COMPUTER VISION NOTES
OBJECT LOCALISATION AND TRACKING

My personal notes on

Object Localisation Techniques; Colour Matching, Mean Shift Tracking, Optical Flow, Lukas Kanade

By

0xLeo (github.com/0xleo)

March 3, 2019

DRAFT X.YY
Missing: . . .

Contents

1 Histogram-Based Methods

1.1 Histogram backprojection

1.1.1 Intuition - model and search image histogram

In image processing, we are usually interested in histograms of greyscale images. However, often the colour histogram can be used to identify an image region or object. RGB histograms are practically not good enough for matching as the R, G, B components are strongly correlation with the illumination hitting the object. In practice, objects are converted from RGB to HSV (Hue, Saturation, Value) domain. Hue represents the colour type (blue, yellow, etc.), saturation represents the vibrancy (how vivid or neutral it is) and value represents the brightness of the colour. Hence HSV decouples the brightness from the colour description. Therefore when performing colour matching we are only interested in the H and S components, which map to a 2D histogram. More about the HSV domain in A.1.

> ☞ **Comment:** The HS components are often but *not always* a good choice for colour-based detection. They may fail detecting black and white objects since black and white can have any colour (H) and in this case the SV components of the HSV or even the YUV domain are a better choice. However, in this article we stick to HS.

Histogram backprojection answers the question "where in the image are the colours that belong to the object being looked for?". We do this by defining a model image (the object we search for – a. k.a. target) and the search (the whole image where we search in), probing the model over search image and calculating their histogram similarity at each position.

Just to illustrate the idea, assume that we want to match the greyscale (instead of the 2D) histogram of the garlic in Fig. 1. A part of the top garlic has been chosen as the model. The histogram of the model is shown as well as that of two matching candidates. In this case, the histogram of "match 2" is more similar to the model's than one "match 1" so we want somehow to register that similarity. The question attempted to be answered in the next section is "how do we measure the similarity of the histogram of the matching candidate to that of the model?".

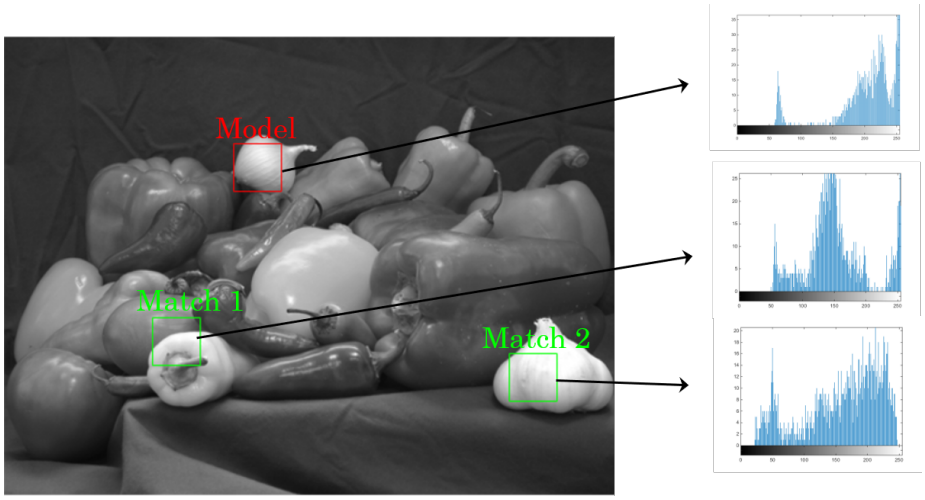

Fig. 1. Model and two matches' greyscale histograms.

1.1.2 (optional) The rationale behind defining the ratio histogram

We assume that

1. the model's histogram (the object we search for) is narrow and tall
2. and the scene's (whole input image) histogram is rather wide.

It has then been proven (it won't be discussed in this article how as the how is out of scope) that a good histogram similarity measure between the model M and a patch of the image we search it in I is the ratio histogram R. M and I need to be pre-calculated and can be divided element-wise, since they have the same range and bins to obtain R. The ratio histogram is a therefore function $R : \mathbb{Z}^2 \to \mathbb{R}$ that maps a colour (h,s) to some value. If $M(h,s) > I(h,s) \Rightarrow R(h,s) > 1$ that means the model has more pixels of colour (h,s) relative to its total number

of pixels compared to $I(h,s)$, and vice versa. If $R(h,s)=1$, then that means that the input and model images contain (h,s) at the same degree.

