Overview

- Introduction to Kinematics
- Joint Motor Control (overview + tutorial)
- Cartesian Control (theory + API + tutorial)
- Gaze Control (theory + API + tutorial)
- Assignment

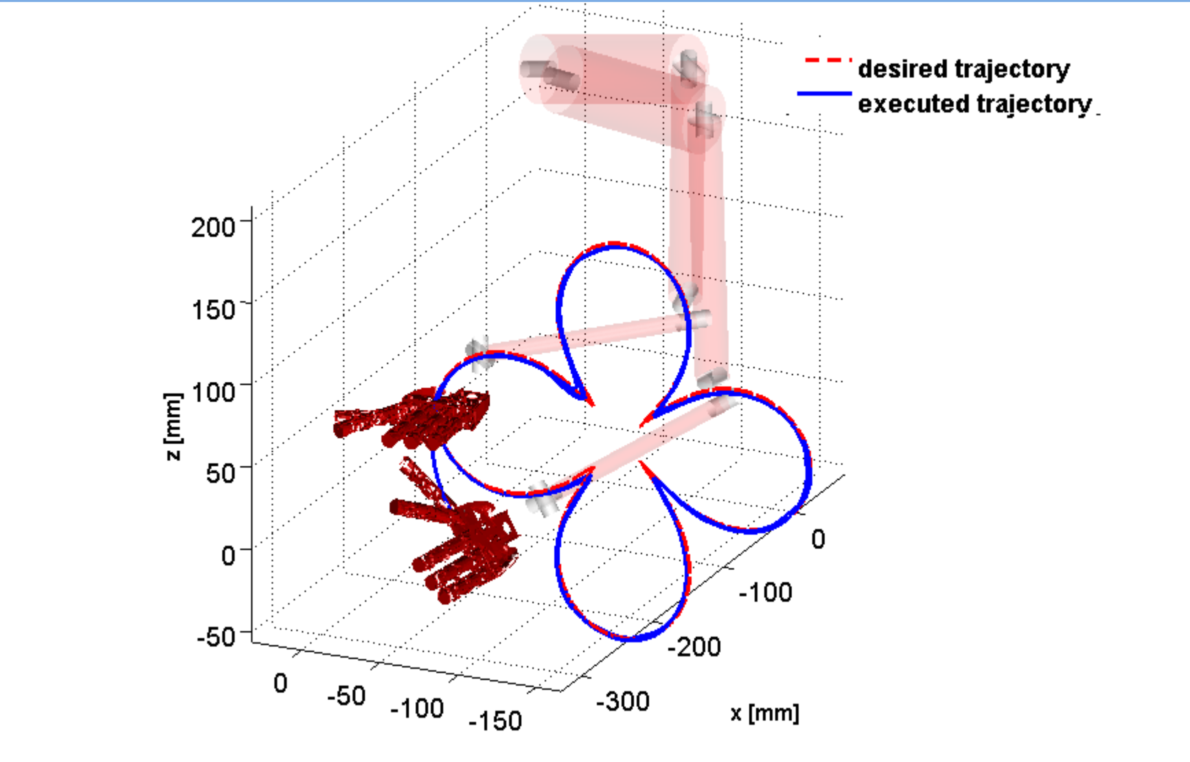
Basic Kinematics (1/2)

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

The Problem

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

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



Basic Kinematics (2/2)

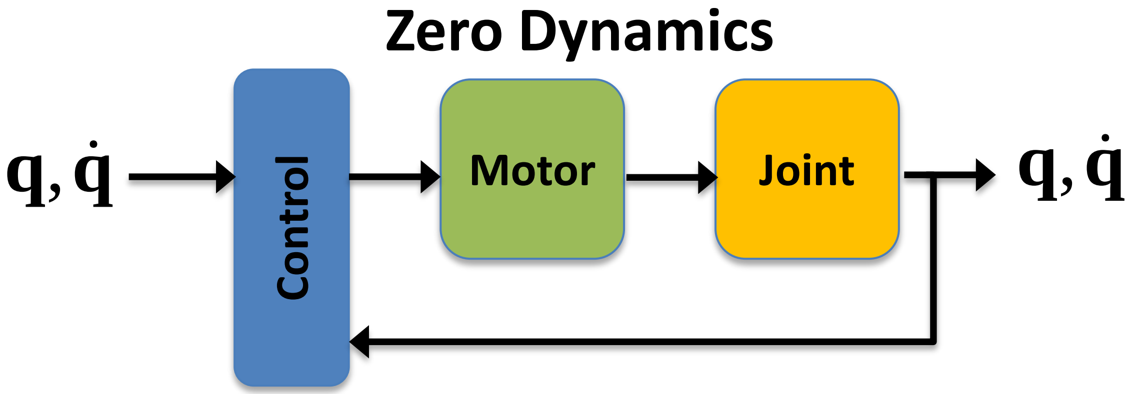
Dynamics – forces, torques, inertias, energy, contact with environment

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

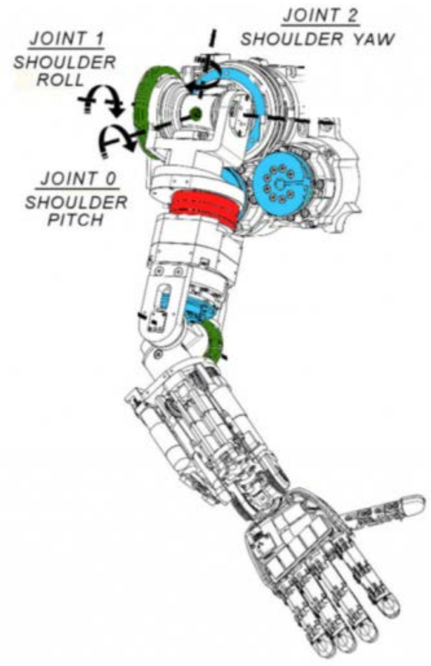
VS.

Kinematics – pure motion imposed to the manipulator's joints

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



Joint Motor Control (1/4)



