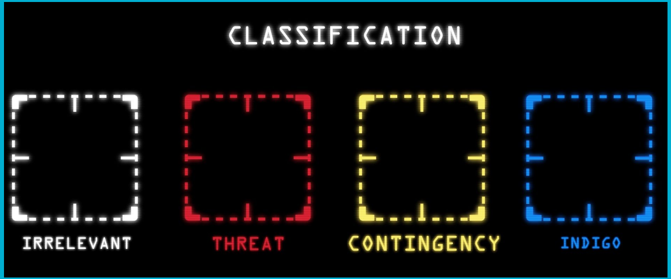


FPGA Object Tracking

Kevin Zhang and Felipe Hofmann

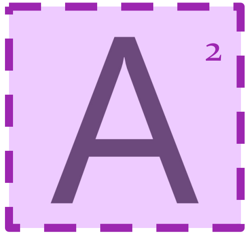
Overview

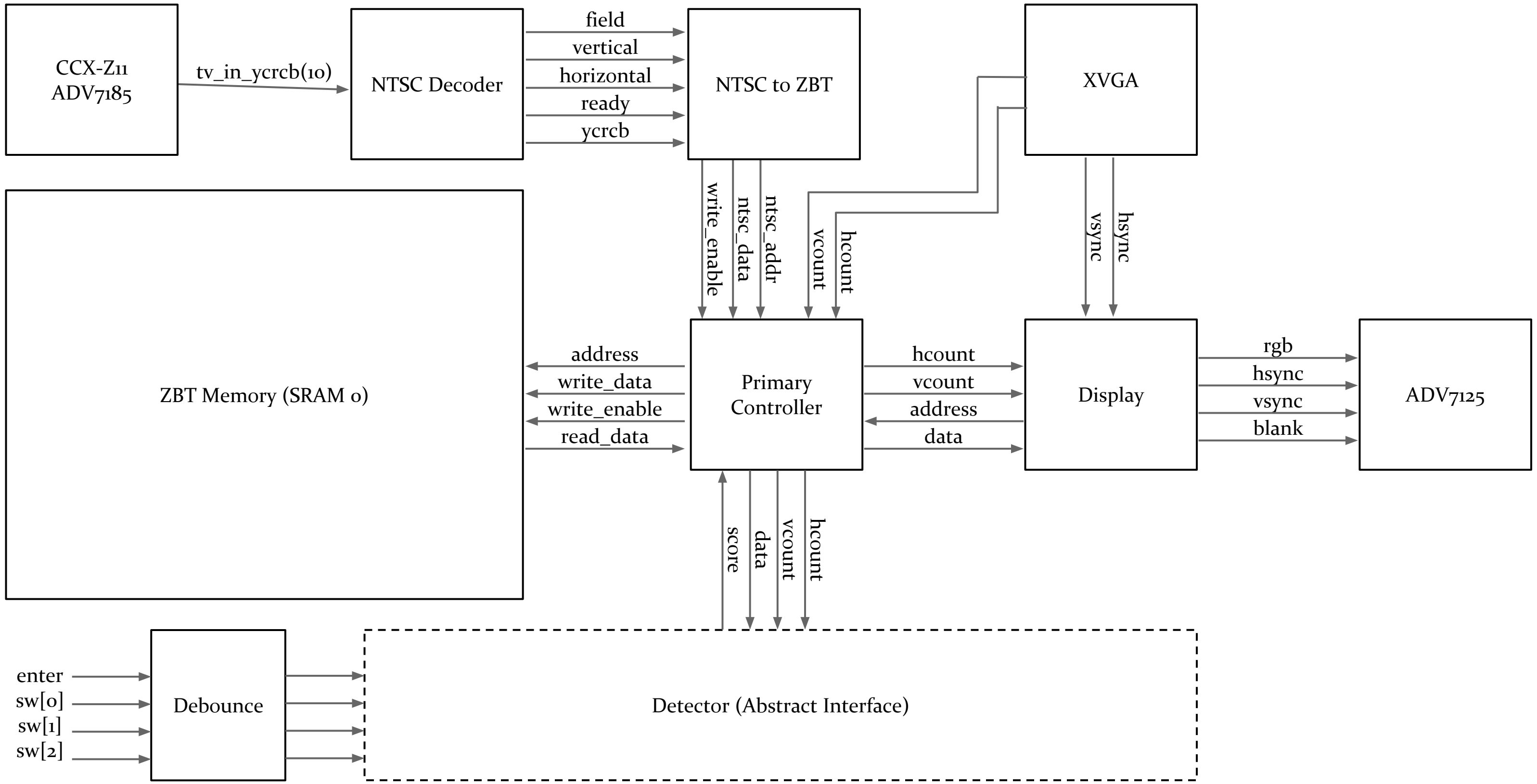
Goal

Fast and energy-efficient object recognition and tracking.

