Overview

This is a framework for setting up and solving optimization problems that involve some combination of objectives, inequality constraints, equality constraints, and slack variables. This software has the same requirements as the cisst library, as it is in a cisst branch in its current form.

Building

The source and documentation can be checked out from its cisst branch with the following command:

```
svn co https://svn.lcsr.jhu.edu/cisst/branches/2014-02-12-control-optimizer
```

A copy of this document should be in the root directory of the branch. The code is split between two folders in the source code. The control optimizer, or the numerical solver part of the code, can be found in /cisst/cisstNumerical/. The header is in this folder, the cpp files are under /code/, and the tests are under /tests/. The virtual fixture data, or the classes that construct virtual fixtures and are used to populate the control optimizer, can be found in /saw/components/sawControllers/. The header files are under /include/, the cpp files are under /code/, the tests are under /tests/, and the example code is under /examples/.

To build this framework, follow the instructions to build cisst normally (https://trac.lcsr.jhu.edu/cisst/wiki/DownloadAndInstallationFAQ), but ensure that you enable CISST_cisstNumerical, CISST_USE_EXTERNAL, CISST_BUILD_SAW, SAW_Controllers, and CISST_BUILD_TESTS (for control optimizer tests). Now all required code should be compiled when cisst is built.

Testing

In the build directory under /cisst/bin/, there is a cisstNumericalTests executable that includes nmrControlOptimizer tests. Run it to ensure cisstNumerical is functioning properly.
In the build directory under /saw/bin/, there is a sawControllersTests executable that includes osaVFData and osaVFDataPiece tests. Run it to ensure these classes are functioning properly.
In the saw bin folder, there is also an executable named osaVFExample, which is a good place to start when attempting to use this code. The example osaVFExample.cpp provides a good framework for designing specific virtual fixtures and using the vf data objects with the control optimizer. Run this executable to ensure that these tests execute properly.

cisstNumerical/nmrControlOptimizer

Summary	This class is used to set up and solve optimization problems that may use slack variables. There are matrices and vectors assigned to represent the objective, inequality constraint, and equality constraint equations of the form $\min_x \| C_x x – d\|^2 + \|C_s s\|^2$ while $A_x x + A_s s <= b$, $E_x x + E_s s = f$, and $L_s <= s <= U_s$ where x represents a vector of variables, s represents a vector of slack variables, and L_s and U_s represents vectors of slack limits.
Variables	vctDoubleMat C: Objective matrix vctDoubleVec d: Objective vector vctDoubleMat A: Inequality constraint matrix vctDoubleVec b: Inequality constraint vector vctDoubleMat E: Equality constraint matrix vctDoubleVec f: Equality constraint vector size_t NumVars: Number of variables in q vector size_t Slacks: Number of slacks assigned size_t CIndex: Objective row index (can be used for allocation/assignment) size_t AIndex: Inequality constraint row index (can be used for allocation/assignment) size_t EIndex: Equality constraint row index (can be used for allocation/assignment) size_t SlackIndex: Slack index (can be used for allocation/assignment)
Methods	nmrControlOptimizer(int n): Constructor that sets the number of variables reset_indices(): Sets all index variables to 0 ReserveSpace(size_t CRows, size_t ARows, size_t ERows, size_t num_slacks): Add each parameter to the current index values allocate(): Resize the matrices and vectors (if necessary) to match the current index values allocate(size_t CRows, size_t CCols, size_t ARows, size_t ACols, size_t ERows, size_t ECols): Resize the matrices and vectors (if necessary) to match the parameters GetRefs(size_t CRows, size_t ARows, size_t ERows, size_t num_slacks, vctDynamicMatrixRef<double> & CData, vctDynamicMatrixRef<double> & CSlacks, vctDynamicVectorRef<double> & dData, vctDynamicMatrixRef<double> & AData, vctDynamicMatrixRef<double> & ASlacks, vctDynamicVectorRef<double> & bData, vctDynamicMatrixRef<double> & EData, vctDynamicMatrixRef<double> & ESlacks, vctDynamicVectorRef<double> & fData): Gets references to the matrix data portion, matrix slack portion, and vectors for the six data objects. The number of rows and slacks included are passed in as parameters

