Tampering Detection in Low-Power Smart Cameras

POLITECNICO DI MILANO

Adriano Gaibotti, Claudio Marchisio,
Alexandro Sentinelli, Giacomo Boracchi

ST Microelectronics (AST), Agrate Brianza (MB), Italy

Politecnico di Milano (DEIB), Milano (MI), Italy

Motivation

Scenario: video monitoring systems operating outdoor and in harsh environments

Tampering Detection: automatic identification of events that could prevent the correct image acquisition

- *Camera sabotage*
- *Natural phenomena* (wind, rain drops, snow…)




Motivation

Scenario: video monitoring systems operating outdoor and in harsh environments

Tampering Detection: automatic identification of events that could prevent the correct image acquisition

- *Camera sabotage*
- *Natural phenomena* (wind, rain drops, snow…)

Degradations on images:

- Change of the camera view-point
- Blurring artefacts

which causes a substantial loss of information




Our Goal

Low-power smart cameras *should be* able to autonomously detect tampering events to activate suitable countermeasures

Idea: perform tampering detection by solely monitoring the video stream acquired form a low-powered device

Our Goal

Low-power smart cameras *should be* able to autonomously detect tampering events to activate suitable countermeasures

Idea: perform tampering detection by solely monitoring the video stream acquired form a low-powered device

Target Device: *SecSoc* (Security System on Chip)

- Two AA batteries
- Simple Imaging analytics tasks
- In case of event detection
  - Send “event detected” message
  - Transfer compressed images sequence (mjpeg)

life.augmented

Challenges

In a low-power smart camera, tampering detection is more challenging because:

- Computational constraints
- Very low frame rates (e.g, < 1 frame per minute)
- Even in normal conditions, the scene content might change much in between two frames

PROBLEM FORMULATION

Problem Formulation

Frames acquired at time t:

$$z_t(x) = D_t[y_t](x)$$

- $x \in \mathcal{X}$: pixel's coordinates
- $z_t(x)$: *intensity* value of x-th pixel in frame at time t
- $D_t[\cdot]$: degradation operator

Problem Formulation

Frames acquired at time t:

$$z_t(x) = D_t[y_t](x)$$

- **No tampering:**

$$D_t[y_t](x) = y_t(x) + \eta_t(x)$$

Problem Formulation

Frames acquired at time t:

$$z_t(x) = D_t[y_t](x)$$

- **No tampering:**

$$D_t[y_t](x) = y_t(x) + \eta_t(x)$$

- **Blur:**

$$D_t[y_t](x) = \int_X y_t(s) h_t(x,s) ds + \eta_t(x)$$

Problem Formulation

Frames acquired at time t:

$$z_t(x) = D_t[y_t](x)$$

- **No tampering:**

$$D_t[y_t](x) = y_t(x) + \eta_t(x)$$

- **Blur:**

$$D_t[y_t](x) = \int_X y_t(s) h_t(x,s) ds + \eta_t(x)$$

- **Displacement:**

$$z_t(x) = \begin{cases} y_t(x) + \eta_t(x), & t < T^* \\ w_t(x) + \eta_t(x), & t \geq T^* \end{cases}$$

Other degradations (e.g. sensor faults, noise increase) could be possibly considered

PROPOSED SOLUTION

Proposed Solution

Our solution three features:

Extract **scalar indicators** from each frame, that should:

- Have low computational cost
- Have low memory requirements

Detect outliers in the indicators to identify tampering events

Segment the scene and monitor each region

- Definition of *regions* in which indicators are *stationary*
- Analysis of the indicators *separately for each region*
- Combining together the obtained results

Extracted indicators

Average luma value

$$l(t) = \sum_{x \in X} z_t(x)$$

Displacements produce changes in $l(t)$

Frame difference

$$d(t) = \sum_{x \in X} \big(z_t(x) - z_{t-1}(x)\big)^2$$

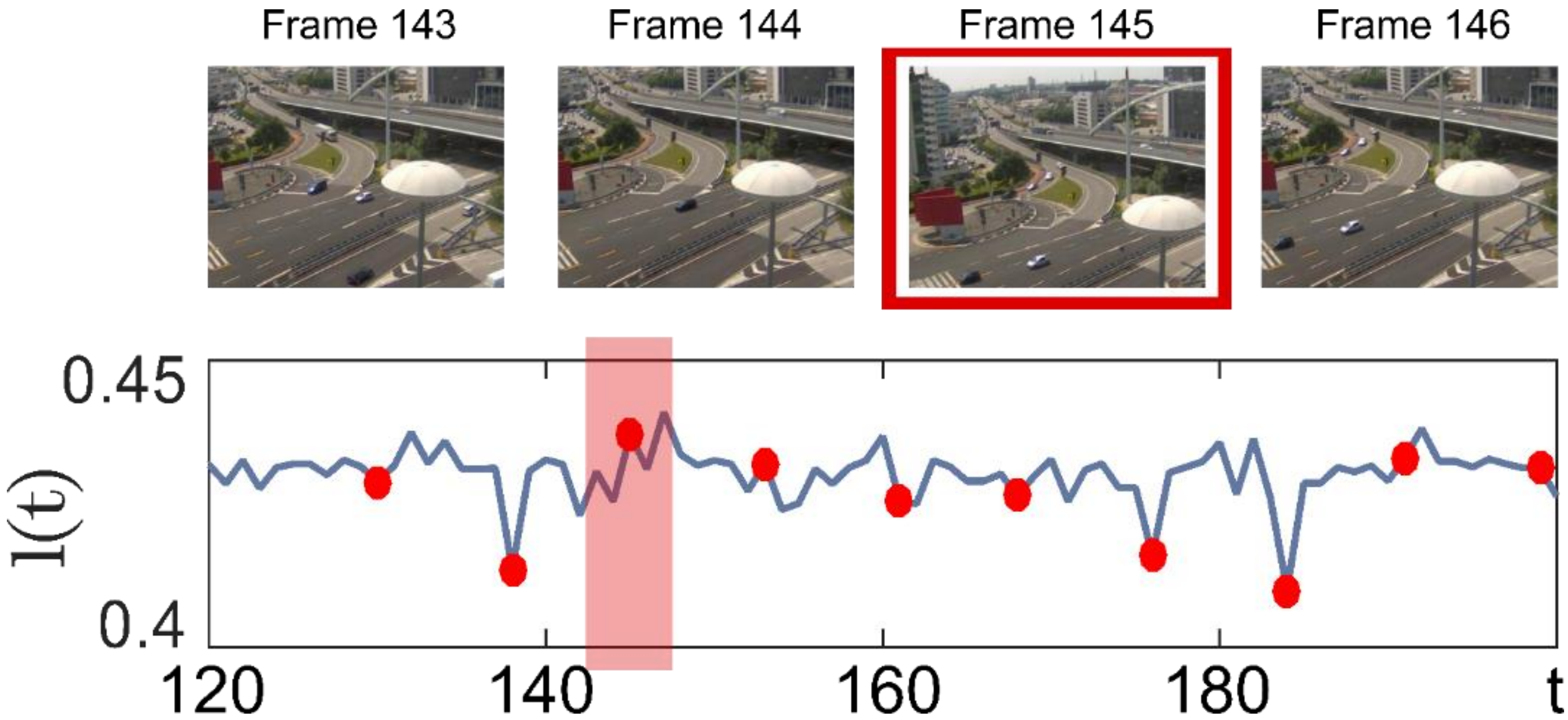
Displacements produce changes in $d(t)$

Average gradient norm

$$g(t) = \sum_{x \in X} \sqrt{(z_t \circledast f_h)^2 + (z_t \circledast f_v)^2}$$

