Background Subtraction using Local SVD Binary Pattern

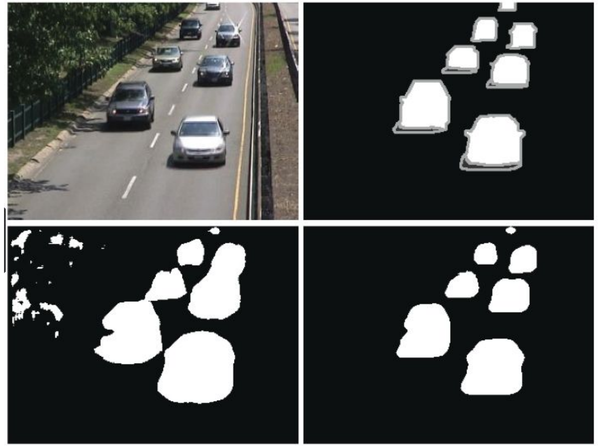
- Diego Javier Quispe
- David Choqueluque Roman

Uses

- Effective method for extracting the foreground (detect changes).

Problem:

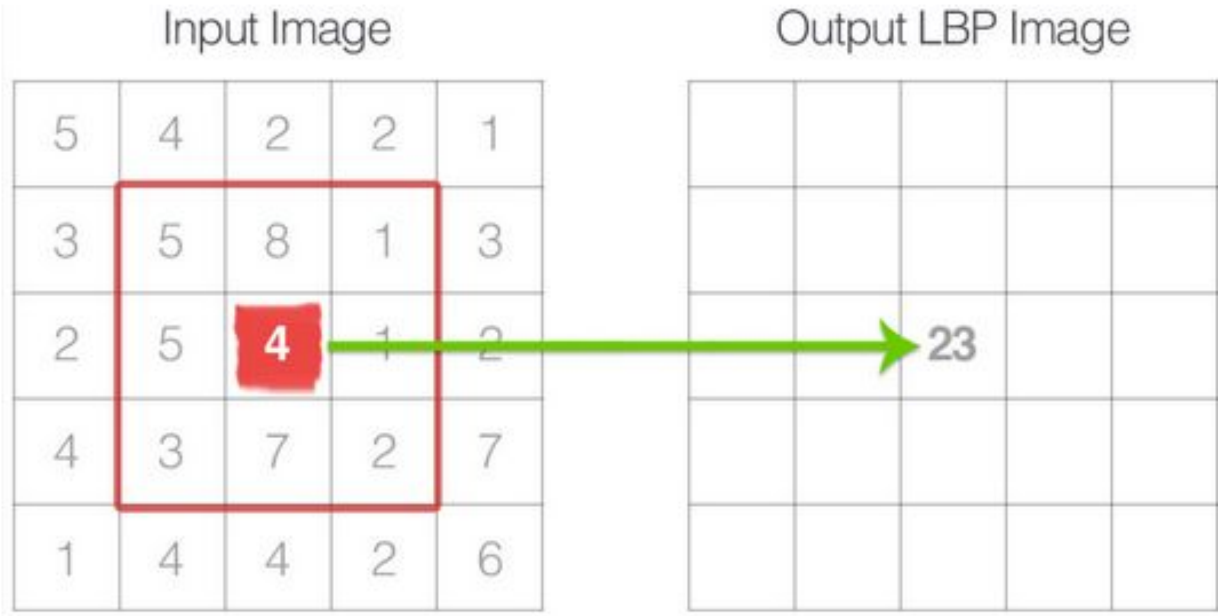
- Illumination variation is one of the major challenge in background subtraction.



