**Controlling Optotrak from Matlab**

Updated in August of 2015 for Visual Studio 2015 and no ODAU.

It is possible to control the Optotrak (3020 or Certus) without the Optotrak software by using otrak application programming interface (API). Unfortunately, Matlab cannot call the API directly. Matlab can call MEX functions (C/C++) which can then call the Optotrak API. Below I will explain the steps required to control Optotrak from within Matlab.

* Install an Optotrak PCI card and connect the card to the Optotrak system control unit (SCU)
* Purchase this card and cable from NDI
* Install Microsoft Visual Studio 2015
* Free download from Microsoft
* Visual C++ is no longer included in Visual Studio. To avoid strange error message, you need to download the C++ compenents for Visual Studio. The following directions are, perhaps, the easiest way to do this.
* Navigate to Control Panel > Programs and Features
* Select Microsoft Visual Studio Community 2015 and click Change and a Visual Studio window will open
* Click Modify
* Under Programming Languages check off Visual C++ , whcih will also select the subitems
* Click UPDATE and the need C++ components will install
* Install Optroak API
* Purchase from NDI
* Use the default installation paths (C:\NDIoapi, C:\ndigital)
* Install Matlab
* You may be entitled to a student discount if you buy it from your University store
* Create a MEX project in Visual C++
* Open Visual Studio 2015
* Navigate to File > New > Project…
* Select Installed > Templates > Visual C++ > Win32 > Win32 Project
* Name the project ‘Matlab\_to\_Optotrak’
* Click OK and the Win32 Application Wizard will appear
* Click Next > and then select DLL and check Empty project
* Click Finish
* Navigate to Project > Add New Item…
* Select Header File (.h)
* Name the file ‘Matlab\_to\_Optotrak.def’
* Click Add
* Enter the following in the new file:

LIBRARY “Optotrak\_to\_Matlab

EXPORTS mexFunction

This definition file (\*.def) tells the compiler and linker that mexFunction should be accessible from outside. In other words, Matlab will be able to call mexFunction.

* Navigate to Project > Add New Item…
* Select C++ File (.cpp)
* Name the file ‘mexFunction.cpp’
* Click Add
* Navigate to Project > Add Existing Item…
* Navigate to the Optotrak API msvc directory with \*.lib files
* For me, it is C:\NDIoapi\ndlib\msvc
* Select both NDItb.lib and oapi.lib and click Add
* Navigate to File > Save All
* Click on Matlab \_to\_Optotrak in the Solution Explorer
* The Solution Explorer should be the left pane of Visual C++
* Selecting Matlab\_to\_Optotrak will ensure the properties we need to change are available (the next steps)
* Navigate to Project > Matlab\_to\_Optotrak Properties…
* Select Configuration Properties > Debugging > Environment
* Click the column beside Environment, click the down arrow, and then <Edit...>
* Enter 'PATH=%PATH%;C:\NDIoapi\dll\'
* Select C/C++ > General > Additional Include Directories
* Click the column beside Additional Include Directories, click the down arrow, and then <Edit…>
* Click on the New Folder icon and a new line will appear
* Click on the … of the new line
* Navigate to the Matlab include directory
* For me, it is C:\Program Files\MATLAB\R2013b\extern\include
* Click Select Folder
* Click on the New Folder icon and a new line will appear
* Click on the … of the new line
* Navigate to the Optotrak realtime directory
* For me, it is C:\ndigital\realtime
* Click Select Folder
* Click on the New Folder icon and a new line will appear
* Click on the … of the new line
* Navigate to the Optotrak API include directory with \*.h files
* For me, it is C:\NDIoapi\ndlib\include
* Click Select Folder and then OK again to be back at the Properties window
* Select C/C++ > Preprocessor > Preprocessor Definitions
* Click the down arrow beside Preprocessor definitions and select <Edit…>
* Add MATLAB\_MEX\_FILE to the bottom of the list and click OK
* Select Linker > General > Output File
* Click the down arrow and then <Edit...>
* Change the Output File to $(OutDir)\Matlab\_to\_Optotrak.mexw32
* Click
* Select Linker > General > Additional Library Directories
* Click the down arrow and select <Edit…>
* Click on the New Folder icon and a new line will appear
* Click on the … of the new line
* Navigate to the Matlab windows directory
* For me, it is C:\Program Files\MATLAB\R2013b\extern\lib\win32\microsoft
* Click Select Folder and then OK to be back at the Properties window
* Select Linker > Input > Additional Dependencies
* Click the down arrow and select <Edit…>
* Add ‘libmat.lib’, ‘libmex.lib’, and ‘libmx.lib’ on new lines and click OK
* Select Linker > Input > Module Definition File
* Click the down arrow and select <Edit…>
* Enter ‘Matlab\_to\_Optotrak.def’ and click OK
* Click OK to close the Properties window
* Navigate to File > Save All

These complicated steps ensure the MEX function can be compiled and linked by Visual C++ and then called from within Matlab. The main output will be Matlab\_to\_Optotrak.mexw32, which will soon have the function mexFunction that can be called by Matlab. If you get lost in the process, there are some pictures in the tutorial Using C/C++ under Matlab (Part 2) (<http://comisef.wikidot.com/tutorial:using-c-c-under-matlab-part-2> ). Another tutorial (Walkthrough: Creating and Using a Dynamic Link Library [C++], <http://msdn.microsoft.com/en-us/library/ms235636.aspx> ) explains the steps required for the MEX function to call the Optotrak API.

