

UNIVERSITAT POLITÈCNICA DE CATALUNYA  
PURDUE UNIVERSITY

FINAL THESIS

---

# Automatic crowdfow estimation enhanced by crowdsourcing

---

*Author:*  
Javier Ribera Prat

*Supervisors:*  
Prof. Edward J. Delp  
Prof. Luis Torres

*A thesis submitted in fulfilment of the requirements  
for the degree of Enginyeria de Telecomunicació*

*in*

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

December 2014

## *Abstract*

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona  
Enginyeria de Telecomunicació

### **Automatic crowdflow estimation enhanced by crowdsourcing**

by Javier Ribera Prat

Video surveillance systems are evolving from simple closed-circuit television (CCTV) towards intelligent systems capable of understanding the recorded scenes. This trend is accompanied by the widespread increase in the amount of cameras, which makes the continuous monitoring of video feeds a practically impossible task. In this scenario, video surveillance systems make intensive use of video analytics and image processing in order to allow their scalability and boost their effectiveness.

One of such video analytics performed in video surveillance systems is crowd analysis. Crowd analysis plays a fundamental role in security applications. For instance, keeping a rough estimate of the amount of people present in a given area or inside a building is critical to prevent jams in an emergency or when planning the distribution of entry and exit nodes.

In this thesis, we focus on crowd flow estimation. Crowd flow is defined as the number of people that have crossed a specific region over time. Hence, the goal of the method is to estimate the crowd flow as accurately as possible in real time. Many automatic methods have been proposed in the literature to estimate the crowd flow.

However, video analytics techniques often face a wide range of difficulties such as occlusions, shadows, environmental conditions changes or distortions in the video. Developed methods struggle to maintain a high accuracy in such situations. Crowdsourcing has been shown as an effective solution to solve to problems that involve complex cognitive tasks. By incorporating human assistantship, the performance of automatic methods can be enhanced in adverse situations.

In this thesis, an automatic crowd flow estimation method, previously developed in the Video and Image Processing Laboratory at Purdue University, is implemented and crowdsourcing is used to enhance its performance. Also, a web platform is developed to control the whole system remotely by the operator of the system, and to allow the crowdsourcing members to perform their tasks.

## *Resum*

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona  
Enginyeria de Telecomunicació

### **Automatic crowdflow estimation enhanced by crowdsourcing**

by Javier Ribera Prat

Els sistemes de videovigilància estan evolucionant des de simples circuits tancats de televisió (CCTV) cap a sistemes intel·ligents capaços d'entendre les escenes enregistrades. A aquesta tendència li acompanya l'extès increment en la quantitat de càmeres, fet que fa que monitoritzar continuament tots els fluxes de video sigui una tasca pràcticament impossible. En aquest escenari, els sistemes de videovigilància fan un ús intensiu d'analítiques de video i processament d'imatge per tal de permetre la seva escalabilitat i impulsar la seva efectivitat.

Una d'aquestes analítiques de video que es realitzen en els sistemes de videovigilància és l'anomenat «crowd analysis» o anàlisi de multituds. El «crowd analysis» duu a terme un rol fonamental en aplicacions de seguretat. Per exemple, mantenir una estimació aproximada de la quantitat de persones presents en una àrea o dintre d'un edifici és crític per prevenir embusos en una emergència o per planejar la distribució de nodes d'entrada o sortida.

En aquesta tesi, ens focalitzem en estimació del «crowd flow» o fluxe de multituds. «Crowd flow» es defineix com el nombre de persones que han creuat una regió específica al llarg del temps. Així, l'objectiu del mètode és estimar el «crowd flow» tan precisament com sigui possible en temps real. En la literatura s'han proposat molts mètodes automàtics per estimar el «crowd flow».

Tot i així, les tècniques d'analítiques de video sovint s'enfronten a una àmplia gamma de dificultats com ara oclusions, ombres, canvis en les condicions ambientals o distorsions en el video. Els mètodes desenvolupats barallen per mantenir una alta precisió en aquestes situacions. El «crowdsourcing» s'ha demostrat com una sol·lució efectiva als problemes que involucren tasques cognitives complexes. Incorporant assistència humana, es pot millorar el rendiment dels mètodes automàtics en situacions adverses.

En aquesta tesi, s'implementa un mètode automàtic d'estimació del «crowd flow», prèviament desenvolupat al Video and Image Processing Laboratory a la universitat de Purdue,

i es fa servir «crowdsourcing» per millorar el seu rendiment. A més, es desenvolupa una plataforma web per controlar tot el sistema remotament per l'operador, i per permetre als membres del «crowdsourcing» portar a terme les seves tasques.

## *Resumen*

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona  
Enginyeria de Telecomunicació

### **Automatic crowdflow estimation enhanced by crowdsourcing**

by Javier Ribera Prat

Los sistemas de videovigilancia están evolucionando desde simples circuitos cerrados de televisión (CCTV) hacia sistemas inteligentes capaces de entender las escenas registradas. A esta tendencia le acompaña el extendido incremento en la cantidad de cámaras, hecho que hace que monitorizar continuamente todos los flujos de vídeo sea una tarea prácticamente imposible. En este escenario, los sistemas de videovigilancia hacen un uso intensivo de analíticas de video y procesamiento de imagen al fin de permitir su escalabilidad e impulsar su efectividad.

Una de estas analíticas de vídeo que se realizan en los sistemas de videovigilancia es el llamado «crowd analysis» o análisis de multitudes. El «crowd analysis» lleva a cabo un rol fundamental en aplicaciones de seguridad. Por ejemplo, mantener una estimación aproximada de la cantidad de personas presentes en una área o dentro de un edificio es crítico para prevenir atascos en una emergencia o para planear la distribución de nodos de entrada o salida.

En esta tesis, nos focalizamos en estimación del «crowd flow» o flujo de multitudes. «Crowd flow» se define como el número de personas que han cruzado una región específica a lo largo del tiempo. Así, el objetivo del método es estimar el «crowd flow» tan precisamente como sea posible en tiempo real. En la literatura se han propuesto muchos métodos automáticos para estimar el «crowd flow».

Aun así, las técnicas de analíticas de vídeo a menudo se enfrentan con una amplia gama de dificultades tales como oclusiones, sombras, cambios en las condiciones ambientales o distorsiones en el vídeo. Los métodos desarrollados pelean por mantener una alta precisión en estas situaciones. El «crowdsourcing» se ha demostrado como una solución efectiva a los problemas que involucran tareas cognitivas complejas. Incorporando asistencia humana, se puede mejorar el rendimiento de los métodos automáticos en situaciones adversas.

En esta tesis, se implementa un método automático de estimación del «crowd flow», previamente desarrollado en el Video and Image Processing Laboratory en la universidad de Purdue, y se usa «crowdsourcing» para mejorar su rendimiento. Además, se desarrolla una plataforma web para controlar todo el sistema remotamente por parte del operador, y permitir a los miembros del «crowdsourcing» llevar a cabo sus tareas.

## *Acknowledgements*

This work would not have been possible without the support of the advisors Professor Luis Torres, from Universitat Politècnica de Catalunya, and Professor Edward J. Delp, from Purdue University. Special gratitude for offering me the opportunity to carry out this project in the Video and Image Processing laboratory at Purdue University.

I would also like to thank Khalid Tahboub for his indispensable help throughout the project.

Lastly, I want to personally express my thankfulness to all the VIPER laboratory members for maintaining such a healthy environment for research and for their warm welcome to West Lafayette.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Resum</b>	<b>ii</b>
<b>Resumen</b>	<b>iv</b>
<b>Acknowledgements</b>	<b>vi</b>
<b>Contents</b>	<b>vii</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>x</b>
<b>Abbreviations</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Literature Review</b>	<b>4</b>
2.1 Crowd flow estimation . . . . .	4
2.2 Crowdsourcing . . . . .	5
<b>3 Crowd flow estimation</b>	<b>7</b>
3.1 Key ideas and overall scheme . . . . .	7
3.2 Foreground pixel count . . . . .	9
3.2.1 Weighting function . . . . .	11
3.2.2 Background subtraction . . . . .	13
3.3 Crowdedness estimation . . . . .	16
3.4 Training data . . . . .	20
<b>4 Crowdsourcing</b>	<b>22</b>
4.1 Uncertainty of the classifier . . . . .	22
4.2 Use of crowdsourcing . . . . .	23
4.3 Crowdsourcing task . . . . .	24



---

<b>5</b>	<b>Web platform</b>	<b>26</b>
5.1	System overview . . . . .	26
5.1.1	Backend . . . . .	27
5.1.2	Front-end . . . . .	28
5.1.3	Real-time streams . . . . .	29
5.2	Graphical User Interface . . . . .	29
5.2.1	Monitor of processes . . . . .	30
5.2.2	Invoker of processes . . . . .	32
5.2.3	Crowdsourcing tasks . . . . .	33
5.2.4	Training . . . . .	33
5.2.5	Command History . . . . .	39
<b>6</b>	<b>Experimental results</b>	<b>40</b>
6.1	Testing conditions . . . . .	40
6.1.1	Dataset . . . . .	40
6.1.2	Parameters . . . . .	41
6.2	Experiments . . . . .	42
6.2.1	First experiment: no crowdsourcing . . . . .	42
6.2.2	Second experiment: crowdsourcing incorporation . . . . .	43
<b>7</b>	<b>Conclusions</b>	<b>45</b>
	<b>Bibliography</b>	<b>46</b>

# List of Figures

3.1	Region of Interest (ROI) and Tripwire examples . . . . .	8
3.2	Overall scheme of the automatic crowd flow estimation method . . . . .	9
3.3	Scheme of the foreground pixel count . . . . .	9
3.4	Foreground mask of a Tripwire . . . . .	10
3.5	Weighting scheme illustration . . . . .	12
3.6	Mixture of gaussians that model the recent history of a pixel for Back-ground Subtraction . . . . .	14
3.7	Different levels of crowdedness . . . . .	17
3.8	GLCM calculations . . . . .	17
3.9	Texture feature scheme . . . . .	18
3.10	Nearest neighbour texture feature vector . . . . .	19
4.1	Certainty areas around two reference vectors corresponding to four values of $\alpha$ . . . . .	24
4.2	Certainty areas around two reference vectors corresponding to four values of $\alpha$ . . . . .	24
5.1	Overview of the system . . . . .	26
5.2	Web platform: processes monitor . . . . .	31
5.3	Web platform: processes invoker selecting a video file . . . . .	32
5.4	Web platform: processes invoker selecting a real-time video stream . . . . .	32
5.5	Web platform: crowdsourcing tasks . . . . .	33
5.6	Web platform: training . . . . .	34
5.7	Web platform: new training data, step 2 . . . . .	34
5.8	Web platform: new training data, step 3 . . . . .	35
5.9	Web platform: new training data, step 4 . . . . .	36
5.10	Web platform: new training data, step 6 . . . . .	36
5.11	Web platform: new training data, step 7 . . . . .	37
5.12	Web platform: new training data, step 8 . . . . .	37
5.13	Web platform: new training data, BS preview . . . . .	38
5.14	Web platform: command history . . . . .	39
6.1	UCSD pedestrian dataset . . . . .	40
6.2	Distribution of the UCSD dataset video segments . . . . .	41
6.3	Results of first experiment: no crowdsourcing . . . . .	43
6.4	Results of second experiment: crowdsourcing incorporation . . . . .	44

# List of Tables

6.1	Provided ground truth of the UCSD dataset for the testing segments of Figure 6.2. . . . .	41
6.2	Results of second experiment: Average error rate appreciably decreases when crowdsourcing is incorporated. . . . .	43

# Abbreviations

<b>AMD</b>	<b>A</b> synchronous <b>M</b> odule <b>D</b> efinition
<b>BS</b>	<b>B</b> ackground <b>S</b> ubtraction
<b>CCTV</b>	<b>C</b> losed- <b>C</b> ircuit <b>T</b> ele <b>V</b> ision
<b>CSS</b>	<b>C</b> ascading <b>S</b> tyle <b>S</b> heets
<b>DOM</b>	<b>D</b> ocument <b>O</b> bject <b>M</b> odel
<b>EI</b>	<b>E</b> lectronic <b>I</b> maging
<b>GLCM</b>	<b>G</b> ray <b>L</b> evel <b>C</b> o-occurrence <b>M</b> atrix
<b>HIT</b>	<b>H</b> uman <b>I</b> ntelligence <b>T</b> ask
<b>HTML</b>	<b>H</b> yper <b>T</b> ext <b>M</b> arkup <b>L</b> anguage
<b>IEEE</b>	<b>I</b> nstitute of <b>E</b> lectric and <b>E</b> lectronic <b>E</b> ngineers
<b>JS</b>	<b>J</b> ava <b>S</b> cript
<b>MOG</b>	<b>M</b> ixture <b>O</b> f <b>G</b> aussians
<b>MVC</b>	<b>M</b> odel- <b>V</b> iew- <b>C</b> ontroller
<b>NAECON</b>	<b>N</b> ational <b>A</b> erospace and <b>E</b> lectronics <b>C</b> onference
<b>ORM</b>	<b>O</b> bject- <b>R</b> elational <b>M</b> apping
<b>PHP</b>	<b>P</b> HP: <b>H</b> ypertext <b>P</b> reprocessor
<b>SPIE</b>	<b>S</b> ociety of <b>P</b> hotographic and <b>I</b> nstrumentation <b>E</b> ngineers
<b>SVM</b>	<b>S</b> upport <b>V</b> ector <b>M</b> achine
<b>ROI</b>	<b>R</b> egion <b>O</b> f <b>I</b> nterest
<b>UCSD</b>	<b>U</b> niversity of <b>C</b> alifornia, <b>S</b> an <b>D</b> iego

*A mis padres, sin cuyo esfuerzo y cariño no habría podido llegar tan lejos...*

# Chapter 1

## Introduction

Video surveillance systems play an important role in commercial, law enforcement and military applications for security purposes [1]. For instance, private buildings, airports or border control points make use of continuous monitoring for prevention and a posteriori analysis of incidents. However, the current trend in surveillance systems shows an exponential increase of the number of cameras. The vast amount of deployed cameras makes the continuous monitoring of all the video feeds a hardly impossible task to be performed by a human operator [2]. This would lead to an underuse of the video surveillance system, as any preventing capability wouldn't be possible.

