Yet Another Human Gesture Controller

Final Design Report

ECE532 – Digital Systems Design

Enoch Lam 995302603
Xihao Li 996382436
Mushfiq Mahmood 996138674

4/3/2011

Overview

Project Description

The team designed and implemented a real-time gesture recognition system, named Yet Another Human Gesture Controller or YAHGEC for short, which serves as a remote control to an external device, such as a personal computer. The system matches gestures to a database of pre-defined gestures and issue a specific control command to the end device. The main motivation for this project comes from a desire to control applications on the computer without being physically at the computer, such as decreasing the volume of music played while on bed through a simple swipe of hand across the camera view. YAHGC can be extended to work with TVs for homemakers in the kitchen, such that the remote control can be deprecated.

This project is inspired from two existing products: the Xbox Kinect and a software called StrokeIt, which recognizes mouse gestures on a computer. Xbox Kinect uses advanced range-camera technologies combined with infrared structured light to capture 3D motion and facial recognition from a person. While the team may have used Kinect as a basis for YAHGC, the team decided to use simple image-processing for a well-defined cursor (such as a red cap of a marker) captured from a video camera.

Original Vision

YAHGC's initial objectives were to recognize a shape-based cursor such as a hand, and improve its recognition ability by employing machine learning on the reference gestures. In addition, the system was to be capable of learning new gestures at runtime.

YAHGC had 5 modules: gesture parsing module, gesture recognition and learning module, remote device I/O module, system commander module, and debugging module. From the video camera, the gestures recognition module was to take in a real-time video feed and look for cursors in video frames. It would then compose a gesture path consisting of each individual cursor points and use this to initialize the gesture buffer in memory. The gesture recognition module would then use a neural network algorithm to match the gesture in memory and compare it with reference gestures. The algorithm in YAHGC would use a two-layer perceptron network—the first layer consists of individual neurons representing a reference gesture buffer, and the second layer which would consist of a single neuron for composing the outputs from the layer one neurons to decide on the best match. Based on the output from the neural network a signal would be sent to the UART interface through remote I/O module. A program on the computer would listen on the USB port to which the board is connected and run appropriate commands based on signals from the board.

Project Goals and System Specifications

Due to various implementation problems and complexity in the original vision, the team made some design changes to address issues with memory access speed and complexity in shape detection that results in long processing time. The details of differences between the original design specification and the actual implementation are outlined in Overview sub-section of Outcome section.

Terminology

Term	Definition
Gesture	A pattern drawn using a cursor in the real world within the video camera's field of view
Cursor	A small object in the real world which can be freely moved by a person in order to draw a pattern
Gesture Frame	A flattened sequence of cursors that has been connected into a gesture path
Buffer	A location in BRAM that stores a gesture frame

Inputs

YAHGC's main input is a real-time video feed of gestures, which are buffered and queued for processing.

Outputs

YAHGC's main output is a serial port from the Xilinx board, which connects to a desktop computer via a serial to USB converter. The output signal is processed through a program on the remote PC that runs a pre-mapped command. For debugging purposes, augmented video can be streamed to a VGA monitor and consist of overlaid information including cursor location and gesture buffer contents.

Constraints

There are several restrictions to the types of gesture that can be detected. A gesture should:

- Be recognized by the system as a 2D pattern within the camera's field of view,
- Be drawn parallel to the camera's plane of view since perpendicular motions present an interpretation challenge,
- Range from a simple line to a series of directional vectors that may vary in size, and
- Be assumed fully connected (one stroke).

Functional Requirements

YAHGC meets the following functional requirements, listed in order of importance

- System must function in "real-time" during normal operations. Normal operation is defined as the process of recognizing gestures from the video stream and outputting keyboard/mouse events to the remote PC. Real-time means there should be a less than 1.5s latency between the completion of a gesture and its corresponding event being sent to the remote PC.
- The gestures database must be scalable. This means there should be some way to expand the database out of hardware (if implemented in hardware) onto memory. An efficient compression schema for the database should be employed to ensure robustness of total memory fetch times. The database must be able to store more than one gesture for usability, preferably more than a hundred.

