Virtual Rock Climbing: A video game using tracking and tactile feedback

Turner Bohlen Chris Lang

December 11, 2013

1 Introduction

This project aimed to create a rock climbing video game in which the player climbs a virtual wall by making motions in the real world, and it achieved its goal. The player is able to reach up, down, left, and right in reality in order to find and grasp virtual handholds. If the player lets go of all handholds, they fall back to the beginning. When the player reaches the top of the wall, the game is over and the movement stops until reset. The users primary input to the game is a set of colored gloves, equipped with haptic feedback motors, and flex sensors. Each glove is separately colored to enable hand tracking by a video camera. The flex sensors on the glove are used to sense when the player makes a physical grabbing motion that is then replicated in the game. A force feedback system was designed and implemented, however it caused a drop in power supply voltage, making the flex sensors unreliable when the motors vibrate. Because the motors were used to indicate that a hold could be grabbed, it was considered too large of a drawback, and not enabled in the final design. Additionally, the stretch goal of including a map editor was partially implemented, however movement within the editor causes glitches.

2 High-Level Design

The system has three major components: the tracking block, the gameplaymeplay block, and the glove block. The tracking block takes in video information from a camera and estimates the x and y coordinates of the players hands. This information is then passed to the gameplay block which models the physics of the game, and outputs video information to the VGA display. The glove block contains all off-chip circuitry, as well as the fpga hardware which controls the vibration of the motors. See figure 1 for their major interactions.

2.1 Tracking Block

The tracking block analyzes video input from a camera and determines the position of the players hands on screen. It takes color and ancillary information as inputs and outputs the position of the players to the gameplay block. This section was constructed by Turner Bohlen.

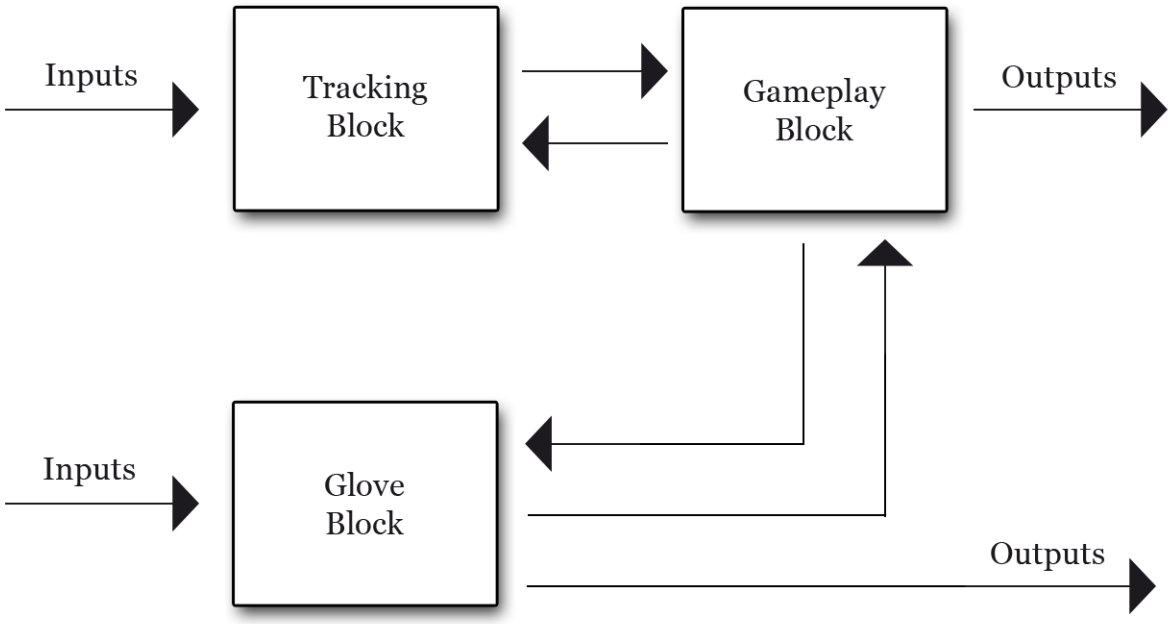

Figure 1: The virtual-reality rock climbing game is implemented with three major components: the tracking block, the glove block, and the gameplay block. Information flow is indicated with arrows. Note that input is processed by either the glove or tracking system before being used by the gameplay block.

2.2 Gameplay Block

The gameplay block handles the physics of the game, and outputs video informationmation to the VGA display. The gameplay block takes the hand position from the tracking block as well as grab information from the glove block. From this information, it determines the location of the player on the rock wall and the handholds, the objects to display on screen, and whether or not the user is grabbing a handhold. This section was constructed by Chris Lang.

2.3 Glove Block

The glove block comprises both the fpga and off-chip circuitry. Its purpose is to manage inputs to and outputs from the gloves. When a player flexes their hand in real life, the off-chip circuitry signals the fpga, and the player grabs in the game. Additionally, it takes information from the gameplay block describing if the user is touching objects on screen and outputs a feedback signal to the corresponding glove. This section was primarily constructed by Chris Lang, although both partners were involved in its creation.