Usage	nmrControlOptimizer co(q.size());
	while(solving) {
	    co.reset_indices();
	    for(int i = 0; i < pieces.size(); i++) {
	        co.ReserveSpace(pieces.at(i).sizes());
	    }
	    allocate();
	    co.reset_indices();
	    for(int i = 0; i < pieces.size(); i++) {
	        co.GetRefs(pieces.at(i).refs());
	        pieces.at(i).UpdateData(co);
	    }
	    co.solve(q);
	}

sawControllers/osaKinematics

Summary	This class is used to hold on to a collection of robot state data.
Variables	std::string name: Name of kinematics object vctFrm3 frame: Current frame vct3 cartVel: Cartesian velocity of frame vct3 angVel: Angular velocity of frame JacobianType jac: Jacobian osaJointState * js;
Methods	osaKinematics(): Constructor updateFrame(vctFrm3 f): Updates the current frame value updateCartVel(vct3 cv): Updates the current Cartesian velocity values updateAngVel(vct3 av): Updates the current angular velocity values updateJacobian(JacobianType j): Updates the current jacobian value setJointState(osaJointState * js): Sets the current joint state

Usage	osaKinematics k();
	k.updateFrame(RobotState.EndEffector());
	k.updateCartVel(CartesianVelocity);
	k.updateAngVel(AngularVelocity);
	k.updateJacobian(EEJacobian);
	k.setJointState(&JS);
	kinematics.push_back(k);
	osaVFData virtualFixture;
	std::vector<std::string> kinNames;
	kinNames.push_back(“End Effector”);
	virtualFixture.SetKinNames(kinNames);
	virtualFixture.Initialize(kinematics, sensorValues);

sawControllers/osaJointState

Summary	This class is used to hold on to a collection robot joint data.
Variables	std::string name: Name of joint state object vctFixedVector<double> jointPos: Current joint positions vctFixedVector<double> jointVel: Current joint velocities
Methods	osaJointState(): Constructor updatePosition(vctFixedVector<double> jp): Updates the current joint positions updateVelocity(vctFixedVector<double> jv): Updates the current joint velocity values

Usage	osaJointState j();
	j.updatePosition(JointPosition);
	j.updateVelocity(JointVelocity);
	kin.setJointState(&j);

sawControllers/osaSensorValue

Summary	This class is used to hold on to a pointer to sensor data.
Variables	vctDynamicVector<double> values;
Methods	nmrSesnorValue(): Constructor updateValues(vctDynamicVector<double>): Updates the current value of the sensor data

Usage	nmrSensorValue f();
	f.updateValues(FTSensor);
	sensorValues.push_back(f);
	osaVFData virtualFixture;
	std::vector<std::string> sensorNames;
	sensorNames.push_back("Force");
	virtualFixture.SetSensorNames(sensorNames);
	virtualFixture.Initialize(kinematics, sensorValues);