- Video input should not be a system constraint, and resolution up to 720p can be scanned into video processing module.
- The cursor in each frame can range from a simple low-variance color cursor to a more advanced shape cursor. There should only be a maximum of one cursor in each frame, i.e. multiple, simultaneous cursors are not expected.
- The resolution of the gesture frames stored in database is reasonable to allow detection of polygonal gestures (with 180 degree turns and curvatures).
- The system can send simple I/O signals to the computer that will be processed by a program on the computer to execute signal-mapped commands.
- Gestures can be detected from anywhere within the range of the camera, even in corners of the camera vision where lighting conditions will vary the most from the center of the camera vision.
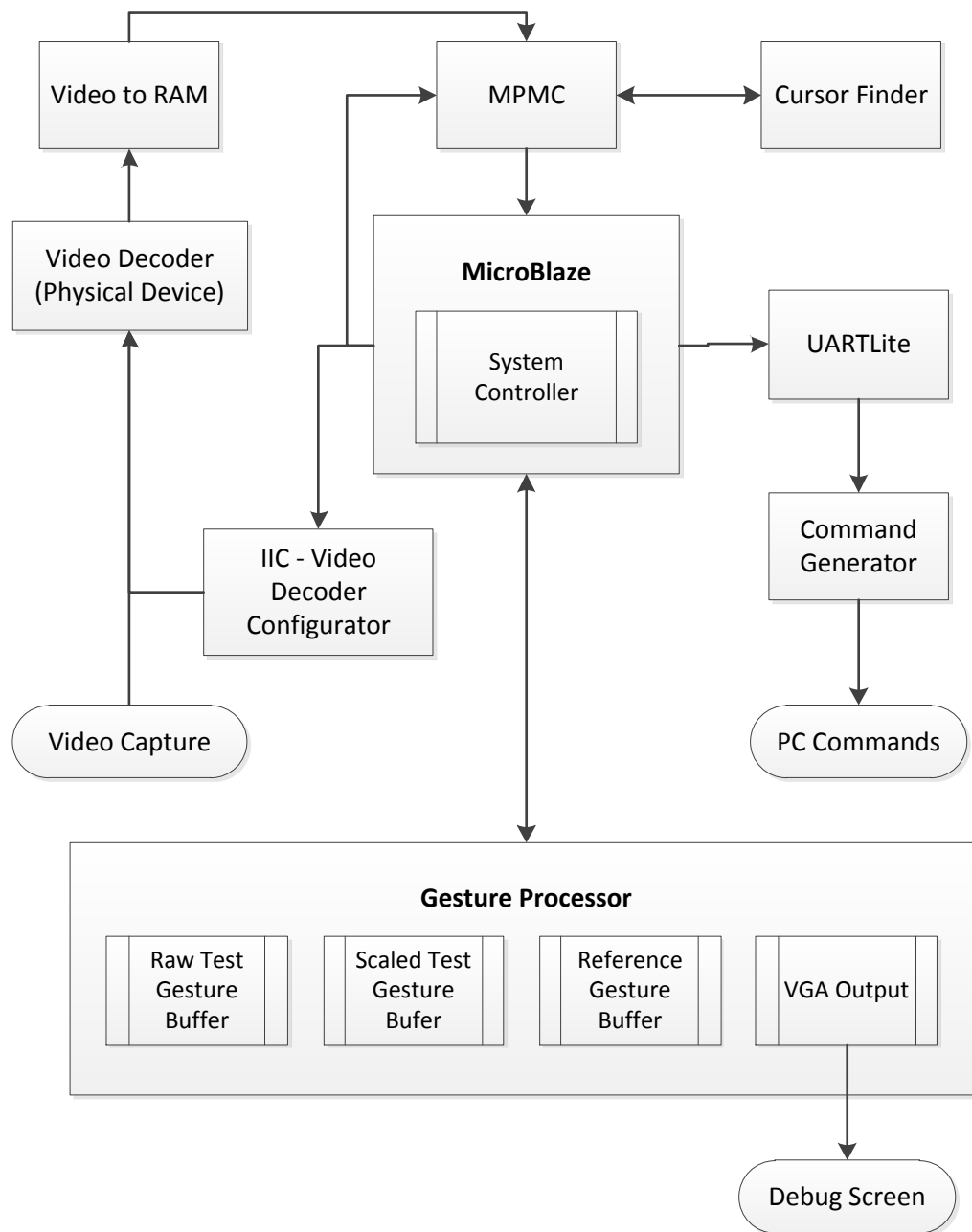
System Modules

