# Software

## Image Processing Algorithm

//---------------------------------------------------------------------------

// Name: pointStack.h

// Author: Nick Bertrand

//---------------------------------------------------------------------------

#ifndef pointStack\_

#define pointStack\_

struct point

**{**

int x**,**y**;**

**};**

// Point stack for floodFill algorithm

// Implented as a linked list

struct pointStackElement

**{**

point elementData**;**

pointStackElement**\*** nextElement**;**

**};**

// Pop/Push. Head set to zero if empty.

point pointStackPop**(**pointStackElement**\*\*** head**);**

void pointStackPush**(**pointStackElement**\*\*** head**,** int x**,** int y**);**

void pointStackPrint**(**pointStackElement**\*** head**);**

#endif

//---------------------------------------------------------------------------

// Name: eyeFeatureExtraction.cpp

// Author: Nick Bertrand, Armeen Taeb

//---------------------------------------------------------------------------

#include "stdafx.h"

#include <highgui.h>

#include <conio.h>

#include <Windows.h>

#include "eyeFeatureExtraction.h"

#include "eFEParam.h"

extern param p**;**

extern point**\*** cRPointList**;**

extern unsigned char**\*** cRBinary**;**

extern int**\*** cRMap**;**

extern int**\*** cRSizes0**;**

extern int**\*** cRSizes1**;**

extern int cRCount**;**

extern enum resultType procResult**;**

extern int cursorCommand**;**

extern int maxRegionSize**;**

extern unsigned char**\*** processedPixels**;**

extern unsigned char**\*** gSImg**;**

extern int**\*** candidateRegionIndices**;**

extern int candidateRegionCount**;**

extern double**\*** cRAspectRatio**;**

extern int doCalibration**;**

extern point**\*** removedPoints**;**

extern int removedPointCount**;**

extern point centroid**;**

extern int consecDirFrame**;**

extern int prevResultType**;**

extern double maxLengthConnected**;**

extern double minLengthConnected**;**

extern point dirArray**[**5**];**

extern int cursorSpeed**;**

extern point prevCentroid**;**

extern int numBlinks**;**

#define RGB2GS(X,Y) 0.1140\*data[(X)\*step+(Y)\*channels+0]+0.5870\*data[(X)\*step+(Y)\*channels+1]+0.2989\*data[(X)\*step+(Y)\*channels+2]

#define I2DFULL(X,Y) ((X)\*width + (Y))

//#define MOUSEEVENTF\_LEFTDOWN 0x0002

//#define MOUSEEVENTF\_LEFTUP 0x0004

void processFrame**(**IplImage **\***img**)**

**{**

point centroid **=** **{**0**,**0**};**

int numAdapt **=** 0**;**

int threshold **=** p**.**initThreshold**;**

procResult **=** isBlink**;** // Assume initially that user is blinking for each frame

int unityIndex **=** **-**1**;**

fillGreyScale**(**img**);**

**while** **(**numAdapt **<** p**.**maxAdaptations **&&** procResult **!=** isPupil**)**

**{**

getConnectedRegions**(**threshold**);**

#ifdef DEBUG\_OUTPUT

printf**(**"Max region size: %u\n"**,** maxRegionSize**);**

#endif

**if(**cRCount **==** 0**)**

**{**

**if(** maxRegionSize **>** p**.**refSizeMax **)**

threshold **-=** p**.**magThreshChange**[**numAdapt**];**

**else** **if(**maxRegionSize **<** p**.**refSizeMin**)**

threshold **+=** p**.**magThreshChange**[**numAdapt**];**

**}**

**else**

**{**

// Compute aspectRatio finds the index of connected region that has closest to unity aspect ratio

unityIndex **=** findUnityRatio**();**

**if(** unityIndex **==** **-**1 **)**

**{**

**if(** maxRegionSize **>** p**.**refSize **)**

threshold **-=** p**.**magThreshChange**[**numAdapt**];**

**else**

threshold **+=** p**.**magThreshChange**[**numAdapt**];**

**}**

**else**

**{**

procResult **=** isPupil**;**

removedPointCount **=** 0**;**

removeAberrations**(**unityIndex**);**

**}**

**}**

**++**numAdapt**;**

**}**

**if** **(**procResult **==** isPupil**)**

**{**

centroid **=** computeCentroid**(**unityIndex**);**

**}**

**else**

**{**

centroid**.**x **=** 0**;** centroid**.**y **=** 0**;**

**}**

#ifdef CALIBRATION\_ACTIVE

**if(**doCalibration **==** 1 **&&** procResult **==** isPupil**)**

**{**

p**.**refCentroid **=** centroid**;**

p**.**refSize **=** cRSizes0**[**unityIndex**];**

p**.**lengthRegion **=** 2**\***sqrt**((**p**.**refSize**)/(**3.1415**));**

p**.**initThreshold **=** threshold**;**

doCalibration **=** 0**;**

**}**

#endif

#ifdef DEBUG\_OUTPUT

printf**(**"Reference Centroid: (%u, %u)\n"**,** p**.**refCentroid**.**x**,** p**.**refCentroid**.**y**);**

printf**(**"Centroid: (%u, %u)\n"**,** centroid**.**x**,** centroid**.**y**);**

#endif

generateCursorCommand**(**centroid**);**

addProcessingOverlay**(**img**,** unityIndex**,** centroid**);**

**}**

void fillGreyScale**(**IplImage **\***img**)**

**{**

int width **=** img**->**width**;**

int height **=** img**->**height**;**

int channels **=** img**->**nChannels**;**

int step **=** img**->**widthStep**;**

int i**,**j**;**

uchar**\*** data **=** **(**uchar **\*)**img**->**imageData**;**

**for(**i **=** p**.**iStart**;** i **<=** p**.**iFinish**;** **++**i**)**

**{**

**for(**j **=** p**.**jStart**;** j **<=** p**.**jFinish**;** **++**j**)**

**{**

gSImg**[**I2DFULL**(**i**,**j**)]** **=** **(**unsigned char**)(**RGB2GS**(**i**,**j**));**

**}**

**}**

**}**

void getConnectedRegions**(**int threshold**)**

**{**

int width **=** p**.**imgWidth**;**

int height **=** p**.**imgHeight**;**

int iStart **=** p**.**iStart**;**

int iFinish **=** p**.**iFinish**;**

int jStart **=** p**.**jStart**;**

int jFinish **=** p**.**jFinish**;**

int totalPixels **=** p**.**totalPixels**;**

int procRegionArea **=** p**.**procRegioniSize **\*** p**.**procRegionjSize**;**

double refSizeMin **=** p**.**refSizeMin**;**

double refSizeMax **=** p**.**refSizeMax**;**

int i**,**j**;**

int coord**;**

int currentPixel**;**

point pt**;**

int currentRegion **=** 0**;**

point **\***pointStack**;**

int stackIndex**;**

int maxSizeOverall **=** 0**;**

int maxSizeProper **=** 0**;**

pointStack **=** **(**point**\*)**malloc**(**procRegionArea **\*** **sizeof(**point**));**

stackIndex **=** 0**;**

memset**(**processedPixels**,** 0**,**width**\***height**\*sizeof(** unsigned char**));**

memset**(**cRBinary**,** 0**,** totalPixels **\*** **sizeof(**unsigned char**));**

memset**(**cRMap**,** 0**,** totalPixels **\*** **sizeof(**int**));**

#ifdef DEBUG\_OUTPUT

printf**(**"Getting connected regions...\n"**);**

printf**(**"Threshold: %u\n"**,** threshold**);**

#endif

**for(**i **=** iStart**;** i **<=** iFinish**;** **++**i**)**

**for(**j **=** jStart**;** j **<=** jFinish**;** **++**j**)**

**{**

**{**

**if(** gSImg**[**I2DFULL**(**i**,**j**)]** **<** threshold **&&** processedPixels**[**I2DFULL**(**i**,**j**)]** **==** 0 **)**

**{**

currentPixel **=** 0**;**

pt**.**x **=** i**;** pt**.**y **=** j**;**

pointStack**[**stackIndex**++]** **=** pt**;**

processedPixels**[**I2DFULL**(**i**,**j**)]** **=** 1**;**

// Do flood fill algorithm

**while(**stackIndex **!=** 0**)**

**{**

pt **=** pointStack**[--**stackIndex**];**

coord **=** I3D**(**currentRegion**,** pt**.**x**-**iStart**,** pt**.**y**-**jStart**);**

cRBinary**[**coord**]** **=** 1**;**

cRMap**[**coord**]** **=** currentPixel**;**

cRPointList**[**I2D**(**currentRegion**,**currentPixel**)]** **=** pt**;**

currentPixel**++;**

**if(** **(** pt**.**x **+** 1 **<=** iFinish **)** **&&** **(** processedPixels**[**I2DFULL**(**pt**.**x **+** 1**,** pt**.**y**)]** **==** 0 **)** **&&** **(** gSImg**[**I2DFULL**(**pt**.**x **+** 1**,** pt**.**y**)]** **<** threshold **))**

**{**

pointStack**[**stackIndex**].**x **=** pt**.**x **+** 1**;**

pointStack**[**stackIndex**++].**y **=** pt**.**y**;**

processedPixels**[**I2DFULL**(**pt**.**x **+** 1**,** pt**.**y**)]** **=** 1**;**

**}**

**if(** **(** pt**.**x **-** 1 **>=** iStart **)** **&&** **(** processedPixels**[**I2DFULL**(**pt**.**x **-** 1**,** pt**.**y**)]** **==** 0 **)** **&&** **(** gSImg**[**I2DFULL**(**pt**.**x **-** 1**,** pt**.**y**)]** **<** threshold **))**

**{**

pointStack**[**stackIndex**].**x **=** pt**.**x **-** 1**;**

pointStack**[**stackIndex**++].**y **=** pt**.**y**;**

processedPixels**[**I2DFULL**(**pt**.**x **-** 1**,** pt**.**y**)]** **=** 1**;**

**}**

**if(** **(** pt**.**y **+** 1 **<=** jFinish **)** **&&** **(** processedPixels**[**I2DFULL**(**pt**.**x**,** pt**.**y **+** 1**)]** **==** 0 **)** **&&** **(** gSImg**[**I2DFULL**(**pt**.**x**,** pt**.**y **+** 1**)]** **<** threshold **))**

**{**

pointStack**[**stackIndex**].**x **=** pt**.**x**;**

pointStack**[**stackIndex**++].**y **=** pt**.**y **+** 1**;**

processedPixels**[**I2DFULL**(**pt**.**x**,** pt**.**y **+** 1**)]** **=** 1**;**

**}**

**if(** **(** pt**.**y **-** 1 **>=** jStart **)** **&&** **(** processedPixels**[**I2DFULL**(**pt**.**x**,** pt**.**y **-** 1**)]** **==** 0 **)** **&&** **(** gSImg**[**I2DFULL**(**pt**.**x**,** pt**.**y **-** 1**)]** **<** threshold **))**

**{**

pointStack**[**stackIndex**].**x **=** pt**.**x**;**

pointStack**[**stackIndex**++].**y **=** pt**.**y **-** 1**;**

processedPixels**[**I2DFULL**(**pt**.**x**,** pt**.**y **-** 1**)]** **=** 1**;**

**}**

**}** // End flood fill

**if(** currentPixel **<=** refSizeMax **&&** currentPixel **>=** refSizeMin **)**

**{**

cRSizes0**[**currentRegion**++]** **=** currentPixel**;**

**if(** currentPixel **>** maxSizeProper**)**

maxSizeProper **=** currentPixel**;**

**}**

**else**

**{**

memset**(**cRBinary **+** currentRegion**\***procRegionArea**,** 0**,** **sizeof(**unsigned char**)\***procRegionArea**);**

memset**(**cRMap **+** currentRegion**\***procRegionArea**,** 0**,** **sizeof(**int**)\***procRegionArea**);**

**}**

**if(** currentPixel **>** maxSizeOverall**)**

maxSizeOverall **=** currentPixel**;**

**if(** currentRegion **==** MAX\_TOTAL\_REGIONS**)**

printf**(**"Maxed number of connected regions reached.\n"**);**

**}**

**else**

processedPixels**[**I2DFULL**(**i**,**j**)]** **=** 1**;**

**}**

**}**

cRCount **=** currentRegion**;**

candidateRegionCount **=** currentRegion**;**

**if(** cRCount **!=** 0 **)**

maxRegionSize **=** maxSizeProper**;**

**else**

maxRegionSize **=** maxSizeOverall**;**

#ifdef DEBUG\_OUTPUT

printf**(**"Connected regions found: %u\n"**,** cRCount**);**

#endif

free**(**pointStack**);**

**}**

// Goal: Compute the aspect ratio of the connected regions and find the ones closest to 1.

