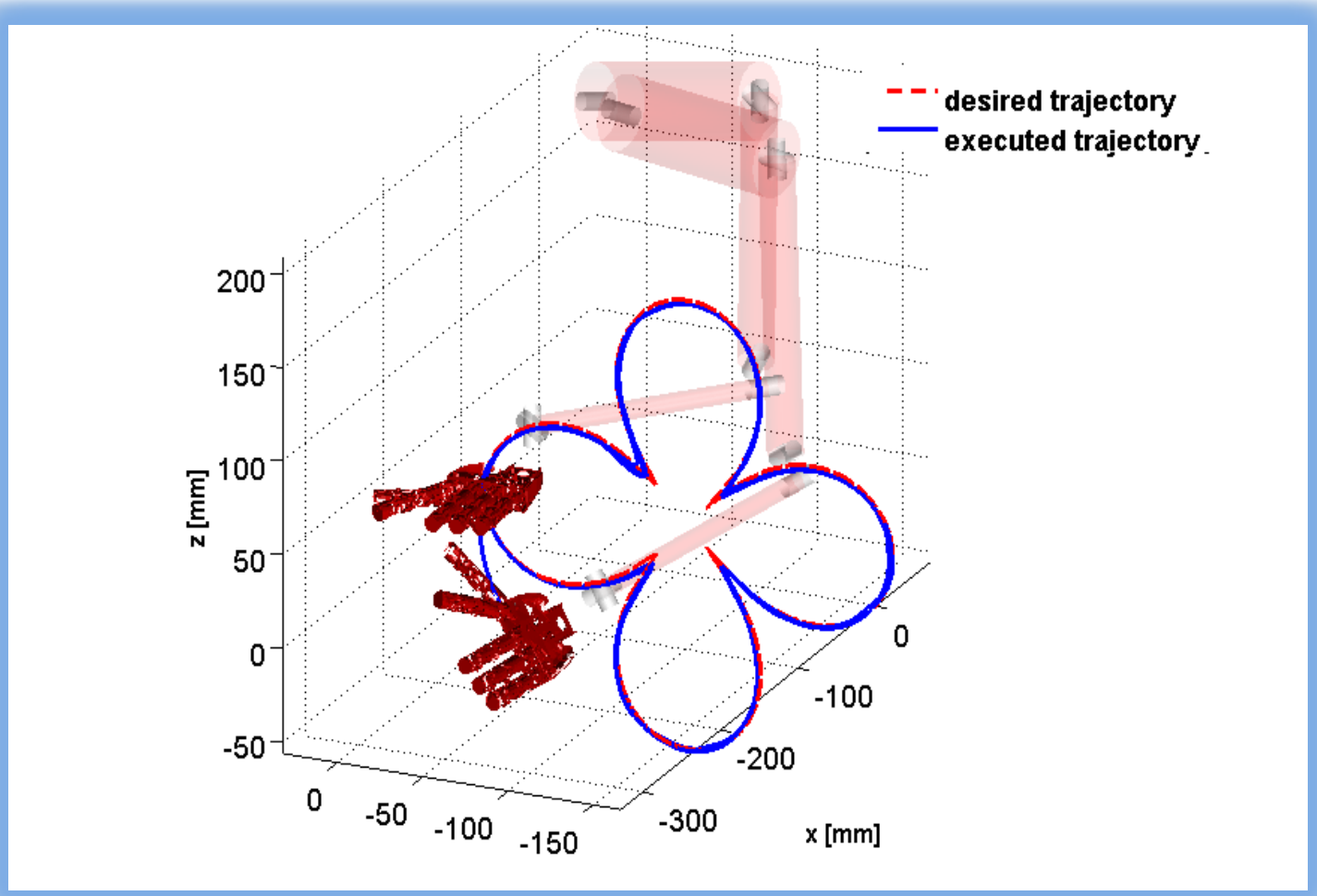
Outline

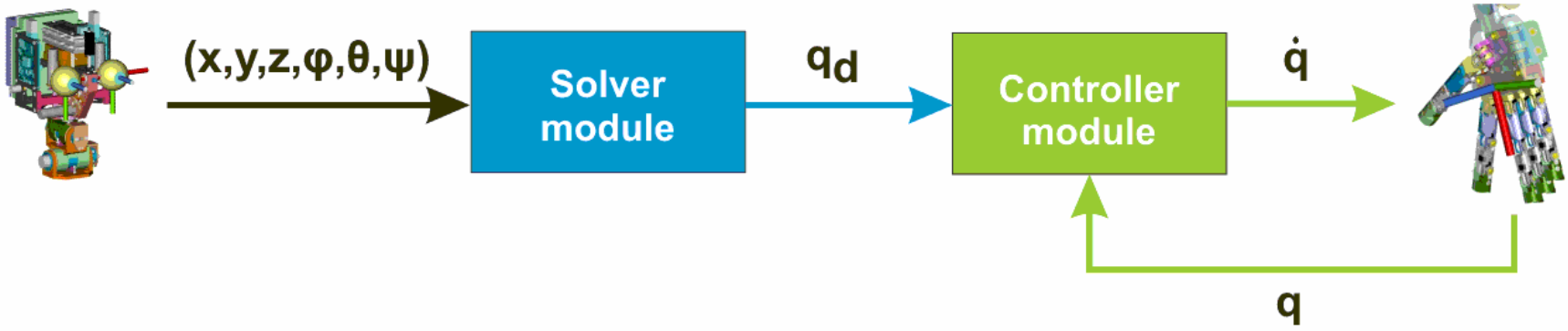
- Theory: Cartesian Controller
- Theory: Gaze Controller

- Installation
- Tutorials

The Problem


desired trajectory
executed trajectory
z [mm]
200
150
100
50
0
-50
0
-50
-100
-150
-300
-200
-100
0
x [mm]

Cartesian Controller Structure



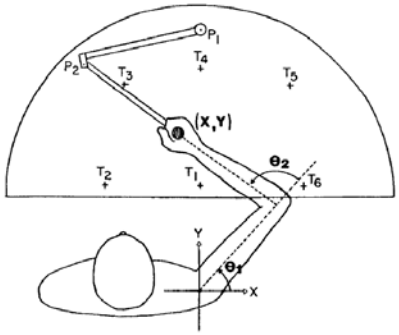
The Solver: the IpOpt choice

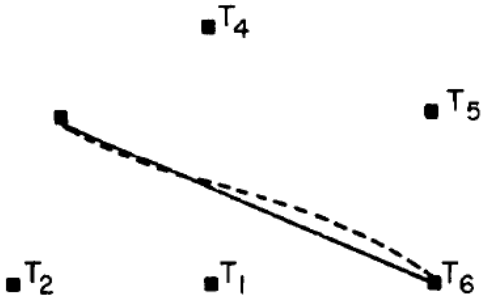
$$\tilde{q}_d = \arg\min_{q \in \mathbb{R}^n} \left(\left\| \alpha_d - K_\alpha(q) \right\|^2 + \lambda \cdot (q_{\text{rest}} - q)^T W (q_{\text{rest}} - q) \right)$$

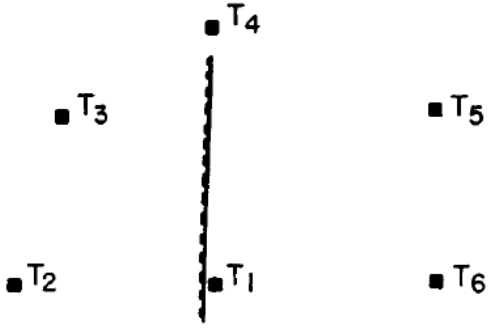
$$\text{s.t.} \begin{cases} \left\| x_d - K_x(q) \right\|^2 < \varepsilon \\ q_L < q < q_U \\ \text{other obstacles ...} \end{cases}$$