3 Hardware

This system required a significant amount of additional hardware. The most important part of the off-chip circuitry was the flex sensors used to detect player grabbing. When the player grabs in reality, the flex sensor is bent, and its resistance changes. The flex sensor is placed in series with another resistor to create a variable voltage divider. A second voltage divider, comprising simply of two resistors, creates a threshold voltage. Both of these are then used as inputs to an op amp, which acts as a comparator, outputting a 0 or 5 volt signal to the fpga. While simple in design, the flex sensors were much more difficult to work with in practice. As the game was tested and the sensors endured strain, they became permanently bent, reducing their change in resistance. Additionally, they did not have a proportionally large resistance change to begin with (at best ranging from 50 to 80 kOhms). This increasingly lessening change in resistance, combined with their tendency to drift over time, led to them being less than ideal for the project. A much simpler and reliable alternative would have been simply including a button

on the players palm to indicate grabbing. Another alternative to be considered for future designs is using two metal contacts on the players fingertips and palms. One helpful design choice that made the current implementation possible was to use a potentiometer instead of a normal resistor in the threshold voltage divider. This allowed a much quicker, and more precise setting of the threshold voltage. This proved to be extremely helpful if any of the resistances drifted. Although not ideal, the flex sensors worked well enough for the project. Additionally, due to lab kit malfunctions, the circuitry was switched onto a new labkit at the last minute, which may have caused additional grabbing issues.

The second main piece of hardware used was the vibrating motors. Although small, they provided a strong force. However, when they vibrated, they drastically lowered the power supply voltage on the fpga, having a large negative effect on the grab detection. Because the motors were used to indicate when a player was able to grab a hold, the haptic feedback was chosen to be removed from the final version of the project. A photo of the gloves, and off-chip circuitry can be seen in figures 2

4 Tracking Block Design

The tracking block uses camera inputs to calculate hand positions, which are then sent to the gameplay block. Four modules comprise the block: the camera module, the frame buffer, the center of mass module, and the position module. Information flow between these modules is shown in figure 3.

4.1 Camera Module

The camera module has the responsibility of taking in and processing the information from the NTSC camera.

The NTSC camera output is connected to the 6.111 lab FPGA board. A set of modules provided by the 6.111 staff processes the input signal and extracts YCrCb color information, horizontal sync, vertical sync, and field data. This data is then passed through a YCrCb to RGB converter and on into the nts_to_zbt module, which prepares the information for storage in the zbt ram.

Color is downsampled to eighteen bits, six each for red, green, and blue. This allows the storage of two pixels in each thirty-six-bit memory slot. Pix-

els are stored in an address indexed by their concatenated y and x coordinates on the screen, dropping the last bit because two pixels share each memory slot. ntsc_to_zbt counts the x and y coordinate of the incoming pixel data and caluclates a memory address based on this information. In doing this calculation, it takes into account the fact that a mirrored screen gives a more intuitive interface for a user and thereby uses $719 - x$ for the x coordinate of the pixel. This mirroring assures that a leftward motion of the user's hand is shown as a leftward motion on the screen, rather that rightward. This module also concatenates the color data for two consecutive pixels, producing the data to be stored in a single memory slot. Finally, ntsc_to_zbt syncronizes this information with the $65Mhz$ system clock where it was previously syncronized with the camera input clock.

4.2 Frame Buffer

The decision was made to use only a single frame buffer for both storage and retrieval. The simplicity of the calculation involving individual pixels made it clear that a two-frame buffer would not be necessary. because of this, data from the ntsc_to_zbt module is passed directly to the 6.111-provided zbt_6111 module, which stores the provided information in the provided address of zbt memory.

The same zbt_6111 module handles output from memory to the center-of-mass module. The 6.111-provided vram_display module calculates the address of the relevant pixel using the xvga module's control signals and sends that to zbt_6111. Read-write is chosen based on the last bit of the current camera hcount.

4.3 Center of Mass Module

The center of mass module calculates the center of mass of certain shades of red and green in each frame, outputting the x and y coordinates of each.

Input from the zbt_6111 read commands is fed into each of two center of mass (centerOfMass in the verilog files) modules along with the value of each of the 8 6.111 lab board switches and two button inputs, one reset button and one set button.

The hcount and vcount information is first checked to see if the current pixel is valid. Any pixel outside of the 720 by 480 camera input area is discarded as unnecessary. Following this check, the pixel is compared to a

certain set of criteria and, if it fits them, deemed part of the glove. These criteria come in three parts. First a color selection signal indicates whether we are tracking a red, green, or blue (the "primary color" object). Next, a minimum value is set for the primary color. Any pixel for which the primary color is below the minimum is ignored. Finally, a minimum difference is set for each of the two remaining "secondary" colors is defined. This minimum difference requires that the value of each of the other colors in the pixel be at least a certain amount below the value for the primary color. If either secondary color is larger than $primary\_color - minimum\_difference$ the pixel is ignored. What remains are a set of pixels that match an expected color profile.

Setting the right color profile is the most difficult part of the problem. Because the lighting within the scene can vary depending on the user's hand position and angle, only very specific settings allow for good tracking across all possible user hand positions. Additionally, every time the system is used the lighting will be slightly different. The only simple way to overcome this was to provide a calibration mode allowing the user to determine the best settings for a given scene and lighting.