int findUnityRatio**()**

**{**

int i**,**j**,**k**;**

int lengthx**,** lengthy **=** 0**;**

int xcount**,** ycount **=** 0**;** // horizontal and vertical length of the image

double dummyratio **=** 0**;** // dummy aspect ratio

int unityIndex **=** **-**1**;** // Output index value

point candidateCentroid**;** // centroid of the candidate region

double centroidChange**;** // The amount that the candidate region's centroid changed from centroid of previously found pupil

maxLengthConnected **=** p**.**lengthRegion**\***p**.**lengthMaxRatio**;**

minLengthConnected **=** p**.**lengthRegion**\***p**.**lengthMinRatio**;**

candidateRegionCount **=** 0**;**

**for** **(**k **=** 0**;** k **<** cRCount**;** **++**k**)**

**{**

xcount **=** 0**;**

ycount **=** 0**;**

**for** **(**i **=** 0**;** i **<** p**.**procRegioniSize**;** **++**i**)**

**{**

lengthx **=** 0**;**

**for** **(**j **=** 0**;** j **<** p**.**procRegionjSize**;** **++**j**)**

**{**

lengthx **+=** **(**int**)**cRBinary**[**I3D**(**k**,**i**,**j**)];**

**}**

**if** **(**lengthx **>** xcount**)**

**{**

xcount **=** lengthx**;**

**}**

**}**

**for** **(**j **=** 0**;** j **<** p**.**procRegionjSize**;** **++**j**)**

**{**

lengthy **=** 0**;**

**for** **(**i **=** 0**;** i **<** p**.**procRegioniSize**;** **++**i**)**

**{**

lengthy **+=** **(**int**)**cRBinary**[**I3D**(**k**,**i**,**j**)];**

**}**

**if** **(**lengthy **>** ycount**)**

**{**

ycount **=** lengthy**;**

**}**

**}**

candidateCentroid **=** computeCentroid**(**k**);**

**if** **(**procResult **!=** isBlink**)**

**{**

centroidChange **=** sqrt**((**double**)(**candidateCentroid**.**x**-**prevCentroid**.**x**)\*(**candidateCentroid**.**x**-**prevCentroid**.**x**)+(**candidateCentroid**.**y**-**prevCentroid**.**y**)\*(**candidateCentroid**.**y**-**prevCentroid**.**y**));**

**}**

**else**

**{**

centroidChange **=** 0**;**

**}**

cRAspectRatio**[**k**]** **=** **(**double**)** **(**ycount**)/(**xcount**);**

#ifdef DEBUG\_OUTPUT

printf**(**"Region: %u\n"**,** k**);**

printf**(**"Size: %u\n"**,** cRSizes0**[**k**]);**

printf**(**"LengthX: %u, LengthY: %u\n"**,** xcount**,** ycount**);**

printf**(**"Aspect ratio for region %u: %lf\n"**,** k**,** cRAspectRatio**[**k**]);**

#endif

**if** **(**cRAspectRatio**[**k**]** **<** p**.**aspectMax **&&** cRAspectRatio**[**k**]** **>** p**.**aspectMin**)**

**{**

#ifdef CALIBRATION\_ACTIVE

**if(**doCalibration **==** 1**)**

**{**

candidateRegionIndices**[**candidateRegionCount**++]** **=** k**;**

**if** **(**abs**(**dummyratio**-**1**)** **>** abs**(**cRAspectRatio**[**k**]-**1**))**

**{**

dummyratio **=** cRAspectRatio**[**k**];**

unityIndex **=** k**;**

**}**

procResult **=** isPupil**;**

**}**

**else**

**{**

// Additional check to make sure that the x length and y length of the candidate region is acceptable

**if** **(**ycount **<** maxLengthConnected **&&** ycount **>** minLengthConnected **&&** xcount **<** maxLengthConnected **&&** xcount **>** minLengthConnected **&&** centroidChange **<** p**.**maxcentroidChange**)**

**{**

candidateRegionIndices**[**candidateRegionCount**++]** **=** k**;**

**if** **(**abs**(**dummyratio**-**1**)** **>** abs**(**cRAspectRatio**[**k**]-**1**))**

**{**

dummyratio **=** cRAspectRatio**[**k**];**

unityIndex **=** k**;**

prevCentroid **=** candidateCentroid**;**

**}**

procResult **=** isPupil**;**

**}**

**}**

#else

**if** **(**ycount **<** maxLengthConnected **&&** ycount **>** minLengthConnected **&&** xcount **<** maxLengthConnected **&&** xcount **>** minLengthConnected**)**

**{**

candidateRegionIndices**[**candidateRegionCount**++]** **=** k**;**

**if** **(**abs**(**dummyratio**-**1**)** **>** abs**(**cRAspectRatio**[**k**]-**1**))**

**{**

dummyratio **=** cRAspectRatio**[**k**];**

unityIndex **=** k**;**

**}**

procResult **=** isPupil**;**

**}**

#endif

**}**

**}**

**return** unityIndex**;**

**}**

// Goal: Find the centroid of the connected region

point computeCentroid**(**int unityIndex**)**

**{**

int i**;**

int sumx **=** 0**,** sumy **=** 0**;**

point centroid**;**

**for** **(**i **=** 0**;** i **<** cRSizes0**[**unityIndex**];** **++**i**)**

**{**

sumx **+=** cRPointList**[**I2D**(**unityIndex**,**i**)].**x**;**

sumy **+=** cRPointList**[**I2D**(**unityIndex**,**i**)].**y**;**

**}**

centroid**.**y **=** **(**int**)** sumx**/(**cRSizes0**[**unityIndex**]-**removedPointCount**);**

centroid**.**x **=** **(**int**)** sumy**/(**cRSizes0**[**unityIndex**]-**removedPointCount**);**

**return** centroid**;**

**}**

// Compares finds reference centroid with the current pupil centroid to determine a direction. Also checks to see how many consecutive frames a particular direction is outputed.

void generateCursorCommand**(**point centroid**)**

**{**

int xdist **=** centroid**.**x **-** p**.**refCentroid**.**x**;**

int ydist **=** centroid**.**y **-** p**.**refCentroid**.**y**;**

int i**;**

**if** **(**procResult **!=** isBlink**)**

**{**

**if** **(**abs**(**xdist**)** **>** abs**(**ydist**))**

**{**

**if** **(**xdist **>** 0 **&&** xdist **>** p**.**minxChangeL**)**

**{**

procResult **=** isLeft**;**

**}**

**else** **if** **(**xdist **<** 0 **&&** abs**(**xdist**)** **>** p**.**minxChangeR**)**

**{**

procResult **=** isRight**;**

**}**

**else**

**{**

procResult **=** isMiddle**;**

**}**

**}**

**else**

**{**

**if** **(**ydist **>** 0 **&&** ydist **>** p**.**minyChangeD**)**

**{**

procResult **=** isDown**;**

**}**

**else** **if** **(**ydist **<** 0 **&&** abs**(**ydist**)** **>** p**.**minyChangeU**)**

**{**

procResult **=** isUp**;**

**}**

**else**

**{**

procResult **=** isMiddle**;**

**}**

**}**

**}**

**if** **(**procResult **==** isBlink**)**

**{**

numBlinks**++;**

**}**

**else**

**{**

numBlinks **=** 0**;**

**}**

**if** **(**prevResultType **==** procResult**)**

**{**

consecDirFrame**++;**

**}**

**else**

**{**

consecDirFrame **=** 0**;**

**}**

// if consecutive frames are in the direction, generate cursor command. Note procResult = isBlink will be used for clicking

**if** **(**consecDirFrame **==** p**.**maxNumFrames**)**

**{**

consecDirFrame **=** 0**;**

cursorCommand **=** procResult**;**

**}**

#ifdef DEBUG\_OUTPUT

printf**(**"The output direction for this frame is: %d\n"**,** procResult**);**

printf**(**"The cursor command is: %d\n"**,** cursorCommand**);**

printf**(**"\n"**);**

#endif

prevResultType **=** procResult**;**

#ifdef MOVE\_CURSOR

INPUT Input**=** **{**0**};**

printf**(**"Proc result: %u\n"**,** procResult**);**

**if(**cursorCommand **!=** isBlink**)**

**{**

POINT mypoint**;**

GetCursorPos**(&**mypoint**);**

SetCursorPos**(**mypoint**.**x **+** cursorSpeed**\***dirArray**[**cursorCommand**].**x**,** mypoint**.**y **+** cursorSpeed**\***dirArray**[**cursorCommand**].**y**);**

**}**

**else**

**{**

**if** **(**numBlinks **>** p**.**minBlinks**)**

**{**

// Left mouse button down

Input**.**type **=** INPUT\_MOUSE**;**

Input**.**mi**.**dwFlags **=** MOUSEEVENTF\_LEFTDOWN**;**

SendInput**(**1**,&**Input**,sizeof(**INPUT**));**

// Left mouse button up

**::**ZeroMemory**(&**Input**,sizeof(**INPUT**));**

Input**.**mi**.**dwFlags **=** MOUSEEVENTF\_LEFTUP**;**

SendInput**(**1**,&**Input**,sizeof(**INPUT**));**

// Left mouse button down

**::**ZeroMemory**(&**Input**,sizeof(**INPUT**));**

Input**.**type **=** INPUT\_MOUSE**;**

Input**.**mi**.**dwFlags **=** MOUSEEVENTF\_LEFTDOWN**;**

SendInput**(**1**,&**Input**,sizeof(**INPUT**));**

// Left mouse button up

**::**ZeroMemory**(&**Input**,sizeof(**INPUT**));**

Input**.**mi**.**dwFlags **=** MOUSEEVENTF\_LEFTUP**;**

SendInput**(**1**,&**Input**,sizeof(**INPUT**));**

numBlinks **=** 0**;**

**}**

**}**

#endif

**}**

// Goal: remove aberrations of the region that was found to be the pupil

void removeAberrations**(**int unityIndex**)**

**{**

int i**,**j**;**

double**\*** rowCount **=** **(**double**\*)**malloc**(**p**.**procRegioniSize **\*** **sizeof(**double**));**

double**\*** colCount **=** **(**double**\*)**malloc**(**p**.**procRegionjSize **\*** **sizeof(**double**));;**

int**\*** rowValue **=** **(**int**\*)**malloc**(**p**.**procRegioniSize **\*** **sizeof(**int**));**

int**\*** colValue **=** **(**int**\*)**malloc**(**p**.**procRegionjSize **\*** **sizeof(**int**));**

int rcIndex **[**300**][**30**];**

int crIndex **[**300**][**30**];**

double sumTotal **=** 0**;**

int sum **=** 0**;**

int z **=** 0**;**

int index**;**

int numCol **=** 0**;**

int numRow **=** 0**;**

int rowIndex **=** 0 **;**

double avgRowCount**;**

double stdRowCount**;**

double avgColCount**;**

double stdColCount**;**

removedPointCount **=** 0**;**

// Vertical Scan

sumTotal **=** 0**;**

int k **=** 0**;**

**for** **(**j **=** 0**;** j **<** p**.**procRegionjSize**;** **++**j**)**

**{**

sum **=** 0**;**

numRow **=** 0**;**

**for** **(**i **=**0**;** i **<** p**.**procRegioniSize**;** **++**i**)**

**{**

**if** **(**cRBinary**[**I3D**(**unityIndex**,**i**,**j**)]** **!=** 0**)**

**{**

sum **+=** cRBinary**[**I3D**(**unityIndex**,**i**,**j**)];**

crIndex**[**k**][**numRow**]** **=** i**;**

**++**numRow**;**

**}**

**}**

**if** **(**sum **!=** 0**)**

**{**

colCount**[**k**]** **=** sum**;**

colValue**[**k**]** **=** j**;**

sumTotal **+=** sum**;**

k**++;**

**}**

**}**

avgColCount **=** sumTotal**/(**k**);**

// Compute the standard deviation of pixel count in vertical scan

sumTotal **=** 0**;**

z **=** removedPointCount**;**

**for** **(**j **=** 0**;** j **<** k**;** **++**j**)**

**{**

sumTotal **+=** **(**colCount**[**j**]-**avgColCount**)\*(**colCount**[**j**]-**avgColCount**);**

