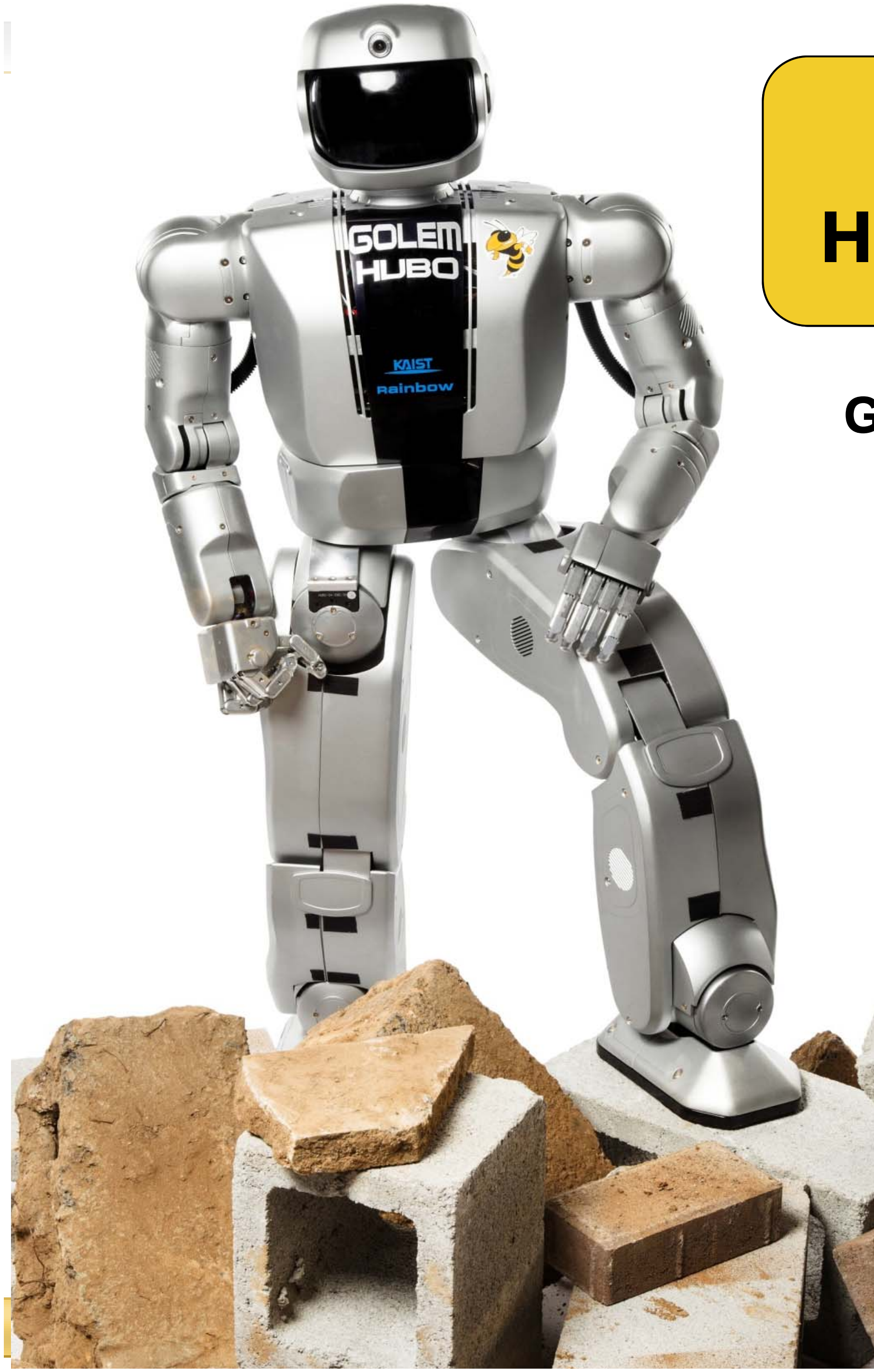


DRC-HUBO TEPRA: Hubo-Ach + Hubo-Motion

Georgia Tech and Drexel University

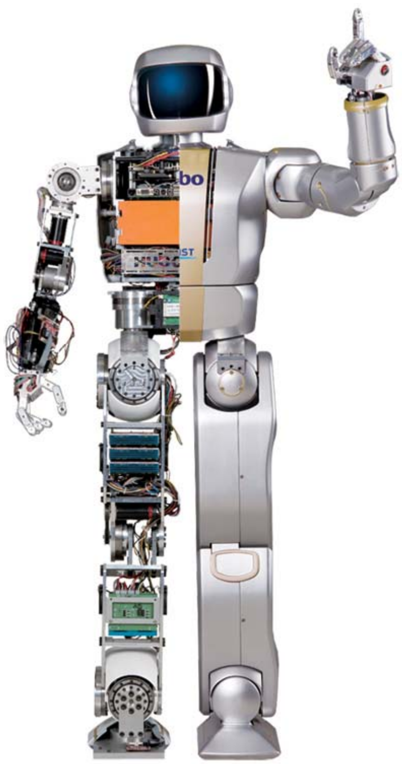
M. Stilman, M.X. Grey, D. Lofaro, P.Oh, N. Dantam, P. Vieira, R. O'Flaherty, S. Joo