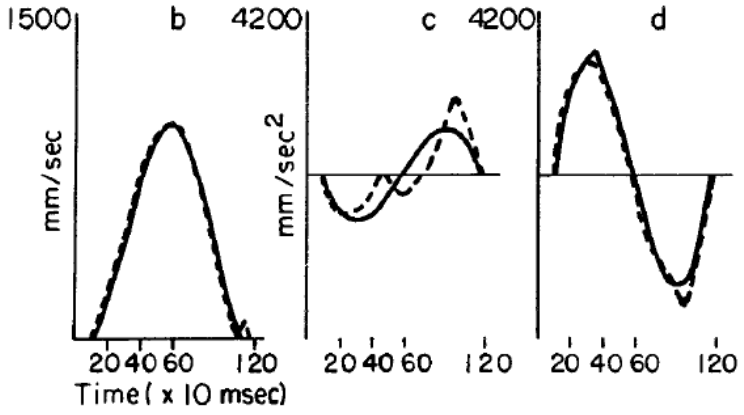
- Quick convergence (real-time compliant: < **20 ms**)
- Scalability
- Singularities and joints bound handling
- Tasks hierarchy
- Complex constraints

The Controller: Trajectory Generation

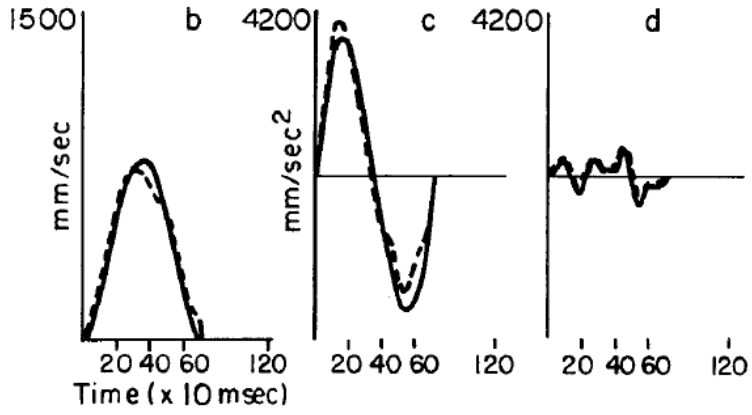






