Università di Padova  Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

24 Ottobre 2011

People Detection and Tracking with Kinect for Mobile Platforms

Supervisor: Pagello Enrico
Cosupervisor: Jensfelt Patric
(KTH - Stockholm)

Student: Campana Riccardo
621996-IF

Academic Year 2010/2011

Abstract

Human detection is a key ability for robot applications that operate in environments where people are present, or in situation where those applications are requested to interact with them. It's the case for social robots like aids for the rehabilitation of inmates in hospitals, assistance in office, guides for museum tours.
In this thesis we will investigate on how we can make use of the new Microsoft's gaming sensor, the Kinect, to address the issues of real-time people detection and tracking, since the sensor has been built in order to detect people and track their movements.
We developed a system that is able of detecting and tracking people in near real-time both on fixed environments and mobile platforms. The system has been developed for an indoor environment and it is composed by three modules that are:

1. segmentation module: looks for values of depth to use as initial centroids for a k-means algorithm in order to label the depth map retrieved by the Kinect;
2. features computation module: computes normalized depth candidates and obtains the likelihood of them being people, using a vector of Relational Depth Similarities (RDSFs) and a Real AdaBoost trained classifier;
3. features tracking module: adds or updates the tracking features for the detected people, that are color histogram, present position and distance covered.

We tested four different classifiers on different situations. The best classifier showed very good detection and tracking results whereas, because of some segmentation problems, the performances of the complete system have been subjected to a lowering with respect to the theoretical ones. We developed also a method for getting rid of some of these segmentation problems and it showed some improvements for the complete system together with some drawbacks that affected the theoretical results. However the complete system works good and with a frame rate of 2 fps on average. Most of the computational load is due again to the segmentation module, so an improvement of this module would lead to both improvements on the real-time performances and on the detection results.

Sommario

La capacità di riconoscere persone é un'abilità chiave per robot che operano in ambienti popolati da esseri umani, o in quelle situazioni in cui tali applicazioni richiedono di interagire con le persone stesse. È il caso di robot a uso sociale come aiuto ai pazienti negli ospedali, assistenti d'ufficio, guide turistiche nei musei.
In questa tesi s'investigherà su come far uso del nuovo sensore giochi della Microsoft, il Kinect, per risolvere le problematiche di riconoscimento e inseguimento di persone in tempo reale, poiché tale sensore è stato costruito appositamente per riconoscere persone e tracciare i loro movimenti.
Abbiamo sviluppato un sistema capace di tracciare e inseguire persone in tempistiche vicine a quelle real-time in grado di funzionare sia in scenari statici che su piattaforme mobili. Il sistema è stato sviluppato per ambienti interni ed è composto da tre moduli, vale a dire:

1. modulo di segmentazione: cerca dei valori di profondità da usare come centroidi iniziali per un algoritmo k-means al fine di ottenere il labeling della mappa di profondità ottenuta dal Kinect;
2. modulo di calcolo delle features: calcola dei candidati di profondità normalizzati e ottiene la verosimiglianza degli stessi di essere persone, usando un vettore di Relational Depth Similarities (RDSFs) e un calssificatore allenato tramite Real AdaBoost;
3. modulo per le features di inseguimento: aggiunde o aggiorna le features di inseguimento per le persone riconosciute, vale a dire istogramma di colore, posizione attuale e distanza percorsa.

Abbiamo testato quattro diversi classificatori in situazioni differenti. Il miglior classificatore ha dimostrato risultati molto buoni di riconoscimento mentre, a causa di alcuni problemi di segmentazione, le prestazioni del sistema completo sono state soggette a un abbassamento rispetto a quelle teoriche. Abbiamo sviluppato anche un metodo per evitare alcuni di questi problemi di segmentazione e tale metodo ha mostrato alcuni miglioramenti per il sistema completo insieme ad alcuni svantaggi che hanno influito sui risultati teorici. Tuttavia, il sistema completo funziona bene e con un frame rate medio di 2 fps. La maggior parte del carico computazionale è dovuto ancora una volta al modulo di segmentazione, di conseguenza un miglioramento di questo modulo porterebbe sia a miglioramenti nelle prestazioni in tempo reale sia nei risultati di riconoscimento.

Contents

Chapter 1

Introduction

Human detection is a key ability for robot applications that operate in environments where people are present, or in situation where those applications are requested to interact with them. It's the case for social robots like aids for the rehabilitation of inmates in hospitals, assistance in office, guides for museum tours.
This problem has been studied quite a lot over the years providing partial solutions that rely on laser scanner and cameras. When dealing with static scenarios, systems that use background subtraction methods are very common. They build a background model of the environment and then by comparing the actual data from the sensors with the model, they look for differences to achieve candidates for the human detection. While this technique yields to good results and low computation load, it cannot be applied in mobile robot applications.
Present solutions rely on face detection algorithms, laser based methods or color skin recognition. The first one has the major issue that the person has to face the robot, the second one has the limitation of exploring the space just on 2D, while the last one can be affected by illumination problems and false candidates. Others applications instead use combination of techniques such as face detection with color segmentation and torso recognition. This because robot applications should often be able not only to detect people, but also to track them and sometimes possibly follow them.
In this thesis we will investigate on how we can make use of the new Microsoft's gaming sensor, the Kinect, to address this problem, since it has been built in order to detect people and track their movements in real-time.
Let's now clarify more precisely the background of human detection. As already introduced, when dealing with the human detection problem we should usually make a distinction between two categories:

1. static sensors;
2. sensors mounted on a mobile platform.

1.1 Previous Work: Static Scenarios

For static scenarios one of the most used method relies on the background subtraction principle. The first step is to create a background model of the environment whereas the second step consists in processing the new data with the model in order to detect people. This procedure can be done in various ways [1].

1.1.1 Gaussian Distribution

Wren et al [2] create a background model by mean of a gaussian distribution on the YUV space at each pixel. This model is continuously updated and the detection is achieved by modeling the human being with multiple blobs with spatial and color components. The parameters of these lasts are estimated by a Kalman filter. Each pixel will be given then a likelihood value of being part either of the background or of the blob.

1.1.2 Intensity Difference

Beleznai [3] instead considers the intensity difference between an input frame and a reference one as a multi-modal probability distribution. Human detection is then achieved by using mean shift computation.

1.1.3 Region Based Model

Another approach was proposed by Eng et al [40]. They built a region based background model of the environment based on the hypothesis that each region has a multi-variate gaussian probability distribution over the colors. In order to build this model, background frames are separated into blocks by means of a k-means algorithm.

1.1.4 Illumination Invariance

The method proposed by Toth and Aach [4] performs illumination invariant background subtraction by using frame differencing, window based sum of absolute differences aggregation, and an adaptive threshold. The method leads to foreground shapes that are then clustered in blobs by mean of a connected components technique. A Fourier transform is then used to describe the shape. This descriptors are used then for the classification process for obtaining the likelihood of a blob being a person. The classifier is represented by a feedforward neural network.

1.1.5 Shape Based Classification

Lee et al [5] combined a background subtraction method with a shape categorization approach. This is achieved by subtracting each new frame with the model

of the environment. On the resulting blobs a contour processing is then applied. The contour is modeled as a polygon approximation expressed as a bend angle in comparison with a normalized length. For each contour achieved then, a similarity measure between the various categories is computed in order to obtain the likelihood of each candidate belonging to one of those categories.

1.1.6 RGBD Data Segmentation

A different approach is the one of Xu and Fujimura [6] also for the type of sensor used. In fact they utilize a device that is able of retrieving depth values together with the RGB ones of the usual image. A range of depth is considered in order to get rid of the background areas. A split and merge algorithm is used then to perform segmentation by depth slicing. At this point foreground objects and people are segmented and, in order to really detect the people between all the blobs, a torso detection is achieved by an ellipse fitting. An heuristic based on the movement is finally used to make the last adjustments.

1.1.7 IR Technique

Another interesting work is the one of Han and Bhanu [7]. They proposed to use an infrared camera together with a standard one. A background subtraction technique is used separately for the two cameras by means of a gaussian probability distributions method. The separated foreground candidates are registered using a hierarchical genetic algorithm and then merged together. Note that the IR camera highly reduces the number of candidates since it detects just object with a thermal signature.
Also Jiang et al [8] developed an approach based on this sensor fusion, but based on relative pixel saliences in the two images.

1.2 Previous Work: Mobile Platforms

None of the background subtraction methods works in situations where the camera or the sensor is moving. In fact in this scenarios the background is continuously changing and having a model results in being almost useless. The methods that deal with mobile scenarios usually involve four steps:

1. preprocessing: the input data have to be prepared in order to be used on the later computation. It is the case of, for example, stereo rectification when using a stereo camera for obtaining depth informations or of color balancing for getting rid of illumination condition in skin detection techniques;

2. segmentation: it is the process of detecting the single objects starting from the global image. This can be done by mean of various techniques like color skin segmentation for detecting people [9], depth or stereo vision segmentation [10], sliding window scan;

3. feature extraction: it is the process of extracting relevant information from the segmented data. Example of features that can be used are edge detector [11], SIFT [12], RDSF [13];

4. classification: the features extracted for each segmented candidate are like descriptors that can be used in order to find the likelihood of the candidate itself belonging to a specific category. Each set of features is given to a classifier. This classifier can be binary or multi-categorical. The first one is capable of determining whether a candidate belongs to a category or not. The multi-categorical one instead returns, from a set of categories, which one is more likely to describe the candidate. There are different options and the most used ones are machine learning methods [14] like support vector machines, AdaBoost and neural networks.

1.2.1 2D Range Data Detection

One of the approaches on a mobile system consists in using boosted features based on two dimensional range scans.
These sensors provide a large field of view and, more important, they work independently from the environment. As a drawback the scans provide little information about people. In fact in cluttered spaces is difficult to do a detection even for people. Luckily for these lasts, it is possible to find some particular geometrical properties that can be used as features for a supervised learning. Arras et al [15] for example proposed an approach that uses AdaBoost both for finding the best features and thresholds to use, and, as usual, to create the classifier. The most popular methods rely on extracting legs by detecting moving blobs that appears as local minima in the range image ([16], [17], [18], [19]) by using motion and geometry features. They demonstrated good results in simple environments but not on cluttered ones.
Another method consists in finding non-static objects in the environment by looking for space that was previously free and now it is occupied [20]. The basic principle is similar to the occupancy grid one but without the grid itself. The method presents low computational load and robustness even on a cluttered environment.

1.2.2 Face Detection

The face detection problem has been studied quite a lot in the past. The face, in fact, represents a very good feature because of its characteristics: it has a low degree of variability in contrast with a high level of texture and, in conjunction with a distinctive color, make it easier to be used for differentiation from other objects.
The first approaches consisted in rely just on the skin color detection as indicator for faces ([21], [22], [23]). However this method is very limited and too sensitive to illumination conditions that yield to a great number of false positive especially in environment with wooden furniture.
For this reason Schlegel et al [24] introduced a facial contour recognition together with the skin color detection. The greatest drawback of their work is the fact that

the system has to be initialized in order to detect a person. So each person in the environment has to be introduced first.
Finally Viola and Jones [25] realized a fast frontal face detection system which will be used also in later researches.

1.2.3 HOG Features

One of the most used feature based detection is the Histogram of Oriented Gradients (HOG) one.
Dalal and Triggs [12] showed how to use these kind of feature in order to build a classifier. Intensity gradients and edge directions appeared, in fact, to be very discriminative features. The process consists in dividing the image in cells, computing the histogram of the edge orientations over all the pixels of the cell itself. Finally a Support Vector Machine was trained in order to classify the candidate for human detection.
Similarly Wang and Lien [26] developed a HOG based detection system which is able also to detect people that present different sizes and orientations. Moreover the system can work under a variety of environments and also in crowded places.
By assigning a dominant orientation to each feature the system gets rid of geometric and rotational variations. This is achieved by mean of both rectangular and circular HOGs since they are not affected by lighting conditions and noise. AdaBoost has a very important role in this approach. In fact it is used to choose a set of meaningful features to achieve a robust detection. The computational time is then reduced by means of a cascade of rejectors, whose parameters are estimated again with AdaBoost.

1.2.4 Multi-Modal Anchoring

Another important feature is the ability to track an object over time. A tracking system can overcome inaccuracies in the feature sequence by mean of temporal information and context knowledge. Particularly these lasts means allow to process just a set of features in order to satisfy the low computational load constraint (limited sensor capabilities or many object to be tracked). One of the techniques employed is the anchoring framework ([27], [28]). In complex environments there are usually lots of sensors that generate different types of perceptions of the object that can vary in a significant way. Kleinhagenbrock et al [17] propose a solution to these problems by a two level anchoring:

1. anchoring composite objects;

2. anchoring the base components of each object.

The fusion of the different sensing modalities can be achieved in three ways:

1. sensor-based fusion methods like Kalman or particle filtering [29], [30];

2. rule-based fusion methods in which the results of individual algorithms can be fused together by combination rules ([31], [32]);

3. hybrid approaches, that are the combination of the two approaches previously described.

The work of Kleinhagenbrock uses the last approach. It showed a facilitation of the distributed and multi-modal anchoring of component symbols that can be also extended. The implementation merged laser range data and color images trying to find percepts for the symbols legs and face.

1.2.5 Audiovisual Approach

Another interesting approach is to use also some audio sensor and combine them with the usual ones (camera, laser scanner...). The introduction of such sensor like a microphone can appear quite useless in the detection step. In fact problems of noise and voice recognition are very hard and they need a lot of computational work. Whereas it can be really useful for example in the tracking of people and in the understanding of whether or not a person wants to interact with the robot (Human-Machine-Interaction). The work of Lang et al [33] fuses three types of information in order to track and focus the attention on a person, that are:

1. camera for face detection: useful when the person is facing the robot (detection and understanding of required attention), but not available usually when the robot is asked to follow a person;

2. laser scanner for object detection: it is used for detecting people in the environment, but it returns no clue on the relative position of the people detected (facing or not the robot) and it is limited on the 2D world;

3. stereo microphone for people localization: this can be used as additional information for localization of people but also to provide another way for knowing which people require attention by the robot.

In this research a color model of the torso of the people detected is also used in order to be able of detecting a person even if it can be lost by the 2D scanner (occlusions). This introduction made the system more robust and reliable.

1.2.6 Startup Tracking

Another method for tracking a single person in real time is the one proposed by Schlegel et al [24]. The person has to undertake a startup procedure in order to be followed. The person introduces himself/herself and the robot, without having any predefined model of a person, builds a model of the person that can now be tracked and followed. In order to have always a good representation of the person tracked, the robot continuously updates the model reducing the sensibility of the

illumination conditions.
The method implements a fast color-blob based approach fused with a sophisticated and computational expensive contour-based approach. By using together these two techniques they introduced a trade off between real-time performances and robustness in the tracking.
In order to be more precise, the model generated consists of a color distribution and a contour of the person. The first one is used in the process of segmentation of the image whereas the second one is useful in finding regions of interest at a high rate for the edge-based approach.
The approach was tested in a natural indoor environment on a real robot and the tests showed real-time performances and a good robustness when combining the two approaches.

1.2.7 Biological Motion Patterns

There is also a method of detection that relies on a biological background. Cutler and Davis [34] built a system able of detecting periodic biological motion patterns such as walking.
The first step is to retrieve moving object and it is achieved by frame differencing after a preprocessing phase. Then by morphological operations, the system returns the tracked objects of whom a temporal selfsimilarity matrix is retrieved since it has the property of being periodic, if the motion is periodic too.
The detection is then finally achieved by a time-frequency analysis based on the Short-Time Fourier Transform (STFT) and a fitting procedure returns the category of the tracked object (human, animal or vehicle).
The system showed robustness and real-time performances.

1.2.8 Shape-Based Template Matching

Another way of dealing with a moving camera is represented by the work of Gavrila and Giebel [11]. They built a system in which a hierarchical tree of shape-based templates is stored in order to retrieve the best match. This tree is automatically generated by a clustering procedure, where the cluster itself represents a node. When we have a candidate for the detection the tree is transversed from the root to the leaves. For each node the Chamfer distance between the candidate and the node itself is computed and, if this distance is greater than a specific value, the scan is not forwarded to its child nodes resulting in an inefficient match. The system provides also a Kalman filter tracker that takes care about temporal informations in order to overcome missed detection.

1.2.9 Pedestrian Pose Estimation

We want to close the summary of the previous works by presenting another system of Viola and Jones [14], that is a classifier trained both on human shape and motion features for pedestrians. The system analyzes input images and uses rectangular

features by mean of integral images. The usual AdaBoost process will then retrieve the best weak classifiers for the detection of the pose. Similarly also a dynamic detector is trained combining static and motion rectangular features. The testing showed how using the system with either the static or the dynamic classifier leads to good detection results when a large dataset is utilized.

1.3 Tools

Before starting the description of the proposed detection and tracking system we want to provide a deeper insight at the tools used.

1.3.1 Kinect

The Kinect is a motion sensing input device by Microsoft for the Xbox 360 video game console. It allows users to control and interact with the console without touching a game controller. This is achieved by means of a natural user interface, gestures and spoken commands.
The Kinect (figure 1.1) is made of different parts:

1. a VGA 640x480 color camera (CMOS) with a Bayer color filter;
2. a IR 1600x1200 camera (CMOS) with outputted sizes of 640x480;
3. an IR projector;
4. servos;
5. 4 microphones.

The majority of gestural control systems were based on the time-of-flight (TOF) method. It consists in sending an infrared light, or similar, into the environment. The time and wavelengths of light that returned to the capture sensor inside the camera, will then give informations on how the environment looks like. The Kinect instead uses another technique that is encoding already information in light patterns that are sent out. The deformations of those patterns will be similarly captured and they will return information on the environment as it was for the other method.
When the camera receives the IR light back, the real processing can start. PrimeSense developed a chip that is located inside the camera itself, which is already able to process the image looking for shapes resembling the one of a person. It tries to detect the head, torso, legs and arms. Once it has achieved this, it starts computing a series of parameters like where the arms are probably going to move.
The system developed by PrimeSense is able to detect any person in the scene, but it is able to process just a limited number of people depending on the power of the processor used.

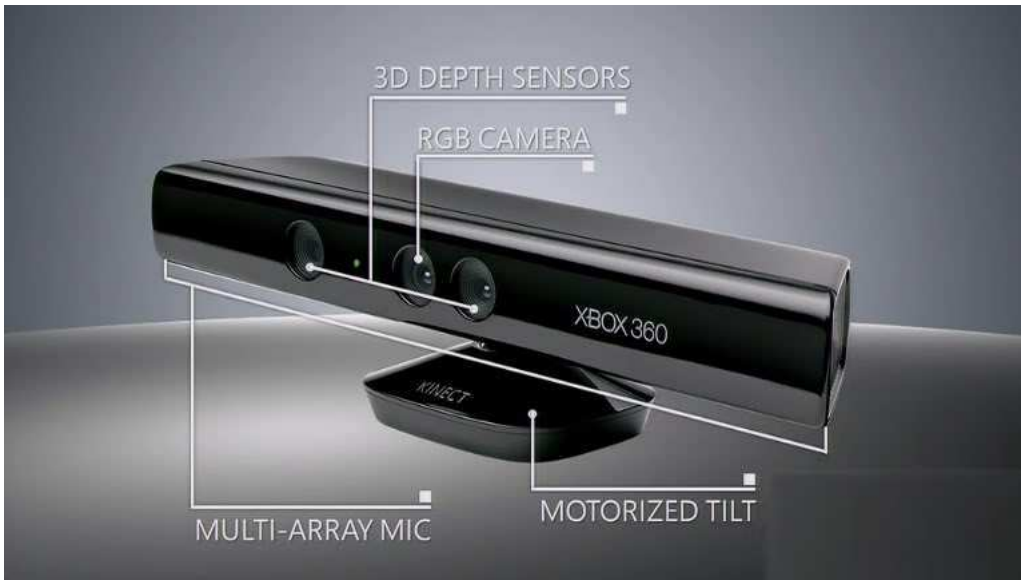

Figure 1.1. The Microsoft Kinect and ts parts.

1.3.2 OpenNI

OpenNI (Open Natural Interaction) is a multi-language, cross-platform framework that defines APIs for writing applications utilizing Natural Interaction.
Natural Interaction (NI) refers to the concept that Human-Machine-Interaction is achieved by human senses and, most of all, vision and hearing. OpenNI aims to define a standard API that is able of dealing with both vision and sensors, and a vision and audio perception middleware, allowing communication between the two components.
OpenNI provide two types of APIs:

1. implemented APIs: allow to deal with the sensor device;
2. not implemented APIs: allow to deal with the middleware components.

The clear distinction between sensors and middleware components is based on the "write once, deploy everywhere" principle. In fact OpenNI allows the porting of applications and moreover enables to write algorithm that works with known raw data independently from the sensor that has generated them. From the producer point of view, instead, OpenNI offers the possibility of building sensors for applications by just providing raw data and not APIs on how to deal with them. An application of OpenNI is for example the tracking of real-life 3D scenes.
OpenNI is an open source API that is publicly available.

The OpenNI Framework is an abstract layer (figure 1.2) that provides the interface for both physical devices and middleware components. Multiple components can

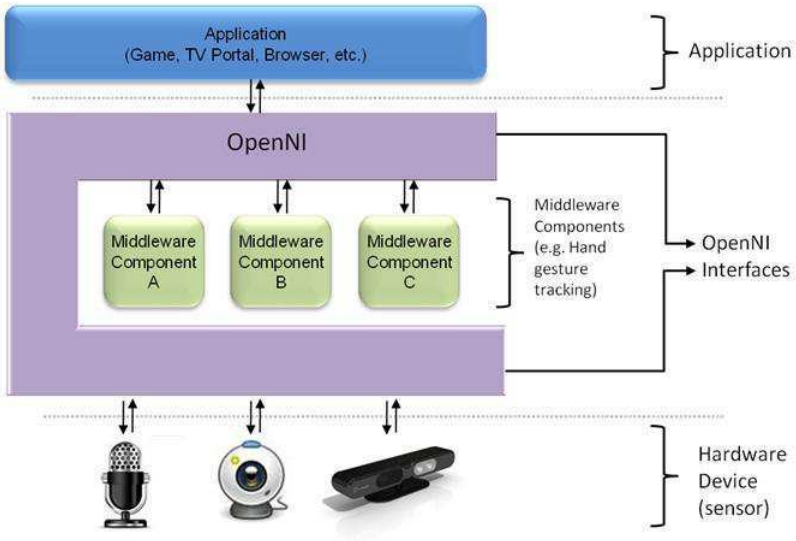

Figure 1.2. The OpenNI abstract layered structure.

register to this framework based on the specific API and they are called modules. A module is responsible for producing and processing the data of the sensor and the currently supported ones are:

1. 3D sensor;
2. RGB camera;
3. IR camera;
4. audio device.

Based on this, OpenNI provides also the following middleware components:

1. full body analysis;
2. hand point analysis;
3. gesture detection;
4. scene analyzer (segmentation, clustering and coordinates framing).

OpenNI relies on Production Nodes. They represents the productive part of the system, that is they create the data required for the interaction. These data can be either low level ones, RGB for example, either composited ones. In fact production nodes can also control lower level production nodes and they can in turn be used by higher level ones. In order to define communication and hierarchy these nodes are organized in production chains.

1.3.3 OpenCV

OpenCV is an open source computer vision library. It is written both in C and C++ and it is available for Linux, Windows and Mac OS X. However there is a lot of research that points to create interfaces for other programming languages like Python, Ruby, Matlab and others.
OpenCV is specifically designed for real-time applications and it can take advantage of multicore processors. Moreover it aims to provide an easy to use infrastructure that can facilitate the creation of computer vision applications.
The OpenCV libray contains over 500 functions that are spread out over many different areas, from camera calibration to medical imaging. In order to completly deal with computer vision problems, OpenCV presents also a complete Machine Learning Library (MLL). This library can be used also outside of the computer vision field for the more general machine learning topics.
OpenCV open source license allows to create commercial products using the tools of the library itself without returning improvements to the library itself. There are a lot of communities though that maintain the software.

1.3.4 ROS

Since the robotic field is continuously growing, it is difficult to provide software for robots. An additional problem is due to the variety of hardware that can be used. In order to overcome these problems, a lot of different frameworks have been developed, specifically designed for focusing on a determined aspect.
The Robot Operating System (ROS) is also a software framework that has been developed considering tradeoffs and priorizations, but its emphasis is on large-scale integrative robotics.
ROS was originally developed in 2007 by the Stanford Artificial Intelligence Laboratory. But from 2008 its development was redirected at Willow Garage, a robotics research institute.
ROS aims to the following goals:

1. peer-to-peer: a number of processes (different hosts) connected at runtime in a peer-to-peer topology;

2. tools-based: ROS has been developed with a microkernel design in which a lot of small tools are used to build and run the ROS components;

3. multi-lingual: different people prefer different programming languages. Interoperability is a key feature for a framework that aims to deal with all these people. In fact ROS has been designed language-neutral and supports C++, Python, Octave and LISP. The support for other programming languages is in state of completion;

4. thin: ROS encourages the development of drivers as standalone libraries. Its build system performs then modular builds inside the source tree by mean

of CMake. The thin ideology consists in having to deal with easier code extraction end reuse beyond the original intent. This is achieved by placing the complexity inside the libraries and creating small executables. For example ROS reuses other open-source projects like drivers, navigation system and simulators from Player, OpenCV, OpenRAVE;

5. free and open-source: ROS is released under the terms of the BSD license, and it is open source that is free for commercial and research use.

There is also another "side" of ROS called ros-pkg. This is a suite of user contributed packages, organized in stacks, that implements various functionalities like mapping, planning, perception. Also the ros-pkg contributed packages are licensed under a variety of open source licenses.

1.4 Organization of the Thesis

This document will proceed in two parts.
In the first one the proposed detection system is described. Chapter 2 explains why there is no need for a preprocessing module. Chapter 3 describes the implemented segmentation module and its phases of detection, filtering, clustering and save. Chapter 4 continues with the candidate shape and Relational Depth Similarity Features computation. Finally in chapter 5 the tracking features are introduced and the person tracking is explained.
In the second part instead the tests and the results achieved are presented together with the conclusions of chapter 8.

Part I

Detection System

Chapter 2

Why No Image Preprocessing

Firstly the system was given a preprocessing module, but it has been deleted since it wasn't improving the detection and tracking results.
The module was based on the "Gray World Assumption". It states that given an image with a sufficient amount of color variations, the average value of the red, green and blue channels (RGB) should average to a common gray color. Since the variations in color are random and independent, it would be safe to say that given a large enough amount of samples, the average should tend to converge to the mean value, which is gray. The result of using this assumption is that the image will no longer have any dominant color, often caused by indoor lighting.

Figure 2.1. Example of a preprocessing with a Gray World Assumption. On the left the original image whereas on the right the processed one.

Hence the module would have been really useful for detection methods that rely heavily on the color information (color skin detection for example). In our method the RGB information becomes useful just in the tracking system. In fact the detection is completely based just on the depth map retrieved by the Kinect whereas, as we will see afterward in this paper, the tracking is based on an hybrid combination of color histogram distances, between candidates, and nearness centroid computation. The tests have shown that the tracking system is not influenced in a relevant way by the illumination condition and for this reason the module has been deleted.

Chapter 3

Segmentation

The segmentation process is achieved in three consequent phases. On the first one the depth image is scanned in order to achieve the possible candidates, then those candidates are filtered in order to get rid of the redundancy, finally the depth imaged is clustered and labeled on the base of the retrieved candidates depth and each cluster is saved as a single object.

