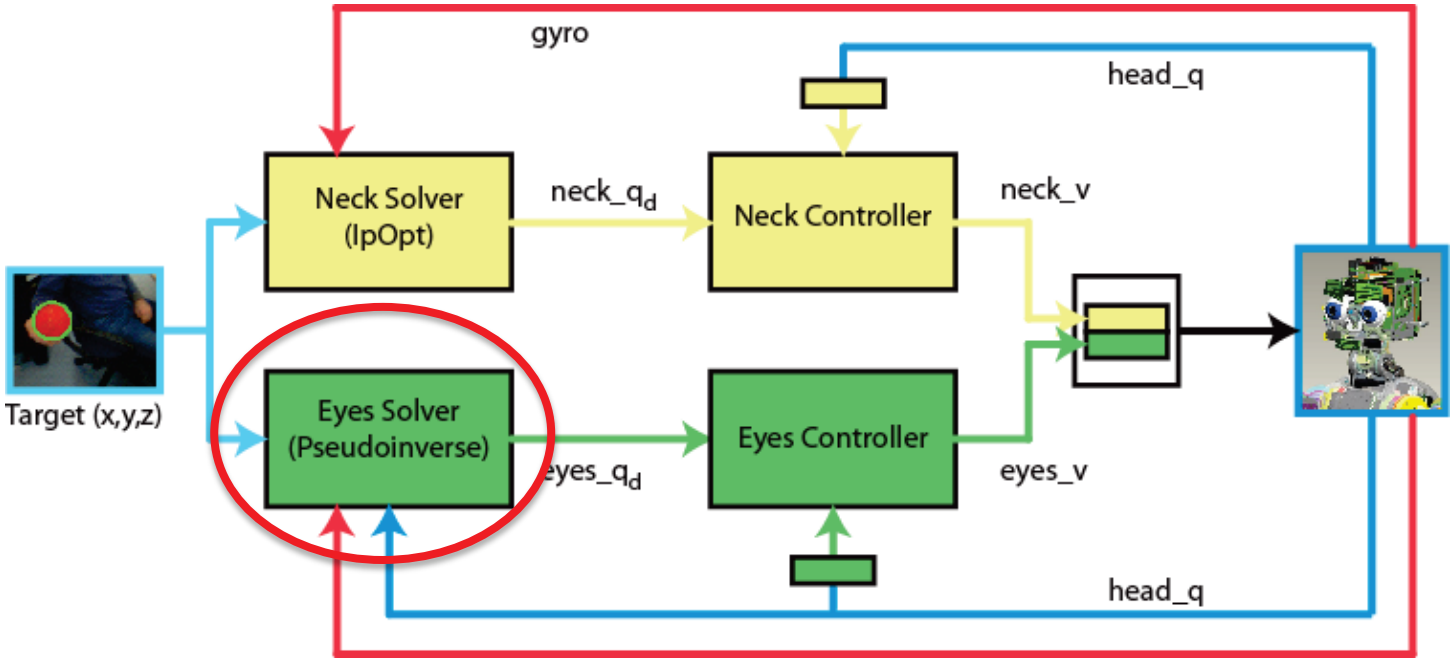




$$q^*_{\text{neck}} = \arg \min_{q_{\text{neck}} \in \mathbb{R}^3} \left\| q_{\text{rest}} - q_{\text{neck}} \right\|^2$$

$$\text{s.t.} \begin{cases} \cos\left(\theta\left(q_{\text{neck}}\right)\right) > 1 - \varepsilon \\ q_{\text{neck}_L} < q_{\text{neck}} < q_{\text{neck}_U} \end{cases}$$

The Gaze Controller (3/7)

