

Real Time Computer Vision Development
for a 2-DOF Gimbal Application

Sakari Kettunen

Center for Self-Organizing and Intelligent Systems (CSOIS)
Department of Electrical and Computer Engineering
College of Engineering, Utah State University
4160 Old Main Hill, Logan Utah 84322-4160, USA
sakari.kettunen@hut.fi
Web: <http://www.csois.usu.edu/>

Jan 24th 2005



Technical Report No. : [USU-CSOIS-TR-05-01](#)

Reports are available from <http://mechatronics.ece.usu.edu/reports/>

Table of Contents

Introduction.....	2
The goal of the project	3
Image acquisition program.....	5
How to use the program	5
Problems during programming	6
Saving the path.....	7
Using ‘savePath.m’	8
Controlling Gimbal	9
Controlling the robot.....	9
Arrangement in this project.....	9
The Simulink models used in this project.....	9
Math-file ‘ILCprocess.m’	12
Using Gimbal and ‘ILCprocess.m’	12
Conclusions.....	14
Bibliography.....	15
Appendix I.....	16
Appendix II	22
Appendix III.....	29

Introduction

This document is about the project that I made for the Center of Self Organizing and Intelligent Systems (CSOIS), Utah State University, Logan, Utah, USA, during the fall semester 2005. The project was about a robot called Gimbal and using it for implementing some parts of Lili Ma's PhD thesis [4]. In this project Gimbal is used for implementing Iterative Learning Control in visual based control.

The files used and produced in this project can be found on the project CD.

Contents of the CD:

/Gimbal/	All necessary files for using ILCprocess.m. Cleaned up from /Sakari/gimbal_interface/.
/Gimbal/ImageAcquire/	Files for image acquisition program. (Cleaned up)
/Matlab/	Working folder for image acquisition program.
/Sakari/	Working folder for controlling Gimbal.
/Sakari/gimbal_interface/	Includes the program used in the demo. original version of the model created by LingHong Zhu [3].

I want to thank everyone who gave me their help and support.

Especially I want to thank

Professor YangQuan Chen
PhD Lili Ma
LingHong Zhu

Jan 24th in Helsinki, Finland

Sakari Kettunen

The goal of the project

The Gimbal is a 2 DOF rotating robot. It consists of a vertically rotating shaft extending into a fork and another shaft placed in between the fork ends, which rotates horizontally [2, App. I]. A laser pointer can be attached to Gimbal's horizontally rotating shaft and the laser can be directed to point certain direction by rotating Gimbal.

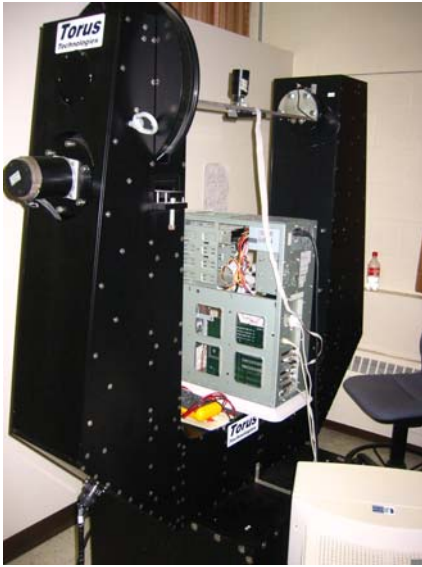


Figure 1 Gimbal



Figure 2 Project arrangement with camera

The goal of the project was to be able to track an undetermined line on the paper/wall with the pointer of the laser. The camera was used as the feedback for the system. The camera can be located without restrains as long as the track can be seen in the picture. (See Figure 2) Because of this arrangement the camera will provide inaccurate and uncalibrated information. Therefore we use Iterative Learning Control (ILC) to find the best output for the system.

Because of the limited time I had in use and the troubles I had with the system during the project I was not able to implement Iterative Learning Control this time. Therefore, the project for me consisted of following parts:

- Programming of image acquisition program.
- Creating a program for saving the curve drawn by the laser.
- Integrating robot control, image acquisition and image processing in to one computer.
- Creating a program to be used as a basis for ILC with camera feedback and testing it in practice.

In other words, I made the preparations for the “real work” with ILC for someone else to be done in the future. Anyhow, this has been a very educative project for me in digital camera solutions and robot control. Here is the list of the programs/technologies I used in this project:

Visual C++ 6.0
MatLab 6.5
Simulink
Quanser Toolbox
Quanser MQ8 data acquisition and control board
C-MEX protocol
Data Translation DT-3153 frame grabber board
Pulnix TMC-7DSP digital camera
WinCon server/client

The specific descriptions of different parts of the project can be read from the following chapters.

Image acquisition program

The first part of the project was to create a program, which grabs a frame from digital camera (Pulnix TMC-7DSP connected to DT 3153 frame grabber board) and transfers it to MatLab workspace. Similar image acquisition is available in MatLab Image Acquisition ToolBox. By making the program by ourselves, it will become more modifiable and cheaper.

The program was programmed by using C programming language and C-MEX protocol. C-MEX protocol enables importing of C-language programs into Matlab. More information about C-MEX is available in [MathWorks support](#). [1]

I used example files ("acq2host.c" and "acq2host.h") provided by Data Translation as a basis for the program. The major modifications I made to the program were adding the C-MEX function (called the gateway routine) and modifying the type of grabbed image data. In the original program acquired image was only shown in a pop-up window. In the new program we needed the picture to be stored into a 3D array, which could be transferred into MatLab. Also the presentation of the grabbed image would be done from MatLab, if needed.

The program must be compiled and run using MatLab. See chapter "How to use the program".

When the program is run for the first time, the camera is initialized. After that program acquires an image or closes the camera, depending if there is any parameter. If there is a parameter, camera will close. NOTE! Camera has to be closed before you exit MatLab.

Image acquisition works as follows:

- The frame is read to the allocated buffer.
- Buffer is read to 3D array created by mex function "mxCreateNumericArray".
- Array is transferred to MatLab.
- Transferred array must be processed in Matlab using matrix functions before it can be handled as a normal RGB picture. See "Acquire.m".

The image buffer which is read from the camera is in different dimensions than the normal MatLab RGB image. Buffer dimensions are 4x640x480, and MatLab RGB dimensions for the same picture are 480x640x3. (Alpha channel can not be used in MatLab.) This is why we have to process the array to be able to see the actual image in MatLab. Matrix functions are easy and fast to use in MatLab which is why operations are done in there instead of the c-program.

How to use the program

To compile program in MatLab, use command

```
'mex FrameCapt.c DtColorSdk.lib DtLineScan.lib OIlg32.lib Olimg32.lib'
```

Make sure you have all the libraries in the same folder as your FrameCapt.c. Select VC++ as the compiler.

