Control Optimizer Framework for use in Enacting Virtual Fixtures

Paul Wilkening, Anton Deguet, and Russell H. Taylor

This document describes a generalized framework for constructing and solving optimization problems based on a collection of virtual fixtures. The data for these fixtures can be collected from multiple external sources to solve for a set of variables, most often new joint velocities for a robot every cycle of a control loop. This software requires the JHU cisst and SAW libraries.

I. Background

The proposed Control Optimizer framework is able to solve optimization problems that involve objectives, inequality constraints, equality constraints, as well as slack variables. Optimization problems must be of the form:

$\min_x \| C_x x - d\|^2 + \|C_s s\|^2$, $A_x x + A_s s <= b$, $E_x x + E_s s = f$, $L_s <= s <= U_s$

where x is a set of variables representing the solution, C_x and C_s are pieces of the objective matrix, d is the objective vector, A_x and A_s are pieces of the inequality matrix, b is the inequality vector, E_x and E_s are pieces of the equality matrix, f is the equality vector, s is a set of slack variables, and L_s and U_s are the slack limits (see [1,2] for more details). Virtual fixtures to be added to the optimizer must first be properly formatted in order to be processed.

II. Workflow

A. Data Integration

The robot controller must maintain a list of active virtual fixtures and a list of robot state data needed to construct virtual fixtures from external sources. These are fed into the control optimizer, which uses it to solve for a set of incremental robot joint movements. An overview of this design is shown in the figure below in Fig. 1.

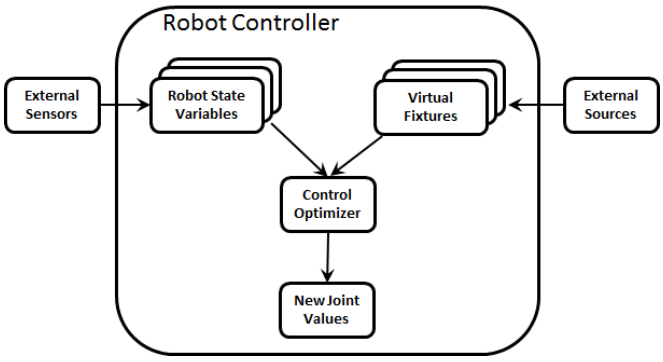

Figure 1 – Overall data flow for combining virtual fixtures with robot data

Note that the "Robot State" variables referenced in Fig. 1 are named for the robot itself but can also include information from external sensors. Virtual fixtures can be created within the robot controller or from external sources as well. The virtual fixtures passed to the robot controller from external sources will usually only be partial sets of data that must be completed by using robot state variables before they can be integrated into the control optimizer.

B. Robot Control Loop

In order to use the control optimizer effectively, its basic usage within a robot control loop must be understood. Pseudocode for putting together the data as in Fig. 1 is shown below in Fig. 2.

```
Process queued commands
Calculate current robot state
Safety check
Reset control optimizer's indices
For every activated virtual fixture
   Increment appropriate control optimizer indices
Allocate space within the control optimizer
Reset control optimizer's indices
For every activated virtual fixture
   Update this virtual fixture's information with robot state data
   Fill in a portion of the control optimizer's tableau
   Increment appropriate control optimizer indices
Solve for dq
Use dq to set the robot's movement for this cycle
```

Figure 2 – Pseudocode for a robot's control loop that takes in new robot state and virtual fixture information and solves for the next incremental motion of the robot.

There are several commands related to this framework that may be under "Process queued commands." These include adding a new virtual fixture to the list, deactivating one already present, or updating the data of one already present. The control optimizer indices are both used to allocate the necessary amount of room in the "tableau," or collection of matrices and vectors as discussed in section I, and provide references to this tableau that each virtual fixture can fill in with its data.

III. Interface

This framework relies upon the control optimizer code found in cisstNumerical as well as abstractions for robot state data and virtual fixtures found in sawControllers. This section serves as an introduction to each of the classes used, but for a more complete guide to the specifics of using this code the document UserManual.docx should be referenced (found in the same directory as this document). osaVFExample.cpp also provides specific examples of how this framework can be used.

A. cisstNumerical

The nmrControlOptimizer class contains the above-mentioned tableau of data and a solver. This is a purely numerical object, and is separate from any virtual fixture-specific logic. This means that the solver can be used for any properly-constructed optimization problems, although its use here is with virtual fixtures. Each piece of the tableau has an

associated index that is set back to zero with the command reset_indices. ReserveSpace increments these indices according to the size of the virtual fixture's tableau being passed as parameters. Once all virtual fixtures are accounted for in the indices, an allocate call will resize all tableau objects to match the indices. These indices are also used to create references to pieces of the tableau in the GetRefs command. The control optimizer passes these references to the tableau to virtual fixtures, which fill them with their data.

B. sawControllers

sawControllers contains classes that represent robot state data and virtual fixture data. osaVFDataPiece is essentially a tableau as seen in the control optimizer, and osaVFData holds on to several of these. osaVFData is able to relate these pieces of the tableau to each other in terms of shared slack variables and contains the names of robot state data objects needed for the subclass of osaVFData to fill its data into the control optimizer in its implementation of the virtual UpdateData command. When a new virtual fixture is passed into the robot controller, these names are used to look up the appropriate robot state data objects maintained in that control loop and initialize the virtual fixture objects.

osaJointState is an object that contains the robot's joint data (positions and velocities). At every cycle, this object is updated with the current joint values. osaKinematics contains information relating to a frame, including its position, velocities, a jacobian and a pointer to an osaJointState. osaSensorValue is an abstract class whose specific implementations can be used to contain information for some kind of external sensor, such as a force sensor. The robot control loop maintains a list of osaKinematics and osaSensorValue indexed by their names. These are used to provide osaVFData objects with the robot state data they require.

REFERENCES

[1] J. Funda, R. Taylor, B. Eldridge, S. Gomory, and K. Gruben, "Constrained Cartesian motion control for teleoperated surgical robots", *IEEE Transactions on Robotics and Automation*, 12- 3 (1996): 453-466.

[2] A. Kapoor, M. Li, and R. H. Taylor "Constrained Control for Surgical Assistant Robots." *IEEE Int. Conference on Robotics and Automation*, Orlando, May 15-19, 2006, pp. 231-236.