3.1 Depth Candidates Detection

The detection of depth candidates is based on a simple hypothesis: a distinct object on the depth map will arise from the background, that is there will be a significant difference between their average depth values. Moreover if an object is partially hiding another object, then the two objects can be detected again by looking at the difference between their average depth values.
Since we don't know the average depth values of all the objects in the scene (it is indeed the first goal to reach or better approximate this result) the best thing to do is looking over the depth value of each pixel. When we found relative depth differences between a pixel and the previous one that exceed a threshold level, we can safely state that we have detected an object.
The detection step is then described by the relation

$$|depth[i][j] - depth[i][j-1]| \overset{?}{>} depth\_threshold$$

where $depth[i][j]$ is the depth map at the row pixel i and column pixel j and where the absolute value allows the detection of objects that are partially occluded by others, or of the background too, starting from nearer ones.
If the relation is satisfied then the value of $depth[i][j]$ is saved as a depth candidate cause it represents a probable object.
There are two expedients that we follow in the scanning of the image:

1. computational load: in order to reduce the computational load the image is not scanned for each row. In fact if there is an object edge at a certain height

then it is highly probable to see that depth value again for the next row in the neighborhood of the column value of the detected edge. For this reason the image is scanned just every 10 row pixels;

2. noise: the Kinect is a noisy sensor. For this reason it happens not so rarely that there are pixel values way different from the real ones (in particular the value 0 appears frequently). Consequently the detection step, as it has been described, is too sensitive to noise. In order to avoid this phenomenon, each time that the relation described before is satisfied, we look forward, for a certain amount of pixel, if the depth value are in the range of the detected object or not. In the first case we really have detected an object and so we save that value as a depth candidate, otherwise we dealt with some noise and we keep going on with the scanning.

The complete detection step will be the following one:

$$|depth[i][j] - depth[i][j-1]| \overset{?}{>} depth\_threshold$$

if so we can have detected an object. So check for the next n pixels if

$$|depth[i][j] - depth[i][j+k]| \overset{?}{<} range\_threshold$$

where $k = 1, ..., n$ and where $j + k <$ x resolution. If also this second condition is satisfied then the value $depth[i][j]$ is saved as a depth candidate.

3.2 Depth Candidates Filtering

From the detection step we obtain an array of depth candidates, but a lot of them can have the same value. In order to clarify this idea let's make two example:

1. as we can see in figure 3.1 at the column j_1 in the row i_1 we have detected an object, so we have saved its value as a possible candidate. At the row i_2 and column j_2 again we have detected an object and saved its value as a candidate. It is clear that we have detected twice the same object.

2. in figure 3.2 the scenario is completely different. At the column j_1 in the row i_1 we have detected an object, so we have saved its value as a possible candidate. At the row i_1 and column j_2 again we have detected an object and saved its value as a candidate. The two object are distinct but their average depth value is the same, so they can belong to the same segmented layer.

Figure 3.1. Example of double detection of one person. The dotted red lines are the scanning of the algorithm at rows i_1 and i_2, whereas the dotted yellow ones represents the detection columns j_1 and j_2.

Figure 3.2. Example of detection of two people belonging to the same depth level. The dotted red line is the scanning of the algorithm at row i_1 whereas the dotted yellow ones represents the detection columns j_1 and j_2.

In both cases we see a clear redundancy in the depth candidates and we have to possibly get rid of it. For doing this we first sort the array of candidates and then we do an averaging of the depth candidates that are within a certain range. All the candidates at depth 0 are merged in one candidate then, from the first non zero value, we scan the vector using a procedure similar to the one used for the detection that is

$$candidates[i+1] - candidates[i] \overset{?}{<} candidate\_threshold$$

where *candidates* is the sorted array containing the depth candidates. If the condition is satisfied we keep track of the values and the number of candidates in the same range and then we proceed to the next value that is looking the condition

$$candidates[i+2] - candidates[i] \overset{?}{<} candidate\_threshold.$$

We can generalize this pruning process. Called k the first index for which it is false that

$$candidates[i+k] - candidates[i] < candidate\_threshold$$

then the true candidate computed as

$$\frac{\sum_{j=0}^{k-1} candidates[i+j]}{k}$$

will be saved.
The scanning of the array will continue then by using $candidates[i+k]$ as first element and iterating the process just described.

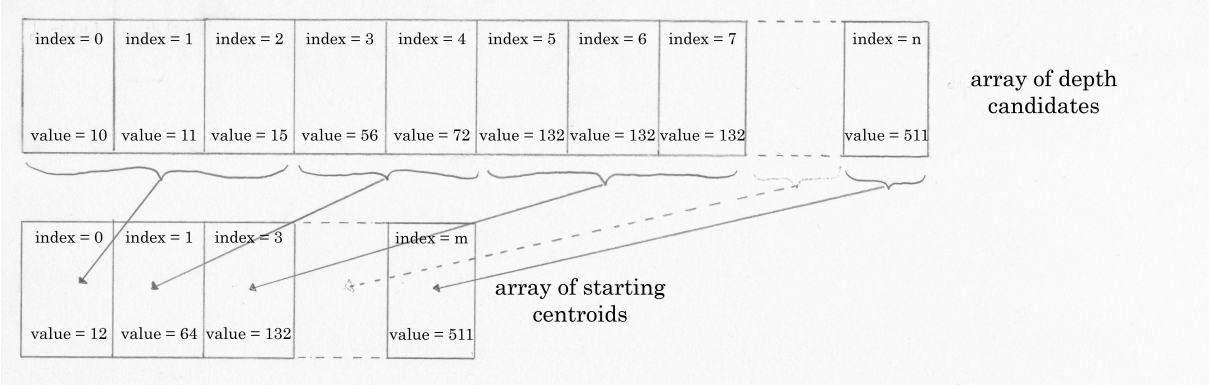

Figure 3.3. Example of the process of candidate filtering.

3.3 Clustering, Labeling and Saving of Segmented Layers

The clustering and labeling of the depth map is then based on the k-means algorithm. The most common version of it uses an iterative refinement technique.

Given an initial set of k means $m_1^{(1)}, ..., m_k^{(1)}$, the algorithm proceeds by alternating two steps:

1. assignment step: every pixel is assigned to the candidate with the closest value:

$$S_i^{(t)} = ||depth[i][j] - m_i^{(t)}|| \leq ||depth[i][j] - m_{i*}^{(t)}||, \forall i* = 1, ..., k;$$

2. update step: compute the new k means as

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{depth[i][j] \in S_i^{(t)}} depth[i][j]$$

The depth candidates computed in the first two phases are used here as starting centroid for the algorithm. There's just a minor issue: since we use the OpenCV implementation of the algorithm we cannot give directly the centroid. So in order to get advantage of the work done until now, we compute an initial labeling of the depth map by applying the assignment step one time and by giving the result as initial labeling for the OpenCV method.
The final result of the k-means algorithm is a labeled image as shown in figure 3.4.

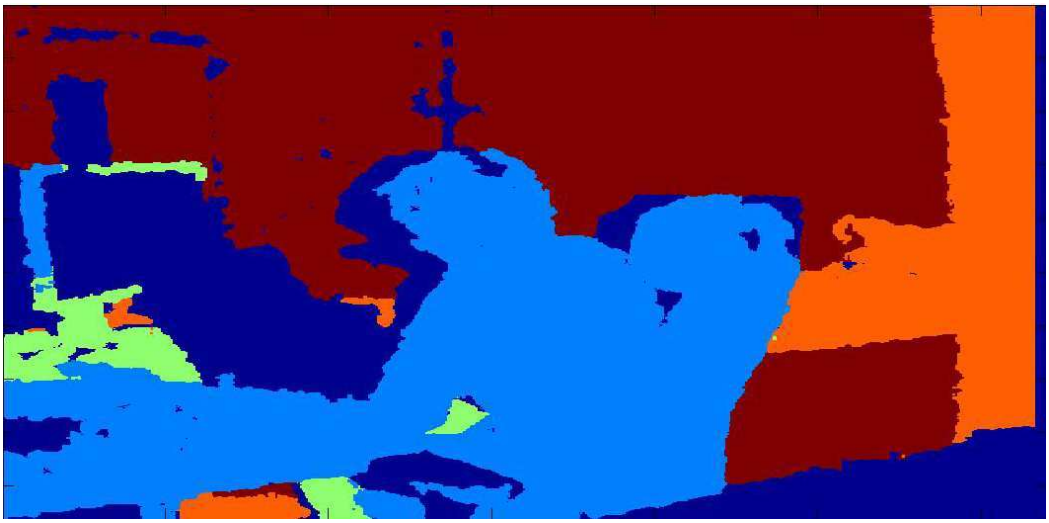

Figure 3.4. Clustered image.

At this point each clustered layer is used as a mask in the depth map. For each segmented layer, the masked points are saved as a single image as shown in the figure 3.5.

Figure 3.5. Image of segmented layer.

Chapter 4

Features Computation

This module is responsible for looking in a segmented layer if there are candidates with a real height and width similar to the ones of a person. If this is the case, then those candidates are normalized to the fixed dimensions of 64x128. This process is done both for the depth image, where the computation takes place, and also for the correspondent RGB image. On the normalized depth image, a vector of Relational Depth Similarity Features is computed. This vector is given then to a classifier in order to achieve the likelihood of being human of the candidates. We will see now the detailed description of the module.

4.1 Shape Computation

Each segmented image is completely scanned. For each column we look every row pixel seeing if it is occupied or not. While doing this we keep track of the consecutive pixels encountered. The maximum value of those is saved as the maximum height in pixel of that column and this statement can be safely done by using both a reasonable hypothesis and an expedient:

1. hypothesis: there can be more than one object in the scene represented in the same column. With this method we take in account just the highest object as a probable candidate. This is perfectly legal since we are looking for people. In fact theoretically there can be two cases:

    a) a person and a smaller distinct object above the person itself: in this case taking the maximum height is the right thing to do in order to detect the person (figure 4.1);

    b) a person and a bigger distinct object over the person itself: this case is really highly unlikely to happen. In fact is like assuming to have an object of the dimensions of a wardrobe flying above the person.

    So the operation is correct;

Figure 4.1. Image showing the first case that is an object above a person.

2. expedient: we have always to take in account that we are dealing with a noisy sensor. Moreover more noise can be introduced by the segmentation process. For these reasons, when scanning each row of a column, if we don't find a consecutive pixel we consider it as noise and we increase a noise counter. We keep then scanning the column and if we find again an object pixel then we were really dealing with noise and we update the height of the object. Instead if we keep having noisy pixels it means that it is not noise and that the object has just the height computed before finding the lasts pixels.

So the complete process can be described as follows. Assuming that at row i and column j we have the first object pixel, that is $depth[i][j] \neq 0$ we look

$$depth[i+1][j] \overset{?}{\neq} 0$$

if it isn't so we increase the noise counter then, in both cases (negative and positive), we look at the row $i+2$. The height saving condition for an object is that for k consecutive pixels we have $depth[i^*+l][j] = 0$ where $0 \leq l < k$. If it is so we save as temporary maximum height in pixel the value

$$pixel\_height = i^* - i.$$

Instead if we find again an object pixel before the k consecutive noisy ones, we consider those pixels as height object pixels and we start again the process from the beginning as already described. If we find more than one object height, we will consider then the maximum one.
At this point we have that for each column a pixel height has been saved. In order to convert it to useful information we convert that pixel height into a real height in cm.
The last step is to do an ordered scan of the column real height. With a similar approach to the one used for computing the pixel height, we compute the width of an object, that is firstly we look if

$$real\_height[j] \overset{?}{\geq} threshold\_height.$$

Assuming that j is the first column that satisfies the above relation, we look if the next column has also a real height satisfying the relation. If this is the case, then we keep looking at the column $j+2$ and we update the object pixel width. Otherwise we use the same expedient that has been used for the computation of the column height, that is, we assume the false result as noise and we keep looking forward if it was really noise or if the object has just the width computed right before. So the width saving condition for an object is that for k consecutive pixels we have $real\_height[j^*+l] < threshold\_height$ where $0 \leq l < k$. If this happens, we take into account the width in pixel described by

$$pixel\_width = j^* - j.$$

Now we have again to convert this pixel width in a real width in cm. Once we have done this we see if the real width exceeds a threshold value. If it is the case then we have finally found a candidate for the human detection.
In order to have now some good proportions and shape for the candidate, we don't take into account the maximum real height and the real width previously computed. Firstly while scanning the object we kept track of the highest point and lowest point of the object itself. This two points are used as height of the rectangle that will

bound the candidate. So if the highest point has row index i_1 and the lowest i_2 then the bounding box height in pixel will be

$$bbp\_height = i_2 - i_1.$$

It doesn't have to appear strange the fact that the height is computed by subtracting the lowest to the highest since it is the lowest that has the higher row index.
The corresponding width is then computed as half the bounding box height that is

$$bbp\_width = \frac{bbp\_height}{2}.$$

Now we can perfectly describe the bounding box by means of another variable that we can keep track of in the scanning of the height, that is the column index j_1 of the highest point of the object. That point represents the middle point of the width.

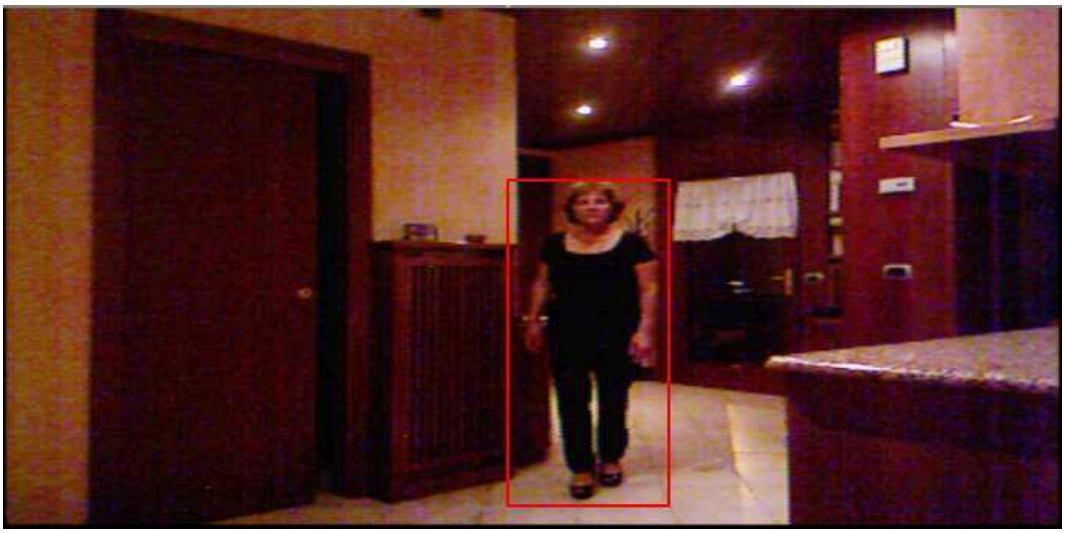

Figure 4.2. Strict bounding box.

Finally the bounding box is increased of the 25% both in width and in height having, as final result, the one showed in figure 4.3.
This procedure has been undertaken since the dimensions of the candidate for the classifier should be normalized to 64x128 so it is important to keep the proportion

$$\frac{bbp\_width}{bbp\_height} = \frac{64}{128} = \frac{1}{2}.$$

The last step of the shaping procedure consists in resizing the candidate to the required sizes of the classifier.

4.1.1 Real Width and Height Computation

In this paragraph we want to clarify how it is possible to convert the width and height in pixels. There are two ways of doing this:

Figure 4.3. Increased bounding box. In transparent red is reported also the strict bounding box

1. exact way: knowing the intrinsic parameters of the camera, we can obtain the parameters for actuating a perfect conversion. There are some matlab scripts that allow the retrieval of those parameters;

2. approximate way: it is the way used in this thesis. Basically it's an empirical method of approximate a conversion between pixel and real width. Firstly the Kinect is carefully positioned parallel to a flat surface (like a table) of known dimensions b, h and with the depth camera that points at the middle point of the surface. Then we move the surface at a distance d where the edges of the surface itself fulfill the x resolution of the camera, that is the edges of the surface correspond to the columns 0 and 639 (if we use a 640x480 resolution as in this case) (figures 4.4 and 4.5). We measure then the distance between the sensor and the surface and we can finally find an equation that relates the width in pixel and the one in cm, that is:

$$width_{max} = b_1 \frac{depth}{10 d_1}$$

$$width_{real} = width_{max} \frac{width_{px}}{y_{resolution}}.$$

Similarly we can proceed for the height. The Kinect is still positioned parallel to a flat surface and with the depth camera that points at the middle point of the surface. This time we move the surface until the edges of it fulfill the y resolution of the camera, that is the edges of the surface correspond to the rows 0 and 479 (again using a 640x480 resolution). We measure then the

distance between the sensor and the surface and we can find the conversion equation for the height too, that is:

$$height_{max} = h_2 \frac{depth}{10 d_2}$$

$$height_{real} = height_{max} \frac{height_{px}}{x_{resolution}}.$$

depth in both cases is the average depth of the segmented layer to which the candidate belongs.

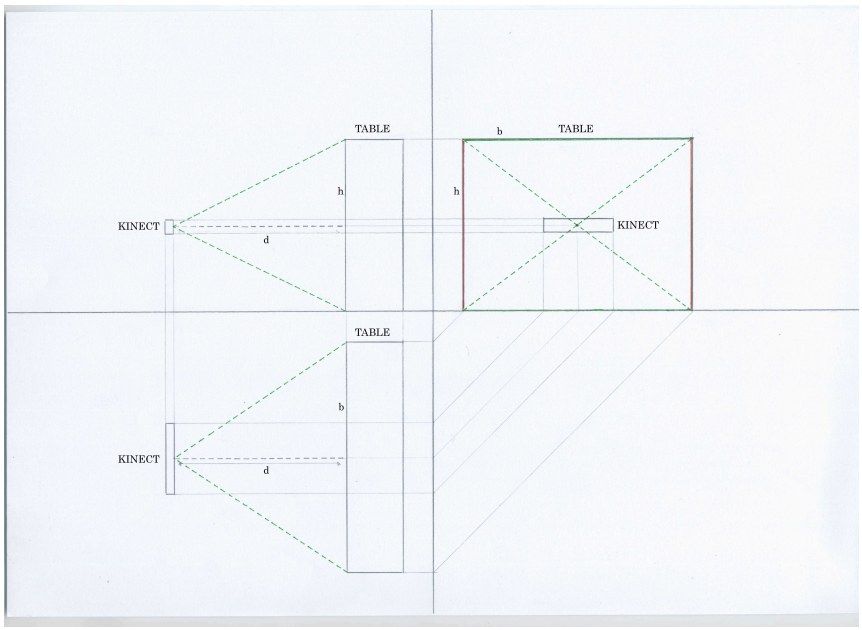

Figure 4.4. Orthographic projection of the process of calibration.

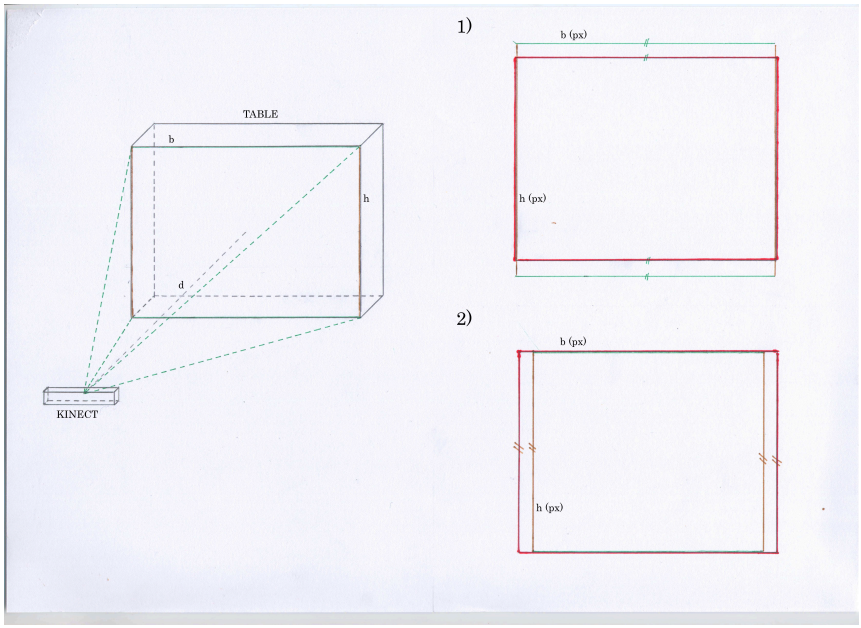

Figure 4.5. Dimetric projection of the process of calibration. We can see the two cases: in the first one the x resolution is fulfilled, whereas in the second the y one. The red rectangle represents the captured 640×480 image.

4.2 Histogram Tracking Preprocessing

In the shaping procedure we obtained a normalized shaped depth candidate of dimensions 64x128. Before resizing it to the normalized dimensions a preprocessing of the RGB image is undertaken, since it will be necessary for the tracking system. This preprocessing is really simple.
The RGB image is masked with the segmented layer of the candidate and then we cut out the portion of the image delimited by the coordinates of the bounding box for the depth candidate. We can see the workflow of the process in figure 4.6.
The colored candidate will be then normalized as well to the sizes of 64x128 and forwarded to the histogram computation if necessary.

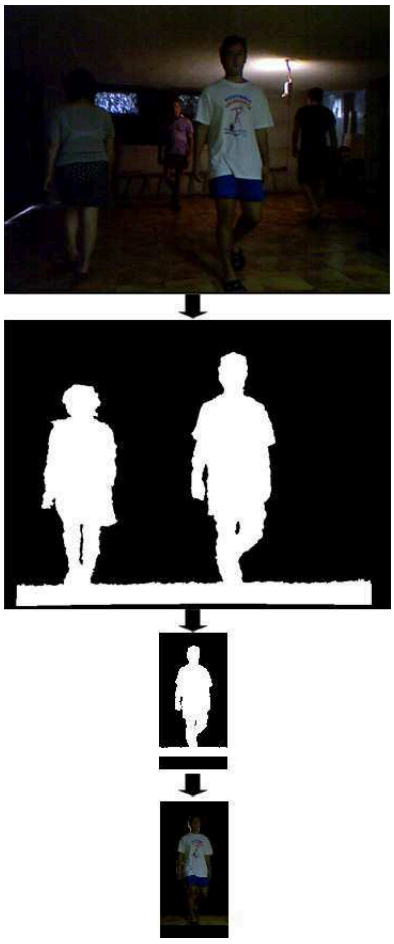

Figure 4.6. Workflow for retrieving color depth candidates.

4.2.1 Depth Registration

A note has to be done in order to clarify that all the operations done in the procedure just explained are safe. The problem in fact is that the camera for grabbing the depth information and the RGB one are distinct. So the two images are not aligned. Luckily the Kinect can register the depth camera together with the RGB one. After this operation every pixel in the depth map is exactly the same of the RGB one.

Figure 4.7. Examples of not registered and registered image.

4.3 RDSF Computation

The features that have been used in order to detect people are called Relational Depth Similarity Features (RDSF) [13]. They are very recent in literature and we have chosen them instead of the more tested HOG features for two main reasons:

1. firstly to test the effectiveness of these features since they have not been used widely;

2. secondly cause they are completely based on the depth information.

RDSF are based on a measure of the degree of similarity between depth histograms obtained from two local regions. First of all we divide the candidate in cells of 8×8 pixels that will represent our local regions. For each couple of cells we compute then the normalized depth histogram. Let's assume to call the two obtained histograms p and q, what happens now is that, knowing the number n of bins of each histogram, we can compute the degree of similarity S between the two local regions by using the Bhattacharyya distance [35], that is

$$S = \sum_{i=1}^{n} \sqrt{p_n q_n}.$$

This distance is used as feature for the classifier and it is clear from its definition that it actually represents a degree of the relational depth similarity.

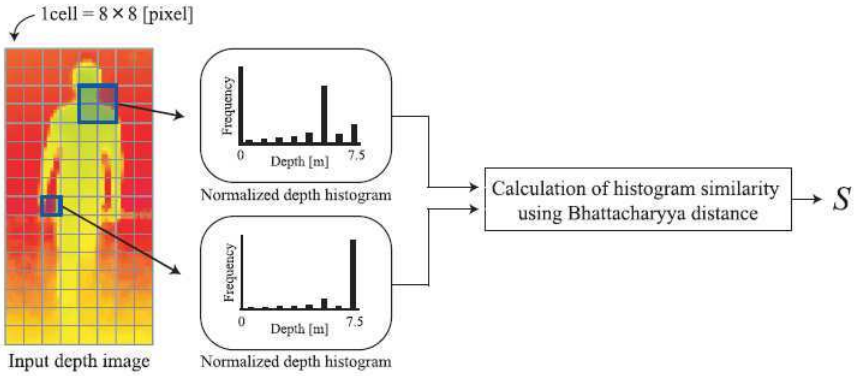

Figure 4.8. RDSF explained.

Since our candidate has the fixed dimensions of 64x128, we have a total of

$$k = \frac{64}{8}\frac{128}{8} = 128$$

cells. We have to compute the distance for every distinct pair of local regions so we obtain a feature vector of length

$$l = \sum_{i=1}^{k-1} i = \left(\sum_{i=1}^{k} i\right) - k = \frac{k(k+1)}{2} - k = \frac{k(k-1)}{2} = 8128.$$

This features vector is given then to the classifier to obtain the likelihood of being human.

4.3.1 Construction of the Classifier

Until now we always assumed to already have a classifier capable of stating if a candidate is likely to be or not to be a person. In reality, this classifier has been constructed by means of a method called Real AdaBoost [36].
First of all let's clarify what boosting means. Boosting is a machine learning algorithm for performing supervised learning and it is based on the demostrated hypothesis that a set of weak learners can create a single strong learner. A weak learner is just slightly correlated to the true classification since it is able of performances that exceed the ones of a random guessing. However they can be far away from the true classification. Whereas, a strong learner is a classifier that is quite near to the true classification.
AdaBoost is the short for Adaptive Boosting and it is a boosting algorithm formulated by Yoav Freund and Robert Schapire [37]. The adaptive part stands for the fact that subsequent built classifiers are tuned in order to be more sensitive of those candidates misclassified by previous classifiers. AdaBoost is sensitive to noisy data

and outliers, but it is usually less sensitive to overfitting.
AdaBoost works in this way: it calls every weak classifier repeatedly in a series of rounds equal to the number n of classifiers. During each round it updates a weights distribution. The function of this distribution is to point out the importance of examples in the data set for the classification. Reasonably if an example is incorrectly classified then its weight will be increased, whereas, if an example is correctly classified, then its weight will be decreased. By doing this the new classifier will be focused more on those examples.
The Real version of this boosting algorithm obtains degrees of separation from the probability density functions for each dimension of features in positive classes and negative classes. The selection is made in order to choose those features that allow the greatest separation between positive and negative classes as weak classifiers. Such degrees of separation is computed in a way so that the output of the classification is a real number (so the name Real AdaBoost). Defined a generic weak classifier selected by the procedure of training as $h_i(x)$, the final classifier $H(x)$ that we will use for obtaining the likelihood of being a human is based on the following equation

$$H(x) = sign\left(\sum_{i=1}^{n} h_i(x)\right).$$

Two specific programs have been created in order to train the classifier. The procedures of the programs are identical to the one already described, but they differs from it in the RDSF computation. In fact we have:

1. positive examples gathering: this first program is responsible for gathering positive examples for the classifier. In order to do that, we give to the program a collection of pictures that contain possible candidates. The program computes all the probable candidates for each image and it returns them as an image on the screen. At this point the supervisor is asked to confirm if the candidate represents a human (positive example) or not (negative example). If it's the case of a positive example then the program adds a line on a text file containing an ordered array with the result of the classification (that is 1) and the vector of RDSF;

2. negative examples gathering: this second program instead is responsible for gathering negative examples for the classifier. In order to do that, we give to the program a collection of pictures that do not contain people. The program computes all the probable candidates for each image. The difference between the previous program is that at this point we know that, since there are no people in any image, the candidate will certainly be a negative example. Indeed the program automatically adds a line on another text file containing an ordered array with the result of the classification (in this case 0) and again the vector of RDSF.

