Lab C : Color Detection

胡晉瑄 林修賢
INSTRUCTOR: Professor Jiin Lai

Overview

- Background Introduction
- Performance Analysis
- Result of color detection
- Suggestion for Improvement

Color Detection

The Color Detection algorithm is basically used for color object tracking and object detection, based on the color of the object. The color based methods are very useful for object detection and segmentation, when the object and the background have a significant difference in color.

(a)

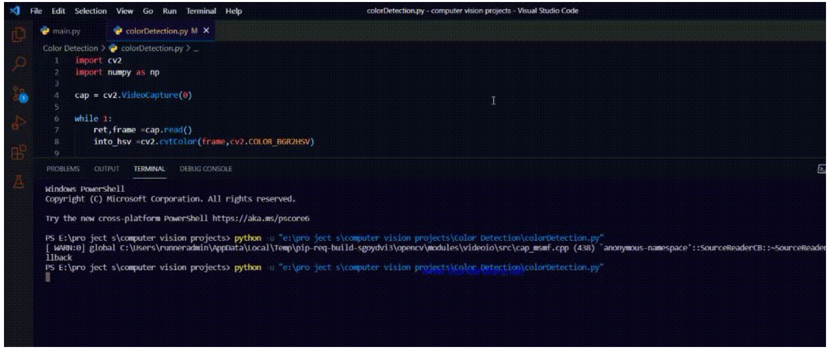
(b)

Why FPGA acceration for color detection?

Nowdays, object detection by color is widely use in many field, such as image annotation, vehicle counting, face detection and also using in tracking objects.

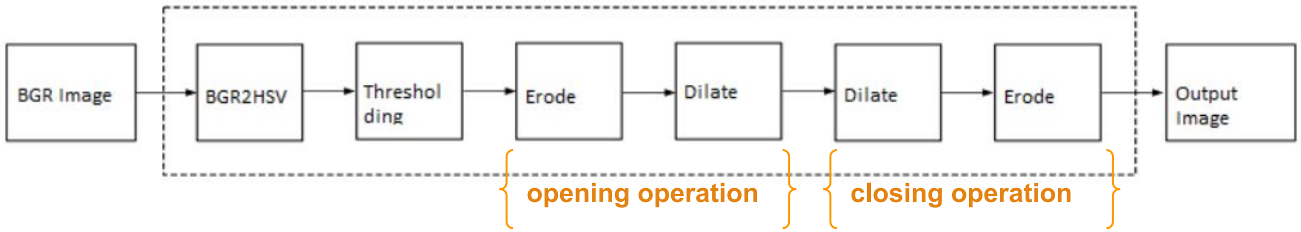
We can find that most of above application are video type and have a real-time need. However, CPU is not suitable for such kind computation. FPGA can solve this problem, we can design a specialized architecture for color detection. With FPGA, we could:

1. Increase the throughput of applications (FPS)
2. Shorten the latency for computing (real time)
3. Utilize hardware of FPGA for data parallelism



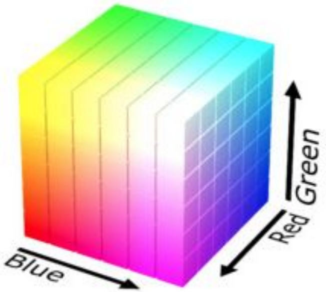
Flow of Color Detection

- ➢ The color space of the original BGR image is converted into an HSV color space.
  - BGR2HSV
- ➢ Applying the thresholding operation on the HSV image and return either 255 or 0.
  - Thresholding
- ➢ Applying morphological operation, reducing unnecessary white patches (noise) in the image.
  - Morphological Opening: Erode -> Dilate
  - Morphological Closing : Dilate -> Erode

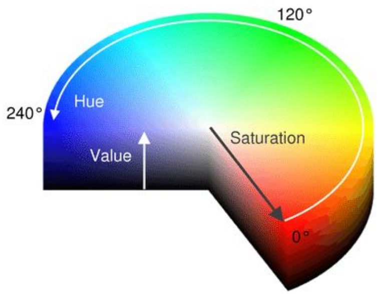


Convert BGR to HSV

- Why HSV?
  - HSV color space is useful for working with color information. It stands for HUE, SATURATION, and VALUE (or brightness). It is very useful in image processing tasks that need to segment objects based on its color
  - **HUE** : The hues are modeled as an angular dimension that encodes color information.
  - **SATURATION** : Saturation encodes the intensity of color.
  - **VALUE** : Value represents the amount to which that respective color is mixed with black.
- RGB colorspace are coded using the three channels, it is more difficult to segment an object based on its color.



RGB colorspace



HSV colorspace

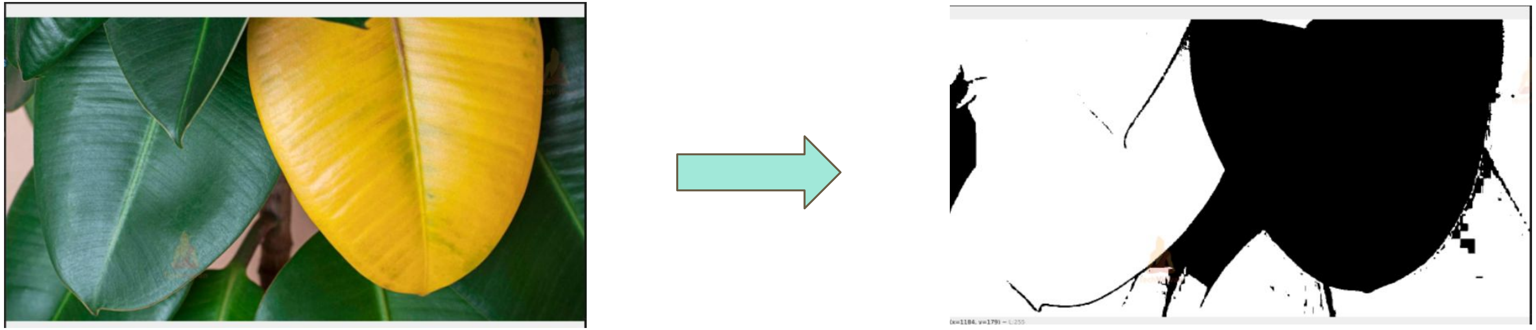
Thresholding

- ➢ Set lower bound and upper bound of detecting objection.
- ➢ Get a binary mask of the frame where the target color is present

Ex: As following example, we set the lower bound to [50, 20, 20], upper bound to [100, 255, 255] (bound for leaves) .