Modules	Description	Type	IP
Video to RAM	Transfers real-time video frames to memory	Hardware	Existing
Cursor Finder	Detects a cursor in each video frame a constructs a gesture frame	Hardware	Modified
Gesture Processor	Parses gesture frame and interfaces with System Controller module	Hardware	New

Sub-modules	Description	IP
Raw Test Gesture Buffer	Grabs raw 640 x 480 x 1 bit raw test gesture	New
Scaled Test Gesture Buffer	Processes raw test gesture and outputs a 256 x 256 x 1 bit scaled test gesture buffer	New
Reference Gesture Buffer	Grabs 256 x 256 x 8 bit reference frame from memory	New
VGA	Outputs scaled gesture buffer and reference gesture buffer	New

MPMC	RAM controller	Hardware	Existing
IIC	Configures video decoder	Hardware	Existing
UartLite	Interface for sending data from MicroBlaze processor to the UART port via serial-to-USB cable	Hardware	Existing
MicroBlaze	A multi-purpose processor that controls the system	Hardware	Existing

System Controller	A program that runs on MicroBlaze to control the entire system (generating reference gestures, learning new reference gestures, interfacing other modules)	Software	New
Command Generator	Program on the remote PC that processes USB data from MicroBlaze and executes scripts or commands on the computer.	Software	New

Block Diagram

Video to RAM

MPMC

Cursor Finder

Video Decoder (Physical Device)

MicroBlaze

System Controller

UARTLite

IIC - Video Decoder Configurator

Command Generator

Video Capture

PC Commands

Gesture Processor

Raw Test Gesture Buffer

Scaled Test Gesture Bufer

Reference Gesture Buffer

VGA Output

Debug Screen

Outcome

Overview

As previously mention, some aspects of the original design were modified to make our system more practical and suitable on an FPGA with limited hardware resources. The differences are as summarized below:

Original Feature	Current Feature	Reason
Neural network algorithm for gesture comparison	Pixel by pixel comparison with probability summation for each reference frame comparison	Impractical to implement a full neural network given FPGA resources and memory bandwidth. The current comparison unit is in essence a serialized neutral network.
Reference gesture frames stored in 640 x 480 resolution	Reference gesture frames stored in 256 x 256 resolution	Memory access speed is limited. Reduction of frame size is required to meet system latency requirements of 1.5s. Given the imprecise nature of the problem the loss of resolution doesn't seem to have a noticeable impact on performance.
Shape + colour based cursor recognition	Only colour based cursor recognition	Shape-based recognition requires complex algorithms, simple colour based recognition was found to be suitable for our needs.
Adjust the probabilities in reference gesture based on the matched test gesture (feedback-based learning)	No feedback implemented	Required significant modifications to how reference gestures are stored. We elected to drop this feature in favour of improving recognition accuracy.

Overall, most of original goals were met with a few minor changes to the algorithms required for practical implementation. YAHGC can do both cursor recognition and comparison between multiple reference gestures in real-time and produces results with an average latency of less than 0.5s.

Future Improvements

Potential features that can be added onto YAHGC include:

- More complex, shape-based cursor recognition
  - Verify the initial cursor location by examining pixels clustered around the cursor based on a pre-specified minimum cursor size

- Use YCrCb instead of RGB when processing video input for Cursor Finder module, which helps avoid problems associated with varying lighting conditions (in YCrCb the colour channels are separate from the luminous channel)
- Hardware acceleration of some software features in System Controller module, such as reference gesture generation and reference gesture loading
    - This allows burst writes to the reference gesture buffer in Gesture Parser module
- Pipelined system that can parse a reference gesture against a test gesture as the reference gesture is loaded
    - This accelerates the processing time and circumvents limits imposed by memory access speed and overall system lag introduced as the database of reference gestures increases in size.

