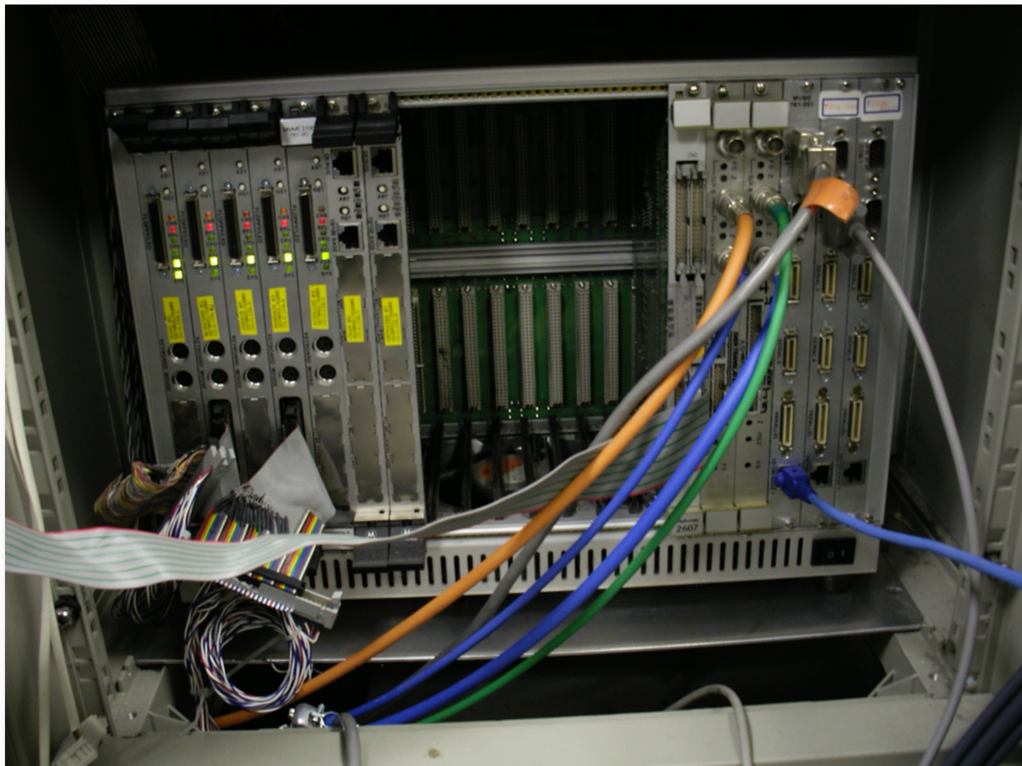
CS545—Lecture SL

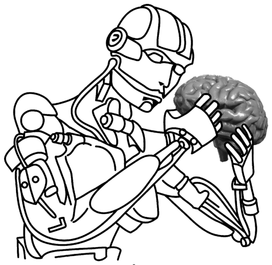
- History: From vxWorks to SL
- Simulation components
  - multi processing, multi-threads
  - configuration files to setup a robot (similar to URDF)
  - Featherstone Rigid-Body Dynamics
  - Contact dynamics from penalty methods with constraint contact points
- Real-Time components
  - RTOS Xenomai interface
    - RTNET, RT-USB, RT-CAN, Analogy
  - ROS interface
- Examples applications
- Pros, Cons, Future
- Data Visualization
  - CLMCPLOT in Matlab
- A Programming Example

From vxWorks to SL



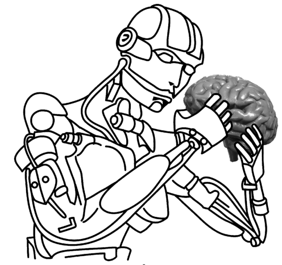
- Originally created as control software for multi-processor real-time control using vxWorks (~1994 at MIT, with Chris Atkeson)

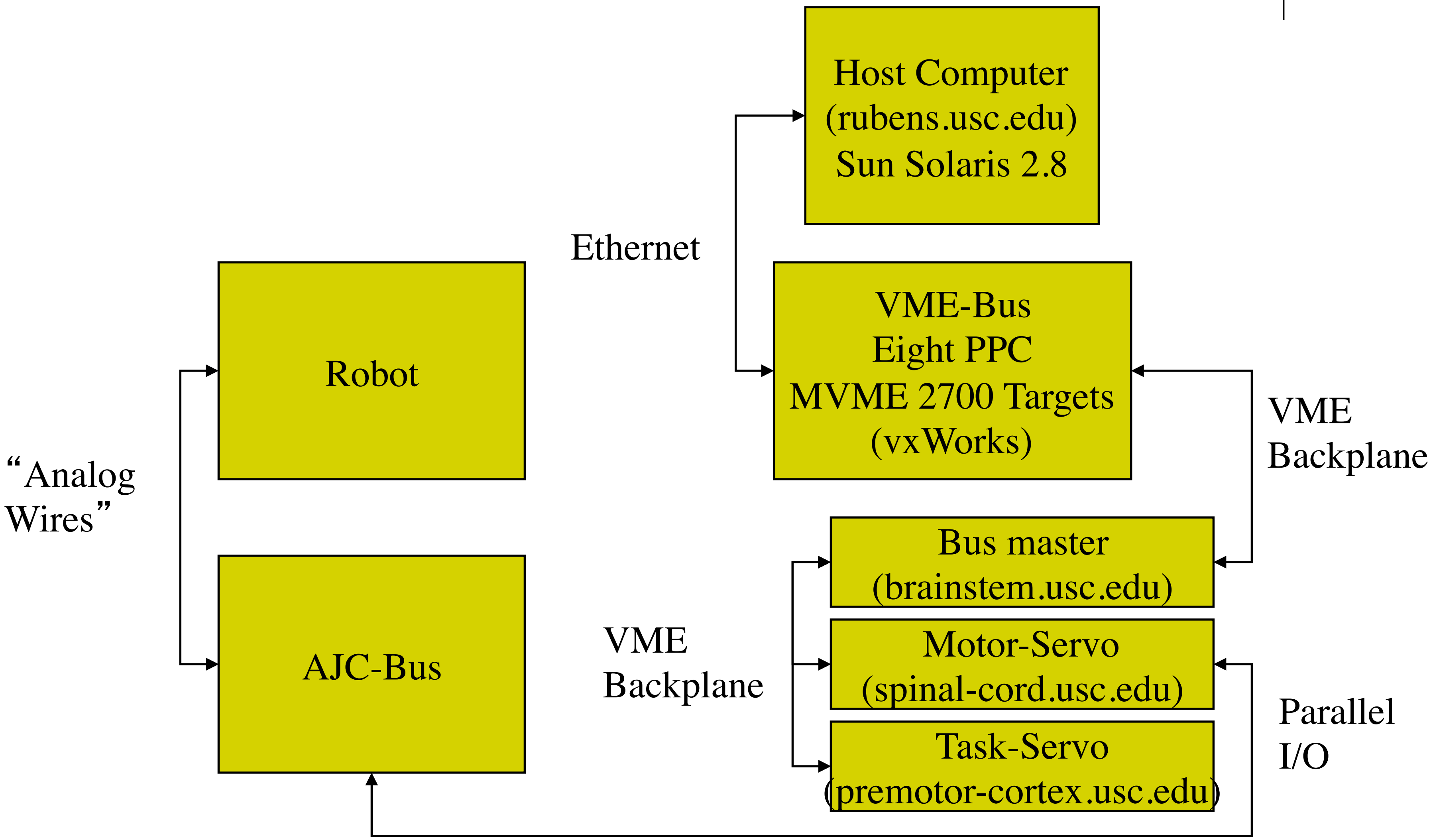
VxWorks: A Professional RTOS for Control



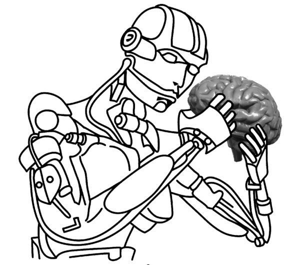
- What does VxWorks do?
  - Offers a development environment on a host computer
  - Offers a UNIX-like real-time operating system on the targets
  - Integrates target and host development smoothly
  - Allows multiple targets
  - Allows target communication and memory sharing
  - Integrates the system smoothly into a TCP/IP computer network
  - Guarantees real-time performance (preemptive priority scheduling, intertask synchronization, interrupt handler, memory management)
  - Allows NFS mounting and normal use of UNIX file systems

A Typical Robot Environment with vxWorks and a VME bus



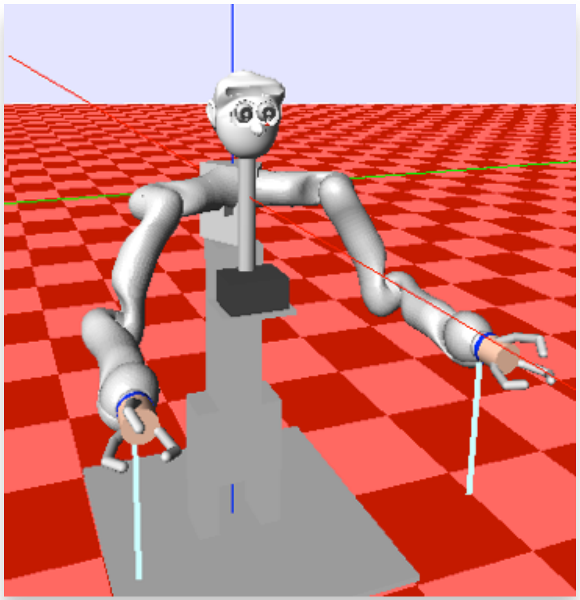


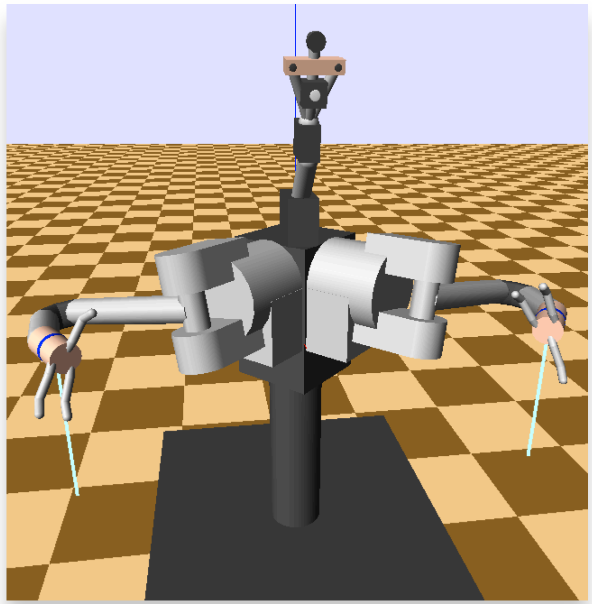
What is SL?