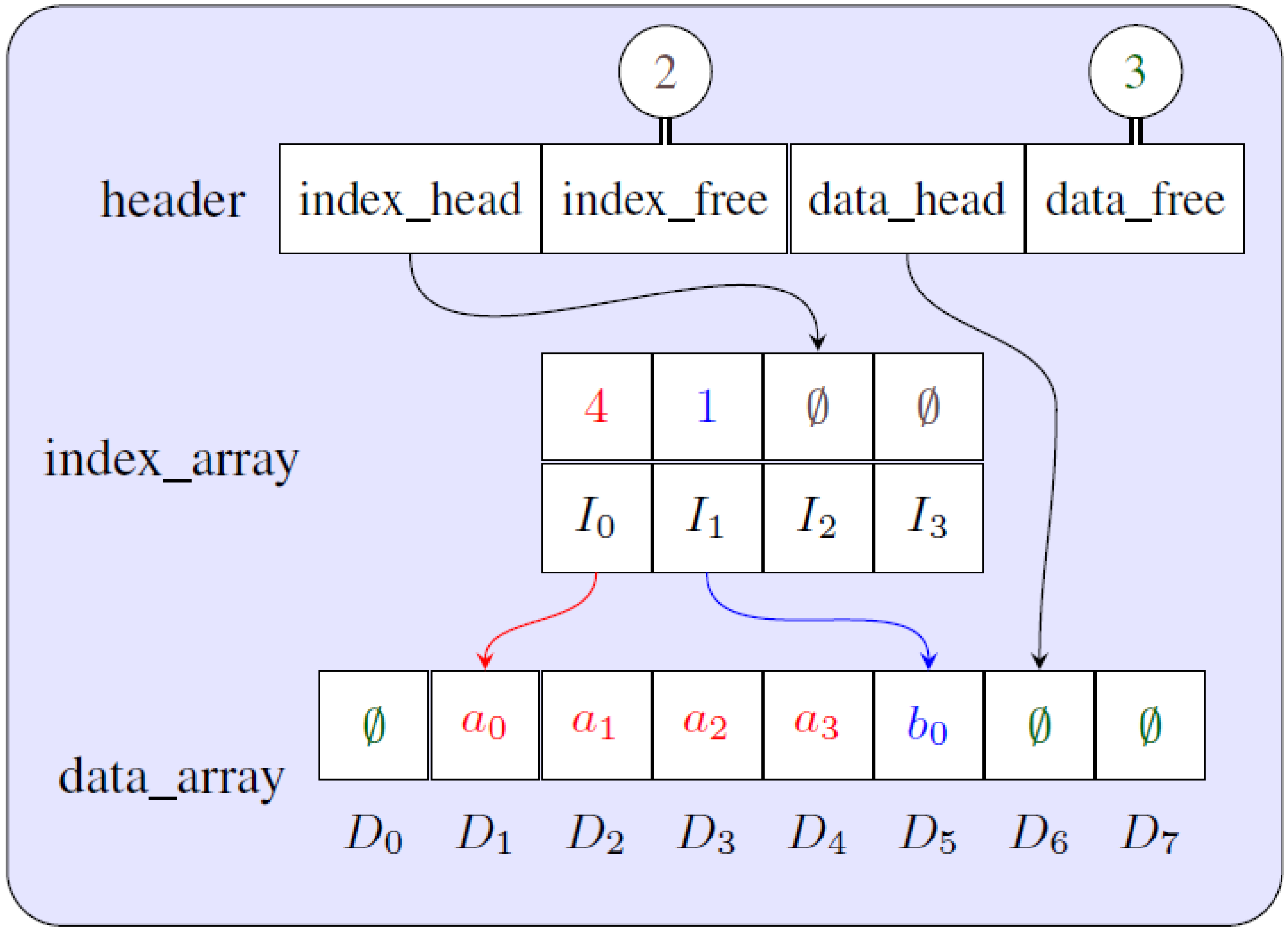
mstilman@cc.gatech.edu

Center for Robotics and Intelligent Machines Georgia Tech

Drexel University

The Ach Multi-Process Linux Architecture

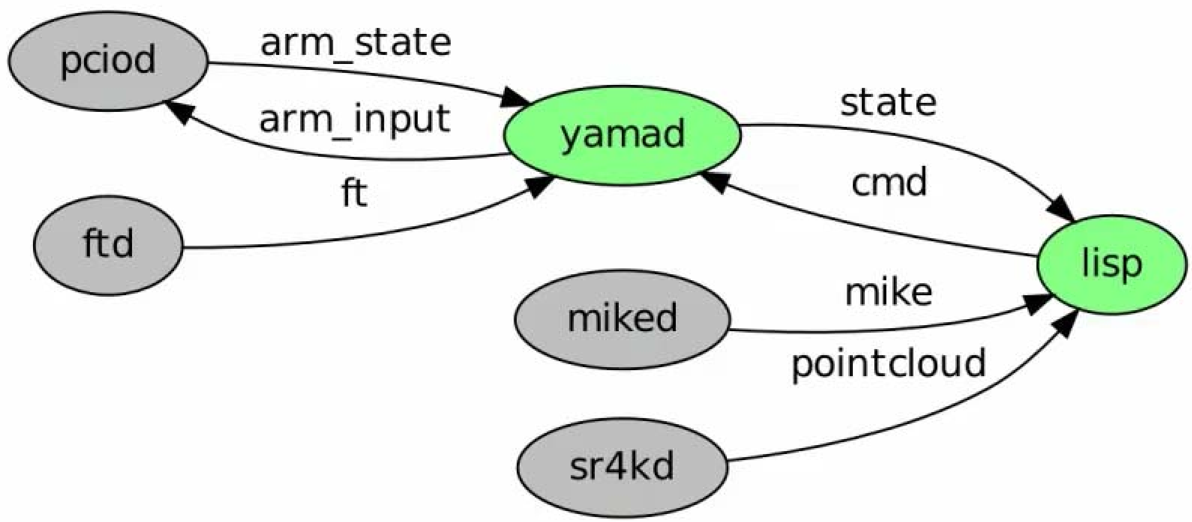




Physical Human-Robot Games: Yamakuzushi

Neil Dantam and Mike Stilman

May 2010



Advantages of Ach (1)

1) **Formally Verified: Promela Model of the core Ach functions and verified it using the SPIN model-checker. No need to worry about synch.**

2) **No HOL Blocking: Always gives the newest data. We can always compute the newest message in the channel in O(1) time.**

3) **Read Older Data: Ach offers older data as best it can. Any messages in the circular buffer which have not been overwritten can still be read.**

Advantages of Ach (2)

4) **Multiple Senders and Receivers**: Ach supports all combinations of communications between M senders and N receivers with only M + N file descriptors. Typical communication methods open sockets between every reader and writer

5) Priorities: Ach obeys **real-time priorities**. Each channel is protected by a mutex and condition variable, thus the kernel decides which process gets the next access to a given channel.

