Korea Summer Institute

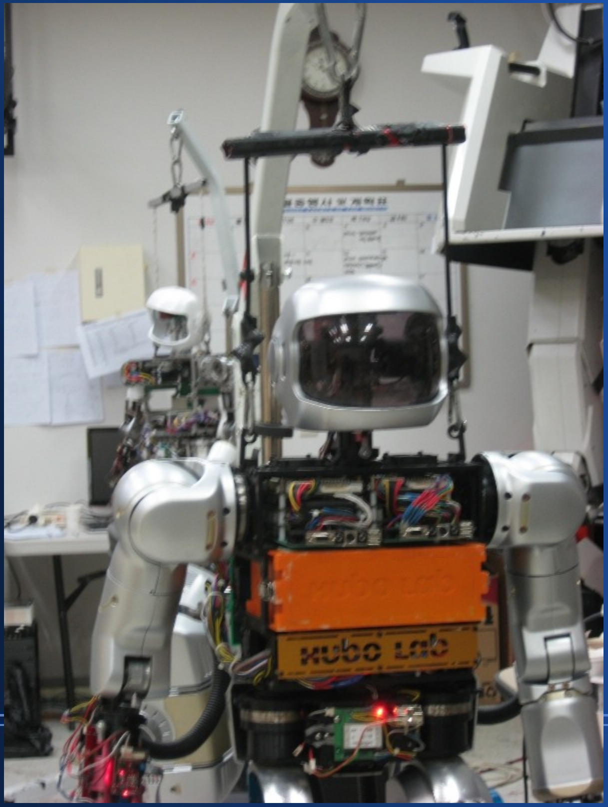
Humanoid Control Software

Bob Sherbert

2010 – 8 – 19

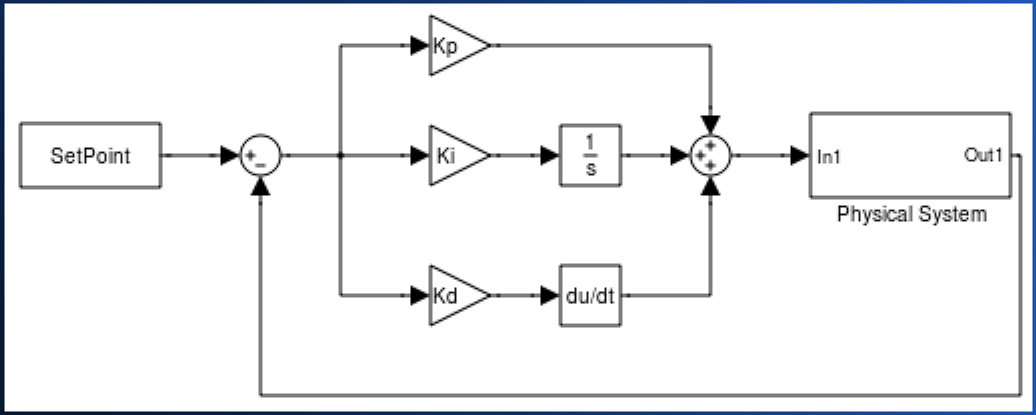
Control Theory

- What do the Segway, cruise missiles, and this guy have in common?



Control Theory

$$u(t) = \mathrm{MV}(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt} e(t)$$



```
#include <vcl.h>
#pragma hdrstop

#include "PIDClass.h"
#include <Math.h>


#pragma package(smart_init)
__fastcall PIDClass::PIDClass()
{
 w = 1;
 r = 1;
 e_old = 0;
 Ue = 0;
 Xy = 1;
 Xe = 1;
 Xp = 1;
 Xi = 0;
 Xd = 0;
}

__fastcall PIDClass::~PIDClass()
{


}

void PIDClass::reset(void)
{
 e = 0;
 e_old = 0;
 KP = 0;
 KI = 0;
 KD = 0;
 y = 0;
}

void PIDClass::calc(void)
{
 e = w - r;

 if (fabs(e * Xe) <= Ue) {
  e_old = 0;
  KP = 0;
  KI = 0;
  KD = 0;
  y = 0;
 } else {

  KP = e * Xe;

  KI += e * Xe;

  KD = (e * Xe) - e_old;

  e_old = e * Xe;

  y = Xy * (KP * Xp + KI * Xi + KD * Xd);
 }

 if (fabs(y) > My) {
  if (y < 0) {
   y = -My;
  } else {
   y = My;
  }
 }



}
```



```
double PIDClass::round(double x) {
 return(floor(x + 0.5));
}
```

