Winter 12-1-2011

Detection-assisted Object Tracking by Mobile Cameras

Li He
University of Nebraska – Lincoln, ritsu1228@gmail.com

DETECTION-ASSISTED OBJECT TRACKING BY MOBILE CAMERAS

by

Li He

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Electrical Engineering

Under the Supervision of
Professors Senem Velipasalar and
Mustafa Cenk Gursoy

Lincoln, Nebraska

December, 2011

DETECTION-ASSISTED OBJECT TRACKING BY MOBILE CAMERAS

Li He, M. S.

University of Nebraska, 2011

Adviser: Senem Velipasalar and Mustafa Cenk Gursoy

Tracking-by-detection is a class of new tracking approaches that utilizes recent development of object detection algorithms. This type of approach performs object detection for each frame and uses data association algorithms to associate new observations to existing targets. Inspired by the core idea of the tracking-by-detection framework, we propose a new framework called detection-assisted tracking where object detection algorithm provides help to the tracking algorithm when such help is necessary; thus object detection, a very time consuming task, is performed only when needed. The proposed framework is also able to handle complicated scenarios where cameras are allowed to move, and occlusion or multiple similar objects exist.

We also port the core component of the proposed framework, the detector, onto embedded smart cameras. Contrary to traditional scenarios where the smart cameras are assumed to be static, we allow the smart cameras to move around in the scene. Our approach employs histogram of oriented gradients (HOG) object detector for foreground detection, to enable more robust detection on mobile platform. Traditional background subtraction methods are not suitable for mobile platforms where the background changes constantly.

ACKNOWLEDGMENTS

First, I would like to thank my advisers, Professors Senem Velipasalar and Mustafa Cenk Gursoy for their guidance throughout my graduate study. I would like to thank them for always encouraging me to work out my own solutions on challenging problems and discussing with me. I highly appreciate the environment that they foster in the group, which is very healthy and helpful for creative research. I also would like to thank Professor Wenbo He for serving on my committee. I have learned a lot from both her class and discussions with her.

Throughout my graduate program, I have been surrounded by an amazing group of colleagues at UNL. My countless conversations with them have been invaluable to my research and to learning about other fields. I especially thank the fellow students from both Smart Vision Systems Laboratory and Wireless Communications and Networking Laboratory; they are Zhe Zhang, Qing Chen, Deli Qiao, Junwei Zhang and Bo Liang. The discussions, chats and laughters with them made my life in Nebraska much more fun. I would like to thank Youlu Wang and Mauricio Casares. They have helped me get started with the camera projects; I also learned a lot from the discussions with them. I also would like to thank my friends — Tiantian Xu, Tongqing Liu, Jinya Pu, Peggy Liu and Yao Wu — you have brought a lot fun to my life; it is really good to meet you here.

Lastly, and most importantly, I would like to thank my parents Zhaoxiang He and Nianyu Ye for always being loving, caring and encouraging. Even in the most difficult times, they never lost their confidence and belief in me. I can always rely on their advices and support. Without them, I could not reach this far.

Contents

Contents **iv**

List of Figures **vii**

List of Tables **ix**

1 Introduction **1**

1.1 Visual Tracking 2

1.1.1 Applications 2

1.1.2 Challenges 3

1.1.3 Visual Tracking on Embedded Smart Cameras 5

1.2 Related Work 5

1.3 Overview of the Thesis 8

2 Probabilistic Visual Tracking and Object Detection **10**

2.1 Overview of Bayesian Filtering 11

2.1.1 Formulation of Bayesian Tracking 12

2.2 Monte Carlo Sampling 14

2.2.1 Problems to Solve 14

2.2.2 Basics of Sampling Methods 15

2.2.3 Importance Sampling 16
2.2.4 Sampling-Importance Resampling 17
2.3 Particle Filters 18
2.3.1 Sequential Importance Sampling (SIS) Filter 19
2.3.2 Sampling-importance Resampling (SIR) Filter 21
2.4 Histogram of Oriented Gradients 22
2.4.1 Rectangular HOG Descriptor 23
2.4.2 Overall Processing Flow 24
2.4.3 Feature Extraction 24
2.4.4 The Classifier 26
2.4.5 Multi-scale Localization 26
2.5 Conclusion 27