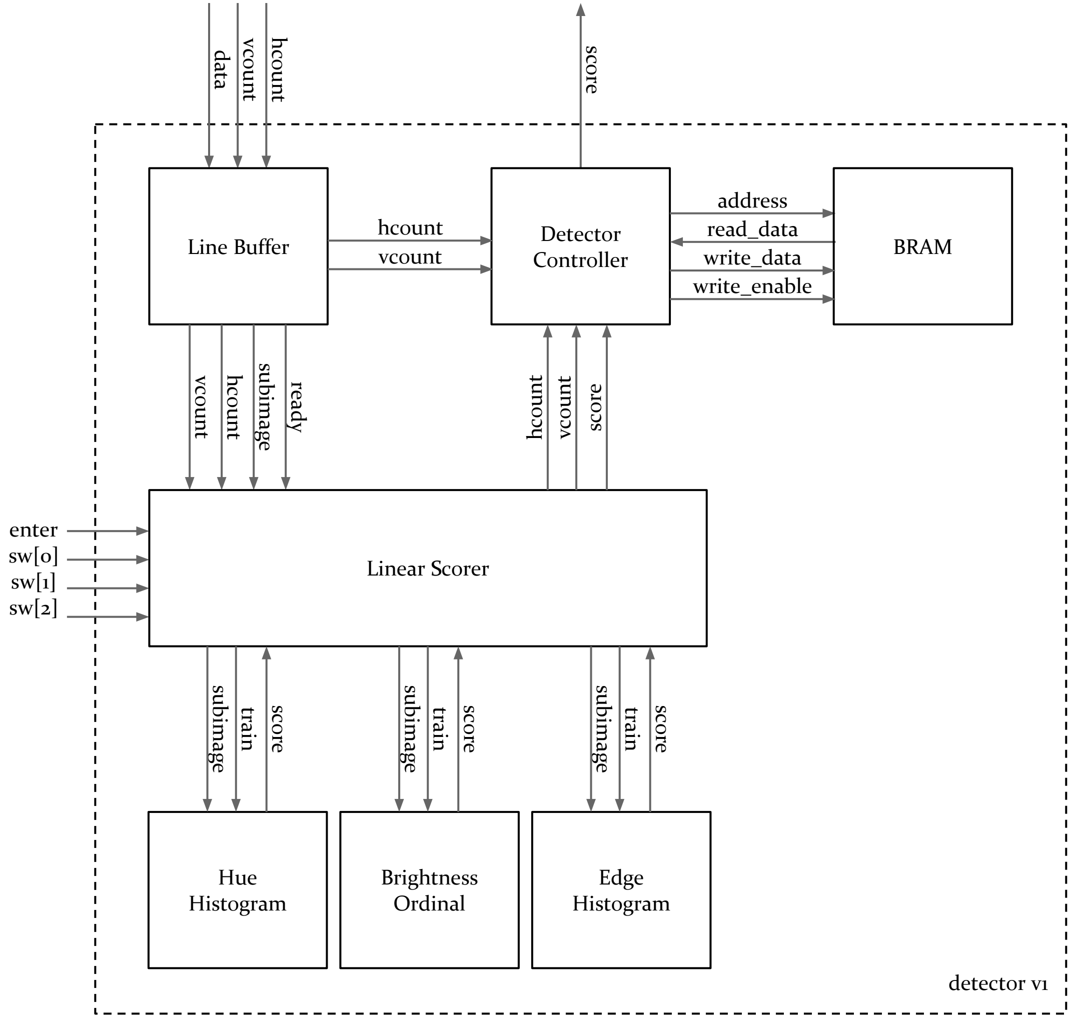
1. Hold an object in front of the camera.
2. Set the switches and hold the train button.
3. Repeat.

Each object (specified by switches 0-2) will automatically be recognized and tracked as it moves around the frame.





CCX-Z11 ADV7185
tv_in_ycrcb(10)
NTSC Decoder
field
vertical
horizontal
ready
ycrcb
NTSC to ZBT
XVGA
write_enable
ntsc_data
ntsc_addr
vcount
hcount
vsync
hsync
ZBT Memory (SRAM 0)
address
write_data
write_enable
read_data
Primary Controller
hcount
vcount
address
data
Display
rgb
hsync
vsync
blank
ADV7125
score
data
vcount
hcount
enter
sw[0]
sw[1]
sw[2]
Debounce
Detector (Abstract Interface)


data
vcount
hcount
score
Line Buffer
hcount
vcount
Detector Controller
address
read_data
write_data
write_enable
BRAM
vcount
hcount
subimage
ready
hcount
vcount
score
enter
sw[0]
sw[1]
sw[2]
Linear Scorer
subimage
train
score
subimage
train
score
subimage
train
score
Hue Histogram
Brightness Ordinal
Edge Histogram
detector v1

Design

Video Modules

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.