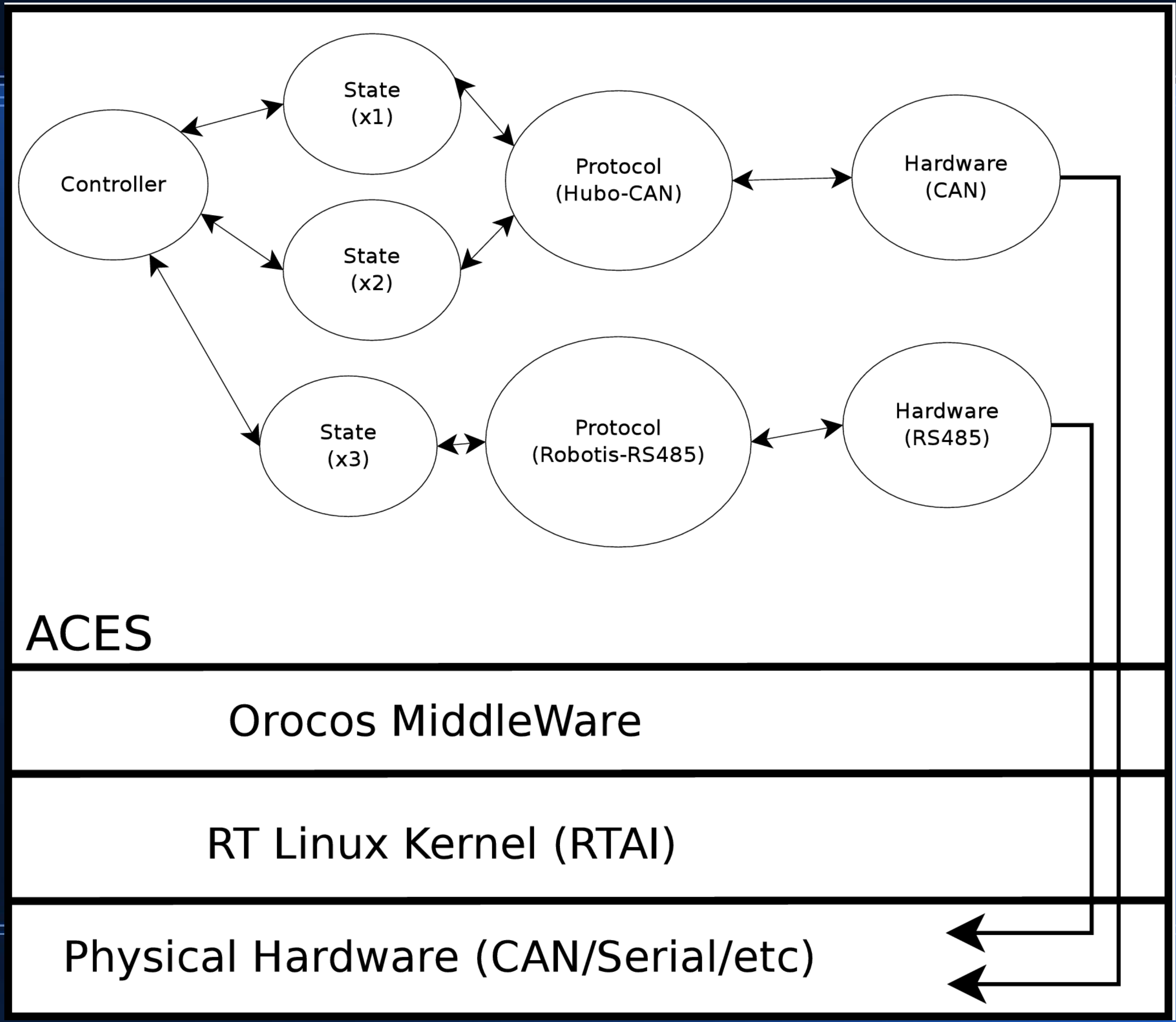
Goals

- Represent physical hardware in something akin to mathematics
- Ideally – Write controllers directly in state space & get automatic hardware response
- Allow easy integration of new hardware

Motivation

- PIRE Grant – Humanoid architecture development
- Multiple platforms – HUBO, Miniature Humanoids, Simulators
- Desire For:
  - Experiments scaling control strategies
  - Ease of exchanging controllers
  - Fault Detection
  - Extensive logging facilities
  - Prototyping in simulation