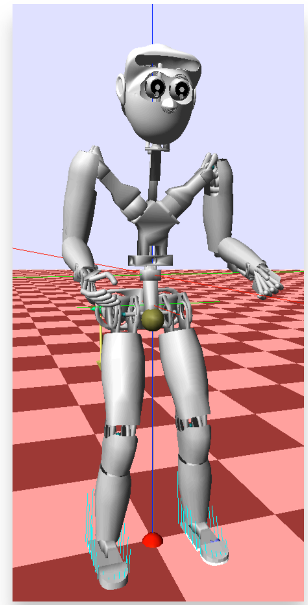


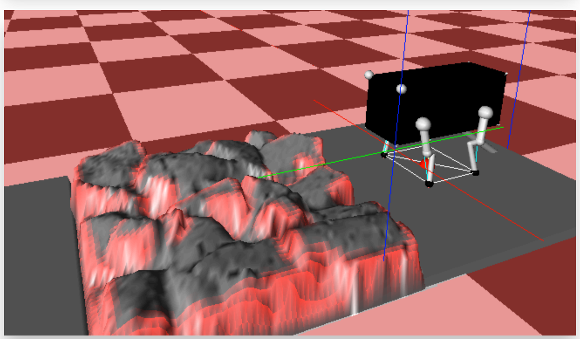
- SL := Simulation Lab
- Goal: Identical software running physical simulations and actual robots
- Design Criteria
  - Fast (super real-time in simulation) and Real-time (for actual robot)
  - Physics simulations and many kinematics and rigid body dynamics functions
  - Multi-processing, multi-threading
  - Visualization tools
  - Easy to reconfigure for different robots
  - Keep the end-user away from complex programming
  - Runs on Unix systems and RTOS Unix systems
  - Minimal dependence on external software packages
  - Interfaces to anything you want (e.g., ROS)

Examples of SL Control Systems



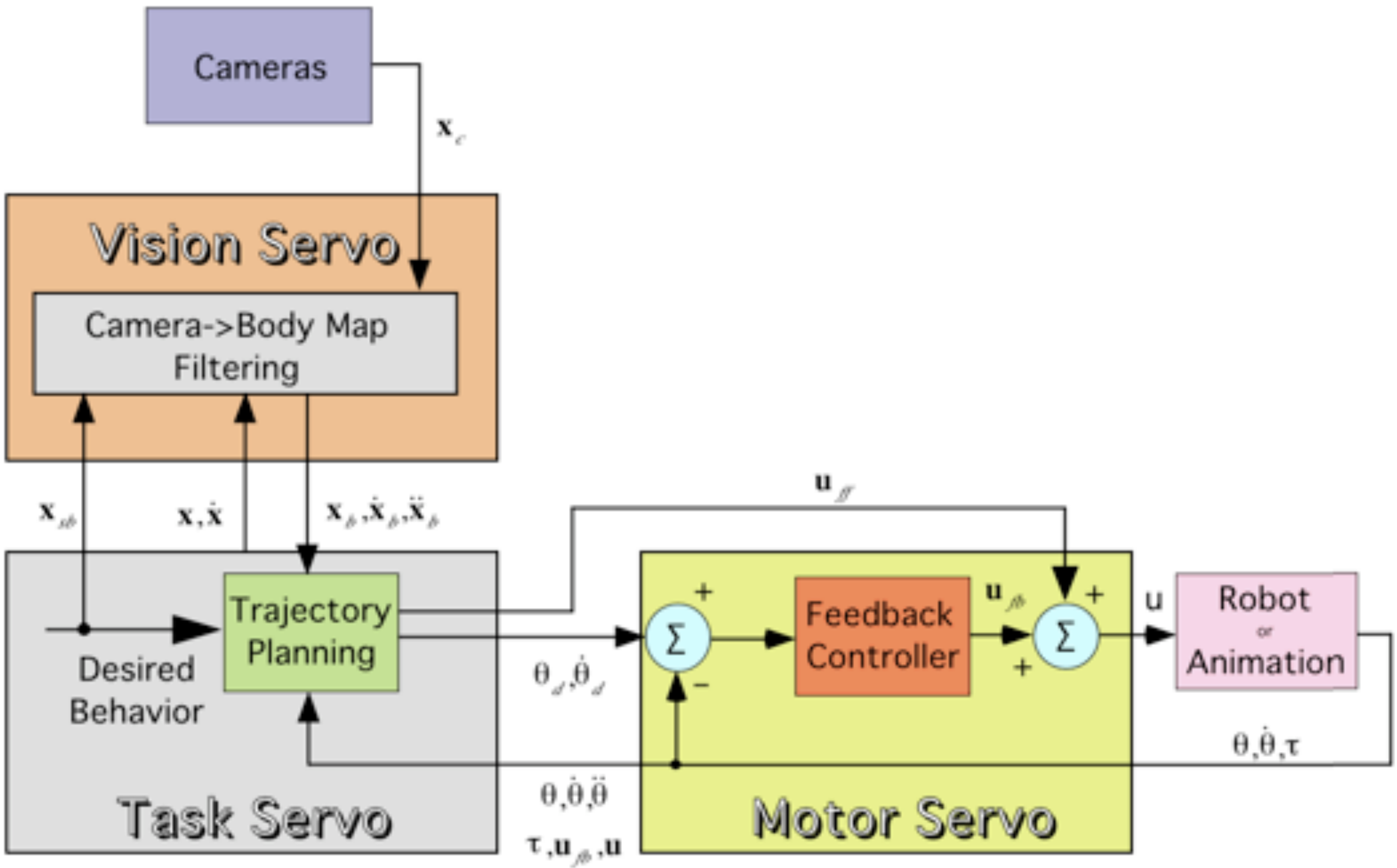




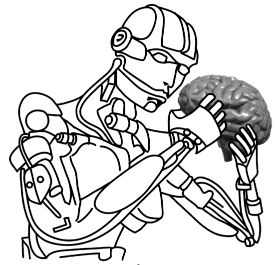


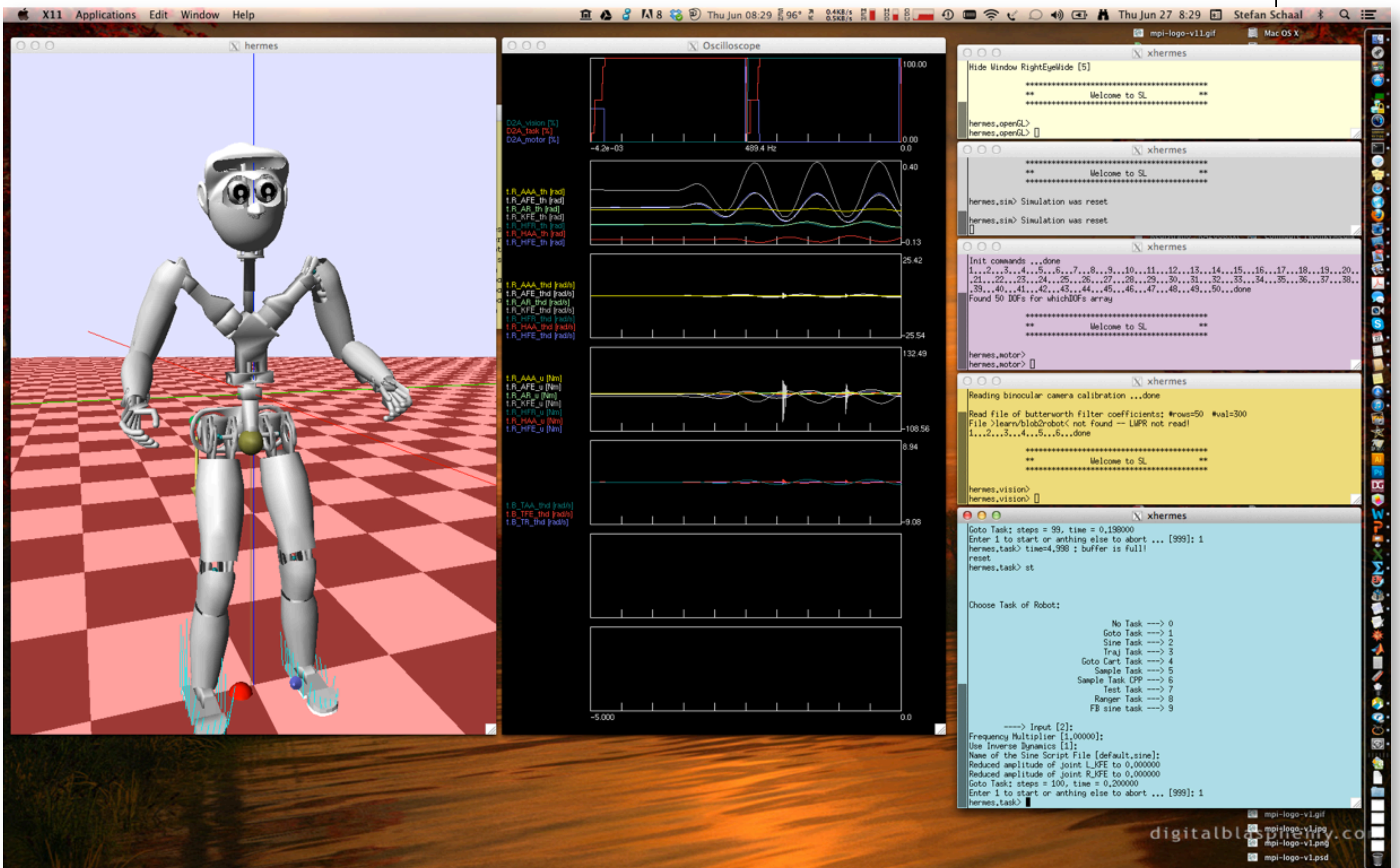
- **Some Key Points of SL**:
- Originally developed as multi-processor real-time control software using vxWorks (~1994 at MIT)
- Extended starting 1996 be add a physical simulator with the goal to have exactly the same simulation and real-time control interface
- Since 2008, real-time version uses open-source Xenomai (hard real-time OS) on Ubuntu Platforms instead of vxWorks
- Used by various partner labs, including CMU, ATR, IIT, ETH, TU Darmstadt, Max-Planck Tübingen, U. Birmingham, and others.