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.

Detector Modules

Line Buffer

Inputs: hcount[10:0], vcount[9:0], data [35:0]
Outputs: ready, hcount[10:0], vcount[9:0], subimage[511:0]

This module is responsible for caching 8-line subimages and feeding each 8x8 patch to the linear scorer by pulling ready high when the data is available.

Linear Scorer

Inputs: hue_score[3:0], edge_score[3:0], ordinal_score[3:0]
Outputs: patch[511:0], train, score, hcount[10:0], vcount[9:0]

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 manually set but 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.

$$c = \sum_i \alpha_i k(s_i, x) + \mathrm{b}$$

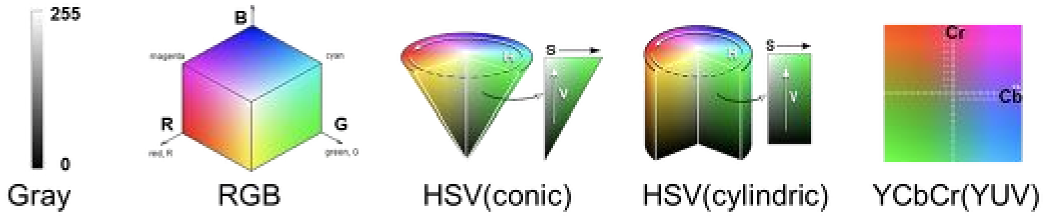
Hue Histogram

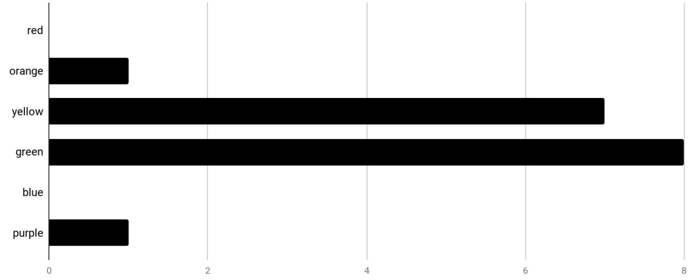
Inputs: subimage[511:0], train
Outputs: score

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.

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



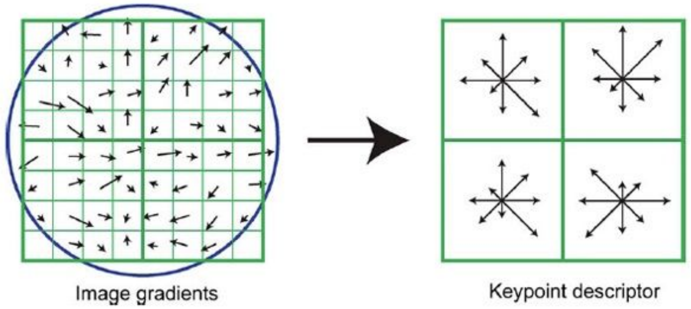


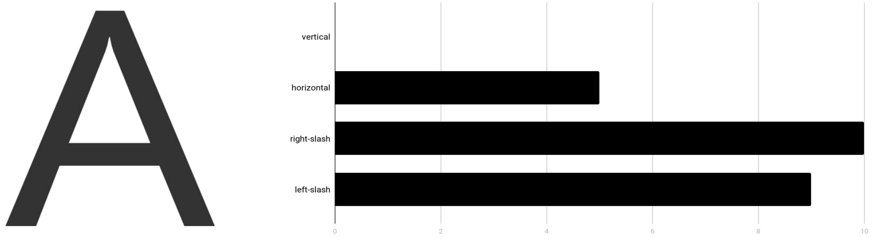
Edge Histogram

Inputs: subimage[511:0], train
Outputs: score

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.

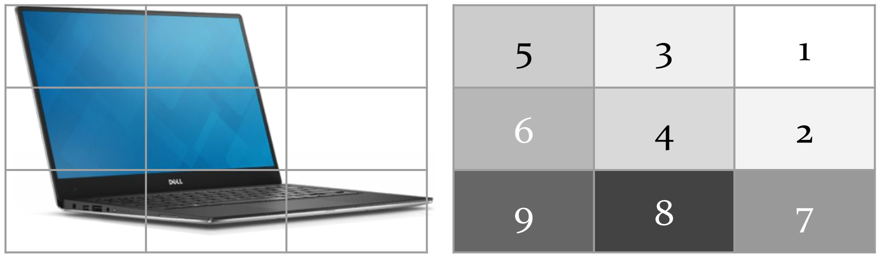




Brightness Ordinal

Inputs: subimage[511:0], train
Outputs: score

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.



