CENTER FOR
MACHINE PERCEPTION

CZECH TECHNICAL
UNIVERSITY

RESEARCH REPORT

ISSN 1213-2365

FGI and plFGI

Maarten Bakker
Lian Ien Oei
Vladimír Smutný

smutny@cmp.felk.cvut.cz

CTU–CMP–2002–19

October 24, 2002

Available at
ftp://cmp.felk.cvut.cz/pub/cmp/articles/smutny/Bakker-TR-2002-19.pdf

This work has been supported by the Czech Ministry of Health under project NN6333-3/2000 and in part by the European Union under project IST-2001-33266, by the Czech Ministry of Education under project MSM 212300013, and by the Grant Agency of the Czech Republic under projects GACR 102/00/1679, GACR 102/01/0971 and GACR 102/01/1371.

Research Reports of CMP, Czech Technical University in Prague, No. 19, 2002

Published by

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

FGI and plFGI
Report 2002

Assignment: Bug fixing FGI and implementing plFGI
Date: June – September 2002
Authors: Maarten Bakker, Lian Ien Oei

Czech Technical University in Prague
Faculty of Electrical Engineering
Center for Machine Perception

Delft University of Technology
Faculty Information Technology and Systems
Sub faculty Technical Informatics

Preface

In 1998 D.J. Oei and A. Kapoerchan implemented a Frame Grabber Interface (FGI) for the people at the Center for Machine Perception (CMP) of the Czech Technical University in Prague. This FGI provides a communication layer between Matlab and the Frame Grabber SDK of Data Translation's DT-Open Layers family. In 2000 R. Heil and C.L. Wauters extended the FGI. Now the people at CMP needed a similar interface for the PixeLINK Camera. Also they wanted some bugs fixed in the existing FGI.

We have been working on this assignment as a training period for our study of Computer Science at Delft University of Technology, from June until September 2002. During this period Assistant Professor Vladimír Smutný has supervised us. The exchange between TUD and ÈVUT was organized by Professor Václav Hlaváè and Dr. Drs. Rothkrantz.

We would like to thank everyone who supported us during our project, especially Professor Václav Hlaváè, Assistant Professor Vladimír Smutný and Miss Eva Matysková from the Center for Machine Perception, Czech Technical University in Prague, and Dr. Drs. Rothkrantz from the Faculty of Information Technology and Systems, Delft University of Technology. Also we would like to thank D.J. Oei, A. Kapoerchan, R. Heil and C.L. Wauters for building FGI and providing documentation on that system, so we did not have to invent everything ourselves.

Maarten Bakker and Lian Ien Oei

September 2002

Abstract

Introduction

The people at the Center for Machine Perception of the Czech Technical University in Prague wanted to be able to grab images with a number of cameras from within the Matlab environment. Therefore they needed an interface between Matlab and the frame grabbers they were using. They already had such an interface, called FGI, for the frame grabbers of Data Translation's DT-Open Layers family. Our task was to update and bug fix this FGI, and to build an interface for cameras from the PixeLINK family.

Background

Matlab is a mathematical program, which can be used to make analyses and computations on images grabbed with a frame grabber. The Frame Grabber Interface provides a communication layer between Matlab and the frame grabber libraries. It is composed of MEX-files: dynamically linked subroutines, which can be called from Matlab as if they were built-in functions.

The original Frame Grabber Interface

The original FGI was a Matlab interface for Data Translation Frame Grabbers. With this interface it was possible to grab pictures from a Data Translation camera into Matlab. You could open and close a device, check whether it has been opened or not, grab an image into Matlab, get and set various variables, load a settings file and reset an opened device. It was composed of separate MEX-files for each function and an additional function '`mainfg`', which was used as a central place to store the device handles.

Updating FGI 2000

One part of the assignment was to update and bug fix the original FGI system.

The 'clear all bug' was a bug, which resulted in the loss of handles to open devices when typing '`clear all`' or '`clear mex`'. This problem was solved by calling the function '`mexLock`' from mainfg.

'Polishing' was another task. This implied that the device information structure should also show the settings for device specific functions, which are not shared by all types of Data Translation frame grabbers. Unfortunately we did not solve this problem.

The next problem was that sometimes the device driver's resources were not released properly. For this a subroutine was added, which is called in those places where problems used to occur.

In addition to this, we also added more descriptive error messages and improved the documentation of grabfg, which was incomplete.

Original design of the new system

Originally we designed a new system, which provides a uniform interface to the Matlab users for frame grabbers of all different families. It is composed of three layers. The command layer is the layer with functions visible to the Matlab users. This layer only provides a uniform interface for all types of cameras; almost no functionality is implemented here. The control layer consists of two functions: fgcontrol passes all commands to the correct device specific interface; fgdevices maintains the 'open device array' in which the device ID's from open devices are stored. The device specific layer consists of a separate dll file for every type of device. This dll file implements the commands and provides communication with the frame grabber libraries.

When studying the PixeLINK API, we discovered that it differs more from the DT Frame Grabber SDK than we thought originally. To deal with these differences, it would either be necessary to move device specific code to the control layer (which is not very nice), or to move fgdevices to the device specific layer (which would take *all* functionality out of the command layer and result in more overhead than benefits from the uniform user interface). Considering this, we decided in the end not to implement our original design.

Implementation of the PixeLINK Frame Grabber Interface

Finally we implemented the PixeLINK Frame Grabber Interface (plFGI) in a way very similar to the original FGI system. It is composed of several MEX-functions, which are callable from Matlab. It supports functionality to open and close a device, check whether a device is opened, get and set a number of parameters and grab an image from the camera. The function '`plDevices`' maintains the 'open device array', in which the device ID's are stored.

Programming environment

The updated FGI system is written in C, for Matlab version 5.3.1. For compiling and linking we used the M-file 'buildall.m', written by our predecessors. For creating plFGI, which is written in C++, we used Microsoft Visual C++ 6.0, with the libraries of Matlab version 6.1 and the PixeLINK Camera API Release 3.0.

Conclusion

The original FGI system is updated. The 'clear all bug' is fixed and the device driver's resources are released properly. More descriptive error messages are added and the documentation of grabfg is improved. The 'polishing' problem still remains unsolved though.

For the PixeLINK cameras plFGI is implemented. It supports most basic functionality: to open and close a device, check whether a device is opened, get and set a number of parameters and grab an image from the camera. Unlike the original FGI system, it only provides 1 way to call plGrab (instead of the 6 ways to call grabfg). Also it has no reset-function, because the PixeLINK API does not provide this.

A full list of topics for improvement is given in Appendix B.

Index

1 Introduction

The people at the Center for Machine Perception of the Czech Technical University in Prague wanted to be able to grab pictures with a number of cameras from within the Matlab environment. In this environment they can make analyses and computations on the images. Therefore they needed an interface between Matlab and the frame grabbers they were using. In 1998 two students from Delft University of Technology developed a Frame Grabber Interface (FGI) to provide a communication layer between Matlab and the Frame Grabber SDK of Data Translation's DT-Open Layers family. In 2000 two other students from Delft bug-fixed and extended FGI.

Our task was to fix a few other bugs in FGI, and to implement a Matlab interface to provide support for cameras from the PixeLINK family. This interface should provide the same functionality for the PixeLINK cameras as the existing FGI does for the Data Translation Frame Grabbers. If possible without losing any functionality, the same commands should be used from within Matlab, to communicate with both DT and PixeLINK cameras.

Chapter 2 starts with a description of the background to this problem. An overview of the Matlab environment and the Data Translation Frame Grabber and PixeLINK interfaces is given. Chapter 3 describes the original Frame Grabber Interface, as it was after the FGI 2000 project. The 4th chapter describes the updates and bug fixes we made to the original FGI system. Chapter 5 describes the architecture we originally had in mind for the new system. In this system the interfaces for the DataTranslation and PixeLINK frame grabbers are integrated into one system. Because of several reasons, this is not the system we finally implemented. In chapter 6 the implementation of the new system, which provides support for only the PixeLINK frame grabbers, is given. It starts with an overview of the system architecture. Then it describes all plFGI functions. Chapter 7 describes the programming environment used.

2 Background

The people at the Center for Machine Perception want to be able to grab images with a camera from within the Matlab environment. Therefore a Frame Grabber Interface is needed, to provide a communication layer between Matlab and the frame grabbers. This chapter first gives an overview of the global architecture. Then it describes the communication interfaces with Matlab and the Data Translation and PixeLINK frame grabbers.

2.1 Overview

The Frame Grabber Interface is a set of functions, which can be called from Matlab, to communicate with a frame grabber. This frame grabber can either be a plug-in board, communicating with a camera, or a camera with an integrated board.

The interface between Matlab and FGI consists of MEX-functions, which are callable directly from the Matlab user interface. For communication between FGI and the frame grabber, the manufacturer of the frame grabber device provides special libraries, with functions that can be called from an application program.

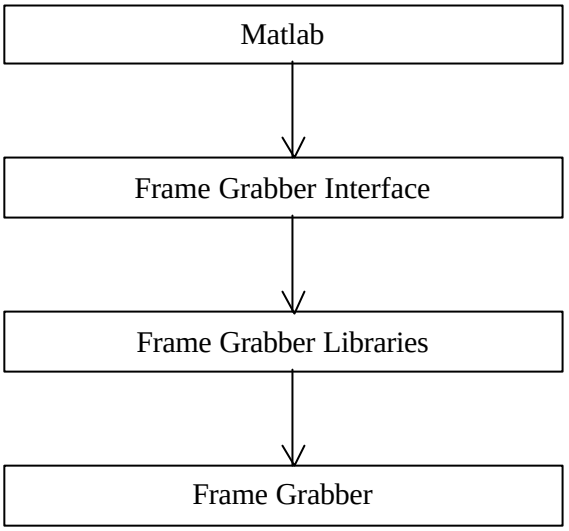

Figure 1 ***Global architecture***

2.2 Matlab

Matlab is a mathematical program for technical computing. It can be used to solve many technical computing problems, especially those with matrix and vector formulations. With Matlab it is possible to make analyses and computations on images grabbed by a frame grabber device.

To interact with data and programs external to the Matlab environment, Matlab provides an Application Program Interface (API). A part of the API are MEX-files: dynamically linked subroutines that the Matlab interpreter can automatically bad and execute. With these it is possible to call your own C subroutines from Matlab as if they were built-in functions.

A MEX-file is just like a regular C-file. It only has a compulsory gateway routine to interface with Matlab: the `mexFunction` with its parameters `nlhs`, `plhs`, `nrhs` and `prhs`, where `prhs` is an array of right-hand input arguments, `nrhs` is the number of right-hand input arguments, `plhs` is an array of left-hand output arguments, and `nlhs` is the number of left-hand output arguments.

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const
mxArray *prhs[])
```

You can access the data in the `mxArray` structure and manipulate it. Most files in the original FGI were MEX-files.

2.3 Data Translation Frame Grabbers

Data Translation (DT) Frame Grabbers are frame grabber boards for the Peripheral Component Interconnect (PCI) bus. You can use them for image analysis and machine vision application. The next picture gives an example of such a board:

Figure 2 ***DT3152 Frame Grabber***

To a frame grabber one can connect several cameras, depending on the number of input channels.

DT Frame Grabbers are Open Layer devices. For the Open Layers-family, Data Translation provides a Software Development Kit (SDK) for Microsoft Windows platforms. This SDK is composed of commands, which are callable from higher-level languages, such as C. For many types of frame grabbers, Data Translation also provides additional function libraries, with functions that are specific for that type of frame grabber. The SDK and function libraries together form the library layer, through which the application layer can communicate with the device interface layer. The device interface layer handles the interface between the API and the board.

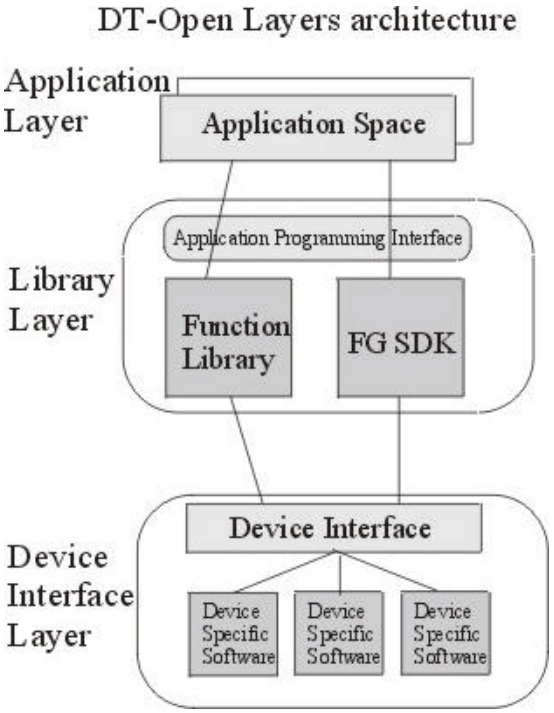

Figure 3 ***DT-Open Layers architecture***

2.4 PixeLINK Cameras

The PixeLINK Megapixel FireWire (IEEE 1394) Cameras are firewire cameras with a CMOS image sensor and integrated image processing electronics. The camera can be used both as a frame grabber and a source of live images. A firewire adapter card is needed to connect the camera to the computer.

Figure 4 ***Megapixel FireWire Cameras***

The PixeLINK Camera Application Programming Interface (API) can be used to develop new software applications for the PixeLINK Cameras. It is a dynamic link library (PimMegaApi.dll), which provides a set of functions to control the camera. With these functions you can set imaging parameters, view images and capture frames and store them as image files. The API functions can be called from a C++ program.

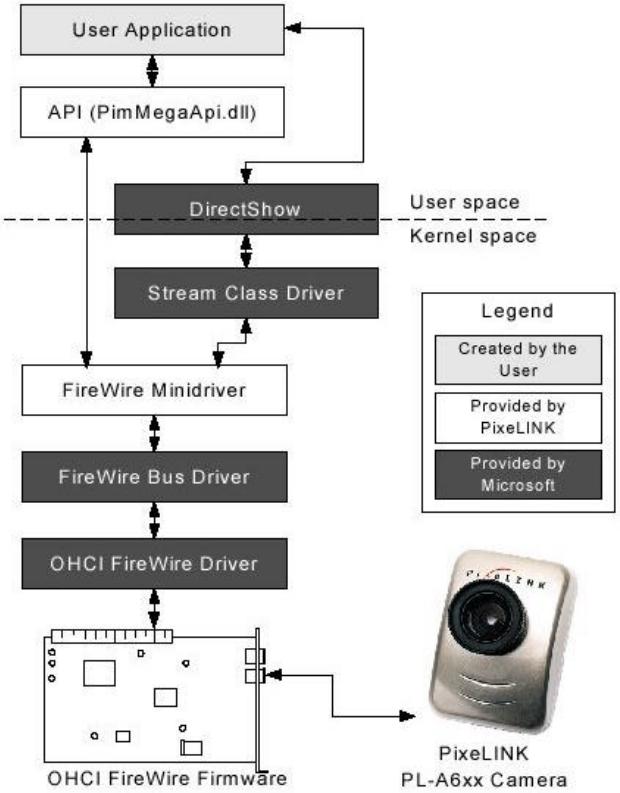

Figure 5 ***PixeLINK Software Architecture***

3 The original Frame Grabber Interface

The original Frame Grabber Interface (FGI) was a Matlab interface for Data Translation Frame Grabbers. With this interface it was possible to grab pictures from a Data Translation camera into Matlab. You could open and close a device, check whether it has been opened or not, grab an image into Matlab, get and set various variables, load a settings file and reset an opened device. This chapter explains the structure of the original FGI and gives a short description of all existing FGI functions.

3.1 Structure of the original system

In the next figure we give an overview of the structure of the original Frame Grabber Interface.

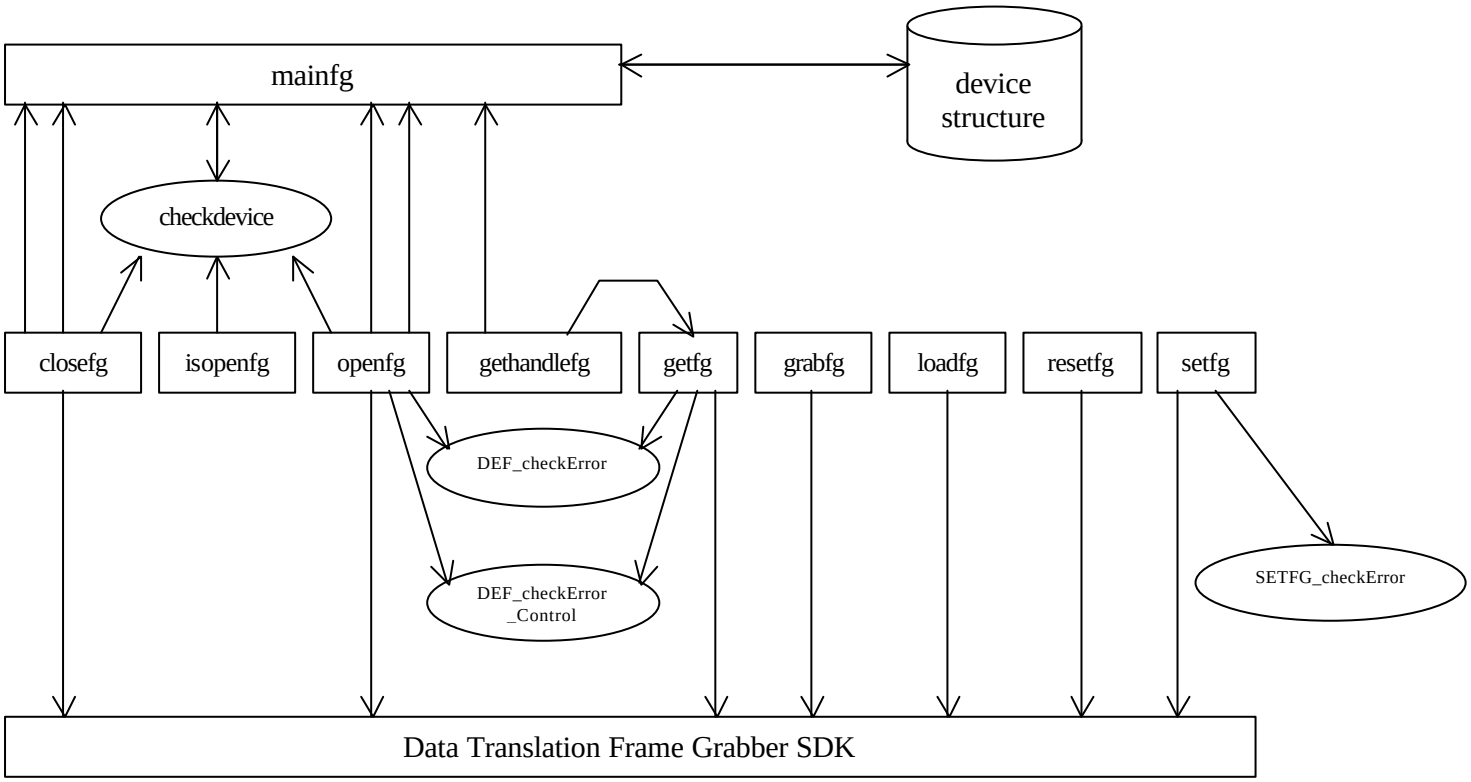

Figure 6 ***Overview of the original Frame Grabber Interface***

The system was composed of several functions (closefg, isopenfg, openfg, ...), which could be called from within the Matlab environment. Each of these functions called the appropriate function from the Frame Grabber SDK, to perform the action needed. Mainfg was used as a central place to store the device handles.

3.2 Description of the existing FGI functions

We give a short description of each of the existing FGI functions[1]. For a more detailed description we refer to 'Matlab Interface For Data Translation Mach Frame grabbers - User Manual & Technical Guide'[2].

- ***Openfg***

DLL name	Openfg.dll
Syntax	Devinfo = openfg('*alias*')
Description	This function opens a DT frame grabber device.
Input	The alias of the frame grabber. The user is supposed to know this alias.
Output	The function will return a structure with the device information.
Example	m = openfg('DT3155');

- ***Closefg***

DLL name	Closefg.dll
Syntax	closefg(*devinfo*) OR closefg('*alias*')
Description	This function closes a DT frame grabber device.
Input	The device information structure (*devinfo*) or the alias (*alias*) of the opened frame grabber
Output	None
Example	closefg(m); or closefg('DT3155');

[1] These descriptions were copied from the 'FGI Report 2000', [Heil/Waut., 2000], p. 28-31, and slightly adjusted to represent the actual situation more accurately.
[2] [Oei/Kap., 1998]

- ***Grabfg***

DLL name	Grabfg.dll
Syntax	imgmatrix = grabfg(*devinfo*) OR moviematrix = grabfg(*devinfo, imgmatrix*) OR [imgmatrix, moviematrix] = grabfg(*devinfo*) OR grabfg(*devinfo, imgmatrix*) OR grabfg(*devinfo, imgmatrix, moviematrix*) OR grabfg(*devinfo, imgmatrix, moviematrix, scaling factor*)
Description	This function grabs an image from an opened frame grabber device. The user has to give the frame grabbers *device information structure (devinfo)* as input argument. When used with the *moviematrix* argument, a standard bitmap will be returned according to the Matlab movie format. The *scaling factor*, can be used for scaling down this bitmap.
Input	The *device information structure* (*devinfo*) of an opened frame grabber device and optionally the pre-allocated *image matrix* (*imgmatrix*), a *movie matrix (moviematrix)* and a *scaling factor* for movies. The movie matrix has size [m, 1] and consists of doubles. Calculate m as follows: m = 388 + (img height / scaling factor) * (img width / scaling factor) / 8. The default value for the movie scaling factor is 2.
Output	If the input isn't valid or the frame grabber isn't able to grab an image, some error message will be shown to the user. If everything's okay it will return the grabbed image stored in a *Matlab matrix* (*imgmatrix*). The image is stored in transposed way. To show the image correctly in Matlab you can use the accent ' on the imgmatrix to correct it (image(imgmatrix'))
Example	im = grabfg(m); or im=uint8(zeros(768,576)); and grabfg(m,im);

- ***Getfg***

DLL name	Getfg.dll
Syntax	Getfg(*devinfo*) OR getfg(*devinfo, parname*)
Description	This function returns an updated structure with the device information or a parameter from the device information structure.
Input	The device information structure (*devinfo*) of the opened frame grabber and the chosen parameter (*parname*).
Output	The function returns an updated device information structure or the chosen parameter from the device information structure.
Example	Getfg(m); or getfg(m, 'frameheight');

- ***Setfg***

DLL name	Setfg.dll
Syntax	Setfg(*devinfo, parname, parvalue*) OR setfg(*devinfo*)
Description	This function sets a parameter in the device information structure. Setfg(m) lists all the possible variables
Input	The device information structure (*devinfo*) of the opened frame grabber, the chosen parameter (*parname*) and the new value (*parvalue*) of the chosen parameter.
Output	None
Example	Setfg(m, 'frameheight', 400); or setfg(m)

- ***Resetfg***

DLL name	Resetfg.dll
Syntax	resetfg(*devinfo*)
Description	This function resets a DT frame grabber device.
Input	The device information structure (*devinfo*) of the opened frame grabber.
Output	None.
Example	resetfg(m);

- ***Loadfg***

DLL name	Loadfg.dll
Syntax	Loadfg(*devinfo, filename*)
Description	This function loads settings from an ini file into the device information structure.
Input	The device information structure (*devinfo*) of the opened frame grabber, and the filename of the settings file (full path).
Output	None.
Example	Loadfg(m, 'c:\camfiles\pulnix.ini');

- ***Isopenfg***

DLL name	Isopenfg.dll
Syntax	Isopenfg('*alias*')
Description	This function will let the user check whether the frame grabber device has been opened or not. It will return a double acting like a Boolean.
Input	The alias of the frame grabber. The user is supposed to know this alias.
Output	The function will return a double indicating the status of frame grabber device. If the function returns 1 the frame grabber has been opened, otherwise the function returns zero.
Example	Isopenfg('DT3155');

- ***Gethandlefg***

DLL name	Gethandlefg.dll
Syntax	Gethandlefg('*alias*')
Description	This function will return the device information structure of the frame grabber device when you lost the handle to the device.
Input	The alias of the frame grabber. The user is supposed to know this alias.
Output	The function will return the device information structure of the opened Frame grabber device with the given alias.
Example	m2=gethandlefg('DT3155');

4 Updating FGI 2000

One part of the assignment was to update and bug fix the original FGI system. At first this included fixing the 'clear all bug' and to do some 'polishing'. Later the 'proper release of the device driver's resources' was added. In this chapter these problems will be explained, and the solutions will be given.

4.1 The 'clear all bug'

One of the problems with FGI was the loss of handles to open devices when typing '`clear all`' in Matlab. The '`clear all`' command in Matlab removes all variables, functions and MEX-files from the Matlab memory, leaving the workspace empty. When this command is used, the variables, in which the device structures are stored, and the MEX-files, in which the handles to open devices are stored, are cleared from memory and the reference to any open devices is lost. However, the underlying Frame Grabber SDK still regards the device as being opened. When one tried to reopen the device after using '`clear all`', an error-message was generated instead. This meant that the user had to close and restart Matlab to continue working with the camera.

We solved this problem by calling the mexLock-function from within the mainfg-function. mexLock locks a MEX-file, so that it cannot be cleared from memory. This means that mainfg remains in memory, even after calling '`clear all`' or '`clear mex`' (which clears all MEX-files from memory). Since the device array, in which the handles to open devices are stored, is stored in mainfg, the handles will remain available.

The following code example shows the use of the mexLock function in mainfg:

```
void mexFunction(int nlhs, mxArray *plhs[],
                         int nrhs, const mxArray *prhs[])
{
  int buflen;
  int i;
  double *y;
  char *task = "";
  char *name = "";
  bool found = false;

  /* Added by M.Bakker 2002/06/24 12:35
   * Use mexLock to prevent 'clear all' or 'clear mex' from clearing
   * mainfg.dll from Matlab memory.  The deviceArray stays intact.
   */
  mexLock();

  . . .
  . . .
  . . .
  . . .
```

Code example 1 ***The use of the mexLock() function in mainfg***

4.2 'Polishing'

The Data Translation Frame Grabber family has several different frame grabbers. These have a lot of functions in common, but some functions are specifically for one type of frame grabber. With the original FGI, it was not possible to see the settings for these specific functions. Only the settings for the shared functions were stored in the device information structure. With 'polishing' we mean that the device information structure also will show the settings for device specific functions.

When studying the source code of the original FGI system, we determined that it would be easiest to immediately redesign the system, in order to integrate the support for the PixeLINK camera. When implementing the new system, we could solve this problem at the same time, instead of solving this first and then possibly having to do it again with the implementation of the new system. Therefore we postponed the solution to this problem.

We designed the new system as described in chapter 5. After a while we decided not to implement that design, but to implement the system described in chapter 6. Unfortunately, after implementing that system, we didn't have enough time left to polish the original FGI system. Therefore this problem still remains unsolved.

4.3 Proper release of the device driver's resources

The original complaint was that in grabfg, when an error was encountered, some of the device driver's resources were not properly released. This resulted in a new error message and the need to close and re-open Matlab after the error had occurred several times. A subroutine was added to properly release resources. A call to this subroutine is made every time before a call to mexErrMsgTxt in a place problems could occur.

4.4 Other changes to the original FGI system

While solving the problems mentioned above, we noticed a few other things that could be changed easily to improve grabfg.

More descriptive error messages in grabfg

To make the source more readable, we had already changed the layout of the source code. After making the changes mentioned above, we noticed that the error messages after calling the Open Layers SDK to acquire a frame were very undescriptive, so we added several new ones.

Documentation of grabfg

While we were working on the source code and checking the documentation, we noticed that not all six different syntaxes for calling grabfg were documented, so we fixed that as well.

Some parts of the source code of grabfg are given in Appendix C to show the changes.

5 Original design of the new system

This chapter describes the original design of the new FGI system. First an overview is given. Then the functions of the three layers (command layer, control layer and device specific layer) are described. Finally we explain why we did not implement this system.

5.1 Overview

In the next figure we give an overview of the original design of the new system.

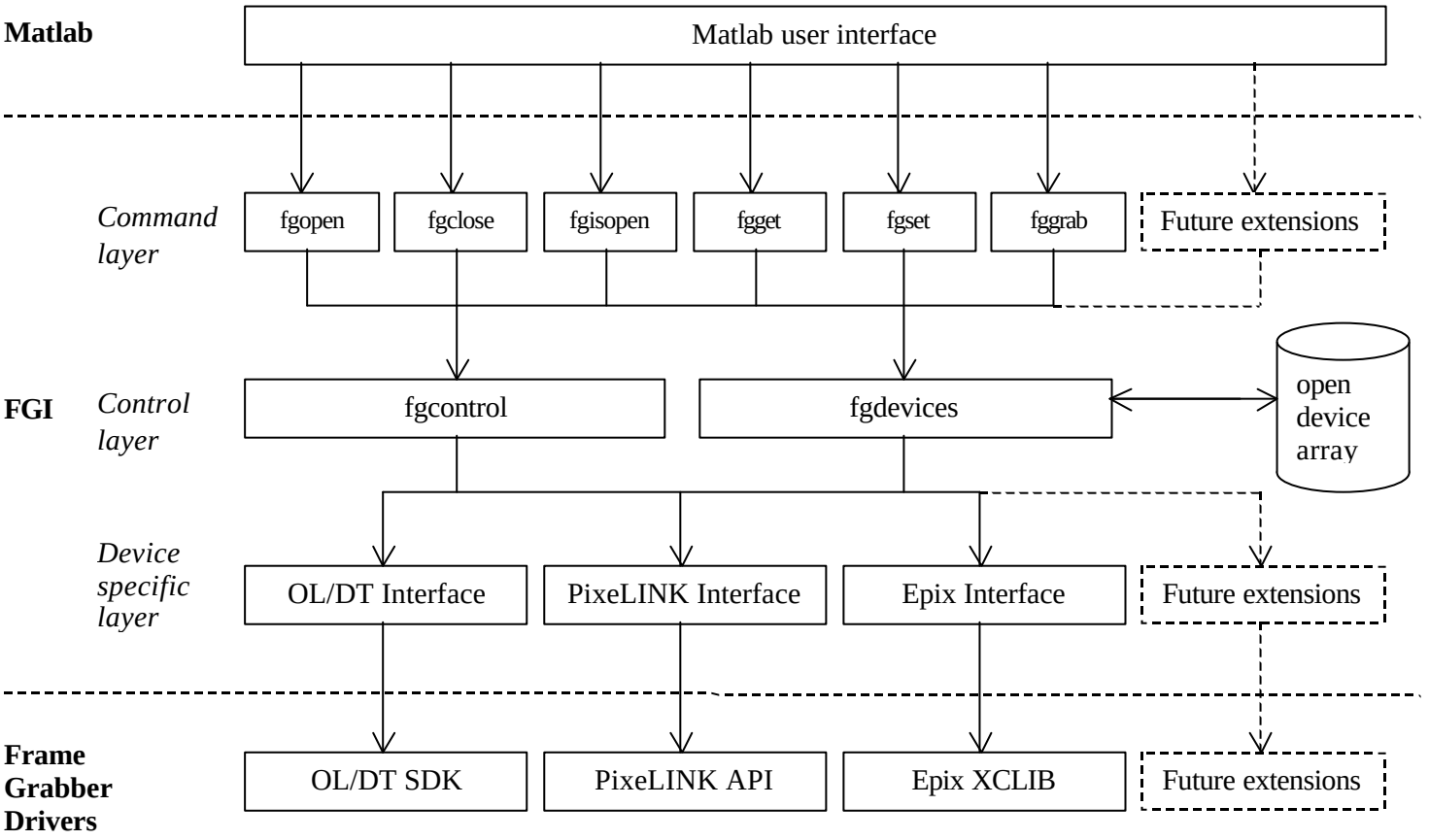

Figure 7 ***Overview of the original design of the new system***

In the new system, there are three layers. We have taken all device specific functionality out of the top layer functions that are callable from Matlab, and moved it into a lower level, device specific layer. This way it becomes easy to implement new frame grabbing devices. When adding new frame grabbers, one only has to add a new device specific interface to the device specific layer. The command and control layers will stay the same.

In the command layer no functions are implemented. It only provides a uniform interface for all types of cameras to the Matlab environment. The control layer takes care of updating the open device structure, loading the appropriate device specific code for the requested device and passing calls to the device specific interfaces. In the device specific layer all functions are implemented separately for each type of camera.

5.2 Description of the three layers

5.2.1 Command layer

This layer provides a uniform Matlab command interface to any frame grabber. In this layer almost no functionality is implemented. It only deals with those parts of the functions, which are common to all frame grabber interfaces. All function calls like fgopen, fgclose, etc. are passed through to fgcontrol in the control layer.

5.2.2 Control layer

The control layer consists of two functions: fgcontrol and fgdevices. fgcontrol passes all commands (fgopen, fgclose, etc.) to the correct device specific interface (DSI). fgdevices maintains the 'open device array' in which the device ID's from open devices are stored. The Matlab mexLock function is used to protect the 'open device structure' and the pointer(s) to the DSI from the Matlab commands '`clear mex`' or '`clear all`'.

Fgcontrol

Fgcontrol is the main MEX-function that is called by the command layer functions with both the name of the task to be performed and the needed parameters as arguments. It uses the device alias to determine which DSI has to be called. If the DSI is called for the first time, it loads the correct dll into memory. Then it calls the DSI and passes the command to the device specific layer.

Fgdevices

Fgdevices keeps track of opened frame grabbing devices. Therefore it maintains the 'open device array' in which a list of the aliases and handles of all open devices is kept. If needed in the future, this structure could be extended to contain other device information for the open device. It provides functionality for manipulating the open device array and getting information stored in the array. Fgdevices is called from control layer as well as the device specific layer.

5.2.3 Device specific layer

This layer consists of a separate dll file for every type of device (one dll for DT grabbers, one for Pixelink, etc). This dll file is composed of a control function, which is called by fgcontrol, and several other functions, which implement the different top layer calls (fgopen, fgclose, etc.). When the control function is called by fgcontrol, it passes the command to the correct function (e.g. plOpen), which in turn implements the command and provides communication with the frame grabber libraries.

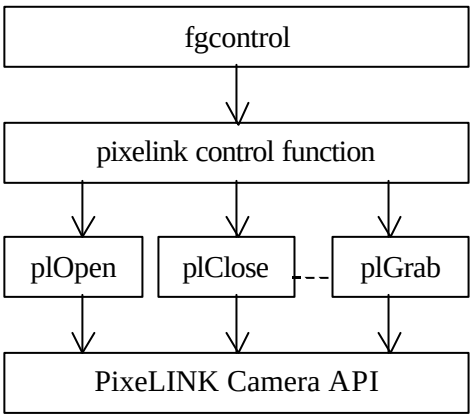

Figure 8 ***Implementation of a DSI***

5.3 Why we did not implement this system

We had already implemented most of the command layer and control layer functions, when we started studying the PixeLINK Camera Application Programming Interface (API) more closely. Then we discovered that, unlike the DataTranslation Frame Grabber SDK, the PixeLINK API doesn't use a unique device alias to identify a specific camera. Also handles to open devices (device ID's) are stored in a slightly different way.

Implementing this in our original design of the new system would mean that fgdevices would have to be able to treat different devices in different ways. This would not be very nice, since then it would be necessary to move device specific code to the control layer (which was meant to remain the same, even after adding new device types).

Another possibility is to move fgdevices to the device specific layer. This however would also mean that *all* functionality has to be taken out of the command layer and moved to the device specific layer, which would leave the command layer rather useless. It would still provide a uniform interface to the Matlab users, but it would also mean a lot of unnecessary overhead when calling one of the functions. This would not be much of a problem when the user would only have to grab one image, but when grabbing 1000 images in a loop for example, it would save a lot of time to call the device specific interface immediately.

This is the reason why in the end we decided not to implement this original design. Instead we implemented plFGI in the same way as the old FGI system for the DataTranslation frame grabbers. How this system is implemented is described in the next chapter.

6 Implementation of the PixeLINK Frame Grabber Interface

This chapter describes the implementation of the PixeLINK Frame Grabber Interface (plFGI), which is very much similar to the original FGI system for the DataTranslation frame grabbers. The system is composed of several MEX-functions, which are callable from Matlab. It supports functionality to open and close a device, check whether a device is opened, get and set a number of parameters and grab an image from the camera. This chapter starts with an overview of the structure of the new system. Then it gives a short description of all functions. Finally it describes the functions in detail.

6.1 Structure of plFGI

In the next figure we give an overview of the structure of plFGI.

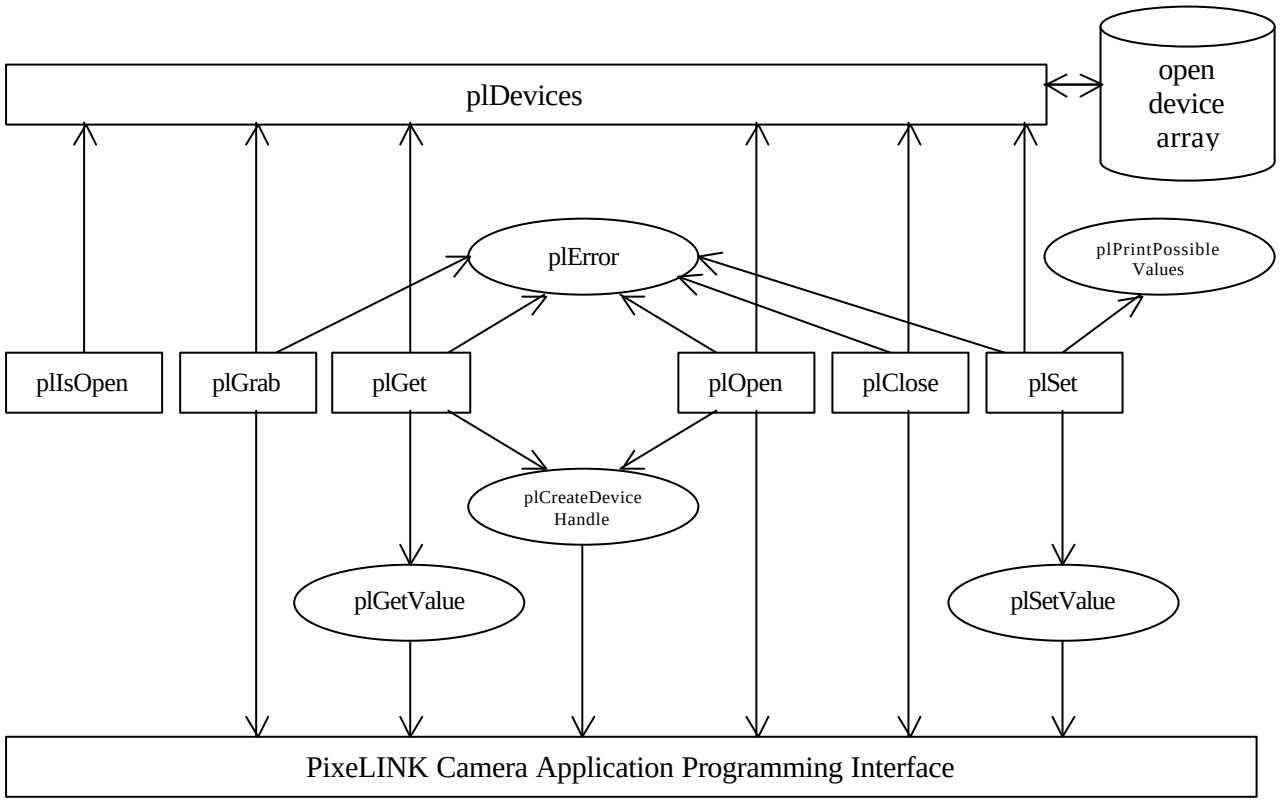

Figure 9 ***Structure of plFGI***

The system is composed of several functions (plIsOpen, plGrab, plGet, plOpen, plClose and plSet), which can be called from Matlab. These functions call the appropriate functions from the PixeLINK Camera Application Programming Interface (API). plDevices maintains the 'open device array', in which the device ID's (handles to open devices) are stored. In figure 9, the relations between the internal and external plFGI functions are shown. An arrow indicates that a function is called by another function. The ↔ arrow between plDevices and the open device array means that data is exchanged in two directions.

6.2 Short description of the plFGI functions

plFGI supports the following functions:

- **plClose** closes a PixeLINK camera.
- **plGet** gets the handle of the camera or the value of a camera or image processing parameter.
- **plGrab** grabs an image from a PixeLINK camera.
- **plIsOpen** checks if a PixeLINK camera is already open.
- **plOpen** opens a PixeLINK camera or returns its handle if it was already open.
- **plSet** sets the value of a camera or image processing parameter or shows the possible parameters or values.

We give a short description of each of the plFGI functions, which are callable from Matlab. In paragraph 6.3 we describe plDevices, which is used as an internal function, called by the other functions. In paragraph 6.4 a more detailed description of the plFGI functions is given.

In the descriptions below, prhs[0] is used to indicate the first input parameter, prhs[1] for the second input parameter, prhs[2] for the third, etcetera. Similarly, plhs[0] is used to indicate the (first) return value of the MEX-function.

- ***plClose***

DLL name	plClose.dll
Syntax	`plClose(handle)` or `plClose(serialnumber)`
Description	plClose closes a PixeLINK camera device.
Input	prhs[0] = (struct) handle of the device, or (U32) serial number of the device
Output	None
Examples	`plClose(m);` `plClose(75122);`

- ***plGet***

DLL name	plGet.dll
Syntax	`plGet(handle)`, `plGet(handle, parametername)`, `plGet(serialnumber)` or `plGet(serialnumber, parametername)`
Description	plGet returns the device's up-to-date handle if 1 argument is given, or returns the value of the given parameter if 2 arguments are given.
Input	prhs[0] = (struct) handle for the device, or (U32) serial number of the device prhs[1] = (string) name of the parameter (optional)
Output	plhs[0] = (struct) handle for the device (with 1 input argument) plhs[0] = value(s) of the given parameter (with 2 input arguments)

Examples	`m = plGet(m);`
	`value = plGet(m, 'Timeout');`
	`m = plGet(75122);`
	`value = plGet(75122, 'Timeout');`

- ***plGrab***

DLL name	plGrab.dll
Syntax	`imagematrix = plGrab(handle)` or `imagematrix = plGrab(serialnumber)`
Description	plGrab grabs a frame from the PixeLINK camera device and places it into an mxArray.
Input	prhs[0] = (struct) handle for the device, or (U32) serial number of the device
Output	plhs[0] = (mxArray) grabbed frame
Examples	`im = plGrab(m);` `im = plGrab(75122);`

- ***plIsOpen***

DLL name	plIsOpen.dll
Syntax	`plIsOpen(handle)` or `plIsOpen(serialnumber)`
Description	plIsOpen checks whether the device has been opened. It returns 0 if the device has not been opened, and 1 if it has been opened.
Input	prhs[0] = (struct) device handle, or (U32) serial number of the device
Output	plhs[0] = (double) 0 if the device has not been opened 1 if the device has been opened
Examples	`boolean = plIsOpen(m);` `boolean = plIsOpen(75122);`

- ***plOpen***

DLL name	plOpen.dll
Syntax	`handle = plOpen(serialnumber)`
Description	plOpen opens a PixeLINK camera device and returns the device's handle. If there is no camera with the given serial number, a list of available serial numbers is printed.
Input	prhs[0] = (int) serial number of the device to be opened
Output	plhs[0] = (struct) handle for the device
Example	`m = plOpen(75122);`

- ***plSet***

DLL name	plSet.dll
Syntax	`plSet(handle)`, `plSet(handle, parametername)`, `plSet(handle, parametername, value)`, `plSet(serialnumber)`, `plSet(serialnumber, parametername)`, or `plSet(serialnumber, parametername, value)`
Description	plSet sets the value of a given parameter. When 1 argument is given a list of all possible parameters is given. When 2 arguments are given, the possible values of the given parameters are given. When called with 3 or more arguments, the device's parameter given in the second argument will be set to the value(s) given in the other argument(s).
Input	prhs[0] = (struct) handle for the device, or (int) serial number of the device prhs[1] = (string) name of the parameter (optional) prhs[2...n] = value of the given parameter (optional)
Output	A list of all possible parameters is printed on the screen (with 1 input argument) The possible values of the parameter are printed on the screen (with 2 input arguments)

Examples	`plSet(m);`
	`plSet(m, 'Timeout');`
	`plSet(m, 'Timeout', value);`
	`plSet(75122)`, `plSet(75122, 'Timeout');`
	`plSet(75122, 'Timeout', 1024);`
	`plSet(m, 'SubWindowSize', 'PCS2112_NO_DECIMATION', 480, 640);`

6.3 plDevices

plDevices is an internal function, which is called by the other functions.

DLL name	plDevices.dll
Syntax	`mexCallMATLAB(nlhs, *plhs[], nrhs, *prhs[], "plDevices");`
Description	plDevices maintains the 'open device array', in which the device ID's (handles to open devices) are stored. Also some settings, which cannot be stored in the PixeLINK Camera API, are stored in plDevices.
Input	prhs[0] = (string) name of task to be performed: print, remove, isopen, get, add, getpar, setpar prhs[1] = (U32) serial number of device; Only for remove, isopen, get, add, getpar, setpar prhs[2] = (int) deviceID; Only for add (string) name of the parameter; Only for getpar, setpar prhs[3] = value of the parameter; Only for setpar
Output	plhs[0] = (double) 1 if device open, 0 if closed; Only for isopen (int) deviceID if open, -1 if closed; Only for get parameter value, -1 if not found; Only for getpar

6.3.1 The open device array

plDevices maintains the 'open device array'. This is an array of structures of type:

```
struct {U32 serialNumber;
        int deviceID;
        U32 grabColorConversion;
        int grabOutputType;}
```

In this array the device ID (handle to the device) and the parameters `GrabColorConversion` and `GrabOutputType` are stored for each open device. The serial number of the device is used to uniquely identify the device. This is the number returned by PimMegaGetSerialNumber, which is called by plOpen immediately after initialising the device.

6.3.2 MEX function

plDevices is implemented as a MEX-function, because of two reasons. First this makes it possible to use the function mexLock() to keep the file in memory, even after typing 'clear all' or 'clear mex'. This also eliminates the need of having to do the memory management ourselves. The other reason is that, for testing other functions, it is convenient to be able to call plDevices directly from Matlab. The users of plFGI however should not use this feature.

6.3.3 Functions of plDevices

The plFGI functions can call plDevices to manipulate the 'open device array', to check whether a device has been opened and to get the settings stored in the 'open device array'. For this, plDevices provides the following functions:

- **print** prints a list of open devices, with SerialNumber, DeviceID, GrabColorConversion and GrabOutputType.
- **remove** removes a device from the 'open device array'.
- **isopen** checks whether a device has been opened.
- **get** returns the device ID (handle to the device) of a camera..
- **add** adds a device to the 'open device array'.
- **getpar** returns the value of the specified parameter.
- **setpar** sets the value of the specified parameter.

Although plDevices is not meant to be called directly from Matlab, in the next part the syntaxes and examples are written down as if it were, because this way it is easier to denote and understand the usage of the left-hand and right-hand side arguments.

- ***print***

Syntax	`plDevices('print')`
Description	prints a list of open devices, with SerialNumber, DeviceID, GrabColorConversion and GrabOutputType.
Input	prhs[0] = (string) 'print'
Output	None
Example	`plDevices('print')`

- ***remove***

Syntax	`plDevices('remove', serialnumber)`
Description	removes a device from the 'open device array'.
Input	prhs[0] = (string) 'remove' prhs[1] = (U32) serialnumber
Output	None
Example	`plDevices('remove', 75122)`

- ***isopen***

Syntax	`plDevices('isopen', serialnumber)`
Description	checks whether a device has been opened.
Input	prhs[0] = (string) 'isopen' prhs[1] = (U32) serialnumber
Output	plhs[0] = (double) 1 if the device is open 0 if the device is closed
Example	`plDevices('isopen', 75122)`

- ***get***

Syntax	`deviceId = plDevices('get', serialnumber)`
Description	returns the device ID (handle to the device) of a camera..
Input	prhs[0] = (string) 'get' prhs[1] = (U32) serialnumber
Output	plhs[0] = (int) deviceID if the device is open -1 if the device is closed
Example	`plDevices('remove', 75122)`

- ***add***

Syntax	`plDevices('add', serialnumber, deviceID)`
Description	adds a device to the 'open device array'.
Input	prhs[0] = (string) 'add' prhs[1] = (U32) serialnumber prhs[2] = (int) deviceID
Output	None
Example	`plDevices('add', 75122, 256085040)`

- ***getpar***

Syntax	`parameterValue = plDevices('getpar', serialnumber, parametername)`
Description	returns the value of the specified parameter.
Input	prhs[0] = (string) 'getpar' prhs[1] = (U32) serialnumber prhs[2] = (string) parametername
Output	plhs[0] = value of the specified parameter
Example	`parameterValue = plDevices('getpar', 75122, 'GrabOutputType')`

- ***setpar***

Syntax	`plDevices('setpar', serialnumber, parametername, parametervalue)`
Description	sets the value of the specified parameter.
Input	prhs[0] = (string) 'setpar' prhs[1] = (U32) serialnumber prhs[2] = (string) parametername prhs[3] = value of the specified parameter
Output	None
Example	`plDevices('setpar', 75122, 'GrabOutputType', RAW)` (in which case RAW is defined as 0x0)

6.4 Detailed description of the plFGI functions

In this paragraph, we give a detailed description of all plFGI functions. First we describe the main functions, and then all internal functions are treated. The complete source files for all functions can be found in Appendix D.

6.4.1 plClose

Description: plClose closes a PixeLINK camera device.

plClose is the MEX-function which takes care of closing an opened PixeLINK device. First it is tested whether one and only one argument was given, using 'if (nrhs == 1)'. If not, the else part is executed, printing an error message with mexPrintf and exiting with mexErrMsgTxt. If the argument is a device handle structure, the device's serial number is extracted from it. If the argument is a number, this is treated as the serial number. If the argument is something else, again an error message will be shown and the function will exit.

Now, it can be assumed that a serial number is present. It is checked whether the device is registered as open, with a call to plDevices. If it is not open, the else-part of the 'if (isOpen)'-construction is executed and the function terminates with an error message. If it is open, again a call to plDevices is made, this time to obtain the PixeLINK API's device-ID. Then the PixeLINK API function 'pimMegaUninitialize' is made to close the device. When an error occurs closing the device, the standard error handling code in plError returns a non-false value and the plClose function exits. If all went well, the device should now be closed and a call to plDevices is made, to update the list of open devices.

6.4.2 plGet

Description: plGet returns the device's up-to-date handle if 1 argument is given, or returns the value of the given parameter if 2 arguments are given.

plGet is the MEX-function which is used for obtaining the value of a parameter, a set of parameters, or a complete device handle for a PixeLINK device. First, it is tested whether 1 or 2 arguments were given. If not, the else part of the check is executed, resulting in an error message and the termination of plGet by using the Matlab call mexErrMsgTxt. The first argument is either a serial number, or a device handle structure. If it is a double, it is stored in the serialNumber variable. If it is a struct containing a field with the name 'SerialNumber', the value of this field is stored in the serialNumber variable (without further checking). If the first argument is another data type, an error message is given and plGet is terminated.

After the serial number has been obtained, it is checked whether the device is open by calling plDevices. If not, an error message is given and plGet is terminated. If the device is open, it is determined whether one or two arguments were given when calling plGet. If only one argument was given, a call to the plCreateDeviceHandle subroutine is made and the result of that is returned. If two arguments were given, it is checked whether the second argument is a string. If it is not, an error message is given and plGet is terminated. If it is, the name of the parameter is copied into a string variable and the subroutine plGetValue is called and the result of that is returned.

6.4.3 plGrab

Description: plGrab grabs a frame from the PixeLINK camera device and places it into an mxArray.

plGrab is the MEX-function that grabs a frame from the PixeLINK device and takes care of the required postprocessing before delivering it to Matlab. Currently only one calling syntax is supported, so it is checked whether there is exactly one input argument and one output argument. If not, the else part of the check is executed, resulting in an error message and the termination of plGrab by using the Matlab call mexErrMsgTxt. The input argument is either a serial number, or a device handle structure. If it is a double, it is stored in the serialNumber variable. If it is a struct containing a field with the name 'SerialNumber', the value of this field is stored in the serialNumber variable (without further checking). If the first argument is another data type, an error message is given and plGrab is terminated.

The serial number is now used to obtain a PixeLINK API device ID by calling plDevices. Using PimMegaStartVideoStream, the video stream is opened. Behind every call to a PixeLINK API function, plError is used to check for an error. If an error occurs while the videostream is supposed to be active, PimMegaStopVideoStream is called before terminating.

Now, it is checked whether the camera is in 'video mode', if it is not, a warning is printed to the screen and the camera is set to video mode. Video mode is used because still mode requires special lighting conditions or a shutter to control the exposure time[3]. After this the program gets some camera parameters and settings:

- get imagerType (colour or monochrome)
- get dataTransferSize (8 bit or 16 bit format)
- get decimation, width and height of the current subwindow[4]
- calculate pixelWidth, pixelHeight according to the rules in the PixeLINK Megapixel FireWire Camera Developer's Manual[5]
- get GrabColorConversion and GrabOutputType parameters using plDevices

Using the above parameters, 3 arrays must set up:

- The capture array, in which the PixeLINK API stores the raw image returned by the camera. This array has the same number of elements (either 8 or 16 bit unsigned integers) as the image has pixels (pixelWidth * pixelHeight) and is arranged as an mxArray of [pixelWidth, pixelHeight].
- The matlab array, which is returned to Matlab. In case of a RAW image, this array has the same size and arrangement as the capture array. In case of a 'Matlab IMAGE', it is an array of unsigned 8 bit integers three times the size of (pixelWidth * pixelHeight). It is arranged as an mxArray of [pixelHeight, pixelWidth, 3]; In memory, this means that 3 column-major arrays of [pixelHeight, pixelWidth] are stored consecutively (like this: RRR....GGG....BBB....).
- The RGB24 array is used as an intermediate array to store the result of PimMegaConvert in case of a non-RAW image, before it is converted to fit into the matlab array. Its size is always (3 * pixelWidth * pixelHeight) and it is arranged as a normal C-type row-major array (like this: RGBRGBRGB....).

[3] [Vitana, 2002-1], §1.5.5
[4] [Vitana, 2002-1], p.92, 'PimMegaSetSubWindow'
[5] [Vitana, 2002-1], p.92, 'PimMegaSetSubWindow'

First, a distinction between the various 'GrabOutputType' settings is made, using a case construction. The size and dim variables are set to the required size and dimensions of the matlab array. After that, the sizes of the capture array and the RGB24 array are defined. A case construction is used to allocate memory for the matlab and capture arrays, differentiating between 8 and 16 bits data size. After this, it is checked whether an error occurred, if so the program is terminated. If everything went all right, the pointer to the matlab array is set. When needed, memory is allocated for the RGB24 array.

Now, the image is captured by making a call to PimMegaReturnVideoData and the videostream is closed using PimMegaStopVideoStream.
For postprocessing, two possibilities are distinguished using an if/else construction.

- The first possibility occurs in case of a RAW image: An 8 bit RAW image is copied into the matlab array using memcpy, a 16 bit RAW image is converted to have all 10 significant bits in the right order, according to the PixeLINK Megapixel FireWire Camera User's Manual[6]. The 2 most significant bits of the first byte, are shifted to be the 2 least significant bits. The second byte is shifted 2 bits to the left (multiplied by 4), then the two bytes are added (equivalent to a logical 'and' operation).
- The second possibility occurs in case of an IMAGE or RGB24 output. Depending on 8 versus 16 bits and black/white versus colour, one of the PimMegaConvert* functions is called to convert the captured image to a 24 bits per pixel image. For colour images, the GrabColorConversion parameter is used to determine the algorithm to be used by the PimMegaConvert* function. The result of PimMegaConvert* is stored into the RGB24 array. If the GrabOutputType is set to RGB24, the RGB24 array is directly copied into the matlab array, using memcpy. In case of an IMAGE type, however, the RGB24 array must be transposed in order for matlab to display it properly. This is done using two nested for loops. Within the for-loops the bytes for R, G and B are copied to the appropriate locations in the matlab array.

All above mentioned case and if constructions contain a default case resulting in an appropriate error message when the distinguishing variable is not recognised.

- Suggestions for improvement:
  - Implementation of the 5 other ways to call plGrab:
    - moviematrix = plGrab(handle, imgmatrix)
    - [imgmatrix, moviematrix] = plGrab(handle)
    - plGrab(handle, imgmatrix)
    - plGrab(handle, imgmatrix, moviematrix)
    - plGrab(handle, imgmatrix, moviematrix, scaling factor)
  - Addition of RGB48 format for 16-bit captures
  - Testing the plGrab function with a colour camera (this should work)
  - Loop optimalisations
  - Capture a RAW image directly into the matlab array.

6.4.4 plIsOpen

Description: plIsOpen checks whether the device has been opened. It returns 0 if the device has not been opened, and 1 if it has been opened.

plIsOpen is the MEX-function for checking if certain PixeLINK device is open.

[6] [Vitana 2002-2], Appendix C

First, it is tested whether only 1 argument was given. If not, the else part of the check is executed, resulting in an error message and the termination of plIsOpen by using the Matlab call mexErrMsgTxt. The argument is either a serial number, or a device handle structure. If it is a double, it is stored in the serialNumber variable. If it is a struct containing a field with the name 'SerialNumber', the value of this field is stored in the serialNumber variable (without further checking). If the first argument is another data type, an error message is given and plGet is terminated.

After the serial number has been obtained, it is checked whether the device is open by calling plDevices. The result of the call to plDevices is stored in the return array of plIsOpen.

6.4.5 plOpen

Description: plOpen opens a PixeLINK camera device and returns the device's handle. If there is no camera with the given serial number, a list of available serial numbers is printed.

plOpen is the MEX-function which takes care of opening a PixeLINK device for use with the other plFGI functions. First it is tested whether the input argument is a number, if it is not, the function terminates printing an appropriate error message using mexErrMsgTxt. Then it is checked whether the device is open by calling plDevices. If the device is already open, the else part of the if-construction is executed printing a warning message, then calling plCreateDeviceHandle and returning the handle.

If the device was not open, a call to PimMegaGetNumberDevices is made to get the amount of available devices. When no devices are attached, plOpen exits with an appropriate error message. When one or more devices are attached, a for-loop is entered to open every device, store the serial number in the array of available serial numbers, check if the serial number matches the requested serial number, then close it and loop. If the serial number matches the requested serial number, the variable found is set to true and the loop is exited by using a break statement.

If found is not true, the array containing all available serial numbers is printed to the screen and the function is terminated using mexErrMsgTxt.

Finally, the open device array is updated by calling plDevices.

- Suggestions for improvement: Use plError after the call to PimMegaGetNumberDevices instead of non standard error code.

6.4.6 plSet

Description: plSet sets the value of a given parameter. When 1 argument is given a list of all possible parameters is given. When 2 arguments are given, the possible values of the given parameters are given. When called with 3 or more arguments, the device's parameter given in the second argument will be set to the value(s) given in the other argument(s).

plSet is the MEX-function used to change the parameter settings of a PixeLINK device. First, it is checked whether any arguments are given, if not, an appropriate error message is printed to the screen using mexPrintf, and the plSet is terminated using mexErrMsgTxt.

If one or more arguments were given, the first argument should be either a handle structure or a serial number. If it is a double, it is stored in the serialNumber variable. If it is a struct

containing a field with the name 'SerialNumber', the value of this field is stored in the serialNumber variable (without further checking). If the first argument is another data type, an error message is given and plGet is terminated. After the serial number has been obtained, it is checked whether the device is open by calling plDevices. If not, an error message is given and plGet is terminated.

If only one argument was given, a list of parameter names is printed on the screen and plSet terminates. If a second argument was given, it is checked whether this is a string. If not plSet is terminated with an appropriate error message. If it is a string, it is stored in the parametername variable. Depending on whether 2 or 3 arguments were given, plPrintPossiblevalues is called (2 parameters), or plSetValue is called (3 arguments).

- Suggestions for improvement:
  - Implementation of PimMegaSetOverlayCallBack
  - Implementation of PimMegaSetPreviewWindow
  - Implementation of PimMegaAutoExposure

6.4.7 plDevices

Description: plDevices maintains the 'open device array', in which the device ID's (handles to open devices) are stored. Also some settings, which are not stored in the PixeLINK Camera API itself, are stored in plDevices.

Since a rather detailed description of plDevices is given in paragraph 6.3 already, here we only added the additional information on the internal working of the function.

For every open device the following information is stored in the open device array: The device's serial number, the device's ID which is used by the PixeLINK API to identify the device, the GrabColorConversion parameter and the GrabOutputType parameter. Another global variable, deviceCount, is incremented every time a device is opened and decremented every time a device is closed, thus counting the number of open devices.

First mexLock is called, to make sure plDevices (containing the open device array) stays in memory. Then it is checked whether any arguments are given and if the first argument is a text string. If not, the error message in the else part of the check is displayed and plDevices terminates. If it is, the first argument is stored in the task variable. If there is a second argument, it should always be the serial number of a device. If it is not a number, plDevices is terminated with an error message.

A case construction is used to make a first selection between the different tasks based on the number of arguments given. For every number of arguments from 1 to 4, a different case exists; the default case gives an error message. Inside a case, the different tasks are selected using an if/else if construction together with strcmp. When the strcmp fails, an appropriate error message is given and plDevices terminates.

- ***1 argument***

print This task is mainly intended for testing and debugging purposes: It prints all entries of the open device array to the screen, one device per line, by using a for loop (for i = 0 to deviceCount - 1).

- ***2 arguments***

remove This task removes a device from the open device list. Using a for-loop (for i = 0 to deviceCount - 1), the given serial number is compared to every serial number in the open device array. As soon as a matching entry is found, a new for loop is entered (for t = <number of matching entry> to deviceCount - 1) to overwrite every entry, from the matching entry up to the last entry, with the entry immediately following it (this goes wrong if the last entry is 31; see 'Suggestions for improvement'). After the inner for loop finishes, deviceCount is decremented.

isopen This task returns 0 if the device is not open, 1 if it is open. First the return value is set to 0, then a for-loop (for i = 0 to deviceCount - 1) is entered. The given serial number is compared to every serial number in the open device array. As soon as a matching entry is found, the return value is set to 1, and the loop is left using a break statement.

get This task returns the device ID for a given serial number. It works the same way as isopen, except that the return value is set to -1, and as soon as a matching entry is found, the return value is set to the appropriate device ID.

- ***3 arguments***

add This task adds a new device to the open device array. First, it is checked whether there is still room in the array to store the information of one more open device. If the array is full, an error message is printed and plDevices is terminated. If there still is room, the entry after the last entry of the open device array (index: deviceCount, because array indexing starts at deviceCount - 1) is filled with the device ID given in the third argument, and suitable default values for the parameters. After that, deviceCount is incremented.

getpar This task reads the value of the given device parameter from the open device array. First, the third argument, which should be a string with the name of the requested parameter, is loaded into the parName variable, then the return value (which should be a double) is initialised. It is determined which parameter should be returned by using strcmp in an if/then/else if/else construction. The procedure is the same for every parameter: A temporary boolean variable t is used to indicate whether a matching serial number was not found. It is set to 1, then a (for i = 0 to deviceCount - 1) loop is entered. As soon as a matching serial number is found, the requested parameter is stored as return value, t is set to 0, and the for loop is terminated using a break statement. Finally, if t still has value 1, an error message is printed stating that the requested serial number could not be found.

- ***4 arguments***

setpar This task writes the given value of the given device parameter in the open device array. Its internal structure is almost the same as that of getpar, but the parameter value is written, not read.

- Suggestions for improvement: In 'remove', the inner for loop tries to copy one entry more than needed. When having 32 camera's open, this could result in a segfault. The inner loop for condition should be: (t = i; t < deviceCount - 1; t++)

6.4.8 plCreateDeviceHandle

plCreateDeviceHandle is a subroutine which is linked to the plOpen and plGet MEX-functions. When given the serial number of a PixeLINK device, it will create a Matlab struct matrix containing all the device's parameter names with their respective values. This is the so-called handle structure, which is returned by plOpen and plGet.

After declaration of the necessary variables and structures, the device-ID for calling the PixeLINK API functions is obtained by calling plDevices. Then, for every parameter, the appropriate API call is made to obtain its value. If the API call returns the value for unsupported function, the string 'Unsupported' is stored as parameter value. If the API returns any other error code, the string 'Could not get value' is stored as parameter value. No other error checking is being done. If everything went okay, the value of the requested parameter is stored in a temporary variable by the PixeLINK API and immediately after that it is stored into the handle structure using mxSetField.

The only exception to this are the parameters 'GrabColorConversion' and 'GrabOutputType', which are not stored in the PixeLINK API, but in plDevices. They are obtained in a similar way to the device-ID by calling plDevices using mexCallMatlab.

- Suggestions for improvement: Convert some of the if-constructions to switch constructions (for example the one used for SubWindowSize).

6.4.9 plError

plError is an error-checking function which is linked to every MEX-function that uses PixeLINK API calls. It is called after every call to such an API function. Required inputs are the PixeLINK API return-code returned by the called API function and a string describing in max. 33 characters what the program was doing when the possible error occurred.

First, the string input of the plError function is used to create an error message 'The device's API encountered a problem while <string>:'. When the PixeLINK API return code (which is defined in PimMegaApiUser.h) indicates an error, this string is printed on the screen.

After that, a case construction is used to determine the nature of the PixeLINK API return code. When everything is okay, plError returns a value of 0 (false). When an error occurred, plError prints an appropriate descriptive message on the screen using mexPrintf and returns a non-zero value (true). To make it possible to distinguish between critical and non-critical errors, a value of 1 is returned on an error that might not be critical. A value of 2 is returned on errors that are always considered critical. Where appropriate, the program can decide on this information to continue or to terminate.

- Suggestions for improvement: The entire building of the string used in 'The device's API encountered...' can be moved within the following 'if (result != ApiSucces)' condition.

6.4.10 plGetValue

plGetValue is a subroutine which is linked to the plGet MEX-function. When given an mxArray for returning the parameter value, the device's serial number and the name of the parameter to be returned, it will return the parameter's value or values in the given mxArray.

plGetValue is very similar to plCreateDeviceHandle. The most important difference is that not all possible parameters of the device are obtained, but only the parameter or structure of which the name was given. This is accomplished by using an if/elseif construction together with multiple strcmp statements. An obvious difference is that the mxArray m can differ in size according to the parameter that is stored in it.

When an unknown parameter name is given, plGetValue prints a list of known parameter names to the screen.

6.4.11 plSetValue

plSetValue is a subroutine linked to the plSet MEX-function. It needs 4 or more input arguments: the serial number of the device, the name of the parameter to be set, the number of values that are passed, all values that are needed to set the parameter.

First, the serial number is used to obtain the PixeLINK device-ID by calling plDevices. For every parameter name, it is checked using an if/strcmp construction whether the right number and type of values are given. If not, a descriptive error message is printed and plSetValue is terminated using a call to mexErrMsgTxt.

After this, another if/elseif/strcmp construction is used to distinguish between the different parameter names. For every parameter name, the passed values are copied into a variable, parsed if necessary and a call is made to the relevant PimMegaSet* function. For the 'GrabColorConversion' and the 'GrabOutputType' parameters, a call is made to plDevices as these are stored there instead of in the PixeLINK API.

- Errors found in the PixeLINK API documentation:

[Vitana 2002-1], p.92, 'PimMegaSetSubWindow'

(uStartColumn + uNumberColumns) must be less than PCS2112_MAX_WIDTH.
(uStartRow + uNumberRows) must be less than PCS2112_MAX_HEIGHT.

'less than' should be 'less than or equal to'.

--

[Vitana 2002-1], p.94, 'PimMegaSetSubWindowPos'

(uStartColumn + current number of columns) must be less than PCS2112_MAX_WIDTH.
(uStartRow + current number of rows) must be less than PCS2112_MAX_HEIGHT.

'less than' should be 'less than or equal to'.

--

[Vitana 2002-1], p.95, 'PimMegaSetSubWindowSize'

(Current start column + Width) must be less than PCS2112_MAX_WIDTH.
(Current start row + Height) must be less than PCS2112_MAX_HEIGHT.

'less than' should be 'less than or equal to'.

6.4.12 plPrintPossibleValues

plPrintPossibleValues is a subroutine linked to the plGet and plSet MEX-functions. When given the name (string) of a parameter, it will use strcmp in an if/else if construction to determine what should be printed. Then a short description of the given parametername and its possible values is printed to the screen using mexPrintf.

6.4.13 plTypes

plTypes.h contains all definitions that are used by the plFGI program. Currently, the only values that are defined here are those used to store the 'GrabOutputType' variable.

7 Programming environment

This chapter describes the programming environment, which was used to create the updated FGI system and plFGI. We used two different ways to build the MEX-files for the FGI system and for plFGI.

7.1 Updated FGI system

The updated FGI system was built in the same way as Ronald Heil and Chris Wauters did. The Microsoft Visual C++ editor was only used for syntax highlighting. For compiling and linking of the MEX-files we used their M-file 'buildall.m'. For more information on this, we refer to 'Matlab Interface For Data Translation Mach Frame grabbers - User Manual & Technical Guide'[7]. We used Matlab version 5.3.1, with the 'Microsoft 6.0 compiler in C:\Program Files\Microsoft Visual Studio'.

7.2 plFGI

For creating plFGI, we used Microsoft Visual C++ 6.0, for writing the source files, as well as for compiling and linking. We used the following procedure to create a project with the correct settings:

- Create a new DLL project (File → New → Projects → Win32 Dynamic-Link Library → A DLL that exports some symbols)
- Throw away StdAfx.cpp, StdAfx.h and Projectname.h
- Disable precompiled headers (Project → Settings → C/C++ → Category Precompiled Headers → Not using precompiled headers)
- Link the necessary libraries (Project → Settings → Link → Category General → Object/library modules):
  - Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmex.lib
  - Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmx.lib
  - Y:\software\framegrag\pixelink\api\pimmegaapi.lib
- Export the mexFunction (Project → Settings → Link → Category General → Project Options): `/export:mexFunction`
- Implement Projectname.cpp, starting with the following template:

```
#define WIN32_LEAN_AND_MEAN
/* This excludes rarely-used stuff from Windows headers. We don't
 * know what that means exactly, but it is generated by MS Visual C++
 * when you create a project and we just copied it.
 */

#include <windows.h>

#include "Y:\soft95\matlab6\extern\include\mex.h"
/* with Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmx.lib
 * and Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmex.lib
 */
```

[7] [Oei/Kap., 1998], p.28-29

```
#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"
/* with Y:\software\framegrag\pixelink\api\pimmegaapi.lib */

#ifdef __cplusplus
    extern "C" {
#endif

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    /* implement the MEX function here */
}

#ifdef __cplusplus
    } /* extern "C" */
#endif
```

Code example 2 ***Template for a plFGI MEX-function***

We used the libraries of Matlab version 6.1 and the PixeLINK Camera API Release 3.0.

The MEX-files are implemented as C++ files (instead of C), because the PixeLINK Camera API is implemented in C++ and it uses language constructions that cannot be used with C files. With the `extern "C"` command, the mexFunction is exported as a C function, so it will be possible to call the function from within Matlab, just like normal C MEX-files.

The PixeLINK dll is load-time linked. This means that when the MEX-file is loaded into memory, the PixeLINK dll is loaded at the same time. Therefore, to use the MEX-files, it is necessary that the file 'PimMegaApi.dll' is in the *Windows*-path (not the Matlab-path). Otherwise Matlab will give an error-message like:

```
Unable to load mex file: g:\fgi\bin\plOpen.dll.
The specified module could not be found.