The text file resulting from the merging of the positive and negative ones, is then used for obtaining a tree of weak classifiers by means of the Real AdaBoost training.

This tree will then be used as the strong classifier of which we have always talked about.

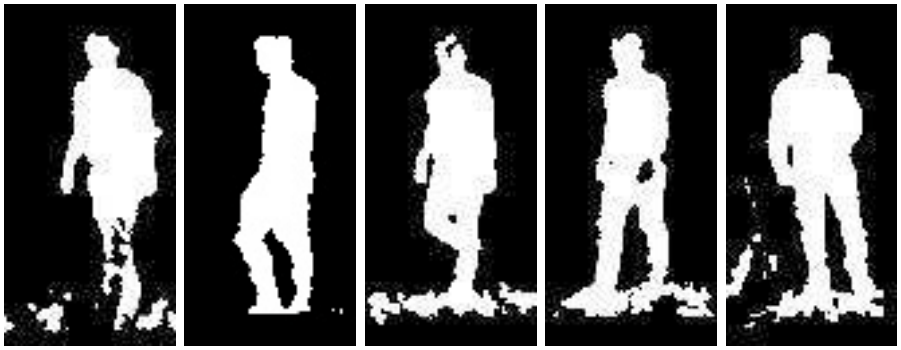

Figure 4.9. Examples of positive training images.

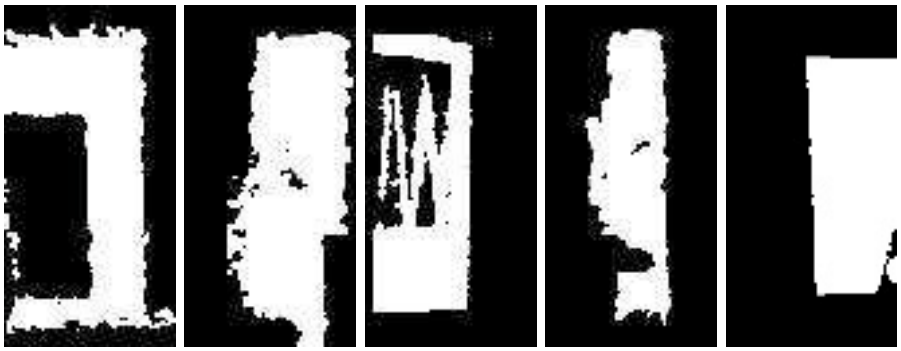

Figure 4.10. Examples of negative training images.

Chapter 5

Features Tracking System

Once a person has been detected, it is important to track its movements during the time in which it is on the scene. This last module is responsible of fulfilling this duty.

5.1 Tracking Features

In this section we will define which features describe a person in our tracking system:

1. color histogram: the final result of section 4.2 is a segmented color candidate as we can see in figure 4.6. We compute for this color candidate the color histogram and we save it as a feature for the detected person. In fact the distribution of color of a person is a good way of describing it as well as it allows to distinguish it from another one;

2. present position: the present position is represented by the centroid of the person itself together with the average depth value of the candidate. These can be easily computed since from the candidate shaping we know both the bounding box characteristics and the centroid depth of the segmented layer d. What we need in fact are the coordinates of the upper left corner x_l, y_l, the bounding box width bbp_width and height bbp_height. The centroid will be identified by the coordinates

$$x_c = x_l + \lfloor \frac{bbp\_width}{2} \rfloor$$

$$y_c = y_l + \lfloor \frac{bbp\_height}{2} \rfloor$$

$$depth_c = d;$$

3. previous position: as explained before the position is represented by the coordinates of a centroid. Consequently the previous position represents the last position in which the specific person has been detected;

Figure 5.1. Centroid computation.

4. distance: it is the space covered between the previous position and the actual position. The details on how it is computed will be given shortly;

5. box color: this is a side feature that has nothing to do with the person itself. It represents the color of the edges of the bounding box that will be used for the visualization of the detection result. This color is simply computed with a random choice of the value of Red, Green and Blue (from 0 to 255) that are merged together in order to achieve the final color.

5.2 Tracking Features Creation and Update

Once a candidate has been given to the classifier, three cases can happen:

1. the classifier states the candidate not being a person;

2. the classifier states the candidate being a person and the tracking system returns that it has never been detected before;

3. the classifier states the candidate being a person and the tracking system returns that it has already been detected.

In the first case we don't have to do anything.
In the second one we have to save for the first time all the tracking features. As we have seen in the previous section, we can easily save the color histogram and the present position given by the centroid. What about the other features? First of all as previous position we save the present position, for the distance we save the value 0 whereas for the color box we do the random operations and we save the

final result as the color for the box edges.
Finally in the third one we have to update the features. We save the new color histogram in place of the old one in order to get rid of illumination differences. Then we move the value saved as present position to the previous position and then the actual position can be safely saved as the present position. We finally compute the distance between those two values and save it in the distance feature. The color box value instead has not to be changed.
We have to point out how the distance is computed. We know the coordinates of the present centroid $(x_c, y_c, depth_c)$ and the ones of the previous centroid $(x_{c'}, y_{c'}, depth_{c'})$. Following the figures 5.2 and 5.3, we will compute now the necessary parameters.

1. $width_c$: it represents the distance in cm between the present centroid and the central column. It can be easily computed knowing the pixel width that is

$$pixel\_width_c = \left| y_c - \frac{y\_resolution}{2} \right|$$

and by applying the conversions explained in the section 4.1.1. We note that in this case the value $\frac{y\_resolution}{2}$ is equal to 320 and more important that this width is given in absolute value;

2. $width_{c'}$: it represents the distance in cm between the previous centroid and the central column. Identically to what we have done for $width_c$, it can be computed knowing the pixel width

$$pixel\_width_{c'} = \left| y_{c'} - \frac{y\_resolution}{2} \right|$$

and by applying again the conversions explained in the section 4.1.1. As before this width is given in absolute value;

3. d_{vc}: it represents the distance between the sensor and the projection of the present centroid on the middle column. It is clear that the figure identified by $width_c$, d_c and $depth_c$ is a right triangle. Of this triangle we know already one leg and the hypotenuse. So we can easily find out the third value by mean of the Pythagorean theorem that is

$$d_{vc} = \sqrt{depth_c^2 - width_c^2};$$

4. $d_{vc'}$: it represents the distance between the sensor and the projection of the previous centroid on the middle column. Identically at what done right before we can compute this value as

$$d_{vc'} = \sqrt{depth_{c'}^2 - width_{c'}^2}.$$

As clearly shown in the images there can be now two possibilities:

1. the present and previous centroid are on the same half of the image;
2. the present and previous centroid are on opposite halves of the image.

In the first case the horizontal distance between the two centroids d_h is equal to the subtraction of the singular widths, that is

$$d_h = width_c - width_{c'}.$$

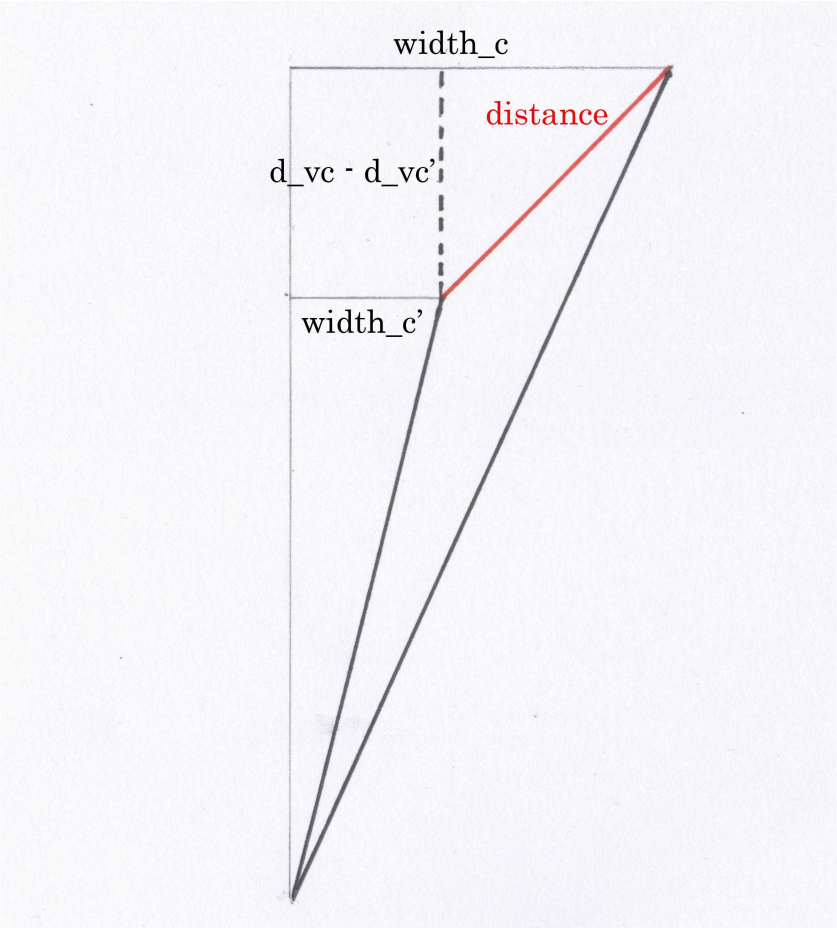

Figure 5.2. Centroids in the same half.

In the second one instead it is represented by the sum of the two widths, that is

$$d_h = width_c + width_{c'}.$$

The vertical distance between the two centroids instead is simply given by the equation

$$d_v = d_{vc} - d_{vc'}.$$

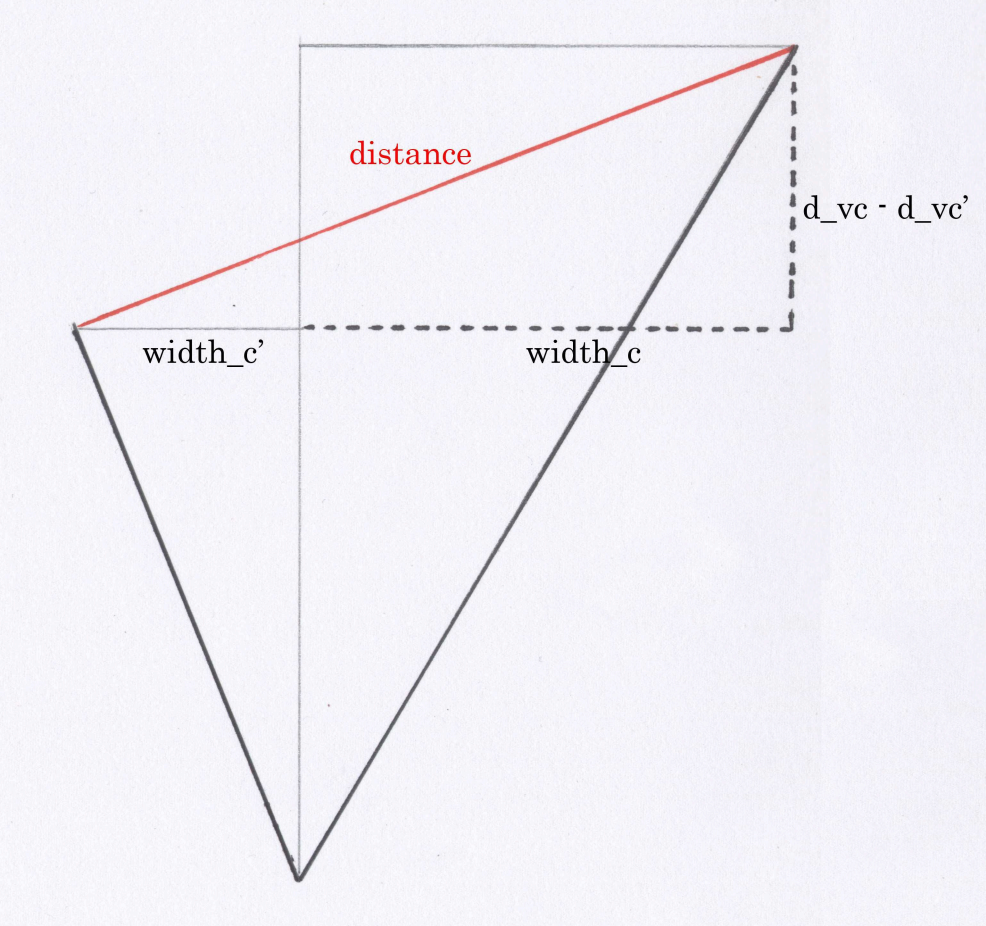

Figure 5.3. Centroids in opposite halves.

So the value of the distance can be computed using again the Pythagorean theorem

$$distance = \sqrt{d_h^2 + d_v^2}.$$

5.3 Person Tracking

This section wants to clarify the last thing on the feature tracking system, that is, how we know if a person that has been correctly detected is new on the scene or if it has already been detected previously.
All the people that has been detected are saved on a database. So each time that we have a detection (a positive result given by the classifier), the first thing that we have to do is to check if the person detected is present on the database. In order to do this we use two discriminant values that are:

1. color histogram distance: for each person in the candidate we have saved its

color histogram and for every candidate we do a preprocessing in order to compute the color histogram as already explained. By means of the Bhattacharyya distance we obtain a degree of similarity between the two histograms s that, for the way in which it is computed, respects the relation $0 \leq s \leq 1$;

2. real distance: that is the distance d_{pt} between the centroid of the person that we want to check and the centroid of the present position of each person in the database. The way in which this distance is computed is identical the procedure described in the section 5.2.

In order to make use of the second value we need to scale it so that its value can follow a relation similar to the one of the color histogram distance, that is $0 \leq scaled\_d_{pt} \leq k$ with k small integer.
Since the images should be taken at a real-time rate then it is good to suppose that the $scaled\_d_{pt} = 0$ will correspond to $d_{pt} = 0$ and that in a frame the maximum movement will be of 1 meter, that is state that $d_{pt} = 100cm$ correspond to $scaled\_d_{pt} = 1$. In this work we wanted to benefit even more the good values of the real distance. For this reason we assumed that $scaled\_d_{pt} = 0$ corresponds to $d_{pt} = 2cm$. So for values of d_{pt} less than $2cm$, $scaled\_d_{pt}$ will have negative values. Given those two couple of values, $(2, 0), (100, 1)$, we can find a linear relation by means of the straight line equation

$$y - y_0 = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) \Rightarrow y = \frac{1}{98}(x - 2) \Rightarrow y = y = \frac{1}{98}x - \frac{1}{49}.$$

By substitution of x and y with the right terms we have

$$scaled\_d_{pt} = \frac{1}{98}d_{pt} - \frac{1}{49}.$$

So the likelihood lkh of a detected person being one of the people in the database is given by the sum of those two values, that is

$$lkh = s + scaled\_d_{pt}.$$

The final step is to find a threshold value for this likelihood in order to affirm that the person can have been previously detected or to state instead that it is new on the scene

$$lkh \overset{?}{\leq} threshold\_lkh.$$

Given this condition three things can happen:

1. the condition is not satisfied for any person in the database: this means that the person detected is new on the scene. Consequently it has to be added following the creation procedure explained in section 5.2;

2. the condition is satisfied for just one person in the database: this means that the person detected was already in the database so we will proceed with the update of its features as explained again in section 5.2;

3. the condition is satisfied for more than one person in the database: it is a strange case but it can happen for very near people that exchange their position or for people almost identically dressed. In this case the person detected was again already in the database and we will choose the one that minimizes the lkh value. Just for this person we will update its tracking features.

5.4 Showing Detections

The final part consists in showing the results achieved in real-time. In order to do that, while processing a new image, the grabbed one is shown together with the bounding boxes identified by the shape computation module and confirmed, as humans, by the classifier. On the bounding boxes there will be written also additional informations deriving by the features tracking module. This are the number of the person tracked and the distance covered, that is the distance between the centroid of the previously detected centroid and the actual one.

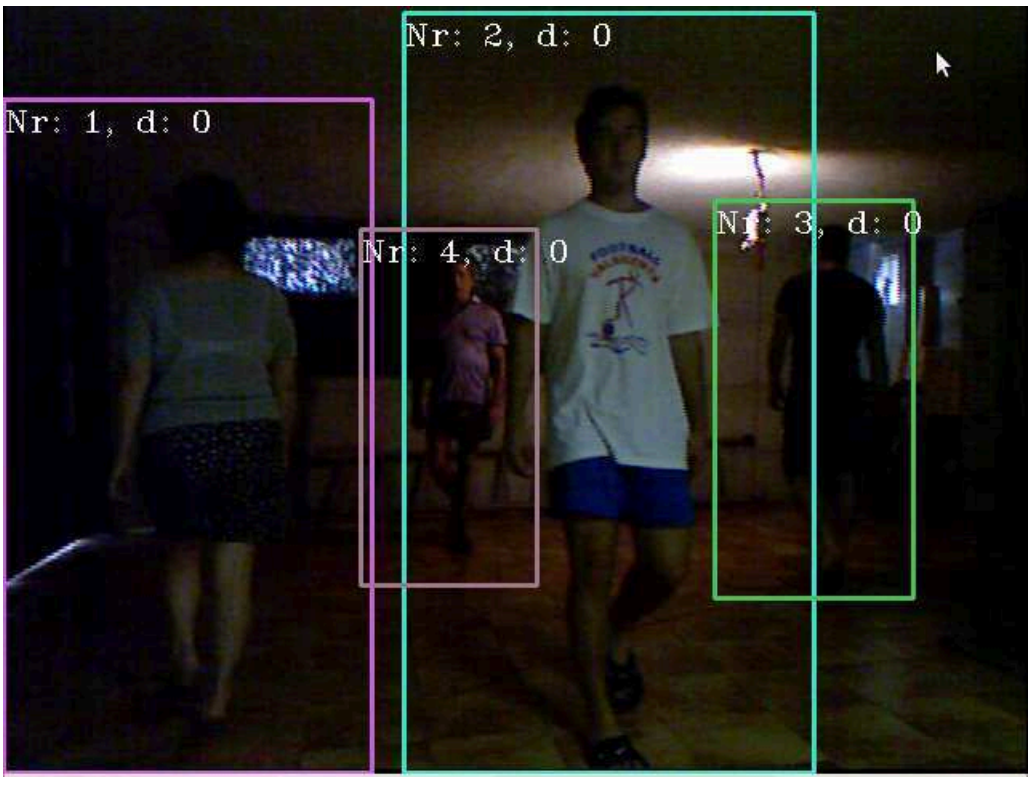

Figure 5.4. Image with detected people and tracking informations.

Part II

Tests and Results

Chapter 6

Introduction

6.1 Performance Measurement

As already seen in the first part of this work, the problems of people detection and tracking have been studied quite a lot in the past years. For this reason it was necessary to introduce also an objective method for measuring the performances of the various algorithms. There are two goals that will be accomplished by using a shared objective measurement method:

1. providing an actual method of performance measurement;
2. allowing the comparison between different algorithms.

6.2 Ground Truth

As seen in section 5.4 the algorithm returns a bounding box for each person detected. In order to determine whether or not this bounding box is right, we should compare it with the true detection of the frame.
This true detection for each frame of the video is often done by hand by a human being, and it is called ground truth.
Comparing the ground truth with the actual detection of the algorithm we can retrieve the following direct parameters:

1. number of right detection (true positive);
2. number of wrong detection (false positive);
3. number of miss (false negative).

In order to check if a detection is right the following comparison equation is used

$$\frac{area(B_a \bigcap B_{gt})}{area(B_a \bigcup B_{gt})} \geq 0.5$$

where B_a represents the bounding box of the algorithm whereas B_{gt} is the ground truth one.
Multiple detections of the same person are considered all false positive except one. From the three direct parameters we can also retrieve the following ones:

1. True Acceptance Rate (TAR):

$$TAR = \frac{n_{tp}}{n_p}$$

   where n_{tp} is the number of true positive and n_p is number of people detected in the ground truth (that can be computed as the sum of the true positives and the misses);

2. False Acceptance Rate (FAR)

$$FAR = \frac{n_{miss}}{n_{miss} + n_{tn}}$$

   where n_{miss} is the number of miss and n_{tn} is a parameter that assumes to have one true negative for each frame;

3. True Rejection Rate (TRR)

$$TRR = 1 - FAR;$$

4. False Rejection Rate (FRR)

$$FRR = 1 - TAR;$$

5. False Positive Per Frame ($FPPF$)

$$FPPF = \frac{n_{fp}}{n_f}$$

   where n_{fp} is the number of false positive and n_f is the number of frames;

6. Miss Per Frame (MPF)

$$MPF = \frac{n_{miss}}{n_f}$$

   where n_{fp} and n_f are again respectively the number of miss and the number of frames.

Moreover, in order to evaluate also the tracking performances, these two metrics are used:

1. Multiple Object Tracking Precision ($MOTP$)

$$MOTP = \frac{100}{n_{tp}} \sum_t \sum_i \left(\frac{area(B_{a_{ti}} \bigcap B_{gt_{ti}})}{area(B_{a_{ti}} \bigcup B_{gt_{ti}})} \right)$$

where $B_{a_{ti}}$ is the bounding box of the algorithm at frame t for the person i and $B_{gt_{ti}}$ is the bounding box of the ground truth again at frame t for the person i;

2. Multiple Object Tracking Accuracy ($MOTA$)

$$MOTA = 1 - \frac{n_{miss} + n_{fp} + n_{mismatches}}{n_p}$$

where the only new parameter is $n_{mismatches}$ that represents, for instance, the number of mismatches in the tracking procedure.

6.3 Receiving Operating Characteristic

The Receiving Operating Characteristic (ROC) curve is the first way of describing in an objective way an algorithm for people detection. On the x axis it is reported the FAR whereas on the y axis there are the corresponding TAR values. In order to achieve different TAR and FAR values, a parameter has to be changed. In this case this parameter is represented by the threshold value of the classifier. In fact as explained in 4.3.1 the classification for detecting people uses the relation

$$H(x) = sign\left(\sum_{i=1}^{n} h_i(x)\right).$$

Instead of using the sign function, we can rewrite the equation for the detection as

$$H(x) = \left(\sum_{i=1}^{n} h_i(x)\right) \geq 0.$$

0 is the right value for achieving the exact correspondence with the sign function, but we can substitute it with any value obtaining the threshold value of the classifier. So we can finally rewrite the equation as

$$H(x) = \left(\sum_{i=1}^{n} h_i(x)\right) \geq threshold_{classifier}.$$

The optimal point of a ROC curve is represented by the coordinates $FAR = 0, TAR = 1$. So the curve is better if it passes nearer to this point. The curve representing a random classifier is the 45 degrees diagonal with origin in the point $FAR = 0, TAR = 0$.

6.4 Detection Error Trade-Off

The Detection Error Trade-Off (DET) curve is the second way of describing in an objective way an algorithm for people detection. The aim of this curve is to relate the two types of errors that can be retrieved, that are false detections and misses. On the x axis the $FPPF$ values are reported whereas on the y axis there are the corresponding FRR values in percentage. Again in order to achieve different $FPPF$ and FRR values the threshold value of the classifier is used as changing parameter. The optimal point of a DET curve is represented by the coordinates $FRR = 100, FPPF = 0$. So the curve is better if it passes nearer to this point. The curve is usually represented with the axis expressed in logarithmic values in order to point out more clearly the differences between the various algorithms.

6.5 Classifiers

For the tests four classifiers have been trained:

1. *tree*: 2814 positive candidates, 7484 negative ones;
2. *tree*2: 4028 positive candidates, 7484 negative ones;
3. *hugeTree*: 5628 positive candidates, 22454 negative ones;
4. *treeLast*: 1115 positive candidates, 10085 negative ones.

6.6 Final Notes

Developing the complete system, it clearly appeared how the segmentation plays a very important role for the detection step. In fact a good candidate segmentation is the base requirement for the classifier. Without it instead the classifier would more likely not recognize real people or an even worse case is that the candidate shaping (section 4.1) will not be able to meet the minimum requirements for giving to the classifier candidates representing real people.
For these reasons the tests have been developed in two ways. In the first one the complete system is tested. In the second one instead the results of the first test are adjusted in order to get rid of the errors deriving from the segmentation module. In order to do this a specific program has been created for counting by hand the number of segmentation errors.
Finally by adjusting a parameter deriving from the segmentation module, we tried to raise the performances of the complete system looking for drawbacks in the new procedure. Tests have been undertaken in order to evaluate also the performances of this last method.

Chapter 7

Test

7.1 Setup

The setup for the test is the following

1. fixed sensor;
2. simple environment;
3. daytime;
4. maximum four people in the scene at the same time;
5. soft and harsh occlusion occurs;
6. 608 frames captured and processed.