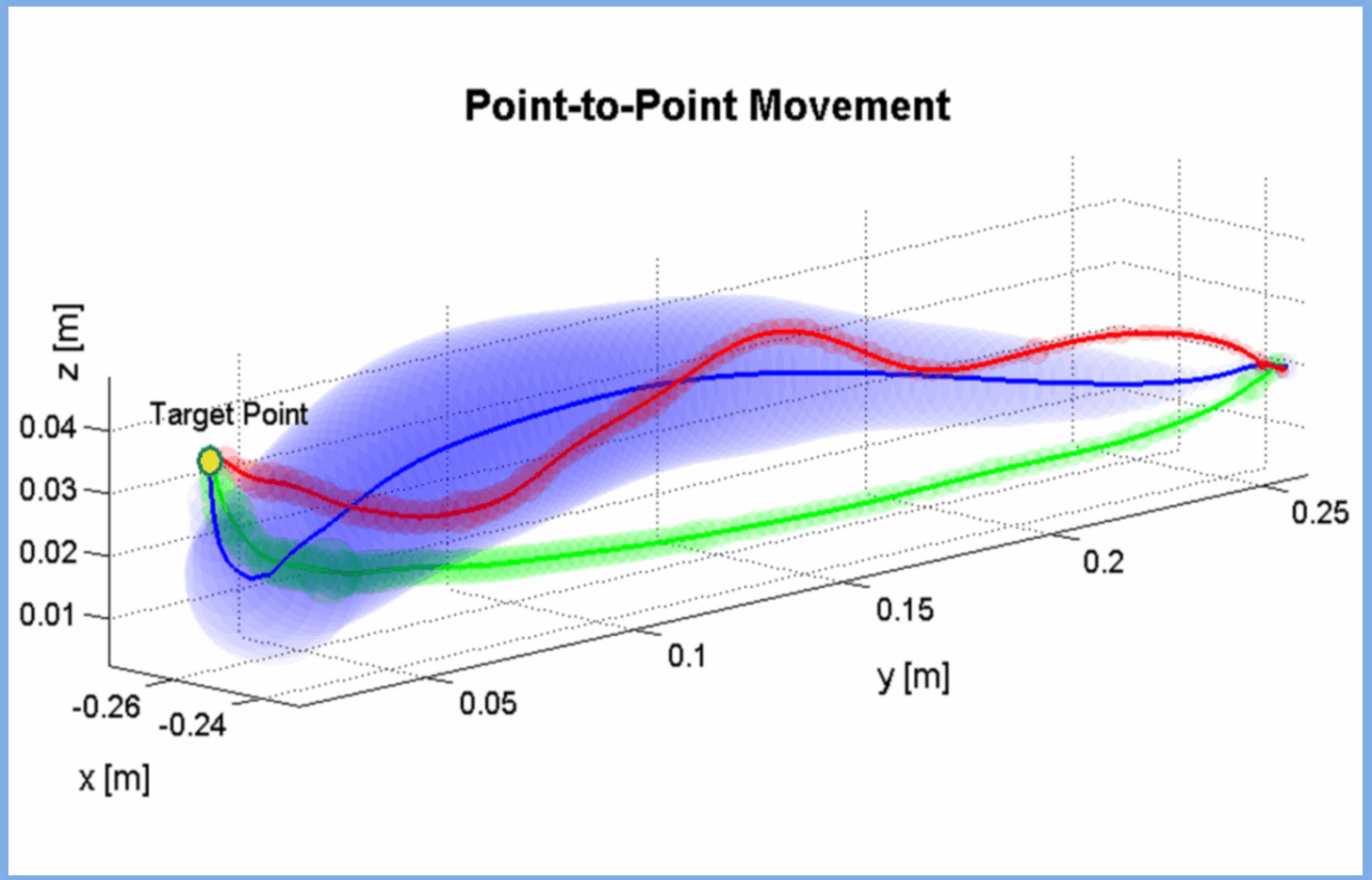


A



B

Evaluation: the P2P case

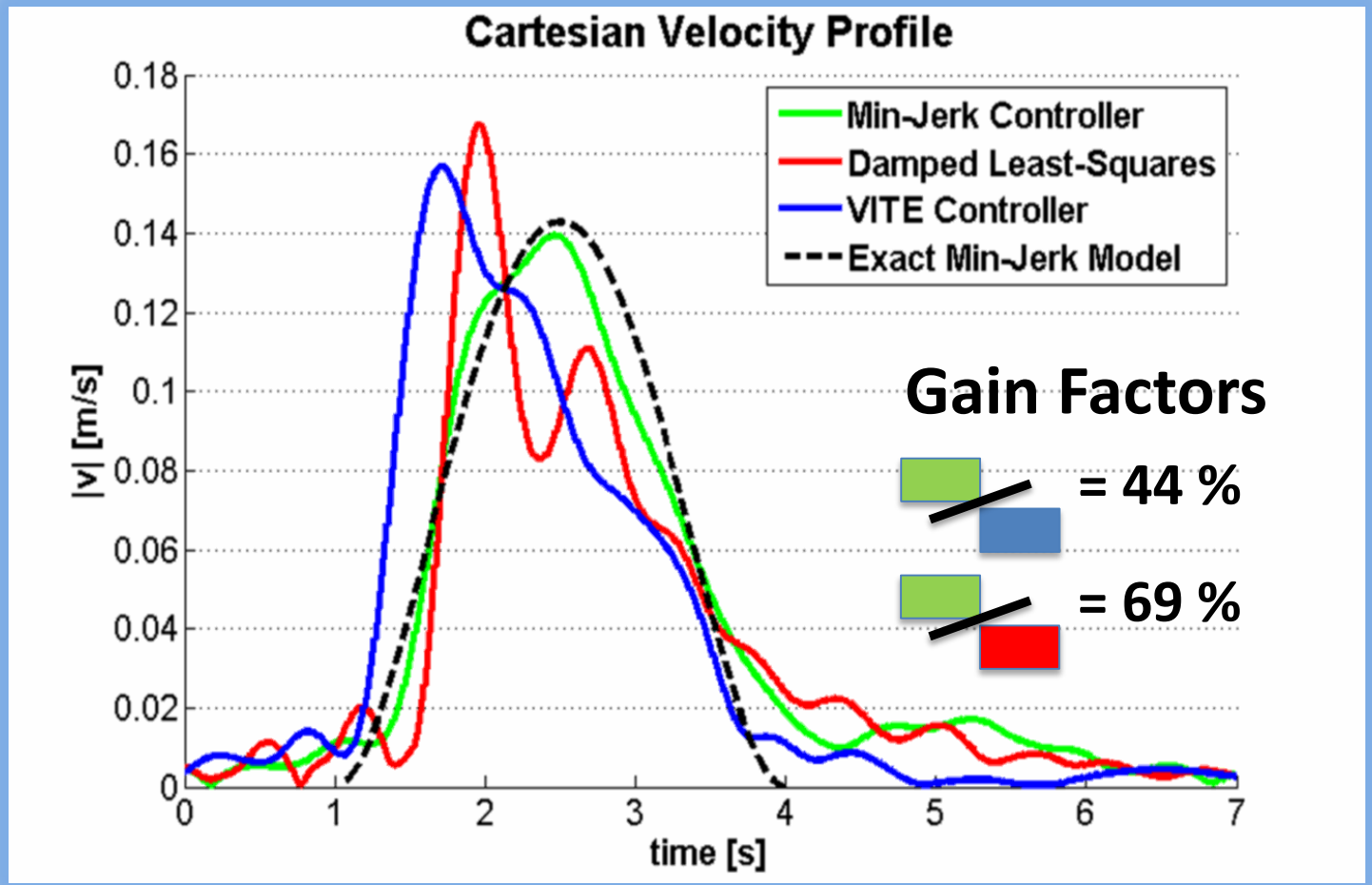


Min-Jerk

DLS

VITE

Is it Minimum-Jerk in the Task-Space?

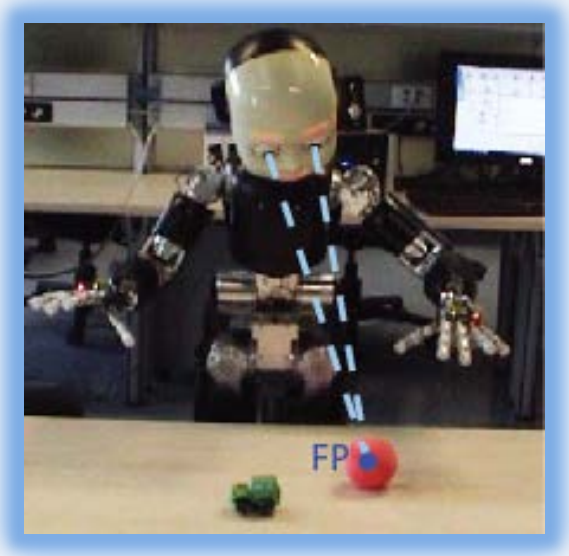


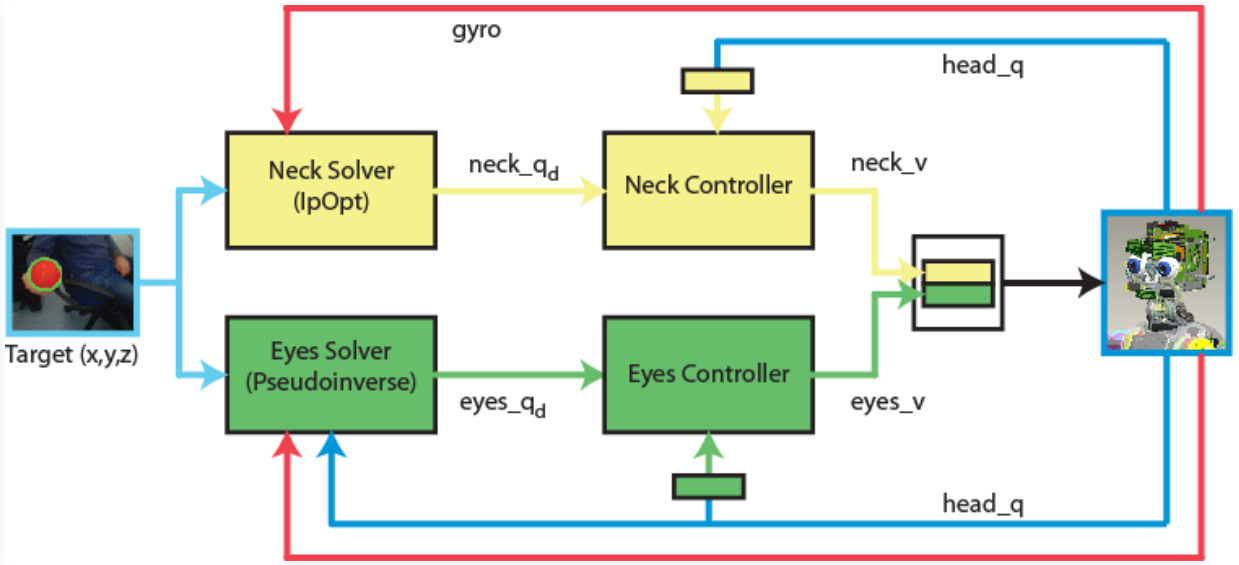
Min-Jerk

DLS

VITE

The Gaze Controller (1/7)

