Real-time covariance tracking algorithm for embedded systems

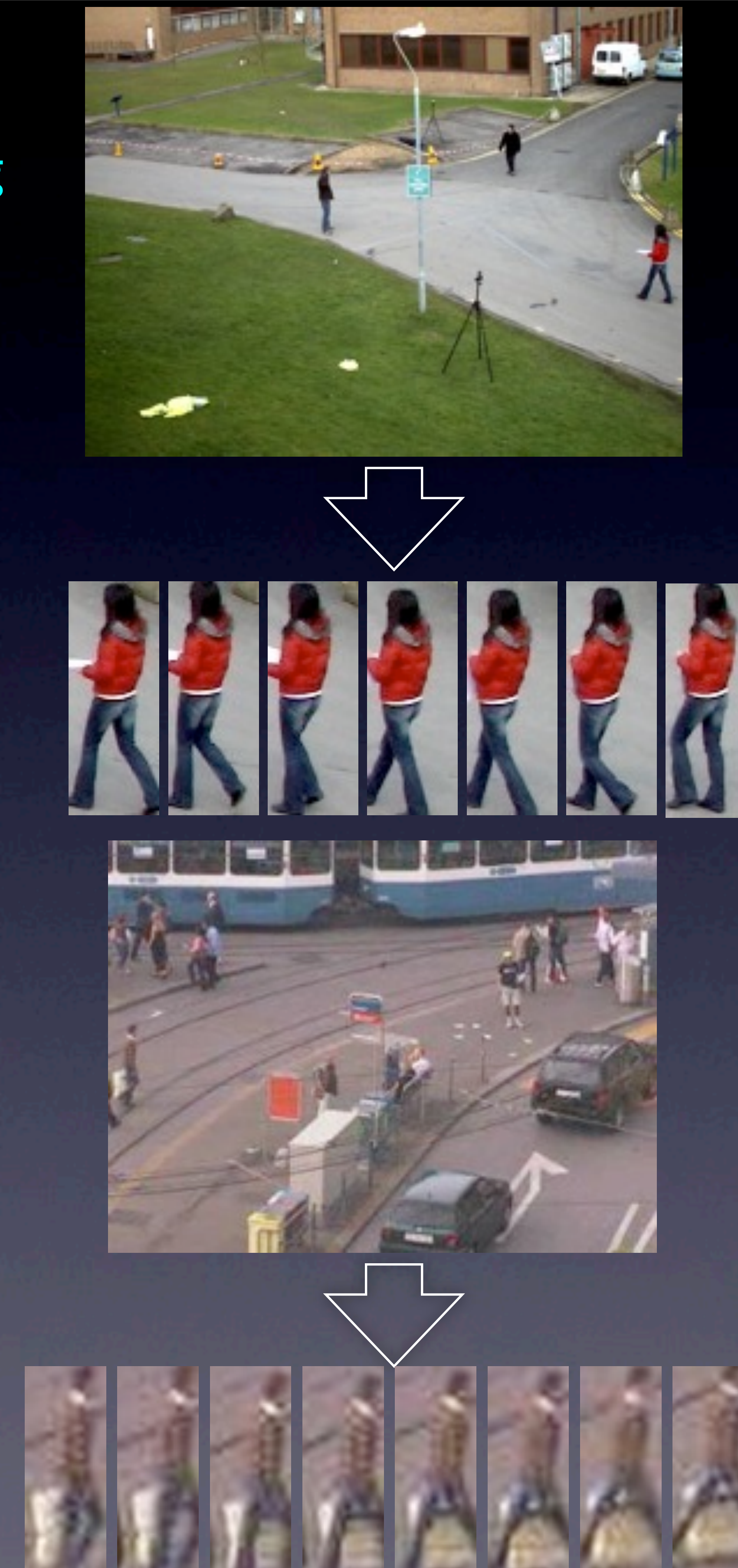
A. Roméro, L. Lacassagne, A. Zahraee, M. Gouiffès

www.lri.fr/~lacas

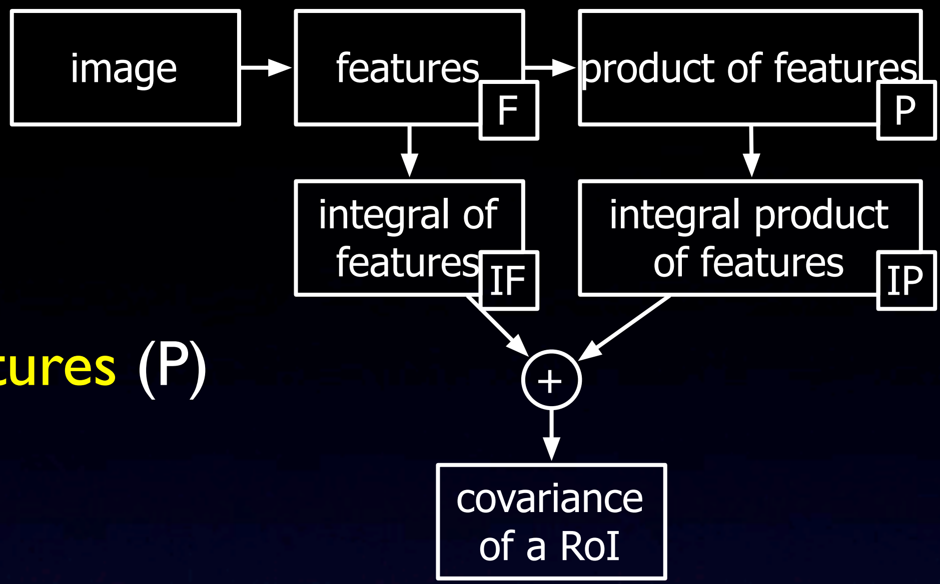
LRI , LIMSI & IEF - University Paris-Sud