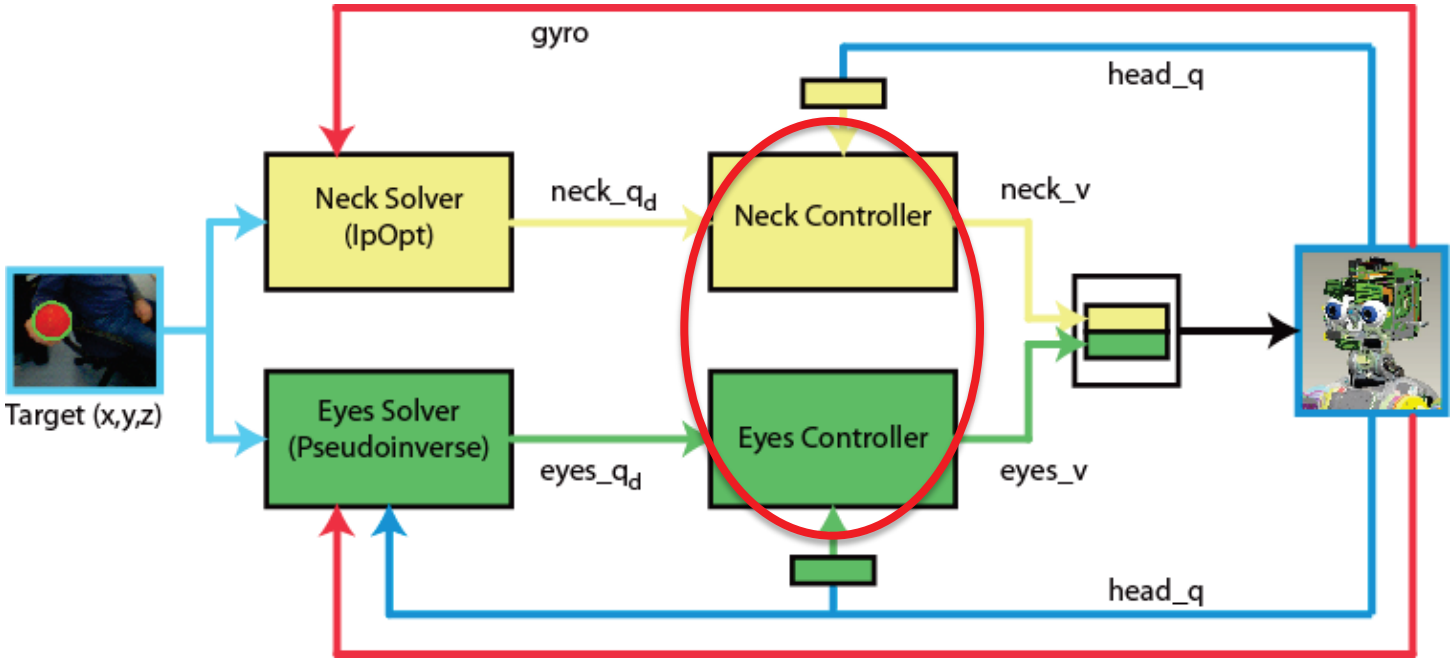


$$q^*_{\text{eyes}} = \arg\min_{q_{\text{eyes}} \in \mathbb{R}^3} \left\| FP_d - K_{FP}\left(q_{\text{eyes}}\right) \right\|^2$$

$$q_{\text{eyes}_{t+1}} = q_{\text{eyes}_t} + \Delta T \left(G \cdot J^{\#} \cdot \left(FP_d - K_{FP}\left(q_{\text{eyes}_t}\right) \right) - \dot{q}_c \right)$$

Gyro

The Gaze Controller (4/7)



Retain Controllers Laws

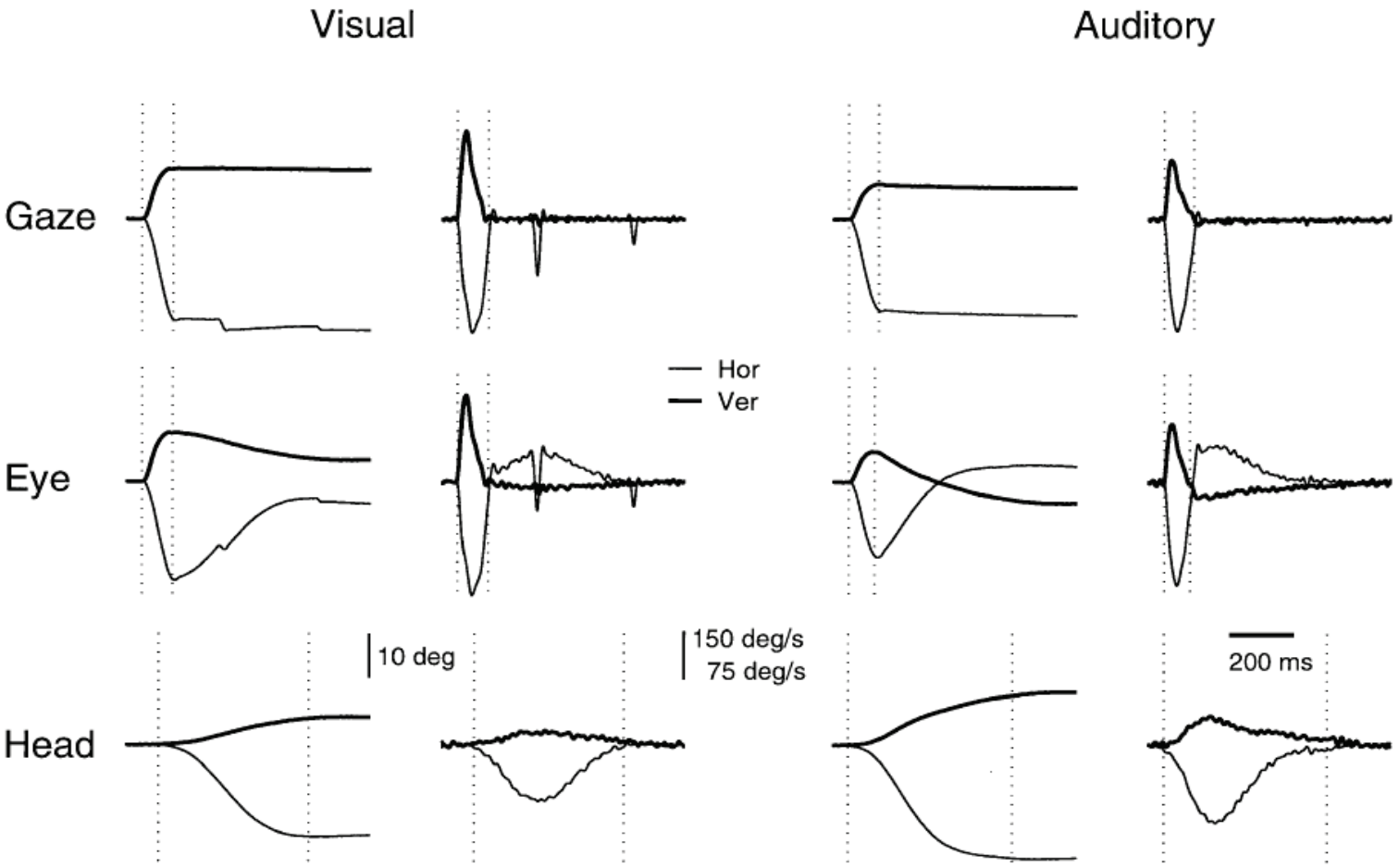
$$\frac{\dot{q}_{\text{neck}}}{q_{\text{neck}_d} - q_{\text{neck}}} = \frac{-a/T_{\text{neck}}}{s^2 - \left(c/T_{\text{neck}}^3\right)s - b/T_{\text{neck}}^2}$$