7.2 tree

7.2.1 Results of the Complete System

TAR	FAR	TRR	FRR	n_{fp}	n_{det}	n_{fp} %
0	0	1	1	0	0	0
0,059803	0	1	0,9402	0	79	0
0,29447	0,012987	0,98701	0,70553	8	397	2,0151
0,48524	0,041009	0,95899	0,51476	26	667	3,8981
0,55488	0,057364	0,94264	0,44512	37	770	4,8052
0,60106	0,099259	0,90074	0,39894	67	861	7,7816
0,63437	0,19683	0,80317	0,36563	149	987	15,096
0,65405	0,33406	0,66594	0,34595	305	1169	26,091
0,67222	0,48865	0,51135	0,32778	581	1469	39,551
0,68206	0,60825	0,39175	0,31794	944	1845	51,165
0,69114	0,73074	0,26926	0,30886	1650	2563	64,378
0,69947	0,78539	0,21461	0,30053	2225	3149	70,657

FPPF	n_{miss}	n_{people}	n_{miss} %	MPF	$threshold_{classifier}$
0	1321	1321	100	2,1727	40
0	1242	1321	94,02	2,0428	20
0,013158	932	1321	70,553	1,5329	10
0,042763	680	1321	51,476	1,1184	0
0,060855	588	1321	44,512	0,96711	-5
0,1102	527	1321	39,894	0,86678	-10
0,24507	483	1321	36,563	0,79441	-15
0,50164	457	1321	34,595	0,75164	-20
0,95559	433	1321	32,778	0,71217	-25
1,5526	420	1321	31,794	0,69079	-30
2,7138	408	1321	30,886	0,67105	-40
3,6595	397	1321	30,053	0,65296	-60

7.2.2 Results Without Segmentation Errors

TAR	FAR	TRR	FRR	n_{fp}	n_{det}	n_{fp} %
0	0	1	1	0	0	0
0,083246	0	1	0,91675	0	79	0
0,40991	0,012987	0,98701	0,59009	8	397	2,0151
0,67545	0,041009	0,95899	0,32455	26	667	3,8981
0,77239	0,057364	0,94264	0,22761	37	770	4,8052
0,83667	0,099259	0,90074	0,16333	67	861	7,7816
0,88303	0,19683	0,80317	0,11697	149	987	15,096
0,91043	0,33406	0,66594	0,089568	305	1169	26,091
0,93572	0,48865	0,51135	0,064278	581	1469	39,551
0,94942	0,60825	0,39175	0,05058	944	1845	51,165
0,96207	0,73074	0,26926	0,037935	1650	2563	64,378
0,97366	0,78539	0,21461	0,026344	2225	3149	70,657

FPPF	n_{miss}	n_{people}	n_{miss} %	MPF	$threshold_{classifier}$
0	949	949	100	1,5609	40
0	870	949	91,675	1,4309	20
0,013158	560	949	59,009	0,92105	10
0,042763	308	949	32,455	0,50658	0
0,060855	216	949	22,761	0,35526	-5
0,1102	155	949	16,333	0,25493	-10
0,24507	111	949	11,697	0,18257	-15
0,50164	85	949	8,9568	0,1398	-20
0,95559	61	949	6,4278	0,10033	-25
1,5526	48	949	5,058	0,078947	-30
2,7138	36	949	3,7935	0,059211	-40
3,6595	25	949	2,6344	0,041118	-60

7.2.3 TAR, ROC and DET

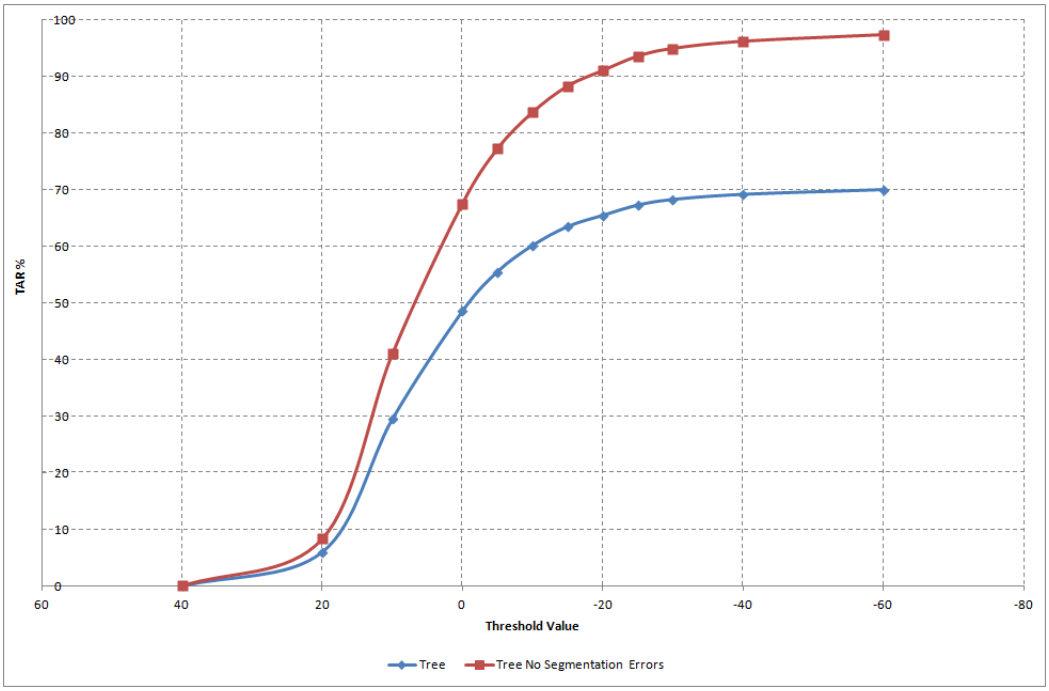

Figure 7.1. TAR curves based on the threshold values for the classifier *tree*.

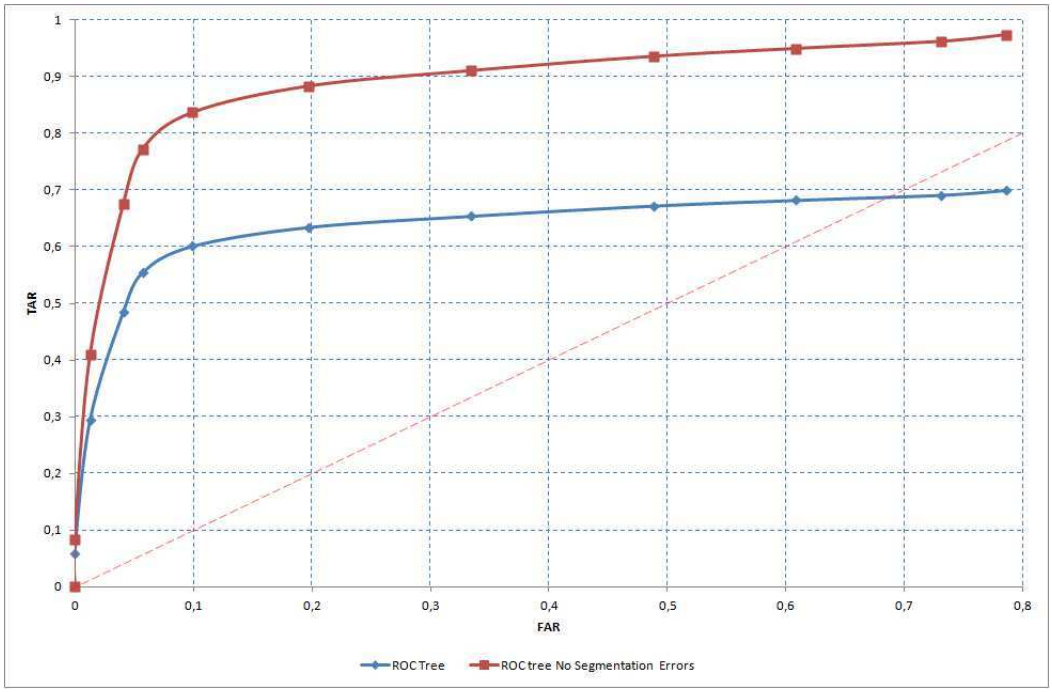

Figure 7.2. ROC curves for the classifier *tree*.

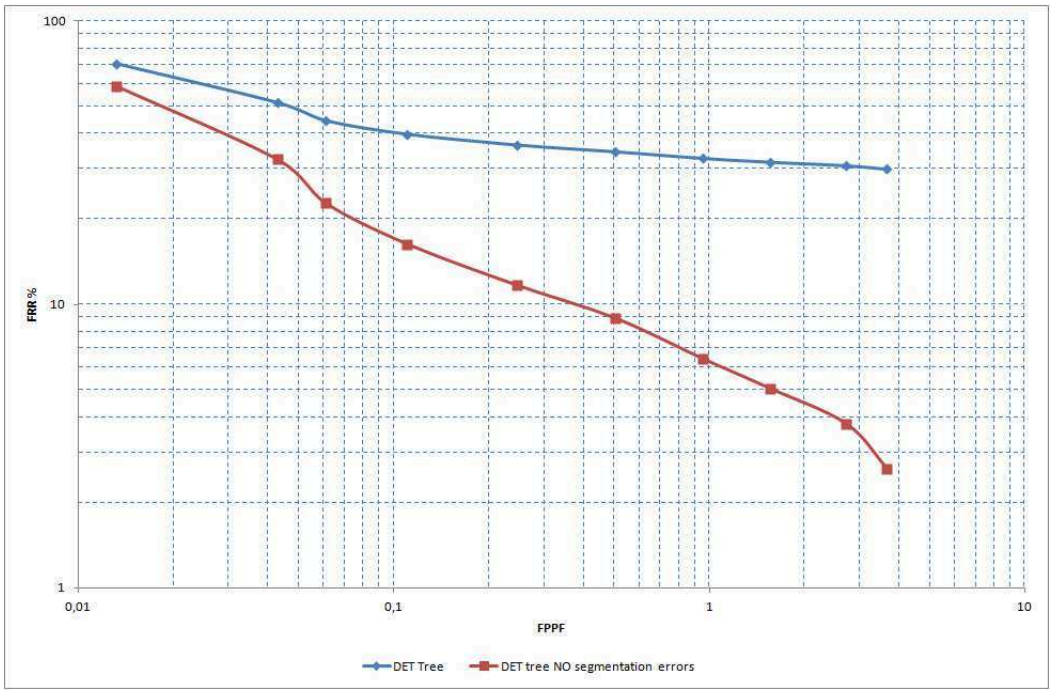

Figure 7.3. DET curves for the classifier *tree*.

7.3 tree2

7.3.1 Results of the Complete System

TAR	FAR	TRR	FRR	n_{fp}	n_{det}	n_{fp} %
0	0	1	1	0	0	0
0,053747	0,00491	0,99509	0,94625	3	74	4,0541
0,37093	0,022508	0,97749	0,62907	14	504	2,7778
0,58743	0,051482	0,94852	0,41257	33	809	4,0791
0,62755	0,073171	0,92683	0,37245	48	877	5,4732
0,65632	0,13759	0,86241	0,34368	97	964	10,062
0,67752	0,22646	0,77354	0,32248	178	1073	16,589
0,69114	0,36	0,64	0,30886	342	1255	27,251
0,70326	0,53374	0,46626	0,29674	696	1625	42,831
0,70704	0,66052	0,33948	0,29296	1183	2117	55,881
0,71461	0,77843	0,22157	0,28539	2136	3080	69,351
0,71612	0,8033	0,1967	0,28388	2483	3429	72,412

FPPF	n_{miss}	n_{people}	n_{miss} %	MPF	$threshold_{classifier}$
0	1321	1321	100	2,1727	40
0,0049342	1250	1321	94,625	2,0559	20
0,023026	831	1321	62,907	1,3668	10
0,054276	545	1321	41,257	0,89638	0
0,078947	492	1321	37,245	0,80921	-5
0,15954	454	1321	34,368	0,74671	-10
0,29276	426	1321	32,248	0,70066	-15
0,5625	408	1321	30,886	0,67105	-20
1,1447	392	1321	29,674	0,64474	-25
1,9457	387	1321	29,296	0,63651	-30
3,5132	377	1321	28,539	0,62007	-40
4,0839	375	1321	28,388	0,61678	-60

7.3.2 Results Without Segmentation Errors

TAR	FAR	TRR	FRR	n_{fp}	n_{det}	n_{fp} %
0	0	1	1	0	0	0
0,074816	0,00491	0,99509	0,92518	3	74	4,0541
0,51633	0,022508	0,97749	0,48367	14	504	2,7778
0,8177	0,051482	0,94852	0,1823	33	809	4,0791
0,87355	0,073171	0,92683	0,12645	48	877	5,4732
0,91359	0,13759	0,86241	0,086407	97	964	10,062
0,9431	0,22646	0,77354	0,056902	178	1073	16,589
0,96207	0,36	0,64	0,037935	342	1255	27,251
0,97893	0,53374	0,46626	0,021075	696	1625	42,831
0,98419	0,66052	0,33948	0,015806	1183	2117	55,881
0,99473	0,77843	0,22157	0,0052687	2136	3080	69,351
0,99684	0,8033	0,1967	0,0031612	2483	3429	72,412

FPPF	n_{miss}	n_{people}	n_{miss} %	MPF	$threshold_{classifier}$
0	949	949	100	1,5609	40
0,0049342	878	949	92,518	1,4441	20
0,023026	459	949	48,367	0,75493	10
0,054276	173	949	18,23	0,28454	0
0,078947	120	949	12,645	0,19737	-5
0,15954	82	949	8,6407	0,13487	-10
0,29276	54	949	5,6902	0,088816	-15
0,5625	36	949	3,7935	0,059211	-20
1,1447	20	949	2,1075	0,032895	-25
1,9457	15	949	1,5806	0,024671	-30
3,5132	5	949	0,52687	0,0082237	-40
4,0839	3	949	0,31612	0,0049342	-60

7.3.3 TAR, ROC and DET

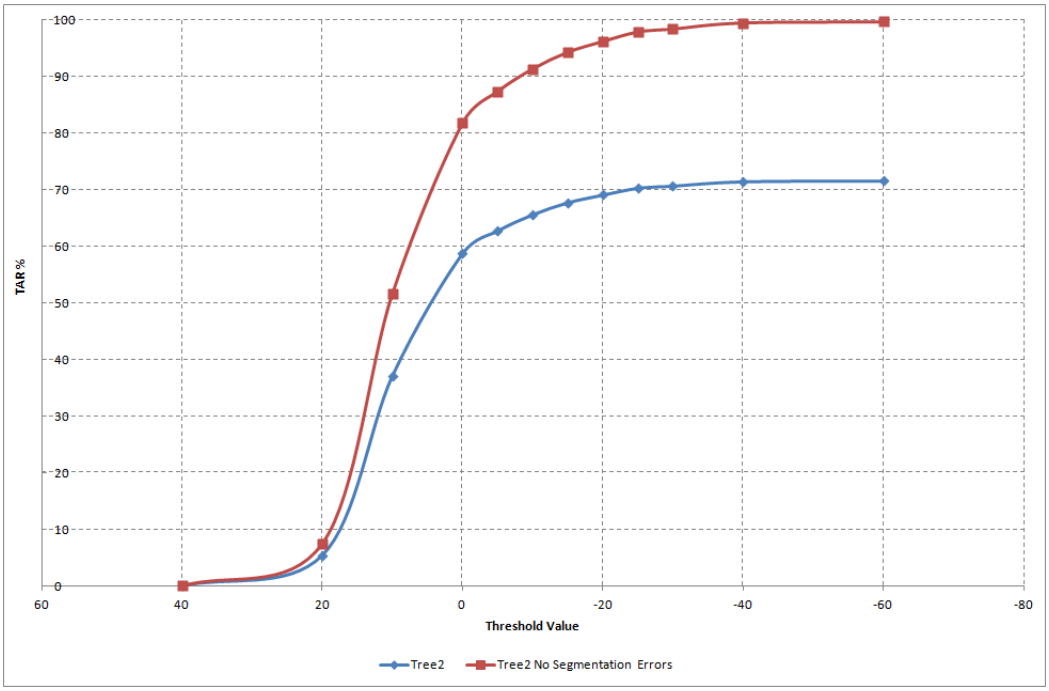

Figure 7.4. TAR curves based on the threshold values for the classifier *tree*2.

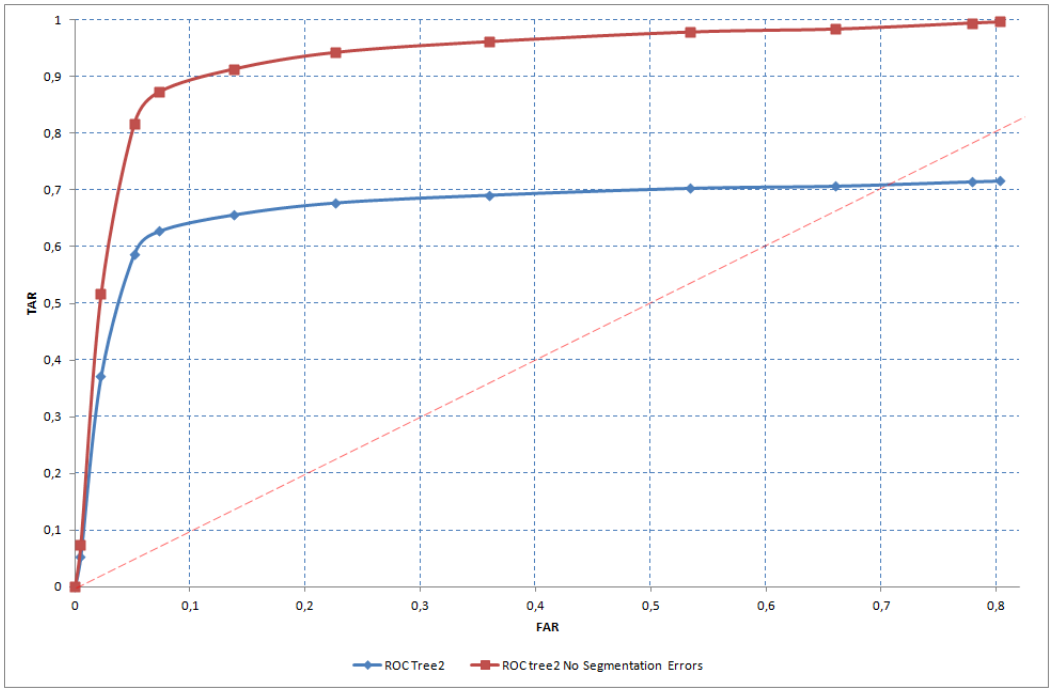

Figure 7.5. ROC curves for the classifier *tree*2.

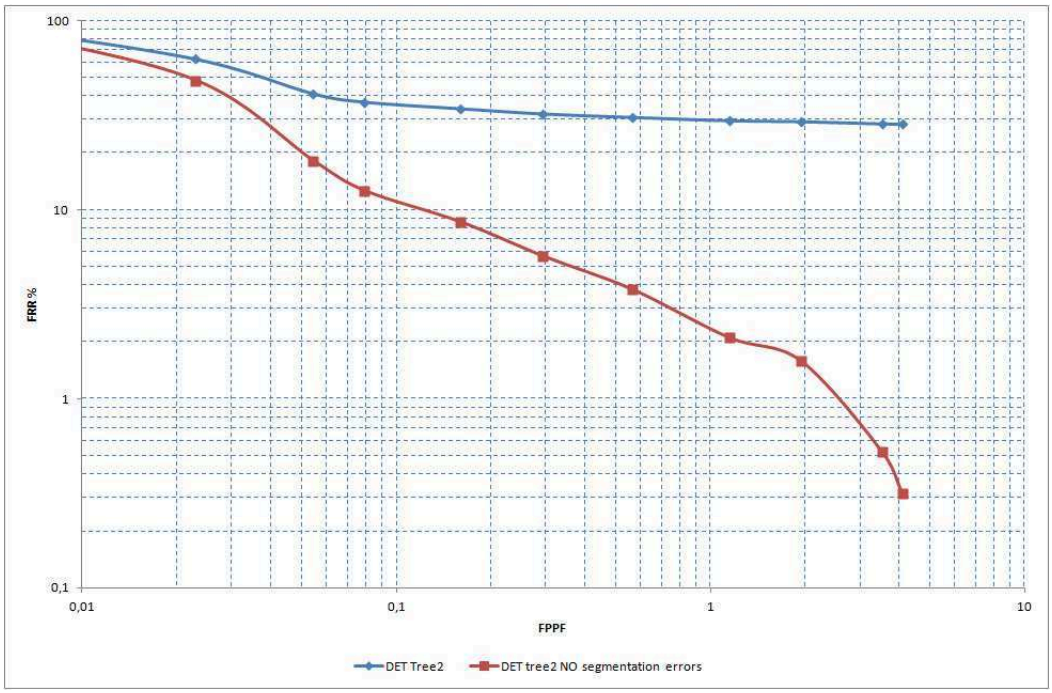

Figure 7.6. DET curves for the classifier *tree*2.

7.4 hugeTree

7.4.1 Results of the Complete System

TAR	FAR	TRR	FRR	n_{fp}	n_{det}	n_{fp} %
0	0	1	1	0	0	0
0,017411	0,001642	0,99836	0,98259	1	24	4,1667
0,2377	0,011382	0,98862	0,7623	7	321	2,1807
0,53747	0,051482	0,94852	0,46253	33	743	4,4415
0,60484	0,073171	0,92683	0,39516	48	847	5,6671
0,65632	0,12894	0,87106	0,34368	90	957	9,4044
0,68055	0,2294	0,7706	0,31945	181	1080	16,759
0,69114	0,41369	0,58631	0,30886	429	1342	31,967
0,7025	0,58156	0,41844	0,2975	845	1773	47,659
0,7078	0,69324	0,30676	0,2922	1374	2309	59,506
0,71461	0,78584	0,21416	0,28539	2231	3175	70,268
0,71612	0,80419	0,19581	0,28388	2497	3443	72,524

FPPF	n_{miss}	n_{people}	n_{miss} %	MPF	$threshold_{classifier}$
0	1321	1321	100	2,1727	40
0,0016447	1298	1321	98,259	2,1349	20
0,011513	1007	1321	76,23	1,6562	10
0,054276	611	1321	46,253	1,0049	0
0,078947	522	1321	39,516	0,85855	-5
0,14803	454	1321	34,368	0,74671	-10
0,2977	422	1321	31,945	0,69408	-15
0,70559	408	1321	30,886	0,67105	-20
1,3898	393	1321	29,75	0,64638	-25
2,2599	386	1321	29,22	0,63487	-30
3,6694	377	1321	28,539	0,62007	-40
4,1069	375	1321	28,388	0,61678	-60

7.4.2 Results Without Segmentation Errors

TAR	FAR	TRR	FRR	n_{fp}	n_{det}	n_{fp} %
0	0	1	1	0	0	0
0,024236	0,001642	0,99836	0,97576	1	24	4,1667
0,33087	0,011382	0,98862	0,66913	7	321	2,1807
0,74816	0,051482	0,94852	0,25184	33	743	4,4415
0,84194	0,073171	0,92683	0,15806	48	847	5,6671
0,91359	0,12894	0,87106	0,086407	90	957	9,4044
0,94731	0,2294	0,7706	0,052687	181	1080	16,759
0,96207	0,41369	0,58631	0,037935	429	1342	31,967
0,97787	0,58156	0,41844	0,022129	845	1773	47,659
0,98525	0,69324	0,30676	0,014752	1374	2309	59,506
0,99473	0,78584	0,21416	0,0052687	2231	3175	70,268
0,99684	0,80419	0,19581	0,0031612	2497	3443	72,524

FPPF	n_{miss}	n_{people}	n_{miss} %	MPF	$threshold_{classifier}$
0	949	949	100	1,5609	40
0,0016447	926	949	97,576	1,523	20
0,011513	635	949	66,913	1,0444	10
0,054276	239	949	25,184	0,39309	0
0,078947	150	949	15,806	0,24671	-5
0,14803	82	949	8,6407	0,13487	-10
0,2977	50	949	5,2687	0,082237	-15
0,70559	36	949	3,7935	0,059211	-20
1,3898	21	949	2,2129	0,034539	-25
2,2599	14	949	1,4752	0,023026	-30
3,6694	5	949	0,52687	0,0082237	-40
4,1069	3	949	0,31612	0,0049342	-60

7.4.3 TAR, ROC and DET

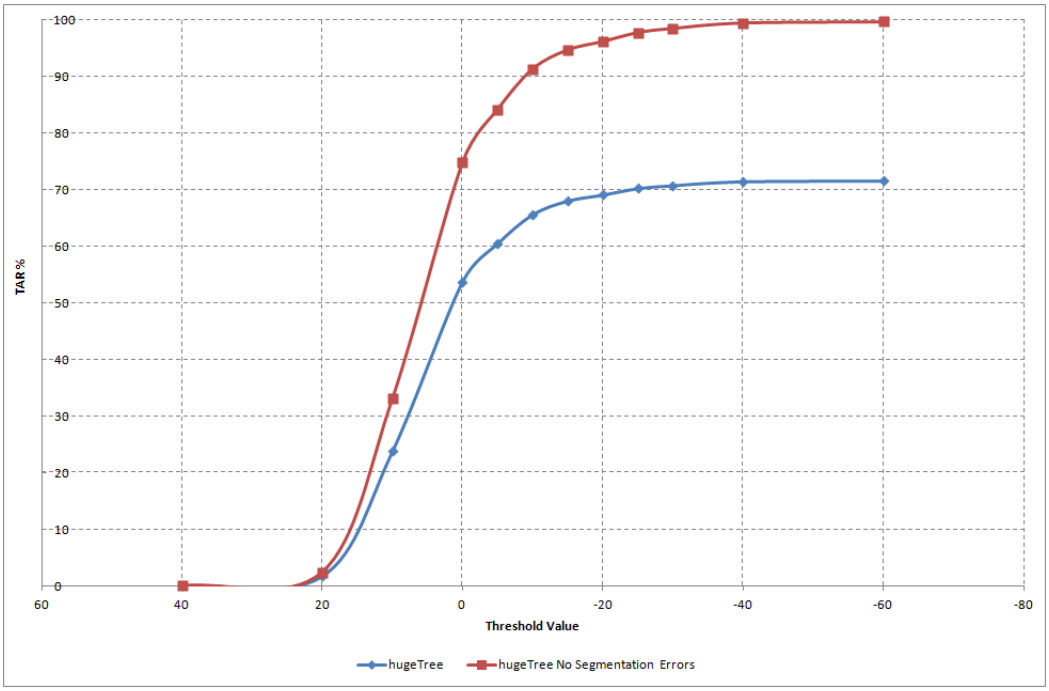

Figure 7.7. TAR curves based on the threshold values for the classifier *hugeTree*.

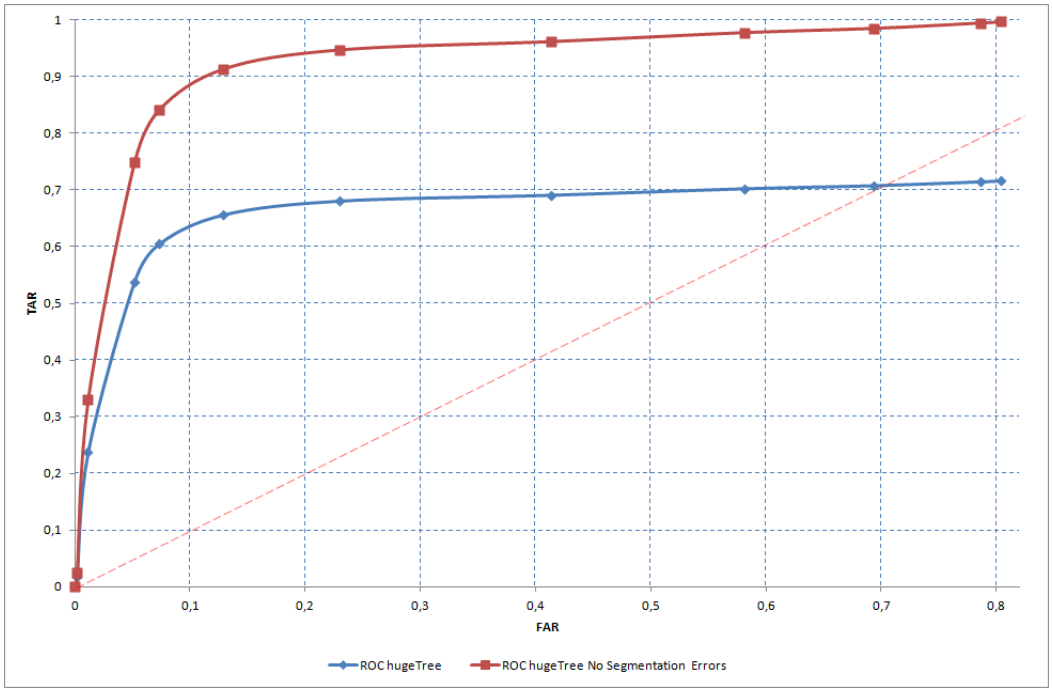

Figure 7.8. ROC curves for the classifier *hugeTree*.

