Project Proposal: Touch Projector

Hope Harrison
Alan Cheng

30 October 2017

Summary

The touch projector allows users to interact with any projected image as if it were a touchscreen. An example would be allowing teachers to give presentations (including demos) without needing to walk back and forth from their computers. The touch projector will consist of a projector, Kinect, and a FPGA. The FPGA acts as a HID-compliant mouse which can plugged into any arbitrary computer. Thus, users can simply touch the surface they are projecting onto, and the computer will receive a click at the point they touched. This project will be divided into three main modules (*Figure 1*):

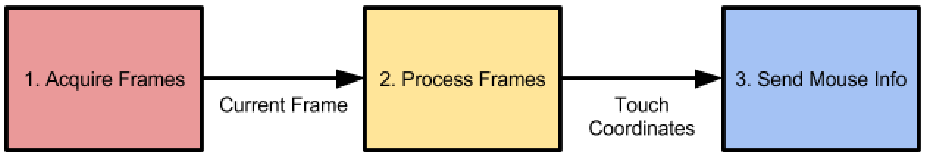

Figure 1: Process breakdown for project

1. Acquire Frames: Kinect acquires depth-image frames and sends them to the FPGA.
2. Process Frames: FPGA processes image frames to calculate the coordinates where the user is touching the surface. Thresholding, noise reduction, and edge detection is needed.
3. Send Mouse Info: Coordinates are transformed into an x and y translation for the mouse and sent to the computer as a HID-compliant mouse signal.

1 Acquiring Frames

In order to detect finger contact with a surface, the distance of a finger is needed. We could calibrate to detect fingers blobs of a certain size, but it is

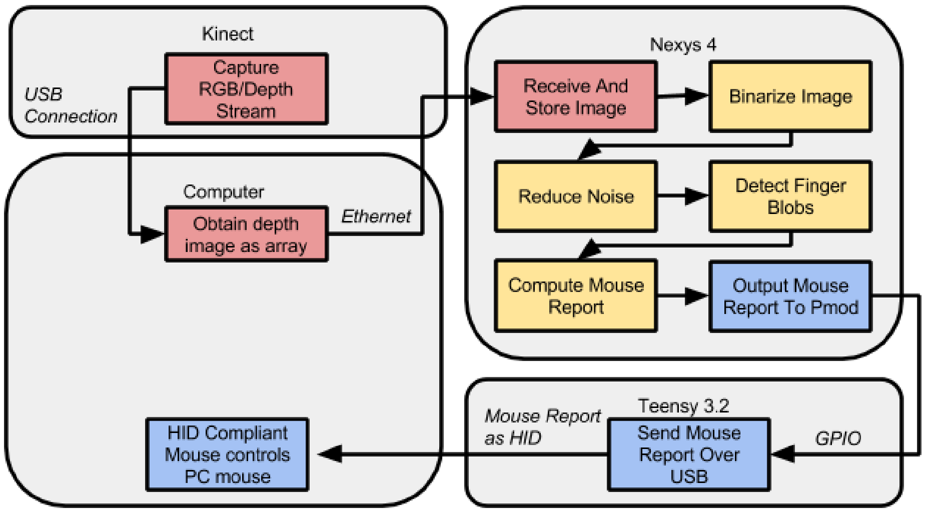

Figure 2. System architecture level design of project

easier to directly use a depth image. Thus, we use the Kinect to extract the current frame's depth image to do processing on. Because it would be difficult to connect the Kinect directly to the Nexys board, and since a computer will be connected to the Nexys, we plan to use existing drivers. Specifically, we anticipate using Microsoft's Kinect SDK coupled with OpenCV to acquire the depth image stream. Then we will send each frame to the Nexys over an ethernet connection.

This module will be tested by having the FPGA display the current frame it is receiving on a monitor over VGA. We can then make sure that the Nexys is receiving the frame correctly. This module will mainly be worked on by Alan.

2 Processing Frames

The main application of the FPGA to this project is to provide high speed image processing, ideally targeting 60 frames per second, but realistically targeting 30 frames per second. Previous work on this idea in software by Alan suffered from many performance issues, and so this FPGA implementation targets those issues.