The eight switch and two button inputs provide this phenomenally helpful configuration control. Two screens are available to the user along with the game screen. Button 0 toggles between them. One allows for configuration of the red glove tracking and the other for configuration of the green glove tracking. When on these screens the user may press button 2 in order to "reset" the tracking and require the system to use the inputs of switches 0 through 3 as the minimum difference value and switches 4 through 7 for the minimum primary color value. These screens highlight all pixels interpreted by th module as part of the glove in green or red, depending on the glove in question, and show the center of mass using one vertical and one horizontal line. This information and experimentation allow the user to determine the best possible settings for tracking each glove. Hitting button 1 sets the current switch values as the tracking settings.

All of this functionality is implemented inside the center of mass module, including detection of the button signal edges, storage of the set values, and selecting between the set values and the switch values depending on the system state. A set of four configurations, each of higher accuracy than the last, can be seen in figure 4.

Two more steps are necessary for accurate tracking. First, weighting based on adjacent pixels biases the center of mass toward large blocks of color

instead of isolated pixels that could be the result of random noise in the video camera. This is done by pushing pixels through three registers. Each pixel is considered when it reaches the second register and weighted based not only on its own color but also on the color of the pixel immediately before and after it. As it is implemented, this has an odd effect of relating the last pixel in each row with the first two of the next. Fixing this would be good but is not strictly necessary from a functionality prospective, considering how unlikely it is for any two specific pixels to encounter unlikely noise on the same frame.

Pixels with no adjacent pixels of the correct color are weighted with a 1, pixels with one adjacent pixel of the correct color are weighted with a 2, and pixels with both adjacent pixels of the correct color are weighted with a 4.

The final step is to accumulate three weighted sums, one of the x coordinates of all matching pixels, one of the y coordinates, and one of the number of pixels matching. When both hcount and vcount hit zero, these values are fed into dividers and reset to zero before the first pixel of the next frame is considered. The divider outputs are the x and y centers of the color in question.

Each center of mass module considers only one color at a time. Two are used, one for green and one for red.

4.4 Position Module

The position module takes as input the center of mass position output and further refines it. Even after the careful work of the center of mass module random noise still resulted in erratic motion of the user hand position on screen. To fix this, the last four center of mass positions are passed through registers and the position of the user's hands in each frame is defined by the average of those four values. This implements a simple low-pass filter and removes any high-frequency noise caused by random glitches. The output of this module is the stabilized hand position. Two such modules are used, one for the green hand and one for the red.

5 Gameplay Block Design

The gameplay module models the physics of the game, as well as outputs video information to be displayed on screen. Based on information from

the grab button, player hand position, and hcount and vcount signals, it determines which of the player's hands are grabbing holds, and simulate their movement accordingly. Additionally, it is able to provide a feedback signal to the glove module to signal when a players hand is above a hold, and is not grabbing. The gameplay module is broken up into four submodules: hand-holds, grabbing, movement, and pixel information, which can be seen in figure 5. The hand-holds module keeps track of where the hand holds are on the game map, and which onscreen pixels are occupied by hand holds. The grabbing module takes this information, in addition to the player hands positions from the tracking module, and grab information from the glove module, and decides whether the player is grabbing onto any of the holds. The movement module uses this grabbing information, and the players hand positions, to simulate movement of the player. Finally, the pixel information module takes the pixel by pixel information from the hand holds, as well as of the positions of the players hands, and outputs pixel by pixel information, as well as its corresponding hcount and vcount signals, to the VGA out component of the labkit.

Initially when integrating all parts of the gameplay module, clocking issues were discovered, which resulted in incorrect colors, and minor artifacting. To solve this problem, the clock, and all other signals, were pipelined throughout the design. Information was passed linearly from hand-hold, to grab, to movement, and finally to the pixel information module, instead of going straight from the handhold to pixel information module, for example. Additionally, if one stage took more than a single clock cycle to complete, its pipeline information was delayed as well. This ensured proper syncing between all modules in the gameplay block. The largest initial source of error in the design of the gameplay module was mislabelled registers and wires. If there was even an error in movement, it was almost always caused by either not declaring a register as signed or indicating an incorrect number of bits. Both of which would lead to sporadic movements. A photo of the game being played can be seen in figure 6.

5.1 Hand-Holds

The hand-hold submodule are in fact multiple submodules, however they act as one. Each instance of the hand-hold module represents a single hold in the map, and has an x and y coordinate associated with it, indicating its absolute position. The holds have a width and a length, over which the holds

are present. Each hold sub-module takes in an hcount and vcount signal, as well as the screen's absolute position information and determine, pixel by pixel, whether a hand hold is present. This is done by adding the screen x and y coordinates to the hcount and vcount coordinates, and seeing if they lie within the width and height of each modules x and y coordinates. If it does, it outputs a one, associated with the corresponding hcount and vcount signal. If not it outputs a zero. The outputs of all of the individual hand hold modules with then be anded together to signal if the current pixel, is occupied by any hand hold. By default, the placement of the hand holds are predetermined. However, a map editor was partially completed. By flipping a switch, the player can enter this editing state. While in this state, the position of the screen tracks the players hands, and they can grab holds to move them to new locations. While the ability to grab and move holds was implemented successfully, the ability to navigate the map while in the editing mode was glitchy, leading to an overall failure of the map editting mode. This was most likely caused by a mislabed register, or perhaps an unwanted feedback loop in the design.