3 Detection-assisted Tracking by Mobile Cameras 28
3.1 System Architecture 28
3.2 Detector 29
3.3 The Tracker 30
3.3.1 The Observation Model 32
3.3.2 The Dynamic Model 35
3.3.3 Correcting Trackers 36
3.3.4 The Timing of Correction 37
3.4 Implementation 38
3.4.1 PC Platform 38
3.4.1.1 Implementation Overview 38
3.4.1.2 TrackerManager and Trackers 40
3.4.2 CITRIC Platform 43

3.5 Conclusion 44

4 Experiments and Evaluation on PC Platform 45

4.1 Overview 45

4.1.1 PC Platform 46

4.2 Evaluation of PC Implementation 46

4.2.1 Tracking without Correction 46

4.2.2 Tracking with Detector 47

4.2.3 Handling Occlusion 48

4.2.4 Handling Similar Objects 49

4.2.5 Handling Crowded Scenes 50

4.3 Conclusion 52

5 Experiments and Evaluation on CITRIC Platform 53

5.1 Overview 53

5.1.1 CITRIC Platform 54

5.2 Evaluation of Embedded Implementation 55

5.2.1 Running on Whole Frame 56

5.2.2 Accelerating by Reducing Frame Size 57

5.2.3 Moving the CITRIC Camera 58

5.2.4 Adaptive Frame Cropping 59

5.3 Conclusion 65

6 Conclusion and Future Work 66

Bibliography 68

List of Figures

2.1 State-space model represented using a graphical model 12
2.2 The overall processing flow of HOG detection algorithm 25

3.1 Overview of the system structure 29
3.2 The divided histogram 33
3.3 The procedure of correcting trackers 36
3.4 Overview of implementation on PC platform 40
3.5 The relationship of TrackerManager, Tracker and ParticleFilter 42

4.1 The performance of the tracker with increasing number of resampling . . 47
4.2 The performance of the tracker with increasing number of resampling on another video sequence 48
4.3 The performance of the track with the assistance of the detector 49
4.4 A scenario where occlusion occurs 50
4.5 A scenario where a similar target exists 50
4.6 A scenario where both the target and the camera are moving forward and multiple similar objects exist 51

5.1 The CITRIC camera mounted on a toy car 54
5.2 The CITRIC camera board 55

5.3 The camera is held by person and is moving around 57
5.4 The camera is fixed and the frames are cropped 58
5.5 The camera is mount on the top of a toy car 60
5.6 The camera is mounted on a toy car and is moving along a corridor . . . 61
5.7 The boundary of a detected object and flow for automatic cropping . . . 63
5.8 Automatic frame cropping is performed 64

List of Tables

4.1 Summary of testing video sequence 46

5.1 Summary of performance on CITRIC platform 64

List of Algorithms

1 The SIS particle filter 21

2 The resampling algorithm 22

3 The SIR particle filter 23

Chapter 1

Introduction

Computer vision is the study of letting computers see, understand and further interact with the world in the way we do. In order to understand and interact with the world, computers must be able to locate objects of interest and continuously follow these objects. This is the task of visual tracking. Visual tracking is also the heart of many computer vision applications such as surveillance, robotic and monitoring applications.

The increasing number of video cameras in our daily lives becomes a more profound driving force of visual tracking. These cameras generate a huge amount of visual data for human analysis and understanding. Although it is possible to have humans involved in analyzing and understanding these visual data, it is more preferable that these tasks can be done in an automatic way. This requirement is even more profound for robotic applications: robots are expected to be self-contained and behave on their own with no or little human involvement.

Since video data is usually composed of background and a few moving objects of interest, in order to automatically analyze and understand video data, it is important to know which parts of a frame correspond to the objects of interest and where the

objects are. The task of determining which parts of a frame correspond to the objects of interest is solved by object detection and the task of following an object at each frame is solved by visual tracking.

1.1 Visual Tracking

Visual tracking is to consistently identify objects throughout a video sequence. A tracking algorithm identifies the objects of interest in each frame and consistently associates the identities of these object over time. It is important that the identification is consistent, that is the same object in different frames must be given the same identification. Any inconsistency causes the algorithm to fail.

In this section we will discuss several interesting applications of visual tracking; these applications show the importance of visual tracking. Then we will discuss some major challenges when we want to perform robust visual tracking.