Next Steps

Future improvements mentioned above can be implemented in the following sequence:

1. Feedback-based learning should be incorporated as a new hardware module that calculates probabilities surrounding the test gesture, and superimpose the probabilities onto the reference gesture probabilities using Naïve Bayes algorithm. Current Gaussian probabilities in reference gesture frames help account for slight variations in test gesture, but are unable to differentiate slight variations between gestures if they are similar. For instance, a backward slash with a slight upward slope at the bottom and a backward slash only gesture may be found identical under the current comparison approach. Instead, differentiation can be improved by tightening the probabilities spread around the test gesture. Variation is then accommodated by learning the new probabilities from the test gesture.

2. The software reference gesture-loading function should be replaced by a hardware module. This can take advantage of MPMC burst reads to quick load reference gestures from the SDRAM to the reference gesture buffer in the comparison module. Implementing this however would require modifications to the reference gesture buffer as currently it doesn't support burst writes to it.

3. More sophisticated cursor recognition can be implemented by adding shape recognition and cursor verification scheme. Verification of cursor improves the robustness of the system as it allows a cursor to be recognized in noisy-backgrounds that consist of many shapes and colors.

4. Reference gestures are be loaded via an external, in-volatile medium such as SD/CF cards or direct transfer from HDD/remote PC over Ethernet. Currently, a set of reference gestures are generated on system startup, and an addition of a reference gesture would require modification to the MicroBlaze System Controller module source.

5. Newly learned reference gestures should be saved onto the reference gesture library in an external, in-volatile medium as they are added.

Detailed Project Description

Design: Algorithms

Frontend: Cursor Position Detection

- The Video to RAM module obtains video frames from the physical video decoder and stores it into a predefined location in SDRAM. As soon as the memory location has been initialized completely for one video frame, a ready signal is set in SDRAM. The Cursor Finder module polls this ready signal, and processes the raw video frame in SDRAM when the signal is high.
- For each pixel in the video frame, Cursor Finder scans for the predefined RGB-based cursor within a range of RGB values, and upon detection, spits out the coordinates into an array in SDRAM-mapped location. These coordinates are then extracted by the System Controller on MicroBlaze, and a gesture path is constructed and written to a test gesture buffer in SDRAM.
- If the distance between consecutive points exceeds a certain distance threshold, the new location is rejected as noise, although the current location is updated as a running sum to include the rejected location.
- After the cursor has been stationary or lost for 0.2 seconds, the System Controller will make a decision on whether the cursor is a valid gesture or noise. This is based upon the total number of pixel locations in the gesture memory.
- If a valid gesture is believed to be found, the process command is sent to the Gesture Processor. Otherwise, the gesture memory buffer clear signal is sent.

Backend: Gesture Processing and Comparison

The hardware contains buffers for the gesture input and gesture database data. The system controller loads these gesture buffers and sends various commands directing this module in its task.

Test Gesture Scaling

Since all database gestures are of fixed size, the input image must first be normalized to the same size. Horizontal and vertical offsets to the output must also be accounted for due to the length to width ratios of the input gesture. The exact algorithm is as follows:

```
// Determine offset and scale ratio
// BUFFER_SIZE = size of square target buffer to map to
length      = max_left - max_right;
width       = max_up - max_down;
if ( length > width )
{
    horizontal_offset = 0
    vertical_offset   = ( length - width ) * BUFFER_SIZE / 2
```

```
        }
        else
        {
            horizontal_offset = ( width – length ) * BUFFER_SIZE / 2
            vertical_offset   = 0
        }
        // Normalize input gesture to target gesture-loading
        for ( /*each element in input gesture*/ )
        {
            x = input_gesture_x
            y = input_gesture_y
            divisor = /* length if ( length > width ), width if ( width >
                        length )*/
            x_normalized = horizontal_offset + ( input_gesture_x *
                                    BUFFER_SIZE ) / divisor
          y_normalized = vertical_offset + ( input_gesture_y * BUFFER_SIZE )
                                    / divisor
        }
```

