Computer Vision Pipeline For Object Recognition

Kevin Zhang and Felipe Hofmann
October 14th, 2017

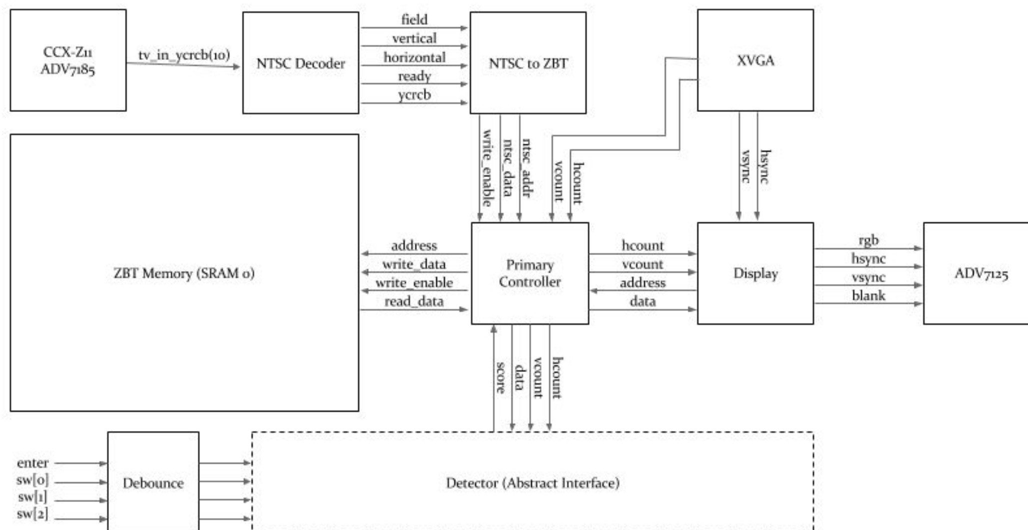
Abstract

We propose to implement a computer vision pipeline for motion tracking. This pipeline will read input from a camera, extract features for object recognition, and identify objects on a VGA monitor. The basic user experience is as follows: the user presses a button and a box appears on the screen - the user holds an arbitrary object in front of the camera so that it shows up in the box. When the button is released, the FPGA learns to recognize the object and draws a box around it whenever it is recognized in the field of view. The target object can range from a simple baseline of a purple ball on a white background to human faces.

Design

The first half of the high-level block diagram for our implementation is shown below. We plan to attach a CCX-Z11 camera to the labkit through the s-video port, decode the video signal with the onboard ADV7185 decoder chip, process the frames, and write to a VGA monitor with the onboard ADV7125 video encoder.

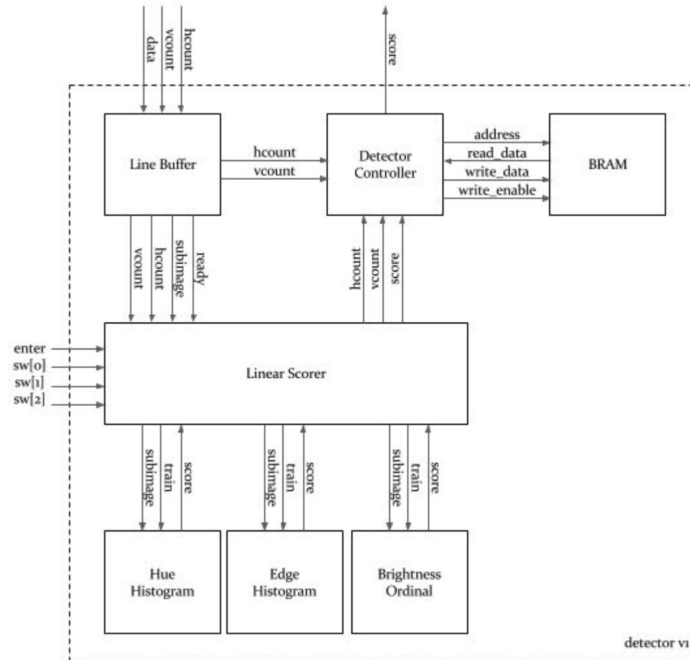
Due to the limited memory size of the BRAMs, we will be using one of the two 512kx36 ZBT SRAMs soldered onto the labkit to store the frame for processing. Because of the 2 clock cycle delay from the read/write signal to retrieval/execution, special care needs to be taken when retrieving pixel values.