Context & goal

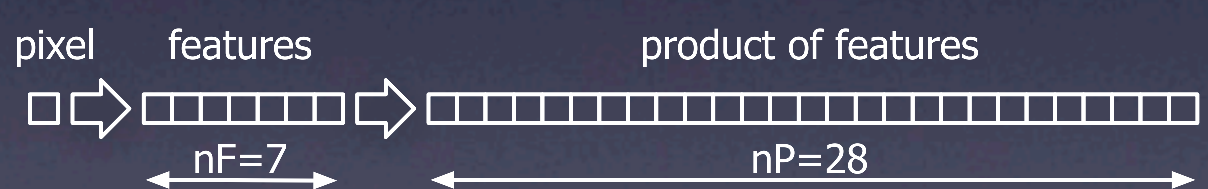
- Covariance matching techniques are interesting :
  - good performance for object retrieval, detection and tracking
  - mixing color and texture information into compact representation
- But ...
  - heavy computations even for State-of-the-Art processors
- So:
  - optimizations are mandatory for embedded systems (Intel mobile proc, ARM Cortex A9)
- Presentation in 4 points
  - algorithm presentation
  - algorithm optimization
  - benchmarks
  - video examples



Covariance algorithm part #1



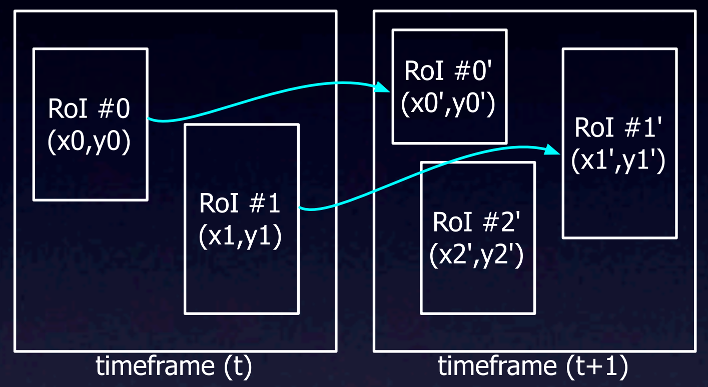
- From an image:
  - a set of features (F) is computed, and a set of product of features (P)
  - the integral images (IF) and (IP) are computed
  - Finally the covariance of a given RoI is easily computed thanks to integral image properties
- Features are tuned to the nature of the image
  - Face tracking & recognition: [x, y, Ix, Iy, Ixx, Iyy] (coordinates, first and second derivatives)
  - pedestrian tracking: [x, y, Intensity, sin(LBP), cos(LBP)] (coordinates, intensity, Local Binary Pattern manipulations)



- But: required a huge amount of memory
  - sizeof(IF) = sizeof(F) = nF x sizeof(float) x N^2 & sizeof(IP) = sizeof(P) = nP x sizeof(float) x N^2
  - with nF=7 and np=28 => 280 bytes per pixel for a 1024x1024 image : 280 MB !!!
  - ... thanks to product symetry, nP = nF(nF+1)/2

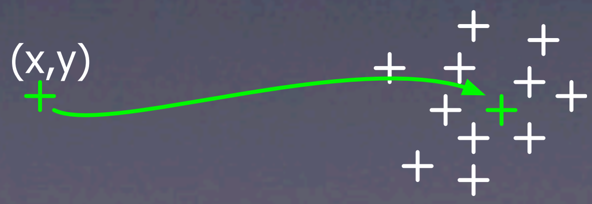
Covariance algorithm part #2

- Two running modes: matching and tracking/searching
- matching of RoI
  - one-on-one matching: RoI association between 1 RoI of image X(t) and 1 RoI of image (t+1)
  - winner takes all strategy.
  - score is the similarity between covariance matrix



- Searching / tracking of RoI
  - each RoI of image X(t) is searched in image X(t+1)
    - with exhaustive search : new position is at best score (winner takes all)

    - with Monte-Carlo search: new position is the average of random positions weighted by the scores (robust to distractors)
    - typically 40 random positions



