Background Subtraction using Local SVD Binary Pattern

Franci Suni Lopez
Universidad Católica San Pablo-Arequipa Peru
franci.suni@ucsp.edu.pe
Joel Oswaldo Gallegos Guillen
Universidad Católica San Pablo-Arequipa Peru
joel.gallegos@ucsp.edu.pe

***Abstract*—Background subtraction is a technique widely used to detect moving objects from static cameras. In the field of video surveillance, it is essential since it allows rapid discrimination when an event has arisen and studies of mobility of objects on the scene. In the last few years, many methods with different strengths and requirements have been proposed. For instance, background subtraction using local SVD binary pattern was a proposed method combining different techniques.**

In this paper we study, replicate and implement the paper before mentioned, and we compare our results with the paper dataset (CDnet 2012). Our results indicate that LSBP does not work well in environments with shadow and in two datasets (copyMachine and sofa) our implementation are better than original paper.

***Keywords*-camera calibration,**

I. Introduction

Background subtraction is a fundamental stage of a large number of applications in which it is necessary to detect movement, identify and / or follow objects in sequences of dynamic images. Among these applications, video surveillance can be named, such as the detection and capture of movement flow, pose estimation, man-computer interaction or video coding based on content, among others. The process of subtraction is also called foreground extraction or foreground extraction, and consists of a series of methods that allow distinguishing between background or static zones (background), and dynamic zones that correspond to the foreground.

In order to extract the background, a representation of the scene is maintained through the digital modeling of the background. This representation must contain the necessary information for each moment to determine, for each new entry image, which areas of the same correspond effectively with the background. Since the background is not invariant in time, it is necessary to perform a dynamic modeling of the background to adapt it to the variations that may occur in it. The first time, any significant change in a region of the image with respect to the background model will be considered foreground. In general, each element of the model is updated for each certain time interval, as long as it has lost significance.

II. Local SVD Binary Pattern

Local Binary Pattern (LBP) is a powerful and fast local image descriptor for analyzing textures. However, LBP is not robust to local image noises. Therefore, the authors add Singular value decomposition (SVD) because is utilized in noise filtering. They presents that SVD coefficients (i.e., singular values) are likely to reveal the illumination-invariant characteristics. Therefore, they try to apply SVD to local regions and they define block B (centered by(x, y)) of MxM pixels:

$$B(x,y) = U \sum V^T$$

where U and V are orthogonal matrixes; $\sum$ = diag $(\lambda_1, \lambda_2, ..., \lambda_n)$ is a nonnegative diagonal matrix with decreasing entries along the diagonal, it is called the singular values of B(x, y). They use blocks of $3x3$ and at each pixel on position B(x,y), they define this scalar for a given frame as:

$$g(x,y) = \sum_{j=2}^{M} \hat{\lambda_j}, \text{ and } \hat{\lambda_j} = \lambda_j / \lambda_1$$

where $\hat{\lambda_j}$ indicates the jth singular value.

The principle of LSBP is to compare a central point value with neighbor values and check whether they are similar or not. And local structural invariant is applied for central point value and neighbor values. Texture at point (x_c, y_c) is modeled using a local neighborhood of radius R, which is sampled at P points. The LSBP binary string at a given location (x_c, y_c) can be derived from the following formula:

$$LSBP(x_c, y_c) = \sum_{p=0}^{p-1} s(i_p, i_c) 2^p$$

where i_c is the central point value, i_p represents the N-neighborhood point value. τ is the similarity threshold which is set to 0.05 in the original paper. $S(.)$ is a sign function defined as follows:

$$s(i_p, i_c) = \begin{cases} 0 & \text{if } |i_p - i_c| \leq \tau \\ 1 & \text{otherwise} \end{cases}$$

Finally, to classify a pixel at coordinate (x, y), the current frame should be matched against their samples. The pixel value $Int(x, y)$ and $LSBP(x, y)$ value are both need to be matched correctly. We call this combined verification.

$$(H(LSBP(x, y), BLSBP_{index}(x, y)) \leq H_{LSBP})$$

$$\&\&(L1dist(Int(x, y), BInt_{index}(x, y)) < R(x, y))$$

$\#min$ is the minimum count of matches needed for classification. They fixed $\#min = 2$ because it is a reasonable balance between noise resistance and computational complexity. For LSBP comparison, they use Hamming distance (XOR) and they select simpler L1 distance for the color intensity comparison.