sawControllers/osaVFData

Summary	This class is used to hold on to part of the virtual fixture data to be processed and the names of the robot state data needed for it to fill in the control optimizer. Note that the addition of slack limits is expected to be implemented in an implementation of the virtual method UpdateData.
Variables	std::string name: Name of the virtual fixture boolean Active: Whether or not to add this fixture's data to the tableau boolean ToDelete: Whether or not to delete this fixture from the vector std::vector<osaVFDataPiece> VFDataPieceVector: Vector of "pieces" of data for the virtual fixture vctDynamicVector<double> SlackLimits: Limits of the slack variables used by the pieces vctDynamicVector<double> SlackCosts: Costs of the slacks variables used by the pieces size_t NumSlacks: The number of slacks used by the pieces std::vector<std::string> KinNames: Strings used to find the kinematics objects needed by the pieces std::vector<std::string> SensorNames: Strings used to find the sensor objects needed by the pieces std::vector<nmrKinematics *> kin: Kinematics objects' current values std::vector<nmrSensorValue *> sensors: Sensor objects' current values
Methods	osaVFData(std::string name): Constructor that sets the name of the VF Activate(): Sets the active flag to true Deactivate(): Sets the active flag to false GetObjectiveRows(): Returns the number of objective rows used by the pieces GetIneqConstraintRows(): Returns the number of inequality constraint rows used by the pieces GetEqConstraintRows(): Returns the number of equality constraint rows used by the pieces GetSlacks(): Returns the number of slacks used by the pieces IsActive(): Returns whether or not the VF is currently active GetName(): Returns the VF name AddData(osaVFDataPiece p): Adds a piece of data SetKinNames(std::vector<std::string> n): Sets the names of the kinematics objects to use GetKinNames(): Returns the kinematics names used SetSlacks(int numSlacks, vctDynamicVector<double> & costs, vctDynamicVector<double> & limits): Sets data relating to slacks GetSensorNames(): Returns the sensor names used SetSensorNames(std::vector<std::string> n): Sets the names of the sensor objects to use reserve_space(nmrControlOptimizer & co): Increments indices in the control optimizer Delete(): Sets the VF for deletion ShouldDelete(): Returns if the VF should be deleted initialize(std::map<std::string,nmrKinematics> k, std::map<std::string,nmrSensorValue> s): Initializes the sensor and kinematics objects based on the parameters associated with the known names

	virtual UpdateData(nmrControlOptimizer & co): Gets references to the control optimizer and uses the kinematics and sensor values to fill in the appropriate data

Usage	osaVFData data(“Data”);
	data.AddData(piece1);
	data.AddData(piece2);
	data.SetKinNames(kinNameVector);
	data.SetSensorNames(sensorNameVector);
	data.SetSlacks(1,cost,limit);
	data.initialize(kinVector,sensorVector);

sawControllers/osaVFDataPiece

Summary	This class is used to hold on to a piece of the virtual fixture data to be processed.
Variables	vctDoubleMat ObjectiveMatrix: Objective matrix vctDoubleVec ObjectiveVector: Objective vector vctDoubleMat IneqConstraintMatrix: Inequality constraint matrix vctDoubleVec IneqConstraintVector: Inequality constraint vector vctDoubleMat EqConstraintMatrix: Equality constraint matrix vctDoubleVec EqConstraintVector: Equality constraint vector std::string VariableName: Name of the variable used in the equations int SlackIndex: Index of this piece's slack among others in the collection of data bool HasObjective: If the objective contains data bool HasIneqConstraint: If the inequality constraint contains data bool HasEqConstraint: If the equality constraint contains data
Methods	GetObjectiveRows(): Returns the number of objective rows in this piece GetIneqConstraintRows(): Returns the number of inequality constraint rows in this piece GetEqConstraintRows(): Returns the number of equality constraint rows in this piece SetObjective(vctDynamicMatrix<double> & m, vctDynamicVector<double> & v): Assigns the objective matrix and vector data SetIneqConstraint(vctDynamicMatrix<double> & m, vctDynamicVector<double> & v): Assigns the inequality constraint matrix and vector data SetEqConstraint(vctDynamicMatrix<double> & m, vctDynamicVector<double> & v): Assigns the equality constraint matrix and vector data SetVarName(std::string vn): Sets the name of the variable represented in the objective and constraint equations GetVarName(): Returns the name of the variable represented in the objective and constraint equations SetSlackIndex(int i): Sets the slack index of this piece's slack variables GetSlackIndex(): Returns the slack index GetObjectiveMatrix(): Returns the objective matrix GetObjectiveVector(): Returns the objective vector GetIneqConstraintMatrix(): Returns the inequality constraint matrix GetIneqConstraintVector(): Returns the inequality constraint vector GetEqConstraintMatrix(): Returns the equality constraint matrix GetEqConstraintVector(): Returns the equality constraint vector

Usage	osaVFData piece1;
	osaVFData piece2;
	piece1.SetObjective(objMat, objVec);
	piece2.SetIneqConstraint(ineqMat, ineqVec);
	osaVFData data("Data");
	data.AddData(piece1);
	data.AddData(piece2);