Hence, video surveillance systems are moving towards intelligent solutions to provide a scalable solution [1]. In such situation, video analytics are responsible of allowing the system to understand the scenes and react to certain events by warning the security operators in real time. Fight, intrusion or left-luggage detection are examples of incidents that can be detected by means of video processing. Many video analytic techniques have been developed to provide automatic analysis of the video data [3].

One type of video analytics performed in video surveillance systems is crowd flow estimation. Crowd flow is defined as the number of people that has crossed a spatial region over time. By analyzing the crowd flow in the entry of a building, system operators can watch the capacity of the building not to be surpassed. It may also help to avoid jams and provide useful information when designing paths or entry and exit nodes.

In this thesis, we focus on crowd flow estimation. A method previously developed in the VIPER laboratory [4] is taken as starting point. This method had been tested as a proof of concept with successful results. However, it lacked consistency throughout the code to be implemented in a real scenario. In addition, it was coded in MATLAB, leading to performance and licensing problems beyond simple testing scenarios. Therefore, the

whole crowd flow estimation method has been implemented from scratch in Python, using convenient libraries, and has been built in such a way that could be easily incorporated to a much wider video surveillance platform.

On the other hand, a different project was carried out by Khalid Tahboub and Neeraj Gadgil also at the VIPER laboratory [5]. In that project, a web-based video annotation system for crowdsourcing surveillance videos was developed. The result of this project was this tool, which centralizes different video feeds coming from security cameras or video files. The operator allows certain users to manually annotate specific events in the video streams. It also can be used to train the users to perform specific tasks.

Any video analytics technique faces challenges when trying to keep a good accuracy level. These may be occlusions, shadows, distortions in the video due to packet losses or video compression, and sudden or gradual environmental changes, such as light conditions. Crowdsourcing has been proven to be an effective solution for utilizing human intelligence, or the “wisdom of the crowd” to perform several tasks. By crowdsourcing here we mean the use of a group of people (e.g. expert observers, or the “o-crowd”) to observe the surveillance video and “help” the automatic analysis method perform better.

In this thesis, an approach is proposed to enhance the performance of the automatic crowd flow estimation method. To avoid confusion, the group of people observing the surveillance video and helping with the crowd flow analysis will be called “o-crowd”. The crowd of people being recorded by the cameras and being counted will be known as the “crowd.” By using crowdsourcing, the automatic method “asks” the o-crowd whenever it is uncertain in making a particular decision. The answers from the o-crowd are read by the system in order to overcome this specific uncertain decision. Not only that, but also the information provided by the o-crowd is assimilated so that it can reduce future uncertainties and keep asking the o-crowd less often. This way, the system “learns” from the o-crowd. The threshold specifies the maximum uncertainty that the system will tolerate before triggering the o-crowd. It can be tuned so the utilization of the o-crowd can be controlled. Because of this reason, it has been denominated crowdsourcing parameter.

A web interface (front-end) has also been to be able to control the whole system. It allows the operator to monitor all the processes running in the backend and their current status. Each process runs a crowd flow estimation analyzing one video feed, coming from a file or a real-time stream. It also lets the operator to invoke new processes, specifying different parameters, and let the o-crowd perform their tasks, among other things. This interface is meant to be easily incorporated in the video annotation tool.

In the backend, for the crowd flow estimation method, the language used is Python 2.7.5. All non-temporary information is stored in PostgreSQL through the Object-Relational Mapper (ORM) SQLAlchemy 0.8.7. For image processing functionalities, OpenCV 2.4.9 and scikit-image 0.10.1 are used. For array manipulations, NumPy 1.8.2 is used to boost performance due to its implementation in C. The API that the frontend will interact with is written in PHP for compatibility reasons with the annotation platform.

The frontend is coded in Javascript using HTML5 functionalities for video displaying and canvas drawing. AngularJS has been the chosen framework. The frontend communicates asynchronously with the backend through AJAX requests. For the design, Bootstrap 3.2.0 has provided templates and animations.

The server where the backend is running is located at the VIPER laboratory and has the following specifications:

- Intel(R) Xeon(R) CPU E3-1225 V2 @ 3.20GHz
- 32 GB DIMM DDR3 1600 MHz
- Gigabit Ethernet interface

Part of this project was published in an IEEE paper and presented in the IEEE NAECON conference, held in Dayton, Ohio, from June 25th to June 27th, 2014. The title of the paper was “Automatic crowd flow estimation enhanced by crowdsourcing” and its authors were Javier Ribera Prat, Khalid Tahboub and Edward J. Delp.

An extension of it will be presented in the IS&T/SPIE EI conference, held in San Francisco, California, from February 8th to February 12th, 2015. For now, the title of the paper will be “Characterizing the uncertainty of classification methods and its impact on the performance of crowdsourcing”. Its authors will be Javier Ribera Prat, Khalid Tahboub and Edward J. Delp.



## Chapter 2

# Literature Review

In this chapter, we will mention and discuss some of the methods present in the literature for crowd flow estimation. Also, a brief review of studies about crowdsourcing will be done.

### 2.1 Crowd flow estimation

Crowd analysis is an extensively studied topic in the video analytics field. Analyzing the crowd includes detecting the direction that the crowd is moving, the speed, the number of people, the density, anomalous events (e.g. fighting) or any other pattern. Its applications extend beyond video surveillance. In [3, 6], a survey of many methods for crowd analysis employed in computer vision can be found.

Regarding crowd flow estimation, the approaches taken by different authors span a wide range of image processing techniques. Direct methods intend to estimate the crowd flow by detecting every individual and tracking them throughout the scene. This approach lacks scalability when dealing with large crowds, as simultaneous tracking of hundreds of targets is not robust and it becomes computationally too expensive. On the other side, indirect methods deduce crowd characteristics from features extracted from the image or video. Thus, this type of methods propose a relation between some low level features and the crowd flow.

In [7], the use of features such as edge orientation and blob size histograms is described. In [8], the number of people in the crowd is estimated using geometric, edge, and dynamic texture features. In [9], a novel spatial-temporal matrix, support vector machine (SVM) and mean-shift clustering are used to count pedestrians.

In [10], a linear relationship between the number of pixels in the foreground segmentation and the number of people is proposed, assuming a constant level of occlusion. In [11, 12], the relation between level of occlusion, or crowdedness, in an image and its texture is shown. In [4], this linear assumption was used to estimate the crowd flow and texture was incorporated to consider changing crowd densities.

This work is based on the indirect method explained in [4]. Some enhancements are proposed and crowdsourcing is incorporated to assist the method in its weak points.

## 2.2 Crowdsourcing

The crowdsourcing paradigm was introduced by J. Howe in 2006 in [13] and a definition proposed in [14]. Originally, it was conceived as a way to obtain services by soliciting them to a large group of undefined people, instead of using the internal resources of the requester [14]. By demanding the work to be done by an external crowd, not only a much more inexpensive, but also a diverse result can be achieved [13]. The vision of the whole picture that the “wisdom of the crowd” holds, usually leads a more accurate outcome [15]. Recently, a more wide and consistent definition of crowdsourcing was presented [16]. In this work, we refer as crowdsourcing to the use of a group of humans, the “o-crowd”, to assist the automatic analysis carried out by a machine in order to make it perform better.

[17] shows a study reviewing the current crowdsourcing systems on the world-wide web. Several commercial platforms that allow requesters to reach the public crowd already exist. Examples are Amazon Mechanical Turk (MTurk) [18], Freelancer [19], Mob4Hire [20], uTest [21], TopCoder [22], CloudCrowd [23], and CrowdFlower [24]. Issues concerning using commercial crowdsourcing systems for law enforcement arise, due to requirements like video contents protection [25]. In [5], a web-based tool to allow fine control over annotations of events on video surveillance videos is described.

Crowdsourcing has been widely used in the computer vision community for building up training data. The idea to make use of human intelligence to assist machines in an automatic method is not new. In [26], a scheme to enhance machine learning by using crowdsourcing is proposed. It is applied to apprehension of new objects by autonomous robots. Although manual human-provided segmentations of the objects of interest and supervised review of the tasks by experts are required, the models for these new objects are built from crowdsourced labels requested through MTurk. In [27], an object detection method was trained and the model of the classifier was continuously refined

---

using crowdsourcing. It iteratively identifies unlabeled data and automatically uploads annotation requests to MTurk.

In this work, the members of the “o-crowd” are reached through a web platform, specifically developed for crowd flow estimation, that allow controlled members to answer very tailored answers, with a view to integrate it into the video annotation tool [5]. A crowdsourcing-helped video surveillance system with many other detectors and applications would be, then, a long-term goal.

## Chapter 3

# Crowd flow estimation

In this chapter, we thoroughly describe the automatic crowd flow estimation method used in this work. It was originally conceived in [4] by Satyam Srivastava, Ka Ki Ng and Edward J. Delp at the Video and Image Processing Laboratory at Purdue University. Also, some proposed improvements are explained.

In Section 3.1, a general scheme of the method is presented as a first approach. An overview of the key ideas and components is explained. In Section 3.2, the process to obtain the foreground pixel count, which is directly related to the final estimation, is described in detail. In Section 3.3

### 3.1 Key ideas and overall scheme

The final goal is to estimate the number of people that have crossed a desired region of the image in a given time, i. e. crowd flow.

As stated before, this method follows an indirect approach. This is, it relates characteristics of the crowd to low level features. In this case, our low level feature is the number of foreground pixels. The assumption is that the number of people present is proportional to the number of foreground pixels.

The region of the space where people crossing will be counted is called “Tripwire” and it must be manually specified by the user.

We compute the crowd flow estimation proportionally to the accumulation of foreground pixels in the Tripwire. Thus, the proportionality rule results in

$$v_N = \frac{\tilde{S}_N}{C} \tag{3.1}$$

where  $v_N$  represents the number of people that have crossed the Tripwire up to the frame number  $N$ .  $S_N$  represents the accumulated foreground pixel count in that period. The details of the foreground pixel count will be discussed in Section 3.2.  $C$  is the proportionality factor, also called scaling factor, used to scale the number of foreground pixels to the amount of people crossing. It represents the amount of pixels that every person shows. Crowdedness is related to occlusions, and more occlusions means less pixels can be seen per person. Because of this reason, the lineal proportion holds true as long as the crowdedness remains constant. Thus, the value of  $C$  depends on the level of crowdedness of the scene. More explanation about the scaling factor and the level of crowdedness will be given in Section 3.3.

In addition to the Tripwire, the operator must also select another region called Region of Interest (ROI) used for the crowdedness estimation. An example of a Tripwire and a ROI over a surveillance videos are depicted in 3.1.

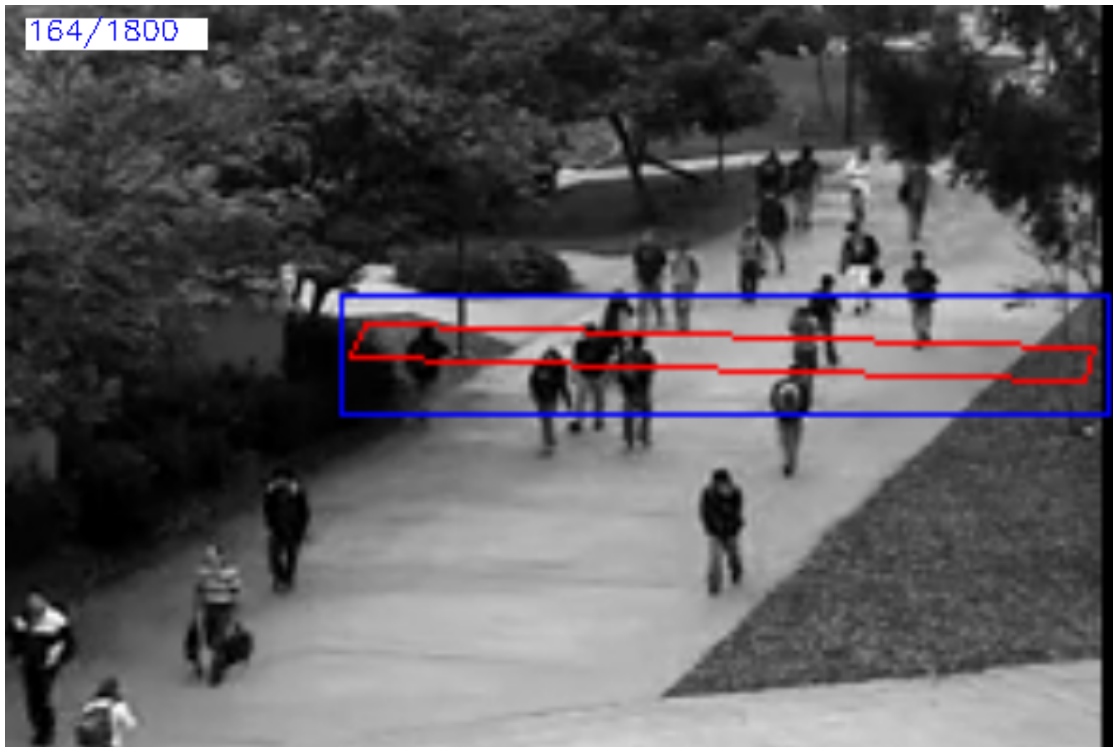


FIGURE 3.1: Example of a Region of Interest (ROI) and a Tripwire drawn over a frame of a surveillance video. ROI is in blue, and Tripwire in red.

An overall scheme of the whole method is shown in Figure 3.2. For each frame, the lower branch computes the foreground pixel count inside the Tripwire and the upper branch estimates the scaling factor from the density level of the ROI. The lower branch is detailed in Section 3.2 and the upper branch in Section 3.3.