After compiling, you can use command 'FrameCapt'

1. To initialize the camera.

If camera is not initialized and you give the command

'FrameCapt',

camera will be initialized. After successful initialization 'Camera initialized' is prompted.

NOTE! You have to wait a while (>0.5 seconds) after the camera initialization before you acquire the image. Otherwise the picture will contain errors.

2. To acquire image data to MatLab workspace.

If the camera is initialized, use command

```
'Acquire'
```

to acquire image.

NOTE! Acquired data is at first in incompatible order in MatLab. You have to change the dimensions of the data to be able to handle it as a RGB image. Therefore, use math-file 'Acquire.m' (or your own similar function) to acquire image, instead of just 'FrameCapt'. See below for details of 'Acquire.m'.

3. To close camera.

You must close the camera before you exit MatLab. Otherwise you will lose the "camera thread" and you have to reboot the computer to be able to access the camera again. To close the camera, give a parameter (what ever, e.g. 0 or 1) for the command 'FrameCapt'. For example:

```
'FrameCapt(1)'
```

'Acquire.m' includes the following code:

```
%Capture frame
pict = FrameCapt;

%From 4x640x480 to 640x480x3
picture(:,:,1) = pict(3,:,:);
picture(:,:,2) = pict(2,:,:);
picture(:,:,3) = pict(1,:,:);

%From 640x480x3 to 480x640x3
newImage(:,:,1) = picture(:,:,1).';
newImage(:,:,2) = picture(:,:,2).';
newImage(:,:,3) = picture(:,:,3).';
```

newImage is now in compatible form for further image processing.

Problems during programming

Acquiring the data to MatLab was relatively easy. However, the picture did not seem to be right at all at first. After going over the data I noticed that it was in wrong dimensions. The alpha channel was always zero, so I was able to see the structure of the image from the data. Using MatLab's matrix operations I was able to find the right dimensions.

The other time consuming problem was solved really simply. The problem was that when I tested my acquiring program, the resulting picture seemed to have errors. Only around every 20th picture seemed to be ok. In my test program I initialized the camera, acquired the image, and closed the camera. It turned out that I Acquired the picture too fast after the initialization of the camera.

Camera needs some time to "wake up". After I added a delay before the acquisition, everything went well.

The code for the program can be found in Appendix I and II.

Saving the path

The second part of the project was to create a program (MatLab m-file) which saves the curve that the laser pointer draws in the paper. The curve was to be saved as set of points in one picture. From this picture we will be able to extract the error for the desired path and the actual path. This error will be used for Iterative Learning Control.

This kind of function is actually really easy to implement. The code ('savePath.m') I wrote looks like this:

```
imSize = size(oldImage);

imSize2 = size(newImage);

if(imSize-imSize2 ~= 0)
    error('Image sizes do not match');
end

pathImage = oldImage;

for i=1 : imSize(2)
    for j=1 : imSize(1)

        %check if pixel is red, then save it
        if(newImage(j,i,1) > 245 && newImage(j,i,2) > 150
            && newImage(j,i,3) > 150)
            pathImage(j,i,1) = newImage(j,i,1);
            pathImage(j,i,2) = newImage(j,i,2);
            pathImage(j,i,3) = newImage(j,i,3);
        end
    end
end
```

In the for-loop we go thru every pixel of the picture, and if the pixel is red, it is copied to 'pathImage'. The pixel is considered red if the red value (of RGB pixel) is over 245 and green and blue values are under 150.

NOTE! If you want to change the sensitivity of the pixel saving, you can change the values in the if-statement

```
if(newImage(j,i,1) > 245 && newImage(j,i,2) > 150 && newImage(j,i,3) > 150),
```

where `newImage(j,i,1)` is the red channel, `newImage(j,i,2)` is the green channel and `newImage(j,i,3)` is the blue channel of the pixel.

NOTE! The line

```
pathImage = oldImage;
```

is unnecessary. It can be removed, if variable 'oldImage' on the first line is changed to 'pathImage'. Also, if this line is removed, you must remove all of the lines

```
oldImage = pathImage;
```

from the file 'ILCprocess.m'. Sorry about that! Those lines are remains of earlier versions of the function. Read more about the file 'ILCprocess.m' from the chapter "Gimbal".

NOTE! I think it would be wise to change this math-file into a function that returns the 'pathImage'. The usage would be easier because you would not have to know in which variables the pictures and the path are saved. Now variables 'newImage' and 'pathImage' are static variables that can not be seen from the main program 'savePath.m'.

Using 'savePath.m'

Before you can use the command 'savePath' in Matlab, you have to take the picture. This can be done using the command 'Acquire' (see chapter "Image Acquisition Program"). Remember to initialize the camera before grabbing the image! 'Acquire' saves the acquired picture to the array 'newImage' in MatLab workspace. This array is used in 'savePath.m' to find the point of the laser in the picture. The found point is copied to array 'pathImage' in MatLab workspace.

Controlling Gimbal

The third part of the project was to integrate image acquisition system and gimbal controlling system in to one computer. Until now image acquisition and image processing has been made in two separate computers. Also, the goal was to be able to do everything from MatLab: Image acquisition, image processing and robot control. Before, there was a different program for image acquisition which wrote the image to a file and the picture was then loaded to MatLab. Two separate computers made the using of the program very complicated.

Controlling the robot

Robot control was done by using Simulink and Quanser Toolbox (WinLib). The Gimbal was connected to the computer thru Quanser data acquisition and control board MQ8. WinCon (provided by Quanser) provides the program interfaces between the Simulink generated C-code with MQ8 board. The WinCon client is installed on the host computer with the MQ8 DACB. The WinCon server is either installed on the host computer or a remote computer. [2, App. I]

Arrangement in this project

Despite several attempts and for some unknown reason, we were not able to install the frame grabber board (DT-3153) and the data acquisition board (MQ8) on the same computer. Therefore we still had to use two separate computers for the project. The other one had the Quanser board installed and the other one had the frame grabber board installed. However, the goal of only one computer was basically achieved; now we just had the WinCon client on the other computer and the WinCon server and the frame grabber board on a “remote” computer. The robot control, the image acquisition and the image processing is now done in one “remote” computer. The remote connection between the two computers was implemented with Ethernet. In the future, when it will be possible to install both boards on one computer, no modifications need to be done. You just do not need to create the remote connection between WinCon server and WinCon client.

The Simulink models used in this project

To move Gimbal, I modified some Simulink models created by Mohua [2] and LingHong[3]. They have controlled Gimble in previous projects. To be able to test this system, I basically needed to move Gimbal and then record the path of the laser with the camera program. Therefore there was no need to create completely new models for Gimbal in this phase. Some modifications had to be done, especially to Mohua’s models, because she had used another data acquisition board, Quanser MULTI-Q3, in her project. The models used in this project are introduced next.

