Università di Pisa

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Specialistica in Computer Engineering

Laurea Specialistica in Computer Engineering

Design and implementation of a system for incremental real-time visual object detection and autonomous recognition

Supervisors:

Giuseppe Amato

Claudio Gennaro

Francesco Marcelloni

Candidate:

Fabio Carrara

February 2015

Abstract

In this work, a system for incremental real-time visual object detection and autonomous recognition is presented. The system is designed for indoor smart cameras and identifies objects appearing on the scene by detecting video changes in the video stream. Object detection is based on a novel interest point-based background subtraction method, which results in a more robust and informative background model with respect to typically color-based approaches. Objects are incrementally learnt by collecting observations in real-time. A similarity function between objects observations relying on local feature matching and geometric consistency checking is defined. The key idea of the system is to relate past and present object observations: clusters of similar observations are maintained exploiting transitivity of similarity between observations and are used to recognize a new observation of an already seen object. Since the system incrementally builds it up from observations during time, no training set for recognition is needed. Experiments have been performed on publicly available datasets to evaluate the detection task and the ability of the system to build good clusters of observations. The system has also been tested on the Raspberry Pi platform equipped with the Pi Camera module.

Contents

List of Figures

List of Tables

Abbreviations

PiCVi	Raspberry **Pi** **C**omputer **Vi**sion experiment
AmI	**Am**bient **I**ntelligence
GMM	**G**aussian **M**ixture **M**odel
MoG	**M**ixture **o**f **G**aussians
IP	**I**nterest **P**oint
FGE	**F**ore**G**round **E**xtraction
BGS	**B**ack**G**round **S**ubtraction

To my parent and my closest relatives who made this work possible.

To Regina and my closest friends who supported me.

Special thanks to my supervisors, particularly Giuseppe Amato and Claudio Gennaro, who made study and work during this thesis pleasant and constructive.

Chapter 1

Introduction

Thanks to the miniaturization of electronics many computing and sensing devices can become part of our environments and help us in our daily activities.

Sensors, actuators and processing units are now very affordable and can be used to build networks of intelligent devices with the capability to understand events in a specific context and take decisions in real-time without the the user thinking explicitly about it. This smart devices grow smaller and are going to completely disappear in the environment, exposing to the user only friendly, human-centric interfaces.

This emerging field of research attempting to bring invisible intelligence in evnironments is known as *Ambient Intelligence* (AmI), defined as [1, Chapter 11]:

> *"a digital environment that proactively, but sensibly, supports people in their daily lives".*

Many technologies and research areas are related to AmI, such as Sensor Networks, Human-Computer Interfaces, Pervasive Computing and Artificial Intelligence.

In the *Sensor Networks* research field, much effort has been put in *Smart Camera Networks*. Due to their descreasing cost, video surveillance systems are installed in locations ranging from big organizations to small personal habitations. Visual sensors such as cameras are capable of bringing a lot of informations that can be used in many applications, such as surveillance, healthcare, teleconferencing, and so on.

Thanks to the possibility to deploy *smart cameras* with some processing power and communication capabilities, decentralized networked systems has been designed and implemented to coordinate cameras in extracting useful informations with less and less need for direct oversigth of a human operator. Unlike single-camera systems, multi-camera systems can

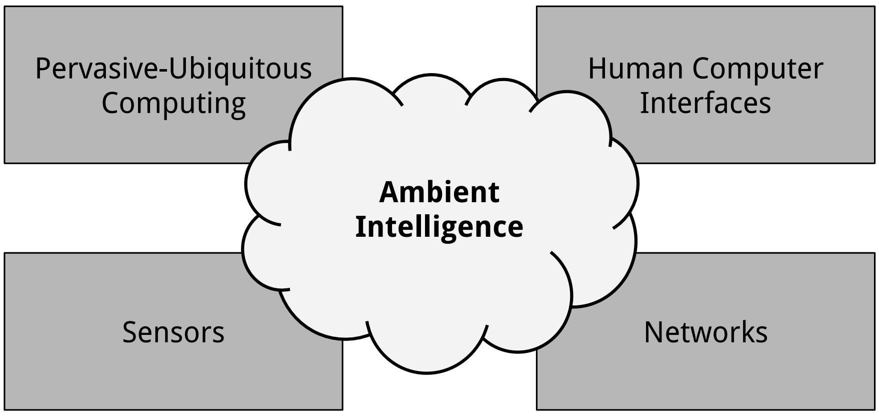


Figure 1.1
Relationship between AmI and other areas (taken from [1]):

exploit spatio-temporal informations for the application purposes, but also for automatic calibration and self-recofiguration.

In this work, PiCVi is presented: a system for incremental real-time visual object detection and autonomous recognition for smart cameras. Object detection and recognition are classical problems in computer vision, but are still challenging when trying to solve them without a priori knowledge of objects and with a limited user interaction. Moreover, the scarce resources available on current smart camera hardware pose challenges for the real-timing constraint. The implemented software tries to enable a single smart camera to detect, learn and recognize objects exploiting change detection in the scene: given the evolution of the scene during time, the system incrementally builds in a semi-supervided way a knowledge it can exploit for the recognition task. The user is queried when ambiguities cannot be resolved.

The system has not been designed for a particular smart camera platform in mind, but it has been tested on the Raspberry Pi platform equipped with a Pi Camera module.

Experiments have been made using public available datasets in order to evaluate the main two aspects of the system: a) the ability to detect objects from a video stream and b) the ability to build a knowledge base for object recognition.

Thesis Structure

In the rest of this Chapter 1, *Smart Camera Newtorks* are presented. The most used hardware architectures are briefly presented and some of their applications are briefly described.

In Chapter 2 the application scenario is described and system assumptions and specifications are reported. Also the object models and the pipelined software architecture that has been chosen are described.

In Chapter 3 the first pipeline stage dedicated to foreground objects detection from the video stream is described. The most commonly used approach for foreground extraction are briefly reported and the implemented technique is described.

In Chapter 4 the second pipeline stage is described. This stage is dedicated to the description and the comparison of the various images of objects collected in the first stage. The choosen description of the objects and the similarity function used to compare them are reported.

In Chapter 5 the third and last pipeline stage dedicated to aggregation of similar objects is described. In this stage, images of objects are clustered together based on their similarity.

In Chapter 6 the experiments done are described and the evaluation of the presented system is reported. In addition, future works and enhancements are discussed.

1.1 Background

Image matching usually relies on a set of visual features of image content. In particular, *local features* of images are used. Local features are peculiar parts of the image (like edges, corners, blobs etc.) whose statistics can be described and compared. Matching images through their local features has the advantage to be robust to clutter and occlusions. Moreover, many algorithms have been proposed to detect and describe local features invariant to scale, rotation and affine transformations. In this section, two algorithms used in this work (one for detection only, one for detection and description of local features) are briefly described.

1.1.1 FAST Interest Point Detector

FAST (Features from Accelerated Segment Test) is a high-speed corner detector good for real-time applications. It uses an machine-learning acceleration of the segment test in order to classify a pixel as corner or not.

Segment Test The segment test on a pixel p considers a circle of 16 pixels around the candidate corner (see Figure 1.2). The intensity I_x of each pixel x on the circle is compared with the intensity of I_p and the pixel is classified as *brighter* if $I_x > I_p + t$, *darker* if $I_x < I_p - t$, *similar* otherwise. If exists at least n contiguos pixels on the circle all *brigther* or *darker*, p is considered a corner.

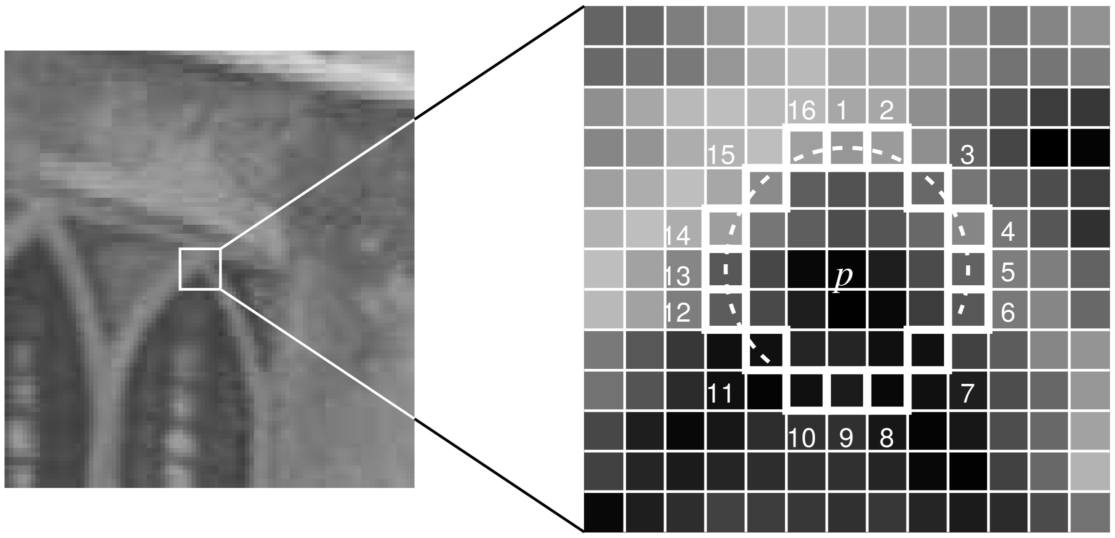


Figure 1.2

Segment Test with $n = 12$: The highlighted squares are the pixels used in the corner detection. The pixel at p is the centre of a candidate corner. The arc formed by the dashed line passes through 12 contiguous pixels which are brighter than p by more than the threshold t.

For $n = 12$, a fast test to exclude non-corner pixels is used: if p is a corner at least three of the four pixels (1, 5, 9, 13) has to be brither or darker, therefore testing those pixel first will exclude many non-corners. However, the full test has to be performed to correctly classify corner pixels.

Accelerated Segment Test Using Machine-Learning The above test is accelerated (but approximated) building a decision tree using a training set of images. Corners are detected using the full segment test on all the training images and for each detected corner its circle of 16 pixels is stored in a feature vector. Choosing a position $x \in 1..16$, the set of all corners P detected from all training images can be divided in three subsets P_b, P_s, P_d containing features having pixel number x in the circle respectively brighter, similar or darker than the central one. A decision tree is built choosing x that yields the most information (based on the entropy of the generated subset) at each level of the tree.

Non-maximal Suppression In order to avoid multiple corner detection in adjacent locations, non-maximal suppression is applied: a score function V is computed for each corner and only the corner with maximum V among the cluster of corners is returned. V is chosen as the sum of absolute differences between p and its 16 surrounding pixels.

1.1.2 ORB Interest Point Detector and Descriptor

ORB (Oriented FAST and Rotated BRIEF) [2] is a high-speed patent-free alternative to SIFT [3] and SURF [4] features detectors and descriptors, offering up to two order of magniture faster computation with almost the same matching results.

Feature Detection with oFAST Keypoints are detected in ORB using FAST feature detector (described in Subsection 1.1.1). Since FAST algorithm is not scale invariant, images pyramids are used to detect scale-invariant features. Harris cornerness measure [5] is computed on the detected keypoints and only the top N keypoints are retained. FAST also does not provide keypoint orientation, therefore rotation-invariant featuers are obtained using the *intensity weighted centroid* C and the center O of the detected image patch: the direction of the vector $\overrightarrow{OC}$ is assumed as orientation θ of the keypoint. This enhanced version of FAST is called by the authors *oFAST* (oriented FAST).

Feature Description with rBRIEF In order to compete with SIFT and SURF descriptors, ORB descriptor should perform well under rotations, still remaining efficient. An enhanced version of BRIEF descriptor called rBRIEF (rotation-aware BRIEF) is used in ORB.

Original BREIF descriptor [6] consists in a bit string representing results of intensity comparisons between fixed pairs of pixels belonging to the smoothed image patch. Let p, x, y be respectively the image patch and the positions of the pixels to be compared. The intensity test is defined as:

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

The descriptor is then defined as a vector packing those bits. Comparisons are made for each pair of pixels (x_i, y_i) belonging to a predefined set S. A subset of S is visually represented in Figure 1.3.

This approach does not perform well under rotation transformations. Therefore in rBRIEF the positions of the pixels (x_i, y_i) to be compared are rotated using the orientation information θ of the keypoint, obtaining a new set of S_θ for comparisons. The orientation θ is discretized in increments of 12 degrees and all possible S_θ are precomputed and stored in a lookup table. A greedy learning algorithm has been developed and used to select the best positions (x_i, y_i) in order to have uncorrelated tests.

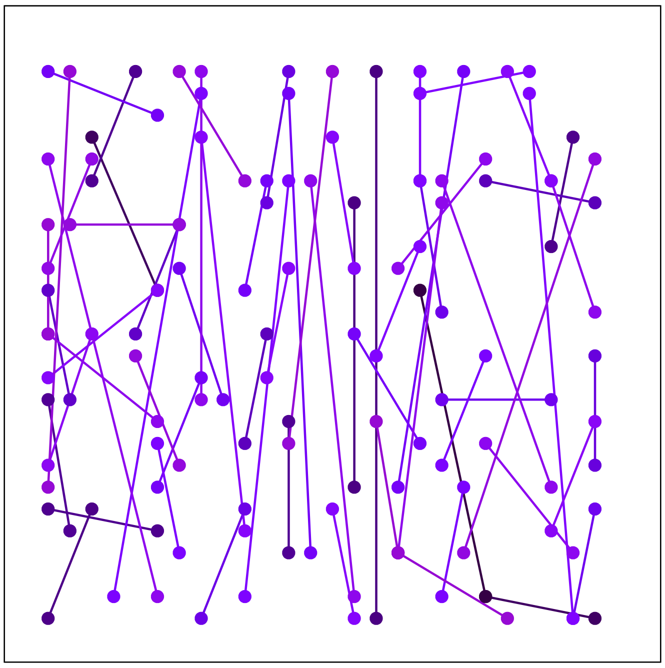


Figure 1.3

A subset of intensity tests performed on pixels nearby the keypoint in ORB.