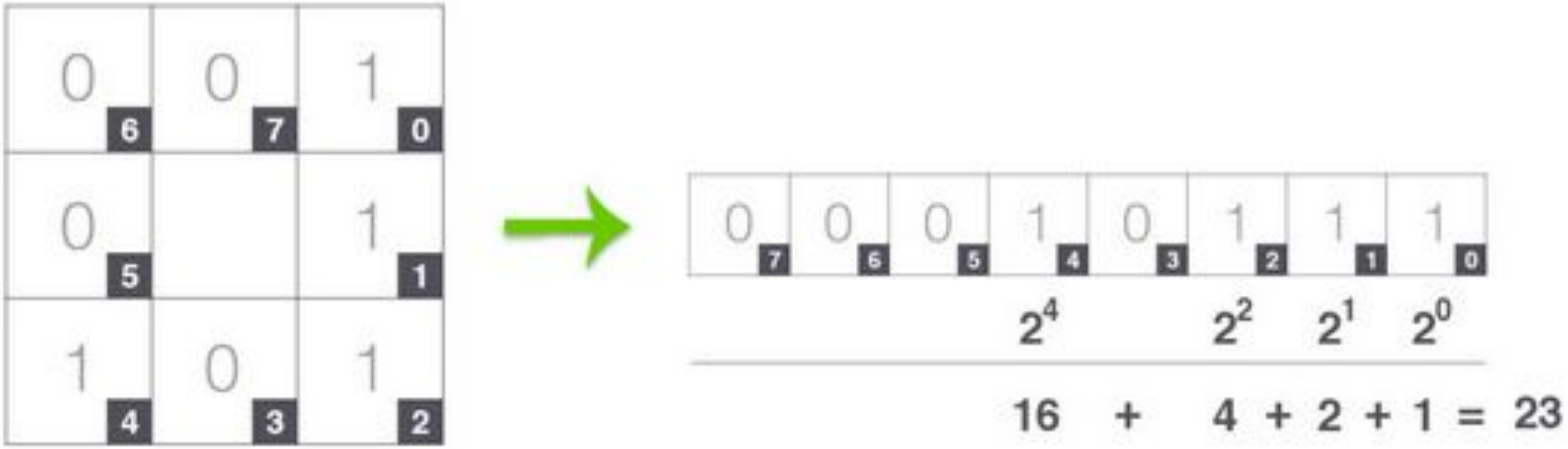
Contributions

- Present an efficient background subtraction model using novel Local SVD Binary Pattern feature (named LSBP).
- Extend LBP with Local singular value decomposition (SVD) operator.