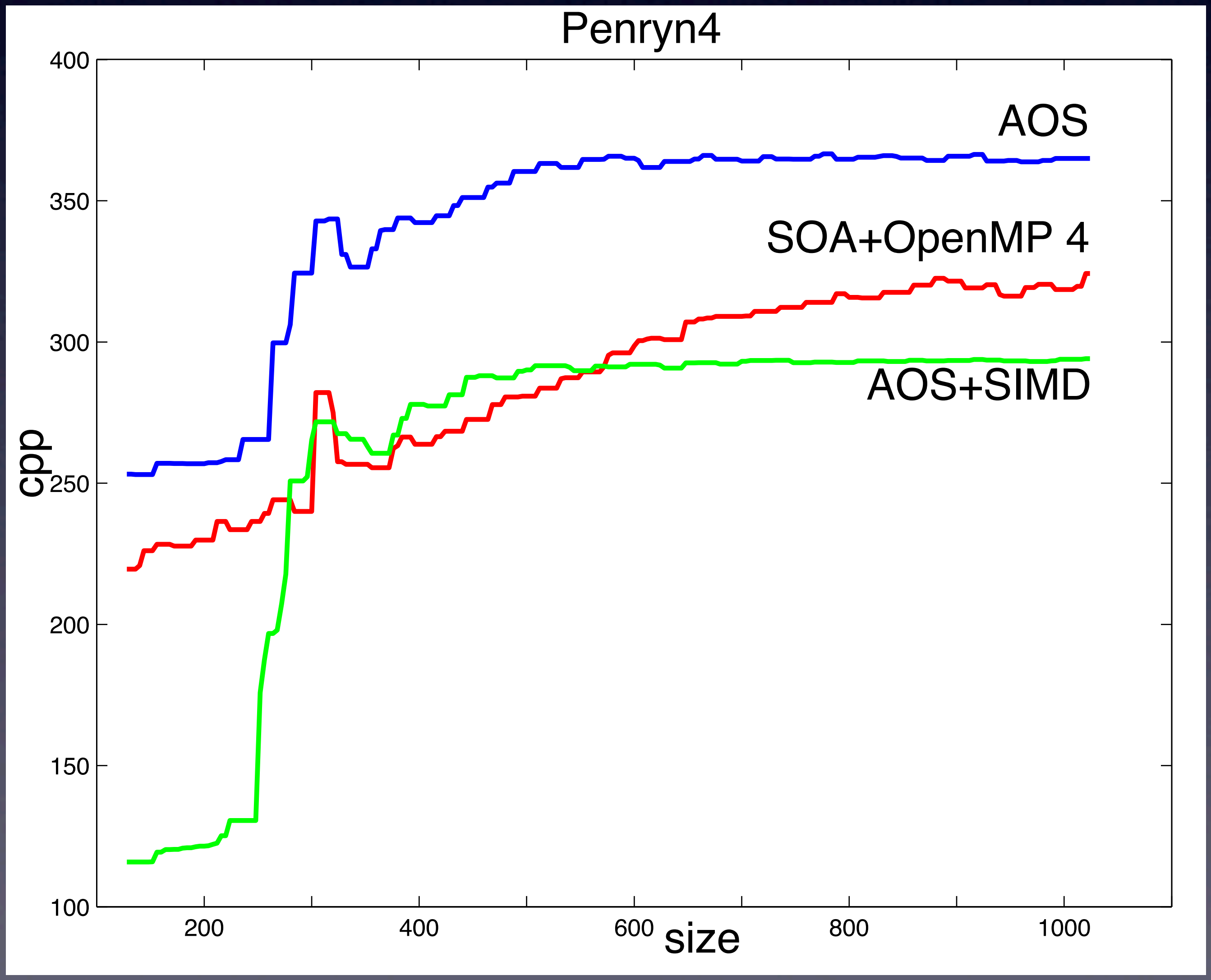
  -

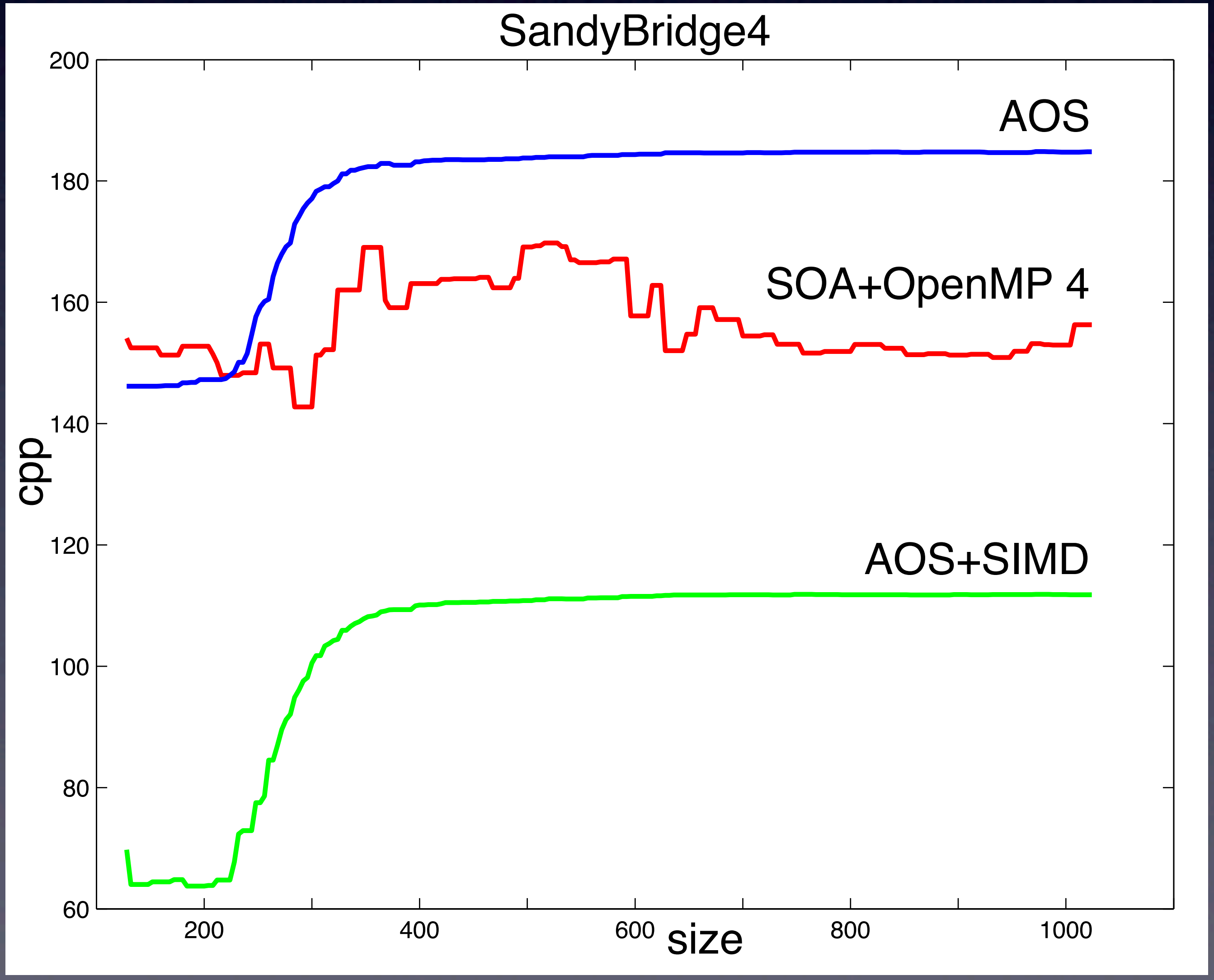
Algorithm optimizations

- The algorithm is composed of 3 parts
  - features computation
  - kernel part = {product of features and integrale image computation}
  - tracking / searching
- First benchmark analysis: the kernel part is the most time consumming: about 80% of total time
- First optimization: cache aware algorithm with models of parallelization
  - two data memory layouts: Array of Structures (AoS) or Structure of Arrays (SoA)
  - AoS enables SIMD computations (Instruction Level Parallelism = ILP)
