CS225A : Experimental Robotics

Lecture 6 : Trajectory Control & Haptic Teleoperation

(Aka.. Do the moves..)

Samir Menon

Oct 09, 2014

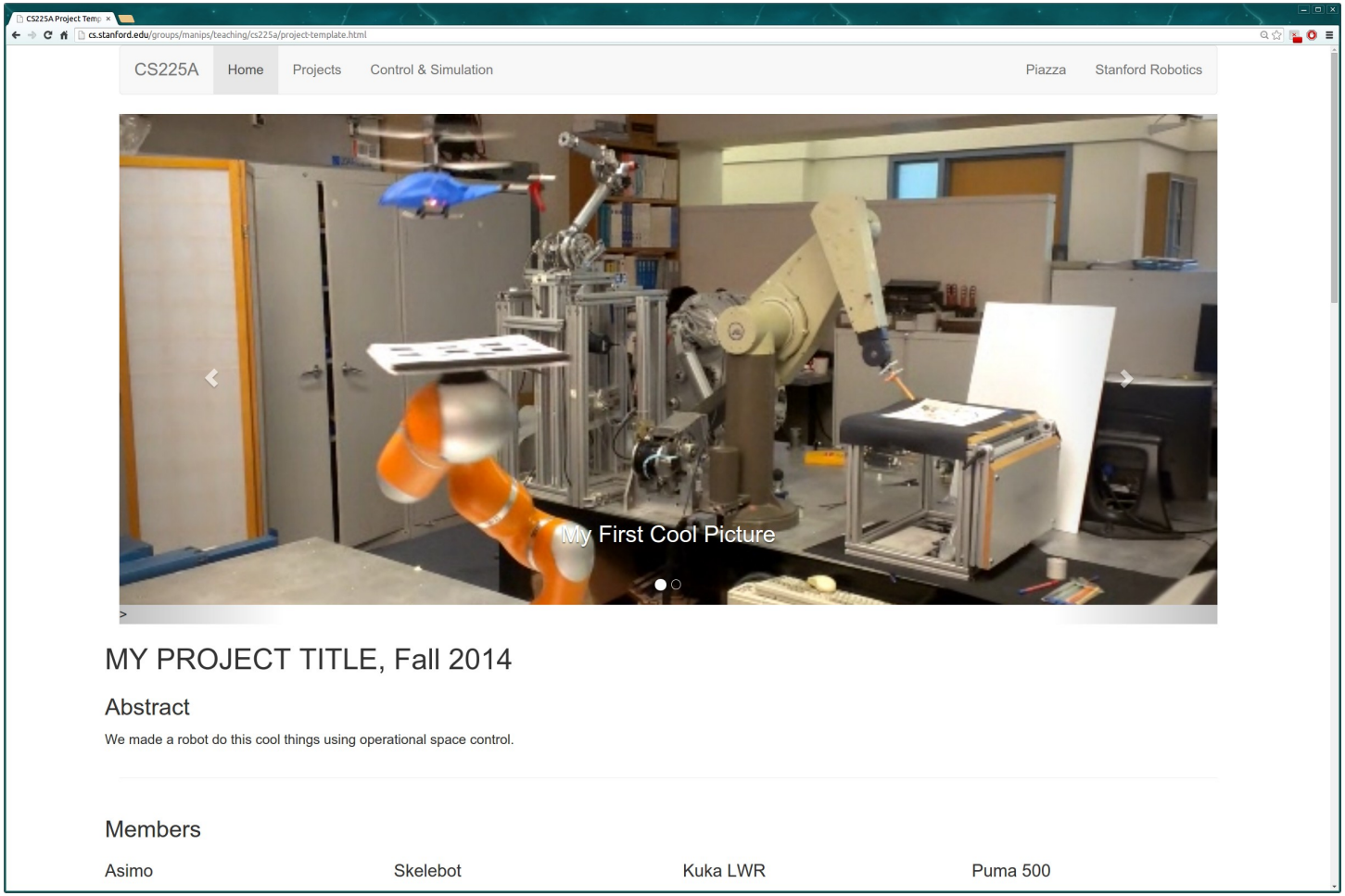
Projects...