Control Loop Over Multiple Processes in SL

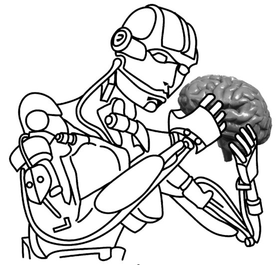


Simulation Components of SL

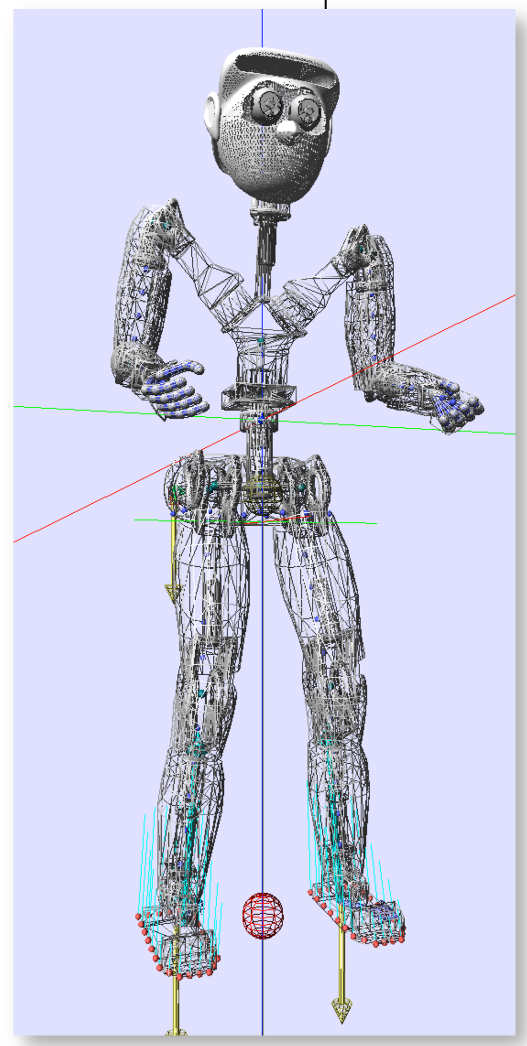




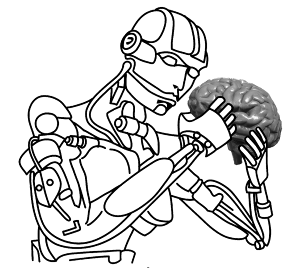
Simulation Components of SL



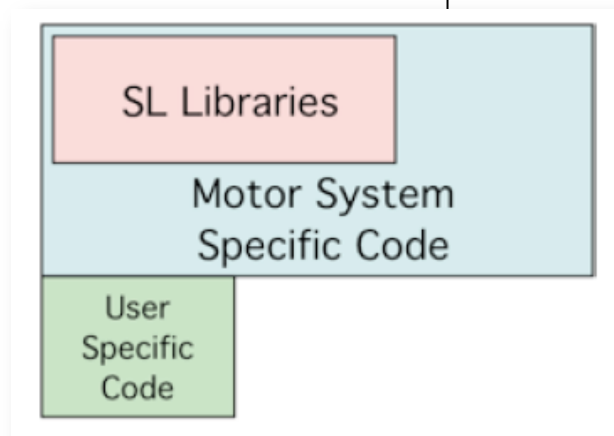
- Multi-processing, multi-threading, shared memory
  - in essence, we mimicked a multi-processor vxWorks systems, which now maps well onto multi-core architectures
  - runs frequently significantly faster than real-time
- Featherstone Algorithms
  - All key Featherstone algorithms implemented (Newton-Euler ID, Composite Inertia ID, Articulated-Body FD, Composite Inertia FD, fixed-base and floating-base)
  - Input: configuration files that describe forward kinematics tree
  - Mathematica programs convert configuration files to C-files
  - We have full access to dynamics/kinematics and change anything
- Contact Dynamics
  - Penalty methods based on contact points
  - Contact points have constraints to allow realistic friction, sliding
  - Various contact models are possible
  - Simple objects in the environment