Overview

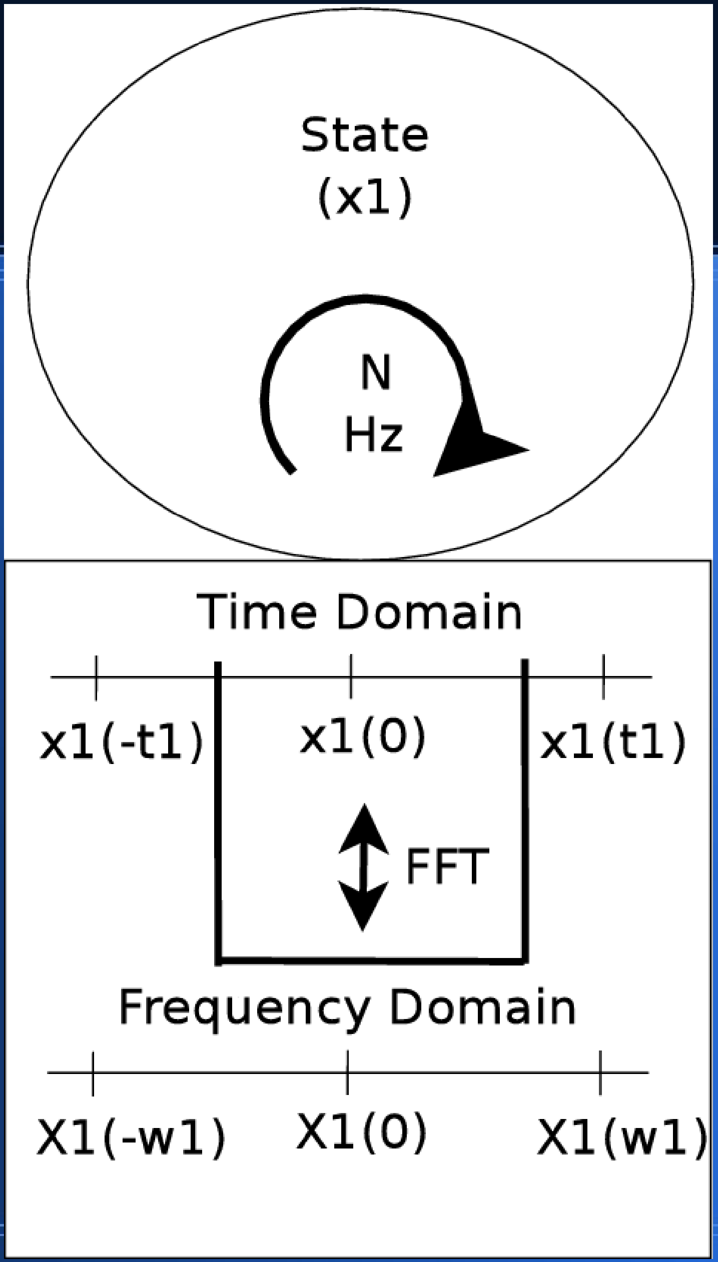

State
(x1)
Controller
Protocol
(Hubo-CAN)
Hardware
(CAN)
State
(x2)
State
(x3)
Protocol
(Robotis-RS485)
Hardware
(RS485)
ACES
Orocos MiddleWare
RT Linux Kernel (RTAI)
Physical Hardware (CAN/Serial/etc)

Overview

- Real Time (RT) services provided by the OS through Orocos
- Each instance of a component is associated w/an RT thread
- Communication between modules is event driven
  - Each component ‘subscribes’ to the events of other components
  - Information in transmissions is available to all, and each makes own decision about reception

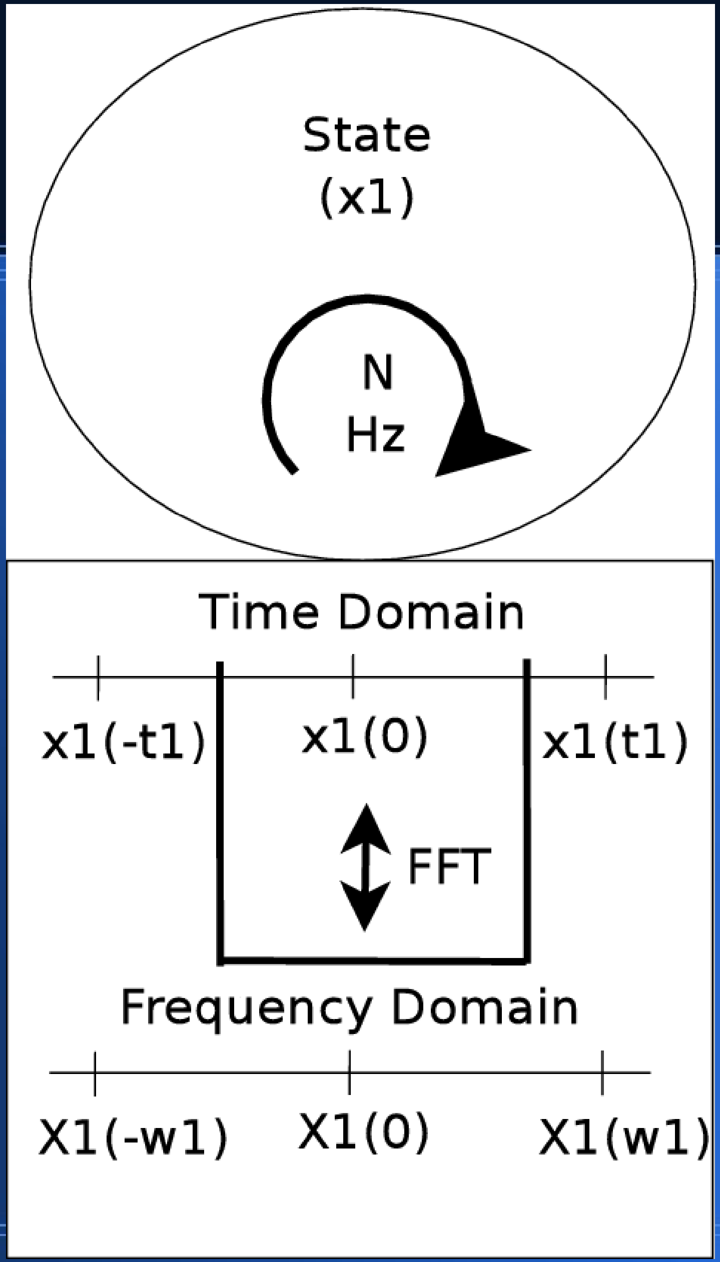
State

- Corresponds with state variable from control theory
- Presents last known value to system internals
- Independent of communication hardware (RS232, CAN, etc)