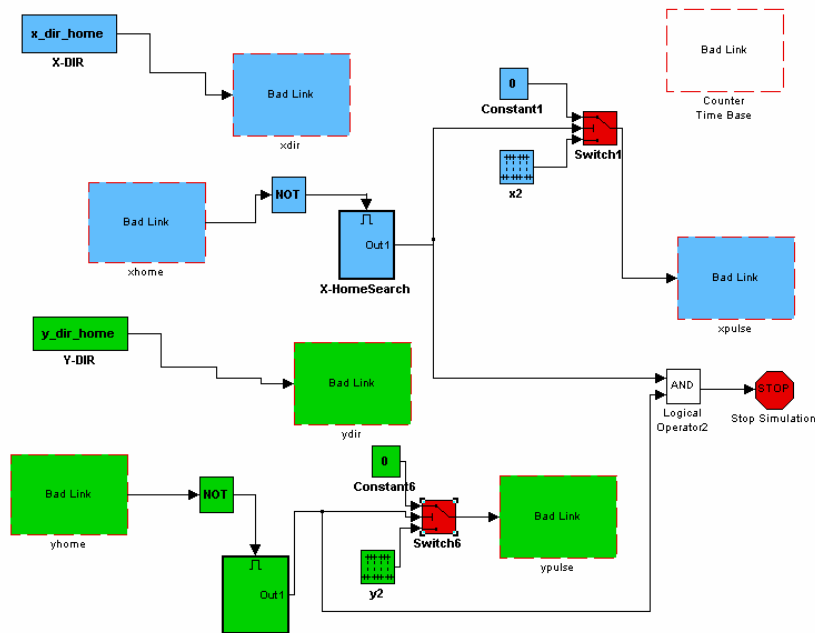


Figure 3 'Homef.mdl' - Home point search

Simulink model 'Homef.mdl' [2] was used to find the home position of Gimbal. That is, the zero angles of both axes. Home point sensors were attached to Gimbal to be able to find the zero angles. Home point search will be used in iteration of the Iterative Learning Control function, because it is the only way to find the same starting point for iterations. Direction variables 'X-DIR' and 'Y-DIR' can be set to values 0 or 4, depending of the desired direction.

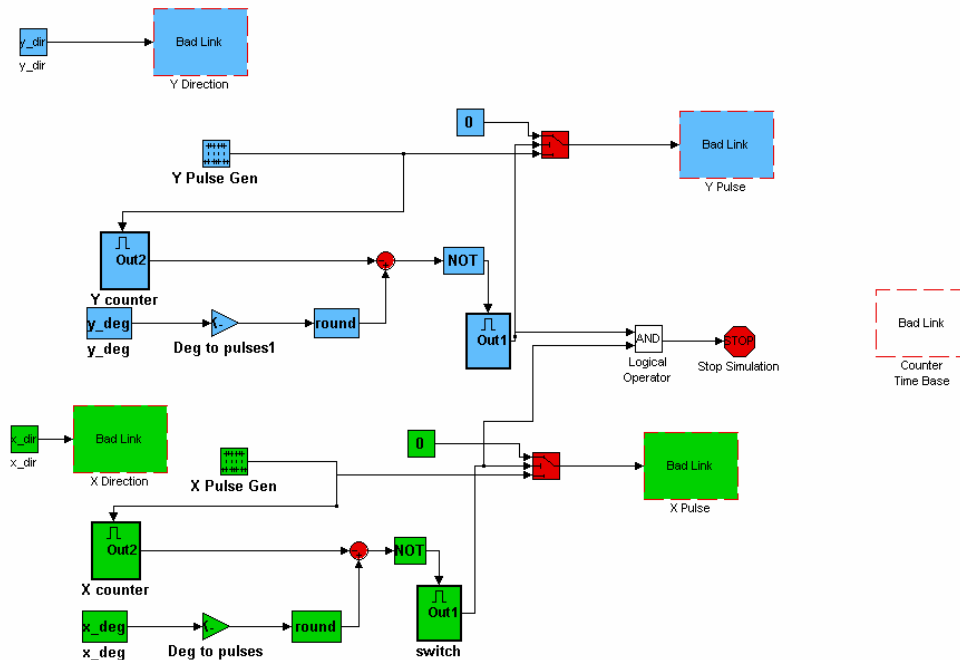


Figure 4 'Point.mdl' - Move to desired point/angle

'Point.mdl'[2] can be used for finding the starting point of the curve. Both axes will move straight to the desired angle (x_deg, y_deg). Simulation will stop when both axes have reached the desired angle.

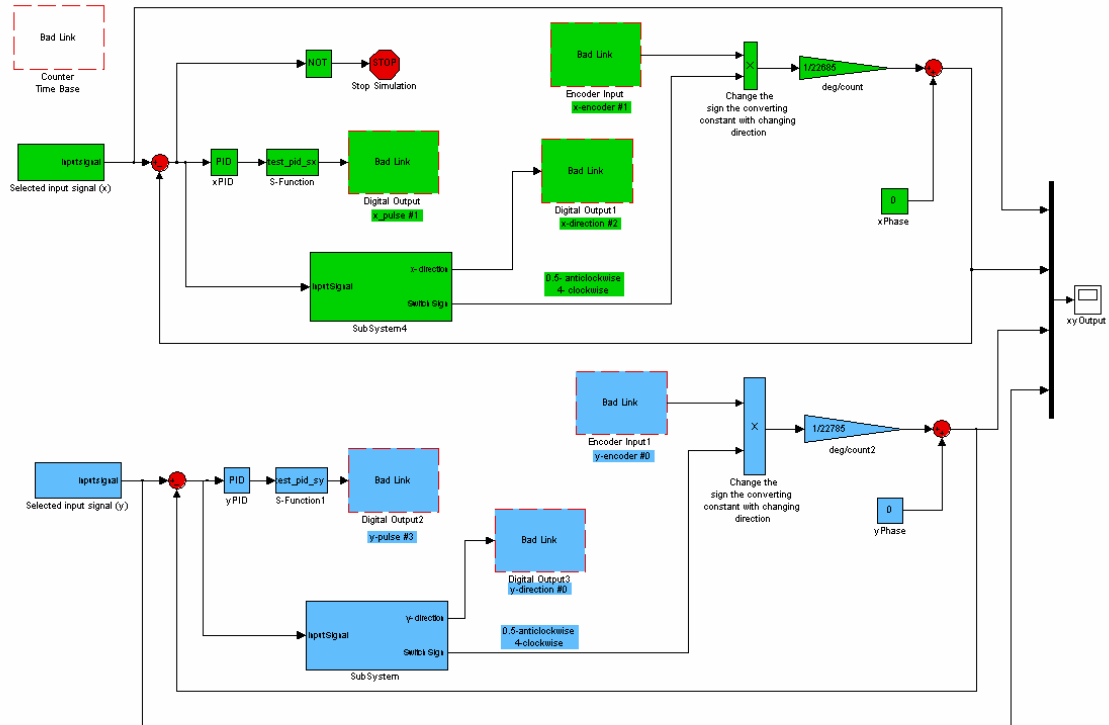


Figure 5 'Model.mdl' - Drawing the curve

I used the model 'Model.mdl', created by LingHong[3], to draw the curve on the paper. It was really easy to use for drawing sine wave or circle. For drawing more complicated curves, a new model should be created.