Advantages of Ach (3)

6) Priority sustenance and Inheritance

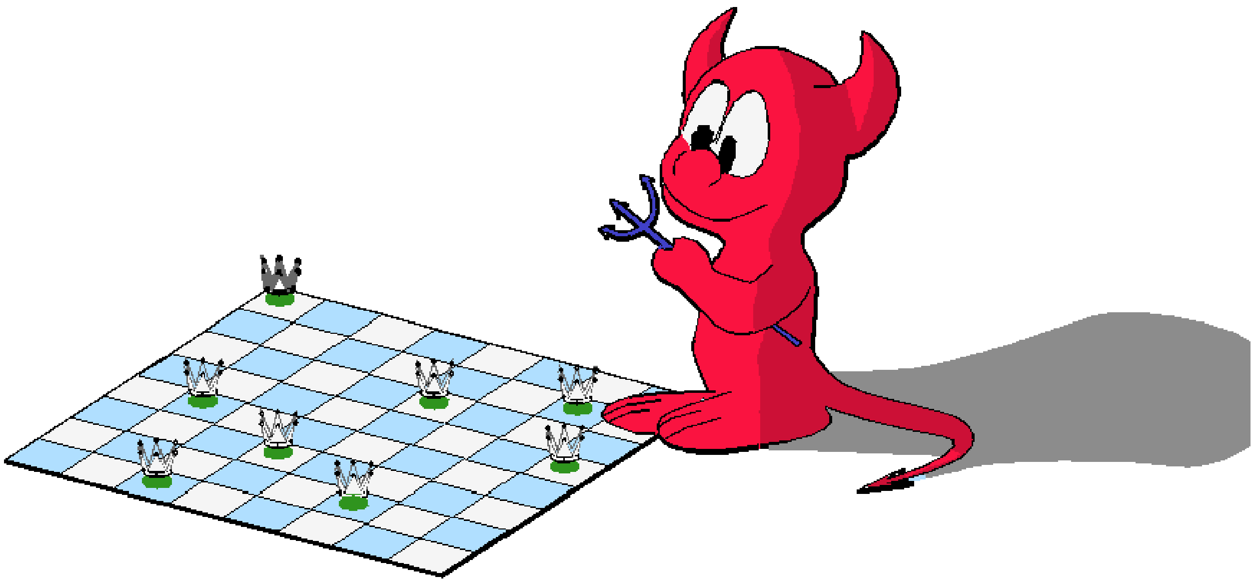
7) Access Control: Because Ach is implemented on top of POSIX shared memory files, access to channels is controlled via the unix permission bits: Per-user and per-group basis.

8) Seamless networking via achd (ACH Daemon). Currently slightly under 1khz (ethernet).

9) Portability: The Ach library is implemented in C using portable POSIX functions. Tested on i386, AMD64, and ARM.

Why Base Ach on POSIX?

- **POSIX: IEEE Standard 1003 & ISO 9945**
- **(Portable Operating System Interface - A family of standards specified by IEEE for maintaining compatibility for Operating Systems).**



Comparison to Xenomai:

- **Xenomai does not guarantee complete POSIX support (Linux does)**
- **Xenomai is a "dual" kernel. Both the Xenomai kernel and Linux kernel run together, which creates two separate "worlds."**
- **With PREEMPT_RT, everything runs as a normal Linux process. To make a real-time process, you make a system call for real-time priority (Standard POSIX)**
- **PREEMPT_RT is part of the mainline LINUX kernel. High likelihood of long-term support. (In contrast Xenomai is a third-party development)**
- **Numerous dev tools work on PREEMPT_RT and don't work on Xenomai. Valgrind is a big and useful one. There are numerous other memory debuggers, leak detectors and other software tools that will all work under PREEMPT_RT.**
- **Drivers are of particular concern! Non-linux OS = very limited support in drivers.**
- **Bottom Line: Our team has decided not to develop Xenomai-Specific Code. This is not portable and the future of Xenomai is uncertain.**

Hard vs. Soft Real Time – "Jitter"