??? Invalid MEX-file
```

For us this caused some confusion, because we thought that there was something wrong with our code. But instead the error-message was generated because the file 'PimMegaApi.dll' could not be found.

7.3 Testing

During development, frequent testing helped us to find bugs in plFGI. After the program reached a stable working state, we have run a few other tests in the limited time that was still available.

On suggestion of Assistant Professor Smutný, it was tested how much time the capturing of multiple frames takes. This could possibly be used as an indication of the quality of the code (memory leaks and resources that are not released would have become apparent this way). Also, the performance of the code, the PixeLINK API and the PixeLINK camera could be verified.

First, some basic tests were performed, showing that capturing the images resulted in a frame rate a bit lower than that of the demo application, but that could be contributed to the

overhead of calling functions from Matlab, and the lack of loop optimalisation in plFGI. When testing different resolutions, the effect on overall capturing speed of the camera itself became apparent: reducing the number of columns in the picture did not affect the capture time very much, as the camera scans row by row. Reducing the number of rows reduced the time needed for capturing accordingly. As a last experiment, the exposure time was variated. Strangely enough, capturing 100 images would take less time than 100 times the exposure time.

As we did not have much time to take a closer look at these tests and do some more advanced testing, this might be a good subject for future examination.

In Appendix E, the test results are included.

Conclusion

For eight weeks we have worked on this project. In the beginning we both had no experience with Matlab, frame grabbers or the C programming language. So we first spent a few days reading documentation and studying the existing FGI. Very soon we were able to fix the 'clear all bug'. The proper release of the device driver's resources, which we were asked to do later during our project, was not very difficult either.

Then we made the original design of the new system. We implemented most of the command layer and control layer. When we started implementing the device specific layer, we encountered several problems, concerning dll's. Neither of us had ever worked with dll's before, so we had to figure out how to load them, as well as how to build them. We spent a lot of time on this, but in the end we succeeded.

When studying the PixeLINK API, we discovered that it differs more from the DT Frame Grabber SDK than we thought originally. Since we did not see a neat solution to deal with this problem, we decided in the end not to implement our original design. Instead we implemented plFGI in a way similar to the implementation of the original FGI system.

So now we have the following results:

The original FGI system is updated. The 'clear all bug' is fixed and the device driver's resources are released properly. More descriptive error messages are added and the documentation of grabfg is improved. The 'polishing' problem still remains unsolved though.

For the PixeLINK cameras plFGI is implemented. It supports most basic functionality: to open and close a device, check whether a device is opened, get and set a number of parameters and grab an image from the camera. Unlike the original FGI system, it only provides 1 way to call plGrab (instead of the 6 ways to call grabfg). Also it has no reset-function, because the PixeLINK API does not provide this.

We made a full list of topics for improvement, which is included in Appendix B.

We both enjoyed working on this project very much. For both of us this was the first time to put into practice everything we had learned during our studies, and it is nice to see that we were able to finish this project successfully.

It is a pity that it took us so long to find out how to work with dll's. We might have been able to deliver a program with more functionality, if that had not taken us so much time. Also it might have been a good idea to start working on the device specific layer first (instead of the command and control layer), since that part involved most of the things unknown to us. Programming the command and control layer was a rather straightforward task, the effort of which was easier to estimate. If we had started with the device specific layer, we would have discovered earlier that our original design would not work out. This would have saved us the time of implementing the command and control layer, which are unused in the end.

Working on this project is not the only thing we enjoyed during our stay here. We also had a lovely time in a different country, during which we learned a lot about the Czech culture and especially the beautiful city of Prague.

Table of figures

Bibliography

[Ammeraal, 1993] AMMERAAL, LEEN (1993). *Ansi C*. Academic Service B.V., Schoonhoven.

[DataTrans., 1996] DATA TRANSLATION (March 1996). *Frame Grabber SDK™*. Data Translation, Inc., Marlboro.

[Heil/Waut., 2000] HEIL, RONALD & WAUTERS, CHRIS (April-June 2000). *FGI Report 2000*. Czech Technical University, Prague & University of Technology, Delft.

[MathWorks, 1998-1] THE MATHWORKS INC. (September 1998). *Matlab Application Program Interface Reference, Version 5 (online version)*. The MathWorks, Inc.

[MathWorks, 1998-2] THE MATHWORKS INC. (October 1998). *Matlab Application Program Interface Guide, Version 5 (online version)*. The MathWorks, Inc.

[Oei/Kap., 1998] OEI, D.J. & KAPOERCHAN, A. (1998). Modified by HEIL, R. & WAUTERS, C.L. (June 2000). *Matlab Interface For Data Translation Mach Frame grabbers - User Manual & Technical Guide*. Czech Technical University, Czech Republic.

[Planz, 1989] PLANZ, ALAN C. (1989). *De kleine C gids*. Academic Service, Schoonhoven.

[Vitana, 2002-1] VITANA CORPORATION (2000-2002). *PixeLINK Megapixel FireWire Camera Developer's Manual, Release 3.0 (online version)*. Vitana Corporation, Ottawa, Ontario, Canada.

[Vitana, 2002-2] VITANA CORPORATION (2000-2002). *PixeLINK Megapixel FireWire Camera User's Manual, Release 3.0 (online version)*. Vitana Corporation, Ottawa, Ontario, Canada.

Appendix A. List of updates

These are all updates made to FGI in June-August 2002:

- Added a call to *mexLock* in mainfg, so the handles to open devices will survive the '`clear all`' and '`clear mex`' commands.
- Added the function *destroyFrameOnErrorExit* in grabfg, to properly release the device driver's resources.
- Changed the layout of grabfg to improve readability.
- Added more descriptive error messages to grabfg.
- Improved the documentation of grabfg, so now all 6 ways of calling grabfg are documented.

Appendix B. Topics for improvement

This is a full list of all topics for improvement on the DataTranslation and PixeLINK Frame Grabber Interfaces:

B.1 DataTranslation FGI

- Polishing: show the settings for device specific functions in the device information structure

B.2 PixeLINK FGI

- plGrab: Implementation of the 5 other ways to call plGrab:
  - moviematrix = plGrab(handle, imgmatrix)
  - [imgmatrix, moviematrix] = plGrab(handle)
  - plGrab(handle, imgmatrix)
  - plGrab(handle, imgmatrix, moviematrix)
  - plGrab(handle, imgmatrix, moviematrix, scaling factor)
- plGrab: Addition of RGB48 format for 16-bit captures
- plGrab: Testing the plGrab function with a colour camera (this should work)
- plGrab: Loop optimalisations
- plGrab: Capture a RAW image directly into the left hand array (*plhs[]) of the mexFunction
- plOpen: Use plError after the call to PimMegaGetNumberDevices instead of non standard error code
- plSet: Implementation of PimMegaSetOverlayCallBack
- plSet: Implementation of PimMegaSetPreviewWindow
- plSet: Implementation of PimMegaAutoExposure
- plDevices: In ‘remove’, the inner for loop tries to copy one entry more than needed. When having 32 camera’s open, this could result in a segfault. The inner loop for condition should be: (t = i; t < deviceCount - 1; t++)
- plCreateDeviceHandle: Convert some of the if-constructions to switch constructions (for example the one used for SubWindowSize)
- plError: The entire building of the string used in ‘The device’s API encountered...’ can be moved within the following ‘if (result != ApiSucces)’ condition.
- Function to check whether the camera is connected to the computer
- Reset function, to set all parameters to their default value
- Demo M-files
- Additional tests on the performance of plFGI

Appendix C. Updated source code of FGI 2000

This is a part of the source of grabfg, to show the changes to FGI 2000, which are described in paragraphs 4.3 (Proper release of the device driver's resources) and 4.4 (Other changes to the original FGI).

```
/*********************************************************************/
/** Filename: grabfg.c                                              **/
/** Authors: D.J. Oei, A. Kapoerchan                                **/
/** Date: 10 July 1998                                              **/
/** Last update by original authors: 17 August 1998                 **/
/** Modified by: R.Heil, C.L.Wauters 17 April 2000                  **/
/** Last update by R.Heil, C.L.Wauters: 01 June 2000                **/
/**                                                                 **/
/** Modified by: L.I.Oei, M.A.E.Bakker                              **/
/** Updates: source cleaned up and properly indented 2002/07/02     **/
/**          destroy frame on errorexit 2002/07/03                  **/
/**          improved error messages after acquire frame 2002/07/04 **/
/**                                                                 **/
/*********************************************************************/
/** This C-file contains the source-code to grab an image.          **/
/*********************************************************************/
/** Input:  device information structure (devinfo)                  **/
/** Output: Matlab mxArray (imgmatrix)                              **/
/** Syntax: imgmatrix = grabfg(devinfo)                             **/
/**         moviematrix = grabfg(devinfo, imgmatrix)                **/
/**         [imgmatrix, moviematrix] = grabfg(devinfo)              **/
/**         grabfg(devinfo, imgmatrix)                              **/
/**         grabfg(devinfo, imgmatrix, moviematrix)                 **/
/**         grabfg(devinfo, imgmatrix, moviematrix, scaling factor) **/
/*********************************************************************/


#include <mex.h>
#include <stdlib.h>
#include "windows.h"
#include "olwintyp.h"
#include "olimgapi.h"
#include "olfgapi.h"
#include "Dtcolors.h"


/* Added by M.Bakker 2002/07/03 18:25
 * this subroutine calls OlFgDestroyFrame to release resources that were not
 * properly released when another error was encountered during the grabbing.
 */
void destroyFrameOnErrorExit(OLT_IMG_DEV_ID DeviceId, OLT_FG_FRAME_ID FrameId)
{
    OLT_APISTATUS result;
    result = OlFgDestroyFrame(DeviceId, FrameId);
    switch (result)
    {
        case OLC_STS_NORMAL:            break;
        case OLC_STS_BUSY:             mexPrintf("Still busy with acquisition.\n");
                                        mexPrintf("Failed to destroy allocated \
                                          resources while recovering from the error \
                                           displayed below.\n");
                                        break;
        case OLC_STS_INVALIDFRAMEHANDLE: mexPrintf("Invalid frame handle.\n");
                                         mexPrintf("Failed to destroy allocated \
                                          resources while recovering from the error \
                                           displayed below.\n");
                                         break;
        case OLC_STS_FRAMENOTALLOCATED: mexPrintf("Frame not allocated.\n");
                                        mexPrintf("Failed to destroy allocated \
                                          resources while recovering from the error \
                                           displayed below.\n");
                                        break;
        case OLC_STS_NONOLMSG:          mexPrintf("Unsupported or unknown message \
                                           passed to the device driver.\n");
                                        mexPrintf("Failed to destroy allocated \
```

```
                                        resources while recovering from the error \
                                         displayed below.\n");
                                      break;
        case OLC_STS_FRAMEISMAPPED:   mexPrintf("Cannot destroy a frame that is \
                                         still mapped.\n");
                                      mexPrintf("Failed to destroy allocated \
                                       resources while recovering from the error \
                                        displayed below.\n");
                                      break;
        default:                      mexPrintf("Unknown error.\n");
                                      mexPrintf("Failed to destroy allocated \
                                       resources while recovering from the error \
                                        displayed below.\n");
                                      break;
    }

    return;
}

. . .
. . .
. . .

case OLC_STS_BUSY:            destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                FrameId); /* M.Bakker 2002/07/03 */
                              mexPrintf("Acquisition section of the device still \
                                busy.\n");
                              mexErrMsgTxt("Unable to grab an image.\n");
                              return;
case OLC_STS_SYSERROR:        destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                FrameId); /* M.Bakker 2002/07/03 */
                             mexPrintf("Unable to communicate with the device.\n");
                              mexErrMsgTxt("Unable to grab an image.\n");
                              return;
case OLC_STS_TIMEOUT:         destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                FrameId); /* M.Bakker 2002/07/03 */
                              mexPrintf("Timeout waiting for acquisition.\n");
                              mexErrMsgTxt("Unable to grab an image.\n");
                              return;
case OLC_STS_FIFO_OVERFLOW:   destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                FrameId); /* M.Bakker 2002/07/03 */
                              mexPrintf("Too much activity on host bus during \
                                transfer.\n");
                              mexErrMsgTxt("Unable to grab an image.\n");
                              return;
case OLC_STS_FIELD_OVERFLOW:  destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                FrameId); /* M.Bakker 2002/07/03 */
                              mexPrintf("Too much activity waiting on host bus \
                                during transfer.\n");
                              mexErrMsgTxt("Unable to grab an image.\n");
                              return;
case OLC_STS_INVALIDFRAMEHANDLE: destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                FrameId); /* M.Bakker 2002/07/03 */
                              mexPrintf("Invalid Framehandle.\n");
                              mexErrMsgTxt("Unable to grab an image.\n");
                              return;
case OLC_STS_CLAMP:           destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                FrameId); /* M.Bakker 2002/07/03 */
                              mexPrintf("Clamp start exceeds clamp end.\n");
                              mexErrMsgTxt("Unable to grab an image.\n");
                              return;
case OLC_STS_VERTICALINC:     destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                FrameId); /* M.Bakker 2002/07/03 */
                              mexPrintf("Vertical increment not equal to 1.\n");
                              mexErrMsgTxt("Unable to grab an image.\n");
                              return;
case OLC_STS_FIRSTACTPIX:     destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                FrameId); /* M.Bakker 2002/07/03 */
                             mexPrintf("First active pixel exceeds backporch start \
                                or clamp start.\n");
                              mexErrMsgTxt("Unable to grab an image.\n");
                              return;
case OLC_STS_ACTPIXCOUNT:     destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                FrameId); /* M.Bakker 2002/07/03 */
                             mexPrintf("Active line area is not between backporch \
                                start and total pixels per line.\n");
```

```
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_ACTLINECOUNT:      destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                               mexPrintf("Active line count exceeds total lines per \
                                  field.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_FRAMELEFT:        destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                                mexPrintf("Frame left exceeds active pixel count.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_RANGE:            destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                                mexPrintf("Difference between whitelevel and \
                                  blacklevel too large.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_FRAMETOP:         destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                                mexPrintf("Frame top exceeds active line count.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_FRAMEWIDTH:       destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                               mexPrintf("Frame width not multiple of 4 or exceeds \
                                  active pixel count.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_FRAMEHEIGHT:     destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                               mexPrintf("Frame height will cause digitalisation to \
                                  exceed active pixel count.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_HSYNCSEARCHPOS:   destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                                mexPrintf("H sync search position precedes H sync \
                                  insert position.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_VSYNCSEARCHPOS:   destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                                mexPrintf("V sync search position precedes V sync \
                                  insert position.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_INTERLACEDHGTGRAN: destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                               mexPrintf("Frame height granularity is illegal.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_INTERLACEDTOPGRAN: destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                                mexPrintf("Frame top granularity is illegal.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_WHITELEVEL:       destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                                mexPrintf("White level cannot be set.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_NONOLMSG:         destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                               mexPrintf("Unsupported or unknown message passed to \
                                  the device driver.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_MODECONFLICT:     destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                                mexPrintf("Mode conflict.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_BUFSIZ:           destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
```

```
                                mexPrintf("Buffersize is too small.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
case OLC_STS_NULL:              destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                                mexPrintf("Pointer supplied is NULL.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
default:                       destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, \
                                  FrameId); /* M.Bakker 2002/07/03 */
                                mexPrintf("Unknown error.\n");
                                mexErrMsgTxt("Unable to grab an image.\n");
                                return;
```

Appendix D. Source code of plFGI

D.1 plClose.cpp

```
/*********************************************************************/
/* Filename:    plClose.cpp                                          */
/* Description: Source code for the function plClose                 */
/* Authors:     M.A.E.Bakker, L.I.Oei                                */
/*              maarten@panic.et.tudelft.nl, l.i.oei@its.tudelft.nl  */
/* Date:        2002/08/07                                           */
/* Updates:                                                          */
/*********************************************************************/
/* plClose closes a pixelink device.                                 */
/*********************************************************************/
/* Input:   prhs[0] = (mxArray)    handle of the device              */
/*                    (U32)        serial number of the device       */
/* Output:  -                                                        */
/* Syntax:  plClose(handle) or plClose(serialnumber)                 */
/*********************************************************************/

#define WIN32_LEAN_AND_MEAN    /* Exclude rarely-used stuff from Windows headers */
#include <windows.h>

#include "Y:\soft95\matlab6\extern\include\mex.h"
/* with Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmex.lib */
/* and  Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmx.lib */

#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"
/* with Y:\software\framegrag\pixelink\api\pimmegaapi.lib */

#include "..\plError.h"

#ifdef __cplusplus
    extern "C" {
#endif

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    PXL_RETURN_CODE nRetValue;       /* the return codes of the pixelink functions */

    mxArray *serialNumberField;
    U32 serialNumber = -1;           /* the serial number of the device */

    HANDLE deviceId;                /* the deviceId PimMegaInitialize returns */

    mxArray *lhs[1], *rhs[2];        /* to call fgdevices */
    double *px;
    int isOpen = 0;

    /* prhs[0] should be the device handle (struct) or serial number (int) */
    if (nrhs==1)
    {
        /* get the serialNumber */
        if (mxIsStruct(prhs[0]))
        {
            serialNumberField = mxGetField(prhs[0], 0, "SerialNumber");
            serialNumber = (U32)mxGetScalar(serialNumberField);
        }
        else if (mxIsDouble(prhs[0]))
        {
            serialNumber = (U32)mxGetScalar(prhs[0]);
        }
        else
        {
            mexPrintf("PixeLINK Interface error.\n");
            mexPrintf("plClose: Wrong arguments given. One argument needed (device \
                       handle  or serial number).\n");
            mexErrMsgTxt("\n");
        }
```

```
        /* check whether the device has been opened */
        rhs[0] = mxCreateString("isopen");
        rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
        px = mxGetPr(rhs[1]);
        px[0] = (U32)serialNumber;

        mexCallMATLAB(1, lhs, 2, rhs, "plDevices");
        isOpen = (int)mxGetScalar(lhs[0]);

        if (isOpen)
        {
            /* get the deviceId */
            rhs[0] = mxCreateString("get");

            rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
            px = mxGetPr(rhs[1]);
            px[0] = (U32)serialNumber;

            mexCallMATLAB(1, lhs, 2, rhs, "plDevices");

            deviceId = (HANDLE *)(int)mxGetScalar(lhs[0]);

            /* uninitialize the device */
            nRetValue = PimMegaUninitialize(&deviceId);
            if (plError(nRetValue, "closing the device"))
            {
                mexErrMsgTxt("\n");
            }

            /* update the open device array in plDevices */
            rhs[0] = mxCreateString("remove");
            rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
            px = mxGetPr(rhs[1]);
            px[0] = (U32)serialNumber;
            mexCallMATLAB(1, lhs, 2, rhs, "plDevices");
        }
        else /* !isOpen */
        {
            mexPrintf("plClose: cannot close device with serial number %d.\n",\
                      serialNumber);
            mexPrintf("Device has not been opened.\n");
            mexErrMsgTxt("\n");
        }
    }
    else /* ! ((nrhs==1) && mxIsStruct(prhs[0])) */
    {
        mexPrintf("PixeLINK Interface error.\n");
        mexPrintf("plClose: Wrong arguments given. One argument needed (device \
                   handle or serial number).\n");
        mexErrMsgTxt("\n");
    }
}

#ifdef __cplusplus
    }    /* extern "C" */
#endif
```

D.2 plCreateDeviceHandle.cpp

```
/*********************************************************************/
/* Filename:    plCreateDeviceHandle.cpp                             */
/* Description: Source code for plCreateDeviceHandle                 */
/* Authors:     M.A.E.Bakker, L.I.Oei                                */
/*              maarten@panic.et.tudelft.nl, l.i.oei@its.tudelft.nl  */
/* Date:        2002/08/09                                           */
/* Updates:                                                          */
/*********************************************************************/
/* plCreateDeviceHandle creates the device handle (i.e. the structure*/
/* with settings), which has to be returned to the matlab user.      */
/*********************************************************************/
/* Input:  returnarray (mxArray)   mxArray for receiving the structure*/
/*         serialNumber (U32)      serial number of the device       */
/* Output: -                                                         */
/* Syntax: plCreateDeviceHandle(returnArray, serialNumber)           */
/*********************************************************************/

#include <windows.h>
#include "Y:\soft95\matlab6\extern\include\mex.h"
#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"

#include "plTypes.h"

void plCreateDeviceHandle(mxArray *returnArray[], U32 serialNumber)
{
    /* m = mxArray structure to be returned as device handle */
    /* lhs, rhs used to call plDevices */
    mxArray *m, *lhs[1], *rhs[3];
    double *px;
    HANDLE deviceId;

    /* variable for receiving the error codes the pixelink api functions return */
    PXL_RETURN_CODE nRetValue;

    int nof=40;    /* number of fields of the device handle structure */

    int rows=1;
    int cols=1;

    char *fieldnames[] =    /* names of the fields of the device handle structure */
    {
        "DeviceID",
        "BlueGain",             "CurrentFrameRate",         "DataTransferSize",
        "Exposure",             "ExposureTime",             "Gamma",
        "Gpo",                  "GreenGain",            "HardwareVersion.ProductID",
        "HardwareVersion.SerialNumber", "HardwareVersion.FirmwareVersion", \
                                                        "HardwareVersion.FpgaVersion",
        "ImagerChipId",         "ImagerClocking",           "ImagerName",
        "ImagerType",           "MonoGain",                 "PreviewWindowPos.Top",
        "PreviewWindowPos.Left", "PreviewWindowSize.Height", \
                                                          "PreviewWindowSize.Width",
        "RedGain",              "Saturation",               "SerialNumber",
        "SoftwareVersion",      "SubWindow.Decimation",     "SubWindow.StartRow",
        "SubWindow.StartColumn", "SubWindow.NumberRows",    "SubWindow.NumberColumns",
        "SubWindowPos.StartRow", "SubWindowPos.StartColumn", \
                                                         "SubWindowSize.Decimation",
        "SubWindowSize.Height", "SubWindowSize.Width",      "Timeout",
        "VideoMode",            "GrabColorConversion",      "GrabOutputType"
    };

    mxArray *tmp, *tmp2, *tmp3, *tmp4, *tmp5;
    double *tmppr, *tmppr2, *tmppr3, *tmppr4, *tmppr5;

    /* variables used for receiving the settings */

    U32      blueGainValue = -1;
    float    currentFrameRateValue = -1;
    U32      dataTransferSizeValue = -1;
    U32      exposureValue = -1;
    float    exposureTimeValue = -1;
    float    gammaValue = -1;
    U32      gpoValue = -1;
```

```
U32      greenGainValue = -1;
HARDWARE_VERSION hardwareVersionValue;
U32      imagerChipIdValue = -1;
U32      imagerClockingValue = -1;
LPSTR    imagerNameValue;
U32      imagerTypeValue = -1;
U32      monoGainValue = -1;

long     previewWindowPosLeftValue = -1;
long     previewWindowPosTopValue = -1;

U32      previewWindowSizeWidthValue = -1;
U32      previewWindowSizeHeightValue = -1;

U32      redGainValue = -1;
U32      saturationValue = -1;
U32      serialNumberValue = -1;
U32      softwareVersionValue = -1;
U32      subWindowDecimationValue = -1;
U32      subWindowStartColumnValue = -1;
U32      subWindowStartRowValue = -1;
U32      subWindowNumberColumnsValue = -1;
U32      subWindowNumberRowsValue = -1;

U32      subWindowPosStartColumnValue = -1;
U32      subWindowPosStartRowValue = -1;

U32      subWindowSizeDecimationValue = -1;
U32      subWindowSizeWidthValue = -1;
U32      subWindowSizeHeightValue = -1;

U32      timeoutValue = -1;
U32      videoModeValue = -1;

U32      grabColorConversion = -1;
U32      grabOutputType = -1;

/*********************************************************************/
/*          Get the deviceId                                         */
/*********************************************************************/
rhs[0] = mxCreateString("get");

rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
px = mxGetPr(rhs[1]);
px[0] = (U32)serialNumber;

mexCallMATLAB(1, lhs, 2, rhs, "plDevices");

deviceId = (HANDLE)(int)mxGetScalar(lhs[0]);


/*********************************************************************/
/*          Get all settings and store them in the mxArray           */
/*********************************************************************/
m = mxCreateStructMatrix(rows, cols, nof, (const char **)fieldnames);

/* DeviceID */
tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
tmppr = mxGetPr(tmp);
tmppr[0] = (int)deviceId;
mxSetField(m, 0, "DeviceID", tmp);

/* Blue Gain */
nRetValue = PimMegaGetBlueGain(deviceId, &blueGainValue);

if (ApiSuccess == nRetValue)
{
    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(tmp);
    tmppr[0] = (U32)blueGainValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
```

```
{
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "BlueGain", tmp);

/* Current Frame Rate */
nRetValue = PimMegaGetCurrentFrameRate(deviceId, &currentFrameRateValue);

if (ApiSuccess == nRetValue)
{
    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(tmp);
    tmppr[0] = currentFrameRateValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "CurrentFrameRate", tmp);

/* Data Transfer Size */
nRetValue = PimMegaGetDataTransferSize(deviceId, &dataTransferSizeValue);

if (ApiSuccess == nRetValue)
{
    if (dataTransferSizeValue == DATA_8BIT_SIZE)
    {
        tmp = mxCreateString("DATA_8BIT_SIZE");
    }
    else if (dataTransferSizeValue == DATA_16BIT_SIZE)
    {
        tmp = mxCreateString("DATA_16BIT_SIZE");
    }
    else
    {
        tmp = mxCreateString("Unknown value");
    }
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "DataTransferSize", tmp);

/* Exposure */
nRetValue = PimMegaGetExposure(deviceId, &exposureValue);

if (ApiSuccess == nRetValue)
{
    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(tmp);
    tmppr[0] = (U32)exposureValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "Exposure", tmp);

/* Exposure Time */
nRetValue = PimMegaGetExposureTime(deviceId, &exposureTimeValue);

if (ApiSuccess == nRetValue)
{
```

```
    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(tmp);
    tmppr[0] = exposureTimeValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "ExposureTime", tmp);

/* Gamma */
nRetValue = PimMegaGetGamma(deviceId, &gammaValue);

if (ApiSuccess == nRetValue)
{
    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(tmp);
    tmppr[0] = gammaValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "Gamma", tmp);

/* Gpo */
nRetValue = PimMegaGetGpo(deviceId, &gpoValue);

if (ApiSuccess == nRetValue)
{
    if (gpoValue == 0)
    {
        tmp = mxCreateString("Off");
    }
    else if (gpoValue == 1)
    {
        tmp = mxCreateString("On");
    }
    else
    {
        tmp = mxCreateString("Unknown value");
    }
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "Gpo", tmp);

/* Green Gain */
nRetValue = PimMegaGetGreenGain(deviceId, &greenGainValue);

if (ApiSuccess == nRetValue)
{
    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(tmp);
    tmppr[0] = (U32)greenGainValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
{
```

```
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "GreenGain", tmp);

/* Hardware Version */
nRetValue = PimMegaGetHardwareVersion(deviceId, &hardwareVersionValue);

if (ApiSuccess == nRetValue)
{
    tmp = mxCreateString((const char *)hardwareVersionValue.ProductID);
    tmp2 = mxCreateString((const char *)hardwareVersionValue.SerialNumber);
    tmp3 = mxCreateString((const char *)hardwareVersionValue.FirmwareVersion);
    tmp4 = mxCreateString((const char *)hardwareVersionValue.FpgaVersion);
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
    tmp2 = mxCreateString("Unsupported");
    tmp3 = mxCreateString("Unsupported");
    tmp4 = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
    tmp2 = mxCreateString("Could not get value");
    tmp3 = mxCreateString("Could not get value");
    tmp4 = mxCreateString("Could not get value");
}
mxSetField(m, 0, "HardwareVersion.ProductID", tmp);
mxSetField(m, 0, "HardwareVersion.SerialNumber", tmp2);
mxSetField(m, 0, "HardwareVersion.FirmwareVersion", tmp3);
mxSetField(m, 0, "HardwareVersion.FpgaVersion", tmp4);

/* Imager Chip ID */
nRetValue = PimMegaGetImagerChipId(deviceId, &imagerChipIdValue);

if (ApiSuccess == nRetValue)
{
    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(tmp);
    tmppr[0] = (U32)imagerChipIdValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "ImagerChipId", tmp);

/* Imager Clocking */
nRetValue = PimMegaGetImagerClocking(deviceId, &imagerClockingValue);

if (ApiSuccess == nRetValue)
{
    if (imagerClockingValue == 0x00)
    {
        tmp = mxCreateString("0x00: External (16MHz) No division");
    }
    else if (imagerClockingValue == 0x01)
    {
        tmp = mxCreateString("0x01: External (16MHz) Division by 2");
    }
    else if (imagerClockingValue == 0x02)
    {
        tmp = mxCreateString("0x02: External (16MHz) Division by 4");
    }
    else if (imagerClockingValue == 0x80)
    {
        tmp = mxCreateString("0x80: Internal (24MHz) No division");
    }
    else if (imagerClockingValue == 0x81)
    {
        tmp = mxCreateString("0x81: Internal (24MHz) Division by 2");
```

```
        }
        else if (imagerClockingValue == 0x82)
        {
            tmp = mxCreateString("0x82: Internal (24MHz) Division by 4");
        }
        else
        {
            tmp = mxCreateString("Unknown value");
        }
    }
    else if (ApiNotSupportedError == nRetValue)
    {
        tmp = mxCreateString("Unsupported");
    }
    else
    {
        tmp = mxCreateString("Could not get value");
    }
    mxSetField(m, 0, "ImagerClocking", tmp);

    /* Imager Name */
    imagerNameValue = (char *)mxCalloc(81, sizeof(char));

    nRetValue = PimMegaGetImagerName(deviceId, imagerNameValue);

    if (ApiSuccess == nRetValue)
    {
        tmp = mxCreateString(imagerNameValue);
    }
    else if (ApiNotSupportedError == nRetValue)
    {
        tmp = mxCreateString("Unsupported");
    }
    else
    {
        tmp = mxCreateString("Could not get value");
    }
    mxSetField(m, 0, "ImagerName", tmp);

    /* Imager Type */
    nRetValue = PimMegaGetImagerType(deviceId, &imagerTypeValue);

    if (ApiSuccess == nRetValue)
    {
        if (imagerTypeValue == PCS2112M_IMAGER)
        {
            tmp = mxCreateString("PCS2112M_IMAGER (Monochrome Camera)");
        }
        else if (imagerTypeValue == PCS2112C_IMAGER)
        {
            tmp = mxCreateString("PCS2112C_IMAGER (Color Camera)");
        }
        else
        {
            tmp = mxCreateString("Unknown value");
        }
    }
    else if (ApiNotSupportedError == nRetValue)
    {
        tmp = mxCreateString("Unsupported");
    }
    else
    {
        tmp = mxCreateString("Could not get value");
    }
    mxSetField(m, 0, "ImagerType", tmp);

    /* Mono Gain */
    nRetValue = PimMegaGetMonoGain(deviceId, &monoGainValue);

    if (ApiSuccess == nRetValue)
    {
        tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(tmp);
        tmppr[0] = (U32)monoGainValue;
    }
```

```
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "MonoGain", tmp);

/* Preview Window Pos */
nRetValue = PimMegaGetPreviewWindowPos(deviceId, &previewWindowPosLeftValue, \
                                       &previewWindowPosTopValue);

if (ApiSuccess == nRetValue)
{
    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(tmp);
    tmppr[0] = (long)previewWindowPosLeftValue;
    tmp2 = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr2 = mxGetPr(tmp2);
    tmppr2[0] = (long)previewWindowPosTopValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
    tmp2 = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
    tmp2 = mxCreateString("Could not get value");
}
mxSetField(m, 0, "PreviewWindowPos.Left", tmp);
mxSetField(m, 0, "PreviewWindowPos.Top", tmp2);

/* Preview Window Size */
nRetValue = PimMegaGetPreviewWindowSize(deviceId, &previewWindowSizeWidthValue, \
                                        &previewWindowSizeHeightValue);

if (ApiSuccess == nRetValue)
{
    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(tmp);
    tmppr[0] = (U32)previewWindowSizeWidthValue;
    tmp2 = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr2 = mxGetPr(tmp2);
    tmppr2[0] = (U32)previewWindowSizeHeightValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
    tmp2 = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
    tmp2 = mxCreateString("Could not get value");
}
mxSetField(m, 0, "PreviewWindowSize.Width", tmp);
mxSetField(m, 0, "PreviewWindowSize.Height", tmp2);

/* Red Gain */
nRetValue = PimMegaGetRedGain(deviceId, &redGainValue);

if (ApiSuccess == nRetValue)
{
    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(tmp);
    tmppr[0] = (U32)redGainValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
```

```
{
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "RedGain", tmp);

/* Saturation */
nRetValue = PimMegaGetSaturation(deviceId, &saturationValue);

if (ApiSuccess == nRetValue)
{
    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(tmp);
    tmppr[0] = (U32)saturationValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "Saturation", tmp);

/* Serial Number */
nRetValue = PimMegaGetSerialNumber(deviceId, &serialNumberValue);

if (ApiSuccess == nRetValue)
{
    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(tmp);
    tmppr[0] = (U32)serialNumberValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "SerialNumber", tmp);

/* Software Version */
nRetValue = PimMegaGetSoftwareVersion(deviceId, &softwareVersionValue);

if (ApiSuccess == nRetValue)
{
    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(tmp);
    tmppr[0] = (U32)softwareVersionValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "SoftwareVersion", tmp);

/* Sub Window */
nRetValue = PimMegaGetSubWindow(deviceId, &subWindowDecimationValue, \
                  &subWindowStartColumnValue, &subWindowStartRowValue, \
                  &subWindowNumberColumnsValue, &subWindowNumberRowsValue);

if (ApiSuccess == nRetValue)
{
    if (subWindowDecimationValue == PCS2112_NO_DECIMATION)
    {
        tmp = mxCreateString("PCS2112_NO_DECIMATION");
    }
    else if (subWindowDecimationValue == PCS2112_DECIMATE_BY_2)
    {
        tmp = mxCreateString("PCS2112_DECIMATE_BY_2");
```

```
        }
        else if (subWindowDecimationValue == PCS2112_DECIMATE_BY_4)
        {
            tmp = mxCreateString("PCS2112_DECIMATE_BY_4");
        }
        else
        {
            tmp = mxCreateString("Unknown value");
        }

        tmp2 = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr2 = mxGetPr(tmp2);
        tmppr2[0] = (U32)subWindowStartColumnValue;
        tmp3 = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr3 = mxGetPr(tmp3);
        tmppr3[0] = (U32)subWindowStartRowValue;
        tmp4 = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr4 = mxGetPr(tmp4);
        tmppr4[0] = (U32)subWindowNumberColumnsValue;
        tmp5 = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr5 = mxGetPr(tmp5);
        tmppr5[0] = (U32)subWindowNumberRowsValue;
    }
    else if (ApiNotSupportedError == nRetValue)
    {
        tmp = mxCreateString("Unsupported");
        tmp2 = mxCreateString("Unsupported");
        tmp3 = mxCreateString("Unsupported");
        tmp4 = mxCreateString("Unsupported");
        tmp5 = mxCreateString("Unsupported");
    }
    else
    {
        tmp = mxCreateString("Could not get value");
        tmp2 = mxCreateString("Could not get value");
        tmp3 = mxCreateString("Could not get value");
        tmp4 = mxCreateString("Could not get value");
        tmp5 = mxCreateString("Could not get value");
    }
    mxSetField(m, 0, "SubWindow.Decimation", tmp);
    mxSetField(m, 0, "SubWindow.StartColumn", tmp2);
    mxSetField(m, 0, "SubWindow.StartRow", tmp3);
    mxSetField(m, 0, "SubWindow.NumberColumns", tmp4);
    mxSetField(m, 0, "SubWindow.NumberRows", tmp5);

    /* Sub Window Pos */
    nRetValue = PimMegaGetSubWindowPos(deviceId, &subWindowPosStartColumnValue, \
                                       &subWindowPosStartRowValue);

    if (ApiSuccess == nRetValue)
    {
        tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(tmp);
        tmppr[0] = (U32)subWindowPosStartColumnValue;
        tmp2 = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr2 = mxGetPr(tmp2);
        tmppr2[0] = (U32)subWindowPosStartRowValue;
    }
    else if (ApiNotSupportedError == nRetValue)
    {
        tmp = mxCreateString("Unsupported");
        tmp2 = mxCreateString("Unsupported");
    }
    else
    {
        tmp = mxCreateString("Could not get value");
        tmp2 = mxCreateString("Could not get value");
    }
    mxSetField(m, 0, "SubWindowPos.StartColumn", tmp);
    mxSetField(m, 0, "SubWindowPos.StartRow", tmp2);

    /* Sub Window Size */
    nRetValue = PimMegaGetSubWindowSize(deviceId, &subWindowSizeDecimationValue, \
                                &subWindowSizeWidthValue, &subWindowSizeHeightValue);

    if (ApiSuccess == nRetValue)
```

```
{
    if (subWindowSizeDecimationValue == PCS2112_NO_DECIMATION)
    {
        tmp = mxCreateString("PCS2112_NO_DECIMATION");
    }
    else if (subWindowSizeDecimationValue == PCS2112_DECIMATE_BY_2)
    {
        tmp = mxCreateString("PCS2112_DECIMATE_BY_2");
    }
    else if (subWindowSizeDecimationValue == PCS2112_DECIMATE_BY_4)
    {
        tmp = mxCreateString("PCS2112_DECIMATE_BY_4");
    }
    else
    {
        tmp = mxCreateString("Unknown value");
    }

    tmp2 = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr2 = mxGetPr(tmp2);
    tmppr2[0] = (U32)subWindowSizeWidthValue;
    tmp3 = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr3 = mxGetPr(tmp3);
    tmppr3[0] = (U32)subWindowSizeHeightValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
    tmp2 = mxCreateString("Unsupported");
    tmp3 = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
    tmp2 = mxCreateString("Could not get value");
    tmp3 = mxCreateString("Could not get value");
}
mxSetField(m, 0, "SubWindowSize.Decimation", tmp);
mxSetField(m, 0, "SubWindowSize.Width", tmp2);
mxSetField(m, 0, "SubWindowSize.Height", tmp3);

/* Timeout */
nRetValue = PimMegaGetTimeout(deviceId, &timeoutValue);

if (ApiSuccess == nRetValue)
{
    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(tmp);
    tmppr[0] = (U32)timeoutValue;
}
else if (ApiNotSupportedError == nRetValue)
{
    tmp = mxCreateString("Unsupported");
}
else
{
    tmp = mxCreateString("Could not get value");
}
mxSetField(m, 0, "Timeout", tmp);

/* Video Mode */
nRetValue = PimMegaGetVideoMode(deviceId, &videoModeValue);

if (ApiSuccess == nRetValue)
{
    if (videoModeValue == STILL_MODE)
    {
        tmp = mxCreateString("STILL_MODE");
    }
    else if (videoModeValue == VIDEO_MODE)
    {
        tmp = mxCreateString("VIDEO_MODE");
    }
    else
    {
        tmp = mxCreateString("Unknown value");
```

```
        }
    }
    else if (ApiNotSupportedError == nRetValue)
    {
        tmp = mxCreateString("Unsupported");
    }
    else
    {
        tmp = mxCreateString("Could not get value");
    }
    mxSetField(m, 0, "VideoMode", tmp);

