POLITECNICO DI MILANO

Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

Master of Science in
Computer Engineering

VISUAL FEATURE EXTRACTION ON LOW-POWER SENSOR NODES

Supervisor: Prof. Marco Tagliasacchi
Assistant Supervisor: Engr. Alessandro Redondi

Master Graduation Thesis by: Luca Baroffio
Student Id. number: 765434

Academic Year 2011/2012

POLITECNICO DI MILANO

Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

Corso di Laurea Magistrale in Ingegneria Informatica

VISUAL FEATURE EXTRACTION ON LOW-POWER SENSOR NODES

Relatore: Prof. Marco Tagliasacchi
Correlatore: Ing. Alessandro Redondi

Tesi di laurea di: Luca Baroffio
Matricola: 765434

Anno Accademico 2011/2012

Abstract

The innovative and forerunner vision about a pervasive Internet of Things in a not too distant future needs some efficient algorithms to provide some low-power nodes of the network with an improved ability to sense data and extract semantic information from the environment. This work aims at adapting a visual feature extraction algorithm to such a scenario and at evaluating the results in terms of performance and efficiency. A survey of the literature about visual feature extraction algorithms and sensor networks with computational efficiency as a key requirement will introduce the choice of SURF as a reference implementation. The development of the algorithm on a sensor node is followed step by step, introducing any issues arising during each stage of the process and their solutions. An exhaustive performance evaluation is then presented, including both a quantitative analysis and some qualitative comments about the results.

Sommario

La visione innovativa di una Internet degli Oggetti in un futuro non molto distante necessita di algoritmi efficienti che permettano ai nodi della rete a basso consumo di rilevare dati dall'ambiente e ricavarne informazione. Questo lavoro ha l'obiettivo di adattare un algoritmo per l'estrazione di visual features a questo scenario e di valutarne i risultati in termini di prestazioni ed efficienza. Una indagine accurata della letteratura riguardo all'estrazione di visual features e alle reti di sensori, con l'efficienza computazionale come requisito principale, farà da introduzione alla scelta di SURF come implementazione di riferimento. Lo sviluppo dell'algoritmo su un nodo sensore è seguito passo dopo passo e le eventuali problematiche sorte nelle diverse fasi del processo sono presentate, insieme alle soluzioni adottate. Infine, gli esiti di una esauriente analisi delle prestazioni saranno riportati, includendo dati quantitativi e alcuni commenti sui risultati ottenuti.

Acknowledgements

Foremost, I would like to express my gratitude to my advisor Prof. Marco Tagliasacchi for his willingness and for the continuous support offered during the preparation of my thesis. This has been a challenging yet rewarding experience and I am really thankful for the opportunity to work on such an interesting topic.

Besides my advisor, I would like to thank my assistant advisor Alessandro Redondi for the support throughout the whole thesis preparation period and especially on resolving the many issues I had to face.

My sincere thanks also goes to Prof. Matteo Cesana for giving me the opportunity to work at the ANTLab and for his cordiality.

I am indebted to my many student colleagues who constituted a stimulating and fun environment in which to learn and grow.

Lastly, but most importantly, I would like to thank my extended family who always believed in me and made all this possible, and Flavia, my personal and beloved motivator to get the best out of me.

Contents

List of Figures

List of Tables

Introduction

Over the last few years a large body of research has focused on visual data analysis to accomplish high-level tasks. Images or videos are often acquired in digital format by sampling and quantizing the lightfield on a fixed set of locations. In most cases image analysis is only the last part of the chain: firstly image acquisition is performed (e.g. using a digital camera), then a compressed representation of such image is stored and transmitted to a central node for the processing. Such paradigm has been successfully employed over many years for a wide range of visual analysis tasks and applications. In recent years, however, the advent of new technologies such as Internet of Things and the continuous improvement of computing devices (in terms of computational power, miniaturization, power requirements, etc.) is leading to a new scenario in which battery-operated sensing nodes are empowered with sight and capable of carrying out visual analysis tasks without resorting to a central unit. Such sensing devices should have the ability to acquire visual data from the environment, to process this data extracting information, to communicate with other nodes over a network. In particular, the ability to interact with other devices could be further exploited delegating some of the computational burden to communicating nodes.

Taking into account these requirements, Wireless Multimedia Sensor Networks seems to be an adequate solution. This technology provides a set of nodes equipped with several different sensors and able to communicate thanks to an ad-hoc network protocol. This topic will be further investigated in section 1.1.

The paradigm in which network nodes will be not only responsible for data

acquisition but also for task processing will try to imitate the human visual system, where stimuli are captured by the eyes, the raw data is analyzed by the brain and then some key concepts are extracted and "stored" into our memory. Moreover, this process has been demonstrated to have a very low energy expenditure and to be very efficient (duration of a few hundreds milliseconds).

The computer vision counterpart of human visual concepts are visual features, i.e. some interest points within the image with a particular "description" that allows a feature to be compared with the other ones. An interest point can be defined as follows:

- it has a clear, preferably mathematically well-founded, definition;
- it has a well-defined position in image space;
- the image structure around the interest point is rich in terms of local information contents allowing for an high descriptiveness and simplifying further processing;
- it is stable under local and global perturbations in the image domain, including perspective transformations as well as illumination/brightness variations, such that the interest points can be reliably computed with high degree of reproducibility;
- optionally, an interest point should include an attribute of scale, to allows similar keypoints obtained from different scales (i.e. under different levels of zoom) to be effectively compared.

Such visual features are obtained by coupling a detector that identifies the interest points and a descriptor that provides an effective characterization. The third requirement is key for an effectual descriptor to be built: several different methods have been proposed to provide a well-defined characterization of the interest point based on its local content (e.g. color information, local gradient, dominant orientation, etc.). A detailed overview of some of the most important algorithms for keypoints detection and description is proposed in section 1.2.

Although sensing nodes performance is constantly getting better, they are not designed to carry out the intensive processing load needed by some

visual feature extraction algorithms. Their limitations in terms of low power consumption, production costs and computational power are to be taken into account while developing the algorithm. A tradeoff between computational performance (in terms of time, required memory and resources, etc.) and operating efficiency (evaluated with different metrics such as repeatability, precision, recall, etc.) is to be found. A detailed analysis of the problem of extracting visual features on a sensor node is presented in Chapter 2 along with the methodology followed during the development of the algorithm.

Then some details about the implementation and the adopted solutions are introduced in Chapter 3 and accompanied by some code snippets.

Finally in Chapter 4, some ad-hoc metrics to evaluate the implemented application are defined and presented; the performance of the algorithms is investigated and compared to other approaches and the results are shown.

Glossary

- W(M)SN: Wireless (Multimedia) Sensor Network
- SURF: Speeded Up Robust Feature
- SIFT: Scale-invariant Feature Transform
- FAST: Features from Accelerated Segment Test
- BRISK: Binary Robust Invariant Scalable Keypoints
- BRIEF: Binary Robust Independent Elementary Features
- DARPA: Defense Advanced Research Projects Agency
- MEMS: MicroElectroMechanical Systems
- WINS: Wireless Integrated Network Sensors
- WPAN: Wireless Personal Area Network
- WLAN: Wireless Local Area Network
- QoS: Quality of Service
- MIPS: Million Instructions Per Second
- RISC: Reduced Instruction Set Computer
- ROM: Read-Only Memory
- RAM: Random-Access Memory
- SDRAM: Synchronous Dynamic Random-Access Memory
- ISO/OSI: International Organization for Standardization/Open Systems Interconnection
- DoG: Difference of Gaussians
- PC: Personal Computer

- TOSSIM: TinyOS SIMulator
- GLOH: Gradient Location and Orientation Histogram

Chapter 1

Review of the State of the Art

1.1 Wireless Multimedia Sensor Networks

A Wireless Multimedia Sensor Network consists of a set of spatially distributed autonomous sensors (a.k.a. sensor network nodes or motes) capable of monitoring physical or environmental conditions and of cooperatively transmitting their data through the network[26]. The forerunner of this kind of technology dates back to the late 70's, when DARPA sponsored the development of Distributed Sensor Nets for military purposes[6]. The introduction of MEMS components in the 90's allows the development of a new generation of low-power yet efficient sensor networks such as WINS[1].

A WSN is composed by a set of nodes, from a few ones up to hundreds or even thousands. Each node consists of several parts, usually a radio transceiver unit, a microcontroller and a source of energy (battery, solar power). The sensing capabilities are provided by modular boards that can contain several different sensors.

The applications of such technologies are really diversified[3]:

- Area monitoring (surveillance, intrusion detection, hazard detection, ..);
- Air monitoring (athmosphering monitoring, gas concentration, pollution, ..);
- Hazards monitoring (landslide, fire, ..)

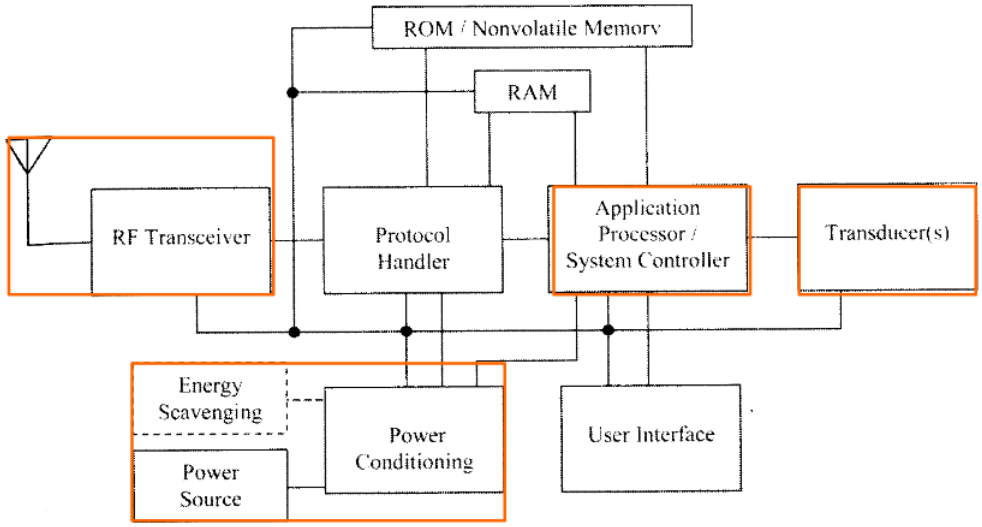


Figure 1.1: WSN architecture[3]

- Localization (people or vehicle tracking and monitoring, ..);
- Machine health monitoring (machinery working conditions, data logging, ..);

1.1.1 Sensors

Sensors are organized into multi-purpose boards (see Figure 1.2) that can be connected with the mainboard. They can be classified into analog and digital sensors depending on the type of the output signal. The former category needs a dedicated analog-to-digital converter for the data to be managed, stored or transmitted[1]. Some popular sensors are[6]:

- camera;
- light sensor (IR, ultraviolet, ..);
- magnetic sensor;
- microphone;

[1]The ADC introduces a new kind of problems about the minimum sampling rate and therefore transmission rate for a signal to be correctly represented.

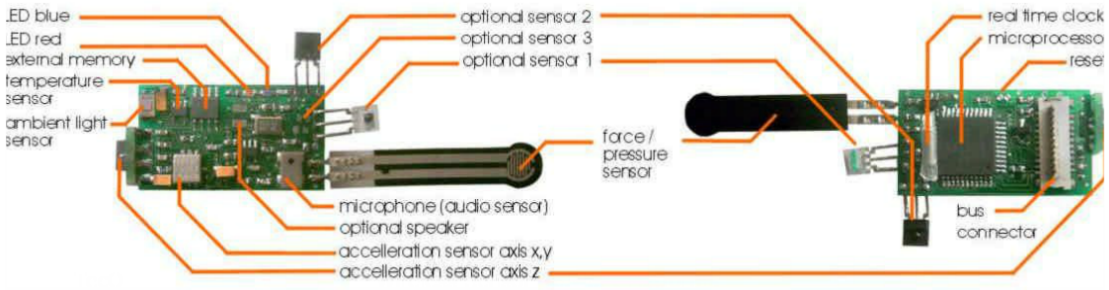


Figure 1.2: a sensor board and its connections with the mainboard[?].

- accelerometer;
- thermometer;
- humidity sensor;
- pressure sensor;
- touch sensor.

The transmission of the data from the sensors to the mainboard can be controlled by different interfaces (Serial Peripheral Interface I2C, Serial, ..).

1.1.2 Microprocessors and memory

In the mainboard design phase MIPS, amount of ROM and RAM are taken into account, as well as power consumption, cost, size and many other variables. Here is a quick overview of some of the most popular microprocessors for the nodes[23]:

Atmel AVR ATmega 128	**TI MSP430**	**Intel PXA271 XScale**
8 bit RISC	16 bit RISC @ 8MHz	32 bit RISC @ 13-416 MHz
32 registers	16 registers	16 registers
4kB RAM	10kB RAM	256kB SRAM
128kB Flash	48kB Flash	32MB Flash
4kB EEPROM	16kB EEPROM	32MB SDRAM
up to 20 MIPS	up to 16 MIPS	-
$<$ 10mA @ 5 MIPS	$<$ 2mA @ 5MIPS	$<$ 50mA @ 104MHz[27]

Table 1.1: microprocessor comparison

1.1.3 Network performance objective and features

- Low power consumption:
  - WSNs typically require an average power consumption significantly lower than the one of existing wireless network implementations (e.g. Bluetooth). This is a key requirement, especially in a large network where frequent battery replacements could be really impractical. Also in industrial equipment long-lasting battery could be needed to respect the usual maintenance schedule;
- Low cost:
  - For several applications the need for a large number of nodes made the low cost of a single mote a key requirement. To meet the objective, expensive components are to be avoided while self-configuration and self-maintenance are to be provided to minimize support expenses;
- Network types:
  - The optimal network topology may vary depending on many variables, such as the area to be covered, the allowed network power (also depending on government regulations), the battery life. In this sense the support of many types of network without large working overhead is a critical factor;
- Network security:
  - Some of the applications of the technology require a high level of security. Besides encryption, also authentication and integrity checking play fundamental roles, not allowing a message to be falsified or maliciously introduced into the network;
- Data throughput:
  - WSNs have a limited data throughput compared to other WPANs and WLANs. Also considering the network overhead, this results in a low efficiency, no matter the chosen design;

- Message latency:
  - WSNs have a really varying Quality of Service (QoS) and do not support isochronous or synchronous communication. Moreover, the real-time stream of video or even voice is too expensive. Therefore message latency is a very relaxed requirement in comparison to that of other WPANs.

