The Standard Control Framework

Samir Menon

December 12, 2011

This document describes the development of a novel force-control and dynamic simulation framework. The framework, developed in C++, is a loose collection of modules that can run a dynamics simulation for articulated bodies, control it, and also render it graphically. It contains standard APIs for the necessary modules, and provides default implementations for each API (Fig. **??**). Its various APIs, their implementations and other components are:

1. A Control API
   (a) The Standard Control Library
2. A Rigid Body Dynamics API
   (a) The Tao dynamics engine (kinematics, and forward and inverse dynamics algorithms)
   (b) Interfaces to Real Robots (forward dynamics)
3. A Graphics API
   (a) The Chai graphics engine
4. A File Parsing API
   (a) The Lotus XML file format parser
   (b) The SAI XML file format parser
   (c) The OpenSim XML file format parser
   (d) The Lotus yaml file format parser
5. Data structures for shared memory between modules
6. Callbacks for message passing between modules
7. Helper functions for calling the APIs through an easy to use, high level interface

Any standard application may use one or more of the above modules. A full robot dynamics simulation will typically use all of them, and might add specific controller and callback extensions.

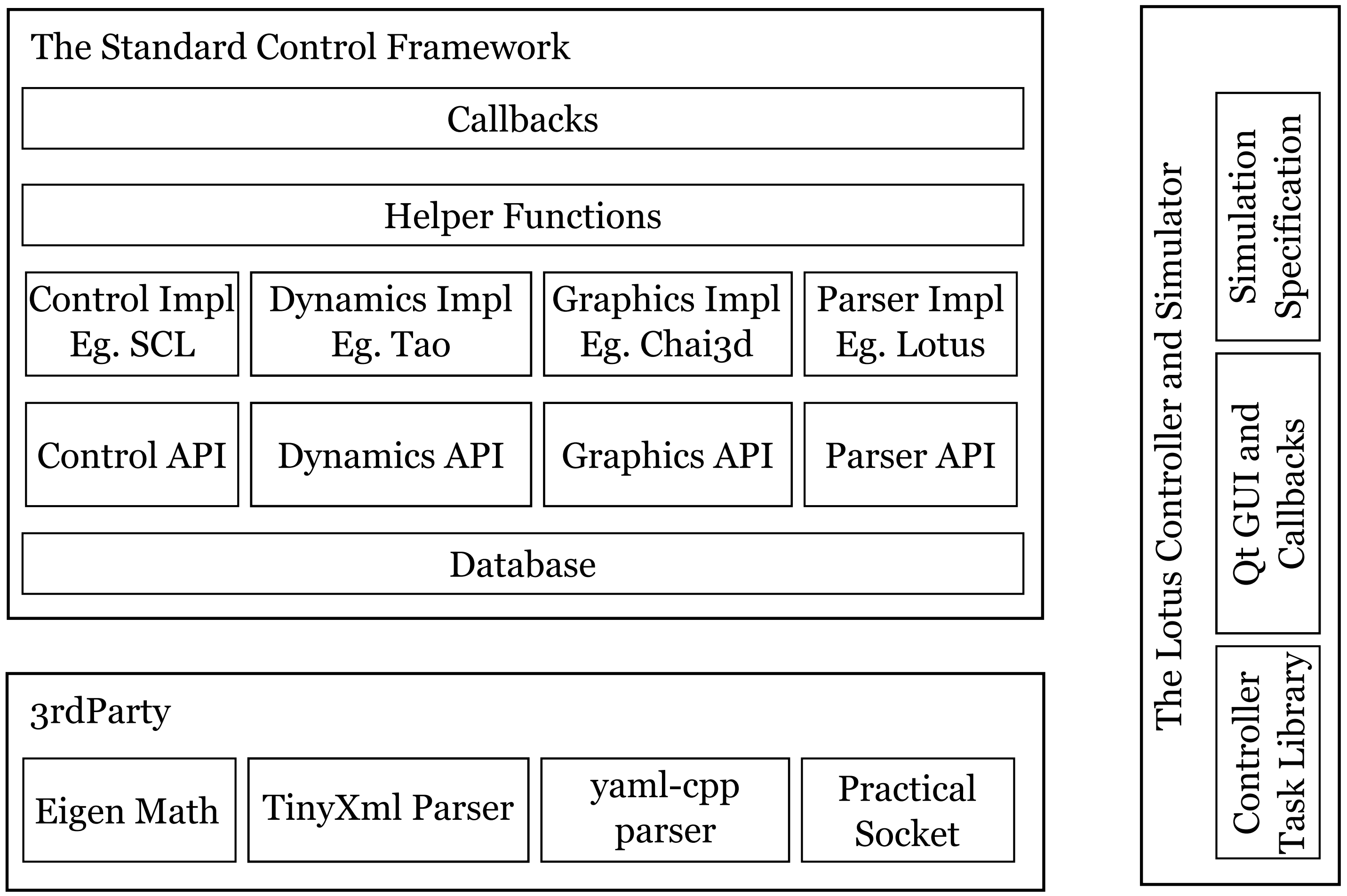

Figure 1: The Lotus Controller and Simulator, the Standard Control Framework, and commonly used third party libraries. Typical applications like Lotus, will link with scf and the third parties and also add their own code to customize user interaction.

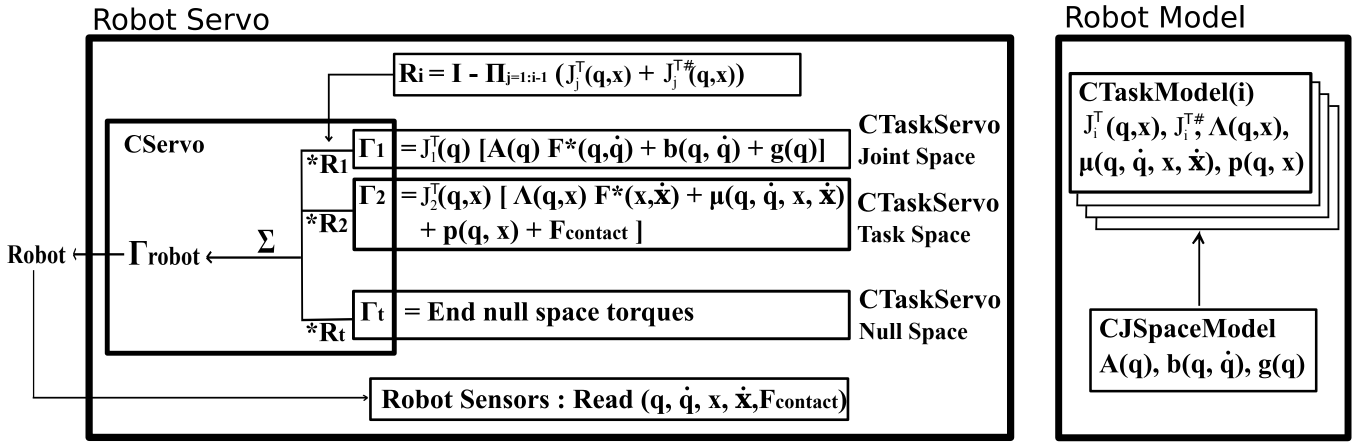
Robot Servo
$R_i = I - \Pi_{j=1:i-1} (J_j^T(q,x) + J_j^{T\#}(q,x))$
CServo
$*R_1$
Γ_1
$= J_1^T(q) [A(q) F^*(q,\dot{q}) + b(q, \dot{q}) + g(q)]$
CTaskServo
Joint Space
$*R_2$
Γ_2
$= J_2^T(q,x) [\Lambda(q,x) F^*(x,\dot{x}) + \mu(q, \dot{q}, x, \dot{x}) + p(q, x) + F_{contact}]$
CTaskServo
Task Space
Robot
Γ_{robot}
Σ
Γ_t
= End null space torques
$*R_t$
CTaskServo
Null Space
Robot Sensors : Read $(q, \dot{q}, x, \dot{x}, F_{contact})$
Robot Model
CTaskModel(i)
$J_i^T(q,x), J_i^{T\#}, \Lambda(q,x), \mu(q, \dot{q}, x, \dot{x}), p(q, x)$
CJSpaceModel
$A(q), b(q, \dot{q}), g(q)$

Figure 2: The Standard Control Library's two-stage task space control architecture.