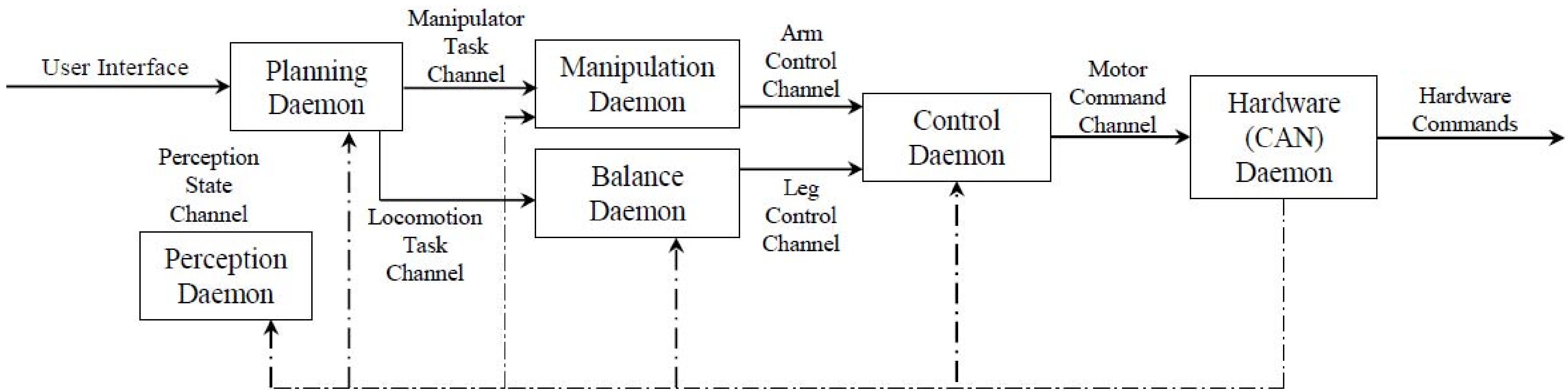
Dr. Jeremy H. Brown Report (2012)

https://www.osadl.org/fileadmin/dam/rtlws/12/Brown.pdf

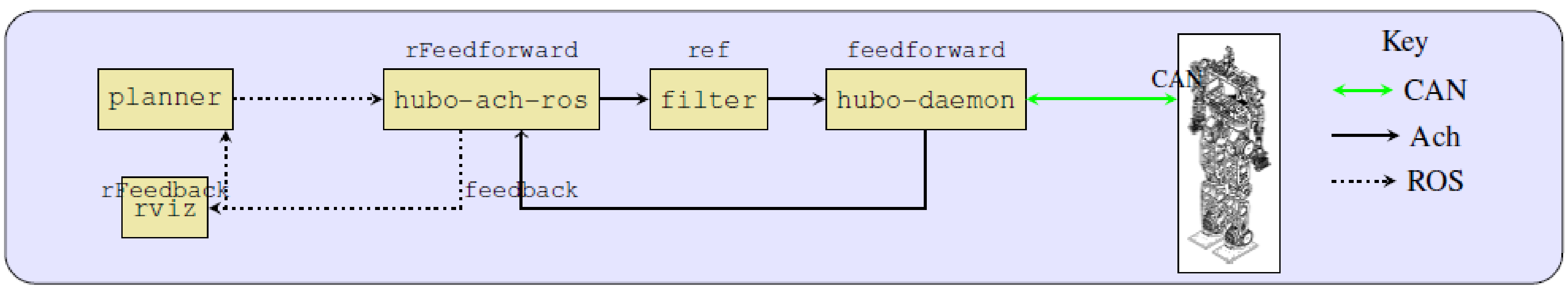
- *Xenomai outperforms PREEMPT_RT*
- *Maximum frequency for both Xenomai and PREEMPT_RT is far beyond 1kHz*
- *Under heavy load Xenomai Crashed!!! (Recall this is 2012)*
- After extensive testing and fix:

% of Tests	PREEMPT_RT	Xenomai
Worst Case – 100%	158us	57us
Close to Worst – 95%	47us	34us
Median	~0	~0