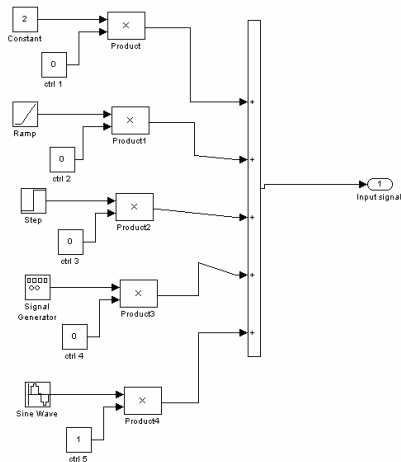


Figure 6 Subsystem 'selected input signal' from 'Model.mdl'

Kuva 1 Subsystem 'selected input signal' from 'Model.mdl'

Math-file 'ILCprocess.m'

To test and demonstrate the working of the system I created a math-file 'ILCprocess.m'. Besides of testing the system for this project, it can be used as a basis for further development of the system. I created the file so that it reminds the ILC process. I also added some (possible) function calls that could be used in ILC process. They are commented away, because I never implemented those functions. They are there just to give ideas for future developers of the system.

'ILCprocess.m' draws iteratively a sine wave on the wall with the laser pointer attached to Gimbal with three different starting points. The iteration process is "dumb" and there is no intelligence in changing the starting point of the curve. Intelligence can be added by creating and using functions that defines new starting points using the data saved from the camera.

The image acquisition program is used to save the path drawn by the laser. It is saved to RGB picture called 'pathImage'. It can be used for finding the errors between the desired and the actual curve.

WinCon commands are used to build and start/stop the process automatically when needed. These commands let us control the WinCon server straight from the math-file, without needing to handle it manually.

'ILCprocess.m' can be found from Appendix III.

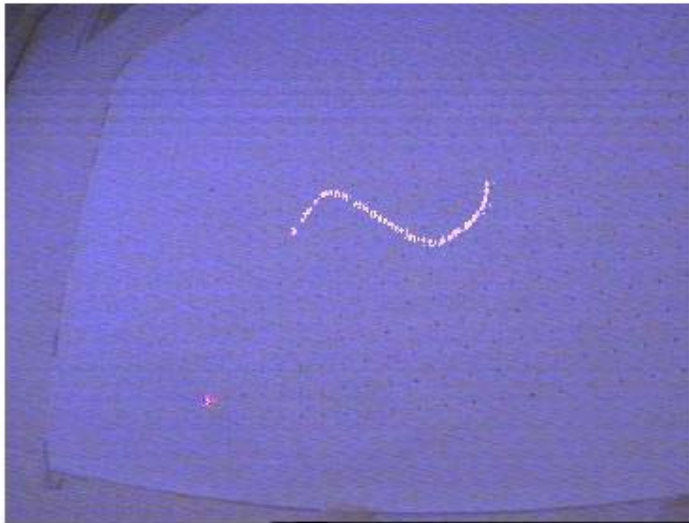


Figure 7 One image saved by 'ILCprocess.m'

Using Gimbal and 'ILCprocess.m'

All the files needed to run ILCprocess can be found in the cd from folder

F:/Gimbal OR

F:/Sakari/gimbal_interface/

Folder 'Gimbal' includes only the necessary files for the project; the other folder is a working folder with extra files. Remember to compile image acquisition program into MatLab workspace before you run ILCprocess. (see chapter "Image Acquisition Program").

At first, because we use two computers in this project, you have to create the remote connection between the WinCon server and the WinCon client. Do the following:

- Start WinCon client in the computer with the Quanser data acquisition board.
- Start WinCon server in the remote computer.
- In WinCon server, select 'Connect...'
- Type in the IP address of the WinCon client computer.

The connection is ok if there is a mark next to the IP address of the client computer in the connection menu of WinCon client.

NOTE! If you run the Simulink control models from the same computer in which you have the data acquisition board installed, you do not have to create this kind of remote connection. Starting the model will automatically connect WinCon server and WinCon client on the same computer.

IMPORTANT! Before you connect the Gimbal to electrical network, you have to set all the variables to zero. This can be done with the Simulink model 'initialization.mdl'. All you need to do is to open the model and select 'Start' from the WinCon menu. After the model has started running, you can connect Gimbal to electrical network. Then stop the process and close the model without saving changes.

IMPORTANT! Remember to unplug Gimbal from electrical network after you stop working. Turning off the computer that is controlling Gimbal when Gimbal is still connected to electricity will cause Gimbal to start turning itself. This will most likely break Gimbal. Usually, when computer is turned off too early, Gimbal will start to make nasty noise. Then, unplug Gimbal as soon as possible. Otherwise the motors of Gimbal will break.

After initialization you have to run 'homef.mdl' to find the initial position for the ILC process. This model will turn the Gimbal to zero angles. If running the model turns Gimbal to wrong direction, you have to change variables 'X-DIR' and 'Y-DIR'. Select from two values, 0 or 4. Changing these variables makes Gimbal turn to the other direction.

When Gimbal is in home position, you can run 'ICLprocess.m'. This is done by opening the models used in the file (homef.mdl, point.mdl and model.mdl) and giving the command 'ILCprocess' in MatLab. Three iterations will be done with different starting points.

Conclusions

The reason why I run out of time with this project, and was not able to implement the ILC process can be blamed on two reasons: first, the equipment used was a bit out-of-date and second, I did not ask help directly. I tried to solve every problem myself first, and then ask help. For some problems, there was no solution available. In addition, you always have to prepare yourself to use about 50 % of the time to common problems with computers (and Windows) in this kind of projects. This time one the biggest problems was the incapability to install the frame grabber board and the data acquisition board to one computer.

Anyhow, the work I have done should be a good basis for the future work in this project.

For improving the reusability of the math files, most of them should be changed into functions. This helps programmers who use the functions in the future. The usage will be clearer and more specific.