1.2 Related work

Healthcare, Teleimmersion, and Surveillance

Chen et al. [7] present variuos domains and applications enabled by markerless motion capture and object tracking systems implemented through smart camera networks. Human body motion capture systems can be useful for applications in *healthcare* (posture analysis, remote rehab feedback, telemonitoring of patients), *teleimmersion* (virtual presence, 3D teleconferencing) and *surveillance* (face and/or gender identification, anomaly detection).

The main objective of this technology is to exploit smart camera networks to build models for objects or human parts in scenes under different conditions and assumptions (with overlapping or nonoverlapping fields of view, with or without markers). Much of the work is on **marker-less** motion caputre systems, since they does not require preparation to collect data, such as markers attached to the body or 3D scanners, hence enabling wider application area. However, they introduce the problem of calibration of cameras and they obtain a less rigorous movement description with respect to marker-based systems.

For teleimmersion applications, multiple cameras offers different viewpoints and can be used to construct a 3D free-viewpoint video. High resolution 3D video can be useful in many fields, such as medical imaging, scientific data and models, remote training and teaching, 3D teleconferencing. Smart camera networks systems capable of obtaining 3D videos are divided in a) passive reconstruction systems, in which 3D video is obtained by analyzing the single 2D videos of each camera and combining them together and b) active vision systems, in which specialized hardware is used to estimate the world 3D model. The former technique is less robust than the latter, but requires no additional hardware.

For surveillance applications, Chen et al. generalize survelliance systems to a four stage processing pipeline: spot the mover (**detection**), find its position (**localization**), predict mover's movements (**track**), label the mover if it is known (**verify and recognize**). The use of smart camera networks (in particular motion capture systems) can help in any of the above stages.

Surveillance of Public Spaces

Abas et al. [8] present a taxonomy based on cost-performance tradeoff of smart camera network systems applied to indoor and outdoor surveillance. System architectures such as Citric[1], HuSIMS, OmniEye, Wi-FLIP, CamInSens[2], MeshEye are analized. For each project the following properties are reported: energy efficiency techniques used, usage of multimodal sensors, bandwidth efficiency techniques used, wireless technology used, computer vision algorithm used, system software used, whether camera overlaps or not, how security is assured. The authors also introduce their solution: *SWEETcam*, a network of specialized hardware that tries to maximize the cost-performance tradeoff. SWEETcam is based on the Raspberry Pi platform running a specialized Linux OS which assures flexibility and code reuse. WiFi is used as wireless link among cameras. The platform is equipped with the Pi's camera module which does not drain as much current as USB cameras and algorithms are implemented in C/C++ using the OpenCV library. A low power MSP430 microcontroller accepts hardware interrupts from a passive motion sensor (PIR) and wakes the Raspberry Pi only if an event occurs, saving energy. The system is equipped with a solar cell for energy harvesting and image processing is based on a simplified MoG foreground extraction and object classification capable of running with the scarce resources available.

Distributed 3D object recognition

Naikal et al. [9] present a scheme for distributed 3D object classification on band-limited smart cameras. Compressive sensing (CS) based codec is used to reduce bandwidth needs and exploiting the reconstruction of jointly sparse feature histograms. A new training set of 3D buildings is created by the authors: the BMW (Berkley Multiview Wireless) DB. The proposed recognition workflow is the following:

[1]`http://www.eecs.berkeley.edu/~yang/software/CITRIC/`
[2]`http://www.caminsens.org/`

1. A vocabulary of features using hierarchical k-means clustering is built offline from the data set.
2. Each camera:
    (a) extract features F_i from the scene,
    (b) quantize them against the build vocabulary and obtain a feature histogram x_i,
    (c) randomly projects it (motivated by CS, obtaining b_i) and send it to central station,
3. The central station:
    (a) receives L projections $b_1...b_L$ from the cameras,
    (b) jointly decodes them with a L1-min algorithm to achieve $\hat{x}_1...\hat{x}_L$,
    (c) classify object using a vocabulary tree built during clustering.

Experiments shown that feature dimensionality reduction obtained with compressive sensing based codec leads to quite the same recognition rates as for the uncoded case, with the advantages of a lower bandwidth usage by the cameras.

Chapter 2

PiCVi: autonomous object detection and recognition system

In this chapter, PiCVi is described: a Raspberry **Pi C**omputer **Vi**sion experiment on visual object detection and autonomous recognition in indoor environments.

2.1 Goals and Assumptions

The main goal of the presented work is to design and implement a system being able to:

- visually detect movable 3D objects in the environment,
- autonomously and incrementally learn to recognize detected objects.

The developed system is intended to be a first processing stage or task for a single smart camera in a larger collaborative project involving a newtork of smart camera, hence no communication between multiple cameras is expected at this stage. Moreover only movable (not fixed) objects are detectable.

Application Scenario In this work, it is assumed that the camera used by the system is fixed and working in indoor environments. In the application scenario, the camera is continuosly screening a room (or part of it) while people are entering and leaving the camera's field of view adding, moving or removing objects (e.g. a book, the remote control, a backpack etc.) from the scene. The scene is therefore composed by a substantially fixed part containing walls and fornitures in which sometimes objects and/or people appear. The camera has the possibility to see various objects in different poses and illuminations over time. Therefore, it can inspect the similarity between observed objects and learn to distinguish them from each other. Each time an object enters the scene, the camera can extract it from the background, recognize and label it if it is known or learn to recognize it in the future.

Future Goals A single learning camera could distinguish many objects over time. Using a network of cameras, the system could refine the recognition task adding new poses of the same object, implementing position triangulation and geometry checks. Moreover the system could proactively ask the user which of the most often seen objects he wants to track and to give them a name. Doing so, the system could be able to answer querys like *where is object X? or where did you last seen it?* and perhaps send a robot to the current object location to fetch it.

Object Model There are many solutions to object detection and recognition already present in computer vision literature. Unfortunately, most of them need a training set or a motion of the object ([10, 11] cites). Since the system has to detect objects only, the detection schemes applied are not based on motion, which is peculiar to living beings. Instead, the system tries to observe changes of the scene that persist over time, assuming changes to be due to objects added, moved or removed from the scene.

Computing Platform The software is written in Java and the computer vision algorithms are implemented using the OpenCV[1] library. This choice ensures a great portability on different smart camera platforms, which usually run an embedded Linux operating system. Moreover, as the system name suggests, the developed software has been tested on the Raspberry Pi platform equipped with a Pi Camera Module in order to have a feedback on the performances on a possible smart camera platform.

Raspberry Pi The Raspberry Pi[2] is a low-power credit card-sized single board computer with a 700MHz ARM processor, VideoCore IV GPU, 512Mb of RAM and around $30 of price. Camera sensors can be attached to it via USB using classic webcams or better via CSI (Camera Serial Interface) using the Raspberry Pi Camera module, which enables a faster visual data acquisition rate and a less current drain with respect to the USB interface. Its specifications make the Raspberry Pi a good candidate for a smart camera hardware platform, indeed it is already present in camera networks literature [8].

[1]http://opencv.org/

[2]http://www.raspberrypi.org/

Figure 2.1

A Raspberry Pi equipped with the Pi Camera Module.

2.2 Software architecture

The system is designed as a pipeline of processing steps applied to the incoming video stream.

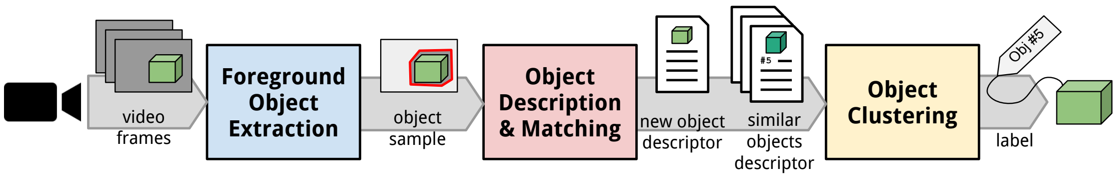

video frames
Foreground Object Extraction
object sample
Object Description & Matching
new object descriptor
#5
similar objects descriptor
Object Clustering
Obj #5
label

Figure 2.2

PiCVi Pipeline Software Architecture

2.2.1 Stage 1: Foreground Object Extraction

In this first stage, the incoming video stream is analized to search for objects. A background model is created and continuosly updated: each incoming frame of the video is compared with the model and non matching parts are interpreted as foreground. Foreground parts stable over time are likely to contain new objects, therefore are extracted and given to the next stage. This stage is described in detail in Chapter 3.

2.2.2 Stage 2: Object Description and Matching

In this stage, the incoming foreground regions of the video frame are analized to search for recognizable objects. The parts with too little or no information are discarded whereas the information-rich parts are considered as particular 2D views (samples) of 3D objects. The visual features of each new *object sample* are extracted and a descriptor is inserted in the local database. Furthermore, the new object sample is compared with already met samples present in the local database and a list of matches is generated and given to the next step. This stage is described in detail in Chapter 4.

2.2.3 Stage 3: Object Online Clustering

In this stage the incoming new object sample and the ones that potentially match are analized and grouped into object clusters in a semi-supervised way. Usually a new object sample is added automatically in an existing cluster, but this is not always the case. After the online clustering, a label is given to the new object sample, which is the output of this last stage. This stage is described in detail in Chapter 5.

2.2.4 Output: Labels

The systems gives in output at certain frames the contour and the label of the object samples detected in the scene. Since objects are autonomously learnt, a numeric label is assigned to each new object. The user will be called to give a name to the objects he is interested to be recognizeable.

Chapter 3

Foreground Object Extraction

In this chapter, the first stage of the PiCVi pipeline is described and the techniques for foreground extraction that have been studied and implemented are reported.

The goal of this stage is to analize frames coming from the still video camera and extract **object samples**, i.e. the parts of the scene containing a view of an object that has to be recognized (see Figure 3.1).

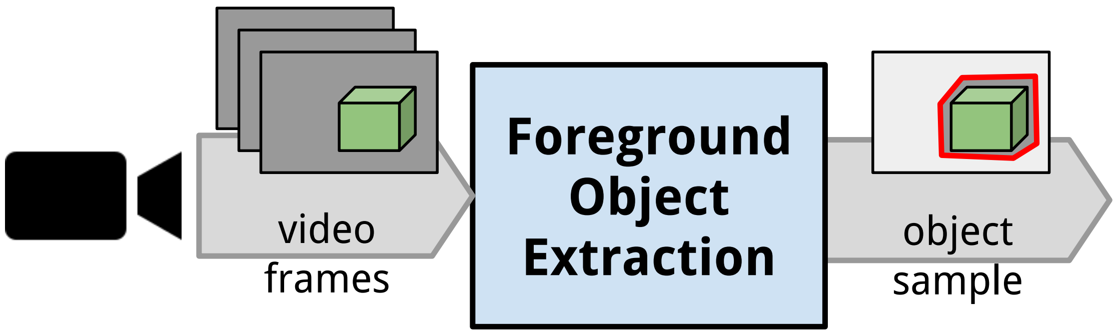


Figure 3.1
First stage of the PiCVi pipeline, having video frames coming from the camera as input and foreground object parts (object samples) as output.

Since the system comes with no information about the objects the user wants to detect and recognize, the system searches for objects where it detects chages in the scene: **foreground extraction** is applied to the video stream.

Foreground extraction is the task of segment an image coming from a video stream in two parts: the relatively fixed and usually not interesting part (the *background*) and the unexpected new part (the *foreground*) which is different from what is usually seen. In the literature, this task is accomplished by appling a **background subtraction** method.

3.1 Background Subtraction Methods

In this section the concept of background subtraction is introduced and the studied state of the art techniques applied to object detection are briefly reported [12].

The idea behind the background subtraction technique is to build and maintain a *background model*, which can be compared with the incoming video frame in order to find spots of the image that differs from the model.

The output of background subtraction method is a **foreground mask**: a binary or grayscale image where a pixel is zero-valued if it is considered background, non-zero otherwise.

Background subtraction methods differs in the type of background model and in the algorithms to evaluate and update the model.

3.1.1 Frame differencing

A very simple background model that can be maintained is the entire frame, i.e. the RGB values of each pixel in the image.

In order to evaluate the current foreground mask, the background model is litterally subtracted from the current frame and a threshold is applied. A pixel value farther more than the threshold from the background model is considered as foreground (see Figure 3.2). This technique is called **frame differencing** and its key idea gave the name of “background subtraction” to the methods executing this type of task.

The background model can be updated in various way: a new frame can substitute or be combined with the old model in order to maintain a model more representative of the current background state.

This method has good performances since no complex operations are done for each frame and can be easly implemented in OpenCV (see Table 3.1). A simple implementation can be attractive in smart camera embedded platforms where usually computational resources have to be saved, but this method is not robust to any type of background changes, such as illumination changes or active background (i.e. moving leaves on tree, see Figure 6.1).