    /* GrabColorConversion */
    rhs[0] = mxCreateString("getpar");
    rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
    px = mxGetPr(rhs[1]);
    px[0] = (U32)serialNumber;
    rhs[2] = mxCreateString("GrabColorConversion");
    mexCallMATLAB(1, lhs, 3, rhs, "plDevices");
    grabColorConversion = (U32)mxGetScalar(lhs[0]);

    switch (grabColorConversion)
    {
        case BAYER_2BY2_COLOR:         tmp = mxCreateString("BAYER_2BY2_COLOR");
                                       break;
        case BAYER_3BY3_COLOR:         tmp = mxCreateString("BAYER_3BY3_COLOR");
                                       break;
        case BAYER_3BY3GGRAD_COLOR:    tmp = mxCreateString("BAYER_3BY3GGRAD_COLOR");
                                       break;
        case BAYER_2PASSGRAD_COLOR:    tmp = mxCreateString("BAYER_2PASSGRAD_COLOR");
                                       break;
        case BAYER_2PASSADAPT_COLOR:   tmp = mxCreateString("BAYER_2PASSADAPT_COLOR");
                                       break;
        case BAYER_VARGRAD_COLOR:      tmp = mxCreateString("BAYER_VARGRAD_COLOR");
                                       break;
        case BAYER_2BY2_MONO:          tmp = mxCreateString("BAYER_2BY2_MONO");
                                       break;
        case BAYER_3BY3_MONO:          tmp = mxCreateString("BAYER_3BY3_MONO");
                                       break;
        case BAYER_ADAPT_MONO:         tmp = mxCreateString("BAYER_ADAPT_MONO");
                                       break;
        case BAYER_NO_CONVERSION:      tmp = mxCreateString("BAYER_NO_CONVERSION");
                                       break;
        default:                       tmp = mxCreateString("Unknown value");
                                       break;
    }
    mxSetField(m, 0, "GrabColorConversion", tmp);

    /* GrabOutputType */
    rhs[0] = mxCreateString("getpar");
    rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
    px = mxGetPr(rhs[1]);
    px[0] = (U32)serialNumber;
    rhs[2] = mxCreateString("GrabOutputType");
    mexCallMATLAB(1, lhs, 3, rhs, "plDevices");
    grabOutputType = (U32)mxGetScalar(lhs[0]);

    switch (grabOutputType)
    {
        case RAW:        tmp = mxCreateString("RAW");
                               break;
        case IMAGE:        tmp = mxCreateString("IMAGE");
                               break;
        case RGB24:        tmp = mxCreateString("RGB24");
                               break;
        default:                tmp = mxCreateString("Unknown value");
                               break;
    }
    mxSetField(m, 0, "GrabOutputType", tmp);

    /* return the device handle array */
    returnArray[0] = m;
    return;
}
```

D.3 plCreateDeviceHandle.h

```
/*************************************************************************/
/* Filename:    plCreateDeviceHandle.h                                   */
/* Description: Header file for plCreateDeviceHandle.cpp                 */
/* Authors:     L.I.Oei, M.A.E.Bakker                                    */
/* Date:        2002/07/31                                               */
/* Updates:                                                              */
/*************************************************************************/
/* Header file for plCreateDeviceHandle.cpp.                             */
/*************************************************************************/

void plCreateDeviceHandle(mxArray *returnArray[], U32 serialNumber);
```

D.4 plDevices.cpp

```
/*************************************************************************/
/* Filename:    plDevices.cpp                                            */
/* Description: Source code for the function plDevices                   */
/* Authors:     M.A.E.Bakker, L.I.Oei                                    */
/*              maarten@panic.et.tudelft.nl, l.i.oei@its.tudelft.nl      */
/* Date:        2002/08/08                                               */
/* Updates:                                                              */
/*************************************************************************/
/* plDevices keeps the open device array, in which the list of open      */
/* devices and some auxilairy parameters that are not stored in the      */
/* device are stored. This is a list of tuples of the form:              */
/* {serialnumber, deviceId, parameter, parameter}                        */
/* plDevices provides functionality to check and update the list.        */
/*************************************************************************/
/* Input:    prhs[0] = (string) name of task to be performed: print,     */
/*                     remove, isopen, get, add, getpar, setpar          */
/*           prhs[1] = (U32) serial number of device; Only for remove,   */
/*                     isopen, get, add, getpar, setpar                  */
/*           prhs[2] = (int) deviceID; Only for add                      */
/*           prhs[2] = (string) name of parameter; Only for getpar,      */
/*                     setpar                                            */
/*           prhs[3] = value of parameter; Only for setpar               */
/* Output:   plhs[0] = (double) 1 if device open, 0 if closed; Only for  */
/*                     isopen                                            */
/*           plhs[0] = (int) deviceID if open, -1 if closed; Only for    */
/*                     get                                               */
/*           plhs[0] = parameter value, -1 if not found; Only for getpar*/
/* Syntax:   not to be used directly from matlab                         */
/*************************************************************************/

/* Exclude rarely-used stuff from Windows headers */
#define WIN32_LEAN_AND_MEAN
#include <windows.h>

#include "Y:\soft95\matlab6\extern\include\mex.h"
/* with Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmex.lib */
/* and  Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmx.lib */

#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"
/* with Y:\software\framegrag\pixelink\api\pimmegaapi.lib */

#include "..\plTypes.h"

#ifdef __cplusplus
    extern "C" {
#endif

/* global variables needed for storing the open device array */
struct {U32 serialNumber; int deviceID; U32 grabColorConversion; int grabOutputType;}
deviceArray[32];
int deviceCount = 0;

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    double *y;
    int buflen, i, t;
    char *task = "", *parName = "";
    U32 serialnumber;

    /* make sure plDevices stays in memory, even when doing 'clear all' or
     * 'clear mex'.
     */
    mexLock();

    if ((nrhs > 0) && mxIsChar(prhs[0]))  /* 1st argument (task) should be a string */
    {
        /* allocate memory for task string and get the 1st argument */
        buflen = (mxGetM(prhs[0]) * mxGetN(prhs[0]) * sizeof(mxChar)) + 1;
        task = (char *)mxCalloc(buflen, sizeof(char));
        mxGetString(prhs[0], task, buflen);

        if (nrhs >= 2) /* if it exists, the 2nd argument is the serial number */
```

```
        {
            if (!mxIsDouble(prhs[1]))
            {
                mexPrintf("Internal error: call to plDevices failed.\n");
                mexPrintf("2nd argument should be the serial number.\n");
                mexErrMsgTxt("\n");
            }
            /* get the 2nd argument (serial number) */
            serialnumber = (U32)mxGetScalar(prhs[1]);
        }
        switch (nrhs)
        {
        /* print has 1 argument (task) */
        case 1:     if (!strcmp(task, "print"))
                    {
                        for (i = 0; i < deviceCount; i++)
                        {
                          mexPrintf("Device nr: %d --> %d, %d: GrabColorConversion = \
%d, GrabOutputType = %d\n", i, deviceArray[i].serialNumber, deviceArray[i].deviceID, \
deviceArray[i].grabColorConversion, deviceArray[i].grabOutputType);
                        }
                    }
                    else
                    {
                        mexPrintf("Internal error: call to plDevices failed.\n");
                        mexPrintf("Wrong number of arguments (1) is given or unknown \
                                   task %s.", task);
                        mexErrMsgTxt("\n");
                    }
                    break;
        /* remove, isopen, get have 2 arguments (task, serialnumber) */
        case 2:     if (!strcmp(task, "remove"))
                    {
                        for (i = 0; i < deviceCount; i++)
                        {
                            if (deviceArray[i].serialNumber == serialnumber)
                            {
                                for (t = i; t < deviceCount; t++)
                                {
                                  deviceArray[t].serialNumber = \
                                                        deviceArray[t+1].serialNumber;
                                  deviceArray[t].deviceID = deviceArray[t+1].deviceID;
                                }
                                deviceCount--;
                            }
                        }
                    }
                    else if (!strcmp(task, "isopen"))
                    {
                        /* initialise return value */
                        plhs[0]=mxCreateDoubleMatrix(1,1,mxREAL);
                        y = mxGetPr(plhs[0]);
                       /* return 0, unless the loop finds a matching serialnumber */
                        y[0] = 0;
                        for (i = 0; i < deviceCount;i++)
                        {
                            if (deviceArray[i].serialNumber == serialnumber)
                            {
                               y[0] = 1; /* found matching serialnumber, return 1 */
                                break;
                            }
                        }
                    }
                    else if (!strcmp(task, "get"))
                    {
                        /* initialise return value */
                        plhs[0]=mxCreateDoubleMatrix(1,1,mxREAL);
                        y = mxGetPr(plhs[0]);
                        /* return -1, unless the loop finds a matching serialnumber */
                        y[0] = -1;
                        for (i = 0; i < deviceCount;i++)
                        {
                            if (deviceArray[i].serialNumber == serialnumber)
                            {
                                y[0] = (double)deviceArray[i].deviceID;
                               break; /* found a matching serialnumber, end search */
```

```
                    }
                }
            }
            else
            {
                mexPrintf("Internal error: call to plDevices failed.\n");
               mexPrintf("Wrong number of arguments (2) is given or unknown \
                          task  %s.", task);
                mexErrMsgTxt("\n");
            }
            break;
/* add has 3 arguments (task, serialnumber, deviceid)
 * getpar has 3 arguments (task, serialnumber, parameter name)
 */
case 3:     if (!strcmp(task, "add"))
            {
                if (deviceCount >= 32)
                    mexErrMsgTxt("Maximum number of open devices (32) \
                                  reached.\n");
                deviceArray[deviceCount].serialNumber = serialnumber;
                deviceArray[deviceCount].deviceID = (int)mxGetScalar(prhs[2]);
                deviceArray[deviceCount].grabColorConversion = \
                                                        BAYER_3BY3_COLOR;
                deviceArray[deviceCount].grabOutputType = RAW;
                deviceCount++;
            }
            else if (!strcmp(task, "getpar"))
            {   /* allocate memory for parName string and get the 3rd
                   argument */
                buflen = (mxGetM(prhs[2])* mxGetN(prhs[2])*sizeof(mxChar)) +1;
                parName = (char *)mxCalloc(buflen, sizeof(char));
                mxGetString(prhs[2], parName, buflen);
                /* initialise return value */
                plhs[0]=mxCreateDoubleMatrix(1,1,mxREAL);
                y = mxGetPr(plhs[0]);
                if (!strcmp(parName, "GrabColorConversion"))
                {
                    t = 1;
                    for (i = 0; i < deviceCount;i++)
                    {
                        if (deviceArray[i].serialNumber == serialnumber)
                        {
                            y[0] = (double)deviceArray[i].grabColorConversion;
                            t = 0;
                            break; /* found a matching serialnumber,
                                      end search */
                        }
                    }
                    if (t)
                    {
                        mexPrintf("Internal error: getpar called with \
                                   unknown serial number %d", serialnumber);
                        mexErrMsgTxt("\n");
                    }
                }
                else if (!strcmp(parName, "GrabOutputType"))
                {
                    t = 1;
                    for (i = 0; i < deviceCount;i++)
                    {
                        if (deviceArray[i].serialNumber == serialnumber)
                        {
                            y[0] = (double)deviceArray[i].grabOutputType;
                            t = 0;
                            break; /* found a matching serialnumber,
                                      end search */
                        }
                    }
                    if (t)
                    {
                        mexPrintf("Internal error: getpar called with \
                                   unknown serial number %d", serialnumber);
                        mexErrMsgTxt("\n");
                    }
                }
                else
```

```
                    {
                        mexPrintf("getpar called with unknown parameter name \
                                %s.", parName);
                        mexErrMsgTxt("\n");
                    }
                }
                else
                {
                    mexPrintf("Internal error: call to plDevices failed.\n");
                   mexPrintf("Wrong number of arguments (3) is given or unknown\
                            task %s.", task);
                    mexErrMsgTxt("\n");
                }
                break;
/* setpar has 4 arguments */
case 4:     if (!strcmp(task, "setpar"))
            {    /* allocate memory for parName string and get the 3rd
                    argument*/
                buflen = (mxGetM(prhs[2])* mxGetN(prhs[2])*sizeof(mxChar)) +1;
                parName = (char *)mxCalloc(buflen, sizeof(char));
                mxGetString(prhs[2], parName, buflen);

                if (!strcmp(parName, "GrabColorConversion"))
                {
                    t = 1;
                    for (i = 0; i < deviceCount;i++)
                    {
                        if (deviceArray[i].serialNumber == serialnumber)
                        {
                            deviceArray[i].grabColorConversion = \
                                                (U32)mxGetScalar(prhs[3]);
                            t = 0;
                            break; /* found a matching serialnumber,
                                      end search */
                        }
                    }
                    if (t)
                    {
                        mexPrintf("Internal error: setpar called with \
                                  unknown serial number %d", serialnumber);
                        mexErrMsgTxt("\n");
                    }
                }
                else if (!strcmp(parName, "GrabOutputType"))
                {
                    t = 1;
                    for (i = 0; i < deviceCount;i++)
                    {
                        if (deviceArray[i].serialNumber == serialnumber)
                        {
                            deviceArray[i].grabOutputType = \
                                                (U32)mxGetScalar(prhs[3]);
                            t = 0;
                            break; /* found a matching serialnumber,
                                      end search */
                        }
                    }
                    if (t)
                    {
                        mexPrintf("Internal error: setpar called with \
                                  unknown serial number %d", serialnumber);
                        mexErrMsgTxt("\n");
                    }
                }
                else
                {
                    mexPrintf("getpar called with unknown parameter name \
                                %s.", parName);
                    mexErrMsgTxt("\n");
                }
            }
            else
            {
                mexPrintf("Internal error: call to plDevices failed.\n");
               mexPrintf("Wrong number of arguments (4) is given or unknown \
                            task %s.", task);
```

```
                        mexErrMsgTxt("\n");
                    }
                    break;
        default:    mexPrintf("Internal error: call to plDevices failed.\n");
                    mexErrMsgTxt("Number of arguments out of range.\n");
        }
    }
    else
    {
        mexPrintf("Internal error: call to plDevices failed.\n");
       mexPrintf("First argument should be a string with the name of the task to be \
                  performed.\n");
        mexErrMsgTxt("\n");
    }
}

#ifdef __cplusplus
    }    /* extern "C" */
#endif
```

D.5 plError.cpp

```
/*************************************************************************/
/* Filename:    plError.cpp                                              */
/* Description: Source code for the error checking of the pl FGI         */
/* Authors:     M.A.E.Bakker, L.I.Oei                                    */
/*              maarten@panic.et.tudelft.nl, l.i.oei@its.tudelft.nl      */
/* Date:        2002/08/02                                               */
/* Updates:                                                              */
/*************************************************************************/
/* plError translates a PixeLINK API status code into a useful error     */
/* message.                                                              */
/*************************************************************************/
/* Input:    (PXL_RETURN_CODE) the API's return code that should be      */
/*           checked                                                     */
/*           (string) description when the possible fault occurred       */
/* Output:   (int)    0 if no error                                      */
/*                 1 on an error that should be interpreted as a warning */
/*                 2 on an error that should be interpreted as fatal     */
/* Syntax:    i = plError(PXL_RETURN_CODE result, char* error);          */
/*************************************************************************/

#define WIN32_LEAN_AND_MEAN    /* Exclude rarely-used stuff from Windows */
#include <windows.h>
#include "Y:\soft95\matlab6\extern\include\mex.h"
#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"

int plError(PXL_RETURN_CODE result, char *error)
{
    char errorStr[49+33];
    strcpy(errorStr, "The device's API encountered a problem while ");
    strncat(errorStr, error, 33);
    strcat(errorStr, ":\n");
    if (result != ApiSuccess)
    {
        mexPrintf(errorStr);
    }
    switch (result)
    {
        case ApiSuccess:          return 0;
        case ApiNullHandleError:  mexPrintf("NULL-handle: the device was not \
                                    sucessfully initialised.\n");
                                  return 2;
        case ApiNullPointerError: mexPrintf("NULL-pointer: problem allocating \
                                    required memory.\n");
                                  return 2;
        case ApiNoDeviceError:    mexPrintf("The camera may be disconnected.\n");
                                  return 1;
        case ApiCOMError:         mexPrintf("Windows COM object error.\n");
                                  return 2;
        case ApiIoctlError:       mexPrintf("I/O control error, API may be \
                                    incompatible with driver.\n");
                                  return 2;
        case ApiFileOpenError:    mexPrintf("File I/O error.\n");
                                  return 1;
        case ApiInvalidParameterError: mexPrintf("Invalid parameter.\n");
                                       return 2;
        case ApiImagerUnknownError: mexPrintf("API doesn't recognize imager, should \
                                        be PCS2112M_IMAGER or PCS2112C_IMAGER.\n");
                                    return 2;
       case ApiOutOfMemoryError: mexPrintf("Unable to allocate required memory.\n");
                                 return 2;
        case ApiUnknownError:     mexPrintf("Error unknown to the API.\n");
                                  return 2;
       case ApiNoPreviewRunningError: mexPrintf("The preview window is not open.\n");
                                      return 2;
        case ApiNotSupportedError: mexPrintf("The camera does not support the \
                                     requested functionality.\n");
                                   return 1;
       case ApiInvalidFunctionCallError: mexPrintf("The function was called in an \
                                           improper sequence.\n");
                                         return 2;
        case ApiHardwareError:    mexPrintf("Hardware error.\n");
                                  return 2;
```

```
        case ApiOSVersionError:   mexPrintf("Unsupported operating system \
                                             version.\n");
                                  return 2;
        case ApiMaximumIterationsError: mexPrintf("Too many internal iterations.\n");
                                        return 2;
        case ApiTimeoutError:     mexPrintf("Time limit exceeded, increasing the \
                                           Timeout setting may solve the problem.\n");
                                  return 2;
        default:                  mexPrintf("Error unknown to the FGI.\n");
                                  return 1;
    }
}
```

D.6 plError.h

```
/*************************************************************************/
/* Filename:    plError.h                                                */
/* Description: Header file for plError.cpp                              */
/* Authors:     L.I.Oei, M.A.E.Bakker                                    */
/* Date:        2002/08/06                                               */
/* Updates:                                                              */
/*************************************************************************/

int plError(PXL_RETURN_CODE, char*);
```

D.7 plGet.cpp

```
/*********************************************************************/
/* Filename:    plGet.cpp                                            */
/* Description: Source code for the function plGet                   */
/* Authors:     L.I.Oei, M.A.E.Bakker                                */
/*              maarten@panic.et.tudelft.nl, l.i.oei@its.tudelft.nl  */
/* Date:        2002/08/01                                           */
/* Updates:                                                          */
/*********************************************************************/
/* plGet returns the device's handle if 1 argument is given, and     */
/* returns the value of the given parameter if 2 arguments are given.*/
/*********************************************************************/
/* Input:    prhs[0] = (struct) handle for the device or             */
/*                     (U32)    serial number of the device          */
/*           prhs[1] = (string) name of the parameter (optional)     */
/* Output:   plhs[0] = (struct) handle for the device (with 1 argument) */
/*           plhs[0] = value of the given parameter (with 2 arguments) */
/* Syntax:   plGet(serialnumber) or plGet(serialnumber, parametername) */
/*           plGet(handle) or plGet(handle, parametername)           */
/*********************************************************************/


#define WIN32_LEAN_AND_MEAN    /* Exclude rarely-used stuff from Windows headers */
#include <windows.h>

#include "Y:\soft95\matlab6\extern\include\mex.h"
/* with Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmex.lib */
/* and  Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmx.lib */

#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"
/* with Y:\software\framegrag\pixelink\api\pimmegaapi.lib */

#include "..\plCreateDeviceHandle.h"
#include "..\plGetValue.h"

#ifdef __cplusplus
    extern "C" {
#endif

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    mxArray *serialNumberField;
    U32 serialNumber = -1;        /* the serial number of the device */

    mxArray *lhs[1], *rhs[2];        /* to call fgdevices */
    double *px;
    int isOpen = 0;

    int buflen;
    char *parametername = "";

    /* prhs[0] should be the device handle (struct) or serial number (int) */
    /* prhs[1] should be the parameter name (optional) */
    if ((nrhs==1) || (nrhs ==2))
    {
        /* get the serialNumber */
        if (mxIsStruct(prhs[0]))
        {
            serialNumberField = mxGetField(prhs[0], 0, "SerialNumber");
            serialNumber = (U32)mxGetScalar(serialNumberField);
        }
        else if (mxIsDouble(prhs[0]))
        {
            serialNumber = (U32)mxGetScalar(prhs[0]);
        }
        else
        {
            mexPrintf("PixeLINK Interface error.\n");
            mexPrintf("plGet: Wrong argument given.\n");
            mexPrintf("First argument should be the device handle or the serial \
                      number.\n");
            mexErrMsgTxt("\n");
        }
```

```
        /* check whether the device has been opened */
        rhs[0] = mxCreateString("isopen");

        rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
        px = mxGetPr(rhs[1]);
        px[0] = (U32)serialNumber;

        mexCallMATLAB(1, lhs, 2, rhs, "plDevices");
        isOpen = (int)mxGetScalar(lhs[0]);

        if (!isOpen)
        {
            mexPrintf("plGet: Device with serial number %lu has not been opened.\n", \
                        serialNumber);
             mexPrintf("Cannot get settings.\n");
             mexErrMsgTxt("\n");
        }

        if (nrhs==1) /* 1 argument: return device handle */
        {
            /* create the device handle (return structure) */
            plCreateDeviceHandle(lhs, serialNumber);
            plhs[0] = lhs[0];
        }
        else /* nrhs==2: 2 arguments: return parameter value */
        {
            /* get the parameter name */
            if (!mxIsChar(prhs[1])) /* 2nd argument (parametername) should be a
                                       string */
            {
                mexPrintf("plGet: Wrong argument given.\n");
                mexPrintf("Second argument should be a string with the \
                           parametername.\n");
                mexErrMsgTxt("\n");
            }

            buflen = (mxGetM(prhs[1]) * mxGetN(prhs[1]) * sizeof(mxChar)) + 1;
            parametername = (char *)mxCalloc(buflen, sizeof(char));
            mxGetString(prhs[1], parametername, buflen);

            /* get the settings for this parameter */
            plGetValue(lhs, serialNumber, parametername);
            plhs[0] = lhs[0];
        }
    }
    else /* ! ((nrhs==1) || (nrhs==2)) */
    {
        mexPrintf("PixeLINK Interface error.\n");
        mexPrintf("plGet: Wrong number of arguments given. One or two arguments \
                   needed:\n");
        mexPrintf("handle or serial number and (optional) a parametername.\n");
        mexPrintf("Example: plGet(handle) or plGet(handle, parametername)\n");
        mexPrintf("     or: plGet(serialnumber) or plGet(serialnumber, \
                   parametername)\n");
        mexErrMsgTxt("\n");
    }
}

#ifdef __cplusplus
    }    /* extern "C" */
#endif
```

D.8 plGetValue.cpp

```
/*************************************************************************/
/* Filename:    plGetValue.cpp                                           */
/* Description: Source code for plGetValue                               */
/* Authors:     M.A.E.Bakker, L.I.Oei                                    */
/*              maarten@panic.et.tudelft.nl, l.i.oei@its.tudelft.nl      */
/* Date:        2002/08/08                                               */
/* Updates:                                                              */
/*************************************************************************/
/* plGetValue returns the value of a given parameter.                    */
/*************************************************************************/
/* Input:   returnarray (mxArray)   mxArray for receiving the structure */
/*          serialNumber (U32)      serialNumber of the device           */
/*          parametername (string)  name of the parameter                */
/* Output:  -                                                            */
/* Syntax:  plGetValue(returnArray, serialNumber, parametername)         */
/*************************************************************************/

#include <windows.h>
#include "Y:\soft95\matlab6\extern\include\mex.h"
#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"

#include "plError.h"
#include "plTypes.h"

void plGetValue(mxArray *returnArray[], U32 serialNumber, char* parametername)
{   /* m = mxArray structure to be returned as device handle */
    /* lhs, rhs used to call plDevices */
    mxArray *m, *lhs[1], *rhs[3];
    double *px;
    HANDLE deviceId;

    /* variable for receiving the error codes the pixelink api functions return */
    PXL_RETURN_CODE nRetValue;

    char *fieldnames[5];
    mxArray *tmp, *tmp2, *tmp3, *tmp4, *tmp5;
    double *tmppr, *tmppr2, *tmppr3, *tmppr4, *tmppr5;

    /* variables used for receiving the settings */

    U32      blueGainValue = -1;
    float    currentFrameRateValue = -1;
    U32      dataTransferSizeValue = -1;
    U32      exposureValue = -1;
    float    exposureTimeValue = -1;
    float    gammaValue = -1;
    U32      gpoValue = -1;
    U32      greenGainValue = -1;
    HARDWARE_VERSION hardwareVersionValue;
    U32      imagerChipIdValue = -1;
    U32      imagerClockingValue = -1;
    LPSTR    imagerNameValue;
    U32      imagerTypeValue = -1;
    U32      monoGainValue = -1;

    long     previewWindowPosLeftValue = -1;
    long     previewWindowPosTopValue = -1;

    U32      previewWindowSizeWidthValue = -1;
    U32      previewWindowSizeHeightValue = -1;

    U32      redGainValue = -1;
    U32      saturationValue = -1;
    U32      serialNumberValue = -1;
    U32      softwareVersionValue = -1;

    U32      subWindowDecimationValue = -1;
    U32      subWindowStartColumnValue = -1;
    U32      subWindowStartRowValue = -1;
    U32      subWindowNumberColumnsValue = -1;
    U32      subWindowNumberRowsValue = -1;
```

```
U32      subWindowPosStartColumnValue = -1;
U32      subWindowPosStartRowValue = -1;

U32      subWindowSizeDecimationValue = -1;
U32      subWindowSizeWidthValue = -1;
U32      subWindowSizeHeightValue = -1;

U32      timeoutValue = -1;
U32      videoModeValue = -1;

U32      grabColorConversionValue = -1;
U32      grabOutputTypeValue = -1;

/*****************************************************************/
/*          Get the deviceId                                     */
/*****************************************************************/
rhs[0] = mxCreateString("get");
rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
px = mxGetPr(rhs[1]);
px[0] = (U32)serialNumber;

mexCallMATLAB(1, lhs, 2, rhs, "plDevices");

deviceId = (HANDLE)(int)mxGetScalar(lhs[0]);

/*****************************************************************/
/*          Get the value of the given parameter                 */
/*****************************************************************/

if (!strcmp(parametername, "DeviceID"))
{   /* DeviceID */
    m = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(m);
    tmppr[0] = (int)deviceId;
}
else if (!strcmp(parametername, "BlueGain"))
{   /* Blue Gain */
    nRetValue = PimMegaGetBlueGain(deviceId, &blueGainValue);

    if (plError(nRetValue, "getting BlueGain value"))
    {
        mexErrMsgTxt("\n");
    }

    m = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(m);
    tmppr[0] = (U32)blueGainValue;
}
else if (!strcmp(parametername, "CurrentFrameRate"))
{   /* Current Frame Rate */
    nRetValue = PimMegaGetCurrentFrameRate(deviceId, &currentFrameRateValue);

    if (plError(nRetValue, "getting CurrentFrameRate value"))
    {
        mexErrMsgTxt("\n");
    }

    m = mxCreateDoubleMatrix(1, 1, mxREAL);
    tmppr = mxGetPr(m);
    tmppr[0] = currentFrameRateValue;
}
else if (!strcmp(parametername, "DataTransferSize"))
{   /* Data Transfer Size */
    nRetValue = PimMegaGetDataTransferSize(deviceId, &dataTransferSizeValue);

    if (plError(nRetValue, "getting DataTransferSize value"))
    {
        mexErrMsgTxt("\n");
    }

    if (dataTransferSizeValue == DATA_8BIT_SIZE)
    {
        m = mxCreateString("DATA_8BIT_SIZE");
    }
    else if (dataTransferSizeValue == DATA_16BIT_SIZE)
    {
```

```
            m = mxCreateString("DATA_16BIT_SIZE");
        }
        else
        {
            m = mxCreateString("Unknown value");
        }
    }
    else if (!strcmp(parametername, "Exposure"))
    {   /* Exposure */
        nRetValue = PimMegaGetExposure(deviceId, &exposureValue);

        if (plError(nRetValue, "getting Exposure value"))
        {
            mexErrMsgTxt("\n");
        }

        m = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(m);
        tmppr[0] = (U32)exposureValue;
    }
    else if (!strcmp(parametername, "ExposureTime"))
    {   /* Exposure Time */
        nRetValue = PimMegaGetExposureTime(deviceId, &exposureTimeValue);

        if (plError(nRetValue, "getting ExposureTime value"))
        {
            mexErrMsgTxt("\n");
        }

        m = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(m);
        tmppr[0] = exposureTimeValue;
    }
    else if (!strcmp(parametername, "Gamma"))
    {   /* Gamma */
        nRetValue = PimMegaGetGamma(deviceId, &gammaValue);

        if (plError(nRetValue, "getting Gamma value"))
        {
            mexErrMsgTxt("\n");
        }

        m = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(m);
        tmppr[0] = gammaValue;
    }
    else if (!strcmp(parametername, "Gpo"))
    {   /* Gpo */
        nRetValue = PimMegaGetGpo(deviceId, &gpoValue);

        if (plError(nRetValue, "getting BlueGain value"))
        {
            mexErrMsgTxt("\n");
        }

        if (gpoValue == 0)
        {
            m = mxCreateString("Off");
        }
        else if (gpoValue == 1)
        {
            m = mxCreateString("On");
        }
        else
        {
            m = mxCreateString("Unknown value");
        }
    }
    else if (!strcmp(parametername, "GreenGain"))
    {   /* Green Gain */
        nRetValue = PimMegaGetGreenGain(deviceId, &greenGainValue);

        if (plError(nRetValue, "getting GreenGain value"))
        {
            mexErrMsgTxt("\n");
        }
```

```
        tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(tmp);
        tmppr[0] = (U32)greenGainValue;
    }
    else if (!strcmp(parametername, "HardwareVersion"))
    {   /* Hardware Version */
        nRetValue = PimMegaGetHardwareVersion(deviceId, &hardwareVersionValue);

        if (plError(nRetValue, "getting HardwareVersion value"))
        {
            mexErrMsgTxt("\n");
        }

        tmp = mxCreateString((const char *)hardwareVersionValue.ProductID);
        tmp2 = mxCreateString((const char *)hardwareVersionValue.SerialNumber);
        tmp3 = mxCreateString((const char *)hardwareVersionValue.FirmwareVersion);
        tmp4 = mxCreateString((const char *)hardwareVersionValue.FpgaVersion);

        fieldnames[0] = "ProductID";
        fieldnames[1] = "SerialNumber";
        fieldnames[2] = "FirmwareVersion";
        fieldnames[3] = "FpgaVersion";
        m = mxCreateStructMatrix(1, 1, 4, (const char **)fieldnames);

        mxSetField(m, 0, "ProductID", tmp);
        mxSetField(m, 0, "SerialNumber", tmp2);
        mxSetField(m, 0, "FirmwareVersion", tmp3);
        mxSetField(m, 0, "FpgaVersion", tmp4);
    }
    else if (!strcmp(parametername, "ImagerChipId"))
    {   /* Imager Chip ID */
        nRetValue = PimMegaGetImagerChipId(deviceId, &imagerChipIdValue);

        if (plError(nRetValue, "getting ImagerChipId value"))
        {
            mexErrMsgTxt("\n");
        }

        m = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(m);
        tmppr[0] = (U32)imagerChipIdValue;
    }
    else if (!strcmp(parametername, "ImagerClocking"))
    {   /* Imager Clocking */
        nRetValue = PimMegaGetImagerClocking(deviceId, &imagerClockingValue);

        if (plError(nRetValue, "getting ImagerClocking value"))
        {
            mexErrMsgTxt("\n");
        }

        if (imagerClockingValue == 0x00)
        {
            m = mxCreateString("0x00\nOscilator type: External (16MHz)\nMax clock \
                                rate divided by: (no division)");
        }
        else if (imagerClockingValue == 0x01)
        {
            m = mxCreateString("0x01\nOscilator type: External (16MHz)\nMax clock \
                                rate divided by: 2");
        }
        else if (imagerClockingValue == 0x02)
        {
            m = mxCreateString("0x02\nOscilator type: External (16MHz)\nMax clock \
                                rate divided by: 4");
        }
        else if (imagerClockingValue == 0x80)
        {
            m = mxCreateString("0x80\nOscilator type: Internal (24MHz)\nMax clock \
                                rate divided by: (no division)");
        }
        else if (imagerClockingValue == 0x81)
        {
            m = mxCreateString("0x81\nOscilator type: Internal (24MHz)\nMax clock \
                                rate divided by: 2");
```

```
        }
        else if (imagerClockingValue == 0x82)
        {
            m = mxCreateString("0x82\nOscilator type: Internal (24MHz)\nMax clock \
                               rate divided by: 4");
        }
        else
        {
            m = mxCreateString("Unknown value");
        }
    }
    else if (!strcmp(parametername, "ImagerName"))
    {   /* Imager Name */
        imagerNameValue = (char *)mxCalloc(81, sizeof(char));

        nRetValue = PimMegaGetImagerName(deviceId, imagerNameValue);

        if (plError(nRetValue, "getting ImagerName value"))
        {
            mexErrMsgTxt("\n");
        }

        m = mxCreateString(imagerNameValue);
    }
    else if (!strcmp(parametername, "ImagerType"))
    {   /* Imager Type */
        nRetValue = PimMegaGetImagerType(deviceId, &imagerTypeValue);

        if (plError(nRetValue, "getting ImagerType value"))
        {
            mexErrMsgTxt("\n");
        }

        if (imagerTypeValue == PCS2112M_IMAGER)
        {
            m = mxCreateString("PCS2112M_IMAGER (Monochrome Camera)");
        }
        else if (imagerTypeValue == PCS2112C_IMAGER)
        {
            m = mxCreateString("PCS2112C_IMAGER (Color Camera)");
        }
        else
        {
            m = mxCreateString("Unknown value");
        }
    }
    else if (!strcmp(parametername, "MonoGain"))
    {   /* Mono Gain */
        nRetValue = PimMegaGetMonoGain(deviceId, &monoGainValue);

        if (plError(nRetValue, "getting MonoGain value"))
        {
            mexErrMsgTxt("\n");
        }

        m = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(m);
        tmppr[0] = (U32)monoGainValue;
    }
    else if (!strcmp(parametername, "PreviewWindowPos"))
    {   /* Preview Window Pos */
        nRetValue = PimMegaGetPreviewWindowPos(deviceId, &previewWindowPosLeftValue, \
                                               &previewWindowPosTopValue);

        if (plError(nRetValue, "getting PreviewWindowPos value"))
        {
            mexErrMsgTxt("\n");
        }

        tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(tmp);
        tmppr[0] = (long)previewWindowPosLeftValue;
        tmp2 = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr2 = mxGetPr(tmp2);
        tmppr2[0] = (long)previewWindowPosTopValue;
```

```
        fieldnames[1] = "PreviewWindowPosLeft";
        fieldnames[0] = "PreviewWindowPosTop";
        m = mxCreateStructMatrix(1, 1, 2, (const char **)fieldnames);

        mxSetField(m, 0, "PreviewWindowPosLeft", tmp);
        mxSetField(m, 0, "PreviewWindowPosTop", tmp2);
    }
    else if (!strcmp(parametername, "PreviewWindowSize"))
    {   /* Preview Window Size */
        nRetValue = PimMegaGetPreviewWindowSize(deviceId, \
                        &previewWindowSizeWidthValue, &previewWindowSizeHeightValue);

        if (plError(nRetValue, "getting PreviewWindowSize value"))
        {
            mexErrMsgTxt("\n");
        }

        tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(tmp);
        tmppr[0] = (U32)previewWindowSizeWidthValue;
        tmp2 = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr2 = mxGetPr(tmp2);
        tmppr2[0] = (U32)previewWindowSizeHeightValue;

        fieldnames[1] = "PreviewWindowSizeWidth";
        fieldnames[0] = "PreviewWindowSizeHeight";
        m = mxCreateStructMatrix(1, 1, 2, (const char **)fieldnames);

        mxSetField(m, 0, "PreviewWindowSizeWidth", tmp);
        mxSetField(m, 0, "PreviewWindowSizeHeight", tmp2);