1.1.4 Network protocols

The requirements listed above lead to strict constraints on network protocols to be adopted in WSNs. Short-range wireless technology represents a cornerstone due to the low power budget of each node. Several institutions are involved in the development of ad-hoc protocols for WSNs. IEEE 802.15.4[25] is a key for the Media Access Control and the Physical ISO/OSI layers, offering a solid foundation, while some different technologies such as ZigBee, WirelessHART an MiWi extend that standard by offering an implementation of the upper layers, not defined in IEEE 802.15.4.

1.1.5 Intel Imote2

The Intel Imote2 is an advanced wireless sensor node platform. It is built around the low power PXA271 XScale CPU and also integrates a 802.15.4 compliant radio. The design is modular and stackable with interface connectors for expansion boards on both the top and bottom sides. A battery board supplying system power can be connected to either side[5].

Hardware Features[24]

- PXA271 XScale Processor
  - Core Frequency: 13/104/208/312/416 MHz
  - 256 KB SRAM
  - 32 MB SDRAM
  - 32 MB Flash
- Zigbee (IEEE 802.15.4) Radio (TI CC2420)

- Mini-USB Client (slave)
  - RS232 console over USB
  - power
- I-Mote2 Basic Sensor connector (31+ 21 pin connector)
- Indicators: Tri-color status LED, power LED, battery charger LED, console LED
- Switches: on/off slider, Hard reset, Soft reset, User programmable switch

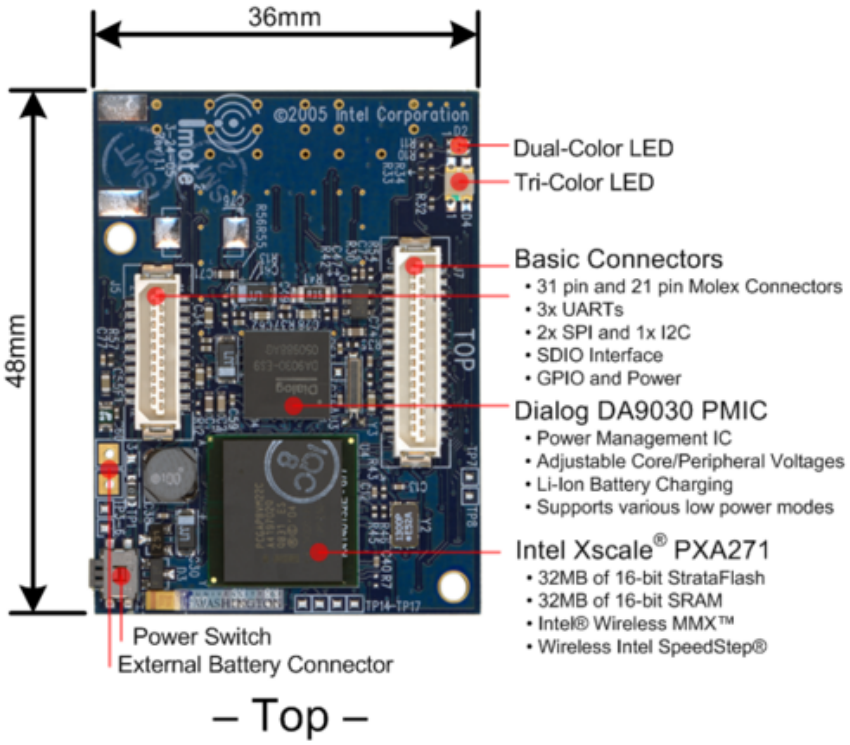


Figure 1.3: top view of an Intel Imote2.

1.2 Visual features extraction algorithms

1.2.1 Harris corner detector

This approach for combined corner and edge detection was introduced by Harris and Stephens[11] in 1988. The key principle is the assumption that, at the location of a corner, the image intensity is widely variable along several directions. Starting from this tenet, let's consider an image patch $I(u, v)$ and its counterpart $I(u + x, v + y)$, shifted by (x, y). Then the sum of squared differences is defined a

$$S(x, y) = \sum_u \sum_v w(u, v) \left(I(u + x, v + y) - I(u, v)\right)^2$$

The shifted patch can be approximated by the following Taylor's expansion:

$$I(u + x, v + y) \approx I(u, v) + I_x(u, v)\, x + I_y(u, v)\, y$$

where I_x and I_y represents the first order partial derivatives in the x and y directions respectively. Then, substituting $I(u + x, v + y)$, $S(x, y)$ can be rewritten as

$$S(x, y) = \sum_u \sum_v w(u, v) \left(I(u + x, v + y) - I(u, v)\right)^2 =$$

$$= \sum_u \sum_v w(u, v) \left(I(u, v) + I_x(u, v)\, x + I_y(u, v)\, y - I(u, v)\right)^2 =$$

$$= \sum_u \sum_v w(u, v) \left(I_x(u, v)\, x + I_y(u, v)\, y\right)^2$$

This can be written as $S(x, y) = \begin{pmatrix} x & y \end{pmatrix} H \begin{pmatrix} x \\ y \end{pmatrix}$, where

$$H = \sum_u \sum_v w(u, v) \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix}$$

Then, a corner is identified if both the eigenvalues of the matrix H are sufficiently large, i.e. if the variation of S in all directions of the vector $\begin{pmatrix} x & y \end{pmatrix}$ is sufficiently large. In particular, the Harris measure quantifies that amount of variation:

$$M_c = \lambda_1 \lambda_2 - k\,(\lambda_1 + \lambda_2)^2$$

1.2.2 SIFT

Scale-Invariant Feature Transform is an algorithm to detect and describe local features in an image proposed by David Lowe[14] in 1999. It aims at identifying locations in image scale space that are invariant with respect to translation, scaling, and rotation, and are minimally affected by noise and small distortions.

Detector

For the detection phase, a scale space is built by applying Gaussian filters with increasing variance to the image[2]. Then the couples of Gaussian-smoothed adjacent images are subtracted to obtain the Difference of Gaussians (see Figure 1.4):

$$D(x, y,) = (G(x, y, k)G(x, y,)) * I(x, y)$$

$$G(x, y,) = \frac{1}{2\pi\sigma^2} e^{-(\mathrm{x}^2+\mathrm{y}^2)/2\sigma^2}$$

$D(x, y, \sigma)$ is an approximation of the Laplacian of Gaussian $L(x, y,)$, defined as

$$^2|\bigtriangledown^2 L(x, y,)| = {}^2|tr[H(x, y,)]| = {}^2|L_{xx}(x, y,) + L_{yy}(x, y,)|$$

where L_{xx} and L_{yy} represent the convolution of the Gaussian second order derivative along the x and y directions with the image.

Once such scale-space is built, key locations are selected in correspondence

[2]Scale space is divided into octaves. For each octave the image is convolved with Gaussian kernels and downsampled by a factor of two to provide the DoG images in a "pyramidal" form.

with its local minima and maxima. For this step to be carried out, each pixel is compared with the neighboring ones in the 3D scale-space (see Figure 1.5).

The location of each keypoint is then refined by fitting a 3D quadratic function to the points and then finding the interpolated location of the maximum. This approach uses the Taylor expansion up to the quadratic terms of the scale space function $D(x, y, \sigma)$:

$$D(\underline{\mathrm{x}}) = D + \frac{\partial D^T}{\partial \underline{\mathrm{x}}}\underline{\mathrm{x}} + \frac{1}{2}\underline{\mathrm{x}}^T\frac{\partial^2 D}{\partial \underline{\mathrm{x}}^2}\underline{\mathrm{x}}$$

where D and its derivatives are evaluated at the sample point and $\underline{\mathrm{x}} = (x, y,)^T$ is the offset from this point. The location of the extremum, $\underline{\hat{x}}$, is determined by taking the derivative of this function with respect to x and setting it to zero, giving

$$\underline{\hat{x}} = -\frac{\partial^2 D^{-1}}{\partial \underline{\mathrm{x}}^2}\frac{\partial D}{\partial \underline{\mathrm{x}}}$$

This method also allows for unstable extrema (i.e. the ones with a low contrast value) to be discarded.

The DoG approach results in a strong response along the edges: poorly defined peak will have a strong response along a principal curvature but a weak one along its perpendicular. The keypoints in these locations should be rejected for stability reasons. For doing this, let's consider

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

and let α be the smallest eigenvalue and β be the largest one. Then, the ratio between the two eigenvalues will be a key to discern corner responses from edge ones.

Descriptor

The first step for the description phase is to assign an orientation to each keypoint, to achieve invariance against image rotation. The scale of the keypoint is used to select the proper Gaussian-smoothed image $L(x, y)$, i.e. the one with the closest value of σ.

Then for each pixel of a patch around the key location, magnitude and

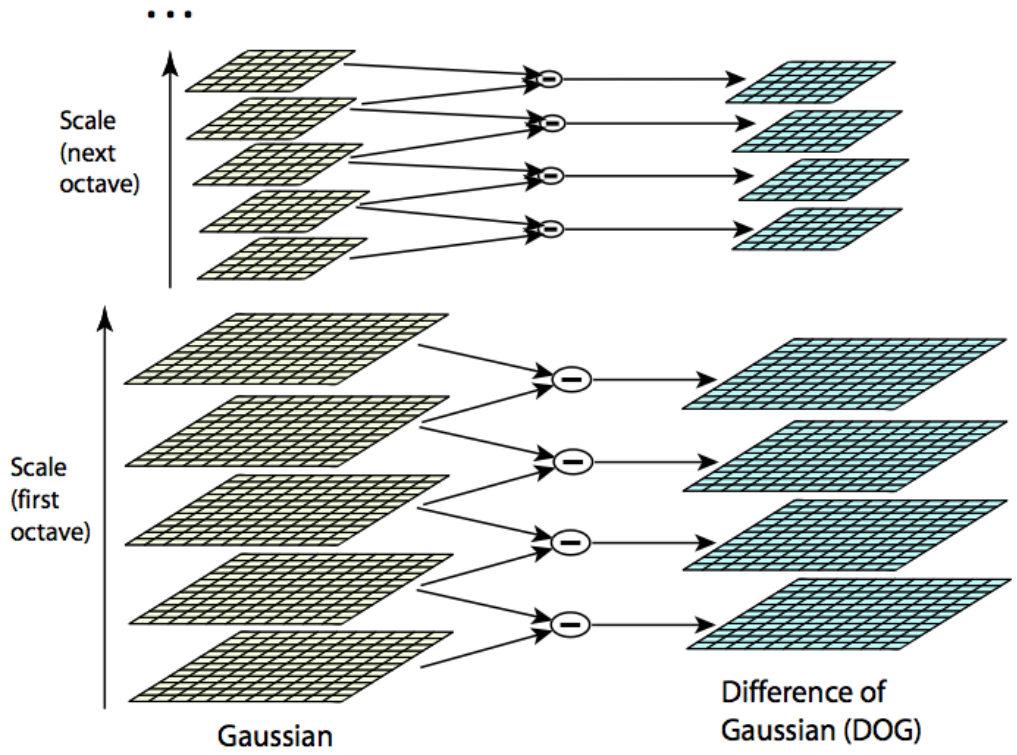


Figure 1.4: scale-space is built by means of Difference of Gaussians.

orientation values are computed starting from $L(x, y)$:

$$m(x, y) = \sqrt{(L(x+1, y)L(x1, y))^2 + (L(x, y+1)L(x, y1))^2}$$

$$(x, y) = tan^{-1}\frac{L(x, y+1)L(x, y1)}{L(x+1, y)L(x1, y)}$$

Each couple of values is added to an histogram: the orientation is used to select the proper bin while the magnitude provides a weighting factor. The maximum of the histogram, after an interpolation process to increase accuracy, will represent the keypoint orientation[3].

A descriptor based on the local properties of the image is to be built and represented in coordinates relative to the estimated orientation, achieving rotation invariance. Gradient magnitudes and orientations are sampled around

[3]if more than one maximum is found, n interest points with different orientations, where n is the number of maxima, are created.

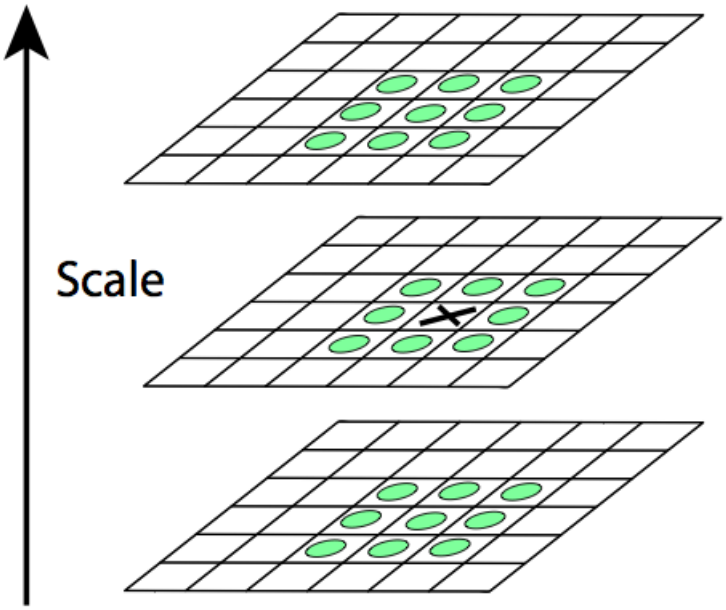


Figure 1.5: maxima and minima of the difference-of-Gaussian images are detected by comparing a pixel (marked with X) to its 26 neighbors in 3x3 regions at the current and adjacent scales (marked with circles).

the keypoint location and then weighted with a Gaussian circular windows to give the less emphasis to samples that are the more distant from the keypoint location. The gradient values will then fill some orientation histograms that, after being normalized to unitary length, will represent the descriptor vector (see Figure 1.6).

1.2.3 SURF

Speeded Up Robust Feature is a local feature detector and descriptor introduced by Herbert Bay et al.[2] in 2006. Partly inspired by SIFT, the standard version of SURF is several times faster than SIFT and claimed by its authors to be robust against different image transformations.