5.2 Grabbing

For each hand, a grabbing module determines whether or not the player is currently grabbing onto a hand hold, updating its output at every vsync. It does it in a similar, pixel by pixel, manner as done in the hand-hold module. Its inputs are hcount and vcount, the pixel by pixel exist signal coming from the holds, hand positions relative to the screen, and signals coming from both of the grab buttons. It has one output registers, grab, for each of the two hands. When hcount and vcount are equal to one of the relative hand positions, it checks to see if the player is pressing the corresponding grab button and if there is a hand hold at that pixel. If these are all true, then it signals the next output true. Otherwise it is set false. This is then sent to the movement module.

5.3 Movement

The movement module is in charge of determining the players absolute position. It uses the player's relative hand positions, grab information coming from the grab module, and his previous velocity to do so. In actuality, the players position is always fixed in relation to the camera, and is always dis-

played as a fixed-position circle onscreen, so the movement module actually updates the screen position. The screen position is then fed back to the hand-hold modules at every vsync.

The movement module has two different states: one for when the player is currently grabbing a hold, and one for free-fall. If the player is grabbing a hold, their position is determined by an anchoring hand. This is to avoid complications due to the player grabbing holds with both hands, and moving them in a way that violates the game. Typically, the anchoring hand is the hand that the player most recently grabbed a hold with. However, if the player lets go with the anchoring hand, the movement module assigns the other as the anchoring hand. Whenever a hand becomes the anchoring hand, its current relative position is stored, as is the absolute position of the screen. When a player is grabbing a hand hold, the movement module updates the camera position based on how far the anchoring hand has moved since it first anchored. It subtracts the current relative hand position from its relative anchored position and adds it to the stored absolute screen position. If a player moves his anchored hand down, the screen also shifts down, allowing the player to reach a higher hand-hold with his non-anchored hand. If the player is not grabbing any hand-hold, he is in freefall, with a corresponding x and y velocity. When he first releases his hand, the game-play module averages his most recent velocities in the x and y directions, and set that to his freefall velocity. With each frame, the movement module updates the camera position based on this velocity, and the y velocity are decremented in order to simulate gravity. This allows the player to quickly pull down and release his grip, propelling him high into the air in order to grab a previously unreachable hold.

5.4 Pixel Information

The pixel information module outputs a 24 bit, RGB color, along with appropriately pipelined hcount, vcount, and vsync information, to a VGA output. It takes in hcount and vcount signals, pixel information from the hand-hold module, and player hand position to determine the value of the current pixel output. It first checks to see if the hcount and vcount signals are within a small radius from each hand position. If so, it sets the pixel output to red or green, signifying player hand position. Otherwise it checks to see if the hcount and vcount signals are within a small radius from the center of the screen. If so it sets the pixel output to white, signifying the players posi-

tion. Otherwise, it check to see if the pixel information from the hand-hold module is true. If so, it sets the pixel output to blue, indicating a hand hold. If none of these are true, it displays a gradient background. It does this by adding the vcount value to the negative screen position (as higher screen position have more negative value). This is the gradient multiplier of each background pixel. It then multiplies the R, G, and B values of each pixel by this multiplier and shifts it 11 registers to the right (because the multiplier can be at most 2^{11}). If $vcount - screeny$ is greater than 2^{11}, it sets the multiplier to 2^{11} to avoid overflow errors.

6 Glove Block Design

The glove block manages inputs from and outputs to the two gloves the player wears while using the game. Two modules comprise the entire block.

6.1 Tactile Feedback

Tactile feedback is provided by this module, which processes an input from the gameplay block and outputs a signal to power the motors attached to the user's gloves. This game used only the most basic tactile feedback. If a hand was touching a hold, the motors on that hand buzzed. As a result the module needed only feed the signal through from the gameplay block to the relevant output on the board.

A modification of this game and potential place for future work is to add more sophisticated feedback. The location of the buzzing on the player's palm could, for example, indicate which direction they need to move in order to find a hold and its strength could indicate how close they are to it.

6.2 Debounce

A simple debouncer syncronizes the noisy input from the flex sensors on the player's gloves and debounces them, providing the clean signal as output to the gameplay block.

7 Integration

While integrating the two major modules was initially frustrating, it did not prove to be as difficult as once thought. The only information passed between each module is the x and y positions of each hand. Because this information is updated only once a frame (which is very infrequent compared to the clock), setup and hold times were very lenient, and clock skew between the two could essentially be ignored. One integration issue that did come up was concerning the output to the VGA display. Each major module has their own VGA output, which then multiplexed. The tracking module has the calibration display, and the gameplay module displays the game. The outputs of both of these were not in sync with each other, as there was a long propagation delay through the gameplay module after already passing through the tracking module. Initially, when the output multiplexer selected which sets of signals to display, it used non-blocking assignments dictated by the tracking blocks clock, which did not meet the setup a holds times of the gameplay block. This initially aroused suspicion of a much larger clocking issue, but after switching the multiplexer to blocking assignments, the issue was fixed.