- Next deadline : Oct 23; Detailed plan due
  - Specific goals for Milestone 1 (Nov 6)
  - Your (rough) control formulation
  - How you will test your controller (in sim)
  - Any hardware requirements for your project
  - Expected time required on your robot

- Milestone 1:
  - Have some controller implemented in sim

- Milestone 2:
  - Have project practically complete (except for minor fine-tuning)

Projects : Submitting a detailed plan

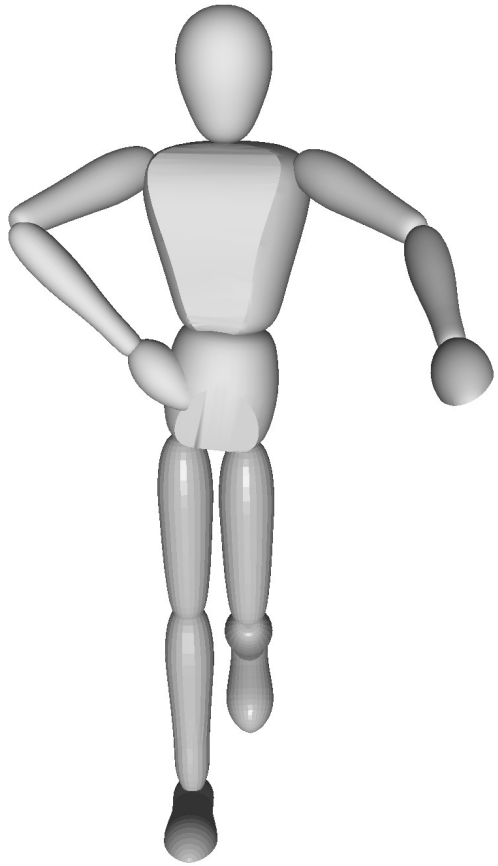


http://cs.stanford.edu/groups/manips/teaching/cs225a/project-template.html

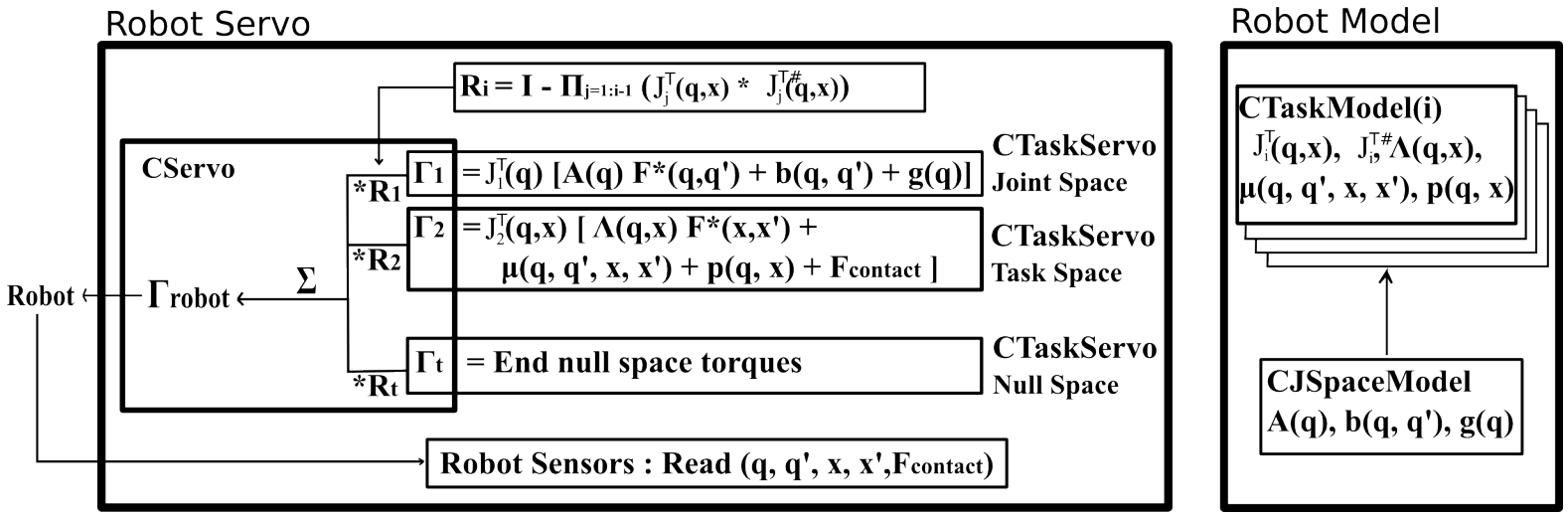
Control Formulation for Multiple Tasks