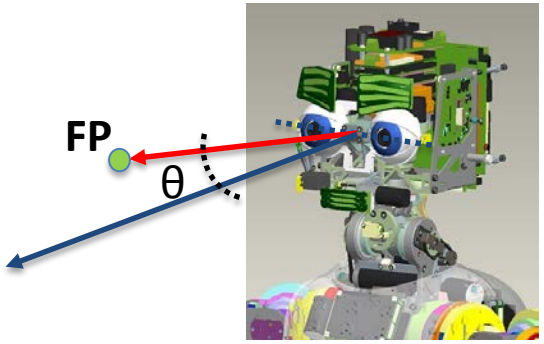




Yet another Cartesian Controller: **reuse ideas** …

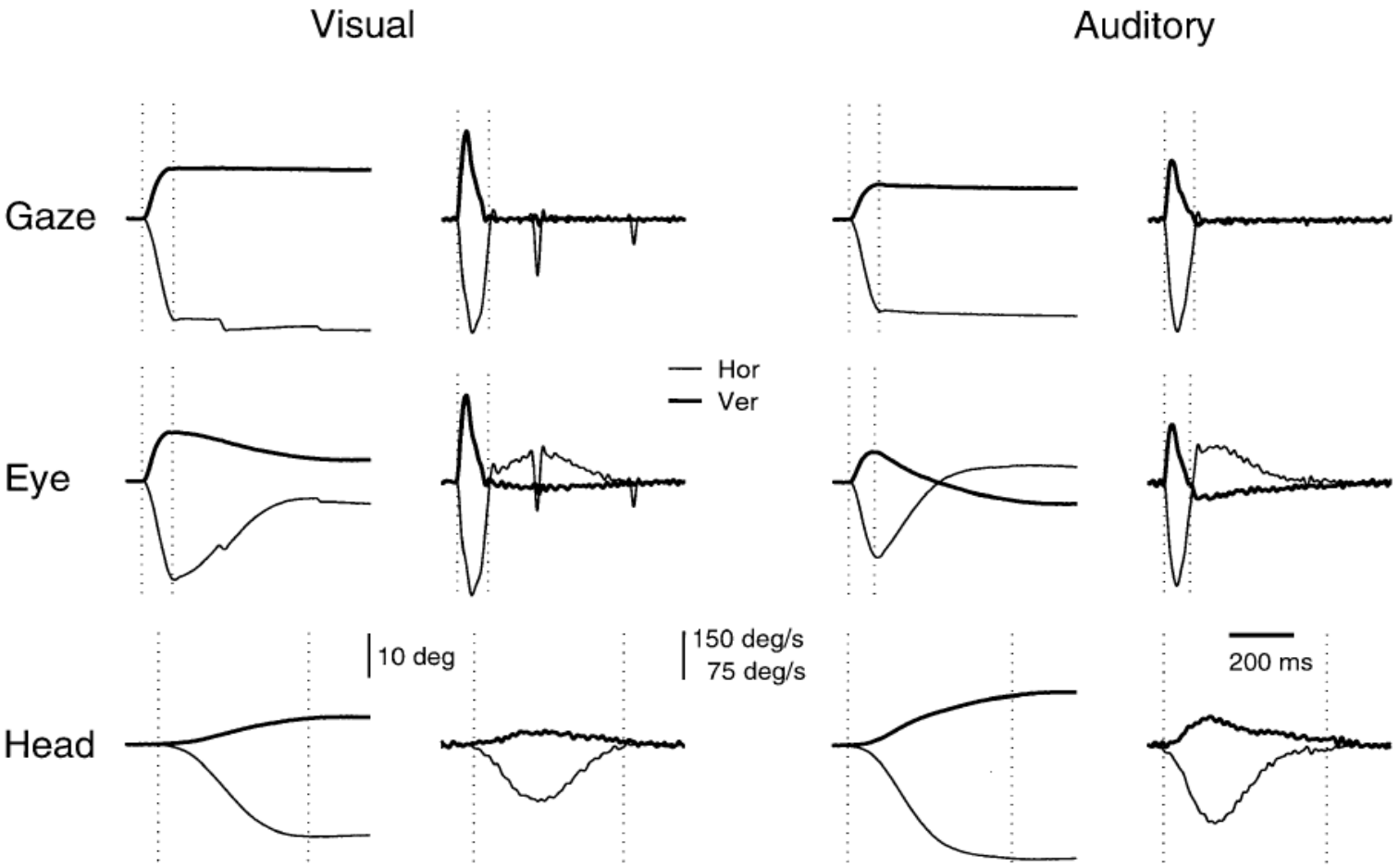
Then, apply easy transformations from Cartesian to …

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



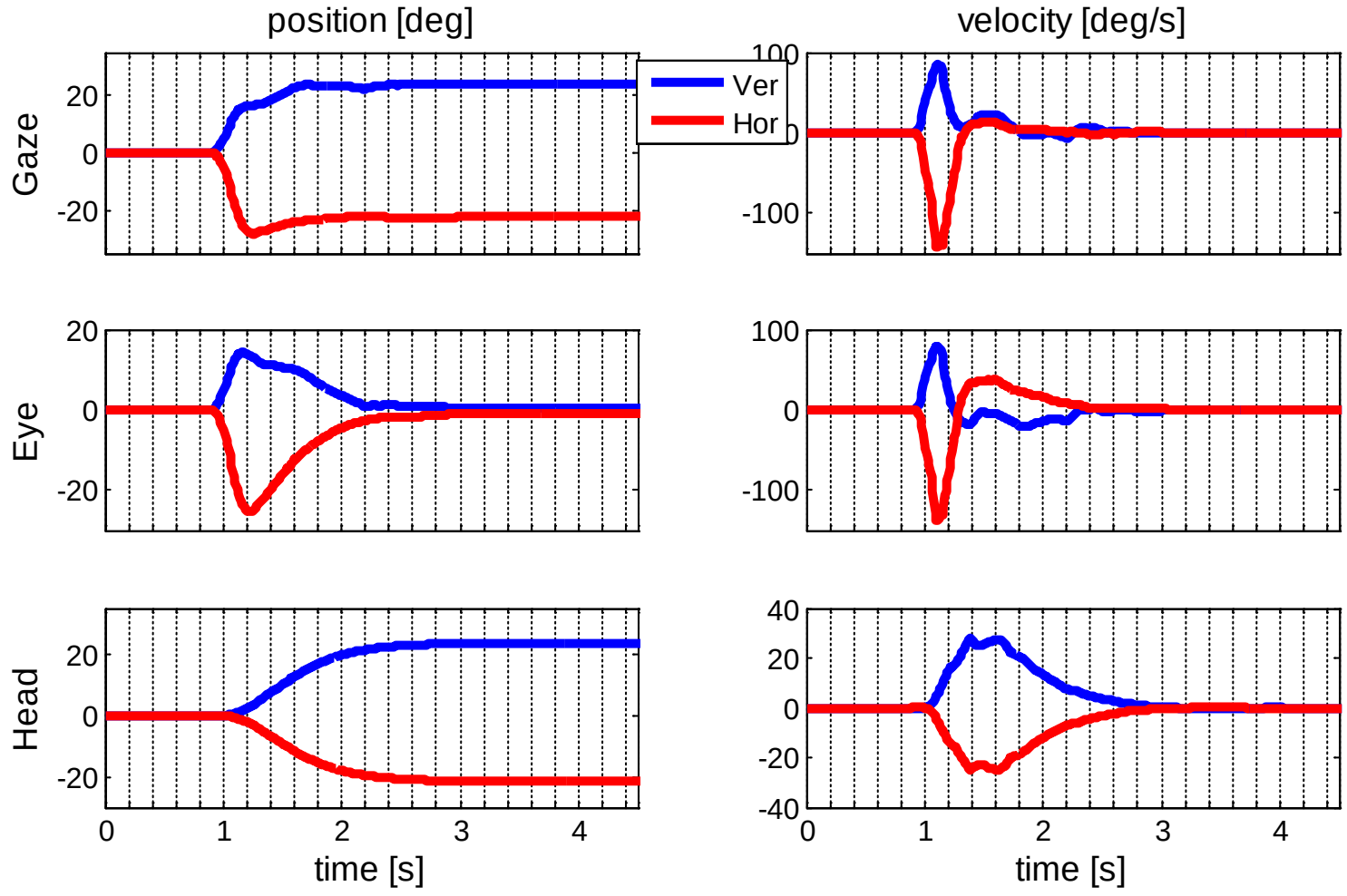
The Gaze Controller (2/7)