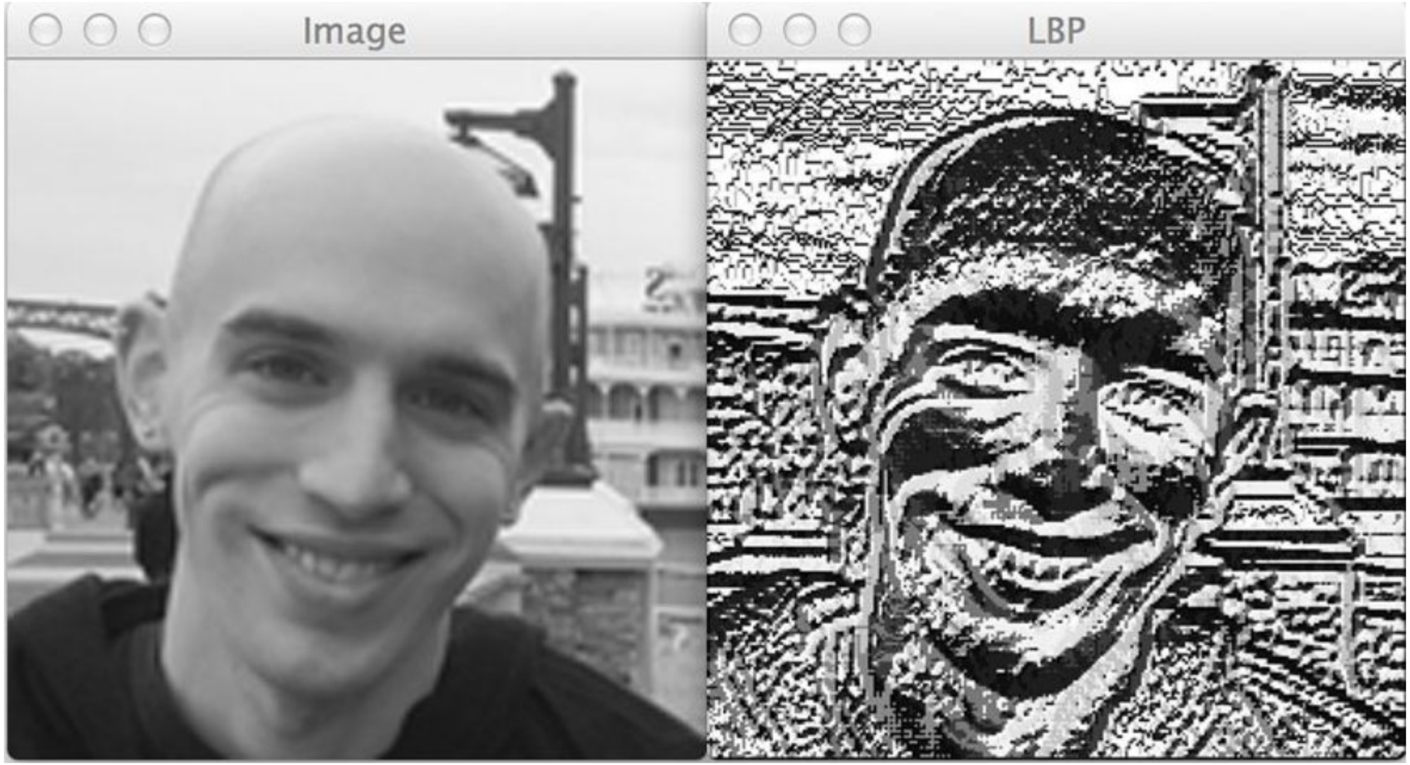
Local Binary Pattern



Local Binary Pattern



Local Binary Pattern

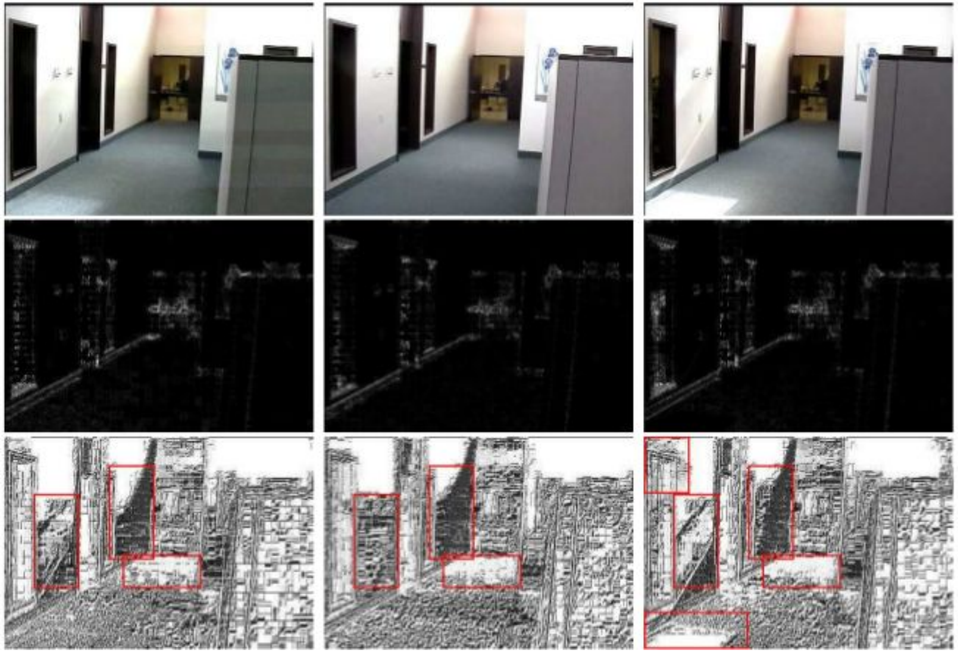


Normalized Coefficients of SVD

The singular values are likely to reveal the illumination invariant characteristics.