**}**

stdColCount **=** sqrt**(**sumTotal**/(**k**));**

**for** **(**j **=** 0**;** j **<** k**;** **++**j**)**

**{**

**if** **(**colCount**[**j**]** **<** avgColCount **-** **(**p**.**numStdVertical**)\***stdColCount**)**

**{**

**for** **(**i **=** 0**;** i **<** colCount**[**j**];** **++**i**)**

**{**

index **=** cRMap**[**I3D**(**unityIndex**,**crIndex**[**j**][**i**],** colValue**[**j**])];**

removedPoints**[**z**].**x **=** cRPointList**[**I2D**(**unityIndex**,**index**)].**x**;**

removedPoints**[**z**].**y **=** cRPointList**[**I2D**(**unityIndex**,**index**)].**y**;**

cRPointList**[**I2D**(**unityIndex**,**index**)].**x **=** 0**;**

cRPointList**[**I2D**(**unityIndex**,**index**)].**y **=** 0**;**

removedPointCount**++;**

z**++;**

**}**

**}**

**}**

// Horizontal Scan

// Compute the average pixel count in a horizontal scan

**for** **(**i **=** 0**;** i **<** p**.**procRegioniSize**;** **++**i**)**

**{**

sum **=** 0**;**

numCol **=** 0**;**

**for** **(**j **=** 0**;** j **<** p**.**procRegionjSize**;** **++**j**)**

**{**

**if** **(**cRBinary**[**I3D**(**unityIndex**,**i**,**j**)]** **!=** 0**)**

**{**

sum **+=** cRBinary**[**I3D**(**unityIndex**,**i**,**j**)];**

rcIndex**[**rowIndex**][**numCol**]** **=** j**;**

**++**numCol**;**

**}**

**}**

**if** **(**sum **!=** 0**)**

**{**

rowCount**[**rowIndex**]** **=** sum**;**

rowValue**[**rowIndex**]** **=** i**;**

sumTotal **+=** sum**;**

rowIndex**++;**

**}**

**}**

avgRowCount **=** sumTotal**/(**rowIndex**);**

// Compute the standard deviation of pixel count in horizontal scan

sumTotal **=** 0**;**

z **=** 0**;**

**for** **(**i **=** 0**;** i **<** rowIndex**;** **++**i**)**

**{**

sumTotal **+=** **(**rowCount**[**i**]-**avgRowCount**)\*(**rowCount**[**i**]-**avgRowCount**);**

**}**

stdRowCount **=** sqrt**(**sumTotal**/(**rowIndex**));**

**for** **(**i **=** 0**;** i **<** rowIndex **;++**i**)**

**{**

**if** **(**rowCount**[**i**]** **<** avgRowCount **-** **(**p**.**numStdHorizontal**)\***stdRowCount**)**

**{**

**for** **(**j **=** 0**;** j **<** rowCount**[**i**];** **++**j**)**

**{**

index **=** cRMap**[**I3D**(**unityIndex**,**rowValue**[**i**],**rcIndex**[**i**][**j**])];**

removedPoints**[**removedPointCount**].**x **=** cRPointList**[**I2D**(**unityIndex**,**index**)].**x**;**

removedPoints**[**removedPointCount**].**y **=** cRPointList**[**I2D**(**unityIndex**,**index**)].**y**;**

cRPointList**[**I2D**(**unityIndex**,**index**)].**x **=** 0**;**

cRPointList**[**I2D**(**unityIndex**,**index**)].**y **=** 0**;**

removedPointCount**++;**

**}**

**}**

**}**

free**(**rowCount**);** free**(**rowValue**);**

**}**

void addProcessingOverlay**(**IplImage **\***img**,** int unityIndex**,** point centroid**)**

**{**

int i**,**j**;**

int channels **=** img**->**nChannels**;**

int step **=** img**->**widthStep**;**

uchar**\*** data **=** **(**uchar **\*)**img**->**imageData**;**

int x**,** y**,** ind**;**

// Draw processing region

#ifdef SHOW\_PROCESSING\_REGION

**for(**i**=**p**.**iStart**;** i **<** p**.**iFinish**;** **++**i**)**

**{**

data**[**i**\***step**+**p**.**jStart**\***channels**+**0**]=**0**;**

data**[**i**\***step**+**p**.**jStart**\***channels**+**1**]=**0**;**

data**[**i**\***step**+**p**.**jStart**\***channels**+**2**]=**255**;**

**}**

**for(**i **=** p**.**iStart**;** i **<** p**.**iFinish**;** **++**i**)**

**{**

data**[**i**\***step**+**p**.**jFinish**\***channels**+**0**]=**0**;**

data**[**i**\***step**+**p**.**jFinish**\***channels**+**1**]=**0**;**

data**[**i**\***step**+**p**.**jFinish**\***channels**+**2**]=**255**;**

**}**

**for(**j **=** p**.**jStart**;** j **<** p**.**jFinish**;** **++**j**)**

**{**

data**[**p**.**iStart**\***step**+**j**\***channels**+**0**]=**0**;**

data**[**p**.**iStart**\***step**+**j**\***channels**+**1**]=**0**;**

data**[**p**.**iStart**\***step**+**j**\***channels**+**2**]=**255**;**

**}**

**for(**j **=** p**.**jStart**;** j **<** p**.**jFinish**;** **++**j**)**

**{**

data**[**p**.**iFinish**\***step**+**j**\***channels**+**0**]=**0**;**

data**[**p**.**iFinish**\***step**+**j**\***channels**+**1**]=**0**;**

data**[**p**.**iFinish**\***step**+**j**\***channels**+**2**]=**255**;**

**}**

#endif

#ifdef SHOW\_CONNECTED\_REGIONS

**for(**i **=** 0**;** i **<** cRCount**;** **++**i**)**

**{**

**for(**j **=** 0**;** j **<** cRSizes0**[**i**];** **++**j**)**

**{**

ind **=** I2D**(**i**,**j**);**

x **=** cRPointList**[**I2D**(**i**,**j**)].**x**;** y **=** cRPointList**[**I2D**(**i**,**j**)].**y**;**

**if** **(**x **!=** 0 **&&** y **!=** 0**)**

**{**

data**[**x**\***step **+** y**\***channels **+** 0**]** **=** 0**;**

data**[**x**\***step **+** y**\***channels **+** 1**]** **=** 0**;**

data**[**x**\***step **+** y**\***channels **+** 2**]** **=** 255**;**

**}**

**}**

**}**

#endif

#ifdef SHOW\_CANDIDATE\_REGIONS

**for(**i **=** 0**;** i **<** candidateRegionCount**;** **++**i**)**

**{**

**for(**j **=** 0**;** j **<** cRSizes0**[**candidateRegionIndices**[**i**]];** **++**j**)**

**{**

ind **=** I2D**(**i**,**j**);**

x **=** cRPointList**[**I2D**(**candidateRegionIndices**[**i**],**j**)].**x**;** y **=** cRPointList**[**I2D**(**candidateRegionIndices**[**i**],**j**)].**y**;**

**if** **(**x **!=** 0 **&&** y **!=** 0**)**

**{**

data**[**x**\***step **+** y**\***channels **+** 0**]** **=** 255**;**

data**[**x**\***step **+** y**\***channels **+** 1**]** **=** 0**;**

data**[**x**\***step **+** y**\***channels **+** 2**]** **=** 0**;**

**}**

**}**

**}**

#endif

// Colors the connected region denoted as the pupil yellow

#ifdef SHOW\_FINAL\_REGION

**if(**unityIndex **!=** **-**1**)**

**{**

**for(**i **=** 0**;** i **<** cRSizes0**[**unityIndex**];** **++**i**)**

**{**

x **=** cRPointList**[**I2D**(**unityIndex**,**i**)].**x**;** y **=** cRPointList**[**I2D**(**unityIndex**,**i**)].**y**;**

data**[**x**\***step **+** y**\***channels **+** 0**]** **=** 0**;**

data**[**x**\***step **+** y**\***channels **+** 1**]** **=** 255**;**

data**[**x**\***step **+** y**\***channels **+** 2**]** **=** 0**;**

**}**

**}**

#endif

#ifdef SHOW\_PIXELS\_REMOVED

**if(**unityIndex **!=** **-**1**)**

**{**

**for** **(**j **=**0 **;** j **<** removedPointCount**;** **++**j**)**

**{**

x **=** removedPoints**[**j**].**x**;** y **=** removedPoints**[**j**].**y**;**

**if** **(**x **!=** 0 **&&** y **!=** 0**)**

**{**

data**[**x**\***step **+** y**\***channels **+** 0**]** **=** 255**;**

data**[**x**\***step **+** y**\***channels **+** 1**]** **=** 255**;**

data**[**x**\***step **+** y**\***channels **+** 2**]** **=** 255**;**

**}**

**}**

**}**

#endif

#ifdef SHOW\_CENTROID\_LOCATION

// Draw vertical line

//for (i = p.iStart; i <= p.iFinish; ++i) // Draw inside processing region

**for** **(**i **=** 0**;** i **<** p**.**imgHeight**;** **++**i**)**

**{**

data**[**i**\***step **+** centroid**.**x**\***channels **+** 0**]** **=** 255**;**

data**[**i**\***step **+** centroid**.**x**\***channels **+** 1**]** **=** 0**;**

data**[**i**\***step **+** centroid**.**x**\***channels **+** 2**]** **=** 255**;**

**}**

// Draw horizontal line

//for (j = p.jStart; j <= p.jFinish; ++j) // Draw inside processing region

**for** **(**j **=** 0**;** j **<** p**.**imgWidth**;** **++**j**)**

**{**

data**[**centroid**.**y**\***step **+** j**\***channels **+** 0**]** **=** 255**;**

data**[**centroid**.**y**\***step **+** j**\***channels **+** 1**]** **=** 0**;**

data**[**centroid**.**y**\***step **+** j**\***channels **+** 2**]** **=** 255**;**

**}**

#endif

**return;**

**}**

void Calibration**(**IplImage **\***img**,** int updateValues**)**

**{**

int width **=** img**->**width**;**

int height **=** img**->**height**;**

int channels **=** img**->**nChannels**;**

int step **=** img**->**widthStep**;**

uchar**\*** data **=** **(**uchar **\*)**img**->**imageData**;**

int i**,**j**;**

int refSize **=** 0**;**

int sumx **=** 0**;**

int sumy **=** 0**;**

**for(**i**=**p**.**iStart**;** i **<** p**.**iFinish**;** **++**i**)**

**{**

**for** **(**j **=** p**.**jStart**;** j **<** p**.**jFinish**;** **++**j**)**

**{**

**if(** **(**unsigned char**)(**RGB2GS**(**i**,**j**))** **<** p**.**initThreshold**)**

**{**

data**[**i**\***step**+**j**\***channels**+**0**]=**0**;**

data**[**i**\***step**+**j**\***channels**+**1**]=**0**;**

data**[**i**\***step**+**j**\***channels**+**2**]=**255**;**

refSize **+=** 1**;**

sumx **+=** j**;**

sumy **+=** i**;**

**}**

**}**

**}**

**if(**refSize **==** 0**)**

refSize **=** 1**;** // Prevent division by zero

**if(**updateValues **==** 1**)**

**{**

p**.**refSize **=** refSize**;**

p**.**refCentroid**.**x **=** **(**int**)(**sumx**/(**p**.**refSize**));**

p**.**refCentroid**.**y **=** **(**int**)(**sumy**/(**p**.**refSize**));**

p**.**lengthRegion **=** 2**\***sqrt**((**p**.**refSize**)/(**3.1415**));**

**}**

#ifdef CAPTURE\_VIDEO

#ifdef SHOW\_CENTROID\_LOCATION

// Draw vertical line

//for (i = p.iStart; i <= p.iFinish; ++i) // Draw inside processing region

**for** **(**i **=** 0**;** i **<** p**.**imgHeight**;** **++**i**)**

**{**

data**[**i**\***step **+** p**.**refCentroid**.**x**\***channels **+** 0**]** **=** 255**;**

data**[**i**\***step **+** p**.**refCentroid**.**x**\***channels **+** 1**]** **=** 0**;**

data**[**i**\***step **+** p**.**refCentroid**.**x**\***channels **+** 2**]** **=** 255**;**

**}**

// Draw horizontal line

//for (j = p.jStart; j <= p.jFinish; ++j) // Draw inside processing region

**for** **(**j **=** 0**;** j **<** p**.**imgWidth**;** **++**j**)**

**{**

data**[**p**.**refCentroid**.**y**\***step **+** j**\***channels **+** 0**]** **=** 255**;**

data**[**p**.**refCentroid**.**y**\***step **+** j**\***channels **+** 1**]** **=** 0**;**

data**[**p**.**refCentroid**.**y**\***step **+** j**\***channels **+** 2**]** **=** 255**;**