    }
    else if (!strcmp(parametername, "RedGain"))
    {   /* Red Gain */
        nRetValue = PimMegaGetRedGain(deviceId, &redGainValue);

        if (plError(nRetValue, "getting RedGain value"))
        {
            mexErrMsgTxt("\n");
        }

        m = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(m);
        tmppr[0] = (U32)redGainValue;
    }
    else if (!strcmp(parametername, "Saturation"))
    {    /* Saturation */
        nRetValue = PimMegaGetSaturation(deviceId, &saturationValue);

        if (plError(nRetValue, "getting Saturation value"))
        {
            mexErrMsgTxt("\n");
        }

        m = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(m);
        tmppr[0] = (U32)saturationValue;
    }
    else if (!strcmp(parametername, "SerialNumber"))
    {   /* Serial Number */
        nRetValue = PimMegaGetSerialNumber(deviceId, &serialNumberValue);

        if (plError(nRetValue, "getting SerialNumber value"))
        {
            mexErrMsgTxt("\n");
        }

        m = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(m);
        tmppr[0] = (U32)serialNumberValue;
    }
    else if (!strcmp(parametername, "SoftwareVersion"))
    {   /* Software Version */
        nRetValue = PimMegaGetSoftwareVersion(deviceId, &softwareVersionValue);

        if (plError(nRetValue, "getting SoftwareVersion value"))
```

```
        {
            mexErrMsgTxt("\n");
        }

        m = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(m);
        tmppr[0] = (U32)softwareVersionValue;
    }
    else if (!strcmp(parametername, "SubWindow"))
    {   /* Sub Window */
        nRetValue = PimMegaGetSubWindow(deviceId, &subWindowDecimationValue, \
                                        &subWindowStartColumnValue, \
                                        &subWindowStartRowValue, \
                                        &subWindowNumberColumnsValue, \
                                        &subWindowNumberRowsValue);

        if (plError(nRetValue, "getting SubWindow value"))
        {
            mexErrMsgTxt("\n");
        }

        if (subWindowDecimationValue == PCS2112_NO_DECIMATION)
        {
            tmp = mxCreateString("PCS2112_NO_DECIMATION");
        }
        else if (subWindowDecimationValue == PCS2112_DECIMATE_BY_2)
        {
            tmp = mxCreateString("PCS2112_DECIMATE_BY_2");
        }
        else if (subWindowDecimationValue == PCS2112_DECIMATE_BY_4)
        {
            tmp = mxCreateString("PCS2112_DECIMATE_BY_4");
        }
        else
        {
            tmp = mxCreateString("Unknown value");
        }

        tmp2 = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr2 = mxGetPr(tmp2);
        tmppr2[0] = (U32)subWindowStartColumnValue;
        tmp3 = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr3 = mxGetPr(tmp3);
        tmppr3[0] = (U32)subWindowStartRowValue;
        tmp4 = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr4 = mxGetPr(tmp4);
        tmppr4[0] = (U32)subWindowNumberColumnsValue;
        tmp5 = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr5 = mxGetPr(tmp5);
        tmppr5[0] = (U32)subWindowNumberRowsValue;

        fieldnames[0] = "SubWindowDecimation";
        fieldnames[2] = "SubWindowStartColumn";
        fieldnames[1] = "SubWindowStartRow";
        fieldnames[4] = "SubWindowNumberColumns";
        fieldnames[3] = "SubWindowNumberRows";

        m = mxCreateStructMatrix(1, 1, 5, (const char **)fieldnames);

        mxSetField(m, 0, "SubWindowDecimation", tmp);
        mxSetField(m, 0, "SubWindowStartColumn", tmp2);
        mxSetField(m, 0, "SubWindowStartRow", tmp3);
        mxSetField(m, 0, "SubWindowNumberColumns", tmp4);
        mxSetField(m, 0, "SubWindowNumberRows", tmp5);
    }
    else if (!strcmp(parametername, "SubWindowPos"))
    {   /* Sub Window Pos */
        nRetValue = PimMegaGetSubWindowPos(deviceId, &subWindowPosStartColumnValue, \
                                           &subWindowPosStartRowValue);

        if (plError(nRetValue, "getting SubWindowPos value"))
        {
            mexErrMsgTxt("\n");
        }

        tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
```

```
        tmppr = mxGetPr(tmp);
        tmppr[0] = (U32)subWindowPosStartColumnValue;
        tmp2 = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr2 = mxGetPr(tmp2);
        tmppr2[0] = (U32)subWindowPosStartRowValue;

        fieldnames[1] = "SubWindowPosStartColumn";
        fieldnames[0] = "SubWindowPosStartRow";
        m = mxCreateStructMatrix(1, 1, 2, (const char **)fieldnames);

        mxSetField(m, 0, "SubWindowPosStartColumn", tmp);
        mxSetField(m, 0, "SubWindowPosStartRow", tmp2);

    }
    else if (!strcmp(parametername, "SubWindowSize"))
    {    /* Sub Window Size */
        nRetValue = PimMegaGetSubWindowSize(deviceId, &subWindowSizeDecimationValue, \
                                            &subWindowSizeWidthValue, \
                                            &subWindowSizeHeightValue);

        if (plError(nRetValue, "getting SubWindowSize value"))
        {
            mexErrMsgTxt("\n");
        }

        if (subWindowSizeDecimationValue == PCS2112_NO_DECIMATION)
        {
            tmp = mxCreateString("PCS2112_NO_DECIMATION");
        }
        else if (subWindowSizeDecimationValue == PCS2112_DECIMATE_BY_2)
        {
            tmp = mxCreateString("PCS2112_DECIMATE_BY_2");
        }
        else if (subWindowSizeDecimationValue == PCS2112_DECIMATE_BY_4)
        {
            tmp = mxCreateString("PCS2112_DECIMATE_BY_4");
        }
        else
        {
            tmp = mxCreateString("Unknown value");
        }

        tmp2 = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr2 = mxGetPr(tmp2);
        tmppr2[0] = (U32)subWindowSizeWidthValue;
        tmp3 = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr3 = mxGetPr(tmp3);
        tmppr3[0] = (U32)subWindowSizeHeightValue;

        fieldnames[0] = "SubWindowSizeDecimation";
        fieldnames[2] = "SubWindowSizeWidth";
        fieldnames[1] = "SubWindowSizeHeight";
        m = mxCreateStructMatrix(1, 1, 3, (const char **)fieldnames);

        mxSetField(m, 0, "SubWindowSizeDecimation", tmp);
        mxSetField(m, 0, "SubWindowSizeWidth", tmp2);
        mxSetField(m, 0, "SubWindowSizeHeight", tmp3);

    }
    else if (!strcmp(parametername, "Timeout"))
    {   /* Timeout */
        nRetValue = PimMegaGetTimeout(deviceId, &timeoutValue);

        if (plError(nRetValue, "getting Timeout value"))
        {
            mexErrMsgTxt("\n");
        }

        m = mxCreateDoubleMatrix(1, 1, mxREAL);
        tmppr = mxGetPr(m);
        tmppr[0] = (U32)timeoutValue;
    }
    else if (!strcmp(parametername, "VideoMode"))
    {   /* Video Mode */
        nRetValue = PimMegaGetVideoMode(deviceId, &videoModeValue);
```

```
        if (plError(nRetValue, "getting VideoMode value"))
        {
            mexErrMsgTxt("\n");
        }

        if (videoModeValue == STILL_MODE)
        {
            m = mxCreateString("STILL_MODE");
        }
        else if (videoModeValue == VIDEO_MODE)
        {
            m = mxCreateString("VIDEO_MODE");
        }
        else
        {
            m = mxCreateString("Unknown value");
        }
    }
    else if (!strcmp(parametername, "GrabColorConversion"))
    {   /* GrabColorConversion is kept in plDevices and used in plGrab */
        rhs[0] = mxCreateString("getpar");
        rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
        px = mxGetPr(rhs[1]);
        px[0] = (U32)serialNumber;
        rhs[2] = mxCreateString("GrabColorConversion");

        mexCallMATLAB(1, lhs, 3, rhs, "plDevices");
        grabColorConversionValue = (U32)mxGetScalar(lhs[0]);

        switch (grabColorConversionValue)
        {
            case (BAYER_2BY2_COLOR):      m = mxCreateString("BAYER_2BY2_COLOR");
                                           break;
            case (BAYER_3BY3_COLOR):      m = mxCreateString("BAYER_3BY3_COLOR");
                                           break;
            case (BAYER_3BY3GGRAD_COLOR): m = mxCreateString("BAYER_3BY3GGRAD_COLOR");
                                           break;
            case (BAYER_2PASSGRAD_COLOR): m = mxCreateString("BAYER_2PASSGRAD_COLOR");
                                           break;
            case (BAYER_2PASSADAPT_COLOR): \
                                          m = mxCreateString("BAYER_2PASSADAPT_COLOR");
                                           break;
            case (BAYER_VARGRAD_COLOR):   m = mxCreateString("BAYER_VARGRAD_COLOR");
                                           break;
            case (BAYER_2BY2_MONO):       m = mxCreateString("BAYER_2BY2_MONO");
                                           break;
            case (BAYER_3BY3_MONO):       m = mxCreateString("BAYER_3BY3_MONO");
                                           break;
            case (BAYER_ADAPT_MONO):      m = mxCreateString("BAYER_ADAPT_MONO");
                                           break;
            case (BAYER_NO_CONVERSION):   m = mxCreateString("BAYER_NO_CONVERSION");
                                           break;
            default:                      m = mxCreateString("Unknown value");
        }
    }
    else if (!strcmp(parametername, "GrabOutputType"))
    {   /* GrabOutputType is kept in plDevices and used in plGrab */
        rhs[0] = mxCreateString("getpar");
        rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
        px = mxGetPr(rhs[1]);
        px[0] = (U32)serialNumber;
        rhs[2] = mxCreateString("GrabOutputType");

        mexCallMATLAB(1, lhs, 3, rhs, "plDevices");
        grabOutputTypeValue = (U32)mxGetScalar(lhs[0]);

        switch (grabOutputTypeValue)
        {
            case (RAW):     m = mxCreateString("RAW");
                            break;
            case (IMAGE):   m = mxCreateString("IMAGE");
                            break;
            case (RGB24):   m = mxCreateString("RGB24");
                            break;
            default:        m = mxCreateString("Unknown value");
        }
```

```
    }
    else
    {
        mexPrintf("plGetValue: Unknown parameter.\n");
        mexPrintf("possible parameters are:\n");
        mexPrintf("DeviceID\n");
        mexPrintf("BlueGain\n");
        mexPrintf("CurrentFrameRate\n");
        mexPrintf("DataTransferSize\n");
        mexPrintf("Exposure\n");
        mexPrintf("ExposureTime\n");
        mexPrintf("Gamma\n");
        mexPrintf("Gpo\n");
        mexPrintf("GreenGain\n");
        mexPrintf("HardwareVersion\n");
        mexPrintf("ImagerChipId\n");
        mexPrintf("ImagerClocking\n");
        mexPrintf("ImagerName\n");
        mexPrintf("ImagerType\n");
        mexPrintf("MonoGain\n");
        mexPrintf("PreviewWindowPos\n");
        mexPrintf("PreviewWindowSize\n");
        mexPrintf("RedGain\n");
        mexPrintf("Saturation\n");
        mexPrintf("SerialNumber\n");
        mexPrintf("SoftwareVersion\n");
        mexPrintf("SubWindow\n");
        mexPrintf("SubWindowPos\n");
        mexPrintf("SubWindowSize\n");
        mexPrintf("Timeout\n");
        mexPrintf("VideoMode\n");
        mexPrintf("GrabColorConversion\n");
        mexPrintf("GrabOutputType\n");
        mexErrMsgTxt("\n");
    }

    returnArray[0] = m;
    return;
}
```

D.9 plGetValue.h

```
/*********************************************************************/
/* Filename:    plGetValue.h                                         */
/* Description: Header file for plGetValue.cpp                       */
/* Authors:     L.I.Oei, M.A.E.Bakker                                */
/* Date:        2002/08/06                                           */
/* Updates:                                                          */
/*********************************************************************/

void plGetValue(mxArray *returnArray[], U32 serialNumber, char* parametername);
```

D.10 plGrab.cpp

```
/*********************************************************************/
/* Filename:    plGrab.cpp                                           */
/* Description: Source code for the function plGrab                  */
/* Authors:     M.A.E.Bakker, L.I.Oei                                */
/*              maarten@panic.et.tudelft.nl, l.i.oei@its.tudelft.nl  */
/* Date:        2002/08/08                                           */
/* Updates:                                                          */
/*********************************************************************/
/* plGrab grabs a frame from the PixeLINK device and places it into a*/
/* mxArray.                                                          */
/*********************************************************************/
/* Input:    prhs[0] =  (U32) serialnumber for the device            */
/*                      (struct) handle for the device               */
/* Output:    plhs[0] = (mxArray) grabbed image                      */
/* Syntax:    imagematrix = plGrab(serialnumber)                     */
/*            imagematrix = plGrab(handle)                           */
/*********************************************************************/

#define WIN32_LEAN_AND_MEAN    /* Exclude rarely-used stuff from Windows */
#include <windows.h>

#include "Y:\soft95\matlab6\extern\include\mex.h"
/* link with Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmex.lib */
/* and libmx.lib */

#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"
/* link with Y:\software\framegrag\pixelink\api\pimmegaapi.lib */

#include "..\plError.h"    /* the pixelink error checking subroutine */
#include "..\plTypes.h"

#ifdef __cplusplus
    extern "C" {
#endif

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    PXL_RETURN_CODE nRetValue;       /* the return codes of the pixelink functions */

    mxArray *serialNumberField;
    U32        serialNumber = -1;    /* the serial number of the device */

    mxArray *lhs[1], *rhs[3];        /* to call fgdevices */
    double *px;
    int isOpen = 0;

    HANDLE deviceId;                 /* the deviceId PimMegaInitialize returns */

    U32 currentVideoMode; /* to get the current video mode; if currentVideoMode != \
                             VIDEO_MODE, set VideoMode to 'VIDEO_MODE' */

    /* image properties */
    U32 imagerType;       /* ImagerType: monochrome or color */
    U32 dataTransferSize; /* DataTransferSize: DATA_8BIT_SIZE or DATA_16BIT_SIZE */
    U32 decimation, width, height;

    U32 grabColorConversion;        /* the Bayer conversion to be used */
    int grabOutputType;          /* what sort of data should be returned to Matlab */

    int *size; /* array used to build the Matlab array (with mxCreateNumericArray) */
    int dim;                         /* number of dimensions of the Matlab array */

    U32 rawImgSize, rawImgByteSize, RGB24ImgSize;

    void *imgOutPtr = NULL; /* pointer to the mxArray in which the image should be \
                               stored (matlab array) */
    void *imgCapturePtr = NULL; /* pointer to the capture array for conversion from \
                                   Bayer or I to RGB24 */
    void *img24BppPtr = NULL; /* pointer to capture array for conversion from RGB24 \
                                 to MatLAB image */

    U32 pixelWidth, pixelHeight;
```

```
    U32 i, j;

    if ((nrhs == 1) && (nlhs == 1))
    {

/*********************************************************************/
/*      Get the serialNumber and deviceId                            */
/*********************************************************************/

        /* get the serial number */
        if (mxIsStruct(prhs[0]))
        {
            serialNumberField = mxGetField(prhs[0], 0, "SerialNumber");
            serialNumber = (U32)mxGetScalar(serialNumberField);
        }
        else if (mxIsDouble(prhs[0]))
        {
            serialNumber = (U32)mxGetScalar(prhs[0]);
        }
        else
        {
            mexPrintf("PixeLINK Interface error.\n");
           mexPrintf("plGrab: Wrong arguments given. First argument should be the \
                      device handle or serial number.\n");
            mexErrMsgTxt("\n");
        }

        /* get the deviceId */
        rhs[0] = mxCreateString("get");

        rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
        px = mxGetPr(rhs[1]);
        px[0] = (U32)serialNumber;

        mexCallMATLAB(1, lhs, 2, rhs, "plDevices");
        deviceId = (HANDLE *)(int)mxGetScalar(lhs[0]);

/*********************************************************************/
/*      Open the video stream and set VideoMode to 'VIDEO_MODE'      */
/*********************************************************************/

        /* open video stream first to do acquisition */
        nRetValue = PimMegaStartVideoStream(deviceId);
        if (plError(nRetValue, "calling PimMegaStartVideoStream"))
        {
            mexErrMsgTxt("\n");
        }

        /* make sure camera is in video mode, it will give better results than stil
         * mode, without shutter or flashlight */
        nRetValue = PimMegaGetVideoMode(deviceId, &currentVideoMode);
        if (plError(nRetValue, "calling PimMegaGetVideoMode"))
        {
            mexErrMsgTxt("\n");
        }
        if (currentVideoMode != VIDEO_MODE)
        {
            nRetValue = PimMegaSetVideoMode(deviceId, VIDEO_MODE);
            if (plError(nRetValue, "calling PimMegaSetVideoMode"))
            {
                mexErrMsgTxt("\n");
            }
            mexPrintf("Setting VideoMode to 'VIDEO_MODE' for better results.\n");
        }

/*********************************************************************/
/*      Get some camera settings                                     */
/*********************************************************************/

        /* get image properties */
        /* monochrome or color */
        nRetValue = PimMegaGetImagerType(deviceId,&imagerType);
        if (plError(nRetValue, "calling PimMegaGetImagerType"))
        {
            PimMegaStopVideoStream(deviceId);
            mexErrMsgTxt("\n");
```

```
        }

        /* DATA_8BIT_SIZE or DATA_16BIT_SIZE */
        nRetValue = PimMegaGetDataTransferSize(deviceId, &dataTransferSize);
        if (plError(nRetValue, "calling PimMegaGetDataTransferSize"))
        {
            PimMegaStopVideoStream(deviceId);
            mexErrMsgTxt("\n");
        }
        nRetValue = PimMegaGetSubWindowSize(deviceId, &decimation, &width, &height);
        if (plError(nRetValue, "calling PimMegaGetSubWindowSize"))
        {
            PimMegaStopVideoStream(deviceId);
            mexErrMsgTxt("\n");
        }
        pixelWidth = (U32) (width / decimation);
        pixelHeight = (U32) (height / decimation);

        /* get postprocessing method from stored configuration in plDevices (needed
         * with processed color images) */
        /* grabColorConversion =
         * plDevices("getpar",serialNumber,"GrabColorConversion") */
        rhs[0] = mxCreateString("getpar");
        rhs[1] = mxCreateDoubleMatrix(1, 1, mxREAL);

        px = mxGetPr(rhs[1]);
        px[0] = (U32)serialNumber;
        rhs[2] = mxCreateString("GrabColorConversion");

        mexCallMATLAB(1, lhs, 3, rhs, "plDevices");

        /* which Bayer conversion should be used? */
        grabColorConversion = (U32)mxGetScalar(lhs[0]);
        /* get raw output flag from stored configuration in plDevices */
        /* grabOutputType = plDevices("getpar", serialNumber, "GrabOutputType") */
        rhs[0] = mxCreateString("getpar");
        rhs[1] = mxCreateDoubleMatrix(1, 1, mxREAL);

        px = mxGetPr(rhs[1]);
        px[0] = (U32)serialNumber;
        rhs[2] = mxCreateString("GrabOutputType");

        mexCallMATLAB(1, lhs, 3, rhs, "plDevices");

        /* what sort of data should be returned to Matlab? */
        grabOutputType = (int)mxGetScalar(lhs[0]);
/*****************************************************************/
/*      Allocate memory for the arrays in which the image is stored */
/*****************************************************************/

        /* calculate array properties
         *
         * 3 arrays are used:
         *   - caputure array: array returned by the pixelink camera; size (pixelWidth
         *                     * pixelHeight)
         *   - matlab array:   array returned to matlab; size depends on postprocessing
         *   - RGB24 array:    array in which the output of the PimMegaConvert function
         *                     is stored (not with raw images)
         *
         * size:      array used to build the Matlab array (with mxCreateNumericArray)
         * dim:       number of dimensions of the Matlab array
         */
        switch (grabOutputType)
        {
            case (RAW):
            { /* raw output: image is returned to matlab exactly the way it is
               * captured. matlab array should be (pixelWidth * pixelHeight) for both
               * colour and monochrome
               */
                size = (int *)mxCalloc(2, sizeof(int));
                size[0] = pixelWidth;
                size[1] = pixelHeight;
                dim = 2;    /* Raw image array has 2 dimensions */
                break;
            }
            case (IMAGE):
```

```
        {   /* processed output: image is processed before it is returned to
             * matlab. matlab array should be (3 * pixelWidth * pixelHeight)
             */
            size = (int *)mxCalloc(3, sizeof(int));
            size[0] = pixelHeight;
            size[1] = pixelWidth;
            size[2] = 3;
            dim = 3;    /* RGB image array has 3 dimensions */
            break;
        }
        case (RGB24):
        {   /* RGB24 output: image is returned as a consequent piece of memory
             * of size (3 * pixelWidth * pixelHeight)
             */
            size = (int *)mxCalloc(2, sizeof(int));
            size[0] = (pixelWidth * pixelHeight * 3);
            size[1] = 1;
            dim = 2;    /* RGB image array has 2 dimensions */
            break;
        }
        default:
        {
           mexPrintf("plGrab: unknown GrabOutputType, should be RAW, IMAGE or \
                      RGB24.");
            PimMegaStopVideoStream(deviceId);
            mexErrMsgTxt("\n");
        }
    }

    /* number of memory locations needed for storing the raw image */
    rawImgSize = (pixelWidth * pixelHeight);
    RGB24ImgSize = rawImgSize * 3;

    /* allocate memory for the matlab and capture arrays */
    switch (dataTransferSize)
    {    /* dimension matrix for pixeldepth */
        case (DATA_8BIT_SIZE): /* 8 bit pixel depth, use 8 bit integer per \
                                  pixel */
                               plhs[0] = mxCreateNumericArray(dim, size, \
                                                              mxUINT8_CLASS, mxREAL);
                               imgCapturePtr = (U8 *)mxCalloc(rawImgSize, \
                                                              sizeof(U8));
                               rawImgByteSize = rawImgSize;
                               break;
        case (DATA_16BIT_SIZE): /* 10 bit pixel depth,use 16 bit integer per \
                                   pixel */
                               if (grabOutputType == RAW)
                                   plhs[0] = mxCreateNumericArray(dim, size, \
                                                                  mxUINT16_CLASS, mxREAL);
                              else /* IMAGE or RGB24 is always 8 bit/subpixel */
                                   plhs[0] = mxCreateNumericArray(dim, size, \
                                                                  mxUINT8_CLASS, mxREAL);
                               imgCapturePtr = (U16 *)mxCalloc(rawImgSize,  \
                                                               sizeof(U16));
                               rawImgByteSize = rawImgSize * 2;
                               break;
        default:               mexPrintf("plGrab: Unknown pixel depth, should \
                                          be DATA_8BIT_SIZE or DATA_16BIT_SIZE.\n");
                               PimMegaStopVideoStream(deviceId);
                               mexErrMsgTxt("\n");
                               return;
    }
    if (plhs[0] == NULL)
    {   /* mxCreateNumericArray returns NULL in case of problems, mxCalloc doesn't
          * return at all
          */
         PimMegaStopVideoStream(deviceId);
        mexErrMsgTxt("plGrab: Cannot allocate enough MatLAB memory to store the \
                      captured image.\n");
        return;
    }
    imgOutPtr = (int *)mxGetPr(plhs[0]);
    /* allocate memory for the RGB24 array if needed */
    if (grabOutputType != RAW) img24BppPtr = (U8 *)mxCalloc(RGB24ImgSize, \
                                                            sizeof(U8));
```

```
/*********************************************************************/
/*      Capture the image and close the video stream                 */
/*********************************************************************/

        /* capture the image */
        nRetValue = PimMegaReturnVideoData(deviceId, rawImgByteSize, imgCapturePtr);
        if (plError(nRetValue, "calling PimMegaReturnVideoData"))
        {
            PimMegaStopVideoStream(deviceId);
            mexErrMsgTxt("\n");
        }

        /* close the video stream */
        nRetValue = PimMegaStopVideoStream(deviceId);
        if (plError(nRetValue, "calling PimMegaStopVideoStream"))
        {
            mexErrMsgTxt("\n");
        }

/*********************************************************************/
/*      Postprocess the image (if necessary) and return the image    */
/*********************************************************************/

        /* return the image as is specified by the GrabColorConversion and
         * GrabOutputType variables
         */
        if (grabOutputType == RAW)
        {    /* 8 bits transfer, just return the image */
            if (dataTransferSize == DATA_8BIT_SIZE)
                memcpy(imgOutPtr, imgCapturePtr, rawImgByteSize);
            else if (dataTransferSize == DATA_16BIT_SIZE)
            {    /* 16 bits transfer: shift some bits */
                j = 0;
                for (i = 0; i < rawImgSize; i++)
                {
                    ((PU16)imgOutPtr)[i] = (((PU8)imgCapturePtr)[j] >> 6) + \
                                                (U16)(((PU8)imgCapturePtr)[j+1]) * 4;
                    j+=2;
                }
            }
            else
            {
                mexPrintf("plGrab: unknown pixel depth.\n");
                mexErrMsgTxt("\n");
                return;
            }
        }
        else /* grabOutputType != RAW */
        {    /* postprocessing by PimMegaConvert */
            if ((dataTransferSize == DATA_8BIT_SIZE)
                 && (imagerType == PCS2112M_IMAGER))
            {    /* 8 bit, monochrome */
                nRetValue = PimMegaConvertMono8BppTo24Bpp(deviceId, \
                       (PU8)imgCapturePtr, pixelWidth, pixelHeight, (PU8)img24BppPtr);
                plError(nRetValue, "converting raw mono 8 to 24 bpp");
            }
            else if ((dataTransferSize == DATA_8BIT_SIZE) && (imagerType == \
                                                                PCS2112C_IMAGER))
            {    /* 8 bit, color */
                nRetValue = PimMegaConvertColor8BppTo24Bpp(deviceId, \
                                        (PU8)imgCapturePtr, pixelWidth, pixelHeight, \
                                         (PU8)img24BppPtr, grabColorConversion);
                plError(nRetValue, "converting raw colour 8 to 24 bpp");
            }
            else if ((dataTransferSize == DATA_16BIT_SIZE) && (imagerType == \

PCS2112M_IMAGER))
            {    /* 16 bit, monochrome */
                nRetValue = PimMegaConvertMono16BppTo24Bpp(deviceId, \
                                        (PU8)imgCapturePtr, pixelWidth, pixelHeight, \
                                                                (PU8)img24BppPtr);
                plError(nRetValue, "converting raw mono 16 to 24 bpp");
            }
            else if ((dataTransferSize == DATA_16BIT_SIZE) && (imagerType == \
                                                                PCS2112C_IMAGER))
            {    /* 16 bit, color */
```

```
                nRetValue = PimMegaConvertColor16BppTo24Bpp(deviceId, \
                                    (PU8)imgCapturePtr, pixelWidth, pixelHeight, \
                                            (PU8)img24BppPtr, grabColorConversion);
                plError(nRetValue, "converting raw colour 16 to 24 bpp");
            }
            else
            {
                mexPrintf("plGrab: Error while postprocessing image.\n");
                mexPrintf("Unknown imager type or pixel depth.\n");
                mexErrMsgTxt("\n");
                return;
            }

            if (grabOutputType == IMAGE)
            {   /* transpose the RGB24 array (rgbrgbrgb) to a MatLAB image array
                 * (rrrgggbbb); copy image into the matlab array in such a way that it
                 * is displayed correctly
                 */
                for (i = 0; i < pixelHeight; i++)
                {
                    for (j = 0; j < pixelWidth; j++)
                    {
                      ((PU8)imgOutPtr)[((j+1) * pixelHeight) - i - 1] = \
                                    ((PU8)img24BppPtr)[((i * pixelWidth) + j) * 3];
                      ((PU8)imgOutPtr)[((j+1) * pixelHeight) - i - 1 + (pixelWidth * \
                                                                pixelHeight)] = \
                                    ((PU8)img24BppPtr)[(((i * pixelWidth) + j) * 3) + 1];
                      ((PU8)imgOutPtr)[((j+1) * pixelHeight) - i - 1 + (pixelWidth * \
                                                            pixelHeight * 2)] = \
                                    ((PU8)img24BppPtr)[(((i * pixelWidth) + j) * 3) + 2];
                    }
                }
            }
            else if (grabOutputType == RGB24)
            {    /* just copy the RGB24 image into the matlab array */
                memcpy(imgOutPtr, img24BppPtr, RGB24ImgSize);
            }
            else /* grabOutputType is neither RAW, RGB24 nor IMAGE */
            {
                mexPrintf("plGrab: unknown value of GrabOutputType: should be RAW, \
                            RGB24 or IMAGE.\n");
                mexErrMsgTxt("\n");
                return;
            }
        }
    }
    else /* !((nrhs==1) && (nlhs == 1)) */
    {
        mexPrintf("plGrab: Wrong arguments given. Calling syntax: image = \
                    plGrab(handle), or image = plGrab(serialnumber)\n");
        mexErrMsgTxt("\n");
    }
}

#ifdef __cplusplus
    }    /* extern "C" */
#endif
```

D.11 plIsOpen.cpp

```
/*********************************************************************/
/* Filename:    plIsOpen.cpp                                         */
/* Description: Source code for the function plIsOpen                */
/* Authors:     M.A.E.Bakker, L.I.Oei                                */
/*              maarten@panic.et.tudelft.nl, l.i.oei@its.tudelft.nl  */
/* Date:        2002/08/07                                           */
/* Updates:                                                          */
/*********************************************************************/
/* plIsOpen checks whether the device has been opened. It returns 0 if  */
/* the device has not been opened, and 1 if it has been opened.      */
/*********************************************************************/
/* Input:    prhs[0] = (struct)   device handle, or                  */
/*                     (U32)     serial number of the device         */
/* Output:   plhs[0] = (double)   0 if the device has not beeen opened  */
/*                                1 if the device has been opened     */
/* Syntax:   plIsOpen(handle) or plIsOpen(serialnumber)              */
/*********************************************************************/

#define WIN32_LEAN_AND_MEAN        // Exclude rarely-used stuff from Windows headers
#include <windows.h>

#include "Y:\soft95\matlab6\extern\include\mex.h"
/* with Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmex.lib */
/* and  Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmx.lib */

#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"
/* with Y:\software\framegrag\pixelink\api\pimmegaapi.lib */

#ifdef __cplusplus
    extern "C" {
#endif

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    mxArray *serialNumberField;
    U32        serialNumber = -1;        /* the serial number of the device */
    mxArray *rhs[2];
    double    *px;

    /* prhs[0] should be the device handle (struct) or serial number (int) */
    if (nrhs==1)
    {
        /* get the serial number */
        if (mxIsStruct(prhs[0]))
        {
            serialNumberField = mxGetField(prhs[0], 0, "SerialNumber");
            serialNumber = (U32)mxGetScalar(serialNumberField);
        }
        else if (mxIsDouble(prhs[0]))
        {
            serialNumber = (U32)mxGetScalar(prhs[0]);
        }
        else
        {
            mexPrintf("PixeLINK Interface error.\n");
            mexPrintf("plIsOpen: Wrong arguments given. One argument needed (device \
                       handle or serial number).\n");
            mexErrMsgTxt("\n");
        }

        /* check whether the device is open */
        rhs[0] = mxCreateString("isopen");

        rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
        px = mxGetPr(rhs[1]);
        px[0] = (U32)serialNumber;

        mexCallMATLAB(1, plhs, 2, rhs, "plDevices");
    }
    else /* ! (nrhs==1) */
    {
        mexPrintf("PixeLINK Interface error.\n");
```

```
        mexPrintf("plIsOpen: Wrong arguments given. One argument needed (device \
                  handle or serial number).\n");
        mexErrMsgTxt("\n");
    }
}

#ifdef __cplusplus
    }    /* extern "C" */
#endif
```

D.12 plOpen.cpp

```
/*********************************************************************/
/* Filename:    plOpen.cpp                                           */
/* Description: Source code for the function plOpen                  */
/* Authors:     M.A.E.Bakker, L.I.Oei                                */
/*              maarten@panic.et.tudelft.nl, l.i.oei@its.tudelft.nl  */
/* Date:        2002/08/08                                           */
/* Updates:                                                          */
/*********************************************************************/
/* plOpen opens a pixelink device and returns the device's handle.   */
/*********************************************************************/
/* Input:   prhs[0] = (int)    serial number of the device to be opened */
/* Output:  plhs[0] = (struct) handle for the device                 */
/* Syntax:  m = plOpen(serialnumber)                                 */
/*********************************************************************/


#define WIN32_LEAN_AND_MEAN    /* Exclude rarely-used stuff from Windows headers */
#include <windows.h>

#include "Y:\soft95\matlab6\extern\include\mex.h"
/* with Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmex.lib */
/* and  Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmx.lib */

#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"
/* with Y:\software\framegrag\pixelink\api\pimmegaapi.lib */

#include "..\plCreateDeviceHandle.h"
#include "..\plError.h"

#ifdef __cplusplus
    extern "C" {
#endif

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    PXL_RETURN_CODE nRetValue;        /* the return codes of the pixelink functions */

    int        isOpen = 0;

    char   *deviceAlias = "PixeLINK(tm) 1394 Camera";           /* device alias */

    /* the number of attached imaging devices with the specified alias */
    U32    numberDevices = -1;
    U32    index = 0;                  /* index of the device (default = 0) */
    bool   found = false;
    HANDLE    deviceId = 0;            /* the deviceId PimMegaInitialize returns */

    U32    serialNumber = -1;          /* the serial number of the device */
    U32    thisSerialNumber = -1;
    int    i = 0;
    U32    availableSerialNumbers[32]; /* to store all available serial numbers */

    mxArray *lhs[1], *rhs[3];          /* to call fgdevices */
    double *px;

    /* prhs[0] should be the serial number */
    if ((nrhs==1) && mxIsDouble(prhs[0]))
    {
        /* get the serial number */
        serialNumber = (U32)mxGetScalar(prhs[0]);

        /* check whether the device has been opened before */
        rhs[0] = mxCreateString("isopen");

        rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
        px = mxGetPr(rhs[1]);
        px[0] = (U32)serialNumber;

        mexCallMATLAB(1, lhs, 2, rhs, "plDevices");
        isOpen = (int)mxGetScalar(lhs[0]);

        /* open the device if it hasn't already been opened */
```

```
if (!isOpen)
{
    /* get the number of attached imaging devices */
    nRetValue = PimMegaGetNumberDevices(deviceAlias, &numberDevices);
    if (ApiSuccess != nRetValue)
    {
        mexPrintf("PixeLINK Interface error.\n");
       mexPrintf("plOpen: Error on checking number of available devices.\n");
        mexErrMsgTxt("\n");
    }
    if (numberDevices < (unsigned long) 1)
    {
        mexPrintf("plOpen: No devices available.\n");
        mexErrMsgTxt("\n");
    }

    /* initialize array with available serial numbers with 0 */
    for (i = 0; i < 32; i++)
        availableSerialNumbers[i] = 0;

    /* open the device with the correct serial number */
    for (index = (unsigned long) 0; (index < numberDevices) && !found; \
         index++)
    {
        /* initialize the device */
        nRetValue = PimMegaInitialize(deviceAlias, index, &deviceId);
        if (plError(nRetValue, "initializing device"))
        {
            mexErrMsgTxt("\n");
        }

        /* get the device's serial number ... */
        nRetValue = PimMegaGetSerialNumber(deviceId, &thisSerialNumber);
        if (plError(nRetValue, "getting serial number"))
        {
            mexErrMsgTxt("\n");
        }

        if (index < 32)
            availableSerialNumbers[index] = thisSerialNumber;

        /* check whether this is the device we're looking for */
        if (serialNumber == thisSerialNumber)
        {
            found = true;
            break;            /* if the correct device is found: break */
        }

        /* if this was not the correct device: uninitialize the device */
        nRetValue = PimMegaUninitialize(&deviceId);
        if (plError(nRetValue, "closing the device"))
        {
            mexErrMsgTxt("\n");
        }
    }

    if (!found)
    {
        mexPrintf("plOpen: No device found with serial number %lu.\n", \
                  serialNumber);
        mexPrintf("Available serial numbers:\n");
        for (i = 0; i < 32; i++)
        {
            if (availableSerialNumbers[i] == (unsigned long)0)
            {
                break;
            }
            mexPrintf("%lu\n", availableSerialNumbers[i]);
        }
        mexErrMsgTxt("\n");
    }


    /* update the open device array in plDevices */
    rhs[0] = mxCreateString("add");
```

```
            rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
            px = mxGetPr(rhs[1]);
            px[0] = (U32)serialNumber;

            rhs[2] = mxCreateDoubleMatrix(1,1,mxREAL);
            px = mxGetPr(rhs[2]);
            px[0] = (int)deviceId;

            mexCallMATLAB(1, lhs, 3, rhs, "plDevices");
        } /* end of opening the device */
        else
        {   /* device was already open, print a warning, then proceed to return the \
               handle */
            mexPrintf("plOpen: device already open, returning device handle.\n");
        }

        /* create the device handle (return structure) */
        plCreateDeviceHandle(lhs, serialNumber);
        plhs[0] = lhs[0];
    }
    else /* ! ((nrhs==1) && mxIsDouble(prhs[0])) */
    {
        mexPrintf("PixeLINK Interface error.\n");
        mexPrintf("plOpen: Wrong arguments given. One argument needed (serial \
                   number).\n");
        mexErrMsgTxt("\n");
    }
}

#ifdef __cplusplus
    }    /* extern "C" */
#endif
```

D.13 plPrintPossibleValues.cpp

```
/*********************************************************************/
/* Filename:    plPrintPossibleValues.cpp                            */
/* Description: Source code for plPrintPossibleValues                */
/* Authors:     M.A.E.Bakker, L.I.Oei                                */
/*              maarten@panic.et.tudelft.nl, l.i.oei@its.tudelft.nl  */
/* Date:        2002/08/09                                           */
/* Updates:                                                          */
/*********************************************************************/
/* plPrintPossibleValues: prints the possible values for 'parametername'*/
/* called when plSet gets 2 arguments                                */
/*********************************************************************/
/* Input:    parametername (string) name of the parameter            */
/* Output:   -                                                       */
/* Syntax:   plPrintPossibleValues(parametername)                    */
/*********************************************************************/