However, because of assumptions (1), (2), $M(h,s) > I(h,s)$ can happen quite often and so long as $M(h,s) > I(h,s) \Rightarrow R(h,s) > 1$, e.g. $R(h,s) = 2,3,10$, the exact value does not give much useful information. To summarise, R de-emphasises pixels with colours that do not belong to the model and emphasises the rest.

1.1.3 A colour matching heuristic

From the previous section, the conclusion is that it is desirable to clip the ratio histogram to 1, as defined by Swain et al.

DEFINITION 1.1. *For each bin j, the ratio histogram is defined as*

$$R_j = \min\left(\frac{M_j}{I_j}, 1\right) \tag{1.1}$$

, where M, I are the model's and input's histograms respectively.

Note that j does not necessarily have to be a pair (h,s), but it if a histogram bin (index) is quantised it can be a rectangle in the 2D space, such as $[20,39]\times[50,69]$. As mentioned before, R associates a colour with its probability of appearing in the model and the next step is the associate each pixel with that probability.

Each pixel of the original image at (x,y) maps to a 2D HS value, by a colour function $c:\mathbb{Z}^2 \leftarrow \mathbb{Z}^2$, by taking $c(x,y)$. Sometimes need an intermediate function $h:\mathbb{Z}^2 \rightarrow \mathbb{Z}^2$ that takes the output of c and quantises it (groups multiple colours in one bin), before it is fed to R. For example, h could convert $[0,1,\ldots,179]\times[0,1,\ldots,255]$ to $[0,19,39,\ldots,179]\times[0,24,49,\ldots,255]$. The output of h is bed to R, which divides M_j to I_j at each bin j.To summarise this paragraph we have defined the following functions in backpropagation:

- $c:\mathbb{Z}^2 \rightarrow \mathbb{Z}^2$: maps a pixel at (x,y) to an HS value (h_j, s_j).
- $h:\mathbb{Z}^2 \rightarrow \mathbb{Z}^2$ maps a set of values $(h_i, s_i, h_{i+1}, s_{i+1}, \ldots, h_n, s_n)$ to another (h,s) value by having quantised the range of h and s.
- $R:\mathbb{Z}^2 \rightarrow [0,1]$ maps an (h,s) value to a probablity.

We therefore want to create a new image b where each pixel (x,y) gets assigned its output of R - the measure of how much its colour appears in the model image.

$$b(x,y) := R\left(h\left(c(x,y)\right)\right) = \min\left(\frac{M\left(h\left(c(x,y)\right)\right)}{I\left(h\left(c(x,y)\right)\right)}, 1\right) \;\forall\, x,y \tag{1.2}$$

The final step is to find compact regions where b is high. If the shape of the object (model) to detect is generic, then this can be done by convolving b with binary disk mask D^r of radius r. Define:

$$D^r_{x,y} = \begin{cases} 1 & \sqrt{x^2+y^2} \le r \\ 0 & \text{otherwise} \end{cases} \tag{1.3}$$

Then the probability image b can be convolved with the mask:

$$b := D^r * b \tag{1.4}$$

The argmax function to returns the pixel (x,y) with the maximum value of its argument, i.e. where the R matrix and the $*$ symbol denotes convolution. Then Histogram Backprojection can be written

Algorithm 1 Colour matching by histogram backprojection according to Swain et al

procedure HIST-BACKPROJ(ImM, ImI) ▷ ImM: model, ImI: search image
  $M \leftarrow \text{histogram}(ImM)$
  $I \leftarrow \text{histogram}(ImI)$
  for each histogram bin j **do** ▷ a bin is a pair (h,s)
    $R_j = \min\left(\frac{M_j}{I_j}, 1\right)$ ▷ Divide element-wise
  $m \leftarrow rows(M)$
  $n \leftarrow cols(M)$
  $b \leftarrow empty_{m \times n}$
  for y in 0...m-1 **do**
    for x in 0...n-1 **do**
      $b_{x,y} \leftarrow R(h(c(x,y)))$ ▷ b matrix of colour probability
  $D^r \leftarrow$ binary disk of radius r
  $b \leftarrow D^r * b$ ▷ Group (by convolving) high probability pixels together.
  $x_{obj}, y_{obj} \leftarrow \arg\max_{x,y}(b)$
  return x_{obj}, y_{obj}

1.1.4 Histogram backprojection implementation from scratch

An implementation of Alg. 1 has been written in A.2. However, instead of finding the location of the object by the argmax function, it applies Otsu's threshold on the R matrix. This automatically selects a threshold T based on the statistics of the histogram of R for which if $R[x,y] < T$, then the pixel at (x,y) is classified as background, else as foreground. Instructions on how to run the implementation code are in A.2 and an output is shown below.

Fig. 2. Input image with a ROI of the objects to detect selected.