Studies on humans …

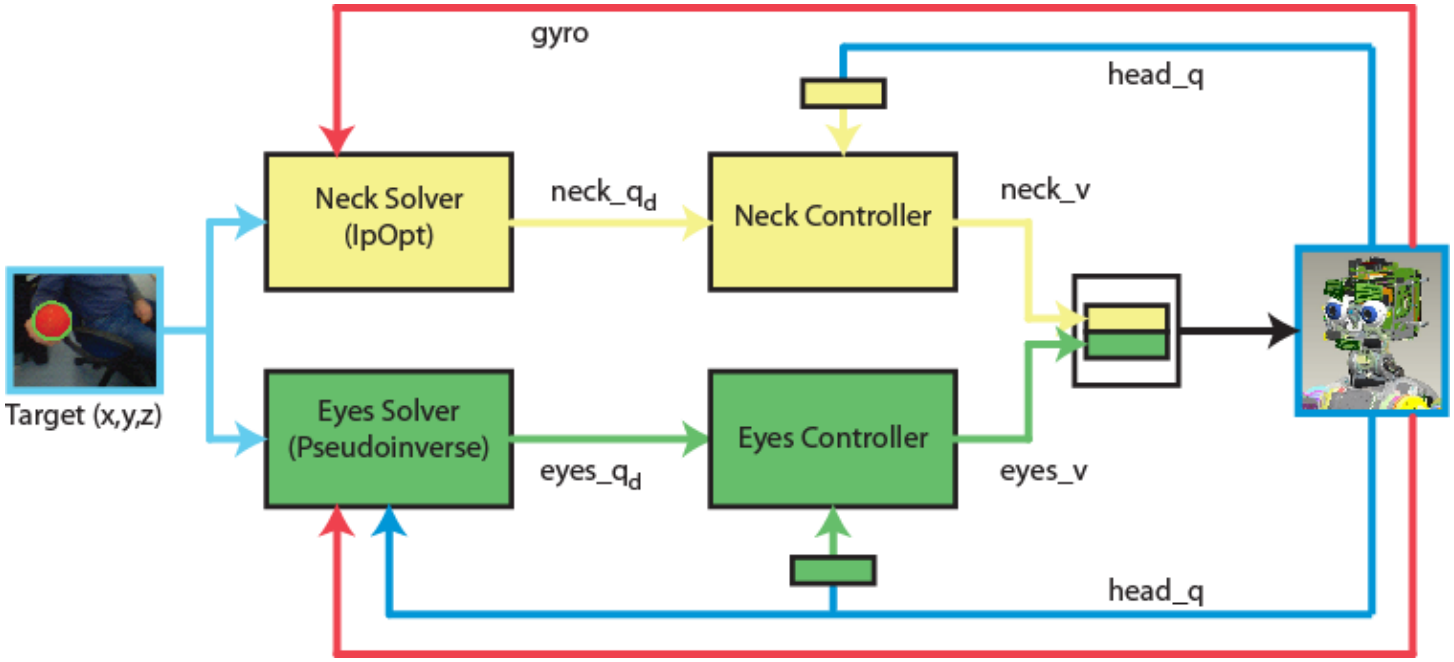


The Gaze Controller (3/7)

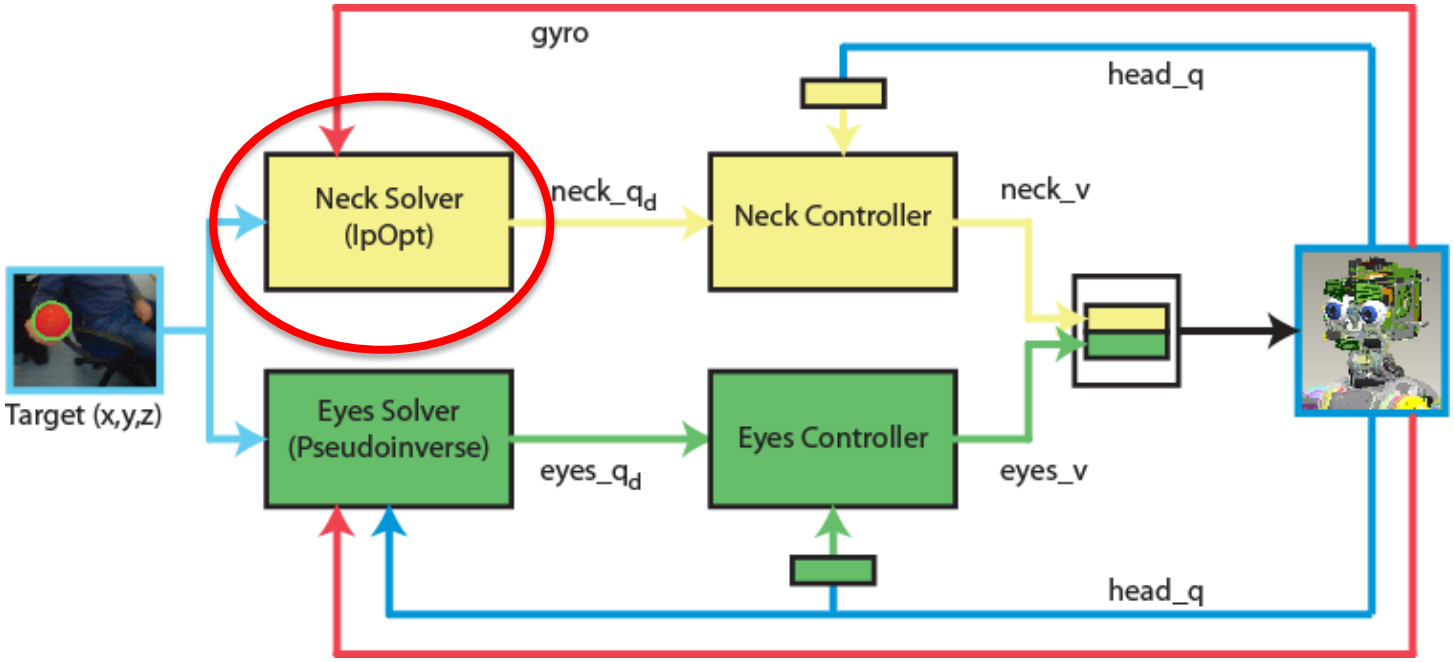
Results on iCub …

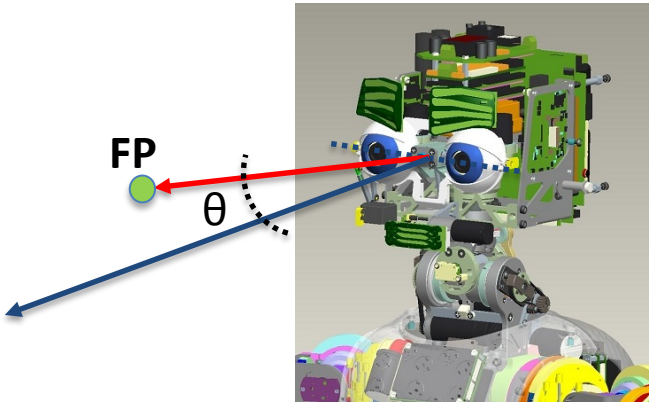


The Gaze Controller (4/7)



The Gaze Controller (5/7)

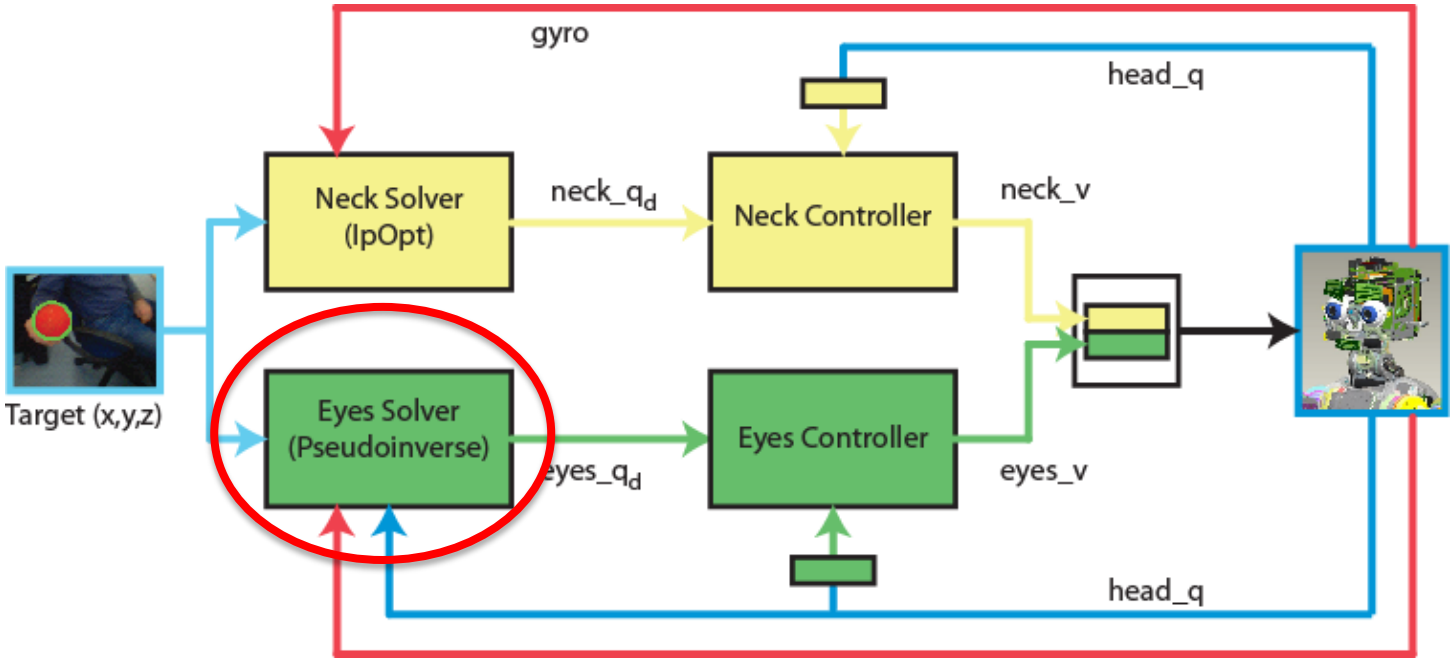




$$q^*_{\text{neck}} = \arg \min_{q_{\text{neck}} \in \mathbb{R}^3} \left\| q_{\text{rest}} - q_{\text{neck}} \right\|^2$$