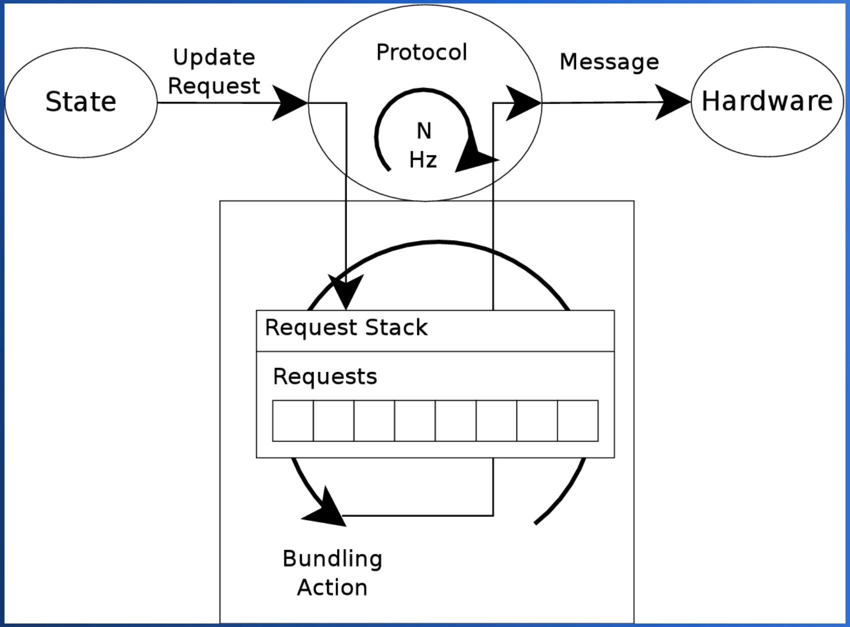
State

- Stores historical data
- Can provide averaged (smoothed) data based on historical data
- Can estimate future data based on user selected algorithm
- Compute Fourier/Laplace transforms for frequency domain controllers



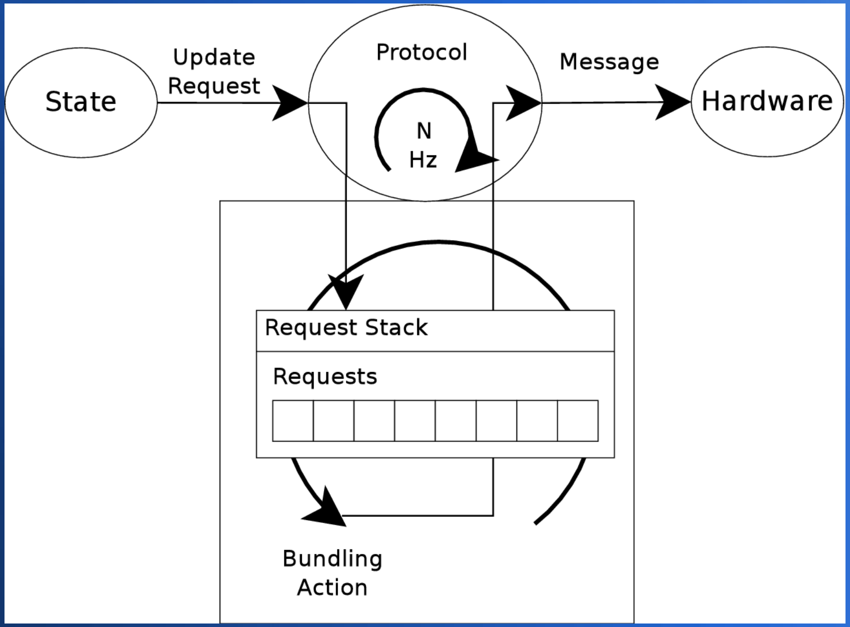
Protocol

- Describes a data format/protocol for communicating with a bus device
  - Arrangement of bytes in packet
  - Data contained within packet
- States communicate w/their corresponding devices through the protocol
  - State emits request event when update required



Protocol

- Collects, buffers, and bundles state requests
- Converts bundles into Messages that the Hardware can transmit to physical devices
- Emits Message as event – Hardware picks up event