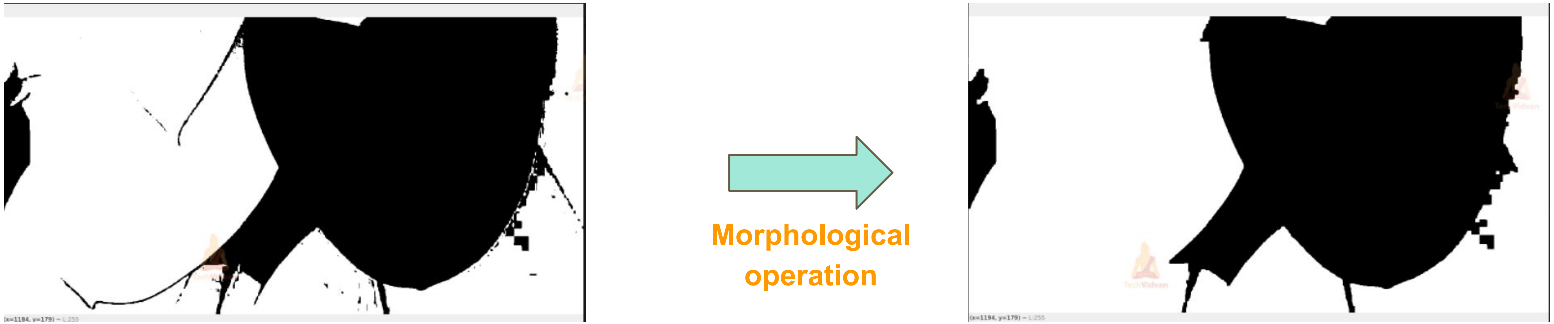
Then, we can use thresholding function to filter the image.

255(white) is for targeting color, 0(black) is for the other color.

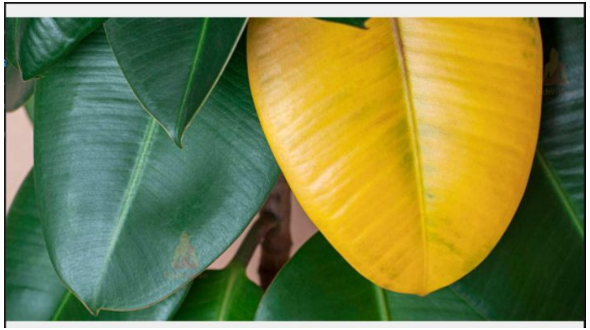


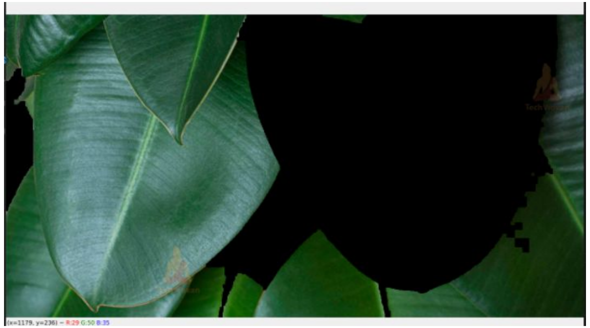
Morphological Opening & Closing

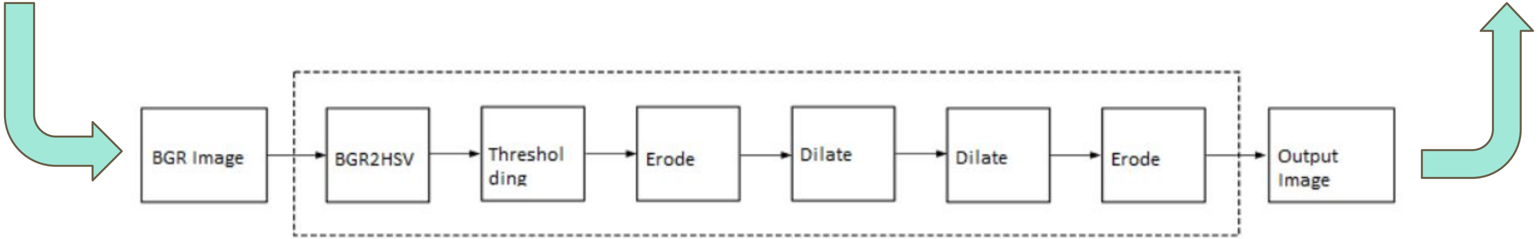
- In the original mask, we can see that there is lots of unnecessary noise. So we have to remove it to get a better result.
- Morphological Opening:
  - Eroding and then dilating. It is aimed to remove little white dot in the black region.
- Morphological Closing :
  - Dilating and then Eroding. It is aimed to remove little black dot in the white region.



Flow result






BGR Image
BGR2HSV
Threshol
ding
Erode
Dilate
Dilate
Erode
Output
Image

Hardware Functions

The Color Detection example uses five hardware functions from the Vitis vision library. They are:

- Array2xfMat
- BGR2HSV
- colorthresholding
- erode
- dilate

Kernel Functions - Array2xfMat

```
static void axiStrm2xfMat(hls::stream<ap_axiu<PTR_WIDTH, 0, 0, 0> >& srcPtr,
                          xf::cv::Mat<T, ROWS, COLS, NPC, XFCVDEPTH>& dstMat) {
    Axi2Mat(srcPtr, dstMat.data, dstMat.rows, dstMat.cols);
}
```

```
static void Axi2Mat(hls::stream<ap_axiu<PTR_WIDTH, 0, 0, 0> >& din,
                    hls::stream<ap_uint<XF_BITS_PER_CLOCK>, XFCVDEPTH>& dout,
                    int rows = ROWS,
                    int cols = COLS) {
// clang-format off
#pragma HLS DATAFLOW
    // clang-format on
    hls::stream<ap_uint<PTR_WIDTH> > ldata;

    ap_uint<log2<ADDRBOUND>::cvalue + 1> axibound = _MMITER::addrbound(rows, cols);
    Axi2AxiStream(din, ldata, axibound);
    AxiStream2Mat(ldata, dout, rows, cols);
}
```

1. Input a 1D array
2. MAXI in - stream out
3. fill with a 2D matrix

Kernel Functions - bgr2hsv

❖ Pseudo code

```
for(rows){
    for(cols){
        src_mat>>pixel_in;
        convert(pixel_in, pixel_out);
        dst_mat<<pixel_out;
    }
}
```

- In bgr2hsv function, it will stream in every bgr pixels and calculate its corresponding hsv representation and then stream out.