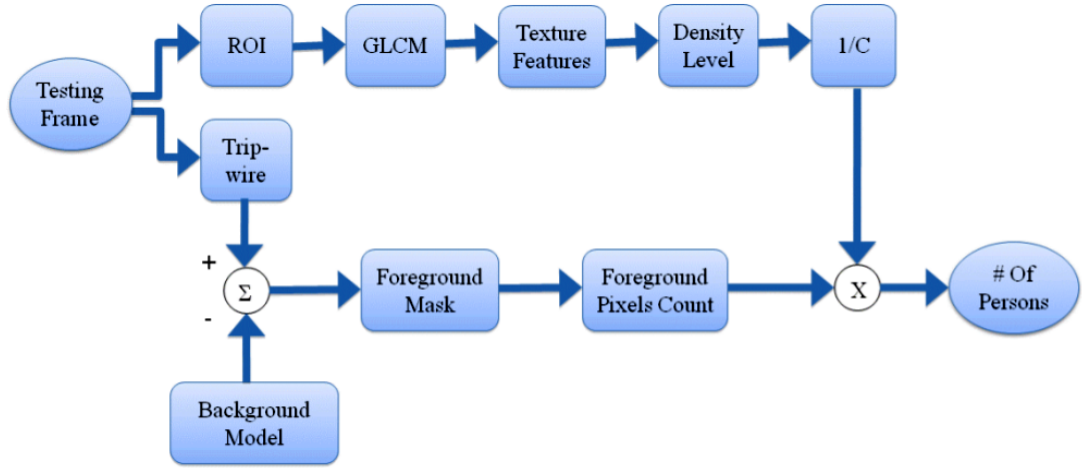


FIGURE 3.2: Overall scheme of the automatic crowd flow estimation method.

### 3.2 Foreground pixel count

In Figure 3.3, the process to obtain the foreground pixel count from the Tripwire is schematized.

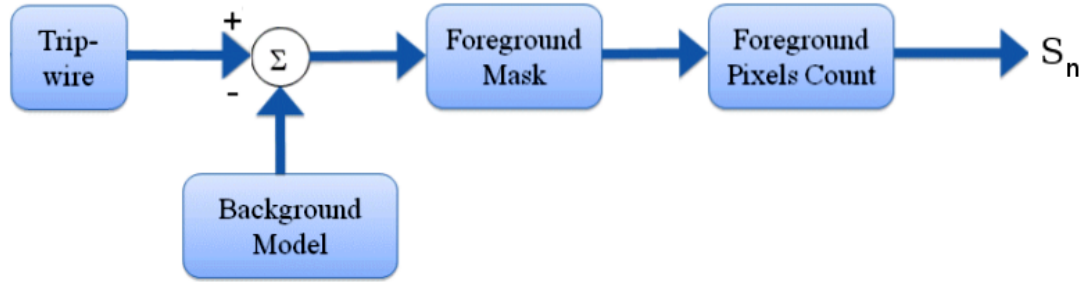


FIGURE 3.3: Scheme of the foreground pixel count.

Let the image of an arbitrary frame with frame number  $n$  be defined as

$$F_n = \{f_n(i, j) | i = 0, 1, \dots, W - 1 \text{ and } j = 0, 1, \dots, H - 1\} \quad (3.2)$$

where the  $f_n(i, j)$  is the value of the pixel (that may be a 3-D RGB value if it is not a gray-scale image) at width position  $i$  and height position  $j$ .  $W$  and  $H$  are the width and height of the frame, respectively. Then, we can represent the foreground mask of  $F_n$  as an indicative function  $I_n$  that is 1 when the pixel belongs to the foreground segmentation and 0 otherwise, i.e.,

$$I_n(i, j) = \begin{cases} 1 & : (i, j) \in \text{foreground segmentation} \\ 0 & : (i, j) \notin \text{foreground segmentation} \end{cases} \quad (3.3)$$

Also, let the Tripwire be represented as the set of pixels  $\mathfrak{R} = \{(i, j) | (i, j) \in \text{Tripwire}\}$ . An example of the foreground mask of the Tripwire is shown in Figure 3.4.

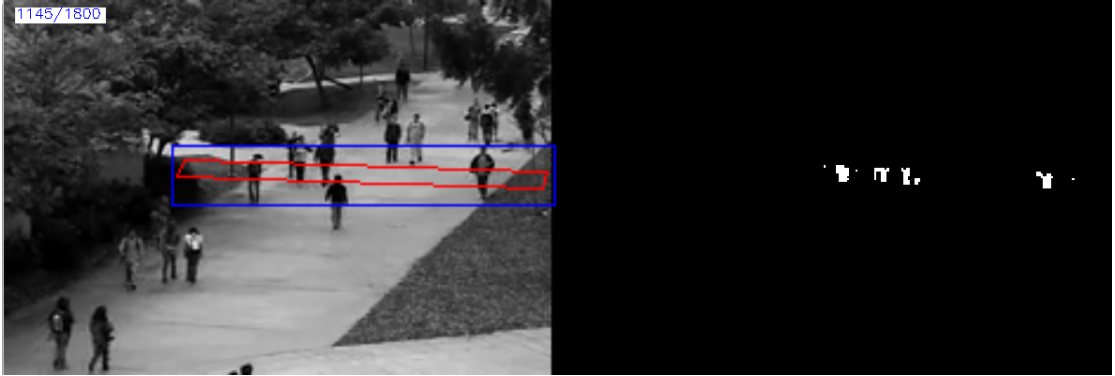


FIGURE 3.4: An arbitrary frame, on the left, and the foreground mask of the Tripwire on the right.

To obtain the foreground mask  $I_n$ , we apply a technique called Background Subtraction (BS), which is explained in detail in Subsection 3.2.2. There is only need to obtain the foreground mask of the Tripwire. Note, then, that the computational cost of performing Background Subtraction in a small region is much smaller than in the whole frame.

The foreground pixel count  $S_n$  of a frame  $F_n$  is then determined as how many pixels belong to the foreground segmentation, i. e.,

$$S_n = \sum_{(i,j) \in \mathfrak{R}} I_n(i, j) \quad (3.4)$$

The foreground pixel accumulation  $S_N$  up to the frame number  $N$  is the accumulation of foreground pixels in that period, i. e.,

$$S_N = \sum_{n=0}^N S_n = \sum_{n=0}^N \sum_{(i,j) \in \mathfrak{R}} I_n(i, j) \quad (3.5)$$

However, Equation (3.5) deals with all frames in the period the same way, regardless of their level of crowdedness. To overcome this limitation, an improvement of this equation that takes into consideration variance in the level of crowdedness will be provided and discussed in Section 3.3.

In addition, due to perspective distortions, the blob size of an object in the foreground mask varies according to the distance from the camera. Unfortunately, if Equation (3.5) is directly used, all pedestrians will be assumed to be at the same distance to the camera, which doesn't always hold true. This may lead to an inaccurate result of the crowd flow

estimation. A weighting function is introduced to consider perspective in the foreground mask.

### 3.2.1 Weighting function

Not only for perspective distortions, but also to account for the effect of perspective, velocities and direction of the movement of the pedestrians, a weighting function is needed. Nevertheless, only perspective distortion remains constant throughout the video, as neither the Tipwire boundaries or the position of the camera are supposed to move. Velocity and direction of movement change every frame. Hence, only perspective distortion was considered to reduce computational complexity.

The weighting function is represented as  $\omega(i, j)$  and weights every pixel  $(i, j)$  of the foreground mask  $I(i, j)$  as in Equation (3.6). The result of using a weighting function results in a weighted foreground count accumulation  $\tilde{S}_N$ .

$$\tilde{S}_N = \sum_{n=0}^N \sum_{(i,j) \in \mathfrak{R}} I_n(i, j) \cdot \omega(i, j) \quad (3.6)$$

The goal is to make the value of  $\omega(i, j)$  higher as the object at pixel  $(i, j)$  is further from the camera. Then, an object that provides less number of pixels to the foreground pixel count because it is far away from the camera will see its contribution increased because of the weighting function. The proposed algorithm to compute  $\omega(i, j)$  works as follows:

First, the user is asked to draw a quadrilateral which corresponds to a rectangle on the floor in the real world. This quadrilateral is defined by its four sides  $L_1$ ,  $L_2$ ,  $L_c$  and  $L_f$  as in Figure 3.5. The user is also asked to indicate the closer and further sides,  $L_c$  and  $L_f$ , respectively.

Due to perspective, lines  $L_1$  and  $L_2$ , which are parallel in the real world, appear to intersect at the vanishing point  $F$ . Second, for each point  $(i, j)$  inside the tripwire  $\mathfrak{R}$ , a line parallel to  $L_1$  and  $L_2$  in the real world is defined as the line that passes through  $(i, j)$  and the vanishing point  $F$ . This line intersects with  $L_c$  at the point  $Q$ . Third, we define a segment  $S_c$  centered at  $Q$  and laying on  $L_c$ . This segment  $S_c$  will always have a predefined length  $W_c$ . Forth, this segment is projected towards the vanishing point,  $F$ , until it reaches the original point  $(i, j)$ , resulting in a segment called  $S_f$  centered at  $(i, j)$  and parallel to  $S_c$ ,  $L_f$  and  $L_c$ . This segment  $S_f$  has length  $W_f$ . As a result of perspective, the resulting length  $W_f$  is not equal to  $W_c$ : it is shorter because it is further from the camera.



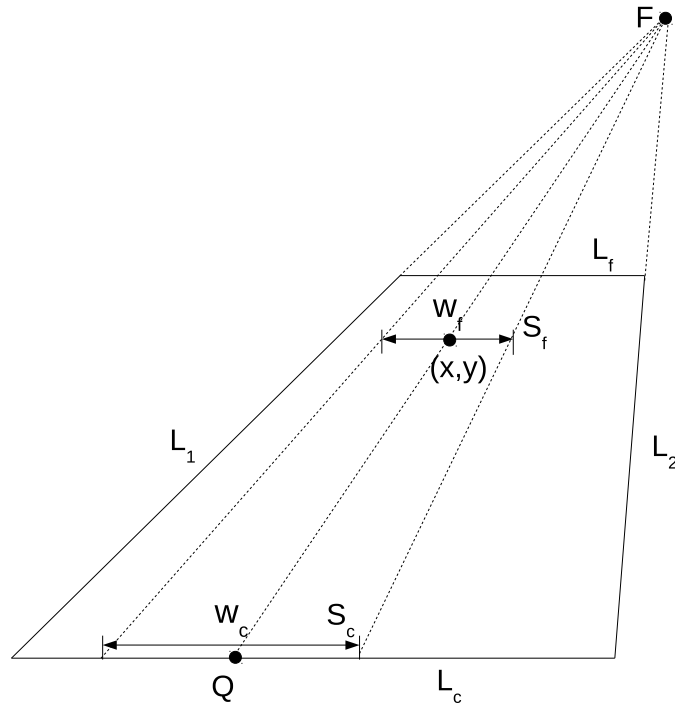


FIGURE 3.5: Weighting scheme illustration. The further a pixel is, the smaller the segment  $W_f$  becomes and the higher its weight results.

Therefore, the weighting function is defined as:

$$\omega(i, j) = \frac{W_c}{W_f} \quad (3.7)$$

$\omega(i, j)$  will not depend on the value of  $W_c$ , as  $W_f$  will be proportional to  $W_c$ . Because of this, the length  $W_c$  can be hardcoded and fixed with the same value for all points  $(i, j)$ .

As stated earlier, the resulting weighting function doesn't depend on  $n$ , so it can be computed only once before the crowd flow estimation even starts.

The implementation of this method was done using homogeneous coordinates [28], as it leads to simpler equations in perspective geometry. The code of the ‘‘Perspective Weight Calculator’’ can be found at `perspective_weight_calculator.py`. The code of classes for 2D homogeneous lines, segments and points, and related geometry functions was also implemented from scratch and can be found in the module `geometry.py` of the `basicClasses` package.

After some initial testing, there was hardly any quantitative improvement on the final estimation when using this weighting function. A reasonable explanation may be that the only dataset against which we tested exhaustively the whole method was the UCSD

dataset [29]. In this dataset, because of the position of the camera, any thin Tripwire drawn perpendicular to the footpath, will contain pixels approximately at the same distance to the camera. Figure 3.1 shows an example of a Tripwire in this dataset. This may have led to an almost constant weighting function.

Because of this, the weighting function was disabled for all testings, i.e,  $\omega(i, j) = 1, \forall(i, j)$ .

### 3.2.2 Background subtraction

To obtain the foreground mask, we employ Background Subtraction (BS). Background Subtraction is a widely used approach in computer vision, where the frame where background will be subtracted from is compared pixelwise to a “reference image” or a “background model”. If the difference in a given pixel is high enough, that pixel is considered as “foreground”. All the pixels considered as foreground conform the foreground segmentation.

Specifically, the technique used in this work, Mixture Of Gaussians (MOG) was conceived by Chris Stauffer and W.E.L Grimson in [30] and later improved in [31–33]. A more recent survey can be found in [34]. This technique is robust against shadows, lighting changes and objects being introduced and removed from the scene. It models the recent history of a pixel as a weighted sum of gaussian functions. Some of these gaussians represent foreground and other background. This means that the background model, in each pixel independently, consists of  $B, B < K$  of the  $K$  different gaussians  $N_g, g = 0, 1, 2, \dots, K - 1$ .

$$N_g(i, j) (f_n | \mu_{g,n}, \Sigma_{g,n}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma_{g,n}|^{1/2}} \exp\left(-\frac{1}{2}(f_n - \mu_{g,n})^T \Sigma_{g,n}^{-1} (f_n - \mu_{g,n})\right) \quad (3.8)$$

where  $f_n$  represents, as in Equation (3.2), the value of the pixel  $(i, j)$  at frame number  $n$ , e.g, scalar for grayscale images or vector for color images,  $\mu_{g,n}$  is the mean value of the gaussian number  $g$  of the pixel  $(i, j)$  at frame  $n$  and  $\Sigma_{g,n}$  its covariance matrix.