Fig. 3. Detected objects on the original image.

1.1.5 Histogram backprojection implementation using OpenCV's API

OpenCV implements the technique using the `cv2.calcBackProject(image, channels, histohram_array, channel_ra` method (in Python). Its invocation looks like:
`cv2.calcBackProject(search_image, channels, model_histogram, channel_ranges, [scale = 1])`

- `search_image`: the input image, e.g. in HSV.
- `channels`: which channels of the original image and the model to select in order to draw its histogram, e.g. `channels = [0,1] -> H, S`.
- `model_histogram`: histogram of the model (ROI), needs to be pre-calculated.
- `channel_ranges`: set it to `[0,180,0,256]` to select the full range of H, S components.

The code listing in A.3 works similarly with the one in A.2, expecting two clicks from the user to define a bounding box around a sample of the object to detect. It also performs similarly on the same images, showing some black spots on roughly the same positions.

1.1.6 Histogram backprojection summary

- ✓ Fast – can easily be used in real time.
- ✓ Relatively immune to noise and illumination changes.
- ✓ Simple to implement.
- ✗ Not effective against non-compact objects.
- ✗ Does not use any knowledge about the shape or position of the detected object – only its colour.

1.2 Mean Shift Tracking

2 Motion-based methods

2.1 Optical Flow

2.2 Lukas-Kanade tracking

3 Edge-based methods

3.1 Hough Transform

A Appendices

A.1 HSV domain

A.2 Histogram implementation from scratch - source code

```
import cv2, numpy as np
import sys
import pdb

g_clicks_xy = []

def qimshow(im, delay = 10, wname = 'display'):
    cv2.imshow(wname,im)
    cv2.waitKey(delay * 1000)
    cv2.destroyAllWindows()


def on_click(event, x, y, flags, param):
    """
        Mouse callback function - write to global list of clicks
    """
    global g_clicks_xy
    if event == cv2.EVENT_LBUTTONDOWN:
        g_clicks_xy.append((x, y))

"""
@im: The input image as read (in BGR)
"""
def get_model(im):
    """
        Process user input (clicks) and Extract the target image
        (model) in hsv.
    """
    global g_clicks_xy
    hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
    cv2.namedWindow('input')
    cv2.setMouseCallback('input', on_click)
    while True:
        cv2.imshow('input', im)
        k = cv2.waitKey()
        if k == ord('q'):
            break
    cv2.destroyAllWindows()
    # for clicks, index 0 = x, index 1 = y
    g_clicks_xy = sorted(g_clicks_xy,
            key = lambda x: x[0]**2 + x[1]**2,
            reverse = True) [-2:]
    click_br, click_tl = g_clicks_xy[0], g_clicks_xy[1]
    w = click_br[0] - click_tl[0]
    h = click_br[1] - click_tl[1]
    hsvt = hsv[click_tl[1]: click_tl[1] + h,
            click_tl[0]: click_tl[0] + w]
    rect = cv2.rectangle(im.copy(), g_clicks_xy[0], g_clicks_xy[1],
            color = (0,255,0))
    qimshow(rect, wname = 'selection', delay = 3)
    return hsvt


"""
@hsv: The whole input (search) image in hsv
@hsvt: The target (model), i.e. the ROI, in hsv
@<return>: The ration histogram of hsvt by hsv, clipped from 0 to 1
```

```
"""
def ratio_histogam(hsv, hsvt):
    # see doc: HS histograms, [0, 180] as in OpenCV 0 <= H <= 179
    M = cv2.calcHist([hsv], [0, 1], None, [180, 256], [0, 180, 0, 256])
    I = cv2.calcHist([hsvt], [0, 1], None, [180, 256], [0, 180, 0, 256])
    R = np.divide(np.array(M, np.float),
            np.array(I, np.float),
            out = np.zeros_like(I),
            where = I != 0)
    R[R > 1.0] = 1.0
    return R


"""
@hsv: the whole input image in HSV
@R: the ratio histogram as returned by the ratio_histogram function
@r: the radius of the disk backprojection convolves with
"""
def backproject(hsv, R, rad = 15):
    """
        Generate a 2D binary image where ones are probably the object(s)
        of interest and zeros the background
    """
    b = np.zeros((hsv.shape[0], hsv.shape[1]), np.uint8)
    for r in range(hsv.shape[0]):
        for c in range(hsv.shape[1]):
            b[r,c] = R[hsv[r,c][0], hsv[r,c][1]]
    b = np.uint8(b)
    disk = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (rad, rad))
    # convolve with disk
    b = cv2.filter2D(b, -1, disk, b)
    b = np.uint8(b)
    b = np.array(cv2.normalize(b, b, 0, 255, cv2.NORM_MINMAX), np.uint8)
    _, thresh = cv2.threshold(np.uint8(b), 0, 255,
            cv2.THRESH_BINARY | cv2.THRESH_OTSU )
    return thresh


def usage():
    str_usage = 'Run with $ python <program_name> <input_image>\n'\
    'When your input image shows up, click 2 times to define\n'\
    'a bounding box around  a sample of your object of interest.\n'\
    'Then press "q" to finish the selection.'
    print str_usage
    sys.exit(0)


def main():
    assert len(sys.argv) > 1,\
        "Run the program with -h or --help to print usage"
    if sys.argv[1] == '-h' or sys.argv[1] == '--help':
        usage()
    else:
        im = cv2.imread(sys.argv[1])
    hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
    hsvt = get_model(im)
    R = ratio_histogam(hsv, hsvt)
    thresh = backproject(hsv, R)
    # final processing - AND the 2D binary threshold image with the
```

```
    # original RGB input image to show the result
    thresh = cv2.merge((thresh, thresh, thresh))
    res = cv2.bitwise_and(im, thresh)
    qimshow(res)


if __name__ == '__main__':
    main()
```