$$\Gamma = J_t^T \left(\Lambda(x)\ddot{x} + \mu(x, \dot{x}) + p(x) \right)$$

Theory

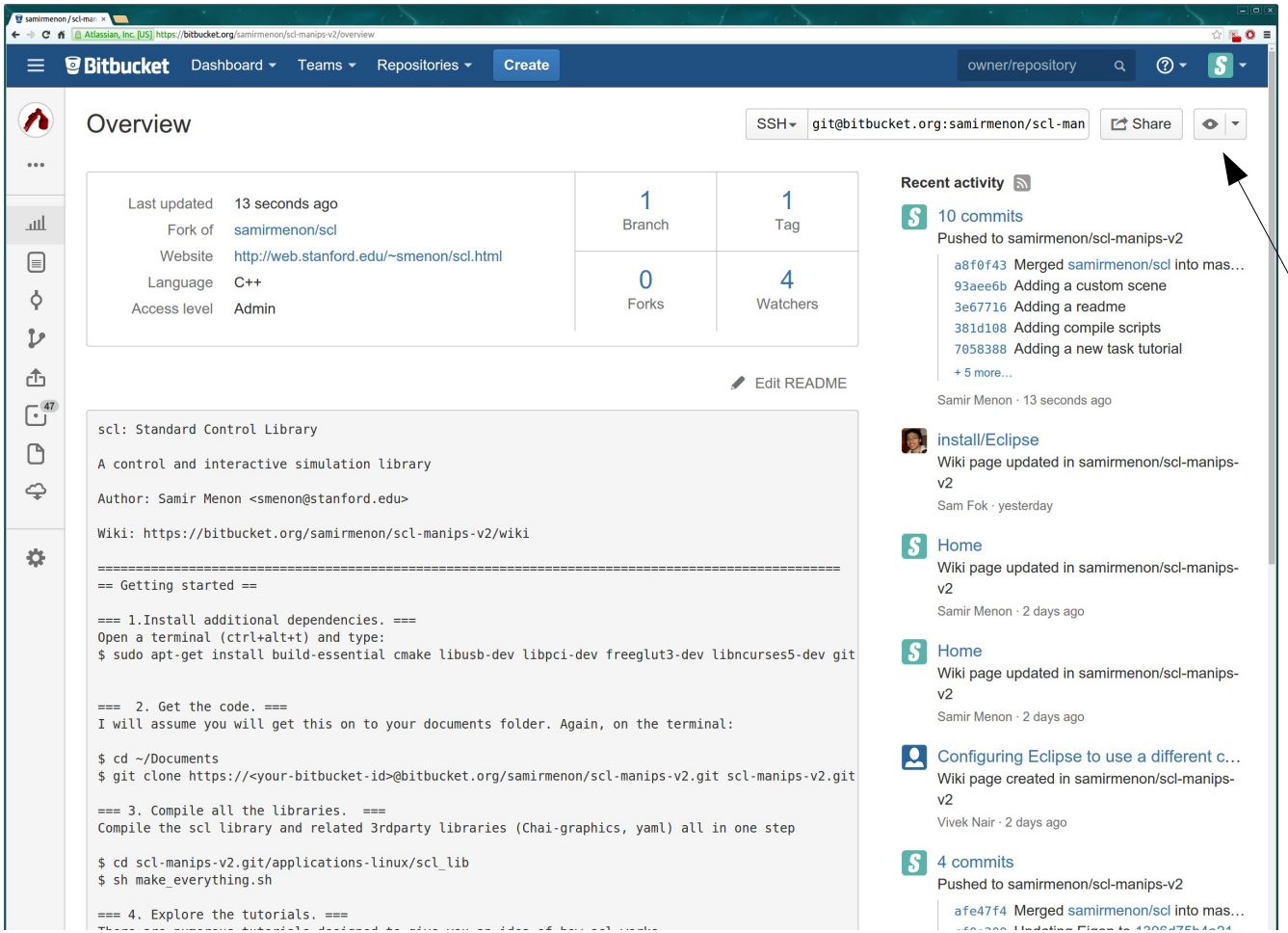


How to implement multiple tasks



- SCL handles most of the code for you
- Your primary job will be to write “Control Tasks” & specify “Control trajectories”
- A task == Some continuous controller in some specific space (e.g., xyz PD control at the hand)

SCL : Watch the repository



Watch

SCL Update : git pull + make_everything

- Please update SCL to get the advanced tutorial

- Get the latest code:
  - $ cd <base_dir>/scl.git
  - $ git stash (save your local changes)
  - $ git pull
  - $ git stash pop (restore your changes)