The gaussians in each pixel are weighted to conform the probability of a pixel  $(i, j)$  to have a specific value:

$$P(f_n) = \sum_{g=1}^K \omega_{g,n} \cdot N(f_n | \mu_{g,n}, \Sigma_{g,n}) \quad (3.9)$$

Accordingly to its definition as a probability function,  $P(f_n)$  should be normalized to 1. Then, given the fact that each gaussian is already normalized to 1, the weights of each pixel at any time instant should sum up to 1:

$$\sum_{g=1}^K \omega_{g,n} = 1 \quad (3.10)$$

The reason of the dependance on the frame instant  $n$  is that the model can be, and in fact is, constantly updated.

A representation of the mixture of gaussians of one pixel ( $P(f_n)$ ) for the case of a grayscale image ( $N = 1$ ) and with  $K = 4$  is shown at Figure 3.6.

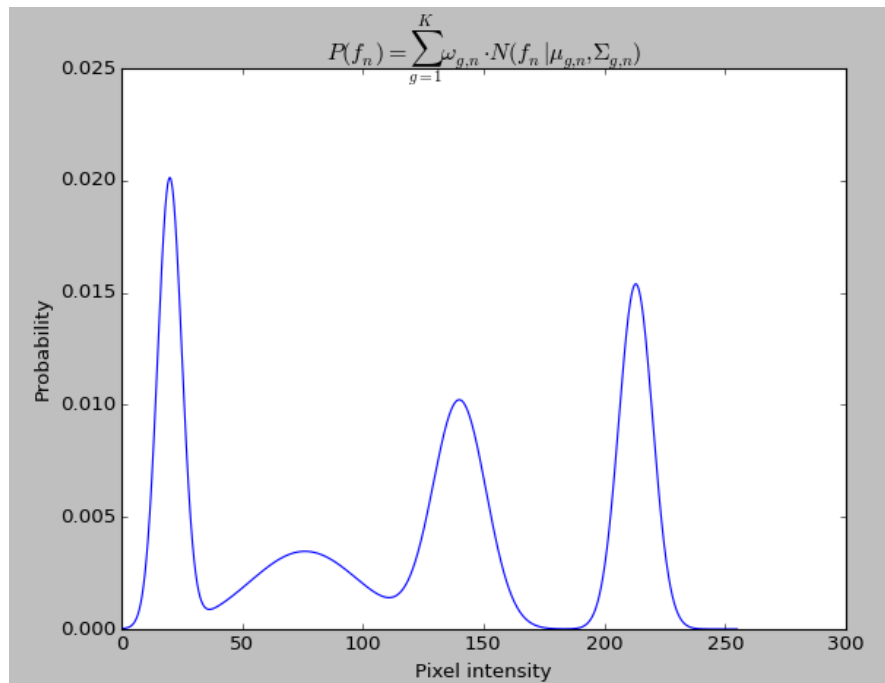


FIGURE 3.6: Mixture of Gaussians representation. Each gaussian is weighted and results in the model for the history of a pixel. In this case, the pixel is in 8 bits grayscale and the number of gaussians is  $K = 4$ . Some of the gaussians would be assigned to foreground and other to background.

C. Staufer *et al.* chose a gaussian function to model the background after observing that the RGB values of the background pixels seem to follow a gaussian distribution [30]. In this experiment they also observed the multi-modal distribution of the background values that led to consider more than one gaussian. Besides, the gaussian function is a very easily described function with only 2 values: mean and covariance.

The number of gaussians,  $K$ , is usually between 3 and 5 [30], but in [32] a mechanism was proposed to constantly and automatically adapt the number of components in the mixture of each pixel.

To reduce memory and processing consumption, the covariance matrix of each gaussian,  $\Sigma_{g,n}$ , is assumed to be:

$$\Sigma_{g,n} = \sigma_{g,n} \mathbf{I} \quad (3.11)$$

While this assumes that red, green and blue values are independent and of the same variance, Equation (3.11) avoids a costly constant matrix inversion and a scalar, instead of a matrix, has to be stored.

For every new frame, the new pixel value is checked against the existing gaussian distributions, so it can be categorized as foreground or background. If the pixel  $f_n(i, j)$  value falls into one of the existing gaussians (gaussian  $g_0$ ), there is a “match”, such gaussian is updated the following way:

$$\omega_{g_0,n} = (1 - \alpha)\omega_{g_0,n-1} + \alpha \quad (3.12)$$

$$\mu_{g_0,n} = (1 - \rho)\mu_{g_0,n-1} + \rho \quad (3.13)$$

$$\sigma_{g_0,n}^2 = (1 - \rho)\sigma_{g_0,n-1}^2 + \rho(f_n - \mu_t)^T(f_n - \mu_t) \quad (3.14)$$

where  $\alpha$  is the learning rate and

$$\rho = \alpha N_{g_0}(f_n | \mu_{g_0,n}, \sigma_{g_0,n}) \quad (3.15)$$

With  $\alpha = 0$  the background model is never updated, and with  $\alpha = 1$  only the last frame is used to build the model. In other words, the component that is still present “corrects” its mean toward the new value and adapts its variance to use the new value. The other gaussians remain untouched. The weights are renormalized after every update.

A match occurs if the pixel value is within 2.5 times the  $\sigma_g$  of a gaussian.

If the pixel value  $f_n(i, j)$  does not match any gaussian distribution, this is interpreted as a sudden appearance of a new object in the scene. In this case, in order to model this new object, a new gaussian is created centered at the current pixel value  $f_n$  with a predefined variance.

Background is expected to have lower variance and appear more often than foreground objects. Thus, a gaussian is more likely to represent a background object if the value of the ratio  $\frac{\omega}{\sigma}$  is higher.

Once a pixel has matched a given gaussian, in order to decide whether the pixel corresponds to foreground or background, the gaussians must be classified to belong to background or foreground objects. To do so, the gaussians are sorted by the value of  $\frac{\omega}{\sigma}$ , resulting in the most likely to be background to be at the top and the most likely to be at the bottom. The first  $B$  distributions that sum enough evidence are considered background:

$$B = \operatorname{argmin}_b \left( \sum_{g=1}^b \omega_g > T \right) \quad (3.16)$$

where  $T$  is the threshold that determines the minimum portion of data that should correspond to background.

OpenCV [35] provides an efficient implementation of a Background Subtractor using Mixture of Gaussians. OpenCV 2.4.9 contains a class called “BackgroundSubtractor-MOG2” that is based on the method explained here and the enhancements explained in [32, 33], which incorporate shadow detection and adaptive number of components. In this thesis, this is the implementation that has been used.

### 3.3 Crowdedness estimation

Once the weighted foreground pixels count  $\tilde{S}_N$  has been computed, it must be scaled properly to get the final people count. As shown in Equation (3.1), the proportionality factor is  $1/C$ . In order to estimate  $C$ , i.e, the number of pixels every person shows, the level of crowdedness must be estimated. In this thesis, we refer to level of crowdedness, crowdedness and level of occlusion as synonyms. Given that more crowdedness results in more occlusion between people and less pixels visible per person, a higher level of crowdedness should correspond to a lower value of  $C$ .

This method uses the approach described in [11, 12], where the relation between crowdedness and texture is detailed. As Figure 3.7 shows, the texture of an image reflects its crowdedness. A sparse scene, with a low level of crowdedness, presents a fine texture. In contrast, a crowded scene, with a high level of crowdedness, presents a coarse texture. Hence, we deduce the level of crowdedness of every frame using texture properties. An extensive study of texture in image processing can be found in [36].



FIGURE 3.7: A sparse scene, with a low level of crowdedness, presents a fine texture. In contrast, a crowded scene, with a high level of crowdedness, presents a coarse texture.

In this study, we characterize the texture of an image by means of a Gray Level Co-occurrence Matrix (GLCM). The GLCM matrix models a texture by characterizing the probability that a pixel with a given gray level is adjacent to another specific gray level. It consists on an  $S \times S$  matrix  $P_{ij}$  that contains the probability of “jumping” from gray level  $i$  to  $j$  when the image is scanned in a given direction, where  $i, j = 0, 1, \dots, S - 1$  and  $S$  is the number of quantized gray tones of the input image. Generally, the GLCM technique analyzes pixels separated by distance  $d$ , but in this method we focus on adjacent pixels, i.e, we fix  $d = 1$ . Figure 3.8 shows an example of GLCM calculations from a very simple image.

0	0	1	1
0	0	1	1
0	2	2	2
2	2	3	3

(a) 4x4 image consisting of 4 gray levels

$$P_H = \begin{pmatrix} 4 & 2 & 1 & 0 \\ 2 & 4 & 0 & 0 \\ 1 & 0 & 6 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}$$

(b) GLCM of 3.8(a) scanned in horizontal ( $\theta = 0^\circ$ )

$$P_V = \begin{pmatrix} 6 & 0 & 2 & 0 \\ 0 & 4 & 2 & 0 \\ 2 & 2 & 2 & 2 \\ 0 & 0 & 2 & 0 \end{pmatrix}$$

(c) GLCM of 3.8(a) scanned in vertical ( $\theta = 90^\circ$ )

$$P_{LD} = \begin{pmatrix} 2 & 1 & 3 & 0 \\ 1 & 2 & 1 & 0 \\ 3 & 1 & 0 & 2 \\ 0 & 0 & 2 & 0 \end{pmatrix}$$

(d) GLCM of 3.8(a) scanned in diagonal ( $\theta = 135^\circ$ )

$$P_{RD} = \begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 2 & 2 & 0 \\ 0 & 2 & 4 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

(e) GLCM of 3.8(a) scanned in diagonal ( $\theta = 45^\circ$ )

FIGURE 3.8: GLCM calculations ( $d = 1$ ).

Often, the GLCM is normalized to 1 by dividing all the elements by the number of elements, so it represents a probability, i. e,  $\sum_{i=1}^S \sum_{j=1}^S P_{ij} = 1$

Our goal is to obtain the level of crowdedness in the Tripwire for each frame. As stated in Section 3.1, the user must also manually select a Region of Interest, a rectangular region of the image, and it must surround the Tripwire. This will be the region where the crowdedness will be estimated from.

This method follows the scheme in Figure 3.9 to characterize the texture of every frame. From the ROI of a given frame, 4 GLCM matrix are created. Each matrix is computed by considering different directions for the adjacent pixels: right ( $0^\circ$ ), top-right ( $45^\circ$ ), top ( $90^\circ$ ), and top-left ( $135^\circ$ ). Then for each matrix, we extract 4 scalars features: energy (3.17), entropy (3.20), homogeneity (3.19) and contrast (3.18). With these 16 scalar features, a 16-D texture feature vector,  $t_n$ , is assembled to represent the texture of the ROI in the frame number  $n$ .

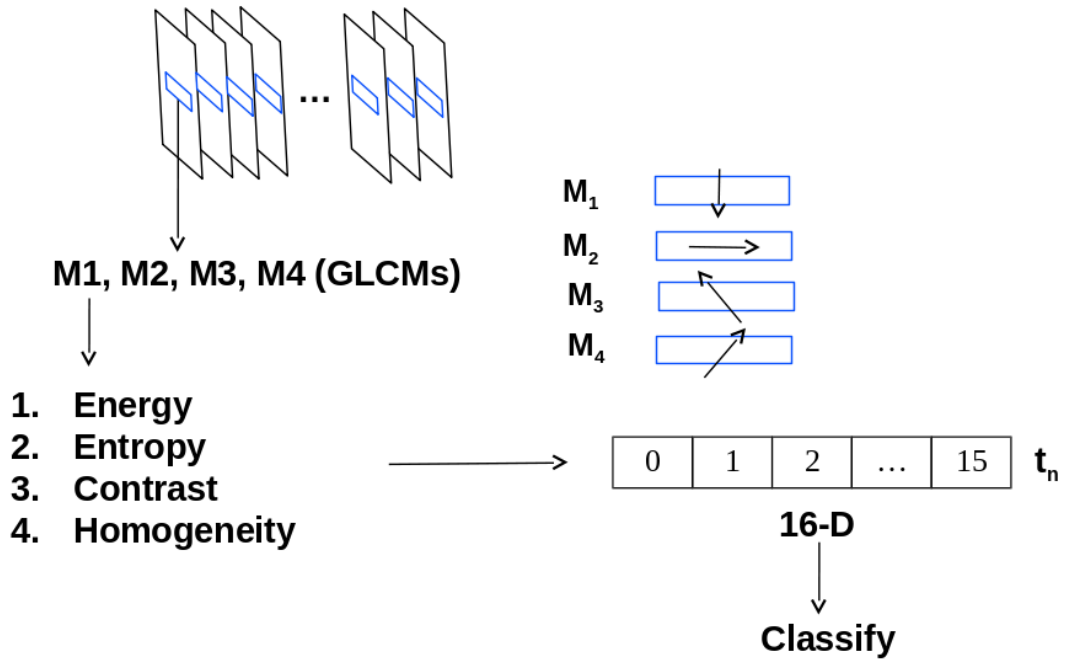


FIGURE 3.9: Scheme to obtain the texture feature vector.

“Energy” is defined to be the square root of the sum of squared elements in the GLCM and is maximum for a constant image. Contrast is a measure of the intensity contrast between a pixel and its neighbors over the whole image. Contrast is minimum for a constant image. Homogeneity measures the closeness of the distribution of elements in the GLCM to the diagonal of the matrix and is 1 for a diagonal GLCM. Entropy is a statistical measure of randomness.

$$Energy(P) = \sqrt{\sum_{i,j} p_{i,j}^2} \quad (3.17)$$

$$\text{Contrast}(P) = \sum_{i,j} p_{ij}(i-j)^2 \quad (3.18)$$

$$\text{Homogeneity}(P) = \sum_{i,j} \frac{p_{ij}}{1+(i-j)^2} \quad (3.19)$$

$$\text{Entropy}(P) = - \sum_{i,j} p_{ij} \log p_{ij} \quad (3.20)$$

Once we have the texture feature vector  $t_n$  of the ROI, we have to classify it into one of the  $L$  levels of crowdedness. As a result of the training process, explained in detail in Section 3.4, every level of crowdedness is represented by a texture feature vector, called “reference” vectors. Each reference feature vector is associated with its corresponding scaling factor  $C_l$ ,  $l = 0, 1, \dots, L-1$ . This is a classification problem in which we classify a 16-D vector into one of the levels of crowdedness represented by the  $L$  reference vectors. The closest reference vector is used for classification, this is a K-Nearest Neighbors (KNN) classifier in which  $K = 1$ . Figure 3.10 shows a 2-D representation of the feature space.