Bibliography

- [1] The MathWorks - Support - MEX-files Guide
<http://www.mathworks.com/support/tech-notes/1600/1605.html>
- [2] Mohua Ghosh, Master's Thesis, ECE Dept. of Utah State University, 2004.
- [3] LingHong Zhu, Master's Thesis, ECE Dept. of Utah State University, 2005.
- [4] Lili Ma, "Vision-Based Measurements for Dynamic Systems and Control", PhD Thesis, ECE Dept. of Utah State University, 2004

Appendix I

```
`FrameCapt.h'

/*
    This program is based on a sample program called
    "acq2host.c"/"acq2host.h"
    produced by Data Translation.

    Program grabs an image from camera (Pulnix TMC-7DSP connected to DT 3153
    frame grabber board) and saves it into an 3D array. This array is transferred
    into
    Matlab using CMEX protocol. Program must be compiled and executed from Matlab.

    Tranferred array must be processed in Matlab using matrix functions before it
    can be handled as a normal RGB picture. See "Acquire.m".

    Date: 11/29/04

    Author: Sakari Kettunen
*/

#if !defined(FrameCapt_H__)
#define FrameCapt_H__ 1

#include <windows.h>          /* required for all Windows applications */
*/
#include <windowsx.h>        /* some nice macros, etc. */

#include <assert.h>          /* for asertion testing */
#include <stdlib.h>          /* for atoi() prototype */
#include <memory.h>          /* for _fmemset() */
#include <stdio.h>           /* for sprintf() */
#include <ctype.h>           /* for ascii test macros */
#include <string.h>

#include "olwintyp.h"        /* standard DT-Open Layers data types
and symbolic constants */
#include "olimgapi.h"        /* interface to DT-Open Layers Imaging
API */
#include "olfgapi.h"        /* interface to DT-Open Layers Frame
Grabber API */
#include "olimgdev.h"
#include "olfg.h"
#include "dtColorSdk.h"
//#include "resource.h"

/* Assume C declarations for C++ callers */
#ifdef __cplusplus
extern "C" {
#endif

/* Data types used by app */

/* This structure holds information about the range of capabilities supported */
```

```

/*    by the current device                                     */
typedef struct tagDEVCAPS
{
    BOOL DoesInputFilter;
    BOOL DoesA2D;
    BOOL DoesVideoType;
    BOOL DoesActiveVideo;
    BOOL DoesFrameSize;
    BOOL DoesClock;
    BOOL DoesSyncSentinel;
    ULNG FrameTypes;
    ULNG SourceCount;
} DEVCAPS, FAR *LPDEVCAPS;

/* This structure is intended to hold current settings and other info about */
/* the current device                                                         */
typedef struct tagDEVINFO
{
    char Alias[OLC_MAX_ALIAS_STR_SIZE];
    OLT_IMG_DEV_ID DevId;
    USHRT InputSource;
} DEVINFO, FAR *LPDEVINFO;

/* This structure is used to hold the limits and current values for the video */
/* area controls of the frame grabber.                                         */
typedef struct tagVIDEOAREAINFO
{
    OLT_LNG_RANGE BackPorchStartRange;
    OLT_LNG_RANGE ClampStartRange;
    OLT_LNG_RANGE ClampEndRange;
    OLT_LNG_RANGE PixelsPerLineRange;
    OLT_LNG_RANGE FirstPixelRange;
    OLT_LNG_RANGE ActivePixelsRange;
    OLT_LNG_RANGE LinesPerFieldRange;
    OLT_LNG_RANGE FirstLineRange;
    OLT_LNG_RANGE ActiveLinesRange;
    ULNG BackPorchStart;
    ULNG ClampStart;
    ULNG ClampEnd;
    ULNG PixelsPerLine;
    ULNG FirstPixel;
    ULNG ActivePixels;
    ULNG LinesPerField;
    ULNG FirstLine;
    ULNG ActiveLines;
} VIDEOAREAINFO, FAR *LPVIDEOAREAINFO;

/* This structure is used to hold the limits and current values for the frame */
/* size controls of the frame grabber.                                         */
typedef struct tagFRAMESIZEINFO
{
    OLT_LNG_RANGE LeftRange;

```

```

        OLT_LNG_RANGE TopRange;
        OLT_LNG_RANGE WidthRange;
        OLT_LNG_RANGE HeightRange;
        OLT_LNG_RANGE HorizontalIncRange;
        OLT_LNG_RANGE VerticalIncRange;
        ULNG Left;
        ULNG Top;
        ULNG Width;
        ULNG Height;
        ULNG HorizontalInc;
        ULNG VerticalInc;
    } FRAMESIZEINFO, FAR *LPFRAMESIZEINFO;

/* This structure is used to store the values read from the sync
   sentinel dialog box edit controls. */

typedef struct tagSYNCSSENTINFO
{
    char szHSearch[20];
    char szVSearch[20];
    char szHInsert[20];
    char szVInsert[20];
} SYNCSSENTINFO, FAR * LPSYNCSSENTINFO;

/* This structure is used for the Cleanup function that is called
   if there is a memory allocation problem when creating a DIB. */

typedef struct tagPointerList
{
    HGLOBAL hPal;
    HPALETTE hGrayscalePal;
    LPLOGPALETTE lpPal;
    HGLOBAL hDIBdata;
    HPUCHR hpDIBdata;
    HGLOBAL hDIBheader;
    BITMAPINFO * lpDIBheader;
} POINTERLIST, FAR *LPPOINTERLIST;

// *****
//
DT3157 - STUFF
// *****

/* DT3157 Digital camera types */
typedef enum DT3157_DIGITAL_CAMERA_TYPES
{
    DT3157_DIGCAM_16BIT_INPUT           = 0x0000,
    DT3157_DIGCAM_14BIT_INPUT          = 0x0001,
    DT3157_DIGCAM_12BIT_INPUT          = 0x0002,
    DT3157_DIGCAM_10BIT_INPUT          = 0x0003,
    DT3157_DIGCAM_8BIT_SINGLE_CHANNEL_INPUT = 0x0004,
    DT3157_DIGCAM_8BIT_DUAL_CHANNEL_INPUT  = 0x0005
} DT3157_DIGCAM_TYPE, *PDT3157_DIGCAM_TYPE, FAR
*LPD3157_DIGCAM_TYPE;

/* DT3157 Sync Controls */
typedef enum DT3157_SYNC_CONTROL
{
    DT3157_SYNC_CTL_UNKNOWN           = 0x0000,
    DT3157_SYNC_CTL_HORIZ_FREQ       = 0x0001,
    DT3157_SYNC_CTL_VERT_FREQ        = 0x0002,

