Team eyeCU

Milestone 1 and 2 Goals

Nick Bertrand, Arielle Blum, Mike Mozingo, Armeen Taeb, Khashi Xiong

Milestone 1

|  |  |
| --- | --- |
| **Module** | Wireless |
| **Inputs** | Wireless data generated by Beagle Bone. UART data from the MSP430. |
| **Outputs** | UART data to the MSP430 board. Wireless data to the Beagle Bone. |
| **Functionality** | The XBEE wireless will have two-way communication. The Beagle Bone XBEE will be able to send cursor commands to the MSP430 XBEE. And the MSP430 XBEE will send algorithm parameter data the Beagle Bone XBEE. |
| **Demonstration Plan** | Send the string, “Hello World”. The string will be sent from the MSP430 development board through UART1 and into an XBEE. This XBEE will transmit this string to another XBEE connected to a USB XBEE explorer and will transmit the string to the computer and display, “Hello World” on a terminal. |

|  |  |
| --- | --- |
| **Module** | Camera board |
| **Inputs** | Visual data |
| **Outputs** | 8-bit data bus with image information in RGB 5:6:5.  HD, VD, DCLK |
| **Functionality** | The camera board houses the camera and supporting hardware. The camera will output da­­ta in the configuration above. Each pulse of DCLK signifies another 8-bit parallel chunk is ready to be read. HD signifies the end of a line in the 640x480 resolution of the final images while VD signifies the end of the image. |
| **Demonstration Plan** | To show functionality of the camera board by displaying the data clock on an oscilloscope. |
| **Module** | Cursor Movement |
| **Inputs** | Command code |
| **Outputs** | Computer cursor movement |
| **Functionality** | Uses Windows API to get and set cursor position. Command code is translated into a direction vector. Updated cursor position will be set to old cursor position plus speed multiplied by the direction vector. |
| **Test Plan** | Write a script to compile a list of simulated cursor commands and visually verify that the cursor moves as desired with the simulated commands as input. |

# Level 2: Host Computer Software

|  |  |
| --- | --- |
| **Module** | Cursor Movement |
| **Inputs** | Command code from a precompiled list, speed parameter |
| **Outputs** | Computer cursor movement |
| **Functionality** | Uses Windows API to get and set cursor position. Command code is translated into a direction vector. Updated cursor position will be set to old cursor position plus speed multiplied by the direction vector. |
| **Test Plan** | Write a script to compile a list of simulated cursor commands and visually verify that the cursor moves as desired with the simulated commands as input. Cursor commands are integer values that correspond to a movement vector. |

# Level 2: DSP Software (Demonstrated on host computer)

|  |  |
| --- | --- |
| **Module Name:** | threshold() |
| **Inputs:** | imageData from a pre-recorded video, initial threshold (manually chosen) |
| **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. |
| **Test Plan:** | Color the dark pixels as red, and visually inspect the image to ensure that pixels that meet the threshold requirement have been marked. |

|  |  |
| --- | --- |
| **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. |
| **Test Plan:** | Color each connected region that meets the size requirement a different color, and visually inspect the resulting image. |

|  |  |
| --- | --- |
| **Module Name:** | getAspectRatio() |
| **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. |
| **Test Plan:** | Print out a list of the aspect ratios computed and visually inspect an image with the connected regions in CR. |

|  |  |
| --- | --- |
| **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. |
| **Test Plan:** | Display the image with the chosen region before and after removal of aberrations and verify that aberrations have indeed been rmoved. |

|  |  |
| --- | --- |
| **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. |
| **Test Plan:** | Indicate the centroid with horizontal and vertical lines and verify by visual inspection that the intersection falls on the centroid of the region. |

|  |  |
| --- | --- |
| **Module Name:** | generateCursorCommand() |
| **Inputs:** | Reference centroid coordinates, current centroid coordinates |
| **Outputs:** | Cursor command code |
| **Functional Description:** | Compares the reference centroid coordinates to the current centroid coordinates. If the difference between the two coordinates exceeds a threshold value for a fixed number of consecutive frames, then the cursor command output will be changed accordingly. Otherwise, the previous cursor command is output. |
| **Test Plan:** | We will manually pick a frame from which to configure the reference centroid. The input video will have a known sequence of gestures (for example, the user will look up, blink, look left, blink, etc.) The cursor command generated will be output in real time on the console as the video is processed. |

|  |  |
| --- | --- |
| **Additional Goals** | **Does “additional goals” mean goals that we will definitely have done that just don’t fit into one of the above modules, or are these stretch goals that may or may not be done?** |
|  | * Have power PCB revision one designed * Have MSP430 board revision one designed |
|  |  |
|  |  |
|  |  |