* Add the following text to mexFunction.cpp and then Save All

#include "mex.h"

void mexFunction(int nlhs, mxArray \*plhs[], int nrhs, mxArray \*prhs[]) {

return;

}

* Select Build > Build Solution
* In the Output at the bottom, it should say Build: 1 succeeded…
* You can ignore the warnings about non-ascii characters and the output filename
* If not, read the error statements (which are cryptic) to correct the problem
* Test the mexFunction in Matlab
* Open Matlab
* Change the Matlab path to the location of Matlab\_to\_Optotrak.mexw32 file
* For me, it is C:\Users\Administrator2\My Documents\Visual Studio 2015\Projects\Matlab\_to\_Optotrak\Debug
* Create a new \*.m file and then save it as hello\_optotrak.m in the current path
* Add in this one command, Matlab\_to\_Optotrak() , and run the file

Although no output was produced, Matlab just successfully called the mexFunction (which doesn’t do anything yet) and then returned control to Matlab. Next, we will have the mexFunction initialize the Optotrak and return a message of success or failure to Matlab.

* Close Matlab
* You cannot build the mexFunction in C++ while Matlab is accessing Matlab\_to\_Optotrak.mexw32
* Instead of closing Matlab, you can run ‘clear mex’ in Matlab to release the Matlab\_to\_Optotrak.mexw32 file
* Go back to mexFunction.cpp and update it as follows

#include "mex.h"

#include "ndtypes.h"

#include "ndpack.h"

#include "ndopto.h"

void mexFunction(int nlhs, mxArray \*plhs[], int nrhs, mxArray \*prhs[]) {

int

return\_code;

// This will cause the Optotrak SCU to beep twice.

return\_code = TransputerLoadSystem("system");

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

if (return\_code != 0) {

plhs[1] = mxCreateString("TransputerLoadSystem: Failed");

TransputerShutdownSystem();

return;

}

plhs[1] = mxCreateString("TransputerLoadSystem: Success");

TransputerShutdownSystem();

return;

}

* Navigate to Build > Build Solution

When mexFunction is now called from Matlab, it will attempt to load the Optotrak. The return code and a message will be return by mexFunction to Matlab. If you would like more information on how to code MEX files, I recommend Writing MATLAB C/MEX Code ( <http://www.mathworks.com/matlabcentral/fileexchange/27151-writing-matlab-cmex-code> ). For more information on the Optrotrak API, refer to the manual Optotrak Application Porgrammer’s Interface Guide.

Before rerunning hello\_optotrak.m in Matlab, ensure the Optotrak camera and SCU are on and the computer you are using is connected to the SCU. Make sure everything is turned off if you need to rewire the Optotrak. Open Matlab, change the path to the location of the hello\_optotrak.m and run hello\_optotrak.m

[ return\_code message ] = Matlab\_to\_Optotrak()

If everything worked, then you should have heard two beeps from the SCU and then received a return code of 0 and the message TransputerLoadSystem: Success. Congratulations, you just controlled the Optotrak from Matlab!

If that didn’t work, then here are a few troubleshooting tips. If the Optotrak SCU didn’t beep at all, then there is likely a communication issue from your computer to the SCU. The most likely problem is the cable between the two is not connected properly. A return code of 1000 and a Failed message indicates the computer is talking to the Optotrak but something went wrong. Check that C:\ndigital\realtime is one of the Additional Include Directories in Visual C++ so the Optotrak can find the system.nif file. It is also possible that the system.nif file is out of date and is trying to configure the Optotrak in an impossible way, for example, looking for the ODAU on the wrong port. To update the system.nif file, run optset32.exe under C:\ndigital\programs . If you still have problems, you can compare the system.nif file on the Matlab computer to the system.nif file on the computer you normally use to control the Optotrak. Errors this early usually indicate that the hardware isn’t connected properly.

Unfortunately, problems at this step are hard to diagnose because Optotrak doesn’t yet create an error log. Messages and errors for later API function calls will be added to opto.err in the same directory as hello\_optotrak.m. This blog post may also help, <https://motorbehaviour.wordpress.com/2015/02/24/controlling-optotrak-from-matlab-ii/>

The next steps involve growing the mexFunction to allow for all sorts of calls to the Optotrak API from Matlab. One option is to pass the name of the API function (with any parameters) from Matlab to the mexFunction. The mexFunction will simply identify which API function you want to call, pass along the parameters, and return a message of success or failure to Matlab. Then we can build a program in Matlab that will initiailize the Optotrak, collect data, convert data, and display the results.

