Recursion Co., Ltd.
Osamu Tamura

Circuit Celler DesignStellaris 2010

Camera-Based Eye-Tracking System

Osamu Tamura

Recursion Co., Ltd.

23 June 2010

Second Prize

This fascinating eye-tracking system was developed with an EKK-LM3S9B96 evaluation board and a USB camera. The Stellaris MCU works as a USB host and controls the camera. A button on the board is for switching the calibration/tracking modes. In Calibration mode, a captured eye image with detected markers is displayed on a computer screen. In the Tracking mode, gaze points are plotted on a white image. Results can be monitored by multiple PCs on a network.

Recursion Co., Ltd.
Osamu Tamura

Camera-Based Eye-Tracking System

1. About Eye Tracking

1-1. Intention

An eye tracking system measures the point of gaze by analyzing eye movement. It is used not only in medical research, but in many fields including commercial services (web usability, advertising, marketing, etc) [1]. The aims of this project are to develop the basic eye tracking system with the minimum cost and to acquire the requirements for the practical system, using Stellaris Cortex-M3 processor board.

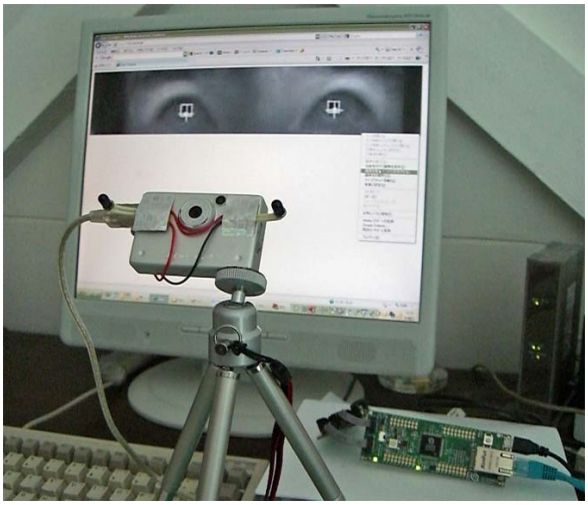

Photo 1: System Overview

1-2. Mechanism

Eye tracking systems typically use the relation between the corneal reflection, and the center of the pupil to identify gaze direction.

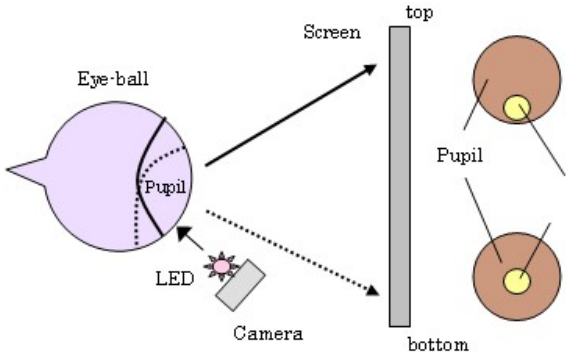

Figure 1: Tracking Eye Movement

The system uses infrared light to distinguish pupil shape from iris.
His irises are dark brown. The pupils become clear in infrared vision (photo 2 and 3).

Photo 2: Normal Vision

Photo3: Infrared Vision

A toy camera has been altered to capture infrared vision. Two infrared LEDs attached have sharp directional beams.

Photo 4: Infrared Camera

1-3. Advantages
The system is implemented on Stellaris processor board. The stand alone type does not consume the processing power of a user PC. The tracking result can be monitored by multiple PCs on network.

2. System Architecture

2-1. Schematic

This is the entire schematic of this system. The USB toy camera has STMicro's STV0680B chipset. The EKK-LM3S9B96 evaluation board was used without external expansion. The ICDI board and PC are necessary to know IP address. Otherwise, use AC/DC5V adapter to provide system power.