**}**

#endif

#endif

#ifdef SHOW\_PROCESSING\_REGION

**for(**i**=**p**.**iStart**;** i **<** p**.**iFinish**;** **++**i**)**

**{**

data**[**i**\***step**+**p**.**jStart**\***channels**+**0**]=**0**;**

data**[**i**\***step**+**p**.**jStart**\***channels**+**1**]=**0**;**

data**[**i**\***step**+**p**.**jStart**\***channels**+**2**]=**255**;**

**}**

**for(**i **=** p**.**iStart**;** i **<** p**.**iFinish**;** **++**i**)**

**{**

data**[**i**\***step**+**p**.**jFinish**\***channels**+**0**]=**0**;**

data**[**i**\***step**+**p**.**jFinish**\***channels**+**1**]=**0**;**

data**[**i**\***step**+**p**.**jFinish**\***channels**+**2**]=**255**;**

**}**

**for(**j **=** p**.**jStart**;** j **<** p**.**jFinish**;** **++**j**)**

**{**

data**[**p**.**iStart**\***step**+**j**\***channels**+**0**]=**0**;**

data**[**p**.**iStart**\***step**+**j**\***channels**+**1**]=**0**;**

data**[**p**.**iStart**\***step**+**j**\***channels**+**2**]=**255**;**

**}**

**for(**j **=** p**.**jStart**;** j **<** p**.**jFinish**;** **++**j**)**

**{**

data**[**p**.**iFinish**\***step**+**j**\***channels**+**0**]=**0**;**

data**[**p**.**iFinish**\***step**+**j**\***channels**+**1**]=**0**;**

data**[**p**.**iFinish**\***step**+**j**\***channels**+**2**]=**255**;**

**}**

#endif

**}**

//---------------------------------------------------------------------------

// Name: gazeTracker.cpp

// Author: Nick Bertrand, Armeen Taeb

//---------------------------------------------------------------------------

#include "stdafx.h"

#include <stdio.h>

#include <highgui.h>

#include <time.h>

#include <conio.h>

#include <Windows.h>

#include "eyeFeatureExtraction.h"

#include "eFEParam.h"

// Initialize program parameters

// \* Indicates value is set based on other parameters

// (computation done in storageInit()

param p **=** **{**

"Nick\_Capstone.avi"**,**// inFile Nick

//"Nick\_Far.avi", // inFile Armeen

"out.avi"**,** // outFile

0**,** // imgWidth

0**,** // imgHeight

170**,** // iStart

270**,** // iFinish

270**,** // jStart

420**,** // jFinish

0.6**,** // aspectMin

2.0**,** // aspectMax

27.91**,** // lengthRegion

1.4**,** // legthMaxRatio

0.6**,** // lengthMinRatio

516**,** // refSize

0.6**,** // refSizeMinRatio

1.8**,** // refSizeMaxRatio

0**,** // refSizeMin\*

0**,** // refSizeMax\*

0**,** // procRegioniSize\*

0**,** // procRegionjSize\*

0**,** // totalPixels (procRegioniSize \* procRegionjSize \* MAX\_TOTAL\_REGIONS)

**{**352**,**214**},** // refCentroid

45**,** // initThreshold

0.7**,** // numStdVertical

1.5**,** // numStdHorizontal

8**,** // minxChangeL

8**,** // minxChangeR

14**,** // minyChangeU

5**,** // minyChangeD

100**,** // maxcentroidChange

5**,** // maxNumFrames

10**,** // minBlinks

4**,** // maxAdaptations

**{**8**,**4**,**2**,**1**},** // magThreshChange

0**,** // nFrames

0 // startFrame

**};**

point dirArray**[**5**]** **=** **{{**0**,**0**},{**1**,**0**},{-**1**,**0**},{**0**,**1**},{**0**,-**1**}};**

point**\*** cRPointList**;** // cRPointList[I2D(k,i)] = ith point of region k

point**\*** removedPoints**;** // Keeps track of points removed in removeAberrations

int removedPointCount**;** // Keeps track of number of points removed in removeAberrations

unsigned char**\*** cRBinary**;** // cRBinary[I3D(k,x,y)] = 1 if (x,y) is in region k

int**\*** cRMap**;** // cRMap[I3D(k,x,y)] = index of point (x,y) in region k in crPointList

int**\*** cRSizes0**;** // Region sizes before clean up

int**\*** cRSizes1**;** // Region sizes after clean up

unsigned char**\*** processedPixels**;** // Used in flood fill algorithm to keep track of already filled pixels

unsigned char**\*** gSImg**;** // Greyscale version of entire image gSImg[2DFULL(x,y)]

int**\*** candidateRegionIndices**;** // Stores index of regions that meet aspect ratio test

int candidateRegionCount**;** // Number of regions that meet aspect ratio test

double**\*** cRAspectRatio**;** // Array storing aspect ratios

int cRCount**;** // Total number of connected regions meeting size requirement

int maxRegionSize **=** 0**;** // Max region size found in getConnectedRegions

int doCalibration **=** 0**;** // Stores pupil size and centroid as reference if set to 1

point centroid**;**

int consecDirFrame**;** // Counts number of consecutive frames user is looking in particular direction

enum resultType procResult**;** // Stores processing result (see definition)

int cursorCommand **=** 2**;** // Stores the cursor command (related to procResult). Originally assume that user is looking in the middle

int prevResultType**;** // stores the processing result of the previous frame. Used to assure consecutive number of frames in particular direction.

double maxLengthConnected**;** // Maximum length of the connected region allowed to pass as the pupil

double minLengthConnected**;** // Minimum length of the connected region allowed to pass as the pupil

int cursorSpeed **=** 5**;**

point prevCentroid**;** // Keeps track of previous centroid

int numBlinks **=** 0**;**

void computeParameters**(**int width**,** int height**)**

**{**

p**.**imgWidth **=** width**;**

p**.**imgHeight **=** height**;**

p**.**refSizeMin **=** p**.**refSize **\*** p**.**refSizeMinRatio**;**

p**.**refSizeMax **=** p**.**refSize **\*** p**.**refSizeMaxRatio**;**

p**.**procRegioniSize **=** p**.**iFinish **-** p**.**iStart **+** 1**;**

p**.**procRegionjSize **=** p**.**jFinish **-** p**.**jStart **+** 1**;**

p**.**totalPixels **=** MAX\_TOTAL\_REGIONS **\*** p**.**procRegioniSize **\*** p**.**procRegionjSize**;**

**}**

void storageInit**()**

**{**

cRPointList **=** **(**point**\*)**malloc**(**p**.**totalPixels **\*** **sizeof(**point**));**

removedPoints **=** **(**point**\*)**malloc**(**p**.**totalPixels **\*** **sizeof(**point**));**

cRBinary **=** **(**unsigned char**\*)**malloc**(**p**.**totalPixels **\*** **sizeof(**unsigned char**));**

cRMap **=** **(**int**\*)**malloc**(**p**.**totalPixels **\*** **sizeof(**int**));**

cRSizes0 **=** **(**int**\*)**malloc**(**MAX\_TOTAL\_REGIONS **\*** **sizeof(**int**));**

cRSizes1 **=** **(**int**\*)**malloc**(**MAX\_TOTAL\_REGIONS **\*** **sizeof(**int**));**

cRAspectRatio **=** **(**double**\*)**malloc**(**MAX\_TOTAL\_REGIONS **\*** **sizeof(**double**));**

candidateRegionIndices **=** **(**int**\*)**malloc**(**MAX\_TOTAL\_REGIONS **\*** **sizeof(**int**));**

processedPixels **=** **(**unsigned char**\*)** malloc**(** **(**p**.**imgWidth **)** **\*** **(**p**.**imgHeight**)** **\*** **sizeof(**unsigned char**)** **);**

gSImg **=** **(**unsigned char**\*)** malloc**(** **(**p**.**imgWidth**)** **\*** **(**p**.**imgHeight**)** **\*** **sizeof(**unsigned char**)** **);**

**if(** cRPointList **==** 0 **||** cRBinary **==** 0 **||** cRMap **==** 0 **||** cRSizes0 **==** 0 **||** cRSizes1 **==** 0 **||** processedPixels **==** 0 **||** gSImg **==** 0 **||** removedPoints **==** 0 **||** cRAspectRatio **==** 0 **||** candidateRegionIndices **==** 0**)**

printf**(**"Error allocating memory\n"**);**

**return;**

**}**

void storageDestroy**()**

**{**

free**(**cRPointList**);**

free**(**removedPoints**);**

free**(**cRBinary**);**

free**(**cRMap**);**

free**(**cRSizes0**);**

free**(**cRSizes1**);**

free**(**cRAspectRatio**);**

free**(**candidateRegionIndices**);**

free**(**processedPixels**);**

free**(**gSImg**);**

**return;**

**}**

#ifdef DEBUG\_MAIN

int \_tmain**(**int argc**,** \_TCHAR**\*** argv**[])**

**{**

int i**;**

point pt **=** **{**1**,**2**};**

pointStackElement**\*** stackHead **=** **(**pointStackElement**\*)**malloc**(sizeof(**pointStackElement**));**

stackHead**->**nextElement **=** 0**;**

stackHead**->**elementData **=** pt**;**

**for(**i **=** 0**;** i **<** 5**;** i**++)**

**{**

**++(**pt**.**x**);** **++(**pt**.**y**);**

pointStackPush**(&**stackHead**,** pt**);**

**}**

pointStackPrint**(**stackHead**);**

printf**(**"Pop top two\n"**);**

pointStackPop**(&**stackHead**);**pointStackPop**(&**stackHead**);**

pointStackPrint**(**stackHead**);**

printf**(**"Pop remaining\n"**);**

**while(**stackHead **!=** 0**)**

pointStackPop**(&**stackHead**);**

pointStackPrint**(**stackHead**);**

printf**(**"Allocating storage\n"**);**

\_getch**();**

storageInit**();**

int loc **=** I3D**(** **(**p**.**maxTotalRegions**-**1**),** **(**p**.**iFinish**-**p**.**iStart**),** **(**p**.**jFinish**-**p**.**jStart**)** **);**

cRBinary**[**loc**]** **=** 25**;**

cRMap**[**loc**]** **=** 25**;**

printf**(**"Deallocating storage\n"**);**

\_getch**();**

storageDestroy**();**

\_getch**();**

**return(**1**);**

**}**

#endif

#ifdef CAPTURE\_CAMERA

int \_tmain**(**int argc**,** \_TCHAR**\*** argv**[])**

**{**

printf**(**"Capturing camera...\n"**);**

int c**;**

int i**;**

int counter**;**

int step **=** 1920**;** // Number of bytes in a row of image data

int channels **=** 3**;** // RGB

int isColor **=** 1**;** //

int fps **=** 30**;** // Frames per second

int depth **=** 8**;** // 8 bits per channel per pixel

uchar**\*** data**;** // Type cast for image data

int cameraIndex **=** 0**;**

int updateValues **=** 1**;**

**if(**argc **>** 1**)**

**{**

cameraIndex **=** atoi**((**char**\*)**argv**[**1**]);**

printf**(**"Using camera %u\n"**,** cameraIndex**);**

**}**

time\_t t\_start**,**t\_end**;**

double sec**,** fps\_measure**;**

// Capture from video device #1

CvCapture**\*** capture **=** cvCaptureFromCAM**(**cameraIndex**);**

**if(**capture **==** 0**)**

**{**

printf**(**"Couldn't open camera.\n"**);**

\_getch**();**

**return** 1**;**

**}**

// Modify capture resolution

cvSetCaptureProperty**(** capture**,** CV\_CAP\_PROP\_FRAME\_WIDTH**,** 640**);**

cvSetCaptureProperty**(** capture**,** CV\_CAP\_PROP\_FRAME\_HEIGHT**,** 480**);**

#ifdef DISPLAY\_OUTPUT

// Create a window to display the images

cvNamedWindow**(**"mainWin"**,** CV\_WINDOW\_AUTOSIZE**);**

// Position the window

cvMoveWindow**(**"mainWin"**,** 5**,** 5**);**

#endif

#ifdef RECORD\_OUTPUT

// Setup output

CvVideoWriter **\***writer **=** cvCreateVideoWriter**(**p**.**outFile**,** CV\_FOURCC**(**'P'**,**'I'**,**'M'**,**'1'**),** fps**,** cvSize**(**640**,**480**),**isColor**);**

#endif

int frameW **=** cvGetCaptureProperty**(**capture**,** CV\_CAP\_PROP\_FRAME\_WIDTH**);**

int frameH **=** cvGetCaptureProperty**(**capture**,** CV\_CAP\_PROP\_FRAME\_HEIGHT**);**

// Allocate memory for an image

IplImage **\***img**;**

// OpenCV reference says we shouldn't modify the output of cvQueryFrame

// so this is used to store a copy.