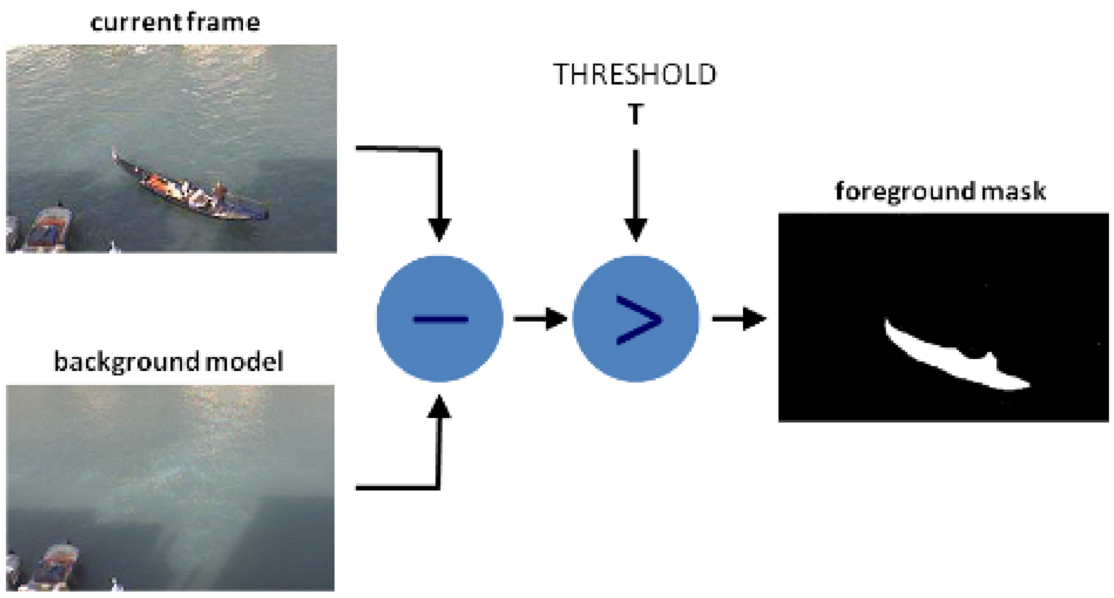


Figure 3.2

Background Subtraction: Frame differencing scheme, taken from [13]

3.1.2 Pixelwise Background Subtraction based on Gaussian Mixture Model

A widely used approach for background modelling is **mixtures of gaussians** [14]. In this approach, statistics of each pixel are modelled by a mixture of a variable number N of gaussians. The probability of of observing a pixel with a certain RGB value $x \in \mathbb{R}^3$ is the following:

$$p(x) = \sum_{i=1}^{N} w_i \cdot n(x; \mu_i, \Sigma_i)$$

where $n(x; \mu_i, \Sigma_i)$ is a multidimensional gaussian probability density with mean vector $\mu_i \in \mathbb{R}^3$ (representing the mean of value the pixel) and covariance matrix $\Sigma_i \in \mathbb{R}^{3\times3}$:

$$n(x; \mu_i, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^3 |\Sigma_i|}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)}$$

and w_i are the weights associated with each gaussian, defined such that

$$\sum_{i=1}^{N} w_i = 1.$$

This kind of model can represent up to N different "sources" of background in a single pixel. For example, given a repetitive moving background such as a tree blowing in the wind, a pixel value could oscillate between green of the leaves and another color from the background. A gaussian component of the mixture could represent the green value from the leaves and another component could represent the underneath color. If the model is correctly trained, both colors can be correctly classified as background.

The model training and update requires to change accordingly the parameters of the mixture, that are the weigths $w_1 \dots w_N$ and the parameters of each gaussian $(\mu_1, \Sigma_1) \dots (\mu_N, \Sigma_N)$, based on incoming frames. A good choice of those parameters is their maximum-likelihood estimates given a set of samples (i.e. pixel values). This can be done using the **EM** (Expectation-Maximization) **algorithm** [15] on a sliding window of pixel values.

The EM algorithm iteratively performs two steps: in the first one (E-step) the likelihood of each sample in the window is calculated evaluating the gaussian mixture with the current parameters. In the second step (M-step) those likelihoods are used to refine the current parameters. The algorithm stops when the relative change in likelihood is small.

An enhanced version [16] of this background subtraction algorithm is implemented and present in OpenCV. In Figure 3.3 a result of the application of this method is shown. Morphological filtering operations (like erosion, dilation, closing and opening) can be applied to the obtained foreground mask in order to remove unwanted noise, such as single pixels foregrounds or background holes in a foreground object.

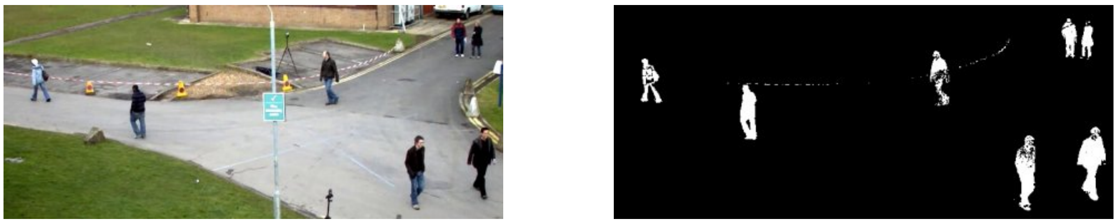

(a) original frame **(b)** foreground mask

Figure 3.3
Result of Background Subtraction based on Gaussian Mixture Model: (a) the current frame on which the foreground needs to be extracted. (b) The foreground mask obtained after the background subtraction.

Performances on the Raspberry Pi platform are good (see Table 3.1), but still this method presented some problems:

- the method is only color-based. Therefore if the foreground object we are trying to detect shares colors with the background, the background subtraction does not perform well (see Figure 3.4).
- training and updating the model with a number $N > 1$ of gaussian components results in an unwanted *memory effect*. For instance consider the following scenario: a new object appearing in the scene is correctly detected and after a while, thanks to the online training, its pixel values becomes part of the background model. If the object

is removed from the scene and then reinserted in approximately the same position, a gaussian component of the model previously trained will match with the object pixel values, marking them as background. This problem is better visualized in Figure 3.5.

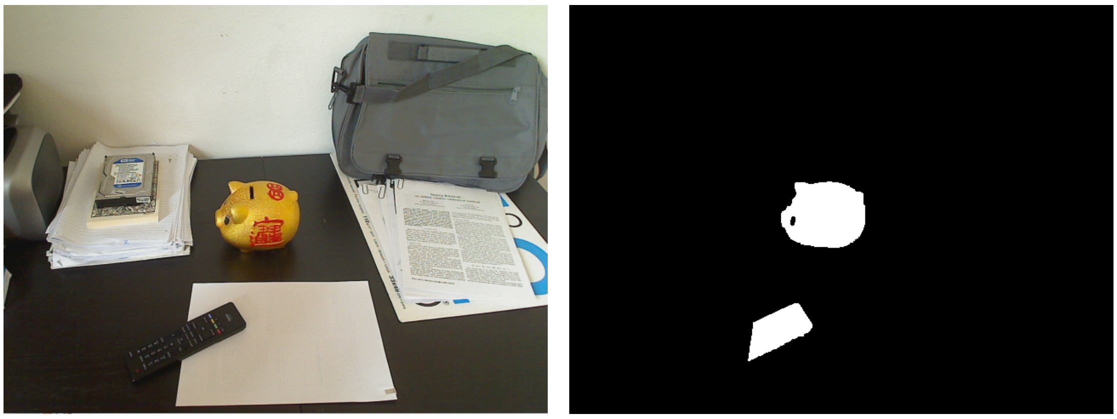


Figure 3.4

Example foreground-background missclassification in background subtraction method based on GMM: half of the remote control shares color with the background and therefore is misclassified.

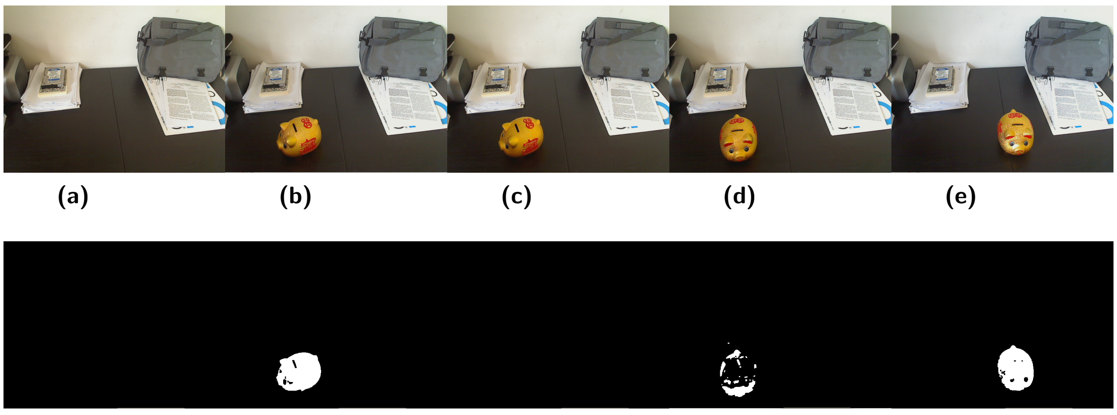


Figure 3.5

Example of unwanted memory effect in background subtraction method based on GMM. Frames (a-e) and extracted foreground masks (f-j) of a testing video are shown: background is initially trained (a), then an object is inserted and detected (b). After a while it is inserted in the background model (c). The object is then removed and reinserted in a slightly different position, but its detection is not correct due to memory effect (d). No problem arises if the object is reinserted in a position non overlapping with the first one (e).

3.2 IP-Based Foreground Extraction Method Implemented in PiCVi

The main drawback of the methods presented is that they are color-based. The key idea of the method implemented in PiCVi is to have a interest point (IP)-based model of the

Method	FPS
Frame Differencing	8.60
Mixture of Gaussians	8.54
IP-based	5.01

Table 3.1
Performances in FPS of tested Foreground Extraction methods applied on the video stream coming from the Raspberry Pi Camera Module.

background: analizing the evolution of the position of the interest points we can construct a model robust to color variations. The choice of this kind of background model is justified by the second stage of the PiCVi pipeline (described in Chapter 4), where object samples are described and matched based on their local features: using a foreground extraction method based on local features we may loose some relevant part of the foreground with respect to color-based methods, but those parts will result in no useful informations for the next stage. Adding the processing step of finding interest points leads to a higher computational demand which decreases the frame processing rate of the camera (Table 3.1), but still remaining acceptable for most smart camera real-time application.

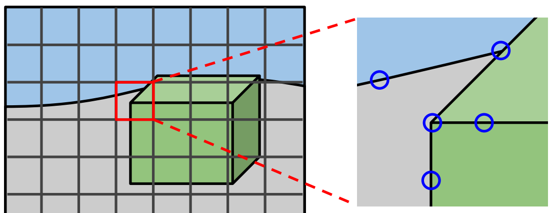
The method presented in this section is an enhanced version of the method described in [17]. The main goal of this method is to segment the local features extracted from the current frame in two sets: the foreground features, potentially belonging to a foreground object, and the background ones, which are fixed and not interesting for our goal.

3.2.1 Background Model

A block-wise background model is built. The image coming from the camera is divided into blocks of $W_b \times H_b$ pixels. After all interest points (IPs), i.e. local features, have been extracted from the image, they are assigned to the appropriate block based on their (x, y) position in the image. In each block, the set of IPs positions is called an **event** (Figure 3.6a). In order to facilitate event labelling, the 2D coordinates of the IPs are mapped in a 1D coordinate by numbering the pixels from 0 at the top left corner of the block, and then counting along each row from left to right to $W_b \times H_b - 1$ at the bottom right corner. Therefore, an event is represented by the set of those 1D coordinate (see Figure 3.6b).

A counter is associated with each event and it is incremented every time that particular event (the simultaneous observation of that group of IPs) occurred.

The whole background model is made by the sets of the occurred events (one set for each block) and their associated counters.



(a)

Event Label	Counter	Timestamp
(28,32,64,67,84)	7	143

(b)

Figure 3.6
(a) The visualization of an event as simultaneous observation of interest point positions inside a block. (b) Numerical representation of the event.

3.2.2 Model Training

The model is continuosly updated at every incoming frame: each detected IP is associated with the correct block and it is going to be part of the current event for that block. Then for each block the event is inserted in the set of the occured events of that block. If the event is already present in the set, its counter is incremented, otherwise a new counter is created.

In order not to store forever all IPs which occurred at least once, an aging technique is applied. A timestamp (frame number) is associated with each event and updated every time that event occurs: if an event does not occur again for a fixed number of frames, it is discarded and removed from the model.

An example of background model for a block is reported in Table 3.2.

3.2.3 Model Evaluation

In each block, a set B of background IP positions is maintained: every time a counter is above a threshold parameter T, all the 1D coordinates belonging to the associated event are inserted in B.

If a position of an incoming IP is present in B, it is considered as a background IP, otherwise it is considered as foreground one.