Kernel Functions - bgr2hsv convertion

```
b = in_pix.range(p * STEP + STEP - 1, p * STEP);
g = in_pix.range(p * STEP + (2 * STEP) - 1, p * STEP + STEP);
r = in_pix.range(p * STEP + (3 * STEP) - 1, p * STEP + 2 * STEP);

v = b;
vmin = b;

CV_CALC_MAX_8U(v, g);
CV_CALC_MAX_8U(v, r);
CV_CALC_MIN_8U(vmin, g);
CV_CALC_MIN_8U(vmin, r);

diff = v - vmin;
vr = v == r ? -1 : 0;
vg = v == g ? -1 : 0;

s = (diff * sdiv[v] + (1 << (11))) >> 12;
h = (vr & (g - b)) + (~vr & ((vg & (b - r + 2 * diff)) + ((~vg) & (r - g + 4 * diff))));
h = (h * hdiv[diff] + (1 << (11))) >> 12;
h += h < 0 ? 180 : 0;

out_pix.range(p * STEP + STEP - 1, p * STEP) = (unsigned char)(h);
out_pix.range(p * STEP + (2 * STEP) - 1, p * STEP + STEP) = (unsigned char)(s);
out_pix.range(p * STEP + (3 * STEP) - 1, p * STEP + 2 * STEP) = (unsigned char)(v);
```

$$V = \max(R, G, B)$$

$$S = \begin{cases} \dfrac{V - \min(R,G,B)}{V} & if \quad V \neq 0 \\ 0 & if \quad otherwise \end{cases}$$

$$H = \begin{cases} 60(G-B)/(V-\min(R,G,B)) & if \quad V = R \\ 120 + 60(B-R)/(V-\min(R,G,B)) & if \quad V = G \\ 240 + 60(R-G)/(V-\min(R,G,B)) & if \quad V = B \end{cases}$$

Kernel Functions - colorthresholding

❖ Pseudo code

```
void colorthresholding(src_mat, dst_mat, low_thresh, high_thresh){
    for(rows){
        for(cols){
            src_mat>>pixel;
            for(channels in pixel){
                val &= inrange(pixel[channel], low_thresh, high_thresh);
            }
            dst_mat<<val;
        }
    }
}
```

$$\mathtt{dst}(I) = \mathtt{lowerb}(I)_0 \leq \mathtt{src}(I)_0 \leq \mathtt{upperb}(I)_0$$

$$\mathtt{dst}(I) = \mathtt{lowerb}(I)_0 \leq \mathtt{src}(I)_0 \leq \mathtt{upperb}(I)_0 \land \mathtt{lowerb}(I)_1 \leq \mathtt{src}(I)_1 \leq \mathtt{upperb}(I)_1 \quad \ldots$$

- In colorthresholding function, it is typically used to get a binary image out of a image or for removing a noise, that is, filtering out pixels with too small or too large values

```cpp
template <int SRC_T, int DST_T, int MAXCOLORS, int ROWS, int COLS, int NPC>
void colorthresholding(xf::cv::Mat<SRC_T, ROWS, COLS, NPC>& _src_mat,
                       xf::cv::Mat<DST_T, ROWS, COLS, NPC>& _dst_mat,
                       unsigned char low_thresh[MAXCOLORS * 3],
                       unsigned char high_thresh[MAXCOLORS * 3]) {
// clang-format off
  #pragma HLS INLINE OFF
  #pragma HLS DATAFLOW
    // clang-format on

    unsigned char low_th[MAXCOLORS][3], high_th[MAXCOLORS][3];
// clang-format off
  #pragma HLS ARRAY_PARTITION variable = low_th dim = 1 complete
  #pragma HLS ARRAY_PARTITION variable = high_th dim = 1 complete
    // clang-format on
    uint16_t j = 0;
    for (uint16_t i = 0; i < (MAXCOLORS); i++) {
// clang-format off
    #pragma HLS PIPELINE
        // clang-format on

        low_th[i][0] = low_thresh[j];
        low_th[i][1] = low_thresh[j + 1];
        low_th[i][2] = low_thresh[j + 2];
        high_th[i][0] = high_thresh[j];
        high_th[i][1] = high_thresh[j + 1];
        high_th[i][2] = high_thresh[j + 2];
        j = j + 3;
    }

    uint16_t img_height = _src_mat.rows;
    uint16_t img_width = _src_mat.cols;

    xFInRange<SRC_T, DST_T, ROWS, COLS, XF_DEPTH(SRC_T, NPC), XF_DEPTH(DST_T, NPC), NPC, XF_WORDWIDTH(SRC_T, NPC),
              XF_WORDWIDTH(DST_T, NPC), MAXCOLORS>(_src_mat, _dst_mat, low_th, high_th, img_height, img_width);
}
} // namespace cv
} // namespace xf
```

set the threshold

use InRange to filter out

```
template <int WORDWIDTH_SRC, int WORDWIDTH_DST, int DEPTH_SRC, int DEPTH_DST, int SRC_T, int NPC>
void inrangeproc(XF_SNAME(WORDWIDTH_SRC) & val_src,
                 XF_SNAME(WORDWIDTH_DST) & tmp_val,
                 XF_PTNAME(DEPTH_DST) channel_out[XF_CHANNELS(SRC_T, NPC)],
                 XF_PTNAME(DEPTH_DST) _lower_thresh[XF_CHANNELS(SRC_T, NPC)],
                 XF_PTNAME(DEPTH_DST) _upper_thresh[XF_CHANNELS(SRC_T, NPC)]) {
    XF_PTNAME(DEPTH_SRC) p;

    ap_uint<8> tmp_val1 = 0;

    for (ap_uint<13> k = 0; k < (1 << XF_BITSHIFT(NPC)); k++) {
// clang-format off
    #pragma HLS unroll
        // clang-format on
        p = val_src.range(k * XF_PIXELDEPTH(DEPTH_SRC) + (XF_PIXELDEPTH(DEPTH_SRC) - 1), k * XF_PIXELDEPTH(DEPTH_SRC));

        for (ap_uint<13> ch = 0, idx = 0; ch < XF_CHANNELS(SRC_T, NPC); ch++, idx += XF_DTPIXELDEPTH(SRC_T, NPC)) {
// clang-format off
      #pragma HLS unroll
            // clang-format on
            tmp_val1 = p.range(idx + (XF_DTPIXELDEPTH(SRC_T, NPC) - 1), idx);
            channel_out[ch] =
                ((tmp_val1 >= _lower_thresh[ch]) && (tmp_val1 <= _upper_thresh[ch])) ? (ap_uint<8>)255 : (ap_uint<8>)0;
        }
        if (XF_CHANNELS(SRC_T, NPC) != 1) {
            tmp_val.range(k * XF_PIXELDEPTH(DEPTH_DST) + (XF_PIXELDEPTH(DEPTH_DST) - 1), k * XF_PIXELDEPTH(DEPTH_DST)) =
                (channel_out[0] & channel_out[1] & channel_out[2]);
        } else {
            tmp_val.range(k * XF_PIXELDEPTH(DEPTH_DST) + (XF_PIXELDEPTH(DEPTH_DST) - 1), k * XF_PIXELDEPTH(DEPTH_DST)) =
                channel_out[0];
        }
    }
}
```