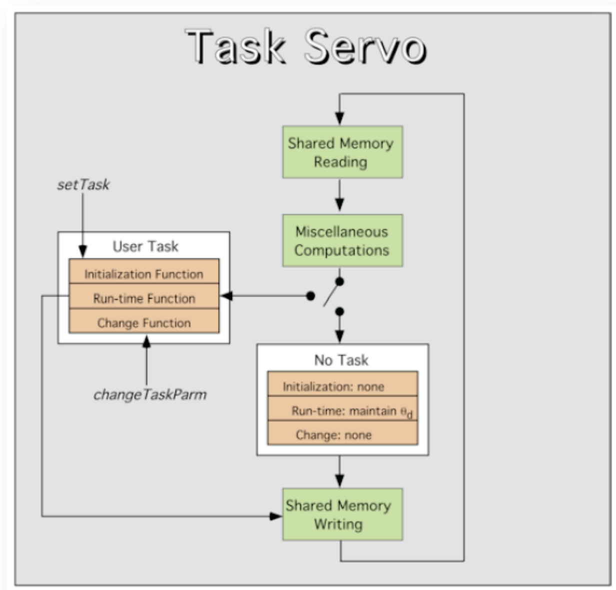


Simulation Components of SL

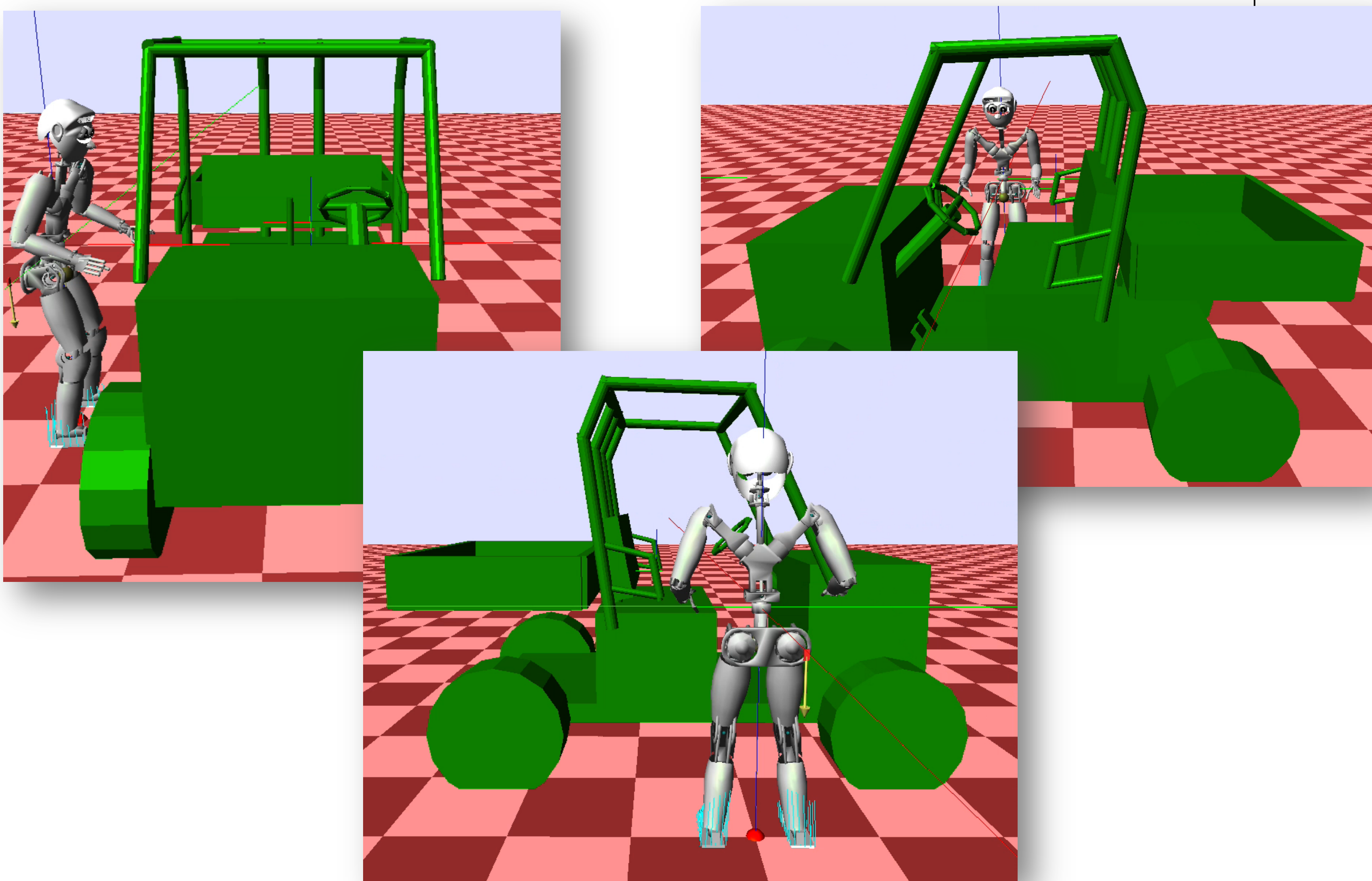


- Programming
  - mostly programmed in C/C++
  - ROS interface (Peter Pastor & Mrinal Kalakrishnan)
  - users can overwrite most code with local function
  - rather lean, simple C-libraries
  - hardly any dependencies on non-standard external libraries (has been compiling for 15 years without problems on Macs, Linux, Dec-Alphas, Solaris, etc.)
  - supports all Unix flavors, but not Windows
- Documentation
  - oh well ...
- http://www-clmc.usc.edu/Resources/Details?id=10259

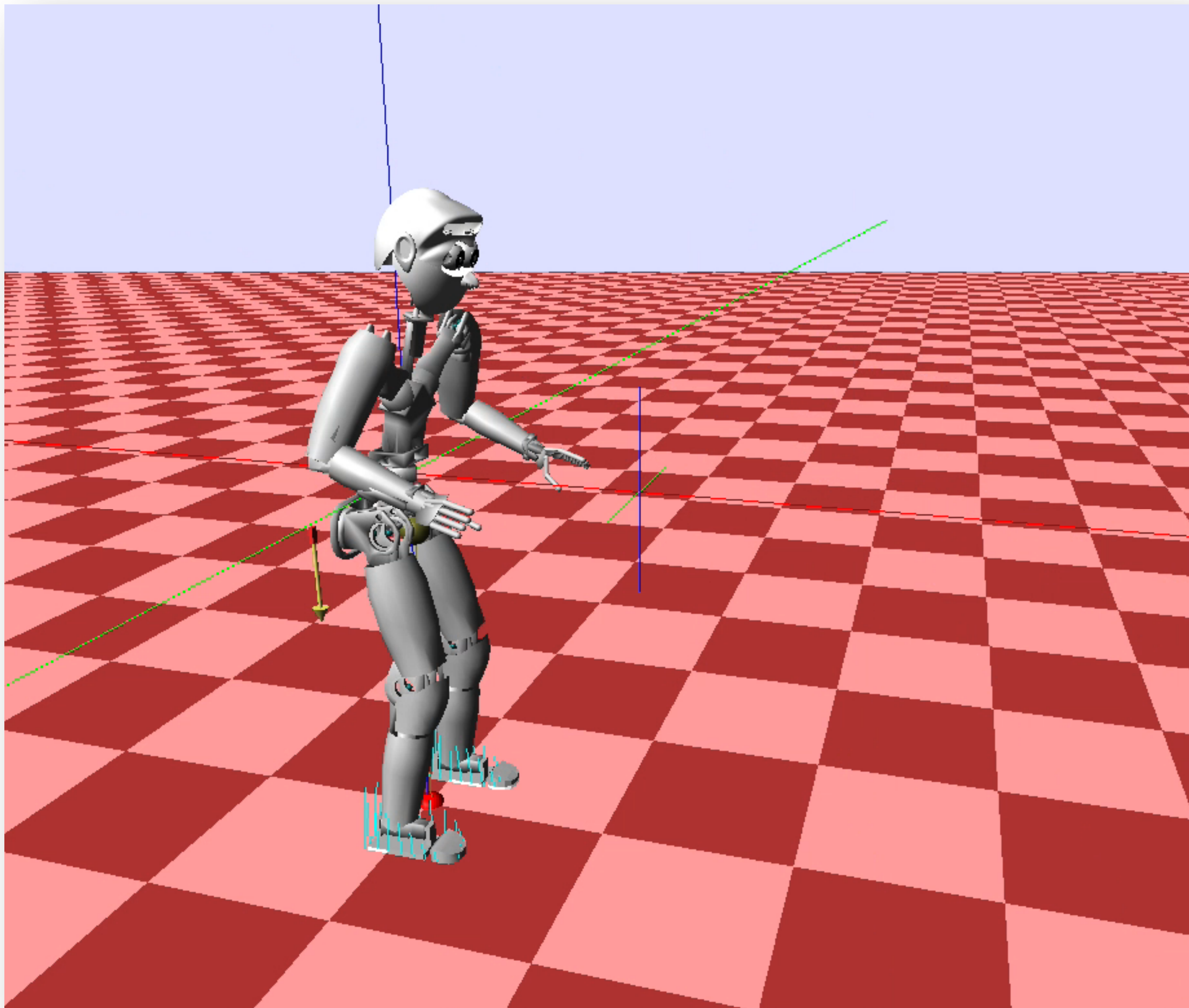




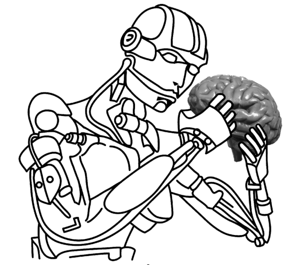
Example: DRC Task



Example: DRC Task



Real-Time Components of SL

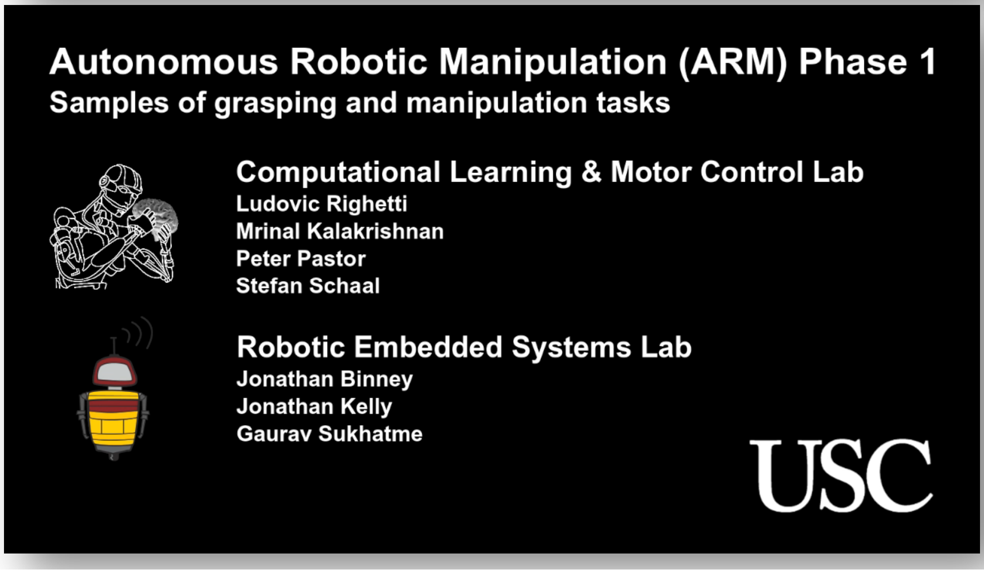


- We Switched to RTOS Xenomai a Few Years Ago
  - Dual kernel Ubuntu patch
  - guaranteed hard real-time when programmed correctly
  - real-time drivers include
    - CAN bus (RT-CAN)
    - Ethernet (RT-NET)
    - USB (RT-USB)
    - Data Acquisition (Analogy)
- Works well with ROS through Interface Process
- Computer Hardware needs to be matched to Xenomai and peripheral boards
- The user code is identical with simulation code, just real-time requirements (no disk access, printf, etc., in real-time threads)

Examples


Learning Locomotion with LittleDog
http://www-clmc.usc.edu
Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry, and Stefan Schaal