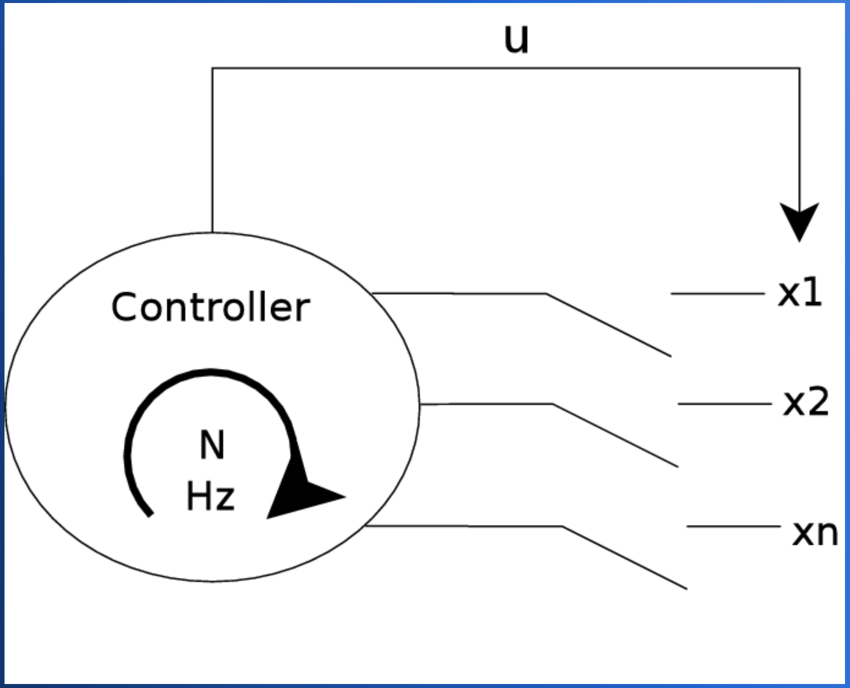


Hardware

- Gatekeeper to physical hardware
  - Prevents multiple access to line in the thread-based environment
- Converts messages from protocol into packets on the physical bus
- Tracks expected responses
- Handles Tx/Rx error reporting to higher levels

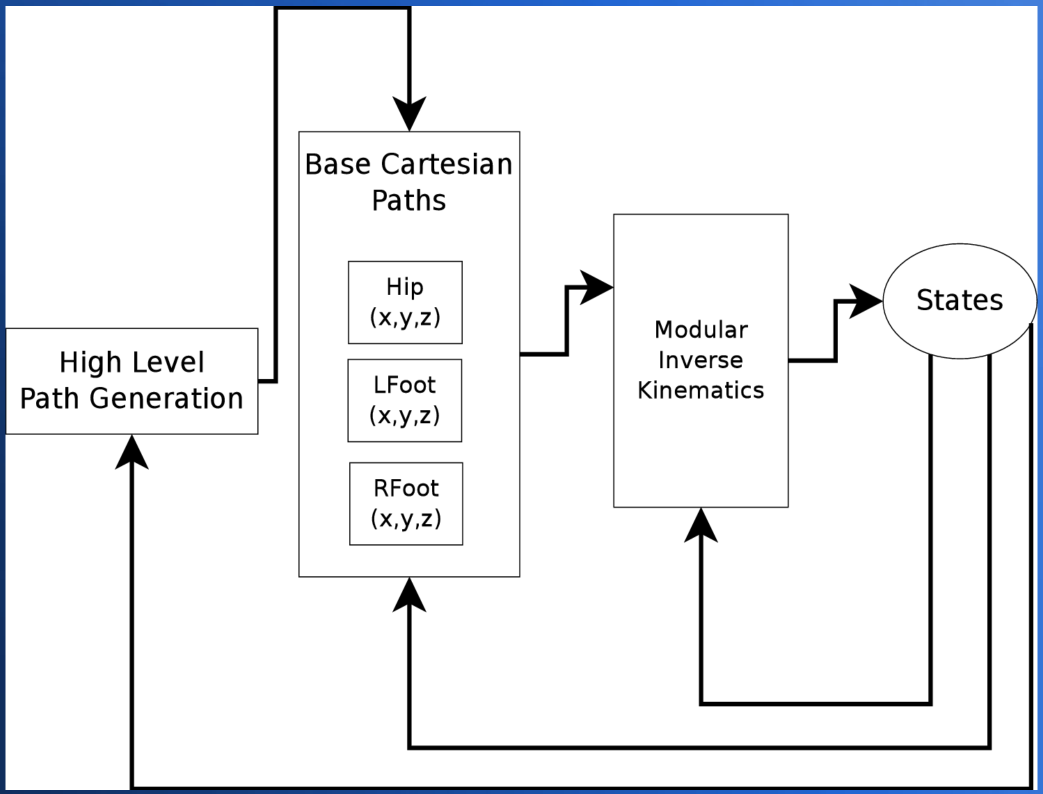
Controller

- Controller implements the feedback laws
- No restrictions placed on controller internals
- May draw from states, external data sources, algorithms, etc
- The determined control signal is broadcast to the entire system
- Subscribers take appropriate action to realize the control signal