$$g(x,y) = \sum_{j=2}^{M} \tilde{\lambda}_j, \quad and \quad \tilde{\lambda}_j = \lambda_j / \lambda_1$$

Comparing LBP vs SVD

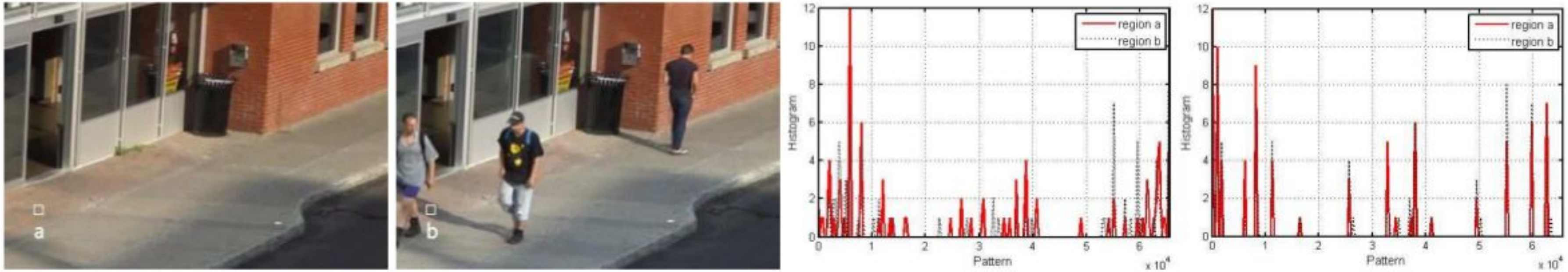


Extend LBP to LSBP

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

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

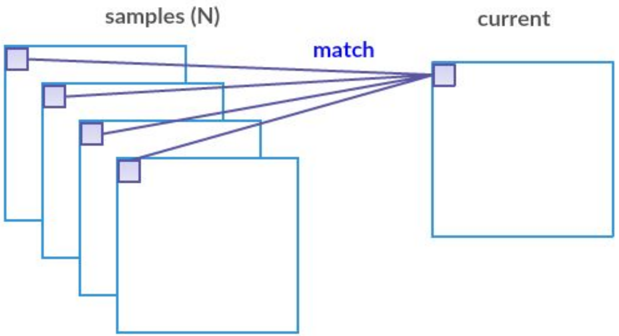
Comparing LBP vs LSBP



a b c d

Figure 3: Comparison of LBP and LSBP features with shadow. (a) and (b) are two frames from the "busStation" video, with two 10×10 regions drawn. Regions contain the same background with and without shadows. (c) LBP histogram of two regions. (d) LSBP histograms of two regions.

Method: Classification



Match:

- Hamming Distance: LSBP
- L1 Distance: Color

Background model

$$B(x,y) = \{B_1(x,y), ..., B_{index}(x,y), ..., B_N(x,y)\}$$

$LSBP(x,y)$ $Int(x,y),$

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

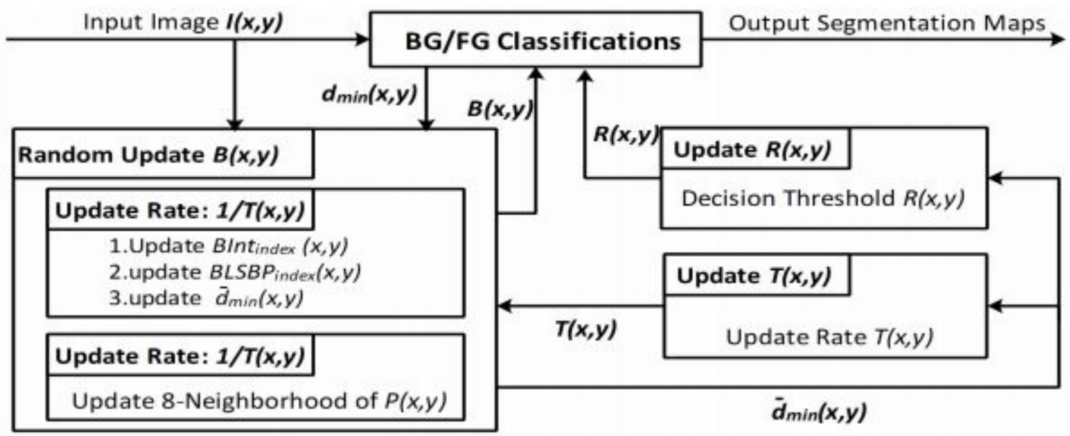
Initialization:

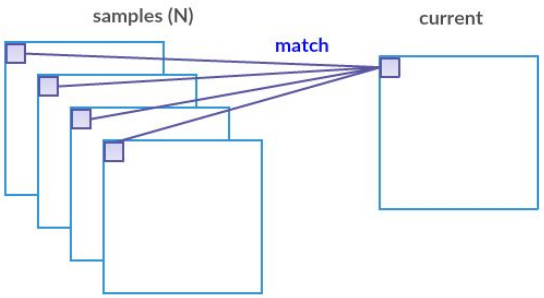
for *each pixel of the first N frames* **do**
  Extract the LSBP descriptor for each pixels using Equation (12)
  Push color intensities into $BInt_{index}(x,y)$ and LSBP features into $BLSBP_{index}(x,y)$ as the background model
  Compute $\overline{d}_{min}(x,y)$ for each pixel.
end for

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

Method: Update





Match:
- Hamming Distance: LSBP
- L1 Distance: Color

Background model

$$B(x,y) = \{B_1(x,y), ..., B_{index}(x,y), ..., B_N(x,y)\}$$

$$LSBP(x,y) \quad Int(x,y),$$

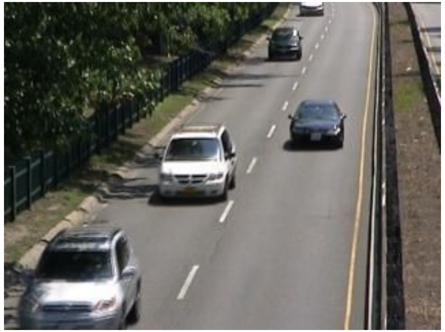
- R(x,y): per-pixel color intensity
- d^min(x,y): average dmin
- dmin: min color distance(L1)(matching)

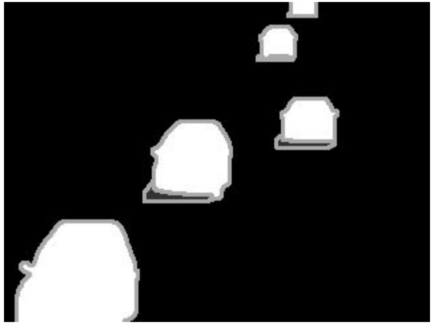
Results highway python

- First 100 elementos of the dataset Highway
- Time to frame: 4.84

Results highway python

- Results of the 1700 elementos
- Time: 2 hours with 52 minutes
- Time to frame: 4.82





