Contents

[Software: 2](#_Toc323755445)

[1. Gaze Tracking Algorithm 2](#_Toc323755446)

[1.1 Introduction to Gaze Tracking: 2](#_Toc323755447)

[1.2 Overview of Approach 2](#_Toc323755448)

[1.3 Algorithm Implementation 4](#_Toc323755449)

[1.4 Lighting Configuration 6](#_Toc323755450)

[1.5 Working Example 7](#_Toc323755451)

[1.6 Results & Discussion 10](#_Toc323755452)

[2. Command Interpretation 11](#_Toc323755453)

[3. Calibration: 12](#_Toc323755454)

[3.1 Why Do We Calibrate? 12](#_Toc323755455)

[3.2 Calibration GUI 12](#_Toc323755456)

[4. Software Interfaces: 13](#_Toc323755457)

[4.1 Host Computer and Beagle Bone Communication: Transferring Frames 13](#_Toc323755458)

[4.2 Beagle Bone and MSP430 Communication 14](#_Toc323755459)

[4.3 MSP430 and Host Computer Communication 14](#_Toc323755460)

[5. Application 15](#_Toc323755461)

[Hardware: 16](#_Toc323755462)

[6. Power Supply 21](#_Toc323755463)

[7. Institutional Review Board (IRB): 26](#_Toc323755464)

[8. Appendix 26](#_Toc323755465)

[8.1 Software 26](#_Toc323755466)

[8.2 Hardware 26](#_Toc323755467)

[9. Bibliography: 26](#_Toc323755468)

# Software:

# Gaze Tracking Algorithm

## Introduction to Gaze Tracking:

The premise behind Gaze tracking is to track the line of the sight of the person. Eye Gaze tracking has been an active research topic for decades because of its potential usages in Human Computer Interaction (HCI). A popular application of gaze tracking is to allow disabled users with controlling common computer task such as clicking or moving the cursor.

Gaze tracking is commonly done using glint-pupil vector tracking. This method uses an infrared light to illuminate the eye and increase contrast in the eye for ease of tracking pupil and other useful features of the eye. Glint is the reflection of the light off of the retina that appears as a bright spot on the cornea [1] [2]. It is used as a reference point in the eye because the glint location remains stationary with respect to the eye as the direction of gaze changes. If the infrared illumination is coaxial with the optical path, then the eye acts as a retroreflector as the light reflects of the retina creating a bright pupil effect

If the illumination source is offset from the optical path, then the pupil appears dark because the reflection from the retina is directed away from the camera. The bright and dark pupil effects enhance the contrast of features in the eye that make the algorithm more accurate. This method tracks the pupil, and ultimately the vector between the pupil and glint. This vector is then mapped (mapping is obtained during calibration process) to location in the screen [2].

While the glint-pupil vector tracking is proved to be very accurate in determining gaze of user’s eyes, it is computationally intensive. Since our algorithm must be running real time on a microprocessor, speed of computation is a huge constraint. Thus we decided to merely track the pupil to find the direction of user’s gaze. Furthermore, while experimenting with using infrared light to illuminate the eye proved to make the algorithm accurate, issues with IRB approval prohibited us from directing infrared light at the eye. This is discussed in greater detail in section 7.

## Overview of Approach

To determine the direction of user’s gaze, we aim to track the pupil of the user. To implement gaze tracking, we take advantage of a few characteristics of human eyes and more specifically of the pupil. Since the pupil is the darkest region of the human eye, we base our algorithm in analyzing dark regions in each frame. Secondly, we expect the pupil to be circular and thus look for circular region. Finally, we expect the pupil to have moved in continuous fashion, e.g. the position of the pupil does not change significantly in small amount of time.

To obtain images of the user’s eyes, we use a commercial Logitech C200 Web Camera shown in Figure 1. The camera is rated to output up to 30 frames per second with frame resolution of 640x480.



Figure 1: Logitech C200 Web Camera. Datasheet for C200 camera is shown in the appendix.

To obtain image data from the web camera, OpenCV 2.0 is used. In particular, using the OpenCV built in library function cvQueryImage(), an image data in RGB format is obtained from the USB camera. Note that this OpenCV function uses the Linux USB video class driver to communicate with the USB camera.

Once we have the image data, we begin implementing our image processing algorithm that is outlined in Figure 2. We begin by thresholding the image which finds pixels in the image that have intensity value below a calibrated threshold value. From the selected pixels, we find connected regions. For a connected region to be passed as the pupil, we have three requirements: 1) connected region has approximately the size of the user’s pupil that is determined from calibration, 2) the region is approximately circular, and 3) continuity condition: the centroid of the candidate region should remain close to previously found centroid. If these requirements are not met, then the threshold parameter is varied until they are met or until we have reached maximum number of iterations that indicate that the user is blinking. Once such region that meets these requirements is found, the region is cleaned up in the “Remove Tails” stage and its centroid is found.