  - SoA enables thread parallelization with OpenMP (Task Level Parallelism = TLP)
- Benchmark on 3 generations of Intel processors
  - 4-C Penryn, 8-C Nehalem and 4-C SandyBridge

Results on GPP #1: SIMD or OpenMP ?

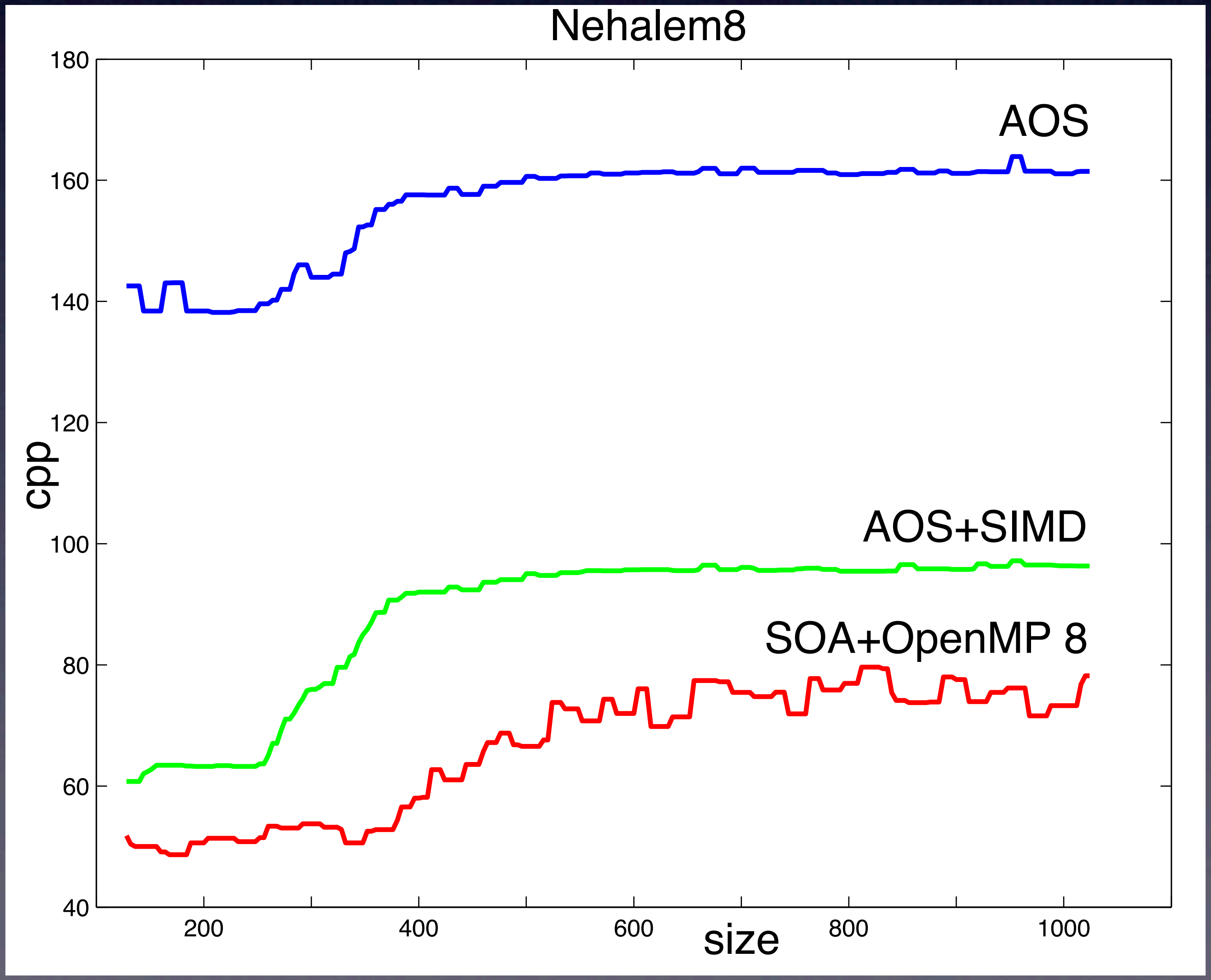
- What is the most efficient parallel model, OpenMP or SIMD ?
  - execution time in cycles per point (cpp) for image size from 128x128 to 1024x1024
  - with 4 cores, AOS+SIMD is more efficient than SOA+OpenMP4
  - with a faster DRAM bus, SandyBridge is x2 faster than Penryn ...
  - very early cache overflow (when data doesn't fit in the cache) (around 200x200)