III. Prerequisites to run available source code

The source code was implemented on ***python***, therefore we need the python compiler installed on our computer. Initially, we installed the the prerequisites on Ubuntu 16.04, but we can not run the code because there was a problem of compatibility between the *numpy* and *maxflow* libraries. Then we install Anaconda2 version 4.4.0 on windows 7 and also install *maxflow* library manually. Finally we run the source code with the following script:

```
python local_svd_2016
-g  highway\groundtruth
-f  highway\input
-k  1700
```

Where *-g* is the Directory with ground-truth frames, *-f* is the directory with input frames and *-k* calculates answer for K-th frame and output.

IV. Match between code and paper

The algorithm proposed by the paper can be summarized by the algorithm 1 in the page 90 of the paper [**?**]. Then in the source code we find the function to extract the LSBP(Local SVD Binary Pattern) using the function SVD (Singular value decomposition) and the conditions to compute the background. Figure IV shows the relationship between the algorithm of the paper with the parts of the source code.

Algorithm 1 Background Subtraction for FG/BG segmentation using LSBP feature.

Initialization:

for $each\ pixel\ of\ the\ first\ N\ frames$ **do**
  Extract the LSBP descriptor for each pixels using Equation (12)
  Push color intensities into $BInt_{index}(x, y)$ and LSBP features into $BLSBP_{index}(x, y)$ as the background model
  Compute $\overline{d}_{min}(x, y)$ for each pixel.
end for

Mainloop:

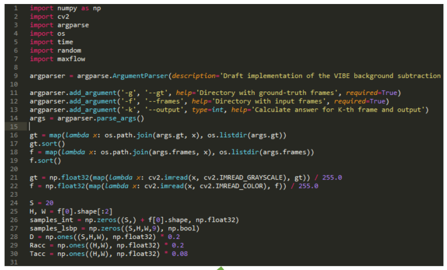
for $each\ pixel\ of\ newly\ appearing\ frame$ **do**
  Extract $Int(x, y)$ and $LSBP(x, y)$
end for
$matches \leftarrow 0$
$index \leftarrow 0$
for each pixel in current frame **do**
  while $((index \leq N)\ \&\&\ (matches < \sharp min))$ **do**
    computer $L1dist(Int(x, y), BInt_{index}(x, y))$ and $H(LSBP(x, y), BLSBP_{index}(x, y))$
    if $((L1dist(x, y) < R(x, y))\&\&(H(x, y) \leq H_{LSBP}))$ **then**
      $matches += matches$
    end if
    $index += index$
  end while
  if $(matches < \sharp min)$ **then**
    $Foreground$
  else
    $Background$
  end if
end for

V. Results

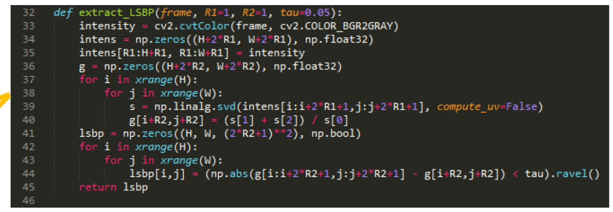
We use the CDnet 2012 dataset[1] to test the source code. From them we take the five datasets that are showed on the paper (*copyMachine*, *highway*, *overpass*, *PETS2006* and *sofa*). Then the results of our experiments is showed on Figure 3.

We can not compute the results of the *copyMachine* and *PETS2006* datasets because our RAM was insufficient (16 Gb). Additionally the resolution of each frame from the different datasets is different; *copyMachine* and *PETS2006* are of $720x480$ and the others are of $320x240$, therefore the compute task in the two firsts are more heavy because compute the *LSBP* is for each pixel.