Milestone 2

|  |  |
| --- | --- |
| **Module** | Wireless |
| **Inputs** | Wireless data generated by Beagle Bone. UART data from the MSP430. |
| **Outputs** | UART data to the MSP430 board. Wireless data to the Beagle Bone. |
| **Functionality** | The XBEE wireless will have two-way communication. The Beagle Bone XBEE will be able to send cursor commands to the MSP430 XBEE. And the MSP430 XBEE will send algorithm parameter data the Beagle Bone XBEE. |
| **Demonstration Plan** | Send cursor commands from Beagle Bone XBEE to the MSP430 XBEE. Send algorithm parameters from MSP430 XBEE to Beagle Bone XBEE. |

|  |  |
| --- | --- |
| **Module** | Camera board |
| **Inputs** | Visual data |
| **Outputs** | 8-bit data bus with image information in RGB 5:6:5.  HD, VD, DCLK |
| **Functionality** | The camera board houses the camera and supporting hardware. The camera will output da­­ta in the configuration above. Each pulse of DCLK signifies another 8-bit parallel chunk is ready to be read. HD signifies the end of a line in the 640x480 resolution of the final images while VD signifies the end of the image. |
| **Demonstration Plan** |  |

# Level 3: Calibration

|  |  |
| --- | --- |
| **Module** | GUI |
| **Inputs** | User adjusted parameters |
| **Outputs** | Visualization of the image processing |
| **Functionality** | Allows the user to control parameters in the algorithm such as initial threshold value and cursor speed. The image processing is visualized by overlaying colored regions over the original image. Also it allows user to pause eye controlled cursor movement. |
| **Test Plan** | Test to see if changes in parameters correspond to correct modification in the overlaying image. |

|  |  |
| --- | --- |
| **Module** | Indicators |
| **Inputs** | User starts calibration |
| **Outputs** | Visual cues to guide the user through calibration, Reference pupil centroid, reference pupil area, processing region |
| **Functionality** | Displays on-screen indicators to tell the user to look at a series of calibration points. From these points, a processing region and reference pupil size and location can be determined. |
| **Test Plan** | Save the frames used for each step in calibration, and manually verify that the parameters generated are correct. |

# Level 2: Host Computer Software

|  |  |
| --- | --- |
| **Module** | Serial Communication (Beagle Bone and host computer) |
| **Inputs** | 640x480 RGB calibration frames from serial communication module |
| **Outputs** | Queue of video frames |
| **Functionality** | Packets of video data are received via USB from the Beagle Bone. Once an entire frame of data has been received, the frame is stored in an image data array, and a pointer to the array is pushed onto a video frame queue.  Note: video is only transmitted from the Beagle Bone during the calibration process. Once calibrated, the device can be used using only the wireless interface. |
| **Test Plan** | First, we will program the Beagle Bone to send a series of known test packets containing trivial “hello world” data. Once we have verified that data can be transmitted and received correctly, we will send a video frame. The frame will then be displayed on the host computer. The final step will be to send and display a continuous stream of video. |

|  |  |
| --- | --- |
| **Module** | Serial Communication (MSP430 and host computer) |
| **Inputs** | Packets from MSP430 over USB |
| **Outputs** | Data extracted from packets |
| **Functionality** | Receives packets from the MSP430 at rate of 30Hz, extracts commands codes from the packet and puts them into a queue to wait to be processed by the cursor movement module. |
| **Test Plan** | Program the MSP430 board to output a set of simulated data and verify that it is correctly received by the host computer. The data is inspected to verify that is the same as the data that was sent. |

|  |  |
| --- | --- |
| **Module** | Calibration |
| **Inputs** | 640x480 RGB calibration frames sent via USB from the Beagle Bone |
| **Outputs** | Reference pupil centroid, reference pupil area, processing region |
| **Functionality** | A GUI interface on the host computer displays the images being captured by the camera with an overlay of the processing. The GUI allows the user to modify algorithm parameters. After some parameters are chosen manually, the user will be guided through a process to collect the remaining calibration values. |
| **Test Plan** | Visually verify that the modified parameters result to a modified overlay. Once this has been tested, verify that the parameters were successfully sent to the Beagle Bone by displaying them on the screen. |

|  |  |
| --- | --- |
| **Additional Goals** |  |
|  | * Have power PCB revision two designed. * Power PCB revision one will power the Beagle Bone, Camera, and XBEE. * Have prototype of the physical setup of the human controlled interface |
|  |  |
|  |  |