Detector

SURF detector is based on an approximation of the Hessian-matrix

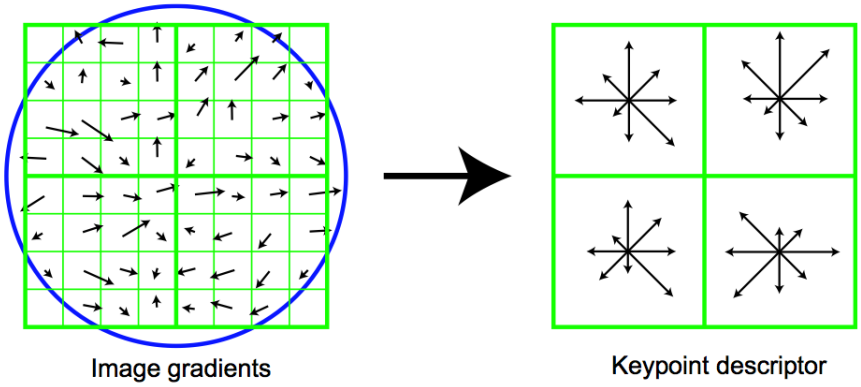


Figure 1.6: On the left, the gradient magnitude and orientation computed at each image sample point in a region around the keypoint location and by a Gaussian window (the blue circle). These samples are then accumulated into orientation histograms, as shown on the right, with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. This figure shows a 2x2 SIFT descriptor array computed from an 8x8 set of samples, whereas the experiments in this paper use 4x4 SIFT descriptors computed from a 16x16 sample array.

$$H(x,\sigma) = \begin{bmatrix} L_{xx} & L_{xy} \\ L_{xy} & L_{yy} \end{bmatrix}$$

For a fast and efficient computation, it exploits boxlets[4] and defines integral images as

$$I_{\Sigma}(x) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i,j)$$

In particular, the samples of the discretized Gaussian kernel are rounded to the nearest integer to provide a boxfilter as in Figure 1.7. This filters are convolved with the image and the approximated Hessian determinant is computed:

$$det(H_{approx}) = D_{xx}D_{yy}(w\, D_{xy})^2$$

$$w = \frac{|L_{xy}(1.2)|_F |D_{yy}(9)|_F}{|L_{yy}(1.2)|_F |D_{xy}(9)|_F} = 0.912... \simeq 0.9$$

[4]boxlets is an algorithm that allows fast convolution with discretized versions of the filters.

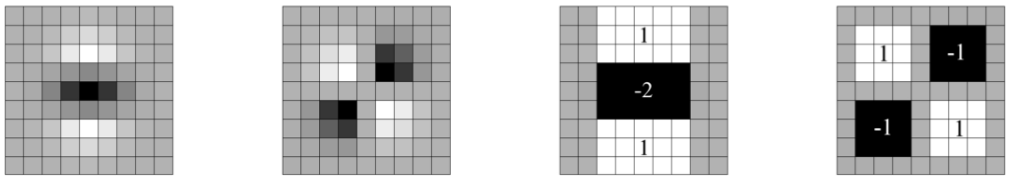


Figure 1.7: Left to right: the (discretised and cropped) Gaussian second order partial derivative in y- (Lyy) and xy-direction (Lxy), respectively; our approximation for the second order Gaussian partial derivative in y- (Dyy) and xy-direction (Dxy). The grey regions are equal to zero.

where $|...|_F$ is the Frobenius norm and w a constant.

As in SIFT, to achieve zoom invariance the keypoints are to be found in a scale-space. Due to the usage of integral images and boxfilters, there is no need to downsample the images, instead the dimension of the boxfilter is iteratively increased. Once the scale-space representation is built, interest points are localized with a non-maximum suppression algorithm[5]. The position and the scale of the keypoints are then refined through an efficient interpolation method.

Descriptor

Similarly to SIFT, SURF descriptor represents the distribution of the intensity content within the interest point neighborhood. First of all, we have to extract a key orientation for each interest point: the Haar wavelet (see figure 1.8) responses in x and y direction within a radius of 6 times the identified scale are calculated. These responses are then Gaussian weighted depending on their distance to the interest point, giving less importance to farther samples. Then the sum of all responses within a sliding window of size $\frac{\pi}{3}$ is calculated and each window represented by a sum vector: the longest one will be chosen as the orientation of the interest point (see Figure 1.9).

The first step for the extraction of a descriptor vector is to select a square patch of side length $20s$ oriented along the estimated direction. The region is then split into 4 x 4 subregions and then, for each one, 5 x 5 regularly spaced Haar wavelet responses d_x (in x direction) and d_y (in y direction) are

[5]For every pixel a 3x3x3 neighborhood is explored and the samples whose value is below the local maximum are set to zero.

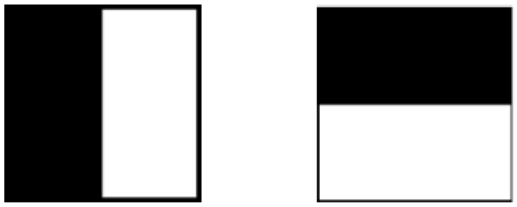


Figure 1.8: Haar wavelet filters to compute the responses in (left) and direction (right). The dark parts have the weight -1 and the light parts +1.

computed (see Figure 1.10). Also in this case a Gaussian window is applied to give less emphasis to outer responses.

Then dx and dy are used to extract four parameters that represent the underlying intensity structure:

- $\sum d_x$;
- $\sum d_y$;
- $\sum |d_x|$;
- $\sum |d_y|$;

Concatenating these results for all the 4 x 4 subregions and after a normalization, a descriptor vector of length 64 is obtained.

1.2.4 FAST

Features from Accelerated Segment Test is a corner detector algorithm proposed by Edward Rosten and Tom Drummond[20, 21] in 2005. The main advantage of FAST corner detector is its computational efficiency.

Detector

The FAST detector consider a circle of 16 pixels around the corner candidate p. The keypoint is accepted if there exists a sequence of n contiguous pixels, belonging to the circle, whose values are greater or lesser than p by a given threshold. FAST sets the parameter n to 12; this choice allows to introduce an efficient method to test the candidate corners: at first only the four pixels at positions 1, 5, 9 and 13 (the four compass directions) are examined (see

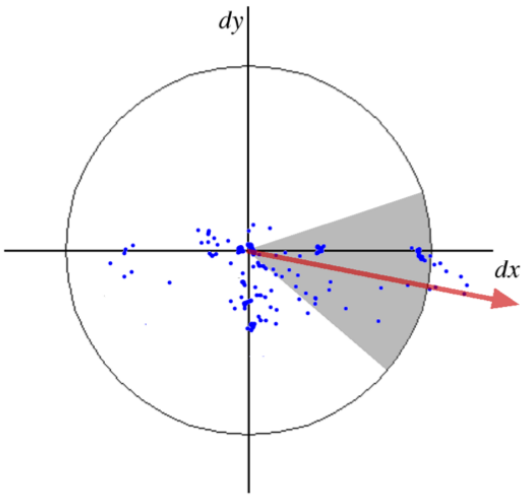


Figure 1.9: dominant orientation estimation is carried on using a sliding window (in grey) and calculating the sum of responses within such window (the red vector).

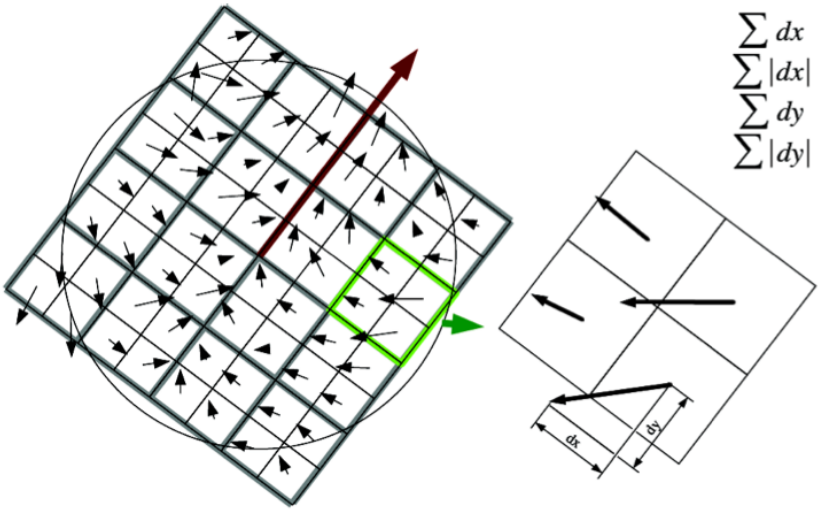


Figure 1.10: the building of SURF descriptor vector. The patch is split into 4 x 4 subregions, Haar wavelet responses are computed and the four descriptive parameters are obtained by summation.

Figure 1.11). If p is a corner, then at least 3 of those pixels should satisfy the test criterion (be greater or lesser than p by a given amount). Examining only four pixels with this method most of the candidates can be efficiently discarded; for the ones that pass this first selection, also the remaining samples on the circle are checked in order to ensure the presence of a true positive.

Machine learning - FAST

The optimization presented above is valid only if n is set to 12, but some tests revealed that the choice $n = 9$ is a better one. Machine learning can be exploited to make the algorithm more efficient also in this case. Provided a training set of images[6], the full algorithm is run to identify which of the 16 pixels on the circle are the most important to decide whether a candidate is a corner.

For each location on the circle $x \in \{1..16\}$ the pixel at that position relative to p (denoted by px) can have one of three states:

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \leq I_p - t & (darker) \\ s, & I_p - t < I_{p\to x} < I_p + t & (similar) \\ b, & I_p + t \leq I_{p\to x} & (brighter) \end{cases}$$

Choosing an x and computing $S_{p\to x}$ for all $p \in P$ (the set of all pixels in all training images) partitions P into three subsets, P_d, P_s, P_b, where each p is assigned to $P_{S_{p\to x}}$.

Let K_p be a boolean variable which is true if p is a corner and false otherwise. The entropy of K for the set P is:

$$H(P) = (c + \bar{c}) \log_2(c + \bar{c}) - c \log_2 c - \bar{c} \log_2 \bar{c}$$

$$\begin{aligned} c &= |\{p|K_p \, is \, true\}| \quad number \, of \, corners \\ \bar{c} &= |\{p|K_p \, is \, false\}| \quad number \, of \, non \, corners \end{aligned}$$

and for each x the information gain corresponds to

$$I_{g_x} = H(P)H(P_d)H(P_s)H(P_b)$$

[6]preferably belonging to a particular application domain.

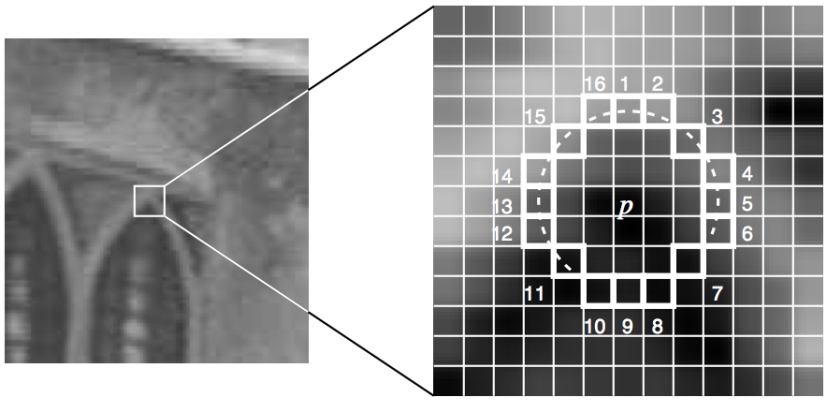


Figure 1.11: FAST segment test corner detection in an image patch. The highlighted squares are the pixels used in the corner detection.

The higher the information gain, the more the point x is important for the classification process. Having selected the x which yields the most information, the process is applied recursively on all three subsets $P_{x,d}, P_{x,s}, P_{x,b}$ and so on and so forth until the entropy of each subset reaches zero.

Exploiting entropy and information gain a decision tree to successfully classify the training data can be generated. Finally, some code generators can build the code to efficiently classify the test data in form of nested if-then-else statements.

Descriptor

Once the keypoints are detected, non-maximal suppression can be used to eliminate the ones that are in the neighborhood of a more defined interest point[7]. As for the descriptor, this type of corner detection naturally suggests using the pixel intensities from the 16 pixel circle as a feature vector, along with a further element describing the type of keypoint (positive if the pixels are greater than the center, negative if they are less than the center).

[7] e.g. a keypoint with $n = n_1$ situated in the neighborhood of another one with $n = n_2$ and $n_2 > n_1$ should be eliminated.

1.2.5 BRIEF

Binary Robust Independent Elementary Features is a feature point descriptor algorithm introduced by Calonder et al.[4] in 2010. It only provides a binary string to describe each keypoint but should rely on a detector to identify the interest points. The key idea is that the image patches could be effectively classified on the basis of a relatively small number of pairwise intensity comparisons. A test on patch p of size $S\text{Œ}S$ is defined as follows:

$$(p; x, y) := \begin{cases} 1 & if\ p(x) < p(y) \\ 0 & otherwise \end{cases}$$

where $p(x)$ is the pixel intensity in a smoothed version of p at $x = (u, v)^T$.

Choosing a set of n_d (x, y)-location pairs defines a set of binary tests. We take our BRIEF descriptor to be the n_d-dimensional bitstring

$$f_{n_d}(p) \triangleq \sum_{1 \le i \le n_d} 2^{i1}(p; x_i, y_i)$$

Several strategies for the choice of the test locations have been tested (see Figure 1.12):

I. $(X, Y) \sim i.i.d.\ Uniform(-\frac{S}{2}, \frac{S}{2})$: the (x_i, y_i) locations are evenly distributed over the patch.

II. $(X, Y) \sim i.i.d.\ Gaussian(0, \frac{1}{25}S^2)$: the tests are sampled from an isotropic Gaussian distribution.

III. $X \sim i.i.d.\ Gaussian(0, \frac{1}{25}S^2)$, $Y \sim i.i.d.\ Gaussian(x_i, \frac{1}{100}S^2)$: x_i is sampled from a Gaussian centered around the origin while y_i is sampled from another Gaussian centered on x_i.

IV. The (x_i, y_i) are randomly sampled from discrete locations of a coarse polar grid.

V. $\forall i : x_i = (0, 0)^T$ and y_i takes all possible values on a coarse polar grid containing n_d points.

The best solution seems to be the sampling of the positions from an isotropic Gaussian distribution with a variance $\sigma^2 = \frac{1}{25}S^2$, where S is the length of

the sides of the image patch.