Reference Gesture Comparison

Once the scaled and oriented input is available, the software populates the gesture database buffer with a target gesture that utilizes a Gaussian function to create a probability area based upon the pixel's closest distance to a gesture pixel. This is performed in the System Controller and may be considered a major bottleneck of the end design. The algorithm goes through pixel-by-pixel in a loop and sums up the Gaussian probabilities for each reference gesture based on the test gesture on/off bits. This would give a very accurate result of how well the test gesture matches up with a particular reference test gesture. System Controller cycles through each database gesture and controls the Gesture Processor interface where the following algorithm is performed:

```
    // Assume that the previous scaling step has been completed:
    number_of_pixles = 0
    total = 0
    for ( /*each pixle in the scaled gesture and the database gesture (same
dimensions )*/ )
    {
        if ( scaled_gesture[i] == 1 )
        {
            number_of_pixles += 1
            total = Gaussian_distribution_database_gesture + total
        }
    }
    return ( total / number_of_pixles )
```

There are two major drawbacks from the above algorithm. The first being that any noise during the gesture memory input stage outside the gesture space will generate an inaccurate offset that will create false negative correlations. Other being that the algorithm determines how well the input gesture fits into the database gesture's probability space, where a partial database gesture will still return large correlations.

One major reason why a full pixel-by-pixel comparison was used instead of a short, partial comparison based, neural network approach was that a complete pixel-by-pixel check against a reference gesture is necessary to find a match. For instance, if there are two reference gestures, "\"-shaped and "V"-shaped, a test gesture "V" may complete a match with "\" reference gesture before it completes with "V" gesture, which would give a wrong match result. By using the full frame comparison, that problem is avoided.

Once the data for all database gestures have been returned and stored within the System Controller sequentially, the database gesture with the highest returning probability is selected as the input gesture and the corresponding actions performed.

Reference Gestures Generation

On a process command from System Controller to Gesture Processor (after constructing a test gesture), System Controller module initializes the SDRAM with a set of reference gestures. Each reference gesture has a custom function that generates the gesture by looping for x and y coordinates, and initializes each f(x) output and its surrounding pixels with Gaussian probabilities. The Gaussian probabilities are obtained from a Gaussian LUT function that maps a delta (distance between the current pixel and f(x)) to an 8-bit integer. The higher the integer is, the higher the probability. The following code shows the Gaussian LUT function:

```
// Gaussian distribution LUT
int gaussian( int delta )
{
    if ( delta < 1 )
      return 255;
    else if ( delta < 2 )
      return 250;
    else if ( delta < 3 )
      return 245;
    else if ( delta < 5 )
      return 230;
    else if ( delta < 7 )
      return 215;
    else if ( delta < 9 )
      return 190;
    else if ( delta < 11 )
      return 170;
    else if ( delta < 14 )
```

```
              return 150;
          else if ( delta < 17 )
              return 125;
          else if ( delta < 20 )
              return 100;
          else if ( delta < 24 )
              return 90;
          else if ( delta < 28 )
              return 80;
          else if ( delta < 33 )
              return 70;
          else if ( delta < 38 )
              return 60;
          else if ( delta < 45 )
              return 50;
          else if ( delta < 52 )
              return 40;
          else
              return 0;
      }
```

Remote PC Commands Generation

When the Gesture Processor completes comparison, a signal is sent to the System Controller via a PLB. The System Controller, with the knowledge of which gesture was matched with, sends the ID of the reference gesture to the Remote PC via UART.

The Remote PC will run a software module called Commands Generator, which constantly polls the USB port containing the UART signal. When a signal is received, it looks up to its database of mapped actions, and executes a preset bash script / keyboard macros.

Implementation: Description of Blocks

Gesture Processor (comparison_module version 1.00.a)

The comparison_module ip core is a new hardware component for this project and is responsible for the isolation the input gesture from the rest of the raw test gesture buffer and comparing it against reference gestures. For the purpose of increasing data access rate, all gesture memory for the immediate computation is store as internal bram blocks which allows for synchronous and one cycle latency data access. Ignoring debug ports, the module has four main input/output ports:

1. SYSTEM_CLOCK – The main clock for driving all logic and memory access. Driven by 100 Mhz plb clock for synchronous behavior.
2. VGA_CLOCK – The clock for driving the vga logic, and the seconds ports of the internal bram dual port memory blocks.