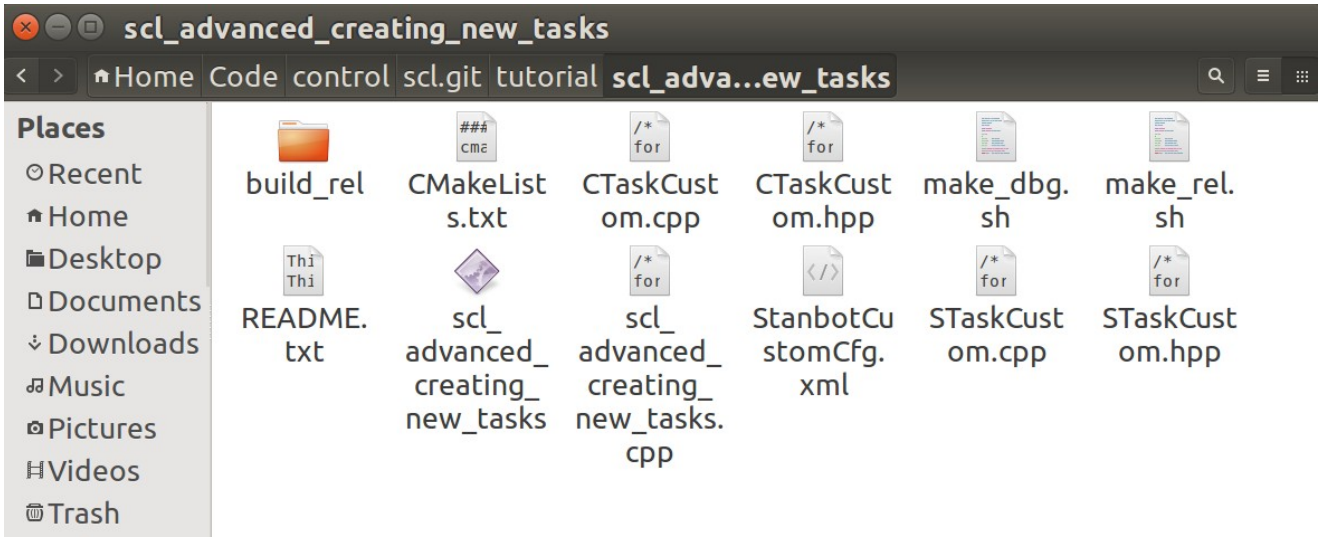
- Rebuild the SCL core library:
  - You have updated scl code, so you need to recompile the library
  - $ cd applications-linux/scl_lib
  - $ sh make_everything.sh

Set up new control task

```
<controller name="opc">
    <type>task</type>
    <must_use_robot>Stanbot</must_use_robot>
    <task name="hand">
        <type>TaskOpPos</type>
        <parent_link>right-hand</parent_link>
        <pos_in_parent>0.01 0.01 0.01</pos_in_parent>
        <priority>0</priority>
        <task_dof>3</task_dof>
        <kp>100</kp>
        <kv>20</kv>
        <ka>0</ka>
        <force_max>1000</force_max>
        <force_min>-1000</force_min>
    </task>

    <task name="hand2">
        <type>TaskCustom</type>
        <parent_link>left-hand</parent_link>
        <pos_in_parent>0.01 0.01 0.01</pos_in_parent>
        <priority>0</priority>
        <task_dof>3</task_dof>
        <kp>100</kp>
        <kv>20</kv>
        <ka>0</ka>
        <force_max>1000</force_max>
        <force_min>-1000</force_min>
    </task>

    <task name="GcTask">
        <type>TaskGc</type>
        <priority>1</priority>
        <task_dof>0</task_dof>
        <kp>100</kp>
        <kv>20</kv>
        <ka>0</ka>
        <force_max>10000</force_max>
        <force_min>-10000</force_min>
    </task>
</controller>
```