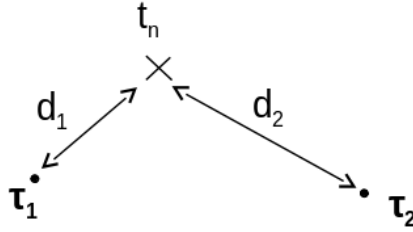


FIGURE 3.10: A 2-D representation of the 16-D feature space.  $t_n$  is the texture feature vector of the ROI and  $\tau_l$ ,  $l = 0, 1, \dots, L-1$  are the reference feature vectors of the  $L$  levels of crowdedness. The closest  $\tau_l$  to  $t_n$  is taken and its corresponding scaling factor is used.

The level of crowdedness  $l_n$  of a given frame is estimated as the level that the closest reference texture feature vector  $\tau_l$  represents, as shown in Equation (3.21):

$$l_n = \arg \min_l d(t_n, \tau_l) \quad (3.21)$$

The Euclidean distance function  $d(\cdot, \cdot)$  is used to compare feature vectors. As the range of the possible values of each feature might differ much, this may lead to the domination of one component in the distance measure. Because of this, we normalize the components of the feature vectors to approximately  $[0, 1]$ .

When the level of crowdedness is determined, the second step is to use the associated scaling factor  $C_n$  for this level of crowdedness and estimate the number of people as



determined in Equation (3.22), which is a combination of Equations (3.1) and (3.4):

$$v_N = \sum_{n=1}^N \frac{\sum_{x,y \in \mathcal{R}} I_n(x,y) \cdot \omega(x,y)}{C_n} \quad (3.22)$$

The code of the ‘‘Crowdedness Estimator’’ can be found at `crowdedness_estimator.py`. To compute the GLCM matrix and to extract the scalar features from it, the Python library `scikit-image` [37] 0.10.1 has been used.

### 3.4 Training data

This method requires a preliminary training stage to be performed by an expert, e.g, the operator of the system. The accuracy of the result is highly dependent on the quality of the training data and the exactitude of the training stage output.

Let  $L$  be the number of crowdedness levels specified by the operator for a video. As mentioned earlier, the scaling factor  $C$  relating the number of foreground pixels to the number of people crossing the Tripwire region is dependent on the level of crowdedness. Therefore, the training process aims to train the classifier to classify the 16-D texture feature vectors into one of the levels of crowdedness. The first output of the training stage is  $L$  texture feature vectors  $(\tau_0, \tau_1, \dots, \tau_{L-1})$  representing  $L$  levels of crowdedness. Each vector is used as a reference vector for the corresponding level of crowdedness. From now on we shall call these  $L$  texture feature vectors as reference vectors. During the training stage, the operator is asked to count the number of people crossing the Tripwire region during a short period of time and  $C_l$  is determined accordingly.  $C_l, l \in 0, 1, \dots, L - 1$ , is the second output of the training stage.

The training process is performed as follows: first, the system operator is asked to mark the Tripwire region and the ROI surrounding it. The perspective weighting function is determined according to Subsection 3.2.1. Figure 3.1 shows an example from our test dataset [29] including the Tripwire region and the ROI surrounding it. Next,  $M$  video frames are chosen randomly from the training video segment. For each frame, the operator is asked to classify each video frame into one of the  $L$  levels of crowdedness.  $\tau_l$ 's are estimated as the average of the training vectors for each corresponding level. To find the scaling factor  $C_l$  for each level, we find the longest stable period for each level of crowdedness. A stable period is a set of consecutive frames with the same level of crowdedness.  $L$  video segments corresponding to each level  $l$  are shown during training and the operator has to count the number of persons crossing the tripwire region,  $v_l$ . The scaling factor  $C_l$  for each level of crowdedness is determined by computing the

accumulation of foreground pixels in each video segment,  $\tilde{S}_l$ , and inverting Equation (3.1), which results in Equation (3.23):

$$C_l = \frac{\tilde{S}_l}{v_l} \quad (3.23)$$

A visual interface to guide the operator in the training data stage will be shown in Subsection 5.2.4.

## Chapter 4

# Crowdsourcing

The classification step and the resulting scaling factor of the current ROI is critical for the performance of the method. As any video analytics technique, shadows, distortions or sudden or gradual environmental changes may lead to poor accuracy of the result. In this crowd flow estimation method, a crowd distribution not captured in the training data or distortions in the video will change the texture, resulting in a misclassification of the texture feature vector. As the scaling factor is so sensitive to the level of occlusion, the foreground pixel count would be improperly scaled. In the end, a misclassification would imply a highly biased crowd flow estimation.

To overcome this, the performance of the classification is enhanced by using crowdsourcing. We aim to reduce the uncertainty in the classification by asking the “o-crowd”, the crowdsourcing members, to assist the automatic method.

### 4.1 Uncertainty of the classifier

For a frame  $n$ , the distance between the texture feature vector  $t_n$  and the nearest reference vector,  $d_1$ , might be very comparable to the distance between  $t_n$  and the second nearest neighbor,  $d_2$ . In this case, the classifier will choose the level of crowdedness corresponding to the nearest reference vector even if the difference between the two distances ( $d_2 - d_1$ ) is very small.

To overcome such uncertain classification decisions, we define “uncertainty”  $\mu$  as the ratio  $\frac{d_1}{d_2}$  to represent the uncertainty of the classification decision:

$$\mu = \frac{d_1}{d_2} \tag{4.1}$$

By definition  $d_1 \leq d_2$  and therefore  $\mu \in [0, 1]$ . Greater values of  $\mu$  represent greater uncertainties as  $d_1$  and  $d_2$  values are comparable.

## 4.2 Use of crowdsourcing

For each frame  $n$ , the uncertainty of its classification  $\mu_n$  is watched. Therefore, whenever  $\mu_n$  is greater than a predefined threshold,  $\alpha$ , the o-crowd is asked to classify the uncertain frame number  $n$ . In other words, this is formally performed when Equation (4.2) is fulfilled.

$$\mu = \frac{d_1}{d_2} > \alpha \quad (4.2)$$

$\alpha$  represents the maximum uncertainty allowed in the classifier and should be between 0 and 1, as uncertainty is. For example,  $\alpha = 0.5$  requires the feedback of the o-crowd whenever the distance between  $t_n$  and the second closest reference is less than twice the distance to the closest reference vector. We call  $\alpha$  the ‘‘crowdsourcing parameter.’’ It is related to how often we refer to the o-crowd, i.e. how often we ask the o-crowd for help. A lower value of alpha implies a lower utilization of the o-crowd, whilst a higher value makes the o-crowd to be asked more often. For instance, if  $\alpha = 0$ , the o-crowd will be referred every frame, as Equation (4.2) will always be fulfilled by definition, and if  $\alpha = 1$  the o-crowd will never be referred, as Equation (4.2) can never be true.

This threshold defines a ‘‘certainty area’’ around the reference vectors. Figure 4.1 illustrates a 2-D representation of the certainty areas around two reference vectors. Feature vectors outside all certainty areas are automatically referred to the o-crowd. The video frame corresponding to this  $t_n$  is shown to the o-crowd and they are asked to estimate its level of crowdedness and the number of people in the crowd.

In Figure 4.1, each circle represents a certainty area around the reference vectors for a specific value of  $\alpha$ . When  $\alpha = 1$ , the entire feature space is divided into two regions, each region is a certainty area around one of the two reference feature vectors. This corresponds to the case when crowdsourcing is not used. Choosing  $\alpha = 0.8$  reshapes the certainty areas to circles, each circle surrounding one of the reference feature vectors. During testing, any video frame which has a resulting texture feature vector outside both circles will be referred to the o-crowd for evaluation. Therefore, by controlling the size of each circle we control how often we ask the o-crowd for help. Choosing  $\alpha = 0.6$  or  $0.4$  clarifies this case as the size of the certainty areas become smaller and it is likely that the o-crowd is engaged more often.

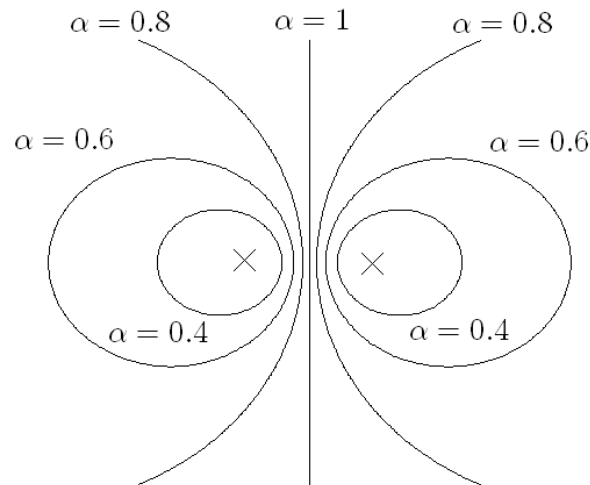
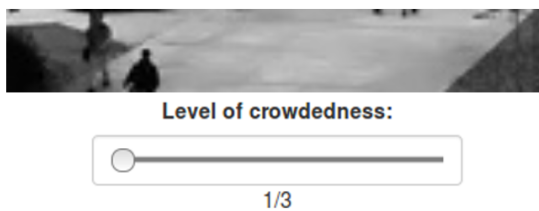


FIGURE 4.1: Certainty areas around two reference vectors corresponding to four values of  $\alpha$ . Reference vectors are represented by cross signs.

### 4.3 Crowdsourcing task

When the o-crowd is asked to assist the classifier because a frame is too uncertain, i.e., higher than the desired threshold, we create a “crowdsourcing task”. In Amazon MTurk, these are called HITs, Human Intelligence Tasks, but in this thesis we will refer to them more generally as crowdsourcing tasks. The members of the o-crowd are requested to solve it. In this case, the task consists of classifying the uncertain frame and counting people crossing the Tripwire.

#### 1. Classify crowdedness



#### 2. People count



FIGURE 4.2: Certainty areas around two reference vectors corresponding to four values of  $\alpha$ . Reference vectors are represented by cross signs.

As shown in Figure 4.2, first, the o-crowd member is asked to classify the ROI of the uncertain frame into one of the  $L$  levels of crowdedness. Second, the number of people crossing the Tripwire in a short video segment centered at the uncertain frame number must be provided. The length of the video must be short enough to contain only one level of crowdedness but long enough to allow the human to be able to count people, as counting fractions of bodies is notably hard. In this work, the length of the video segment was fixed empirically to 200 frames <sup>1</sup>.

With this information, a new scaling factor can be computed the same way as in the training stage as in Equation (4.3).

$$C = \frac{\tilde{S}}{v} \quad (4.3)$$

where  $v$  is the people count provided by the o-crowd and  $\tilde{S}$  is the accumulation of foreground pixels in the video segment of the crowdsourcing task. The formerly uncertain feature vector  $t_n$  now has been provided a new scaling factor and can be used in the estimation.

In addition, when using crowdsourcing, we make sure not to ask the o-crowd for redundant information. This is particularly important because consecutive video frames are likely to have similar feature vectors. Therefore, we store the  $t$ 's for which the o-crowd has provided feedback as new reference vectors, as well as their recently computed scaling factors. Accordingly to a reference vector, implicit certainty areas centered at the  $t$ 's are created. Hence, future feature vectors in next frames will not be referred to the o-crowd, as they will likely fall into the certain area of the new reference vector. Also, future classifications will perform better by taking into consideration this new reference vector.

However, the reference feature vectors coming from the training data have been averaged from many frames and the training process has been performed by an expert operator. Thus, the original reference vectors are more trustworthy than a reference computed from only one frame. To take this into account, certainty areas surrounding o-crowd reference vectors are scaled down. This is done by reducing  $\alpha$  by a factor of 0.9 only for reference vectors produced by crowdsourcing.

In conclusion, we make use of crowdsourcing to help the automatic method whenever the classifier analyzes a frame that would lead to a poor estimation. Furthermore, we achieve active learning by incorporating the information provided by the o-crowd to the classifier in order to make the method learn from crowdsourcing.

---

<sup>1</sup>200 frames in our dataset [29], which runs at 30 fps, corresponds to 6.6 s.

# Chapter 5

## Web platform

In this chapter, a brief overview of the whole system from the code point of view will be shown. Also, the Graphical User Interface (GUI) of the front-end of the web platform is presented. With this web application, the user can interact with the system. It allows the operator to control every aspect of the crowd flow estimation and lets the o-crowd members to solve crowdsourcing tasks.

### 5.1 System overview

Figure 5.1 shows a conceptual scheme of the structure of the whole system, divided in the backend, run in the server, and the front-end, run in the client web browser.

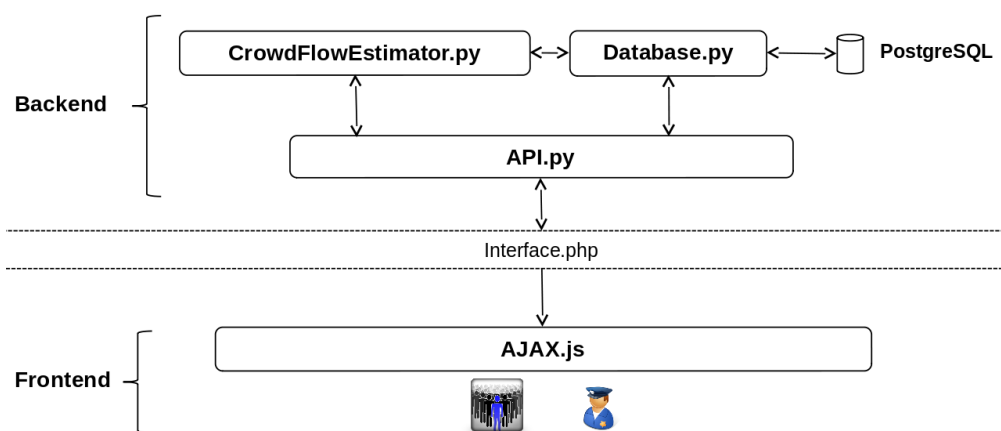


FIGURE 5.1: Overview of the system. Video analytics are performed in the backend server and the user interacts with the front-end, that sends asynchronous requests to the API in the server.