Figure 2: Overview of Image Processing Algorithm

## Algorithm Implementation

Since our algorithm is being implemented in real time, computational efficiency is extremely important. To maximize the efficiency of our algorithm, we organize our data in the following data structures shown in Table 1 :

|  |  |  |
| --- | --- | --- |
| Data Structure Name: | Data Structure Type | Data Stored |
| imageData | Three dimensional array of bytes | The RGB pixel values for each coordinate in a given frame. |
| crPointList | Two dimensional array of points | Each row contains the coordinates of a connected region. Only regions that meet the area requirement are stored. |
| crBinary | Three dimensional array of binary Value | Binary(i,x,y) = 1 if the coordinate (x,y) is in region i, and 0 otherwise |
| crMap | Three dimensional array of integers | Contains the integer index which maps a point from the matrix form of the region to an element in crPointList. |
| crSize | Array of integers | Elements are the sizes of the connected regions. |
| crCount | Integer | Number of connected region stored. |

Table 1: Data structures

|  |  |
| --- | --- |
| Module Name: | threshold() |
| Inputs: | imageData, initial threshold |
| Outputs: | List of points that satisfy threshold criteria |
| Functional Description: | Scans each pixel in the region of interest in a frame and checks to see which pixels are dark enough to belong to the pupil. This process is repeated until a region (computed with getConnectedRegions()) with an area close to a reference area is found, or until a maximum number of iterations has been reached. If the maximum number of iterations is reached and no suitable regions are detected, identify the user as blinking. |

|  |  |
| --- | --- |
| Module Name: | getConnectedRegions() |
| Inputs: | List of dark points identified in threshold() |
| Outputs: | crPointList, crSize, crCount,crBinary |
| Functional Description: | Uses a stack based implementation of the flood fill algorithm to identify connected regions of dark points. |

|  |  |
| --- | --- |
| Module Name: | findUnityRatio() |
| Inputs: | crBinary, crCount |
| Outputs: | Aspect ratio for each connected region in CR, index of the connected region with aspect ratio nearest to one |
| Functional Description: | Computes the ratio of the longest horizontal and longest vertical lengths. The connected region with the aspect ratio closest to one is identified as the pupil. |

|  |  |
| --- | --- |
| Module Name: | removeAberrations() |
| Inputs: | crPointList, crMap, crSize, Index indicating chosen region |
| Outputs: | Updated crPointList and crSize |
| Functional Description: | Computed the number of pixels in each row of the connected region and find the mean and standard deviation of the pixel counts. Remove rows that have pixel counts that fall out of a certain number of standard deviations away from the mean. Repeat the process in the vertical direction. |

|  |  |
| --- | --- |
| Module Name: | computeCentroid() |
| Inputs: | crPointList, crSize, Index indicating chosen region |
| Outputs: | Coordinates of the centroid |
| Functional Description: | Sum the coordinates of all points belonged to the pupil region and divide by the total number of points. The result is the coordinate of the centroid. |

|  |  |
| --- | --- |
| Module Name: | generateCursorCommand() |
| Inputs: | Reference centroid coordinates, current centroid coordinates |
| Outputs: | Cursor command code (integer values ranging from 0 to 5) |
| Functional Description: | Compares the reference centroid coordinates to the current centroid coordinates. If the difference between the two coordinates exceeds a threshold value for 10 consecutive frames, then the cursor command output will be changed accordingly. Otherwise, the previous cursor command is output. |

|  |  |
| --- | --- |
| Cursor Command | Direction of Movement |
| 0 | No change |
| 1 | Right |
| 2 | Left |
| 3 | Up |
| 4 | Down |
| 5 | Left Click |

Table 2: Translation between cursor command and direction

## Lighting Configuration

After testing our image processing algorithm in Capstone without a controlled lighting environment, we saw that reflection and glares from the fluorescent light on the pupil confused our algorithm. To combat this issue, we have a lamp that point to the back wall in capstone and away from the user. This resolved most of our issues with reflections, though the conditions are still not ideal for accurate gaze tracking. An image of the lamp we used is shown in Figure 3.

Note as we discussed in section 1.1, IR light enhances contrast in the eye that makes the image processing easier. To take advantage of this, we have removed the IR filter in the Logitech C200 camera, allowing the minimal IR light from fluorescent light in capstone to be captured by the camera.



Figure 3: Configuration of lamp in Capstone

## Working Example

For better understanding of our algorithm and its main steps, we will demonstrate the result of each step of our algorithm on the image shown in Figure 4 obtained from out webcam:



Figure 4: Original Image

Suppose that from a calibration process, the initial threshold value was chosen to be 34 and the reference pupil area was computed to be 554. After applying thresholding and coloring pixels that have intensity value less than the threshold value, we obtain the result shown in Figure 5.



Figure 5: After Thresholding: Where biggest connected region is colored yellow and the second largest colored red

Note that two connected regions that have area relatively close to the reference pupil area are the eyebrow and the pupil. The eyebrow is computed to have 2416 pixels and the pupil has 1024 pixels. Since both connected regions have areas that fall outside of the acceptable deviance from the reference pupil area, we do not consider them as candidates for the pupil. Since both regions have area greater than reference pupil area, out algorithm decreases the threshold value to 31. The result after adjusting the threshold value is shown in Figure 6.



Figure 6: Image after adjusting threshold value

The area of each connected region in the processed image is:

Eyebrow 1: 564

Eyebrow 2: 196

Pupil: 512

Note that the first and third connected regions have area within the acceptable range to be considered as the pupil. To distinguish between these two connected regions, we compute the aspect ratio of both connected regions:

Eyebrow Aspect ratio = 0.49

Pupil Aspect Ratio = 1.07

The pupil aspect ratio is within the acceptable range but the eyebrow aspect ratio is not. Thus our algorithm correctly identifies the pupil as the region of interest. Finally, we remove aberrations of the pupil and obtain the processed image shown in Figure 7 with the centroid of the region denoted as intersection of the two lines:

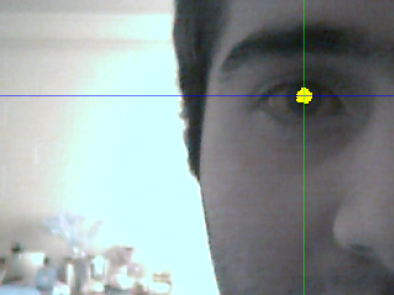


Figure 7: Final Processed Image

## Results & Discussion

To evaluate our results, we consider efficiency (timing) and accuracy of our gaze tracking algorithm.