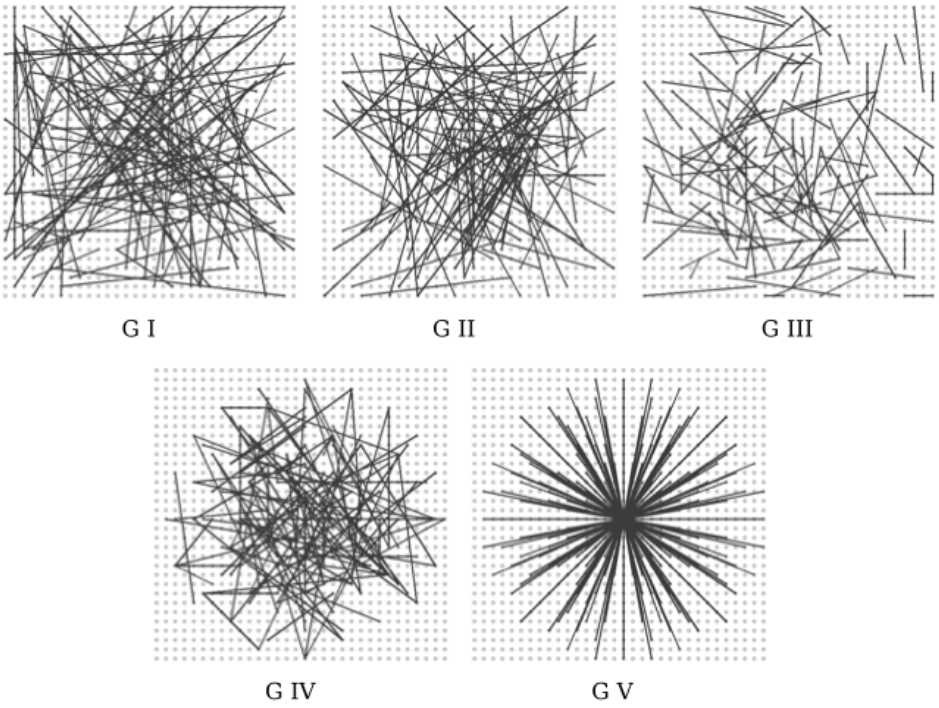


Figure 1.12: BRIEF approaches for the sampling of tests locations.

1.2.6 BRISK

Binary Robust Invariant Scalable Keypoints is a framework for efficient feature detection and description introduced by Leutenegger et al.[12] in 2011.

Detector

The key idea is to apply a FAST-like detector in a scale-space with the aim of achieving invariance to scale. Such scale-space is built as a pyramid of different layers, each one with a progressive half-sampling of the original image. Once the structure is built, FAST[8] is applied on each layer to obtain the score s of each point. Then non-maximal suppression is exploited to

[8]FAST 9-16 mask is employed in BRISK: the criterion for a candidate to be chosen as a keypoint is the presence of at least 9 adjacent samples (belonging to the 16-point circle) whose values are lower or greater than the reference one by a given threshold T.

retrieve keypoints in the scale-space. For an interest point to be selected its FAST score s should be a local maxima with respect to both the 8 neighboring samples belonging to the same scale layer and the remaining neighboring samples belonging to adjacent layers (see Figure 1.13). Finally, a quadratic function is fit to the keypoint to refine its location and its scale.

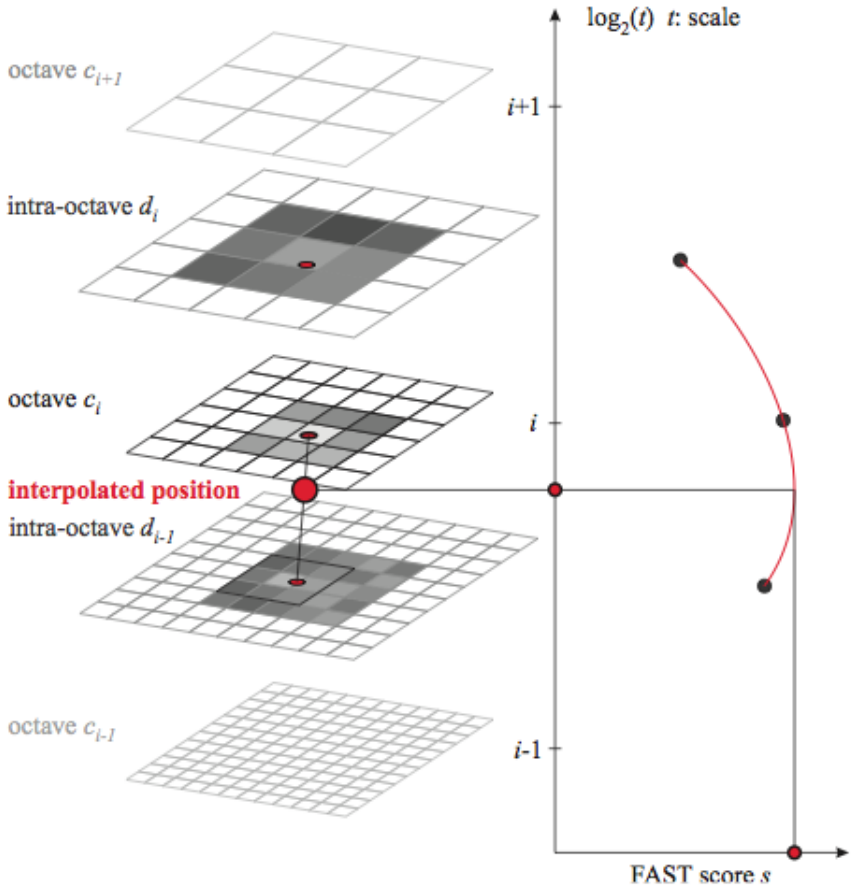


Figure 1.13: BRISK detector: a keypoint is identified at octave by analyzing the 8 neighboring saliency scores in as well as in the corresponding scores-patches in the immediately-neighboring layers. Then the sub-pixel location of the keypoint is refined with interpolation and the scale value is obtained fitting a 1-D quadratic function and finding its maximum.

Descriptor

Similarly to BRIEF the descriptor consists of a binary string representing results of simple brightness comparisons. BRISK descriptor exploits a particular pattern used for sampling the neighborhood of a keypoint. The pattern

defines N locations equally spaced on circles concentric with the keypoint (see Figure 1.14). In order to avoid aliasing and to give less emphasis to distant samples, a Gaussian smoothing with a standard deviation σ proportional to the distance from the center of the interest point is applied. Once the sampling locations and the smoothing have been defined, then the comparisons are exploited both to assign the orientation of the interest point and the descriptor vector. In particular, the comparisons between long-distance pairings are used to estimate the key orientation of the interest points while the ones between short-distance pairings contribute to the generation of the descriptor vector.

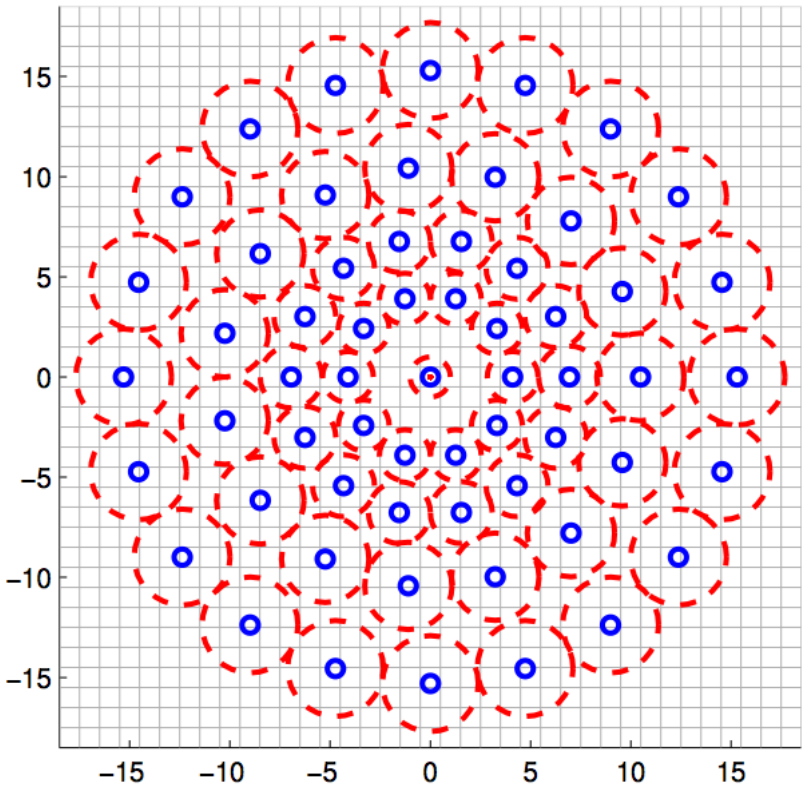


Figure 1.14: The BRISK sampling pattern with N = 60 points: the small circles denote the sampling locations while the bigger dashed ones are drawn at a radius corresponding to the standard deviation of the Gaussian kernel used to smooth the intensity values at the sampling points.

Chapter 2

Local features extraction on a sensor node

Sensing nodes have a particular architecture with peculiar features that should be taken into account in the development of a visual features extraction algorithm. Since they are designed with low power consumption as a main goal and they are commonly battery powered, the available computational power is limited. Therefore, the algorithm should be as efficient as possible to meet the requirements of such devices, with a possibly low computational complexity. Moreover, memory poses a strict constraint on the data structures to be employed: the storage of an image with double precision for each pixel, for example, is quite expensive in terms of on-board memory[1]. Considering the most common types of motes on the market, such as for example MicaZ, Mica2, TelosB, the memory requirements of our algorithm could not be met, since the amount of available RAM is in the order of hundreds of kilobytes. Then the Intel Imote2 node seems to be an adequate solution to meet both the memory and computational power requirements, featuring a 32 MB on-board SDRAM and a quite fast microprocessor with an up to 416 MHz clock frequency.

The traditional approaches for corner detection, starting from the one introduced by Harris & Stephens, are computational-intensive algorithms also

[1]Considering a 512x512 image and a 64-bit double precision data type to describe each pixel, the amount of memory required to store such data structure is equal to 2 megabytes.

for modern computing architectures. Then, more and more efficient algorithms inspired by the previous approaches have been introduced over the years. SIFT is a vastly and successfully employed method for this purpose, but still has a quite high computational cost for sensing nodes. SURF introduces several approximations to SIFT, such as the use of boxlets and integral images, trying to reduce the complexity of corner detection yet retaining good performance. Moreover, several different implementations have been released over the years together with the source code, and a comprehensive literature about algorithm development, implementation choices and performance evaluation have been published. This detailed background provides a solid foundation on which to develop a brand new implementation of the SURF algorithm for sensing nodes.

2.1 TinyOS

TinyOS is an open source, embedded operating system designed for low-power wireless devices, such as those used in sensor networks[13]. TinyOS started as a collaboration between the University of California, Berkeley, Intel Research and Crossbow Technology, and has since grown to be an international consortium, the TinyOS Alliance. It is written in the nesC programming language, a component-based and event-driven dialect of the C programming language, tailored around the requirements of WSNs. The event-based approach supports the concurrency intensive operations required by sensor networks with minimal hardware requirements, not allowing a single task to block the entire system and promoting a fair use of resources.

TinyOS provides not only the basic operating system but also a set of interfaces and components that can be easily integrated and exploited while developing applications for motes, that have to be written in nesC. Each Component is described by both its interface, containing commands and events, and its internal implementation. Commands are requests made to lower level components, while Event Handlers are invoked to deal with hardware events (at lower levels they are directly linked to hardware interrupts). Tasks are responsible for the execution of code and instructions and they can call commands, signal events and put other tasks in the Scheduler, that manages the execution of tasks according to a FIFO policy. A TinyOS appli-

cation is the union of a configuration file, describing the used components and the links between them, and a module, providing the implementation as a sequence of instructions or tasks that makes use of the declared components.

TOSSIM, a TinyOS motes simulator is provided to ease the development of TinyOS applications, that can be tested and debugged directly on a PC before being installed on the nodes.

2.2 From a high-level algorithm to a nesC implementation of SURF

As a starting point for the implementation of a brand new version of the SURF algorithm, an analysis of the open-source releases has taken place. In particular, two candidate versions of the algorithm have been investigated and compared in terms of implementative choices, performance, documentation:

- OpenSURF[7] is a dedicated library implementing the SURF algorithm. Originally written in C++, has also been ported in matlab and in Java;
- Pan-o-matic 0.9.4[19] is a tool for computing interest point correspondences between images released as a plugin for the panorama photo stitcher application Hugin. It contains an open source C++ implementation of SURF.

Some detailed analyses and comparison of the performance of different SURF implementations are well documented in literature. In particular, [10] shows that the Pan-o-matic version outperforms the OpenSURF counterpart in terms of repeatability, precision and recall, achieving results that are similar to the original SURF ones.

So Pan-o-matic was an obvious choice as a trace to follow in the development of the brand new implementation. For simplicity, an upright version of the SURF algorithm has been chosen, providing keypoints that are not invariant with respect to rotation.

From C++ to a plain C implementation

The first step towards a nesC version is a porting from the object-oriented

C++ implementation to a plain C one. References to external libraries have to be avoided, since the final version of the algorithm could not rely on such external resources.

Some tools guarantee an automatic compilation of C++ code into the C counterpart, but the resulting code is not user-friendly nor maintainable, so that kind of solutions does not meet the requirements. The object-oriented implementation has been manually analyzed, the instructions have been progressively inspected and translated to the target language. With some reverse engineering the C++ objects are mapped to simpler C data types and the relative functions are modified to accept such data structures as input parameters. The final result is a purely sequential code, easier to be translated to the nesC programming language.

Memory is also an important aspect to deal with; its usage and the allocation method are to be kept into account while developing the application. The Pan-o-matic SURF implementation makes use of dynamic memory allocation for the instantiation of some big data structures needed by the algorithm. In nesC every data structure employed at runtime has to be declared before executing any command and, although some primitives for dynamic memory allocation (e.g. malloc(), dealloc(), etc.) are implemented, they are not fully supported nor safe.

For these reasons, as regards the C prototype, dynamic memory allocation is to be avoided and some big data structures are instead declared as global variables at the beginning of the code. The original implementation receives the image path as a input parameter; in nesC it is not possible to store that data in a file system. Then the C prototype includes a header file that contains a test image data structure in the form of a bidimensional array of integers describing the image pixel map.

Towards a nesC implementation

The nesC and C programming languages are strictly related: their syntaxes are nearly identical, as the semantic of their instructions. Moreover, the plain C prototype offers an implementation that is in some way much closer to the target with respect to the original one.

As a first step for this second porting, every variable declaration has to be moved at global scope. Doing this, care must be taken in managing

duplicated variable names: as an example, in C a variable can be declared in the scope of a loop and is accessible only for the loop instructions. If another variable with the same name but possibly another type is declared after the loop, no problems arise. Since in nesC variables are at global scope, the same scenario would represents a problem, and some variable renaming could be required.

The code needs also to be reorganized in a set of tasks that represents the flow of the algorithm. Such tasks are then sequentially called after the boot and the initialization of the system.