$$\frac{\dot{q}_{\text{eyes}}}{q_{\text{eyes}_d} - q_{\text{eyes}}} = \frac{-a/T_{\text{eyes}}}{s^2 - \left(c/T_{\text{eyes}}^3\right)s - b/T_{\text{eyes}}^2}$$

Feed Forward term delivered with low-level Position Control to implement **fast saccades**

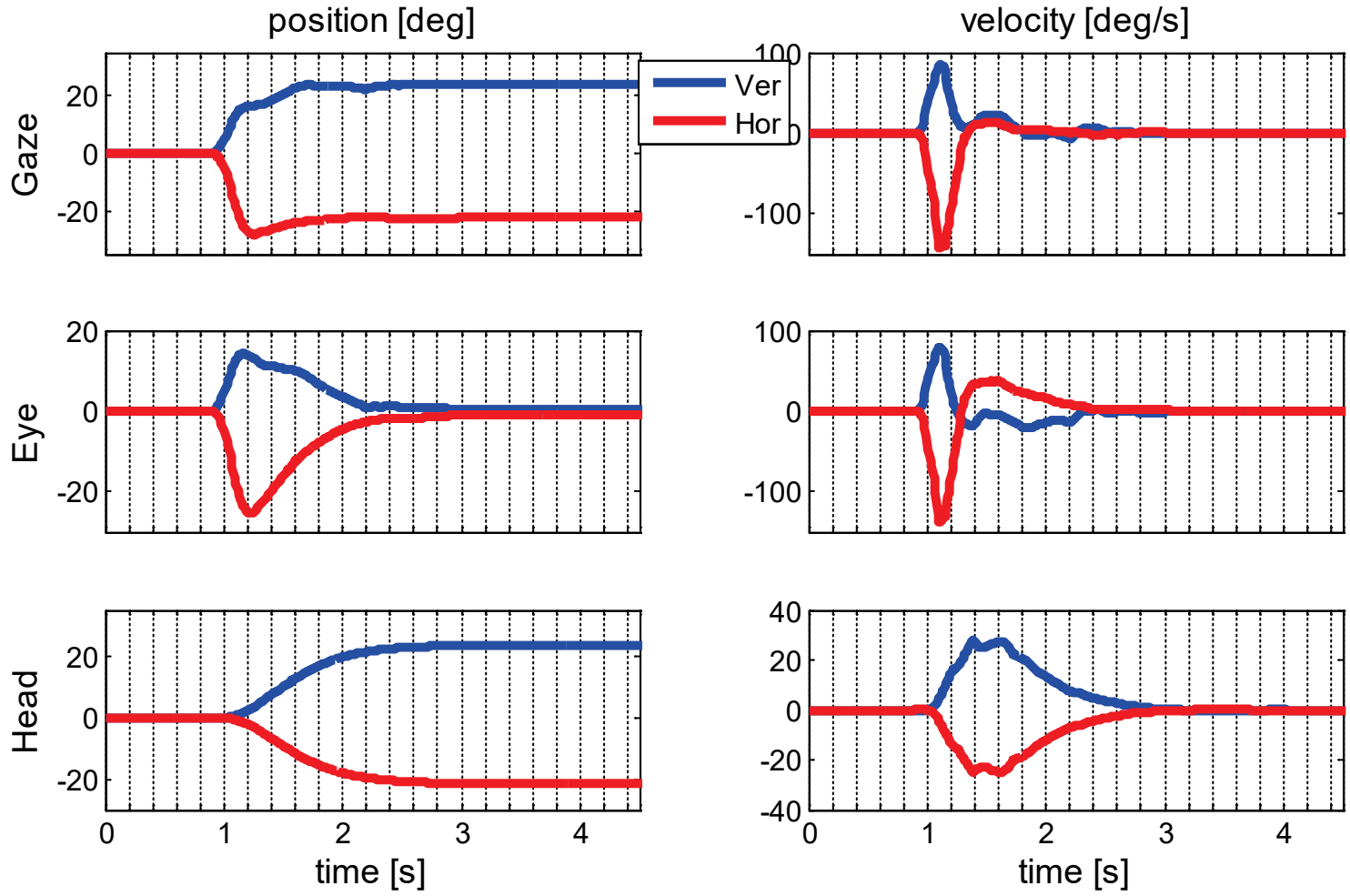
The Gaze Controller (5/7)