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;
}
```

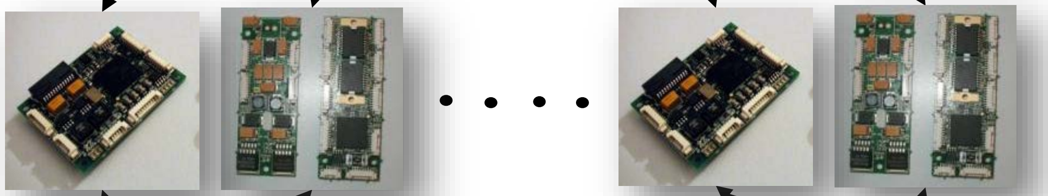
Gbit Ethernet

PC104

FIRMWARE @ 1 KHz

CONTROLLERS

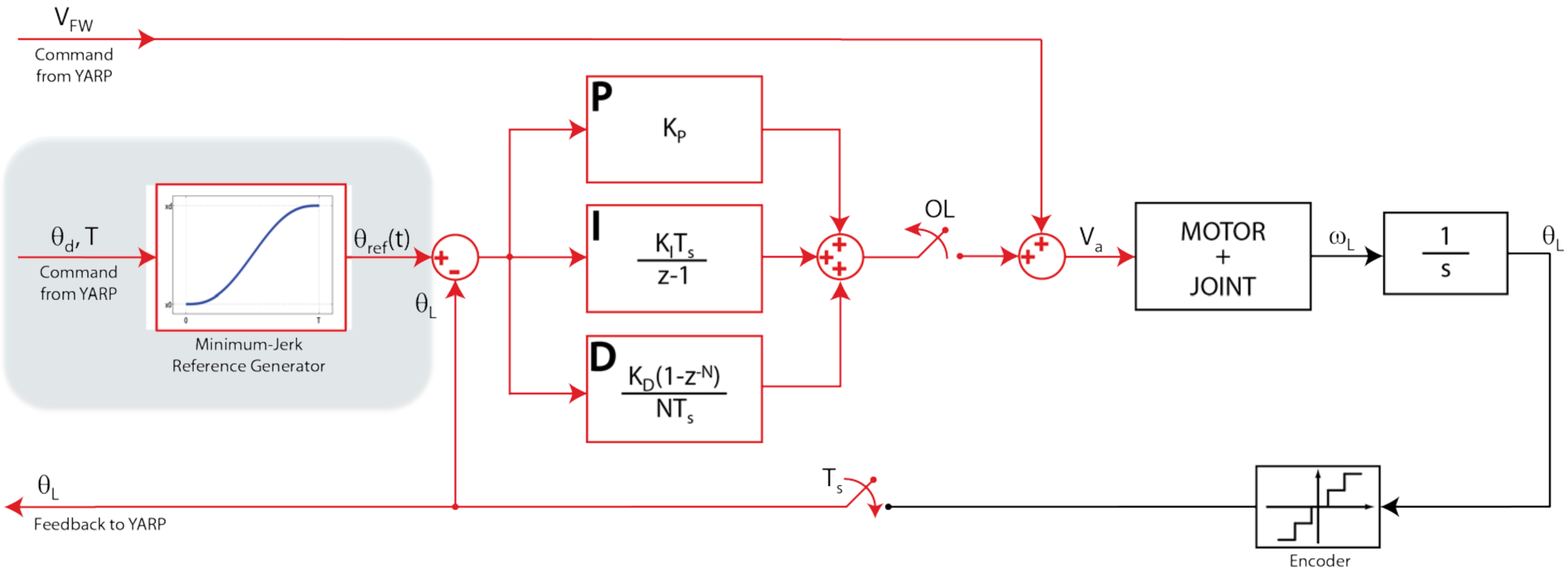
DC MOTORS



Sensors & Actuators

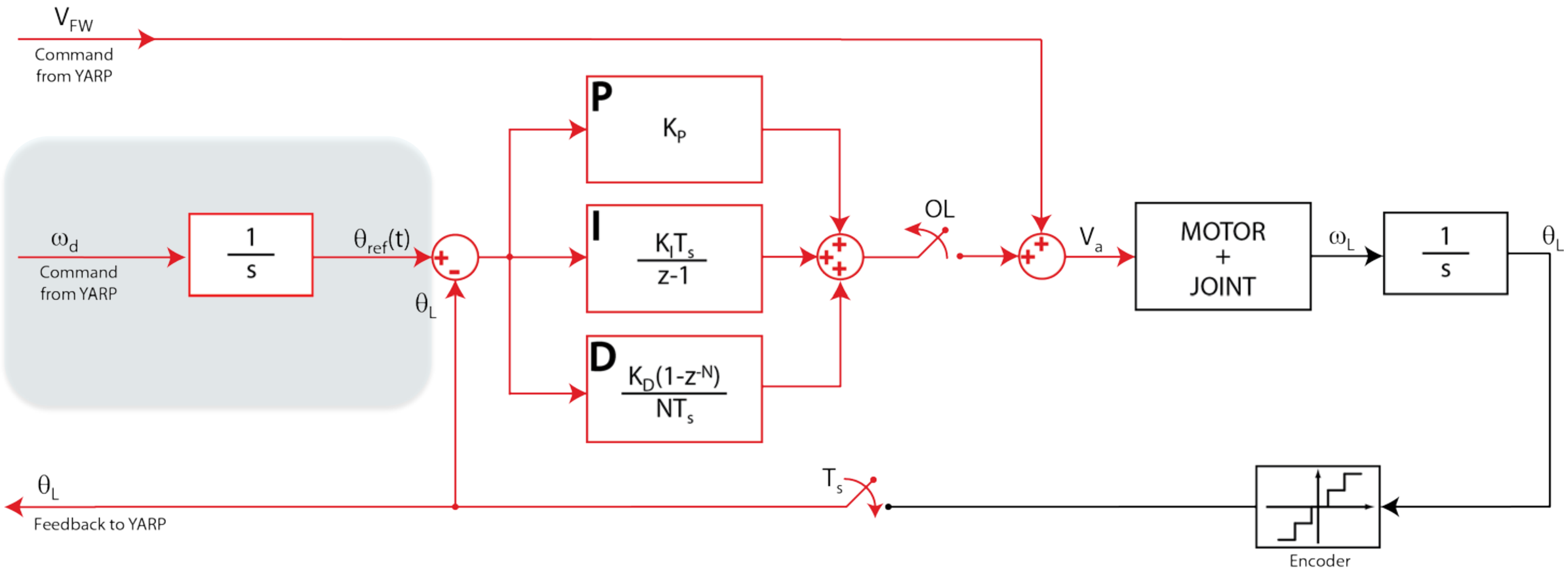
Joint Motor Control (2/4)

Position Control



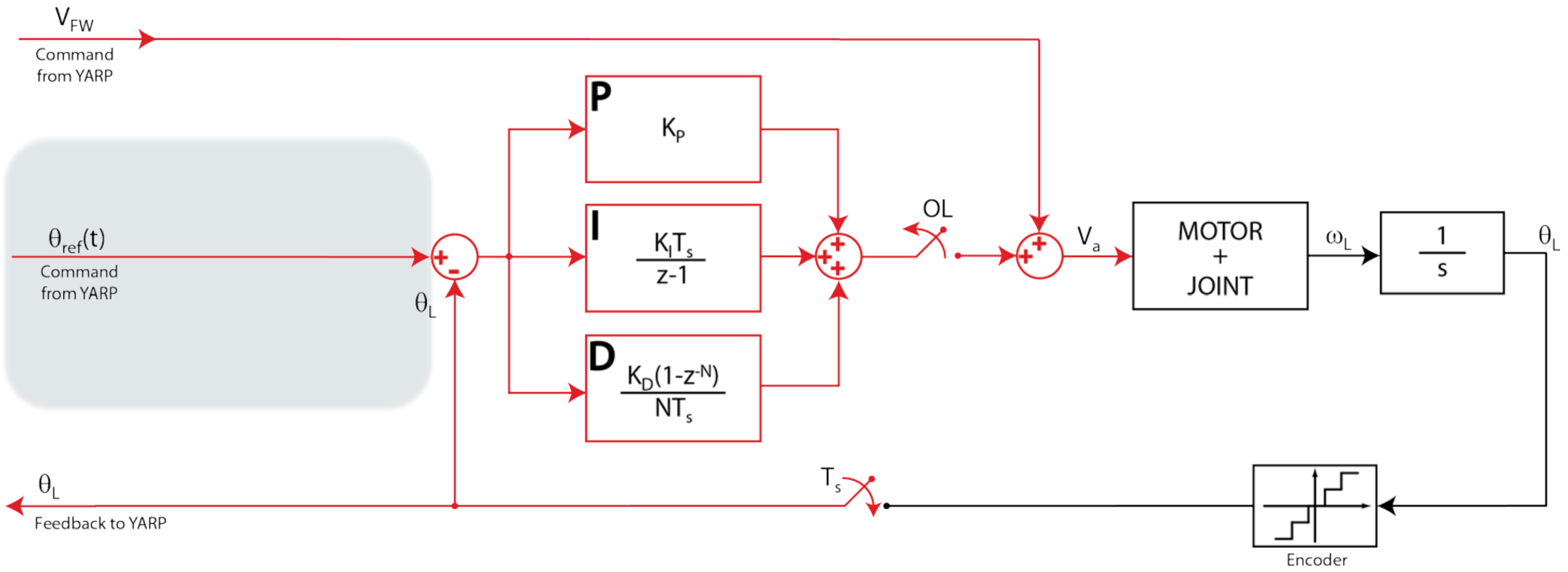
Joint Motor Control (3/4)

Velocity Control



Joint Motor Control (4/4)

Position-Direct Control



Tutorial Time !

The Cartesian Controller (1/5)

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.

very "robotic" movements:

- snap onset
- exponential decay

Jacobian Transpose

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

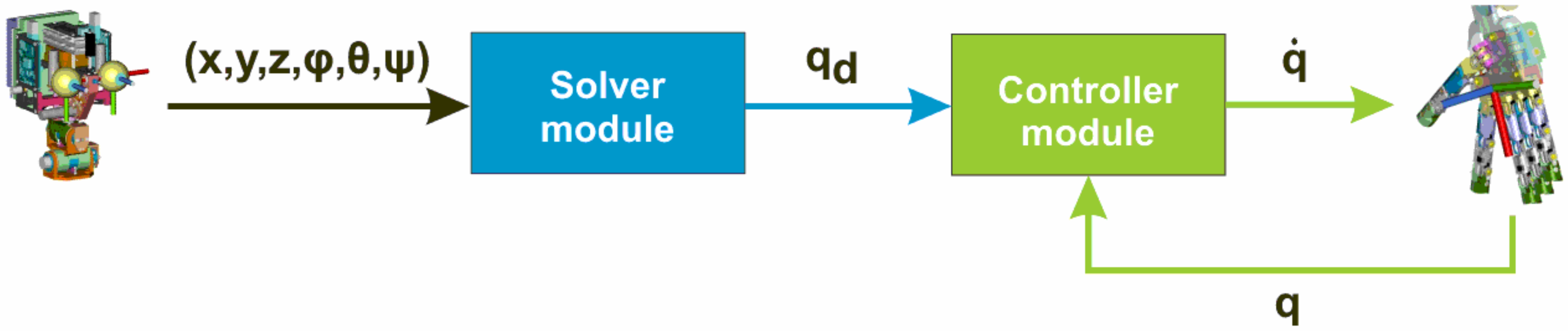
Jacobian Pseudoinverse

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

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 (2/5)