The accuracy of gaze tracking is essentially determined whether we accurately calculated the direction of user’s gaze. From experiments, we have demonstrated that we are almost always successful at obtaining the direction of user’s gaze. There is however room for improvements. Currently, our algorithm is sensitive to lighting conditions. If we had more time, we would have preferred to make our algorithm more robust to various kinds of lighting.

For the efficiency of our algorithm, we tested our algorithm both on a pre-recorded video and on live mode. Table 3 shows the efficiency of our algorithm in number of frames per second processed for both modes.

|  |  |
| --- | --- |
| Mode | Frames Per Second(fps) |
| Live Mode | ~10 |
| Pre-Recorded Video | 60 |

Table 3: Frames per second for both modes of operation

The efficiency of our algorithm is clearly visible in the number of frames per second to process pre-recorded video. This mode is more representative of the timing of our algorithm since there is not much overhead on reading video. On the other hand, processing in live mode is slower because of the overhead from grabbing frames from the camera before we process. In fact, we notice that the CPU on the Beagle Bone reaches 95% from processing in live mode.

# Command Interpretation

The output of the gaze tracking algorithm is a command that indicates the direction of user’s gaze. The cursor command value ranges from 0 to 5. The interpretation of cursor commands to an associated cursor response is shown in Table 4.

|  |  |  |
| --- | --- | --- |
| Cursor Command | Direction of Movement | Vector |
| 0 | No change | (0,0) |
| 1 | Right | (1,0) |
| 2 | Left | (-1,0) |
| 3 | Up | (0,-1) |
| 4 | Down | (0,1) |
| 5 | Left Click | N/A |

Table 4: Translation of cursor command to vector for direction of movement

Since the human eyes often move involuntary, we require that the output of our algorithm is a particular cursor command for ten consecutive processed frames before executing cursor movement or click. Once this requirement has been met, cursor movement is implemented using Windows API (Application Programming Interface).

For simplicity, the implementation of cursor movements/clicking is done in Python. The cursor commands 0-4 received from the processing module is associated with a vector as demonstrated in Table 4. For such cursor commands, the new cursor position is obtained by the following expression:

where the parameter, speed, is adjustable by the user. Finally, the cursor command “5” corresponds to a left mouse click. This was also done with Windows API call.

# Calibration:

## Why Do We Calibrate?

Since characteristics of the human eyes vary between users, the user is guided through a calibration process where the characteristics of their eyes are obtained. Parameters that are obtained through calibration stage are:

1. Initial Threshold Value: the initial value that is used by algorithm to locate pixels that have intensity value below the initial threshold value.
2. Region of Processing: a user adjustable box that would encapsulate the eye.
3. Minimum change in distance from reference pupil to associate with change in direction of gaze.
4. Reference pupil centroid when user is looking at the center of screen.
5. Reference pupil area: the number of pixels of the centroid when user is looking at the center of screen.

Note that parameters 1 and 2 are manually adjusted by the user whereas the others are computed by the gaze tracking algorithm

## Calibration GUI

For ease of adjusting calibration parameters, we designed an interactive calibration process, in which the user will be able to modify algorithm parameters via a GUI on the computer.

The GUI interface will contain slide bars for each adjustable parameter for the user to manually modify the parameters. More specifically, the will be an adjustable bar for initial threshold, and boundaries of the box that encapsulates the eye. In the calibration process, the user is asked to manually adjust the threshold value until the overlay image has his/her pupil colored red. The user will also adjust the boundaries of processing region to enclose the eye. Note that the adjusted calibration parameters are communicated to the beagle bone via Xbee (for more details on this communication, please refer to section 4). The beagle bone will process the image by marking the pixels with intensity value below a threshold value as red. The processed calibration frames from the camera will be transferred via Ethernet on the beagle bone to the computer so that the user can view the effect of these parameters in real time and choose suitable values (for more details on this communication, please refer to section 4.1).

The Calibration GUI was implemented in python using the Tkinter, a standard python GUI interface. Screen captures of a few steps of calibration process are shown in the following figures.

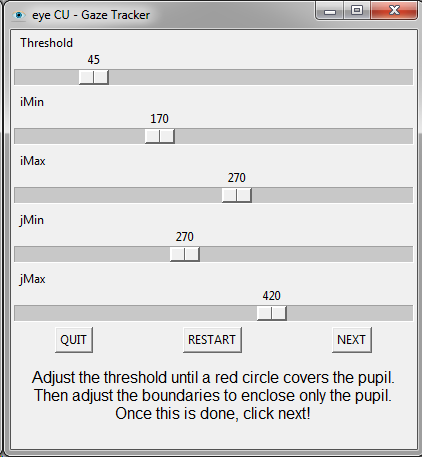


Figure 8: Calibration GUI

# Software Interfaces:

## Host Computer and Beagle Bone Communication: Transferring Frames

For purposes of debugging and calibration, frames that are being processed need to be sent from the Beagle Bone to the Host Computer. We implemented frame transfer in two methods, the first one being through USB and the second being through Ethernet.

The transfer via USB was done as a bulk transfer. This transfer was done in embedded environment on the Beagle Bone. To develop USB transfer, an available USB bulk transfer example on StellarisWare software was modified and used. The transfer rate via USB is approximately a frame per second. Though this rate is rather slow, it is not an issue because this transfer is only used for debugging purposes. The code for video transfer on the host computer side as well as the beagle bone can be found in appendix

Though USB bulk transfer was sufficient for transferring video, it was not suitable at the end of our project since we switched to Linux environment for the beagle bone to interface with the USB camera. As a result, transferring video Ethernet was developed. With the available linux drivers for Ethernet communication and software tools provided by Sam Siewert, one way Ethernet communication over Ethernet was written. To set up communication between Beagle Bone and Host computer, a port and the IP Address is agreed upon both on the Beagle Bone and the host computer. When Beagle Bone sends frame over Ethernet, the size of frame is specified. The Host computer continuously reads from the specified port until the full data is obtained. The host computer displays the frame using python. Note the code for this can be found in Appendix \*\*\*.