3. data_in[31:0] - 32 bit generic input signals, connected to PLB bus. All supported commands are translated into instructions via a op-code like system.
4. data_out[31:0] – 32 bit generic output signal, connected to PLB bus. 8 bits reserved for return status, 24 bits reserved for data. Allows memory writes to location for acknowledgment command.

The comparison_module core is memory mapped as a PLB slave. All communications with the MicroBlaze is done via memory writes and reads.

The comparison_module contains the following sub modules:

1. One 640 by 480 1 bit dual port bram – input cursor history buffer.
2. One 256 by 256 1 bit dual port bram – input normalized cursor gesture buffer.
3. One 256 by 256 8 bit dual port bram – database gesture distribution buffer.
4. One vga_controller custom module – debug module also used as user interface, allows for viewing of all three memory buffers.

The comparison_module supports the following functionalities which can be selectively used by the System Controller:

1. Direct write of 1 pixel to any of the three memory buffers; no acknowledgment required.
2. Direct read of 1 pixel from any of the three memory buffers; acknowledgment required.
3. Hardware cleaning to all three buffers; acknowledgment required.
4. Initiate main scaling and calculation algorithm, acknowledgment required. (see algorithm section)

Video to RAM Module (video_to_ram version 1.00.a)

This module interfaces with the video decoder and pulls raw video frames from the decoder into SDRAM. Other modules (such as Cursor Finder) that require reads on the raw video frame would then read directly from SDRAM for a video frame. This module is an existing IP core.

Input: nothing, **Output**: video frame in a specific address range in SDRAM

The Video to SDRAM block has a ready signal byte in memory that a module can poll to ensure the memory contents has been fully initialize with a complete video frame.

Cursor Finder (version 1.00.a)

The Cursor Finder module is implemented in hardware, modified from an existing core. This module locates an RGB-based cursor within a specified RGB range in a video frame. Cursor Finder scans the video frames buffered in SDRAM and attempts to find the pre-defined cursor (a slightly dark red cursor) based on parameters configured by the System Controller (by writing them to known locations in memory). If successful, it will write the coordinates of the cursor into an known location in SDRAM, which the System Controller reads and passes on to the comparison module. If unsuccessful, the previous successful coordinate stays in the output slot in SDRAM, and no change is made to memory.

Input: video frame from SDRAM

Output: coordinate of the cursor found, written to SDRAM

The existing core modified to create this module is part of the Virtual Pong project from 2010 (can be found here: http://www.eecg.toronto.edu/~pc/courses/432/2010/projects/virtualpong.zip). The original core was called "paddle_detector". The original core essentially looked for 4 cursors, we only require one cursor.

IIC (xps_iic version 2.00.a)

An existing Xilinx hardware module used to for communication to off-chip components. Used in our project to configure the video decoder on the VDEC-1 card.

Input: takes in configuration parameters from System Controller on system initialization

Output: Communicates configuration to VDEC-1 Card.

MPMC (mpmc version 4.03.a)

The multi-port memory controller is an existing Xilinx hardware module provides an interface to perform I/O operations to SDRAM. When used with our board, changes to the system UCF was required (details can be found here: http://www.eecg.toronto.edu/~pc/courses/edk/doc/512MBfix.txt) as our board had a 515 MB memory stick.

Input/Output: SDRAM reads and writes.

UartLite (xps_uartlite version 1.00.a)

UartLite is an existing hardware module that interfaces the serial-to-USB cable for communication with a remote device over UART terminal. This module allows System Controller to communicate data to the remote PC's Commands Generator module. It has both a receive and transmit channel that operates on a FIFO 16-character queue. Control signals on its registers indicate the status of transmit/receive FIFO queues, and are read by the System Controller to control the flow of information to the remote PC.