8 Conclusion

Two major learnings from this project are to be careful with the construction of any hardware components and, of course, to make sure that all registers have the correct length and are appropriately signed.

Connections between the motors, flex sensors, and wires on each glove came loose and broke very easily, and the flex sensors in particular deteriorated over time. Any project using flexible gloves and sensors like this one needs to be aware of the wear and tear endured by any components attached directly to the glove.

Overall, though, this project was a success. The use of colored gloves and an NTSC camera coupled with a high-quality tracking algorithm provided the smooth hand tracking that the system relied on. Erratic hand motion would have resulted in an unacceptably jittery screen because of the direct tie between hand position and screen output.

The gameplay module used the available inputs to display a set of handholds to the screen and allowed the player to "climb" the virtual wall by

closing their hand while hovering over a hold and moving their hand downward. It provided simple haptic feedback based on whether or not each hand was in contact with a hold, but was chosen to be removed due to power supply issues.

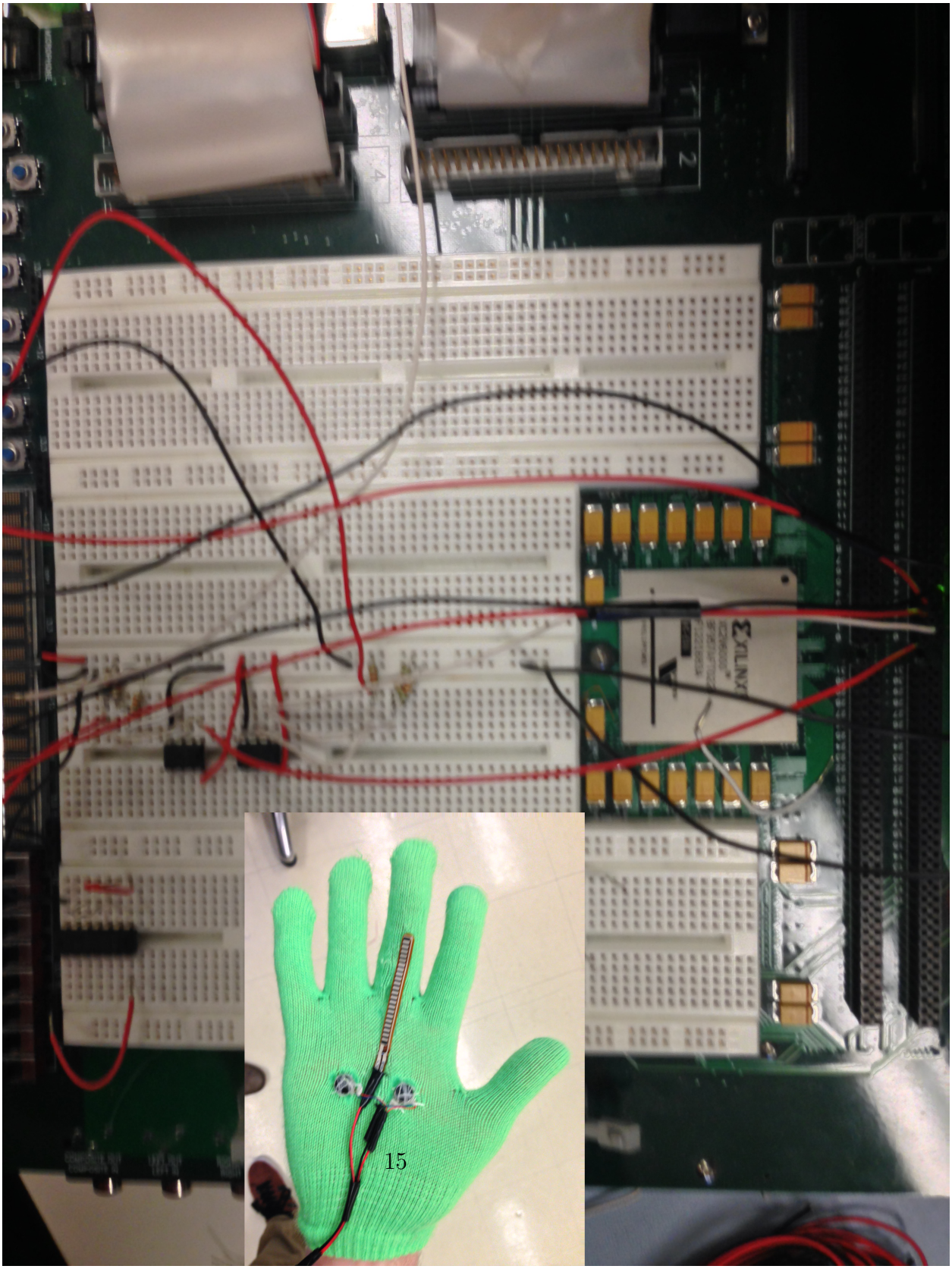

Figure 2: Apair of circuits built on to the lab kit convert the variable resistance of the flex sensors into a digital signal. This is done by comparing the

