# CIS II: Paper Presentation Report

By Zachary Zhou
Group 16
Project: CISST to MATLAB Interface

Paper Source:
Vincent Chu, Ghassan Hamarneh "MATLAB-ITK Interface for Medical Image Filtering, Segmentation, and Registration". <www.cs.sfu.ca/~hamarneh/ecopy/medical_showcase2005a.pdf>

## Project Summary:

The cisst package is a collection of libraries designed to ease the development of computer assisted intervention systems. Currently the cisst libraries are available only in the C/C++ programming language, requiring anyone wishing to use the cisst libraries to be proficient in the C language to some degree. However, there are many potential applications for the cisst library which doesn't require a deep understanding of the C language. As a result, it is profitable to wrap the cisst package in a more easily understood language.
Recently, MATLAB has become a very popular language for scientific researchers and biomedical engineers. Intrinsically, it much easier to learn and understand MATLAB than C. There is no need to use clearly defined types in MATLAB, no need for separate header/source files, no need to create makefiles and compile, etc.. In addition, MATLAB offers a command terminal which allows the user to test code without having to stop and compile.
In addition, many methods of analyzing data collected by the cisst librarys have been coded in MATLAB. A MATLAB wrapper for the cisst libraries will allow for the user to direcly analyze collected data instead of having to send it to MATLAB via File IO.
As a result, MATLAB is one of the best languages to wrap the cisst libraries in.

## Background

MATLAB, short for MATrix LABoratory, is an interpretative language and environment which is optimized for matrix manipulation. As a result, MATLAB is commonly used for matrix computations, numerical analysis, and graphing.
The ITX (Insight ToolKit) library, which is used in this paper is a open-source toolkit written in the C++ environment. The toolkit contains various filtering, segmentation and registration algorithms designed for medical image analysis. ITX provides various algorithms and methods which are not commonly available in MATLAB and utilizes the superior speeds of compiled C methods.
As an interpretative language, MATLAB allows for C libraries to be directly loaded into the MATLAB workspace. There are two ways in which this can be accomplished. Firstly, one may recompile C source files/libraries into MEX files which may be called in a similar manner as standard MATLAB function. In addition, C library files can be loaded directly onto the MATLAB workspace and functions on the library can be called by MATLAB indirectly through the calllibaray() method.

**Paper Purpose and Relevance**

The purpose of this paper was to present the design and usage of a MATLAB wrapper for the ITK library. This paper is of strong relevance to my chosen project as it documents the creation of a MATLAB wrapper for a C library. Similarly, I am trying to create a MATLAB wrapper for the cisst libraries which is also coded in C. The paper covers both and overview of their design and implementation of their wrapper. In addition, it includes documentation and examples of usage of the MATLAB-ITK wrapper in the MATLAB workspace. From this paper, I may be able to drawn some design ideas for the implementation of the cisst-MATLAB wrapper.

**Design, Implementation, and Results**

The following presents an overview of the design of the MATLAB-ITK wrapper, its implementation, and results as they are presented in the paper.

Objective:

The stated objective of this paper is design a wrapper which will allow a greater number of researchers access to the ITK libraries. In addition, allow for data from ITK to be reduced on MATLAB without file IO. This is expected to result in a speed up several orders in magnitude.

MATLAB Setup for Building MEX:

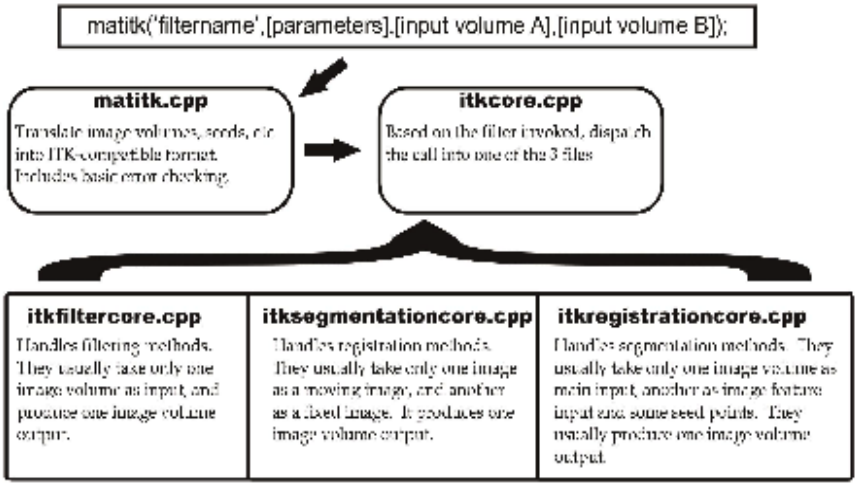


**Figure 1.** MATITK Execution Flowchart.

Utilizes CMake to compile standard ITK files into MEX files. Automatically generates mexopt.bat file which contains header and library paths for ITK files.