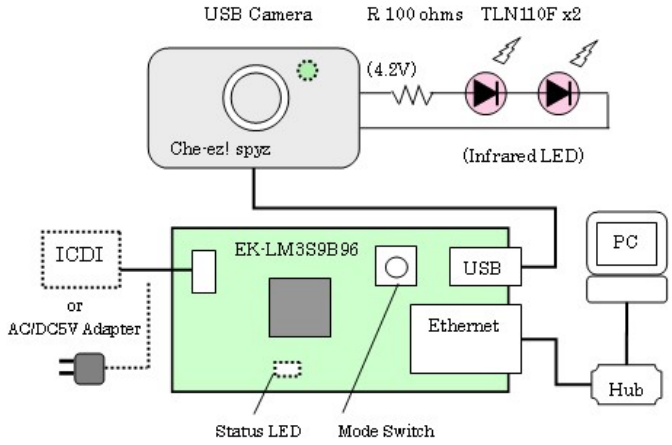

Figure 2: Schematic

2-2. Block Diagram

The Stellaris MCU works as a USB host, and controls the camera. The frame speed is rather slow because of the STV0680B commands response (IMAGE_GRAB and SDRAM_UPLOAD).

The user-button on the board switches the calibration/tracking modes. In the calibration mode, the captured eye image with detected markers is displayed on the browser screen of web clients. In the tracking mode, gaze points are plotted on a white image.

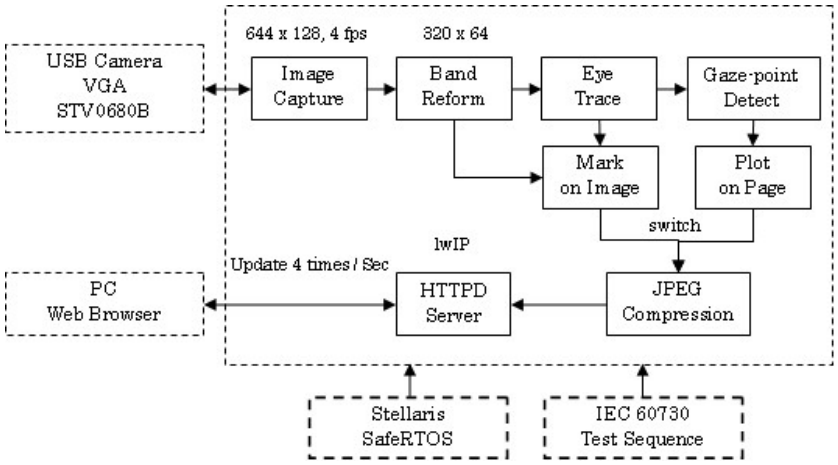

Figure3: Block Diagram

2-3. Infrared Camera

The camera used here is "Che-ez! spyz". This is not available now, but many toy cameras have the same chip (STV0680B) inside. Use the camera of which the VGA frame is supported. Since invisible rays are blocked before reaching to the image sensor, it is necessary to replace a square pink glass with an infrared-pass/visible-cut filter. I used a piece of infrared sheet. You can attach an infrared lens, instead.

Photo 5: Altered Camera

2-4. Image Memory Control

STV0680B chipset has a command to retrieve the image data on SDRAM buffer, which is formatted in 644x482 pixels of the Bayer strip lines. This system uses R pixel portion of the middle band (320x64 pixels). To save the MCU memory (96KB) and the time, the SDRAM image is transferred in two steps.

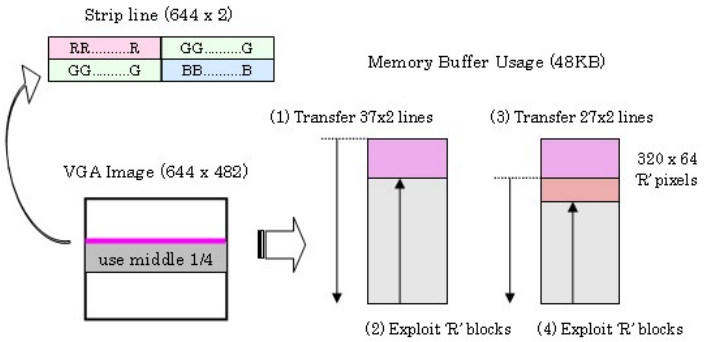

Figure 4: Memory Usage

2-5. Gaze Point Detection

To separate corneal reflection and pupil in image, threshold levels are determined based on histograms of the captured image. This program finds the bounding rectangles and centers of both corneal reflection and pupil to calculate the gazing point (trace.c).

