Notes and Restrictions on Coding

- The code must follow the C++98 standards to compile and run on the real robot. Please don't use any other standards.
- No dynamic memory allocation. You may not use malloc/free, new/delete. Please allocate space needed statically. Not following this rule may result in not completing the servo loop in time.
- The library has classes for matrices and vectors (`PrMatrix, PrVector, etc`). You can refer to the header files in the `include` directory. A PrVector can be indexed like an array (`gv.tau[0]`), and a PrMatrix can be indexed like a two-dimensional array (`gv.Lambda[0][5]`) or a function (`gv.Lambda(0, 5)`). You can add, subtract, and multiply them using the usual rules of linear algebra, so multiplying a matrix by a vector does exactly what you'd expect. (There's one exception: If you multiply two vectors together, the result is a vector produced by an element-wise multiplication. This is useful for multiplying by k_p or k_v vectors. Use the dot() method if you want a scalar.)
- printf() is too slow to run from a servo loop, so it is aliased to the Ui::Display() method provided in UiAgent.h. The output is delegated to a low-priority task. However, if the program unexpectedly crashes, any enqueued printf() statements will be lost. If you need to force an immediate printf() for debugging purposes, use fprintf(stdout,...) instead. But use fprintf() sparingly: on a QNX machine, an fprintf() statement is likely to `cause` a crash because the servo loop can't finish in time.
- One way to debug crashes, which you might need to do while working on your projects at the end of the quarter, is to analyze a core dump. Run "gdb `program name core dump`", and use the "bt" command to see a stack trace. On the QNX machines, the core dump is in /var/dumps/servo.core. Remember to clean up your core dumps on the computers, or you will run out of disk space!

Global Variables

All the variables and parameters of relevance for the student to access are declared in GlobalVariables.h; All generated code for the assignments and the final project will be located in control.cpp. The student will be able to declare global variables and functions in control.cpp as well.
Note: even though the GUI measures angles in degrees, the internal variables that you'll use in the code use radians. The angles are converted when the data is sent from the GUI to the server. The gv. denotes a variable of a GlobalVariables instance.

State variables

gv.dof	: Degrees of freedom
gv.curTime	: Current simulator time
gv.tau	: Vector of joint torques [in newton-meters]
gv.q	: Vector of current joint space positions [rad]
gv.dq	: Vector of current joint space velocities [rad / sec]
gv.kp	: Vector of position gains (k_p)
gv.kv	: Vector of velocity gains (k_v)
gv.qd	: Vector of desired joint positions [rad]
gv.dqd	: Vector of desired joint velocities [rad / sec]
gv.ddqd	: Vector of desired joint accelerations [rad / sec2]
gv.x	: Vector of current operational space positions
gv.dx	: Vector of current operational space velocities
gv.xd	: Vector of desired operational space positions
gv.dxd	: Vector of desired operational space velocities
gv.ddxd	: Vector of desired operational space accelerations

gv.elbow	: Desired elbow configuration for track control mode
gv.T	: Linear transformation for end-effector position/orientation
gv.Td	: Linear transformation for desired end-effector position/orientation

Kinematics & Dynamics Variables

gv.J	: Jacobian
gv.Jtranspose	: Jacobian transpose
gv.A	: Mass matrix. Also called "M" in some robotics classes.
gv.B	: Centrifugal/coriolis vector
gv.G	: Gravity vector
gv.Lambda	: Mass matrix in operational space
gv.mu	: Centrifugal/coriolis vector in operational space
gv.p	: Gravity vector in operational space
gv.singularities	: Bitmap of singularities
gv.E	: Matrix converting linear/angular velocity to configuration parameters
gv.Einverse	: Inverse of g E matrix

Limit Variables

gv.qmin	: Minimum joint positions allowance [rad]
gv.qmax	: Maximum joint positions allowance [rad]
gv.dqmax	: Maximum joint velocities allowance [rad / sec]
gv.ddqmax	: Maximum joint accelerations allowance [rad / sec2]
gv.taumax	: Maximum joint torques allowance [newton-meter]
gv.xmin	: Vector of minimum operational-space coordinates
gv.xmax	: Vector of maximum operational-space coordinates
gv.dxmax	: Maximum operational space velocity allowance (scalar)
gv.ddxmax	: Maximum operational space acceleration allowance (scalar)
gv.wmax	: Maximum angular velocity in operational space (scalar)