This code is available in the attached mexFunction.cpp and collect\_a\_trial.m. You will also need open\_ndi\_bin\_file.m to run collect\_a\_trail.m. Copy and paste mexFunction.cpp into C++ and build it. Now copy collect\_a\_trail.m into the same folder as hello\_optotrak.m, open matlab, change the path to the location of collect\_a\_trial.m, and run it. I’ve also copied and pasted the code for the three files below.

**collect\_a\_trial.m**

% Written by Jarrod Blinch, November 13th, 2010

% Available from motorbehaviour.wordpress.com

clear all;

collection\_num\_markers = 3; % optotrak

collection\_num\_channels = 4; % odau

collection\_frequency = 500; % Hz

collection\_duration = 2; % s

%cam\_filename = 'Registered20150807'; % used when there are multiple sensors

[a b] = Matlab\_to\_Optotrak('TransputerLoadSystem',

%[a b] = Matlab\_to\_Optotrak('TransputerLoadSystem', collection\_num\_markers, collection\_frequency, collection\_duration, cam\_filename); % used when there are multiple sensors

collection\_num\_markers, collection\_num\_channels, collection\_frequency, collection\_duration); %#ok<NASGU>

if (a ~= 0)

error('TransputerLoadSystem died!');

end

[a b] = Matlab\_to\_Optotrak('OptotrakActivateMarkers'); %#ok<NASGU>

if (a ~= 0)

error('OptotrakActivateMarkers died!');

end

[a b] = Matlab\_to\_Optotrak('DataBufferInitializeFile', '', '001.P01');

% Array b will contain the current IRAD locations.

[a b] = Matlab\_to\_Optotrak('DataGetLatest3D');

display('IRED locations:');

display(b(4:end));

% Array d will contain the current ODAU input.

[c d] = Matlab\_to\_Optotrak('DataGetLatestOdauRaw');

display('ODAU signals:');

display(d(4:end));

[a b] = Matlab\_to\_Optotrak('DataBufferStart');

data\_buffer\_done = false;

while (data\_buffer\_done == false)

[a b] = Matlab\_to\_Optotrak('DataBufferWriteData');

if (a ~= 0)

display('Warning: DataBufferWriteData died!');

end

if (b == 1)

data\_buffer\_done = true;

end

% Can be used to peek at the data during collection.

[a b] = Matlab\_to\_Optotrak('DataGetLatest3D');

[c d] = Matlab\_to\_Optotrak('DataGetLatestOdauRaw');

end

% Convert the R and O files to C and V.

[a b] = Matlab\_to\_Optotrak('FileConvert', '', '001.P01');

optotrak\_array = open\_ndi\_bin\_file('C#001.P01');

odau\_array = open\_ndi\_bin\_file('V1#001.P01');

figure;

hold on;

title('IRED locations');

plot3(optotrak\_array(:,1), optotrak\_array(:,2), optotrak\_array(:,3), 'r');

plot3(optotrak\_array(:,4), optotrak\_array(:,5), optotrak\_array(:,6), 'g');

plot3(optotrak\_array(:,7), optotrak\_array(:,8), optotrak\_array(:,9), 'b');

figure;

hold on;

title('ODAU signals');

plot(odau\_array);

[a b] = Matlab\_to\_Optotrak('OptotrakDeActivateMarkers'); %#ok<NASGU>

if (a ~= 0)

display('Warning: OptotrakDeActivateMarkers died!');

end

[a b] = Matlab\_to\_Optotrak('TransputerShutdownSystem');

if (a ~= 0)

display('Warning: TransputerShutdownSystem died!');

end

% Release the Matlab\_to\_Optotrak.mexw32 file, which allows it

% to be compiled in Visual C++ if needed.

clear mex;

**open\_ndi\_bin\_file.m**

% [ndi\_array] = open\_ndi\_bin\_file(filename)

%

% Written by Jarrod Blinch, November 6th, 2010

% Available from motorbehaviour.wordpress.com

function [ndi\_array] = open\_ndi\_bin\_file(filename)

fid = fopen(filename, 'r');

fread(fid, 1, 'char'); % 32

item\_total = fread(fid, 1, 'short'); % items per frame

subitem\_total = fread(fid, 1, 'short'); % subitems per frame

column\_total = item\_total \* subitem\_total;

frame\_total = fread(fid, 1, 'int'); % number of frames

fread(fid, 1, 'float'); % collection frame frequency

fread(fid, 60, 'char=>char'); % user comments

fread(fid, 60, 'char=>char'); % system comments

fread(fid, 30, 'char=>char'); % file description

fread(fid, 1, 'short'); % cutoff filter frequency

fread(fid, 8, 'char=>char'); % time of collection

fread(fid, 1, 'short'); % unused?