1 The Control API

The control API enables a dual-rate dynamic force-controllers, where a fast servo loop computes actuator forces and torques, while a possibly slower model update loop computes the dynamic state f the articulated rigid body system (Fig. ??). Our default implementations of this API is:

1.1 SCL : The Standard Control Library

The SCL contains two types of controllers to meet the APIs requirements

Generalized Coordinate Control: The generalized coordinate (gc) controller takes the state of the robot, a gc trajectory and uses a dynamic model to compute actuator control forces. For robots, which are typical articulated rigid body systems, the gc controller generates joint torques.

Task Space Control: The task space controller implements a prioritized multi-level task control strategy that takes multiple tasks, arranges them so that lower priority tasks operate in the null space of higher priority tasks, and finally sums the gc forces that each task generates within its range space. Our task control implementation also includes common operational space control tasks such as operational point position, frame orientation, and null space control.

The tasks are easily specified in an xml file, and typical controllers may be built without writing any C++ code.

2 The Rigid Body Dynamics API

The rigid body dynamics API serves two functions. First to compute the forward dynamics, which involves soving the physics equations of motion for articulated rigid bodies. And second to compute various kinematic and dynamic quantities like kinematic transformations, Jacobian matrices, generalized inertia matrices etc.

2.1 The Tao Dynamics Library

Our default implementation of the dynamics API is Tao. Tao is an open source library released under the MIT license and provides forward and inverse dynamics algorithms along with algorithms to compute the kinematic transformations and Jacobians.

2.2 Interfaces to Robots

Monitoring a real robot's physical motion can be a substitute for solving forward dynamics equations. We are presently working on providing interfaces to the Kuka Lightweight Arm, the Puma robot, and the Pr2 robot.

3 The Graphics API

The graphics API provides basic functions to render a set of rigid and articulated bodies. Essentially, it implements a scene graph of articulated and rigid bodies, and uses some underlying graphics implementation (like OpenGL) to render them. It also computes contacts between different graphical objects, and sends them to the dynamics engine for contact resolution.

3.1 The Chai3d Graphics

The default graphics implementation we use is called Chai3d. Please see "http://www.chai3d.org/" for more details.

4 The Parser API

Since there may be multiple API implementations for the other modules, we decided to standardize a *file format independent* parser API that could allow each module to maintain a specific file format, and at the same time provide a clean interface to wrap all the module file formats together. This has allowed us to simultaneously support multiple file formats:

4.1 Lotus XML

The lotus XML is the most general of our implementations, and provides xml parsing for all the default module implementations.

World: This allows specifying global physical properties such as gravity

Robot: This allows specifying one or more robots in a tree-like branching structure. The robot links are arranged in flat (not recursive) xml tags. Ie. the links are all at the same level in an xml schema, and are arranged into a tree by name and parent name tags at the time of parsing.

Graphics: This allows specifying non-physics objects, lighting, and cameras.

Controller: This allows specifying one or more controllers and their tasks.

Since almost every static parameter can be specified in the file format, writing code is only required for dynamic user interaction.

4.2 SAI XML

We provide legacy file format parsing to convert older SAI XML into Lotus XML.

4.3 OpenSim XML

We also provide file format parsing to convert the biomechanics OpenSim XML file format into Lotus XML.

4.4 Lotus Yaml

We also plan to support a file format based on yaml. Yaml has numerous advantages over xml because it is usually more compact, expressive, easily readable and can also serve as a serialized format for message passing.

5 Shared Memory

The primary method for modules to share information with each other is shared memory, implemented through a shared data structure singleton called the Database.

6 Message Passing

As an alternative to accessing shared memory, we also support a set of statically defined callbacks. The advantage of using template based static callbacks, against using dynamic callbacks using virtual functions, is that our code is more modular, cleaner, and much faster.

7 Helper functions and classes

While the scf exposes its api at a very low level and is primarily geared towards advanced users, most users prefer a simpler high-level interface. To meet their needs, we also support a Robot API, a collection of high-level helper functions and classes that can help quickly build an application.