IplImage **\***dst **=** cvCreateImage**(**cvSize**(**frameW**,** frameH**),** depth**,** channels**);**

// Record FPS in camerea mode

time**(&**t\_start**);**

counter **=** 0**;**

computeParameters**(**frameW**,** frameH**);**

storageInit**();**

**while(**1**)**

**{**

img **=** cvQueryFrame**(**capture**);**

cvCopy**(** img**,** dst**,** **NULL);**

Calibration**(**dst**,** updateValues**);**

#ifdef DISPLAY\_OUTPUT

// Show the image in the window

cvShowImage**(**"mainWin"**,** dst **);**

#endif

// Calculate FPS and output

time**(&**t\_end**);**

**++**counter**;**

double sec **=** difftime**(**t\_end**,** t\_start**);**

fps\_measure **=** counter**/**sec**;**

printf**(**"FPS: %.2lf, T: %u, S: %.2lf, C: (%u,%u)\n"**,**fps\_measure**,**p**.**initThreshold**,** p**.**refSize**);**

c **=** cvWaitKey**(**30**);**

**if(**c **==** 'g'**)**

**break;**

**switch(**c**)**

**{**

**case** 'r'**:**

p**.**initThreshold**++;**

**break;**

**case** 'f'**:**

p**.**initThreshold**--;**

**break;**

**case** 'a'**:**

p**.**jStart **-=** 20**;**

**break;**

**case** 'A'**:**

p**.**jStart **+=** 20**;**

**break;**

**case** 's'**:**

p**.**iFinish **+=** 20**;**

**break;**

**case** 'S'**:**

p**.**iFinish **-=** 20**;**

**break;**

**case** 'd'**:**

p**.**jFinish **+=** 20**;**

**break;**

**case** 'D'**:**

p**.**jFinish **-=** 20**;**

**break;**

**case** 'w'**:**

p**.**iStart **-=** 20**;**

**break;**

**case** 'W'**:**

p**.**iStart **+=** 20**;**

**break;**

**}**

**}**

time**(&**t\_start**);**

counter **=** 0**;**

**while(**1**)**

**{**

// Retrieve the captured frame

img **=** cvQueryFrame**(**capture**);**

cvCopy**(** img**,** dst**,** **NULL);**

// In this version, getConnectedRegions thresholds and modifies dst

processFrame**(**dst**);**

#ifdef DISPLAY\_OUTPUT

// Show the image in the window

cvShowImage**(**"mainWin"**,** dst **);**

#endif

// Calculate FPS and output

time**(&**t\_end**);**

**++**counter**;**

sec **=** difftime**(**t\_end**,** t\_start**);**

fps\_measure **=** counter**/**sec**;**

printf**(**"FPS: %.2lf, T: %u, S: %.2lf, C: (%u,%u)\n"**,**fps\_measure**,**p**.**initThreshold**,** p**.**refSize**,** p**.**refCentroid**.**x**,** p**.**refCentroid**.**y**);**

printf**(**"MinL: %u, MinR: %u, MinD: %u, MinU: %u\n"**,** p**.**minxChangeL**,** p**.**minxChangeR**,** p**.**minyChangeD**,** p**.**minyChangeU**);**

printf**(**"Speed: %u\n"**,** cursorSpeed**);**

// Wait 10 ms for a key to be pressed

c **=** cvWaitKey**(**10**);**

// escape key terminates program

**if(**c **==** 27**)**

**break;**

**switch(**c**)**

**{**

**case** 'r'**:**

p**.**initThreshold**++;**

**break;**

**case** 'f'**:**

p**.**initThreshold**--;**

**break;**

**case** 't'**:**

p**.**refSize **+=** 20**;**

**break;**

**case** 'g'**:**

p**.**refSize **-=** 20**;**

**break;**

**case** 'c'**:**

doCalibration **=** 1**;**

**break;**

**case** 'a'**:**

p**.**jStart **-=** 20**;**

**break;**

**case** 'A'**:**

p**.**jStart **+=** 20**;**

**break;**

**case** 's'**:**

p**.**iFinish **+=** 20**;**

**break;**

**case** 'S'**:**

p**.**iFinish **-=** 20**;**

**break;**

**case** 'd'**:**

p**.**jFinish **+=** 20**;**

**break;**

**case** 'D'**:**

p**.**jFinish **-=** 20**;**

**break;**

**case** 'w'**:**

p**.**iStart **-=** 20**;**

**break;**

**case** 'W'**:**

p**.**iStart **+=** 20**;**

**break;**

**case** 'j'**:**

p**.**minxChangeL **+=** 1**;**

**break;**

**case** 'J'**:**

p**.**minxChangeL **-=** 1**;**

**break;**

**case** 'k'**:**

p**.**minyChangeD **+=** 1**;**

**break;**

**case** 'K'**:**

p**.**minyChangeD **-=** 1**;**

**break;**

**case** 'i'**:**

p**.**minyChangeU **+=** 1**;**

**break;**

**case** 'I'**:**

p**.**minyChangeU **-=** 1**;**

**break;**

**case** 'l'**:**

p**.**minxChangeR **+=** 1**;**

**break;**

**case** 'L'**:**

p**.**minxChangeR **-=** 1**;**

**break;**

**case** 'y'**:**

cursorSpeed**++;**

**break;**

**case** 'h'**:**

cursorSpeed**--;**

**break;**

**}**

**if(**c **!=** **-**1**)**

**{**

storageDestroy**();**

computeParameters**(**p**.**imgWidth**,** p**.**imgHeight**);**

storageInit**();**

**}**

#ifdef RECORD\_OUTPUT

// Output to video file

cvWriteFrame**(**writer**,**dst**);**

#endif

**}**

#ifdef RECORD\_OUTPUT

// Release video writer

cvReleaseVideoWriter**(&**writer**);**

#endif

// Release capture

cvReleaseCapture**(&**capture**);**

storageDestroy**();**

**return** 0**;**

**}**

#endif

#ifdef CAPTURE\_VIDEO

int \_tmain**(**int argc**,** \_TCHAR**\*** argv**[])**

**{**

printf**(**"Capturing video...\n"**);**

int c**;**

int i**;**

int step **=** 1920**;** // Number of bytes in a row of image data

int channels **=** 3**;** // RGB

int isColor **=** 1**;** //

int fps **=** 30**;** // Frames per second

int depth **=** 8**;** // 8 bits per channel per pixel

int stepVideoOn **=** 1**;**

int stepVideoCount **=** 0**;**

int threshold **=** p**.**initThreshold**;**

#ifdef GET\_PARAMETERS\_FROM\_PYTHON

// Memory mapping

HANDLE hMapFile**;**

LPCTSTR pBuf**;**

TCHAR buffName**[]** **=** TEXT**(**"Local\\GazeTrackerFileMapping"**);**

hMapFile **=** OpenFileMapping**(**

FILE\_MAP\_ALL\_ACCESS**,** // read/write access

FALSE**,** // do not inherit the name

buffName**);**

**if** **(**hMapFile **==** **NULL)**

**{**

\_tprintf**(**TEXT**(**"Could not create file mapping object (%d). Please ensure that the Python GUI is running first.\n"**),**

GetLastError**());**

\_getch**();**

**return(**1**);**

**}**

pBuf **=** **(**LPTSTR**)** MapViewOfFile**(**hMapFile**,** // handle to map object

FILE\_MAP\_ALL\_ACCESS**,** // read/write permission

0**,**

0**,**

256**);**

**if** **(**pBuf **==** **NULL)**

**{**

\_tprintf**(**TEXT**(**"Could not create file mapping object (%d). Please ensure that the Python GUI is running first.\n"**),** GetLastError**());**

CloseHandle**(**hMapFile**);**

\_getch**();**

**return(**1**);**

**}**

int**\*** fileParam **=** **(**int**\*)**pBuf**;**

**\***fileParam **=** p**.**initThreshold**;**

#endif

// End memory mapping

// Capture video from file

CvCapture**\*** capture **=** cvCaptureFromAVI**(**p**.**inFile**);**

int numFrames **=** **(**int**)** cvGetCaptureProperty**(**capture**,** CV\_CAP\_PROP\_FRAME\_COUNT**);**

#ifdef DISPLAY\_OUTPUT

// Create a window to display the images

cvNamedWindow**(**"mainWin"**,** CV\_WINDOW\_AUTOSIZE**);**

// Position the window

cvMoveWindow**(**"mainWin"**,** 5**,** 5**);**

#endif

#ifdef RECORD\_OUTPUT

// Setup output

CvVideoWriter **\***writer **=** cvCreateVideoWriter**(**p**.**outFile**,** CV\_FOURCC**(**'P'**,**'I'**,**'M'**,**'1'**),** fps**,** cvSize**(**640**,**480**),**isColor**);**

#endif

int frameW **=** **(**int**)**cvGetCaptureProperty**(**capture**,** CV\_CAP\_PROP\_FRAME\_WIDTH**);**

int frameH **=** **(**int**)**cvGetCaptureProperty**(**capture**,** CV\_CAP\_PROP\_FRAME\_HEIGHT**);**

// Allocate memory for an image

IplImage **\***img**;**

IplImage**\*\*** dst **=** **(**IplImage**\*\*)**malloc**(**numFrames **\*** **sizeof(**IplImage**\*));**

IplImage**\*** tempImg **=** cvCreateImage**(**cvSize**(**frameW**,** frameH**),** depth**,** channels**);**

// OpenCV reference says we shouldn't modify the output of cvQueryFrame

// so this is used to store a copy.

printf**(**"Reading %u frames..."**,** numFrames**);**

**for(**i **=** 0**;** i **<** numFrames**;** **++**i**)**

**{**

img **=** cvQueryFrame**(**capture**);**

dst**[**i**]** **=** cvCreateImage**(**cvSize**(**frameW**,** frameH**),** depth**,** channels**);**

cvCopy**(**img**,** dst**[**i**],** **NULL);**

**}**

printf**(**"Done\n"**);**

computeParameters**(**frameW**,** frameH**);**

storageInit**();**

#ifdef CALIBRATION\_ACTIVE

printf**(**"Adjust the boundaries to pinpoint the pupil. Then adjust threshold until pupil region is approximately filled\n"**);**

printf**(**"Press G when finished\n"**);**

int updateValues **=** 1**;**

**while(**1**)**

**{**

cvCopy**(**dst**[**0**],** tempImg**,** **NULL);**

Calibration**(**tempImg**,** updateValues**);**

#ifdef DISPLAY\_OUTPUT

// Show the image in the window

cvShowImage**(**"mainWin"**,** tempImg **);**

c **=** cvWaitKey**(**100**);**

#endif

//c = \_getch();

**if(**c **==** 'g'**)**

**{**

**if(**updateValues **==** 1**)**

updateValues **=** 0**;**

**else**

**break;**

**}**

// escape key terminates program

**switch(**c**)**

**{**

**case** 'r'**:**

p**.**initThreshold**++;**

**break;**

**case** 'f'**:**

p**.**initThreshold**--;**

**break;**

**case** 'a'**:**

p**.**jStart **-=** 10**;**

**break;**

**case** 'A'**:**

p**.**jStart **+=** 10**;**

**break;**

**case** 's'**:**

p**.**iFinish **+=** 10**;**

**break;**

**case** 'S'**:**

p**.**iFinish **-=** 10**;**

**break;**

**case** 'd'**:**

p**.**jFinish **+=** 10**;**

**break;**

**case** 'D'**:**

p**.**jFinish **-=** 10**;**

**break;**

**case** 'w'**:**

p**.**iStart **-=** 10**;**

**break;**

**case** 'W'**:**

p**.**iStart **+=** 10**;**

**break;**

**case** 27**:**

exit**(**1**);**

**break;**

**}**

printf**(**"T: %u, S: %.2lf, C: (%u,%u)\n"**,**p**.**initThreshold**,** p**.**refSize**,** p**.**refCentroid**.**x**,** p**.**refCentroid**.**y**);**

printf**(**"MinL: %u, MinR: %u, MinD: %u, MinU: %u\n"**,** p**.**minxChangeL**,** p**.**minxChangeR**,** p**.**minyChangeD**,** p**.**minyChangeU**);**

prevCentroid **=** p**.**refCentroid**;**

#ifdef GET\_PARAMETERS\_FROM\_PYTHON

p**.**initThreshold **=** fileParam**[**0**];**

p**.**iStart **=** fileParam**[**1**];**

p**.**iFinish **=** fileParam**[**2**];**

p**.**jStart **=** fileParam**[**3**];**

p**.**jFinish **=** fileParam**[**4**];**

#endif

storageDestroy**();**

computeParameters**(**p**.**imgWidth**,** p**.**imgHeight**);**

storageInit**();**

**}**

#ifdef GET\_PARAMETERS\_FROM\_PYTHON

UnmapViewOfFile**(**pBuf**);**

CloseHandle**(**hMapFile**);**

#endif

#endif

clock\_t t\_start **=** clock**();**

**for(**i **=** 0**;** i **<** numFrames**;** **++**i**)**

**{**

processFrame**(**dst**[**i**]);**

#ifdef DISPLAY\_OUTPUT

// Show the image in the window

cvShowImage**(**"mainWin"**,** dst**[**i**]** **);**

cvWaitKey**(**10**);**

#endif

#ifdef VIDEO\_STEP\_THROUGH

**if(**stepVideoOn **==** 1**)**

**{**

**if(**stepVideoCount **==** 0**)**

**{**

printf**(**"Current Frame: %u\n"**,** i**+**1**);**

**while(**1**)**

**{**

c **=** cvWaitKey**(**100**);**

**if(**c **!=** **-**1**)**

**break;**

**}**

**if(**c **==** 't'**)** // Skip 10 frames

stepVideoCount **=** 10**;**

**else** **if(**c **==** 'f'**)** // Skip 50 frames

stepVideoCount **=** 50**;**

**else** **if(**c **==** 'e'**)** // Skip to end

stepVideoOn **=** 0**;**

**}**

**else**

**--**stepVideoCount**;**

**}**

#endif

**}**

printf**(**"Time elapsed: %f\n"**,** **((**double**)**clock**()** **-** t\_start**)** **/** CLOCKS\_PER\_SEC**);**