fread(fid, 8, 'char=>char'); % date of collection

fread(fid, 73, 'char'); % extended headed and unused

ndi\_array = ones(frame\_total,column\_total) .\* NaN;

for frame\_num = 1:frame\_total

for column\_num = 1:column\_total

data = fread(fid, 1, 'float');

if (data < -100000) % technically, it is EE EE EE EE or -3.697314e+28

data = NaN;

end

ndi\_array(frame\_num,column\_num) = data;

end

end

fclose(fid);

end

**mexFunction.cpp [with an Optotrak Data Acquisition Unit (ODAU)]**

// Written by Jarrod Blinch, November 13th, 2010

// Available from motorbehaviour.wordpress.com

#include "mex.h"

#include <matrix.h>

#include <string.h>

#include <windows.h>

#include <math.h>

#include "ndtypes.h"

#include "ndpack.h"

#include "ndopto.h"

void mexFunction(int nlhs, mxArray\* plhs[], int nrhs, mxArray\* prhs[]) {

int

return\_code;

unsigned int

uRealtimeDataReady = 0,

uSpoolComplete = 0,

uSpoolStatus = 0,

uFrameNumber,

uElements,

uFlags,

ui,

uj;

static int

puRawData[3+1]; // number of ODAU channels + 1

static Position3d

p3dData[1];

char

StrBuffer[65],

StrBuffer2[65],

StrBuffer3[65],

filename\_input[68],

filename\_output[68];

//szNDErrorString[MAX\_ERROR\_STRING\_LENGTH + 1]; // 2047 + 1 (ndopto.h)

double

\*y,

dk,

dl;

// Make sure there is only one argument.

if (nrhs < 1) {

mexErrMsgTxt("At least one string argument must be passed!\n");

}

// Make sure the argument is a string.

if (!mxIsChar(prhs[0])) {

mexErrMsgTxt("The first argument should be a string!\n");

}

// Read the string into StrBuffer.

if (mxGetString(prhs[0], StrBuffer, sizeof(StrBuffer)-1)) {

mexErrMsgTxt("Unable to read the argument string!\n");

}

if (strcmp(StrBuffer, "TransputerDetermineSystemCfg") == 0) {

// This will beep twice.

return\_code = TransputerLoadSystem( "system" );

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("TransputerDetermineSystemCfg");

} else if (strcmp(StrBuffer, "TransputerLoadSystem") == 0) {

// Ensure we have a second and third integer arguments (optotrak markers, odau channels).

if (nrhs != 5) {

mexErrMsgTxt("Four additional arguments (#markers, #channels, hz, time) must be passed with TransputerLoadSystem!\n");

}

// Make sure the second and third arguments are unsigned integers.

if (!mxIsDouble(prhs[1])) {

mexErrMsgTxt("The second argument (num oprotrak markers) should be an unsigned integer!\n");

}

if (!mxIsDouble(prhs[2])) {

mexErrMsgTxt("The third argument (num odau channels) should be an unsigned integer!\n");

}

if (!mxIsDouble(prhs[3])) {

mexErrMsgTxt("The fourth argument (colection frequency) should be a double!\n");

}

if (!mxIsDouble(prhs[4])) {

mexErrMsgTxt("The fifth argument (collection duration) should be a double!\n");

}

// Read the second and third arguments into unsigned integers.

ui = (unsigned int)mxGetScalar(prhs[1]); // number of Optotrak markers

uj = (unsigned int)mxGetScalar(prhs[2]); // number of ODAU1 channels

dk = mxGetScalar(prhs[3]); // collection frequency

dl = mxGetScalar(prhs[4]); // collection duration (s)

// This will beep twice.

return\_code = TransputerLoadSystem( "system" );

if (return\_code != 0) {

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("TransputerLoadSystem");

TransputerShutdownSystem();

return;

}

return\_code = TransputerInitializeSystem(OPTO\_LOG\_ERRORS\_FLAG | OPTO\_LOG\_MESSAGES\_FLAG);

if (return\_code != 0) {

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("TransputerInitializeSystem");

TransputerShutdownSystem();

return;

}

return\_code = OptotrakLoadCameraParameters("standard");

if (return\_code != 0) {

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("OptotrakLoadCameraParameters");

TransputerShutdownSystem();

return;

}

return\_code = OdauSetupCollection(

ODAU1, // nOdauId ODAU1

uj, // nChannels (1-256)

1, // nGain

ODAU\_DIGITAL\_PORT\_OFF, // nDigitalMode ODAU\_DIGITAL\_PORT\_OFF

(float)dk, // fFrameFrequency (1-100000)

(float)50000.0, // fScanFrequency (1-100000)