Studies on humans …

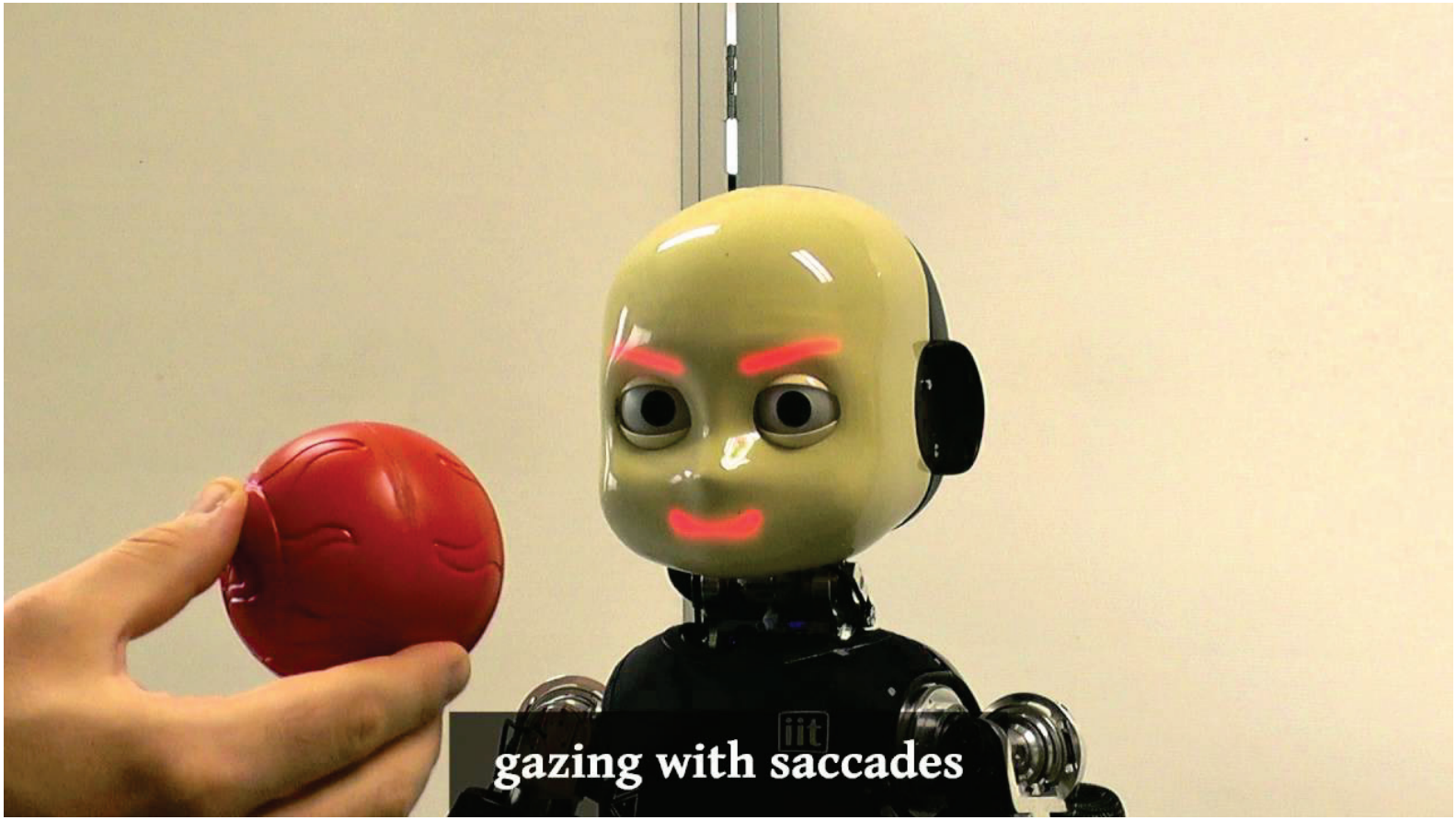


The Gaze Controller (6/7)

Results on iCub …



The Gaze Controller (7/7)

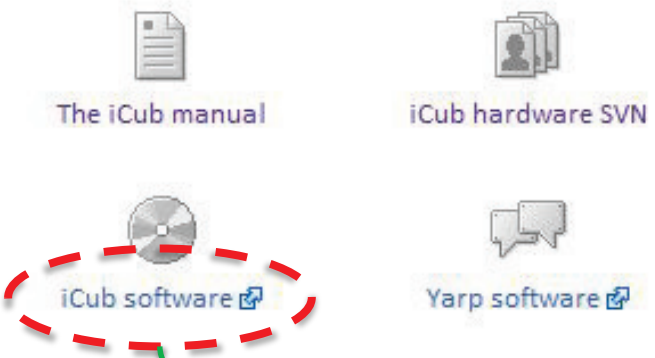


http://y2u.be/I4ZKfAvs1y0

Interfaces Documentation

In the search field: type **ICartesianControl/IGazeControl**

Interfaces Tutorials



- **Software** - most of the software (including **iCub modules**)
- **Applications** - a list of documented applications (collections of modules)
- **Tutorials** - a set of tutorials to learn how to use the software
- The documentation for contributed software is here: Contrib documentation
- Programmer's checklist:
  - **Compile status** - check if your code is compiling on a test server
  - Licensing - have you declared your authorship, and rights granted?
  - Coding guidelines - some tips on how to write your code
  - Modules and CMake - some tips on how to make your code compilable
  - Committing to the repository - things to check before committing files to the repository
- Reference material:
  - The The iCub manual
  - The RobotCub Website.
  - Getting the software.
  - Our software architecture, YARP.