#include <windows.h>

#include "Y:\soft95\matlab6\extern\include\mex.h"
/* with Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmex.lib */
/* and  Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmx.lib */

#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"
/* with Y:\software\framegrag\pixelink\api\pimmegaapi.lib */

void plPrintPossibleValues(char* parametername)
{
    /* BlueGain */
    if (!strcmp(parametername, "BlueGain"))
    {
        mexPrintf("BlueGain: U32\n");
        mexPrintf("Value should not exceed %lu.\n", PCS2112_MAX_GAIN);
    }
    /* DataTransferSize */
    else if (!strcmp(parametername, "DataTransferSize"))
    {
        mexPrintf("DataTransferSize: string\n");
        mexPrintf("Valid values are 'DATA_8BIT_SIZE' and 'DATA_16BIT_SIZE'.\n");
    }
    /* Exposure */
    else if (!strcmp(parametername, "Exposure"))
    {
        mexPrintf("Exposure: U32\n");
        mexPrintf("Value must not exceed %lu.\n", PCS2112_MAX_EXPOSURE);
    }
    /* ExposureTime */
    else if (!strcmp(parametername, "ExposureTime"))
    {
        mexPrintf("ExposureTime: float fExposure, \
                  string bChangeClockSpeed='FALSE'\n");
        mexPrintf("fExposure is the exposure time, in milliseconds.\n");
        mexPrintf("bChangeClockSpeed should be a string with value 'TRUE' or \
                  'FALSE'.\n");
       mexPrintf("if no bChangeClockSpeed argument is given, the default 'FALSE' \
                  will be used.\n");
    }
    /* Gamma */
    else if (!strcmp(parametername, "Gamma"))
    {
        mexPrintf("Gamma: float\n");
        mexPrintf("To turn of gamma correction, specify a value of 0 or 1.\n");
    }
    /* Gpo */
    else if (!strcmp(parametername, "Gpo"))
    {
        mexPrintf("Gpo: string Value, float PulseLength\n");
        mexPrintf("Value should be 'On' or 'Off'.\n");
        mexPrintf("If PulseLength = 0, GPO is set to Value.\n");
        mexPrintf("If PulseLength > 0, A pulse of PulseLength milliseconds is sent \
                  via the GPO;\n");
        mexPrintf("  the value of the pulse is set by Value.\n");
    }
```

```
/* GreenGain */
else if (!strcmp(parametername, "GreenGain"))
{
    mexPrintf("GreenGain: U32\n");
    mexPrintf("Value should not exceed %lu.\n", PCS2112_MAX_GAIN);
}
/* ImagerClocking */
else if (!strcmp(parametername, "ImagerClocking"))
{
    mexPrintf("ImagerClocking: string\n");
    mexPrintf("Valid values are:\n");
    mexPrintf("'0x00' Oscilator type: External (16MHz)\n");
    mexPrintf("       Max clock rate divided by: (no division)\n");
    mexPrintf("'0x01' Oscilator type: External (16MHz)\n");
    mexPrintf("       Max clock rate divided by: 2\n");
    mexPrintf("'0x02' Oscilator type: External (16MHz)\n");
    mexPrintf("       Max clock rate divided by: 4\n");
    mexPrintf("'0x80' Oscilator type: Internal (24MHz)\n");
    mexPrintf("       Max clock rate divided by: (no division)\n");
    mexPrintf("'0x81' Oscilator type: Internal (24MHz)\n");
    mexPrintf("       Max clock rate divided by: 2\n");
    mexPrintf("'0x82' Oscilator type: Internal (24MHz)\n");
    mexPrintf("       Max clock rate divided by: 4\n");
}
/* ImagerName */
else if (!strcmp(parametername, "ImagerName"))
{
    mexPrintf("ImagerName: string (max. 80 characters)\n");
}
/* MonoGain */
else if (!strcmp(parametername, "MonoGain"))
{
    mexPrintf("MonoGain: U32\n");
    mexPrintf("Value should not exceed %lu.\n", PCS2112_MAX_GAIN);
}
/* OverlayCallBack */
else if (!strcmp(parametername, "OverlayCallBack"))
{
    /* TODO */
    mexPrintf("PimMegaSetOverlayCallBack not supported yet.\n");
}
/* PreviewColorConversion */
else if (!strcmp(parametername, "PreviewColorConversion"))
{
    mexPrintf("PreviewColorConversion: string\n");
    mexPrintf("Valid values are:\n");
    mexPrintf("'BAYER_2BY2_COLOR'        Fastest\n");
    mexPrintf("'BAYER_3BY3_COLOR'        Fast--default\n");
    mexPrintf("'BAYER_3BY3GGRAD_COLOR'   Best quality for real-time\n");
    mexPrintf("'BAYER_2PASSGRAD_COLOR'   Best for captured images\n");
    mexPrintf("'BAYER_2PASSADAPT_COLOR'  Best for captured images\n");
    mexPrintf("'BAYER_VARGRAD_COLOR'     Best for captured images\n");
    mexPrintf("'BAYER_2BY2_MONO'         Fastest (converts to monochrome)\n");
    mexPrintf("'BAYER_3BY3_MONO'        Best quality for real-time (to mono)\n");
    mexPrintf("'BAYER_ADAPT_MONO'       Best for caputured images (to mono)\n");
    mexPrintf("'BAYER_NO_CONVERSION'     No Bayer conversion\n");
}
/* PreviewWindow */
else if (!strcmp(parametername, "PreviewWindow"))
{
    /* TODO */
    mexPrintf("PimMegaSetPreviewWindow not supported yet.\n");
}
/* RedGain */
else if (!strcmp(parametername, "RedGain"))
{
    mexPrintf("RedGain: U32\n");
    mexPrintf("Value should not exceed %lu.\n", PCS2112_MAX_GAIN);
}
/* Saturation */
else if (!strcmp(parametername, "Saturation"))
{
    mexPrintf("Saturation: U32\n");
    mexPrintf("Value should not exceed 0xFF.\n");
}
/* SubWindow */
```

```
    else if (!strcmp(parametername, "SubWindow"))
    {
        mexPrintf("SubWindow: string uDecimation\n");
        mexPrintf("           U32 uStartRow\n");
        mexPrintf("           U32 uStartColumn\n");
        mexPrintf("           U32 uNumberRows\n");
        mexPrintf("           U32 uNumberColumns\n");
        mexPrintf("uDecimation should be 'PCS2112_NO_DECIMATION',\n");
        mexPrintf("  'PCS2112_DECIMATE_BY_2' or 'PCS2112_DECIMATE_BY_4'\n");
    }
    /* SubWindowPos */
    else if (!strcmp(parametername, "SubWindowPos"))
    {
        mexPrintf("SubWindowPos: U32 uStartRow, U32 uStartColumn\n");
    }
    /* SubWindowSize */
    else if (!strcmp(parametername, "SubWindowSize"))
    {
        mexPrintf("SubWindowSize: string uDecimation\n");
        mexPrintf("               U32 uHeight\n");
        mexPrintf("               U32 uWidth\n");
        mexPrintf("uDecimation should be 'PCS2112_NO_DECIMATION',\n");
        mexPrintf("  'PCS2112_DECIMATE_BY_2' or 'PCS2112_DECIMATE_BY_4'\n");
    }
    /* Timeout */
    else if (!strcmp(parametername, "Timeout"))
    {
        mexPrintf("Timeout: U32\n");
    }
    /* VideoMode */
    else if (!strcmp(parametername, "VideoMode"))
    {
        mexPrintf("VideoMode: string\n");
        mexPrintf("Valid values are 'STILL_MODE' and 'VIDEO_MODE',\n");
    }
    /* GrabColorConversion */
    else if (!strcmp(parametername, "GrabColorConversion"))
    {
        mexPrintf("GrabColorConversion: string\n");
        mexPrintf("Valid values are:\n");
        mexPrintf("'BAYER_2BY2_COLOR'        Fastest\n");
        mexPrintf("'BAYER_3BY3_COLOR'        Fast--default\n");
        mexPrintf("'BAYER_3BY3GGRAD_COLOR'   Best quality for real-time\n");
        mexPrintf("'BAYER_2PASSGRAD_COLOR'   Best for captured images\n");
        mexPrintf("'BAYER_2PASSADAPT_COLOR'  Best for captured images\n");
        mexPrintf("'BAYER_VARGRAD_COLOR'     Best for captured images\n");
        mexPrintf("'BAYER_2BY2_MONO'         Fastest (converts to monochrome)\n");
        mexPrintf("'BAYER_3BY3_MONO'         Best quality for real-time (to mono)\n");
        mexPrintf("'BAYER_ADAPT_MONO'        Best for caputured images (to mono)\n");
        mexPrintf("'BAYER_NO_CONVERSION'     No Bayer conversion\n");
    }
    /* GrabOutputType */
    else if (!strcmp(parametername, "GrabOutputType"))
    {
        mexPrintf("GrabOutputType: string\n");
        mexPrintf("Valid values are:\n");
        mexPrintf("'RAW'    Image exactly as produced by the camera\n");
        mexPrintf("'IMAGE'  24 bpp colour bitmap converted to Matlab image format\n");
        mexPrintf("'RGB24'  24 bpp colour bitmap in RGB24 format\n");
    }
    else
    {
        mexPrintf("plSet: Unknown parameter name.\n");
        mexErrMsgTxt("\n");
    }
}
```

D.14 plPrintPossibleValues.h

```
/*****************************************************************************/
/* Filename:    plPrintPossibleValues.h                                      */
/* Description: Header file for plPrintPossibleValues.cpp                    */
/* Authors:     L.I.Oei, M.A.E.Bakker                                        */
/* Date:        2002/08/06                                                   */
/* Updates:                                                                  */
/*****************************************************************************/

void plPrintPossibleValues(char* parametername);
```

D.15 plSet.cpp

```
/*********************************************************************/
/* Filename:    plSet.cpp                                            */
/* Description: Source code for the function plSet                   */
/* Authors:     M.A.E.Bakker, L.I.Oei                                */
/*              maarten@panic.et.tudelft.nl, l.i.oei@its.tudelft.nl  */
/* Date:        2002/08/06                                           */
/* Updates:                                                          */
/*********************************************************************/
/* plSet sets the value of a given parameter. When 1 argument is given  */
/* a list of all possible paramters is given. When 2 arguments are      */
/* given, the possible values of the given parameters are given.        */
/*********************************************************************/
/* Input:   prhs[0] = (struct) handle for the device or              */
/*                    (int)       serial number of the device        */
/*          prhs[1] = (string) name of the parameter (optional)      */
/*          prhs[2...n] = value of the given parameter (optional)    */
/* Output:  A list of all possible paramters (with 1 argument)       */
/*          The possible values of the parameter (with 2 arguments)  */
/* Syntax:  plSet(serialnumber), plSet(serialnumber, parametername), or */
/*          plSet(serialnumber, parametername, value)                */
/*********************************************************************/


#define WIN32_LEAN_AND_MEAN    /* Exclude rarely-used stuff from Windows headers */
#include <windows.h>

#include "Y:\soft95\matlab6\extern\include\mex.h"
/* with Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmex.lib */
/* and  Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmx.lib */

#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"
/* with Y:\software\framegrag\pixelink\api\pimmegaapi.lib */

#include "..\plPrintPossibleValues.h"
#include "..\plSetValue.h"

#ifdef __cplusplus
    extern "C" {
#endif

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    mxArray *serialNumberField;
    U32        serialNumber = -1;        /* the serial number of the device */

    mxArray *lhs[1], *rhs[5];        /* to call fgdevices */
    double *px;
    int isOpen = 0;

    int buflen, i;
    char *parametername = "";

    /* prhs[0] should be the device handle (struct) or serial number (int) */
    /* prhs[1] should be the parameter name (optional) */
    /* prhs[2...n] should be the parameter value (optional) */
    if (nrhs >= 1)
    {
        /* get the serialNumber */
        if (mxIsStruct(prhs[0]))
        {
            serialNumberField = mxGetField(prhs[0], 0, "SerialNumber");
            serialNumber = (U32)mxGetScalar(serialNumberField);
        }
        else if (mxIsDouble(prhs[0]))
        {
            serialNumber = (U32)mxGetScalar(prhs[0]);
        }
        else
        {
            mexPrintf("PixeLINK Interface error.\n");
            mexPrintf("plSet: Wrong argument given.\n");
           mexPrintf("First argument should be the device handle or the serial \
```

```
                    number.\n");
            mexErrMsgTxt("\n");
        }

        /* check whether the device has been opened */
        rhs[0] = mxCreateString("isopen");

        rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
        px = mxGetPr(rhs[1]);
        px[0] = (U32)serialNumber;

        mexCallMATLAB(1, lhs, 2, rhs, "plDevices");

        isOpen = (int)mxGetScalar(lhs[0]);

        if (!isOpen)
        {
            mexPrintf("plSet: Device with serial number %lu has not been opened.\n", \
                    serialNumber);
            mexErrMsgTxt("\n");
        }

        /* 1 argument: give list of possible parameters */
        if (nrhs == 1)
        {
            mexPrintf("Possible parameters (if supported):\n");
            mexPrintf("BlueGain                    DataTransferSize\n");
            mexPrintf("Exposure                    ExposureTime\n");
            mexPrintf("Gamma                       Gpo\n");
            mexPrintf("GreenGain                   ImagerClocking\n");
            mexPrintf("ImagerName                  MonoGain\n");
            mexPrintf("OverlayCallBack             PreviewColorConversion\n");
            mexPrintf("PreviewWindow               RedGain\n");
            mexPrintf("Saturation                  SubWindow\n");
            mexPrintf("SubWindowPos                SubWindowSize\n");
            mexPrintf("Timeout                     VideoMode\n");
            mexPrintf("GrabColorConversion         GrabOutputType\n");
        }
        else /* 2 or 3 arguments */
        {
            /* get the parameter name */
            if (!mxIsChar(prhs[1])) /* 2nd argument (parametername) should be a \
                                       string */
            {
                mexPrintf("plSet: Wrong argument given.\n");
                mexPrintf("Second argument should be a string with the \
                           parametername.\n");
                mexErrMsgTxt("\n");
            }

            buflen = (mxGetM(prhs[1]) * mxGetN(prhs[1]) * sizeof(mxChar)) + 1;
            parametername = (char *)mxCalloc(buflen, sizeof(char));
            mxGetString(prhs[1], parametername, buflen);


            /* 2 arguments: give possible values for this parameter */
            if (nrhs == 2)
            {
                plPrintPossibleValues(parametername);
            }

            /* 3 or more arguments: set value for this parameter */
            else
            {
                /* copy value arguments into an mxArray */
                for (i = 2; i < nrhs; i++)
                {
                    rhs[i-2] = (struct mxArray_tag *)prhs[i];
                }

                /* call plSetValue to set these values */
                plSetValue(serialNumber, parametername, nrhs-2, rhs);
            }
        }
    }
    else /* ! (nrhs >= 1) */
```

```
    {
        mexPrintf("PixeLINK Interface error.\n");
        mexPrintf("plSet: Wrong number of arguments given.\n");
        mexPrintf("One or more arguments needed:\n");
        mexPrintf("1.    handle or serial number\n");
        mexPrintf("2.    parametername (optional)\n");
        mexPrintf("3...n parametervalue (optional)\n");
        mexPrintf("Example: plSet(handle), plSet(handle, parametername), or\n");
        mexPrintf("         plSet(handle, parametername, value)\n");
        mexPrintf("     or: plSet(serialnumber), plSet(serialnumber, parametername), \
                or\n");
        mexPrintf("         plSet(serialnumber, parametername, value)\n");
        mexErrMsgTxt("\n");
    }
}

#ifdef __cplusplus
    }    /* extern "C" */
#endif
```

D.16 plSetValue.cpp

```
/*********************************************************************/
/* Filename:    plSetValue.cpp                                       */
/* Description: Source code for plSetValue                           */
/* Authors:     M.A.E.Bakker, L.I.Oei                                */
/*              maarten@panic.et.tudelft.nl, l.i.oei@its.tudelft.nl  */
/* Date:        2002/08/09                                           */
/* Updates:                                                          */
/*********************************************************************/
/* plSetValue sets the value of a given parameter.                   */
/*********************************************************************/
/* Input:    serialNumber (U32)        serialNumber of the device    */
/*            parametername (string)    name of the parameter        */
/*            nvalues (int)             number of values             */
/*            values (mxArray)        mxArray with values            */
/* Output:    -                                                      */
/* Syntax:    plSetValue(serialNumber, parametername, nvalues, values)  */
/*********************************************************************/

#include <windows.h>
#include "Y:\soft95\matlab6\extern\include\mex.h"
#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"

#include "plError.h"
#include "plTypes.h"
#include "plPrintPossibleValues.h"


void plSetValue(U32 serialNumber, char* parametername, int nvalues, mxArray *values[])
{
    /* m = mxArray structure to be returned as device handle */
    mxArray *lhs[1], *rhs[4];
    double *px;
    HANDLE deviceId;

    /* variable for receiving the error codes the pixelink api functions return */
    PXL_RETURN_CODE nRetValue;

    int    buflen, i;
    U32    tmpU32, tmpU322, tmpU323, tmpU324, tmpU325;
    char*  tmpstring;
    float  tmpfloat;
    BOOL   tmpBOOL;

    /*********************************************************************/
    /*          Get the deviceId                                         */
    /*********************************************************************/
    rhs[0] = mxCreateString("get");

    rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
    px = mxGetPr(rhs[1]);
    px[0] = (U32)serialNumber;

    mexCallMATLAB(1, lhs, 2, rhs, "plDevices");

    deviceId = (HANDLE)(int)mxGetScalar(lhs[0]);

    /*********************************************************************/
    /*       Check whether the correct parameters are given.             */
    /*********************************************************************/

    /* U32 */
    if (!strcmp(parametername, "BlueGain") ||
        !strcmp(parametername, "Exposure") ||
        !strcmp(parametername, "GreenGain") ||
        !strcmp(parametername, "MonoGain") ||
        !strcmp(parametername, "RedGain") ||
        !strcmp(parametername, "Saturation") ||
        !strcmp(parametername, "Timeout"))
    {
        if (! ( (nvalues == 1) && mxIsDouble(values[0]) ) )
        {
            mexPrintf("plSetValue: wrong number of parameters given or wrong \
```

```
                    parametertype.\n");
            mexPrintf("%s expects 1 parameter of type U32.\n", parametername);
            mexErrMsgTxt("\n");
        }
    }
    /* string */
    else if (!strcmp(parametername, "DataTransferSize") ||
        !strcmp(parametername, "ImagerClocking") ||
        !strcmp(parametername, "ImagerName") ||
        !strcmp(parametername, "PreviewColorConversion") ||
        !strcmp(parametername, "VideoMode") ||
        !strcmp(parametername, "GrabColorConversion") ||
        !strcmp(parametername, "GrabOutputType"))
    {
        if (! ( (nvalues == 1) && mxIsChar(values[0]) ) )
        {
            mexPrintf("plSetValue: wrong number of parameters given or wrong \
                    parametertype.\n");
            mexPrintf("%s expects 1 parameter of type string.\n", parametername);
            mexErrMsgTxt("\n");
        }
    }
    /* float, [string] */
    else if (!strcmp(parametername, "ExposureTime"))
    {
        if (! ((nvalues == 1) || (nvalues == 2)))
        {
            mexPrintf("plSetValue: wrong number of parameters given.\n");
            mexPrintf("%s expects 1 or 2 parameters.\n", parametername);
            mexErrMsgTxt("\n");
        }
        if (! mxIsDouble(values[0]) )
        {
            mexPrintf("plSetValue: wrong parametertype given.\n");
            mexPrintf("%s expects a float as 1st argument.\n", parametername);
            mexErrMsgTxt("\n");
        }
        if ((nvalues == 2) && ! mxIsChar(values[0]) )
        {
            mexPrintf("plSetValue: wrong parametertype given.\n");
            mexPrintf("%s expects a string as 2nd argument.\n", parametername);
            mexErrMsgTxt("\n");
        }
    }
    /* float */
    if (!strcmp(parametername, "Gamma"))
    {
        if (! ( (nvalues == 1) && mxIsDouble(values[0]) ) )
        {
            mexPrintf("plSetValue: wrong number of parameters given or wrong \
                    paramtertype.\n");
            mexPrintf("%s expects 1 parameter of type float.\n", parametername);
            mexErrMsgTxt("\n");
        }
    }
    /* U32, [float] */
    else if (!strcmp(parametername, "Gpo"))
    {
        if (! ((nvalues == 1) || (nvalues == 2)))
        {
            mexPrintf("plSetValue: wrong number of parameters given.\n");
            mexPrintf("%s expects 1 or 2 parameters.\n", parametername);
            mexErrMsgTxt("\n");
        }
        if (! mxIsDouble(values[0]) )
        {
            mexPrintf("plSetValue: wrong parametertype given.\n");
            mexPrintf("%s expects a U32 as 1st argument.\n", parametername);
            mexErrMsgTxt("\n");
        }
        if ((nvalues == 2) && ! mxIsDouble(values[0]) )
        {
            mexPrintf("plSetValue: wrong parametertype given.\n");
            mexPrintf("%s expects a float as 2nd argument.\n", parametername);
            mexErrMsgTxt("\n");
        }
```

```
}
/* string, U32, U32, U32, U32 */
if (!strcmp(parametername, "SubWindow"))
{
    if (! (nvalues == 5))
    {
        mexPrintf("plSetValue: wrong number of parameters given.\n");
        mexPrintf("%s expects 5 parameters of types string, U32, U32, U32, \
                  U32.\n", parametername);
        mexErrMsgTxt("\n");
    }
    if (! mxIsChar(values[0]))
    {
        mexPrintf("plSetValue: wrong parametertype given.\n");
        mexPrintf("%s expects 5 parameters of types string, U32, U32, U32, \
                  U32.\n", parametername);
        mexErrMsgTxt("\n");
    }
    for (i = 1; i < 5; i++)
    {
        if (! mxIsDouble(values[i]) )
        {
            mexPrintf("plSetValue: wrong parametertype given.\n");
           mexPrintf("%s expects 5 parameters of types string, U32, U32, U32, \
                     U32.\n", parametername);
            mexErrMsgTxt("\n");
        }
    }
}
/* U32, U32 */
else if (!strcmp(parametername, "SubWindowPos"))
{
    if (! (nvalues == 2))
    {
        mexPrintf("plSetValue: wrong number of parameters given.\n");
        mexPrintf("%s expects 2 parameters.\n", parametername);
        mexErrMsgTxt("\n");
    }
    if (! (mxIsDouble(values[0]) && mxIsDouble(values[1])))
    {
        mexPrintf("plSetValue: wrong parametertype given.\n");
        mexPrintf("%s expects a two arguments of type U32.\n", parametername);
        mexErrMsgTxt("\n");
    }
}
/* string, U32, U32 */
else if (!strcmp(parametername, "SubWindowSize"))
{
    if (! (nvalues == 3))
    {
        mexPrintf("plSetValue: wrong number of parameters given.\n");
        mexPrintf("%s expects 3 parameters.\n", parametername);
        mexErrMsgTxt("\n");
    }
    if (! mxIsChar(values[0]))
    {
        mexPrintf("plSetValue: wrong parametertype given.\n");
        mexPrintf("%s expects a 3 arguments of types string, U32, U32.\n", \
                  parametername);
        mexErrMsgTxt("\n");
    }
    if (! (mxIsDouble(values[1]) && mxIsDouble(values[2])))
    {
        mexPrintf("plSetValue: wrong parametertype given.\n");
        mexPrintf("%s expects a 3 arguments of types string, U32, U32.\n", \
                  parametername);
        mexErrMsgTxt("\n");
    }
}

/*********************************************************************/
/*          Set the value of the given parameter                     */
/*********************************************************************/

/* Blue Gain */
if (!strcmp(parametername, "BlueGain"))
```

```
{
    tmpU32 = (U32)mxGetScalar(values[0]);

    /* check whether the given value is within the correct range */
    if (tmpU32 > PCS2112_MAX_GAIN)
    {
        mexPrintf("plSetValue: Error setting BlueGain: value too large.\n");
        mexPrintf("BlueGain value should not exceed %lu.\n", PCS2112_MAX_GAIN);
        mexErrMsgTxt("\n");
    }

    /* set the value */
    nRetValue = PimMegaSetBlueGain(deviceId, tmpU32);

    if (plError(nRetValue, "setting BlueGain"))
    {
        mexErrMsgTxt("\n");
    }
}
/* DataTransferSize */
else if (!strcmp(parametername, "DataTransferSize"))
{
    buflen = (mxGetM(values[0]) * mxGetN(values[0]) * sizeof(mxChar)) + 1;
    tmpstring = (char *)mxCalloc(buflen, sizeof(char));
    mxGetString(values[0], tmpstring, buflen);

    /* check whether this is a known value */
    if (!strcmp(tmpstring, "DATA_8BIT_SIZE"))
    {
        tmpU32 = DATA_8BIT_SIZE;
    }
    else if (!strcmp(tmpstring, "DATA_16BIT_SIZE"))
    {
        tmpU32 = DATA_16BIT_SIZE;
    }
    else
    {
        mexPrintf("plSetValue: Error setting DataTransferSize: unknown value.\n");
        mexPrintf("Valid values are 'DATA_8BIT_SIZE' and 'DATA_16BIT_SIZE'\n");
        mexErrMsgTxt("\n");
    }

    /* set the value */
    nRetValue = PimMegaSetDataTransferSize(deviceId, tmpU32);

    if (plError(nRetValue, "setting DataTransferSize"))
    {
        mexErrMsgTxt("\n");
    }
}
/* Exposure */
else if (!strcmp(parametername, "Exposure"))
{
    tmpU32 = (U32)mxGetScalar(values[0]);

    /* check whether the given value is within the correct range */
    if (tmpU32 > PCS2112_MAX_EXPOSURE)
    {
        mexPrintf("plSetValue: Error setting Exposure: value too large.\n");
        mexPrintf("Exposure value should not exceed %lu.\n", \
                  PCS2112_MAX_EXPOSURE);
        mexErrMsgTxt("\n");
    }

    /* set the value */
    nRetValue = PimMegaSetExposure(deviceId, tmpU32);

    if (plError(nRetValue, "setting Exposure"))
    {
        mexErrMsgTxt("\n");
    }
}
/* ExposureTime */
else if (!strcmp(parametername, "ExposureTime"))
{
    tmpfloat = (float)mxGetScalar(values[0]);
```

```
        tmpBOOL = FALSE;

        if (nvalues >= 2) /* if a second value is given, get the value */
        {
            buflen = (mxGetM(values[1]) * mxGetN(values[1]) * sizeof(mxChar)) + 1;
            tmpstring = (char *)mxCalloc(buflen, sizeof(char));
            mxGetString(values[1], tmpstring, buflen);

            /* check whether this is a known value */
            if (!strcmp(tmpstring, "FALSE"))
            {
                tmpBOOL = FALSE;
            }
            else if (!strcmp(tmpstring, "TRUE"))
            {
                tmpBOOL = TRUE;
            }
            else
            {
                mexPrintf("plSetValue: Error setting ExposureTime: unknown value.\n");
                mexPrintf("Second value should be 'TRUE' or 'FALSE'.\n");
                mexErrMsgTxt("\n");
            }
        }

        /* set the value */
        nRetValue = PimMegaSetExposureTime(deviceId, tmpfloat, tmpBOOL);

        if (plError(nRetValue, "setting ExposureTime"))
        {
            mexErrMsgTxt("\n");
        }
    }
    /* Gamma */
    else if (!strcmp(parametername, "Gamma"))
    {
        tmpfloat = (float)mxGetScalar(values[0]);

        nRetValue = PimMegaSetGamma(deviceId, tmpfloat);

        if (plError(nRetValue, "setting Gamma"))
        {
            mexErrMsgTxt("\n");
        }
    }
    /* GPO */
    else if (!strcmp(parametername, "Gpo"))
    {
        tmpU32 = 0;

        /* get Value */
        buflen = (mxGetM(values[0]) * mxGetN(values[0]) * sizeof(mxChar)) + 1;
        tmpstring = (char *)mxCalloc(buflen, sizeof(char));
        mxGetString(values[0], tmpstring, buflen);

        if (!strcmp(tmpstring, "On"))
        {
            tmpU32 = 1;
        }
        else if (!strcmp(tmpstring, "Off"))
        {
            tmpU32 = 0;
        }
        else
        {
            mexPrintf("plSetValue: Error setting Gpo: unknown value.\n");
            mexPrintf("First value should be 'On' or 'Off'.\n");
            mexErrMsgTxt("\n");
        }

        /* get PulseLength */
        tmpfloat = 0;

        if (nvalues >= 2)
        {
            tmpfloat = (float)mxGetScalar(values[1]);
```

```
        }

        /* set the value */
        nRetValue = PimMegaSetGpo(deviceId, tmpU32, tmpfloat);

        if (plError(nRetValue, "setting Gpo"))
        {
            mexErrMsgTxt("\n");
        }
    }
    /* Green Gain */
    else if (!strcmp(parametername, "GreenGain"))
    {
        tmpU32 = (U32)mxGetScalar(values[0]);

        /* check whether the given value is within the correct range */
        if (tmpU32 > PCS2112_MAX_GAIN)
        {
            mexPrintf("plSetValue: Error setting GreenGain: value too large.\n");
            mexPrintf("GreenGain value should not exceed %lu.\n", PCS2112_MAX_GAIN);
            mexErrMsgTxt("\n");
        }

        /* set the value */
        nRetValue = PimMegaSetGreenGain(deviceId, tmpU32);

        if (plError(nRetValue, "setting GreenGain"))
        {
            mexErrMsgTxt("\n");
        }
    }
    /* ImagerClocking */
    else if (!strcmp(parametername, "ImagerClocking"))
    {
        tmpU32 = 0x00;

        buflen = (mxGetM(values[0]) * mxGetN(values[0]) * sizeof(mxChar)) + 1;
        tmpstring = (char *)mxCalloc(buflen, sizeof(char));
        mxGetString(values[0], tmpstring, buflen);

        /* check whether this is a known value */
        if (!strcmp(tmpstring, "0x00"))
        {
            tmpU32 = 0x00;
        }
        else if (!strcmp(tmpstring, "0x01"))
        {
            tmpU32 = 0x01;
        }
        else if (!strcmp(tmpstring, "0x02"))
        {
            tmpU32 = 0x02;
        }
        else if (!strcmp(tmpstring, "0x80"))
        {
            tmpU32 = 0x80;
        }
        else if (!strcmp(tmpstring, "0x81"))
        {
            tmpU32 = 0x81;
        }
        else if (!strcmp(tmpstring, "0x82"))
        {
            tmpU32 = 0x82;
        }
        else
        {
            mexPrintf("plSetValue: Error setting ImagerClocking: unknown value.\n");
            mexPrintf("Valid values are: '0x00', '0x01', '0x02', '0x80', '0x81' and \
                       '0x82'.\n");
            mexErrMsgTxt("\n");
        }

        /* set the value */
        nRetValue = PimMegaSetImagerClocking(deviceId, tmpU32);
```

```
        if (plError(nRetValue, "setting ImagerClocking"))
        {
            mexErrMsgTxt("\n");
        }
    }
    /* ImagerName */
    else if (!strcmp(parametername, "ImagerName"))
    {
        buflen = (mxGetM(values[0]) * mxGetN(values[0]) * sizeof(mxChar)) + 1;
        tmpstring = (char *)mxCalloc(buflen, sizeof(char));
        mxGetString(values[0], tmpstring, buflen);

        if (buflen > 81)
        {
            mexPrintf("plSetValue: Error setting ImagerName: string too long.\n");
            mexPrintf("ImagerName should not exceed 80 characters.\n");
            mexErrMsgTxt("\n");
        }

        /* set the value */
        nRetValue = PimMegaSetImagerName(deviceId, tmpstring);

        if (plError(nRetValue, "setting ImagerName"))
        {
            mexErrMsgTxt("\n");
        }
    }
    /* MonoGain */
    else if (!strcmp(parametername, "MonoGain"))
    {
        tmpU32 = (U32)mxGetScalar(values[0]);

        /* check whether the given value is within the correct range */
        if (tmpU32 > PCS2112_MAX_GAIN)
        {
            mexPrintf("plSetValue: Error setting MonoGain: value too large.\n");
            mexPrintf("MonoGain value should not exceed %lu.\n", PCS2112_MAX_GAIN);
            mexErrMsgTxt("\n");
        }

        /* set the value */
        nRetValue = PimMegaSetMonoGain(deviceId, tmpU32);

        if (plError(nRetValue, "setting MonoGain"))
        {
            mexErrMsgTxt("\n");
        }
    }
    /* OverlayCallBack */
    else if (!strcmp(parametername, "OverlayCallBack"))
    {
        /* TODO */
        mexErrMsgTxt("PimMegaSetOverlayCallBack not supported yet.\n");
    }
    /* PreviewColorConversion */
    else if (!strcmp(parametername, "PreviewColorConversion"))
    {
        tmpU32 = BAYER_3BY3_COLOR;

        buflen = (mxGetM(values[0]) * mxGetN(values[0]) * sizeof(mxChar)) + 1;
        tmpstring = (char *)mxCalloc(buflen, sizeof(char));
        mxGetString(values[0], tmpstring, buflen);

        /* check whether this is a known value */
        if (!strcmp(tmpstring, "BAYER_2BY2_COLOR"))
            tmpU32 = BAYER_2BY2_COLOR;
        else if (!strcmp(tmpstring, "BAYER_3BY3_COLOR"))
            tmpU32 = BAYER_3BY3_COLOR;
        else if (!strcmp(tmpstring, "BAYER_3BY3GGRAD_COLOR"))
            tmpU32 = BAYER_3BY3GGRAD_COLOR;
        else if (!strcmp(tmpstring, "BAYER_2PASSGRAD_COLOR"))
            tmpU32 = BAYER_2PASSGRAD_COLOR;
        else if (!strcmp(tmpstring, "BAYER_2PASSADAPT_COLOR"))
            tmpU32 = BAYER_2PASSADAPT_COLOR;
        else if (!strcmp(tmpstring, "BAYER_VARGRAD_COLOR"))
            tmpU32 = BAYER_VARGRAD_COLOR;
```

```
        else if (!strcmp(tmpstring, "BAYER_2BY2_MONO"))
            tmpU32 = BAYER_2BY2_MONO;
        else if (!strcmp(tmpstring, "BAYER_3BY3_MONO"))
            tmpU32 = BAYER_3BY3_MONO;
        else if (!strcmp(tmpstring, "BAYER_ADAPT_MONO"))
            tmpU32 = BAYER_ADAPT_MONO;
        else if (!strcmp(tmpstring, "BAYER_NO_CONVERSION"))
            tmpU32 = BAYER_NO_CONVERSION;
        else
        {
            mexPrintf("plSetValue: Error setting PreviewColorConversion: unknown \
                       value.\n");
            plPrintPossibleValues("PreviewColorConversion");
            mexErrMsgTxt("\n");
        }

        /* set the value */
        nRetValue = PimMegaSetImagerClocking(deviceId, tmpU32);

        if (plError(nRetValue, "setting PreviewColorConversion"))
        {
            mexErrMsgTxt("\n");
        }
    }
    /* PreviewWindow */
    else if (!strcmp(parametername, "PreviewWindow"))
    {
        /* TODO */
        mexErrMsgTxt("PimMegaSetPreviewWindow not supported yet.\n");
    }
    /* RedGain */
    else if (!strcmp(parametername, "RedGain"))
    {
        tmpU32 = (U32)mxGetScalar(values[0]);

        /* check whether the given value is within the correct range */
        if (tmpU32 > PCS2112_MAX_GAIN)
        {
            mexPrintf("plSetValue: Error setting RedGain: value too large.\n");
            mexPrintf("RedGain value should not exceed %lu.\n", PCS2112_MAX_GAIN);
            mexErrMsgTxt("\n");
        }

        /* set the value */
        nRetValue = PimMegaSetRedGain(deviceId, tmpU32);

        if (plError(nRetValue, "setting RedGain"))
        {
            mexErrMsgTxt("\n");
        }
    }
    /* Saturation */
    else if (!strcmp(parametername, "Saturation"))
    {
        tmpU32 = (U32)mxGetScalar(values[0]);

        /* check whether the given value is within the correct range */
        if (tmpU32 > 0xFF)
        {
            mexPrintf("plSetValue: Error setting Saturation: value too large.\n");
            mexPrintf("Saturation value should not exceed 0xFF.\n");
            mexErrMsgTxt("\n");
        }

        /* set the value */
        nRetValue = PimMegaSetSaturation(deviceId, tmpU32);

        if (plError(nRetValue, "setting Saturation"))
        {
            mexErrMsgTxt("\n");
        }
    }
    /* SubWindow */
    else if (!strcmp(parametername, "SubWindow"))
    {
        tmpU32 = PCS2112_NO_DECIMATION;
```