1, // nStreamData

(float)dl, // fCollectionTime in seconds

(float)0.0, // fPreTriggerTime, not supported and must be 0

0); // uFlags

if (return\_code != 0) {

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("OdauSetupCollection");

TransputerShutdownSystem();

return;

}

return\_code = OptotrakSetStroberPortTable(ui, 0, 0, 0); // Number of markers connected to each port

if (return\_code != 0) {

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("OptotrakSetStroberPortTable");

TransputerShutdownSystem();

return;

}

return\_code = OptotrakSetupCollection(

ui, // Number of markers in the collection

(float)dk, // Frequency to collect data frames at

(float)2500.0, // Marker frequency for marker maximum on-time

30, // Dynamic or Static Threshold value to use

160, // Minimum gain code amplification to use

1, // Stream mode for the data buffers

(float)0.35, // Marker Duty Cycle to use

(float)7.0, // Voltage to use when turning on markers

(float)dl, // Number of seconds of data to collect

(float)0.0, // Number of seconds to pre-trigger data by, not suppoted and must be 0

OPTOTRAK\_NO\_FIRE\_MARKERS\_FLAG | OPTOTRAK\_BUFFER\_RAW\_FLAG); // OPTOTRAK\_GET\_NEXT\_FRAME\_FLAG often used with realtime data

if (return\_code != 0) {

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("OptotrakSetupCollection");

TransputerShutdownSystem();

return;

}

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("OptotrakSetupCollection");

} else if (strcmp(StrBuffer, "TransputerShutdownSystem") == 0) {

return\_code = TransputerShutdownSystem();

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("TransputerShutdownSystem");

} else if (strcmp(StrBuffer, "OptotrakActivateMarkers") == 0) {

return\_code = OptotrakActivateMarkers();

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("OptotrakActivateMarkers");

} else if (strcmp(StrBuffer, "OptotrakDeActivateMarkers") == 0) {

return\_code = OptotrakDeActivateMarkers();

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("OptotrakDeActivateMarkers");

} else if (strcmp(StrBuffer, "DataBufferInitializeFile") == 0) {

// Second argument example: ../data/raw/

// Third argument example: 001.P1

// The program will append R# or O1#

// Ensure we have a second and third string argument with the filename (path, part of filename).

if (nrhs != 3) {

mexErrMsgTxt("A second and third string arguments must also be passed with DataBufferInitializeFile!\n");

}

// Make sure the second and third arguments are strings.

if (!mxIsChar(prhs[1])) {

mexErrMsgTxt("The second filepath argument should be a string!\n");

}

if (!mxIsChar(prhs[2])) {

mexErrMsgTxt("The third part-filename argument should be a string!\n");

}

// Read the strings into StrBuffers.

if (mxGetString(prhs[1], StrBuffer2, sizeof(StrBuffer2)-1)) {

mexErrMsgTxt("Unable to read second filepath argument string!\n");

}

if (mxGetString(prhs[2], StrBuffer3, sizeof(StrBuffer3)-1)) {

mexErrMsgTxt("Unable to read third part-filename argument string!\n");

}

filename\_input[0] = '\0';

strcat\_s(filename\_input, StrBuffer2);

strcat\_s(filename\_input, "R#");

strcat\_s(filename\_input, StrBuffer3);

return\_code = DataBufferInitializeFile(OPTOTRAK, filename\_input);

if (return\_code != 0) {

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("DataBufferInitiazeFile OPTOTRAK");

return;

}

filename\_input[0] = '\0';

strcat\_s(filename\_input, StrBuffer2);

strcat\_s(filename\_input, "O1#");

strcat\_s(filename\_input, StrBuffer3);

return\_code = DataBufferInitializeFile(ODAU1, filename\_input);

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("DataBufferInitializeFile ODAU1");

} else if (strcmp(StrBuffer, "DataBufferStart") == 0) {

return\_code = DataBufferStart();

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("DataBufferStart");

} else if (strcmp(StrBuffer, "DataBufferWriteData") == 0) {

// Returns -1 and DataBufferWriteData || uSpoolStatus on error.

// Returns 0 and uSpoolComplete on success.

return\_code = DataBufferWriteData(&uRealtimeDataReady, &uSpoolComplete, &uSpoolStatus, NULL);

nlhs = 2;

if (return\_code != 0 || uSpoolStatus != 0) {

plhs[0] = mxCreateDoubleScalar(-1);

if (return\_code != 0) {

plhs[1] = mxCreateString("DataBufferWriteData");

} else {

plhs[1] = mxCreateString("uSpoolStatus");

}

} else {

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateDoubleScalar(uSpoolComplete);

}

} else if (strcmp(StrBuffer, "FileConvert") == 0) {

// Ensure we have a second and third string argument with the filename (path, part of filename).