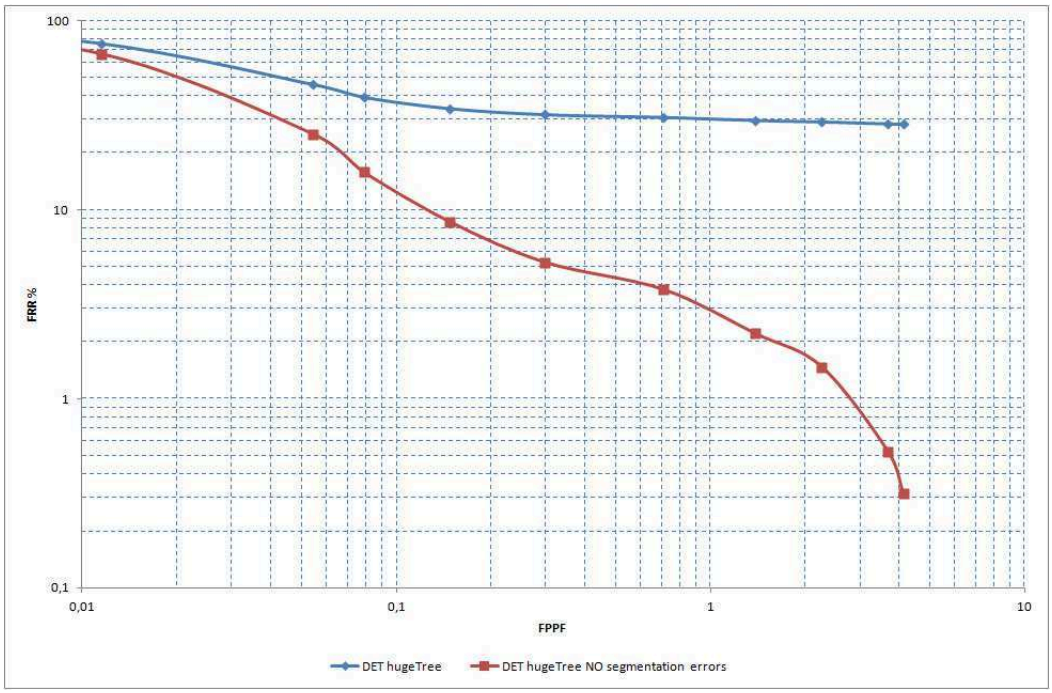

Figure 7.9. DET curves for the classifier *hugeTree*.

7.5 treeLast

7.5.1 Results of the Complete System

TAR	FAR	TRR	FRR	n_{fp}	n_{det}	n_{fp} %
0	0	1	1	0	0	0
0,000757	0	1	0,99924	0	1	0
0,033308	0,003279	0,99672	0,96669	2	46	4,3478
0,1461	0,003279	0,99672	0,8539	2	195	1,0256
0,22104	0,00491	0,99509	0,77896	3	295	1,0169
0,32627	0,014587	0,98541	0,67373	9	440	2,0455
0,49357	0,034921	0,96508	0,50643	22	674	3,2641
0,59652	0,14487	0,85513	0,40348	103	891	11,56
0,65254	0,44169	0,55831	0,34746	481	1343	35,815
0,6866	0,70699	0,29301	0,3134	1467	2374	61,794
0,70098	0,77915	0,22085	0,29902	2145	3071	69,847

FPPF	n_{miss}	n_{people}	n_{miss} %	MPF	$threshold_{classifier}$
0	1321	1321	100	2,1727	30
0	1320	1321	99,924	2,1711	20
0,0032895	1277	1321	96,669	2,1003	10
0,0032895	1128	1321	85,39	1,8553	0
0,0049342	1029	1321	77,896	1,6924	-5
0,014803	890	1321	67,373	1,4638	-10
0,036184	669	1321	50,643	1,1003	-20
0,16941	533	1321	40,348	0,87664	-30
0,79112	459	1321	34,746	0,75493	-40
2,4128	414	1321	31,34	0,68092	-50
3,528	395	1321	29,902	0,64967	-60

7.5.2 Results Without Segmentation Errors

TAR	FAR	TRR	FRR	n_{fp}	n_{det}	n_{fp} %
0	0	1	1	0	0	0
0,001054	0	1	0,99895	0	1	0
0,046365	0,003279	0,99672	0,95364	2	46	4,3478
0,20337	0,003279	0,99672	0,79663	2	195	1,0256
0,30769	0,00491	0,99509	0,69231	3	295	1,0169
0,45416	0,014587	0,98541	0,54584	9	440	2,0455
0,68704	0,034921	0,96508	0,31296	22	674	3,2641
0,83035	0,14487	0,85513	0,16965	103	891	11,56
0,90832	0,44169	0,55831	0,091675	481	1343	35,815
0,95574	0,70699	0,29301	0,044257	1467	2374	61,794
0,97576	0,77915	0,22085	0,024236	2145	3071	69,847

FPPF	n_{miss}	n_{people}	n_{miss} %	MPF	$threshold_{classifier}$
0	949	949	100	1,5609	30
0	948	949	99,895	1,5592	20
0,0032895	905	949	95,364	1,4885	10
0,0032895	756	949	79,663	1,2434	0
0,0049342	657	949	69,231	1,0806	-5
0,014803	518	949	54,584	0,85197	-10
0,036184	297	949	31,296	0,48849	-20
0,16941	161	949	16,965	0,2648	-30
0,79112	87	949	9,1675	0,14309	-40
2,4128	42	949	4,4257	0,069079	-50
3,528	23	949	2,4236	0,037829	-60

7.5.3 TAR, ROC and DET

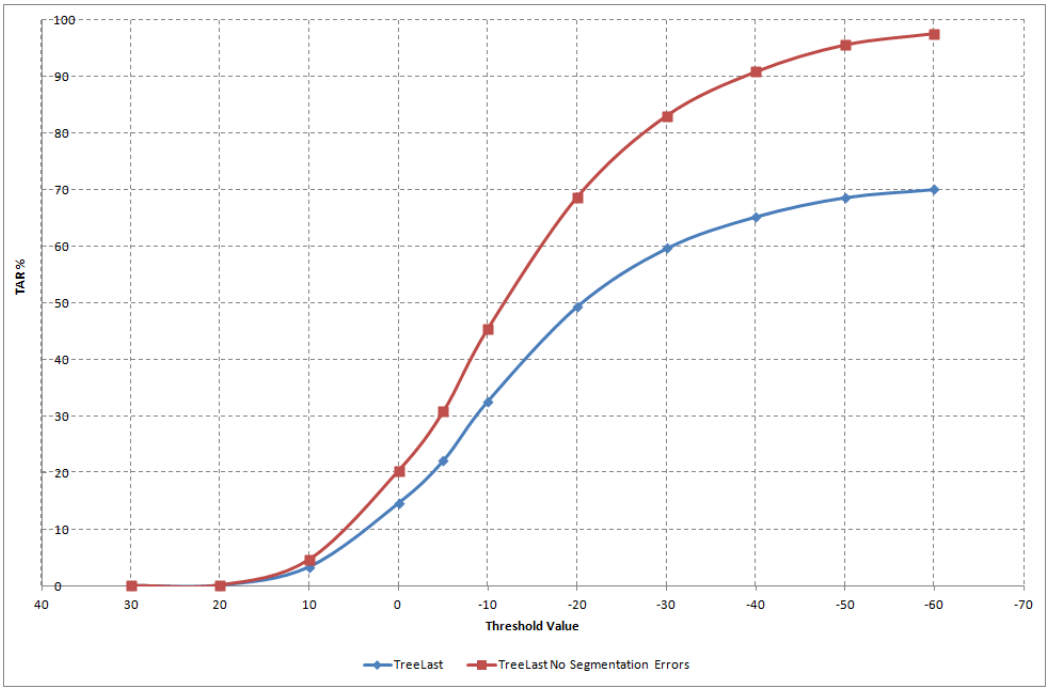

Figure 7.10. TAR curves based on the threshold values for the classifier *treeLast*.

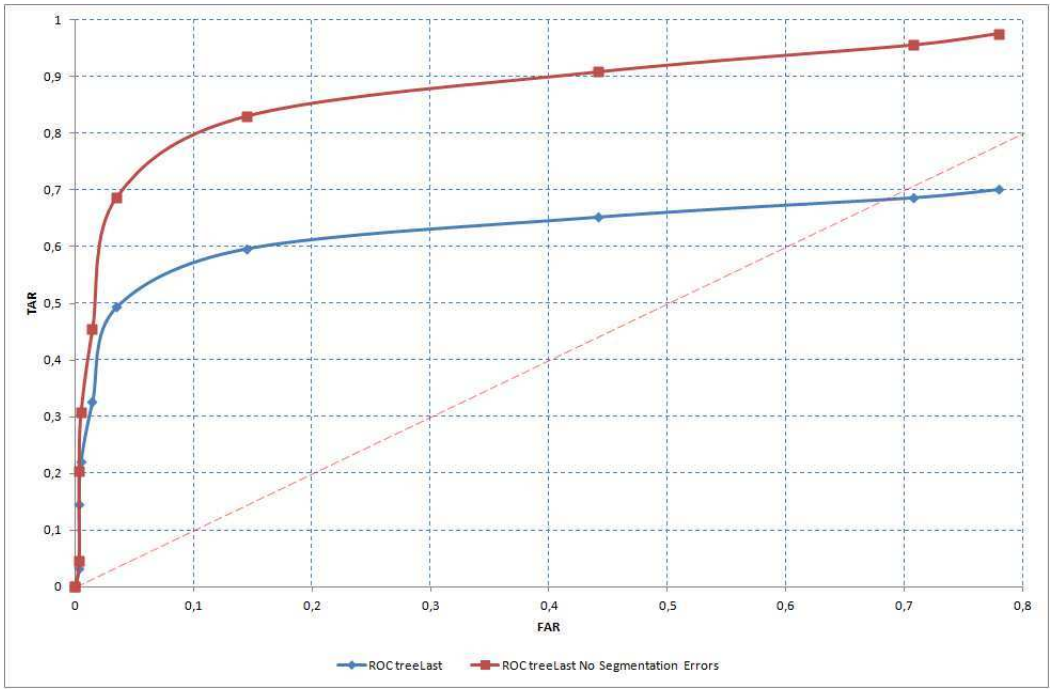

Figure 7.11. ROC curves for the classifier *treeLast*.

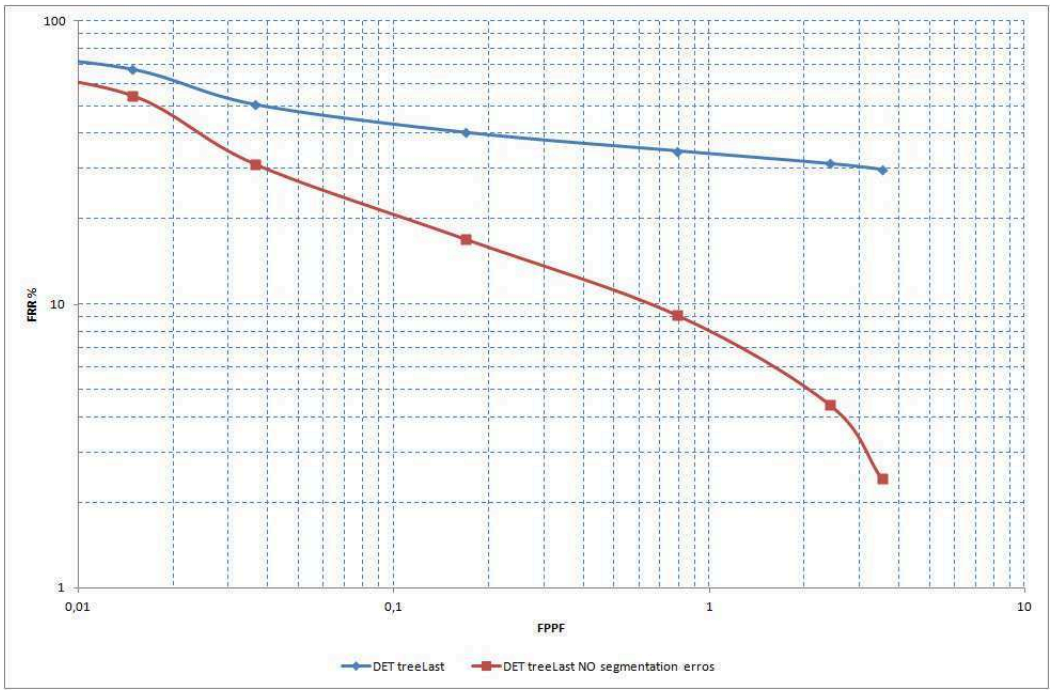

Figure 7.12. DET curves for the classifier *treeLast*.

7.6 Classifiers Comparison

The classifiers *hugeTree* and *tree*2 among all the classifiers showed the best results. In particular *tree*2 performed also slightly better with respect to *hugeTree*.
This result is clearly showed by the ROC curves in figures 7.16 and 7.17. In fact those two classifiers come closer to the optimal point ($TAR = 1,\ FAR = 0$) and are all the time above the other two curves. Among them the ROC curves for *tree*2 and *hugeTree* are basically identically for values of FAR greater than 0.1 whereas under that value the first one performs better.
The DET curves in figure 7.18 instead give no useful informations for comparing the classifiers. In fact they show that the various classifiers perform almost identically.
Additionally we will report also the TAR curves both with and without segmentation errors (figures 7.13 and 7.14), and the TRR ones (figure 7.15). We can also now explain why we don't need to create a TRR curve without considering segmentation errors for each classifier.
The TRR curve depends just on the parameters TRR and $threshold_{classifier}$ that don't depend on the value of misses (the parameter affected by the segmentation problems) as explained in section 6.2.

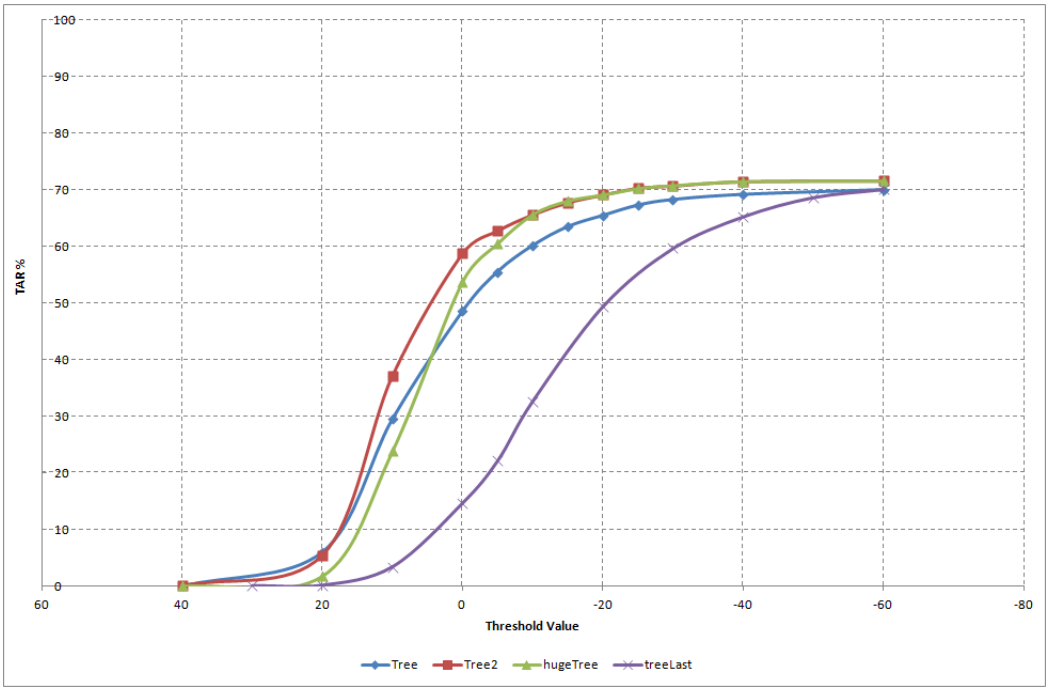

Figure 7.13. TAR curves based on the threshold values for all the classifiers.

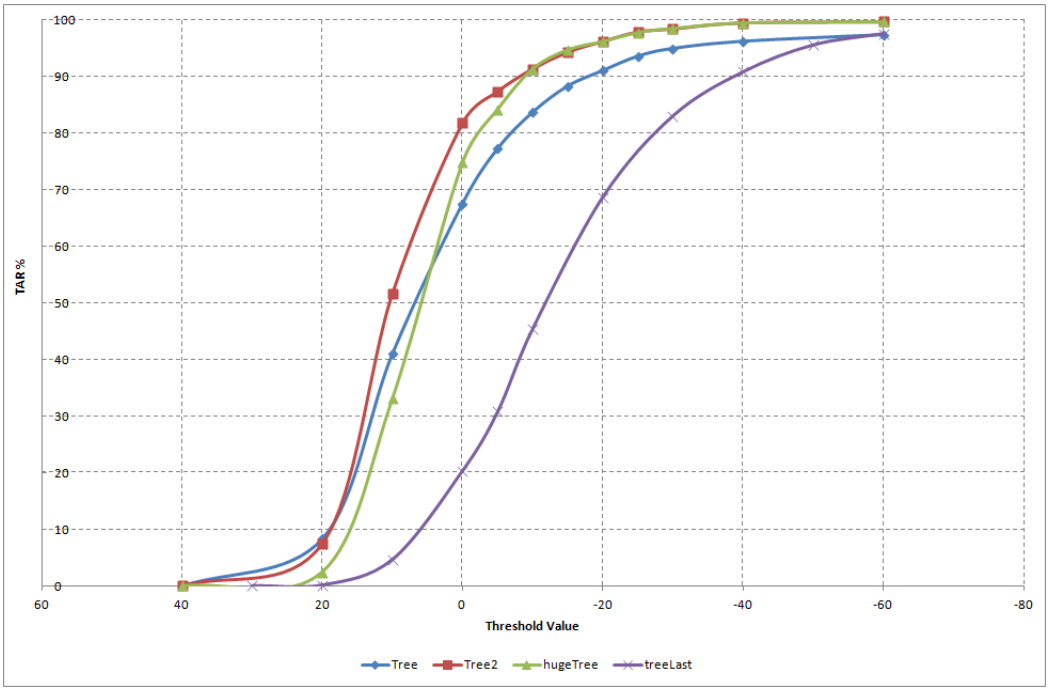

Figure 7.14. TAR curves based on the threshold values for all the classifiers without segmentation errors.

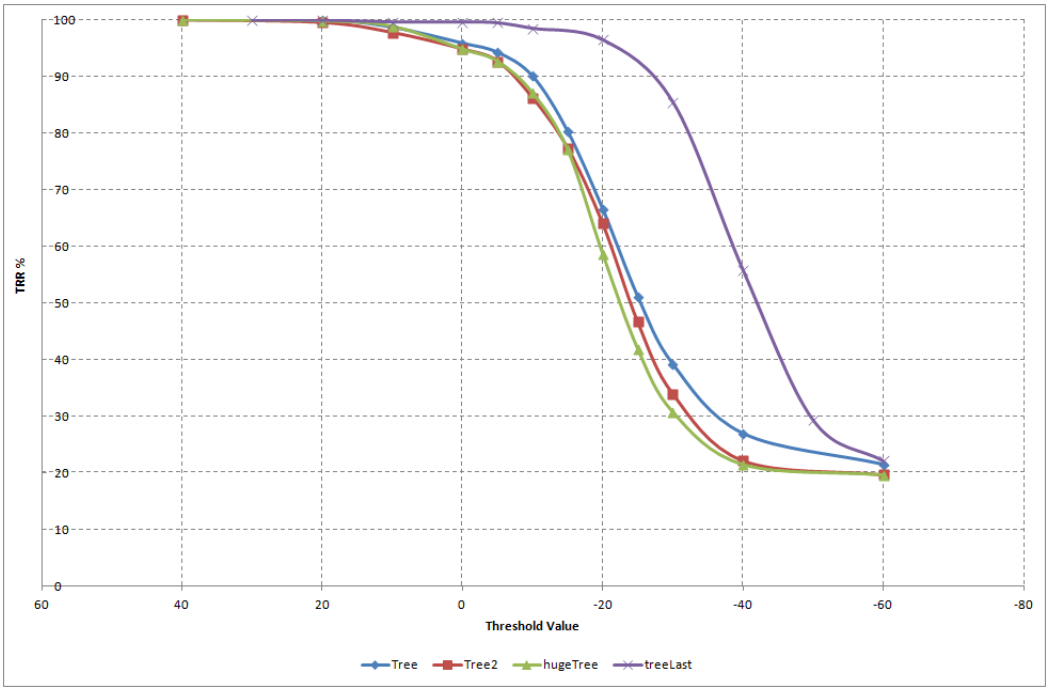

Figure 7.15. TRR curves based on the threshold values for all the classifiers.

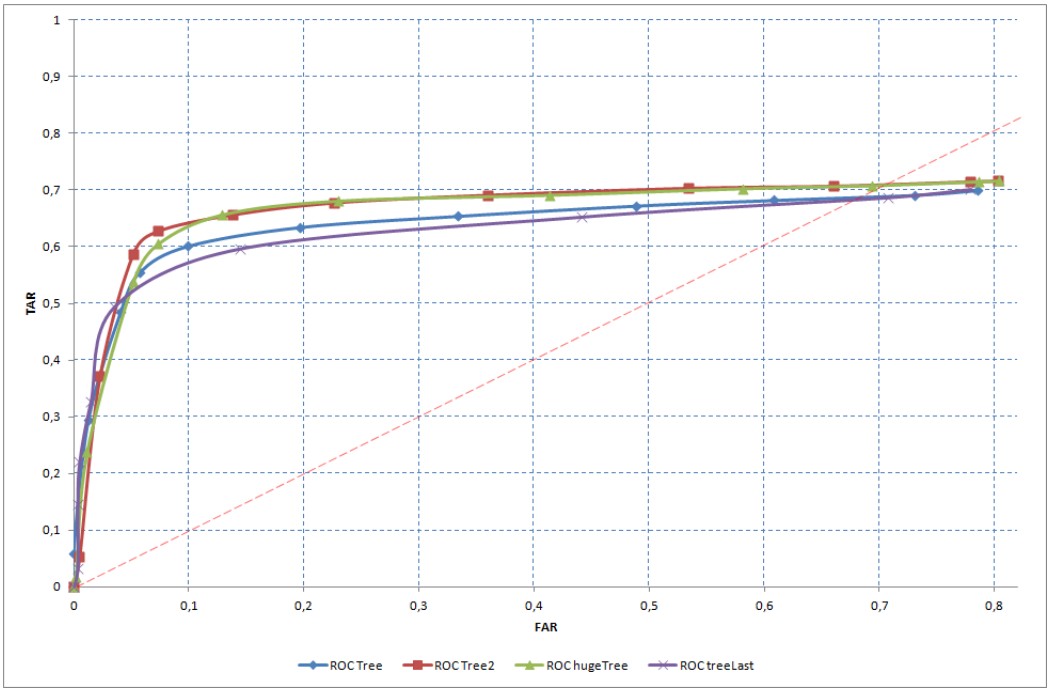

Figure 7.16. ROC curves for all the classifiers.

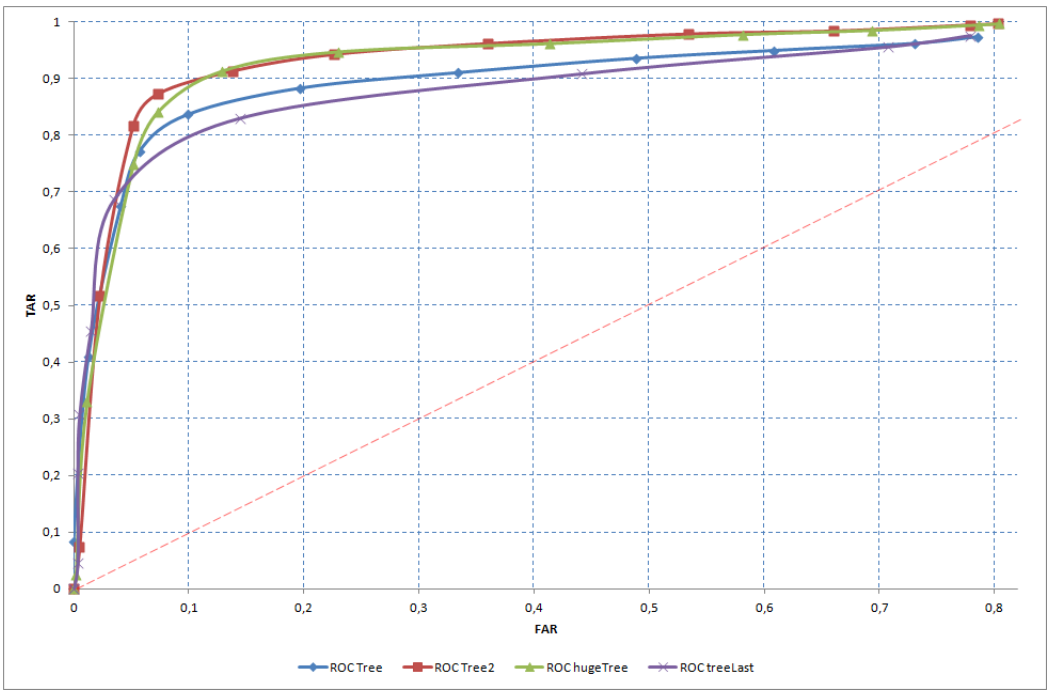

Figure 7.17. ROC curves for all the classifiers without segmentation errors.

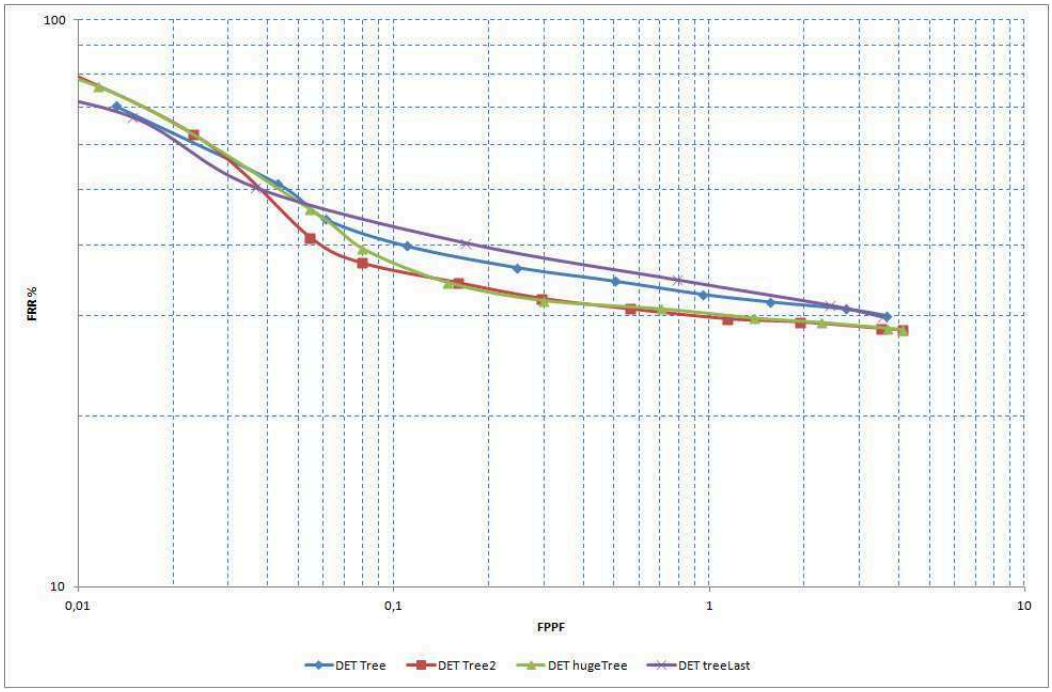

Figure 7.18. DET curves for all the classifiers.