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

The Gaze Controller (6/7)

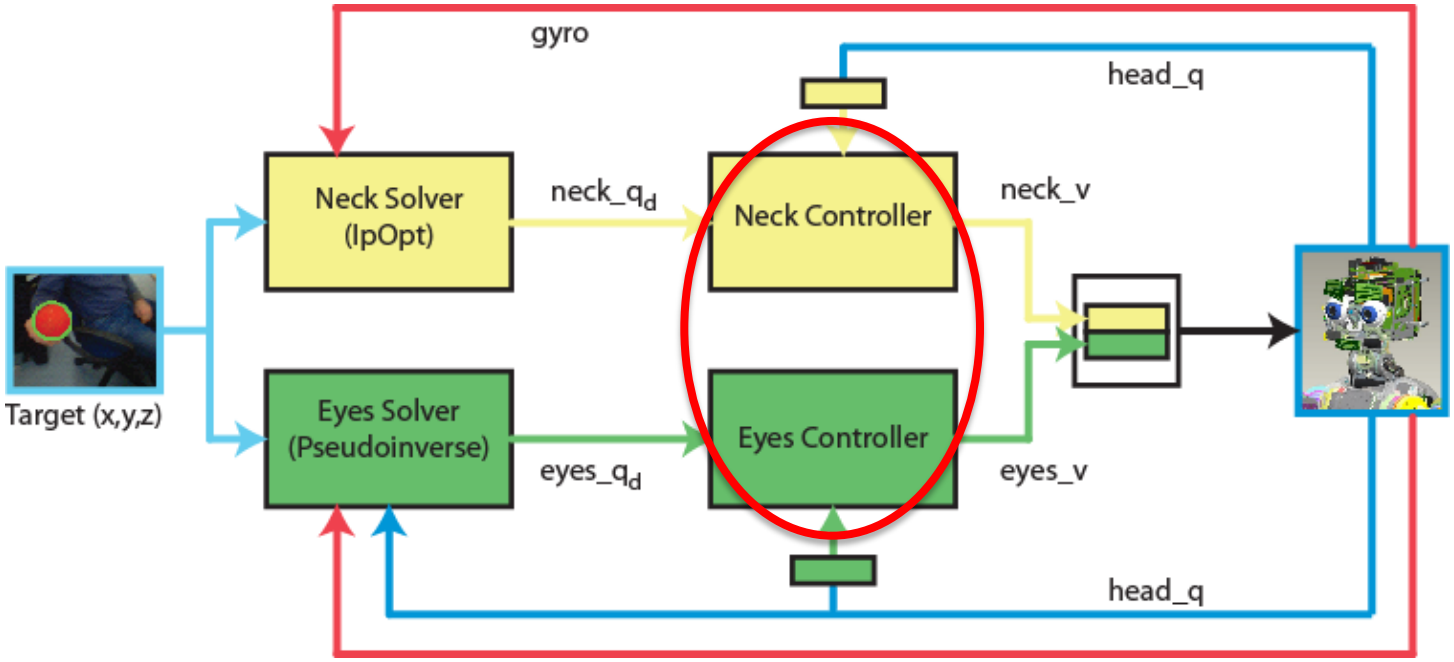


$$q^{*}_{\text{eyes}} = \arg \min_{q_{\text{eyes}} \in \mathbb{R}^3} \left\| FP_d - K_{FP}\left(q_{\text{eyes}}\right) \right\|^2$$

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

Gyro

The Gaze Controller (7/7)