if (nrhs != 3) {

mexErrMsgTxt("A second and third string arguments must also be passed with DataBufferInitializeFile!\n");

}

// Make sure the second and third arguments are strings.

if (!mxIsChar(prhs[1])) {

mexErrMsgTxt("The second filepath argument should be a string!\n");

}

if (!mxIsChar(prhs[2])) {

mexErrMsgTxt("The third part-filename argument should be a string!\n");

}

// Read the strings into StrBuffers.

if (mxGetString(prhs[1], StrBuffer2, sizeof(StrBuffer2)-1)) {

mexErrMsgTxt("Unable to read second filepath argument string!\n");

}

if (mxGetString(prhs[2], StrBuffer3, sizeof(StrBuffer3)-1)) {

mexErrMsgTxt("Unable to read third part-filename argument string!\n");

}

filename\_input[0] = '\0';

filename\_output[0] = '\0';

strcat\_s(filename\_input, StrBuffer2);

strcat\_s(filename\_output, StrBuffer2);

strcat\_s(filename\_input, "R#");

strcat\_s(filename\_output, "C#");

strcat\_s(filename\_input, StrBuffer3);

strcat\_s(filename\_output, StrBuffer3);

return\_code = FileConvert(filename\_input, filename\_output, OPTOTRAK\_RAW);

if (return\_code != 0) {

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("FileConvert OPTOTRAK");

//OptotrakGetErrorString(szNDErrorString, MAX\_ERROR\_STRING\_LENGTH + 1);

//mexPrintf("API Error: %s", szNDErrorString); % printed weird text...

return;

}

filename\_input[0] = '\0';

filename\_output[0] = '\0';

strcat\_s(filename\_input, StrBuffer2);

strcat\_s(filename\_output, StrBuffer2);

strcat\_s(filename\_input, "O1#");

strcat\_s(filename\_output, "V1#");

strcat\_s(filename\_input, StrBuffer3);

strcat\_s(filename\_output, StrBuffer3);

return\_code = FileConvert(filename\_input, filename\_output, ANALOG\_RAW);

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("FileConvert ODAU");

} else if (strcmp(StrBuffer, "DataGetLatest3D") == 0) {

//(float)-3.0E28

return\_code = DataGetLatest3D(&uFrameNumber, &uElements, &uFlags, p3dData);

if (return\_code != 0) {

plhs[1] = mxCreateString("DataGetLatest3D");

} else {

// Send back frame number, elements, and raw data in a float array.

plhs[1] = mxCreateDoubleMatrix(1, (uElements \* 3) + 3, mxREAL);

y = mxGetPr(plhs[1]);

y[0] = (double)uFrameNumber;

y[1] = (double)uElements;

y[2] = (double)uFlags;

for (ui = 1; ui <= uElements; ui++) {

uj = ui \* 3;

y[uj] = (double)p3dData[ui-1].x;

y[uj+1] = (double)p3dData[ui-1].y;

y[uj+2] = (double)p3dData[ui-1].z;

}

}

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

} else if (strcmp(StrBuffer, "DataGetLatestOdauRaw") == 0) {

return\_code = DataGetLatestOdauRaw(ODAU1, &uFrameNumber, &uElements, &uFlags, puRawData);

if (return\_code != 0) {

plhs[1] = mxCreateString("DataGetLatestOdauRaw");

} else {

// Send back frame number, elements, and raw data in a float array.

plhs[1] = mxCreateDoubleMatrix(1, uElements + 3, mxREAL);

y = mxGetPr(plhs[1]);

y[0] = (double)uFrameNumber;

y[1] = (double)uElements;

y[2] = (double)uFlags;

for (ui = 1; ui <= uElements; ui++) {

y[ui+2] = (double)((int)(puRawData[ui-1])) \* 0.000305175 / (float)1; // final divide is ODAU gain

}

}

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

} else {

mexErrMsgTxt("A command was not found for that string argument!\n");

}

}

**mexFunction.cpp [for an Optotrak Certus with 2 sensors and no Optotrak Data Acquisition Unit (ODAU)]**

// Written by Jarrod Blinch, August 2015

// Available from motorbehaviour.wordpress.com

#include "mex.h"

#include <matrix.h>

#include <string.h>

#include <windows.h>

#include <math.h>

#include "ndtypes.h"

#include "ndpack.h"

#include "ndopto.h"

void mexFunction(int nlhs, mxArray\* plhs[], int nrhs, mxArray\* prhs[]) {

int

return\_code;

unsigned int

uRealtimeDataReady = 0,

uSpoolComplete = 0,

uSpoolStatus = 0,

uFrameNumber,

uElements,

uFlags,

ui,

uj;

static int

puRawData[3 + 1]; // number of ODAU channels + 1

static Position3d

p3dData[1];

char

StrBuffer[65],

StrBuffer2[65],

StrBuffer3[65],

filename\_input[68],

filename\_output[68];