1.1.1 Applications

To see the importance of visual tracking, we start by discussing some of its applications.

Video Surveillance This is the direct application of visual tracking and is still one of the most important applications [1] [2] [3]. Human tracking is mainly used for analyzing and monitoring human behaviors. For example, we may be interested in where a specific person has been. Vehicle tracking is mainly used for traffic monitoring. For example, transportation department may be interested in installing cameras at intersections and using these cameras to collect traffic statistics at different moments.

Mobile Robot Navigation This is another important area of for visual tracking [4] [5]. A mobile robot is expected to have the ability to acquire and respond properly to moving or static targets or obstacles. To perform this task, a mobile robot should be able to detect and track such targets or obstacles. By tracking predefined landmarks a mobile robot can know where it is and how it can reach certain target locations.

Human Computer Interface By tracking human and human actions (head, shoulder, whole body actions, eye gaze or facial expressions) and further understanding these actions, it is possible to control computers by these actions [6] [7]. This is a new type of human computer interface; it allows more natural interaction with computers in certain scenario. One real world example of this type of human computer interface is Microsoft Kinect.

1.1.2 Challenges

Consistent labeling is the heart of visual tracking, meaning that the same object in different frames must be given the same identification. The whole task loses its meaning if we are unable to perform consistent labeling.

Visual tracking has received a great deal of attention due to its importance. Yet from the perspective of computer vision, visual tracking is quite a challenging task. A good tracking algorithm has to be both robust and efficient. When performing visual tracking, we may encounter several challenges.

Abrupt Motion It is common to assume that the motion of a specific target is continuous, i.e. the difference between consecutive frames is small, when developing a tracking algorithm. Whenever the motion of the target shows discontinuity, the assumption may not hold and the algorithms fail. However, discontinuity of motion

is common in real world. One cause for the discontinuity is low frame rate, which may be caused by low processing power.

Cluttered Background When the background contains objects that have similar appearance as the target, confusion occurs and the tracking results degenerate. Therefore it is important to develop a measurement model that can discriminate between to different targets in order to reduce the impact of cluttered background.

Changing Illumination Many tracking algorithms are based on background subtraction assuming that the background always remain the same throughout the whole process. Changes of illumination may break this assumption. Such algorithms usually fail when there are abrupt changes of illumination. To handle this, it is necessary to develop a robust background model that can adapt to illumination changes.

Moving Cameras Moving camera settings are quite useful in mobile applications. In such scenarios, the cameras are moving instead of being static. Since most tracking algorithms assume a static background model, these algorithms fail when the background is no longer static. To handle moving camera settings, new algorithms and methods must be developed.

Occlusions It is possible that a specific target is occluded by other objects or it moves out of the field of view (FOV) of the camera in a complex scene. Occlusions usually cause loss of information preventing the tracking algorithm from obtaining necessary information to track the target, leading to tracking failures. Therefore, it is important to correctly handle occlusions in order to achieve robust tracking. The key to solving this problem is figuring out how to track the location of the target and keep the identity when the target is occluded.

1.1.3 Visual Tracking on Embedded Smart Cameras

Embedded smart cameras are self-contained, standalone vision systems with built-in vision sensors and processors. The discriminative feature of smart cameras is that they not only captures visual data but also perform higher-level visual processing locally. Smart cameras are usually equipped with communication interfaces for exchanging information with other smart cameras. They are becoming more popular with the advances in VLSI technology and embedded system architecture.

Multiple smart cameras can form a distributed camera network. A distributed camera network has many advantages over traditional centralized camera systems. For example, distributed camera network may be formed ad-hoc; it is convenient to add or remove camera nodes from the network. Visual tracking can also be performed on smart cameras. Smart cameras enable many applications in the areas of surveillance, traffic control, health care, home assistance, environmental monitoring, wildlife monitoring and industrial process control[8]. As discussed in 1.1.1, many of these applications require visual tracking. However, due to the limited resources of these embedded platforms, implementing visual tracking on smart cameras can be more challenging.