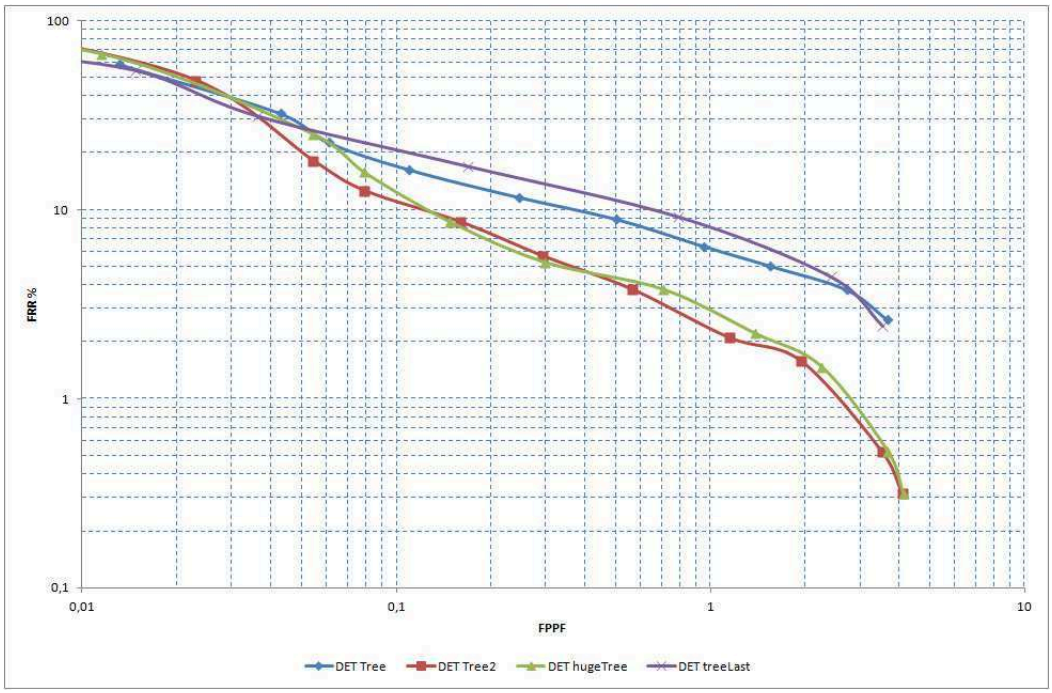

Figure 7.19. DET curves for all the classifiers without segmentation errors.

7.7 Shape Parameter

In order to raise the performances of the complete system, we can change a parameter deriving from the section 4.1, that is the *threshold_height*.
In fact, on the previous tests, we used a quite strict *threshold_height* value. In particular:

1. if the average depth of the segmented layer was smaller than 1.5 meters, we looked for candidates with *real_height* greater than 50 *cm*;

2. if the average depth of the segmented layer was greater than 1.5 meters, we looked for candidates with *real_height* greater than 90 *cm*.

For the following tests we changed this parameter so that for every candidate the only requirement is a *real_height* greater than 50 *cm*, independently from the average depth of the segmented layer. As we can see from figures 7.20, 7.21, 7.22, 7.23, 7.24, 7.25, 7.26, the classifier that on the whole showed the best performances is again the *tree*2 one, even if, for values of FAR smaller than 5%, *tree*2 doesn't show the best performances on the ROC curves.

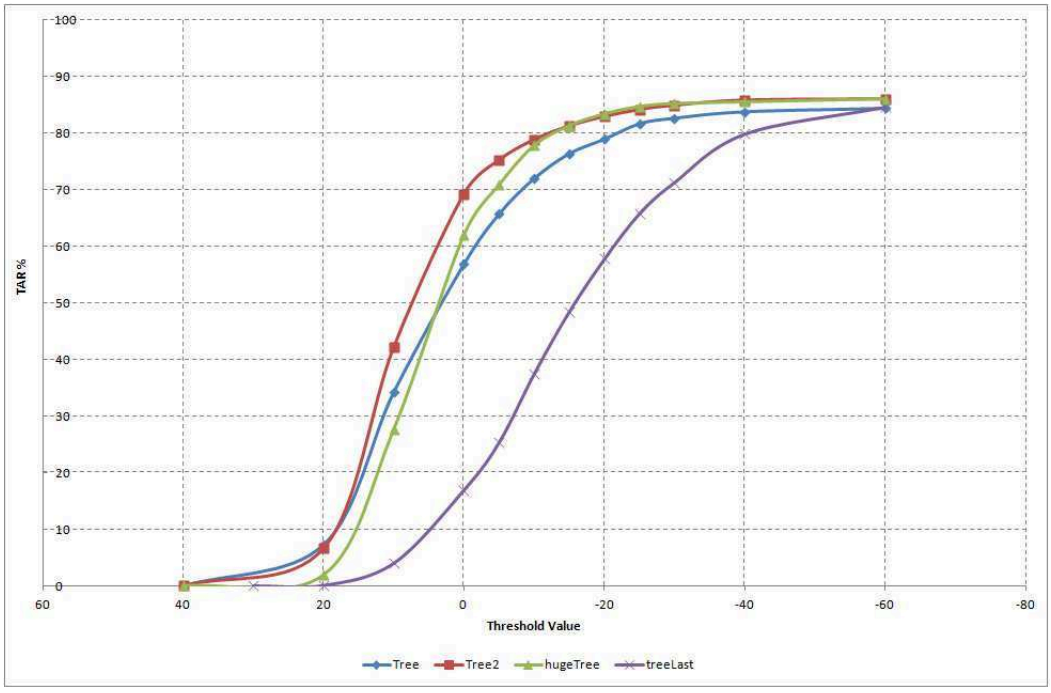

Figure 7.20. TAR curves based on the threshold values for all the classifiers.

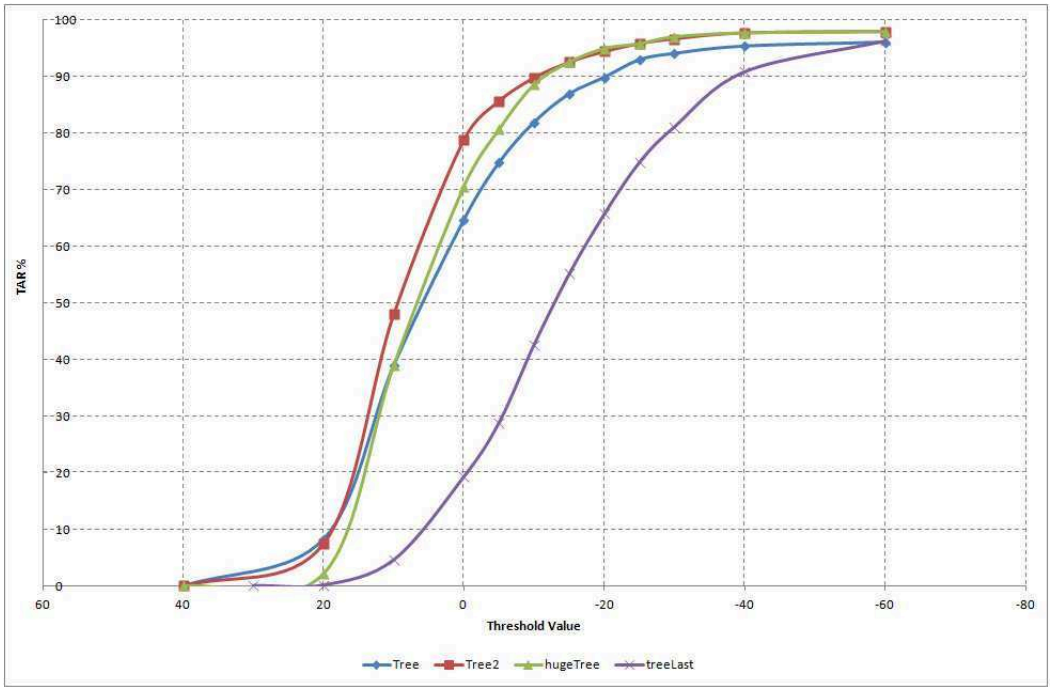

Figure 7.21. TAR curves based on the threshold values for all the classifiers without segmentation errors.

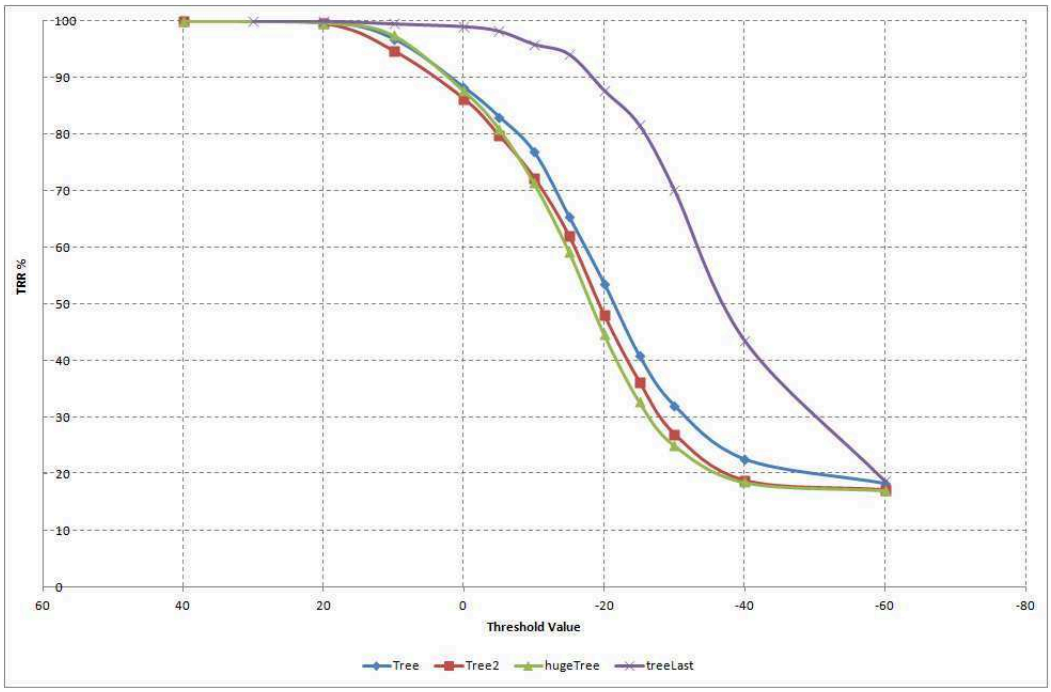

Figure 7.22. TRR curves based on the threshold values for all the classifiers.

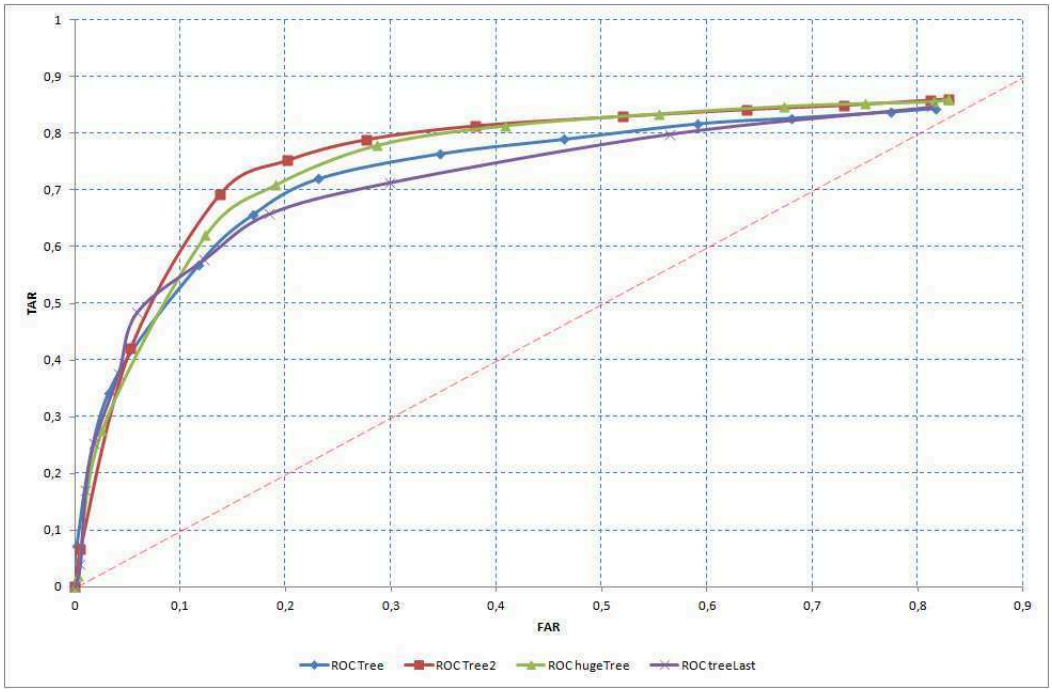

Figure 7.23. ROC curves for all the classifiers.

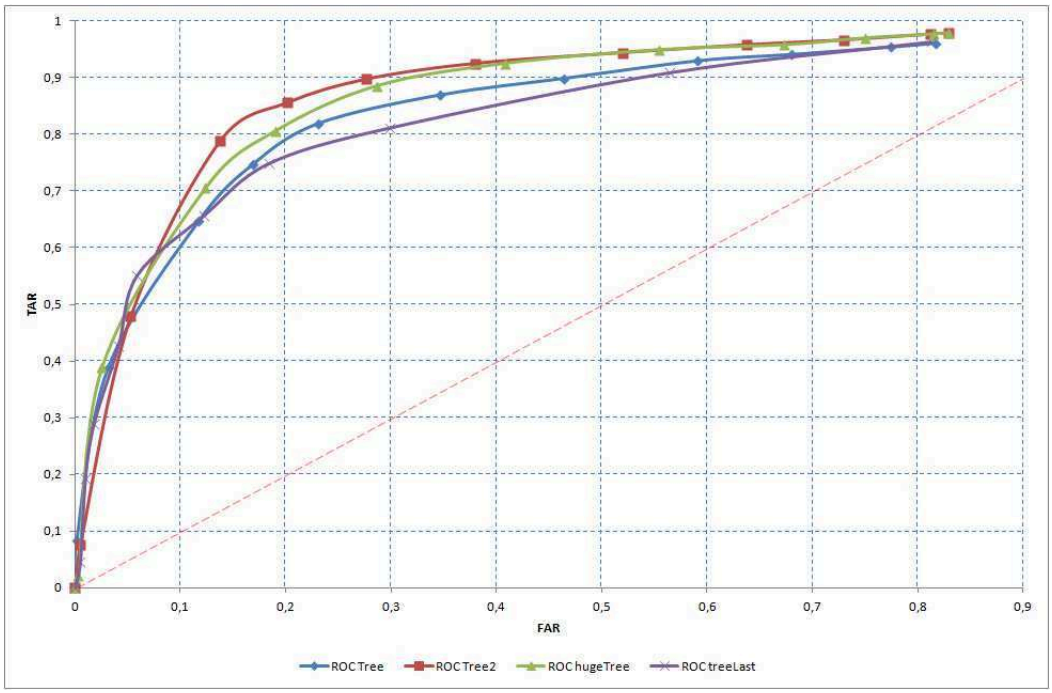

Figure 7.24. ROC curves for all the classifiers without segmentation errors.

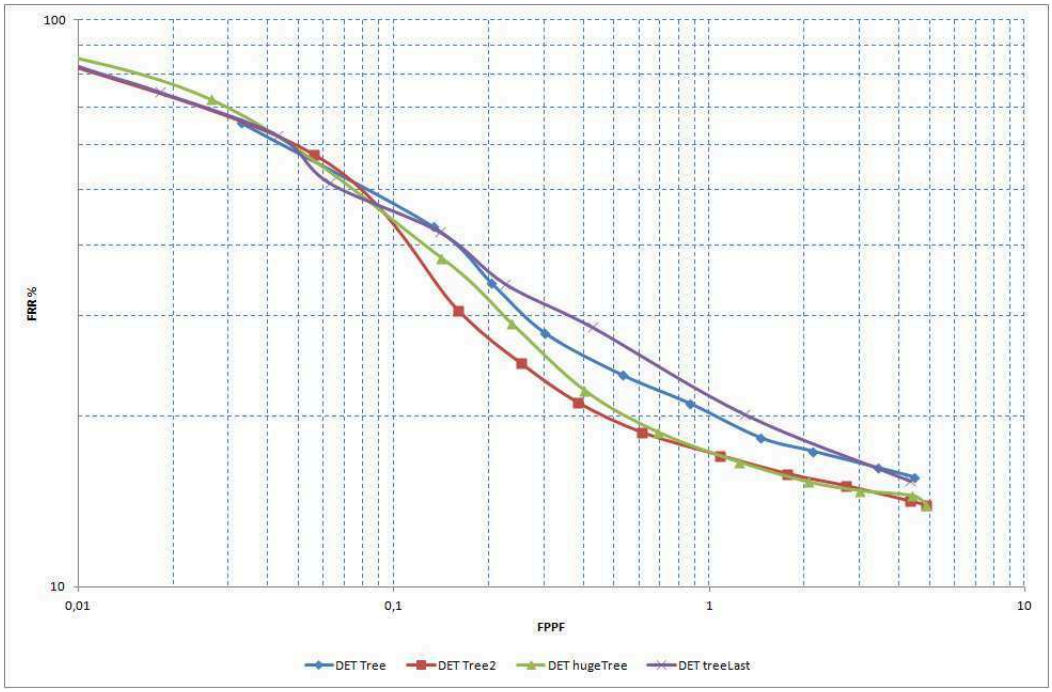

Figure 7.25. DET curves for all the classifiers.

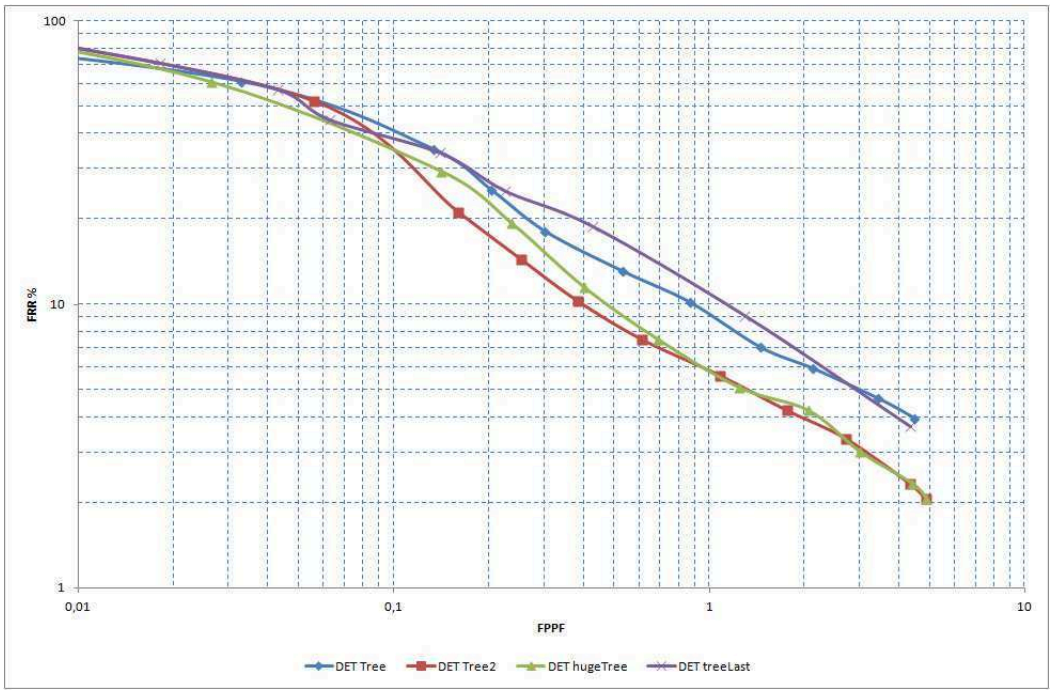

Figure 7.26. DET curves for all the classifiers without segmentation errors.

So we can now make a comparison between the performances of the classifier $tree2$ with the strict shape parameter and the ones of the same classifier but with the changed $threshold\_height$.
The comparison for the TAR curves of the complete system (figure 7.27) shows clearly how the performances have been raised. In fact on the parameterized test the maximum value of true positive is of 86% with respect to the 72% of the original one. As expected, instead, we can see on figure 7.28 that the two different tests perform almost identically when we are considering segmentation errors.
The ROC curves are more interesting. We can see that for the complete system (figure 7.29), the less strict threshold works better for FAR values smaller than roughly 13%. After this point instead the strict version is more efficient. However the shapes of the curves seem to show that potentially the strict version of the tests should perform better than the less strict one. This is confirmed when taking into account segmentation errors. In fact figure 7.30 shows how, in this case, the strict version works all the time better. The explanation for this phenomenon is quite simple actually. By being less strict on the height threshold value, we achieve two results. The first one is that we actually get rid of some segmentation problems whereas the second one is that we give to the classifier a lot more candidates which present a higher probability of not being a person. The probability for a candidate with height of 60 cm being a segmentation problem is lower than the one of being an object.
Finally the DET curves show also a similar behavior with respect to the ROC ones. In fact, for the complete system, the classifier with the strict $height\_threshold$ performs better for values of $FPPF$ smaller than roughly 0.12 whereas after this point, the parameterized one becomes better (figure 7.31). If we consider instead also the segmentation errors, the original tests show again continuously the best performances (figure 7.32).

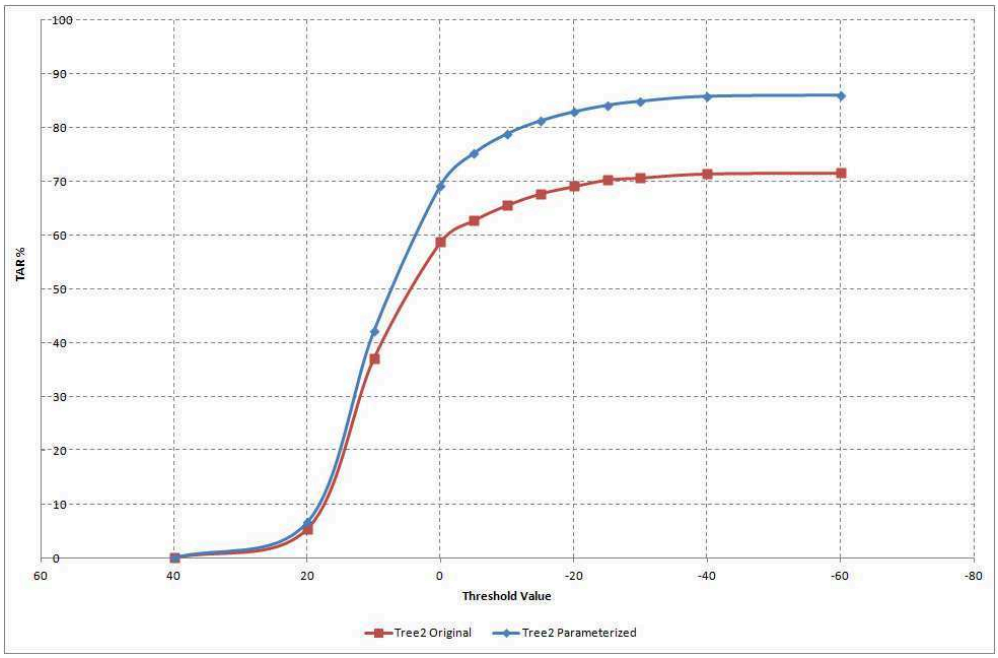

Figure 7.27. Comparison of the TAR curves for the classifier *tree*2 in the two situation described.

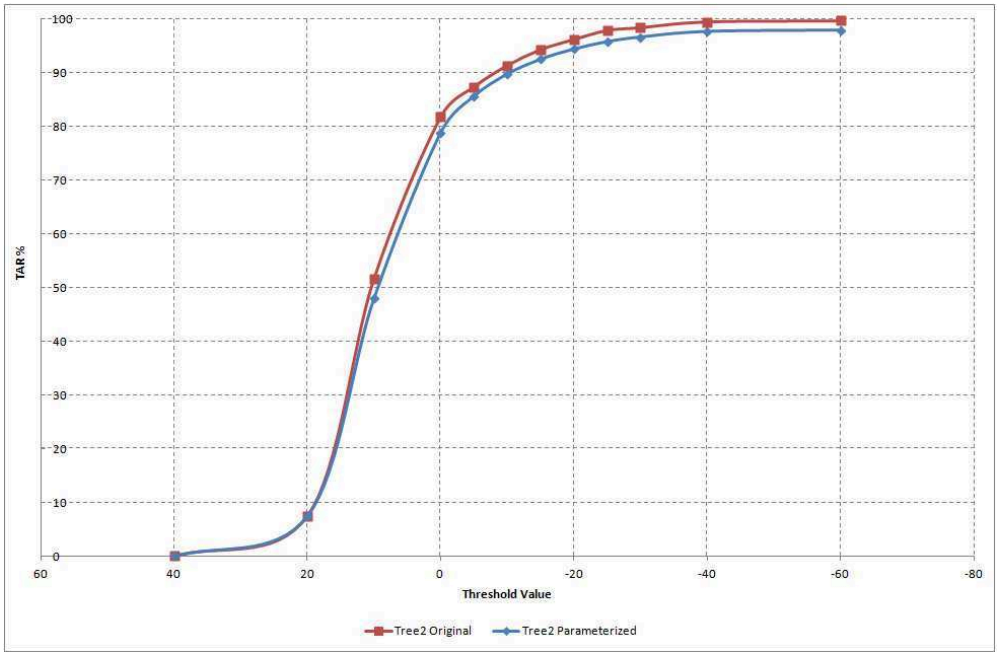

Figure 7.28. Comparison of the TAR curves for the classifier *tree*2 in the two situation described without segmentation errors.

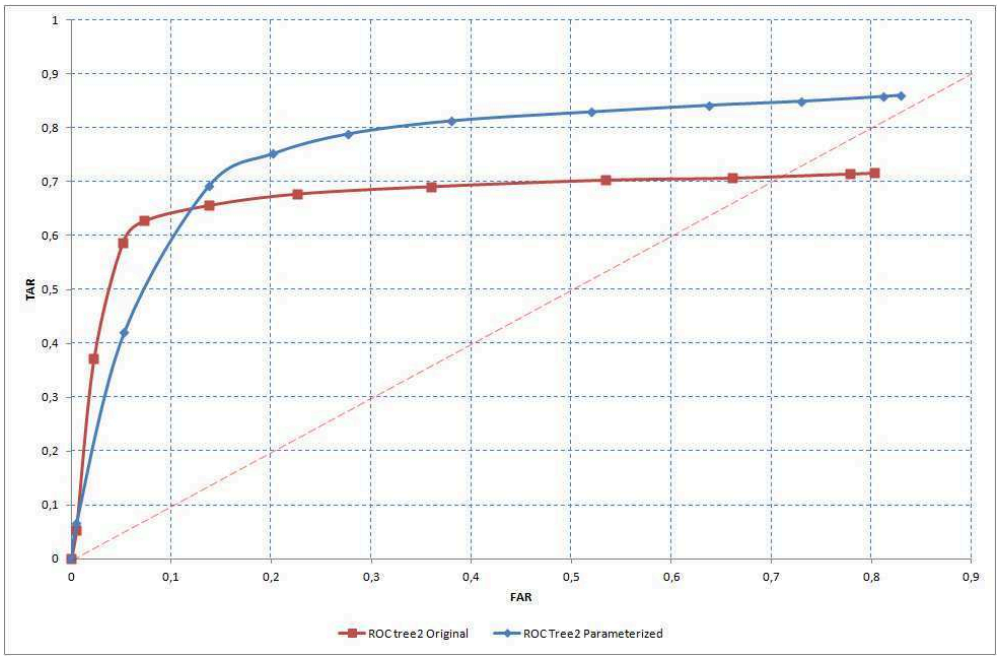

Figure 7.29. Comparison of the ROC curves for the classifier *tree*2 in the two situation described.