Results on GPP #2: SIMD or OpenMP ?

- On bi-quad Nehalem
  - 8 cores with scalar computations match 1-core with SIMD
  - SOA+OpenMP is not efficient on GPP
  - and even more on embedded systems with a smaller number of core (Cortex-A9: up to 4 cores, Cortex-A15: 2 cores only)
  - => AOS+SIMD is the memory layout / parallelism chosen



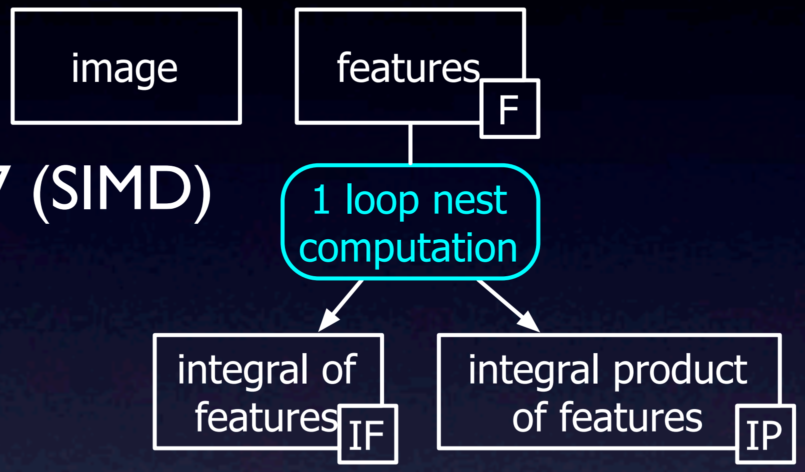
Covariance complexity

- Two embedded systems, focus on kernel part of the algorithm
  - 4 configurations {Intel Penryn ULV, ARM Cortex-A9} x {scalar, SIMD}
  - complexity = arith {MUL+ADD}, memory access {LOAD+STORE}, Arithmetic Intensity (AI) (arith/mem)
- Observation
  - low AI due to too many memory accesses == SIMD won't be efficient :-(
  - => reduce memory accesses by loop fusion (quite tricky ...)

instructions	MUL	ADD	LOAD	STORE	AI
AoS scalar version with 3 loops					
product of features	n_P	0	$2n_P$	n_P	-
integral of features	0	$3n_F$	$4n_F$	n_F	-
integral of products	0	$3n_P$	$4n_P$	n_P	-
total	n_P	$3(n_P+n_F)$	$6n_P+4n_F$	$2n_P+n_F$	-
total with $n_P = n_F(n_F+1)/2$	$2n_F^2+5n_F$		$4n_F^2+9n_F$		-
total with $n_F = 7$	133		259		0.5
AoS SIMD (with $n_F = 7$) version with 3 loops					
product of features	7	0	2	7	-
integral of features	0	21	28	7	-
integral of products	0	6	2	2	-
total SSE (+ 15 PERM)	49		54		0.9
total Neon (+ 48 PERM)	82		54		1.5

advanced Loop Transform (multiple fusions)

- Loop Fusion
  - instead of 3 loop nests to produce Products (P), Integral Features (IF), Integral Products (IP),
  - only 1 loop nest to produce IF and IP, without access (load & store) to Products