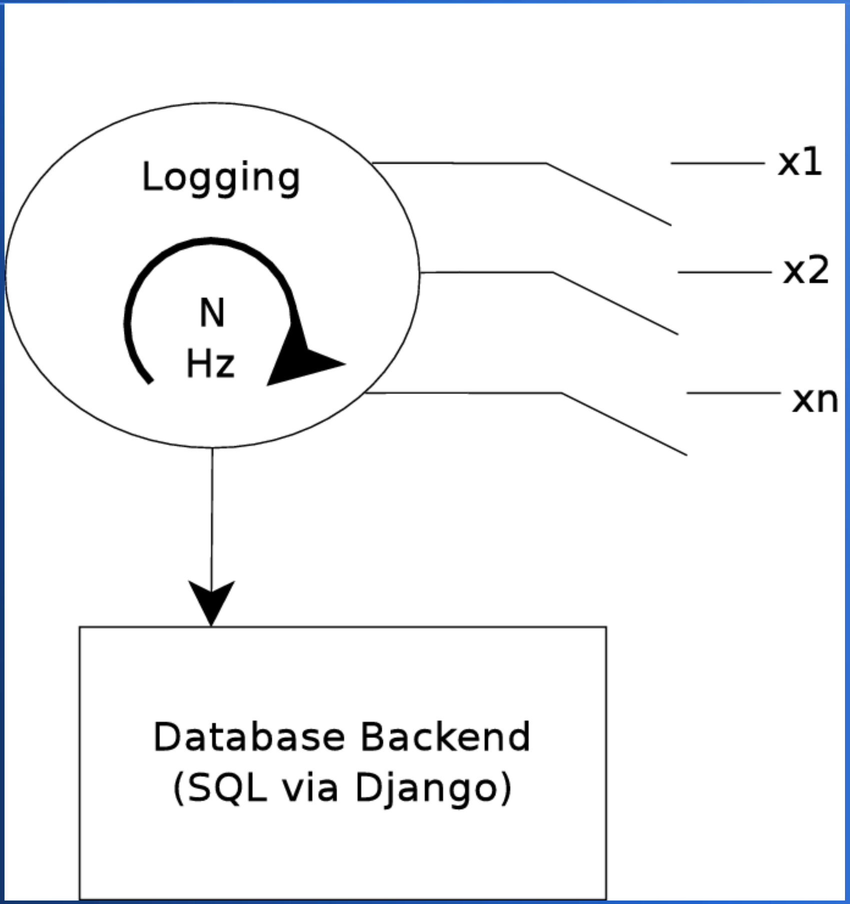
Template Controller (ex.)

- Modular design for humanoid control
- Cartesian path, IK, and joint space are each modules
  - Different module used for Hubo/Mini/Virtual
- Feedback can be provided at any level
  - Landing control vs Ankle roll control



Logging

- Internal sampling of states
- Data stored to an SQL database
- Data easily accessible through web
- Easily exported to analysis packages (Matlab)

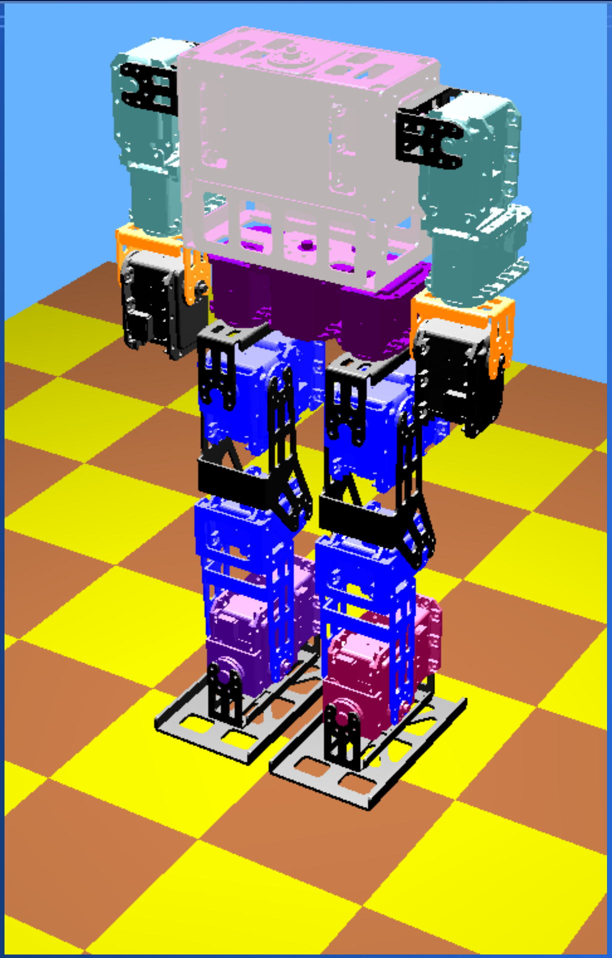


Playback

- Specialize controller can be loaded which reads logged data back from database
- The recorded data can be played back within the simulator
- Simulator provides contact highlighting and single step advance
- Allows for failure analysis