Sample Daemon & Communication Structure

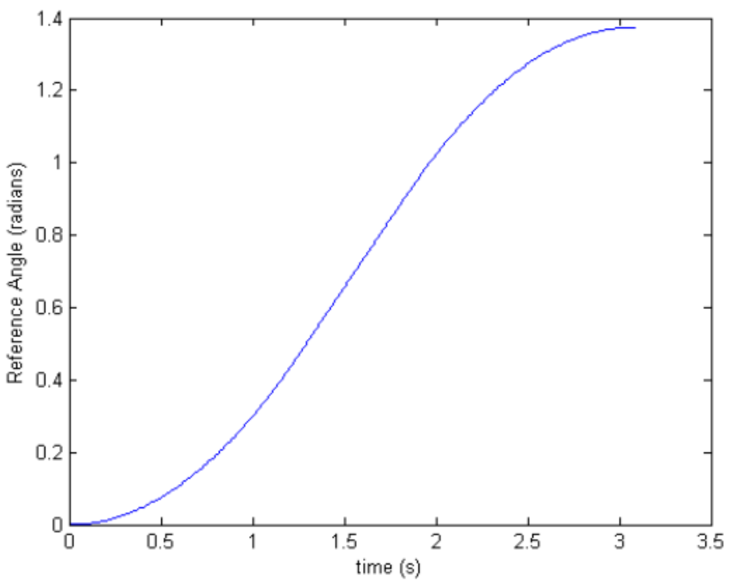


On Hubo – Here's the basic concept:



Hubo-Motion-RT: Control Daemon

- **Position**
  - **Specify a goal angle for each joint**
    (Options)
    - ❑ **Nominal Speed**
    - ❑ **Nominal Acceleration**
  - **Joint is moved from its current position to the goal position**
  - **Nominal Speed and Nominal Acceleration are <u>never</u> exceeded**
    - **The joint decelerates before arriving at the goal**
  - **Control is Asynchronous**
    - **Goal position can be changed at any time with no delay**
    - **Nominal Speed and Acceleration can be changed at any time**
  - **Carried through to completion**
    - **A command only needs to be sent once and it will be carried all the way through to completion**



Hubo-Motion-RT: Control Daemon

- **Velocity**
  - **Specify a desired velocity**
    (Optional)
    - **Nominal Acceleration**
  - **Joint is driven at the desired velocity without exceeding the Nominal Acceleration**
  - **Time-Out Feature**
    - **Velocity commands need to be streamed in**
    - **If a new velocity command is not sent within a pre-decided time, the joint's velocity will be driven down to zero at the rate of the nominal acceleration**
    - **This way joints will stop if their commanding programs crash**
- **Pass-Through**
  - **Motor board reference values are sent directly to the hardware daemon with no alterations**

Hubo-Motion-RT: Control Daemon

- **Safety Features**
  - **Minimum and Maximum Joint Values**
    - **A joint value which exceeds these limits will never be sent**
    - **These are independent of the hardware limits**
    - **They can be set at any point in run time**
    - **You can tailor them to your particular constraints**
  - **Error Limit**
    - **A joint whose reference/state error exceeds a limit will be frozen**
    - **This can be used to prevent burnout or detect hardware issues**
  - **Seamless transitions between position and velocity control modes**
  - **Enables multiple processes to control different limbs seamlessly**

Hubo-Motion-RT: Programming Interface

- C++ Library
  - Considering Python support in the future (still to be determined)
  - Sample code:

```
#include "Hubo_Tech.h"
int main( int argc, char **argv )
{
    Hubo_Tech hubo;
    hubo.setJointAngle( REB, -M_PI/2, true );
}
```

  - Moves the right elbow 90-degrees at the default Nominal Speed and Acceleration values

Hubo-Motion-RT: Programming Interface

- **Uses Eigen C++ Library**
  - **Sample code:**

```
#include "Hubo_Tech.h"
int main( int argc, char **argv )
{
    Hubo_Tech hubo;
    Vector6d q;
    q << -20.0/180.0*M_PI, 0, 0, -M_PI/2+20.0/180.0*M_PI, -0.3, 0;
    hubo.setRightArmAngles( q, true );
}
```

  - **Moves the right arm positions to the angles specified by the "q" vector**
  - **Uses the default Nominal Speeds and Accelerations of each joint**