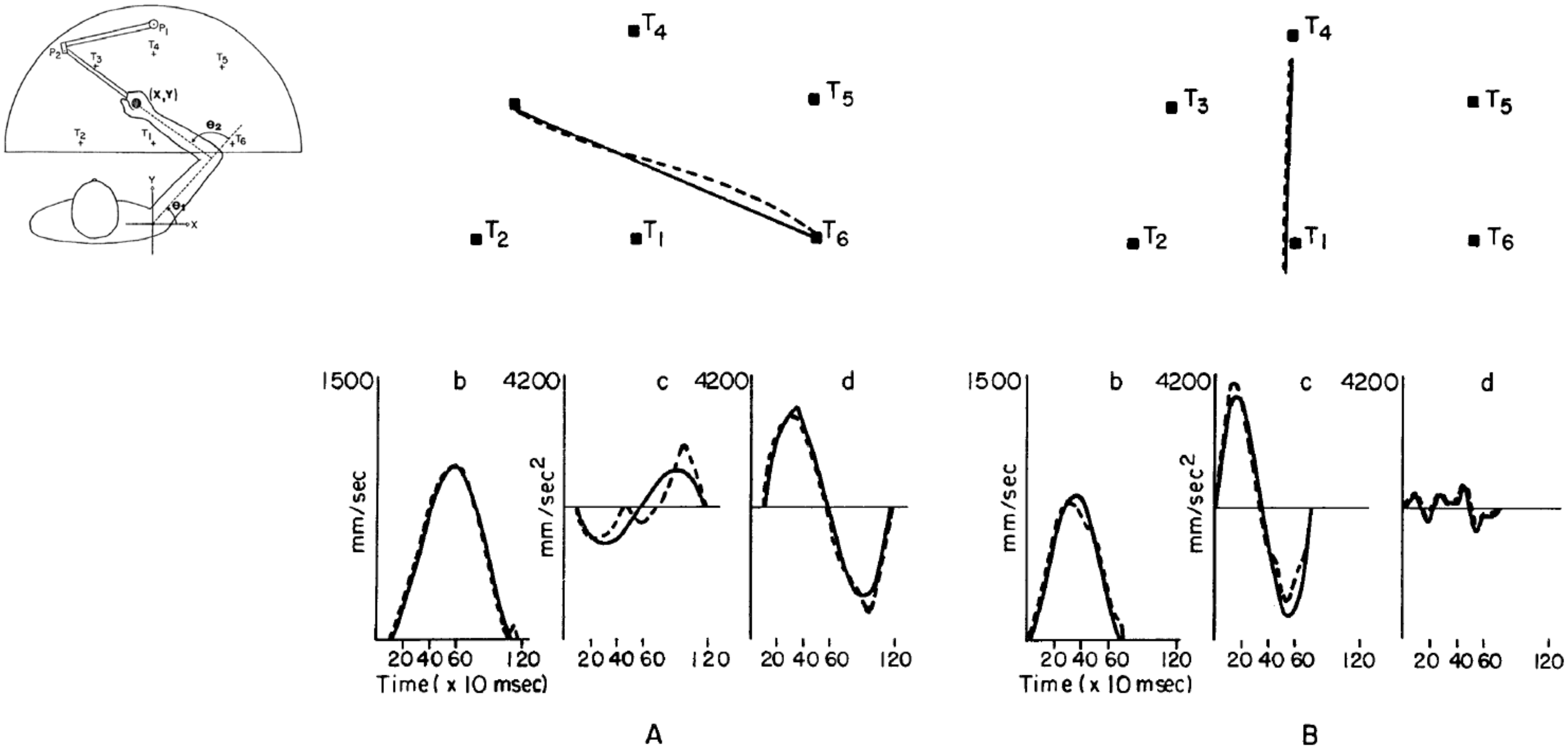
nonlinear constrained optimization:

- fast
- scalable

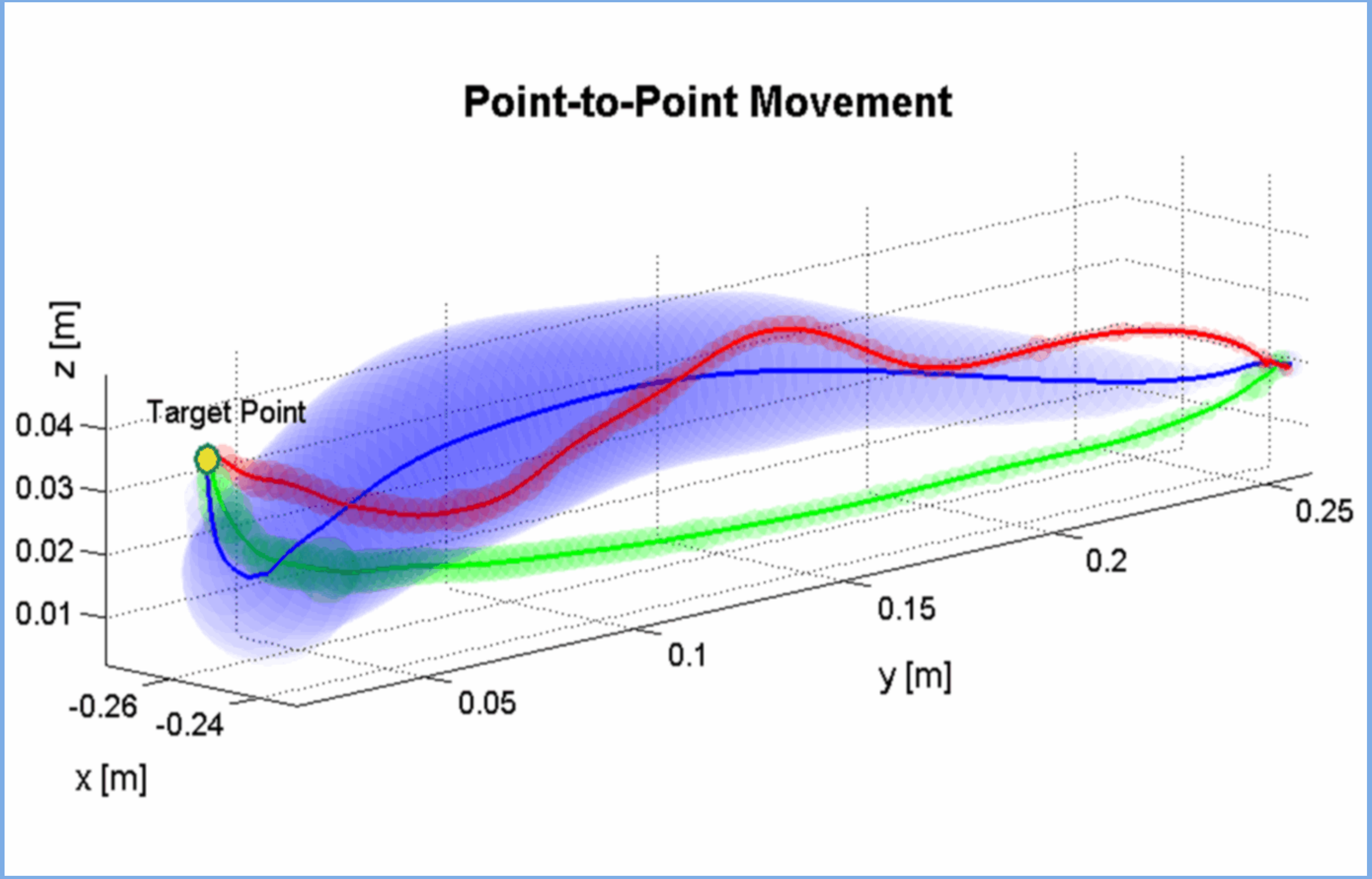
human-like movements (min-jerk):

- bell-shape velocity profile
- quasi-straight path

The Cartesian Controller (3/5)



The Cartesian Controller (4/5)

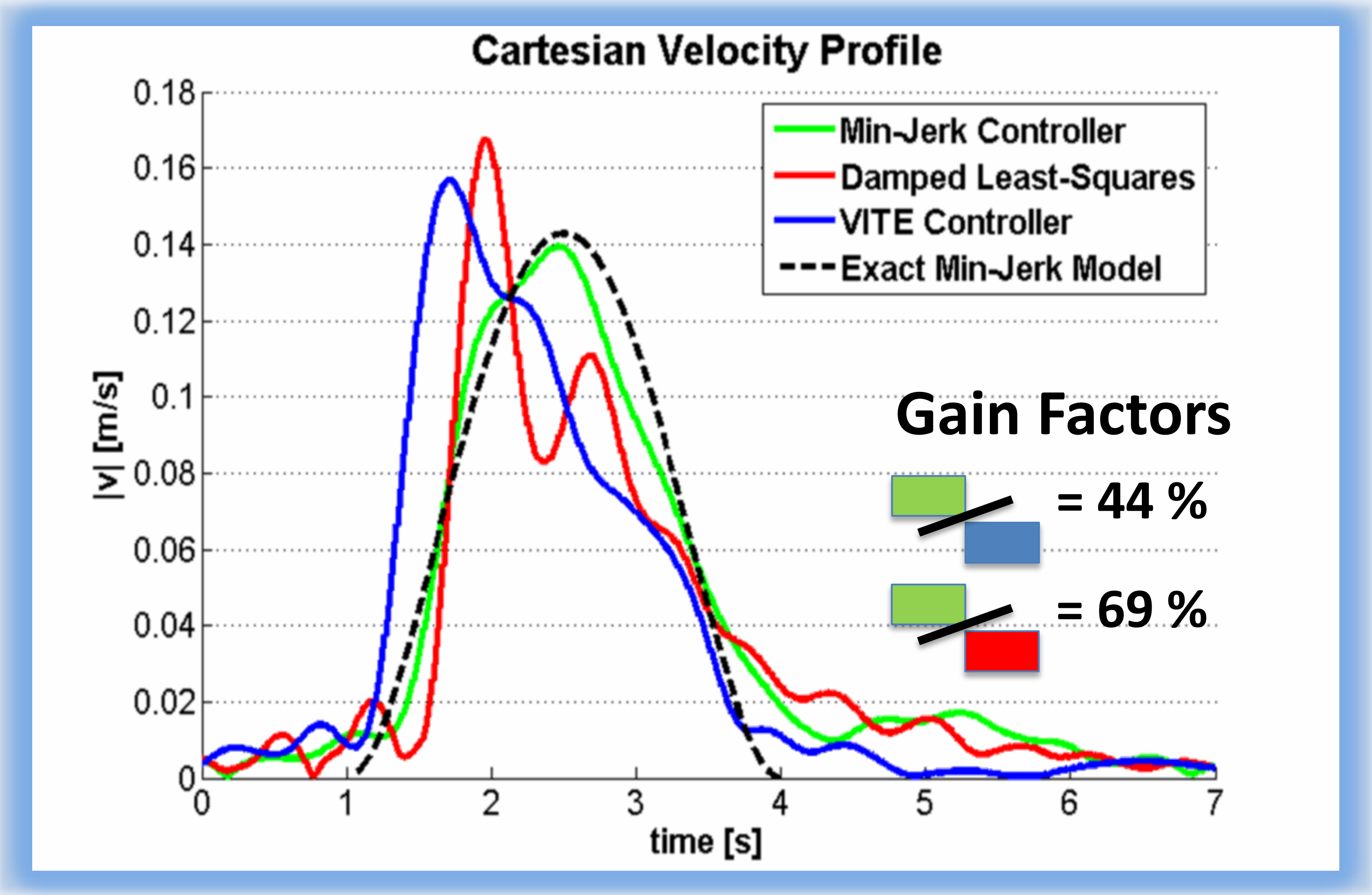


Min-Jerk

DLS

VITE

The Cartesian Controller (5/5)



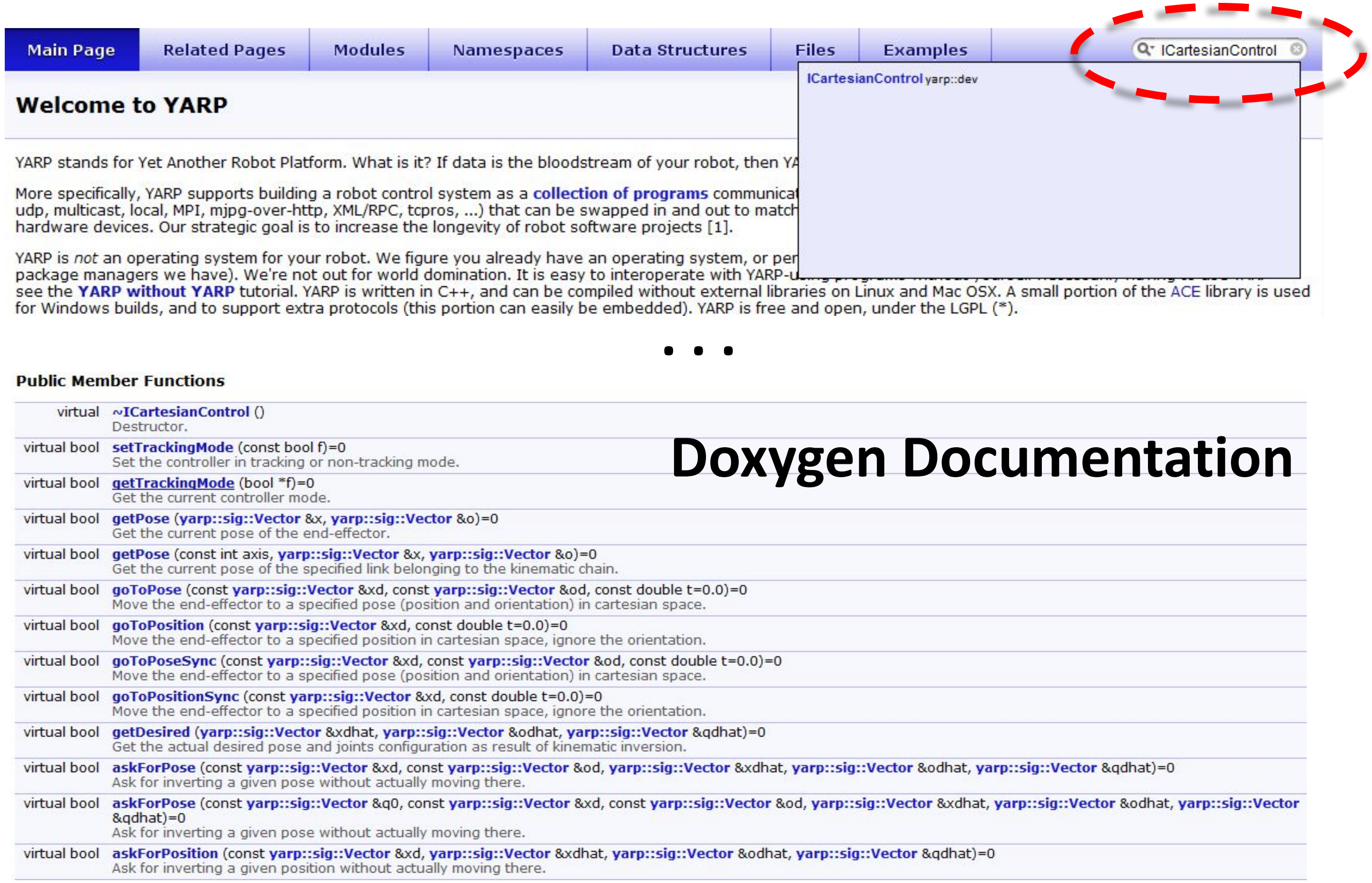
Min-Jerk

DLS

VITE

Interfaces Documentation

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



Doxygen Documentation

Interfaces Tutorials

The iCub manual

iCub hardware SVN

iCub software

Yarp software

- **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

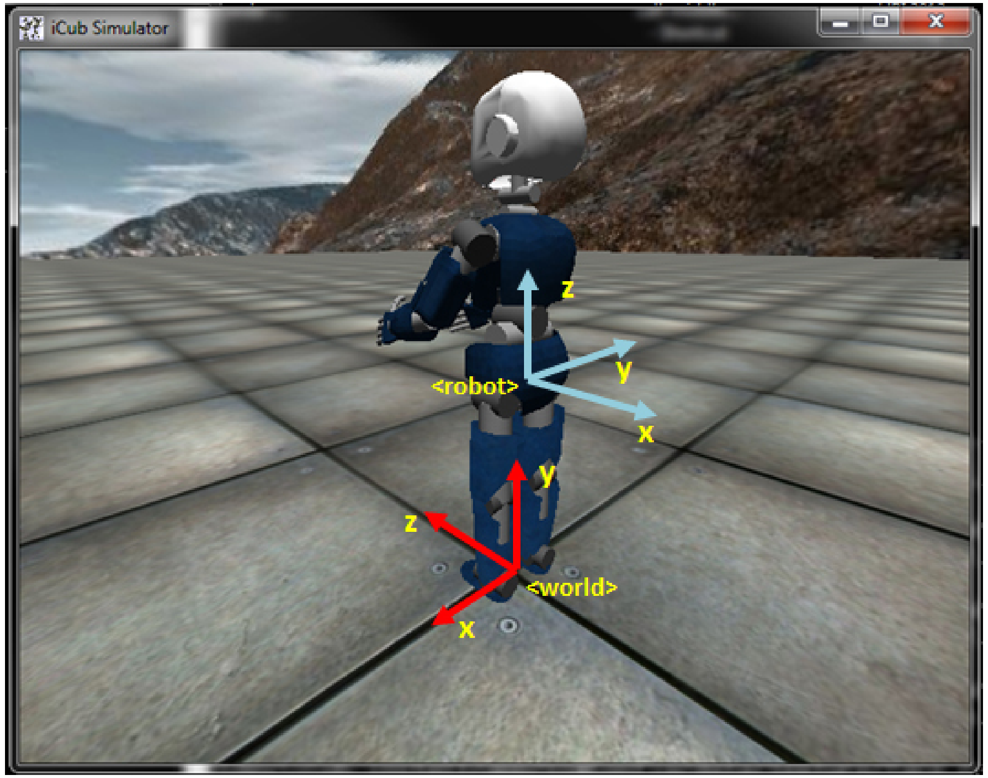
Cartesian Interface (1/9)

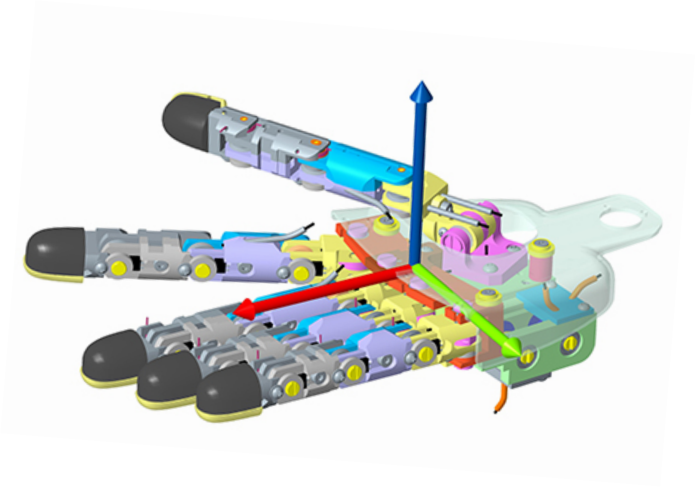
OPENING THE CARTESIAN INTERFACE