```

```

        DT3157_SYNC_CTL_HPULSE_WIDTH    = 0x0003,
        DT3157_SYNC_CTL_VPULSE_WIDTH    = 0x0004,
        DT3157_SYNC_CTL_PHASE           = 0x0005
    } DT3157_SYNC_CONTROL, *PDT3157_SYNC_CONTROL, FAR
*LPDT3157_SYNC_CONTROL;

/* Integration pulse (exposure) settings */

typedef struct DT3157T_Exposure
{
    USHORT input_channel; /* input channel to apply exposure */
    ULONG PulseDuration; /* Integration pulse duration (in usec) */
    ULONG Polarity;      /* logic polarity of pulse to
generate*/
} DT3157T_EXPOSURE, *PDT3157T_EXPOSURE, FAR *LPDT3157T_EXPOSURE;

// These are the function pointer to use if we have a dt3157 board in
// the PC. Otherwise, they won't be used

typedef char STRING[255]; // Standard array of char for a
lpstr

// *****
// *****

/* Global variables. "GLOBAL__" is defined only in the main module. In all
*/
/* other modules, it preprocesses to a declaration, rather than a definition.
*/

#ifdef GLOBAL__
#define GLOBAL
#define EQU(init) = init
#else
#define GLOBAL extern
#define EQU(init)
#endif

/* Flags used by the RdAndChkEditCtlValue to determine the type of operation */

#define INCREMENT 1
#define DECREMENT -1
#define SET 0

GLOBAL DEVCAPS CurDevCaps EQU({0}); /* capabilities of current
device */
//GLOBAL DEVINFO CurDevInfo EQU({0}); /* current device information
*/

//int PASCAL WinMain(HINSTANCE, HINSTANCE, LPSTR, int);
BOOL SelectDevice(int);
BOOL NewDevice(LPCSTR);
BOOL AcquireToHost();

/* Function Prototypes for acq2hst.c */
int PASCAL WinMain(HINSTANCE, HINSTANCE, LPSTR, int);
BOOL PRIVATE InitApplication(HINSTANCE);

```

```

BOOL PRIVATE InitInstance(HINSTANCE, int);
LPARAM CALLBACK __export MainWndProc(HWND, UINT, WPARAM, LPARAM);
BOOL CALLBACK __export About(HWND, UINT, WPARAM, LPARAM);
BOOL CALLBACK __export SelectDeviceProc(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam);
LRESULT PRIVATE CreatedIBDisplay(HPUCHAR hpAcquireBuf, ULONG ulHeight, ULONG
ulWidth, ULONG ulPixelDepth);
BOOL PRIVATE InitDeviceList(HWND hComboBox);
void PRIVATE CloseCurrentDevice(void);
//BOOL PRIVATE NewDevice(LPCSTR lpszAlias);
//void PRIVATE AcquireToHost(DEVINFORM DevInfo);

void PRIVATE CenterWindow(HWND hWnd, HWND hOwner);
void PRIVATE CleanupDIBPointers(LPPOINTERLIST lpPointerList);
BOOL RdAndChkEditCtlVal(HWND hDlg, int Ctl_ID, LPOLTLNG_RANGE lpRange, LPCHR
lpszTitle, LPLNG lpResult, int IncDecSet);
LNG PRIVATE GetDlgItemLong(HWND hDlg, int idControl, LPBOOL lpbWorked);
double PRIVATE GetDlgItemDouble(HWND hDlg, int idControl, LPBOOL lpbWorked);
void PRIVATE PrintStatus (HWND hDlg, OLT_APISTATUS status, LPSTR lpszOpMsg,
LPSTR lpszMBTitle);
void PRIVATE ResetControlFocus (HWND hDlg, int idControl, long lValue);

BOOL CALLBACK __export InputFilterProc(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam);

BOOL CALLBACK __export InputSourceProc(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam);

BOOL CALLBACK __export A2DProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam);
void PRIVATE SetA2DInfo(HWND, LPLNG , LPLNG );

BOOL CALLBACK __export VideoSelectProc(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam);
void PRIVATE InitComboBoxFromControllist(HWND hComboBox,
OLT_FG_INPUT_CAP_KEY ListLimitsKey, OLT_FG_INPUT_CAP_KEY
ListKey);
void PRIVATE InitCSyncSourceList(HWND hComboBox);
void PRIVATE SetCsyncInfo(HWND hDlg);
void PRIVATE SetVarscanInfo(BOOL bLSyncLowToHigh, BOOL bFSyncLowToHigh);

BOOL CALLBACK __export FrameTypeProc(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam);

BOOL CALLBACK __export SyncSentinelProc(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam);
void PRIVATE InitSyncSentinelDlg(HWND hDlg, LPBOOL lpSyncSentinel,
LPSYNCSSENTINFO lpInfo);
void PRIVATE SetSyncSentinelInfo(HWND hDlg, BOOL bSyncSentinel, LPSYNCSSENTINFO
lpInfo);

BOOL CALLBACK __export PixelClockProc(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam);
void PRIVATE InitPixelClockDlg(HWND hDlg, LPBOOL lpInternal, LPBOOL
lpExtClockOnLowToHigh, LPULNG lpCurFreq);
void PRIVATE SetPixelClockInfo(HWND hDlg, BOOL bInternal, BOOL
bExtClockOnLowToHigh, LPULNG lpCurFreq);

```

```

BOOL CALLBACK __export VideoAreaProc(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam);
void PRIVATE InitVideoAreaDlg(HWND hDlg, LPVIDEOAREAINFO lpInfo);
void PRIVATE IncrementVideoControl(HWND hDlg, HWND hControl, LPVIDEOAREAINFO
lpInfo);
void PRIVATE DecrementVideoControl(HWND hDlg, HWND hControl, LPVIDEOAREAINFO
lpInfo);
void PRIVATE MinMaxVideoControl(HWND hDlg, HWND hControl, LPVIDEOAREAINFO
lpInfo, BOOL bMaximize);
void PRIVATE SetVideoAreaInfo(HWND hDlg, LPVIDEOAREAINFO lpInfo);

BOOL CALLBACK __export FrameSizeProc(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam);
void PRIVATE InitFrameSizeDlg(HWND hDlg, LPFRAMESIZEINFO lpInfo);
void PRIVATE IncrementFrameControl(HWND hDlg, HWND hControl, LPFRAMESIZEINFO
lpInfo);
void PRIVATE DecrementFrameControl(HWND hDlg, HWND hControl, LPFRAMESIZEINFO
lpInfo);
void PRIVATE MinMaxFrameControl(HWND hDlg, HWND hControl, LPFRAMESIZEINFO
lpInfo, BOOL bMaximize);
void PRIVATE SetFrameSizeInfo(HWND hDlg, LPFRAMESIZEINFO lpInfo);

BOOL CALLBACK __export TimeoutProc(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam);

/* End of extern "C" { declaration for C++ callers */
#ifdef __cplusplus
}
#endif

#endif /* FrameCapt_H__ */