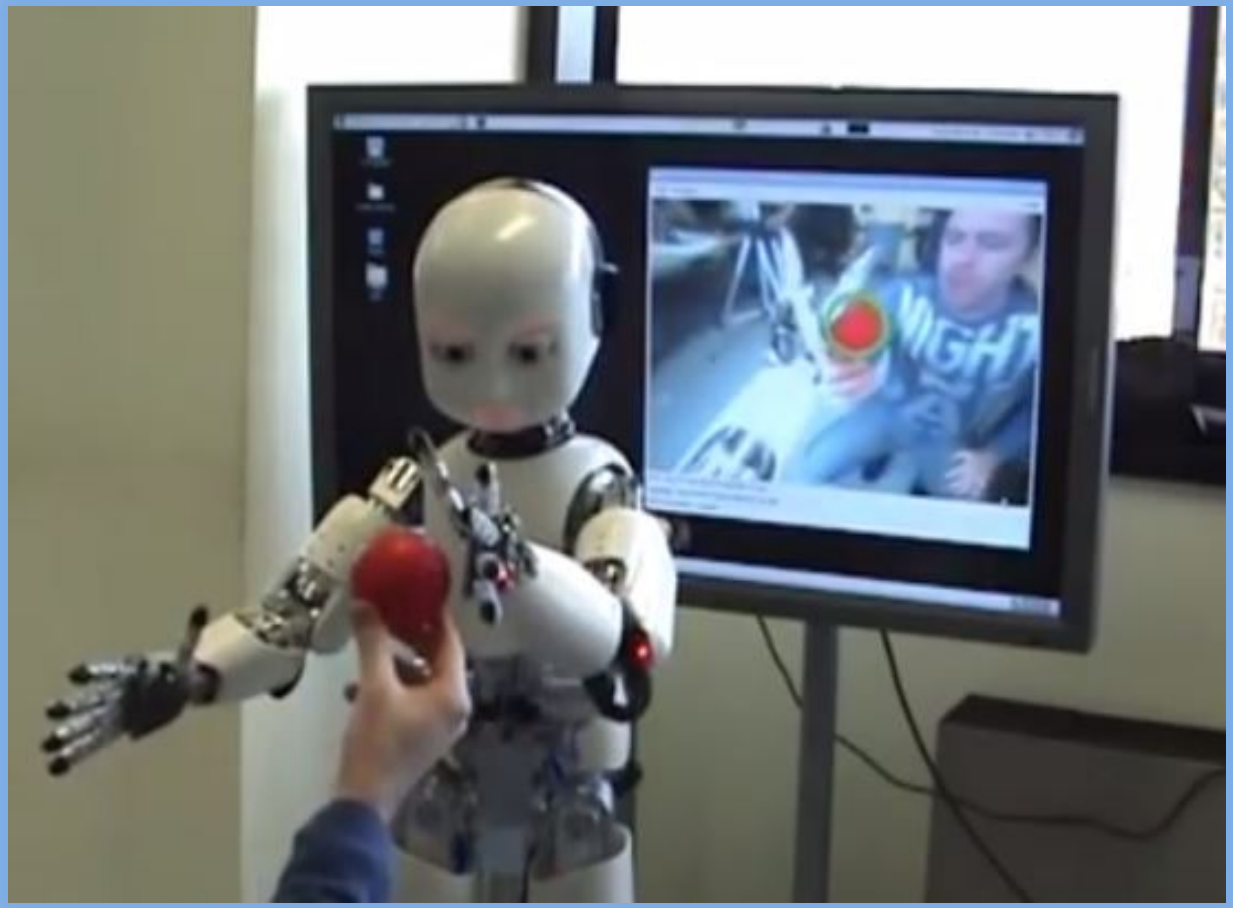
Retain Controllers Laws

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

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

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

An old video…

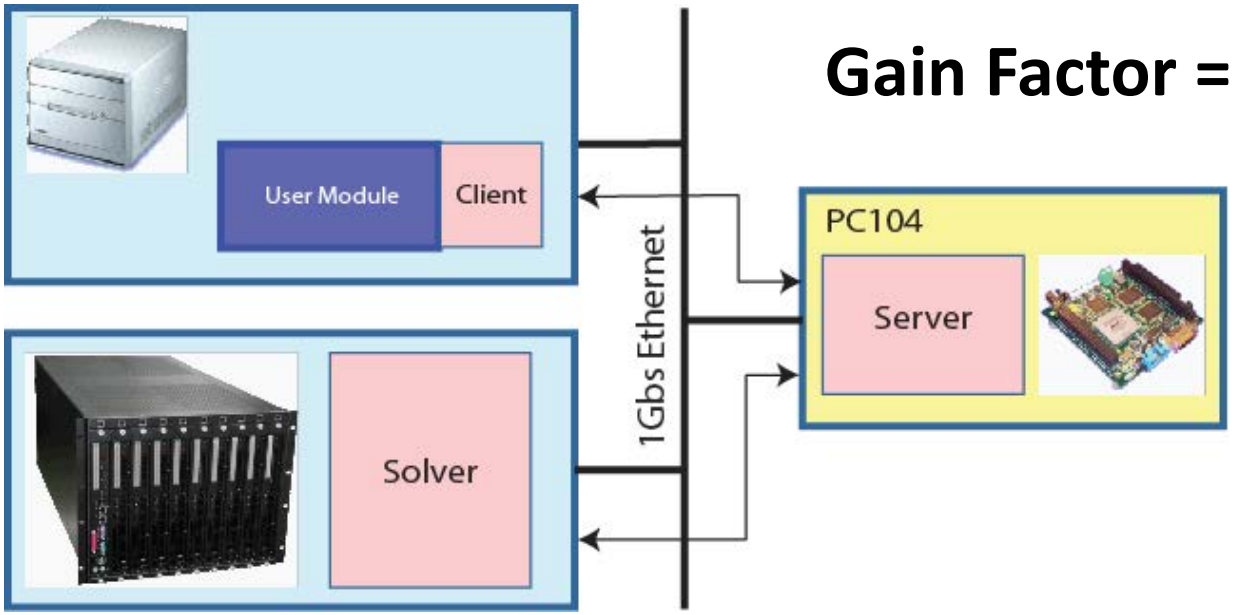


https://www.youtube.com/watch?v=LMGSok5tN4A

Software Development

Abstraction Layers from low to high:

1. Open-source library **iKin** for generic kinematic chains (vs. KDL)
2. Design YARP **Cartesian Interfaces**



Gain Factor = 50%

Function Call → Task-Space Movement

Enabling Device Drivers (1/2)

IpOpt installation: http://wiki.icub.org/wiki/Installing_IPOPT

Tick the drivers from within the CMake mask

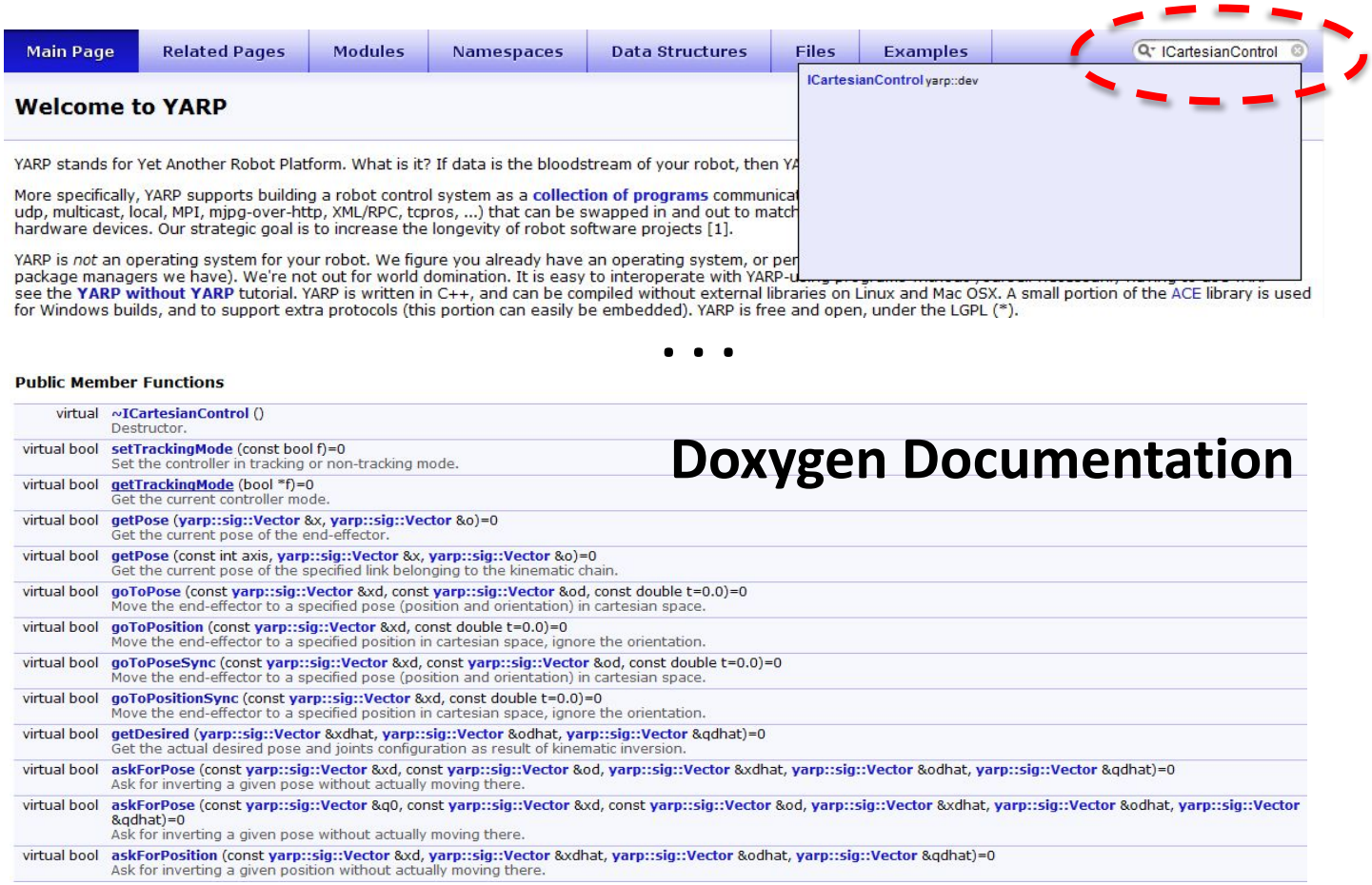
Enabling Device Drivers (2/2)

Check the final availability with "*icubmoddev --list*"

```
C:\DEV\work>icubmoddev --list
Here are devices listed for your system:
Device "test_grabber", C++ class TestFrameGrabber, wrapped by "grabber"
Device "test_motor", C++ class TestMotor, wrapped by "controlboard"
Device "remote_grabber", C++ class RemoteFrameGrabber, wrapped by "grabber"
Device "grabber", C++ class ServerFrameGrabber, is a network wrapper.
Device "inertial", C++ class ServerInertial, is a network wrapper.
Device "sound_grabber", C++ class ServerSoundGrabber, is a network wrapper.
Device "pipe", C++ class DevicePipe, has no network wrapper
Device "group", C++ class DeviceGroup, has no network wrapper
Device "remote_controlboard", C++ class RemoteControlBoard, wrapped by "controlboard"
Device "controlboard", C++ class ServerControlBoard, is a network wrapper.
Device "analogsensorclient", C++ class AnalogSensorClient, has no network wrapper
Device "cartesiancontrollerserver", C++ class ServerCartesianController, wrapped by "cartesiancontrollerclient"
Device "cartesiancontrollerclient", C++ class ClientCartesianController, is a network wrapper.
Device "gazecontrollerclient", C++ class ClientGazeController, is a network wrapper.
```