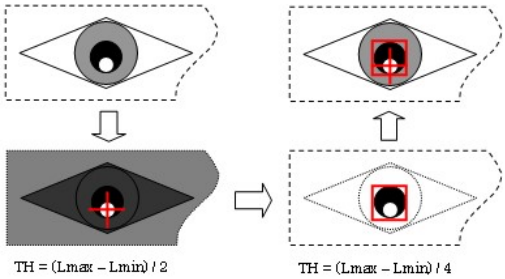

Figure 5: Eye Marking

3. Implementation

3-1. USB Host Control

The STV0680B camera is a vendor-specific class device [2][3][4]. It requires an alternate-interface to activate a bulk transfer. This function switches the interface.

The usb_control_msg() wraps USBHCDControlTransfer(), to be called by LibUsb style.

```
unsigned long usb_set_altinterface( unsigned int alternate )
{
    tUSBRequest SetupPacket;

    SetupPacket.bmRequestType =
                    USB_RTYPE_DIR_OUT|USB_RTYPE_STANDARD|USB_RTYPE_INTERFACE;
    SetupPacket.bRequest = USBREQ_SET_INTERFACE;
    SetupPacket.wValue = (unsigned short)alternate;
    SetupPacket.wIndex = 0;
    SetupPacket.wLength = 0;

    return USBHCDControlTransfer(0,
                            &SetupPacket,
                            pDevice->ulAddress,
                            NULL,
                            0,
                            pDevice->DeviceDescriptor.bMaxPacketSize0);
}

unsigned long usb_control_msg( unsigned char dir, unsigned char cmd,
    unsigned short value, unsigned short tlen, void *rbuf, unsigned short rlen )
{
    tUSBRequest SetupPacket;

    SetupPacket.bmRequestType = dir | USB_RTYPE_VENDOR | USB_RTYPE_DEVICE;
    SetupPacket.bRequest = cmd;
    SetupPacket.wValue = value;
    SetupPacket.wIndex = 0;
    SetupPacket.wLength = tlen;

    return USBHCDControlTransfer(0,
                            &SetupPacket,
                            pDevice->ulAddress,
                            (unsigned char *)rbuf,
                            rlen,
                            pDevice->DeviceDescriptor.bMaxPacketSize0);
}
```

List 1: USB Host Controll Functions

The code snippet of the image capture sequence is shown below.

```
void camera_start( unsigned long ulBulkInPipe )
{
        //      activate bulk transfer
        if( usb_set_altinterface(1)!=0 ) {      ...

        xTaskDelay( 2000 );

        //      clear all error and data
        if( usb_control_msg( USB_RTYPE_DIR_OUT, CMDID_CLEAR_COMMS_ERROR, ...

        //      set SDRAM entry pointers
        if( usb_control_msg( USB_RTYPE_DIR_OUT, CMDID_SET_IMAGE_INDEX, ...
        if( usb_control_msg( USB_RTYPE_DIR_IN, CMDID_GET_TABLE_ENTRY, ...

        //      initialize gaze point detector
        trace_init( data, w8, HGT );

        for( ;; ) {
                USBHCDMain();

                //      grab image
                if( usb_control_msg( USB_RTYPE_DIR_OUT, CMDID_GRAB_IMAGE, ...

                //      detect gaze point while processing the grab
                trace_gazepoint();

                //      wait until grabbing is done
                do {
                        ...
                } while( errinf.error==CAMERR_BUSY );

                //      retrieve band data (twice)
                for( i=0; i<HGN; i++ ) {
                        USBHCDMain();

                        //      request band data
                        if( usb_control_msg( USB_RTYPE_DIR_OUT, CMDID_UPLOAD_SDRAM, ...

                        //      obtain band data
                        if( USBHCDPipeRead( ulBulkInPipe, dptr, sdramex[i].size ) ...

                        //      retry transfer - it frequently fails if the data size
                        //      is not the multiples of a strip size (1288 bytes).
                        if( usb_control_msg( USB_RTYPE_DIR_IN, CMDID_GET_LAST_ERROR, ...
                                i--;
                                continue;
                        }

                        //      reform band data - exploit 'R' pixels
                        ...
                        //      mirroring - flip horizontally
                        ...
                }
        }
}
```

List 2: Image Capture Sequence

3-2. JPEG Compression

Without data buffering, it is unable to capture the next image and overwrite during the ethernet transmission. Since the RAM is already exhausted, image needs to be compressed before the transmission. The size of JPEG image is about 4K bytes.