Blur attenuates high frequency components of the image

Example: camera displacements

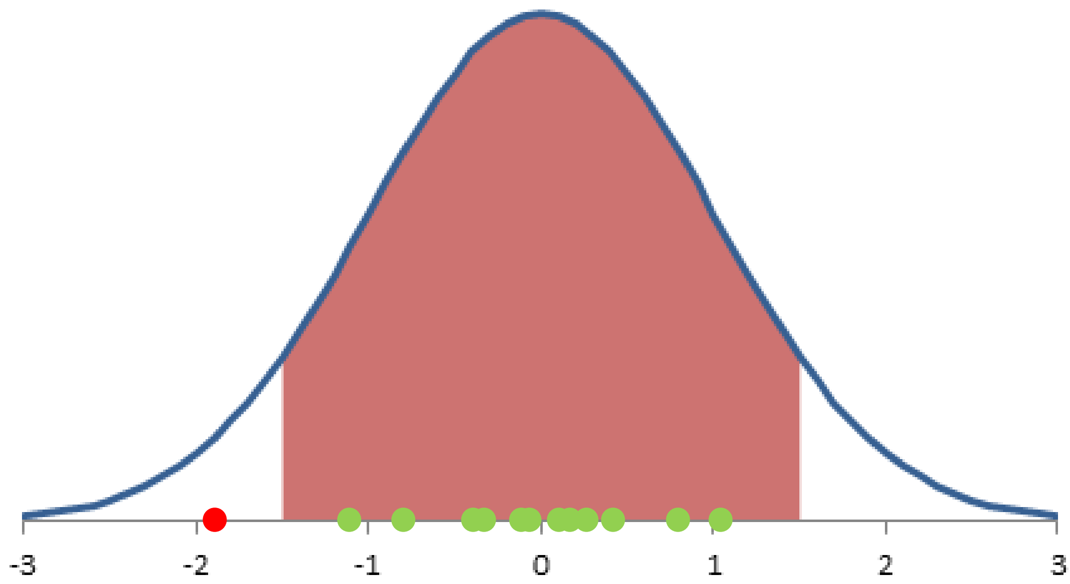


Outlier detection

Indicators are monitored by ***confidence interval***:

$$[\bar{l} - \gamma_l \sigma_l, \bar{l} + \gamma_l \sigma_l]$$

- $\bar{l}$: temporal mean of ∂l
- σ_l: temporal standard deviation of ∂l
- γ_l: tuning parameter



Outlier detection

Indicators are monitored by ***confidence interval***:

$$[\bar{l} - \gamma_l \sigma_l, \bar{l} + \gamma_l \sigma_l]$$

- $\bar{l}$: temporal mean of ∂l
- σ_l: temporal standard deviation of ∂l
- γ_l: tuning parameter

Very **efficient** to run, but these techniques **are meant for i.i.d. random variables**

Unfortunately, changes in the scene or in the illumination bring **unpredictable trends** inside our indicators

Detrending

Compute the temporal derivative of the indicators

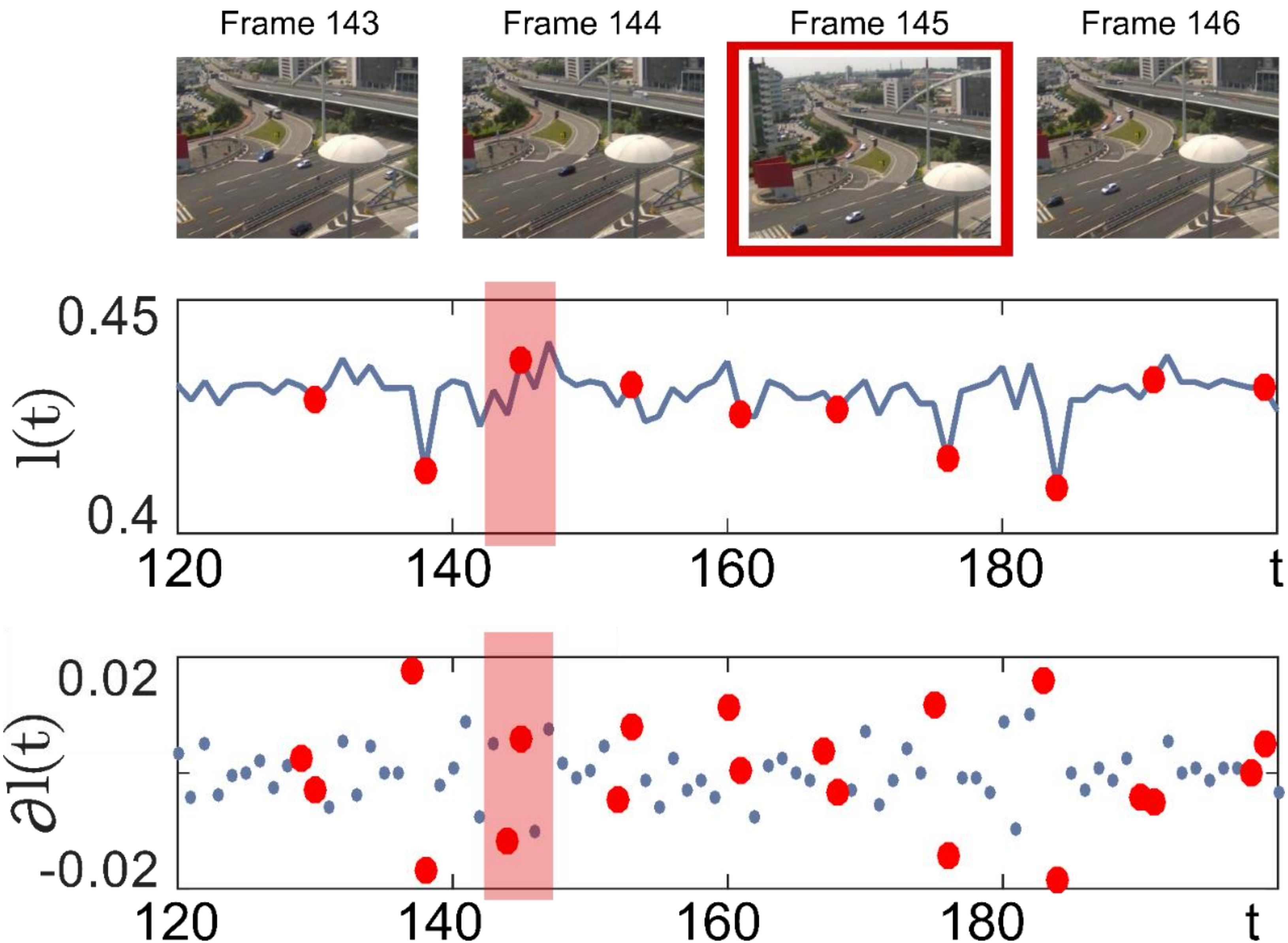
$$\partial l(t) = l(t) - l(t-1)$$

Outliers in the detrended indicators are found as values falling outside the confidence interval

$$\left[\overline{\partial l} - \gamma_l \sigma_l, \overline{\partial l} + \gamma_l \sigma_l\right]$$

- $\overline{\partial l}$: temporal mean of ∂l
- σ_l: temporal standard deviation of ∂l
- γ_l: tuning parameter

Example: camera displacements



Segmentation

- Video parts with different degrees of texture and dynamics
- Indicators applied in different areas of the image have different behaviours

Static and high textured

High dynamics

Static and low textured

Segmentation

- Video parts with different degrees of texture and dynamics
- Indicators applied in different areas of the image have different behaviours
- An *adaptive segmentation* of the image is able to make the tampering detection more robust.

