ISTITUTO ITALIANO
DI TECNOLOGIA

iDynTree :
Free Floating Dynamics Library

Gabriele Nava

Stefano Dafarra

iCub Facility

Istituto Italiano di Tecnologia, Genova, Italy

Slides taken from VVV18 International Winter School on Humanoid Robot Programming
Credits Silvio Traversaro

Loading model

Models can be built programmatically but typically are loaded from **URDF** files (the same robot description format used in **ROS**).

As sometimes we are not interested in all joints contained in a robot, we can explicitly specify the **joints** that we want to load, and their order, using the **iDynTree::ModelLoader** class.

From this class, you obtain a **iDynTree::Model** that contain the structure and the parameters of the model, that you can pass to **iDynTree::KinDynComputations** that

```
ModelLoader mdlLoader;

std::vector<std::string> joints; // specify the used joints
joints.push_back("torso_pitch");
...
joints.push_back("r_wrist_pitch");

mdlLoader.loadReducedModelFromFile("./model.urdf", joints);

Model model = mdlLoader.model();

KinDynComputations kinDynComp;
kinDynComp.loadRobotModel(mdlLoader.model());

// You can now use the kinDynComp object to compute terms
// of the dynamics equations, transformation between
// frames, jacobians
```

Caveat on iDynTree <--> YARP integration

- Conversion utilities between **YARP** and **iDynTree** are available in the `<iDynTree/yarp/YARPConversions.h>` header.

- **iDynTree** uses *radians* for all its interfaces, **YARP** typically uses *degrees*.

- **iDynTree** assume in input and in output always *floating base quantities*, you will need to do to the appropriate conversions if you want to use them for **fixed base robots** control.

- See the **impedance_control-tutorial** for more on this!

ISTITUTO ITALIANO
DI TECNOLOGIA

Joint Level Motor Control

(for whole-body dynamic control)

Gabriele Nava

Stefano Dafarra

iCub Facility

Istituto Italiano di Tecnologia, Genova, Italy

Slides taken from VVV18 International Winter School on Humanoid Robot Programming
Credits Silvio Traversaro

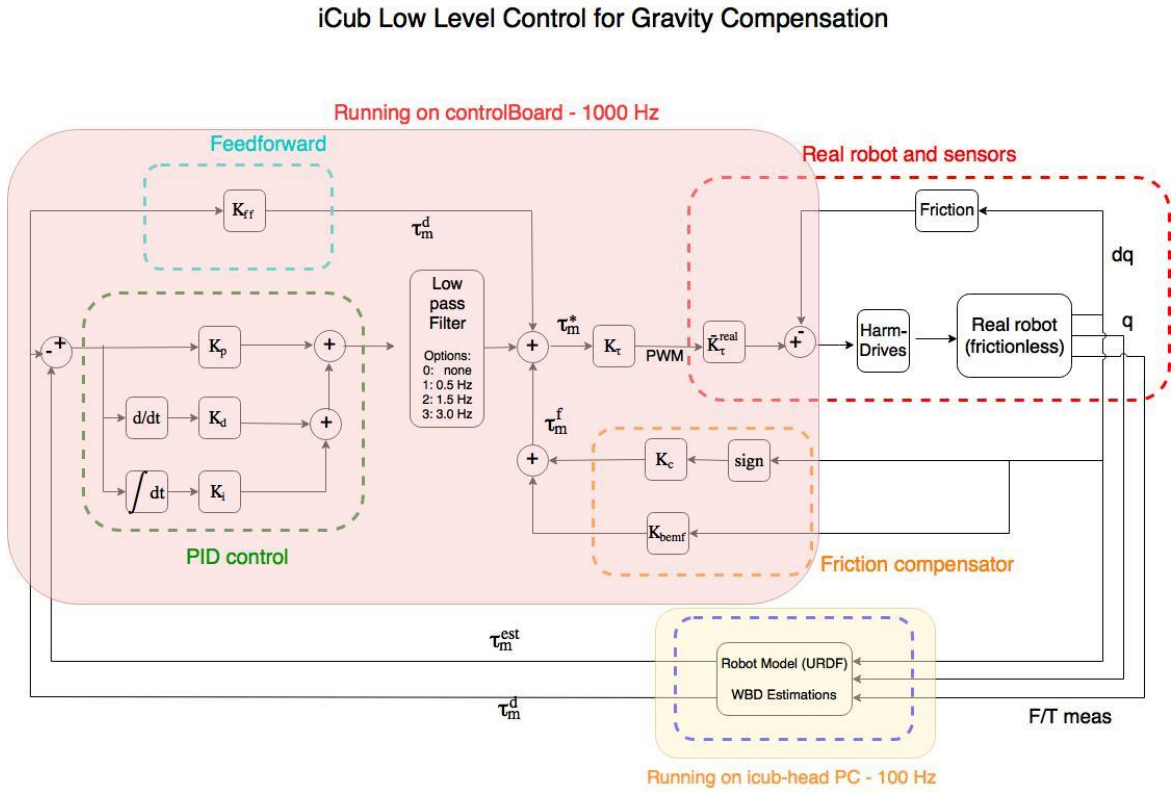
Torque Control (iCub, Feb 2018)