- Issue : Specify new task type

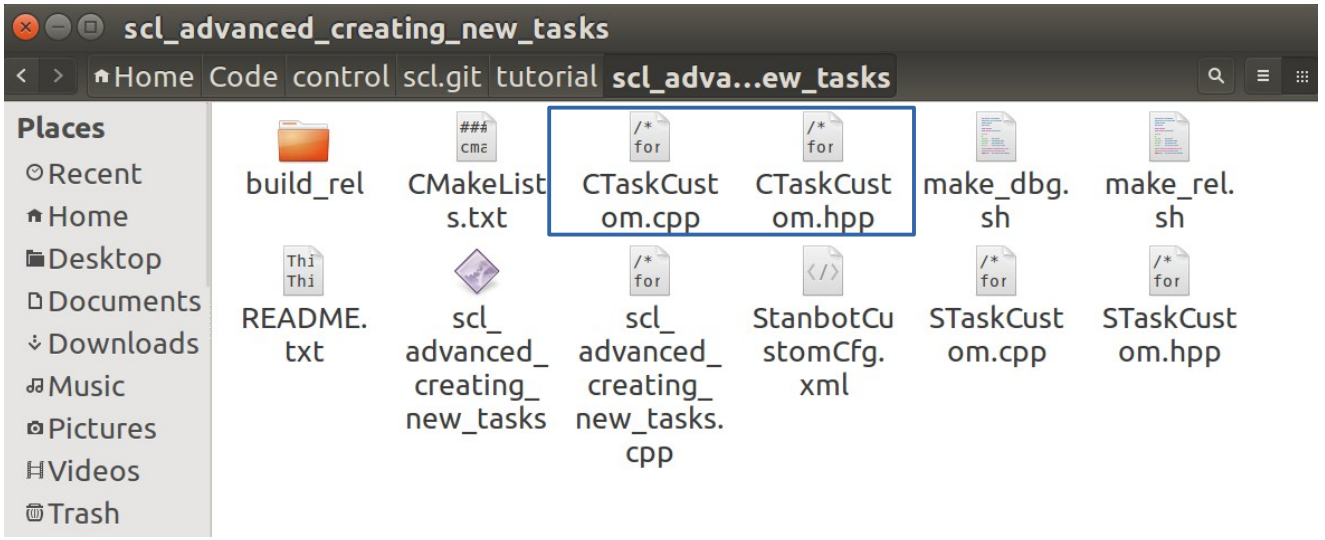
Set up new control task



Advanced tutorial :

scl.git/tutorial/scl_advanced_creating_new_tasks

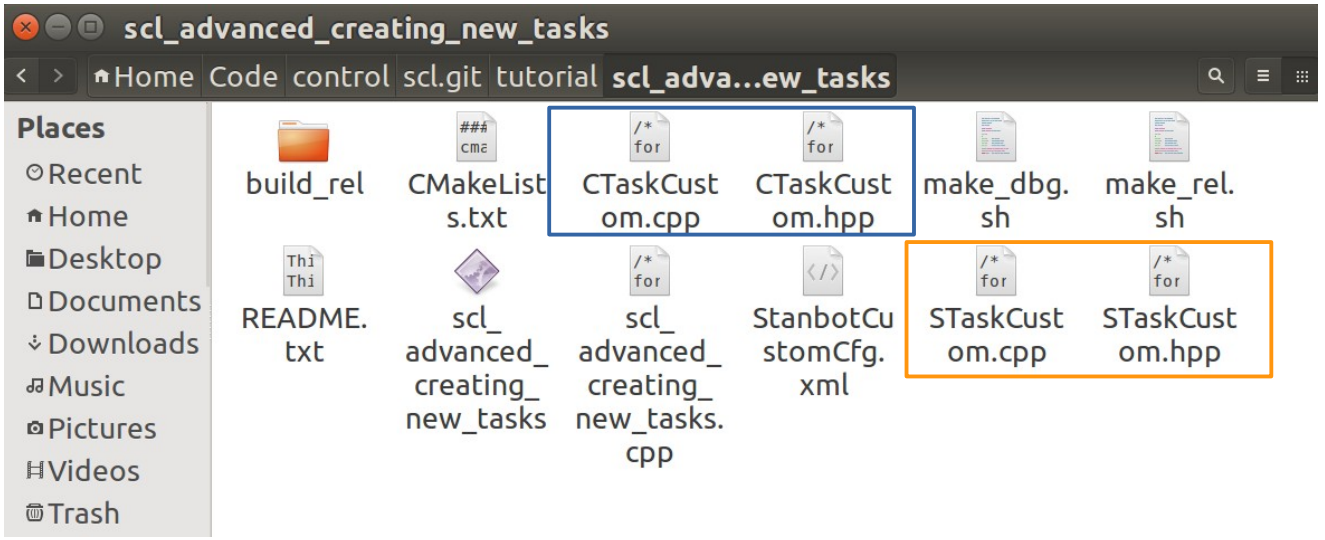
Set up new control task (computation)