set in-range channel to 1

& all the channel and get the result

Kernel Functions - erode/dilate

❖ Pseudo code

```
void erode/dilate(src_mat,dst_mat,rows,cols,k_rows,k_cols){
    for(rows){
        for(cols){
            Process_function(...);
        }
    }
}
```

erode

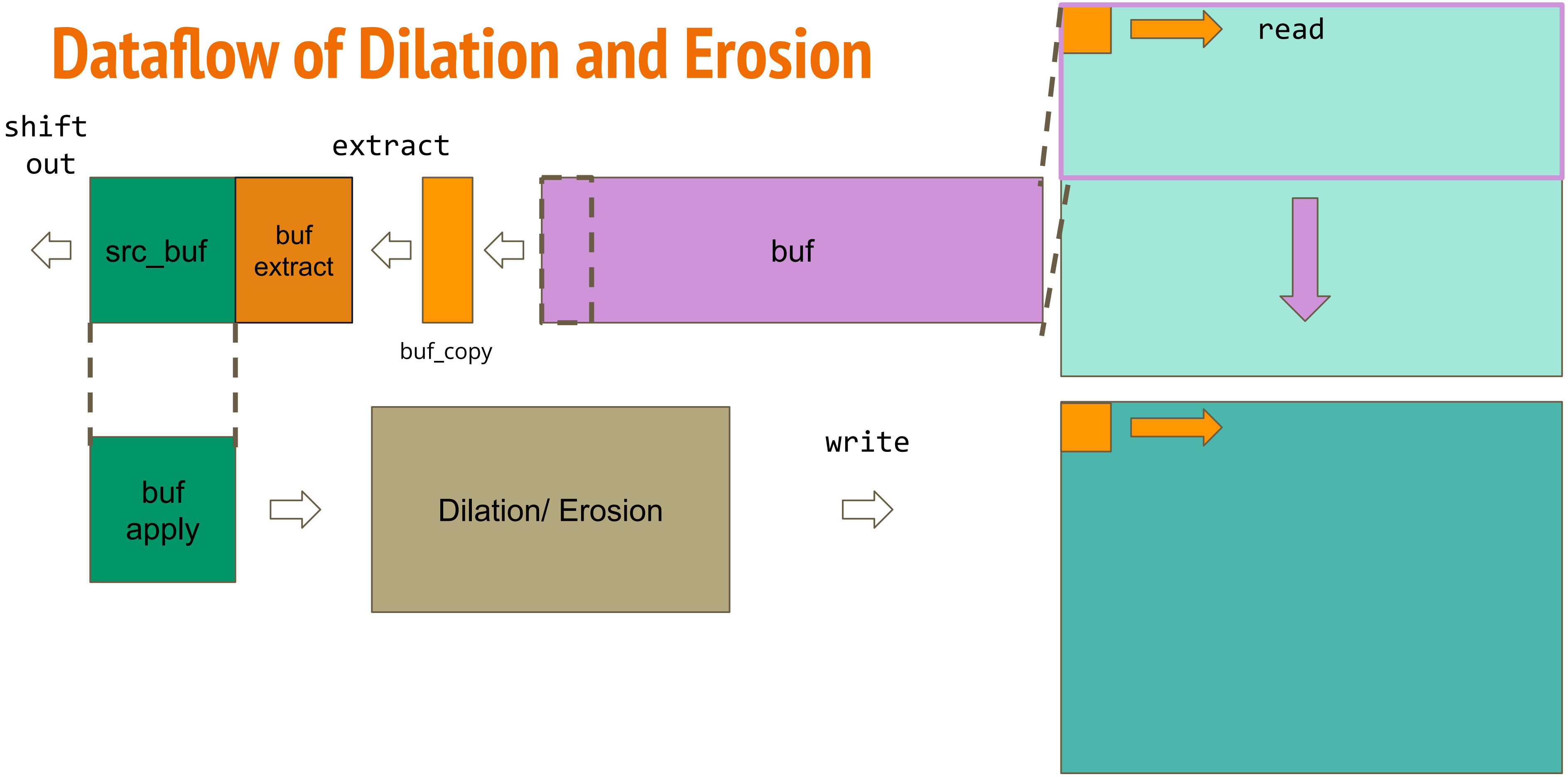
$$dst(x, y) = \min_{\substack{x-1 \le x' \le x+1 \\ y-1 \le y' \le y+1}} src(x', y')$$

dilate

$$dst(x, y) = \max_{\substack{x-1 \le x' \le x+1 \\ y-1 \le y' \le y+1}} src(x', y')$$

kernel erode and kernel dilate have same architecture, the only difference is function_apply in the process_function.

Dataflow of Dilation and Erosion



```
template <int ROWS, int COLS, int PLANES, int TYPE, int NPC, int WORDWIDTH, int TC, int K_ROWS, int K_COLS>
void Process_function(...) {
...
Col_Loop:
    for (ap_uint<13> col = 0; col < ((img_width) >> XF_BITSHIFT(NPC)) + col_loop_var;
         col++) // Image width should be multiple of NPC
    {
        #pragma HLS LOOP_TRIPCOUNT min=1 max=TC
        #pragma HLS pipeline
        #pragma HLS LOOP_FLATTEN OFF
        ...
        Row_Loop:
        for (int idx = 0; idx < K_ROWS; idx++) {
            #pragma HLS LOOP_TRIPCOUNT min=K_ROWS max=K_ROWS
            #pragma HLS UNROLL
            // clang-format on
            if (col < (img_width >> XF_BITSHIFT(NPC)))
                buf_cop[idx] = buf[(row_ind[idx])][col];
            else {
                buf_cop[idx] = 0; // buf_cop[idx];
                buf_cop[idx].b_not();
            }
        }
        //////////////////////  Actual Process function cal   ////////////////////////////////////
        for (int iter = 0; iter < npc; iter++) {
// clang-format off
            #pragma HLS pipeline
            // clang-format on
            for (int copyi = 0; copyi < K_ROWS; copyi++) {
                for (int copyj = 0; copyj < K_COLS; copyj++) {
                    src_buf_temp_med_apply[copyi][copyj] = src_buf[copyi][copyj + iter];
                }
            }
            function_apply<PLANES, TYPE, K_ROWS, K_COLS>(&OutputValue, src_buf_temp_med_apply,
                                                         kernel); // processing 8-pixels
            OutputValues[iter] = OutputValue;
        }
```