```
#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);
}
```

Cartesian Interface (2/9)

Coordinate Systems



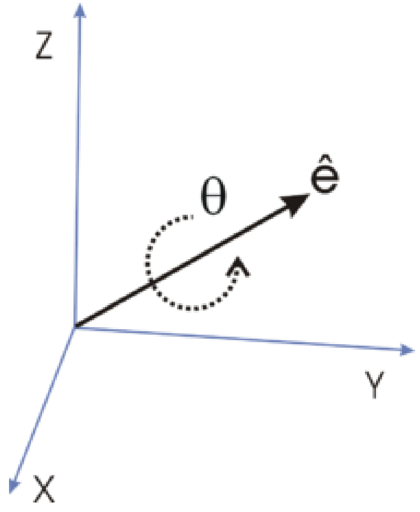


Cartesian Interface (3/9)

Orientation: Axis-Angle

$$r = [e_x \ e_y \ e_z \ \theta]$$

$\|e\| = 1$ (for $e_x \ e_y \ e_z$), $\theta \rightarrow rad$



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 (4/9)

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 (5/9)

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 (6/9)

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 (7/9)

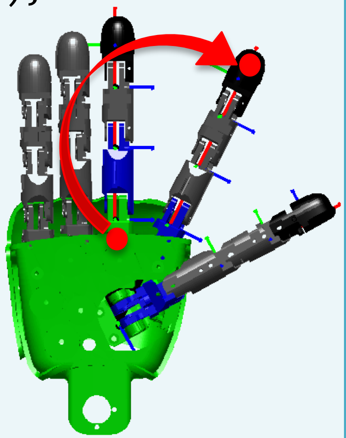
CONTEXT SWITCH