Interfaces Documentation

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



Doxygen Documentation

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 (1/4)

CMAKE

```
find_package(ICUB)
…
include_directories($ICUB_INCLUDE_DIRS)
…
target_link_libraries(${PROJECTNAME} icubmod)
```

CODE SKELETON

```
...
#include <yarp/dev/all.h>
YARP_DECLARE_DEVICES(icubmod)
...
int main()
{
YARP_REGISTER_DEVICES(icubmod)
...
}
```

Interfaces Communalities (2/4)

OPENING THE CARTESIAN INTERFACE

```
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);
}
```

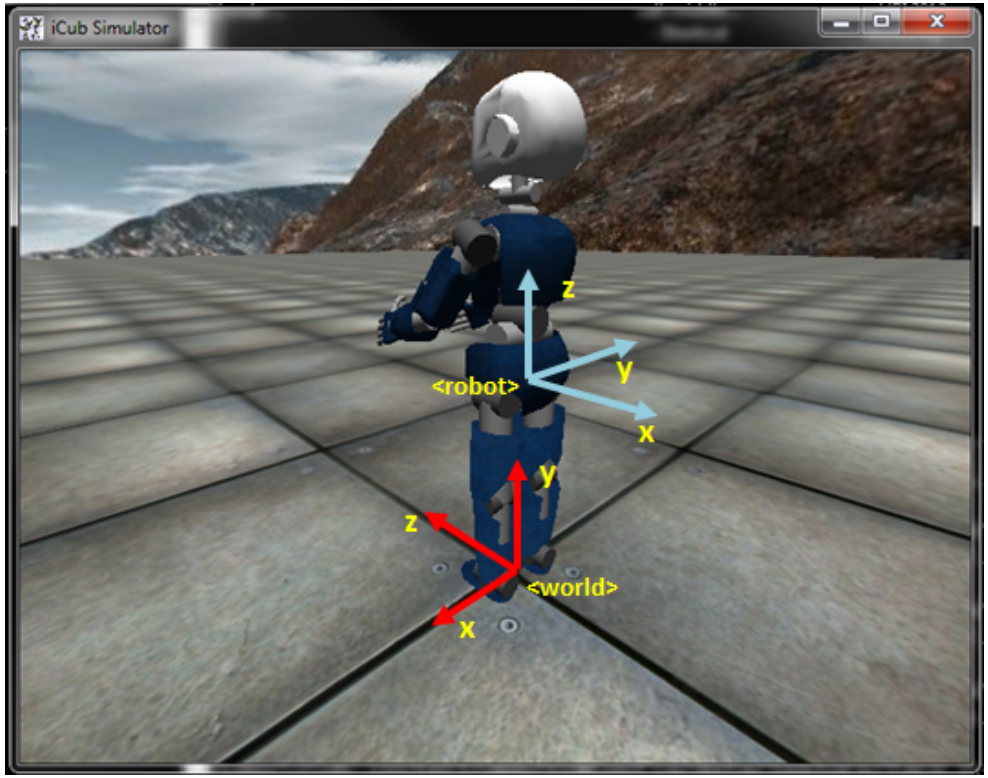
Interfaces Communalities (3/4)

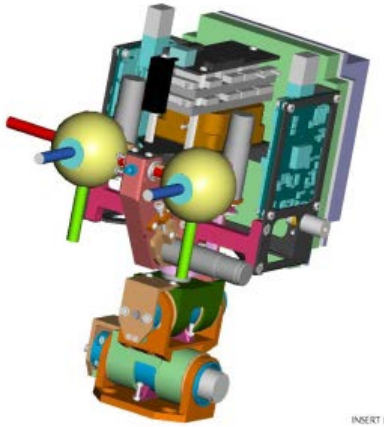
OPENING THE GAZE INTERFACE

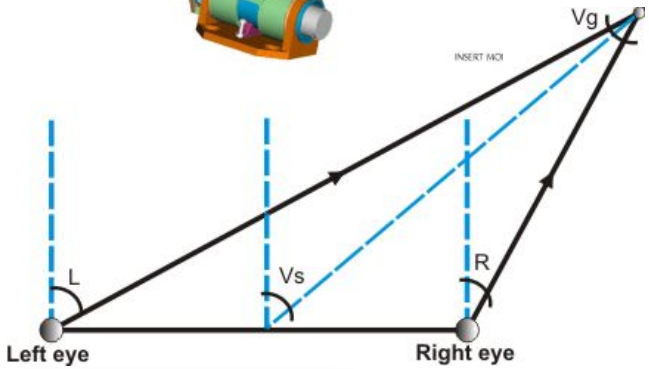
```
Property option;

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);
}
```

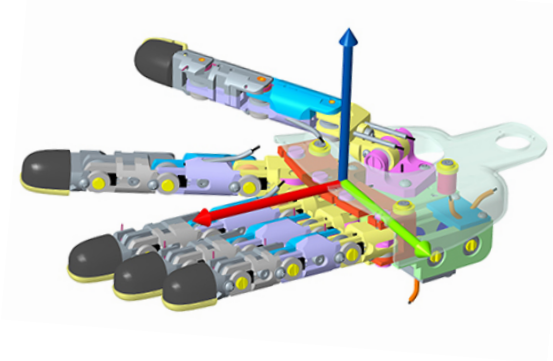
Interfaces Communalities (4/4)

Coordinate Systems





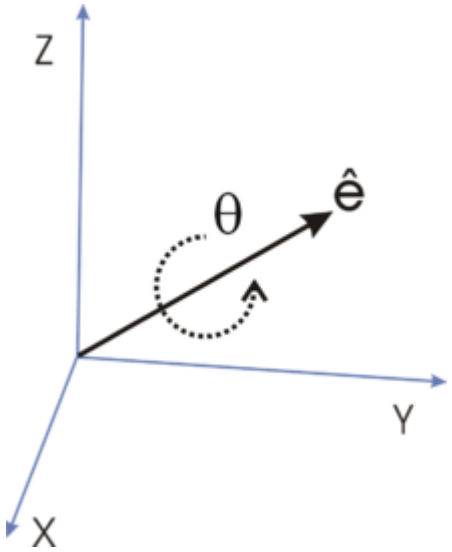




Cartesian Interface (1/7)

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)

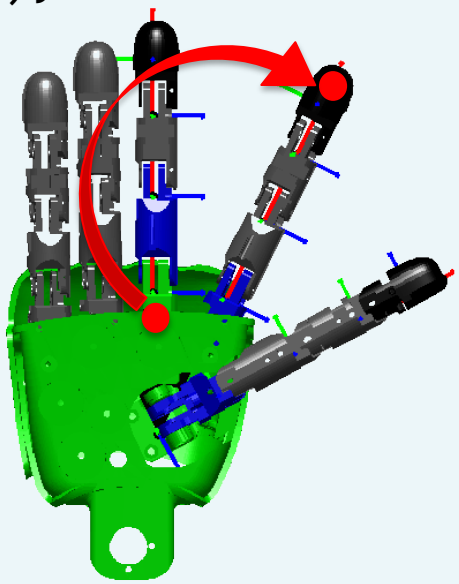
CONTEXT SWITCH

```
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> 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 (1/4)

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] => elevantion [deg]
// ang[2] => vergence [deg]
```

LOOK AT 3D POINT

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

… IN ANGULAR DOMAIN

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

Gaze Interface (2/4)

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 (3/4)

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 (4/4)

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");
```