[1] http://jacarini.dinf.usherbrooke.ca/dataset2012/



```python
import numpy as np
import cv2
import argparse
import os
import time
import random
import maxflow

argparser = argparse.ArgumentParser(description='Draft implementation of the VIBE background subtraction

argparser.add_argument('-g', '--gt', help='Directory with ground-truth frames', required=True)
argparser.add_argument('-f', '--frames', help='Directory with input frames', required=True)
argparser.add_argument('-k', '--output', type=int, help='Calculate answer for K-th frame and output')
args = argparser.parse_args()

gt = map(lambda x: os.path.join(args.gt, x), os.listdir(args.gt))
gt.sort()
f = map(lambda x: os.path.join(args.frames, x), os.listdir(args.frames))
f.sort()

gt = np.float32(map(lambda x: cv2.imread(x, cv2.IMREAD_GRAYSCALE), gt)) / 255.0
f = np.float32(map(lambda x: cv2.imread(x, cv2.IMREAD_COLOR), f)) / 255.0

S = 20
H, W = f[0].shape[:2]
samples_int = np.zeros((S,) + f[0].shape, np.float32)
samples_lsbp = np.zeros((S,H,W,9), np.bool)
D = np.ones((S,H,W), np.float32) * 0.2
Racc = np.ones((H,W), np.float32) * 0.2
Tacc = np.ones((H,W), np.float32) * 0.08
```



```python
def extract_LSBP(frame, R1=1, R2=1, tau=0.05):
    intensity = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    intens = np.zeros((H+2*R1, W+2*R1), np.float32)
    intens[R1:H+R1, R1:W+R1] = intensity
    g = np.zeros((H+2*R2, W+2*R2), np.float32)
    for i in xrange(H):
        for j in xrange(W):
            s = np.linalg.svd(intens[i:i+2*R1+1,j:j+2*R1+1], compute_uv=False)
            g[i+R2,j+R2] = (s[1] + s[2]) / s[0]
    lsbp = np.zeros((H, W, (2*R2+1)**2), np.bool)
    for i in xrange(H):
        for j in xrange(W):
            lsbp[i,j] = (np.abs(g[i:i+2*R2+1,j:j+2*R2+1] - g[i+R2,j+R2]) < tau).ravel()
    return lsbp
```

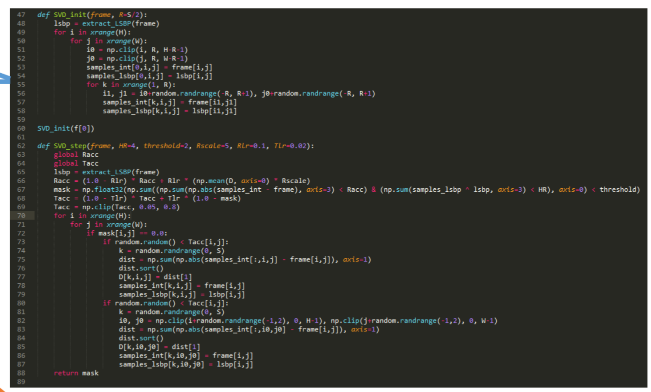
Algorithm 1 Background Subtraction for FG/BG segmentation using LSBP feature.

Initialization:

for *each pixel of the first N frames* **do**
  Extract the LSBP descriptor for each pixels using Equation (12)
  Push color intensities into $BInt_{index}(x,y)$ and LSBP features into $BLSBP_{index}(x,y)$ as the background model
  Compute $\vec{d}_{min}(x,y)$ for each pixel.
end for

