Software

- main emphasis of this course
- essentially the product of the material in CS223A
  - but is now VERY actionable
- consists of two parts: simulator and controller
Simulator

- input: joint torques, current state \((q, dq)\)
- output: state at next time step
- perform integration over time (time constant matters)
- may also report/resolve contact forces
  - advanced feature, we will say more about this later
Controller

- input: current state, possibly other sensor data (visual, force, etc.)
- output: joint torques to apply
- from current state and kinematics you can compute all necessary quantities
  - like in CS223A...
  - but we do this for you!
- you will implement everything from joint space to operational space control
SAI

▶ used to be SCL but we are working on repackaging it with more utilities
  – thank you to Samir Menon for developing this!
▶ our simulator + controller package
▶ flexible specification of robots
  – and then we compute everything for you
▶ allows for combining multiple tasks into a single controller
▶ framework makes it easy to mix and match robots and controllers
▶ provides visualization (incredibly helpful for debugging)

▶ examples provided through tutorials
  – in fact, it is the homework to go through these

▶ will definitely take some work learn how to use
  – but again, we are here for you
Demo time...
But wait, how many are used to command line?
Real demo time!
SCL architecture

- large but only a few key pieces
  - data: separated static and dynamic
  - dynamics: integration and support for computing important quantities
Data

- static data does not change over the lifetime of the simulation
  - examples: mass properties, link structure
- dynamic data changes over simulated time
  - transformation matrices
  - sensor values (q,dq)
- this is done so separate out where to find things and do “efficient” update

hint: in SCL, most data classes start with “S”
Data in code

scl::SRobotParsed rds;  //Robot data structure....
scl::SGcModel rgcm;  //Robot data structure with dynamic quantities...
scl::SRobotIO rio;  //I/O data structure

▶ the first holds static (parsed) data
  – you won’t need to access this directly much
▶ the second stores dynamic quantities
  – you will want to read values from here
▶ the third stores information about sensors and actuators
  – think of this as the basic interface to the robot

▶ you are encouraged to read the header files at least
  – generally well commented
Dynamics

- contains functions for computing dynamic quantities from code
  - needs state and static information
- also contains integration for stepping simulation

- hint: in SCL, most classes that operate on data start with “C”
Dynamics in code

scl::CDynamicsScl dyn_scl; // Robot kinematics and dynamics computation
scl::CDynamicsTao dyn_tao; // Robot physics integrator...

- SCL supports two dynamics engines, prefer to use CDynamicsScl
  - Tao soon to be deprecated, but still used for integration
- how would you integrate when running a real robot?
Putting it together

Initialization is straightforward, and will be the same for most applications

```plaintext
scl::SRobotParsed rds;
scl::SGcModel rgcm;
scl::SRobotIO rio;
scl::CDynamicsScl dyn_scl;
scl::CDynamicsTao dyn_tao;
scl::CParserScl p;

p.readRobotFromFile("<robo_cfg_file>" ,"<dir>" ,"<robo_name>" , rds);
rgcm.init(rds); //Simple way to set up dynamic tree...
dyn_scl.init(rds); //Set up dynamics object
dyn_tao.init(rds); //Set up integrator object
rio.init(rds.name_, rds.dof_);
```
Putting it together

- to compute all dynamic quantities
  
  ```
  dyn_scl.computeGCModel(...)
  ```

- computed quantities can then be accessed from SGcModel

- in the simple case, use SRobotIO to set robot joint values
  
  ```
  rio.sensor_.q_ << 0,0,1,0,0,1;
  rio.actuators_.force_gc_commanded_ = ...
  ```
To come

- next week: writing direct controllers
- later: reusable method of writing tasks/controllers
  - see src/scl/control/task/tasks/CTaskOpPos.{hpp|cpp} for more info
Implementing a controller on a real robot

- derive/design the control
- test in simulator
- run on robot

- essentially the order of the class
- must go in this order!
  - robots are very fun to use… but very expensive to break
- should alternate between steps 2 and 3 as you add more features
  - for instance, connecting the robot to visual input