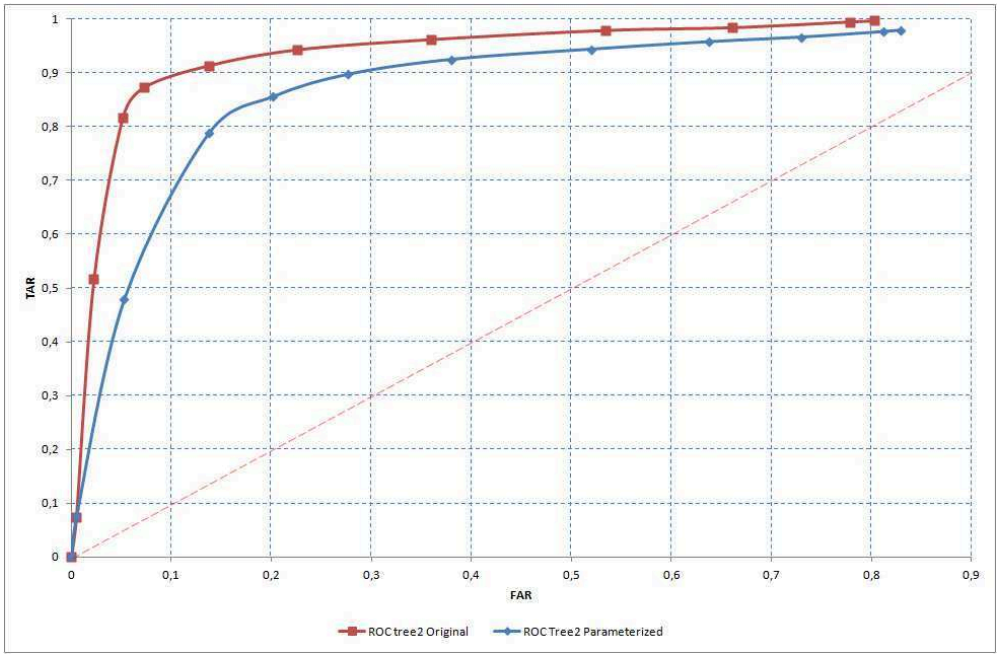

Figure 7.30. Comparison of the ROC curves for the classifier *tree*2 in the two situation described without segmentation errors.

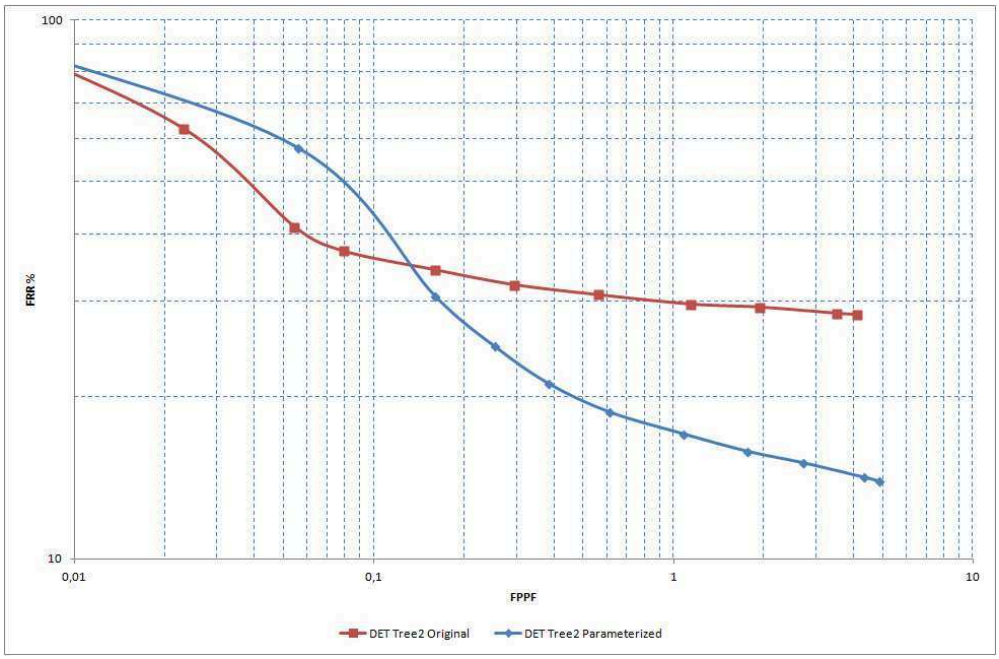

Figure 7.31. Comparison of the DET curves for the classifier *tree*2 in the two situation described.

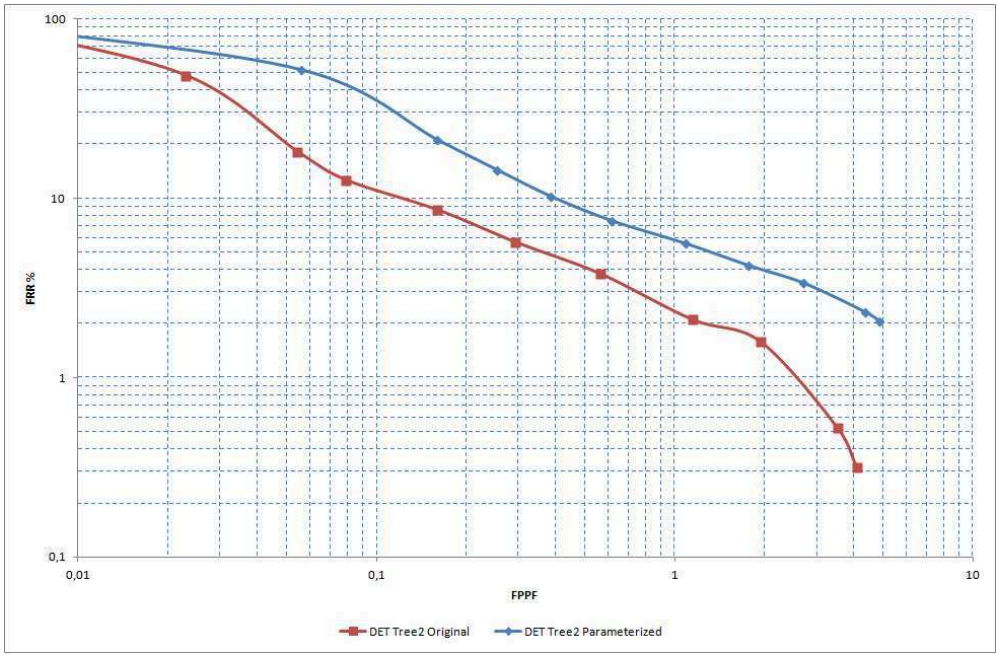

Figure 7.32. Comparison of the DET curves for the classifier *tree*2 in the two situation described without segmentation errors.

7.8 Tracking Evaluation

7.8.1 Strict Algorithm

Tracking informations for the classifier *hugeTree*

$threshold_{classifier}$	$MOTP$	n_{fp}	n_{miss}	$n_{mismatches}$	$MOTA$	$MOTA$ no seg
20	70,39	1	1298	7	1,135503	1,58
10	71,34	7	1007	26	21,27176	29,6
0	69,91	33	611	11	50,41635	70,2
-5	69,89	48	522	8	56,24527	78,3
-10	69,54	90	454	11	57,98637	80,7
-15	69,49	181	422	10	53,59576	74,6
-20	69,4	429	408	8	36,03331	50,2

Tracking informations for the classifier *tree2*

$threshold_{classifier}$	$MOTP$	n_{fp}	n_{miss}	$n_{mismatches}$	$MOTA$	$MOTA$ no seg
20	64,53	3	1250	19	3,709311	5,16
10	69,04	14	831	17	34,7464	48,4
0	69,88	33	545	13	55,26117	76,9
-5	69,64	48	492	10	58,36488	81,2
-10	69,45	97	454	14	57,22937	79,7
-15	69,41	178	426	8	53,67146	74,7
-20	69,17	342	408	9	42,54353	59,2

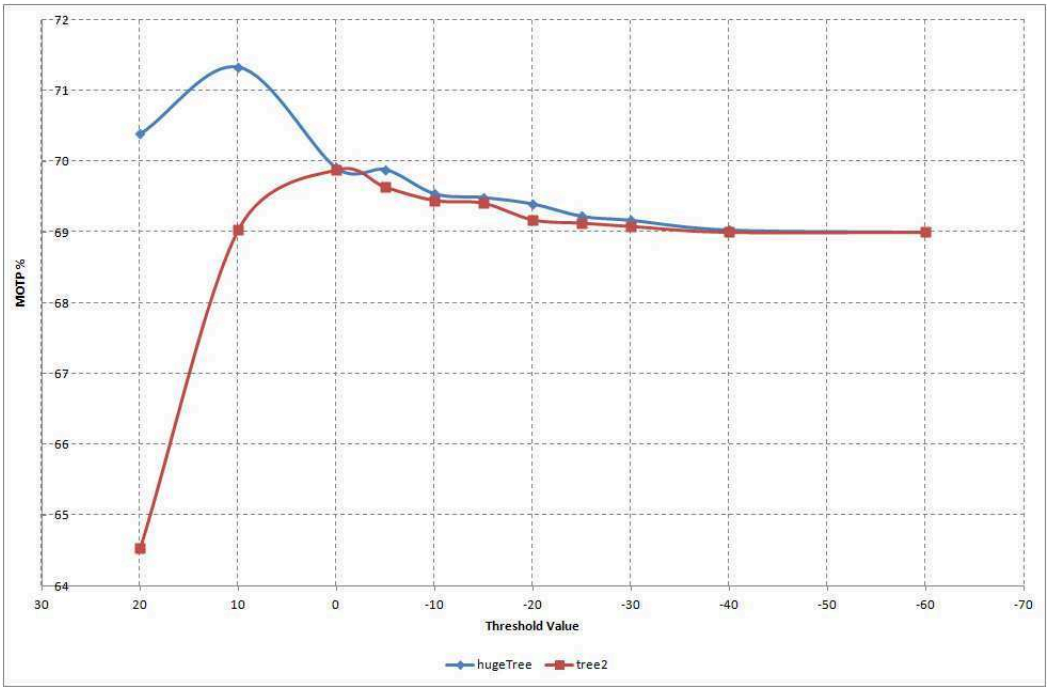

Figure 7.33. *MOTP* curves for the classifiers *hugeTree* and *tree*2.

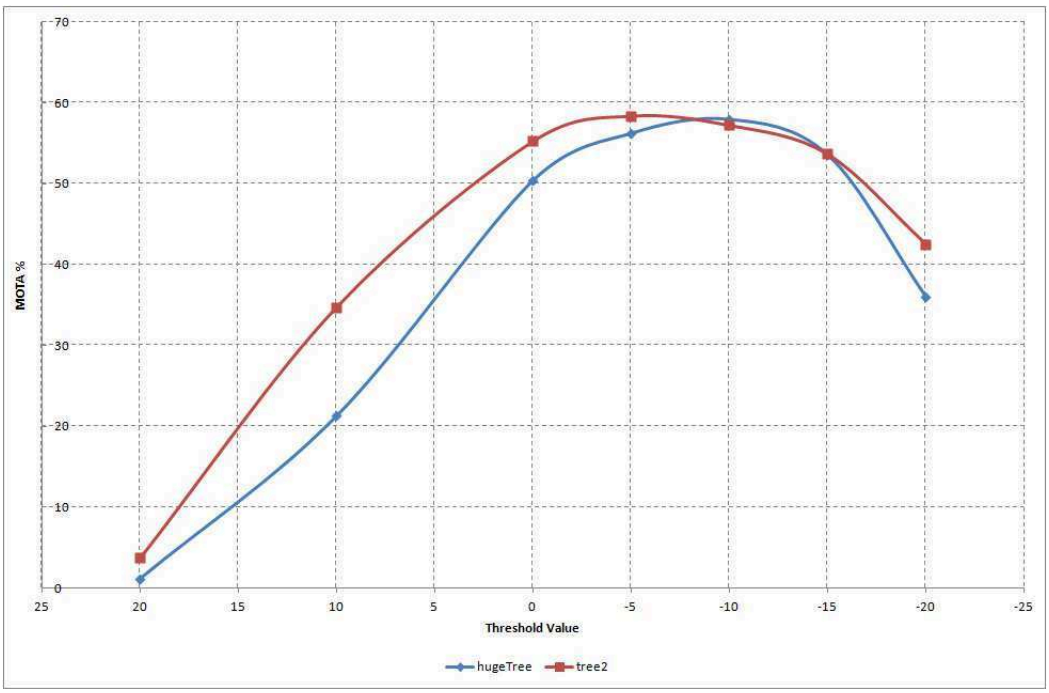

Figure 7.34. *MOTA* curves for the classifiers *hugeTree* and *tree*2.

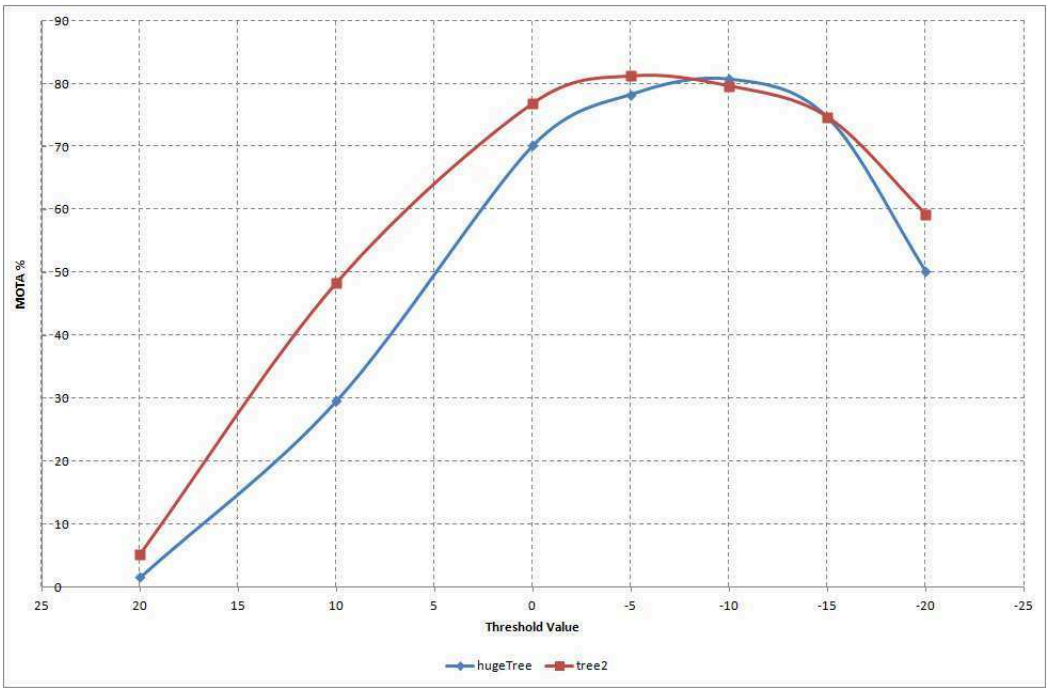

Figure 7.35. $MOTA$ curves for the classifiers $hugeTree$ and $tree2$ without segmentation errors.

7.8.2 Non Strict Algorithm

Tracking informations for the classifier *hugeTree*

$threshold_{classifier}$	$MOTP$	n_{fp}	n_{miss}	$n_{mismatches}$	$MOTA$	$MOTA$ no seg
20	69,5	2	1296	11	0,908403	1,03
10	69,93	16	956	24	24,60257	35,1
0	68,59	86	502	19	54,04996	61,5
-5	68,33	143	385	22	58,36488	66,4
-10	67,91	244	293	34	56,77517	64,6
-15	67,77	419	247	39	46,63134	53,1
-20	67,51	756	219	61	21,57456	24,5

Tracking informations for the classifier *tree2*

$threshold_{classifier}$	$MOTP$	n_{fp}	n_{miss}	$n_{mismatches}$	$MOTA$	$MOTA$ no seg
20	69,5	3	1234	19	4,920515	5,6
10	69,93	34	764	19	38,15291	43,4
0	68,59	97	406	15	60,78728	69,2
-5	68,33	154	327	21	61,99849	70,5
-10	67,91	233	279	25	59,34898	67,5
-15	67,77	373	247	37	50,26495	57,2
-20	67,51	659	225	60	28,53899	32,5

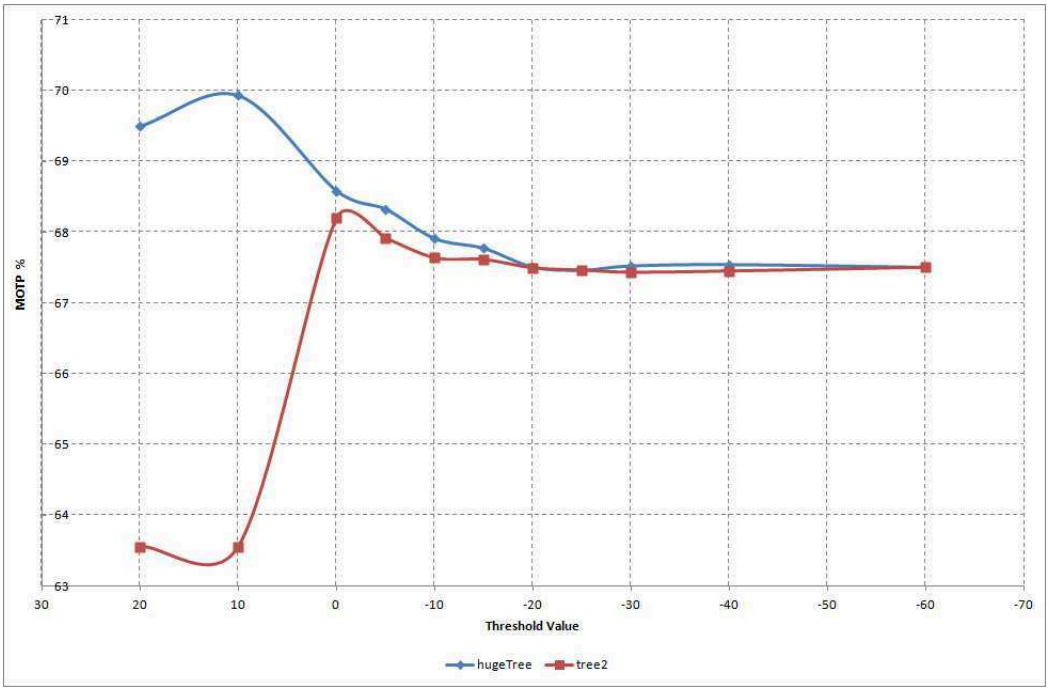

Figure 7.36. *MOTP* curves for the classifiers *hugeTree* and *tree*2.

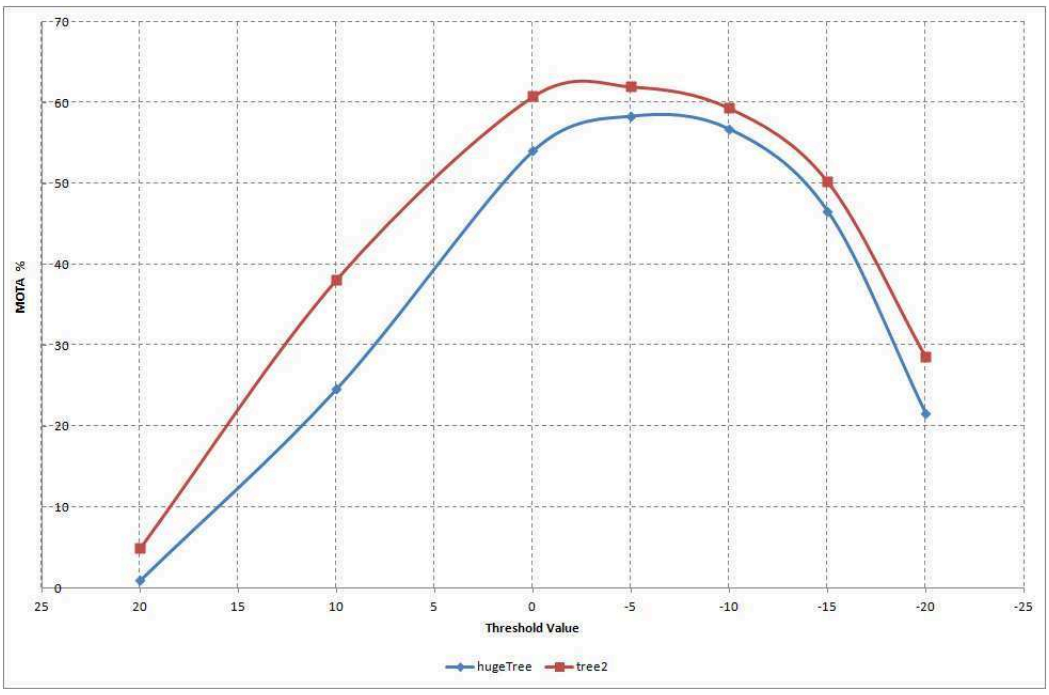

Figure 7.37. *MOTA* curves for the classifiers *hugeTree* and *tree*2.

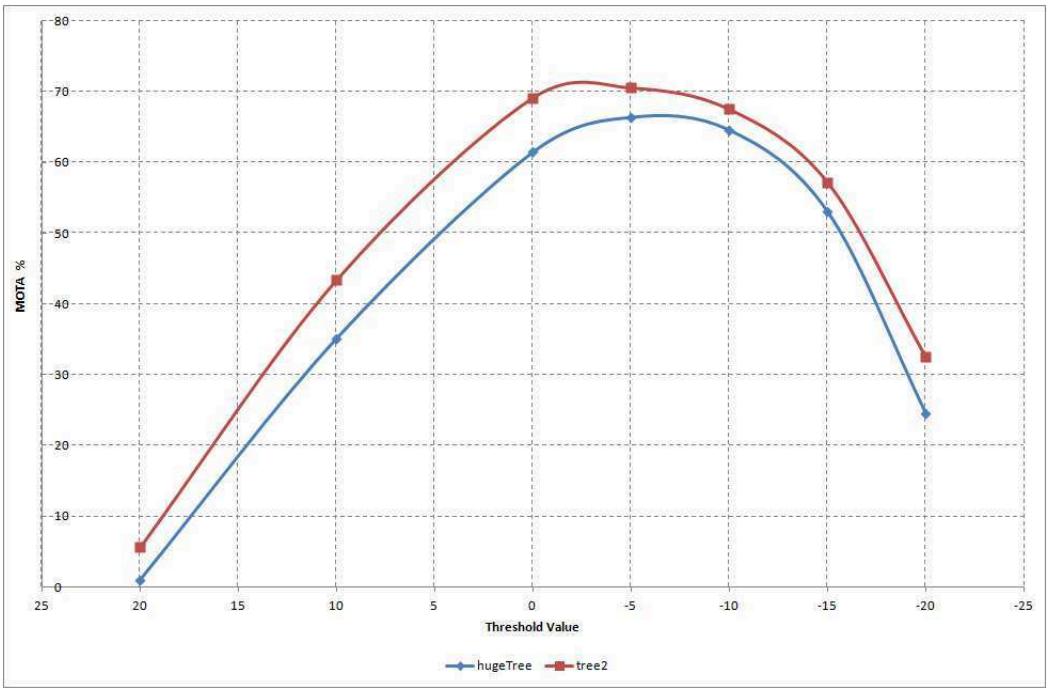

Figure 7.38. *MOTA* curves for the classifiers *hugeTree* and *tree*2 without segmentation errors.

7.8.3 Comparison

The performances of the tracking module have been evaluated just for the two classifiers that showed the best performances, that are *tree*2 and *hugeTree*. The classifier *hugeTree* is generally more precise with respect to the *tree*2 one (figures 7.33, 7.36), even if for values of $threshold_{classifier}$ greater than 0, the two classifiers appear to be almost identical.
The best precision is reached by the classifier *hugeTree* in the strict version of the algorithm (71.3% for the threshold value of 10, whereas the *tree*2 best performance is of 69.9% for the threshold value of 0). We can also see from figure 7.39, that the non strict version of the algorithm is all the time worse than the strict one for both the classifiers. The best performances achieved are however near to the ones of the strict version (roughly 2% smaller).
On the contrary, the classifier *tree*2 is the most accurate, both for the complete system (figures 7.34, 7.37) and when taking into account the segmentation errors (figures 7.35, 7.38).
The best accuracy is reached by the classifier *tree*2 when taking into account segmentation errors and using the strict version of the algorithm. The performance is of 81.2% with $threshold_{classifier} = -5$. If we consider the complete system instead, the best result is again reached by the classifier *tree*2 but using the non strict version of the algorithm. Figures 7.40 and 7.41 show more clearly that this time the non strict version actually raises the accuracy of the complete system as expected from the considerations made in section 7.7 and the detection results achieved.
However the raising is not significant. In fact we achieved performances that in the best case have been raised of just 3%. On the contrary when we take into account the segmentation errors, we see a more significant lowering in the performances (up to 13%). This is due to a phenomenon that has been introduced by the new candidates allowed by the shape filtering. The segmentation module in fact can sometimes cut a person in halves (external and internal or vertical halves for side detections). This doubling of a person was filtered out by the shaping procedure, but now they are allowed to be computed as candidates. So it happens not so rarely that a person can be detected twice by the classifier. This two detections represent the same person and we can safely affirm that they are both correct. For this reason the probability of achieving a mismatch becomes a lot higher than in the strict version of the algorithm. In fact it can be the case that the additional detection has a centroid nearer than the previous detected one. In this case we will obtain a mismatch. So it is explained also the lowering of the performances.

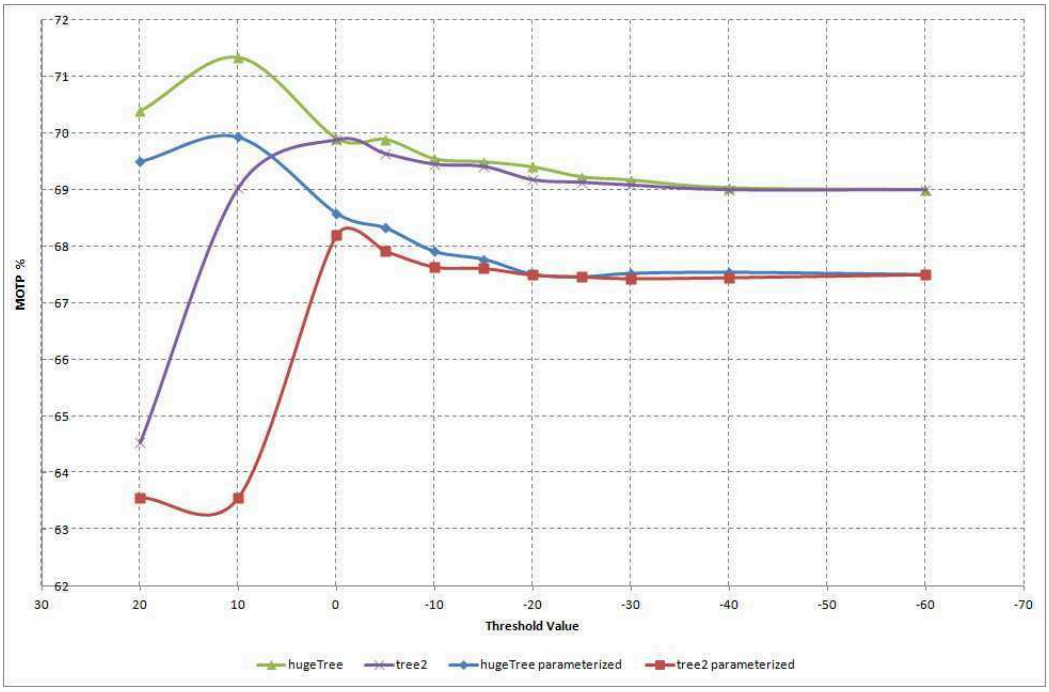

Figure 7.39. Comparison of the $MOTP$ curves for the two versions of the algorithm.