```
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 (8/9)

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 (9/9)

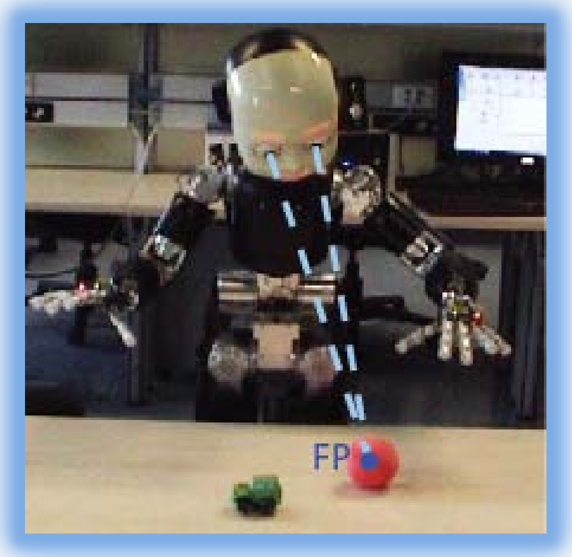
Find out more (e.g. **Events Callbacks** …):
http://wiki.icub.org/iCub/main/dox/html/icub_cartesian_interface.html

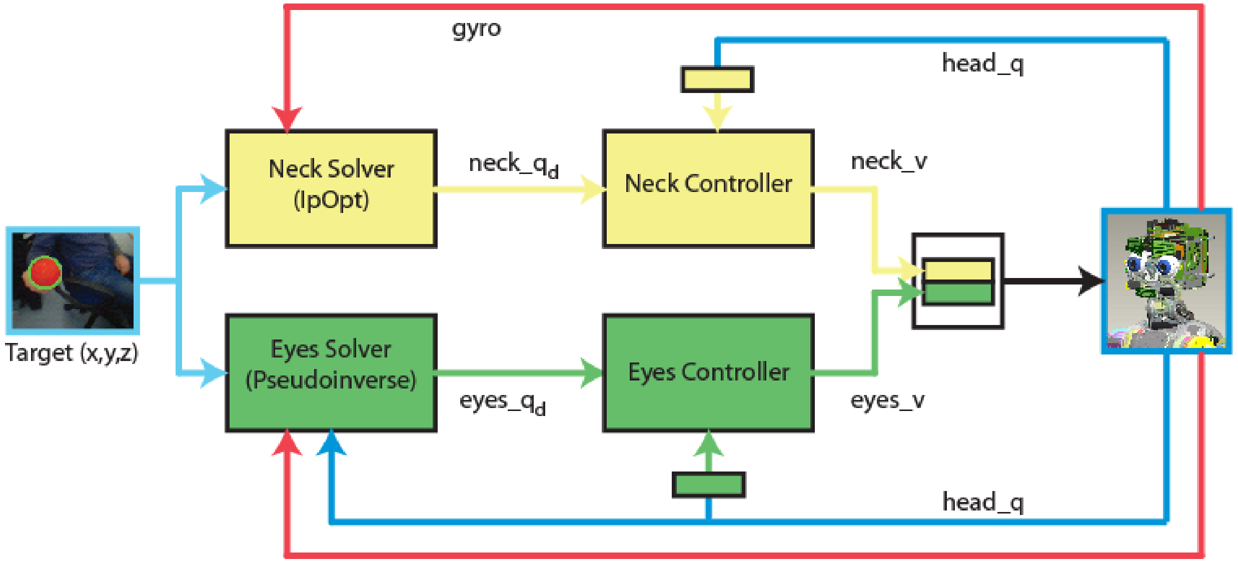
USING THE INTERFACE ALONG WITH THE SIMULATOR

```
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");
```

Tutorial Time !