- **The classic hello world** - how to write the very first program
- **Getting accustomed with motor interfaces** - a tutorial on how to use the motor interfaces
- **Getting accustomed with torque/impedance interfaces** - a tutorial on how to use the joint level torque/impedance interface
- **Basic Image Processing** - a tutorial on a basic image processing
- **The ResourceFinder Class (basic)** - a tutorial on how to organize the command line parameters of your modules
- **The ResourceFinder Class (advanced)** - organizing parameters: advanced tutorial
- **The RFModule Class** - a tutorial on how to use the module helper class to write a program
- **The RateThread Class** - a tutorial on how to write a control loop using threads
- **The Cartesian Interface** - a tutorial on how to control a robot's limb in the operational space
- **The Gaze Interface** - a tutorial on how to control the robot gaze through a Yarp interface
- **A short introduction to iDyn** - a short introduction to the iDyn library
  - **Computation of torques in a single chain, using iDyn** - how to compute torques in a single chain, using iDyn library

Interfaces Communalities (1/3)

OPENING THE CARTESIAN INTERFACE

```
#include <yarp/dev/all.h>
Property option;

option.put("device","cartesiancontrollerclient");
option.put("remote","/icub/cartesianController/right_arm");
option.put("local","/client/right_arm");

PolyDriver clientCartCtrl(option);

ICartesianControl *icart=NULL;
if (clientCartCtrl.isValid()) {
   clientCartCtrl.view(icart);
}
```

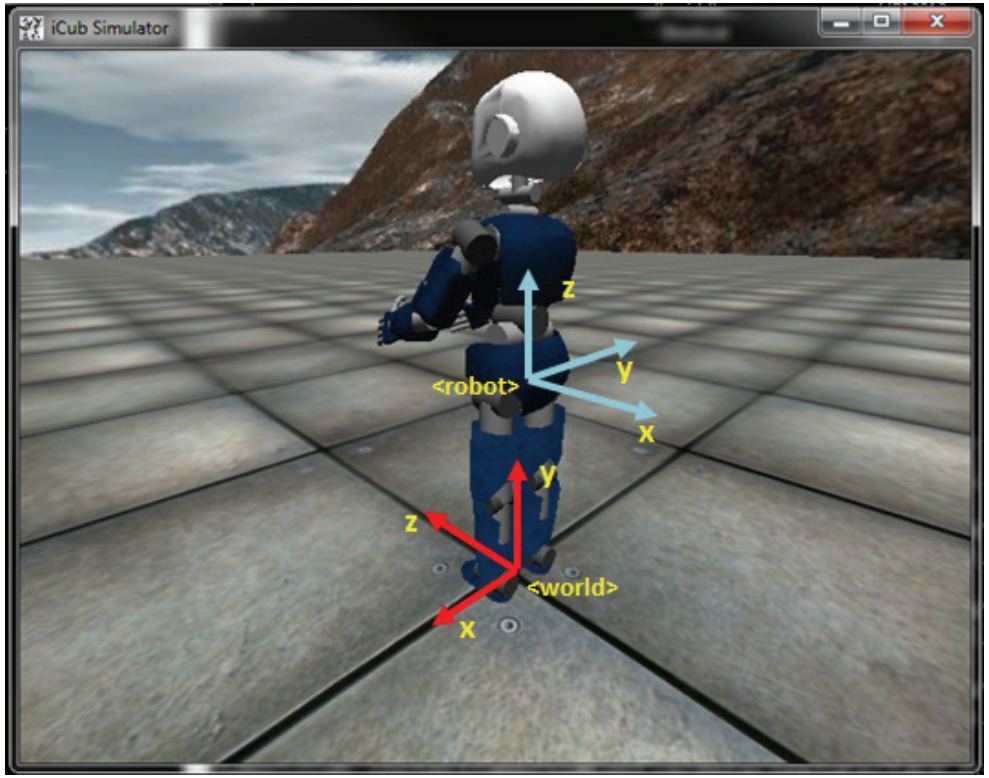
Interfaces Communalities (2/3)