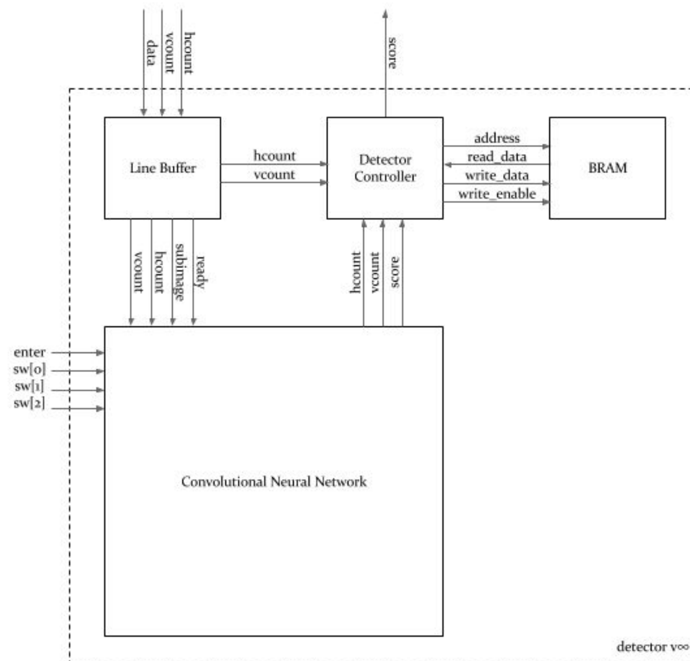
Note that the detector module in the above block diagram - see dashed lines - is an abstract interface which ingests pixel data and button presses and returns scores for each location in the frame. The key responsibility of this component is to recognize and track objects, and we propose 2 different concrete implementations based on traditional and modern computer vision techniques, respectively.

Furthermore, note that bus widths are omitted from the above diagram and will be discussed along with implementation details in a later section.

Our first concrete implementation of the detector module will rely on hue and edge features. The pixel data is accumulated in a line buffer which transmits subimages of size 16x16 to the linear scorer. The linear scorer returns a linear combination of the scores produced by the hue, edge, and ordinal feature detectors.



Our second optional concrete implementation which will only be implemented if time allows will use a shallow convolutional neural network with a ReLU nonlinearity to score the subimages. Some of the challenges of this implementation are speed and the lack of floating point support.



Both detectors respond identically to the user input. When the enter button is pressed, the detector learns to recognize the active object as object X, where X is the value indicated by the switches.

Video/VGA Modules[1]

NTSC Decoder

Inputs: tv_in_ycrcb [9:0]
Outputs: ycrcb [29:0], field, vertical, horizontal, ready

This module reads the NTSC stream produced by the ADV7185 decoder and produces YCrCb values as well as the appropriate field/vertical/horizontal signals. These outputs are passed to the *NTSC to ZBT* module.

NTSC to ZBT

Inputs: field, vertical ,horizontal, ready, ycrcb [29:0]
Outputs: ntsc_addr [18:0], ntsc_data [35:0], write_enable

This module uses the field/vertical/horizontal signals and counters to compute the current pixel position and outputs the memory address associated with that pixel. It also extracts the luminescence component of the ycrcb signal. These outputs are used by the primary controller to fill the ZBT memory.

XVGA

Outputs: hsync, vsync, hcount [10:0], vcount [9:0]

Generates XVGA display signals (1024 x 768 @ 60Hz). This is the standard module provided by the staff and is used without modifications. Due to timing concerns, we may downgrade this to 800x600 or even 640x480 resolution.

Display

Inputs: hsync, vsync, hcount [10:0], vcount [9:0], data [35:0]
Outputs: hsync, vsync, address [18:0], rgb [7:0]

This module accepts hcount/vcount and computes the appropriate memory address, taking into account the 2 cycle clock delay in reading from SRAM. Furthermore, since the memory is 36 bits wide with 4 pixels packed into each memory location and the extra 4 bits used to store the chroma values, this only needs to read 1 value every 4 clock cycles.

ZBT Memory

Inputs: write_enable, address [18:0], write_data [35:0]
Outputs: read_data [35:0]

This module interfaces with the onboard SRAM. There is a two cycle delay on read/write and accepts/return 36-bit values.

Primary Controller

Inputs: hcount [10:0], vcount [9:0], ntsc_addr [18:0], ntsc_data [35:0], write_enable, read_data [35:0],_____ address [18:0], score [3:0]

[1] Many of the modules used for video encoding/decoding are derived from the sample Verilog provided by the 6.111 staff on the course website at http://web.mit.edu/6.111/www/f2017/index.html.

Outputs: hcount [10:0], vcount [9:0], data [35:0], address [18:0], write_data [35:0], write_enable

The primary controller connects all of our submodules and handles switching between read and write for our SRAM frame buffer. The decoder runs at 27 MHz while the XVGA runs at 65 MHz, but XVGA reads only happen once every 4 cycles due to the pixel packing described above, allowing us to store NTSC data on the free cycles.

Detector Modules

Line Buffer

This module is responsible for caching 16-line subimages and feeding each 16x16 patch to the linear scorer by pulling ready high when the data is available.

Detector Controller

This module is responsible for storing scores produced by the linear scorer and retrieving them from BRAM upon request.