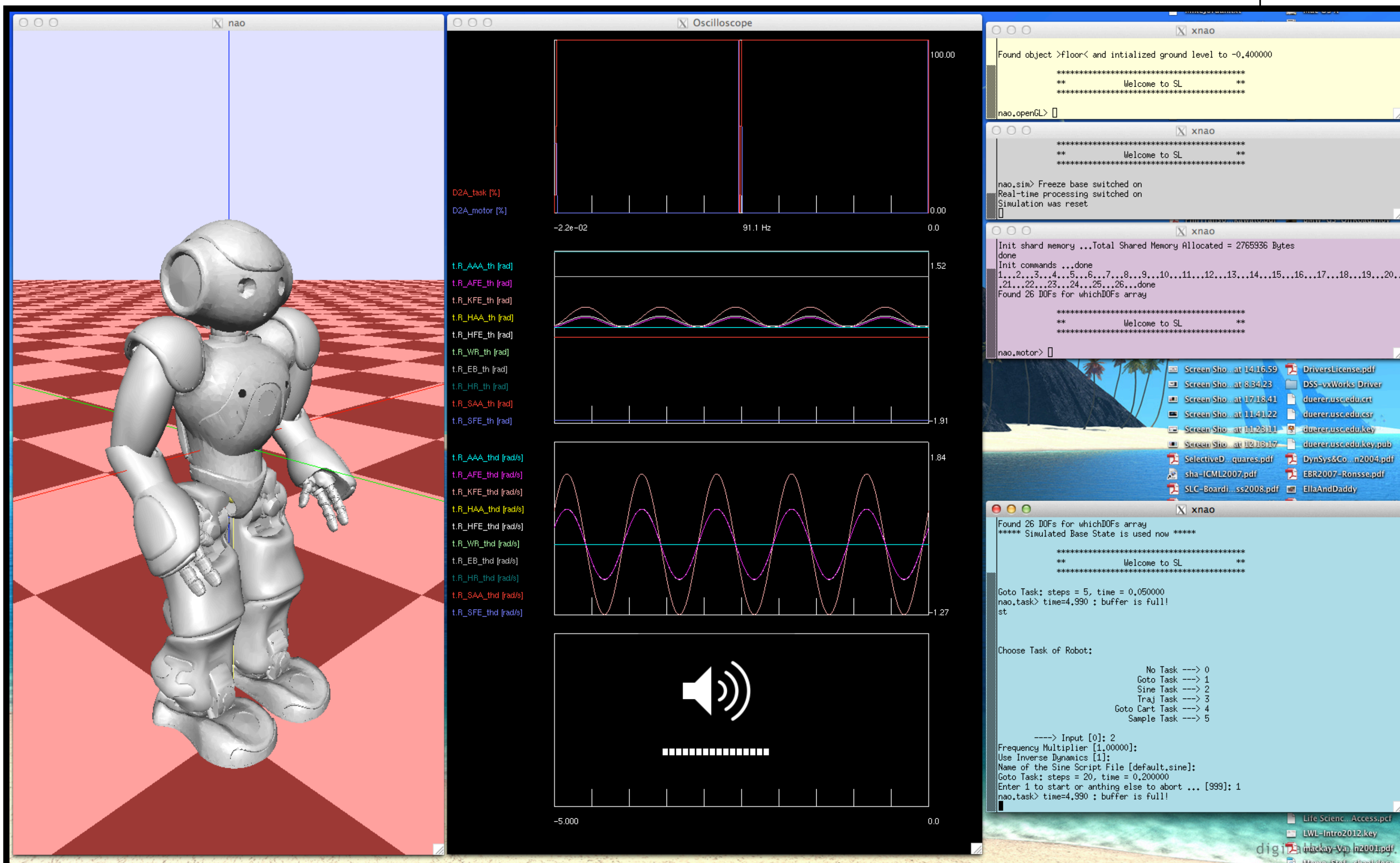

Momentum-based Balance Control for Torque-controlled Humanoids
Alexander Herzog+, Ludovic Righetti+*, Felix Grimminger+, Peter Pastor*, Stefan Schaal+*
+Autonomous Motion Department
Max-Planck-Institute for Intelligent Systems
*Computational Learning and Motor Control Lab
University of Southern California
USC


Autonomous Robotic Manipulation (ARM) Phase 1
Samples of grasping and manipulation tasks
Computational Learning & Motor Control Lab
Ludovic Righetti
Mrinal Kalakrishnan
Peter Pastor
Stefan Schaal
Robotic Embedded Systems Lab
Jonathan Binney
Jonathan Kelly
Gaurav Sukhatme
USC

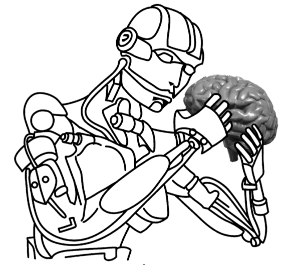
Pros, Cons, Future

- Pros
  - simple, lightweight
  - the same software for real-time control and simulation
  - rapid setup of new robots (days to a week at most)
- Cons
  - should be upgraded to newer software engineering (C++)
  - need better documentation
  - physical contacts based on penalty methods are painful
- Future
  - EIGEN to create Featherstone algorithms?
  - combine Featherstone for RBD with something else for contact dynamics
  - update of user interface
  - maybe RT patch instead of Xenomai?

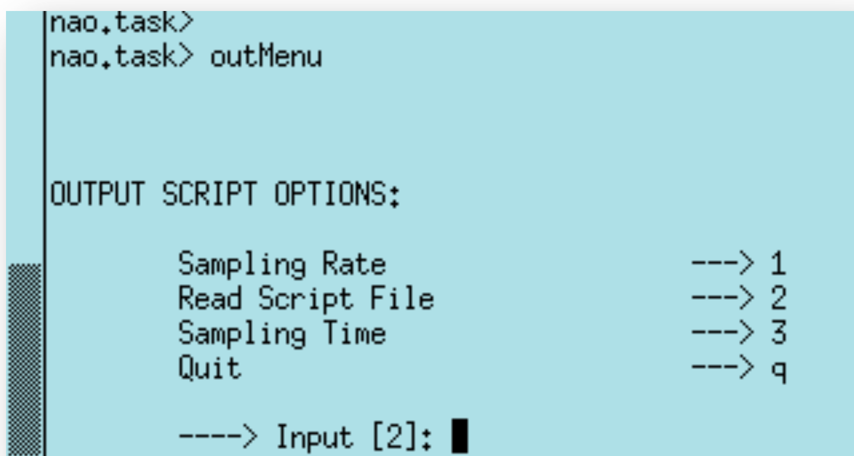
Data Visualization



Data Visualization



- Visualization and debugging tools are CRTICALLY important when working with robot
- SL has
  - Graphics Windows
  - A real-time Oscilloscope
  - CLMCPLOT, a Matlab data visualization
    - Collects select variables in real-time into a memory buffer
    - Allows saving memory buffer to file
    - Visualization in a special Matlab program called CLMCPLOT



Typical Directory Structure of an SL End-User

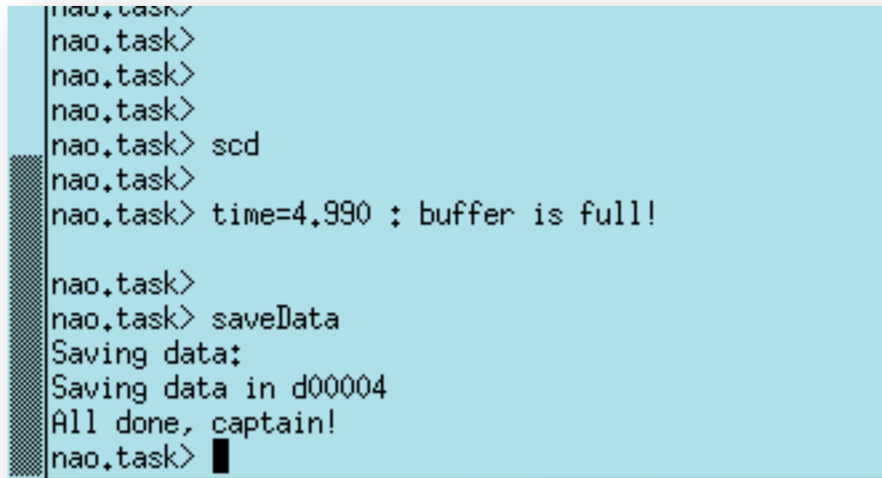
- naoUser/
  - Makefile
  - src/
  - prefs/
    - task_default.script
    - task_sample.script
    - task_default.osc
    - default.sine
    - default_script
    - …
  - config/
  - x86_64mac
  - x86_64
  - x86_64xeno

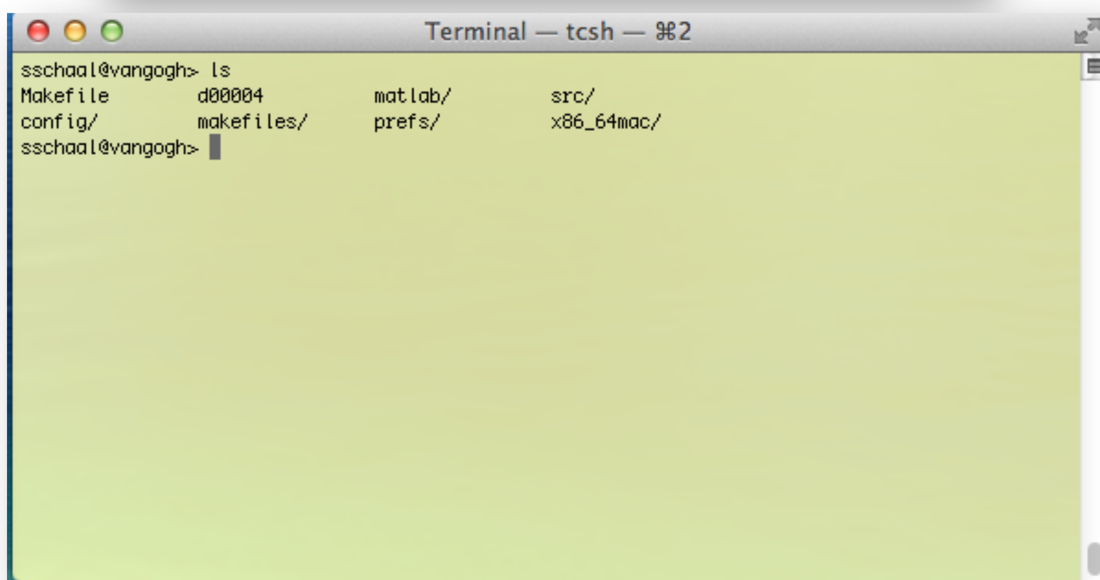
A Data Collection Script: task_default.script