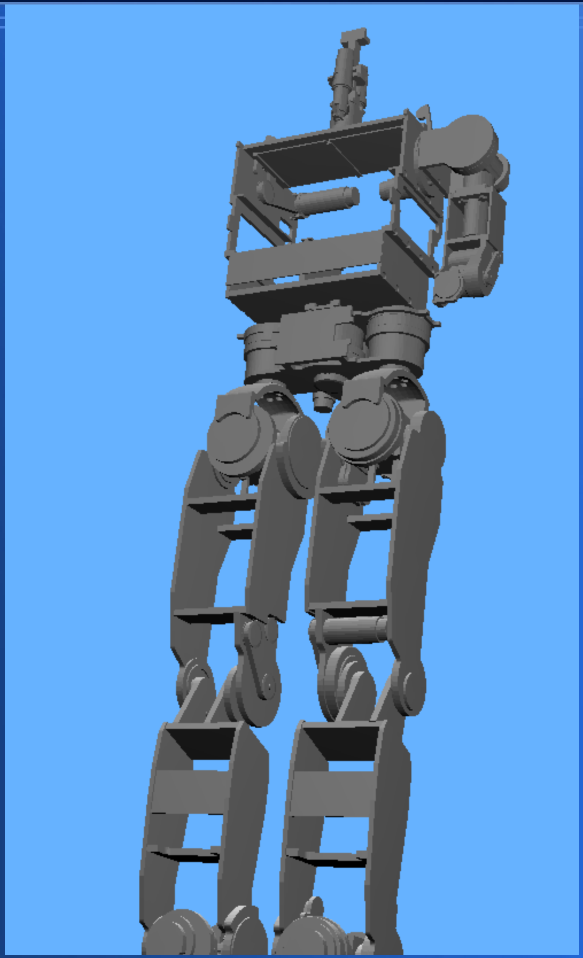
Current Progress

- Miniature Humanoid physical properties fully implemented in simulation
- Prototype software connected to playback simple statically stable walking scripts
- Requires addition of sensors



Current Progress

- HUBO model ~70% complete
- Physically geometry mostly imported
- Appropriate inertial data collected
- Sensors need to be implemented



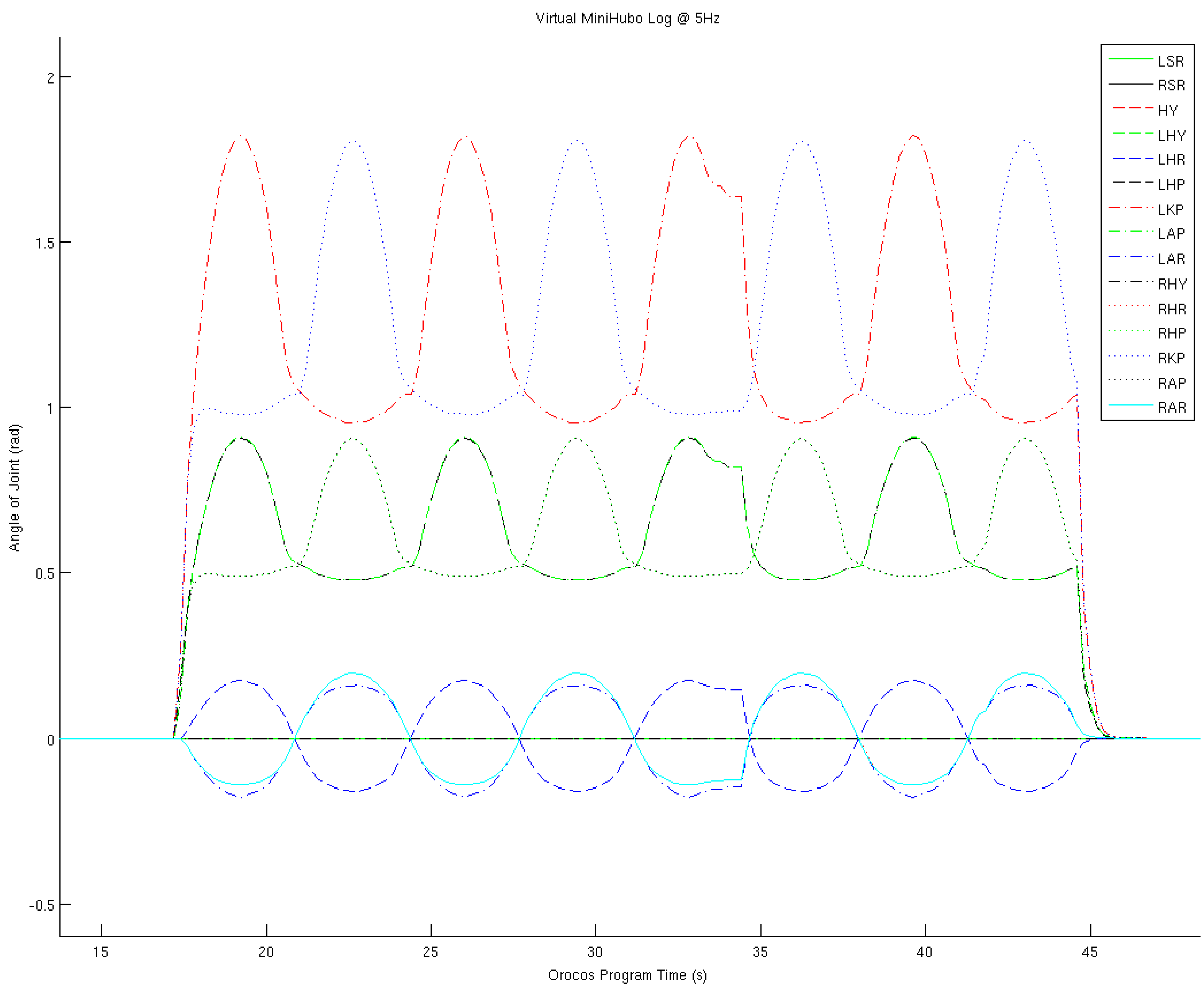
Current Progress

```
program main {
    do addHardware("wbHW 1 600", "Webots", "");
    do addProtocol("wbPcol 2 600", "Webots", "");

    do addController("wbNull 10 15", "Webots Null", "");
    do addController("wbctrl 10 15", "Webots Mini",
                     "IKscript1.txt");
    do addController("wbArm 10 15", "Webots Arm", "sins.txt");

    do addDevice("dHY 5 31", "Webots",  "HY  0.0 -1.0");
    do addDevice("dLSP 5 31", "Webots", "LSP 0.0  1.0");
    do addDevice("dLSR 5 31", "Webots", "LSR 0.0  1.0");
    do addDevice("dRSP 5 31", "Webots", "RSP 0.0  1.0");
    do addDevice("dRSR 5 31", "Webots", "RSR 0.0  1.0");
    do addDevice("dLHY 5 31", "Webots", "LHY 0.0  1.0");
    do addDevice("dLHR 5 31", "Webots", "LHR 0.0  1.0");
    do addDevice("dLHP 5 31", "Webots", "LHP 0.0 -1.0");
    do addDevice("dLKP 5 31", "Webots", "LKP 0.0  1.0");
    do addDevice("dLAP 5 31", "Webots", "LAP 0.0 -1.0");
    do addDevice("dLAR 5 31", "Webots", "LAR 0.0  1.0");
    do addDevice("dRHY 5 31", "Webots", "RHY 0.0  1.0");
    do addDevice("dRHR 5 31", "Webots", "RHR 0.0 -1.0");
    do addDevice("dRHP 5 31", "Webots", "RHP 0.0 -1.0");
    do addDevice("dRKP 5 31", "Webots", "RKP 0.0  1.0");
    do addDevice("dRAP 5 31", "Webots", "RAP 0.0 -1.0");
    do addDevice("dRAR 5 31", "Webots", "RAR 0.0  1.0");
launch.ops                                    2,1           Top
```

```
    do addState("HY 5 31",  "Webots", "Joint");
    do addState("LSP 5 31", "Webots", "Joint");
    do addState("LSR 5 31", "Webots", "Joint");
    do addState("RSP 5 31", "Webots", "Joint");
    do addState("RSR 5 31", "Webots", "Joint");
    do addState("LHY 5 31", "Webots", "Joint");
    do addState("LHR 5 31", "Webots", "Joint");
    do addState("LHP 5 31", "Webots", "Joint");
    do addState("LKP 5 31", "Webots", "Joint");
    do addState("LAP 5 31", "Webots", "Joint");
    do addState("LAR 5 31", "Webots", "Joint");
    do addState("RHY 5 31", "Webots", "Joint");
    do addState("RHR 5 31", "Webots", "Joint");
    do addState("RHP 5 31", "Webots", "Joint");
    do addState("RKP 5 31", "Webots", "Joint");
    do addState("RAP 5 31", "Webots", "Joint");
    do addState("RAR 5 31", "Webots", "Joint");

    do linkDS("dHY",  "HY"); do linkPD("wbPcol", "dHY");
    do linkPD("wbPcol", "dLSP"); do linkDS("dLSP", "LSP");
    do linkPD("wbPcol", "dLSR"); do linkDS("dLSR", "LSR");
    do linkPD("wbPcol", "dRSP"); do linkDS("dRSP", "RSP");
    do linkPD("wbPcol", "dRSR"); do linkDS("dRSR", "RSR");
    do linkPD("wbPcol", "dLHY"); do linkDS("dLHY", "LHY");
    do linkPD("wbPcol", "dLHR"); do linkDS("dLHR", "LHR");
launch.ops                                    52,1          28%
"launch.ops" 116L, 4505C written
```