```
        /* uDecimation */
        buflen = (mxGetM(values[0]) * mxGetN(values[0]) * sizeof(mxChar)) + 1;
        tmpstring = (char *)mxCalloc(buflen, sizeof(char));
        mxGetString(values[0], tmpstring, buflen);

        if (!strcmp(tmpstring, "PCS2112_NO_DECIMATION"))
        {
            tmpU32 = PCS2112_NO_DECIMATION;
        }
        else if (!strcmp(tmpstring, "PCS2112_DECIMATE_BY_2"))
        {
            tmpU32 = PCS2112_DECIMATE_BY_2;
        }
        else if (!strcmp(tmpstring, "PCS2112_DECIMATE_BY_4"))
        {
            tmpU32 = PCS2112_DECIMATE_BY_4;
        }
        else
        {
            mexPrintf("plSetValue: Error setting SubWindow: unknown value.\n");
            mexPrintf("uDecimation should be 'PCS2112_NO_DECIMATION',\n");
            mexPrintf("  'PCS2112_DECIMATE_BY_2' or 'PCS2112_DECIMATE_BY_4'\n");
            mexErrMsgTxt("\n");
        }

        tmpU322 = (U32)mxGetScalar(values[2]); /* uStartColumn */
        tmpU323 = (U32)mxGetScalar(values[1]); /* uStartRow */
        tmpU324 = (U32)mxGetScalar(values[4]); /* uNumberColumns */
        tmpU325 = (U32)mxGetScalar(values[3]); /* uNumberRows */

        /* check whether the given values are within the correct range */
        if ((tmpU322 + tmpU324) > PCS2112_MAX_WIDTH)
        {
            mexPrintf("plSetValue: Error setting SubWindow: value too large.\n");
            mexPrintf("(uStartColumn + uNumberColumns) should not exceed %lu.\n", \
                      PCS2112_MAX_WIDTH);
            mexErrMsgTxt("\n");
        }
        if ((tmpU323 + tmpU325) > PCS2112_MAX_HEIGHT)
        {
            mexPrintf("plSetValue: Error setting SubWindow: value too large.\n");
            mexPrintf("(uStartRow + uNumberRows) should not exceed %lu.\n", \
                      PCS2112_MAX_HEIGHT);
            mexErrMsgTxt("\n");
        }
        if (tmpU324 < PCS2112_MIN_WIDTH)
        {
            mexPrintf("plSetValue: Error setting SubWindow: value too small.\n");
            mexPrintf("uNumberColumns must be at least %lu.\n", PCS2112_MIN_WIDTH);
            mexErrMsgTxt("\n");
        }
        if (tmpU325 < PCS2112_MIN_HEIGHT)
        {
            mexPrintf("plSetValue: Error setting SubWindow: value too small.\n");
            mexPrintf("uNumberRows must be at least %lu.\n", PCS2112_MIN_HEIGHT);
            mexErrMsgTxt("\n");
        }

        /* set the values */
        nRetValue = PimMegaSetSubWindow(deviceId, tmpU32, tmpU322, tmpU323, tmpU324, \
                                                                            tmpU325);

        if (plError(nRetValue, "setting SubWindow"))
        {
            mexErrMsgTxt("\n");
        }
    }
    /* SubWindowPos */
    else if (!strcmp(parametername, "SubWindowPos"))
    {
        tmpU32 = (U32)mxGetScalar(values[1]); /* uStartColumn */
        tmpU322 = (U32)mxGetScalar(values[0]); /* uStartRow */

        /* tmpU323 = subWindowSizeDecimationValue */
        /* tmpU324 = subWindowSizeWidthValue (number of columns) */
```

```
        /* tmpU325 = subWindowSizeHeightValue (number of rows) */
        nRetValue = PimMegaGetSubWindowSize(deviceId, &tmpU323, &tmpU324, &tmpU325);

        /* check whether the given values are within the correct range */
        if ((tmpU32 + tmpU324) > PCS2112_MAX_WIDTH)
        {
            mexPrintf("plSetValue: Error setting SubWindowPos: value too large.\n");
            mexPrintf("(uStartColumn + current number of columns) should not exceed \
                       %lu.\n", PCS2112_MAX_WIDTH);
            mexErrMsgTxt("\n");
        }
        if ((tmpU322 + tmpU325) > PCS2112_MAX_HEIGHT)
        {
            mexPrintf("plSetValue: Error setting SubWindowPos: value too large.\n");
            mexPrintf("(uStartRow + current number of rows) should not exceed \
                       %lu.\n", PCS2112_MAX_HEIGHT);
            mexErrMsgTxt("\n");
        }

        /* set the values */
        nRetValue = PimMegaSetSubWindowPos(deviceId, tmpU32, tmpU322);

        if (plError(nRetValue, "setting SubWindowPos"))
        {
            mexErrMsgTxt("\n");
        }
    }
    /* SubWindowSize */
    else if (!strcmp(parametername, "SubWindowSize"))
    {
        tmpU32 = PCS2112_NO_DECIMATION;

        /* uDecimation */
        buflen = (mxGetM(values[0]) * mxGetN(values[0]) * sizeof(mxChar)) + 1;
        tmpstring = (char *)mxCalloc(buflen, sizeof(char));
        mxGetString(values[0], tmpstring, buflen);

        if (!strcmp(tmpstring, "PCS2112_NO_DECIMATION"))
        {
            tmpU32 = PCS2112_NO_DECIMATION;
        }
        else if (!strcmp(tmpstring, "PCS2112_DECIMATE_BY_2"))
        {
            tmpU32 = PCS2112_DECIMATE_BY_2;
        }
        else if (!strcmp(tmpstring, "PCS2112_DECIMATE_BY_4"))
        {
            tmpU32 = PCS2112_DECIMATE_BY_4;
        }
        else
        {
            mexPrintf("plSetValue: Error setting SubWindowSize: unknown value.\n");
            mexPrintf("uDecimation should be 'PCS2112_NO_DECIMATION',\n");
            mexPrintf("  'PCS2112_DECIMATE_BY_2' or 'PCS2112_DECIMATE_BY_4'\n");
            mexErrMsgTxt("\n");
        }

        tmpU322 = (U32)mxGetScalar(values[2]); /* uWidth */
        tmpU323 = (U32)mxGetScalar(values[1]); /* uHeight */

        /* check whether the given values are within the correct range */

        /* tmpU324 = subWindowPosStartColumnValue */
        /* tmpU325 = subWindowPosStartRowValue */
        nRetValue = PimMegaGetSubWindowPos(deviceId, &tmpU324, &tmpU325);

        if ((tmpU324 + tmpU322) > PCS2112_MAX_WIDTH)
        {
            mexPrintf("plSetValue: Error setting Subwindow size: value too large.\n");
            mexPrintf("(Current start column + uWidth) should not exceed %lu.\n", \
                       PCS2112_MAX_WIDTH);
            mexErrMsgTxt("\n");
        }
        if ((tmpU325 + tmpU323) > PCS2112_MAX_HEIGHT)
        {
            mexPrintf("plSetValue: Error setting Subwindow size: value too large.\n");
```

```
            mexPrintf("(Current start row + uHeight) should not exceed %lu.\n", \
                      PCS2112_MAX_HEIGHT);
            mexErrMsgTxt("\n");
        }
        if (tmpU322 < PCS2112_MIN_WIDTH)
        {
            mexPrintf("plSetValue: Error setting Subwindow size: value too small.\n");
            mexPrintf("uWidth must be at least %lu.\n", PCS2112_MIN_WIDTH);
            mexErrMsgTxt("\n");
        }
        if (tmpU323 < PCS2112_MIN_HEIGHT)
        {
            mexPrintf("plSetValue: Error setting Subwindow size: value too small.\n");
            mexPrintf("uHeight must be at least than %lu.\n", PCS2112_MIN_HEIGHT);
            mexErrMsgTxt("\n");
        }

        /* set the values */
        nRetValue = PimMegaSetSubWindowSize(deviceId, tmpU32, tmpU322, tmpU323);

        if (plError(nRetValue, "setting SubWindowSize"))
        {
            mexErrMsgTxt("\n");
        }
    }
    /* Timeout */
    else if (!strcmp(parametername, "Timeout"))
    {
        tmpU32 = (U32)mxGetScalar(values[0]);

        nRetValue = PimMegaSetTimeout(deviceId, tmpU32);

        if (plError(nRetValue, "setting Timeout"))
        {
            mexErrMsgTxt("\n");
        }
    }
    /* VideoMode */
    else if (!strcmp(parametername, "VideoMode"))
    {
        tmpU32 = STILL_MODE;

        buflen = (mxGetM(values[0]) * mxGetN(values[0]) * sizeof(mxChar)) + 1;
        tmpstring = (char *)mxCalloc(buflen, sizeof(char));
        mxGetString(values[0], tmpstring, buflen);

        /* check whether this is a known value */
        if (!strcmp(tmpstring, "STILL_MODE"))
        {
            tmpU32 = STILL_MODE;
        }
        else if (!strcmp(tmpstring, "VIDEO_MODE"))
        {
            tmpU32 = VIDEO_MODE;
        }
        else
        {
            mexPrintf("plSetValue: Error setting VideoMode: unknown value.\n");
            mexPrintf("Valid values are 'STILL_MODE' and 'VIDEO_MODE',\n");
            mexErrMsgTxt("\n");
        }

        /* set the value */
        nRetValue = PimMegaSetVideoMode(deviceId, tmpU32);

        if (plError(nRetValue, "setting VideoMode"))
        {
            mexErrMsgTxt("\n");
        }
    }
    else if (!strcmp(parametername, "GrabColorConversion"))
    {
        tmpU32 = BAYER_3BY3_COLOR;

        buflen = (mxGetM(values[0]) * mxGetN(values[0]) * sizeof(mxChar)) + 1;
        tmpstring = (char *)mxCalloc(buflen, sizeof(char));
```

```
        mxGetString(values[0], tmpstring, buflen);

        /* check whether this is a known value */
        if (!strcmp(tmpstring, "BAYER_2BY2_COLOR"))
            tmpU32 = BAYER_2BY2_COLOR;
        else if (!strcmp(tmpstring, "BAYER_3BY3_COLOR"))
            tmpU32 = BAYER_3BY3_COLOR;
        else if (!strcmp(tmpstring, "BAYER_3BY3GGRAD_COLOR"))
            tmpU32 = BAYER_3BY3GGRAD_COLOR;
        else if (!strcmp(tmpstring, "BAYER_2PASSGRAD_COLOR"))
            tmpU32 = BAYER_2PASSGRAD_COLOR;
        else if (!strcmp(tmpstring, "BAYER_2PASSADAPT_COLOR"))
            tmpU32 = BAYER_2PASSADAPT_COLOR;
        else if (!strcmp(tmpstring, "BAYER_VARGRAD_COLOR"))
            tmpU32 = BAYER_VARGRAD_COLOR;
        else if (!strcmp(tmpstring, "BAYER_2BY2_MONO"))
            tmpU32 = BAYER_2BY2_MONO;
        else if (!strcmp(tmpstring, "BAYER_3BY3_MONO"))
            tmpU32 = BAYER_3BY3_MONO;
        else if (!strcmp(tmpstring, "BAYER_ADAPT_MONO"))
            tmpU32 = BAYER_ADAPT_MONO;
        else if (!strcmp(tmpstring, "BAYER_NO_CONVERSION"))
            tmpU32 = BAYER_NO_CONVERSION;
        else
        {
            mexPrintf("plSetValue: Error setting GrabColorConversion: unknown \
                       value.\n");
            plPrintPossibleValues("GrabColorConversion");
            mexErrMsgTxt("\n");
        }

        /* set the value */
        rhs[0] = mxCreateString("setpar");
        rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
        px = mxGetPr(rhs[1]);
        px[0] = (U32)serialNumber;
        rhs[2] = mxCreateString("GrabColorConversion");
        rhs[3] = mxCreateDoubleMatrix(1, 1, mxREAL);
        px = mxGetPr(rhs[3]);
        px[0] = tmpU32;
        mexCallMATLAB(0, lhs, 4, rhs, "plDevices");
    }
    else if (!strcmp(parametername, "GrabOutputType"))
    {
        buflen = (mxGetM(values[0]) * mxGetN(values[0]) * sizeof(mxChar)) + 1;
        tmpstring = (char *)mxCalloc(buflen, sizeof(char));
        mxGetString(values[0], tmpstring, buflen);

        /* check whether this is a known value */
        if (!strcmp(tmpstring, "RAW"))
            tmpU32 = RAW;
        else if (!strcmp(tmpstring, "IMAGE"))
            tmpU32 = IMAGE;
        else if (!strcmp(tmpstring, "RGB24"))
            tmpU32 = RGB24;
        else
        {
            mexPrintf("plSetValue: Error setting GrabOutputType: unknown value.\n");
            plPrintPossibleValues("GrabOutputType");
            mexErrMsgTxt("\n");
        }

        /* set the value */
        rhs[0] = mxCreateString("setpar");
        rhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
        px = mxGetPr(rhs[1]);
        px[0] = (U32)serialNumber;
        rhs[2] = mxCreateString("GrabOutputType");
        rhs[3] = mxCreateDoubleMatrix(1, 1, mxREAL);
        px = mxGetPr(rhs[3]);
        px[0] = tmpU32;
        mexCallMATLAB(0, lhs, 4, rhs, "plDevices");
    }
    else
    {
        mexPrintf("plSetValue: Unknown parameter name.\n");
```

```
        mexErrMsgTxt("\n");
    }
}
```

D.17 plSetValue.h

```
/*********************************************************************/
/* Filename:    plSetValue.h                                         */
/* Description: Header file for plSetValue.cpp                       */
/* Authors:     L.I.Oei, M.A.E.Bakker                                */
/* Date:        2002/08/06                                           */
/* Updates:                                                          */
/*********************************************************************/

void plSetValue(U32 serialNumber, char* parametername, int nvalues, mxArray
*values[]);
```

D.18 plTypes.h

```
/*************************************************************************/
/* Filename:    plTypes.h                                                */
/* Description: Header file containing (type) definitions.               */
/* Authors:     M.A.E.Bakker, L.I.Oei                                    */
/* Date:        2002/08/06                                               */
/* Updates:                                                              */
/*************************************************************************/

#define RAW    0x0
#define IMAGE 0x1
#define RGB24 0x2
```

Appendix E. Test results

Below are the screen printouts of the tests performed on plFGI:

```
>> plset(m,'DataTransferSize','DATA_8BIT_SIZE');
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

   65.8000

>> plset(m,'SubWindowSize','PCS2112_NO_DECIMATION', 512, 640);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

   29.6600

>> plset(m,'SubWindowSize','PCS2112_NO_DECIMATION', 256, 640);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

   17.3000

>> plset(m,'SubWindowSize','PCS2112_NO_DECIMATION', 1024, 640);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

   55.8000

>> plset(m,'SubWindowSize','PCS2112_NO_DECIMATION', 1024, 320);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

   52.7800

>> plset(m,'ExposureTime',1);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

   51.3500

>> plget(m)

ans =

                          DeviceID: 256106896
                          BlueGain: 'Unsupported'
                 CurrentFrameRate: 'Could not get value'
                 DataTransferSize: 'DATA_8BIT_SIZE'
                          Exposure: 1028
                      ExposureTime: 1.0400
                             Gamma: 1
                               Gpo: 'Off'
                         GreenGain: 'Unsupported'
        HardwareVersion.ProductID: 'PL-A630 Series Monochrome EC Module'
     HardwareVersion.SerialNumber: '12572'
  HardwareVersion.FirmwareVersion: '1.0.3.0'
      HardwareVersion.FpgaVersion: '4'
                    ImagerChipId: 805306368
                  ImagerClocking: '0x01 External (16Mhz) Division by 2'
                      ImagerName: 'Imager0'
                      ImagerType: 'PCS2112M_IMAGER (Monochrome Camera)'
                        MonoGain: 0
            PreviewWindowPos.Top: 'Could not get value'
           PreviewWindowPos.Left: 'Could not get value'
        PreviewWindowSize.Height: 'Could not get value'
```

```
 PreviewWindowSize.Width: 'Could not get value'
                  RedGain: 'Unsupported'
              Saturation: 'Unsupported'
            SerialNumber: 75122
         SoftwareVersion: 16777984
    SubWindow.Decimation: 'PCS2112_NO_DECIMATION'
      SubWindow.StartRow: 0
   SubWindow.StartColumn: 0
    SubWindow.NumberRows: 1024
 SubWindow.NumberColumns: 320
   SubWindowPos.StartRow: 0
SubWindowPos.StartColumn: 0
SubWindowSize.Decimation: 'PCS2112_NO_DECIMATION'
    SubWindowSize.Height: 1024
     SubWindowSize.Width: 320
                 Timeout: 1000
               VideoMode: 'VIDEO_MODE'
     GrabColorConversion: 'BAYER_3BY3_COLOR'
          GrabOutputType: 'RAW'
```

plFGI

MatLAB Interface
for
PixeLINK FireWire Cameras

User Manual & Technical Guide

Last updated: September, 2002

MatLAB Interface
for
PixeLINK FireWire Cameras

User Manual & Technical Guide

M.A.E.Bakker (maarten@panic.et.tudelft.nl)
L.I.Oei (L.I.Oei@its.tudelft.nl)
September, 2002

Index

PART I USER MANUAL

1 General introduction

This manual describes the Matlab interface 'plFGI' for Vitana PixeLINK digital cameras. The name plFGI stands for 'PixeLINK Frame Grabber Interface'. In this document you will find described the functionality of this interface and information about the working conditions. This document can be used as a reference while working with the interface.

1.1 Short description of Matlab

Matlab is a mathematical program for technical computing. It is used by research institutes and universities all over the world, mostly to solve problems, which involve matrix and vector calculations. It can, for example, be used to make analyses and computations on images, which are stored as matrices. This is the main purpose of the plFGI software. One of the important features of Matlab is the ability to use external compiled programs from within Matlab, called MEX-files. The plFGI consists of such external programs. In a windows environment, the native .dll format is used for MEX-files. Variables are passed between Matlab and the MEX-files using mxArrays in which all required variables can be stored and returned.

1.2 Short description of the PixeLINK camera

The PixeLINK PL-A6xx camera series consists of megapixel cameras with a CMOS image sensor and integrated image processing electronics. The IEEE-1394 'FireWire' interface is used for communicating with the computer. The cameras are available in both in colour and black & white versions. Some of the camera models are equipped with a machine vision connector to connect to external shutters etcetera.

1.3 Short description of plFGI

The plFGI is a simple interface for grabbing images directly from the PixeLINK camera into a Matlab mxArray. It follows the concept of the existing Data Translation FGI closely. Several picture formats are supported. Currently, the interface features grabbing an image from a videostream and changing most settings of the camera. To accomplish this, functionality of the PixeLINK API library - which is supplied with the camera - is used.

1.4 Hardware and software requirements

To use this interface, the following is required, newer versions may also work, older versions probably won't give good results:

- A Vitana PixeLINK PL-A6xx series camera.
- PixeLINK drivers / API version 3.0
- A PC with at least a Pentium II processor, 64MB of RAM and a IEEE-1394 'FireWire' interface.
- MS Windows 98SE, Windows ME, Windows 2000 SP1 or Windows XP.
- Mathworks Matlab 6.

2 Introduction to plFGI

In this chapter, some background information and a description of the plFGI functions are given..

2.1 About data types

The following Matlab data types are used when working with the plFGI:

- string

A vector of which the components are ASCII codes. Used with the plFGI for passing names of camera parameters to plGet and plSet.

- double

The default data type, used by Matlab to store numbers in the double precision floating point format. Range from 1.7E-308 to 1.7E+308. Used with the plFGI for passing all variables that are marked U32, int or double. The (U32) and (int) designations used below in this document are only used for indicating the allowed range (32 bit unsigned or signed integer) of the passed variable.

- uint8

8 bit unsigned integer. Used by the plFGI within an mxArray for returning a grabbed image.

- uint16

16 bit unsigned integer. Used by the plFGI within an mxArray for returning a grabbed image.

- struct

An array containing fieldnames (string) and their values (any type). Used by plFGI for the device handle structure.

- mxArray

The default way of storing arrays of any data type in Matlab. Data is stored in column major format. Used by the plFGI for returning a grabbed image and for storing the device's handle structure.

2.2 About the camera's serial number

The camera's serial number is used as a unique identifier for opening and using the camera. The PixeLINK camera we used for testing the plFGI had as many as 3 different serial numbers. One serial number on a sticker on the board the image sensor was mounted on (this number is returned in the HardwareInfo structure, and also used in the PixeLINK application that comes with the camera). Another serial number is on the board on which the other electronics of the camera are located. The third serial number is returned when you get the serial number using the function `'PimMegaGetSerialNumber'`. This last number is the one we decided to use as 'official' serial number, on basis of the PixeLINK documentation suggesting this. If you don't know what serial number the camera you want to use has, call plOpen with a non-existing number to get a listing of the serial numbers of all cameras that are attached to the computer.

2.3 The plFGI functions

The plFGI supports the following functions (in alphabetical order):

- **plClose** closes a PixeLINK camera.
- **plGet** gets the value of a camera or image processing parameter or the complete handle structure of the camera.
- **plGrab** grabs an image from a PixeLINK camera.
- **plIsOpen** checks if a PixeLINK camera is already open.
- **plOpen** opens a PixeLINK camera and return its handle structure or only returns its handle if the camera was already open.
- **plSet** sets the value of a camera or image processing parameter or shows the possible parameters or values.

Nota bene: In Matlab, the function names are not case sensitive, although it may be easier to read back code if cases are used.

Below, descriptions are given of each of the plFGI functions callable from Matlab. For a description of plDevices, which is used as an internal function, we refer to the technical documentation. In the descriptions below, prhs[0] is used to indicate the first input parameter, prhs[1] for the second input parameter, prhs[2] for the third, etcetera. Similarly, plhs[0] is used to indicate the (first) return value of the MEX-function.

- ***plClose***

DLL name	plClose.dll
Syntax	`plClose(handle)` or `plClose(serialnumber)`
Description	plClose closes an open PixeLINK camera device.
Input	prhs[0] = (struct) handle of the device, or (U32) serial number of the device
Remarks	It will be verified whether the input is a valid handle or number. plClose will then try to close the device. If the device was not open, does not exist or returns an error code, an error message is printed and plClose exits.
Output	No return value, the device is closed and the internal list of open devices is updated.
Examples	`plClose(m);` `plClose(75122);`

- ***plGet***

DLL name	plGet.dll
Syntax	`plGet(handle)`, `plGet(handle, parametername)`, `plGet(serialnumber)` or `plGet(serialnumber, parametername)`
Description	plGet returns the device's up-to-date handle if 1 argument is given, or returns the value of the given parameter if 2 arguments are given.
Input	prhs[0] = (struct) handle for the device, or (U32) serial number of the device prhs[1] = (string) name of the parameter (optional)
Remarks	It will be verified whether the input is a valid handle or number. plGet will then try to get the requested parameter. If the parameter is not valid, the device is not open, does not exist or returns an error code, and error message is printed and plGet exits.
Output	plhs[0] = (struct) handle for the device (with 1 input argument) plhs[0] = value(s) of the given parameter (with 2 input arguments)

Examples	`m = plGet(m);`
	`value = plGet(m, 'Timeout');`
	`m = plGet(75122);`
	`value = plGet(75122, 'Timeout');`

- ***plGrab***

DLL name	plGrab.dll
Syntax	`imagematrix = plGrab(handle)` or `imagematrix = plGrab(serialnumber)`
Description	plGrab grabs a frame from the PixeLINK camera device and places it into an mxArray.
Input	prhs[0] = (struct) handle for the device, or (U32) serial number of the device
Remarks	It will be verified whether the input is a valid handle or number. plGrab will then try to capture and return an image. If there is not enough free memory, the device is not open, does not exist or returns an error code, and error message is printed and plGrab exits.
Output	plhs[0] = (mxArray) grabbed frame
Examples	`im = plGrab(m);` `im = plGrab(75122);`

- ***plIsOpen***

DLL name	plIsOpen.dll
Syntax	`plIsOpen(handle)` or `plIsOpen(serialnumber)`
Description	plIsOpen checks whether the device has been opened. It returns 0 if the device has not been opened, and 1 if it has been opened.
Input	prhs[0] = (struct) device handle, or (U32) serial number of the device
Remarks	It will be verified whether the input is a valid handle or number. If the camera was unplugged from the computer without being closed first, this function will still show it as being open.
Output	plhs[0] = (double) 0 if the camera device is not open 1 if the camera device is open
Examples	`boolean = plIsOpen(m);` `boolean = plIsOpen(75122);`

- ***plOpen***

DLL name	plOpen.dll
Syntax	`handle = plOpen(serialnumber)`
Description	plOpen opens a PixeLINK camera device and returns the device's handle. If there is no camera with the given serial number, a list of available serial numbers is printed.
Input	prhs[0] = (int) serial number of the device to be opened
Remarks	It will be verified whether the input is a number. If it is a valid serial number plOpen will try to open the camera device and return a handle structure. If the camera is already open, a warning will be printed, and the handle structure will be returned. If the input was not a number, or the camera could not be opened, an error message is printed. If there is no camera with the given serial number, a list of available serial numbers is printed.
Output	plhs[0] = (struct) handle for the device
Example	`m = plOpen(75122);`

- ***plSet***

DLL name	plSet.dll
Syntax	`plSet(handle)`, `plSet(handle, parametername)`, `plSet(handle, parametername, value)`, `plSet(serialnumber)`, `plSet(serialnumber, parametername)`, or `plSet(serialnumber, parametername, value)`
Description	plSet sets the value of a given parameter. When 1 argument is given a list of all possible parameters is given. When 2 arguments are given, the possible values of the given parameter are given. When called with 3 or more arguments, the device's parameter given in the second argument will be set to the value(s) given in the other argument(s).
Input	prhs[0] = (struct) handle for the device, or (int) serial number of the device prhs[1] = (string) name of the parameter (optional) prhs[2...n] = value of the given parameter (optional)
Remarks	It will be verified whether the input is a valid handle or number. If it is not, an error message will be printed. plSet will then try to write the specified parameter. If the parameter does not exist, the value is out of range under the given circumstances or if the device returns an error code, an error message is printed and plSet exits.
Output	A list of all possible parameters is printed on the screen (with 1 input argument) The possible values of the parameter are printed on the screen (with 2 input arguments)

Examples	`plSet(m);`
	`plSet(m, 'Timeout');`
	`plSet(m, 'Timeout', value);`
	`plSet(75122);`
	`plSet(75122, 'Timeout');`
	`plSet(75122, 'Timeout', 1024);`
	`plSet(m, 'SubWindowSize', 'PCS2112_NO_DECIMATION', 480, 640);`

2.4 Notes on plGrab

The properties of the grabbed image depend on several parameters. For most parameter settings, a good overview is given in the PixeLINK manuals. Two parameters are not covered directly by those manuals: GrabColorConversion and GrabOutputType.

The setting of GrabOutputType (default: **RAW**) determines what image format should be returned. The following settings are possible:

- **RAW** returns the image exactly as produced by the camera (row major array, one position for every pixel). Colour images are returned as a Bayer pattern.
- **IMAGE** returns an mxArray that is suitable for direct viewing with the Matlab image command. In case of colour images, the Bayer pattern is first translated into a suitable RGB format. The conversion method is determined by the setting of GrabColorConversion.
- **RGB24** returns a single column mxArray that is filled according to the standardised RGB24 format. It can be used for example to save pictures to a file, as many image file formats are in one way or the other based on this format. In case of colour images, the Bayer pattern is first translated into a suitable RGB format. The conversion method is determined by the setting of GrabColorConversion.

The setting of GrabColorConversion (default: **BAYER_3BY3_COLOR**) determines, in case of colour images that are returned as IMAGE or RGB24, what method should be used to convert the Bayer pattern of the camera to a suitable image. Possible values are:

- **BAYER_2BY2_COLOR** Fastest
- **BAYER_3BY3_COLOR** Fast—Default
- **BAYER_3BY3GGRAD_COLOR** Best quality for real-time
- **BAYER_2PASSGRAD_COLOR** Best for captured images
- **BAYER_2PASSADAPT_COLOR** Best for captured images
- **BAYER_VARGRAD_COLOR** Best for captured images
- **BAYER_2BY2_MONO** Fastest (converts to monochrome)
- **BAYER_3BY3_MONO** Best quality for real-time (to mono.)
- **BAYER_ADAPT_MONO** Best for captured images (to mono.)
- **BAYER_NO_CONVERSION** No Bayer conversion

The captured image is returned in an mxArray. To arrange things in a bit more convenient way, let's define 'Height' to be (SubWindowSize.Height / SubWindowSize.Decimation) and define 'Width' to be (SubWindowSize.Width / SubWindowSize.Decimation). Then, the image is returned as described below:

- For **RAW** images an mxArray of size [Width, Height] filled with uint8 or uint16 is returned.
  Whether the contents of the mxArray consist of uint8 or uint16 depends on the setting of the DataTransferSize parameter.
- For **IMAGE** images an mxArray of size [Height, Width, 3] filled with uint8 is returned. This is true for colour and black & white images, both are returned in the Matlab colour image format.
- For **RGB24** format, an array of size [Height * Width * 3, 1] of uint8 is returned.

2.5 Notes on plGet and plSet

Care should be taken on the following points:

- Not all combinations of parameter values are valid, for example setting ImagerClocking to 0x00 (16MHz, no division) while DataTransferSize is set to 'DATA_16BIT_SIZE' will result in an error message.
- Some values have a certain granularity, for example setting SubWindowSize.Height to 475 will result in a value of 472 being set.
- Parameter names are case sensitive.
- For a complete list of parameters, refer to Appendix B of this document and to the PixeLINK documentation.

2.6 Note on using handles

The handle obtained when using plOpen will of course not automatically be updated to reflect the recent settings of the device. This is not important for the plFGI functions, as they only use non- settable parameters from the handle. When you want to view the most recent settings, please use plOpen or plGet to obtain the most recent information. It is even possible to make use of the construction `m = plGet(m);` to update the handle to reflect changed settings, if one wants to.

3 Working with plFGI

3.1 Getting started

To be able to use plFGI functions, Matlab should be able to find and load the right files. First make sure the PixeLINK API library file 'PimMegaApi.dll' is in a directory that is in the Windows path. If it is not, the following error message will occur whenever you try to use a plFGI function:

```
Unable to load mex file: g:\fgi\bin\plOpen.dll.
The specified module could not be found.

??? Invalid MEX-file
```

This can be quite confusing, as it doesn't directly point to the actual cause of the error.

Also, Matlab should be able to find the plFGI executable files. If the plFGI distribution is stored on the network path Y:\Software\plFGI for example, the following commands should be typed into the Matlab command window:

```
system_dependent RemotePathPolicy Reload;
system_dependent RemoteCWDPolicy Reload;
addpath Y:\software\plFGI\bin;
```

To view black and white RAW images with the Matlab image function, make sure the right colour map is set by typing:

```
colormap(gray(255));
```

When viewing raw images in 16-bit format, it should be taken into consideration that values of 255 and above will be interpreted as maximum white level by the Matlab image function. To be able to view the image correctly, use some scaling.

3.2 Some examples

The following example will open a camera having serial number 75122, get the current value of the gamma correction parameter displaying it on the screen, set the gamma correction parameter, then get the new value of the gamma correction parameter displaying it on the screen, then close the camera. To have the parameter not display on the screen, just put a **;** behind the plGet lines.

```
m = plOpen(75122);
g = plGet(m, 'Gamma')
plSet(m, 'Gamma', 0.65);
g = plGet(m, 'Gamma')
plClose(m);
```

The following example will open a (black & white) camera having serial number 75122, grab an image using the default settings (480 rows, 640 columns, 8-bit raw image), display it and then close the camera.

```
m = plOpen(75122);
im = plGrab(m);
colormap(gray(255));
image(im');
plClose(m);
```

The following example will open a (colour) camera having serial number 98765, set the image size to 200 rows, 320 columns, decimation factor 2, set the GrabOutputType, set the GrabColorConversion and capture an image. Then it will set the GrabColorconversion to another value and capture another image. It will then close the camera, display the first image, wait for a key to be pressed, then display the second image.

```
m = plOpen(98765);
plSet(m, 'SubWindowSize', 'PCS2112_DECIMATE_BY_2', 200, 320);
plSet(m, 'GrabOutputType', 'IMAGE');
plSet(m, 'GrabColorConversion', 'BAYER_VARGRAD_COLOR');
im = plGrab(m);
plSet(m, 'GrabColorConversion', 'BAYER_ADAPT_MONO');
im2 = plGrab(m);
plClose(m);
image(im);
waitforbuttonpress;
image(im2);
```

The following example will capture 100 images and display them on the screen one after the other as in a movie. It is presumed that the camera is already open with handle h, that all parameters are set correctly and that it will be closed later. Note: the tic and toc Matlab functions can be used to measure the average frame rate. To measure the pure grabbing speed without the time Matlab needs to display the image, just leave out the line with `image(i');`

```
for im=1:100
   im = plgrab(m);
   image(im');
end
```

PART II TECHNICAL GUIDE

1 Introduction to plFGI development

1.1 The plFGI distribution package

The plFGI distribution package currently consists of 3 folders:

/bin contains all .dll and .m executable files
/doc contains all documentation
/src contains the source code and project files used for building the plFGI

In every folder and subfolder, a readme.txt file is present, in which all files that should be in the folder are listed, along with the purpose they are for.

1.2 Building plFGI

plFGI was built using the Microsoft Visual C++ 6.0 Integrated Development Environment. Every MEX-function has its own subfolder, containing the necessary files for building it. Double click on the appropriate workspace file (.dsw) to open the project. All relevant header and source files (see Appendix C for an overview of the dependencies) are included in the workspace. Before opening the project, you may want to check the pathnames used in the various project configuration files, or just place all files in the same folders as used during development:

G:\NewFGI\src\	plFGI source files and project folders
Y:\software\framegrag\pixelink\api\	PixeLINK libraries
Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\	Matlab libraries

At some point, you may want to create a new project, below is the procedure that was used to create plFGI:

- Create a new DLL project (File → New → Projects → Win32 Dynamic-Link Library → A DLL that exports some symbols)
- Throw away StdAfx.cpp, StdAfx.h and Projectname.h
- Disable precompiled headers (Project → Settings → C/C++ → Category Precompiled Headers → Not using precompiled headers)
- Link the necessary libraries (Project → Settings → Link → Category General → Object/library modules):
  - Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmex.lib
  - Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmx.lib
  - Y:\software\framegrag\pixelink\api\pimmegaapi.lib
- Export the mexFunction (Project → Settings → Link → Category General → Project Options): `/export:mexFunction`
- Implement Projectname.cpp, starting with the following template:

```
#define WIN32_LEAN_AND_MEAN
/* This excludes rarely-used stuff from Windows headers. We don't
 * know what that means exactly, but it is generated by MS Visual C++
 * when you create a project and we just copied it.
 */

#include <windows.h>

#include "Y:\soft95\matlab6\extern\include\mex.h"
/* with Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmx.lib
 * and Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmex.lib
 */

#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"
/* with Y:\software\framegrag\pixelink\api\pimmegaapi.lib */

#ifdef __cplusplus
    extern "C" {
#endif

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    /* implement the MEX function here */
}

#ifdef __cplusplus
    } /* extern "C" */
#endif
```

Code example 1 ***Template for a plFGI MEX-function***

The libraries of Matlab version 6.1 and the PixeLINK Camera API Release 3.0 were used for the development of the plFGI.

1.3 General notes on plFGI development

The MEX-files are implemented as C++ files (instead of C), because the PixeLINK Camera API is implemented in C++ and it uses language constructions, which cannot be used with C files. With the `extern "C"` command, the mexFunction is exported as a C function, so it will be possible to call the function from within Matlab, just like normal C MEX-files.

The PixeLINK dll is load-time linked. This means that when the MEX-file is loaded into memory, the PixeLINK dll is loaded at the same time. Therefore, to use the MEX-files, it is necessary that the file 'PimMegaApi.dll' is in the *Windows*-path (not the Matlab-path). Otherwise Matlab will give an error-message like:

```
Unable to load mex file: g:\fgi\bin\plOpen.dll.
The specified module could not be found.