The nesC code is then built, tested and debugged with TOSSIM. The simulator implements the main features of a mote and it is really useful for the testing process, providing a way more user-friendly interface than the motes themselves. In particular, the debugging process is eased, since some debug instructions can be trivially nested inside to code to observe the value of variables or the general behavior of the application. Then, with TOSSIM, a first debug session takes place directly on a PC without the need of installing the application on a mote. The identified problems in the porting of the algorithm can be easily solved within this stage.

Once the code is working in the simulation environment, the following steps consist in the installation over a mote and in a second debug session to solve any potential problem arising at this stage. TOSSIM is a useful tool but it by no means offers a complete coverage in terms of simulation of motes features. For example, regarding memory, TOSSIM does not pose tight limits about the amount of available RAM, while Intel Imote2 amount of primary memory is not sufficient to satisfy the needs of the algorithm. Then, the largest data structures are to be stored in the SDRAM thanks to some ad-hoc primitives. Unfortunately, a given amount of memory can be reserved in that sector but it is not possible to directly store a variable by assignment. The usage of multidimensional data structures is then discouraged, since it is not possible to define that data types in the SDRAM sector nor to assign the content of such structures.

To solve this problem, the algorithm is modified to operate on monodimensional array instead of using multidimensional ones and the data structures are modified accordingly. For doing this, multidimensional arrays are sequentially scanned as a raster and stored in a simple arrays. Indexes are

then modified according to a particular transformation to guarantee that the right field is accessed.

The resulting code is finally compiled and installed on the mote via a debugging board and using serial communication. Unlike in TOSSIM, debug is a difficult task and the values of the variables cannot be directly inspected. In fact, it is not possible to print information on the standard output or on some log files. The data has to be transmitted in form of messages over a serial communication channel to the PC and then it can be suitably received, managed and stored. For doing this, a simple java application has been implemented to manage the messages received via the serial communication port and to print the results in some log files.

Chapter 3

Implementation details

3.1 Storing variables in the SDRAM

Some large variables should be necessarily stored in the SDRAM section. For doing that, a header file called "sdram.h" is introduced and the employed data structures are to be declared according to a particular syntax. In particular some flags, interpreted by the compiler, discern the variables that are to be stored in a particular memory section. Then, an address to each variable is defined to be later recalled from the main file, where a pointer to that SDRAM address is assigned to a local variable's one.

```
1  #ifndef _SDRAM_H_
2  #define _SDRAM_H_
3
4
5  #define VGA_SIZE_RGB (25600)     //im_r*im_c
6  #define II_SIZE_RGB (25921)      //(im_r+1)*(im_c+1)
7  #define ASH_SIZE (128000)        //_maxscales*im_r*im_c
8  #define KP_SIZE (2620)           //max_kp*131
9
10 uint32_t base_f[VGA_SIZE_RGB] __attribute__((section(".sdram")));
11 double ima_f[VGA_SIZE_RGB] __attribute__((section(".sdram")));
12 double ii_f[II_SIZE_RGB] __attribute__((section(".sdram")));
13 double aSH_f[ASH_SIZE] __attribute__((section(".sdram")));
14 double kp_f[KP_SIZE] __attribute__((section(".sdram")));
15
```

```
16 #define BASE_FRAME_ADDRESS base_f
17 #define IMA_FRAME_ADDRESS ima_f
18 #define II_FRAME_ADDRESS ii_f
19 #define ASH_FRAME_ADDRESS aSH_f
20 #define KEYPOINTS_FRAME_ADDRESS kp_f
21
22 #endif //_SDRAM_H_
```

Listing 3.1: an excerpt of "sdram.h"

```
1  #include "sdram.h"
2
3  implementation {
4
5   uint8_t *Ima = (void*)BASE_FRAME_ADDRESS;
6   double *B = (void*)IMA_FRAME_ADDRESS;
7   double *_ii = (void*)II_FRAME_ADDRESS;
8   double *aSH=(void*)ASH_FRAME_ADDRESS;
9   double *keypoints=(void*)KEYPOINTS_FRAME_ADDRESS;
10
11 ...
```

Listing 3.2: an excerpt of "WSN_SURFM.nC"

3.2 Test image initialization

Since the declaration of a variable larger then the mote's primary RAM capacity is not allowed, the test image should be stored in the SDRAM of the node with a pixel by pixel assignment to the relative array cell at the beginning of the execution. Unfortunately, this approach only allows for small test images to be used, since a sufficiently large number of cell by cell assignment via the serial communication channel prevents the node from being programmed with the executable files.

```
1 Ima[0]= 161;
2 Ima[1]= 161;
3 Ima[2]= 158;
4 Ima[3]= 159;
5 Ima[4]= 163;
```

```
Ima[5]= 159;
Ima[6]= 156;
Ima[7]= 156;
Ima[8]= 150;
Ima[9]= 158;

...

Ima[25592]= 99;
Ima[25593]= 101;
Ima[25594]= 124;
Ima[25595]= 153;
Ima[25596]= 187;
Ima[25597]= 217;
Ima[25598]= 240;
Ima[25599]= 248;
```

Listing 3.3: an excerpt of "WSN_SURFM.nC"

3.3 Box filtering

The box filtering of an image requires only a few simple instructions. For a given box-like shape, only the values at the corners of such rectangle are retrieved from the integral image (see variables a, b, c, d in the code excerpt). Then, the final result of the filtering is provided by simply summing and subtracting the contributions of the boxes.

```
/* Calculate aDxy, the filter has the following shape:
 *
 *    + contributions are to be summed;
 *    - contributions are to be subtracted.
 *
 *          |+++++|   |-----|
 *          |+ a +|   |- c -|
 *          |+++++|   |-----|
 *
 *          |-----|   |+++++|
 *          |- d -|   |+ b +|
 *          |-----|   |+++++|
 *
```

```
14  */
15
16
17  a = _ii[(im_c+1)*(_y_plus_lxy_d2+1)+aXPS+_lxy_d2+1] + ...
18      _ii[(im_c+1)*_y+aXPS] - ...
19      _ii[(im_c+1)*(_y_plus_lxy_d2+1)+aXPS] - ...
20      _ii[(im_c+1)*_y+aXPS+_lxy_d2+1];
21
22  b = _ii[(im_c+1)*(_y+1)+aXPS+1] + ...
23      _ii[(im_c+1)*(_y_minus_lxy_d2)+aXPS - _lxy_d2] - ...
24      _ii[(im_c+1)*(_y+1)+aXPS - _lxy_d2] - ...
25      _ii[(im_c+1)*(_y_minus_lxy_d2)+aXPS+1];
26
27  c = _ii[(im_c+1)*(_y+1)+aXPS + _lxy_d2+1] + ...
28      _ii[(im_c+1)*(_y_minus_lxy_d2)+aXPS] - ...
29      _ii[(im_c+1)*(_y+1)+aXPS] - ...
30      _ii[(im_c+1)*(_y_minus_lxy_d2)+aXPS + _lxy_d2+1];
31
32  d = _ii[(im_c+1)*(_y_plus_lxy_d2+1)+aXPS+1] + ...
33      _ii[(im_c+1)*_y+aXPS - _lxy_d2] - ...
34      _ii[(im_c+1)*(_y_plus_lxy_d2+1)+aXPS - _lxy_d2] - ...
35      _ii[(im_c+1)*_y+aXPS+1];
36
37  aDxy = a+b-c-d;
```

Listing 3.4: an excerpt of "WSN_SURFM.nC"

3.4 SURF-128

An enhanced SURF descriptor, made up of 128 entries, has been proposed by Bay in [2] and implemented in WSN-SURF. For each keypoint, the common SURF-64 descriptor is made up of the 4 components

$$\begin{cases} \sum d_x \\ \sum d_y \\ \sum |d_x| \\ \sum |d_y| \end{cases}$$

sampled on a 4x4 grid around the interest point.

In the enhanced SURF-128 descriptor, each of the 16 samples belonging to the 4x4 grid is made up of the following eight components (replacing the previous four ones):

$$\begin{cases} \sum d_x & for \quad d_y < 0 \\ \sum |d_x| & \dots \\ \sum d_x & for \quad d_y \geq 0 \\ \sum |d_x| & \dots \\ \sum d_y & for \quad d_x < 0 \\ \sum |d_y| & \dots \\ \sum d_y & for \quad d_x \geq 0 \\ \sum |d_y| & \dots \end{cases}$$

In the code of WSN-SURF, setting the flag "_extended" to 1 the SURF-128 descriptor is computed, otherwise the SURF-64 is chosen.

```
if (_extended) {
 aBin_ = (aWavYR_ <= 0) ? 0 : 1;
 _cmp[aBin1V_][aBin1U_][aBin_] += aWB1V_ * aWB1U_ * aWavXR_;
 _cmp[aBin2V_][aBin1U_][aBin_] += aWB2V_ * aWB1U_ * aWavXR_;
 _cmp[aBin1V_][aBin2U_][aBin_] += aWB1V_ * aWB2U_ * aWavXR_;
 _cmp[aBin2V_][aBin2U_][aBin_] += aWB2V_ * aWB2U_ * aWavXR_;

 aBin_ += 2;
 aVal_ = fabs(aWavXR_);
 _cmp[aBin1V_][aBin1U_][aBin_] += aWB1V_ * aWB1U_ * aVal_;
 _cmp[aBin2V_][aBin1U_][aBin_] += aWB2V_ * aWB1U_ * aVal_;
 _cmp[aBin1V_][aBin2U_][aBin_] += aWB1V_ * aWB2U_ * aVal_;
 _cmp[aBin2V_][aBin2U_][aBin_] += aWB2V_ * aWB2U_ * aVal_;

 aBin_ = (aWavXR_ <= 0) ? 4 : 5;
 _cmp[aBin1V_][aBin1U_][aBin_] += aWB1V_ * aWB1U_ * aWavYR_;
 _cmp[aBin2V_][aBin1U_][aBin_] += aWB2V_ * aWB1U_ * aWavYR_;
 _cmp[aBin1V_][aBin2U_][aBin_] += aWB1V_ * aWB2U_ * aWavYR_;
 _cmp[aBin2V_][aBin2U_][aBin_] += aWB2V_ * aWB2U_ * aWavYR_;

 aBin_ += 2;
 aVal_ = fabs(aWavYR_);
```

```
23  _cmp[aBin1V_][aBin1U_][aBin_] += aWB1V_ * aWB1U_ * aVal_;
24  _cmp[aBin2V_][aBin1U_][aBin_] += aWB2V_ * aWB1U_ * aVal_;
25  _cmp[aBin1V_][aBin2U_][aBin_] += aWB1V_ * aWB2U_ * aVal_;
26  _cmp[aBin2V_][aBin2U_][aBin_] += aWB2V_ * aWB2U_ * aVal_;
27
28 }
29 else {
30
31  aBin_ = (aWavXR_ <= 0) ? 0 : 1;
32  _cmp[aBin1V_][aBin1U_][aBin_] += aWB1V_ * aWB1U_ * aWavXR_;
33  _cmp[aBin2V_][aBin1U_][aBin_] += aWB2V_ * aWB1U_ * aWavXR_;
34  _cmp[aBin1V_][aBin2U_][aBin_] += aWB1V_ * aWB2U_ * aWavXR_;
35  _cmp[aBin2V_][aBin2U_][aBin_] += aWB2V_ * aWB2U_ * aWavXR_;
36
37  aBin_ = (aWavYR_ <= 0) ? 2 : 3;
38  _cmp[aBin1V_][aBin1U_][aBin_] += aWB1V_ * aWB1U_ * aWavYR_;
39  _cmp[aBin2V_][aBin1U_][aBin_] += aWB2V_ * aWB1U_ * aWavYR_;
40  _cmp[aBin1V_][aBin2U_][aBin_] += aWB1V_ * aWB2U_ * aWavYR_;
41  _cmp[aBin2V_][aBin2U_][aBin_] += aWB2V_ * aWB2U_ * aWavYR_;
42 }
```

Listing 3.5: an excerpt of "WSN_SURFM.nC"

3.5 Sending features via the serial communication channel

The features extracted with the algorithm can be sent to another node or to a central unit exploiting the serial communication channel in the form of messages. Since the communication protocol has several limitations in terms of speed and message size, the features should be adapted and modified to meet the requirements. In particular, the components of the keypoint descriptor vector are quantized to 8 bits, and a whole vector does not fit the maximum dimension of a serial message payload[1]. The vector is then split up in several parts that are sent separately along with an identification field

[1] in TinyOS the maximum number of bytes for the payload amounts to 255. The number of components that can be transmitted depends on the size of the data types describing it. In this example, 31 entries consisting of an 8 bit value each accounts for 31 * 8 = 248 bytes of data to be transmitted, fitting the maximum payload size of the serial messages.

that allows for an easy reconstruction of the features at the receiver.

```
test_serial_msg_t* rcm = (test_serial_msg_t*)call Packet.↩
    getPayload(&packet, sizeof(test_serial_msg_t));
if (rcm == NULL) {
 return;
}
if (call Packet.maxPayloadLength() < sizeof(test_serial_msg_t)) {
 return;
}
rcm->id = sent;
rcm->seq = seq;
rcm->aX = keypoints[sent*131]*100.0;
rcm->aY = keypoints[sent*131+1]*100.0;
rcm->aS = keypoints[sent*131+2]*1000.0;

for (jjj=0; jjj<15; ++jjj){
 if (15*seq+jjj<iLen_){

  rcm->cmp[jjj] = keypoints[sent*131+3+15*seq+jjj]*255.0;
 }
 else rcm->cmp[jjj]=0.0;
}

if (call AMSend.send(AM_BROADCAST_ADDR, &packet, sizeof(↩
    test_serial_msg_t)) == SUCCESS) {

 locked = TRUE;
}
```

Listing 3.6: an excerpt of "WSN_SURFM.nC"

```
#ifndef TEST_SERIAL_H
#define TEST_SERIAL_H

typedef nx_struct test_serial_msg {
 nx_uint16_t id;
 nx_uint8_t seq;
 nx_uint16_t aX;
 nx_uint16_t aY;
 nx_uint16_t aS;
```

```
10   nx_int8_t cmp[15];
11 } test_serial_msg_t;
12
13 enum {
14   AM_TEST_SERIAL_MSG = 0x89,
15 };
16
17 #endif
```

Listing 3.7: an excerpt of "TestSerial.h"

Chapter 4

Performance evaluation

The performance of both the detector and the descriptor are evaluated according to [18, 17], along with an analysis of the required processing time and of the resource usage with respect to some key parameters. An ad-hoc dataset is used for this purpose: six sets of images with increasing changes in some imaging conditions are provided along with the ground-truth correspondences (see Figure 4.1). For the tests, the images have been downsampled by a factor of four to meet the requirements of the WSN-SURF application.