Static and high textured

High dynamics

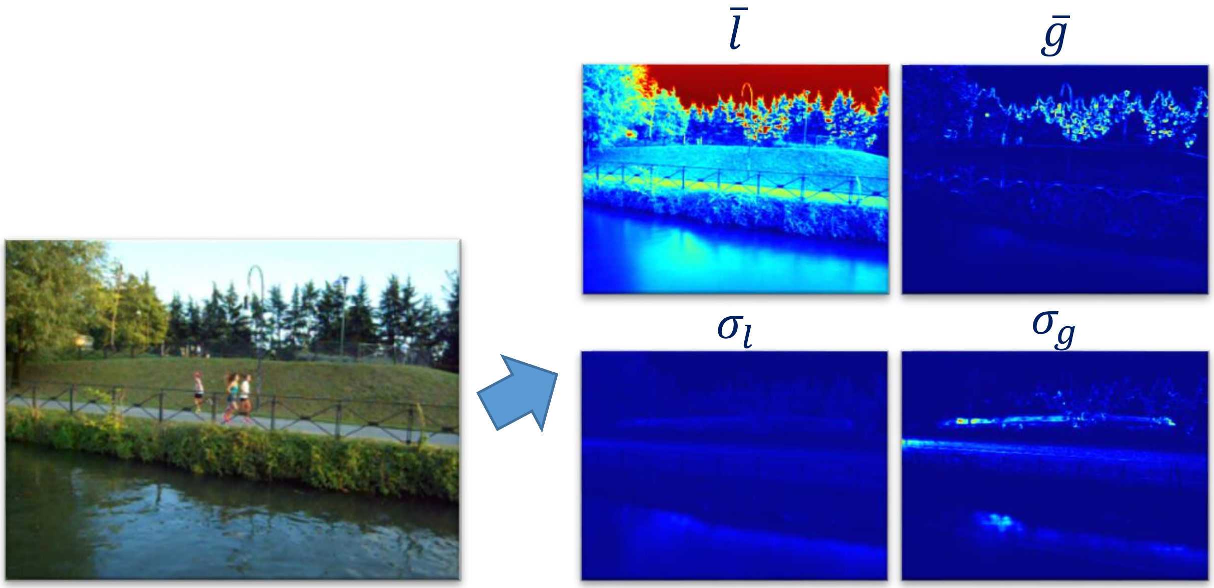
Static and low textured

Segmentation

1. **Feature vector $\boldsymbol{f}(x)$** from training set

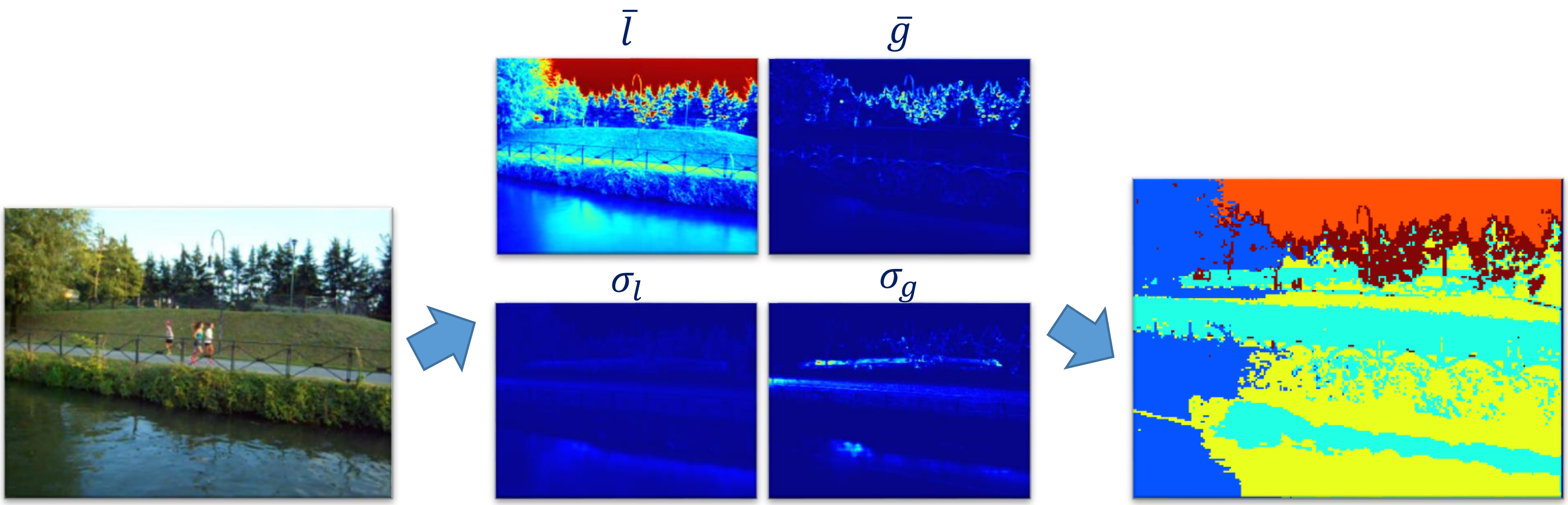
$$\boldsymbol{f}(x) = \left[r(x); c(x); \bar{l}(x); \sigma_l(x); \bar{g}(x); \sigma_g(x)\right], \forall x \in X$$

- $r(x), c(x)$: row and column number of pixel x
- $\bar{l}(x), \sigma_l(x)$: mean and standard deviation of the l at x during time
- $\bar{g}(x), \sigma_g(x)$: mean and standard deviation of the gradient norm at x during time