After this preliminary classification of the incoming IPs, two post-processing tasks are executed in order to decrease misclassifications:

Background Zone Enlargment Due to acquisition noise the exact positions of the IPs can slightly change during time. Therefore a background point could be incorrectly classified as a foreground one when its position is not the same as the background

#	Event Label	Counter	Timestamp	#	Event Label	Counter	Timestamp
1	(16,19,22)	1	187	24	(18,20,23)	1	46
2	(10,16,20,23)	2	117	25	(11,22,32,48)	1	192
3	(20)	1	14	26	(22)	2	141
4	(10,23)	1	3	27	(11,16,22)	1	113
5	**(11)**	**11**	**204**	28	(16,18,20,22)	1	195
6	**(19,23)**	**29**	**220**	29	(20,23,32)	1	89
7	(16,18,20,23)	1	76	**30**	**(20,23)**	**12**	**214**
8	**(20,22)**	**5**	**212**	31	(18,22)	1	53
9	(11,22)	3	215	**32**	**(11,23)**	**34**	**208**
10	(18,23)	2	209	33	(11,23,32)	1	205
11	(20,22,26)	1	72	34	(16,23)	1	139
12	(12,23,25)	1	4	35	(9,19,23)	1	42
13	(11,16,23,32)	1	151	36	(12,16)	2	96
14	**(23)**	**11**	**216**	37	(12,18,23)	1	19
15	**(11,16,23)**	**6**	**210**	**38**	**(12,23)**	**15**	**218**
16	(11,23,48)	2	95	39	(10,12,22)	1	104
17	(19,22,32)	1	217	40	(12,18)	2	31
18	(18)	1	196	41	(12)	2	116
19	(19,22)	3	200	42	(9,11,13,23)	1	219
20	(13)	1	213	43	(9,11,23)	1	107
21	(4,20)	1	105	44	(12,16,23,26,32)	1	197
22	(10,20,23)	1	181	45	(11,23,25)	2	69
23	(16,20,22)	3	171	**46**	**(12,16,23)**	**6**	**178**

(a)

Background-IP-List: 11, 12, 16, 19, 20, 22, 23

(b)

Table 3.2
(a) Example of background model for an image block (without post-processing tasks), containing all the events occurred (background events are bold). In this example, the background events are obtained applying a threshold $T = 5$ to the counter column. (b) The obtained background positions list B used to classify incoming interest points. It is obtained grouping together the positions of background events.

point present in the B set. In order to limit this effect, the 3×3 neighbors pixels of any background IP are considered as background and are inserted in B.

Neighbor Blocks Analisys The blocks of the image can be divided in three different types: 1) *background blocks*, that contains only background IPs, 2) *foreground blocks*, containing only foreground IPs and 3) *mixed blocks*, that contains both. Some blocks are spourious and all their IPs can be correctly reclassified in foreground or background points. For example a mixed or foreground block with no foreground blocks in its neghborhood is probably spourious and has to be reclassified as background block. Similarly a background block with many foreground blocks in its neighborhood is probably a foreground one.

To do this, a binary image with one pixel per block is created: each pixel has value 1 if the correspondent block is a mixed or foreground one, 0 otherwise. A 2D 3×3 filter, reported in Table 3.3, is applied to compute the number of foreground neighbors for each block. If the neighbors count is below a threshold parameter T_{FG} the block and all its IPs are reclassified as background since it has not enough foreground neighbors to be considered foreground. If the neighbors count is above a threshold parameter

1	1	1
1	0	1
1	1	1

Table 3.3
The 2D 3×3 filter used to compute the number of foreground neighbors of a block.

$T_{BG} > T_{FG}$, the block is surrounded by enough foreground blocks to be considered a foreground one and all its IPs are reclassified as foreground.

At the end, the output of this algorithm is the partition of the initial set of IPs in foreground and background ones, respectively shown in Figure 3.7 as blue and red circles.

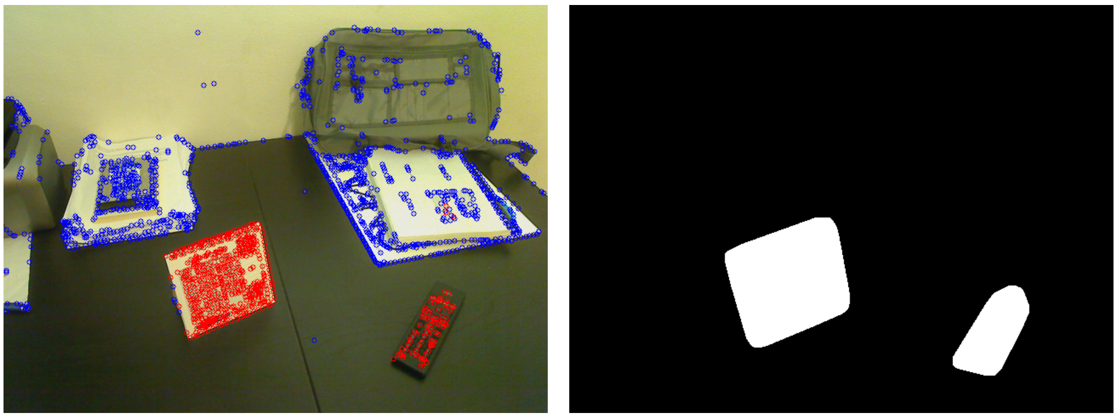

(a) **(b)**

Figure 3.7
IP based Background Subtraction output: (a) features are partitioned in Foreground IPs (red circles) and Background IPs (blue circles). (b) The related foreground mask.

A foreground mask similar to the ones produced by the background subtraction algorithms previously described is obtained "drawing white filled" circles centered in the foreground IPs on a completely black mask. The radius of the drawn circles is fixed and is not depending on the interest point size since in this stage we are not detecting them at different scales. Morphological opening operation is applied to the mask, in order to discard singular spourious foreground spots and to fill background holes inside a foreground area. Finally, the convex hull of each isolated white spot is found and filled (see Figure 3.7).

3.2.4 Foreground Extraction State Machine

Given the assumption that the system has to detect and recognize non-moving objects, the foreground is extracted from the scene only when a new object is present and the scene is stable, i.e. there is no movement of the new objects. To do so, the system is defined as a 3-state machine that cyclically executes the following tasks:

Background Training Initially the system trains and evaluates the main background model applying the algorithms described until almost every incoming IP is classified as background. A running average of the percentage of foreground points is maintained:

$$\text{fgRatio}^{(0)} = 1$$

$$\text{fgRatio}^{(N+1)} = \alpha \cdot \text{fgRatio}^{(N)} + (1-\alpha) \cdot \frac{\#\text{ fg IPs}}{\#\text{ all IPs}}$$

When fgRatio is below a threshold parameter T_{train}, the model training is suspended and the system goes in the **Foreground Detection** state.

Foreground Detection In this state, the system is ready to detect new objects appearing in the scene. A new background model is created in parallel to the main one and it is fed with the foreground IPs classified by the main model, creating a second level of classification (see Figure 3.8). The foreground IPs are divided in **fixed foreground points**, which are the background ones of the new model, and **transient foreground points**. A running average of the fixed foreground points is maintained:

$$\text{fixedFgRatio}^{(0)} = 1$$

$$\text{fixedFgRatio}^{(N+1)} = \beta \cdot \text{fixedFgRatio}^{(N)} + (1-\beta) \cdot \frac{\#\text{ fixed fg IPs}}{\#\text{ all IPs}}$$

When fixedFgRatio is stable to a value greater than 0, the system assumes there is at least a new non-moving object in the scene and goes to the **Object Extraction** state.

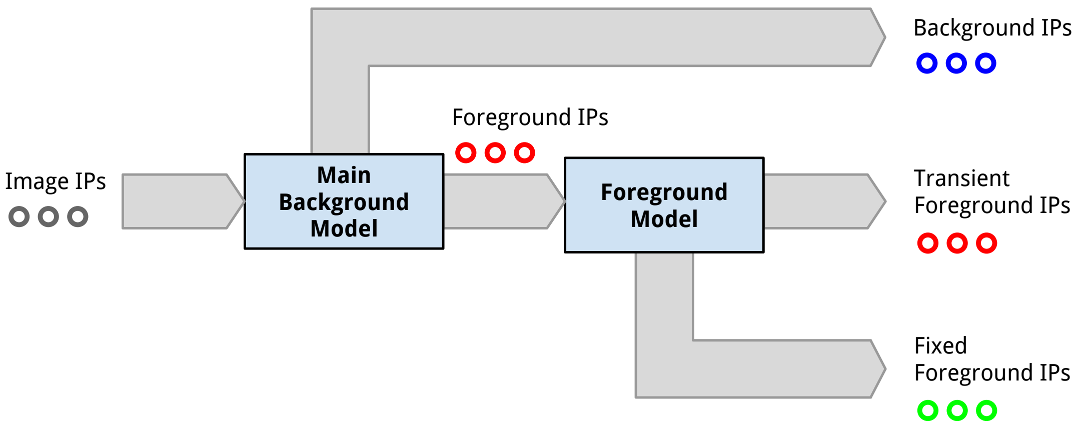
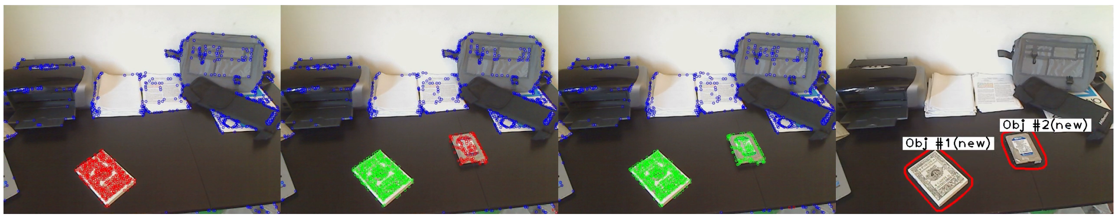


Figure 3.8
Scheme of the 2-tier classification of IPs

Object Extraction The system computes the foreground mask of the current frame using the fixed foreground points obtained from the previous state. Morphological filters and contour-finding algorithms are applied and foreground image patches are extracted and given as input to the second stage of the PiCVi pipeline, described in Chapter 4. At the end of this task, the system returns to the **Background Training** state and the IPs of the new objects are inserted in the background model in order to avoid re-detection.

An example of output of this pipeline stage is shown in Figure 3.9.



(a) **(b)** **(c)** **(d)**

Figure 3.9

Foreground Extraction Method: background features are drawn in the scene (blue). (a) A new object is inserted in the scene and its features are classified as transient foregroung features *(red). (b) After a while they became* fixed foreground features *(green), another object is inserted and its features are classified again as* transient foregroung features *(red). (c) Both objects are considered fixed and (d) object samples are extracted.*

The above method is independent from the interest point detection algorithm. Among the detectors already implemented and shipped in OpenCV, FAST [18] detection algorithm has been chosen for different reasons:

- its implementation does not use image pyramids. In this stage, we are searching for parts of the frame containing detectable features and we are not interested in finding features robust to scale transformations, hence image pyramids are unnecessary.
- it does not limit the number of detected points. Since the background model is based on interest point positions, detected points should not disappear from a part of the image because of the insertion of something which has stronger points.
- it is faster than other detectors (see Figure 3.10), which is relevant in a low resource computing platform.

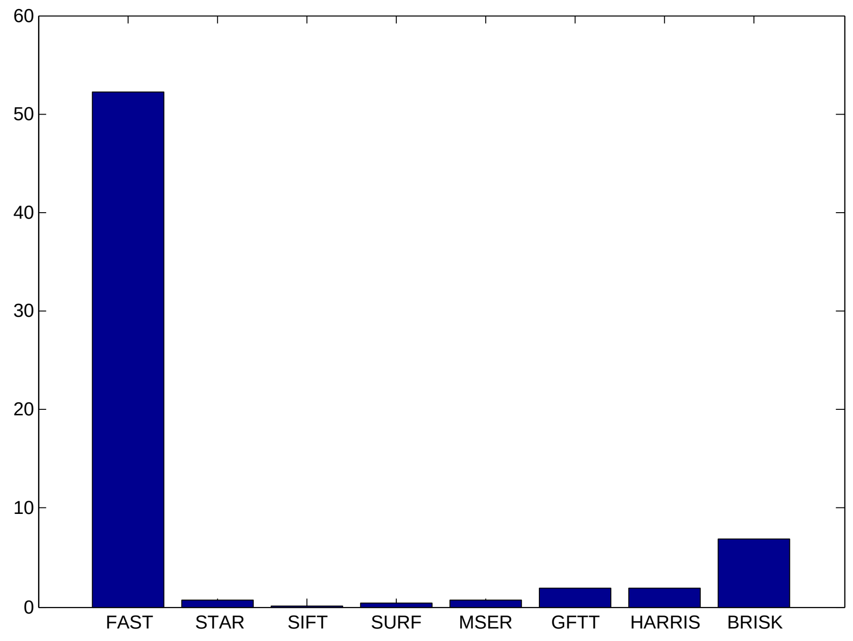


Figure 3.10

Performances in FPS of Feature Detectors available in OpenCV applied on the video stream coming from the Raspberry Pi Camera Module

Chapter 4

Object Description and Matching

In this chapter, the second stage of the PiCVi pipeline is described. The goal of this stage is to extract a description of the *object sample* (i.e. a patch of the original video frame containing only the object to be recognized) coming from previous stage, store it in a database and compare it to other descriptions of other already met object samples. Therefore the output of this stage is a list of descriptors of object samples that are visually similar (match) with the incoming one (see Figure 4.1).

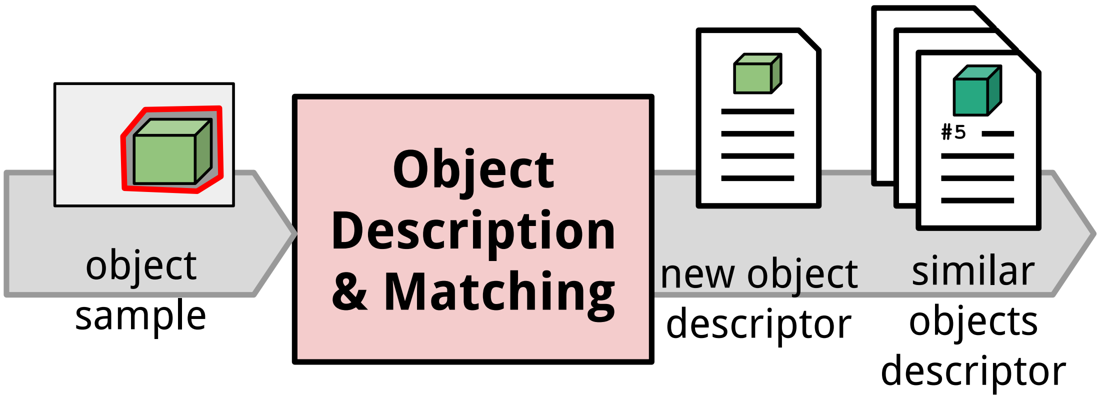


Figure 4.1
Second stage of the PiCVi pipeline, having object samples as input and object descriptors as output.