The JPEG encoder was introduced from the "Embedded JPEG Codec Library" project [5]. I optimized the code for ARM processor and gray-scale image. These codes (jpeg.c, dct.c, and huffman.c) are not my original work, and are published under GPL license.

3-3. Network Interface

After the evaluation on uIP, I decided to use lwIP since the high speed TCP transaction is required for image transfer. The task of httpd here is to show and refresh images on client browser. "fs.c" generates a JavaScripted-HTML and transmits JPEG images four times per second.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
   <meta http-equiv="Content-Script-Type" content="text/javascript" />
   <title>Eye-Tracker</title>
   <script type="text/javascript">
   <!--
    function ReloadPage(){
     now = new Date();
     document.getElementById("imgEye").src =
       "eye.jpg?"+now.getTime()+now.getMilliseconds();
    }
    setInterval(ReloadPage,250);
   //-->
   </script>
  </head>
  <body>
      <img src="eye.jpg" width="98%" id="imgEye" style="position:absolute;
      top:6px;" alt="eye.jpg" />
  </body>
</html>
```

List 3: HTML Script to Refresh Image

3-4. SafeRTOS

The Stellaris SafeRTOS is very simple, but is enough for this application. There is not much to do to adapt into it, but I redefined some locally defined arrays globally to avoid stack-overflow.

3-5. IEC60730

IEC60730 libraries are linked to check CPU, I/O ports, ADC, and memory. The system is tested at the startup time, and is kept testing until camera is connected.

4. Result

4-1. Operation

The measurement should be done with special attention paid to lighting. Daylight prevents the proper marking on eyes. The head should be kept still during eye tracking. Make sure your eyes are marked in captured view. Switch to the plot mode, and gaze at a corner of the screen for a while. The plots will gradually move to that direction.

4-2. Marking

Under the appropriate light, the detected markers appear on eyes.

Photo 6: Markers on Eyes

4-3. Tracking

This is a sample plot gazing here and there (black area). There are some insensitive zones.

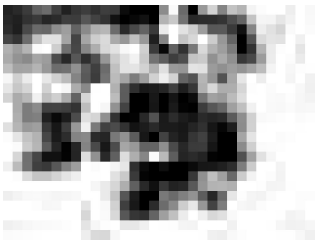

Photo 7: Eye Tracking

4-4. Code and Performance

The generated codes are mostly occupied by lwIP and other libraries.

	Code	**Data**
Eye tracking	1.8K	18K
USB control	1.7K	48K
JPEG encoder	1K	
Other tasks	2K	
lwIP library	30K	17K
Other library	16K	
Total	53K	88K

The system clock is 80MHz. It worked with 31MHz, with accessing from two Web clients without problems.

5. Conclusion

Although the eye tracking was barely achieved with this system, its performance falls far short of that of the commercial systems. Several improvements are needed to fit it for practical use. Some problems are:

(1) An interference with the environmental light, especially daylight
(2) An individual difference of human face, and glasses
(3) Unsettled head position
(4) Slow framing rate
(5) Rough resolution and co-ordinates conversion.

These are impressions through this project with Sterallis processor:

USB imaging devices are easily controlled without Linux and a huge memory.
The code tested on an x86 machine can be ported with minimum (almost no) modifications.

I could extend my knowledge to design a practical eye tracking system through this experiment.

References:

[1] http://en.wikipedia.org/wiki/Eye_tracking
[2] STV0680B+VV6410/6411/6500 Version 3.4, STMicroelectronics
[3] STV0680 Software Development Kit (SDK) User Manual
[4] http://sourceforge.net/projects/gphoto/
[5] http://sourceforge.net/projects/mb-jpeg/

Source Code:
[6] http://www.recursion.jp/prose/eyetrack/eyetrack_src.zip
Video Clip:
[7] http://www.recursion.jp/prose/eyetrack/eyetrack.wmv