## Beagle Bone and MSP430 Communication

To communicate cursor commands and calibration data between the MSP430 board and the Beagle Bone, we are using XBEEs. Interfacing to an XBEE is straightforward; all you have to do is use a standard UART (universal asynchronous receiver transmitter) interface. On each of the processors chosen we have several UART ports built in for us to use. Programmatically communicating using XBEEs is just like talking over a serial port, the XBEE modules handle all of the wireless transmission.

## MSP430 and Host Computer Communication

Communication between the host computer and the MSP430 board is another UART serial interface. This time we have a FTDI chip that converts UART to USB for transmission. On the host computer side there is a USB to serial driver. In this way we can communicate with the board using a serial module in any programming language. For our project, we decided to use python, and the pySerial module. The python program uses a blocking read to look for data on the serial port. Once the cursor command is received, the host computer processes the command. The communication is two way, so that we can send calibration data back to the Beagle Bone.

The development on the MSP430 UART was done with Code Composer Studio version 5 and OLIMEX MSP430F169 prototype board. Appendix A contains the code for a single UART communication. The OLIMEX prototype board has only one available UART and to test UART communication a string sent to the MSP430 would be echoed back to the user. Communication tests used an XBee connect to the MSP430 prototype board and another XBee on a XBee explorer from SparkFun.

Double UART MSP430 code was created. However, testing of this code could not be achieved. The code would enable UART0 and UART1 of the MSP430. If one UART received information then it would be sent to the other UART and vice versa. The idea was to connect one UART to an XBee and another to USB. If data is received from the XBEE, it would be transmitted through the USB to the host computer. If data was sent through the USB, it would be transmitted through the XBee to the Beagle Bone.

# Application

To display the functionality of our eye tracking system, the duck hunt game was developed. Duck Hunt is implemented in python and uses the pygame library. It is a simple 2-D point and click shooting game. Ducks fly across the screen from left to right. The player has an infinite amount of shots to hit the duck on the screen. However, if a duck makes it across the screen, the game will end and the iconic dog pops out of the grass to indicate the game is over. The player has two options once the game ends. The player can hit the up arrow key on the keyboard to restart the game or hit the escape key to end the game. Credit to Johann Gomes for the original game code in which our game is based off of. The original game code can be found on pygame.org. An image of the Duck Hunt game is shown in Figure 9.



Figure 9: Duck Hunt Game

# Hardware:

**JTAG Connector**

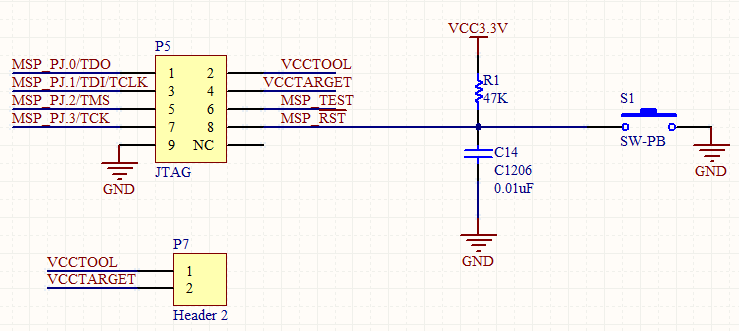


Figure 10: JTAG Programmer Connector Schematic

In order to program the MSP430, we needed a way to load code. The MSP430F5438A requires a 4-wire JTAG programming protocol. Using the Texas Instruments SLAU278H JTAG configuration, a schematic of this interface was developed as shown in Figure 10 [3].s The MSP-FET430UIF was used as the JTAG programmer for the MSP430F5438A through Code Composer Studio version 5. Note that the MSP-FET430UIF cannot be used to program the MSP430F5438A in Code Composer Studio version 4 due to an incompatible firmware update for the programmer. This information is currently not well documented in the literature provided by Texas Instruments; however, information on this issue may be found on the TI E2E Community Support Forums. As shown in Figure 10, the four pins, TDO, TDI, TMS, and TCK are used to program the MSP430F5438A. Additionally, the two pins, VCC TOOL and VCC TARGET are used to either power the programmer via USB power supplied by the host computer or powering from the target MSP430F5438A board. Depending on the user’s requirements, the user may choose to power the programmer off of the board or via USB through a jumper connection on the P7 header.

The programmer also has a connection to the /RST pin of the processor for resetting the processor after programming. The reset switch provides a quick way to re-run code on the MSP430F5438A in the event that the processor gets hung-up, or to simply reset the processor. In order to reset the processor, /RST is brought to ground. When the processor is not in reset, and in order to maintain normal operation, /RST is pulled up to VCC. The reset circuit consisting of R1, C14, and the momentary push button is called a button debounce circuit. This circuit prevents spikes of high and low voltages corresponding to high and low input to the reset pin of the processor. This debounce circuit relies principally on the capacitor which serves to resist sudden voltage changes on the line.

