Matlab Interface for the CISST Libraries

Group 16

Team Members: Zachary Zhou

Mentors: Anton Deguet

Background, Aims, Significance:

"The *cisst* package is a collection of libraries designed to ease the development of computer assisted intervention systems. The Surgical Assistant Workstation (SAW) is a platform that combines robotics, stereo vision, and intraoperative imaging (e.g., ultrasound) to enhance a surgeon's capabilities. The SAW package therefore consists of implemented components (e.g., interfaces to many of the devices used for computer-integrated surgery) as well as reusable applications."

Although the *cisst* library is a very powerful tool utilized in many of the medical robots in this lab, the entire library is written in C++. As a result, one requires a sufficient understanding of C++ in order to fully utilize the library. However, not necessarily all of the researchers who would like the use the cisst libraries are proficient in the C++ language.

In order to make the cisst package more accessible, it would be beneficial to port the library onto a different language. For example, MATLAB happens to be a popular language that is fairly easy to learn and utilize.

There are several advantages to utilizing MATLAB over C++. First of all, the MATLAB package includes many numerical methods for matrix manipulation and mathematical interfaces, making it a powerful package for reducing and analyzing data. In addition, MATLAB uses very loose type definitions which, in addition the command console, makes it very user friendly. Finally, researchers often reduce data acquired through the cisst libraries on MATLAB.

Overview of Goals:

Creation of a general purpose dynamic wrapper that will allow the use of the *cisst* packages from MATLAB. A breakdown of the goals are listed below:

- General MATLAB wrapper for the cisst classes
- Utilize CMake to create the plug-in library
- Dynamically load cisst components onto MATLAB
- Handle data manipulation between C++ and MATLAB

Technical Approach:

One approach to porting the cisst libraries would be to hard code each component into MATLAB. For obvious reasons, this approach is not adequate given the size and the changing nature of the cisst package.

Another approach would be to create a code generate which would convert C/C++ functions to MATLAB. However, this approach is very unstable and would potentially fail at multiple locations. In addition, every time the source code is changed on the cisst library, it would need to be generated with every update.

We know that MATLAB is capable of utilizing C/C++ libraries up to some degree due to the MEX capabilities. Thus, we are able to call C functions from MATLAB. The question then how will we know the names of the functions we would like to call form the cisst package.

Fortunately, all classes in the cisst libraries have the ability to identify all the methods in its own class and return the names of these methods in string form. Using the names of these functions, we hope to dynamically create MATLAB classes which will "wrap" the cisst classes.

Using the names of the classes/methods, we should be able to create basic MATLAB objects that simply call their corresponding methods in C++. By taking all these functions names, we should be able to dynamically generate

One issue that we will encounter is that MATLAB and C do not necessarily store their variables in memory the same way. As a result, we will have to code methods which will convert objects from C to MATLAB. In addition, as MATLAB is made for matrix manipulation, we should convert user defined types into matrix form when sending them from C to MATLAB (and vice versa).

We expect to be able to create C objects that may be called one of two ways. In the most basic case, we will simply create general MATLAB wrapper function that will take in string inputs and call the corresponding C method (ex: pos = cisstMatlab.Execute("daVinci", "PSM1", GetPositionCartesian");). However, we would prefer to dynamically create objects in MATLAB using the string names returned by cisst (ex: pos = daVinci.PSM1.GetPositionCartesian();). This will be done in a matter similar to using function pointers in C.

Deliverables:

Minimum:

- Be able to load a single component without configuration file onto MATLAB
- Get dynamic loading to work
- Write basic data conversion methods for native types

Expected:

- Utilize CMake to built MATLAB plug-in library
- Create MATLAB object on the fly with string names
- Populate MATLAB with component interfaces, names, and commands
- Conversion methods for vectors and matrices
- Proper documentation of completed portions

Maximum

- Conversion methods for composite types (cisstDataGenerator)
- Test on multiple machines from MATLAB
- Try running MATLAB wrapper from command-line
- Extensive documentation/readme

Milestones:

- Read/understand cisst library (esp. cisstMultiTask)
    - Expected date: 2/22/12
    - Status: Complete
- Explore MATLAB/C interfaces
    - Expected date: 3/1/12
    - Status: pending
- Call a C method from MATLAB
    - Expected date: 2/28/12
    - Status: complete
- Call a MATLAB method from C
    - Expected Date: 4/1/12
    - Status: pending
- Pass variables between C/MATLAB
    - Expected Date: 4/2/12
    - Status: pending
- Utilize CMake to create library
    - Expected Date: 4/27/12
- Dynamically load component
    - Expected Date: 4/2/12
- Load single component on Matlab
    - Expected Date: 4/2/12
- Data Conversion of Basic Types
    - Expected Date: 3/1/12
- Data Conversion of Hybrid Tasks
    - Expected Date: 3/1/12

Timeline:

Deliverables	20-Feb	1-Mar	9-Mar	16-Mar	23-Mar	2-Apr	6-Apr	13-Apr	20-Apr	27-Apr	4-May	10-May
Read/understand cisst library												
Explore MATLAB/C interfaces												
Call a C method from MATLAB												
Call MATLAB from C												
Pass Variables between C/MATLAB												
Dynamically create cisst objects												
Load single component on MATLAB												
Conversion of Basic Data Types												
Conversion of user defined types (cisstDataGenerator)												
Software Documentation												
Final Report												

In progress
Complete

Management:

- Work on project several hours a day (~30-40 hours a week)
- Maintain regular email contact with Anton
- Meet on a bi-weekly basis to go over progress

References:

- https://trac.lcsr.jhu.edu/cisst
- https://trac.lcsr.jhu.edu/cisst/wiki/cisstMultiTaskTutorial
- http://www.mathworks.com/support/tech-notes/1600/1605.html
- http://www.cmake.org/cmake/resources/resources.html