Input: Receive Data FIFO, 16-bit register

Output: Transmit Data FIFO, 16-bit register

MicroBlaze (microblaze version 7.10.d)

A Xilinx hardware module that implements a general-purpose soft processor. It is used in this project to run the System Controller which is written in C.

System Controller (software, running on MicroBlaze)

This is the controller software that oversees the entire system. It is a new software module implemented in the MicroBlaze. Its responsibilities are:

- Configuring the video decoder
- Configuring the cursor finder's parameters (colour ranges)

- Reading output from the cursor finder module and transferring cursor coordinates to raw gesture buffer in the comparison module (this is done in a tight loop faster than the cursor finder module can produce new coordinates)
- Detecting the end of a gesture (by checking if the cursor location hasn't changed for a specified period of time) and signaling the comparison module to begin processing at the end of a gesture
- Loading reference gestures into the comparison module buffers
- Clearing the comparison module buffers at the end of a comparison
- Interpreting results from the comparison module and sending gesture ID to Commands Generator over the UART connection

Commands Generator (software, running on remote PC)

This is a new software module that runs on the remote PC. It monitors the USB port to which the board is connected for signals from the System Controller. The signals received indicate the ID of a reference gesture. This ID is then looked up in a table to find the corresponding action. Currently the lookup table contains shell scripts which are executed on a match. This module makes use of Linux syscalls (to communicate with the USB port) and thus will only work on POSIX systems.

Input: matched gesture ID

Output: execution of matched script

Design Tree

The project tree is rooted at the system's EDK project root directory. Here is a list of important directories and contents they contain:

- docs: Contains a copy of this design report.
- pcores: Contains non-Xilinx ip cores. In particular, contains our custom cores: the comparsion_module and cursor_finder.
  - pcores/comparison_module/implementation: Contains the pre-synthesized hardware modules (.ngc files) requires by the comparison_module. These should be placed in the implementation directory in the root of the EDK project directory prior to synthesizing the project.
- YAHGC_SW/src: Contains the source for the system controller module running on the MicroBlaze.
- commandgen: Contains the source for the Commands Generator module

Appendix

Project Schedule

Week 1 (Feb 10)
- Video buffering in memory complete
- System skeleton code complete
    - Skeleton contains empty modules with connections

Week 2 (Feb 17)
- Debugging video output framework complete
    - fully functional video buffering process, which takes video from camera, buffers it as frames in memory which can be operated on and allows writing to the video
    - frame to output debugging information to VGA monitor
- If not completed by this point, abandon debugging video output, fall back to other debugging methods

Week 3 (Feb 24)
- Pixel processing unit for gesture comparison sub-module completed
    - Given two pixels in a gesture buffer, can compute the probability of gesture pixel matching reference pixel
    - Should begin planning on how to chain them together
- Gesture buffer design complete

Week 4 (March 3)
- Added test gesture scaling sub-module into Gesture Processor
- Started integrating the Gesture Processor sub-modules
- VGA output now shows both scaled test gesture buffer and reference gesture buffer

Week 5 (March 10)
- Gesture Processor module complete and correctly isolates and scales gestures to reference gesture sizes
    - Pixel-by-pixel comparison and probability summing works well
    - Reference gestures and test gestures are hard-coded into the reference buffer
- Start on Cursor Finder module to process video frames from memory

Week 6 (March 17)
- System Controller now has functions to generate a full set of reference gestures
- Cursor Finder module is completed and tested
    - This means for every frame in video buffer, we can get a coordinate which corresponds to the centre of the cursor
    - No more hacks where we hard code test gestures into gesture buffer

- Start writing function to send signals to UART on System Controller

Week 7 (March 24)
- Start integrating Cursor Finder, System Controller and Gesture Processor via signals on PLB and MPMC accesses
- Start writing the Commands Generator module

Week 8 (March 30)
- Integration completed (Cursor Finder, System Controller, and Gesture Processor)
- A gesture from video input can be detected and matched to a correct reference gesture.
- Starts adding a function in System Controller to learn new gestures and add them into the reference gesture database.

Week 9 (April 6)
- Integration between Commands Generator and YAHGC completed
- Demo Day, full system functional