**RS232 Interface**

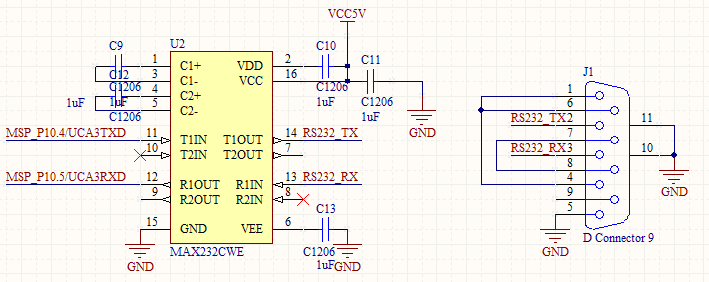


Figure 11: RS232 Interface Schematic

As a back-up in case there was an issue with the USB interface with the MSP430F5438A we employed an RS232 interface as shown in Figure 11. This would provide an additional interface between the MSP430F5438A and the host computer for XBee RX and TX data. As you can see in the schematic, the MAX232 is powered externally by 5V but it requires 10V to operate which is supplied internally through dual charge-pump DC-DC voltage converters. The first charge pump on the MAX232 takes in 5V doubling the voltage to approximately 10V. While the second charge pump takes the +10V, inverts the input, outputting -10V. The charge pump, capacitor values, and pin configuration for typical operation can be found in the datasheet in the appendix for the MAX232. Furthermore, for interfacing a host computer over serial, a DB9 serial connector is necessary as shown in Figure 11.

**USB Circuit**

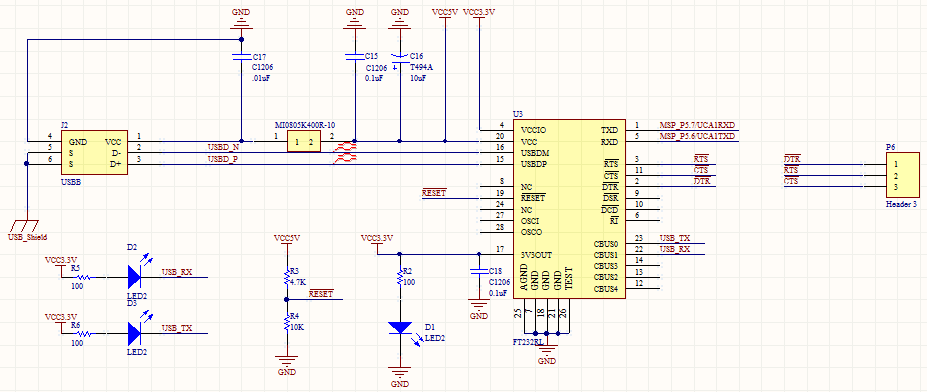


Figure 12: USB Interface Schematic

Figure 12 displays the USB Interface Schematic for the custom MSP430 board. The FT232RL chip from FTDI is a USB to serial interface allowing the MSP430F5438A to communicate XBee RX and TX data over a USB connection with the host computer. The design of this circuit was facilitated using an application note by FTDI on hardware design guidelines for FTDI ICs [4]. As shown in Figure 12, the FT232RL is in a self-powered or bus powered configuration, drawing power from the USB-B connector connected to the host computer. Additionally, the FT232RL has a +3.3V output from an integrated LDO regulator which the datasheet advises should be decoupled to ground using a 100nF capacitor. The pin VCCIO is the power supply for the UART interface and CBUS group pins. The MSP430F5438A datasheet recommends a maximum of +3.6V at any pin, the VCCIO pin on the FT232RL must be set to +3.3V in order to ensure recommended operating conditions have been met. Since the FT232RL outputs +3.3V at pin 17, VCCIO could be powered by the FT232RL. Due to an oversight, VCCIO was tied to +5V so the PCB trace had to be removed and a white wire was added to connect pin 17 to VCCIO. Note that Figure 12 shows the proper configuration.

You can see in Figure 12 that a ferrite bead is connected in series with the USB power supply to reduce EMI noise from being radiated down the USB cable to the USB host from the FT232RL and associated circuitry. Obviously, the value of the ferrite bead depends on the total current drawn. A 1.5A 40Ω ferrite bead was employed by Laird Technologies. Furthermore, decoupling capacitors C15 and C16 were also added to minimize EMI emission and improve ESD immunity. The signals D- and D+ were routed as a differential pair in order to reject common-mode noise. This means that D- and D+ had very equal lengths in order to maintain the timing of signals. In addition, the overall length of these two signals was made to be as short as possible.

As a visual indication that the FT232RL was correctly powered, a power green LED was connected to the 3V3OUT pin. Additionally, since any of the CBUS I/O pins can be configured to drive an LED, LEDs were placed on CBUS0 and CBUS1 as visual indicators when data is being transmitted or received.

In Figure 12, /RESET on the FT232RL is connected to voltage divider consisting of R3 connected to 5V, in series with R4 connected to ground. This circuit serves to hold the IC in reset while a USB cable is not connected to the peripheral.

**XBee Circuit**

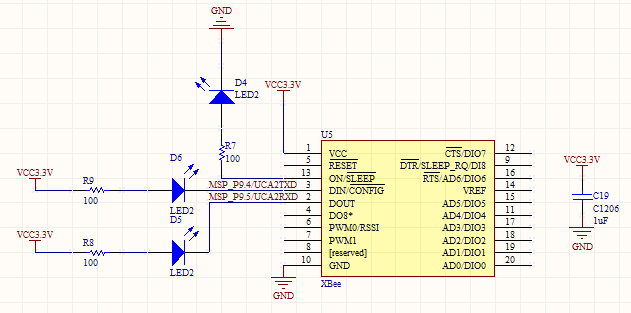


Figure 13: XBee Schematic

The XBee Series 1 chip antenna wireless transceiver was used to send as well as receive data from the XBee on the Beagle Bone to the host computer and vice versa. This XBee is controlled by the MSP430F5438A through a UART on the MSP.