Architecture:

Main wrapper file is matitk.cpp, which contains the mexfunction. As a result, matitk() is the only function which will be called from the MATLAB console. Matitk will then parce the input string values and pass results to itkcore.cpp, which performs further reduction of input values.
Itkcore will then invoke one of 3 procedures depending on command invoked.

Filtering

```
void segmentationGeodesicActiveContourLevelSet(){
    const char* PARAM[]={"propagationScaling", .../*some more parameters*/};
    const char* SUGGESTVALUE[]={"","1.0","1.0","0.02","800"};
    const int nParam = sizeof(PARAM)/sizeof(*PARAM);
    ParameterContainer paramIterator(PARAM,SUGGESTVALUE,nParam);
    if (emptyImportFilter[IMPORTFILTER8]){mexErrMsgTxt("...")}
    mexPrintf("\nThis method requires two image volumes....\n");
    ////////////////////Begin Core Filter Code//////////////////////
    double propagationScaling=paramIterator.getCurrentParam(0);
    //... edited for brevity.  The other 4 parameters can be accessed in a similar fashion
    typedef itk::GeodesicActiveContourLevelSetImageFilter<InputImageType,OutputImageType>...
    GeodesicActiveContourFilterType::Pointer filter = GeodesicActiveContourFilterType::New();
    filter->SetPropagationScaling( propagationScaling );
    //... edited for brevity.  The other 4 parameters are set in a similar fashion as the line above.
    filter->SetInput(importFilter[IMPORTFILTER8]->GetOutput());
    filter->SetFeatureImage( importFilter[IMPORTFILTERA]->GetOutput());
    filter->Update();
    //...omitted code for setting up and connecting additional filters ..
    pixelContainer = thresholder->GetOutput()->GetPixelContainer();
    ////////////////////End Core Filter Code//////////////////////////
}
```

Sample pseudocode:
First lines defines parameters required by the filter, following lines are relevant code. There is no return type for filters. They access variables stored in itkcore.cpp which are then passed to MATLAB via matitk. Filtering code mush start with an input with the letter 'f'.

Segmentation:

```
void segmentationGeodesicActiveContourLevelSet(){
    const char* PARAM[]={"propagationScaling", .../*some more parameters*/};
    const char* SUGGESTVALUE[]={"","1.0","1.0","0.02","800"};
    const int nParam = sizeof(PARAM)/sizeof(*PARAM);
    ParameterContainer paramIterator(PARAM,SUGGESTVALUE,nParam);
    if (emptyImportFilter[IMPORTFILTER8]){mexErrMsgTxt("...")}
    mexPrintf("\nThis method requires two image volumes....\n");
    ////////////////////Begin Core Filter Code//////////////////////
    double propagationScaling=paramIterator.getCurrentParam(0);
    //... edited for brevity.  The other 4 parameters can be accessed in a similar fashion
    typedef itk::GeodesicActiveContourLevelSetImageFilter<InputImageType,OutputImageType>...
    GeodesicActiveContourFilterType::Pointer filter = GeodesicActiveContourFilterType::New();
    filter->SetPropagationScaling( propagationScaling );
    //... edited for brevity.  The other 4 parameters are set in a similar fashion as the line above.
    filter->SetInput(importFilter[IMPORTFILTER8]->GetOutput());
    filter->SetFeatureImage( importFilter[IMPORTFILTERA]->GetOutput());
    filter->Update();
    //...omitted code for setting up and connecting additional filters ..
    pixelContainer = thresholder->GetOutput()->GetPixelContainer();
    ////////////////////End Core Filter Code//////////////////////////
}
```

Similar to filtering, first lines define parameters, following lines are relevant code. As in case above, there is no return type as the method modifies variables stored in itkcore. Segmentation calls must start with the letter 's'.

Registration:
Procedure for registration is similar to that of filtering and segmentation. Registration calls much begin with the letter 'r'.