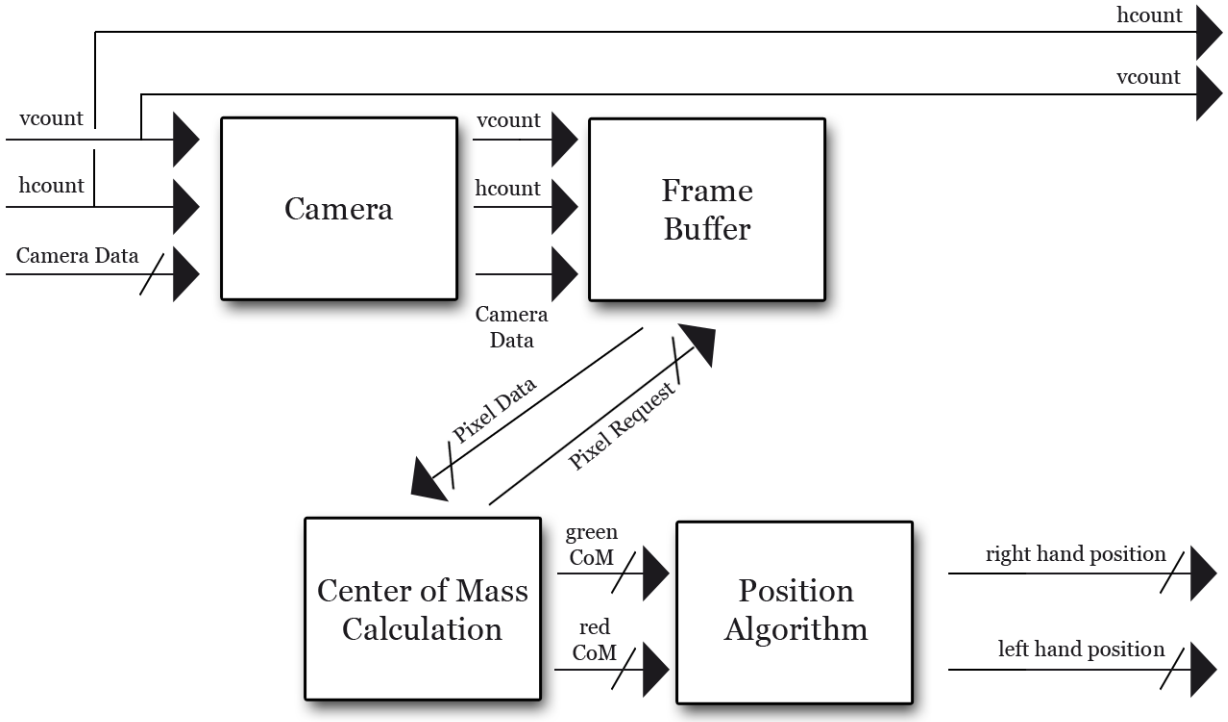

Figure 3: The tracking system has four primary steps. First, information is processed from the raw camera input. Second, frames are stored for later use. Third, center of mass information is calculated for each glove. Finally, a smoothing function using past position information to prevents errors and noise. The result is accurate information describing the position of the player's hands in the real world.