Figure 4.1: the dataset contains six sequences consisting of six images each with progressively worsened imaging conditions such as viewpoint (Graffiti and Wall), blur (Bikes and Trees), JPEG compression ratio (UBC), illumination (Leuven).

4.1 Detector

The proposed metric to evaluate the performance of the detector is an extended version of the repeatability measure. Although the framework has been thought to compare affine covariant regions, represented by a bounding ellipse, it can also adapt to SURF features, represented by a circle of radius proportional to the detected scale.

In particular, to compute the repeatability score for a couple of images, each keypoint belonging to the first one (and its corresponding region) is projected on the second one according to the provided ground-truth transformation, then if the projected ellipse overlaps with another region by a given amount, a correspondence is found. The percentage of such matches over the number of interest points represents the score.

In details, the repeatability score is defined as

$$R_{score} = \frac{C(A,B)}{min(n_a, n_b)}$$

where $C(A,B)$ is the number of correspondences between keypoints of image A and B and n_i is the number of interest points detected in image I. To check the correspondences between couples of interest points an overlap error is defined as

$$\epsilon_S = 1 - \frac{\mu_a \cap (H^T \mu_b H)}{\mu_a \cup (H^T \mu_b H)}$$

where μ_a and μ_b represents the two elliptical regions of the keypoints a and b (see Figure 4.2). Typically, values below a threshold of 0.4 are assumed to represent a correspondence (see Figure 4.3).

In Figures 4.4, 4.5 and 4.6 the repeatability values for each dataset are presented and compared with some other features detectors:

- openSURF[7];
- FAST 9-16[20, 21];
- SIFT[14];
- MSER[15];

- Harris-affine[16];
- Hessian-affine[16].

The implementation of the SURF algorithm on a WSN achieves quite good results in most cases, while more expensive and affine invariant approaches definitely seems to have an edge on it for some strong viewpoint transformations, while the more computationally efficient FAST detector seems to achieve comparable results for several datasets[1].

4.2 Descriptor

As for the detector, also for the descriptor an ad-hoc metric is introduced: the matching score represents once again the ratio of correct matches over the total number of regions. Differently from the previous case, however, the correspondences between the keypoints are established relying on the whole set of values of each interest point (including the descriptor). In particular, the process includes two steps:

- A ground truth for correct matches is created starting from the detector evaluation, deeming correct matches the ones having an overlap error ϵ_S lower than 0.4;
- For each of the above correspondences, a match is assigned if the two keypoints are the nearest neighbors in the descriptor space[2], i.e. the two descriptor vectors are the most similar ones. Then, the percentage of such matches compared to the number of regions represents the matching score.

For evaluation purposes, the interest points detected with the WSN-SURF detector are described with several different algorithms:

- steerable filters[9];
- SIFT[14];

[1] unlike the other algorithms, FAST is not scale-invariant.

[2] the nearest neighbor is identified according to the euclidean distance between the descriptor vectors.

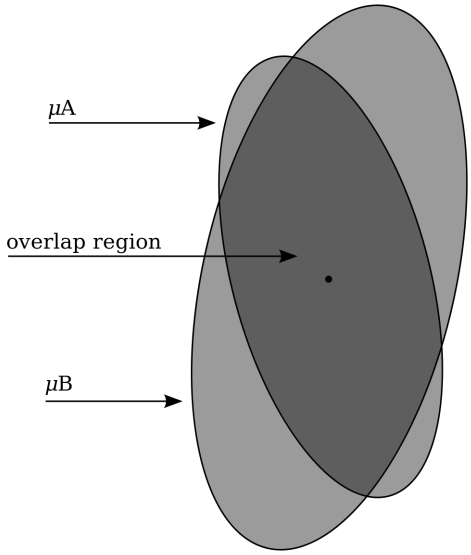


Figure 4.2: the overlap error of two elliptical regions μ_a and μ_b .

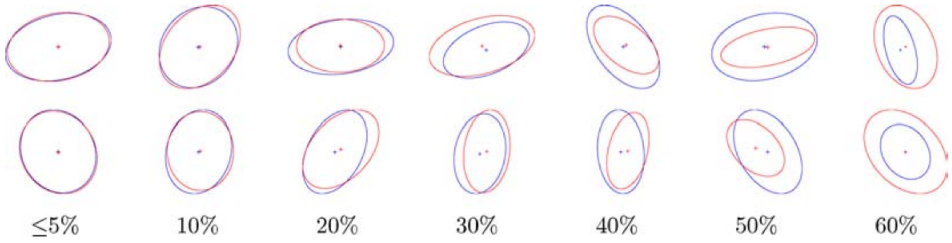


Figure 4.3: examples of regions projected on the corresponding ellipse with the ground truth transformation along with the relative overlap error.

- GLOH[17];
- differential invariants[8];
- complex filters[22];
- WSN-SURF.

In Figures 4.7, 4.8 and 4.9 the matching scores for each set of images is reported. The WSN-SURF descriptor performs very well in every scenario, outperforming the other descriptors in several cases.

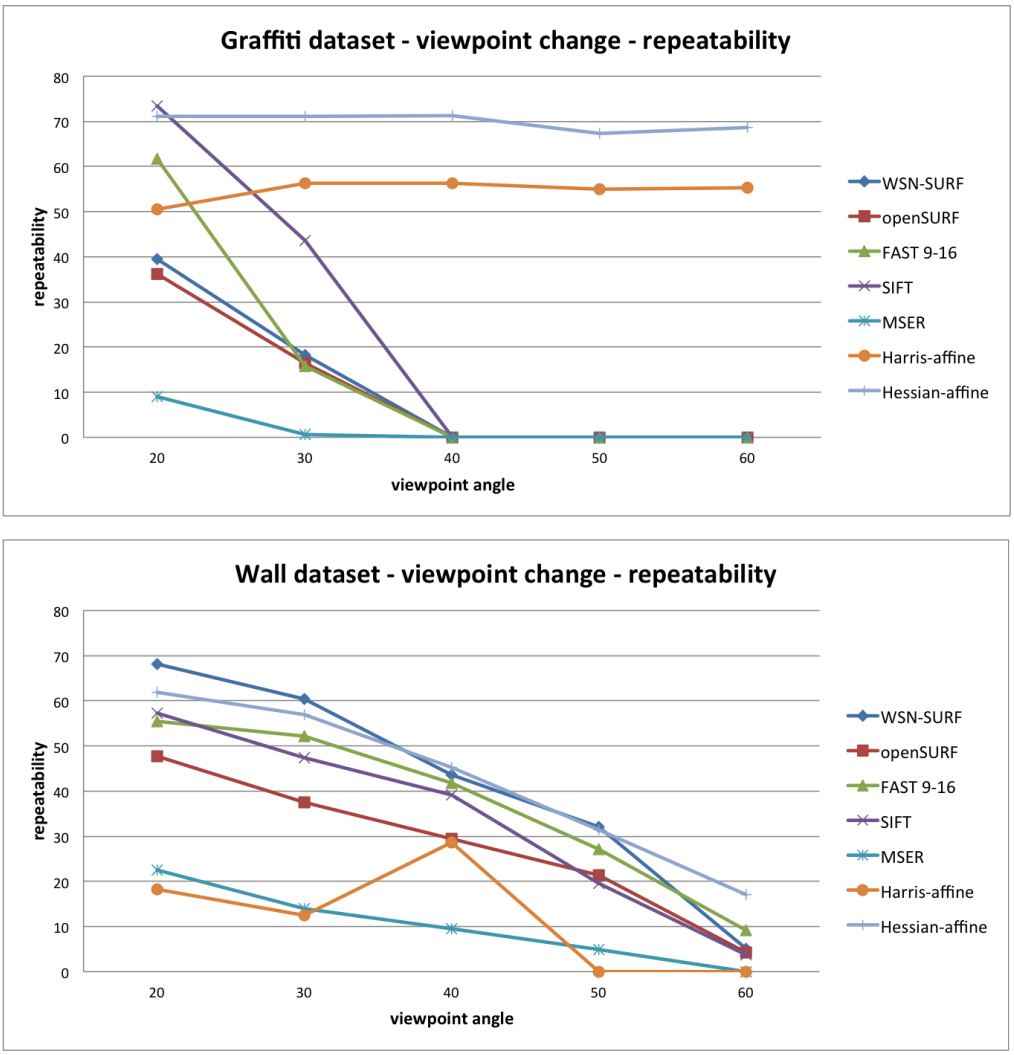
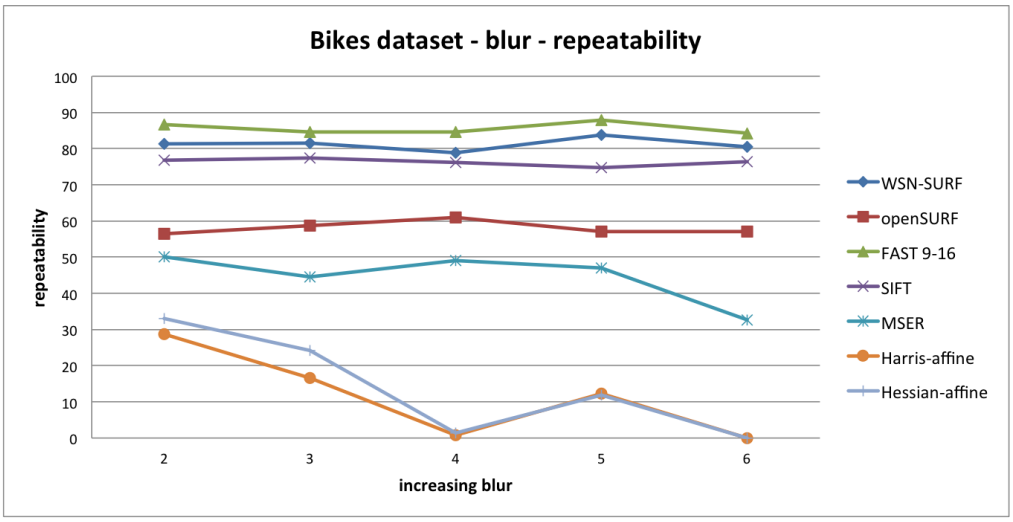


Figure 4.4: repeatability figures for Graffiti and Wall datasets, representative of changes in viewpoint.



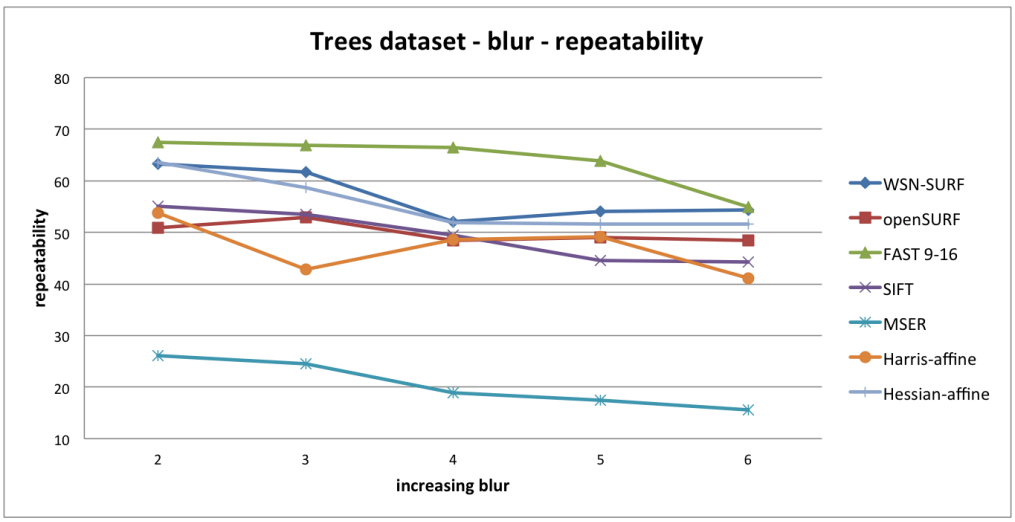


Figure 4.5: repeatability figures for Bikes and Trees datasets, representative of increasing blurring.

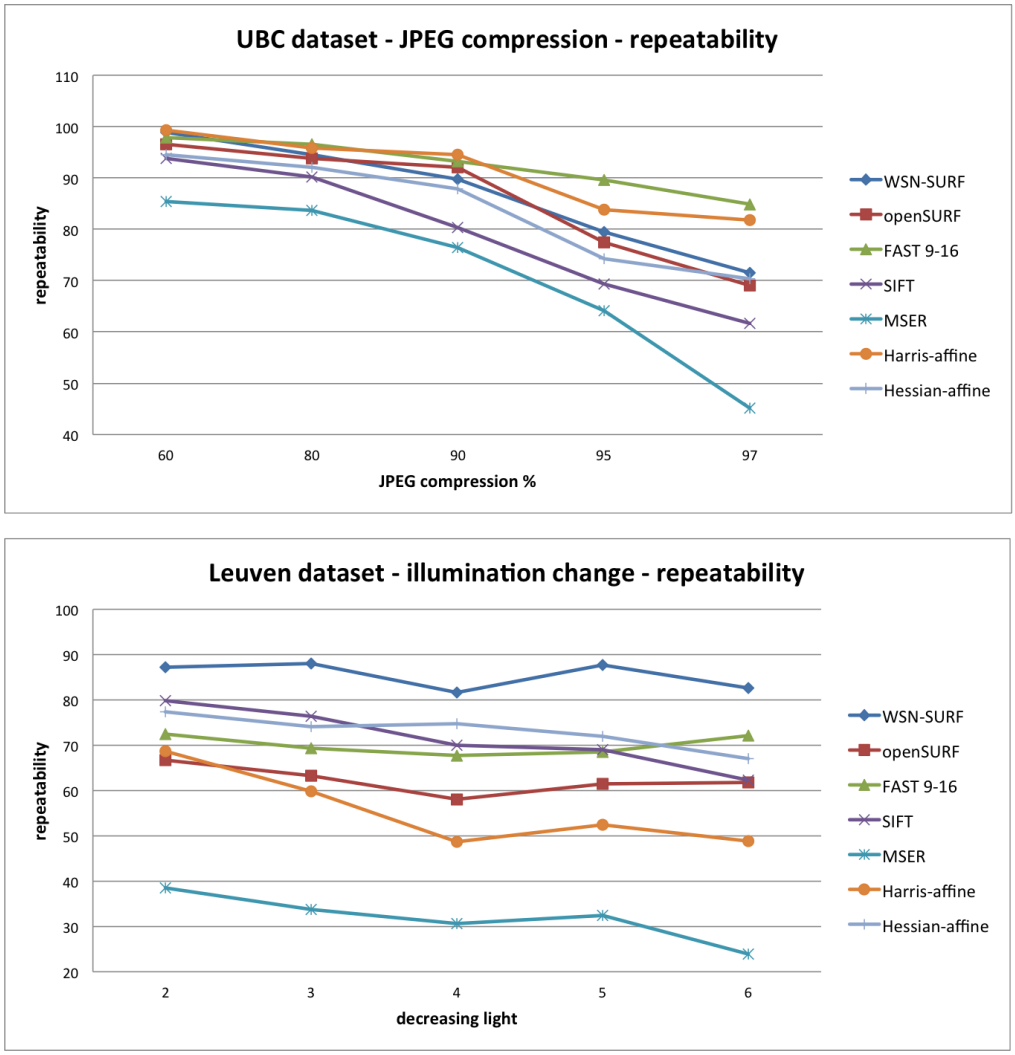


Figure 4.6: repeatability figures for UBC and Leuven datasets, representative of JPEG compression and illumination changes, respectively.