Hubo-Motion-RT: Programming Interface

- **Delivers all state data as well**
  - **Sample code:**

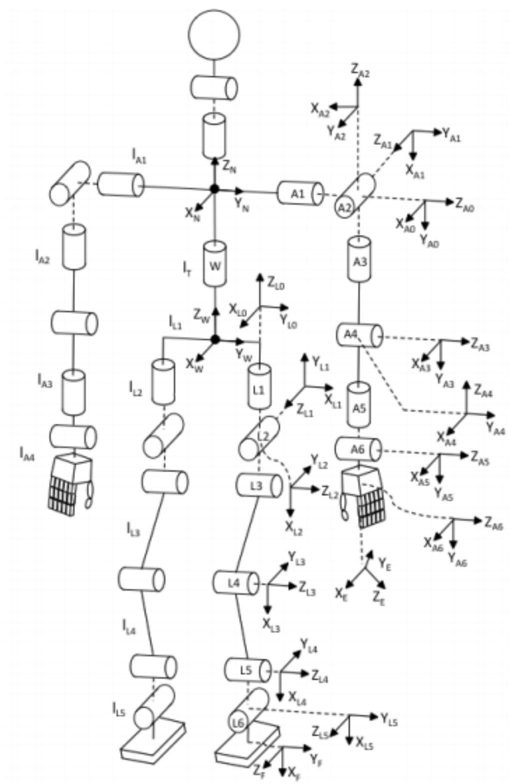
```
hubo.update();

hubo.getRightArmAngles( q );
mx = hubo.getRightHandMx();
```

  - **Grabs the latest state data**
  - **Fills the vector "q" with the current positions of the right arm joints**
  - **Fills "mx" with the value of the right hand FT-Sensor's reading of moment about the x-axis**

Hubo-Motion-RT: Kinematics

- **Forward and Inverse Kinematics**
  - **Sample code:**
    - **T is an Eigen::Transform describing the end effector pose**
    - **q is an Eigen::Vector6**
    - **qprev is the previous value of q**

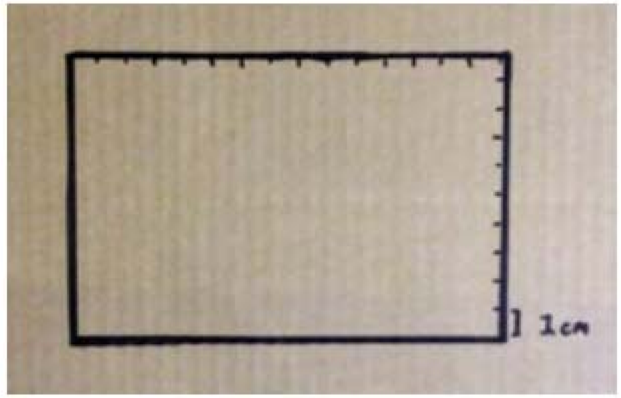
```
hubo.huboArmFK( T, q, LEFT );

hubo.huboArmIK( q, T, qprev, RIGHT );
```

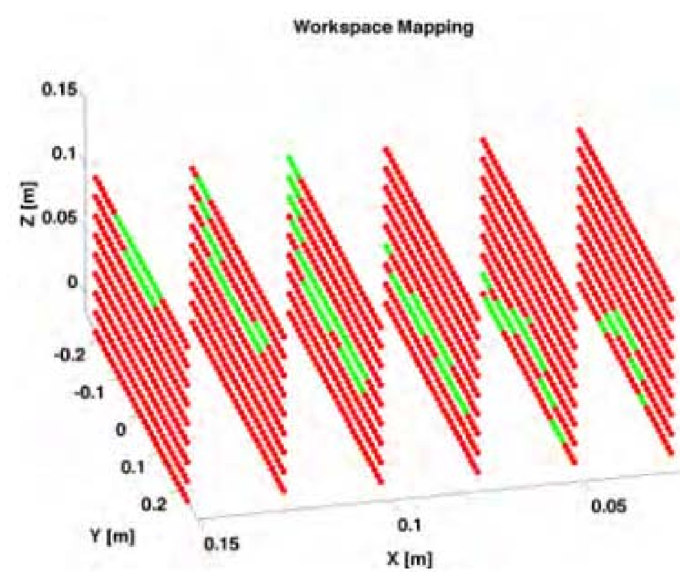


  - **Fills in the end effector pose (T) for the left arm given the joint angles described by q**
  - **Fills in the required joint angles (q) to produce the desired end effector pose (T) for the right arm**

Hubo-Motion-RT: Kinematics

- **Results:**





[redacted]