#ifdef RECORD\_OUTPUT

**for(**i **=** 0**;** i **<** numFrames**;** **++**i**)**

**{**

// Output to video file

cvWriteFrame**(**writer**,**dst**[**i**]);**

**}**

// Release video writer

cvReleaseVideoWriter**(&**writer**);**

#endif

// Release capture and images

cvReleaseCapture**(&**capture**);**

**for(**i **=** 0**;** i **<** numFrames**;** **++**i**)**

cvReleaseImage**(&(**dst**[**i**]));**

cvReleaseImage**(&**tempImg**);**

storageDestroy**();**

#ifdef DISPLAY\_OUTPUT

**while(**1**)**

**{**

c **=** cvWaitKey**(**100**);**

**if(**c **!=** **-**1**)**

**break;**

**}**

#else

\_getch**();**

#endif

**return** 0**;**

**}**

#endif

//---------------------------------------------------------------------------

// Name: pointStack.cpp

// Author: Nick Bertrand

//---------------------------------------------------------------------------

#include "stdafx.h"

#include <stdlib.h>

#include <stdio.h>

#include "pointStack.h"

point pointStackPop**(**pointStackElement**\*\*** head**)**

**{**

**if(\***head **!=** 0**)**

**{**

pointStackElement**\*** tempLink **=** **\***head**;** // Remember old head

point tempData **=** tempLink**->**elementData**;** // Grab data

**\***head **=** tempLink**->**nextElement**;** // Update head

free**(**tempLink**);** // Free up old head

**return(**tempData**);**

**}**

**}**

void pointStackPush**(**pointStackElement**\*\*** head**,** int x**,** int y**)**

**{**

pointStackElement **\*** ele **=** **(**pointStackElement **\*)** malloc**(sizeof(**pointStackElement**));** // Create new head

**if(**ele **==** 0**)**

printf**(**"Error allocating memory\n"**);**

ele**->**elementData**.**x **=** x**;**

ele**->**elementData**.**y **=** y**;**

ele**->**nextElement **=** **\***head**;**

**\***head **=** ele**;** // Update head

**return;**

**}**

void pointStackPrint**(**pointStackElement**\*** head**)**

**{**

**if(**head **==** 0**)**

**{**

printf**(**"Stack Empty\n"**);**

**return;**

**}**

**while(**head **!=** 0**)**

**{**

printf**(**"X:%u Y:%u\n"**,**head**->**elementData**.**x**,** head**->**elementData**.**y**);**

head **=** head**->**nextElement**;**

**}**

**}**

//---------------------------------------------------------------------------

// Name: eFEParam.h

// Author: Nick Bertrand, Armeen Taeb

//---------------------------------------------------------------------------#ifndef eFEParam\_

#define eFEParam\_

//#define DEBUG\_MAIN // Exits directly after compil

//#define DEBUG\_OUTPUT

#define CALIBRATION\_ACTIVE // Allow calibration

//#define CAPTURE\_CAMERA // Capture from camera if defined, else from video file

#define DISPLAY\_OUTPUT // Shows video output in a window

#define CAPTURE\_VIDEO

#define RECORD\_OUTPUT // Records output to p.outFile

#define VIDEO\_STEP\_THROUGH // Requires a key press to advance to next frame in video

#define MOVE\_CURSOR

//#define GET\_PARAMETERS\_FROM\_PYTHON

#define SHOW\_PROCESSING\_REGION // Draw processing region

#define SHOW\_THRESHOLD\_PIXELS // Color thresholded pixels

#define SHOW\_CONNECTED\_REGIONS // Color connected regions

#define SHOW\_CANDIDATE\_REGIONS // Color regions that meet size and aspect ratio requirements

#define SHOW\_FINAL\_REGION // Color final pupil region

#define SHOW\_PIXELS\_REMOVED // Color the the part of the final region that was removed in removeAberration

#define SHOW\_CENTROID\_LOCATION // Draw cross over pupil centroid

#define MAX\_TOTAL\_REGIONS 10 // Total number of connected regions to store

//Macros

#define I2D(K,I) ((K) \* p.procRegioniSize \* p.procRegionjSize + (I))

#define I3D(K,X,Y) ((K) \* p.procRegioniSize \* p.procRegionjSize + (X) \* p.procRegionjSize + (Y))

struct point

**{**

int x**,**y**;**

**};**

struct param

**{**

// Paths for video input/output

char inFile**[**500**];**

char outFile**[**500**];**

int imgWidth**;**

int imgHeight**;**

// Processing region

int iStart**;**

int iFinish**;**

int jStart**;**

int jFinish**;**

// Acceptable aspect ratio

double aspectMin**;**

double aspectMax**;**

// Acceptable length of the connected region (Note we assume the same constraints for x and y direction).

double lengthRegion**;**

double lengthMaxRatio**;**

double lengthMinRatio**;**

// Acceptable pupil dimensions

double refSize**;**

double refSizeMinRatio**;**

double refSizeMaxRatio**;**

double refSizeMin**;**

double refSizeMax**;**

int procRegioniSize**;**

int procRegionjSize**;**

int totalPixels**;**

// Reference centroid and initial threshold

point refCentroid**;**

int initThreshold**;**

// Number of standard deviations away from the mean in removeAberrations for vertical and horizontal scan.

double numStdVertical**;**

double numStdHorizontal**;**

// How much the eye moves before moving the cursor

int minxChangeL**;**

int minxChangeR**;**

int minyChangeU**;**

int minyChangeD**;**

double maxcentroidChange**;** // The tolerance for change in centroid of candidate pupil from previously found pupil to consider the candidate region as pupil.

// Number of frames with a particular direction before moving the cursor in that direction

int maxNumFrames**;**

int minBlinks**;**

// Maximum number of threshold adaptations and the change in magnitude of threshold:

int maxAdaptations**;**

int magThreshChange**[**4**];**

// Define which video frames are processed

int nFrames**;**

int startFrame**;**

**};**

// Results from processing

enum resultType

**{**

isMiddle**,**

isRight**,**

isLeft**,**

isDown**,**

isUp**,**

isBlink**,**

isPupil

**};**

#endif

//---------------------------------------------------------------------------

// Name: eyeFeatureExtraction.h

// Author: Nick Bertrand, Armeen Taeb

//---------------------------------------------------------------------------

#ifndef eyeFeatureExtraction\_

#define eyeFeatureExtraction\_

#include "eFEParam.h"

// Current function: modifies img by indicated threshold values

// Goal: Create cR, cRSizes, cRCount

void processFrame**(**IplImage **\***img**);**

// Goal: Has the user select the boundaries for calibration.

void Calibration**(**IplImage **\***img**,** int updateValues**);**

// Fill 2D array with greyscale values

void fillGreyScale**(**IplImage **\***img**);**

// Fill connected region structures

void getConnectedRegions**(**int threshold**);**

// Compute Aspect Ratio and find selected region that has ratio equaling to 1

int findUnityRatio**();**

//remove abberations from the image

void removeAberrations**(**int unityIndex**);**

// Draw lines to indicate the pupil centroid, and color the chosen region yellow

void addProcessingOverlay**(**IplImage **\***img**,** int unityIndex**,** point centroid**);**

// Finds the centroid of the region selected as the pupil

point computeCentroid**(**int unityIndex**);**

// Compares the reference centroid and the computed centroid to determine direction

void generateCursorCommand**(**point centroid**);**

#endif

## Calibration GUI

# -----------------------------------------------------------------------

# Name: calibrationGUI.py

# Authors: Nick Bertrand, Armeen Taeb

# -----------------------------------------------------------------------

**from** Tkinter **import** **\***

**import** tkMessageBox

**import** struct

**import** mmap

**import** os

**import** sys

**class** **App:**

# Status bar text

step1Text **=** "Adjust the threshold until a red circle covers the pupil. Then adjust the boundaries to enclose only the pupil. Once this is done, click next!"

step2Text **=** "Adjust the boundaries to include the maximum range of motion of your eye."

step3Text **=** "For this part of the calibration, center yourself in front of the monitor. Then look at each black circle without moving your head and press space. Press space to start."

widgetWidth **=** 400

**def** \_\_init\_\_**(**self**,** master**):**

# Initialize Frame

frame **=** Frame**(**master**)**

frame**.**grid**()**

self**.**w **=** Canvas**(**frame**,** width **=** self**.**widgetWidth**,** height **=** 90**)**

self**.**w**.**grid**(**row**=**10**,** columnspan**=**3**)**

self**.**statusLabel **=** Label**(**

frame**,**

text **=** self**.**step1Text**,**

font **=** **(**"Helvectica"**,** "12"**),**

wraplength **=** self**.**widgetWidth

**)**

self**.**statusLabel**.**grid**(**row **=** 10**,** sticky **=** N**,** columnspan**=**3**,** pady **=** 10**)**

# Initialize Scales

self**.**scaleThreshold **=** Scale**(**

frame**,**

label**=**"Threshold"**,**

orient**=**HORIZONTAL**,**

from\_**=**1**,**

to**=**255**,**

command**=**self**.**writeParams**,**

length **=** self**.**widgetWidth**,**

relief**=**FLAT**,**

cursor**=**'gumby'

**)**

self**.**scaleThreshold**.**set**(**45**)**

self**.**scaleThreshold**.**grid**(**row**=**0**,** columnspan**=**3**)**

self**.**scaleiMin **=** Scale**(**

frame**,**

label**=**"iMin"**,**

orient**=**HORIZONTAL**,**

from\_**=**0**,**

to**=**480**,**

command**=**self**.**writeParams**,**

length **=** self**.**widgetWidth**,**

relief**=**FLAT**,**

cursor**=**'umbrella'

**)**

self**.**scaleiMin**.**set**(**170**)**

self**.**scaleiMin**.**grid**(**row**=**1**,** columnspan**=**3**)**

self**.**scaleiMax **=** Scale**(**

frame**,**

label**=**"iMax"**,**

orient**=**HORIZONTAL**,**

from\_**=**0**,**

to**=**480**,**

command**=**self**.**writeParams**,**

length **=** self**.**widgetWidth**,**

relief**=**FLAT**,**

cursor**=**'coffee\_mug'

**)**

self**.**scaleiMax**.**set**(**270**)**

self**.**scaleiMax**.**grid**(**row**=**2**,** columnspan**=**3**)**

self**.**scalejMin **=** Scale**(**

frame**,**

label**=**"jMin"**,**

orient**=**HORIZONTAL**,**

from\_**=**0**,**

to**=**640**,**

command**=**self**.**writeParams**,**

length **=** self**.**widgetWidth**,**

relief**=**FLAT**,**

cursor**=**'boat'

**)**

self**.**scalejMin**.**set**(**270**)**

self**.**scalejMin**.**grid**(**row**=**3**,** columnspan**=**3**)**

self**.**scalejMax **=** Scale**(**

frame**,**

label**=**"jMax"**,**

orient**=**HORIZONTAL**,**

from\_**=**0**,**

to**=**640**,**

command**=**self**.**writeParams**,**

length **=** self**.**widgetWidth**,**

relief**=**FLAT**,**

cursor**=**'pirate'

**)**

self**.**scalejMax**.**set**(**420**)**

self**.**scalejMax**.**grid**(**row**=**4**,** columnspan**=**3**)**

# Initialize Buttons

self**.**button **=** Button**(**frame**,** text **=** "QUIT"**,** command **=** frame**.**quit**)**

self**.**button**.**grid**(**row**=**6**,**pady**=**2**,**column**=**0**)**

self**.**button **=** Button**(**frame**,** text **=** "RESTART"**,** command **=** self**.**restartCalibration0**)**

self**.**button**.**grid**(**row**=**6**,**pady**=**2**,**column**=**1**)**

self**.**nextbutton **=** Button**(**frame**,** text **=** "NEXT"**,** command **=** self**.**startFirstStage**)**

self**.**nextbutton**.**grid**(**row**=**6**,** pady**=**2**,** column**=**2**)**

# Write threshold value from scale to file map

**def** writeParams**(**self**,** val**):**

map**.**seek**(**0**)**

map**.**write**(**struct**.**pack**(**"i"**,** self**.**scaleThreshold**.**get**()))**

map**.**write**(**struct**.**pack**(**"i"**,** self**.**scaleiMin**.**get**()))**

map**.**write**(**struct**.**pack**(**"i"**,** self**.**scaleiMax**.**get**()))**

map**.**write**(**struct**.**pack**(**"i"**,** self**.**scalejMin**.**get**()))**

map**.**write**(**struct**.**pack**(**"i"**,** self**.**scalejMax**.**get**()))**