padding over bounding value with 0

copy data to kernel buffer from src

run function_applt & write back

erode: function_apply

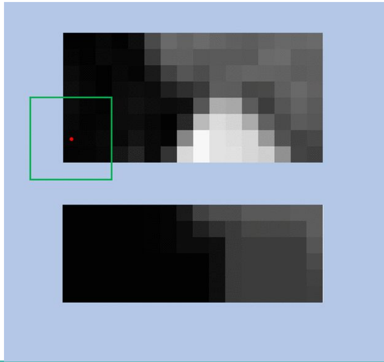
```
void function_apply(XF_PTUNAME(DEPTH) * OutputValue,
                    XF_PTUNAME(DEPTH) src_buf[K_ROWS][K_COLS],
                    unsigned char kernel[K_ROWS][K_COLS]) {
...

Apply_erode_Loop:
    for (ap_uint<5> c = 0, k = 0; c < PLANES; c++, k += depth) {
// clang-format off
        #pragma HLS LOOP_TRIPCOUNT min=1 max=PLANES
        #pragma HLS UNROLL
        // clang-format on
        XF_PTNAME(DEPTH) max = 255;
        for (ap_uint<13> k_rows = 0; k_rows < K_ROWS; k_rows++) {
// clang-format off
            #pragma HLS LOOP_TRIPCOUNT min=1 max=K_ROWS
            #pragma HLS UNROLL
            // clang-format on
            for (ap_uint<13> k_cols = 0; k_cols < K_COLS; k_cols++) {
// clang-format off
                #pragma HLS LOOP_TRIPCOUNT min=1 max=K_COLS
                #pragma HLS UNROLL
                // clang-format on
                if (kernel[k_rows][k_cols]) {
                    if (src_buf[k_rows][k_cols].range(k + (depth - 1), k) < max) {
                        max = src_buf[k_rows][k_cols].range(k + (depth - 1), k);
                    }
                }
            }
        }
        packed_val.range(k + (depth - 1), k) = max;
    }

    *OutputValue = packed_val;
    return;
}
```

$$dst(x, y) = \min_{\substack{x-1 \le x' \le x+1 \\ y-1 \le y' \le y+1}} src(x', y')$$

If any pixel in the kernel is 0,
output of this kernel would be set 0



dilate: function_apply

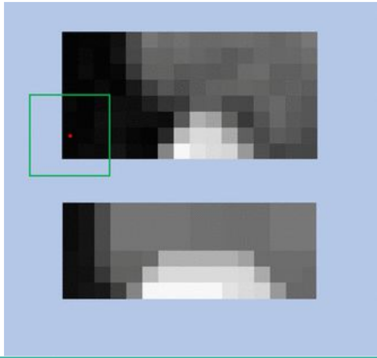
```
void dilate_function_apply(XF_PTUNAME(DEPTH) * OutputValue,
                           XF_PTUNAME(DEPTH) src_buf[K_ROWS][K_COLS],
                           unsigned char kernel[K_ROWS][K_COLS]) {
...

Apply_dilate_Loop:
    for (ap_uint<5> c = 0, k = 0; c < PLANES; c++, k += depth) {
// clang-format off
        #pragma HLS LOOP_TRIPCOUNT min=1 max=PLANES
        #pragma HLS UNROLL
        // clang-format on
        XF_PTNAME(DEPTH) max = 0;
        for (ap_uint<13> k_rows = 0; k_rows < K_ROWS; k_rows++) {
// clang-format off
            #pragma HLS LOOP_TRIPCOUNT min=1 max=K_ROWS
            #pragma HLS UNROLL
            // clang-format on
            for (ap_uint<13> k_cols = 0; k_cols < K_COLS; k_cols++) {
// clang-format off
                #pragma HLS LOOP_TRIPCOUNT min=1 max=K_COLS
                #pragma HLS UNROLL
                // clang-format on
                if (kernel[k_rows][k_cols]) {
                    if (src_buf[k_rows][k_cols].range(k + (depth - 1), k) > max) {
                        max = src_buf[k_rows][k_cols].range(k + (depth - 1), k);
                    }
                }
            }
        }
        packed_val.range(k + (depth - 1), k) = max;
    }

    *OutputValue = packed_val;
    return;
}
```

$$dst(x, y) = \max_{\substack{x-1 \le x' \le x+1 \\ y-1 \le y' \le y+1}} src(x', y')$$

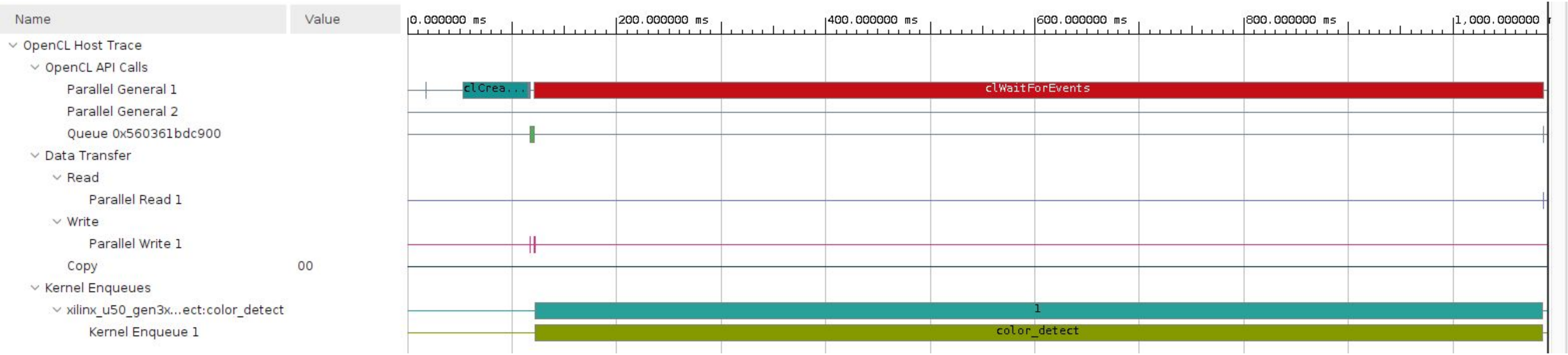
If any pixel in the kernel is 255,
output of this kernel would be set 255



Performance Analysis

SW_emulation

Profilie summary : Kernel execution time: 19015ms



HW_emulation

Profilie summary : Kernel execution time: 1.203ms

& Compute Units

ecution

	Enqueues	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)
_detect	1	19015.500	19015.500	19015.500	19015.500

el Execution

	Kernel Instance Address	Context ID	Command Queue ID	Device	Start Time (ms)	Duration (ms)
_detect	0x55a4a2344b10	0	0	xilinx_u50_gen3x16_xdma_201920_3-0	53.115	19015.500

Unit Utilization

t	Kernel	Device	Calls	Dataflow Execution	Max Parallel Executions	Dataflow Acceleration	CU Utilization (%)	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)	Clock Freq (MHz)
ect_1	color_detect	xilinx_u50_gen3x16_xdma_201920_3-0	1	No	0	1.000000x	100	19015.500	19015.500	19015.500	19015.500	300.000