```

Appendix II

'FrameCapt.c'

```
/*
    This program is based on a sample program called
"acq2host.c"/"acq2host.h"
    produced by Data Translation.

    Program grabs an image from camera (Pulnix TMC-7DSP connected to DT 3153
    frame grabber board) and saves it into an 3D array. This array is transferred
into
    Matlab using CMEX protocol. Program must be compiled and executed from Matlab.

    Tranferred array must be processed in Matlab using matrix functions before it
    can be handled as a normal RGB picture. See "Acquire.m".

    Date: 11/29/04

    Author: Sakari Kettunen
*/

/* Let's use strict type checking in the example */
#define STRICT

#define GLOBAL__
#include <math.h>

#include "FrameCapt.h"
#include "mex.h" /*
Matlab Executable library */

#define NDIMS 3
#define TOTAL_ELEMENTS 1228800//307200// 921600
#define TRIGGER_EXTERNAL_LINE 2

static HINSTANCE hInst;
static BOOL RGBDataFlag =0;

static HBITMAP image;
static BOOL init;

GLOBAL DEVINFO CurDevInfo EQU({0});

BOOL freeMemory();

// These have been moved from Acquire to this place for Drawing RGB data , hence
they need
// to be initialised in Acquire
    ULONG ulHeight = 0L;
    ULONG ulWidth = 0L;
    HGLOBAL hAcquireBuf = NULL;
    HPUCHR hpAcquireBuf = NULL;
    OLT_FG_FRAME_ID FrameId = 0;
    OLT_FG_FRAME_INFO *Frame;
```

```

char *array;

/* The gateway routine */
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    const int dims[] = {4, 640, 480};
    unsigned char *start_of_pr;
    int bytes_to_copy;
    double *x;

    if(nrhs==1) { // if there is a parameter, close camera
        freeMemory();
        init = FALSE;
    } else if(!init){ // if camera not initialized

        if(!SelectDevice(0)){ //Select the first in the list (0)

            mexErrMsgTxt("SelectDevice failed\n");
        }

        if(!NewDevice(CurDevInfo.Alias))
        {
            (void) OlImgCloseDevice(CurDevInfo.DevId);

            mexErrMsgTxt("Init failed\n\nCamera might be
already open. Restart system, if necessary.\n");
        }

        init = TRUE;
        mexPrintf("Camera initialized.\n");
    }else{ // capture image

        if(!AcquireToHost())
        {
            GlobalUnlock (hpAcquireBuf);
            GlobalFree (hAcquireBuf);

            (void)

OlImgCloseDevice (CurDevInfo.DevId);

            mexErrMsgTxt("Acquire Failed\n");
        }

        /* Create a 4-by-640-by-480 array of unsigned
8-bit integers. */
        plhs[0] =
mxCCreateNumericArray (NDIMS, dims, mxUINT8_CLASS, mxREAL);

        /* Populate the real part of the created array.
*/
        start_of_pr = (unsigned char
*)mxGetData (plhs[0]);
        bytes_to_copy = TOTAL_ELEMENTS *
mxGetElementSize (plhs[0]);

```

```

        if(!memcpy(start_of_pr, hpAcquireBuf,
bytes_to_copy))
        {
            GlobalUnlock(hpAcquireBuf);
            GlobalFree(hAcquireBuf);
            OlFgDestroyFrame( CurDevInfo.DevId,
FrameId);
            (void)
OlImgCloseDevice(CurDevInfo.DevId);
            mexErrMsgTxt("Cannot copy.\n");
        }
        GlobalUnlock(hpAcquireBuf);
        GlobalFree(hAcquireBuf);
        OlFgDestroyFrame( CurDevInfo.DevId, FrameId);
    }
}

//free allocated memory

BOOL freeMemory()
{
    GlobalUnlock(hpAcquireBuf);
    GlobalFree(hAcquireBuf);

    OlFgDestroyFrame( CurDevInfo.DevId, FrameId);

    (void) OlImgCloseDevice(CurDevInfo.DevId);
    mexPrintf("Camera closed.\n");
}

BOOL SelectDevice(int index)
{
    int iCount = 0;
    OLT_APISTATUS Status = OLC_STS_NORMAL;
    HGLOBAL hDevInfoList = NULL;
    LPOLT_IMGDEVINFO lpDevInfoList = NULL;
    int i = 0;

    /* How many aliases do we have */

    if ( (Status = OlImgGetDeviceCount(&iCount)) != OLC_STS_NORMAL )
    {
        FILE *file_out;
        file_out = fopen ("ERROR4.txt", "w");
        fprintf(file_out, "OlImgGetDeviceCount failed (status =
%#08lx).", Status);
        return FALSE;
    }

    /* Anything? */

    if ( !iCount )
        return TRUE;
}

```

```

/* Allocate space for info */

    hDevInfoList = GlobalAlloc(GHND, iCount * sizeof(OLT_IMGDEVINFO));
    if ( !hDevInfoList )
    {
        FILE *file_out;
        file_out = fopen ("ERROR0.txt", "w");
        fprintf(file_out, "Unable to allocate memory for device list;
GlobalAlloc failed.");
        return FALSE;
    }

    lpDevInfoList = (LPOLT_IMGDEVINFO) GlobalLock(hDevInfoList);
    if ( !lpDevInfoList )
    {
        FILE *file_out;
        file_out = fopen ("ERROR1.txt", "w");
        fprintf(file_out, "Unable to lock memory for device list;
GlobalLock failed.");
        GlobalFree(hDevInfoList);
        return FALSE;
    }

/* Fill in the device info */

    lpDevInfoList->StructSize = sizeof(OLT_IMGDEVINFO);
    if ( (Status = OlImgGetDeviceInfo(lpDevInfoList, iCount *
sizeof(OLT_IMGDEVINFO))) )
    {
        FILE *file_out;
        file_out = fopen ("ERROR2.txt", "w");
        fprintf(file_out, "OlImgGetDeviceInfo failed (status =
%#08lx).", Status);
        GlobalUnlock(hDevInfoList);
        GlobalFree(hDevInfoList);
        return FALSE;
    }

/* Select wanted alias */

    strcpy(CurDevInfo.Alias,lpDevInfoList[index].Alias);

/* Free the memory */

    GlobalUnlock(hDevInfoList);
    GlobalFree(hDevInfoList);

    return TRUE;
}

/*****

BOOL  NewDevice(LPCSTR lpCszAlias)
{
{
    OLT_IMG_DEV_ID DevId = NULL;
    OLT_APISTATUS Status = OLC_STS_NORMAL;
    char DeviceName[20];