Mainloop:

for *each pixel of newly appearing frame* **do**
  Extract $Int(x,y)$ and $LSBP(x,y)$
end for
$matches \leftarrow 0$
$index \leftarrow 0$
for each pixel in current frame **do**
  while $((index \leq N)$ && $(matches < \sharp min))$ **do**
    computer $L1dist(Int(x,y), BInt_{index}(x,y))$ and $H(LSBP(x,y), BLSBP_{index}(x,y))$
    if $((L1dist(x,y) < R(x,y))$&&$(H(x,y) \leq H_{LSBP}))$ **then**
      $matches += matches$
    end if
    $index += index$
  end while
  if $(matches < \sharp min)$ **then**
    $Foreground$
  else
    $Background$
  end if
end for



```python
def SVD_init(frame, R=S/2):
    lsbp = extract_LSBP(frame)
    for i in xrange(H):
        for j in xrange(W):
            i0 = np.clip(i, R, H-R-1)
            j0 = np.clip(j, R, W-R-1)
            samples_int[0,i,j] = frame[i,j]
            samples_lsbp[0,i,j] = lsbp[i,j]
            for k in xrange(1, R):
                i1, j1 = i0+random.randrange(-R, R+1), j0+random.randrange(-R, R+1)
                samples_int[k,i,j] = frame[i1,j1]
                samples_lsbp[k,i,j] = lsbp[i1,j1]

SVD_init(f[0])

def SVD_step(frame, HR=4, threshold=2, Rscale=5, Rlr=0.1, Tlr=0.02):
    global Racc
    global Tacc
    lsbp = extract_LSBP(frame)
    Racc = (1.0 - Rlr) * Racc + Rlr * (np.mean(D, axis=0) * Rscale)
    mask = np.float32(np.sum((np.sum(np.abs(samples_int - frame), axis=3) < Racc) & (np.sum(samples_lsbp ^ lsbp, axis=3) < HR), axis=0) < threshold)
    Tacc = (1.0 - Tlr) * Tacc + Tlr * (1.0 - mask)
    Tacc = np.clip(Tacc, 0.05, 0.8)
    for i in xrange(H):
        for j in xrange(W):
            if mask[i,j] == 0.0:
                if random.random() < Tacc[i,j]:
                    k = random.randrange(0, S)
                    dist = np.sum(np.abs(samples_int[:,i,j] - frame[i,j]), axis=1)
                    dist.sort()
                    D[k,i,j] = dist[1]
                    samples_int[k,i,j] = frame[i,j]
                    samples_lsbp[k,i,j] = lsbp[i,j]
                if random.random() < Tacc[i,j]:
                    k = random.randrange(0, S)
                    i0, j0 = np.clip(i+random.randrange(-1,2), 0, H-1), np.clip(j+random.randrange(-1,2), 0, W-1)
                    dist = np.sum(np.abs(samples_int[:,i0,j0] - frame[i,j]), axis=1)
                    dist.sort()
                    D[k,i0,j0] = dist[1]
                    samples_int[k,i0,j0] = frame[i,j]
                    samples_lsbp[k,i0,j0] = lsbp[i,j]
    return mask
```

```python
if args.output is not None:
    for i in xrange(f.shape[0]):
        sec = time.time()
        out = SVD_step(f[i])
        mrf = postprocessing(f[i], out)
        print('Frame %d, %.3f sec.' % (i, time.time() - sec))
        if i >= args.output:
            cv2.imwrite('svd-frame.png', f[i] * 255)
            cv2.imwrite('svd-mask.png', out * 255)
            cv2.imwrite('svd-gt.png', gt[i] * 255)
            cv2.imwrite('svd-mrf.png', mrf * 255)
            break
else:
    for i in xrange(f.shape[0]):
        cv2.imshow('Frame', f[i])
        cv2.imshow('Ground-truth', gt[i])
        out = SVD_step(f[i])
        cv2.imshow('Output', out)
        cv2.imshow('Output + MRF', postprocessing(f[i], out))
        k = cv2.waitKey(0)
        if k == 27:
            break
```

```python
def postprocessing(im, unary):
    unary = np.float32(unary)
    unary = cv2.GaussianBlur(unary, (9, 9), 0)

    g = maxflow.Graph[float]()
    nodes = g.add_grid_nodes(unary.shape)

    for i in xrange(im.shape[0]):
        for j in xrange(im.shape[1]):
            v = nodes[i,j]
            g.add_tedge(v, -unary[i,j], -1.0+unary[i,j])

    def potts_add_edge(i0, j0, i1, j1):
        v0, v1 = nodes[i0,j0], nodes[i1,j1]
        w = 0.1 * np.exp(-((im[i0,j0] - im[i1,j1])**2).sum() / 0.1)
        g.add_edge(v0, v1, w, w)

    for i in xrange(1,im.shape[0]-1):
        for j in xrange(1,im.shape[1]-1):
            potts_add_edge(i, j, i, j-1)
            potts_add_edge(i, j, i, j+1)
            potts_add_edge(i, j, i-1, j)
            potts_add_edge(i, j, i+1, j)

    g.maxflow()
    seg = np.float32(g.get_grid_segments(nodes))
    return seg
```