4.1 Object Sample Description

A descriptor of an object sample is created and related to other descriptions of object samples seen in the past: the system may alredy seen several times the same object under different lightning conditions, different 3D poses or different scales, but should be able to relate each observation to each other.

Many computer vision methods are reported in the literature for recognizing a known object in a scene [10, 19–21]. In this work, the comparison of two object samples is based on **local features** of the image [2–4]. Many of the widely used features detection, extraction and matching algorithms are already implemented and bundled in OpenCV. Other type of features, like the ones based on shape or color, may be integrated in order to obtain a better description of the object sample, but are not covered in this work and left to future work (see Subsection 6.2.1).

In PiCVi, an object sample is described by a) the position of its local features (keypoints) and b) their extracted descriptors. ORB detector and extractor algorithms [2] have been chosen among the methods available in OpenCV, mainly because of its good tradeoff between performance (see Figure 4.2) and robustness. It can be seen in Section 4.2 that local features matching is only the first of many processing steps for computing similarity among descriptors and is not required to be precise: a low false negative rate is required in order not to loose potentially good matches while false positives are filtered out by subsequent processing steps.

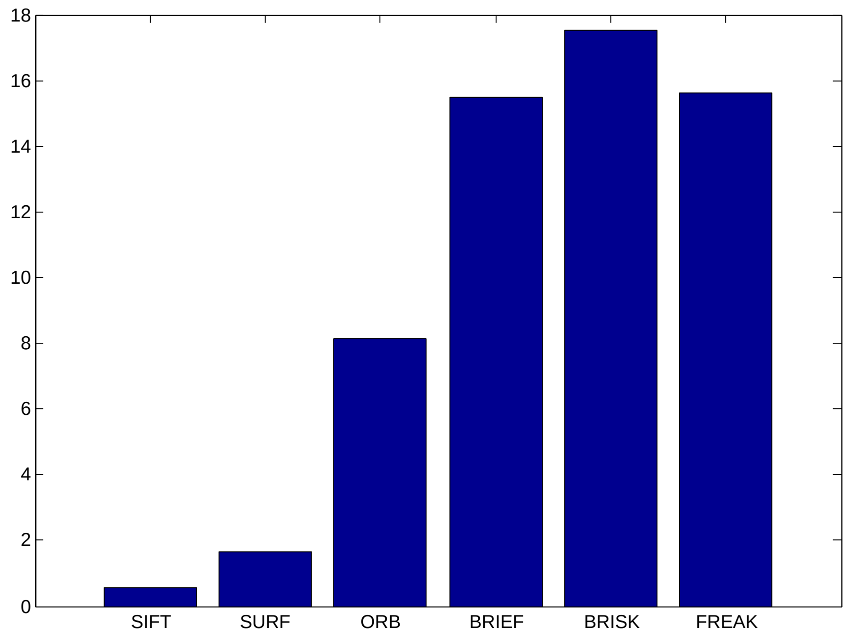


Figure 4.2

Performances in FPS of Feature Descriptor Extractors available in OpenCV applied on the video stream coming from the Raspberry Pi Camera Module

Keypoints position and ORB descriptors of each object sample are stored in a SQLite[1] database together with other useful informations. The complete description of an object sample is reported in Table 4.1.

[1]http://www.sqlite.org/

FIELD	Description
ID	an integer unique identifier of the sample
URL	the url of the image of the sample stored in the filesystem (used only for debugging purposes)
TIMESTAMP	the time the object sample descriptor has been created
KEYPOINTS	the binary serialization of the keypoints detected in the sample
DESCRIPTORS	the binary serialization of the descriptors extracted from the sample
CLASS-TAG	the label of the class the sample belongs to (used during the third stage of the PICVI pipeline: Object Clustering, described in Chapter 5)

Table 4.1
Information stored in the DB for each object sample.

It may happen that the incoming sample has not enough features to be described and to be compared with other samples. In this case, the sample is discarded.

4.2 Similarity Function

A similarity function $S : (o_1, o_2) \rightarrow [0, 1]$ is defined on a pair of object samples (o_1, o_2), representing the quality of the match between the two object descriptors. Similarity value goes from 0 (object samples do not match at all) to 1 (perfect match, usually obtained only comparing an object sample to itself).

The similarity value among two objects is computed in steps shown in Figure 4.3 and described.

In each step, the system tries to stop the similarity computation in case of bad match, returning a value 0 of similarity and avoiding further processing. In this way bad matches are fastly recognized and filtered out.

4.2.1 Feature Matching

Let K_1, K_2 be the sets of keypoints of the compared objects and D_1, D_2 their sets of corresponding descriptors.

Bruteforce Nearest Neighbor Match A preliminary list of descriptor matches is created finding for each descriptor in D_1 its nearest neighbor in D_2 using the bruteforce method (Figure 4.3d). The nearest neighbor descriptor is the one having smallest distance with respect

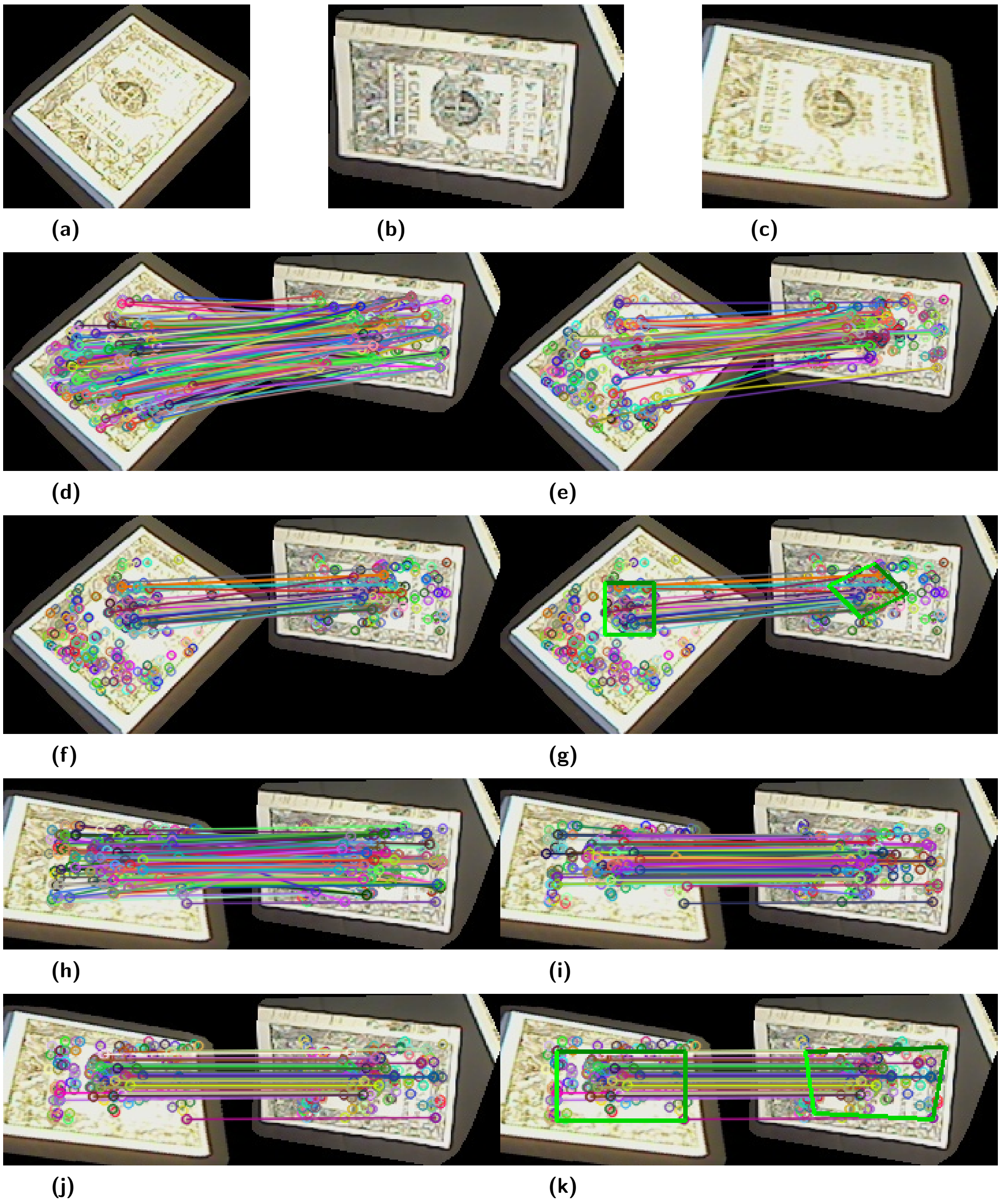


Figure 4.3

Example of similarity computation among object samples (a) and (b). Matches are obtained finding nearest neighbor for each descriptor (d) and then are thresholded (e). RANSAC is applied to find an homography and only inliers are kept (f). Found homography is shown in (g). The sample (a) is transformed using the found homography in (c) and all matching steps are reapplied: all matches (h), thresholded matches (i), RANSAC inliers (j) and homography found (k). The computed similarity value is 0.48.

to the descriptor in D_1 we are considering. Distances between descriptors are computed using the method suggested by the authors of the descriptor. In case of ORB, Hamming distance between the binary representation of descriptors is used.

Thresholding Matches Matches in the list are then filtered maintaining only the ones having distance between descriptors below a parameter threshold T_m (Figure 4.3e). If there are less than N_m matches left in the list, the system cannot correctly compare the object samples and assumes there is no similarity among them, returning a similarity value of 0.

4.2.2 RANSAC Filtering of Matches

Thresholded matches are not sure to be good ones: although a matching couple of descriptors have small Hamming distance among them, their respective keypoints may not be detected from the same point of the object (see Figure 4.3e). In this case, the match is a false positive one. These kind of matches can be filtered out checking whether the points that match are geometrically consistent.

Homography Estimation Two images of the same planar surface in space are related by a **homography** [22]. A homography is a invertible transformation that maps the 2D coordinates of points in a image plane into the 2D coordinates in another plane. The 2D coordinates (x, y) of a point p are represented in *homogeneous coordinates*, stored in a 3D vector $p_h = (x_1, x_2, x_3)$ where $x = \frac{x_1}{x_3}$ and $y = \frac{x_2}{x_3}$. Using this kind of coordinates, homography transformations can be represented by a linear operation: a multiplication by H, a 3×3 real matrix. After the transformation, the original 2D coordinates can be retrieved dividing the first two coordinates for the third one.

$$p = \begin{bmatrix} x \\ y \end{bmatrix}, \quad p_h = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix}$$

$$p'_h = Hp_h = \begin{bmatrix} \omega x' \\ \omega y' \\ \omega \end{bmatrix} \quad \rightarrow \quad p' = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

Let $K_1^\star, K_2^\star$ be the sets of keypoints corresponding to the descriptors belonging to the filtered list of matches. Taking four points from each set we can find the 8-degree-of-freedom homography that relates those points.

In order to find the homography that relates correctly the most of the points in $K_1^\star$ and $K_2^\star$, RANSAC is applied [22, 23]. RANSAC (**RAN**dom **SA**mple **C**onsensus) is an non-deterministic algorithm to estimate parameters of a mathematical model from a set of observed data which contains outliers.

RANSAC algorithms iteratevly executes the following steps:

(a) takes 4 matches (couple of points) at random from $K_1^\star$ and $K_2^\star$,

(b) computes the homography H relating those points,

(c) counts the number of other matches that are correctly related by H (called **inliers**).

After a certain number of iterations, the matrix H which gave the maximum number of inliers is returned.

Filter By Homography Using the homography found by the RANSAC algorithm (Figure 4.3g), we can further filter the list of matching descriptors, keeping only the inliers of the perspective transformation (Figure 4.3f).

Not all the homographies returned by the RANSAC algorithm are good. Sometimes, quasi-degenerate or flipping homographies are reported and in most of the cases they are not representing a good match (see Figure 4.4).

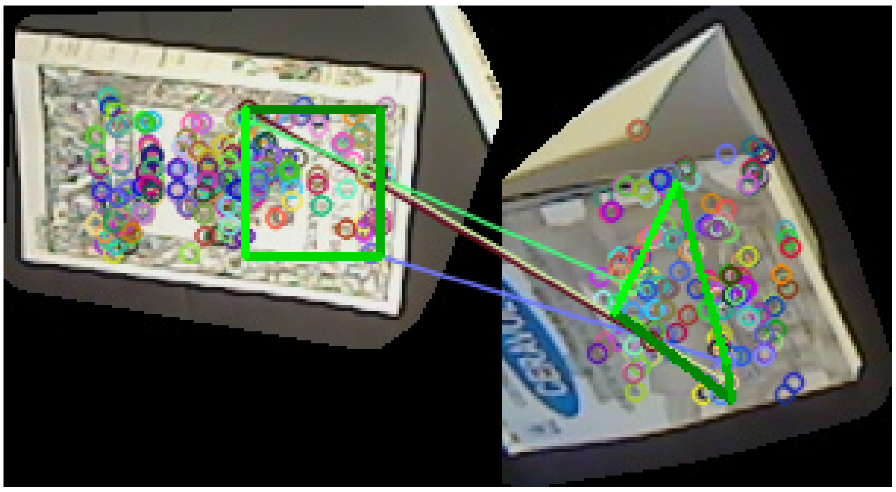


Figure 4.4

Ugly Homography found by RANSAC algorithm, usually indicating a bad match.