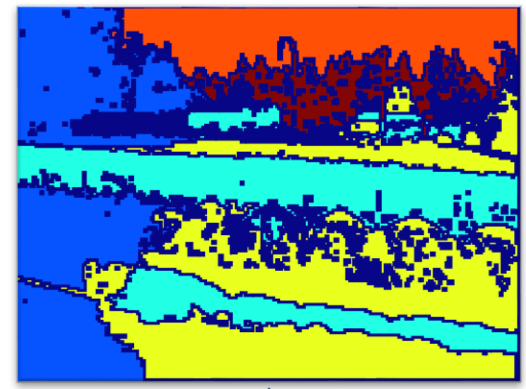
Segmentation

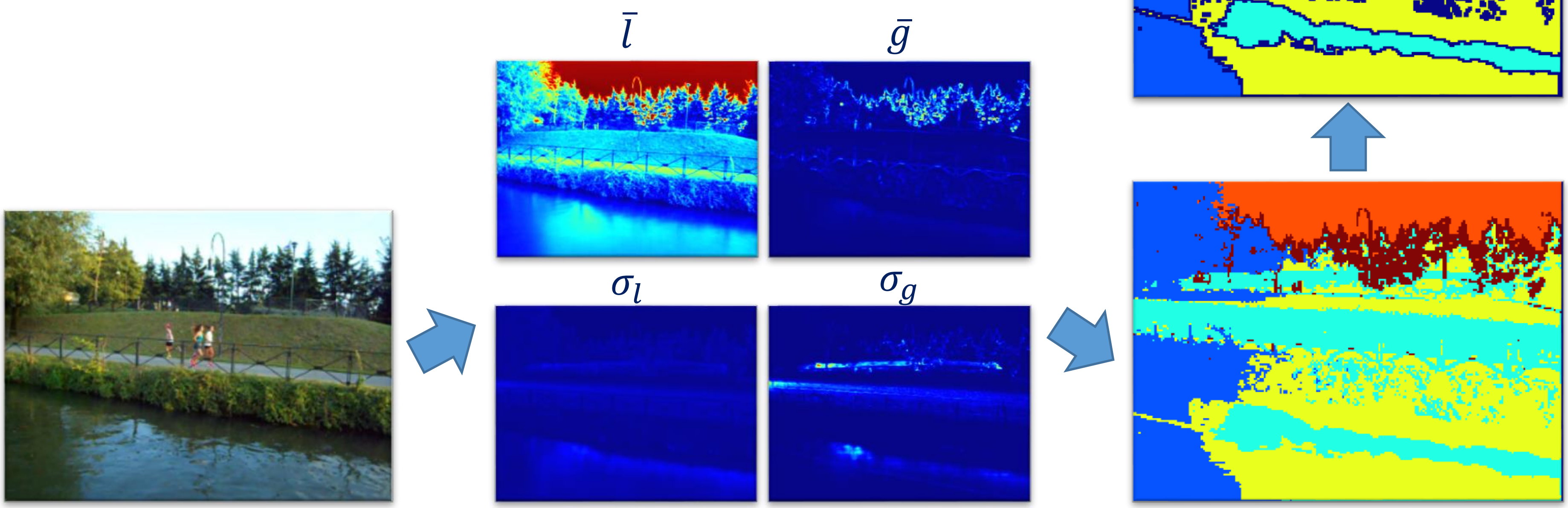
1. **Feature vector** $f(x)$ from training set
2. ***Weighted* k-means** clustering over feature vectors
   - Euclidean distances are scaled by a weight inversely proportional to the standard deviation over the cluster
   - **Calinski-Harabasz criterion** for number of cluster choice



Segmentation

1. **Feature vector** $f(x)$ from training set
2. ***Weighted* k-means** clustering over feature vectors
3. **Refinement** with ***morphological operators*** to remove boundaries and tiny regions





Segmentation

1. **Feature vector** $f(x)$ from training set
2. ***Weighted* k-means** clustering over feature vectors
3. **Refinement** with ***morphological operators*** to remove boundaries and tiny regions

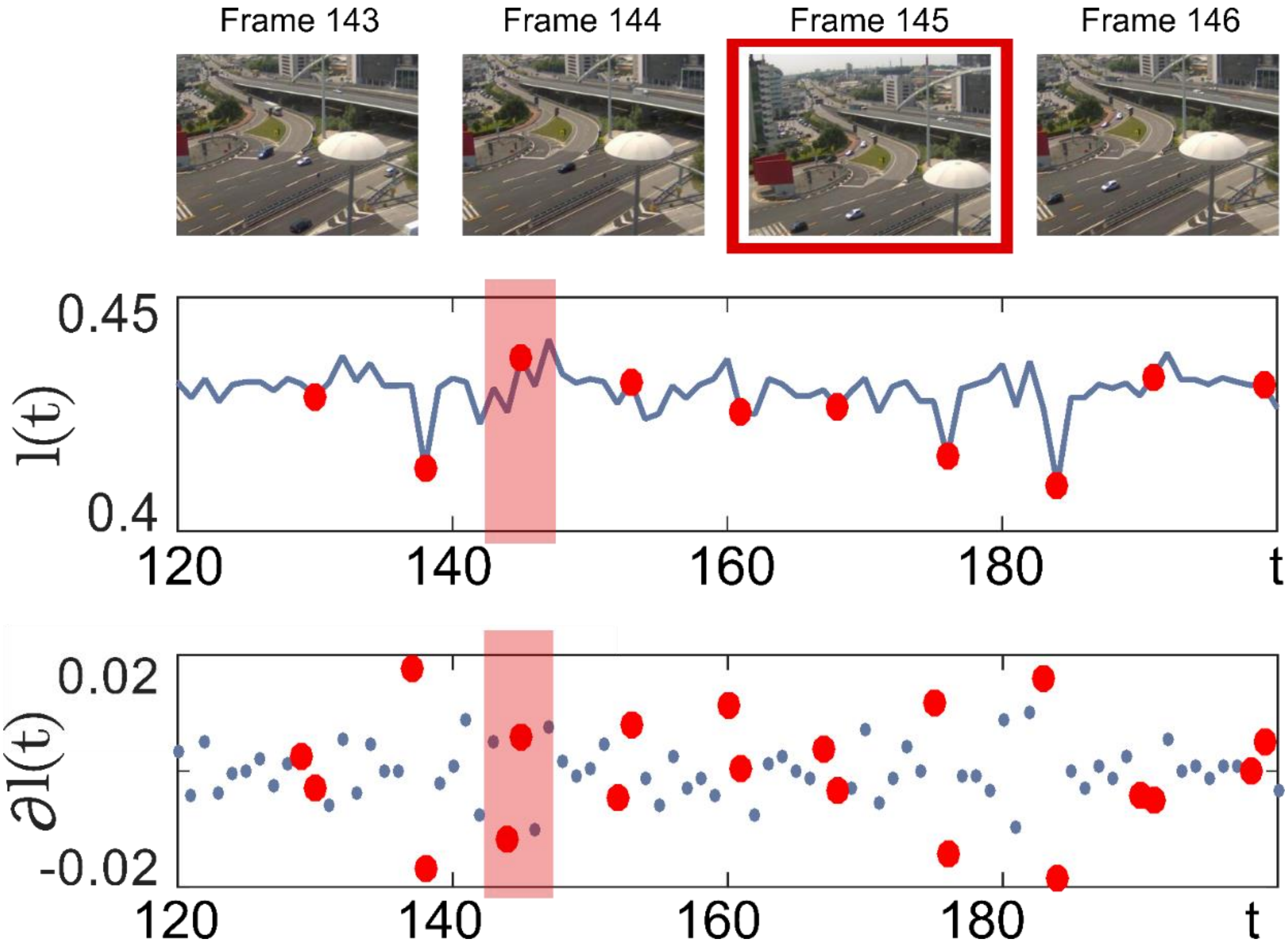
Indicators are then computed **for each region**:

$$l_k(t) = \frac{1}{\#R_k} \sum_{x \in R_k} z_t(x)$$

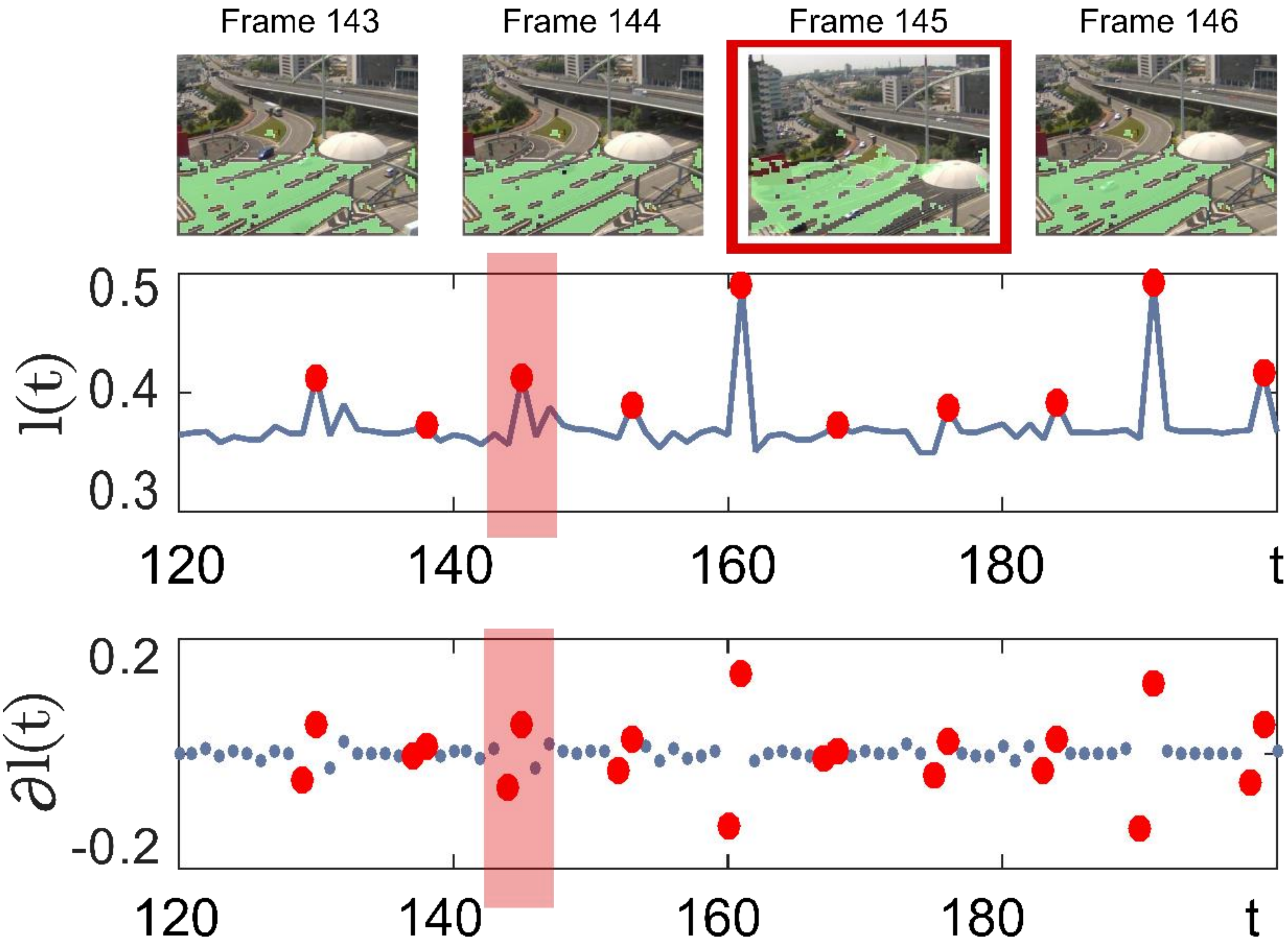
$$d_k(t) = \frac{1}{\#R_k} \sum_{x \in R_k} \left(z_t(x) - z_{t-1}(x)\right)^2$$

R_k: k-th region extracted from image, $k = 1, \dots, K$
$\#R_k$: number of pixels in the k-th region, $k = 1, \dots, K$