Given a depth image frame from the computer via ethernet, we must construct an efficient representation of image data, as well as an algorithm to detect finger touch. Depending on the amount of available memory, we can construct a 2D grid structure of modules, wherein each module represents (ideally) a pixel or (realistically) a cluster of pixels, and where each module can communicate with neighboring pixels. However, depending on complexity of the finger touch detection algorithm, we may have to simplify the image representation.

We can generalize the image processing to the following subsections:

Binarizing the Image
Image binarization is provided by image thresholding. If we were to update each pixel individually, a single frame of 1080p resolution would take 2073600 clock cycles to update. However, if we parallelize the image representation as with the proposed 2D grid, we can reduce the operation time by a large magnitude. Specifically, the complexity is reduced to the number of pixels represented by one cell in the grid.

Reducing Noise in the Image
We plan to subtract the current frame (received from computer) from the previous frame (stored from last sample) to filter out much of the noise in the image. In our application, a finger touch takes multiple frames to occur, so subtraction can remove noise. Complexity of subtraction depends on both data access times for the stored frame, as well as the image representation. We can also apply kernels, such as those taught in lecture, or a gaussian blur to reduce grainy noise in the image.

Detecting Finger Blobs
The last subsection requires edge detection. Given a filtered image, we can represent blobs with chain codes. A 2D grid model would allow us to see values of neighboring pixels, and the chain codes can be calculated from there. We could also opt to use moments or any other centroid finding algorithm of blobs to direct calculate the midpoint.

For debugging and testing of the image processing algorithms, we will include switches which will allow the image in between any of these stages of processing to be viewed on the monitor. In this way, we will be able to see, for example, what blob is being detected and make sure it is right. The image processing will be worked on together by Alan and Hope.

3 Sending Mouse Info

In order to control the computer by touching the projection, we must have our device appear as an HID mouse to the computer. Since the Nexys 4 is only able to function as a USB host (but not slave), we are not able to output directly from the Nexys 4 to the computer. Therefore, we will use a Teensy as an intermediate.

See *Figure 2* for a depiction of the following processes:

Computing Mouse Report
Within the Nexys, this module will receive the coordinates of the currently tracked position of the users finger, as well as whether they are touching the surface. The module keeps track of the last coordinates and computes the difference between the two, which will be the x- and y-translation values. Currently, we will only send a left click button if the user is touching the surface, and no button press otherwise, but we could expand the project if we have time to also send right clicks for certain gestures or other options.

Outputting Mouse Report (Pmod)
The Nexys will also have a module that takes in the buttons, x-translation, and

y-translation and prepares it to send to the Teensy. Each signal will be output to a pmod port, along with a shared clock to indicate when new data is ready. We will use a circuit board to wire between the pmod of the Nexys and the digital I/O lines of the Teensy.

Sending Mouse Report (USB)
The Teensy will be programmed to read the data from its digital inputs on the ready clock. Then it will send mouse click() and move() commands to the computer over USB.

This module will also have a debug mode, so that it can be tested on its own, without relying on other modules working. When in debug mode, it will use the left, right, up, down, and enter buttons on the Nexys as the mouse movement and clicking, rather than taking it from the image processing. In this way, we can make sure that the code for outputting the mouse report data to the Teensy as well as the code for sending the mouse moves and clicks in the Teensy is working correctly. This module will be worked on mainly by Hope.

Performance/Hardware Limitations

A major point of concern is with the data transfer rate between PC and FPGA, and vica-versa. A stable ethernet connection should allow us a fast data transfer rate with gigabit transfer rates. This should be sufficient for our application, but if we had to opt for a slower communication method, then we may run into a performance bottleneck.

Another major point of concern is the image representation in hardware. Ideally, we could use a highly parallelized representation that allows us to process multiple pixels at the same time to reduce runtime (as compared to software). If we are limited to calculating pixel by pixel one at a time, we negate one of the biggest benefits of using an FPGA, and may run into a performance bottleneck. This would also require us to reduce the quality of image processing techniques, such as in filtering or edge detection, which may lead to much worse in quality results.