The primary purpose of the LEDs on ON, DIN, and DOUT of the XBee is for debugging purposes. Additionally, a decoupling capacitor was placed across power and ground on the XBee. In an attempt to reduce reflections from the wireless transmitter, no components were placed immediately beneath the XBee in order to prevent turning a component into a disruptive antenna.

The XBee communication was set to standard baud rate of 9600Hz. This baud rate is more than sufficient for calibration and cursor command signals to be sent wirelessly. Calibration data is sent only on initial use. Cursor command signals are only sent at 30Hz.

**Microprocessor**

The design of the XBee transceiver board centers around the processor. This processor needed to be able to arbitrate data between an XBee and the host computer. The processor had to be reliable, powerful enough to arbitrate data, easy to develop with from both a software and hardware perspective as well as readily available. Since Capstone provides MSP430 development boards by Olimex and Texas Instruments offers free samples of most MSP430s we reasoned that we could begin development immediately on the MSP430 architecture. We choose the MSP430F5438A processor because we needed a processor with at least two UARTs, one for XBee communication and one for USB communication with the host computer. In addition, we wanted to add an RS232 serial interface as a backup in case the USB interface did not operate as desired. Therefore we needed a processor with at least three UARTs. Texas Instruments only has MSP430s with either two or four UARTs but not three so we decided to not sacrifice a UART and go with four. The MSP430F5438A has four UARTs, in addition to coming in a 100 pin LQFP package.

**MSP430F5438A**

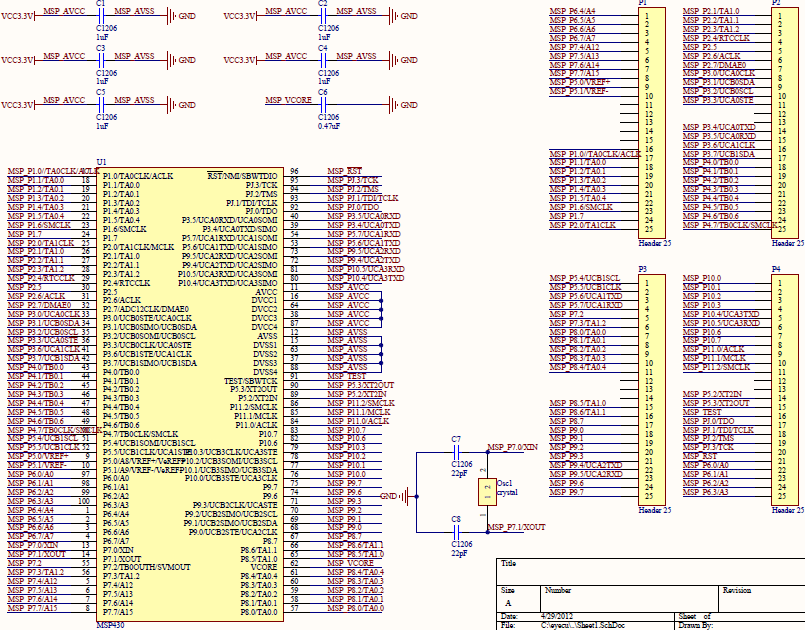


Figure 14: MSP430F5438A Processor Pin Breakout

The processor we are using on the MSP430 board is a MSP430F5438A. A schematic of the processor can be seen in Figure 14. As you can see in the schematic, the MSP430 runs off of 3.3V which is generated by the USB circuit. The MSP430 is supported by an external oscillator circuit with a 32.768kHz crystal. The MSP430F5438A has an internal 32.768kHz oscillator; however, it is recommended to use an external oscillator for time sensitive applications.

## Power Supply

The power supply is designed to supply power to the Beagle Bone. It utilizes a linear regulator which takes in an input voltage range of 6V to 15V and regulates this voltage to 5V. The power supply is powered by a 6.6V lithium iron phosphate battery (LiFePO4). Figure 15 shows the LiFePO4 battery used in the power supply.



Figure 15: 6.6V LiFePO4 Battery

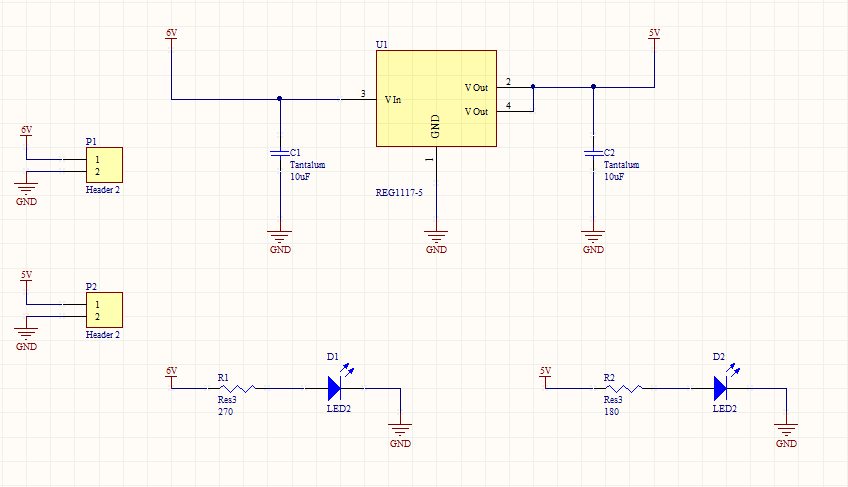


Figure 16: Power Schematic Revision 1

The first revision of the power system, shown in Figure 16, utilizes a single linear regulator with two decoupling capacitors. Status LEDs are added to the Vin and Vout rails of the power supply to provide a visual indication of an input voltage present and output voltage prepared for use. Figure 17 shows the first revision of the power supply PCB. P1 is the input supplied by the lithium iron phosphate battery via a Deans Ultra Plug. P1 is the output of the power supply and uses a DC barrel jack to connect to the Beagle Bone.