The Gaze Controller (1/6)

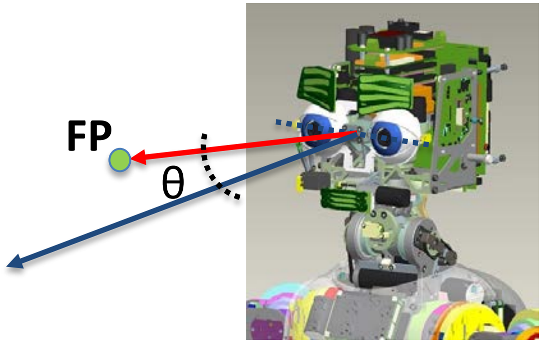




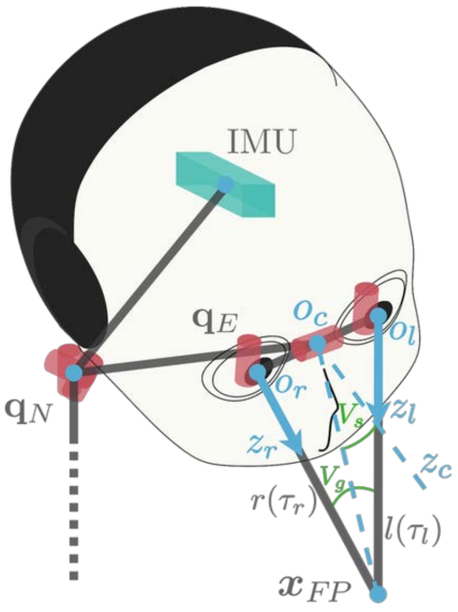
Yet another Cartesian Controller: **reuse ideas** …

Then, apply easy transformations from Cartesian to …

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

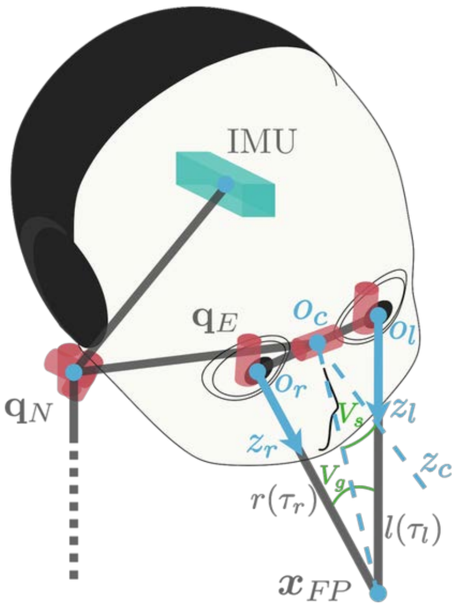


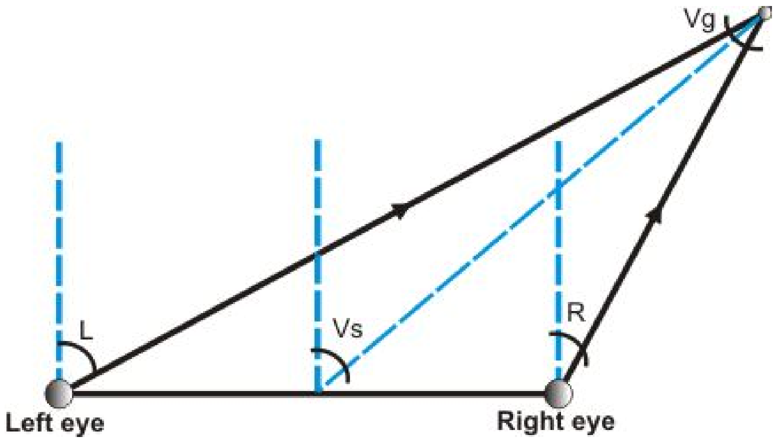
The Gaze Controller (2/6)



Joint #	Part	Joint Name	Range	Unit
0	Neck	**Pitch**	$+/-$	[deg]
1	Neck	**Roll**	$+/-$	[deg]
2	Neck	**Yaw**	$+/-$	[deg]
3	Eyes	**Tilt**	$+/-$	[deg]
4	Eyes	**Version**	$+/-$	[deg]
5	Eyes	**Vergence**	≥ 0	[deg]

The Gaze Controller (3/6)