# Transition from stage zero to stage one

**def** startFirstStage**(**self**):**

self**.**stage **=** 1

self**.**statusLabel**[**'text'**]** **=** self**.**step2Text

self**.**nextbutton**[**'command'**]** **=** self**.**startSecondStage

self**.**scaleThreshold**[**'state'**]** **=** DISABLED

# Transition from stage one to stage two

**def** startSecondStage**(**self**):**

# Create new window

self**.**top **=** Toplevel**()** # toplevel = window

self**.**top**.**overrideredirect**(**1**)** # Override Redirect Flag -- Use for fullscreen

self**.**top**.**state**(**"zoomed"**)** # Fullscreen

self**.**stage **=** 2**;** #

# Get screen dimensions

width **=** self**.**top**.**winfo\_screenwidth**()**

height **=** self**.**top**.**winfo\_screenheight**()**

self**.**top**.**focus\_set**()** # Set focus to new window

# Create large blue canvas

self**.**w **=** Canvas**(**self**.**top**,**width**=**width**,** height **=** height**,** background**=**"LightSkyBlue4"**,** highlightbackground**=**"LightSkyBlue4"**)**

self**.**w**.**grid**(**row**=**0**,**column**=**0**)**

# Create buttons

self**.**top**.**button **=** Button**(**self**.**top**,** text **=** "PREVIOUS"**,** command **=** self**.**restartDirectionalCalibration**,** bd**=**3**,** font**=(**"Helvectica"**,** "12"**,** "bold"**))**

self**.**top**.**button**.**grid**(**row**=**0**,**sticky **=** SE**,** padx**=**120**,** pady**=**20**)**

self**.**top**.**button **=** Button**(**self**.**top**,** text **=** "RESTART"**,** command **=** self**.**restartCalibration**,** bd**=**3**,** font**=(**"Helvectica"**,** "12"**,** "bold"**))**

self**.**top**.**button**.**grid**(**row**=**0**,**sticky **=** SE**,** padx**=**20**,** pady**=**20**)**

# Show directions

self**.**directionText **=** Label**(**

self**.**top**,**

width **=** 0**,**

text**=**self**.**step3Text**,**

font**=(**"Helvectica"**,** "24"**,** "bold"**),**

wraplength **=** 600**,**

background**=**"LightSkyBlue4"

**)**

self**.**directionText**.**grid**(**row**=**0**,** column**=**0**)**

# Draw circle

self**.**t **=** Canvas**(**self**.**top**,**width**=**50**,**height**=**50**,** background**=**"LightSkyBlue4"**,** bd**=**0**,** highlightbackground**=**"LightSkyBlue4"**)**

self**.**t**.**create\_oval**(**2**,**2**,**50**,**50**,**fill **=** "black"**)**

self**.**top**.**bind**(**"<KeyRelease-space>"**,** self**.**LookMiddle**)**

# Transition to next step of calibration

**def** LookMiddle**(**self**,**event**):**

self**.**t**.**grid**(**row**=**0**,**column**=**0**)**

self**.**directionText**[**'text'**]** **=** " "

self**.**top**.**bind**(**"<KeyRelease-space>"**,** self**.**LookLeft**)**

self**.**calibDirection **=** 0**;**

**def** LookLeft**(**self**,**event**):**

self**.**t**.**grid**(**row**=**0**,**column**=**0**,**sticky **=** W**)**

self**.**top**.**bind**(**"<KeyRelease-space>"**,** self**.**LookRight**)**

self**.**calibDirection **=** 1**;**

**def** LookRight**(**self**,**event**):**

self**.**t**.**grid**(**row**=**0**,**column**=**0**,**sticky **=** E**)**

self**.**top**.**bind**(**"<KeyRelease-space>"**,** self**.**LookUp**)**

self**.**calibDirection **=** 2**;**

**def** LookUp**(**self**,**event**):**

self**.**t**.**grid**(**row**=**0**,**column**=**0**,**sticky **=** N**)**

self**.**top**.**bind**(**"<KeyRelease-space>"**,** self**.**LookDown**)**

self**.**calibDirection **=** 3**;**

**def** LookDown**(**self**,**event**):**

self**.**t**.**grid**(**row**=**0**,**column**=**0**,**sticky **=** S**)**

self**.**top**.**bind**(**"<KeyRelease-space>"**,** self**.**killWindow**)**

self**.**statusLabel**[**'text'**]** **=** "Calibration complete!"

self**.**calibDirection **=** 4**;**

# Used for 'PREVIOUS' button event

# Restarts directional calibration

**def** restartDirectionalCalibration**(**self**):**

self**.**top**.**bind**(**"<KeyRelease-space>"**,** self**.**LookMiddle**)**

self**.**t**.**grid\_forget**()**

self**.**directionText**[**'text'**]** **=** self**.**step3Text

self**.**directionText**.**grid**()**

# Restart for use from main window

**def** restartCalibration0**(**self**):**

self**.**stage **=** 0

self**.**statusLabel**[**'text'**]** **=** self**.**step1Text

self**.**nextbutton**[**'command'**]** **=** self**.**startFirstStage

self**.**scaleThreshold**[**'state'**]** **=** ACTIVE

# Restart for use from direction calibration window

**def** restartCalibration**(**self**):**

self**.**top**.**destroy**()**

self**.**stage **=** 0

self**.**statusLabel**[**'text'**]** **=** self**.**step1Text

self**.**nextbutton**[**'command'**]** **=** self**.**startFirstStage

self**.**scaleThreshold**[**'state'**]** **=** ACTIVE

# Close direction calibration window

**def** killWindow**(**self**,**event**):**

self**.**top**.**destroy**()**

root **=** Tk**()** # Initialize root

root**.**wm\_title**(**'eye CU - Gaze Tracker'**)** # Set window title

# Set window icon

scriptpath **=** os**.**path**.**dirname**(**sys**.**argv**[**0**])**

scriptpath **+=** '\\eye.ico'

**if** os**.**path**.**exists**(**scriptpath**):**

root**.**iconbitmap**(**default**=**scriptpath**)**

root**.**grid**()**

app **=** App**(**root**)**

map **=** mmap**.**mmap**(**0**,**256**,** tagname**=**'Local\\GazeTrackerFileMapping'**)** # Create file map

root**.**mainloop**()**

map**.**close**()** # Close file map

## Interfaces:

### Video Transfert Between Beagle Bone and Host Computer:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// gazeTracker\_DebugVideo.c – Host computer side of video transfer

// Written By: Nick Bertrand, Armeen Taeb

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <windows.h>

#include <stdio.h>

#include <highgui.h>

#include <stdlib.h>

#include <initguid.h>

#include "lmusbdll.h"

#include "luminary\_guids.h"

#include <strsafe.h>

// Buffer size definitions.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define MAX\_STRING\_LEN 256

#define MAX\_ENTRY\_LEN 256

#define USB\_BUFFER\_LEN 930000

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// The build version number

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define BLDVER "8555"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// The number of bytes we read and write per transaction if in echo mode.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define ECHO\_PACKET\_SIZE 64

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Buffer into which error messages are written.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TCHAR g\_pcErrorString[MAX\_STRING\_LEN];

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// The number of bytes transfered in the last measurement interval.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ULONG g\_ulByteCount = 0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// The total number of packets transfered.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ULONG g\_ulPacketCount = 0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Return a string describing the supplied system error code.

//

// \param dwError is a Windows system error code.

//

// This function returns a pointer to a string describing the error code

// passed in the dwError parameter. If no description string can be found

// the string "Unknown" is returned.

//

// \return Returns a pointer to a string describing the error.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LPTSTR GetSystemErrorString(DWORD dwError)

{

DWORD dwRetcode;

//

// Ask Windows for the error message description.

//

dwRetcode = FormatMessage(FORMAT\_MESSAGE\_FROM\_SYSTEM, "%0", dwError, 0,

g\_pcErrorString, MAX\_STRING\_LEN, NULL);

if(dwRetcode == 0)

{

return((LPTSTR)L"Unknown");

}

else

{

//

// Remove the trailing "\n\r" if present.

//

if(dwRetcode >= 2)

{

if(g\_pcErrorString[dwRetcode - 2] == '\r')

{

g\_pcErrorString[dwRetcode - 2] = '\0';

}

}

return(g\_pcErrorString);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// The main application entry function.

//

// \param None.

//

// \return Set the exit code to 0 of no errors cause the application to end

// or a non-zero value otherwise.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main(int argc, char \*argv[])

{

BOOL bResult;

BOOL bDriverInstalled;

char\* szBuffer;

ULONG ulWritten;

ULONG ulRead;

DWORD dwError;

LMUSB\_HANDLE hUSB;

char\* frame;

char c;

unsigned int i;

IplImage \*img;

HANDLE hMapFile;

LPCTSTR pBuf;

TCHAR szName[]=TEXT("Local\\BeagleVideoTransfer");

hMapFile = CreateFileMapping(

INVALID\_HANDLE\_VALUE,

NULL,

PAGE\_READWRITE,

0,

930000,

szName);

if(hMapFile == NULL)

{

printf("Can't open file map\n");

return 1;

}

pBuf = (LPTSTR)MapViewOfFile(hMapFile,

FILE\_MAP\_ALL\_ACCESS,

0,0,

930000);

if(pBuf == NULL)

{

printf("Could not open file view\n");

return 1;

}

frame = (char\*)pBuf;

cvNamedWindow("mainWin", CV\_WINDOW\_AUTOSIZE);

cvMoveWindow("mainWin", 5,5);

img = cvCreateImage(cvSize(640,480), 8, 3);

szBuffer = (char\*)malloc(sizeof(char)\*USB\_BUFFER\_LEN);

//

// Find our USB device and prepare it for communication.

//

hUSB = InitializeDevice(BULK\_VID, BULK\_PID,

(LPGUID)&(GUID\_DEVINTERFACE\_STELLARIS\_BULK),

&bDriverInstalled);

if(hUSB)

{

szBuffer[0] = 1;

while(1)

{

bResult = WriteUSBPacket(hUSB, szBuffer, 1, &ulWritten);

if(!bResult)

{

//

// We failed to write the data for some reason.

//

dwError = GetLastError();

printf("Error %d (%S) writing to bulk OUT pipe.\n", dwError,

GetSystemErrorString(dwError));

\_getch();

return 1;

}

//

// We wrote data successfully so now read it back.

//

printf("Wrote %d bytes to the device\n",

ulWritten);

dwError = ReadUSBPacket(hUSB, szBuffer, 921600, &ulRead,INFINITE, NULL);

for(i = 0; i < ulRead; ++i)

{

//frame[i] = szBuffer[i];

img->imageData[i] = szBuffer[i];

}

c = cvWaitKey(10);

if(c == 27)

break;

cvShowImage("mainWin", img);

if(dwError != ERROR\_SUCCESS)

{

//

// We failed to read from the device.

//

printf("Error %d (%S) reading from bulk IN pipe.\n", dwError,

GetSystemErrorString(dwError));

}

}

}

else

{

//

// An error was reported while trying to connect to the device.

//

dwError = GetLastError();

printf("\nUnable to initialize the Stellaris Bulk USB Device.\n");

printf("Error code is %d (%S)\n\n", dwError, GetSystemErrorString(dwError));

printf("Please make sure you have a Stellaris USB-enabled evaluation\n");

printf("or development kit running the usb\_dev\_bulk example\n");

printf("application connected to this system via the \"USB OTG\" or\n");

printf("\"USB DEVICE\" connectors. Once the device is connected, run\n");

printf("this application again.\n\n");

printf("\nPress \"Enter\" to exit: ");

fgets(szBuffer, MAX\_STRING\_LEN, stdin);

printf("\n");

return(2);

}

TerminateDevice(hUSB);

UnmapViewOfFile(pBuf);

CloseHandle(hMapFile);

cvReleaseImage(&img);

return(0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// usb\_dev\_bulk.c – USB Bulk transfer on Beagle Bone

// Written By: Nick Bertrand, Armeen Taeb

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "ustdlib.h"

#include "hw\_uart.h"

#include "hw\_types.h"

#include "interrupt.h"

#include "soc\_AM335x.h"

#include "beaglebone.h"

#include "usblib.h"

#include "usb.h"

#include "usb-ids.h"

#include "usbdevice.h"

#include "usbdbulk.h"

#include "uartStdio.h"

#include "usb\_bulk\_structs.h"

#include "hw\_usb.h"

#include "delay.h"

#include "cache.h"

#include "mmu.h"

//! StellarisWare/tools/usb\_bulk\_example.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// The system tick rate expressed both as ticks per second and a millisecond

// period.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define SYSTICKS\_PER\_SECOND 100

#define SYSTICK\_PERIOD\_MS (1000 / SYSTICKS\_PER\_SECOND)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// The global system tick counter.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

volatile unsigned int g\_ulSysTickCount = 0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Variables tracking transmit and receive counts.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

volatile unsigned int g\_ulTxCount = 0;

volatile unsigned int g\_ulRxCount = 0;

#ifdef DEBUG

unsigned int g\_ulUARTRxErrors = 0;

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Debug-related definitions and declarations.