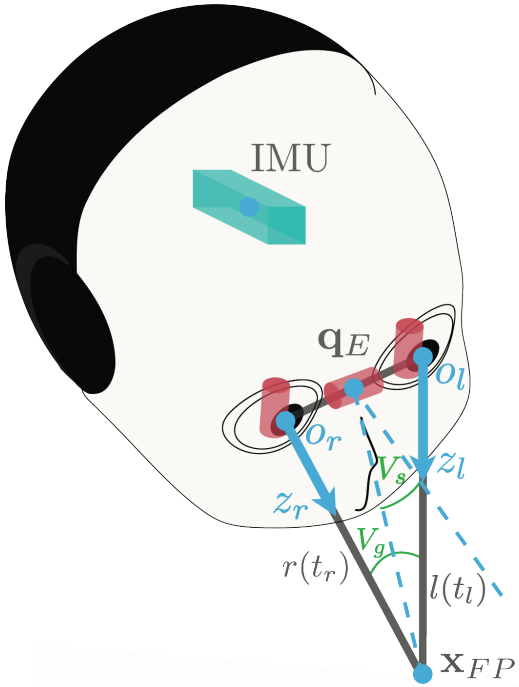
OPENING THE GAZE INTERFACE

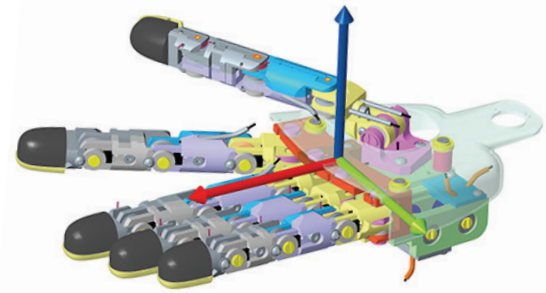
```
#include <yarp/dev/all.h>
Property option;

option.put("device",“gazecontrollerclient");
option.put("remote","/iKinGazeCtrl");
option.put("local","/client/gaze”);

PolyDriver clientGazeCtrl(option);

IGazeControl *igaze=NULL;
if (clientGazeCtrl.isValid()) {
   clientGazeCtrl.view(igaze);
}
```

Interfaces Communalities (3/3)

Coordinate Systems



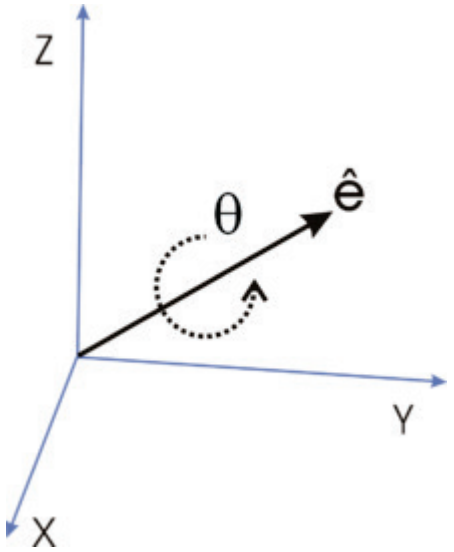




Cartesian Interface (1/7)

Orientation: Axis-Angle

$$r = [\underbrace{e_x\ e_y\ e_z}_{\|e\| = 1}\ \theta] \rightarrow rad$$



TARGET ORIENTATION through DIRECTION COSINE MATRIX

```
Matrix R(3,3);
// pose x-axis   y-axis         z-axis
   R(0,0)= 0.0;  R(0,1)= 1.0;  R(0,2)= 0.0;  // x-coordinate
   R(1,0)= 0.0;  R(1,1)= 0.0;  R(1,2)=-1.0;  // y-coordinate
   R(2,0)=-1.0;  R(2,1)= 0.0;  R(2,2)= 0.0;  // z-coordinate

Vector o=ctrl::dcm2axis(R);
```

Cartesian Interface (2/7)

RETRIEVE CURRENT POSE

```
Vector x,o;
icart->getPose(x,o);
```

REACH FOR A TARGET POSE (SEND-AND-FORGET)

```
icart->goToPose(xd,od);
icart->goToPosition(xd);
```

REACH FOR A TARGET POSE (WAIT-FOR-REPLY)

```
icart->goToPoseSync(xd,od);
icart->goToPositionSync(xd);
```

REACH AND WAIT

```
icart->goToPoseSync(xd,od);
icart->waitMotionDone();
```

Cartesian Interface (3/7)

ASK FOR A POSE (without moving)

```
Vector xdhat,odhat,qdhat;
icart->askForPose(xd,xdhat,odhat,qdhat);
```

MOVE FASTER/SLOWER

```
icart->setTrajTime(1.5);  // point-to-point trajectory time
```

REACH WITH GIVEN PRECISION

```
icart->setInTargetTol(0.001);
```

KEEP THE POSE ONCE DONE

```
icart->setTrackingMode(true);
```

Cartesian Interface (4/7)

ENABLE/DISABLE DOF

```
Vector curDof;
icart->getDOF(curDof); // [0 0 0 1 1 1 1 1 1 1]

Vector newDof(3);
newDof[0]=1; // torso pitch: 1 => enable
newDof[1]=2; // torso roll: 2 => skip
newDof[2]=1; // torso yaw: 1 => enable
icart->setDOF(newDof,curDof);
```

GIVE PRIORITY TO REACHING IN POSITION/ORIENTATION

```
icart->setPosePriority("position"); // default
icart->setPosePriority("orientation");
```

Cartesian Interface (5/7)

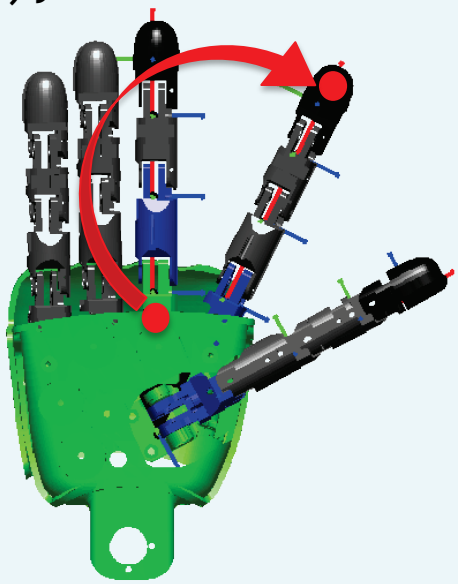
CONTEXT SWITCH

```
icart->setDOF(newDof1,curDof1);    // prepare the context
icart->setTrackingMode(true);

int context_0;
icart->storeContext(&context_0);   // latch the context

icart->setDOF(newDof2,curDof2);    // perform some actions
icart->goToPose(x,o);

icart->restoreContext(context_0); // retrieve context_0
icart->goToPose(x,o);             // perform with context_0
```