Most of these spourious results can be detected analizing the homography matrix.

Three checks are done:

- flipping homographies can be discarded checking if $det(H) < 0$.
- very skewed or prospective homographies can be discarded if $det(H)$ is too small or too big: given a parameter N, H is discarded if $det(H) > N$ or $det(H) < \frac{1}{N}$.
- homographies transforming the matching keypoints bounding box in a concave polygon can be filtered out with a convexity check.

In those cases, it is very unlikely that the samples under analysis are really related by this perspective transformation, therefore the system assumes there is no similarity between them and returns a similarity value of 0.

Second Stage RANSAC Some samples may pass the homography matrix check even if the perspective transform described by H is very unlikely to be observed. In order to filter out false positives homography matrices, the image of the first sample o_1 is transformed in $\hat{o_1}$ using the homography to be validated (Figure 4.3c) and the similarity computation steps are repeated considering the samples $\hat{o_1}$ and o_2. Features are re-detected and re-extracted from $\hat{o_1}$, matched with o_2 and a second RANSAC is executed to estimate a new homography $\hat{H}$ describing the prospective transformation among $\hat{o_1}$ and o_2 (Figure 4.3h-k). If the original samples o_1 and o_2 were really different views of the same object, $\hat{H}$ should be very near to the identity transformation (Figure 4.3k). If not the similarity between o_1 and o_2 is set to 0 and returned.

4.2.3 Similarity Output

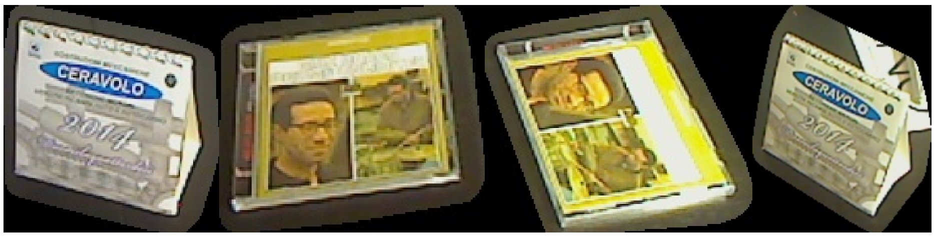
After the system found a good homography relating the samples, the ratios $\hat{r_1}, r_2$ among the number of inliers and the total number of detected features are computed for each sample:

$$\hat{r_1} = \frac{I}{|\hat{K_1}|}, \qquad r_2 = \frac{I}{|K_2|}$$

where I are the number of inliers of the homography estimated between samples $\hat{o_1}$ and o_2, $|\hat{K_1}|$ and $|K_2|$ are respectively the number of detected keypoints in $\hat{o_1}$ and in o_2. The similarity value among original samples under analysis $S(o_1, o_2)$ is defined as the harmonic mean between $\hat{r_1}$ and r_2:

$$S(o_1, o_2) = \frac{2}{\frac{1}{\hat{r_1}} + \frac{1}{r_2}}$$

Some similarity outputs among four object samples are reported in Figure 4.5.



	A	B	C	D
A	1	0	0	0.63
B	0	1	0.57	0
C	0	0.62	1	0
D	0.68	0	0	1

(e)

Figure 4.5

Values of similarity among object samples a-d reported in table (e).

Similar Objects List Generation Given the new object sample $\hat{o}$, a list L of similar object samples has to be generated by the system.

This step is completed in this stage scanning all object samples in the local database. For each object sample o_i the similarity value $s_i = S(\hat{o}, o_i)$ is computed and if it is above a similarity threshold T_s, o_i is inserted in L.

Chapter 5

Online Object Clustering

In this chapter the third stage of the PiCVi pipeline is described. The goals of this stage are a) to label the new object sample and b) to maintain clusters of object samples that potentially represent the same real 3D object.

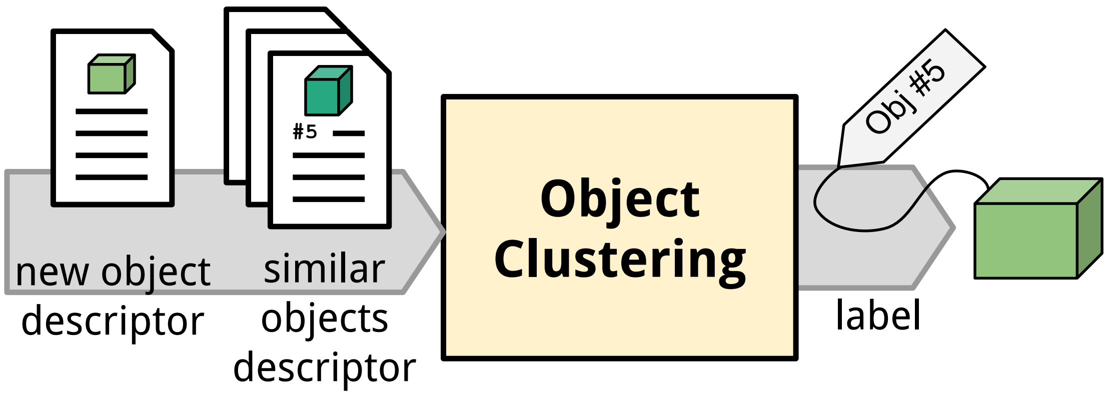
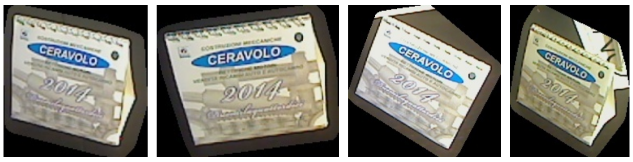


Figure 5.1
Third stage of the PiCVi pipeline, having object descriptors as input and a label as output.

Each cluster is identified by a *label* assigned to object samples. Each time a new object sample descriptor is created, the system tries to put it in a cluster relying on the similarity it has with other already clustered samples. To do so, the system uses the list of already clustered object samples similar to the new one, which has been already computed in the previous stage (see Figure 5.1).

The new object sample can bring informations useful to cluster reorganization: for example let c_1 and c_2 be two clusters of object samples representing the same real object but viewed from different poses. A new object sample representing the real object in an intermediate pose could suggest the system to merge c_1 and c_2 in a unique cluster (see Figure 5.3).


(a) table calendar

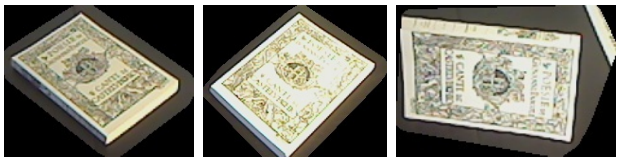
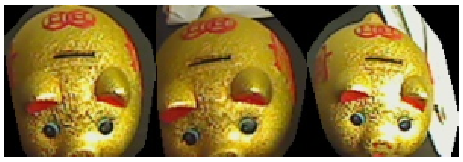
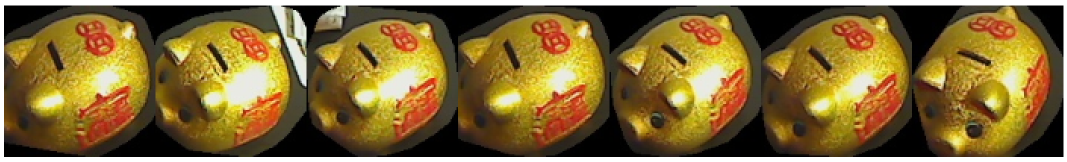

(b) poetry book

Figure 5.2

Example of two object sample clusters: (a) and (b).


(a) frontal view cluster


(b) side view cluster

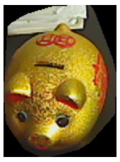

(c)

Figure 5.3

Example of **cluster merging***: cluster (a) and (b) group together respectively the frontal view and a side view of the object. The new sample (c) is similar to both clusters and can lead to a cluster merge.*

5.1 Semi-supervised Clustering Technique

When trying to label a new object sample, the following scenarios can occur:

1. the new sample does not match with old samples.
2. the new sample matches with one or more old samples all belonging to the same cluster.
3. the new sample matches with more than two samples beloging to different clusters.

In order to handle the above scenarios, a semi-supervised online clustering technique is adopted. Most of the times the system will be able to reorganize the new object together with the old ones, but sometimes a correct reorganization is difficult without user interaction.

5.1.1 Unsupervised Clustering

In the case of the first two scenarios, the incoming object can be labelled in an unsupervised way without messing up with the current clusters configuration.

In scenario 1, no samples similar to the new one exist, hence a new label is created an assigned to the sample. This is equivalent to create a new cluster containing only the new object. It may happen that a new sample representing an object already present in the database is not similar (in terms of *similarity value*) to the other samples. In this case the new sample is incorrectly put in a new cluster instead of being grouped with the other samples representing the same object, but this situation can be recovered in the future performing a **cluster merge** operation (see Subsection 5.1.2) if new samples of the object will be collected by the system.

In scenario 2, the new sample is similar to samples belonging to a same cluster, hence is a good candidate for that cluster. Hence, the system assigns the corresponding cluster label to the new sample.

In general, expanding a cluster with a new sample is a simple operation, since only the label of the new samples has to be set. On the contrary, splitting clusters is a difficult operation that requires many similarity evaluations and database accesses. In order to maintain a good performance and to be able to do clustering online (each time a new sample is collected), the system does not perform cluster splitting, hence clusters are expanded only if there is a strong similarity among samples. This is obtained designing the similarity function (see Section 4.2) in such way it is very unlikely that samples which are not visually similar have a high similarity value.

5.1.2 Supervised Clustering

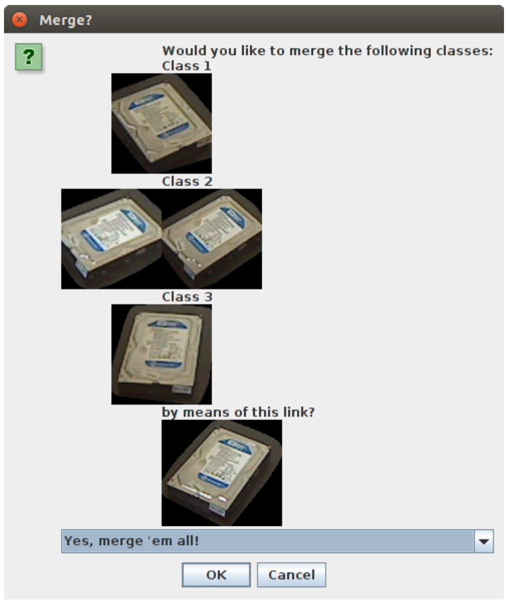
In scenario 3, the new object is similar to members of different clusters. Many actions may be taken by the system in this situation:

1. the clusters containing the samples similar to the new one are merged together in a unique bigger cluster to which the new object will belong (**cluster merging**, Figure 5.3).

2. the new sample is inserted into only one among the candidates clusters.

3. a new cluster is created containing only the new object.

The **cluster merging** operation is the way the system aggregates multiple views of a single object into a single truth set, creating a base knowledge for the object recognition. It has to be performed carefully since no cluster splitting is available. Using a single camera with no a priori knowledge about the objects the user want to recognize, it is difficult to decide when to perform cluster merging rather than the other actions listed above.



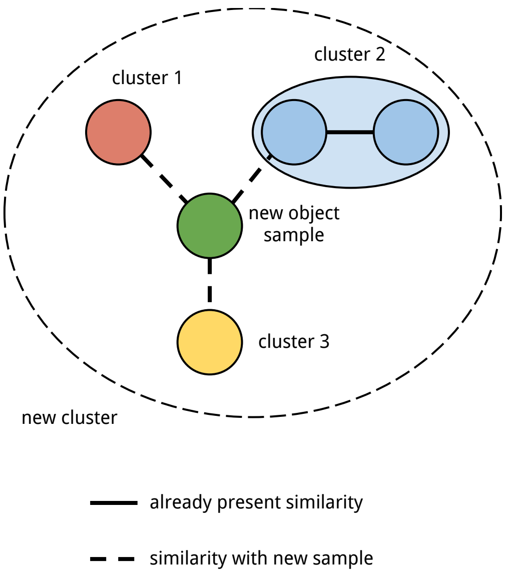


Figure 5.4
On the left, the supervised clustering dialog: the system asks the user what to do when three clusters are similar to each other. On the right, a visual representation of similarities between samples affected by the merge.

Up to now, the system does not decide automatically and asks the user which action should be taken (Figure 5.4). This may lead to many queries to the user that may rapidly get bored. Interaction between multiple cameras and similarity values between samples and clusters may be exploited to take the correct action automatically. Possible solution are briefly discussed in Subsection 6.2.1 and are left to future work.

Chapter 6

Experiments and Results

In this chapter, the experiments conducted to evaluate the system are described and results are reported. Conclusions of this work and are summarized and some directions for future work are discussed.

6.1 Experiments and Results

No labelled datasets are available for evaluating the whole PiCVi pipeline, moreover the first stage is loosely coupled with the following stages and suited to be tested separately too. Hence two experiments, described in the following subsections, were done for evaluating a) object detection (done through the foreground extraction taks) and b) object recognition (done through description, matching, labelling and clustering of object samples).

6.1.1 Evaluation of Foreground Extraction

For the evaluation of the foreground extraction task accomplished by the first pipeline stage of PiCVi the following publicly available videos were used:

- Street (Fifth Configuration) Video and
- Rotary (Fifth Configuration) Video, both taken from the BMC[1] dataset [24],
- Sofa Video, taken from the *Intermittent Object Motion* category of the CDW-2012[2] dataset [25].