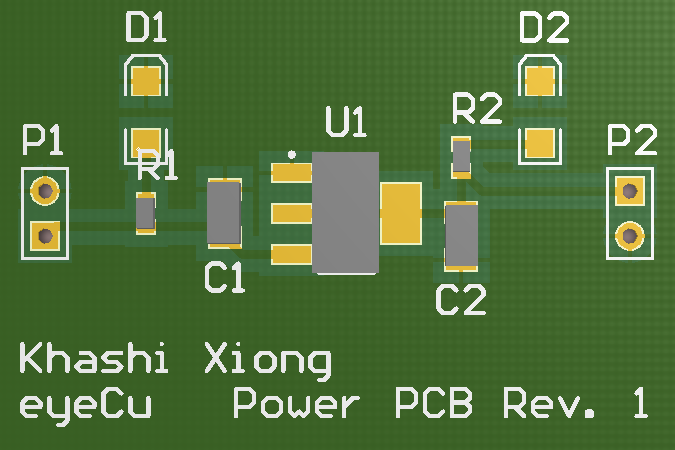


Figure 17: Power Supply PCB Revision 1

The second revision of the power supply utilizes the same circuit as the first revision power supply, but the second revision power supply also includes a recharging circuit capable of recharging the lithium iron phosphate battery. The charge controlling integrated circuit is a BQ2057 from Texas Instruments and utilizes a PNP transistor, decoupling capacitors, and a charging LED. When the battery is fully charged the charging LED turns off. Figure 18 shows the second revision power supply with battery recharging circuit.

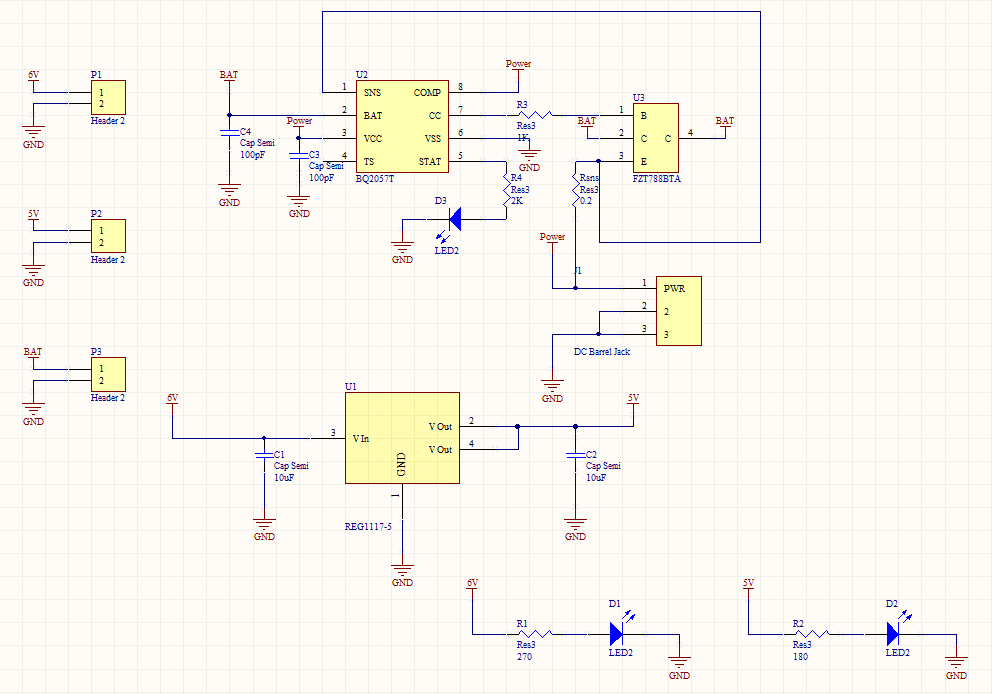


Figure 18: Power Schematic Revision 2

Figure 19 shows the second revision PCB. Power for the recharging circuit takes an input DC power supply through J1 and outputs through a Deans Ultra Plug via P3.