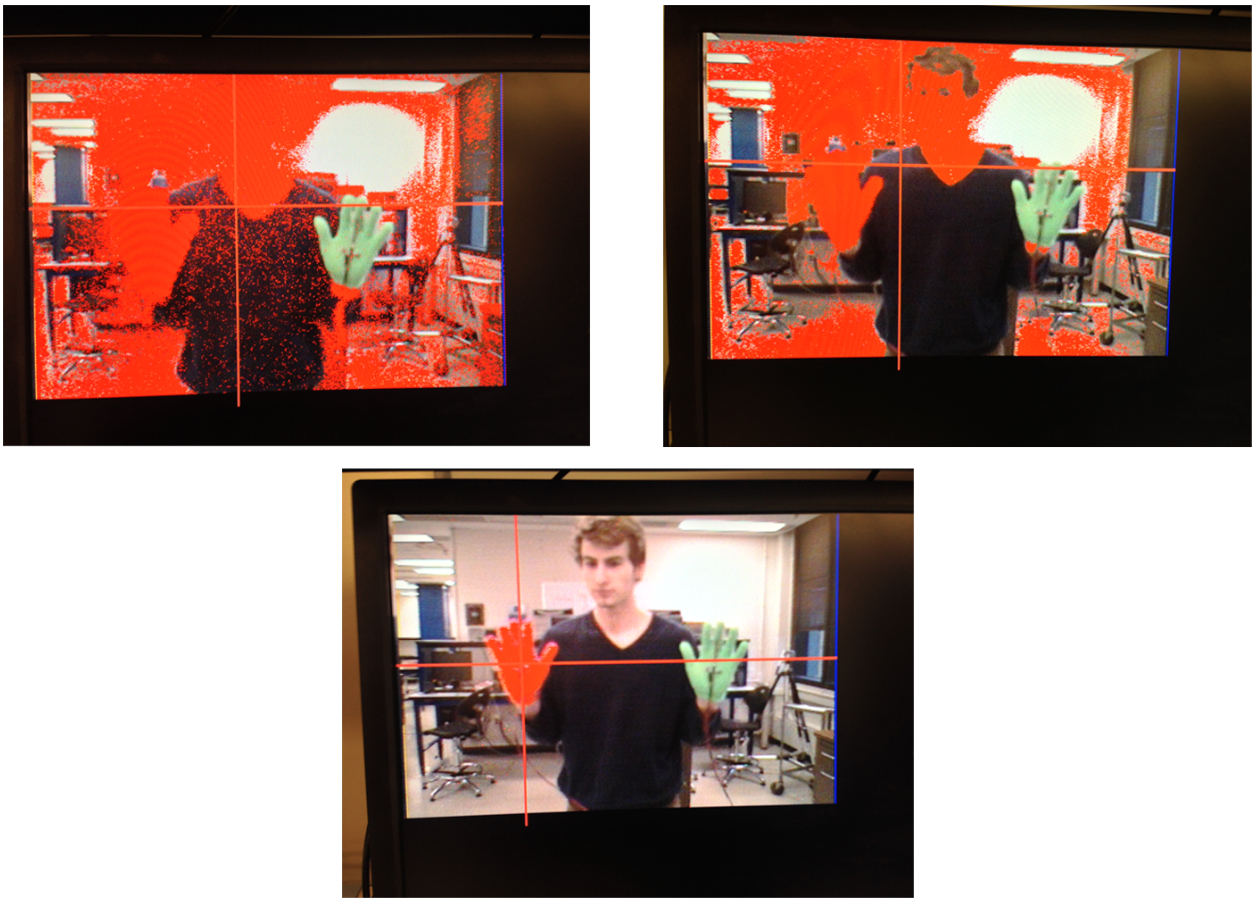

Figure 4: Configuration involves the modification of two variables, the minimum primary color value and the minimum color difference, to define which pixels are part of the glove. Three stages of configuration are shown, demonstrating the importance of this configuration step.

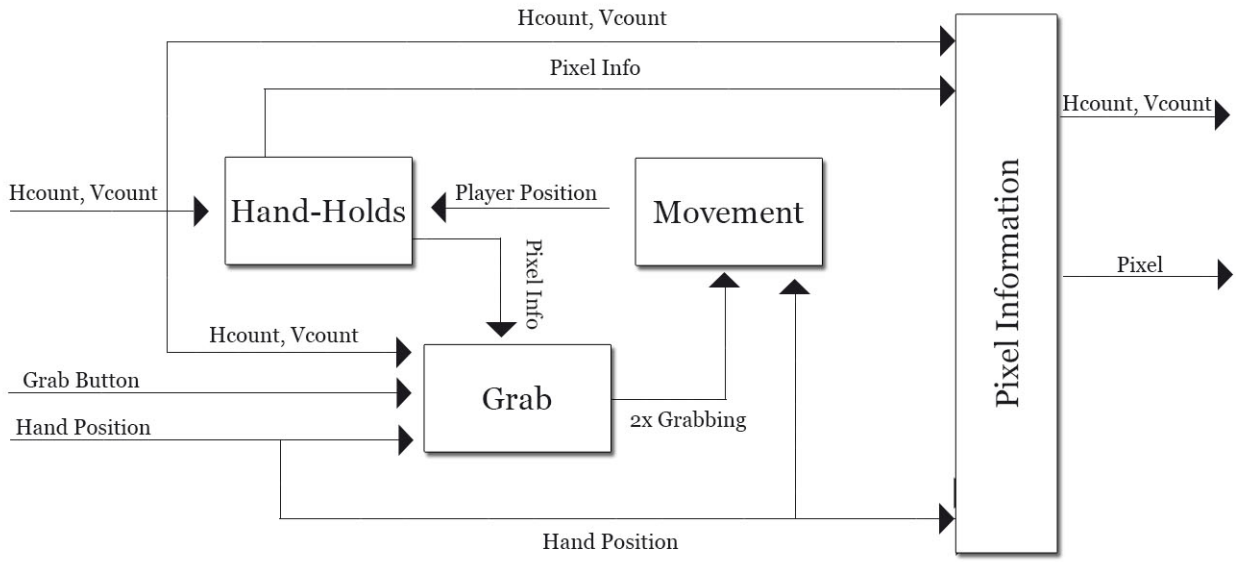

Figure 5: The gameplay block is comprised of four modules: hand-holds, grabbing, movement, and pixel information