[1]Background Models Challenge: http://bmc.univ-bpclermont.fr/
[2]Change Detection Workshop: http://changedetection.net/

The first two are synthetic videos presenting two outdoor scenes. Camera acquisition noise, illumination changes and background movements are artificially added emulating a windy and cloudy environment. The last one is a real video of an indoor scene more representative of the application scenario, where some objects are added, moved and removed from the scene by human actors. Each frame of each video comes with a groundtruth mask indicating which is the foreground part of it.

The three foreground extraction algorithms presented in Chapter 3 (Frame Differencing, Subsection 3.1.1; Mixture Of Gaussians, Subsection 3.1.2 and IP-based, Section 3.2) are evaluated. Each algorithm is applied to each video: since we are interested in extracting images of objects rich of local features in order to match them later, at each frame FAST keypoints are extracted and classified as foreground or background keypoint based on the foreground mask obtained by the algorithm. The correct classification is given by the groundtruth foreground mask (Figure 6.1).

Measures related to binary classification problems are extracted. The positive class represents foreground points and the negative background ones. The following measures are extracted for each algorithm and reported in Figure 6.2:

Precision

$$p = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}},$$

where TP is the number of correctly classified foreground points and FP is the number of background points incorrectly classified as foreground. This measure represents the fraction of keypoints classified as foreground that are really foreground keypoints.

Recall

$$r = \frac{\mathrm{TP}}{\mathrm{P}},$$

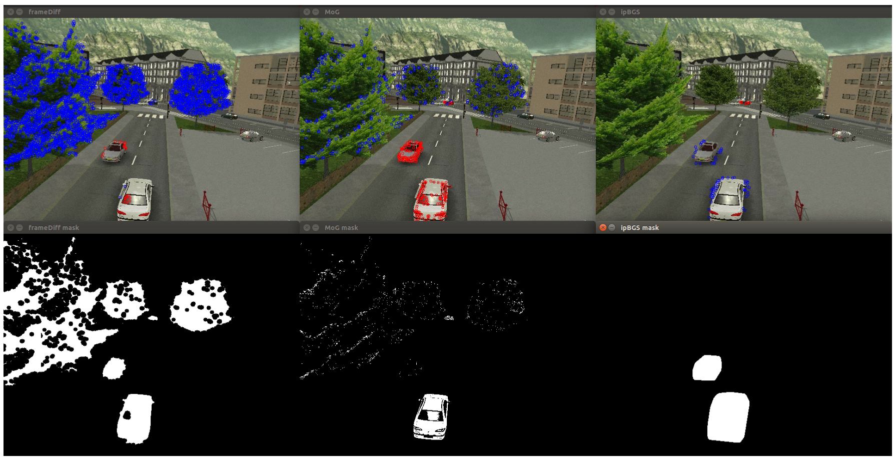
where P is the number of foreground keypoints. This measure represents the fraction of all foreground keypoints correctly classified as foreground points.

F_1 **Score**

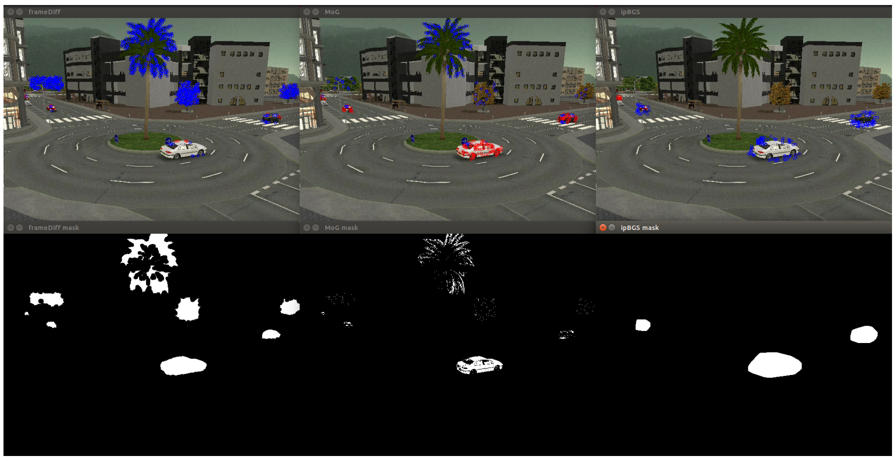
$$F_1 = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

which is the harmonic mean of precision and recall.

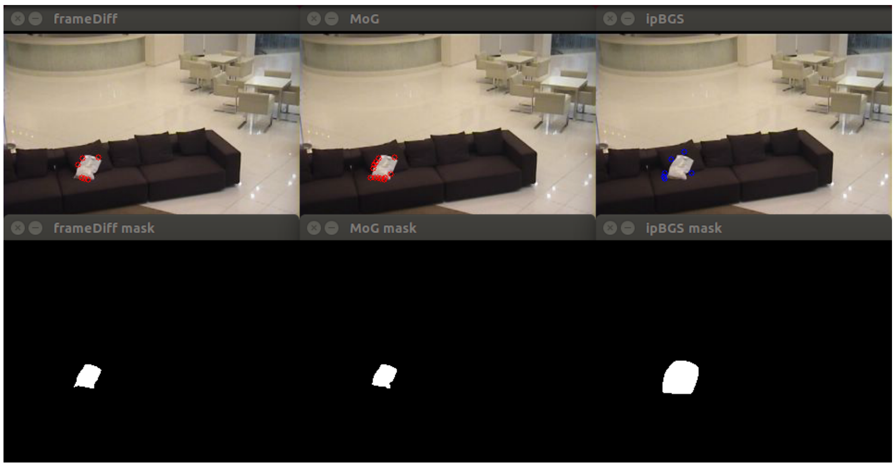
The implemented method has higher recalls than other tested methods, since less foreground points are left behind. In some tests, precision may be lower due to false positives, but for the application it is more important to have a high recall, since spourious results can be filtered out in further processing steps.



(a) street



(b) rotary



(c) sofa

Figure 6.1

Foreground Extraction Test Scenes: an example frame of each test scene and the foreground mask generated by each algorithm are shown. Red and blue circles drawn on the frame shows respectively misclassified foreground and background keypoints.

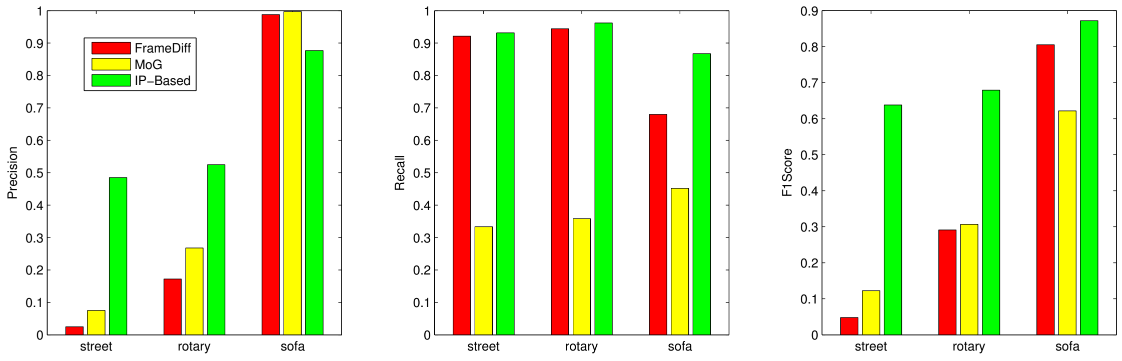


Figure 6.2

Foreground Extraction Evaluation Metrics: the figure plots Precision, Recall and F1 Score obtained by each algorithm tested in each video sequence.

6.1.2 Evaluation of Object Recognition

In order to evaluate the second and the third stage of the PiCVi pipeline, the publicly available *3D Objects* dataset [26] has been used. This dataset is composed by images of 10 object categories. For each category, 9-10 objects are present and for each object, several images are reported in which the specific object is shown in different poses. Each image comes with a foreground mask which denotes exactly in which part of the image the object is located. Images are taken from 8 different angles using 3 different scales and 3 different heigths for the camera, leading to around 5500 labelled images of 100 specific objects (see Table 6.1).

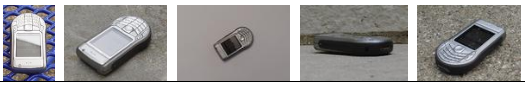
Class	**Object**	**Samples**
cellphone	cellphone_1	. . .
	. . .	. . .
	cellphone_9	 . . .
mouse	mouse_1	. . .
	. . .	. . .
toaster	toaster_1	. . .
	. . .	. . .

Table 6.1

Excerpt from the Stanford "3D Objects" dataset: only some samples of some objects of some classes are reported.

The presented system autonomously groups objects into clusters without knowing their labels, but cannot recognize them before the user labels at least some of them, hence the system cannot be compared with traditional trained classifiers. Instead the ability of the system to build good and easy to label clusters is measured.

Let $O = \{(o_1, l_1), (o_2, l_2), \ldots\}$ the set of labelled samples. The entire dataset O is randomly shuffled and splitted in training set O_{train} (90%) and testing set O_{test} (10%): training samples are presented to PiCVi as coming from the output of the foreground extraction stage. The system builds clusters of samples while they are processed. In the case a supervised clustering is needed, the test code simulates the user interaction choosing the best action to be taken knowing the labels of the samples involved. The complete clustering test code is reported in Algorithm 1.

Algorithm 1 Clustering algorithm simulating user interaction used during tests

```
for all $(o_i, l_i) \in O_{train}$ do
    $O_s = \{o_j \in D : S(o_i, o_j) > T_s\}$          ▷ $O_s$ is the set of samples similar to $o_i$
    if $|O_s| = 0$ then
        $newC \leftarrow newC + 1$
        $c_i \leftarrow newC$
    else if $|O_s| = 1$ then
        $k \leftarrow k : o_k \in O_s$
        $c_i \leftarrow c_k$
    else
        $C_s = \{c_j : o_j \in O_s, \forall(o_j, c_j) \in D\}$ ▷ $C_s$ is the set of all cluster IDs to which each
                                                  sample in $O_s$ belongs
        if $|C_s| = 1$ then
            $k \leftarrow k : c_k \in C_s$
            $c_i \leftarrow c_k$
        else
            $L_s \leftarrow \textsc{majorLabels}(C_s)$          ▷ $L_s$ is the set of the major labels for each clus-
                                                  ter in $C_s$
            $CI \leftarrow \{c : l = l_i, \forall(c, l) \in L_s\}$ ▷ $CI$ is the set of cluster IDs having major label
                                                  equal to $l_i$
            $OI \leftarrow \{j : c_j \in CI\}$          ▷ $OI$ is the set indices of objects belonging to
                                                  one of the clusters in $CI$
            $newC \leftarrow newC + 1$
            for all $k \in OI$ do
                $c_k \leftarrow newC$
            end for
            $c_i \leftarrow newC$
        end if
    end if
    $D \leftarrow D \cup \{(o_i, c_i)\}$
end for
```

Once the clusters are built, they must be labelled to produce a labelled training set. Since the user usually does not want to waste time in cleaning clusters or label singular objects, the test code simulates a labelling technique based on *major voting*: an entire cluster is labelled with the label of the most frequent object present in it.

The training set thus labelled is used for training a k-NN classifier. The *cluster* k-NN classifier finds the k most similar samples (the ones with the higher value of similarity S) and assigns a score for each label of those samples. The winning label is assigned to the

Algorithm 2 Subroutine that finds major labels of a set of clusters

```
function MAJORLABELS($D, C_s$)
    $L_s \leftarrow \emptyset$
    for all $c \in C_s$ do
        $L_c \leftarrow \{(i, l_i) : (o_i, c) \in D\}$    ▷ $L_c$ is the set of all couples (sample, label) belonging to the cluster $c$
        $uL_c \leftarrow \{l_i : (i, l_i) \in L_c\}$    ▷ $uL_c$ is the set of labels present in cluster $c$
        $C \leftarrow \emptyset$    ▷ $C$ is the set of (label, count) couples
        for all $l \in uL_c$ do
            $L_{c,l} \leftarrow \{(i, l_i) \in L_c : l_i = l\}$    ▷ $L_{c,l}$ is the set of all couples (sample, label) belonging to the cluster $c$ having label $l$
            $C \leftarrow C \cup \{(l, |L_{c,l}|)\}$
        end for
        $majorL \leftarrow \text{argmax}_{\hat{l}:(\hat{l},\hat{n}) \in C}\{n : \forall (l, n) \in C, l = \hat{l}\}$
        $L_s \leftarrow L_s \cup \{(c, majorL)\}$
    end for
    return $L_s$
end function
```

processed test sample. Another k-NN classifier is trained using the training set with correct labels and another labelling of the test set is generated in the same way.