??? Invalid MEX-file
```

This can be quite confusing, as it doesn't directly point to the actual cause of the error.

1.4 Testing

During development, frequent testing helped to find bugs in plFGI. After the program reached a stable working state, a few other tests were run in the limited time that was still available.

On suggestion of Assistant Professor Smutný, it was tested how much time the capturing of multiple frames takes. This could possibly be used as an indication of the quality of the code (memory leaks and resources that are not released would have become apparent this way). Also, the performance of the code, the PixeLINK API and the PixeLINK camera could be verified.

First, some basic tests were performed, showing that capturing the images resulted in a frame rate a bit lower than that of the demo application, but that could be contributed to the overhead of calling functions from Matlab, and the lack of loop optimalisation in plFGI. When testing different resolutions, the effect on overall capturing speed of the camera itself became apparent: reducing the number of columns in the picture did not affect the capture time very much, as the camera scans row by row. Reducing the number of rows reduced the time needed for capturing accordingly. As a last experiment, the exposure time was variated. Strangely enough, capturing 100 images would take less time than 100 times the exposure time.

As there was not much time to take a closer look at these tests and do some more advanced testing, this might be a good subject for future examination.

In Appendix E, the test results are included.

2 Technical description of plFGI

This chapter describes the implementation of the plFGI, the structure of which is similar to the original FGI system for the DataTranslation frame grabbers. The system is composed of several MEX-functions, which are callable from Matlab. It supports functionality to open and close a device, check whether a device is opened, get and set a number of parameters and grab an image from the camera. This chapter starts with an overview of the structure of the plFGI, after that follow descriptions of the individual source files. For a functional specification from the user's point of view, please refer to the first section of this document.

2.1 Structure diagram

The system is composed of several functions (plIsOpen, plGrab, plGet, plOpen, plClose and plSet), which can be called from Matlab. These functions call the appropriate functions from the PixeLINK Camera Application Programming Interface (API). plDevices maintains the 'open device array', in which the device ID's (handles to open devices) are stored. In the diagram below, the relations between the internal and external plFGI functions are shown. An arrow indicates that a function is called by another function. The ↔ arrow between plDevices and the open device array means that data is exchanged in two directions.

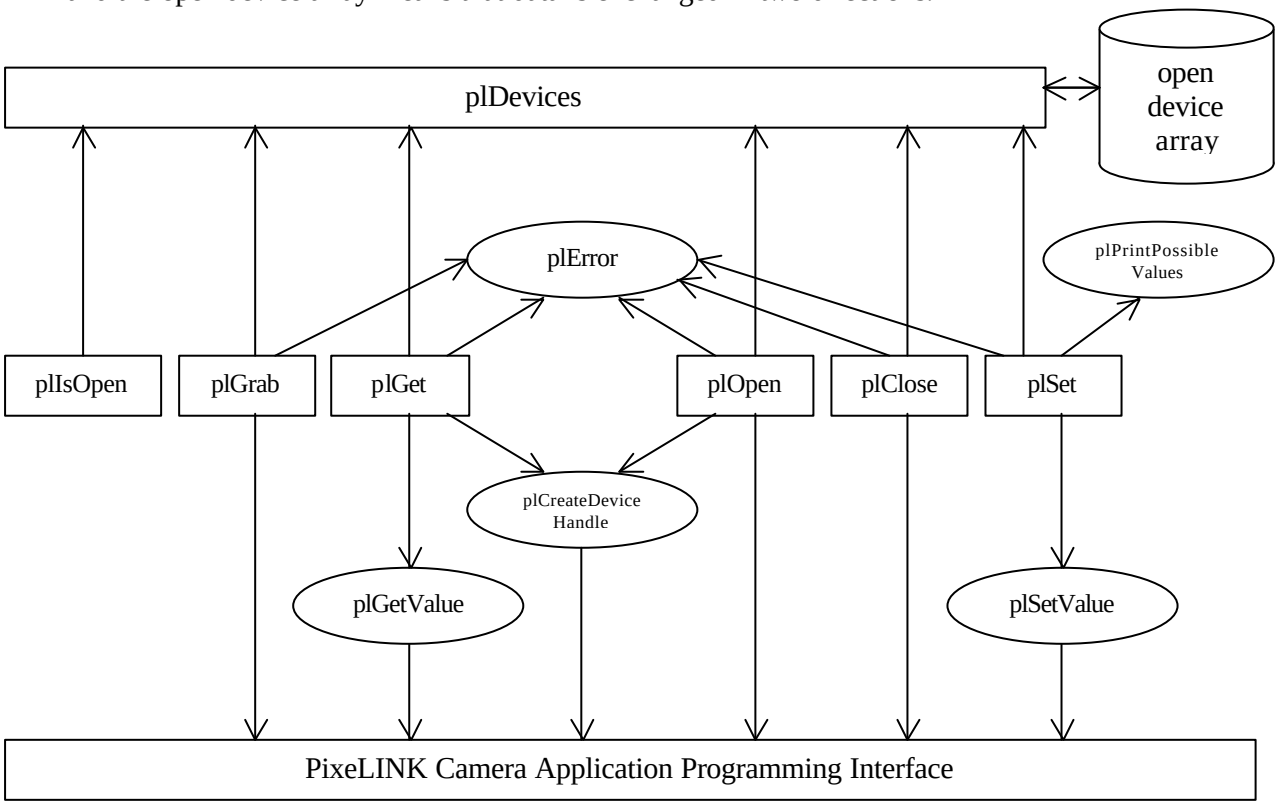

Figure 1 ***Structure of plFGI***

2.2 Description of plClose

plClose is the MEX-function which takes care of closing an opened PixeLINK device. First it is tested whether one and only one argument was given, using 'if (nrhs == 1)'. If not, the else part is executed, printing an error message with mexPrintf and exiting with mexErrMsgTxt. If the argument is a device handle structure, the device's serial number is extracted from it. If the argument is a number, this is treated as the serial number. If the argument is something else, again an error message will be shown and the function will exit.

Now, it can be assumed that a serial number is present. It is checked whether the device is registered as open, with a call to plDevices. If it is not open, the else-part of the 'if (isOpen)'-construction is executed and the function terminates with an error message. If it is open, again a call to plDevices is made, this time to obtain the PixeLINK API's device-ID. Then the PixeLINK API function 'pimMegaUninitialize' is made to close the device. When an error occurs closing the device, the standard error handling code in plError returns a non-false value and the plClose function exits. If all went well, the device should now be closed and a call to plDevices is made, to update the list of open devices.

2.3 Description of plGet

plGet is the MEX-function which is used for obtaining the value of a parameter, a set of parameters, or a complete device handle for a PixeLINK device. First, it is tested whether 1 or 2 arguments were given. If not, the else part of the check is executed, resulting in an error message and the termination of plGet by using the Matlab call mexErrMsgTxt. The first argument is either a serial number, or a device handle structure. If it is a double, it is stored in the serialNumber variable. If it is a struct containing a field with the name 'SerialNumber', the value of this field is stored in the serialNumber variable (without further checking). If the first argument is another data type, an error message is given and plGet is terminated.

After the serial number has been obtained, it is checked whether the device is open by calling plDevices. If not, an error message is given and plGet is terminated. If the device is open, it is determined whether one or two arguments were given when calling plGet. If only one argument was given, a call to the plCreateDeviceHandle subroutine is made and the result of that is returned. If two arguments were given, it is checked whether the second argument is a string. If it is not, an error message is given and plGet is terminated. If it is, the name of the parameter is copied into a string variable and the subroutine plGetValue is called and the result of that is returned.

2.4 Description of plGrab

plGrab is the MEX-function that grabs a frame from the PixeLINK device and takes care of the required postprocessing before delivering it to Matlab. Currently only one calling syntax is supported, so it is checked whether there is exactly one input argument and one output argument. If not, the else part of the check is executed, resulting in an error message and the termination of plGrab by using the Matlab call mexErrMsgTxt. The input argument is either a serial number, or a device handle structure. If it is a double, it is stored in the serialNumber variable. If it is a struct containing a field with the name 'SerialNumber', the value of this field is stored in the serialNumber variable (without further checking). If the first argument is another data type, an error message is given and plGrab is terminated.

The serial number is now used to obtain a PixeLINK API device ID by calling plDevices. Using PimMegaStartVideoStream, the video stream is opened. Behind every call to a PixeLINK API function, plError is used to check for an error. If an error occurs while the videostream is supposed to be active, PimMegaStopVideoStream is called before terminating.

Now, it is checked whether the camera is in 'video mode', if it is not, a warning is printed to the screen and the camera is set to video mode. Video mode is used because still mode requires special lighting conditions or a shutter to control the exposure time[1]. After this the program gets some camera parameters and settings:

- Get imagerType (colour or monochrome)
- Get dataTransferSize (8 bit or 16 bit format)
- Get decimation, width and height of the current subwindow[2]
- Calculate pixelWidth, pixelHeight according to the rules in the PixeLINK Megapixel FireWire Camera Developer's Manual[3]
- Get GrabColorConversion and GrabOutputType parameters using plDevices

Using the above parameters, 3 arrays must set up:

- The capture array, in which the PixeLINK API stores the raw image returned by the camera. This array has the same number of elements (either 8 or 16 bit unsigned integers) as the image has pixels (pixelWidth * pixelHeight) and is arranged as an mxArray of [pixelWidth, pixelHeight].
- The matlab array, which is returned to Matlab. In case of a RAW image, this array has the same size and arrangement as the capture array. In case of a 'Matlab IMAGE', it is an array of unsigned 8 bit integers three times the size of (pixelWidth * pixelHeight). It is arranged as an mxArray of [pixelHeight, pixelWidth, 3]; In memory, this means that 3 column-major arrays of [pixelHeight, pixelWidth] are stored consecutively (like this: RRR....GGG....BBB....). In case of an RGB24 image, an array of size [Height * Width * 3, 1] of uint8 is returned.
- The RGB24 array is used as an intermediate array to store the result of PimMegaConvert in case of a non-RAW image, before it is converted to fit into the matlab array. Its size is always (3 * pixelWidth * pixelHeight) and it is arranged as a normal C-type row-major array (like this: RGBRGBRGB....).

First, a distinction between the various 'GrabOutputType' settings is made, using a case construction. The size and dim variables are set to the required size and dimensions of the matlab array. After that, the sizes of the capture array and the RGB24 array are defined. A case construction is used to allocate memory for the matlab and capture arrays, differentiating between 8 and 16 bits data size. After this, it is checked whether an error occurred, if so the program is terminated. If everything went all right, the pointer to the matlab array is set. When needed, memory is allocated for the RGB24 array.

Now, the image is captured by making a call to PimMegaReturnVideoData and the videostream is closed using PimMegaStopVideoStream.

For postprocessing, two possibilities are distinguished using an if/else construction.

- The first possibility occurs in case of a RAW image: An 8 bit RAW image is copied into the matlab array using memcpy, a 16 bit RAW image is converted to have all 10

[1] [Vitana, 2002-1], §1.5.5
[2] [Vitana, 2002-1], p.92, 'PimMegaSetSubWindow'
[3] [Vitana, 2002-1], p.92, 'PimMegaSetSubWindow'

significant bits in the right order, according to to the PixeLINK Megapixel FireWire Camera User's Manual[4]: The 2 most significant bits of the first byte, are shifted to be the 2 least significant bits. The second byte is shifted 2 bits to the left (multiplied by 4), then the two bytes are added (equivalent to a logical 'and' operation).

- The second possibility occurs in case of an IMAGE or RGB24 output format. Depending on 8 versus 16 bits and black/white versus colour, one of the PimMegaConvert* functions is called to convert the captured image to a 24 bits per pixel image. For colour images, the GrabColorConversion parameter is used to determine the algorithm to be used by the PimMegaConvert* function. The result of PimMegaConvert* is stored into the RGB24 array. If the GrabOutputType is set to RGB24, the RGB24 array is directly copied into the matlab array, using memcpy. In case of an IMAGE type, however, the RGB24 array must be transposed in order for Matlab to display it properly. This is done using two nested for loops. Within the for loops the bytes for R, G and B are copied to the appropriate locations in the matlab array.

All above mentioned case and if constructions contain a default case resulting in an appropriate error message when the distinguishing variable is not recognised.

- Suggestions for improvement:
  - Implementation of the 5 other ways to call plGrab:
    - moviematrix = plGrab(handle, imgmatrix)
    - [imgmatrix, moviematrix] = plGrab(handle)
    - plGrab(handle, imgmatrix)
    - plGrab(handle, imgmatrix, moviematrix)
    - plGrab(handle, imgmatrix, moviematrix, scaling factor)
  - Addition of RGB48 format for 16-bit captures
  - Testing the plGrab function with a colour camera (this should work)
  - Loop optimalisations
  - Capture a RAW image directly into the matlab array.

2.5 Description of plIsOpen

plIsOpen is the MEX-function for checking if certain PixeLINK device is open.

First, it is tested whether only 1 argument was given. If not, the else part of the check is executed, resulting in an error message and the termination of plIsOpen by using the Matlab call mexErrMsgTxt. The argument is either a serial number, or a device handle structure. If it is a double, it is stored in the serialNumber variable. If it is a struct containing a field with the name 'SerialNumber', the value of this field is stored in the serialNumber variable (without further checking). If the first argument is another data type, an error message is given and plGet is terminated.

After the serial number has been obtained, it is checked whether the device is open by calling plDevices. The result of the call to plDevices is stored in the return array of plIsOpen.

2.6 Description of plOpen

plOpen is the MEX-function which takes care of opening a PixeLINK device for use with the other plFGI functions. First it is tested whether the input argument is a number, if it is not, the function terminates printing an appropriate error message using mexErrMsgTxt. Then it is

[4] [Vitana 2002-2], Appendix C

checked whether the device is open by calling plDevices. If the device is already open, the else-part of the if-construction is executed printing a warning message, then calling plCreateDeviceHandle and returning the handle.

If the device was not open, a call to PimMegaGetNumberDevices is made to get the amount of available devices. When no devices are attached, plOpen exits with an appropriate error message. When one or more devices are attached, a for loop is entered to open every device, store the serial number in the array of available serial numbers, check if the serial number matches the requested serial number, then close it and loop. If the serial number matches the requested serial number, the variable found is set to true and the loop is exited by using a break statement.

If found is not true, the array containing all available serial numbers is printed to the screen and the function is terminated using mexErrMsgTxt.

Finally, the open device array is updated by calling plDevices.

- Suggestions for improvement: Use plError after the call to PimMegaGetNumberDevices instead of non standard error code.

2.7 Description of plSet

plSet is the MEX-function used to change the parameter settings of a PixeLINK device. First, it is checked whether any arguments are given, if not, an appropriate error message is printed to the screen using mexPrintf, and the plSet is terminated using mexErrMsgTxt.

If one or more arguments were given, the first argument should be either a handle structure or a serial number. If it is a double, it is stored in the serialNumber variable. If it is a struct containing a field with the name 'SerialNumber', the value of this field is stored in the serialNumber variable (without further checking). If the first argument is another data type, an error message is given and plGet is terminated. After the serial number has been obtained, it is checked whether the device is open by calling plDevices. If not, an error message is given and plGet is terminated.

If only one argument was given, a list of parameter names is printed on the screen and plSet terminates. If a second argument was given, it is checked whether this is a string. If not plSet is terminated with an appropriate error message. If it is a string, it is stored in the parametername variable. Depending on whether 2 or 3 arguments were given, plPrintPossibleValues is called (2 parameters), or plSetValue is called (3 arguments).

- Suggestions for improvement:
  - Implementation of PimMegaSetOverlayCallBack
  - Implementation of PimMegaSetPreviewWindow
  - Implementation of PimMegaAutoExposure

2.8 Description of plDevices

plDevices is a MEX-function, just like the 6 functions described above. However, it only contains plFGI internal functionality and should not be called by the user. It is used to keep track of open devices. This is its functional specification:

DLL name	plDevices.dll
Syntax	`mexCallMATLAB(nlhs, *plhs[], nrhs, *prhs[], "plDevices");`
Description	plDevices maintains the 'open device array', in which the device ID's (handles to open devices) are stored. Also some settings, which are not stored in the PixeLINK Camera API itself, are stored in plDevices.
Input	prhs[0] = (string) name of task to be performed: print, remove, isopen, get, add, getpar, setpar prhs[1] = (U32) serial number of device; Only for remove, isopen, get, add, getpar, setpar prhs[2] = (int) deviceID; Only for add (string) name of the parameter; Only for getpar, setpar prhs[3] = value of the parameter; Only for setpar
Output	plhs[0] = (double) 1 if device open, 0 if closed; Only for isopen (int) deviceID if open, -1 if closed; Only for get parameter value, -1 if not found; Only for getpar

plDevices is implemented as a MEX-function, because of two reasons. First this makes it possible to use the function `mexLock()` to keep the file in memory, even after typing 'clear all' or 'clear mex'. This also eliminates the need of having to do the memory management ourselves. The other reason is that, for testing purposes, it is convenient to be able to call plDevices directly from Matlab. The users of the plFGI, however, should not use this feature.

The open device array

plDevices maintains the 'open device array'. This is an array of structures of type:

```
struct {U32 serialNumber;
        int deviceID;
        U32 grabColorConversion;
        int grabOutputType;}
```

In this global array of structs the device ID (handle to the device) and the parameters `GrabColorConversion` and `GrabOutputType` are stored for every open device. The serial number of the device is used to uniquely identify the device. This is the number returned by PimMegaGetSerialNumber, which is called by plOpen immediately after initialising the device.

The workings of plDevices

For every open device the following information is stored in the open device array described above: The device's serial number, the device's ID which is used by the PixeLINK API to identify the device, the GrabColorConversion parameter and the GrabOutputType parameter. Another global variable, deviceCount, is incremented every time a device is opened and decremented every time a device is closed, thus counting the number of open devices.

First mexLock is called, to make sure plDevices (containing the open device array) stays in memory. Then it is checked whether any arguments are given and if the first argument is a

text string. If not, the error message in the else part of the check is displayed and plDevices terminates. If it is, the first argument is stored in the task variable. If there is a second argument, it should always be the serial number of a device. If it is not a number, plDevices is terminated with an error message.

A case construction is used to make a first selection between the different tasks based on the number of arguments given. For every number of arguments from 1 to 4, a different case exists; the default case gives an error message. Inside a case, the different tasks are selected using an if/else if construction together with strcmp. When the strcmp fails, an appropriate error message is given and plDevices terminates.

Although plDevices is not meant to be called directly from Matlab, in the next part the syntaxes and examples are written down as if it were, because this way it is easier to denote and understand the usage of the left-hand and right-hand side arguments.

- ***1 argument***

print

Syntax	`plDevices('print')`
Description	prints a list of open devices, with SerialNumber, DeviceID, GrabColorConversion and GrabOutputType.
Input	prhs[0] = (string) 'print'
Output	None
Example	`plDevices('print')`

This task is mainly intended for testing and debugging purposes: It prints all entries of the open device array to the screen, one device per line, by using a for-loop (for i = 0 to deviceCount - 1).

- ***2 arguments***

remove

Syntax	`plDevices('remove', serialnumber)`
Description	removes a device from the 'open device array'.
Input	prhs[0] = (string) 'remove' prhs[1] = (U32) serialnumber
Output	None
Example	`plDevices('remove', 75122)`

This task removes a device from the open device list. Using a for-loop (for i = 0 to deviceCount - 1), the given serial number is compared to every serial number in the open device array. As soon as a matching entry is found, a new for-loop is entered (for t = <number of matching entry> to deviceCount - 1) to overwrite every entry, from the matching entry up to the last entry, with the entry immediately following it (this goes wrong if the last entry is 31; see 'Suggestions for improvement'). After the inner for-loop finishes, deviceCount is decremented.

isopen

Syntax	`plDevices('isopen', serialnumber)`
Description	checks whether a device has been opened.
Input	prhs[0] = (string) 'isopen' prhs[1] = (U32) serialnumber
Output	plhs[0] = (double) 1 if the device is open 0 if the device is closed
Example	`plDevices('isopen', 75122)`

This task returns 0 if the device is not open, 1 if it is open. First the return value is set to 0, then a for-loop (for i = 0 to deviceCount - 1) is entered. The given serial number is compared to every serial number in the open device array. As soon as a matching entry is found, the return value is set to 1, and the loop is left using a break statement.

get

Syntax	`deviceId = plDevices('get', serialnumber)`
Description	returns the device ID (handle to the device) of a camera..
Input	prhs[0] = (string) 'get' prhs[1] = (U32) serialnumber
Output	plhs[0] = (int) deviceID if the device is open -1 if the device is closed
Example	`plDevices('remove', 75122)`

This task returns the device ID for a given serial number. It works the same way as isopen, except that the return value is set to -1, and as soon as a matching entry is found, the return value is set to the appropriate device ID.

- ***3 arguments***

add

Syntax	`plDevices('add', serialnumber, deviceID)`
Description	adds a device to the 'open device array'.
Input	prhs[0] = (string) 'add' prhs[1] = (U32) serialnumber prhs[2] = (int) deviceID
Output	None
Example	`plDevices('add', 75122, 256085040)`

This task adds a new device to the open device array. First, it is checked whether there is still room in the array to store the information of one more open device. If the array is full, an error message is printed and plDevices is terminated. If there still is room, the entry after the last entry of the open device array (index: deviceCount, because array indexing starts at deviceCount - 1) is filled with the device ID given in the third argument, and suitable default values for the parameters. After that, deviceCount is incremented.

getpar

Syntax	`parameterValue = plDevices('getpar', serialnumber, parametername)`
Description	returns the value of the specified parameter.
Input	prhs[0] = (string) 'getpar' prhs[1] = (U32) serialnumber prhs[2] = (string) parametername
Output	plhs[0] = value of the specified parameter
Example	`parameterValue = plDevices('getpar', 75122, 'GrabOutputType')`

This task reads the value of the given device parameter from the open device array. First, the third argument, which should be a string with the name of the requested parameter, is loaded into the parName variable. Then the return value (which should be a double) is initialised. It is determined which parameter should be returned by using strcmp in an if/then/else if/else construction. The procedure is the same for every parameter: A temporary boolean variable t is used to indicate whether a matching serial number was not found. It is set to 1, then a (for i = 0 to deviceCount - 1) loop is entered. As soon as a matching serial number is found, the requested parameter is stored as return value, t is set to 0, and the for loop is terminated using a break statement. Finally, if t still has value 1, an error message is printed stating that the requested serial number could not be found.

- ***4 arguments***

setpar

Syntax	`plDevices('setpar', serialnumber, parametername, parametervalue)`
Description	sets the value of the specified parameter.
Input	prhs[0] = (string) 'setpar' prhs[1] = (U32) serialnumber prhs[2] = (string) parametername prhs[3] = value of the specified parameter
Output	None
Example	`plDevices('setpar', 75122, 'GrabOutputType', RAW)` (in which case RAW is defined as 0x0)

This task writes the given value of the given device parameter in the open device array. Its internal structure is almost the same as that of getpar, but the parameter value is written, not read.

- Suggestions for improvement: In 'remove', the inner for loop tries to copy one entry more than needed. When having 32 camera's open, this could result in a segfault. The inner loop for condition should be: (t = i; t < deviceCount - 1; t++)

2.9 Description of plCreateDeviceHandle

plCreateDeviceHandle is a subroutine which is linked to the plOpen and plGet MEX-functions. When given the serial number of a PixeLINK device, it will create a Matlab struct matrix containing all the device's parameter names with their respective values. This is the so-called handle structure, which is returned by plOpen and plGet.

After declaration of the necessary variables and structures, the device-ID for calling the PixeLINK API functions is obtained by calling plDevices. Then, for every parameter, the appropriate API call is made to obtain its value. If the API call returns the value for unsupported function, the string 'Unsupported' is stored as parameter value. If the API returns any other error code, the string 'Could not get value' is stored as parameter value. No other error checking is being done. If everything went okay, the value of the requested parameter is stored in a temporary variable by the PixeLINK API and immediately after that it is stored into the handle structure using mxSetField.

The only exception to this are the parameters 'GrabColorConversion' and 'GrabOutputType', which are not stored in the PixeLINK API, but in plDevices. They are obtained in a similar way to the device-ID by calling plDevices using mexCallMatlab.

- Suggestions for improvement: Convert some of the if-constructions to switch constructions (for example the one used for SubWindowSize).

2.10 Description of plError

plError is an error-checking function which is linked to every MEX-function that uses PixeLINK API calls. It is called after every call to such an API function. Required inputs are the PixeLINK API return-code returned by the called API function and a string describing in max. 33 characters what the program was doing when the possible error occurred.

First, the string input of the plError function is used to create an error message 'The device's API encountered a problem while <string>:'. When the PixeLINK API return code (which is defined in PimMegaApiUser.h) indicates an error, this string is printed on the screen.

After that, a case construction is used to determine the nature of the PixeLINK API return code. When everything is okay, plError returns a value of 0 (false). When an error occurred, plError prints an appropriate descriptive message on the screen using mexPrintf and returns a non-zero value (true). To make it possible to distinguish between critical and non-critical errors, a value of 1 is returned on an error that might not be critical. A value of 2 is returned on errors that are always considered critical. Where appropriate, the program can decide on this information to continue or to terminate.

- Suggestions for improvement: The entire building of the string used in 'The device's API encountered...' can be moved within the following 'if (result != ApiSucces)' condition.

2.11 Description of plGetValue

plGetValue is a subroutine which is linked to the plGet MEX-function. When given an mxArray for returning the parameter value, the device's serial number and the name of the parameter to be returned, it will return the parameter's value or values in the given mxArray.

plGetValue is very similar to plCreateDeviceHandle. The most important difference is that not all possible parameters of the device are obtained, but only the parameter or structure of which the name was given. This is accomplished by using an if/elseif construction together with multiple strcmp statements. An obvious difference is that the mxArray m can differ in size according to the parameter that is stored in it.

When an unknown parameter name is given, plGetValue prints a list of known parameter names to the screen.

2.12 Description of plSetValue

plSetValue is a subroutine linked to the plSet MEX-function. It needs 4 or more input arguments: the serial number of the device, the name of the parameter to be set, the number of values that are passed, all values that are needed to set the parameter.

First, the serial number is used to obtain the PixeLINK device-ID by calling plDevices. For every parameter name, it is checked using an if/strcmp construction whether the right number and type of values are given. If not, a descriptive error message is printed and plSetValue is terminated using a call to mexErrMsgTxt.

After this, another if/elseif/strcmp construction is used to distinguish between the different parameter names. For every parameter name, the passed values are copied into a variable, parsed if necessary and a call is made to the relevant PimMegaSet* function. For the 'GrabColorConversion' and the 'GrabOutputType' parameters, a call is made to plDevices as these are stored there instead of in the PixeLINK API.

- Errors found in the PixeLINK API documentation:

[Vitana 2002-1], p.92, 'PimMegaSetSubWindow'

(uStartColumn + uNumberColumns) must be less than PCS2112_MAX_WIDTH.
(uStartRow + uNumberRows) must be less than PCS2112_MAX_HEIGHT.

'less than' should be 'less than or equal to'.

--

[Vitana 2002-1], p.94, 'PimMegaSetSubWindowPos'

(uStartColumn + current number of columns) must be less than PCS2112_MAX_WIDTH.
(uStartRow + current number of rows) must be less than PCS2112_MAX_HEIGHT.

'less than' should be 'less than or equal to'.

--

[Vitana 2002-1], p.95, 'PimMegaSetSubWindowSize'

(Current start column + Width) must be less than PCS2112_MAX_WIDTH.
(Current start row + Height) must be less than PCS2112_MAX_HEIGHT.

'less than' should be 'less than or equal to'.

2.13 Description of plPrintPossibleValues

plPrintPossibleValues is a subroutine linked to the plGet and plSet MEX-functions. When given the name (string) of a parameter, it will use strcmp in an if/else if construction to determine what should be printed. Then a short description of the given parametername and its possible values is printed to the screen using mexPrintf.

2.14 Description of plTypes

plTypes.h contains all definitions that are used by the plFGI program. Currently, the only values that are defined here are those used to store the 'GrabOutputType' variable.

2.15 General hints on modifying

- When adding or removing a device parameter, this should be done in the obvious places in the following files: plCreateDeviceHandle.cpp, plGetValue.cpp, plPrintPossibleValues.cpp, plSetValue.cpp and plSet.cpp. Also the 'nof' variable and the 'fieldnames' structure in plCreateDeviceHandle should be updated accordingly. Please also don't forget to update the parameter overview in plGetValue.

Bibliography

[Vitana, 2002-1] VITANA CORPORATION (2000-2002). *PixeLINK Megapixel FireWire Camera Developer's Manual, Release 3.0 (online version)*. Vitana Corporation, Ottawa, Ontario, Canada.

[Vitana, 2002-2] VITANA CORPORATION (2000-2002). *PixeLINK Megapixel FireWire Camera User's Manual, Release 3.0 (online version)*. Vitana Corporation, Ottawa, Ontario, Canada.

Appendix A Parameters in the device's handle structure

DeviceID
BlueGain
CurrentFrameRate
DataTransferSize
Exposure
ExposureTime
Gamma
Gpo
GreenGain
HardwareVersion.ProductID HardwareVersion.SerialNumber HardwareVersion.FirmwareVersion HardwareVersion.FpgaVersion
ImagerChipId
ImagerClocking
ImagerName
ImagerType
MonoGain
PreviewWindowPos.Top PreviewWindowPos.Left PreviewWindowSize.Height PreviewWindowSize.Width
RedGain
Saturation
SerialNumber
SoftwareVersion
SubWindow.Decimation SubWindow.StartRow SubWindow.StartColumn SubWindow.NumberRows SubWindow.NumberColumns
SubWindowPos.StartRow SubWindowPos.StartColumn
SubWindowSize.Decimation SubWindowSize.Height SubWindowSize.Width
Timeout
VideoMode
GrabColorConversion
GrabOutputType

Table 1 ***A list of all parameters that are present in the handle structure of the device***

Appendix B Parameters for use with get and set

Parameter name		Type	Range	Default	Granularity	availability	
						get	set
BlueGain		U32	0 .. 63	31	1	+	+
CurrentFrameRate		FLOAT		-	-	+	-
DataTransferSize		STRING	DATA_8BIT_SIZE, DATA_16BIT_SIZE	8BIT	-	+	+
Exposure		U32	0 .. 2046	2046		+	+
ExposureTime	Time	FLOAT				+	+
	ChangeClockSpeed	STRING	TRUE, FALSE	FALSE	-	-	+
Gamma		FLOAT	0 .. 1	1		+	+
Gpo		U32	0, 1	0	1	+	+
GreenGain		U32	0 .. 63	31	1	+	+
HardwareVersion	ProductID	STRING	-	-	-	+	-
	SerialNumber	STRING	-	-	-	+	-
	FirmwareVersion	STRING	-	-	-	+	-
	FpgaVersion	STRING	-	-	-	+	-
ImagerChipId		U32	-	-	-	+	-
ImagerClocking		STRING	0x00, 0x01, 0x02, 0x80, 0x81,0x82	0x00	-	+	+
ImagerName		STRING	-	ImagerX	-	+	+
ImagerType		STRING	PCS2112M_IMAGER, PCS2112C_IMAGER	-	-	+	-
MonoGain		U32	0 .. 63	31	1	+	+
OverlayCallBack		-	-	-	-	-	N/I[5]
PreviewColorConversion		STRING	(see table 3)	(table 3)	-	-	+
PreviewWindow		-	-	-	-	-	N/I
PreviewWindowPos	Top	LONG		200		+	-
	Left	LONG		200		+	-
PreviewWindowSize	Height	U32	0 .. 1024		8	+	-
	Width	U32	0 .. 1280		8	+	-
RedGain		U32	0 .. 63	31	1	+	+
Saturation		U32		96		+	+
SerialNumber		U32	-	-	-	+	-
SoftwareVersion		U32	-	-	-	+	-
SubWindow	Decimation	STRING	(see table 4)	(table 4)	-	+	+
	StartRow	U32		0		+	+
	StartColumn	U32		0		+	+
	NumberRows	U32	0 .. 1024	480	8	+	+
	NumberColumns	U32	0 .. 1280	640	8	+	+
SubWindowPos	StartRow	U32		0		+	+
	StartColumn	U32		0			
SubWindowSize	Decimation	STRING	(see table 4)	(table 4)	-	+	+
	Height	U32		480	8		
	Width	U32		640	8		
Timeout		U32		1000		+	+
VideoMode		STRING	STILL_MODE, VIDEO_MODE	VIDEO	-	+	+
GrabColorConversion		STRING	(see table 3)	(table 3)	-	+	+

[5] N/I means 'Not Implemented'

GrabOutputType	STRING	RAW, IMAGE, RGB24	RAW	-	+	+

Table 2

BAYER_2BY2_COLOR	Fastest
BAYER_3BY3_COLOR	Fast – Default
BAYER_3BY3GGRAD_COLOR	Best quality for real-time
BAYER_2PASSGRAD_COLOR	Best for captured images
BAYER_2PASSADAPT_COLOR	Best for captured images
BAYER_VARGRAD_COLOR	Best for captured images
BAYER_2BY2_MONO	Fastest (converts to monochrome)
BAYER_3BY3_MONO	Best quality for real-time (to monochrome)
BAYER_ADAPT_MONO	Best for captured images (to monochrome)
BAYER_NO_CONVERSION	No Bayer conversion

Table 3

PCS2112_NO_DECIMATION	Original resolution – Default
PCS2112_DECIMATE_BY_2	Resolution / 2, resulting height and width may be rounded down to accommodate the camera's decimation algorithms
PCS2112_DECIMATE_BY_4	Resolution / 4, resulting height and width may be rounded down to accommodate the camera's decimation algorithms

Table 4

Appendix C Source file dependencies

plClose	plClose.cpp, plError.cpp, plError.h
plDevices	plDevices.cpp, plTypes.h
plGet	plGet.cpp, plCreateDeviceHandle.cpp, plGetValue.cpp, plError.cpp, plCreateDeviceHandle.h, plGetValue.h, plError.h, plTypes.h
plGrab	plGrab.cpp. plError.cpp, plError.h, plTypes.h
plIsOpen	plIsOpen.cpp
plOpen	plOpen.cpp, plCreateDeviceHandle.cpp, plError.cpp, plCreateDeviceHandle.h, plError.h
plSet	plSet.cpp, plPrintPossibleValues.cpp, plSetValue.cpp, plError.cpp, plPrintPossibleValues.h, plSetValue.h, plError.h, plTypes.h

Table 5

Appendix D Topics for improvement

This is a full list of all topics for improvement on plFGI:

- plGrab: Implementation of the 5 other ways to call plGrab:
  - moviematrix = plGrab(handle, imgmatrix)
  - [imgmatrix, moviematrix] = plGrab(handle)
  - plGrab(handle, imgmatrix)
  - plGrab(handle, imgmatrix, moviematrix)
  - plGrab(handle, imgmatrix, moviematrix, scaling factor)
- plGrab: Addition of RGB48 format for 16-bit captures
- plGrab: Testing the plGrab function with a colour camera (this should work)
- plGrab: Loop optimalisations
- plGrab: Capture a RAW image directly into the left hand array (*plhs[]) of the mexFunction
- plOpen: Use plError after the call to PimMegaGetNumberDevices instead of non standard error code
- plSet: Implementation of PimMegaSetOverlayCallBack
- plSet: Implementation of PimMegaSetPreviewWindow
- plSet: Implementation of PimMegaAutoExposure
- plDevices: In 'remove', the inner for loop tries to copy one entry more than needed. When having 32 camera's open, this could result in a segfault. The inner loop for condition should be: (t = i; t < deviceCount - 1; t++)
- plCreateDeviceHandle: Convert some of the if-constructions to switch constructions (for example the one used for SubWindowSize)
- plError: The entire building of the string used in 'The device's API encountered...' can be moved within the following 'if (result != ApiSucces)' condition.
- Function to check whether the camera is connected to the computer
- Reset function, to set all parameters to their default value
- Demo M-files
- Additional tests on the performance of plFGI

Appendix E Test results

Below are the screen printouts of the tests performed on plFGI:

```
>> plset(m,'DataTransferSize','DATA_8BIT_SIZE');
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

   65.8000

>> plset(m,'SubWindowSize','PCS2112_NO_DECIMATION', 512, 640);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

   29.6600

>> plset(m,'SubWindowSize','PCS2112_NO_DECIMATION', 256, 640);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

   17.3000

>> plset(m,'SubWindowSize','PCS2112_NO_DECIMATION', 1024, 640);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

   55.8000

>> plset(m,'SubWindowSize','PCS2112_NO_DECIMATION', 1024, 320);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

   52.7800

>> plset(m,'ExposureTime',1);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

   51.3500

>> plget(m)

ans =

                        DeviceID: 256106896
                        BlueGain: 'Unsupported'
                CurrentFrameRate: 'Could not get value'
                DataTransferSize: 'DATA_8BIT_SIZE'
                        Exposure: 1028
                    ExposureTime: 1.0400
                           Gamma: 1
                             Gpo: 'Off'
                       GreenGain: 'Unsupported'
      HardwareVersion.ProductID: 'PL-A630 Series Monochrome EC Module'
   HardwareVersion.SerialNumber: '12572'
HardwareVersion.FirmwareVersion: '1.0.3.0'
    HardwareVersion.FpgaVersion: '4'
                  ImagerChipId: 805306368
                ImagerClocking: '0x01 External (16Mhz) Division by 2'
                    ImagerName: 'Imager0'
                    ImagerType: 'PCS2112M_IMAGER (Monochrome Camera)'
                      MonoGain: 0
          PreviewWindowPos.Top: 'Could not get value'
         PreviewWindowPos.Left: 'Could not get value'
      PreviewWindowSize.Height: 'Could not get value'
```

```
 PreviewWindowSize.Width: 'Could not get value'
                  RedGain: 'Unsupported'
              Saturation: 'Unsupported'
            SerialNumber: 75122
         SoftwareVersion: 16777984
    SubWindow.Decimation: 'PCS2112_NO_DECIMATION'
      SubWindow.StartRow: 0
   SubWindow.StartColumn: 0
    SubWindow.NumberRows: 1024
 SubWindow.NumberColumns: 320
   SubWindowPos.StartRow: 0
SubWindowPos.StartColumn: 0
SubWindowSize.Decimation: 'PCS2112_NO_DECIMATION'
    SubWindowSize.Height: 1024
     SubWindowSize.Width: 320
                 Timeout: 1000
               VideoMode: 'VIDEO_MODE'
     GrabColorConversion: 'BAYER_3BY3_COLOR'
          GrabOutputType: 'RAW'
```