Unit Stalls

No data. To generate kernel stall data, see Profiling the Application

		(ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)
		1.858	41.858	41.858	41.858
		0.010	0.010	0.010	0.010
		0.779	0.779	0.779	0.779
		0.326	0.326	0.326	0.326
clEnqueueWriteBuffer	1	0.233	0.233	0.233	0.233
clFinish	1	0.001	0.001	0.001	0.001
clGetDeviceIDs	2	0.002	0.000	0.001	0.001
clGetDeviceInfo	2	0.008	0.001	0.004	0.006
clGetExtensionFunctionAddress	2	0.016	0.002	0.008	0.015
clGetExtensionFunctionAddressForPlatform	2	0.005	0.001	0.003	0.004
clGetPlatformInfo	6	0.003	0.000	0.000	0.001
clReleaseCommandQueue	1	0.002	0.002	0.002	0.002
clReleaseContext	1	1.375	1.375	1.375	1.375
clReleaseDevice	2	0.002	0.001	0.001	0.002
clReleaseEvent	1	0.024	0.024	0.024	0.024
clReleaseKernel	1	0.208	0.208	0.208	0.208
clReleaseMemObject	11	0.007	0.000	0.001	0.002
clReleaseProgram	1	0.061	0.061	0.061	0.061
clRetainDevice	2	0.001	0.000	0.000	0.001
clRetainMemObject	6	0.004	0.000	0.001	0.002
clSetKernelArg	7	0.013	0.000	0.002	0.007
clWaitForEvents	1	19018.300	19018.300	19018.300	19018.300

Writes

ess	Context ID	Command Queue ID	Start Time (ms)	Duration (ms)	Buffer Size (KB)	Writing Rate (MB/s)
	0	0	51.223	N/A	1,041.35	N/A
	0	0	50.891	N/A	0.027	N/A
	0	0	50.759	N/A	1,041.32	N/A

Reads

ess	Context ID	Command Queue ID	Start Time (ms)	Duration (ms)	Buffer Size (KB)	Reading Rate (MB/s)
	0	0	19069.600	N/A	347.108	N/A

Kernels & Compute Units

Kernel Execution (includes estimated device times)

Kernel	Enqueues	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)
color_detect	1	1.203	1.203	1.203	1.203

Top Kernel Execution

Kernel	Kernel Instance Address	Context ID	Command Queue ID	Device	Start Time (ms)	Duration (ms)
color_detect	0x55f83a255470	0	0	xilinx_u50_gen3x16_xdma_201920_3-0	0.121	1.203

Host Data Transfers

Host Transfer

Context:Number of Devices	Transfer Type	Number of Buffer Transfers	Transfer Rate (MB/s)	Avg Bandwidth Utilization (%)	Avg Size (KB)	Total Time (ms)	A T
context0:1	READ	1	224.240	N/A	347.108	N/A	N
context0:1	WRITE	3	0.916	N/A	694.234	N/A	N

Top Memory Writes

Buffer Address	Context ID	Command Queue ID	Start Time (ms)	Duration (ms)	Buffer Size (KB)	Writing Rate (MB/s)
0x101000	0	0	21799.700	N/A	1,041.35	N/A
0xff000	0	0	21791.200	N/A	0.027	N/A
0x0	0	0	21791.000	N/A	1,041.32	N/A

Top Memory Reads

Buffer Address	Context ID	Command Queue ID	Start Time (ms)	Duration (ms)	Buffer Size (KB)	Reading Rate (MB/s)
0x102000	0	0	142077.000	N/A	347.108	N/A

API Calls

OpenCL API Calls

API Name	Calls	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)
clCreateBuffer	5	1.212	0.073	0.242	0.635
clCreateCommandQueue	1	0.002	0.002	0.002	0.002
clCreateContext	1	0.040	0.040	0.040	0.040
clCreateKernel	1	1.382	1.382	1.382	1.382
clCreateProgramWithBinary	1	21752.700	21752.700	21752.700	21752.700
clEnqueueMigrateMemObjects	1	0.005	0.005	0.005	0.005
clEnqueueReadBuffer	1	1.611	1.611	1.611	1.611
clEnqueueTask	1	0.028	0.028	0.028	0.028
clEnqueueWriteBuffer	1	0.240	0.240	0.240	0.240
clFinish	1	0.002	0.002	0.002	0.002
clGetDeviceIDs	2	0.002	0.000	0.001	0.001
clGetDeviceInfo	2	0.007	0.001	0.003	0.006
clGetExtensionFunctionAddress	2	0.015	0.002	0.007	0.013
clGetExtensionFunctionAddressForPlatform	2	0.004	0.001	0.002	0.004
clGetPlatformInfo	6	0.002	0.000	0.000	0.001
clReleaseCommandQueue	1	0.003	0.003	0.003	0.003
clReleaseContext	1	9835.720	9835.720	9835.720	9835.720
clReleaseDevice	2	0.004	0.001	0.002	0.003
clReleaseEvent	1	0.018	0.018	0.018	0.018
clReleaseKernel	1	0.688	0.688	0.688	0.688

IP Synthesis Report

```
================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+---------+----------+------------+
    |  Clock |  Target | Estimated| Uncertainty|
    +--------+---------+----------+------------+
    |ap_clk  |  3.33 ns|  3.650 ns|     0.90 ns|
    +--------+---------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+----------+----------+-----+---------+----------+
    |  Latency (cycles) |  Latency (absolute) |    Interval    | Pipeline |
    |   min   |   max   |    min   |    max   | min |   max   |   Type   |
    +---------+---------+----------+----------+-----+---------+----------+
    |      115|  8328075| 0.420 us | 30.394 ms|   88|  8328014| dataflow |
    +---------+---------+----------+----------+-----+---------+----------+
```

