**Level 1:**

Image from Camera

Configuration Parameters

Host Computer Software

Beagle Bone Software

Control Codes

Cursor Movement

**Level 2: Beagle Bone DSP**



|  |  |
| --- | --- |
| **Module** | **DSP Algorithm** |
| **Inputs** | 640x480 RGB image data |
| **Outputs** | Command indicating the direction of the user’s gaze |
| **Functionality** | Processes the image to obtain the centroid of the pupil. It then compares the pupil centroid to the reference centroid to determine the direction of gaze. |
| **Test Plan** | Before the Beagle Bone is ready, sample images will be collected via a webcam and processed using an implementation of the algorithm that runs on a computer. Once the Beagle Bone and camera are ready, real-time testing can be performed and the direction of the user’s gaze can be indicated on the LCD. |

**Level 3: Beagle Bone DSP**

**Primary Software Data Structures:**

|  |  |  |
| --- | --- | --- |
| **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. |
| **CR** | Two dimensional array of points | Each row contains the coordinates of a connected region. Only regions that meet the area requirement are stored. |
| **CRSize** | Array of integers | Elements are the sizes of the connected regions. |
| **CRCount** | Integer | Number of connected regions stored. |

|  |  |
| --- | --- |
| **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. |
| **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:** | CR, CRSize, CRCount |
| **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:** | CR, CRSize, 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:** | CR, CRSize, Index indicating chosen region |
| **Outputs:** | Updated CR 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 removed. |

|  |  |
| --- | --- |
| **Module Name:** | **computeCentroid()** |
| **Inputs:** | CR, CRSize |
| **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. |

**Level 2: Calibration**

|  |  |
| --- | --- |
| **Module** | **Calibration** |
| **Inputs** | 640x480 RGB calibration frames |
| **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** | Verify that 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. |

**Level 2: Cursor Movement**

|  |  |
| --- | --- |
| **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** | Compile a list of simulated cursor commands and verify that the cursor moves as desired. |

**Level 2: Serial Communication**

|  |  |
| --- | --- |
| **Module** | **Serial Communication** |
| **Inputs** | Packets from MSP430 |
| **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 placed into the queue. |

**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. |