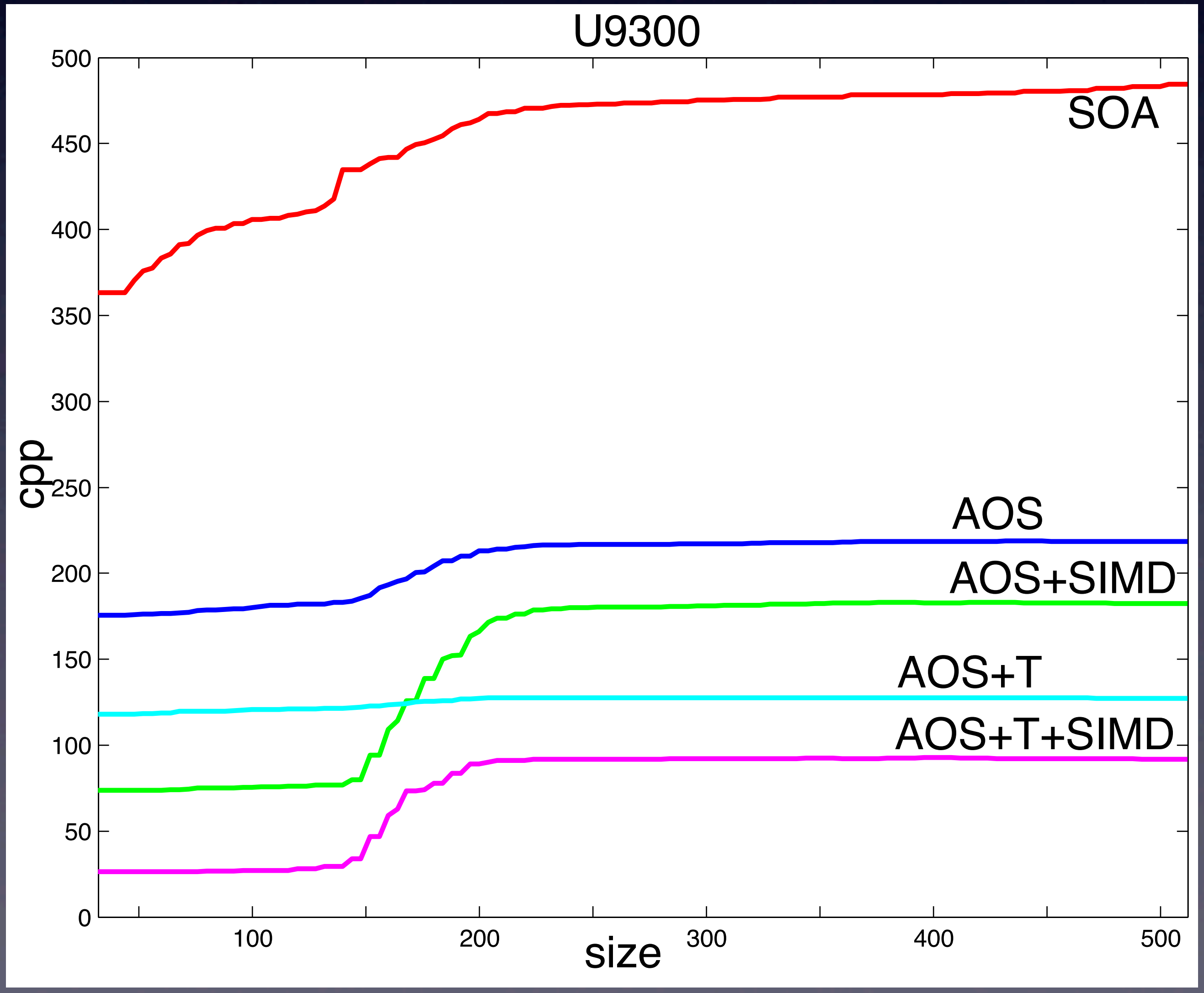


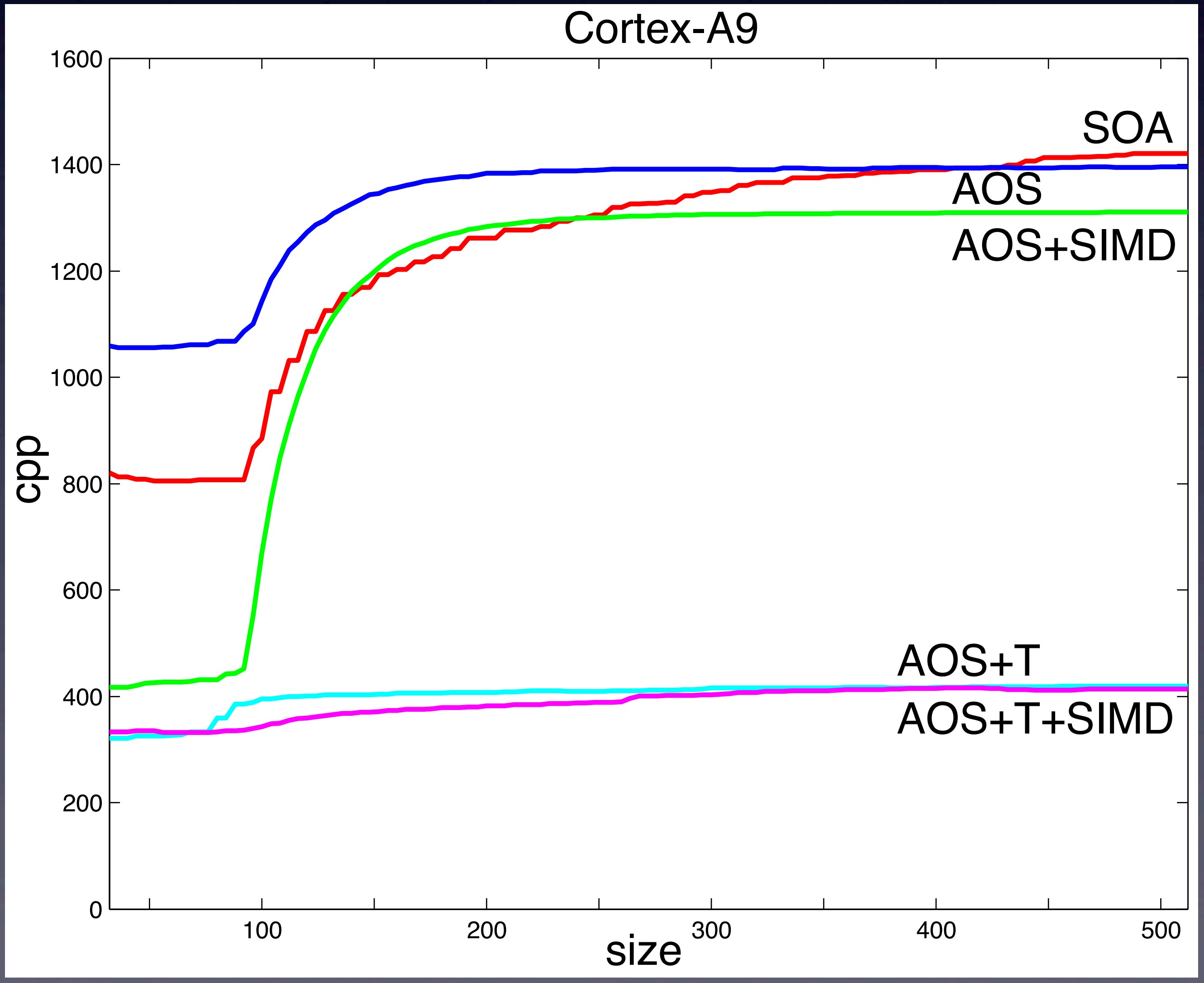
  - amount of memory accesses has been divided by 3.36 (scalar) 2.7 (SIMD)
  - less stress on memory buses

instructions	MUL	ADD	LOAD	STORE	AI
AoS scalar version + Loop Fusion					
integral of features	0	$2n_F$	$2n_F$	n_F	-
integral product of features	n_P	$2n_P$	n_P	n_P	-
total	n_P	$2(n_P + n_F)$	$n_P + 2n_F$	$n_P + n_F$	-
total with $n_P = n_F(n_F+1)/2$	$1.5n_F^2 + 3.5n_F$		$n_F^2 + 4n_F$		-
total with $n_F = 7$	98		77		1.3
AoS SIMD (with $n_F = 7$) version + Loop Fusion					
integral of features	0	4	4	2	-
integral product of features	7	14	7	7	-
total SSE (+ 15 PERM)	40		20		2.0
total Neon (+ 48 PERM)	73		20		3.7

Benchmarks - Loop Transform (Fusion)

- Intel Penryn ULV 9300 (1,2 GHz)
  - Loop Transform provides a ~ x2 compared to AOS & AOS+SIMD. Total speedup = x5.3
- ARM Cortex A9 (1.0 GHz)
  - AoS & AoS+SIMD are not efficient compared to SoA (reasons: memory bandwidth, cache perf)
  - advanced loop transforms are mandatory : speedup x3.4





Benchmarks - Intel Penryn ULV U9300

- Observation
  - kernel duration divided by x6.9 => total duration divided by x2.9
  - real-time execution on 1 core for 312x233, 2 cores for 640x480

sequence	panda		pedxing	
size	312 x 233		640 x 480	
algorithm version	SoA	AoS+SIMD+T	SoA	AoS+SIMD+T