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+---------------------+---------+------+---------+--------+-----+
|         Name        | BRAM_18K|  DSP |    FF   |   LUT  | URAM|
+---------------------+---------+------+---------+--------+-----+
|DSP                  |        -|     -|        -|       -|    -|
|Expression           |        -|     -|        0|      88|    -|
|FIFO                 |        -|     -|      587|     705|    -|
|Instance             |       37|    12|    11673|   17679|    0|
|Memory               |        -|     -|        -|       -|    -|
|Multiplexer          |        -|     -|        -|     162|    -|
|Register             |        -|     -|       21|       -|    -|
+---------------------+---------+------+---------+--------+-----+
|Total                |       37|    12|    12281|   18634|    0|
+---------------------+---------+------+---------+--------+-----+
|Available SLR        |     1344|  2976|   871680|  435840|  320|
+---------------------+---------+------+---------+--------+-----+
|Utilization SLR (%)  |        2|    ~0|        1|       4|    0|
+---------------------+---------+------+---------+--------+-----+
|Available            |     2688|  5952|  1743360|  871680|  640|
+---------------------+---------+------+---------+--------+-----+
|Utilization (%)      |        1|    ~0|       ~0|       2|    0|
+---------------------+---------+------+---------+--------+-----+
```

Debugging Process

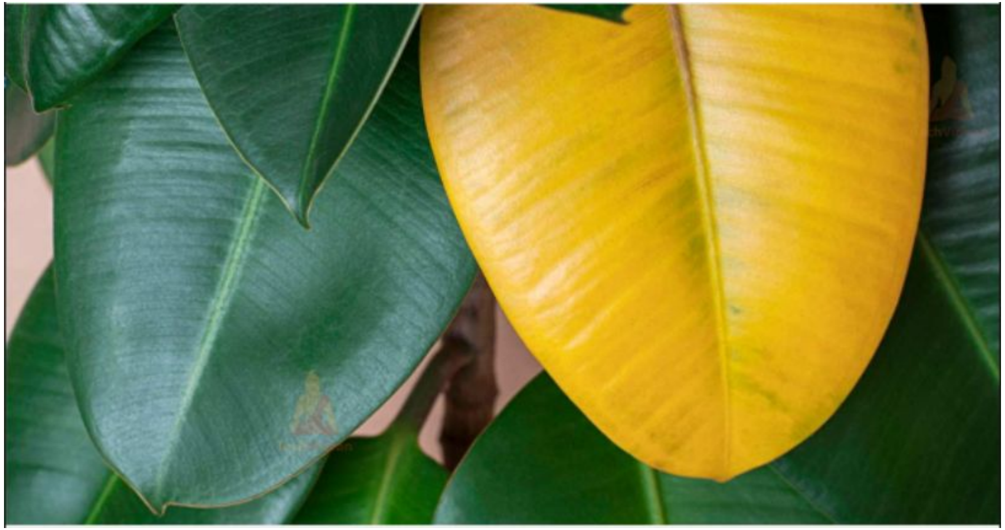
- Can't find opencv path:
  - solution: `$(SYSROOT)/usr/include/opencv4`
- Can't run hardware FPGA run
  - `Maybe ask server manager?`

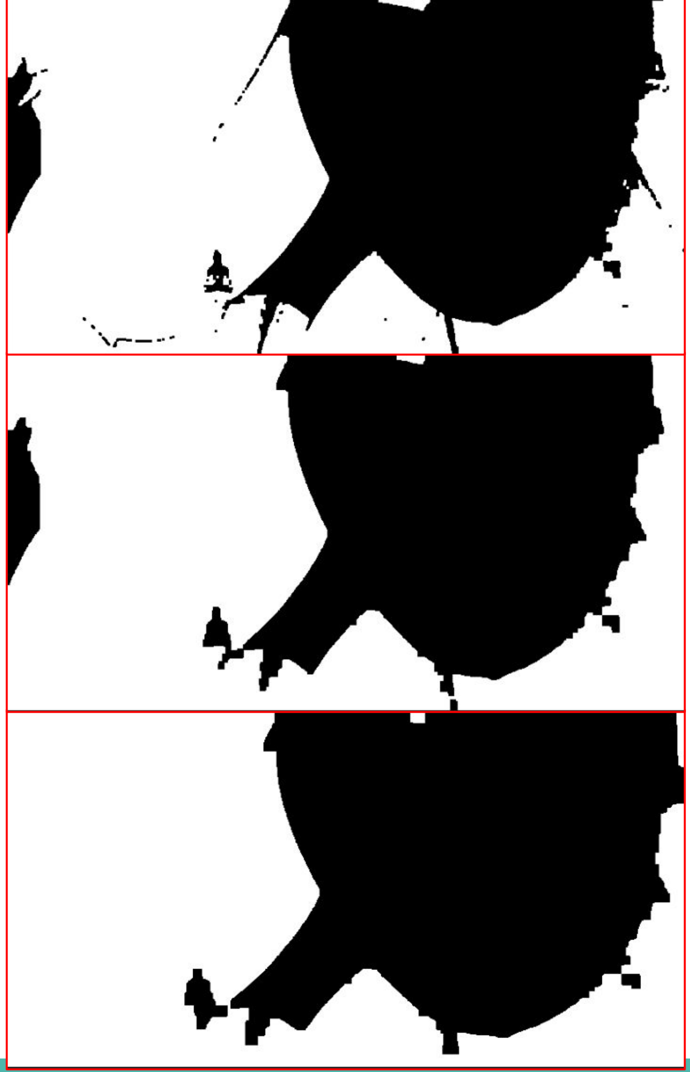
```
[XRT] WARNING: XRT
[XRT] ERROR: Permission denied Device index 0
[XRT] ERROR: Could not open device
/mnt/HLSNAS/nycuhls11/vitis/Vitis_Libraries/vision/L3/examples/colordetect/xf_colordetect_tb.cpp:182 Error calling cl::Context context(device, NULL, NULL, NULL
, &err), error code is: -6
[XRT] WARNING: Profiling may contain incomplete information. Please ensure all OpenCL objects are released by your host code (e.g., clReleaseProgram()).
[XRT] WARNING: XRT
[XRT] ERROR: Permission denied Device index 0
terminate called after throwing an instance of 'std::runtime_error'
  what():  Could not open device
make: *** [Makefile:352: run] Aborted (core dumped)
```

The Result of Color Detection

- Yellow threshold : [50,20,20] - [100,255,255]
- Increase **FILTER_SIZE** from 3,7,10

The erodsion and dilation can clean image better.

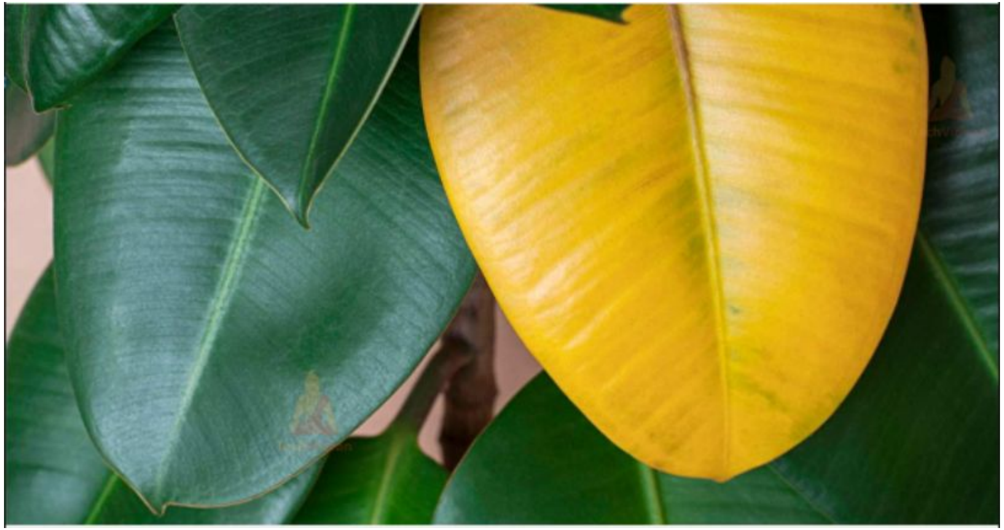


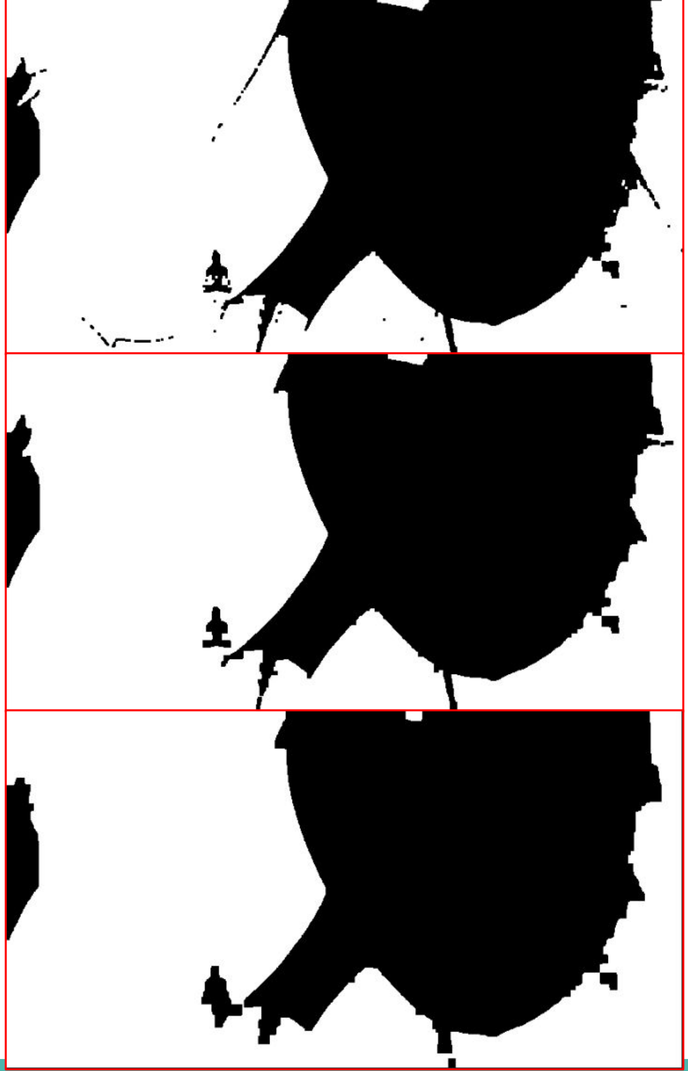


The Result of Color Detection

- **ITERATIONS** : 1,2,4

The erodsion and dilation can clean image better.





Suggestion for Improvement-1

❖ Setting NPC argument of kernel erode and kernel dilate to accelerate.

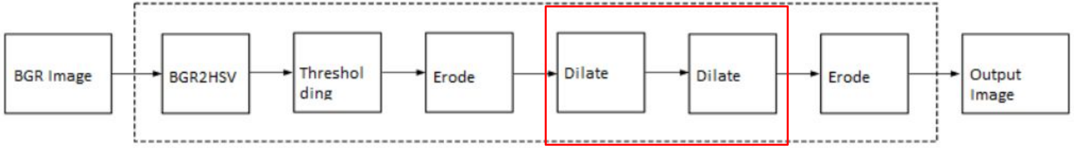
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.

Operating Mode	Latency Estimate	
	Min Latency (ms)	Max Latency (ms)
1 pixel operation (300 MHz)	7.0	7.0
8 pixel operation (150 MHz)	1.85	1.85

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	3	6
DSP48E	0	0
FF	411	657
LUT	392	1249
CLB	96	255

Suggestion for Improvement-2

❖ Change the iterations number seperately.