//

// Debug output is available via UART0 if DEBUG is defined during build.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifdef DEBUG

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Map all debug print calls to UARTprintf in debug builds.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define DEBUG\_PRINT UARTprintf

#else

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Compile out all debug print calls in release builds.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define DEBUG\_PRINT while(0) ((int (\*)(char \*, ...))0)

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Flags used to pass commands from interrupt context to the main loop.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define COMMAND\_PACKET\_RECEIVED 0x00000001

#define COMMAND\_STATUS\_UPDATE 0x00000002

volatile unsigned int g\_ulFlags = 0;

char \*g\_pcStatus;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Global flag indicating that a USB configuration has been set.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static volatile tBoolean g\_bUSBConfigured = false;

#define START\_ADDR\_DDR (0x80000000)

#define START\_ADDR\_DEV (0x44000000)

#define START\_ADDR\_OCMC (0x40300000)

#define NUM\_SECTIONS\_DDR (512)

#define NUM\_SECTIONS\_DEV (960)

#define NUM\_SECTIONS\_OCMC (1)

/\* page tables start must be aligned in 16K boundary \*/

#ifdef \_\_TMS470\_\_

#pragma DATA\_ALIGN(pageTable, 16384);

static volatile unsigned int pageTable[4\*1024];

#elif defined(\_\_IAR\_SYSTEMS\_ICC\_\_)

#pragma data\_alignment=16384

static volatile unsigned int pageTable[4\*1024];

#else

static volatile unsigned int pageTable[4\*1024] \_\_attribute\_\_((aligned(16\*1024)));

#endif

static void MMUConfigAndEnable(void);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// The error routine that is called if the driver library encounters an error.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifdef DEBUG

void

\_\_error\_\_(char \*pcFilename, unsigned int ulLine)

{

UARTprintf("Error at line %d of %s\n", ulLine, pcFilename);

while(1)

{

}

}

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Receive new data and echo it back to the host.

//

// \param psDevice points to the instance data for the device whose data is to

// be processed.

// \param pcData points to the newly received data in the USB receive buffer.

// \param ulNumBytes is the number of bytes of data available to be processed.

//

// This function is called whenever we receive a notification that data is

// available from the host. We read the data, byte-by-byte and swap the case

// of any alphabetical characters found then write it back out to be

// transmitted back to the host.

//

// \return Returns the number of bytes of data processed.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static unsigned int

EchoNewDataToHost(tUSBDBulkDevice \*psDevice, unsigned char \*pcData,

unsigned int ulNumBytes)

{

unsigned int ulWriteIndex;

tUSBRingBufObject sTxRing;

static unsigned char currentPixel = 0;

int i,j,k;

int colorIndex;

static int currentOffset = 0;

int colStarts[8] = {0, 91, 182,273,364,455,546,640};

unsigned char IntensityLookup[7][3] = {{255,255,255},

{255,255,0},

{0,255,255},

{0,255,0},

{255,0,255},

{255,0,0},

{0,0,255}};

unsigned char frame[921600];

//

// Get the current buffer information to allow us to write directly to

// the transmit buffer (we already have enough information from the

// parameters to access the receive buffer directly).

//

USBBufferInfoGet(&g\_sTxBuffer, &sTxRing);

//

// Update our receive counter.

//

g\_ulRxCount += ulNumBytes;

ulWriteIndex = sTxRing.ulWriteIndex;

// Set up a new test card frame

for(k = 0; k < 7; ++k)

{

colorIndex = (currentOffset+k) % 7;

for(i = colStarts[k]; i < colStarts[k+1]; ++i)

{

for(j = 0; j < 480; ++j)

{

frame[3\*(640\*j+i) + 2] = IntensityLookup[colorIndex][0];

frame[3\*(640\*j+i) + 1] = IntensityLookup[colorIndex][1];

frame[3\*(640\*j+i) + 0] = IntensityLookup[colorIndex][2];

}

}

}

currentOffset++;

USBBufferFlush(&g\_sTxBuffer);

ulWriteIndex = sTxRing.ulWriteIndex;

for(j=0; j < 921600; ++j)

{

g\_pucUSBTxBuffer[ulWriteIndex] = frame[j];

ulWriteIndex++;

ulWriteIndex = (ulWriteIndex == BULK\_BUFFER\_SIZE) ? 0 : ulWriteIndex;

}

currentPixel++;

USBBufferDataWritten(&g\_sTxBuffer, 921600);

return(921600);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Handles bulk driver notifications related to the transmit channel (data to

// the USB host).

//

// \param pvCBData is the client-supplied callback pointer for this channel.

// \param ulEvent identifies the event we are being notified about.

// \param ulMsgValue is an event-specific value.

// \param pvMsgData is an event-specific pointer.

//

// This function is called by the bulk driver to notify us of any events

// related to operation of the transmit data channel (the IN channel carrying

// data to the USB host).

//

// \return The return value is event-specific.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

unsigned int

TxHandler(void \*pvCBData, unsigned int ulEvent, unsigned int ulMsgValue,

void \*pvMsgData)

{

//

// We are not required to do anything in response to any transmit event

// in this example. All we do is update our transmit counter.

//

if(ulEvent == USB\_EVENT\_TX\_COMPLETE)

{

g\_ulTxCount += ulMsgValue;

}

//

// Dump a debug message.

//

DEBUG\_PRINT("TX complete %d\n", ulMsgValue);

return(0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Handles bulk driver notifications related to the receive channel (data from

// the USB host).

//

// \param pvCBData is the client-supplied callback pointer for this channel.

// \param ulEvent identifies the event we are being notified about.

// \param ulMsgValue is an event-specific value.

// \param pvMsgData is an event-specific pointer.

//

// This function is called by the bulk driver to notify us of any events

// related to operation of the receive data channel (the OUT channel carrying

// data from the USB host).

//

// \return The return value is event-specific.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

unsigned int

RxHandler(void \*pvCBData, unsigned int ulEvent,

unsigned int ulMsgValue, void \*pvMsgData)

{

//

// Which event are we being sent?

//

switch(ulEvent)

{

//

// We are connected to a host and communication is now possible.

//

case USB\_EVENT\_CONNECTED:

{

g\_bUSBConfigured = true;

g\_pcStatus = "Host connected.";

g\_ulFlags |= COMMAND\_STATUS\_UPDATE;

//

// Flush our buffers.

//

USBBufferFlush(&g\_sTxBuffer);

USBBufferFlush(&g\_sRxBuffer);

break;

}

//

// The host has disconnected.

//

case USB\_EVENT\_DISCONNECTED:

{

g\_bUSBConfigured = false;

g\_pcStatus = "Host disconnected.";

g\_ulFlags |= COMMAND\_STATUS\_UPDATE;

break;

}

//

// A new packet has been received.

//

case USB\_EVENT\_RX\_AVAILABLE:

{

tUSBDBulkDevice \*psDevice;

//

// Get a pointer to our instance data from the callback data

// parameter.

//

psDevice = (tUSBDBulkDevice \*)pvCBData;

//

// Read the new packet and echo it back to the host.

//

return(EchoNewDataToHost(psDevice, pvMsgData, ulMsgValue));

}

//

// Ignore SUSPEND and RESUME for now.

//

case USB\_EVENT\_SUSPEND:

case USB\_EVENT\_RESUME:

break;

//

// Ignore all other events and return 0.

//

default:

break;

}

return(0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Sets up the AINTC Interrupt

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static void USB0AINTCConfigure(void)

{

/\* Initializing the ARM Interrupt Controller. \*/

IntAINTCInit();

/\* Registering the Interrupt Service Routine(ISR). \*/

IntRegister(SYS\_INT\_USB0, USB0DeviceIntHandler);

/\* Setting the priority for the system interrupt in AINTC. \*/

IntPrioritySet(SYS\_INT\_USB0, 0, AINTC\_HOSTINT\_ROUTE\_IRQ);

/\* Enabling the system interrupt in AINTC. \*/

IntSystemEnable(SYS\_INT\_USB0);

}

static void USBInterruptEnable(void)

{

/\* Enabling IRQ in CPSR of ARM processor. \*/

IntMasterIRQEnable();

/\* Configuring AINTC to receive UART0 interrupts. \*/

USB0AINTCConfigure();

}

/\*

\*\* Function to setup MMU. This function Maps three regions (1. DDR

\*\* 2. OCMC and 3. Device memory) and enables MMU.

\*/

void MMUConfigAndEnable(void)

{

/\*

\*\* Define DDR memory region of AM335x. DDR can be configured as Normal

\*\* memory with R/W access in user/privileged modes. The cache attributes

\*\* specified here are,

\*\* Inner - Write through, No Write Allocate

\*\* Outer - Write Back, Write Allocate

\*/

REGION regionDdr = {

MMU\_PGTYPE\_SECTION, START\_ADDR\_DDR, NUM\_SECTIONS\_DDR,

MMU\_MEMTYPE\_NORMAL\_NON\_SHAREABLE(MMU\_CACHE\_WT\_NOWA,

MMU\_CACHE\_WB\_WA),

MMU\_REGION\_NON\_SECURE, MMU\_AP\_PRV\_RW\_USR\_RW,

(unsigned int\*)pageTable

};

/\*

\*\* Define OCMC RAM region of AM335x. Same Attributes of DDR region given.

\*/

REGION regionOcmc = {

MMU\_PGTYPE\_SECTION, START\_ADDR\_OCMC, NUM\_SECTIONS\_OCMC,

MMU\_MEMTYPE\_NORMAL\_NON\_SHAREABLE(MMU\_CACHE\_WT\_NOWA,

MMU\_CACHE\_WB\_WA),

MMU\_REGION\_NON\_SECURE, MMU\_AP\_PRV\_RW\_USR\_RW,

(unsigned int\*)pageTable

};

/\*

\*\* Define Device Memory Region. The region between OCMC and DDR is

\*\* configured as device memory, with R/W access in user/privileged modes.

\*\* Also, the region is marked 'Execute Never'.

\*/

REGION regionDev = {

MMU\_PGTYPE\_SECTION, START\_ADDR\_DEV, NUM\_SECTIONS\_DEV,

MMU\_MEMTYPE\_DEVICE\_SHAREABLE,

MMU\_REGION\_NON\_SECURE,

MMU\_AP\_PRV\_RW\_USR\_RW | MMU\_SECTION\_EXEC\_NEVER,

(unsigned int\*)pageTable

};

/\* Initialize the page table and MMU \*/

MMUInit((unsigned int\*)pageTable);

/\* Map the defined regions \*/

MMUMemRegionMap(&regionDdr);

MMUMemRegionMap(&regionOcmc);

MMUMemRegionMap(&regionDev);

/\* Now Safe to enable MMU \*/

MMUEnable((unsigned int\*)pageTable);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// This is the main application entry function.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int

main(void)

{

unsigned int ulTxCount;

unsigned int ulRxCount;

MMUConfigAndEnable();

//

//USB module clock enable

//

USB0ModuleClkConfig();

//

//USB interrupt enable

//

USBInterruptEnable();

//

//Delay timer setup

//

DelayTimerSetup();

//

// Initialize the transmit and receive buffers.

//

USBBufferInit((tUSBBuffer \*)&g\_sTxBuffer);

USBBufferInit((tUSBBuffer \*)&g\_sRxBuffer);

//

// Pass our device information to the USB library and place the device

// on the bus.

//

USBDBulkInit(0, (tUSBDBulkDevice \*)&g\_sBulkDevice);

//

// Clear our local byte counters.

//

ulRxCount = 0;

ulTxCount = 0;

//

// Main application loop.

//

while(1)

{

//

// Have we been asked to update the status display?

//

if(g\_ulFlags & COMMAND\_STATUS\_UPDATE)

{

//

// Clear the command flag

//

g\_ulFlags &= ~COMMAND\_STATUS\_UPDATE;

}

//

// Has there been any transmit traffic since we last checked?

//

if(ulTxCount != g\_ulTxCount)

{

//

// Take a snapshot of the latest transmit count.

//

ulTxCount = g\_ulTxCount;

//

// Update the display of bytes transmitted by the UART.

//

}

//

// Has there been any receive traffic since we last checked?

//

if(ulRxCount != g\_ulRxCount)

{

//

// Take a snapshot of the latest receive count.

//

ulRxCount = g\_ulRxCount;

}

}

}

### Beagle Bone and MSP430 Communication

### Beagle Bone and MSP430 Communication

## Application