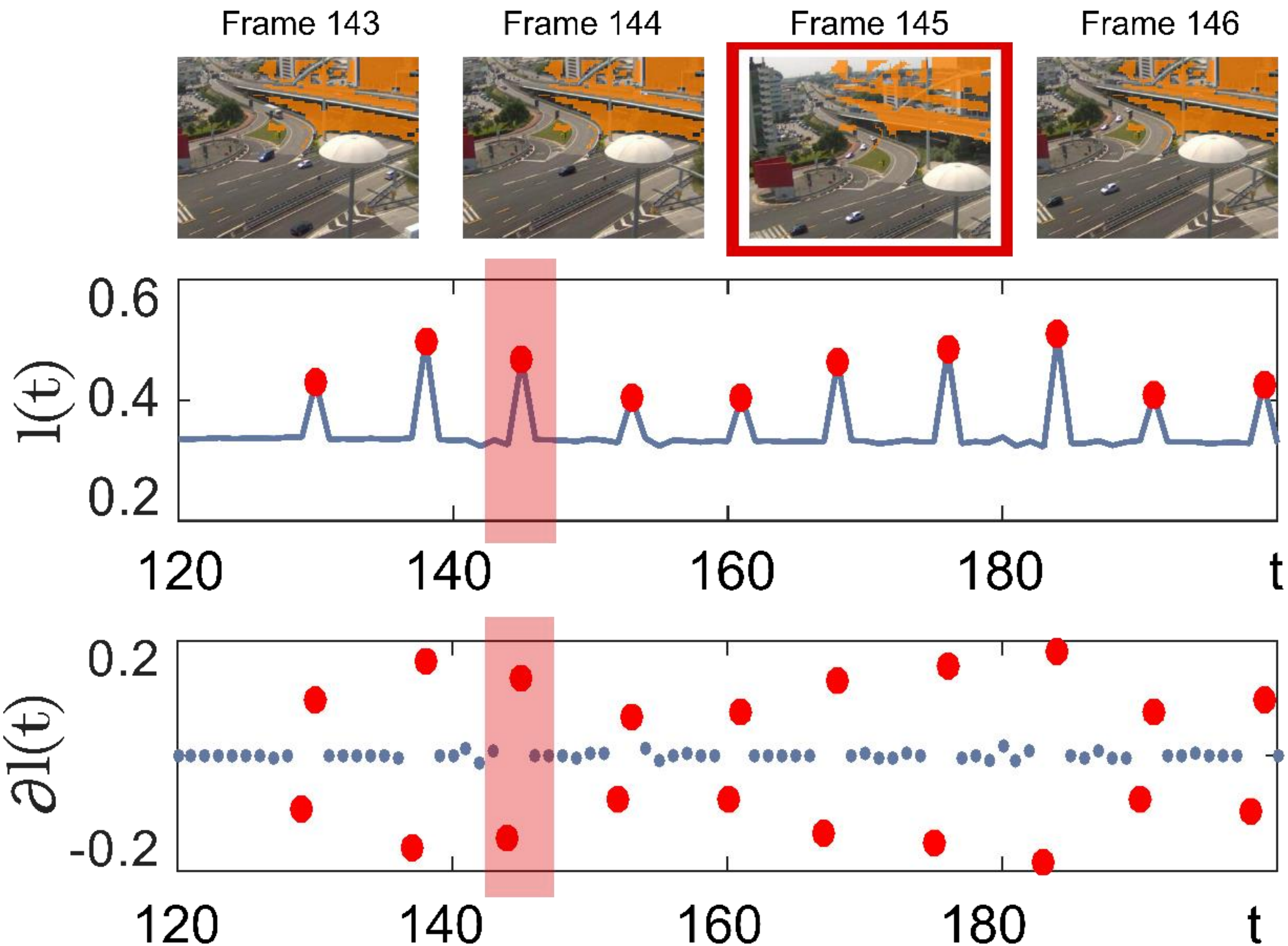
Example: camera displacements



Example: camera displacements



Example: camera displacements



Algorithm

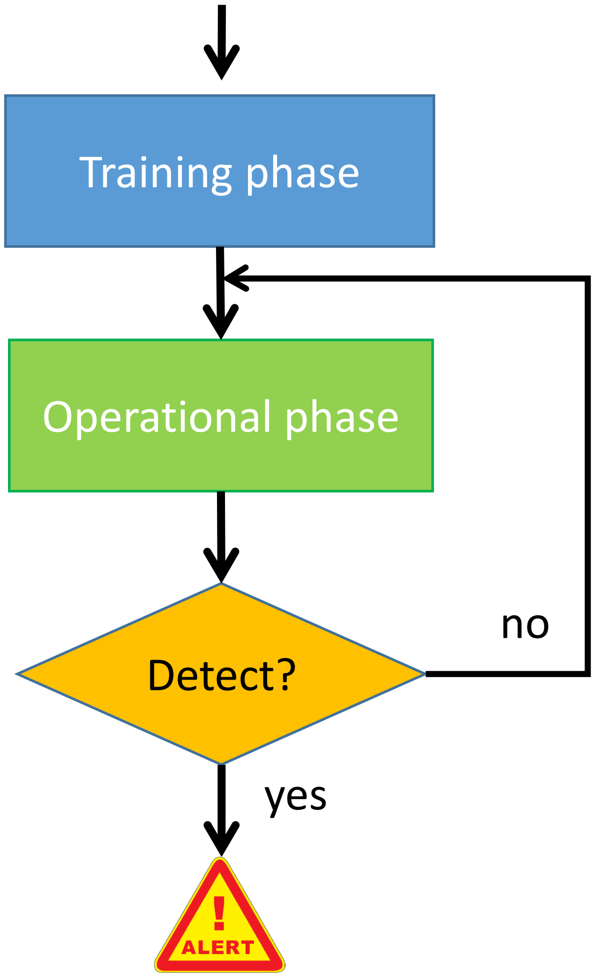
1. Training phase, compute

$$\partial l_k(t), \partial g(t), \qquad t = 1 \dots, T_o$$
$$\overline{\partial l}_k, \overline{\partial g}, \sigma_{l_k}, \sigma_g, \qquad k = 1, \dots, K$$

1. Get frame z_t
2. Compute $\{\partial l_k(t), k = 1, \dots, K\}, \partial g(t)$

3. *If* $\partial g(t) < -\gamma_g \sigma_g \vee \partial g(t) > \gamma_g \sigma_g$ *then*
   raise a blurring alert
4. *If* $\partial l_k(t) < -\gamma_{l_k} \sigma_{l_k} \vee \partial l_k(t) > \gamma_{l_k} \sigma_{l_k}$ *for at least* Γ_l regions *then*
   raise a camera displacement alert

INPUT:
Training frames,
$\gamma_l, \gamma_g, \Gamma_l, \{R_k, k = 1, \dots, K\}$



Algorithm

1. Training phase, compute

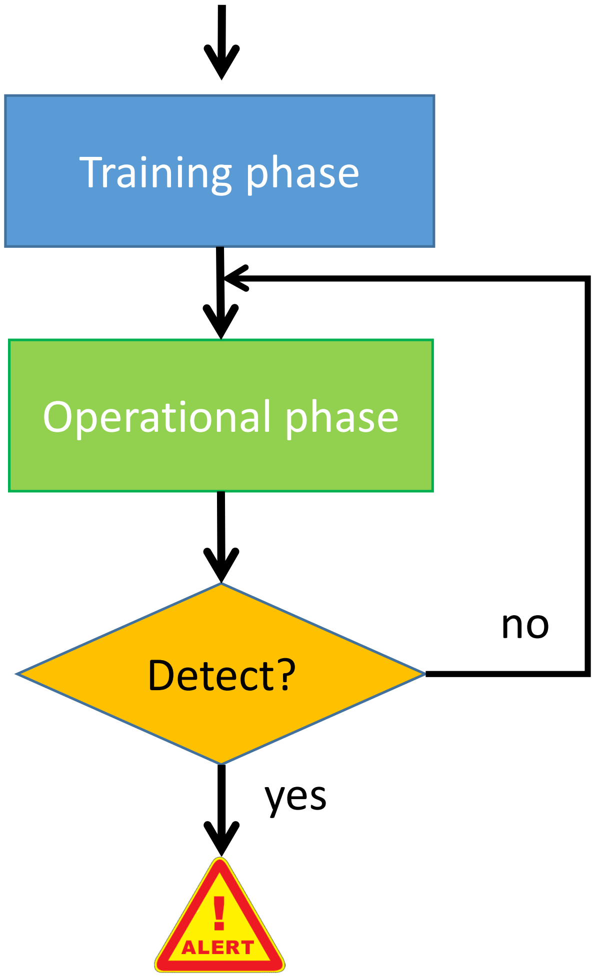
$$\partial l_k(t), \partial g(t), \qquad t = 1 \dots, T_o$$
$$\overline{\partial l_k}, \overline{\partial g}, \sigma_{l_k}, \sigma_g, \qquad k = 1, \dots, K$$

1. Get frame z_t
2. Compute $\{\partial l_k(t), k = 1, \dots, K\}, \partial g(t)$

3. *If* $\partial g(t) < -\boxed{\gamma_g}\sigma_g \vee \partial g(t) > \boxed{\gamma_g}\sigma_g$ *then*
   raise a blurring alert
4. *If* $\partial l_k(t) < -\boxed{\gamma_{l_k}}\sigma_{l_k} \vee \partial l_k(t) > \boxed{\gamma_{l_k}}\sigma_{l_k}$ *for at least* $\boxed{\Gamma_l}$ regions *then*
   raise a camera displacement alert

INPUT:

Training frames,
$\gamma_l, \gamma_g, \Gamma_l, \{R_k, k = 1, \dots, K\}$



The parameters

The parameters Γ determine the **minimum number of regions firing an outlier** before detecting a camera displacement

- $\Gamma = 1$ implies that the first region firing an outliers causes a detection
- $\Gamma = K$ not advisable, in some cases at least one region might not change (e.g. the sky region when moving the camera downward)
- $\Gamma = K - 1$ all but the sky (or the ground) regions have to fire an alarm