features computation (cpp)	128	150	128	150
kernel computation (cpp)	599	87	618	91
tracking (cpp)	23	23	11	11
total (cpp)	738	248	769	264
kernel / total ratio	81 %	35 %	80 %	34 %
total speedup	x 2.9		x 2.9	
1-core execution time (ms)	45	15	197	68
2-core execution time (ms)	36	9	158	38

cpp & execution time (ms) for Intel Penryn ULV U9300

Benchmarks - ARM Cortex-A9

- Observation
  - kernel duration divided by x3.7 => total duration divided by x2.2
  - real-time execution on 2 core for 312x233

sequence	panda		pedxing	
size	312 x 233		640 x 480	
algorithm version	SoA	AoS+SIMD+T	SoA	AoS+SIMD+T
features computation (cpp)	461	461	486	486
kernel computation (cpp)	1491	395	1600	415
tracking (cpp)	96	96	19	19
total (cpp)	2048	952	2106	921
kernel / total ratio	73 %	42 %	73 %	45 %
total speedup	x 2.2		x 2.2	
1-core execution time (ms)	149	69	647	283
2-core execution time (ms)	108	36	492	149

cpp & execution time (ms) for ARM Cortex-A9

Conclusion & future works

- Conclusion
  - Covariance matching / tracking is a robust and parametrizable algorithm
  - agility to tune features to nature of image

  - Real-time execution on embedded processors (ARM Cortex, Intel ULV)
  - agility to adapt the number of features to the computation power
  - huge impact of High Level Transforms (x6.9 x3.7) : an efficient compiler is not enough !