A.3 Histogram implementation using OpenCV - source code

```
import cv2
import numpy as np
import sys


g_clicks_xy = []

def qimshow(im, delay = 10, wname = 'display'):
    cv2.imshow(wname,im)
    cv2.waitKey(delay * 1000)
    cv2.destroyAllWindows()


def on_click(event, x, y, flags, param):
    """
        Mouse callback function - write to global list of clicks
    """
    global g_clicks_xy
    if event == cv2.EVENT_LBUTTONDOWN:
        g_clicks_xy.append((x, y))

def check_args():
    assert len(sys.argv) == 2,\
        "the program expects exactly 1 argument (image file)"

def input_rectangle(im):
    im_cpy = im.copy()
    try:
        cv2.imshow('Original', im)
        cv2.waitKey()
        cv2.destroyAllWindows()
        print im.shape
    except:
        print "Cannot read image supplied"
        return
    x0 = input("x0: ")
    y0 = input("y0: ")
    x1 = input("x1: ")
    y1 = input("y1: ")
    cv2.rectangle(im_cpy, (x0,y0), (x1,y1), (0,0,255), 2)
    cv2.imshow('Original w/ ROI', im_cpy)
    cv2.waitKey()
    cv2.destroyAllWindows()
    return [x0, y0, x1, y1]

def main():
        im = cv2.imread(sys.argv[1]) # input image
        cv2.namedWindow('select area')
        cv2.setMouseCallback('select area', on_click)
        while True:
            cv2.imshow('select area', im)
            k = cv2.waitKey()
            if k == ord('q'):
                break
        cv2.destroyAllWindows()
        hsv = cv2.cvtColor(im,cv2.COLOR_BGR2HSV)
        click_br, click_tl = g_clicks_xy[0], g_clicks_xy[1]
```

```
        w = click_br[0] - click_tl[0]
        h = click_br[1] - click_tl[1]
        roi = im[click_tl[1]: click_tl[1] + h,
            click_tl[0]: click_tl[0] + w]
        target_hsv = cv2.cvtColor(roi2, cv2.COLOR_BGR2HSV)
        # Calculating object histogram
        # In OpenCV, hue lies within [0,179]!
        roihist = cv2.calcHist([target_hsv],    # image\
                [0, 1],                         # channel selection\
                None,                           # mask\
                [90, 128],                      # no of bins\
                [0, 180, 0, 256]                # channel ranges\
                )
        # normalize histogram to [0,255] range and apply backprojection
        cv2.normalize(roihist, roihist, 0, 255, cv2.NORM_MINMAX)
        dst = cv2.calcBackProject([hsv],  # image\
                [0,1],                          # channel selection\
                roihist,                        # histogram array\
                [0,180,0,256],                  # channel ranges\
                scale = 1)
        # Now convolute with circular disc
        disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5,5))
        cv2.filter2D(dst, -1, disc,dst)
        # Otsu's threshold
        _, thresh = cv2.threshold(dst, 0, 255,  cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        # Make it 3D to AND it with the model image
        thresh = cv2.merge((thresh, thresh, thresh))
        res = cv2.bitwise_and(im,thresh)

        cv2.imshow('Original', im)
        cv2.imshow('Result', res)
        cv2.waitKey()
        cv2.destroyAllWindows()
        # res = np.vstack((model, thresh,res))
        # cv2.imwrite('res.jpg',res)

if __name__ == '__main__':
    main()
```