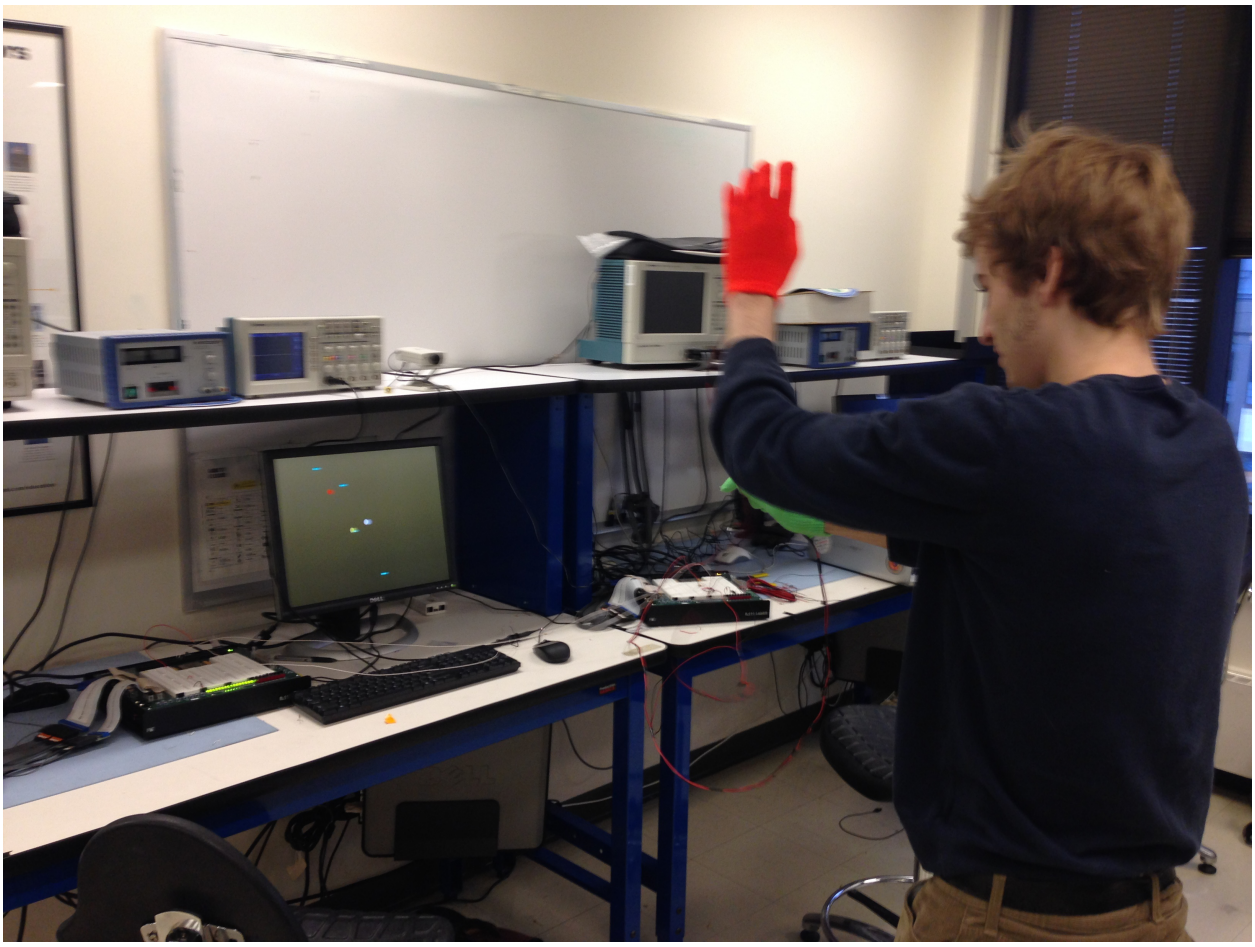

Figure 6: A user plays the game by standing a short distance from the camera and controlling their motion using their gloved hands.

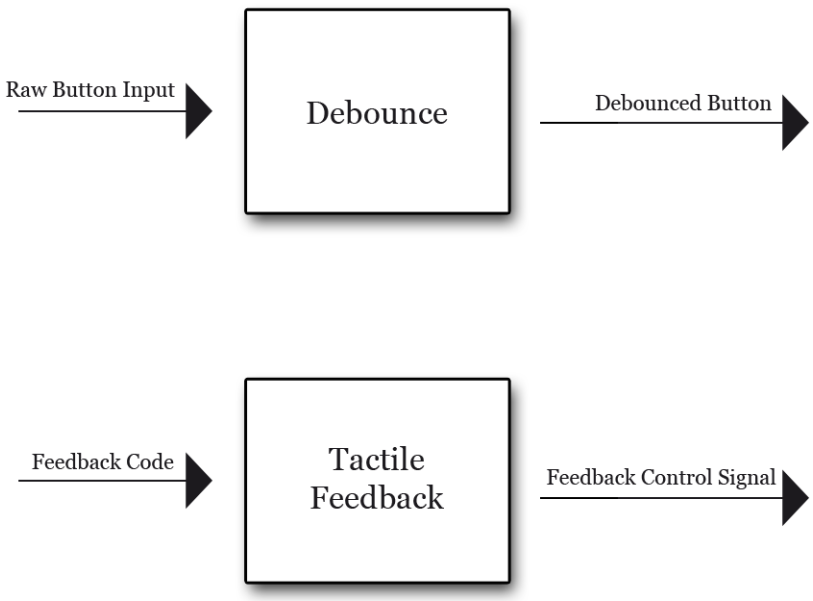

Figure 7: The glove system is comprised of two data processing steps, one for the debouncing of button input and the other for processing of tactile feedback duty cycles.