### 5.1.1 Backend

Originally, the crowd flow estimation method was coded in MATLAB. Although it was enough for proof-of-concept testings, such development environment has strong drawbacks in the long term, such as poor performance and licensing problems. Besides, the original implementation consisted on a plethora of independent scripts that lacked consistency to be employed in a real scenario. Because of this reason, the original idea was implemented from scratch.

The main programming language used to code the backend was Python. The reason was its rapid prototyping and easy readability. Also, it has many scientific libraries built in C that boost the performance of the code, like NumPy [38] for operations with arrays and OpenCV [35] and scikit-image [37] for image processing.

The server holds all the computations. The `CrowdFlowEstimator` class, in the module `crowd_flow_estimator.py`, is the point of entry where the estimation starts. The foreground segmentation is relegated to the class `ForegroundDetector`, in the module `foreground_detector.py`, which is a wrapper around OpenCV's class `BackgroundSubtractorMOG2` with custom added functionality. The `CrowdednessEstimator` class, in the module `crowdedness_estimator.py`, is in charge of computing texture feature vectors, looking for the nearest neighbour and computing the uncertainty. The `PerspectiveWeightsCalculator` class, in the module `perspective_weights_calculator.py`, computes the weighting function explained in Subsection 3.2.1. The `Trainer` class, in module `trainer.py`, is responsible of generating new training data at the operator's request. The `basicClasses` package contains modules that take care of geometry (`geometry.py`), building, drawing and managing ROIs and Tripwires (`roi.py` and `tripwire.py`, respectively), and packaging and normalizing texture feature vectors (`texture_features.py`). Miscellaneous classes and functions for image and video manipulations, mathematical and coding functions, and control of UNIX processes are contained in the modules of the `utils` package.

Also, any non-volatile information is stored in a PostgreSQL database through the Object-Relational Mapper (ORM) SQLAlchemy [39]. This allows to access the database in a high level with Python language. The ORM provides a mapping between an instantiation of a Python class with a row of a database table. In addition, it makes trivial the change to any other database such as MySQL, Oracle or SQLite, as it is dialect-agnostic.

The interaction with the database is performed with the package `database`. Any reading or writing to the database is controlled by the class `Database` of the module `database.py`, that maps any object stored in the database to a class in the module `sql_classes.py`. This package makes sure that every time that is used, the tables on the server database



correspond to the tables definitions in the `definition.py` module. If not, it creates or modifies the database tables accordingly.

Any external interaction with the backend is done through an API composed of some scripts in Python. This public API makes sure the input parameters are acceptable. It can be accessed manually by the operator (through the command line) or via web through an interface that is exposed to the web server (in this case Apache). The interface is, in turn, composed of scripts written in PHP. The choice of PHP was because of compatibility reasons with the annotation platform, that had already been developed in PHP. We wanted to stick to use Python for the backend but make it easy to incorporate to the annotation platform in the future.

The versions used in this work were: Python 2.7.5, SQLAlchemy 0.8.7, OpenCV 2.4.9, scikit-image 0.10.1, NumPy 1.8.2, PHP 5.5.19, Apache 2.4.10 and PostgreSQL 9.3.5.

The machine which acts as server is located at the VIPER laboratory and has the following specifications:

- Intel(R) Xeon(R) CPU E3-1225 V2 @ 3.20GHz
- 32 GB DIMM DDR3 1600 MHz
- Gigabit Ethernet interface

### 5.1.2 Front-end

The front-end was programmed using Javascript (JS) and makes use of CSS3 and HTML5 elements like the `<video>` tag. It should run properly on any modern web browser that respects the web standards, as it uses no proprietary extension.

Google's AngularJS [40] 1.3.0 was used as web framework to build a web application following the model-view-controller (MVC) architecture and to allow modularization of the code. Data binding to DOM elements was of special interest to write clear code.

RequireJS [41] 2.1.14 allows to manage JS dependencies. It was used to download and load AngularJS modules that depend on each other asynchronously without breaking dependencies. This leads to improvements in loading times and better code encapsulation in small files. Such approach is called Asynchronous Module Definition (AMD).

Twitter's Bootstrap [42] 3.2.0 provided templates and animations for a nice design of the front-end.

Files in `services` are pieces of code that offer small functionalities and are built as Angular services that are injected into Angular modules as dependencies. Files in `services/backend` send AJAX requests to the appropriate URI of the server where the desired API functionality is exposed. `services/backend/crowdsourcing.js` file is used for interactions relating crowdsourcing tasks, `services/backend/processes.js` for interactions relating crowd flow estimation processes and `services/backend/trainingData.js` for interactions relating training data. `services/ZoneSelector.js` contains a class that allows the user to select an area in a `<canvas>` HTML5 tag. These services are injected into the modules in `apps` folder when needed. Each file in `apps` isolates the functionality of the sections of the front-end that correspond to the sections of the GUI explained in Section 5.2.

### 5.1.3 Real-time streams

The system can analyze real-time video streams that may come from camera networks. To test this capability, a real-time stream is generated from a video file using `ffmpeg`:

---

```
ffmpeg -re -i videos/ucsd/long.mp4 -vcodec vp8 -f rtp rtp://localhost:70000 \  
> ~/PycharmProjects/crowd/streams/1.sdp ; \  
rm ~/PycharmProjects/crowd/streams/1.sdp
```

---

This generates an `.sdp` file describing the incoming real-time stream. Reading from this `.sdp` file from the crowd flow method is equivalent to read directly from a video file. This is an `ffmpeg` functionality and `OpenCV` makes use of `ffmpeg` underneath.

In order to be able to generate the video segment needed for the crowdsourcing task, a custom buffer mechanism specifically tailored to this application was developed. In general, this is a sliding buffer centered at the frame to be analyzed and the same length as the video segment of the crowdsourcing task. To generate a video segment, it is enough to join all the frames in the buffer into a video file. This buffer is implemented in the class `Buffer`, in the module `programming.py`, in the package `utils`.

## 5.2 Graphical User Interface

A user-friendly GUI is crucial for any web application that is intended to be used by a regular user. This web platform will be used by the operator of the system to control the video analytics and by the members of the o-crowd to solve crowdsourcing tasks. Neither of these can be assumed to be programmers or engineers. The following subsections will cover the functionalities of every section of the GUI.

### 5.2.1 Monitor of processes

The first tab (Figure 5.2) lets the operator (e.g, a police officer or the owner of the system) to monitor every process which is running a crowd flow estimation in the server. Each process can analyze only one video file or one real-time stream. For each process, these are the aspects that can be monitored:

- The Process ID (PID) of the UNIX process
- The video being analyzed (it can be displayed in a modal box)
- The progress of the analysis (frame position over total number of frames)
- The people that have been counted crossing the Tripwire so far
- The last time when we have information about this processed
- Whether this process has emitted a crowdsourcing task and it has not been solved yet
- How many crowdsourcing tasks have been emitted
- The value of the crowdsourcing parameter
- The name (or label) used to reference the training data that this process is using
- The number of reference feature vectors that the classifier contains
- Whether the “time machine” option is enabled or not
- Whether the process has reached the end of the video or not

The operator can also stop the analysis using the “×” button. This implies killing the UNIX process running in the server.

Finally, the slider in the lower part sets the interval to retrieve or refresh information from the server.

The time machine option makes the crowd flow estimation process store the frame number at the instant when a crowdsourcing request is created. Then, whenever this task is solved, it goes back to that frame number. This rewinding allows to use all the new information the method has available to improve the estimation as much as it can, as by default the estimation doesn’t stop because a crowdsourcing task has been emitted. Obviously, when a real-time stream is being analyzed, it makes no sense to go back in time because we the method to be real-time. This is why in the tab to invoke processes, as shown in Subsection 5.2.2, this option is disabled when the operator selects a real-time video stream.

# AUTOMATIC CROWDFLOW ESTIMATION ENHANCED BY CROWDSOURCING

PROCESS MONITOR
Process invoker
Crowdsourcing tasks
Training

## Processes monitor

PID	Video feed	Frame position	Total frames	Accumulated people count	Last update	Pending crowdsourcing request	Number of times referred to o-crowd	Crowdsourcing parameter	Training data	Number of references	Time machine	DONE	KILL
15322	<a href="#">videos/ucsd/long.mp4</a>	2730	97291	74.30	2014-12-08_16:33:02:547090	YES	1	0.6	training1	3	YES	NO	✘
15497	<a href="#">videos/1.ogg</a>	12	2118	0.00	2014-12-08_16:33:04:119663	NO	0	0.6	training2	3	NO	NO	✘

Keep updated

Frequency of update [s]:  2 seconds

FIGURE 5.2: Web platform: processes monitor. This tab lets the operator watch processes running a crowd flow estimations in the server.

### 5.2.2 Invoker of processes

In the second tab, the operator can spawn new processes in the server that will analyze a video file or a real-time stream. Figures 5.3 and 5.4 display this screen.

The screenshot shows the 'Process invoker' interface with the following settings:

- Video input type:**  File,  Stream
- Video source:**
- Training data (label):**
- Automation factor:**
- Time machine:**
- 

FIGURE 5.3: Web platform: processes invoker. A video file is selected as video input type.

The screenshot shows the 'Process invoker' interface with the following settings:

- Video input type:**  File,  Stream
- Video source:**
- Training data (label):**
- Automation factor:**
- Time machine:**  *Disabled when a real-time stream is the input video feed.*
- 

FIGURE 5.4: Web platform: processes invoker. A real-time video stream is selected as video input type.

If the operator has selected a video file, he must write the path to the file (a warning will be arisen if the file does not exist). If he has selected a real time video stream, a list of incoming streams will be shown and one must be selected.

### 5.2.3 Crowdsourcing tasks


Process monitor Process invoker CROWDSOURCING TASKS Training

## Pending crowdsourcing tasks

Select	id	PID	Label
<input type="radio"/>	151	24043	2014-12-08_20:05:777636
<input checked="" type="radio"/>	154	26259	2014-12-08_20:11:903711

### 1. Classify crowdedness

Classify the level of crowdedness of the next image:




Level of crowdedness:

 /3

### 2. People count

Count how many people are crossing the Tripwire in this video:



How many?

[Solve Task](#)

FIGURE 5.5: Web platform: crowdsourcing tasks. The o-crowd can select and solve a pending crowdsourcing task.

The screen shown in Figure 5.5 is the only one available for the non-operator users. In it, the members of the o-crowd can select one of the pending crowdsourcing tasks, i.e., a crowdsourcing task that has not been solved yet. A crowdsourcing task contains the PID of the process that generated it and when it was generated. Once the desired task is selected, the questions to the o-crowd will appear on the bottom. The o-crowd member can move the slider to indicate the level of crowdedness of the uncertain frame and type how many people are crossing the Tripwire in the video segment. Once all answers are provided, the button “Solve Task” sends the solved crowdsourcing task to the server so it can be incorporated to the process that emitted it.

### 5.2.4 Training

In this tab, the operator can either check in the upper section all the available training data and their parameters, or create new training data through the guide of process of the bottom section.

## AUTOMATIC CROWDFLOW ESTIMATION ENHANCED BY CROWDSOURCING

Process monitor Process invoker Crowdsourcing tasks **TRAINING**

## Training data



id	Label	Training video	Levels of crowdedness	Reference texture features ids	ROI id	Tripwire id	BS threshold	BS learning rate	Delete
4	training1	videos/ucsd/long.mp4	3	859,860,861	28	30	400	-1	
5	training2	videos/1.ogg	3	852,853,854	27	29	400	-1	

## New training data

## Steps

1. Video
2. Label
3. Tripwire
4. ROI
5. Weighting Map
6. Crowdedness classification
7. People count
8. BS Parameters

Clear all

## 1. Training video

Video to use for training:

Type a video file.

FIGURE 5.6: Web platform: training. The upper section shows the available training data and the lower section allows to create new training data.

The process to create new training data consists of the following steps. If a step is not satisfactory, i.e, no answer or a nonsense is provided, the next step will not be available. Answers of a given step may be undone and overwritten by clicking on the back button (arrow to the left). First, as shown in the lower section of Figure 5.6, the operator must type the video path. Its existence will be checked. Second, as shown in Figure 5.7, the name or label for the future training data must be written. It must be unique, so the backend will check if such label already exists.

## New training data

## Steps

1. Video
2. Label
3. Tripwire
4. ROI
5. Weighting Map
6. Crowdedness classification
7. People count
8. BS Parameters

Clear all

## 2. Label

New training data label:

new\_training\_data

FIGURE 5.7: Web platform: new training data, step 2. The label of the future training data can be chosen.

Third, as shown in Figure 5.8, the Tripwire must be provided. The GUI allows to draw a Tripwire on the surveillance video by clicking and dragging its vertices.

