Introduction to Matlab Engine - Getting the best out of C++ and Matlab.

Compiled by Jai Pillai (jsp AT umiacs DOT umd DOT edu)

Introduction

“Matlab or C++, which one to use for programming? ”, is a question which haunts most graduate student (and sometimes even senior researchers) in computer vision. Both there development platforms have their own merits and demerits. Coding in C++ decreases the run time significantly and lets one use powerful libraries like OpenCV, STL and Boost. But debugging and visualizing data is difficult, which increases the code development time. Matlab on the other hand provided good visualization tools like plot, mesh and surf. But it is inefficient in executing loops. Again the run time of code increases significantly with Matlab, leading to the common excuse in research “Our code is currently implemented in Matlab and is slow, but can be improved by a C++ implementation”.

In this tutorial, we look at a way of utilizing the best of both worlds - the speed of C++ codes and the visualizing capabilities of Matlab. How is that possible? The trick we employ is to do the coding in C++ and write modules for transferring data from C++ to an adjacent Matlab environment for visualization and debugging. Once the code is developed, this transfer module can be commented out or removed to get the fully working code in C++. This will reduce the code development time and the code run time. This is made possible by the Matlab Engine. In this tutorial, we provide a step by step procedure for using the Matlab Engine. Many of the details below are not the author’s and have merely been collected from internet and put in a common place. Sample functions implementing the data transfer can be downloaded from authors website.

At first, we look at how to implement the code engdemo.cpp, which is the demo program provided by Matlab. Since getting this code to work is tricky and took significant time for the author, the various steps are enumerated below both in Windows and UNIX.

Using Matlab Engine with Visual C++

1. Open Microsoft Visual Studio.
2. Create a New Project
    a. Select File->New->Project.
    b. Select Visual C++ -> General -> Empty Project
    c. Enter the name of the project and its location in the indicated text areas and click OK.
3. Add a source file to the project.
    a. Right-click on the SOURCE FILES folder in the SOLUTION EXPLORER and click Add -> "New Item...". Choose "C++ file (.cpp)", enter the name of the file as enginedemo.cpp. Click OK.
    b. Copy the code from the source file: engwindemo.c, and paste it into this file enginedemo.cpp. The file engwindemo.c may be obtained from the following location: - $MATLABROOT/extern/examples/eng_mat where $MATLABROOT is the MATLAB root directory, and may be determined by entering the command: matlabroot at the MATLAB command prompt.

4. Modify the project properties
    a. Right click on the project name and select PROPERTIES in the solution explorer to open the project properties. Once this window has opened, make the following changes:
    b. Under C/C++ General, add the directory $MATLABROOT\extern\include to the field ADDITIONAL INCLUDE DIRECTORIES.
    c. Under C/C++ Precompiled Headers, select "Not Using Precompiled Headers".
    d. Under Linker General, add the directory MATLABROOT\extern\lib\win32\microsoft to the field: ADDITIONAL LIBRARY DIRECTORIES.
    e. Under Linker Input, add the following names: libmx.lib, libmat.lib,libeng.lib to the field marked ADDITIONAL DEPENDENCIES.
5. Registering Matlab as a COM server on your system. Entering the following commands in a DOS command window:
    a. cd *matlabroot*\bin\win32
    b. matlab /regserver
    c. Close the MATLAB window that appears.
6. Now you are all set. Build and run the project in Visual C++ and hopefully, it will open Matlab and perform the required data transfer and plotting, implemented in enginedemo.cpp.

Using Matlab Engine with Ubuntu, using Eclipse IDE.

1. Install C shell csh in your system at /bin/csh.
2. Include the required paths below to the PATH variable
    a. Location of matlab - $matlabroot/bin and
    b. $matlabroot/sys/os using
       setenv PATH $matlabroot/bin:$matlabroot/sys/os:$PATH
3. Navigate the terminal to the location of the demo file engdemo.cpp. (using cd command). This file can be found in $matlabroot/extern/examples/eng_mat/ engdemo.cpp
4. Call the g++ compiler with the required include files and libraries as follows
```
        g++ engdemo.cpp -o engdemo.o -I $matlabroot/extern/include -L $matlabroot/
bin/glnax64 -leng -lmat -lmex -lut
```
5. Now the object file will be created. Run the object file as follows

./engdemo.o

How to transfer data from C++ to Matlab and vice versa?

In this section,we look into the steps to transfer data from C++ to Matlab and back. You should include the header file engine.h for calling the required functions. I have implemented a function **sendMatlab() to transfer OpenCV Mat files to Matlab as matrices.** It can be downloaded from **http://www.umiacs.umd.edu/~jsp/Downloads/MatlabEngine.rar**

The three main steps in transfering data from C++ to Matlab are

a. Opening Matlab using Matlab Engine.
b. Converting the data into mxArray format.
c. Transfering the data to Matlab using the engPutVariable function.

Opening Matlab - This is done by defining a pointer to Engine and calling the engOpen function as shown below :-

```
Engine *Eg;
        Eg = engOpen("\0");
```

Converting data into mxArray format :- Here we first have to allocate memory for an mxArray pointer and transfer data into the allocated memory. mxArray is the main MATLAB structure used for holding data in MEX-Files. It can hold real data, complex data, arrays, matrices, sparse-arrays, strings, and other MATLAB data-structures.

```
        mxArray *Temp;
        Temp = mxCreateDoubleMatrix(1,3,mxREAL);
        double data[3] = {1.0,2.0,3.0};
        memcpy((void *)mxGetPtr(Temp),(void *)data,sizeof(data));
```

Transfering data to Matlab - Use the engPutVariable function. Its second argument is the variable name and the third one is the pointer to mxArray storing the data.

```
        engPutVariable(eng,”VariableName”,Temp);
```

Additional documentation of the various supported functions can be found at
http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_external/f29148.html
I will try to update this document periodically as I spot more bugs and techniques. Feel free to email me (jsp AT umiacs DOT umd DOT edu) with your comments, suggestions and bugs :).