Features

This presents a macroscopic view of the features extracted by detector v1. Rather than convolving over 16x16 (or 8x8) patches, here we aggregate over the entire image.

Hue Histogram

red	196911
purple	28992
green	55371
blue	15169

Edge Histogram

up	3766
down	4117
left	3247
right	3074

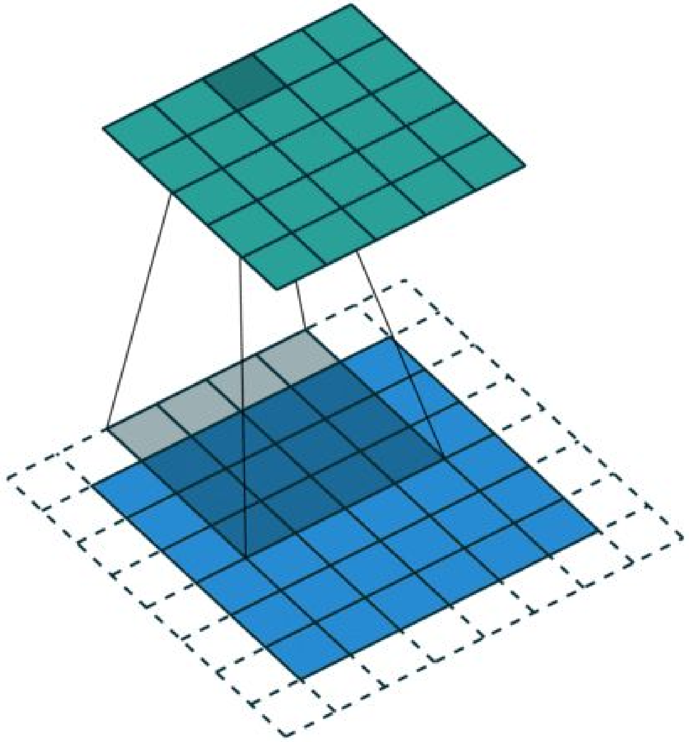
Brightness Ordinal

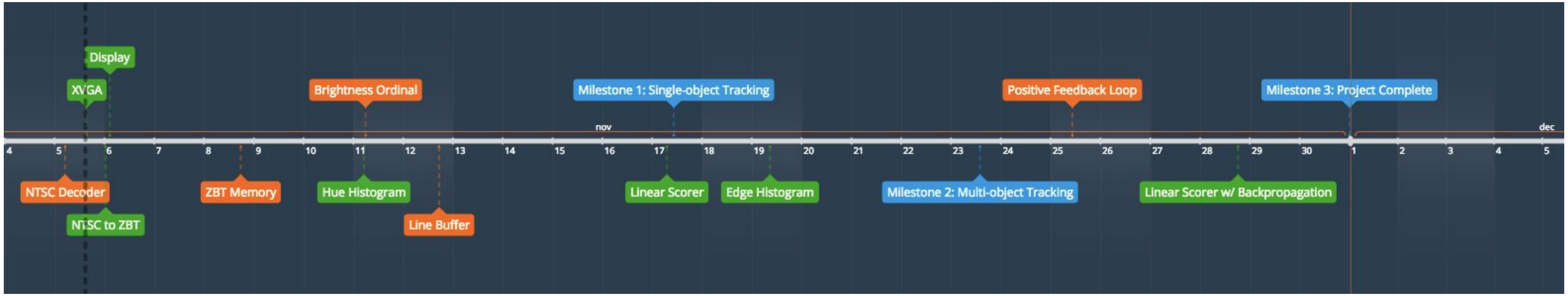
0	1
3	2

demo: https://kevz.me/app/6.111/

Aside: Convolutions

- All of the examples given so far have been computed on the "full" image.
- This isn't how it is actually computed… we have to convolve each feature extractor over 8x8 patches.
- Think convolutional neural networks… except with untrainable layers.




Display
XVGA
Brightness Ordinal
Milestone 1: Single-object Tracking
Positive Feedback Loop
Milestone 3: Project Complete
nov
dec
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
1
2
3
4
5
NTSC Decoder
NTSC to ZBT
ZBT Memory
Hue Histogram
Line Buffer
Linear Scorer
Edge Histogram
Milestone 2: Multi-object Tracking
Linear Scorer w/ Backpropagation

Timeline