Cartesian Interface (6/7)

DEFINING A DIFFERENT EFFECTOR

```
iCubFinger finger("right_index");
Vector encs; iencs->getEncoders(encs.data());
Vector joints; finger.getChainJoints(encs,joints);
Matrix tipFrame=finger.getH((M_PI/180.0)*joints);

Vector tip_x=tipFrame.getCol(3);
Vector tip_o=ctrl::dcm2axis(tipFrame);

icart->attachTipFrame(tip_x,tip_o);

icart->getPose(x,o);
icart->goToPose(xd,od);

icart->removeTipFrame();
```



Cartesian Interface (7/7)

Find out more (e.g. **Events Callbacks** …):
http://wiki.icub.org/iCub/main/dox/html/icub_cartesian_interface.html

USING THE INTERFACE ALONG WITH THE SIMULATOR

```
1> iCub_SIM
2> yarprobotinterface --context simCartesianControl
3> iKinCartesianSolver --context simCartesianControl --part left_arm


option.put("device","cartesiancontrollerclient");
option.put("remote","/icubSim/cartesianController/left_arm");
option.put("local","/client/right_arm");
```

Gaze Interface (1/6)

GET CURRENT FIXATION POINT IN CARTESIAN DOMAIN

```
Vector x;
igaze->getFixationPoint(x);
```

GET CURRENT FIXATION POINT IN ANGULAR DOMAIN

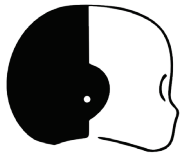
```
Vector ang;
igaze->getAngles(ang);
// ang[0] => azimuth [deg]
// ang[1] => elevation [deg]
// ang[2] => vergence [deg]
```

LOOK AT 3D POINT

```
igaze->lookAtFixationPoint(xd);
```

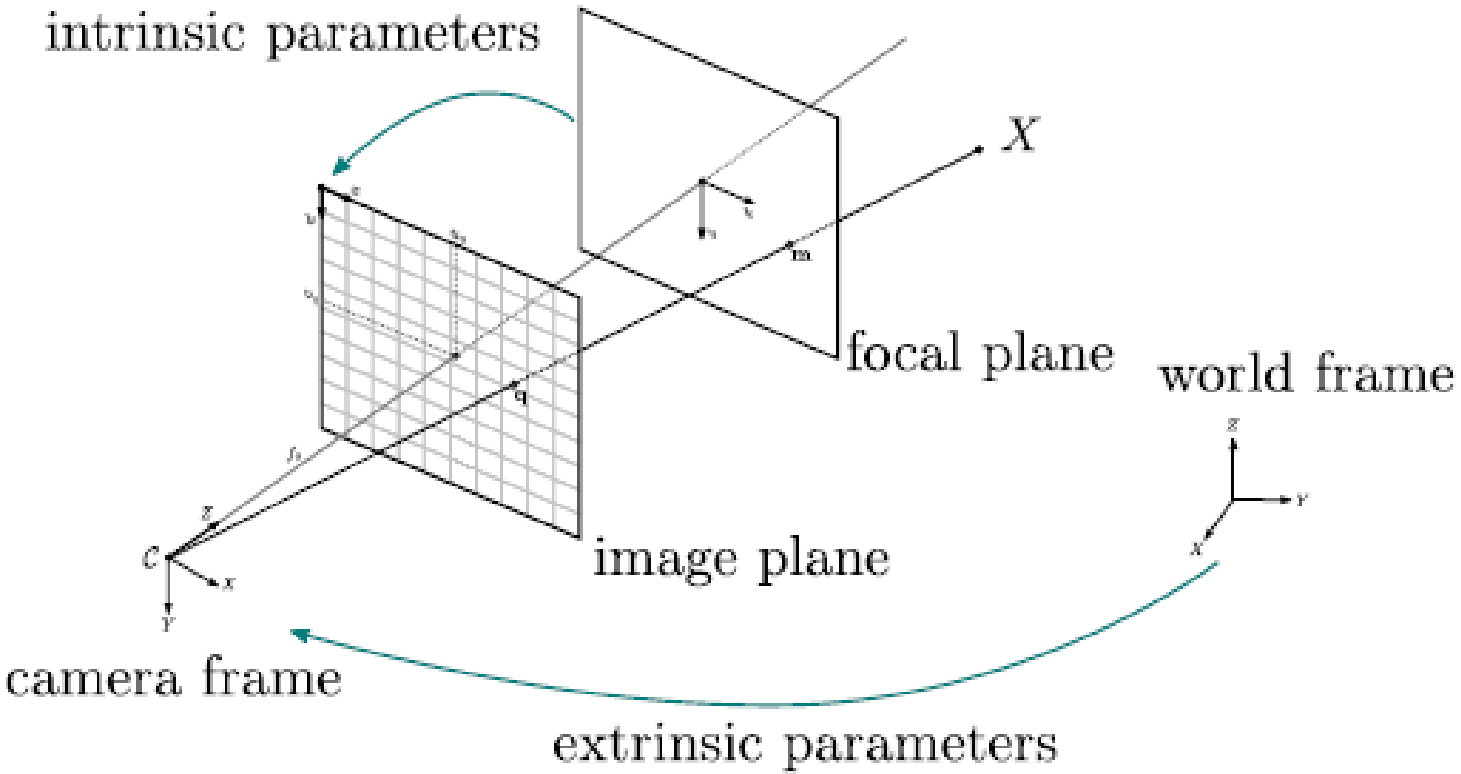
… IN ANGULAR DOMAIN

```
igaze->lookAtAbsAngles(ang);
igaze->lookAtRelAngles(ang);
```