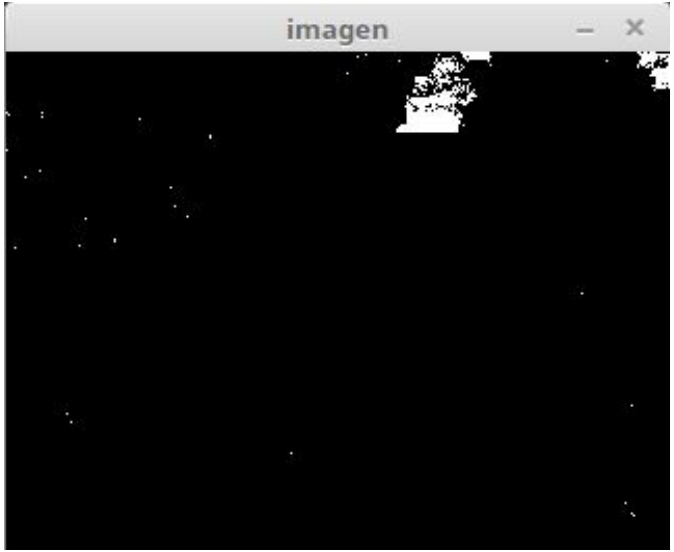
Results office python

- Results of the 100 elementos
- Time to frame: 7.23

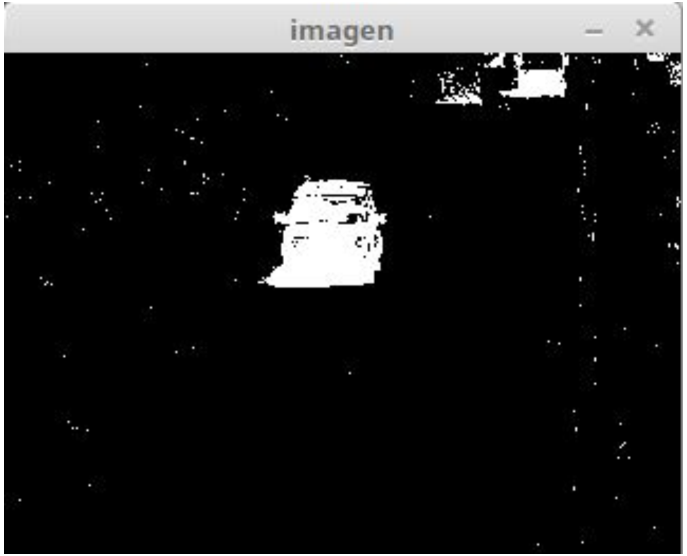
Processing time

Stage	Time per frame(in c++)	Time per frame(in python)
highway	0.864s	5.0141
peopleInShape	1.096s	5.8016

Results highway c++

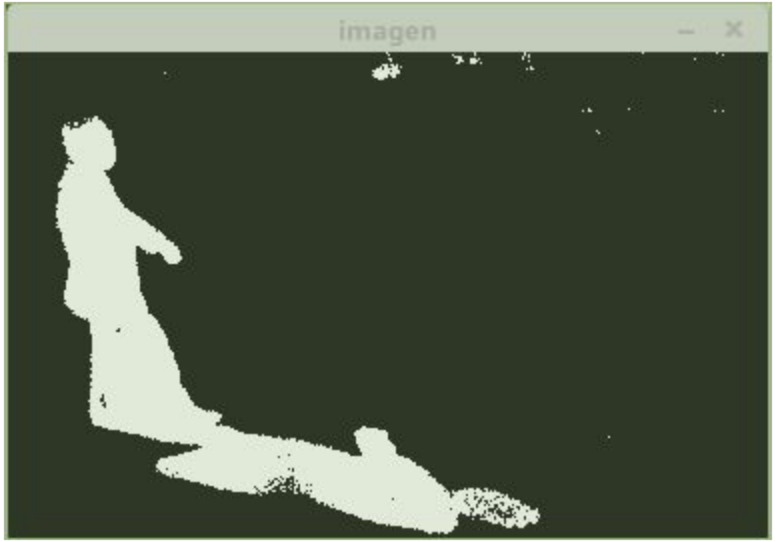


Results highway c++



Results highway c++

Results peopleInShape c++



Results peopleInShape c++

Results peopleInShape c++

Conclusion

- Compare with the implementation in python, our implementation in c++ reduce the time per frame in a proportion of 5:1.
- We reduce noise by identifying and correcting errors in the code.