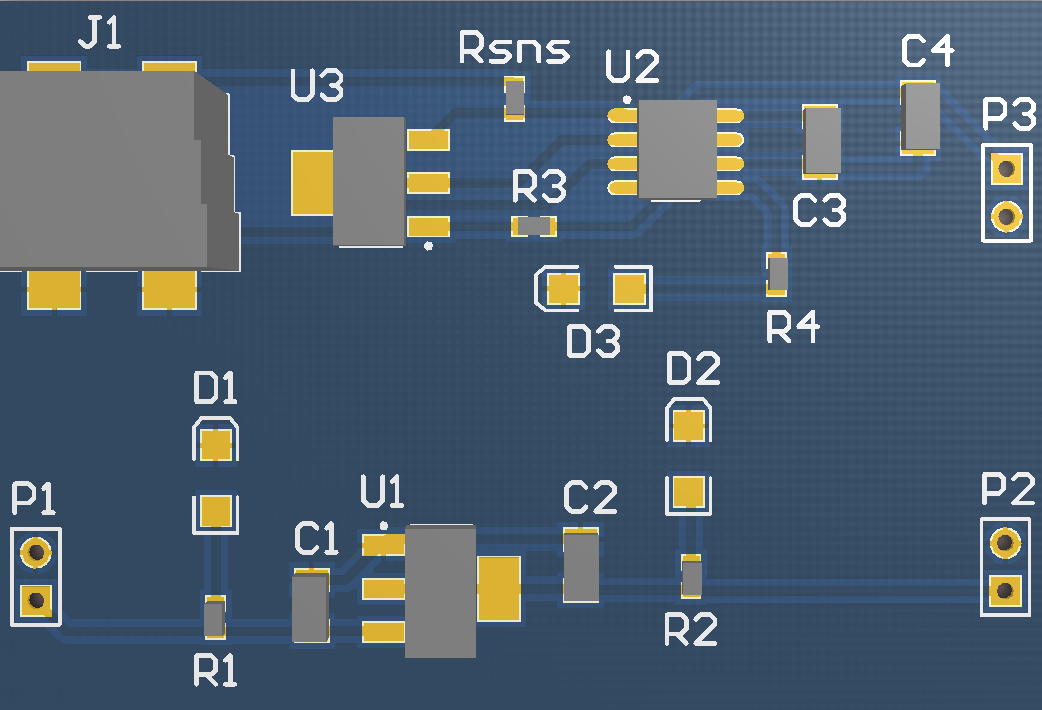


Figure 19: Power PCB Revision 2

Figure 20 shows the flowchart for the recharging circuit.

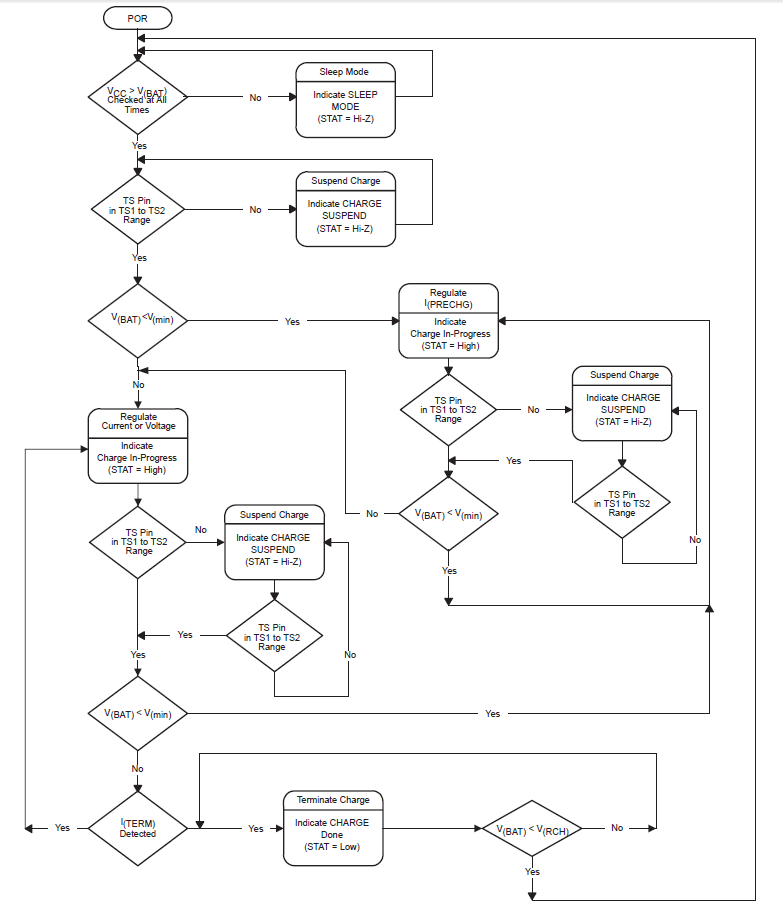


Figure 20: Recharging Circuit Flowchart

# Institutional Review Board (IRB):

# Appendix

## Software

## Image Processing Algorithm

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

// Name: pointStack.h

// Author: Nick Bertrand

// Goal: Defines the stack used for finding connected regions

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

#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.h

// Author: Nick Bertrand, Armeen Taeb

// Goal: Defines the header for functions in eyeFeatureExtraction

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

#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

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

// Name: eFEParam.h

// Author: Nick Bertrand, Armeen Taeb

// Goal: Defines parameters

//---------------------------------------------------------------------------#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.cpp

// Author: Nick Bertrand, Armeen Taeb

// Goal: contains functions for image processing

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

#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

// Goal: Grab frames and call processing functions

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

#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

// Goals: function for handling stack for doing connected regions

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

#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**;**

**}**

**}**

## Calibration

# -----------------------------------------------------------------------

# 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

## Software Interfaces

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

// Name: gazeTracker\_DebugVideo.c

// Author: Nick Bertrand,Armeen Taeb

// Goal: C code on the host computer side. Reads a frame from the Beagle Bone and displays the image on the screen.

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

#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**);**

**}**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Name: usb\_dev\_bulk.c

// Authors: Modified fNick Bertrand, Armeen Taeb

// Goal: Sets up Bulk Transfer on the Beagle Bone

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#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"

//

// 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**;**

**}**

**}**

**}**

## 8.2 Hardware

# Bibliography:

|  |  |
| --- | --- |
| [1] | L. Dongheng, D. Winfield and D. Parkhurst, "Starburst: A hybrid algorithm for vide-based eye tracking combining feature-based and model-based approaches," *Computer Vision and Pattern Recognition - Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on,* p. 79, 2005. |
| [2] | E. Guestrin and E. M, "General theory of remote gaze estimation using pupil center and corneal reflections," *Biomedical Engineering, IEEE Transactions,* vol. 53, no. 6, pp. 1124-1133, 2006. |
| [3] | Texas Instruments Incorporated, "MSP430 Hardware Tools: User's Manual SLAU278H," 2009, Revised 2011. |
| [4] | Future Technology Devices International Ltd., "Application Note AN\_146 USB Hardware Design Guidelines for FTDI ICs," Future Technology Devices International Limited, 2010. |