R_SFE_th
R_SFE_thd
R_SFE_thdd
R_SFE_u
R_SFE_ufb
R_SFE_load
R_SFE_des_th
R_SFE_des_thd
R_SFE_des_thdd
R_SFE_uff

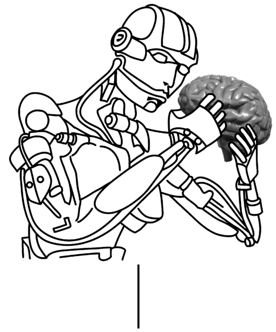
R_SAA_th
R_SAA_thd
R_SAA_thdd
R_SAA_u
R_SAA_ufb
R_SAA_load
R_SAA_des_th
R_SAA_des_thd
R_SAA_des_thdd
R_SAA_uff

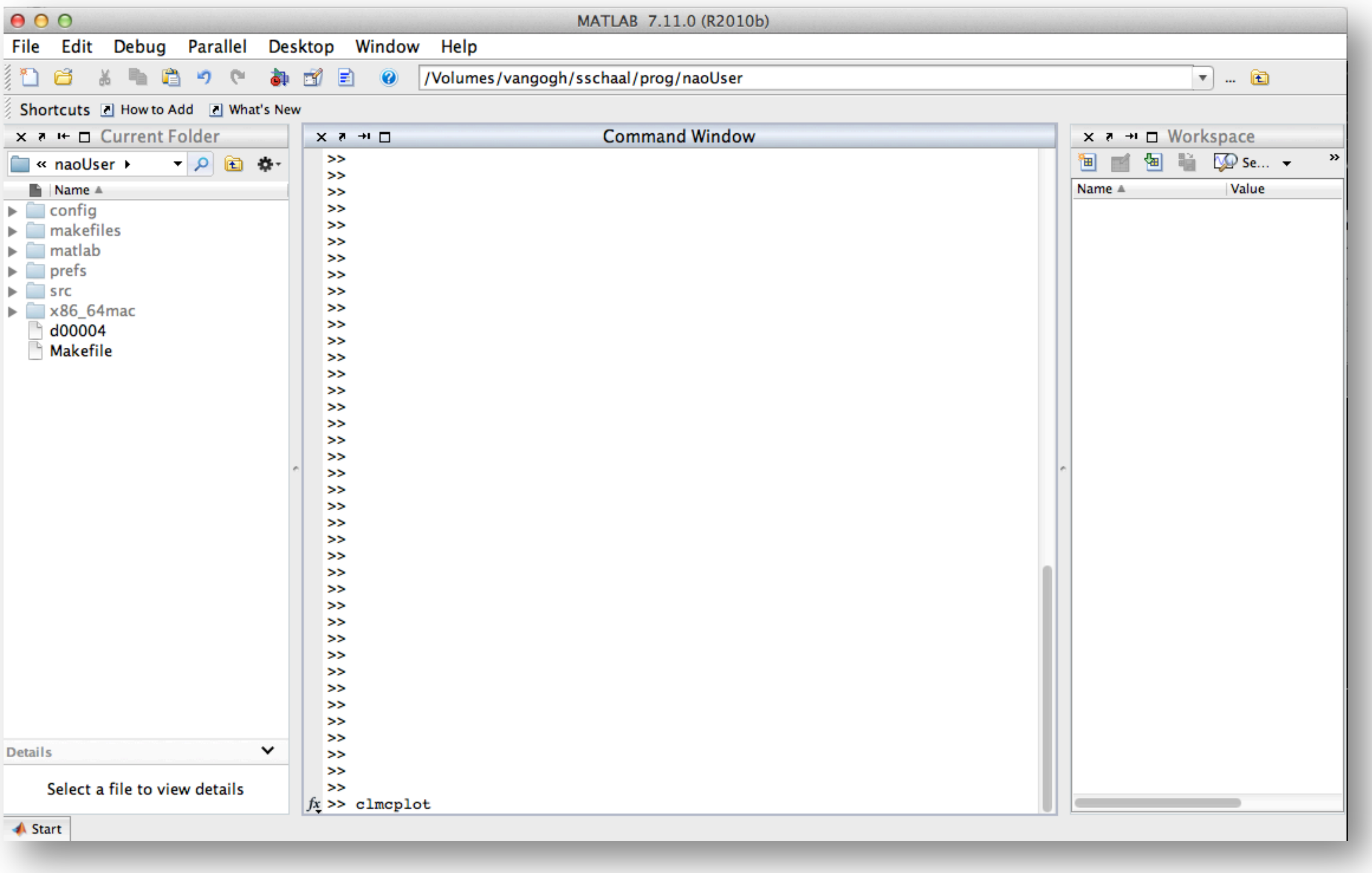
R_HR_th
R_HR_thd
R_HR_thdd
R_HR_u
R_HR_ufb
R_HR_load
R_HR_des_th
…