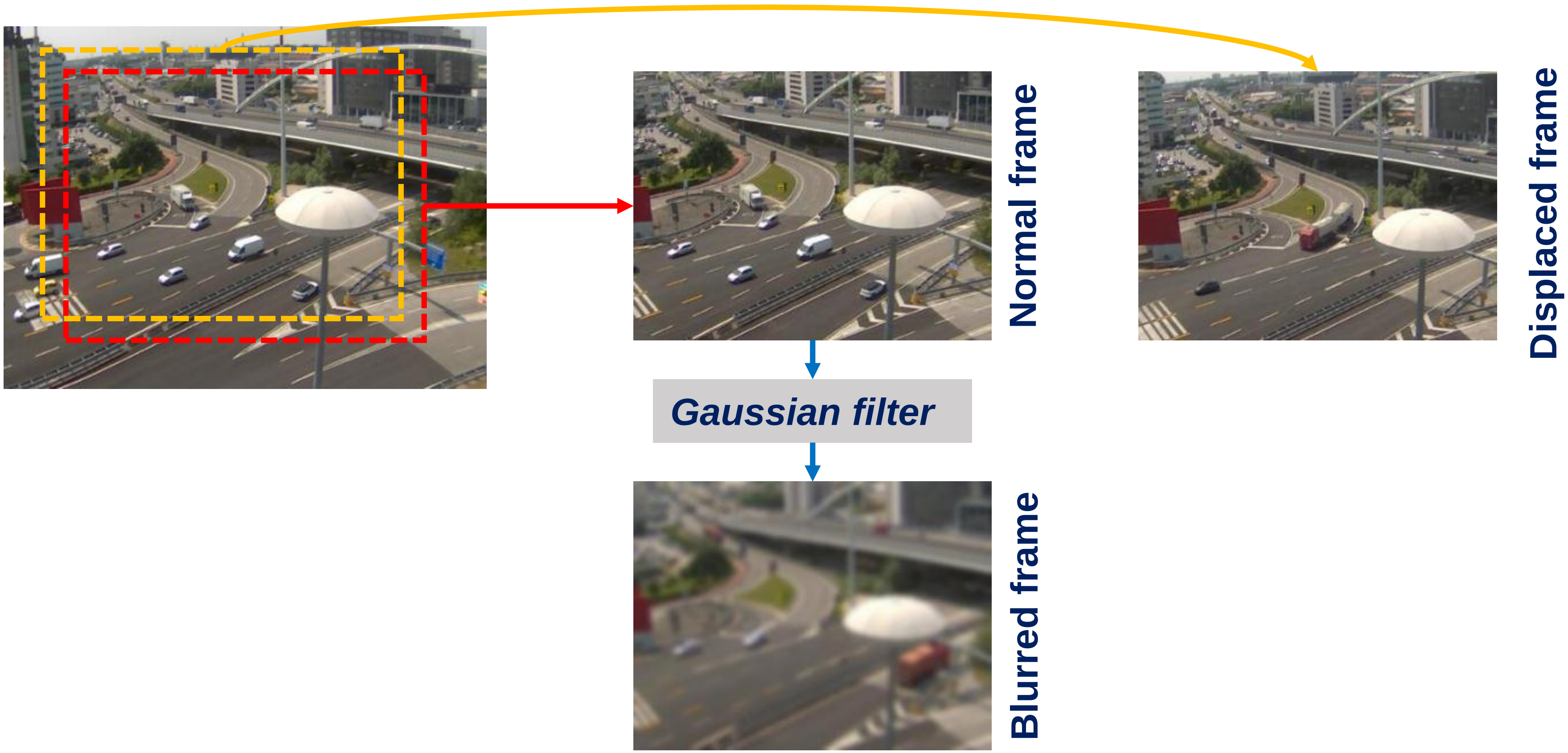
The parameters γ determine **how far an outlier should be** from the expected value of the indicator.

We investigate these parameters empirically.

EXPERIMENTS

Experiments

- 8 sequences taken from webcams monitoring different urban areas (more than 12200 frames overall)
- Tamper was introduced *synthetically* in 10% of frames



Experiments

- 8 sequences taken from webcams monitoring different urban areas (more than 12200 frames overall)
- Tamper was introduced *synthetically* in 10% of frames
- Different configurations have been compared:
  - Segmentation vs. whole image
  - Adaptive regions (Algorithm 1) vs. Voronoi regions
  - $\Gamma_x = 1$ vs. $\Gamma_x = K - 1$
- ROC curves have been computed by varying γ_l, γ_d and γ_g between 0.1 and 50

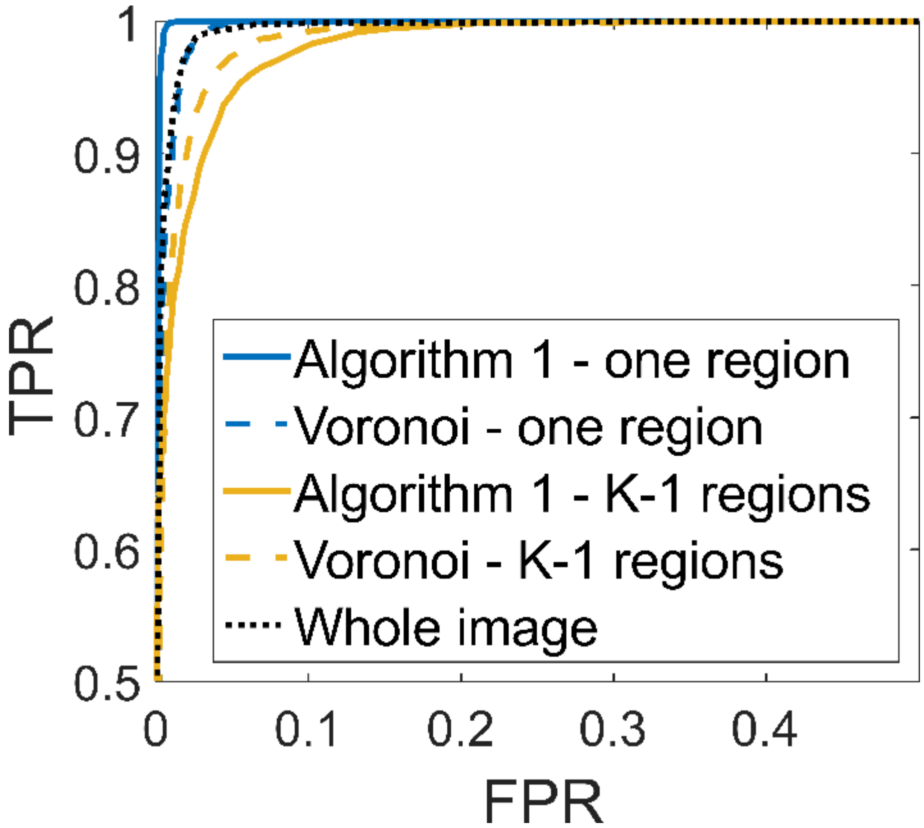
- Sequences can be provided upon request.

Results

Displacement detection - FD

AUCs:

- Algorithm 1 – one region: 99.89%
- Whole image: 99.65%

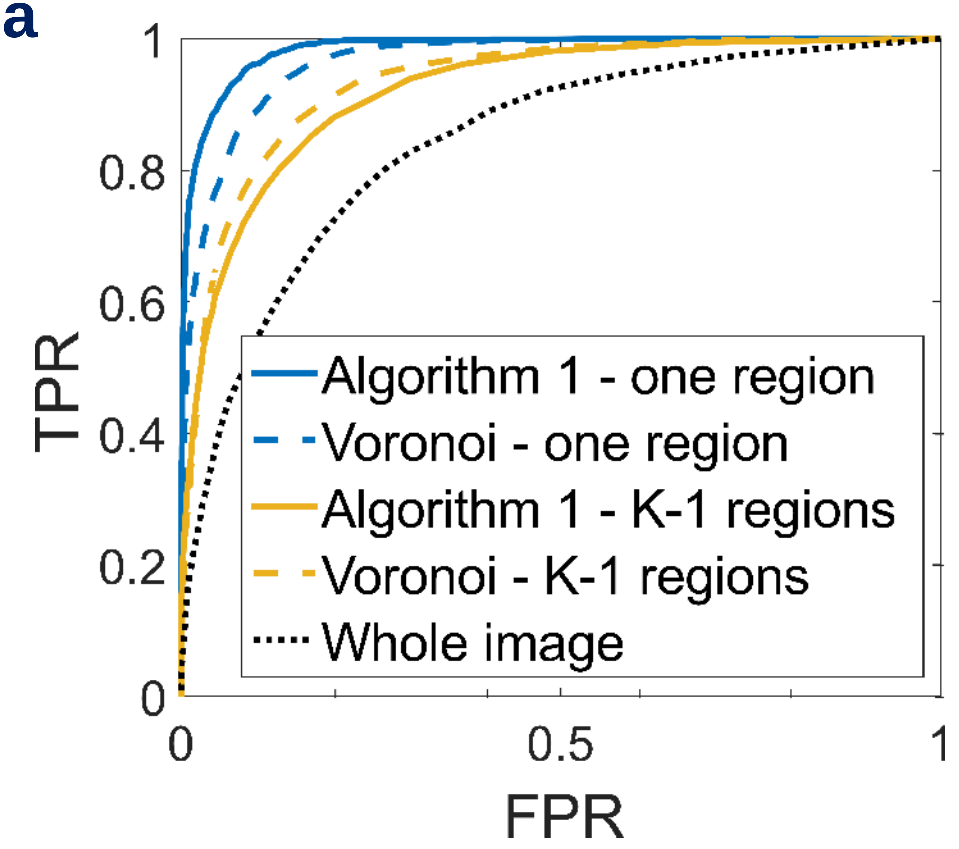


Limiting FPR to 1%

- Algorithm 1:
  - $\gamma_d = 15.6$
  - $TPR = 99.92\%$
- Whole image:
  - $\gamma_d = 6.5$
  - $TPR = 91.67\%$
- Voronoi:
  - $\gamma_d = 18.64$
  - $TPR = 89.05\%$