```

```

/* Try to open the device */

Status = OIImgOpenDevice(lpVtblAlias, &DevId);

if ( ! OIImgIsOkay( Status ) )
{
    mexPrintf("OIImgOpenDevice failed.\n");
    return FALSE;
}

CurDevInfo.DevId = DevId;
CurDevInfo.InputSource = 0;

OIImgQueryDeviceCaps (CurDevInfo.DevId, OLC_IMG_DC_DEVICE_NAME, DeviceName, sizeof (STRING));

if ( (!strcmp("DT3153", DeviceName)) ||
      (!strcmp("DT3154", DeviceName)) ||
      (!strcmp("DT3133", DeviceName)) ||
      (!strcmp("DT3132", DeviceName)) ||
      (!strcmp("DT3131", DeviceName)) ||
      (!strcmp("DT3130", DeviceName)) ||
      (!strcmp("DT3120", DeviceName)) )

{

    // This is hardcoded to avoid having to include
    // header files that are related only to the
    // color SDK.
    #define WRITE 8
    #define RGB_32 7
    #define RGB_24 6
    #define RGB_16 4
    #define MONO 1

    OLT_APISTATUS (*StorageMode) (OLT_IMG_DEV_ID, unsigned
int, unsigned int, unsigned int*);
    HINSTANCE DtColorLib;
    unsigned int value = RGB_32;

    StorageMode = NULL;
    DtColorLib = LoadLibraryEx("DtColorSDK", NULL, 0);

    StorageMode = (void*)
GetProcAddress (DtColorLib, "DtColorStorageMode");

    if (StorageMode)
    {
        // Set the board in 32 bits RGB
        StorageMode (CurDevInfo.DevId, 0, WRITE, &value);
    }
}

return TRUE;
}
}

```

```

/*          Function:  AcquireToHost      */
BOOL  AcquireToHost()
{
    OLT_APISTATUS Status = 0L;
    ULNG  ulPixelDepth = 0L;
    ULNG  ulMinBufSize = 0L;

    Frame = GlobalAlloc (GHND, sizeof(OLT_FG_FRAME_INFO));

    ulHeight = 0L;
    ulWidth = 0L;
    GlobalUnlock(hAcquireBuf);
    GlobalFree (hAcquireBuf);

    hAcquireBuf = NULL;
    hpAcquireBuf = NULL;

    // Query for Height, Width and Pixel Depth of the frame

    (void) OlFgQueryInputControlValue (CurDevInfo.DevId,
                                        CurDevInfo.InputSource,
                                        OLC_FG_CTL_FRAME_HEIGHT,
                                        &ulHeight);

    (void) OlFgQueryInputControlValue (CurDevInfo.DevId,
                                        CurDevInfo.InputSource,
                                        OLC_FG_CTL_FRAME_WIDTH,
                                        &ulWidth);

    (void) OlFgQueryInputCaps (CurDevInfo.DevId,
                              OLC_FG_IC_PIXEL_DEPTH,
                              &ulPixelDepth,
                              sizeof(ULNG));

    // calculate minimum size of buffer in bytes *

    ulMinBufSize = ulHeight * ulWidth * ulPixelDepth;

    hAcquireBuf = GlobalAlloc (GHND, ulMinBufSize);

    if ( !hAcquireBuf )
    {
        // unable to allocate memory
        mexPrintf("Unable to allocate enough memory for
acquire.\n");
        return FALSE;
    }

    hpAcquireBuf = (HPUCHAR) GlobalLock(hAcquireBuf);
    if ( !hpAcquireBuf)
    {
        // unable to lock memory

        TCHAR  szBuf[80];
        LPVOID lpMsgBuf;
        DWORD  dw = GetLastError();

```

```

        FormatMessage(
FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM,
NULL,
dw,
MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
(LPTSTR) &lpMsgBuf,
0, NULL );

        sprintf(szBuf,
"Failed with error %d: %s", dw, lpMsgBuf);

        mexPrintf("Unable to lock memory for
acquire.\n%s\n", szBuf);

        LocalFree(lpMsgBuf);

        return FALSE;
    }

    Status = OlFgAllocateBuiltInFrame ( CurDevInfo.DevId,

        OLC_FG_DEV_MEM_VOLATILE,

        OLC_FG_NEXT_FRAME,

        &FrameId);
    if ( !OlImgIsOkay(Status) )
    {
        mexPrintf("Unable to Allocate Frame.");
    }

    // Have the memory, attempt the acquire
    Status = OlFgAcquireFrameToHost(CurDevInfo.DevId, FrameId,
hpAcquireBuf, ulMinBufSize);

    if ( !OlImgIsOkay(Status) )
    {
        mexPrintf("Unable to acquire.");
        (void) OlFgDestroyFrame( CurDevInfo.DevId, FrameId);
        GlobalUnlock(hAcquireBuf);
        GlobalFree(hAcquireBuf);
        return FALSE;
    }

    return TRUE;
}

```

Appendix III

'ILCprocess.m'

```
clear;
```

```
k = 0;
```

```
while(k < 3)
```

```
    %starting point search model name:  
    mn = 'point';
```

```
    FrameCapt; %Initialize camera
```

```
    pause(1);
```

```
    Acquire; %Take the first picture
```

```
    oldImage = newImage; %Saves taken image to the pathImage
```

```
    if(k == 0)
```

```
        y_dir = 4; %remove these, when you have destination point functions  
        y_deg = 5;  
        x_dir = 4;  
        x_deg = 5;
```

```
        %SetDestinationPoint(newImage) % Define starting point and destination  
point from
```

```
                                % picture and set parameters for model  
                                % (x_dir, x_deg, y_dir, y_deg)
```

```
        wc_build(mn) % let's build with new parameters  
    end
```

```
    wc_start(mn)  
    pause(1);  
    while(wc_isrunning(mn))  
        pause(1);  
    end
```

```
    wc_stop(mn);
```

```
    Acquire; %Take picture at the end  
    savePath;  
    oldImage = pathImage;
```

```
    %figure;  
    %imshow(pathImage);
```

```
    %DefineError(pathImage) % Define error of destination point  
    %SetDestinationPoint() % and actual point. Set model parameters  
                                % again, according to defined error.
```

```
    x_deg = x_deg - 1; % remove these, when you have the error functions above  
    y_deg = y_deg + 1;
```

```
    %% End of start point search
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The actual curve: % -----
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pause(2);

%curve model name:

%mn = 'point';
%mn = 'homef';
mn = 'model';
%mn = 'circle3';
%mn = 'ysin9'

% Uncomment these two, if you don't want the home point in the picture

%Acquire; %Take the first picture
%oldImage = newImage; %Saves taken image to the pathImage

if(k == 0)
    %SetCurveParameters(newImage) % Define desired curve from
                                % picture and set parameters for model
    wc_build(mn) % let's build with new parameters
end

wc_start(mn)
pause(1);
while(wc_isrunning(mn))

    % Save laser point every 0.5 second
    Acquire;
    savePath;
    oldImage = pathImage;
    pause(0.1);

end

wc_stop(mn);

figure;
imshow(pathImage);

%error = DefineError(pathImage) % Define error for actual and desired
                                % curves. ILC. Set model parameters
                                % again, according to defined error.

%if(error < e)
%    break;
%end

x_dir_home = abs(x_dir - 4); % Home is in the opposite direction to start
point (usually)
y_dir_home = abs(y_dir - 4);

wc_start('homef'); % Go back to home point
pause(1);
while(wc_isrunning('homef'))
    pause(1);
end

```

```
    wc_stop('homef');  
    %pause(1);  
  
    k = k + 1;  
  
end  
  
FrameCapt(1); %Close camera
```