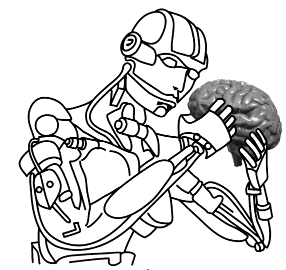


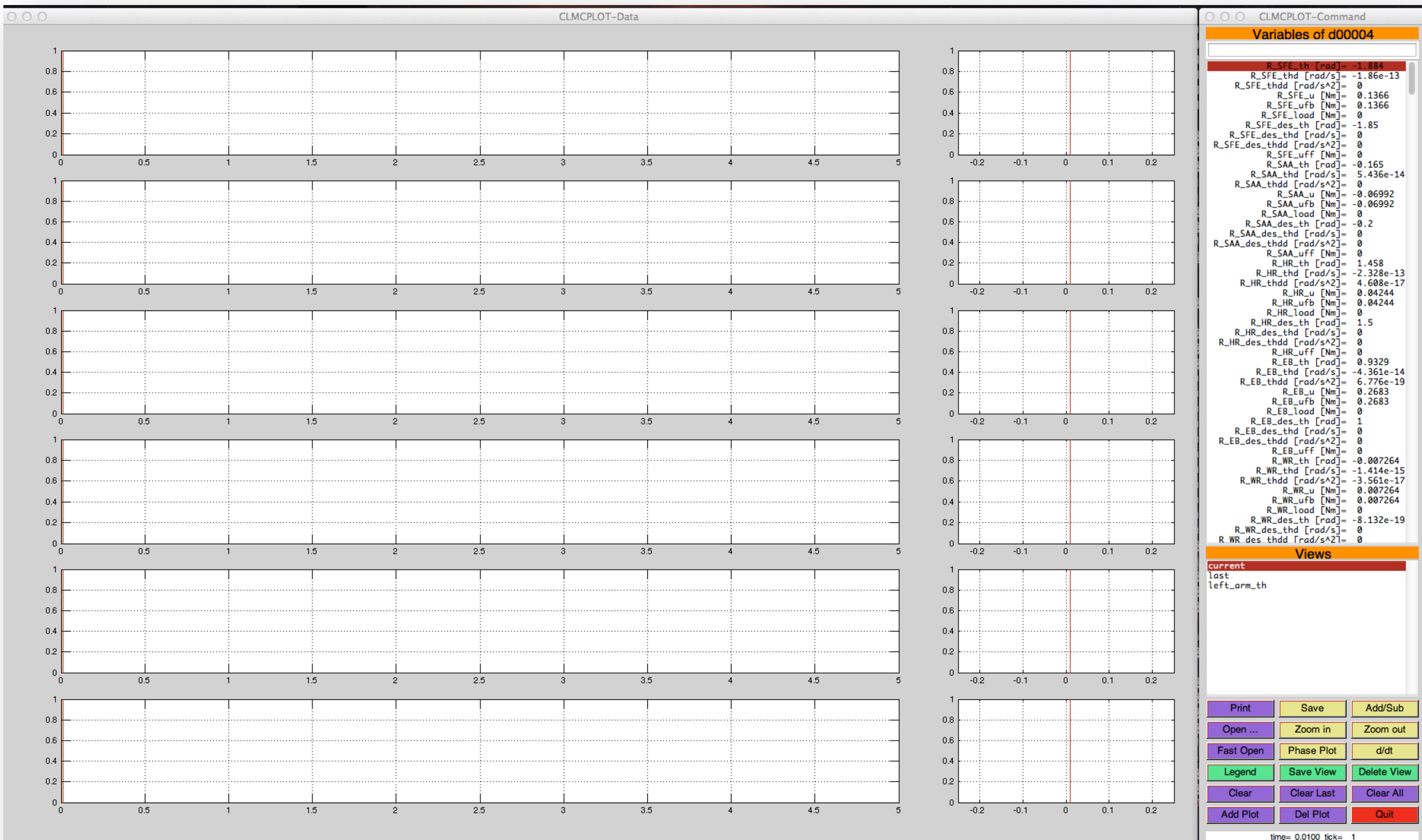
CLMCPLOT in Matlab



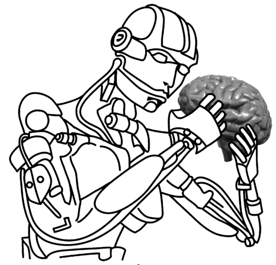


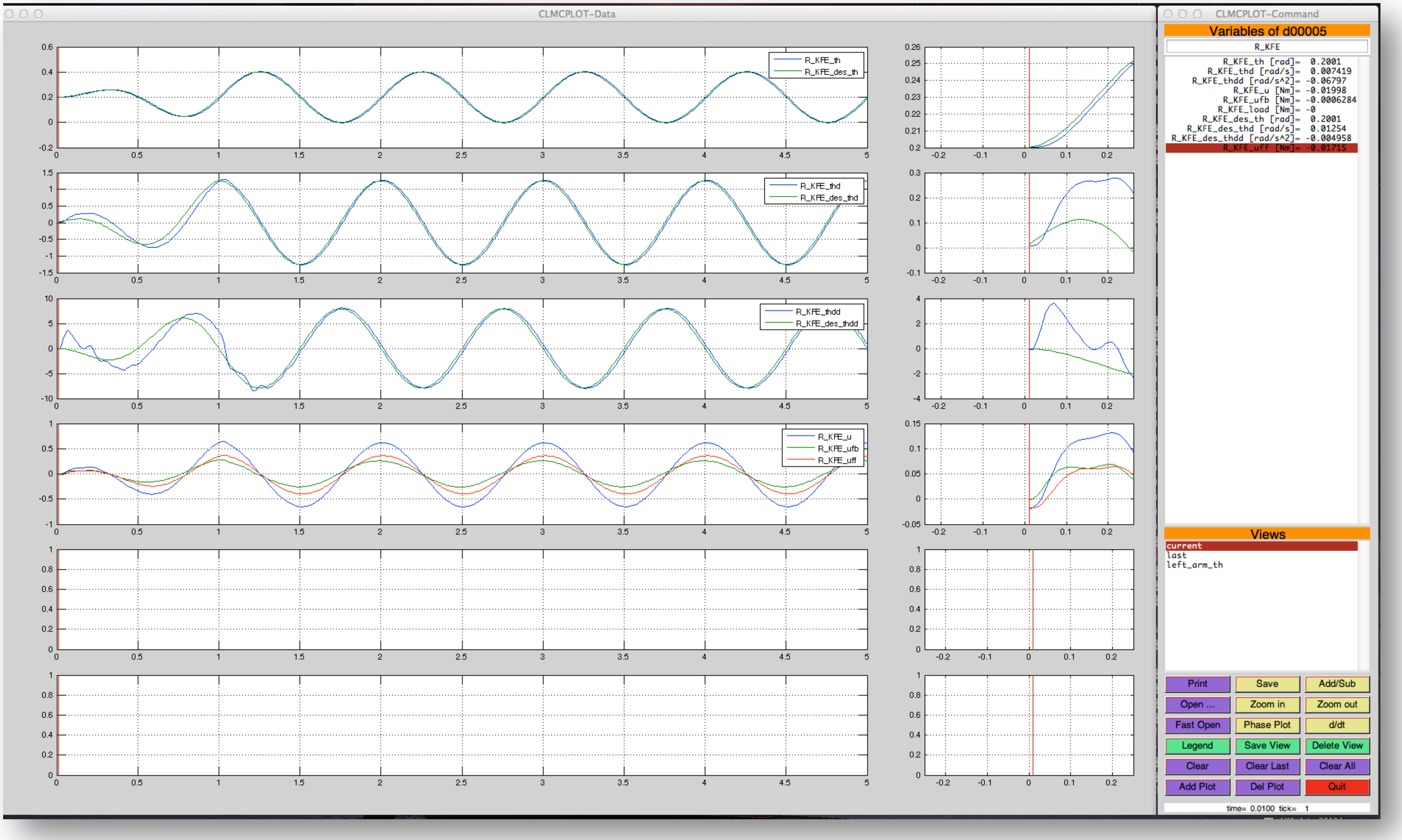
CLMCPLOT in Matlab



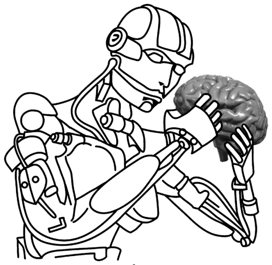


CLMCPLOT in Matlab



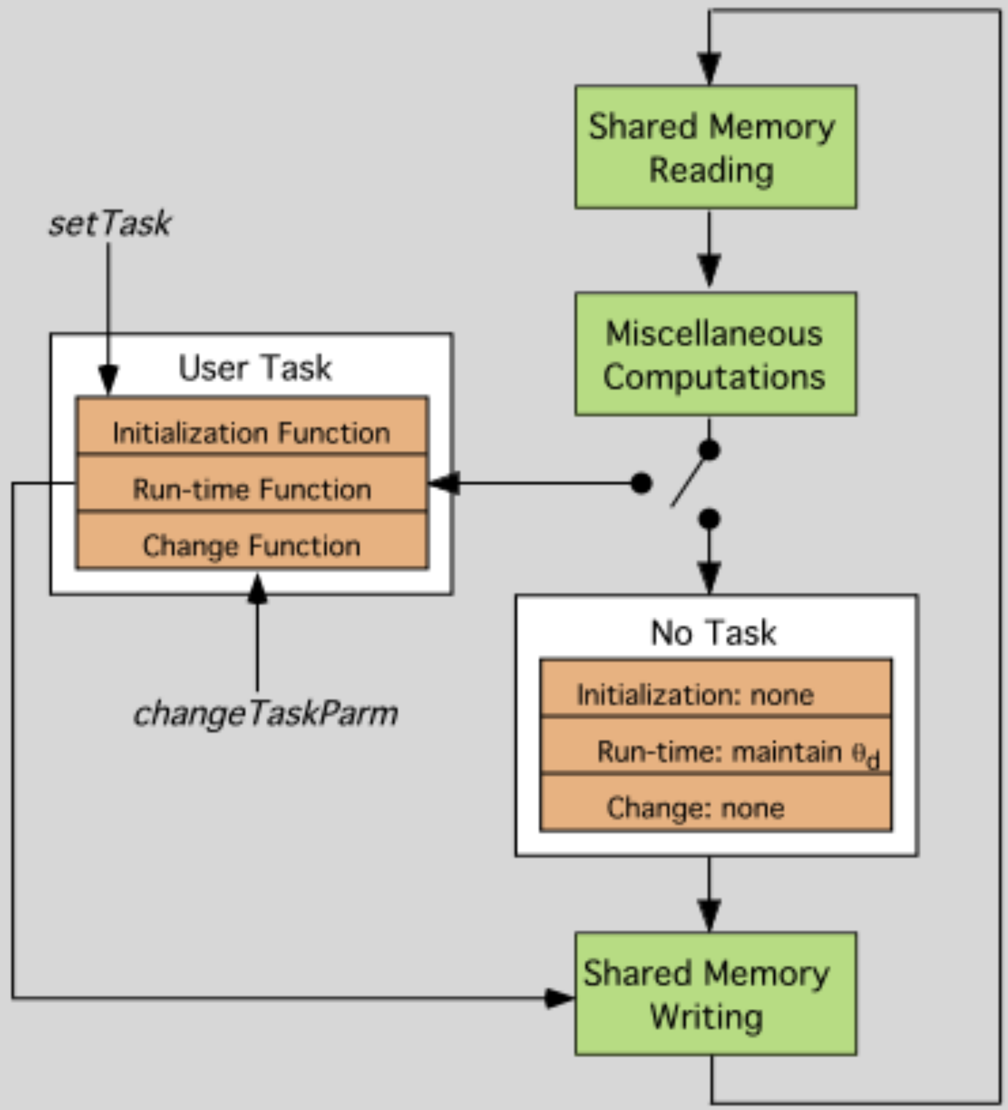


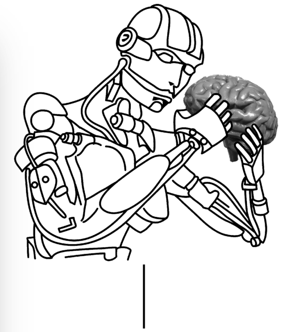
Programming SL: What is happening on the Task-Servo?



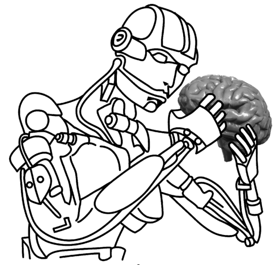
- The Task-Servo just executes Tasks
  - At high sampling rate (e.g., 100Hz for the NAO)
    - Read sensory date from shared memory
    - Generate desired trajectory and feedforward commands
    - Write desired trajectory and feedforward commands to shared memory
- Tasks need to consist of (at least) 3 function
  - Initialization function of the task (not time critical)
  - Run function of the task (real-time)
  - Function to change the parameters of the task (not time critical)

Task Servo


Shared Memory Reading
Miscellaneous Computations
setTask
User Task
Initialization Function
Run-time Function
Change Function
changeTaskParm
No Task
Initialization: none
Run-time: maintain θd
Change: none
Shared Memory Writing

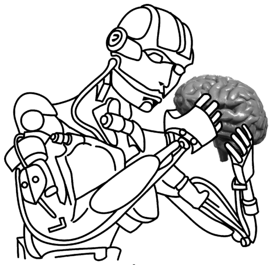


Adding a New Task



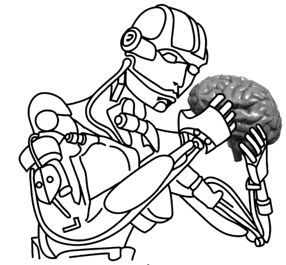
- Write C/C++-functions that contain the 3 required routines
    - (templates: sample_task.c or sample_task_cpp.cpp will be provided)
- Compile the C-code
- Use the setTask (short: st) command in the task_servo to start the task

What is happening in the INIT function?



- Bring the robot to an initial (safe) posture
- Initialize variables
- Trigger task execution

What happens in the RUN function?