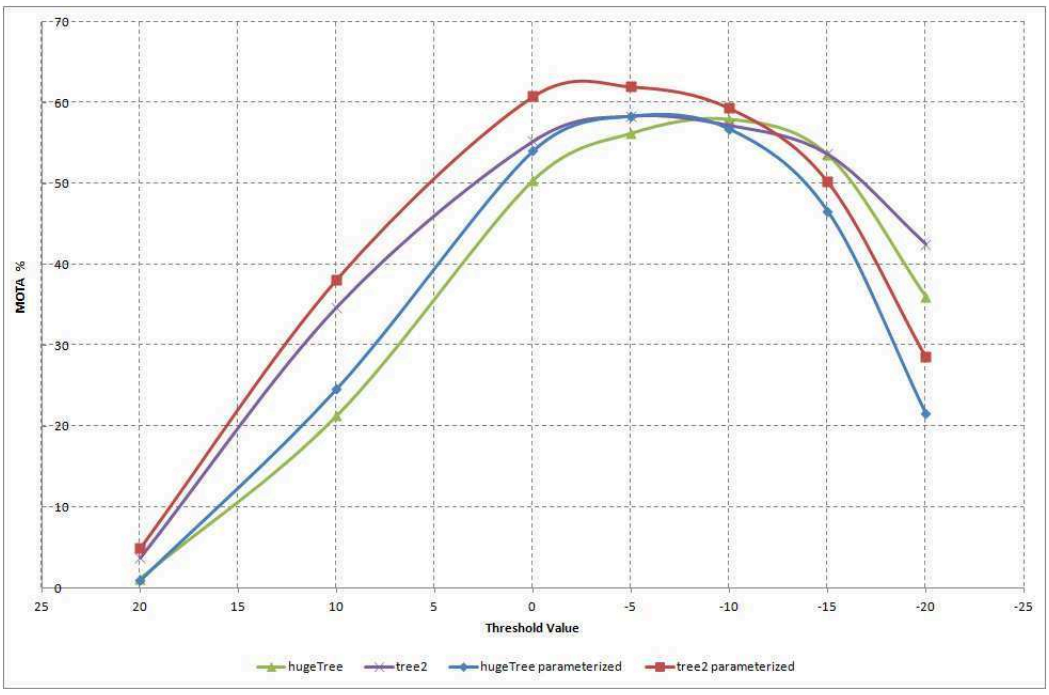

Figure 7.40. Comparison of the $MOTA$ curves for the two versions of the algorithm.

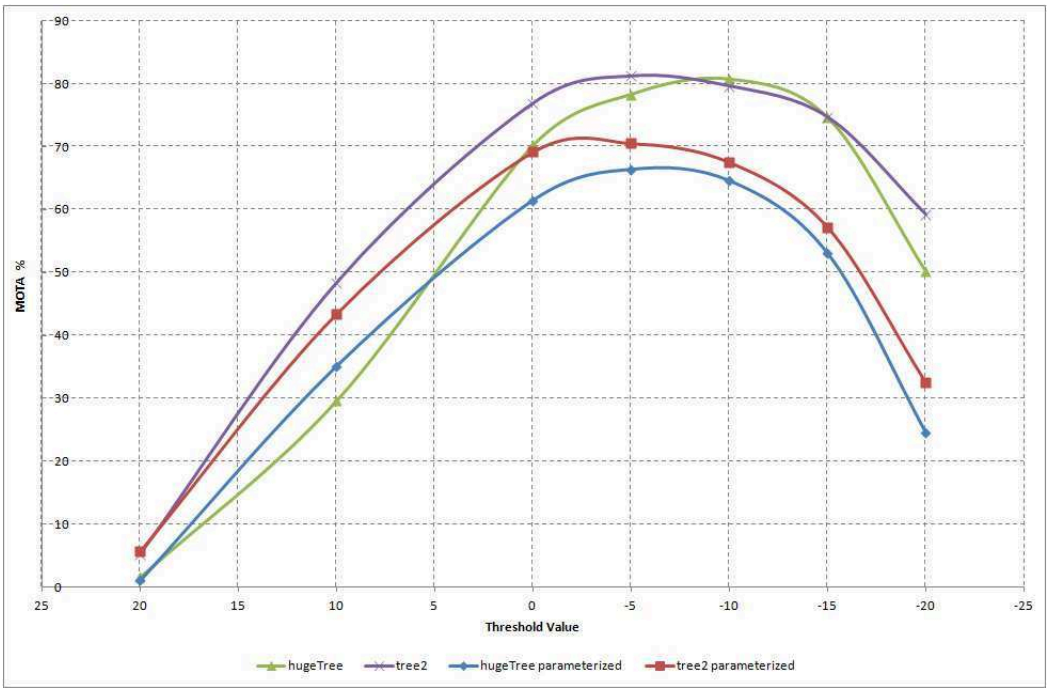

Figure 7.41. Comparison of the $MOTA$ curves for the two versions of the algorithm without segmentation errors.

7.9 Moving Camera

A second test has been done also in a scenario where the Kinect was moving. The tests showed results almost identical to the ones we have already seen for the static sensor, but with a little lowering of the performances. In order to be more complete we report just a little comparison for the TAR, ROC and DET curves of the best classifier ($tree2$) in both situations.

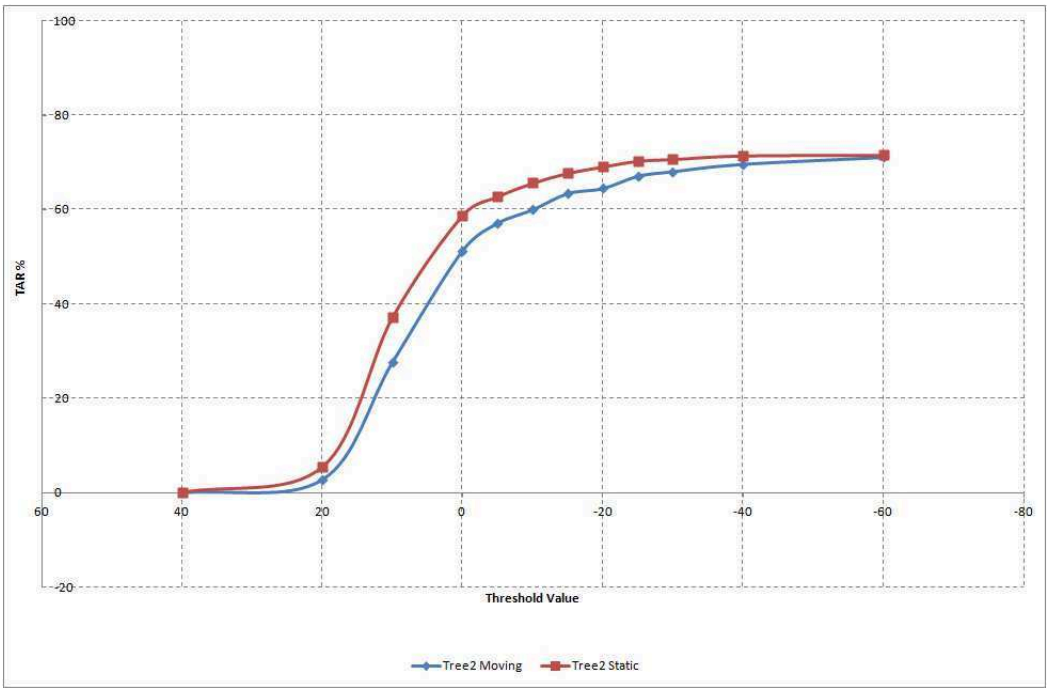

Figure 7.42. Comparison of the TAR curves for the classifier $tree2$ on the static and mobile scenario.

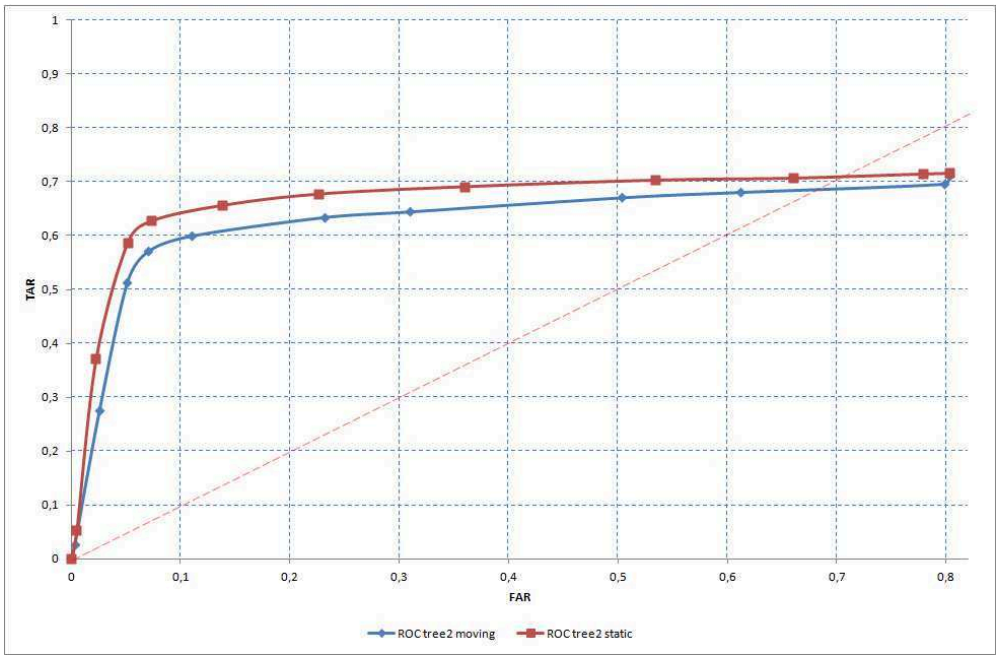

Figure 7.43. Comparison of the *ROC* curves for the classifier *tree*2 on the static and mobile scenario.

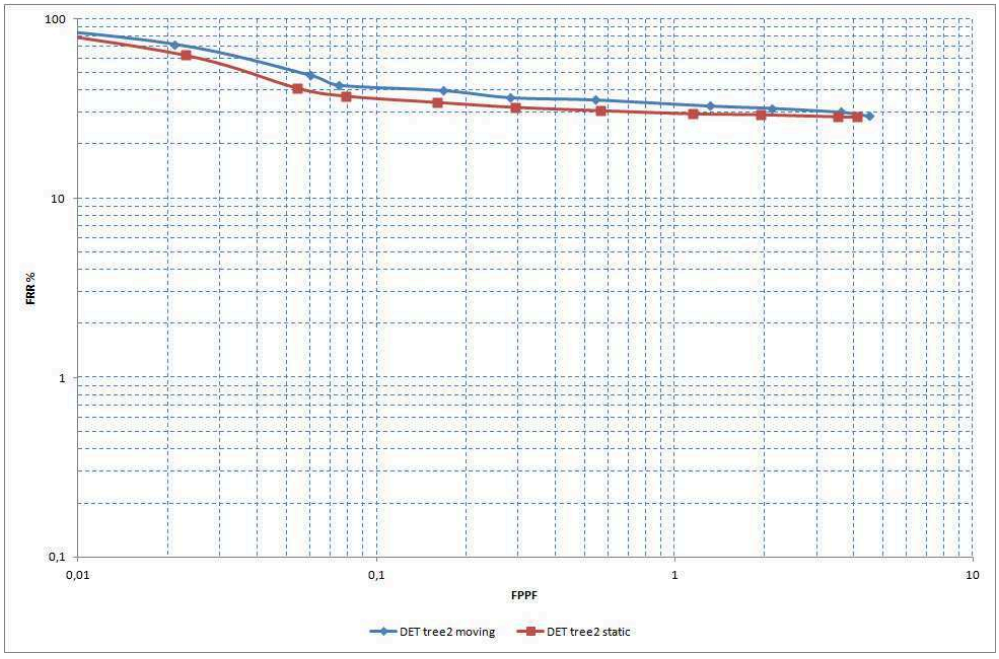

Figure 7.44. Comparison of the *DET* curves for the classifier *tree*2 on the static and mobile scenario.

7.10 Timing Constraints

One of the requirement for the developed system is to run in real-time. The performances achieved are instead near real-time. We divided the measuring into seven parts, that are:

1. segmentation: it represents the time needed for segmenting the depth map, retrieved by the Kinect, in depth layers;
2. candidate initialization: it represents the time needed for computing the pixel width and height of each candidate;
3. candidate filtering: it represents the time needed for filtering out a candidate, on the base of the real width and height informations;
4. candidate preprocessing: it represents the time needed for all the preprocessing procedures for each candidate required by the feature computation module;
5. classifier: it represents the time needed for creating the vector of RDSFs and for retrieving the answer of the classifier for each candidate;
6. tracking update: it represents the time needed for creating or updating the tracking features of each candidate;
7. showing: it represents the time needed for showing the results on the screen.

The tests demonstrated that four of this parts always run in less than 1 ms, that are candidate filtering and processing, tracking update and showing. More interesting are the remaining three parts.
The candidate initialization takes continuously roughly 10 ms for each candidate.
The processing required by the classifier instead varies mostly between 10 and 30 ms with an average of 16 ms for each candidate. Sometimes it happened that this part ran incredibly fast (less than 1 ms).
The heaviest part of the computation is represented by the segmentation. In fact the best performances achieved by this part has been of 200 ms whereas the worst one recorded a timing of 320 ms. If we sum up all the contributions of the other parts we obtain, in the best and worst cases, timings of 110 ms and 340 ms. It doesn't have to appear strange that the computation time went up to these values since the timing of the other parts (apart from the showing one) is given for each candidate. So it has to be multiplied for all the candidates given by the system to the various parts.
So the complete system runs at 2.2 fps on average. The best performance recorded the value of 3.1 fps whereas the worst one has been attested to 1.8 fps.

Chapter 8

Conclusions

In this thesis we have investigated on how we can make use of the new Microsoft's gaming sensor, the Kinect, to address the issues of real-time people detection and tracking, since the sensor has been built in order to detect people and track their movements.
We developed a system that is able of detecting and tracking people in near real-time both on fixed environments and mobile platforms. The system has been developed for an indoor environment and it is successfully able to detect people standing upright that are at least 1.5 meters from the sensor which can be affected by a not too severe occlusion. Those people can be walking or standing still. The developed system presents three main characteristics:

1. good detection results of the classifier;
2. good tracking results;
3. segmentation problems.

8.1 Classifier Performances

The classifier by itself showed in the best case (section 7.3) good performances. It is capable of detecting the 82% of people in the scene with a false positive detection percentage of 5%. The TAR percentage then keeps growing but from now on the FAR one grows faster. It is still acceptable until the combination $TAR = 91\%,\ FAR = 13\%$ but after this we can see that an increase of 3% of TAR correspond to an increase of 10% in the FAR percentage.
This results don't completely describe the accuracy of the classifier. In fact since the RDSF are based on the human detection shape (measured by the Bhattacharyya distance between relative histograms as explained in section 4.3), it is sensitive to the person pose. In fact the classifier has been tested mainly on frontal and backward pose of people with some side and occluded images. This process has been done in order to give a classifier consistent data on which relying on.

This fact has led to two things: a good detection of people facing or showing their back to the sensor, and a weaker detection of people showing instead their sides. These people can be still detected often also because of the head which is a very strong characteristic in the human shape. In an opposite way the occlusion of the head will often lead to not recognize a person as a human being.
So we considered the different types of pose on this specific test case: classifier $tree2$, $threshold_{classifier} = -10$. In this case we have $FRR = 13\%$ and of this the 13% is due to frontal or backward poses whereas the 51% for side ones. So an improvement that can be done is to train another classifier for just sides pose of people and use it as a secondary test for candidates that are not recognized as human being by the first classifier.
The remaining 36% instead is due to severe occlusion or to half-length detections. In fact one of the limitation is that the classifier has been trained for full body detection and so there has to be a minimum distance between the sensor and each person of 1.5 meters. However on this test people were moving also closer to the sensor in order to test this limitation.

8.2 Tracking Performances

The system showed good tracking results. By using the $MOTP$ and $MOTA$ metrics, we achieved a detection precision of 71.3 % with the classifier $hugeTree$, and an accuracy of 62 % and 81.2 % respectively for the complete system and taking into account for segmentation errors. Both best accuracies are achieved with the classifier $tree2$.

8.3 Segmentation Problems

As already explained in section 6.6 the segmentation module can make some errors. These errors will result in not giving to the classifier good candidates or in not giving candidates at all. For this reason the performances of the complete system drop down sometimes more than 25% with respect to the ones of the classifier itself. In fact we can see for the classifier $tree2$ with $threshold_{classifier} = -10$ the performances of the complete system go down from $TAR = 91\%$ to $TAR = 66\%$. The false positive measure instead, as already explained, isn't affected by the segmentation module, in fact we have in both cases $FAR = 13\%$.
Also the tracking system is affected by these problems. In fact the $MOTA$ evaluation depends on the number of miss, false positive and mismatches, whereas the $MOTP$ measurement is independent from these values. So the performances of the complete system drop down again sometimes more than 20% with respect to the ones of the classifier itself. We can see for the classifier $tree2$ with $threshold_{classifier} = -5$ that the performances of the complete system go down from $MOTA = 81.2\%$ to

$MOTA = 58.4\%$.
An advantage of the method developed is instead that there is a limitation on the number of candidates produced. This will results in a limitation as well of possible false positives. In fact even by lowering noteworthily the threshold value for the classifier, the FAR percentage doesn't go higher than 85%.
However in order to get rid of some segmentation problems, we tried to relax a constrain in the shape filtering of the candidates. This has led to three results:

1. raising of the performances of the complete system for detection results: in fact for the classifier $tree2$ with $threshold_{classifier} = -10$, the performances of the complete system that went down from $TAR = 91\%$ to $TAR = 66\%$ have been raised till $TAR = 79\%$;

2. no significant improvement in the tracking results: the $MOTA$ evaluation for the classifier $tree2$ with $threshold_{classifier} = -5$ that went down from $MOTA = 81.2\%$ to $MOTA = 58.4\%$ has been raised just till $MOTA = 62\%$;

3. worsening of the results when taking into account segmentation errors: the DET and ROC curves for the detection show that the performances for the complete system become worse and this is due to the fact that by being less strict on the height threshold value, we actually get rid of some segmentation problems but we give to the classifier a lot more candidates which presents a higher probability of not being a person. Indeed, the probability for a candidate with height of 60 cm being a segmentation problem is lower than the one of being an object. Moreover the tracking algorithm suffers a lot the introduction of these new candidates. In fact it happens not so rarely that a person can be detected twice. This leads to a lot higher probability of achieving a mismatch than it happens in the strict version of the algorithm. So the lowering of the performances.

8.4 Real-Time Measurement

The complete system runs at $2.2fps$ on average. The best performance recorded the value of $3.1fps$ whereas the worst one has been attested to $1.8fps$.
The module that produces the greatest computational load is again the segmentation one. In fact, by looking at the partial timing of the algorithm, this module showed the best run at $200ms$ and the worst one at $320ms$. In order to make a comparison all the other processing has be done in the best case in $110ms$ and in the worst one in $340ms$.

8.5 Final Notes and Future Work

The entire program has been developed in two ways:

1. a standalone executable for fixed installations of the sensor;

2. a ROS node for installations of the sensor on mobile platform.

Still the program need to convert the data from the navigation node in order to update correctly the tracking system. This update is really simple in fact it is possible to compute where the previous centroid of each person should be, by mean of a geometrical computation almost identical to the one explained in section 5.2. Then the computation will be the same of the one already explained again in section 5.2 by using the computed centroids in place of the previous ones.
In addition, the greatest improvement has to be done on the segmentation module for all the reasons already explained. Moreover an improvement of this module will lead also to a lowering of the timing of all the remaining processing by giving less and more accurate candidates.

Bibliography

[1] N. Ogale, "A survey of techniques for human detection from video," Master's thesis, University of Maryland, 2006. Unpublished.

[2] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland, "Pfinder: real-time tracking of the human body," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 780–785, 1997.

[3] C. Beleznai, B. Fruhstuck, and H. Bischof, "Human detection in groups using a fast mean shift procedure," *International Conference on Image Processing*, vol. 1, pp. 349–352, 2004.

[4] D. Toth and T. Aach, "Detection and recognition of moving objects using statistical motion detection and fourier descriptors," *International Conference on Image Analysis and Processing*, pp. 430–435, 2003.

[5] D. Lee, P. Zhan, A. Thomas, and R. Schoenberger, "Shape-based human intrusion detection," *SPIE International Symposium on Defense and Security, Visual Information Processing XIII*, pp. 81–91, 2004.

[6] F. Xu and K. Fujimura, "Human detection using depth and gray images," *EEE Conference on Advanced Video and Signal Based Surveillance*, pp. 115–121, 2003.

[7] J. Han and B. Bhanu, "Detecting moving humans using color and infrared video," *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, vol. 30, pp. 228–233, 2003.

[8] L. Jiang, F. Tian, L. Shen, S. Wu, S. Yao, Z. Lu, and L. Xu, "Perceptual-based fusion of ir and visual images for human detection," *International Symposium on Intelligent Multimedia, Video and Speech Processing*, pp. 514–517, 2004.

[9] M. Yang, D. Kriegman, and N. Ahuja, "Detecting faces in images: A survey," *IEEE Pattern Analisis and Machine Intelligence*, vol. 24, pp. 34–58, 2002.

[10] D. Geronimo, A. Lopez, and A. Sappa, "Computer vision approaches to pedestrian detection: Visible spectrum survey," *LNCS, Pattern Recognition and Image Analysis*, pp. 547–554, 2007.

[11] D. Gavrila and J. Giebel, "Shape-based pedestrian detection and tracking," *IEEE Intelligent Vehicle Symposium*, pp. 8–14, 2002.

[12] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1063–6919, 2005.

[13] S. Ikemura and H. Fujiyoshi, "Real-time human detection using relational depth similarity features," *Proceedings of the 10th Asian conference on Computer vision*, vol. 4, pp. 1–14, 2010.

[14] P. Viola, M. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," *IEEE International Conference on Computer Vision*, pp. 734–741, 2003.

[15] K. Arras, O. Mozos, and W. Burgard, "Using boosted features for the detection of people in 2d range data," *IEEE International Conference on Robotics and Automation, Roma, Italy*, vol. 2, pp. 3402–3407, 2007.

[16] A. Fod, A. Howard, and M. Mataric, "Laser-based people tracking," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[17] M. Kleinhagenbrock, S. Lang, J. Fritsch, F. Lomker, G. Fink, and G. Sagerer, "Person tracking with a mobile robot based on multi-modalanchoring," *IEEE International Workshop on Robot and Human Interactive Communication (RO-MAN), Berlin, Germany*, 2002.

[18] M. Scheutz, J. McRaven, and G. Cserey, "Fast, reliable, adaptive, bimodal people tracking for indoor environments," *IEEE/RSJ Int. Conference on Intelligent Robots and Systems, Sendai, Japan*, 2004.

[19] D. Schulz, W. Burgard, D. Fox, and A. Cremers, "People tracking with a mobile robot using sample-based joint probabilistic data association filters," *International Journal of Robotics Research (IJRR)*, vol. 22, pp. 99–116, 2003.

[20] M. Lindstrom and J. Eklundh, "Detecting and tracking moving objects from a mobile platform using a laser range scanner," *IROS*, pp. 1364–1369, 2001.

[21] H. Asoh, Y. Motomura, F. Asano, I. Hara, S. Hayamizu, and K. Itou, "Jijo-2: An office robot that communicates and learns," *IEEE Intelligent Systems*, vol. 16, pp. 46–55, 2001.

[22] H. Sidenbladh, D. Kragic, and H. Christensen, "A person following behaviour for a mobile robot," *Proceeding of the IEEE International Conference on Robotics and Automation*, pp. 670–675, 1999.

[23] S. Waldherr, R. Romero, and S. Thrun, "A gesture based interface for human-robot interaction. autonomous robots," 2000.

[24] C. Schlegel, J. Illmann, H. Jaberg, M. Schuster, and R. Worz, "Vision based person tracking with a mobile robot," *Proceeding British Machine Vision Conference*, vol. 3, pp. 418–427, 1998.

[25] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceeding of IEEE conference on computer vision and pattern recognition*, vol. 1, pp. 228–235, 2001.

[26] C. Wang and J. Lien, "Adaboost learning for human detection based on histograms of oriented gradients," *Proceedings of the 8th Asian conference on Computer vision*, vol. 1, pp. 885–895, 2007.

[27] S. Coradeschi and A. Saffiotti, "Anchoring symbols to sensor data: preliminary report," *Proceeding of the 17th AAAI Conference*, pp. 129–135, 2000.

[28] S. Coradeschi and A. Saffiotti, "Perceptual anchoring of symbols for action," *Proceeding of the 17th IJCAI Conference*, pp. 407–412, 2001.

[29] J. Sherrah and S. Gong, "Fusion of perceptual cues for robust tracking of head pose and position. in pattern recognition," *Pattern Recognition, special issue on Data and Information Fusion in Image Processing and Computer Vision*, 2000.

[30] J. Vermaak, A. Blake, M. Gangnet, and P. Perez, "Sequential monte carlo fusion of sound and vision for speaker tracking," *Proceeding International Conference on Computer Vision*, vol. 1, pp. 741–746, 2001.

[31] T. Darrell, G. Gordon, M. Harville, and J. Woodfill, "Integrated person tracking using stereo, color, and pattern detection," *International Journal of Computer Vision*, vol. 37, pp. 175–185, 2000.

[32] A. Gern, U. Franke, and P. Levi, "Robust vehicle tracking fusing radar and vision," *International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pp. 323–328, 2001.

[33] J. Fritsch, M. Kleinehagenbrock, S. Lang, G. Fink, and G. Sagerer, "Audiovisual person tracking with a mobile robot," *Proceeding International Conference on Intelligent Autonomous Systems*, pp. 898–906, 2004.

[34] R. Cutler and L. Davis, "Robust real-time periodic motion detection, analysis, and applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 781–796, 2000.

[35] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by probability distributions," *Bull Calcutta Math Soc*, vol. 35, pp. 99–109, 1943.

[36] R. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, pp. 297–336, 1999.

[37] E. Schapire, "Using output codes to boost multiclass learning problems," *Machine Learning: Proceedings of the Fourteenth International Conference*, pp. 313–321, 1997.

[38] G. Buchsbaum, "A spatial processor model for object color perception," *Journal of the Franklin Institute*, 1980.

[39] B. Funt, K. Barnard, and L. Martin, "Is machine colour constancy good enough?," *Lecture Notes in Computer Science*, p. 1406, 1998.

[40] H. Eng, J. Wang, A. Kam, and W. Yau, "A bayesian framework for robust human detection and occlusion handling using a human shape model," *International Conference on Pattern Recognition*, 2004.