Advanced tutorial :

scl.git/tutorial/scl_advanced_creating_new_tasks

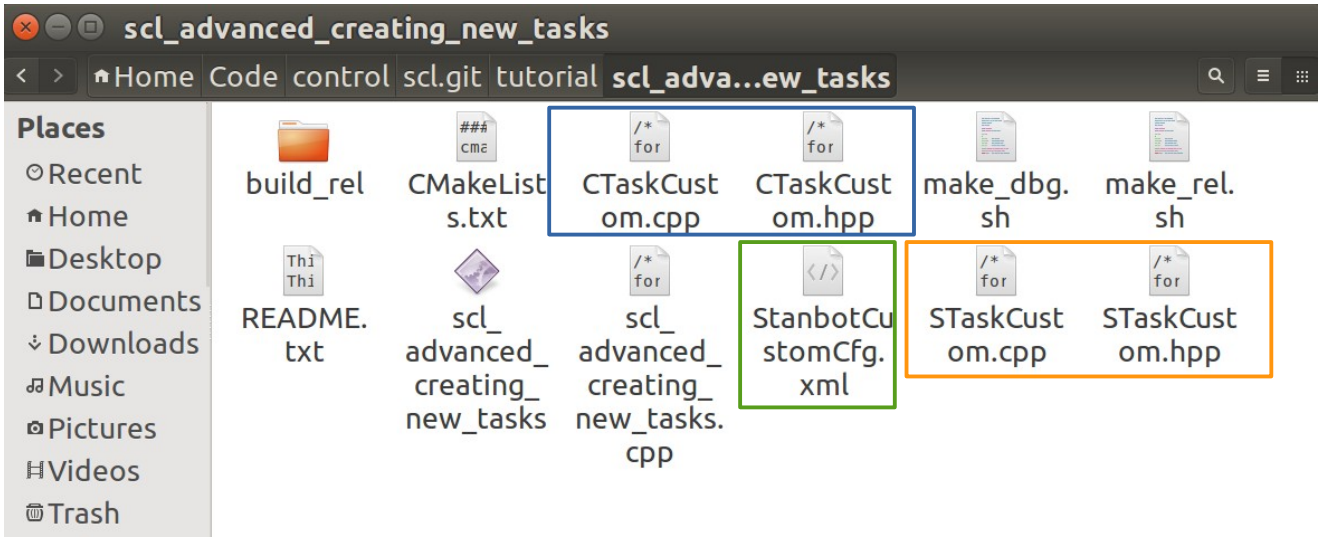
Set up new control task (data structures)



Advanced tutorial :

scl.git/tutorial/scl_advanced_creating_new_tasks

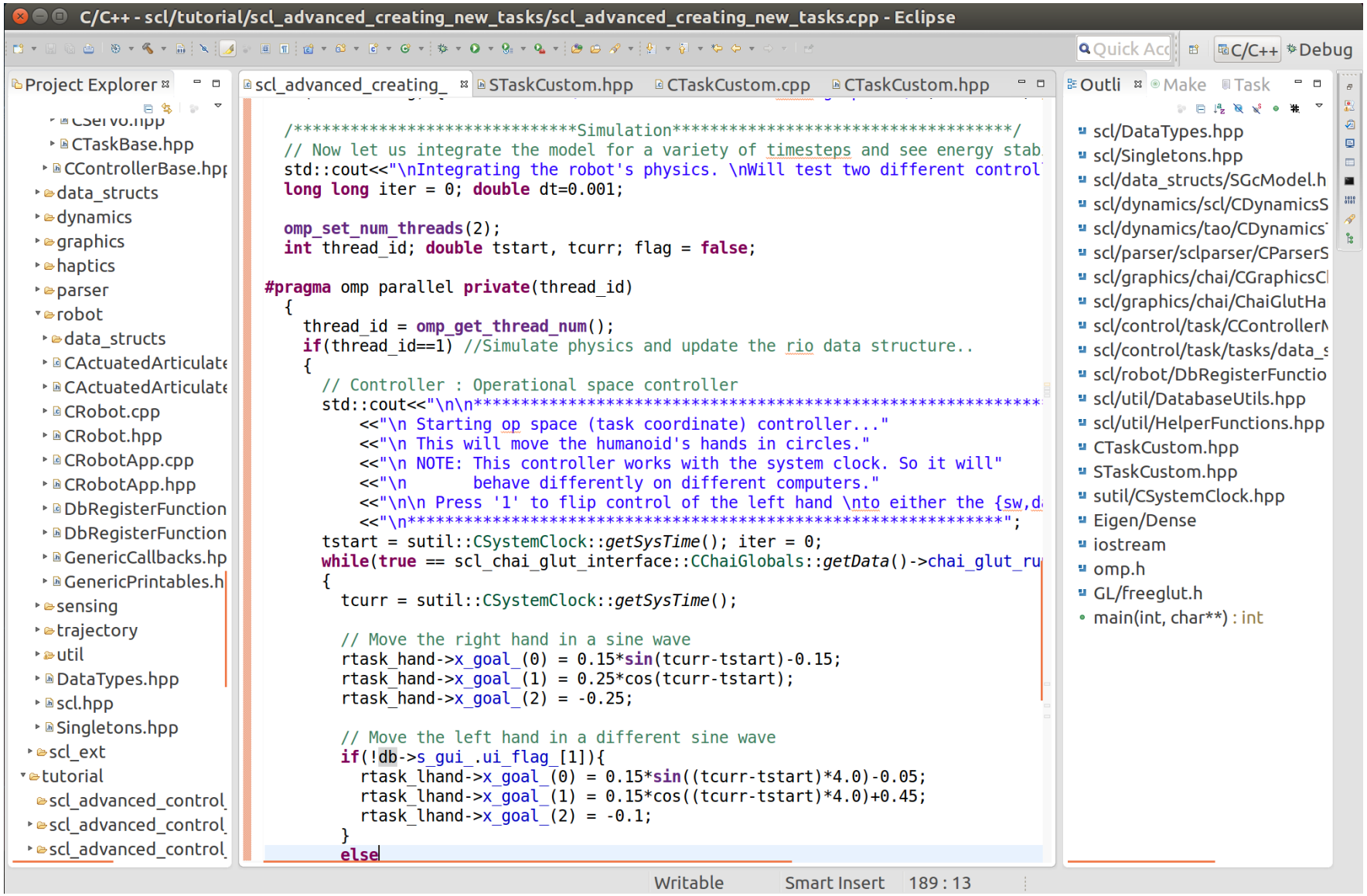
Set up new control task (custom config file)