- Future works
  - enhanced feature-matching with kinematic tracking
  - benchmark algorithm on Cortex A15 (better pipeline throuput)
  - port algorithm to many-cores architecture :
    - embedded system Kalray MPPA and or Tilera TileGX (640x480 & 720p multi-target tracking)
    - High Performance Computing Intel Xeon-Phi (HD 1080p multi-target tracking)

video examples

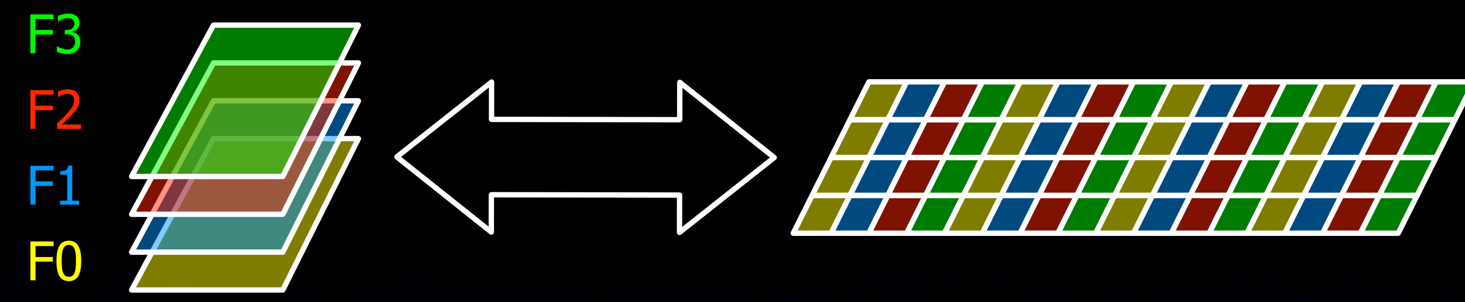
- Pedxing
  - pedestiran crossing
  - lot of cluter due to jpeg/mpeg compression (block boundaries)
- Panda
  - "slow motion" panda but with
  - high variability (black & white != white & black)
- PETS 2009
  - multi-target tracking

Thanks !

www.lri.fr/~lacas

SoA vs AoS: product of features

- Memory layout
  - SoA = a cube of 1 matrix by feature
  - AoS =1 matrix of interlaced features
  - SIMDization is possible because nF >= cardinal(SIMD) = 4 with SSE and Neon



SoA versus AoS for nF = 4 features

Algorithm 1: product of features - *SoA* version

$k \leftarrow 0$
foreach $k_1 \in [0..n_F - 1]$ **do**
  foreach $k_2 \in [k_1..n_F - 1]$ **do**
    foreach $i \in [0..h - 1]$ **do**
      foreach $j \in [0..w - 1]$ **do**
        $P[k][i][j] \leftarrow F[k_1][i][j] \times F[k_2][i][j]$
        $k \leftarrow k + 1$

Algorithm 1: product of features - *AoS* version

foreach $i \in [0..h - 1]$ **do**
  foreach $j \in [0..w - 1]$ **do**
    $k \leftarrow 0$
    foreach $k_1 \in [0..n_F - 1]$ **do**
      foreach $k_2 \in [k_1..n_F - 1]$ **do**
        $P[i][j \times n_P + k] \leftarrow F[i][j \times n_F + k] \times F[i][j \times n_F + k]$
        $k \leftarrow k + 1$

SoA vs AoS: integrale images

- Memory layout
  - SoA = a cube of 1 matrix by feature
  - AoS =1 matrix of interlaced features

Algorithm 1: integral image - *SoA* version, $n \in \{n_F, n_P\}$

foreach $k \in [0..n-1]$ **do**
  foreach $i \in [0..h-1]$ **do**
    foreach $j \in [0..w-1]$ **do**
      $I[k][i][j] \leftarrow I[k][i][j] + I[k][i][j-1] + I[k][i-1][j] - I[k][i-1][j-1]$

Algorithm 1: integral image - *AoS* version, $n \in \{n_F, n_P\}$

foreach $i \in [0..h-1]$ **do**
  foreach $j \in [0..w-1]$ **do**
    foreach $k \in [0..n-1]$ **do**
      $I[i][j \times n + k] \leftarrow I[i][j \times n + k] + I[i][(j-1) \times n + k] + I[i-1][j \times n + k] - I[k][i-1][(j-1) \times n + k]$

Covariance tracking initialisation

- How to initialize tracking / searching ?
  - not addressed in the paper but:
  - still camera:
    - background substraction [DASIP 2012] [ICIP 2009] [JRTIP 2008]
    - mixture of Gaussian (abnormal intensity) [ISIVC 2012]
  - camera in motion:
    - HoG (Histogram of Gradient) or features recognition [Viola Jones]
    - optical flow segmentation (abnormal flow) [ISIVC 2012]