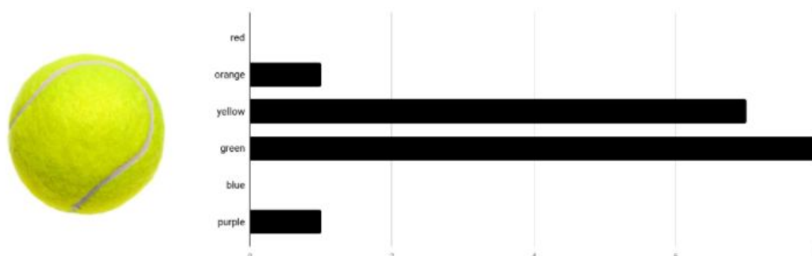
Linear Scorer

This passes the subimages to the individual detector modules and returns a linear combination of the scores returned by the detectors. The linear combination will initially be hardcoded but eventually can be automatically optimized. In addition, this module is responsible for interpreting the user input - i.e. button down means that the user is training the module to recognize object X, where X is determined by the switches - and pulling *train* high.

Hue Histogram

This module accepts *subimage[64*8-1:0]* and *train* as inputs and returns a *score[3:0]*. The subimage contains the cr/cb values scaled down to 4 bits each for the central 8x8 patch of the subimage. When *train* is pulled high, the *score* is set to 16 and the hue histogram module stores the current histogram; when train is pulled low, this module returns a score indicating how similar the current histogram is to the stored histogram.

An example of a histogram is shown below; as you may expect, a tennis ball will have high yellow and green values but low counts for the other colors.



We propose storing the target histogram for an object - i.e. the histogram computed from the center of the screen when the enter button is held down - in 48 registers where each of the 6 colors takes up 8 bits; the scoring function for comparing the histograms is shown here:

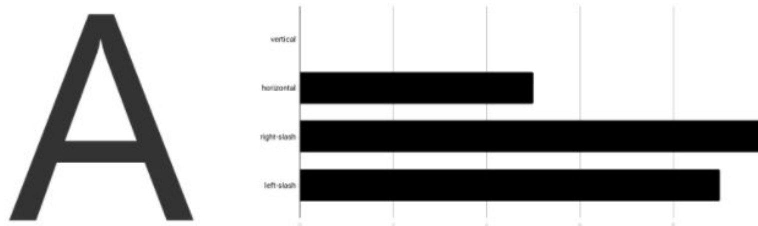
$$score = \sum_{color} |target_{count} - current_{count}|$$

Although *train* needs to be high for at least 1 clock cycle, once the histogram is stored in registers, the *score* computation can be purely combinatorial. The only arithmetic operations used by this module are comparisons, additions, and subtractions.

If we run into synthesis issues due to the large number of arithmetic operations, we can implement an alternative which uses dynamic programming to reduce it to 3 operations per subimage.[2]

Edge Histogram[3]

This module accepts *subimage[64*8-1:0]* and train as inputs and returns a *score[3:0]*. It computes a simplified "histogram of oriented gradients" in the subimage. For example, the below diagram shows the 4 primary types of edges found in the letter A.

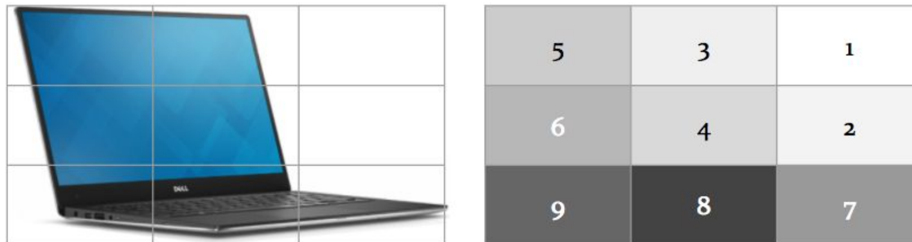


The edge detection will be performed via straightforward comparison operations (similar to the FAST corner detection method) allowing us to *train* in 1 clock and then compute scores with purely combinatorial logic.

Once again, any synthesis issues due to the large number of comparators can be resolved with a dynamic programming solution as indicated above.

Brightness Ordinal

This module accepts *subimage[64*8-1:0]* and train as inputs and returns a *score[3:0]*. It computes a brightness ordinal which splits the subimage up into an even grid and returns a sorted list of cells from brightest to darkest. This gives us a coarse measure of the morphology of the object.



As described above, this detector module also trains in 1 clock cycle and produces scores with purely combinatorial logic.

[2] We can store $\pi[i,j]$ to indicate the sum of colors up to column i and j such that $\pi[i,j] - \pi[i-16,j] - \pi[i,j-16] + \pi[i-16,j-16]$ returns an equivalent value with only 3 arithmetic operations.
[3] A variant of this is the histogram of oriented gradients (HoG) found in computer vision literature.