Automated Generation of Filtering Script:
Uses perl script, matitkcode.pl to generate c source files that contain auto-generated mex compatible filter methods. There are potential bugs as it merely follows the pseudocode of the filtering methods listed above.

Ultilizing the wrapper:
MEX source files and libraries is compiled via CMake into the library matitk.dll. When this library is placed in MATLAB's working directory, MATLAB will be automatically be able to call the MEX function of matitk.
MATLAB function calls to ITK library are as follows:
matitk(operationName,[parameters],[inputArray1],[inputArray2],[seed(s)Array],[Image(s)Spacing]);

**Conclusions**

As the end result, the authors were able to generate a MEX library with the abilities to call the following ITK methods.

The wrapper contains full compatibility between C/MATLAB for the ITK library and includes parsing of basic data types. In addition, the authors perovide a PERL script which can be potentially used to add additional filtering methods into the library.

| Opcode | Corresponding filter name |
|---|---|
| FGA | filterGaussian |
| FCA | filterCurvatureAnsio |
| FCF | filterCurvatureFlow |
| FMMCF | filterMinMaxCurvatureFlow |
| FGM | filterGradientMagnitude |
| FGMS | filterGradientMagnitudeWithSmoothing |
| FSN | filterSigmoidNonlinearMapping |
| FBD | filterDilate |
| FBE | filterErode |
| FDM | filterDanielssonDistanceMapImageFilter |
| FDMV | filterDanielssonDistanceMapImageFilterGetVoronoiMap |
| FBL | filterBilateral |
| FBT | BinaryThresholdImageFilter |
| FBB | BinomialBlurImageFilter |
| FD | DerivativeImageFilter |
| FDG | DiscreteGaussianImageFilter |
| FF | FlipImageFilter |
| FGAD | GradientAnisotropicDiffusionImageFilter |
| FGMRG | GradientMagnitudeRecursiveGaussianImageFilter |
| FLS | LaplacianRecursiveGaussianImageFilter |
| FMEANF | MeanImageFilter |
| FMEDIANF | MedianImageFilter |
| SCC | segmentationConfidenceConnected |
| SIC | segmentationIsolatedConnected |
| SNC | segmentationNeighbourhoodConnected |
| SCT | segmentationConnectedThreshold |
| SFM | segmentationFastMarch |
| SOT | segmentationOtsuThreshold |
| SGAC | segmentationGeodesicActiveContourLevelSet |
| SLLS | segmentationLaplacianLevelSetLevelSet |
| RTPS | registerThinPlateSpline |
| RD | registerDemon |

**Table 1.** MATITK available opcodes and the corresponding opnames.

**Personal Assessment**

I believe that this paper provides a good example of how to wrap an existing C library to MATLAB. It provides details how to overcome the single entry point weakness of MEX files. In addition, the paper includes some analysis of why they chose to filter all function calls through one method instead of adding multiple MEX functions (one per ITK method). Includes a way to add new filtering methods into MATITK library.

However, I believe there are several weaknesses to this paper. Primarily, one of the stated goals is to reduce run time of MATLAB analysis by skipping file IO. The authors claim that utilizing the wrapper will result in orders of magnitude speed-up. However, calling C functions from MATLAB are inherently slower than calling them from a compiled program. The authors do not even attempt to analysis potential speed-ups gained via the wrapper.

In addition, I feel that their use of the one MEX function results in very awkward function calls in MATLAB. The result of their design is a lot of String manipulation. Also, the function call itself seems very bulky on the MATLAB side and requires careful analysis in order to even call a simple method from the ITK library. I believe this somewhat mitigates the gain in simplicity going from C to MATLAB.

I believe that the PERL script is a weakness. Automatic generation of code is useful, but because it is simply a code generator, there is potentially many errors that can occur when generating the code. The generator seems inflexible and prone to failure.

Also, I will not be able to implement this design for the cisst libraries. One big difference between the cisst libraries and the ITK library is that the cisst library utilizes object oriented design. As a result, it it fairly unlikely that a successful wrapper can be created with the single method of entry necessitated by the MEX architecture.

However, on the whole this paper was very relevant to cisst-MATLAB interface project. Both involve a wrapper for a C library in MATLAB. In addition, the stated goals of both wrappers are virtually the same. Even though I will not be able to use much of the design in my project, reading and understanding the design choices made by the authors was really interesting.