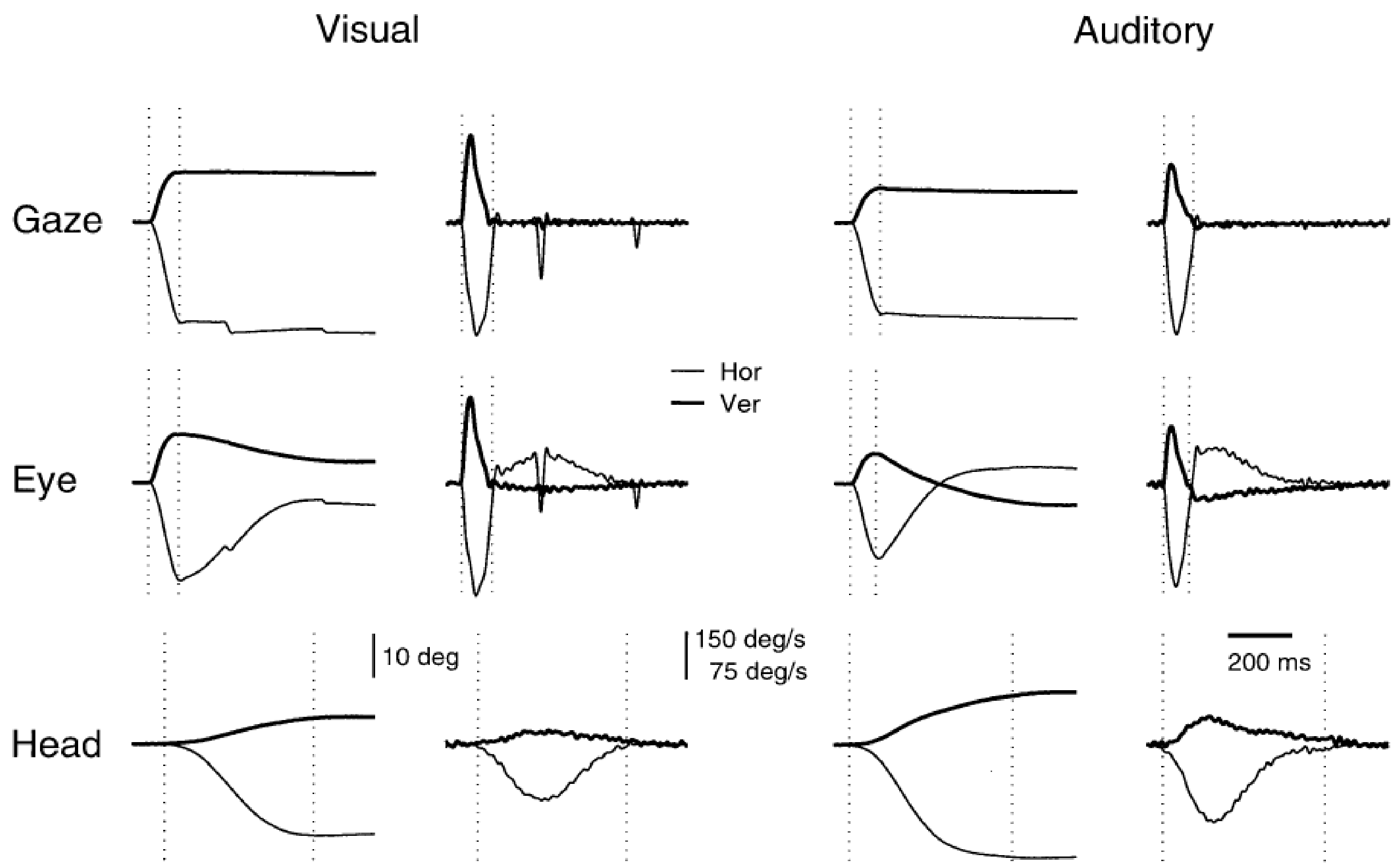




$$\begin{cases} V_g = L - R \\ V_s \approx (L + R)/2 \end{cases}$$

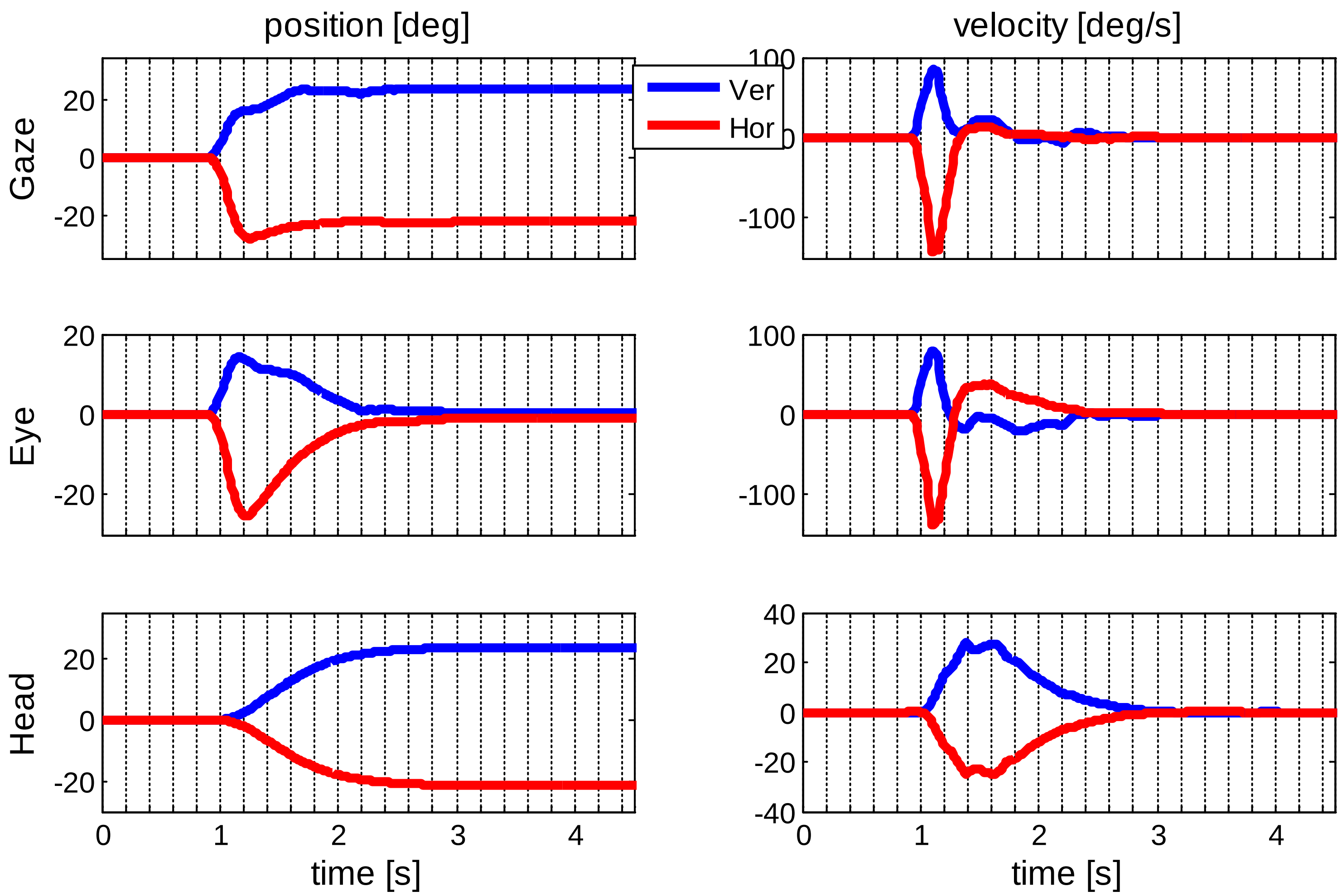
The Gaze Controller (4/6)

Studies on humans …

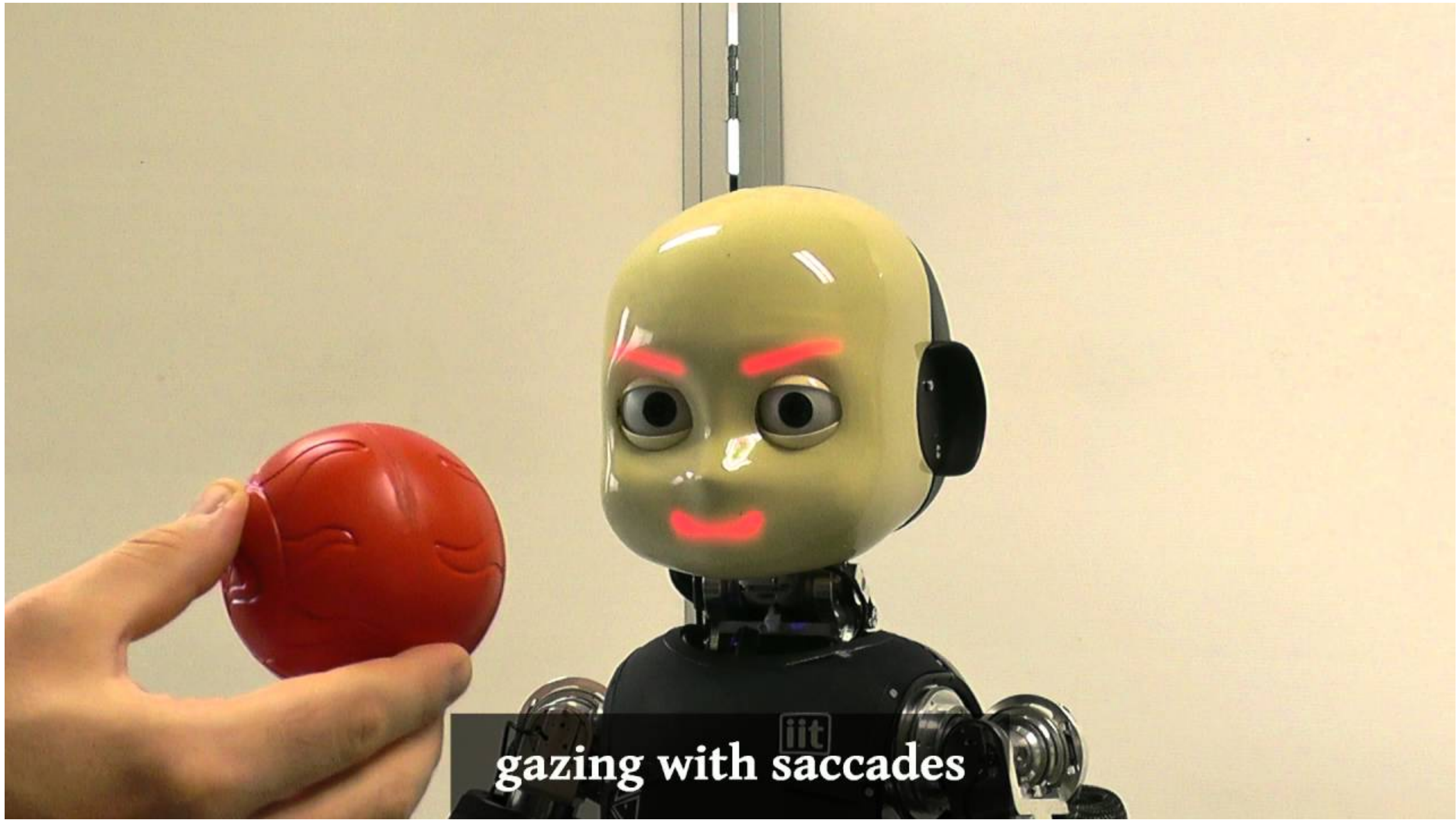


The Gaze Controller (5/6)

Results on iCub ...