Test set labellings are evaluated with the same metrics used in Subsection 6.1.1: precision, recall and F-score are computed for each class and then are aggregated using macro- and micro-averaging techniques. Let n be the number of object classes, TP_i the true positives, FP_i the false positives, P_i the total number of samples, p_i the precision, r_i the recall and F_i the F1-score of class i, then:

Micro-averaged precision

$$p_{micro} = \frac{\sum_{i=1}^{n} \mathrm{TP}_i}{\sum_{i=1}^{n} (\mathrm{TP}_i + \mathrm{FP}_i)},$$

Micro-averaged recall

$$r_{micro} = \frac{\sum_{i=1}^{n} \mathrm{TP}_i}{\sum_{i=1}^{n} \mathrm{P}_i},$$

Micro-averaged F-score

$$F_{micro} = \frac{2 p_{micro} r_{micro}}{p_{micro} + r_{micro}},$$

Macro-averaged precision

$$p_{macro} = \frac{\sum_{i=1}^{n} p_i}{n},$$

Macro-averaged recall

$$r_{macro} = \frac{\sum_{i=1}^{n} r_i}{n},$$

Macro-averaged F-score

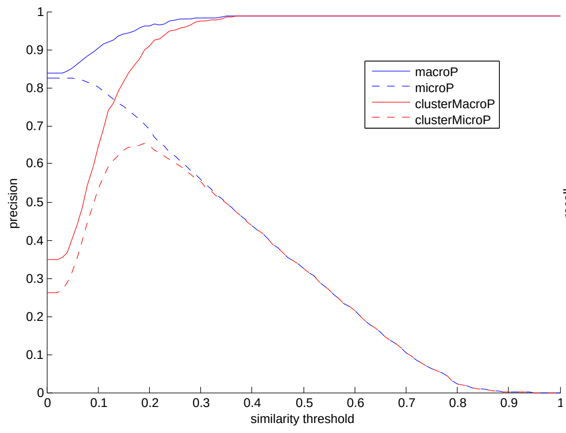
$$F_{macro} = \frac{2p_{macro}r_{macro}}{p_{macro} + r_{macro}}$$

Macro-averaged metrics tends to give the same weight to each class, while micro-averages metrics takes into account possible biases introduced by each class and gives a more accurate global performance index. Another measured metric is the number of interactions the system must have with the user in order to label the training set: the *groundtruth* k-NN classifier needs the user to label each training sample individually, which corresponds to a number of query to the user equal to the number of samples in the training set. The *cluster* k-NN classifier needs to interact with the user a) when a cluster merging can not be resolved automatically during the online clustering and b) when a cluster has to be labelled.

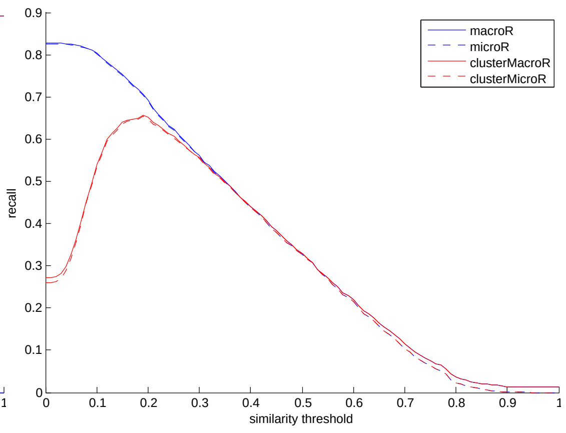
In Figure 6.3, the performances of the two classifiers for various similarity thresholds T_s are reported. It can be seen that for T_s around 0.2, the *cluster* k-NN classifier has almost the same performances of the *groundtruth* k-NN classifier, having only around half the interactions with the user. However, performance degradation of the *cluster* k-NN classifier is due to the fact that we simulated a unique user interaction after the training phase which used *major voting* paradigm to label all clusters at once. Since the system is incrementally building richer and richer clusters, this is not the best way to interact with the user asking labels: user interaction may be proactively requested only when big homogeneous clusters are involved, maximizing the amount of information collected. Moreover, smarter techniques than major voting may be implemented to simulate a more precise user labelling session. In the performed tests, many singleton or small clusters are present at the end of the training phase, raising the number of queries to the user needed to label the entire training set.

6.2 Conclusions

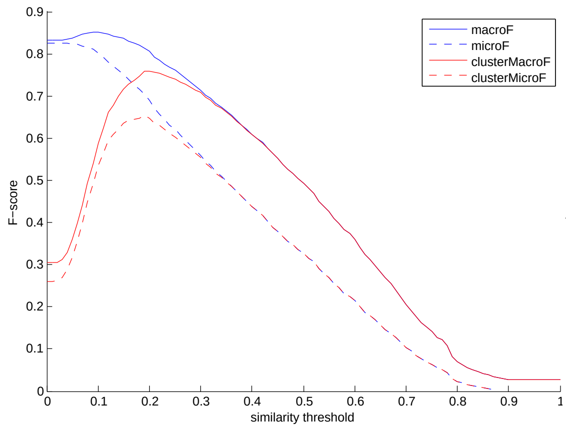
In this work, a system for incremental real-time object detection and automonous recognition in indoor environments have been presented. The system detects movable foreground objects relying on FAST interest points. Once the object has been segmented, the system relies on ORB features to create its descriptor, store it and compare it with descriptors of previously seen objects. A visual similarity function of two samples has been defined relying on ORB features matching and geometry consistency checking through the use of homographies. The system groups together similar objects into clusters relying on the transitivity of similarity among objects. Each cluster identifies a class and the system learn to autonomously recognize an object assessing cluster membership. No publicly benchmarks have been found for evaluating the entire processing pipeline. Hence, experiments were done in order to asses:



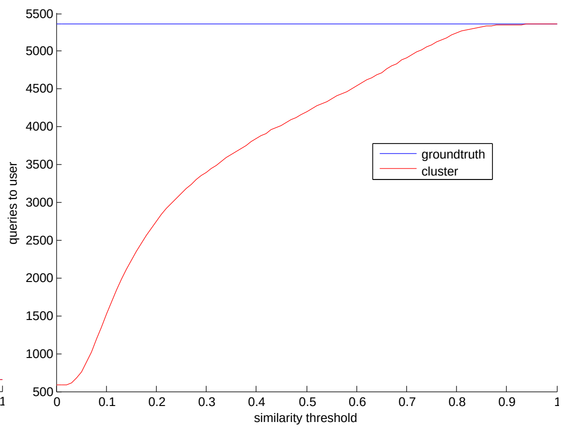
(a) Micro- and macro-averaged precision values when varying the similarity threshold



(b) Micro- and macro-averaged recall values when varying the similarity threshold



(c) Micro- and macro-averaged F1-score values when varying the similarity threshold



(d) The number of queries to the user when varying the similarity threshold

Figure 6.3

Comparison of the perfromances of the recognition task, solved by a k-NN classifier trained with the groundtruth training set (blue lines) and by a k-NN classifier with training set made by cluster labelling (red lines).

- the ability of the system to detect foreground objects in an active background and to extract patches of the video frame containing their interest points. The discussed methods for foreground extraction has been tested with labelled datasets built for background subtraction evaluation. During experiments it has been noticed that color-based background subtraction method typically used in computer vision may omit informative part or include a useless part of the image, hence a interest point-based method has been implemented.

- the ability of the system to group together unlabelled samples, reducing the labelling work of the user. A dataset of 5500 images has been processed and it has been point out that in a k-NN classification task based on similarity as distance the presented labelling strategy requires about half of the user interaction without degrading too much the classification results with respect to the groundtruth k-NN classifier.

6.2.1 Future Work

Future work will explore two main aspects: a) the enhancement of the implemented methods for object detection, description, clustering, labelling and recognition in the scope of a single camera, b) the design of the distributed multi-camera system, the implementation of the camera network protocol and of the tasks enabled by inter-camera communication. For the former, some practical solutions are described in the following paragraphs. For the latter, possible adoptable solutions for the complete system are suggested.

Exploit similarity values Similarity values has been defined to belong to the interval $[0, 1]$ and a single threshold T_s is used to decide whether the compared samples are similar or not. Multiple similarity thresholds may be adopted to define different level of confidence of the similarity between two samples. Levels of confidences can be exploited to automatically take some decisions during the clustering of samples and limit the user interaction.

Proactive user interaction Up to now, the user is queried for each decision the system could not take autonomously during the clustering of samples. This may lead to query the user too often. Unresolved decision should be stored and the user queried in a proactive way only when a considerably amount of samples can be labelled with little effort. In this way the ratio between the collected information and the number of queries to the user is maximized. An example of this behaviour can be observed in bulk face tagging in multiple photos in popular social networks, where the user is prompted to tag only substantial groups of similar faces at once.

Enhance object detection and description In this work, detection and description of objects is mainly based on local features of images, describing the statistics pixel values of the grayscale channel around relevant points of the image such as corners and edges. Other types of features may be integrated to enhance each task:

object detection the implemented foreground extraction method may be combined with color-based methods (such as Mixture of Gaussians Background Subtraction [14, 16]) to increase the confidence of the presence of a foreground object and to produce a sharper contour of the extracted object. This may lead to the inclusion of less noise in the image to be processed in the following steps.

object description color-based features (such as color histograms, dominant colors etc.) and shape informations may be included to obtain a better object sample description: coherently the similarity function defined between two object samples shuold be updated to consider the comparison of those elements as part of the final similarity score. This may increase the rate of correct matches among samples.

Many samples collected from the same camera during time may be very similar to each other and introduce redundant information. Moreover, handling many samples leads to storage and computational issues. Hence, a technique for sample aggregation or deletion may be implemented in the scope of a single camera. In the case of a network of cameras, sample redistribution and query may be implemented in order to equally distribute the load among cameras.

Local Optimizations: Metric-space Indexing and Hardware Exploitation For each object sample under analysis, the list of similar objects is generated by the system scanning the local database. The problem of this approach is that the execution time is linear with the number of samples present in the database. Even if most of similarity values are equal to zero and its computation has fast rejection of non matching couples, retrieving similar objects may become too computational demanding when too many samples are present in the database: the limited resources of a smart camera will fastly limit the amount of samples it can handle.

Metric-space indices (such as M-trees [27, Chapter 3], Locality Sensitive Hashing etc.) may be added in order to efficiently retrieve similar samples from the local database: an intermediate distance between object descriptors or the similarity value itself may be used as metric for the index. Doing so, a single smart camera may handle more samples without degrading its performaces during similarity searches.

In the special case of the Raspberry Pi computing platform, no OpenCV interfaces are available to directly exploit the equipped GPU (VideoCore IV). Specific interfaces may be developed in order to to accelerate computer vision algorithms and save CPU time for other application tasks, such as network communications.

Multi-camera enabled tasks In this work, the system is limited to a single camera. Building a network of cameras may increase available space and computational resources and enables new type of tasks. The confidence level of object detection and recognition may be increased through inter-camera communication and cross-checking, enabling the system to interact fewer times with the user. Object tracking with handover can be implemented. If cameras have overlapping field of views automatic camera calibration and object localization can be done exploiting point triangulation. The complete distributed network may be managed and queried by smartphone or other user-friendly interfaces.

Bibliography

[1] Juan Carlos Augusto. Artificial intelligence in challenging environments. In Alfons J Schuster, editor, *Intelligent computing everywhere*. Springer, 2007.

[2] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[3] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006*, pages 404–417. Springer, 2006.

[5] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[6] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *Computer Vision–ECCV 2010*, pages 778–792. Springer, 2010.

[7] Ching-Hui Chen, Julien Favre, Gregorij Kurillo, Thomas P Andriacchi, Ruzena Bajcsy, and Rama Chellappa. Camera networks for healthcare, teleimmersion, and surveillance. *Computer*, 47(5):26–36, 2014.

[8] Kevin Abas, Caio Porto, and Katia Obraczka. Smart camera networks for the surveillance of public spaces. *Computer*, 2014.

[9] Nikhil Naikal, Allen Y Yang, and S Shankar Sastry. Towards an efficient distributed object recognition system in wireless smart camera networks. In *Information Fusion (FUSION), 2010 13th Conference on*, pages 1–8. IEEE, 2010.

[10] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[11] Rita Cucchiara, Costantino Grana, Massimo Piccardi, and Andrea Prati. Detecting moving objects, ghosts, and shadows in video streams. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(10):1337–1342, 2003.

[12] Kinjal A Joshi and Darshak G Thakore. A survey on moving object detection and tracking in video surveillance system. *International Journal of Soft Computing and Engineering*, 2(3):44–48, 2012.

[13] OpenCV Developer Team. Opencv documentation: How to use background subtraction methods. URL http://docs.opencv.org/trunk/doc/tutorials/video/background_subtraction/background_subtraction.html.

[14] Thierry Bouwmans, Fida El Baf, and Bertrand Vachon. Background modeling using mixture of gaussians for foreground detection-a survey. *Recent Patents on Computer Science*, 1(3):219–237, 2008.

[15] Jeff A Bilmes et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998.

[16] Pakorn KaewTraKulPong and Richard Bowden. An improved adaptive background mixture model for real-time tracking with shadow detection. In *Video-Based Surveillance Systems*, pages 135–144. Springer, 2002.

[17] Dehghani A. and Sutherland A. A novel interest-point-based background subtraction algorithm. *ELCVIA*, 13(1):Gowri Srinivasa, 2014.

[18] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006*, pages 430–443. Springer, 2006.

[19] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[20] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pages I–900. IEEE, 2002.

[21] Shengcai Liao, Xiangxin Zhu, Zhen Lei, Lun Zhang, and Stan Z Li. Learning multi-scale block local binary patterns for face recognition. In *Advances in Biometrics*, pages 828–837. Springer, 2007.

[22] Elan Dubrofsky. *Homography estimation*. PhD thesis, UNIVERSITY OF BRITISH COLUMBIA, 2009.

[23] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[24] Antoine Vacavant, Thierry Chateau, Alexis Wilhelm, and Laurent Lequièvre. A benchmark dataset for outdoor foreground/background extraction. In *Computer Vision-ACCV 2012 Workshops*, pages 291–300. Springer, 2013.

[25] Nil Goyette, Pierre-Marc Jodoin, Fatih Porikli, Janusz Konrad, and Prakash Ishwar. Changedetection. net: A new change detection benchmark dataset. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 1–8. IEEE, 2012.

[26] Silvio Savarese and Fei-Fei Li. 3d generic object categorization, localization and pose estimation. In *ICCV*, pages 1–8, 2007.

[27] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity search: the metric space approach*, volume 32. Springer, 2006.