Current Progress

```
    marshalling ( Read and write Properties to a file. )
           main ( Orocos Program Script )
 Programs     : main[R]

 Peers        : HY[R] LAP[R] LAR[R] LHP[R] LHR[R] LHY[R] LKP[R] LSP[R] LSR[R] RAP[R] RAR[R] RHP[R] RHR[R] RHY[R] RKP[R] RSP[R
] RSR[R] dHY[R] dLAP[R] dLAR[R] dLHP[R] dLHR[R] dLHY[R] dLKP[R] dLSP[R] dLSR[R] dRAP[R] dRAR[R] dRHP[R] dRHR[R] dRHY[R] dRKP[
R] dRSP[R] dRSR[R] flog[R] launch[R] wbArm[R] wbHW[R] wbNull[R] wbPcol[R] wbctrl[R]

 In Task dispatch[R]. (Status of last Command : none )
 (type 'ls' for context info) :HY.value
      Got :HY.value
 = 5.16191e-08

 In Task dispatch[R]. (Status of last Command : none )
 (type 'ls' for context info) :HY.go(1.15)
      Got :HY.go(1.15)
 = (void)

 In Task dispatch[R]. (Status of last Command : none )
 (type 'ls' for context info) :HY.value
      Got :HY.value
 = 1.15

 In Task dispatch[R]. (Status of last Command : none )
 (type 'ls' for context info) :RSP.go(HY.value + .2)
      Got :RSP.go(HY.value + .2)
 = (void)

 In Task dispatch[R]. (Status of last Command : none )
 (type 'ls' for context info) :
```


Virtual MiniHubo Log @ 5Hz
LSR
RSR
HY
LHY
LHR
LHP
LKP
LAP
LAR
RHY
RHR
RHP
RKP
RAP
RAR
Angle of Joint (rad)
Orocos Program Time (s)
2
1.5
1
0.5
0
-0.5
15
20
25
30
35
40
45