Advanced tutorial :

scl.git/tutorial/scl_advanced_creating_new_tasks

Before we start : Improve coding/debugging



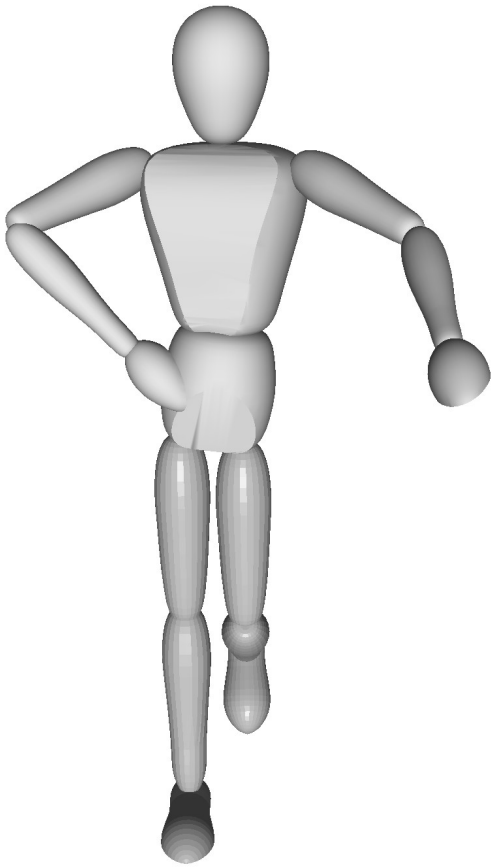

```
/*****************************Simulation***********************************/
// Now let us integrate the model for a variety of timesteps and see energy stab
std::cout<<"\nIntegrating the robot's physics. \nWill test two different control
long long iter = 0; double dt=0.001;

omp_set_num_threads(2);
int thread_id; double tstart, tcurr; flag = false;

#pragma omp parallel private(thread_id)
  {
    thread_id = omp_get_thread_num();
    if(thread_id==1) //Simulate physics and update the rio data structure..
    {
      // Controller : Operational space controller
      std::cout<<"\n\n*************************************************************
          <<"\n Starting op space (task coordinate) controller..."
          <<"\n This will move the humanoid's hands in circles."
          <<"\n NOTE: This controller works with the system clock. So it will"
          <<"\n       behave differently on different computers."
          <<"\n\n Press '1' to flip control of the left hand \nto either the {sw,d
          <<"\n*************************************************************";
      tstart = sutil::CSystemClock::getSysTime(); iter = 0;
      while(true == scl_chai_glut_interface::CChaiGlobals::getData()->chai_glut_ru
      {
        tcurr = sutil::CSystemClock::getSysTime();

        // Move the right hand in a sine wave
        rtask_hand->x_goal_(0) = 0.15*sin(tcurr-tstart)-0.15;
        rtask_hand->x_goal_(1) = 0.25*cos(tcurr-tstart);
        rtask_hand->x_goal_(2) = -0.25;

        // Move the left hand in a different sine wave
        if(!db->s_gui_.ui_flag_[1]){
          rtask_lhand->x_goal_(0) = 0.15*sin((tcurr-tstart)*4.0)-0.05;
          rtask_lhand->x_goal_(1) = 0.15*cos((tcurr-tstart)*4.0)+0.45;
          rtask_lhand->x_goal_(2) = -0.1;
        }
        else
```

Show how to set up compile/debug...