//szNDErrorString[MAX\_ERROR\_STRING\_LENGTH + 1]; // 2047 + 1 (ndopto.h)

double

\*y,

dk,

dl;

// Make sure there is only one argument.

if (nrhs < 1) {

mexErrMsgTxt("At least one string argument must be passed!\n");

}

// Make sure the argument is a string.

if (!mxIsChar(prhs[0])) {

mexErrMsgTxt("The first argument should be a string!\n");

}

// Read the string into StrBuffer.

if (mxGetString(prhs[0], StrBuffer, sizeof(StrBuffer) - 1)) {

mexErrMsgTxt("Unable to read the argument string!\n");

}

if (strcmp(StrBuffer, "TransputerDetermineSystemCfg") == 0) {

// This will beep twice.

return\_code = TransputerLoadSystem("system");

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("TransputerDetermineSystemCfg");

}

else if (strcmp(StrBuffer, "TransputerLoadSystem") == 0) {

// Ensure we have a four additional arguments.

if (nrhs != 5) {

mexErrMsgTxt("Three additional arguments (#markers, Hz, time, cam filename) must be passed with TransputerLoadSystem!\n");

}

// Make sure the arguments are as expected.

if (!mxIsDouble(prhs[1])) {

mexErrMsgTxt("The second argument (num oprotrak markers) should be an unsigned integer!\n");

}

if (!mxIsDouble(prhs[2])) {

mexErrMsgTxt("The third argument (colection frequency) should be a double!\n");

}

if (!mxIsDouble(prhs[3])) {

mexErrMsgTxt("The fourth argument (collection duration) should be a double!\n");

}

if (!mxIsChar(prhs[4])) {

mexErrMsgTxt("The fifth argument (cam filename) should be a string!\n");

}

// Read the second and third arguments into unsigned integers.

ui = (unsigned int)mxGetScalar(prhs[1]); // number of Optotrak markers

dk = mxGetScalar(prhs[2]); // collection frequency

dl = mxGetScalar(prhs[3]); // collection duration (s)

if (mxGetString(prhs[4], StrBuffer2, sizeof(StrBuffer2) - 1)) {

mexErrMsgTxt("Unable to read fifth cam filename argument string!\n");

}

return\_code = TransputerLoadSystem("system"); // This will beep twice.

if (return\_code != 0) {

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("TransputerLoadSystem");

TransputerShutdownSystem();

return;

}

return\_code = TransputerInitializeSystem(OPTO\_LOG\_ERRORS\_FLAG | OPTO\_LOG\_MESSAGES\_FLAG);

if (return\_code != 0) {

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("TransputerInitializeSystem");

TransputerShutdownSystem();

return;

}

return\_code = OptotrakLoadCameraParameters(StrBuffer2);

if (return\_code != 0) {

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("OptotrakLoadCameraParameters");

TransputerShutdownSystem();

return;

}

return\_code = OptotrakSetStroberPortTable(ui, 0, 0, 0); // Number of markers connected to each port

if (return\_code != 0) {

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("OptotrakSetStroberPortTable");

TransputerShutdownSystem();

return;

}

return\_code = OptotrakSetupCollection(

ui, // Number of markers in the collection

(float)dk, // Frequency to collect data frames at

(float)2500.0, // Marker frequency for marker maximum on-time

30, // Dynamic or Static Threshold value to use

160, // Minimum gain code amplification to use

1, // Stream mode for the data buffers

(float)0.35, // Marker Duty Cycle to use

(float)7.0, // Voltage to use when turning on markers

(float)dl, // Number of seconds of data to collect

(float)0.0, // Number of seconds to pre-trigger data by, not suppoted and must be 0

OPTOTRAK\_NO\_FIRE\_MARKERS\_FLAG | OPTOTRAK\_BUFFER\_RAW\_FLAG); // OPTOTRAK\_GET\_NEXT\_FRAME\_FLAG often used with realtime data

if (return\_code != 0) {

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("OptotrakSetupCollection");

TransputerShutdownSystem();

return;

}

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("OptotrakSetupCollection");

}

else if (strcmp(StrBuffer, "TransputerShutdownSystem") == 0) {

return\_code = TransputerShutdownSystem();

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("TransputerShutdownSystem");

}

else if (strcmp(StrBuffer, "OptotrakActivateMarkers") == 0) {

return\_code = OptotrakActivateMarkers();

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("OptotrakActivateMarkers");

}

else if (strcmp(StrBuffer, "OptotrakDeActivateMarkers") == 0) {

return\_code = OptotrakDeActivateMarkers();

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("OptotrakDeActivateMarkers");

}

else if (strcmp(StrBuffer, "DataBufferInitializeFile") == 0) {

// Second argument example: ../data/raw/

// Third argument example: 001.P1

// The program will append R#

// Ensure we have a second and third string argument with the filename (path, part of filename).