Fig. 1. Match between source code and paper algorithm.

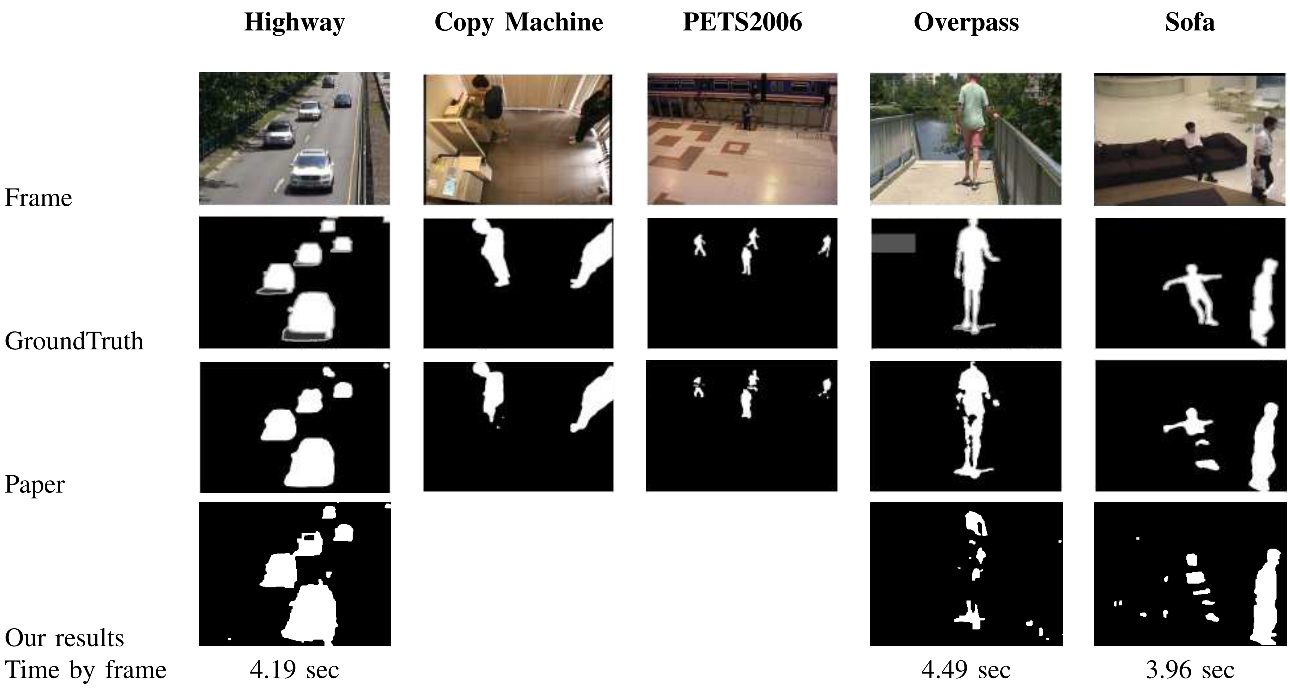


Fig. 2. Comparison between original frames, GroundTruth, results of paper and our results.