Displacement detection - Luma

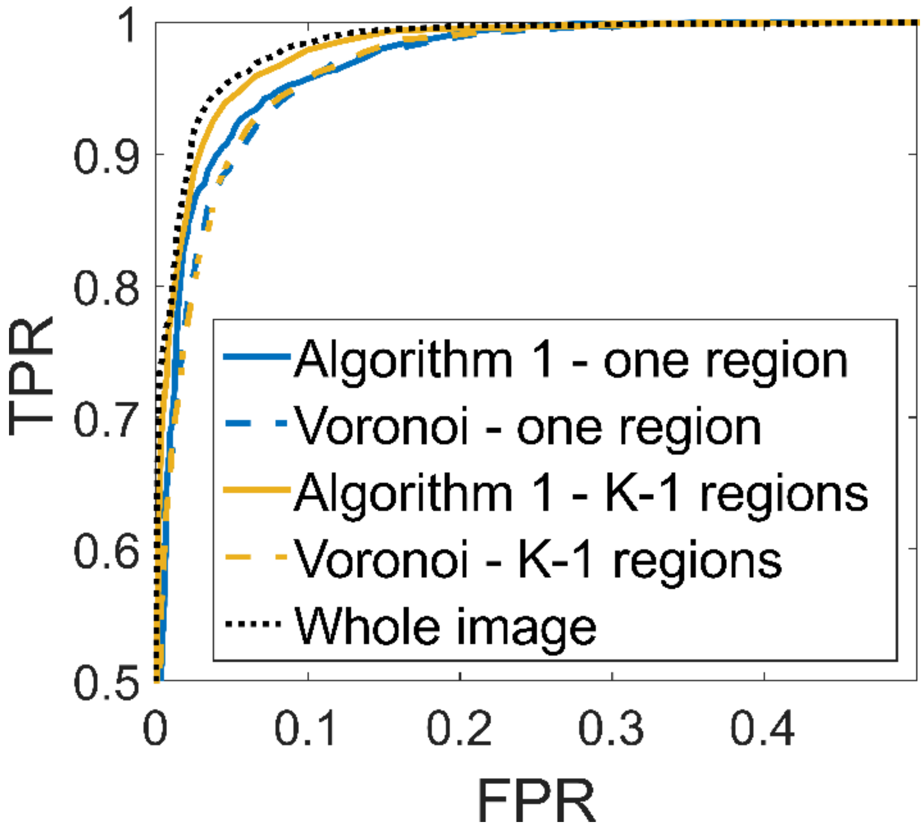
AUCs:
- Algorithm 1 – one region: 98.44%
- Whole image: 84.07%



Limiting FPR to 1%
- Algorithm 1:
  - $\gamma_l = 5.8$
  - $TPR = 73.98\%$
- Whole image:
  - $\gamma_l = 3.7$
  - $TPR = 17.85\%$
- Voronoi:
  - $\gamma_l = 6$
  - $TPR = 53.02\%$

Results

Blurring detection - Gradient



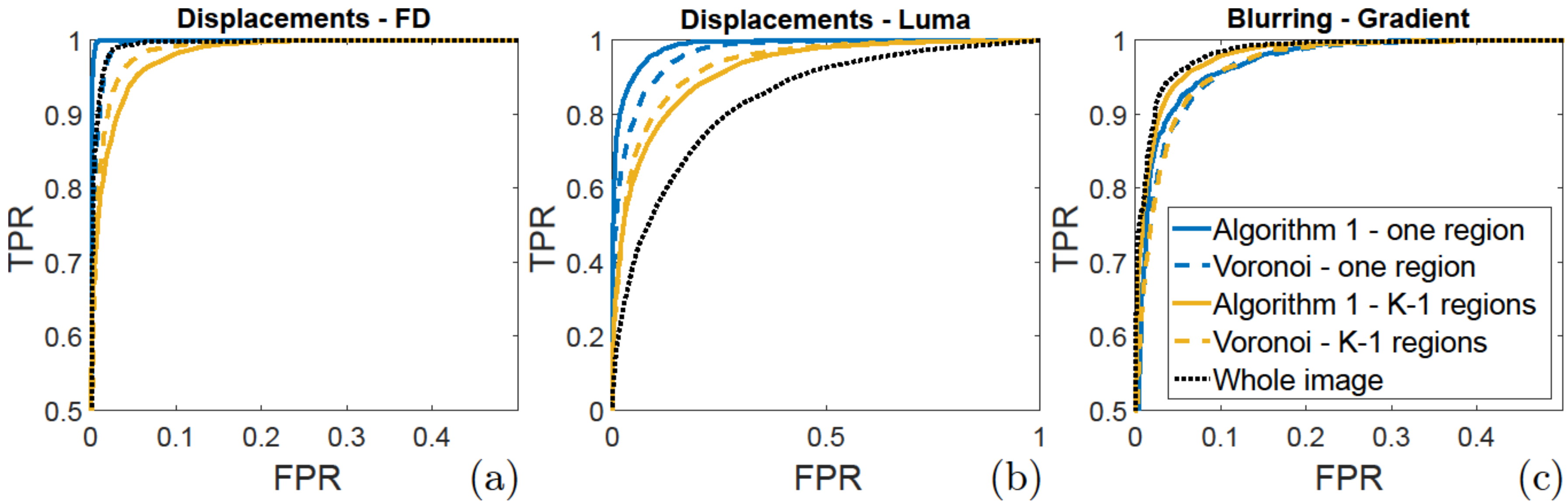
AUCs:

- Algorithm 1 – one region: 98.4%
- Whole image: 99.13%

Limiting FPR to 3%

- Algorithm 1:
  - $\gamma_g = 6{,}2$
  - $TPR = 90{,}77\%$
- Whole image:
  - $\gamma_g = 3{,}4$
  - $TPR = 92{,}85\%$
- Voronoi:
  - $\gamma_g = 8{,}1$
  - $TPR = 83{,}21\%$

Results



- Camera displacements can be better detected by monitoring FD than luma
  - But luma requires less computational and memory resources
- Blurring is more effectively detected by monitoring the whole image at once
- In low-power scenario, it is important to operate at low FPR
  - Prevent useless data transmission (reduce battery lifetime)

CONCLUSIONS

Conclusions

- **Tampering detection** for *embedded smart cameras*
  - Blur and displacements detection
  - Operating on image regions improves the detection of camera displacements
  - Low-computational/memory requirements
- **Ongoing works:**
  - Approacing other types of tampering
    - e.g. Degradation of imaging sensor
  - Integration of ***sequential monitoring schemes***
    - to detect subtle tampering persisting over time
  - Investigate ***superpixels*** methods to segment the image
    - exploiting the temporal information in the training sequence

THANKS FOR YOUR ATTENTION!
ANY QUESTIONS?