if (nrhs != 3) {

mexErrMsgTxt("A second and third string arguments must also be passed with DataBufferInitializeFile!\n");

}

// Make sure the second and third arguments are strings.

if (!mxIsChar(prhs[1])) {

mexErrMsgTxt("The second filepath argument should be a string!\n");

}

if (!mxIsChar(prhs[2])) {

mexErrMsgTxt("The third part-filename argument should be a string!\n");

}

// Read the strings into StrBuffers.

if (mxGetString(prhs[1], StrBuffer2, sizeof(StrBuffer2) - 1)) {

mexErrMsgTxt("Unable to read second filepath argument string!\n");

}

if (mxGetString(prhs[2], StrBuffer3, sizeof(StrBuffer3) - 1)) {

mexErrMsgTxt("Unable to read third part-filename argument string!\n");

}

filename\_input[0] = '\0';

strcat\_s(filename\_input, StrBuffer2);

strcat\_s(filename\_input, "R#");

strcat\_s(filename\_input, StrBuffer3);

return\_code = DataBufferInitializeFile(OPTOTRAK, filename\_input);

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("DataBufferInitiazeFile OPTOTRAK");

}

else if (strcmp(StrBuffer, "DataBufferStart") == 0) {

return\_code = DataBufferStart();

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("DataBufferStart");

}

else if (strcmp(StrBuffer, "DataBufferWriteData") == 0) {

// Returns -1 and DataBufferWriteData || uSpoolStatus on error.

// Returns 0 and uSpoolComplete on success.

return\_code = DataBufferWriteData(&uRealtimeDataReady, &uSpoolComplete, &uSpoolStatus, NULL);

nlhs = 2;

if (return\_code != 0 || uSpoolStatus != 0) {

plhs[0] = mxCreateDoubleScalar(-1);

if (return\_code != 0) {

plhs[1] = mxCreateString("DataBufferWriteData");

}

else {

plhs[1] = mxCreateString("uSpoolStatus");

}

}

else {

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateDoubleScalar(uSpoolComplete);

}

}

else if (strcmp(StrBuffer, "FileConvert") == 0) {

// Ensure we have a second and third string argument with the filename (path, part of filename).

if (nrhs != 3) {

mexErrMsgTxt("A second and third string arguments must also be passed with DataBufferInitializeFile!\n");

}

// Make sure the second and third arguments are strings.

if (!mxIsChar(prhs[1])) {

mexErrMsgTxt("The second filepath argument should be a string!\n");

}

if (!mxIsChar(prhs[2])) {

mexErrMsgTxt("The third part-filename argument should be a string!\n");

}

// Read the strings into StrBuffers.

if (mxGetString(prhs[1], StrBuffer2, sizeof(StrBuffer2) - 1)) {

mexErrMsgTxt("Unable to read second filepath argument string!\n");

}

if (mxGetString(prhs[2], StrBuffer3, sizeof(StrBuffer3) - 1)) {

mexErrMsgTxt("Unable to read third part-filename argument string!\n");

}

filename\_input[0] = '\0';

filename\_output[0] = '\0';

strcat\_s(filename\_input, StrBuffer2);

strcat\_s(filename\_output, StrBuffer2);

strcat\_s(filename\_input, "R#");

strcat\_s(filename\_output, "C#");

strcat\_s(filename\_input, StrBuffer3);

strcat\_s(filename\_output, StrBuffer3);

return\_code = FileConvert(filename\_input, filename\_output, OPTOTRAK\_RAW);

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

plhs[1] = mxCreateString("FileConvert OPTOTRAK");

//OptotrakGetErrorString(szNDErrorString, MAX\_ERROR\_STRING\_LENGTH + 1);

//mexPrintf("API Error: %s", szNDErrorString); % printed weird text...

}

else if (strcmp(StrBuffer, "DataGetLatest3D") == 0) {

//(float)-3.0E28

return\_code = DataGetLatest3D(&uFrameNumber, &uElements, &uFlags, p3dData);

if (return\_code != 0) {

plhs[1] = mxCreateString("DataGetLatest3D");

}

else {

// Send back frame number, elements, and raw data in a float array.

plhs[1] = mxCreateDoubleMatrix(1, (uElements \* 3) + 3, mxREAL);

y = mxGetPr(plhs[1]);

y[0] = (double)uFrameNumber;

y[1] = (double)uElements;

y[2] = (double)uFlags;

for (ui = 1; ui <= uElements; ui++) {

uj = ui \* 3;

y[uj] = (double)p3dData[ui - 1].x;

y[uj + 1] = (double)p3dData[ui - 1].y;

y[uj + 2] = (double)p3dData[ui - 1].z;

}

}

nlhs = 2;

plhs[0] = mxCreateDoubleScalar(return\_code);

}

else {

mexErrMsgTxt("A command was not found for that string argument!\n");

}

}