Potential-field Variables

gv.jlimit	: (simulator only) Joint Limits potential field flag
gv.q0	: Vector of minimum distances to apply joint limit potential fields
gv.kj	: Vector of gains for the joint limit potential field
gv.rho0	: Minimum distance to apply potential field controller
gv.eta	: Gain for potential field controller
gv.sbound	: Boundary of singularity [rad]
gv.line	: Structure holding a line for the Line Trajector controller
gv.numObstacles	: Number of obstacles, for the potential field controller
gv.obstacles	: Array of obstacles, for the potential field controller

Functions to be modified in `control.cpp`

The following functions are run once every time a different control mode is invoked :

void InitControl()	: Runs before the first servo loop
void initFloatControl()	: Float Control Mode
void initOpenControl()	: Open-Loop Control Mode
void initNjholdControl()	: Joint Space Non-Dynamic Hold Mode
void initJholdControl()	: Joint Space Dynamic Hold Mode
void initNholdControl()	: Operational Space Non-Dynamic Hold Mode
void initHoldControl()	: Operational Space Dynamic Hold Mode
void initNjmoveControl()	: Joint Space Non-Dynamic Move Mode
void initJmoveControl()	: Joint Space Dynamic Move Mode
void initNjgotoControl()	: Joint Space Non-Dynamic Velocity Saturation Mode
void initJgotoControl()	: Joint Space Dynamic Velocity Saturation Mode
void initNjtrackControl()	: Joint Space Non-Dynamic Cubic Spline Track Mode
void initJtrackControl()	: Joint Space Dynamic Cubic Spline Track Mode
void initNxtrackControl()	: Cartesian Space Non-Dynamic Cubic Spline Track Mode
void initXtrackControl()	: Cartesian Space Dynamic Cubic Spline Track Mode
void initNgotoControl()	: Operational Space Non-Dynamic Velocity Saturation Mode
void initGotoControl()	: Operational Space Dynamic Velocity Saturation Mode
void initNtrackControl()	: Operational Space Non-Dynamic Cubic Spline Track Mode
void initTrackControl()	: Operational Space Dynamic Cubic Spline Track Mode
void initPfmoveControl()	: Potential Field Move Mode
void initLineControl()	: Potential Field Line Track Mode
void initProj1Control()	: User-defined Final Project Control Mode 1

void initProj2Control() : User-defined Final Project Control Mode 2
void initPsroj3Control() : User-defined Final Project Control Mode 3

The following control functions are be continuously executed (computing and sending torque values at the clock rate) as long as a associated control mode is active:

void PreprocessControl() : Runs before the control function
void PostprocessControl() : Runs after the control function
void noControl() : No Control, all Torque set to 0.0
void floatControl() : Arm Floats (gravity compensation)
void openControl() : Open-Loop (no feedback) control (caution !)
void njholdControl() : Arm holds current joint position (Non-Dynamic)
void jholdControl() : Arm holds current joint position (Dynamic)
void nholdControl() : Arm holds current end-effector position (Non-Dynamic)
void holdControl() : Arm holds current end-effector position (Dynamic)
void njmoveControl() : Joint Space, Non-Dynamic Feedback Control
void jmoveControl() : Joint Space, Dynamic Feedback Control
void njgotoControl() : Arm joints move to desired angles with vel.sat.(Non-Dynamic)
void jgotoControl() : Arm joints move to desired angles with vel.sat.(Dynamic)
void njtrackControl() : Arm joints move following cubic spline traject.(Non-Dynamic)
void jtrackControl() : Arm joints move following cubic spline traject.(Dynamic)
void nxtrackControl() : End-effector moves to cartesian coordinates (Non-Dynamic)
void xtrackControl() : End-effector moves to cartesian coordinates (Dynamic)
void ngotoControl() : End-effector follows trajectory with vel.sat. (Non-Dynamic)
void gotoControl() : End-effector follows trajectory with vel.sat. (Dynamic)
void ntrackControl() : End-effector follows a cubic spline trajectory (Non-Dynamic)
void trackControl() : End-effector follows a cubic spline trajectory (Dynamic)
void pfmoveControl() : Potential Field Method, manipulator avoids obstacles
void lineControl() : Line Tracking Method (with potential field)
void proj1Control() : User-defined Final Project Control Mode 1
void proj2Control() : User-defined Final Project Control Mode 2
void proj3Control() : User-defined Final Project Control Mode 3

The following function is useful for debugging. It will execute whenever you type pdebug at the "CS225A:>" prompt:

void PrintDebug() : Print debugging information in response to the pdebug command