The Gaze Controller (6/6)



http://y2u.be/I4ZKfAvs1y0

Gaze Interface (1/7)

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);
}
```

Gaze Interface (2/7)

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

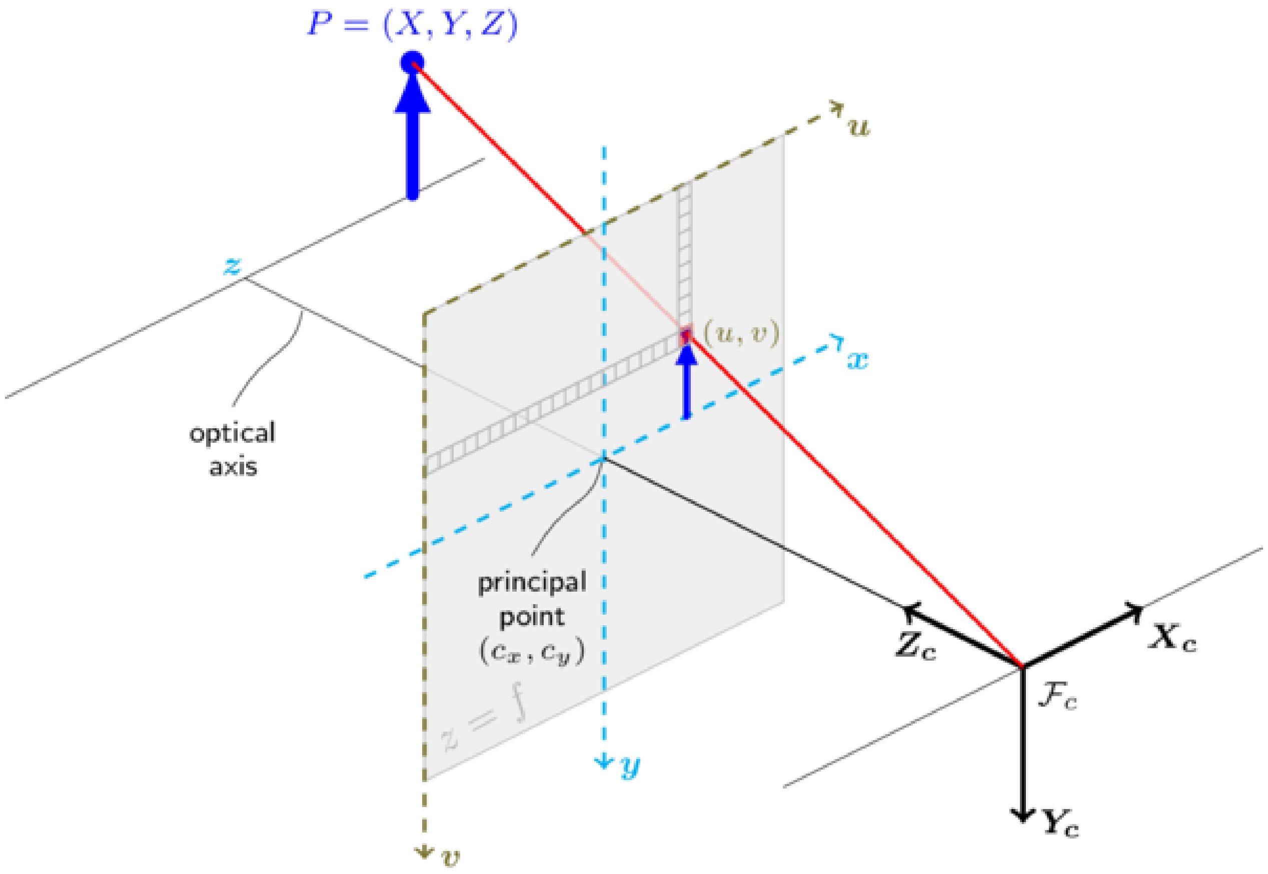
```
igaze->lookAtFixationPoint(xd);
```

… IN ANGULAR DOMAIN

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

Gaze Interface (3/7)

LOOK AT POINT IN IMAGE DOMAIN



Gaze Interface (4/7)

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);
```

=

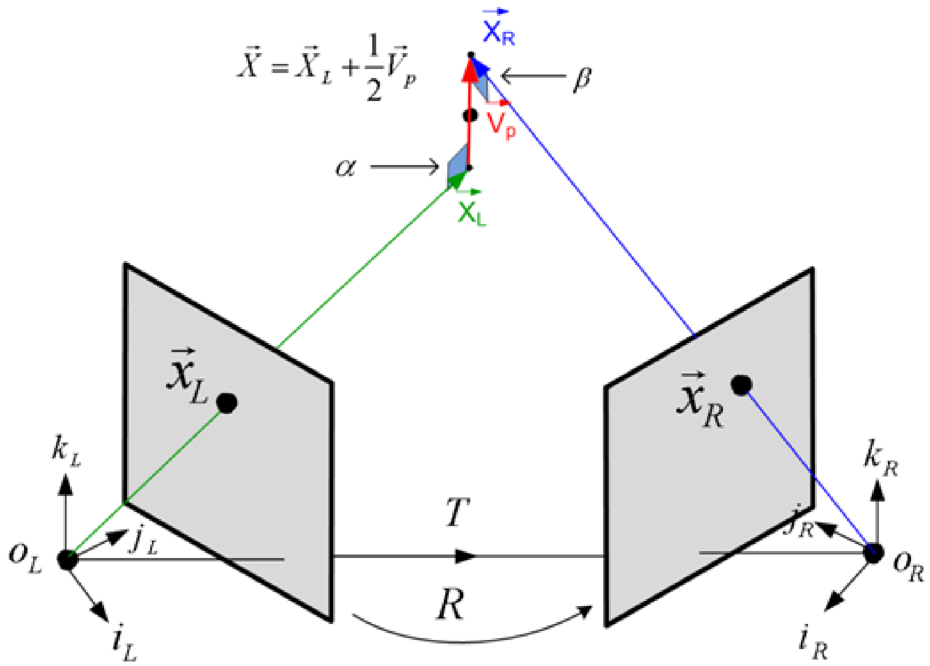
… EQUIVALENT TO

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

Gaze Interface (5/7)

GEOMETRY OF PIXELS

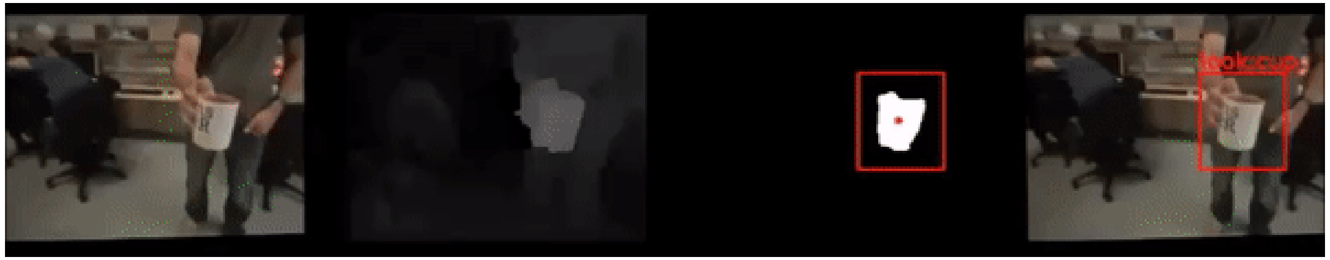
```
Vector x;
igaze->get3DPointOnPlane(camSel,px,plane,x);
igaze->get3DPointFromAngles(mode,ang,x);
igaze->triangulate3DPoint(pxl,pxr,x);
```



Gaze Interface (6/7)

GEOMETRY OF PIXELS

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



Gaze Interface (7/7)

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");
```

Tutorial Time !