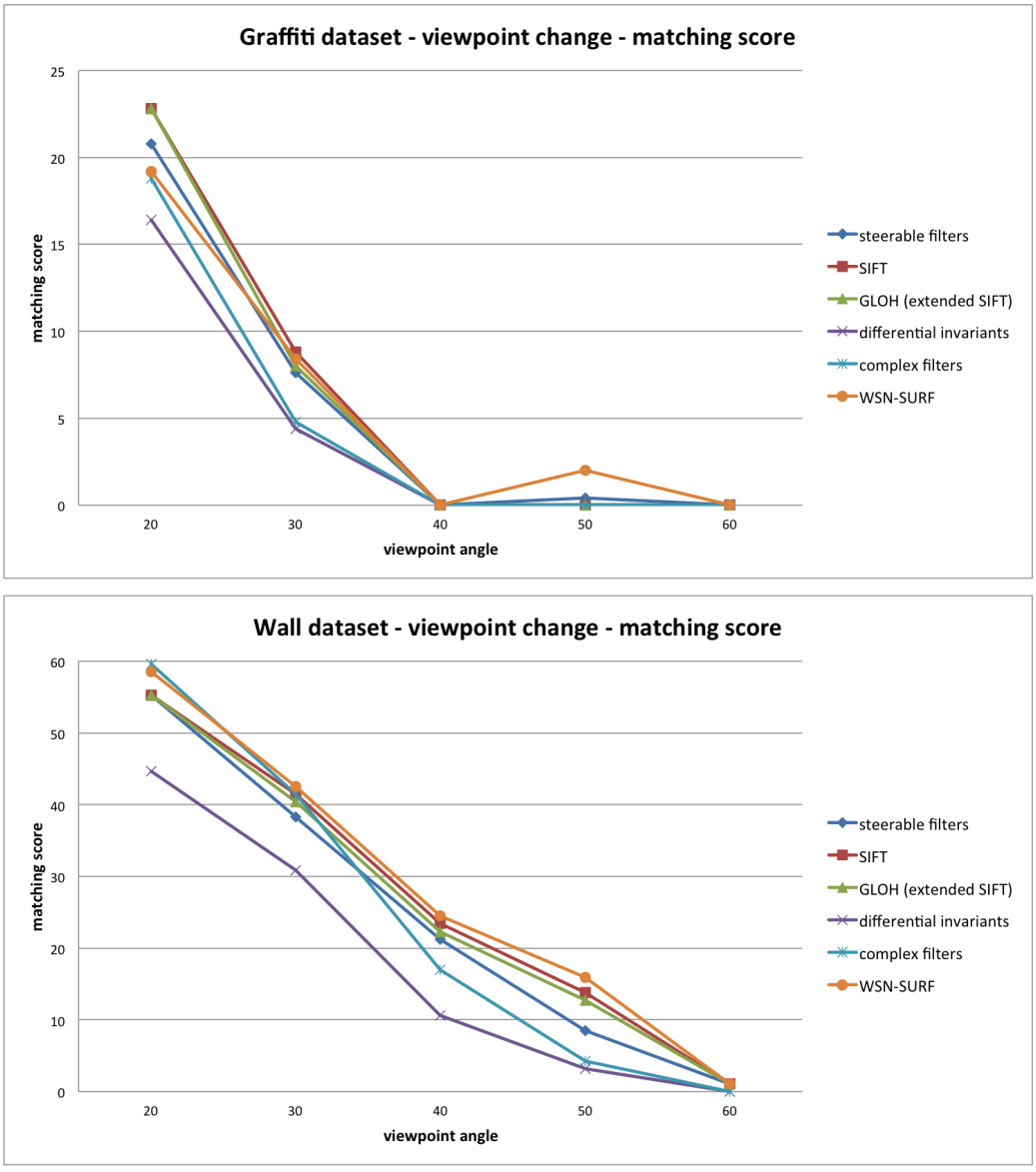


Figure 4.7: matching score figures for Graffiti and Wall datasets, representative of changes in viewpoint.

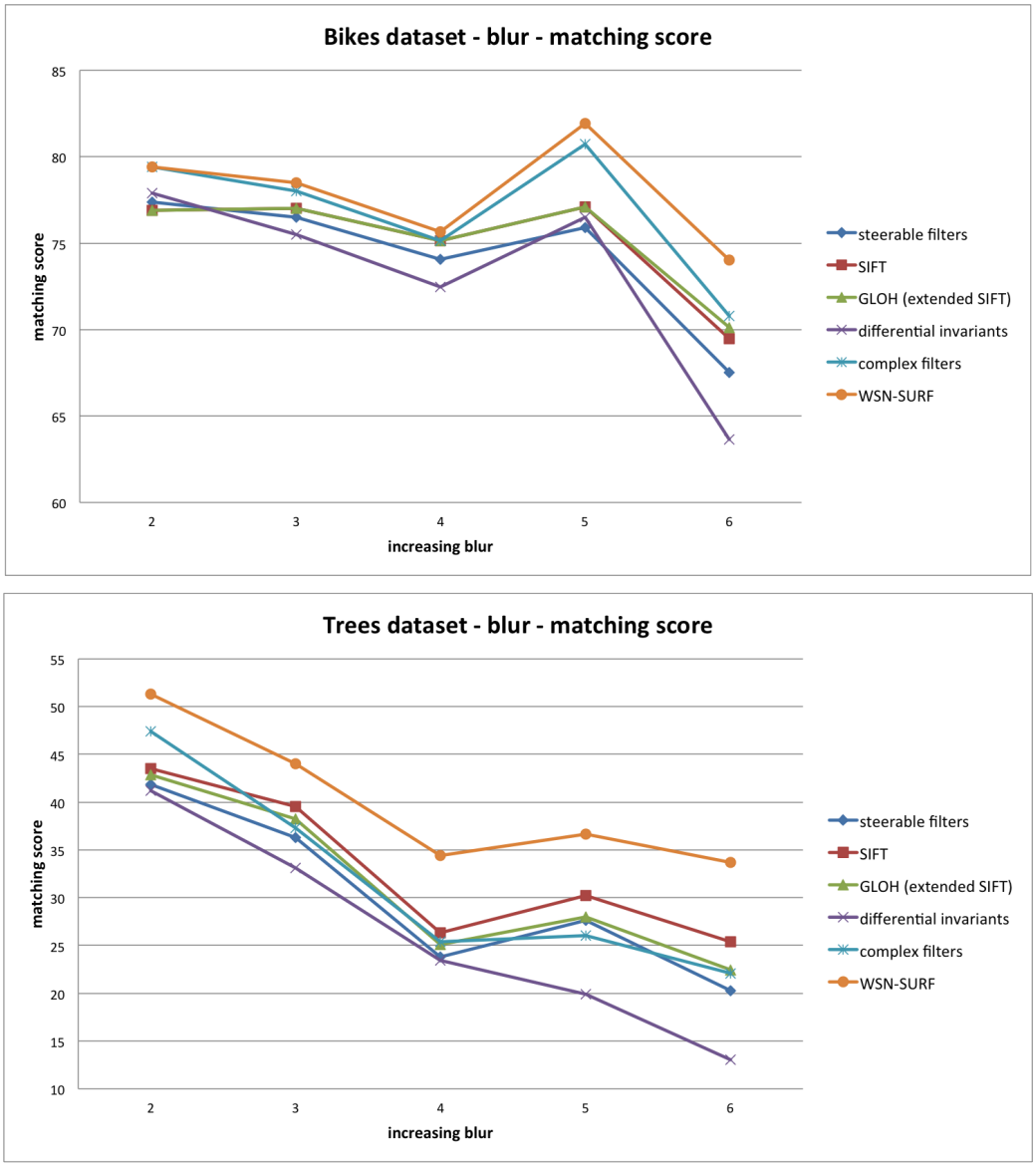


Figure 4.8: matching score figures for Bikes and Trees datasets, representative of increasing blurring.

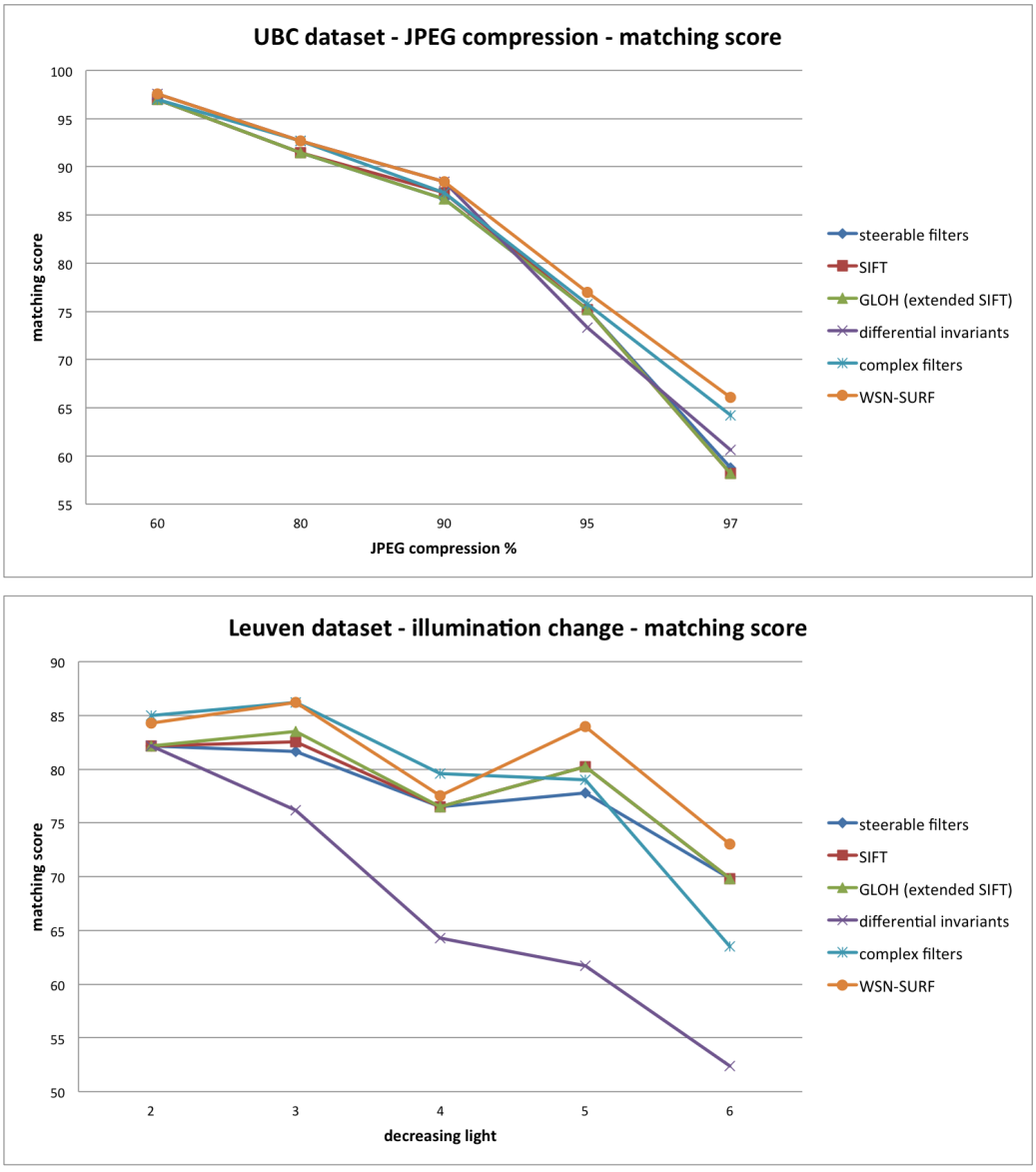


Figure 4.9: matching score figures for UBC and Leuven datasets, representative of JPEG compression and illumination changes, respectively.

4.3 Processing time

The processing time for the algorithm has been measured resorting to TinyOS Timer interface. In particular, for each image the processing time is estimated for the following operations:

- image initialization: the time needed to store the image in the SDRAM and to convert each pixel value from int format (8 bit, 0-255 range) to double precision format (64 bit, 0-1 range)[3];
- integral image: the time needed to build an integral image (see 1.2.3 for further details) starting from the previous step;
- scale-space building: the time needed for the computation of the scale-space layers for the first (and the most computational-intensive one) octave. Each value of such "cube" represents an approximated Hessian determinant for the particular location and scale;
- detection: the time needed to retrieve the interest points location and scale with non-maximal suppression. It also includes the interpolation process to refine the values of these three parameters;
- descriptor: the time needed to compute the descriptor vector for every keypoint.

Please note that the whole detector chain includes both the scale-space building and the detection tasks. Here the two operations are kept separated for a more detailed analysis.

The processing time values are reported in Tables 4.1, 4.2, 4.3.