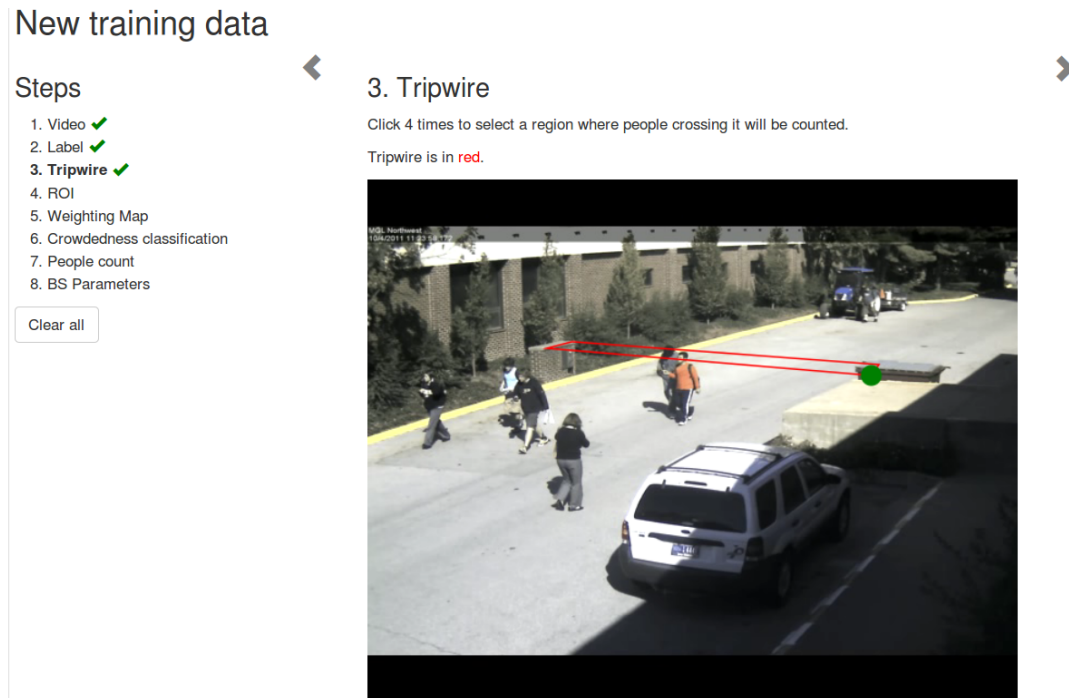


FIGURE 5.8: Web platform: new training data, step 3. The Tripwire must be drawn over the surveillance video.

Forth, as shown in Figure 5.9, the ROI must be provided. The same way as with the Tripwire in the previous step, the GUI also allows to draw a ROI on the surveillance video by clicking and dragging its vertices.

The fifth step corresponds to the weighting scheme explained in Subsection 3.2.1. As stated previously, the improvements were not noticeable, so this step is disabled.

Sixth, as shown in Figure 5.10, the operator is asked to classify the level of crowdedness of some random snapshots focused on the ROI. The number of levels of crowdedness, as well as the number of random snapshots to be asked for classification, is configurable.

Seventh, as shown in Figure 5.11, the backend computes and returns as many video segments with a Tripwire drawn on them as levels of crowdedness. These video segments contain only one level of crowdedness in all their frames. The user is asked to provide the people count of each of them.

Lastly, Figure 5.12 shows the final step, where the values of the parameters of the Background Subtraction can be specified. The button labeled “Assemble new training data” will send all the information collected from the user to the backend and new training data will be computed, hence appearing its corresponding entry to the table of the upper part of the screen shown in Figure 5.6.



## New training data

Steps

1. Video ✓
2. Label ✓
3. Tripwire ✓
4. ROI ✓
5. Weighting Map
6. Crowdedness classification
7. People count
8. BS Parameters

[Clear all](#)

### 4. Region of Interest

Click and select a region that represents the level of crowdedness of the Tripwire.

Tripwire is in **red**.  
ROI is in **blue**.



The image shows a surveillance video frame of an outdoor area with a white SUV in the foreground. A red line (tripwire) and a blue rectangle (ROI) are overlaid on the scene. The ROI is a blue rectangle that encompasses the tripwire and the area where people are walking. The tripwire is a red line that crosses the ROI.

FIGURE 5.9: Web platform: new training data, step 4. The ROI must be drawn over the surveillance video.

## New training data

Steps

1. Video ✓
2. Label ✓
3. Tripwire ✓
4. ROI ✓
5. Weighting Map ✓
6. Crowdedness classification
7. People count
8. BS Parameters

[Clear all](#)

### 6. Crowdedness classification

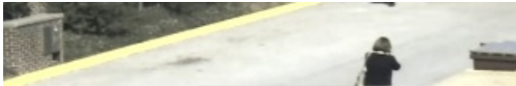
Levels of crowdedness:

Number of random snapshots to get:

[Retrieve 10 random snapshots](#)


Classify the level of crowdedness of each of these snapshots as {empty|low|...|high} density:

Snapshot 1/10:



Level of crowdedness:  1/3 [Change this snapshot](#)

Snapshot 2/10:



Level of crowdedness:  2/3 [Change this snapshot](#)

FIGURE 5.10: Web platform: new training data, step 6. The random snapshots must be classified into one of the levels of crowdedness.

## New training data

### Steps


1. Video ✓
2. Label ✓
3. Tripwire ✓
4. ROI ✓
5. Weighting Map ✓
6. Crowdedness classification ✓
- 7. People count**
8. BS Parameters

Clear all

### 7. People count

Retrieve videos

Count how many people are crossing the next videos (can be decimal):  
Level of crowdedness: 1 / 3



People crossing the Tripwire: 2.5

FIGURE 5.11: Web platform: new training data, step 7. The people count for the video segment corresponding to each level of crowdedness must be provided.

## New training data

### Steps

1. Video ✓
2. Label ✓
3. Tripwire ✓
4. ROI ✓
5. Weighting Map ✓
6. Crowdedness classification ✓
7. People count ✓
- 8. BS Parameters**

Clear all

### 8. Background Subtractor parameters

**Threshold** 50

**Learning rate** 0.001  Automatic

**Background Model initialization**

Before background is subtracted, the Background Model will be initialized using the following frames of the training video:

**Initial frame** 0

**Final frame** 300

Build background model "on-the-go"

[Preview](#)

Assemble new training data ↻ This may take up to some minutes...

FIGURE 5.12: Web platform: new training data, step 8. The parameters of the Background Subtraction can be configured in this screen.

However, before sending the definitive training data to the server, it is convenient to check the effect of the BS parameters on the foreground segmentation. The button “Preview” of Figure 5.12 raises a modal box that shows such preview. The foreground segmentation of the training video is computed in the server with the provided parameters and the resulting video is shown to the user. This way the user can fine-tune the BS parameters to achieve the desired foreground segmentation.




FIGURE 5.13: Web platform: new training data, BS preview. The effect of the BS parameters on the foreground segmentation can be tested using this preview feature.

### 5.2.5 Command History

Finally, this complementary feature of the web application may be used for debugging purposes. With the screen shown in Figure 5.14, the operator can watch a history of the commands run in the server. This may be useful to verify that the system is being used in a proper manner.

Command history

🔄 Last  commands 

Command	Timestamp
<code>python '/home/javiribera/PycharmProjects/crowd/public_api/command_history.py' '--num-commands-to-retrieve' '50'</code>	2014-12-09_11:48:38:585812
<code>python '/home/javiribera/PycharmProjects/crowd/public_api/processes_running.py'</code>	2014-12-09_11:48:38:236379
<code>python '/home/javiribera/PycharmProjects/crowd/public_api/processes_running.py'</code>	2014-12-09_11:48:36:199429
<code>python 'public_api/kill_process.py' '--pid' '20217'</code>	2014-12-09_11:48:33:008664
<code>python '/home/javiribera/PycharmProjects/crowd/public_api/processes_running.py'</code>	2014-12-09_11:48:32:651054
<code>python '/home/javiribera/PycharmProjects/crowd/public_api/processes_running.py'</code>	2014-12-09_11:48:30:700921
<code>python '/home/javiribera/PycharmProjects/crowd/public_api/processes_running.py'</code>	2014-12-09_11:48:28:658235
<code>python '/home/javiribera/PycharmProjects/crowd/public_api/processes_running.py'</code>	2014-12-09_11:48:26:581538
<code>python 'crowd_flow_estimator.py' '--webserver' '--automation' '1' '--training' 'training1' '--video' 'videos/ucsd/long.mp4'</code>	2014-12-09_11:48:23:562794
<code>python '/home/javiribera/PycharmProjects/crowd/public_api/processes_running.py'</code>	2014-12-09_11:48:18:557483
<code>python '/home/javiribera/PycharmProjects/crowd/public_api/processes_running.py'</code>	2014-12-09_11:48:16:578923
<code>python '/home/javiribera/PycharmProjects/crowd/public_api/command_history.py' '--num-commands-to-retrieve' '50'</code>	2014-12-09_11:47:43:411092
<code>python '/home/javiribera/PycharmProjects/crowd/public_api/training_datas.py'</code>	2014-12-09_11:47:42:899553

FIGURE 5.14: Web platform: command history.

To avoid overload, only the last commands are shown, but it can be completely configured. Also, the operator can clean the history with the trash button.

## Chapter 6

# Experimental results

### 6.1 Testing conditions

#### 6.1.1 Dataset

The dataset used to test the proposed method is the University of California, San Diego (UCSD) pedestrian dataset [29]. It contains video of pedestrians on UCSD walkways, taken from a stationary camera at the resolution of  $238 \times 158$ .



FIGURE 6.1: A frame of the University of California, San Diego pedestrian dataset

As the video was originally split into a set of .png files, it had to be compiled to a video file using ffmpeg [43], as in real scenarios the method would analyze a real-time stream or a video file.

This dataset is suitable for testing purposes since it is long enough (54 minutes) and contains different crowd density levels. We used 8 minutes of video and segmented it

into 6 clips, the first clip was used solely for training and is 3 minutes long. The remaining 5 minutes are segmented into 5 clips each of 1 minute duration. The clip segmentation is depicted in Figure 6.2.

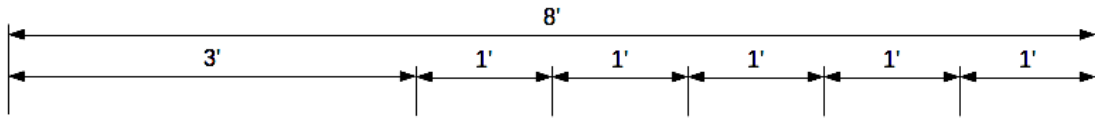


FIGURE 6.2: The first 3 minutes of the UCSD dataset were used solely for training. The testing was performed only using the following 5 minutes, segmented into 1 minute long clips.

The UCSD dataset does not include ground truth information. Therefore, we used our best judgment to provide the ground truth data. However, the numbers might be “0.5” person from the true values due to the fact that some people are crossing the tripwire at the beginning or the end of each clip. The provided ground truth appears in Table 6.1 and this is the ground truth we used in this study.

	3'-4'	4'-5'	5'-6'	6'-7'	7'-8'
Ground Truth	46	46.5	46	34.5	25

TABLE 6.1: Provided ground truth of the UCSD dataset for the testing segments of Figure 6.2.

### 6.1.2 Parameters

The parameters of the method were manually selected, as they heavily depend on the video being analyzed. For our dataset, we chose the number of levels of crowdedness and the background subtraction parameters empirically, as the operator of the surveillance system would do.

The selected number of crowdedness levels was  $L = 3$ . In [4], it was noted that a larger number of levels of crowdedness gives better results. However, in this dataset, 3 distinct levels of crowdedness could be observed.

During training and testing, moving object detection was done using the background subtraction method explained in 3.2.2. The learning rate, that represents how fast the background model is updated, was set to 0.01 during both stages. The threshold, that represents the distance to decide whether a pixel is well described by the background model, was set to 400 also for both training and testing.

The o-crowd consisted on two expert members that jointly solved the crowdsourcing tasks.

## 6.2 Experiments

Several tests were conducted to examine various aspects of our proposed method. We measured the error rate of the final crowd flow estimation result, defined as the absolute difference between the estimated value and the ground truth, as a percentage of the ground truth value:

$$ER = \frac{|v_{estimation} - v_{GT}|}{v_{GT}} \quad (6.1)$$

We also measured the utilization of the o-crowd, this is, the number of crowdsourcing tasks emitted.

The first is to evaluate the performance of the automatic method with no use of crowdsourcing ( $\alpha = 1$ ). The aim is to introduce contrast changes and compression attacks to the video and evaluate the drop in performance. The second experiment considers crowdsourcing as a solution to quality degradation. We compare the performance and evaluate the utilization of the crowd for two values of  $\alpha$ .

### 6.2.1 First experiment: no crowdsourcing

Figure 6.3 shows the increase in the error rate when the video is considerably degraded. The degradation consisted of H.264 compression using ffmpeg. The Constant Rate Factor (CRF) was set to 33. Also, contrast was boosted to 1.68 using eq2 filter. As a result, the video size was reduced to a 40%.

The average error rate increases from 9.2% to 34.4% when distortions are introduced. The average error rate is the average of the error rates in all the 5 video segments used for testing. We can conclude that degradation of the video quality can significantly impact the automatic analysis.