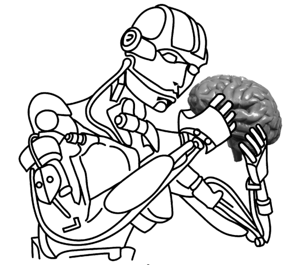


- Assign appropriate values to feedforward commands and desired trajectory variables
  - “joint_des_state” structure receives desired states and u_ff
  - “joint_state” structure has all current state information
- Definition of these structures (see SL.h)
  - SL_Jstate joint_state[N_DOF +1]
  - SL_Dstate joint_des_state[N_DOF+1]
- Possible DOFs: see left.

```
enum RobotDOFs {
  R_SFE = 1,
  R_SAA,
  R_HR,
  R_EB,
  R_WR,
  R_FING,

  L_SFE,
  L_SAA,
  L_HR,
  L_EB,
  L_WR,
  L_FING,

  R_FB,
  R_HFE,
  R_HAA,
  R_KFE,
  R_AFE,
  R_AAA,

  L_FB,
  L_HFE,
  L_HAA,
  L_KFE,
  L_AFE,
  L_AAA,

  B_HR,
  B_HN,

  N_ROBOT_DOFS
};
```

```
typedef struct { /* joint space state for each DOF */
  real   th;   /* theta */
  real   thd;  /* theta-dot */
  real   thdd; /* theta-dot-dot */
  real   u;    /* torque command */
  real   load; /* sensed torque */
} SL_Jstate;

typedef struct { /* desired values for controller */
  real   th;   /* desired theta */
  real   thd;  /* desired theta-dot */
  real   uff;  /* feedforward command */
} SL_DJstate;
```

What happens in the CHANGE function?



- Interactively change variable assignments, e.g., change some gains for gain tuning.
  - Be careful: you can change variables that are in the running program, and a typo could be terrible
  - Read variables into temp variables, check min/max values, and only then assign to variables that are used

CMAKE for creating Makefiles

- CMAKE is open source software
- src/CMakeList.list is the only file you need to change if you add new files for compilation

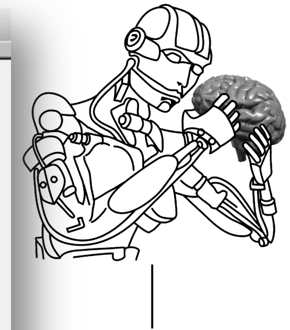
emacs@vangogh

```
###########################################################################
###########################################################################
#
#  This is a CMakeList.txt file originally programmed for the CLMC/AMD labs
#  at the University of Southern California and the Max-Planck-Institute for
#  Intelligent Systems. We use a mixutre of explicit makefiles and cmake, but
#  primarily we rely on cmake for all major compile dependencies. All our
#  software is provided under a slightly modified version of the LGPL license
#  to be found at http://www-clmc.usc.edu/software/license.
#
#  Copyright by Stefan Schaal, 2014
#
###########################################################################
###########################################################################
# which version are we using

cmake_minimum_required(VERSION 2.8)

###########################################################################
# include common cmake components

include($ENV{LAB_ROOT}/config/cmake/LAB.cmake)

###########################################################################
# user defined cmake components

# set global compile type
set(CMAKE_BUILD_TYPE RelWithDebInfo) # Optimization with debugging info
#set(CMAKE_BUILD_TYPE Release)       # Optimization
#set(CMAKE_BUILD_TYPE Debug)         # Debug

# the robot name
set(NAME "nao")

# local defines
include_directories(BEFORE $ENV{LAB_ROOT}/${NAME}/include)
include_directories(BEFORE $ENV{LAB_ROOT}/${NAME}/math)
include_directories(BEFORE ../include)
include_directories(BEFORE ../src)


# ------------------------------------------------------------------------


set(SRCS_XTASK
        initUserTasks.c
        sample_task.c
        sample_task_cpp.cpp
        )

set(SRCS_XOPENGL
        initUserGraphics.c
        )

set(SRCS_XSIM
        initUserSimulation.c
        )
```

An Example C-Program

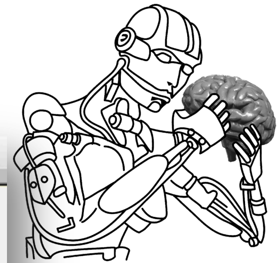
emacs@vangogh

```c
/*=========================================================================
===========================================================================

                              sample_task.c

===========================================================================
Remarks:

      sekeleton to create the sample task

=========================================================================*/

// system headers
#include "SL_system_headers.h"

// SL includes
#include "SL.h"
#include "SL_user.h"
#include "SL_tasks.h"
#include "SL_task_servo.h"
#include "SL_kinematics.h"
#include "SL_dynamics.h"
#include "SL_collect_data.h"
#include "SL_shared_memory.h"
#include "SL_man.h"

// defines

// local variables
static double start_time = 0.0;
static double freq;
static double amp;
static SL_DJstate  target[N_DOFS+1];

// global functions

// local functions
static int  init_sample_task(void);
static int  run_sample_task(void);
static int  change_sample_task(void);

/*****************************************************************************
******************************************************************************
Function Name   : add_sample_task
Date            : Feb 1999
Remarks:

adds the task to the task menu

******************************************************************************
Paramters:  (i/o = input/output)

none

*****************************************************************************/
void
add_sample_task( void )

{
  int i, j;

  addTask("Sample Task", init_sample_task, 
          run_sample_task, change_sample_task);

}
```



An Example C-Program

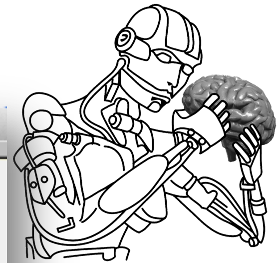
```c
/*****************************************************************************
******************************************************************************
  Function Name : init_sample_task
  Date          : Dec. 1997

  Remarks:

  initialization for task

******************************************************************************
  Paramters:  (i/o = input/output)

       none

 *****************************************************************************/
static int
init_sample_task(void)
{
  int j, i;
  int ans;
  static int firsttime = TRUE;

  if (firsttime){
    firsttime = FALSE;
    freq = 0.1; // frequency
    amp  = 0.5; // amplitude
  }

  // prepare going to the default posture
  bzero((char *)&(target[1]),N_DOFS*sizeof(target[1]));
  for (i=1; i<=N_DOFS; i++)
    target[i] = joint_default_state[i];

  // go to the target using inverse dynamics (ID)
  if (!go_target_wait_ID(target))
    return FALSE;

  // ready to go
  ans = 999;
  while (ans == 999) {
    if (!get_int("Enter 1 to start or anthing else to abort ...",ans,&ans))
      return FALSE;
  }

  // only go when user really types the right thing
  if (ans != 1)
    return FALSE;

  start_time = task_servo_time;
  printf("start time = %.3f, task_servo_time = %.3f\n",
         start_time, task_servo_time);

  return TRUE;
}
```



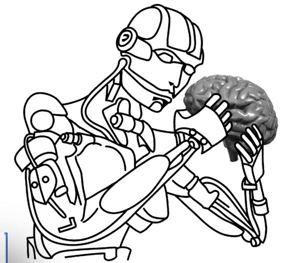
An Example C-Program

emacs@vangogh

```c
/*****************************************************************************
******************************************************************************
  Function Name : run_sample_task
  Date          : Dec. 1997

  Remarks:

  run the task from the task servo: REAL TIME requirements!

******************************************************************************
  Paramters:  (i/o = input/output)

  none

 *****************************************************************************/
static int
run_sample_task(void)
{
  int j, i;

  double task_time;
  double omega;
  int    dof;

  // NOTE: all array indices start with 1 in SL

  task_time = task_servo_time - start_time;
  omega     = 2.0*PI*freq;

  // osciallates one DOF
  dof = 1;
  for (i=dof; i<=dof; ++i) {
    target[i].th   = joint_default_state[i].th +
      amp*sin(omega*task_time);
    target[i].thd   = amp*omega*cos(omega*task_time);
    target[i].thdd  =-amp*omega*omega*sin(omega*task_time);
  }

  // the following variables need to be assigned
  for (i=1; i<=N_DOFS; ++i) {
    joint_des_state[i].th   = target[i].th;
    joint_des_state[i].thd  = target[i].thd;
    joint_des_state[i].thdd = target[i].thdd;
    joint_des_state[i].uff  = 0.0;
  }

  // compute inverse dynamics torques
  SL_InvDynNE(joint_state,joint_des_state,endeff,&base_state,&base_orient);

  return TRUE;
}
```



An Example C-Program



emacs@vangogh

```c
    target[i].thdd  =-amp*omega*omega*sin(omega*task_time);
  }

  // the following variables need to be assigned
  for (i=1; i<=N_DOFS; ++i) {
    joint_des_state[i].th   = target[i].th;
    joint_des_state[i].thd  = target[i].thd;
    joint_des_state[i].thdd = target[i].thdd;
    joint_des_state[i].uff  = 0.0;
  }

  // compute inverse dynamics torques
  SL_InvDynNE(joint_state,joint_des_state,endeff,&base_state,&base_orient);

  return TRUE;
}

/*****************************************************************************
******************************************************************************
  Function Name : change_sample_task
  Date          : Dec. 1997

  Remarks:

  changes the task parameters

******************************************************************************
  Paramters:  (i/o = input/output)

  none

 *****************************************************************************/
static int
change_sample_task(void)
{
  int    ivar;
  double dvar;

  get_int("This is how to enter an integer variable",ivar,&ivar);
  get_double("This is how to enter a double variable",dvar,&dvar);

  return TRUE;

}
```