Some notes:

- the computation of the descriptor vector is by far the most computationally intensive task (see Figure 4.10) and its execution time is strongly dependent on the number of interest points to be described (see Figure 4.12);

[3] the conversion is needed for the following steps of the algorithm.

image id	bikes 1	bikes 2	bikes 3	graf 1	graf 2	graf 3
image initialization	2.7	2.6	2.8	3.0	3.0	3.2
integral image	2.0	1.8	2	2.0	1.8	1.8
scale-space building	77.8	77.9	77.6	77.8	77.9	77.4
detector	14.4	13.8	13.4	15.3	15.8	14
descriptor	364.3	312.4	250.3	750.6	788.8	586.4
number of keypoints	87	76	67	180	200	143

Table 4.1: processing time values (in seconds) for the Bikes and Graffiti image sequences.

image id	leuv 1	leuv 2	leuv 3	trees 1	trees 2	trees 3
initialization	2.9	2.7	2.5	3.1	3.1	3.1
integral image	2.1	1.9	2	1.9	2.0	2.0
scale-space building	77.3	77.8	77.7	77.4	77.4	77.4
detector	13.9	13.3	12.8	13.8	13.6	13.9
descriptor	483	314.9	108.4	462.8	413.2	552.9
number of keypoints	116	80	29	121	109	135

Table 4.2: processing time values (in seconds) for the Leuven and Trees image sequences.

image id	ubc 1	ubc 2	ubc 3	wall 1	wall 2	wall 3
initialization	2.8	3.0	2.9	3.2	3	2.9
integral image	1.9	2.0	2.0	1.7	2.1	1.9
scale-space building	77.7	77.6	77.5	77.5	77.4	77.7
detector	13.8	13.7	13.9	12.7	12.7	13
descriptor	482.1	482.6	560.5	105.5	120	159
number of keypoints	122	121	131	20	23	37

Table 4.3: processing time values (in seconds) for the UBC and Wall image sequences.

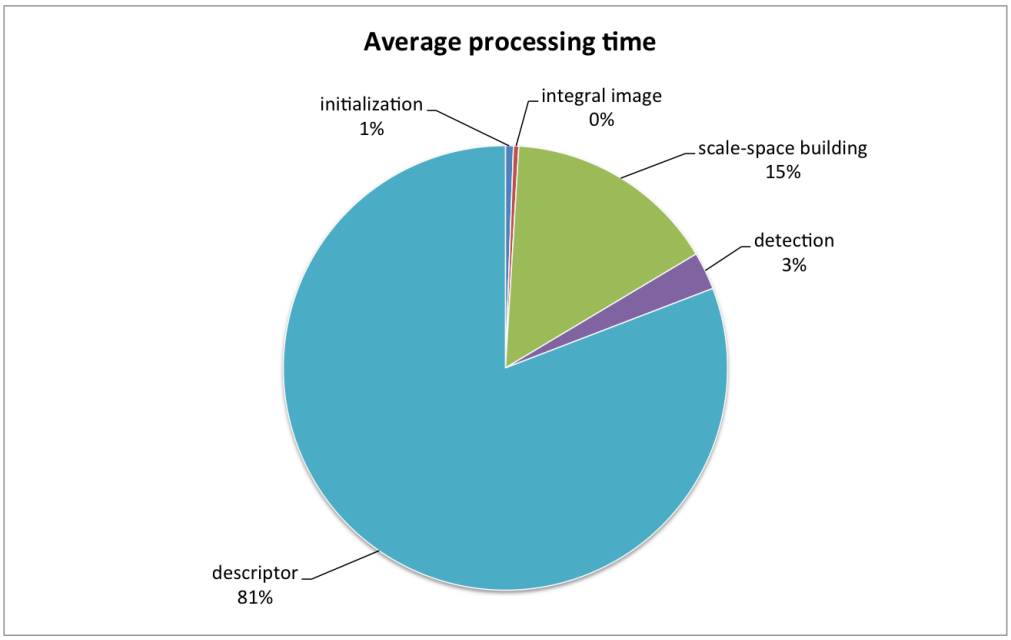


Figure 4.10: average processing time percentage for each task of the algorithm. The descriptor accounts for a rough 81% of the total execution time, while the whole detector chain, including scale-space building and detection, results in a rough 18%.

- the time required for initialization, integral image building, scale-space building is constant with respect to the number of keypoints, as conceivable;
- the time needed for the detection stage is slightly dependent on the number of identified interest points (see Figure 4.11);
- the single instruction of the descriptor that computes an exponential function[4] accounts for a rough 15% of the total processing time.

4.4 Memory

Regarding the memory space required by the program, it is strongly affected by the size of the image to be analyzed. In particular, calling n the number

[4]such a function is used to Gaussian weight the gradient responses according to their distance from the reference point.

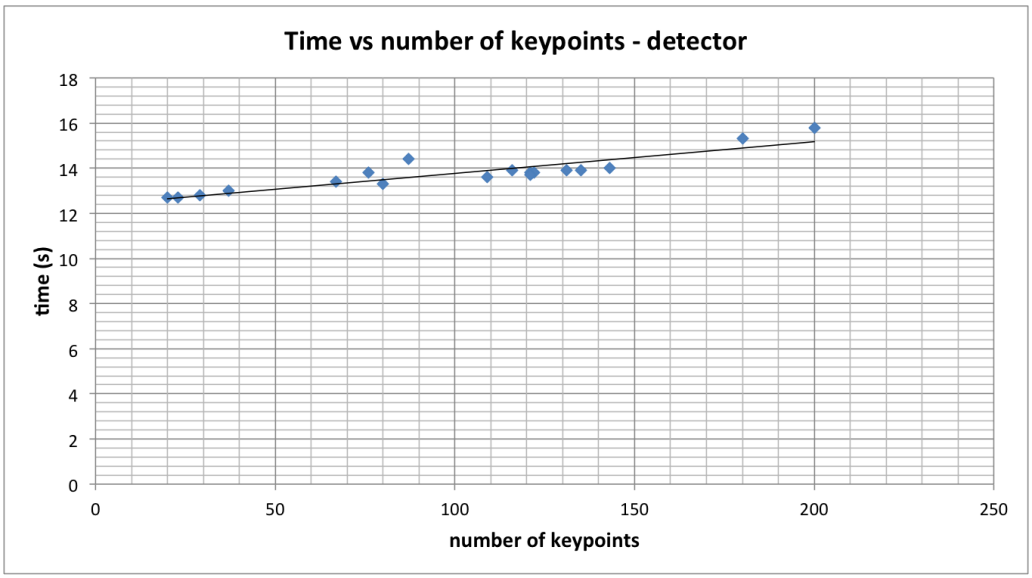


Figure 4.11: the processing time (in seconds) for the detection task versus the number of identified keypoints. The dots represents the experimental values, while the straight line the estimated trend.

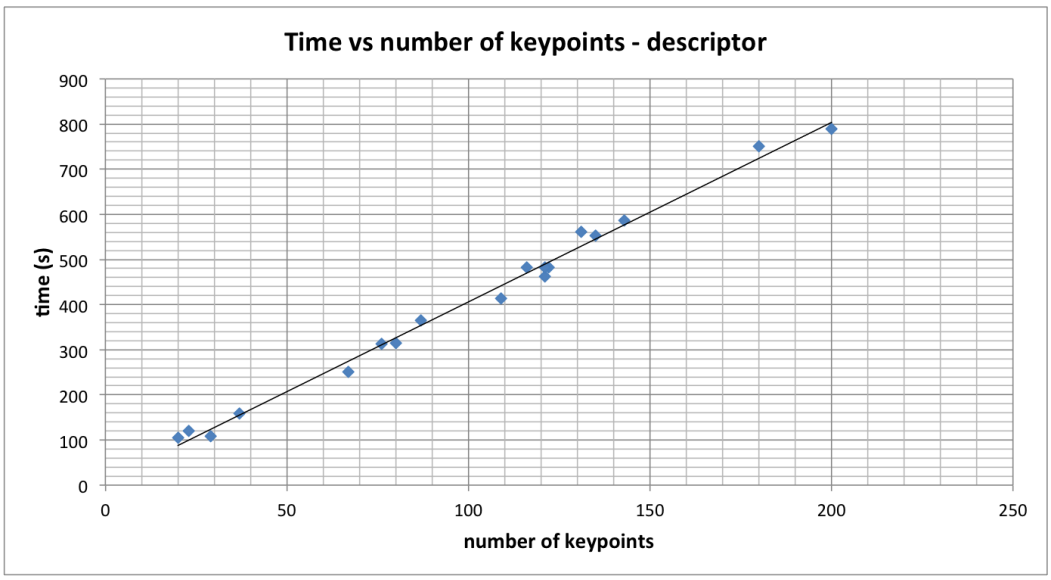


Figure 4.12: the processing time (in seconds) for the descriptor task versus the number of identified keypoints. The dots represents the experimental values, while the straight line the estimated trend.

of pixels of the image, the algorithm requires a rough amount of $n \times 9 \times 64 = 576 \times n$ bits for the execution.

For example, considering a grayscale 0.3 megapixel[5] image, the memory usage amounts to

$$M_u \approx 576 \times 0.3 = 172.8 \; megabits = 21.6 \; megabytes$$

[5]e.g. a 640×480 grayscale image.

Chapter 5

Conclusion and future work

In a scenario where a pervasive network of things is empowered with improved sensing abilities, the efficiency of the algorithms in terms of power consumption, processing time, network and memory usage is a key. The performance evaluation of such implementation shows that the algorithm, despite being efficient on a modern PC, shows several weakness on a mote. The hardware features of the network nodes and the required low-power consumption pose several limitations on the performance of the application. Moreover, TinyOS does not provide an efficient support to computational intensive algorithms, being in some sense network-oriented.

Therefore, although the possible applications of this implementation are limited by its performance, this work aims at evaluating the feasibility and at identifying the problems of such a project, providing a solid foundation for further experiments.

As regards this implementation, some improvements could be possible:

- The use of an exponential function to smooth the contributes of local gradients to the final descriptor vector represents a bottleneck for the whole process. From some experimental results reported in section 4.3, this operation accounts for a rough 15% of the total processing time. In the reference implementations the function has been substituted with a look-up table that samples its values at some points with a fixed interval. This could provide a great benefit in terms of processing time at a slightly higher cost in terms of memory occupation;

- The code is now implemented as a single task, preventing the operating system to interleave it with other operations. For this purpose, the code could be split into different tasks, introducing some global variables representing the state of the algorithm[1], allowing for an improved and more fair execution flow;

- SURF has been introduced as a more efficient alternative to other common feature extraction algorithms such as SIFT. Although it is way more efficient with respect to its predecessors, its performance in terms of computational time is not sufficient to meet the requirements of the possible applications. More and more efficient algorithms are being constantly introduced as time goes by, suggesting new methods for the detection and the description of interest points. Since the detector and the descriptor are detachable, a possible solution could be to adopt an hybrid approach combining different kinds of detectors and descriptors to achieve an higher efficiency[2].

- Since the programming of the nodes is carried out exploiting a serial communication channel, only small test images have been tested. For live-sized images to be processed, an alternative method for the programming has to be found. A possible solution could be the use of a digital camera, whose interface is already implemented in TinyOS, allowing for the execution of the feature extraction algorithm on real-time photos taken from that camera.

[1] the information about the scale and the coordinates of the point to be processed in the next execution of the processing task.

[2] e.g. good results are documented with an hybrid approach consisting of FAST as a detector along with SURF as a descriptor.

Bibliography

[1] Asada, G., Dong, M., Lin, T. S., Newberg, F., Pottie, G., and Kaiser, W. J. Wireless integrated network sensors: Low power systems on a chip. In *Proceedings of the 24th European Solid-State Circuits Conference (ESSCIRC '98)* (1998), pp. 9–16. WINS.

[2] Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. Speeded-up robust features (surf). *Comput. Vis. Image Underst. 110*, 3 (June 2008), 346–359.

[3] Callaway, E. H. *Wireless Sensor Networks: Architectures and Protocols*. CRC Press, Inc., Boca Raton, FL, USA, 2003.

[4] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. Brief: binary robust independent elementary features. In *Proceedings of the 11th European conference on Computer vision: Part IV* (Berlin, Heidelberg, 2010), ECCV'10, Springer-Verlag, pp. 778–792.

[5] Crossbow Technology, I. Intel imote2 data sheet.

[6] Decker, C. Wireless sensor network (wsn) platforms.

[7] Evans, C. Notes on the opensurf library. Tech. Rep. CSTR-09-001, University of Bristol, January 2009.

[8] Florack, L., ter Haar Romeny, B. M., Koenderink, J. J., and Viergever, M. A. General intensity transformations and differential invariants. *Journal of Mathematical Imaging and Vision 4*, 2 (1994), 171–187.

[9] Freeman, W. T., and Adelson, E. H. The design and use of steerable filters. *IEEE Trans. Pattern Anal. Mach. Intell. 13*, 9 (Sept. 1991), 891–906.

[10] Gossow, D., Decker, P., and Paulus, D. Robocup 2010. Springer-Verlag, Berlin, Heidelberg, 2011, ch. An evaluation of open source SURF implementations, pp. 169–179.

[11] Harris, C., and Stephens, M. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference* (1988), pp. 147–151.

[12] Leutenegger, S., Chli, M., and Siegwart, R. Brisk: Binary robust invariant scalable keypoints. In *ICCV'11* (2011), pp. 2548–2555.

[13] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., and Culler, D. Tinyos: An operating system for sensor networks. In *in Ambient Intelligence* (2004), Springer Verlag.

[14] Lowe, D. G. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2* (Washington, DC, USA, 1999), ICCV '99, IEEE Computer Society, pp. 1150–.

[15] Matas, J., Chum, O., Urban, M., and Pajdla, T. Robust wide-baseline stereo from maximally stable extremal regions. *Image Vision Comput.* (2004), 761–767.

[16] Mikolajczyk, K., and Schmid, C. An affine invariant interest point detector, 2002. Copenhagen.

[17] Mikolajczyk, K., and Schmid, C. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell. 27*, 10 (Oct. 2005), 1615–1630.

[18] Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., Kadir, T., and Gool, L. V. A comparison of affine region detectors. *Int. J. Comput. Vision 65*, 1-2 (Nov. 2005), 43–72.

[19] Orlinski, A. Pan-o-matic.

[20] Rosten, E., and Drummond, T. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision* (October 2005), vol. 2, pp. 1508–1511.

[21] Rosten, E., and Drummond, T. Machine learning for high-speed corner detection. In *European Conference on Computer Vision* (May 2006), vol. 1, pp. 430–443.

[22] Schaffalitzky, F., and Zisserman, A. Multi-view matching for unordered image sets, or "how do i organize my holiday snaps?". In *Proceedings of the 7th European Conference on Computer Vision-Part I* (London, UK, UK, 2002), ECCV '02, Springer-Verlag, pp. 414–431.

[23] Schonwalder, J., and Harvan, M. Wireless sensor networks: Motes, nesc, and tinyos. Jacobs University, Bremen.

[24] TinyOS. Intelmote2 support wiki.

[25] Wikipedia. Ieee 802.15.4 — Wikipedia, the free encyclopedia, 2012. [Online; accessed 01-November-2012].

[26] Wikipedia. Wireless sensor network — Wikipedia, the free encyclopedia, 2012. [Online; accessed 01-November-2012].

[27] Yale. Power modes and energy consumption for the imote2 sensor node.