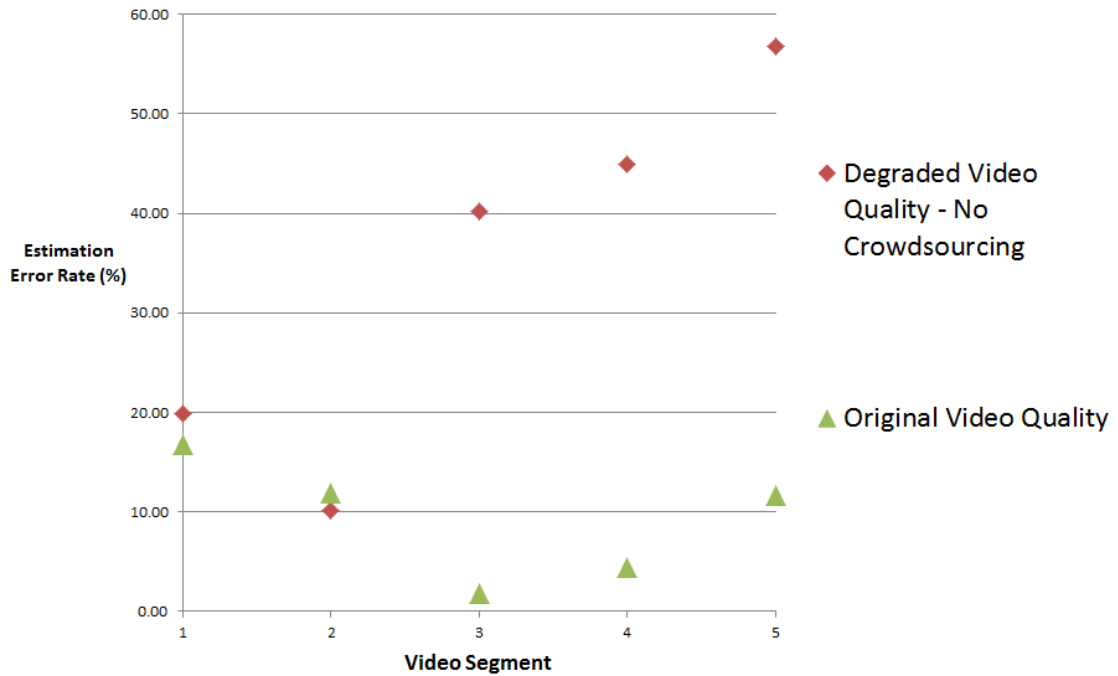


FIGURE 6.3: Results of first experiment: Average error rate increases from 9.2% to 34.4% when distortions are introduced

### 6.2.2 Second experiment: crowdsourcing incorporation

In this experiment, we incorporate crowdsourcing to the automatic method to increase its performance when the video is degraded. We check the average error rate and track the evolution of the utilization of the o-crowd throughout the video segments. The values of  $\alpha = 0.5, 0.6$  and 1 were used. Table 6.2 shows the reduction of the average error rate when crowdsourcing is incorporated.

	Average error rate
$\alpha$ (alpha) = 1 (no crowdsourcing)	34.4%
$\alpha$ (alpha) = 0.6	14.3%
$\alpha$ (alpha) = 0.5	18.0%

TABLE 6.2: Results of second experiment: Average error rate appreciably decreases when crowdsourcing is incorporated.

The average error rate decreases appreciably when crowdsourcing is incorporated. We can conclude that the proposed method manages to identify correctly the uncertainty and crowdsourcing effectively enhances the performance of the automatic method.

Furthermore, when using crowdsourcing, we would like the method to learn from the o-crowd input, so it would be desirable that the utilization of the o-crowd decrease as we progress in time. In other words, the number of crowdsourcing tasks, when the o-crowd is asked to assist the method, should decrease throughout the video segments. Figure 6.4 displays the number of emitted crowdsourcing tasks per video segment.



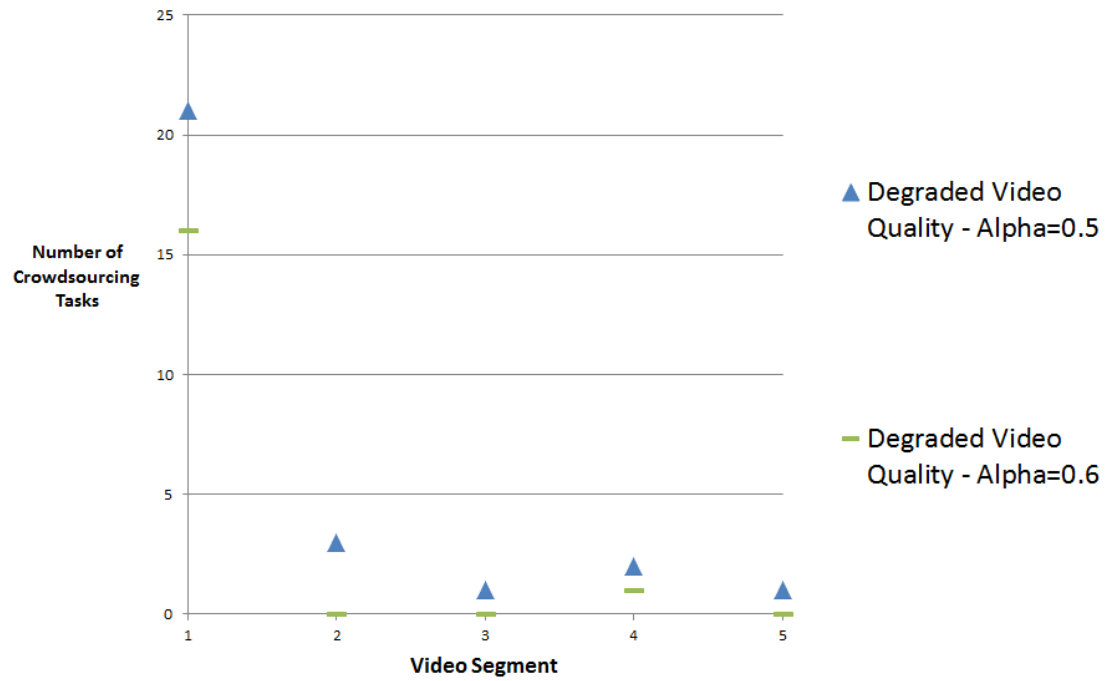


FIGURE 6.4: Results of second experiment: Utilization of the o-crowd

Figure 6.4 clearly depicts that our proposed method not only reduces the error rates, but also trains the classifier in an effective way, such that the number of crowdsourcing tasks decreases as we progress in time. Also, as expected, it shows that for lower value of  $\alpha$  the o-crowd is engaged more often.

## Chapter 7

# Conclusions

In this work, we presented enhancements to a crowd flow estimation method previously developed in the VIPER laboratory. Experimental evaluation on a publicly available dataset demonstrates that our proposed method identifies uncertainties and uses crowdsourcing in an effective way. The automatic method makes use of new information provided by the o-crowd and incorporates it into the model, hence achieving active learning. The final result is that the crowd flow estimation error rate is significantly reduced and the rate at which the o-crowd is engaged decreases in time.

Also, a web platform was developed to be able to control the whole system. It can be used by the operator to tune, start and oversight the crowd flow estimation of the video feeds. It also lets the o-crowd members perform their crowdsourcing tasks, so crowdsourcing can be incorporated into the automatic method. This platform is crucial for the scalability of the proposed method and it can be integrated into larger video surveillance systems.

As for future work, we will propose new characterizations of the uncertainty, for example by using Support Vector Machines (SVMs) [44], a technique widely used in machine learning. Also, a future extension of this work may include building a pyramid model to differentiate the accuracy of various o-crowd members. For example, assuming that the “wisdom of the crowd” leads to an accurate result in average, different roles and trustworthiness can be assigned to each o-crowd member, using as a reference the average of the answers of the whole o-crowd. Finally, testing a wider range of combination of parameters with a larger o-crowd size may also increase the understanding of the method and make the results more accurate.

# Bibliography

- [1] M. Valera and S. Velastin. Intelligent distributed surveillance systems: a review. *IEE Proceedings of Vision, Image and Signal Processing*, 152(2):192–204, April 2005. doi: [10.1049/ip-vis:20041147].
- [2] H. M. Dee and S. A. Velastin. How close are we to solving the problem of automated visual surveillance? *Machine Vision and Applications*, 19(5-6):329–343, 2008. ISSN 0932-8092. doi: [10.1007/s00138-007-0077-z].
- [3] Weiming Hu, Tieniu Tan, Liang Wang, and S. Maybank. A survey on visual surveillance of object motion and behaviors. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 34(3):334–352, Aug 2004. ISSN 1094-6977. doi: [10.1109/TSMCC.2004.829274].
- [4] S. Srivastava, K. K. Ng, and E. J. Delp. Crowd flow estimation using multiple visual features for scenes with changing crowd densities. *Proceedings of the 8th IEEE International Conference on Advanced Video and Signal-Based Surveillance*, pages 60–65, August–September 2011. doi: [10.1109/AVSS.2011.6027295]. Klagenfurt, Austria.
- [5] N. J. Gadgil, K. Tahboub, D. Kirsh, and E. J. Delp. A web-based video annotation system for crowdsourcing surveillance videos. *Proceedings of the SPIE/IS&T Electronic Imaging, Imaging and Multimedia Analytics in a Web and Mobile World*, 9027:90270A–1–12, March 2014. doi: [10.1117/12.2042440].
- [6] B. Zhan, D. Monekosso, S. Velastin P. Remagnino, and L. Xu. Crowd analysis: A survey. *Machine Vision and Applications*, 19(5-6):345–357, October 2008. doi: [10.1007/s00138-008-0132-4].
- [7] D. Kong, D. Gray, and H. Tao. Counting pedestrians in crowds using viewpoint invariant training. *Proceedings of the British Machine Vision Conference*, September 2005. Oxford, UK.
- [8] A. B. Chan, Z. J. Liang, and N. Vasconcelos. Privacy preserving crowd monitoring: Counting people without people models or tracking. *Proceedings of the IEEE*

- Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–7, June 2008. doi: [10.1109/CVPR.2008.4587569]. Anchorage, AK.
- [9] Zhongjie Yu, Chen Gong, Jie Yang, and Li Bai. Pedestrian counting based on spatial and temporal analysis. *Proceedings of IEEE International Conference on Image Processing (ICIP)*, October 2014. Paris, France.
- [10] J. H. Yin, S. A. Velastin, and A. C. Davies. Image processing techniques for crowd density estimation using a reference image. *Proceedings of the Asian Conference on Computer Vision (ACCV)*, pages 489–498, December 1995. doi: [10.1007/3-540-60793-5\\_102]. Singapore.
- [11] A. N. Marana, S. A. Velastin, L. F. Costa, and R. A. Lotufo. Automatic estimation of crowd density using texture. *Safety Science*, 28(3):165–175, April 1998. doi: [10.1016/S0925-7535(97)00081-7].
- [12] A. N. Marana and S. A. Velastin and L. F. Costa and R. A. Lotufo. Estimation of crowd density using image processing. *Proceedings of the IEE Colloquium on Image Processing for Security Applications*, pages 11/1–11/8, March 1997. doi: [10.1049/ic:19970387]. London, United Kingdom.
- [13] J. Howe. The rise of crowdsourcing. *Wired magazine*, 14(6):1–4, 2006. Dorsey Press.
- [14] J. Howe. Crowdsourcing: A Definition. June 2006. URL: [http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing\\_a.html](http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html).
- [15] J. Surowiecki. *The Wisdom of Crowds*. Knopf Doubleday Publishing Group, 2005. ISBN 9780307275059.
- [16] Enrique Estellés-Arolas and Fernando González-Ladrón-De-Guevara. Towards an integrated crowdsourcing definition. *J. Inf. Sci.*, 38(2):189–200, April 2012. ISSN 0165-5515. doi: [10.1177/0165551512437638].
- [17] Anhai Doan, Raghu Ramakrishnan, and Alon Y Halevy. Crowdsourcing systems on the world-wide web. *Communications of the ACM*, 54(4):86–96, 2011.
- [18] Amazon Mechanical turk. URL: <https://www.mturk.com>.
- [19] Freelancer. URL: <https://www.freelancer.com>.
- [20] Mob4hire. URL: <http://www.mob4hire.com>.
- [21] uTest. URL: <https://www.utest.com>.
- [22] TopCoder. URL: <https://www.topcoder.com>.

- 
- [23] ClowdCrowd. URL: <http://www.cloudcrowd.com>.
- [24] CrowdFlower. URL: <http://www.crowdflower.com>.
- [25] D. Brabham. *Crowdsourcing*. The MIT Press, 2013. ISBN 9780262518475.
- [26] A. Sorokin, D. Berenson, S. Srinivasa, and M. Hebert. People helping robots helping people: Crowdsourcing for grasping novel object. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2117–2122, October 2010. doi: [10.1109/IROS.2010.5650464]. Taipei, Taiwan.
- [27] S. Vijayanarasimhan and K. Grauman. Large-scale live active learning: Training object detectors with crawled data and crowds. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1449–1456, June 2011. doi: [10.1109/CVPR.2011.5995430]. Providence, RI.
- [28] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003. ISBN 9780521540513.
- [29] A. B. Chan and N. Vasconcelos. Modeling, clustering, and segmenting video with mixtures of dynamic textures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):909–926, May 2008. doi: [10.1109/TPAMI.2007.70738].
- [30] C. Stauffer and W.E.L. Grimson. Adaptive background mixture models for real-time tracking. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2, June 1999. ISSN 1063-6919. doi: [10.1109/CVPR.1999.784637]. Fort Collins, CO.
- [31] P. KaewTraKulPong and R. Bowden. An improved adaptive background mixture model for real-time tracking with shadow detection. pages 135–144, 2002. doi: [10.1007/978-1-4615-0913-4\_11].
- [32] Z. Zivkovic. Improved adaptive gaussian mixture model for background subtraction. 2:28–31 Vol.2, August 2004. ISSN 1051-4651. doi: [10.1109/ICPR.2004.1333992].
- [33] Z. Zivkovic and F. van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters*, 27(7): 773–780, May 2006. doi: [10.1016/j.patrec.2005.11.005].
- [34] T. Bouwmans, F. El Baf, and B. Vachon. Background modeling using mixture of gaussians for foreground detection - a survey. pages 219–237, 2008.
- [35] OpenCV. URL: <http://www.opencv.org>.
- [36] R. M. Haralick. Statistical and structural approaches to texture. *Proceedings of the IEEE*, 67(5):786–804, May 1979. doi: [10.1109/PROC.1979.11328].

- 
- [37] scikit-image. URL: <http://scikit-image.org>.
  - [38] NumPy. URL: <http://www.numpy.org>.
  - [39] SQLAlchemy. URL: <http://www.sqlalchemy.org>.
  - [40] AngularJS. URL: <https://angularjs.org>.
  - [41] RequireJS. URL: <http://www.requirejs.org>.
  - [42] Bootstrap. URL: <http://getbootstrap.com>.
  - [43] FFmpeg. URL: <https://www.ffmpeg.org>.
  - [44] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20(3):273–297, 1995. doi: [10.1007/BF00994018].