τ_d

Command
from YARP

τ_m

Feedback to YARP

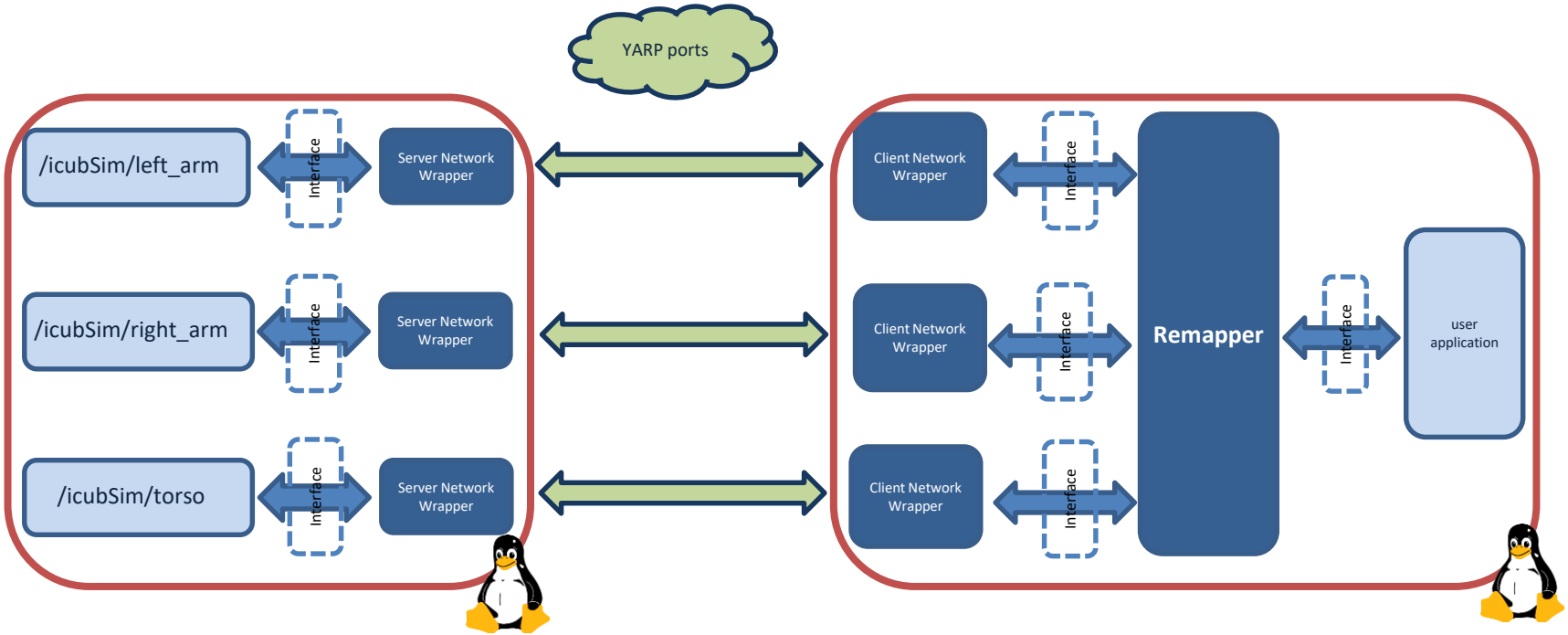


Interface: ITorqueControl

Like IPositionControl2 and IVelocityControl2, but for torque control

```
ITorqueControl::getAxes() = 0;
ITorqueControl::setRefTorque(…) = 0;
ITorqueControl::getTorque(…) = 0;
```

Whole-body hardware abstraction



The remapper only exposes the joint required by the user application.
Similar concept in `ros_control` : **CombinedRobotHW**

Getting Interfaces only for the desired joints

The device that combines multiple "remote_controlboard" devices by specifying the desired joints is "remotecontrolboardremapper"

Required parameter to configure it are:

- List of remote port prefixes: remoteControlBoards
- Local port name: localPortPrefix
- Ordered list of desired joints: axesNames

```
PolyDriver poly;
Property config;

config.put("device", "remotecontrolboardremapper");
config.put("localPortPrefix", "/<myApplication>");

Bottle boards;
Bottle & boardsList = boards.addList();
boards.addString("/icubSim/torso");
boards.addString("/icubSim/left_arm");
boards.addString("/icubSim/right_arm");
options.put("remoteControlBoards", boards.get(0));

Bottle joints;
Bottle & jointsList = joints.addList();
joints.addString("torso_pitch");
...
joints.addString("r_wrist_yaw");
options.put("axesNames", joints.get(0));

...

poly.open(config);
```

```
ITorqueControl *trqControl = NULL;
poly.view(trqControl);                  // Get the interface


int joints;
trqControl->getAxes(&joints);           // Get number of joints

yarp::sig::vector desTorques(joints, 0.0);
trqControl->setRefTorques(desTorques.data()); // Set a desired torques setpoint

yarp::sig::vector measTorques(joints, 0.0);
trqControl->getTorques(measTorques.data());   // Get the measured (or estimated) torques
```