Gaze Interface (2/6)

LOOK AT POINT IN IMAGE DOMAIN



Gaze Interface (3/6)

LOOK AT POINT IN IMAGE DOMAIN

```
int camSel=0;  // 0 => left, 1 => right
Vector px(2);
px[0]=100;
px[1]=50;
double z=1.0;

igaze->lookAtMonoPixel(camSel,px,z);
```

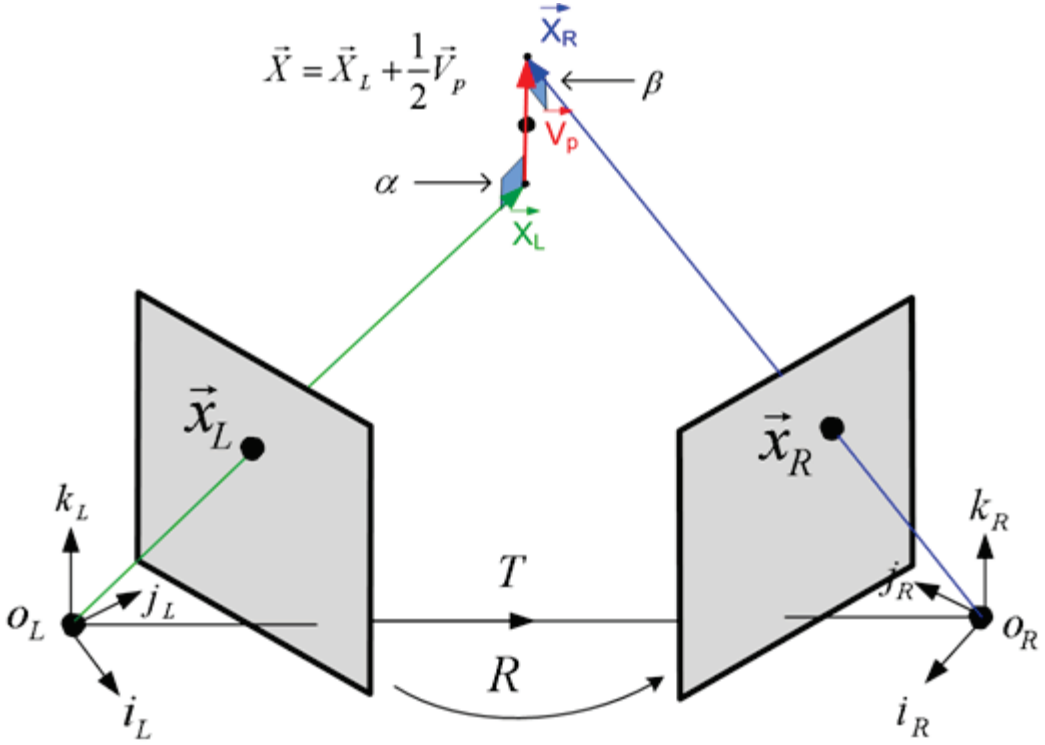
… EQUIVALENT TO

```
Vector x;
igaze->get3DPoint(camSel,px,z,x);
igaze->lookAtFixationPoint(x);
```

Gaze Interface (4/6)

GEOMETRY OF PIXELS

```
Vector x;
igaze->get3DPointOnPlane(camSel,px,plane,x);
igaze->get3DPointFromAngles(mode,ang,x);
igaze->triangulate3DPoint(pxl,pxr,x);
```



Gaze Interface (5/6)

GEOMETRY OF PIXELS

```
Vector x;
igaze->get3DPointOnPlane(camSel,px,plane,x);
igaze->get3DPointFromAngles(mode,ang,x);
igaze->triangulate3DPoint(pxl,pxr,x);
```

LOOK AT POINT WITH STEREO APPROACH => LOOPING!

```
Vector c(2); c[0]=160.0; c[1]=120.0;
bool converged=false;

while (!converged) {
    Vector pxl(2),pxr(2);
    pxl[0]=...; pxl[1]=...;  // retrieve data from vision
    pxr[0]=...; pxr[1]=...;

    igaze->lookAtStereoPixels(pxl,pxr);
    converged=(0.5*(norm(c-pxl)+norm(c-pxr))<5);
}
```

Gaze Interface (6/6)

Find out more (e.g. **Events Callbacks, Fast Saccadic Mode** …):
http://wiki.icub.org/iCub/main/dox/html/icub_gaze_interface.html

USING THE INTERFACE ALONG WITH THE SIMULATOR

```
1> iCub_SIM
2> iKinGazeCtrl --from configSim.ini

option.put("device",“gazecontrollerclient");
option.put("remote","/iKinGazeCtrl”);
option.put("local","/client/right_arm");
```