```
// Use erode and dilate to fully mark color areas:
xf::cv::erode<XF_BORDER_CONSTANT, OUT_TYPE, HEIGHT, WIDTH, XF_KERNEL_SHAPE, FILTER_SIZE, FILTER_SIZE, ITERATIONS,
              NPC1>(imgHelper1, imgHelper2, _kernel);
xf::cv::dilate<XF_BORDER_CONSTANT, OUT_TYPE, HEIGHT, WIDTH, XF_KERNEL_SHAPE, FILTER_SIZE, FILTER_SIZE, ITERATIONS,
              NPC1>(imgHelper2, imgHelper3, _kernel);
xf::cv::dilate<XF_BORDER_CONSTANT, OUT_TYPE, HEIGHT, WIDTH, XF_KERNEL_SHAPE, FILTER_SIZE, FILTER_SIZE, ITERATIONS,
              NPC1>(imgHelper3, imgHelper4, _kernel);
xf::cv::erode<XF_BORDER_CONSTANT, OUT_TYPE, HEIGHT, WIDTH, XF_KERNEL_SHAPE, FILTER_SIZE, FILTER_SIZE, ITERATIONS,
              NPC1>(imgHelper4, imgOutput, _kernel);
```

```
xf::cv::erode<XF_BORDER_CONSTANT, OUT_TYPE, HEIGHT, WIDTH, XF_KERNEL_SHAPE, FILTER_SIZE, FILTER_SIZE, ITERATIONS,
              NPC1>(imgHelper1, imgHelper2, _kernel);
xf::cv::dilate<XF_BORDER_CONSTANT, OUT_TYPE, HEIGHT, WIDTH, XF_KERNEL_SHAPE, FILTER_SIZE, FILTER_SIZE, 2*ITERATIONS,
          NPC1>(imgHelper2, imgHelper3, _kernel);
xf::cv::erode<XF_BORDER_CONSTANT, OUT_TYPE, HEIGHT, WIDTH, XF_KERNEL_SHAPE, FILTER_SIZE, FILTER_SIZE, ITERATIONS,
          NPC1>(imgHelper4, imgOutput, _kernel);
```

Suggestion for Improvement-3

- IP utilization is very low using dataflow(<1%) => Not good for FPGA.
  - Dataflow can deal with arbitrary input size and fixed hardware utilization
    - However, latency grows up by the size of input

If the input size is known and the memory is also sufficient

- Optimization
  - Treat image as a big matrix
  - Using pipline or unroll mechanism to calculate mutiple pixels at same time.
  - Fully utilize FPGA resource to get the best performance