- Set up the compile and debug process "*for your application*" in Eclipse
  - Will make debugging easier
  - The example on the wiki does this for an example control application
  - Replace the application's paths with your own

- Follow the wiki instructions here:
  https://bitbucket.org/samirmenon/scl-manips-v2/wiki/configure/Eclipse2

SCL : Stanbot w/ Prioritized Hand Tasks



Run New Task on Different robot

- Let's do it from scratch...
- Watch and learn how to set up a new scene

Modify the controller : STaskCustom

```
class STaskCustom : public scl::STaskBase
{
public:
  //Computed attributes (last measured, in x dimensional task-space)
  Eigen::VectorXd x_;             //Position in the global frame
  Eigen::VectorXd dx_;            //Velocity in the global frame
  Eigen::VectorXd ddx_;           //Acceleration in the global frame

  Eigen::VectorXd x_goal_;        //Goal Position in the global frame
  Eigen::VectorXd dx_goal_;       //Goal Velocity in the global frame
  Eigen::VectorXd ddx_goal_;      //Goal Acceleration in the global frame

  Eigen::Vector3d pos_in_parent_; //Position in the parent link's local frame (x,y,z)
  std::string link_name_;         //The parent link
  const scl::SRigidBody *link_ds_;     //The parent link's parsed data structure

  scl::sFloat spatial_resolution_;     //Meters

  const scl::SRigidBodyDyn *rbd_;   //For quickly obtaining a task Jacobian

  bool enable_wonky_behavior_;
```

Modify the controller : CTaskCustom

```
//Obtain force to be applied to a unit mass floating about
//in space (ie. A dynamically decoupled mass).
data_->ddx_ = data_->ka_.array() * (data_->ddx_goal_ - data_->ddx_).array();
data_->ddx_ += tmp2 + tmp1;

data_->ddx_ = data_->ddx_.array().min(data_->force_task_max_.array());//Min of self and max
data_->ddx_ = data_->ddx_.array().max(data_->force_task_min_.array());//Max of self and min

if(flag_compute_gravity_)
{ data_->force_task_ = data_->M_task_ * data_->ddx_ - data_->force_task_grav_;  }
else
{ data_->force_task_ = data_->M_task_ * data_->ddx_;  }

if(data_->enable_wonky_behavior_)
{
  std::cout<<"\n  Good force : "<<data_->force_task_.transpose();
  Eigen::Vector3d tmp = Eigen::Vector3d::Random() * data_->force_task_.maxCoeff()*3;
  data_->force_task_ += tmp;
  std::cout<<"\n Wonky force : "<<data_->force_task_.transpose();
}
```

Modify the app : Set up trajectory
scl_advanced_creating_new_tasks.cpp

```
// Move the left hand in a different sine wave
if(!db->s_gui_.ui_flag_[1]){
  rtask_lhand->x_goal_(0) = 0.15*sin((tcurr-tstart)*4.0)-0.05;
  rtask_lhand->x_goal_(1) = 0.15*cos((tcurr-tstart)*4.0)+0.45;
  rtask_lhand->x_goal_(2) = -0.1;
}
else
{ rtask_lhand->x_goal_ = db->s_gui_.ui_point_[0]; }

// Enable wonky behavior if we set a different flag
if(db->s_gui_.ui_flag_[9])
{ rtask_lhand->enable_wonky_behavior_ = true; }
else
{ rtask_lhand->enable_wonky_behavior_ = false;  }
```

Modify the app : Do other stuff
scl_advanced_creating_new_tasks.cpp

```
// Move the left hand in a different sine wave
if(!db->s_gui_.ui_flag_[1]){
  rtask_lhand->x_goal_(0) = 0.15*sin((tcurr-tstart)*4.0)-0.05;
  rtask_lhand->x_goal_(1) = 0.15*cos((tcurr-tstart)*4.0)+0.45;
  rtask_lhand->x_goal_(2) = -0.1;
}
else
{ rtask_lhand->x_goal_ = db->s_gui_.ui_point_[0]; }

// Enable wonky behavior if we set a different flag
if(db->s_gui_.ui_flag_[9])
{ rtask_lhand->enable_wonky_behavior_ = true; }
else
{ rtask_lhand->enable_wonky_behavior_ = false;  }
```

Groups introduce themselves!!!