1.2 Related Work

From the above discussion, we see that robust visual tracking is an interesting, important and yet challenging topic in computer vision. The inherent difficulty of this task roots from the variability of object, background appearances as well as changes of environment illumination. A mainstream of methods dealing with object tracking is based on background modeling and sometimes calibrated cameras [9], [10], [11]. In these methods, a background model is built based upon the fact that the background

is static and remains the same throughout the entire tracking process. This type of methods can be extremely lightweight and fast [12]. However, their performance mainly rely on the underlying model and the assumption of static cameras. If the background model is not properly developed and updated, the performance is vulnerable to shadows and illumination changes. On the other hand, calibrated cameras with known internal and external parameters are also used to facilitate tracking using geometric relations. But when the system contains a relatively large number of cameras, e.g. 5, the calibration process becomes quite painstaking. Yet, the use of calibrated camera also assumes static cameras.

To cope with the difficulties mentioned above, a new class of tracking methods, called *tracking-by-detection*, has been introduced. These methods utilize object detection algorithms rather than background subtraction. The main idea of these approaches is to employ an object detector in the place of background subtraction. Since most object detection algorithms only require a single image to work, they are much more robust to the changing background and illumination compared to traditional background subtraction algorithms. In a typical tracking-by-detection framework, the detection algorithm is invoked at each frame and then the detection results are associated to existing targets in the system.

The tracking-by-detection framework is driven by the recent development in object detection and has been employed in recent works [13], [14], [15], [16], [17]. However, most of them is based on associating detection responses with trajectories using similarities of appearance, position and size. For example, Huang et al. [14] employed an approach that is entirely based on data association; Li et al. [17] improved the method used in [14] by integrating learning algorithms and the target of learning is an affinity model. Zhang et al. [15] also used a data association based algorithm with the exception that network flow was employed to perform the optimization.

Visual tracking on embedded smart cameras is another interesting yet challenging research area. On embedded platforms, we no longer have intensive processing power and large memory as we do in desktop or workstation computing environments. Limited resource is one of the main constraints for embedded systems. Wang et al. [10], used a lightweight background subtraction algorithm described in [12], to detect the foreground objects. The lightweight algorithm employs a temporal difference method until a complete background model is built. The algorithm is designed for embedded smart cameras, and can run fast on an embedded system. However, the algorithm designed for static cameras, generates false positives when there are shadows.

The HOG descriptor for human detection is proposed in by Dalal et al. [18] and is widely used for human detection. The basic idea of HOG is that local visual features can be characterized well by the distribution of local intensity gradients or edge directions. An image is divided into cells and 1-D histograms of gradient direction are calculated. The calculated histograms are then normalized over cells to get the HOG. A support vector machine (SVM) is then trained to classify human and non-human regions.

The HOG detection algorithm has also been widely used for foreground detection. Although detection algorithms are more suitable for moving cameras and are more robust towards background changes, they have not yet been used on embedded smart cameras. In [19], we have introduced some initial work towards applying HOG detection algorithm on the CITRIC embedded smart camera platform and reported the performance. In this thesis, we continue this work introduced and try to make the implementation more applicable. By applying HOG detector to embedded smart cameras we can build robust tracking algorithm for real-world mobile applications. The experimental results and performance also reveal certain limitations of such embedded platforms.

1.3 Overview of the Thesis

By carefully studying the *tracking-by-detection* framework, we notice that the detection is performed for every frame. However, the continuous application of the detection algorithm and data association is not necessary on a frame basis. In *tracking-by-detection* framework, detection algorithms have the same role as background subtraction algorithms in the traditional tracking framework. New observations are not necessary at each frame. For example, Bayesian tracking algorithms, such as particle filter, can predict the location of the target with quite high accuracy. New observations are only needed at the initialization stage or when the tracking algorithm is unacceptably degenerated. In this thesis, we propose a new framework called *detection-assisted-tracking*. In this framework, we combine particle filter and object detection algorithm in a different way. Particle filter still has the main role while the detection algorithm is only meant to assist the tracking algorithm when it needs help. This is in contrast to the tracking-by-detection framework, where the detection algorithm has a major role.

In Chapter 2, we discuss tracking algorithms and object detection algorithms in detail. Tracking and object detection algorithms are the heart of the proposed *detection-assisted-tracking* framework. In this thesis, we will use particle filter for tracking and histogram of oriented gradient (HOG) detector for object detection. These two topics are also covered in detail in this chapter.

In Chapter 3, we describe the proposed framework and its implementation in great detail. We will discuss how the particle filter algorithm and HOG detector are connected and several design consideration. From the software engineering perspective, the architecture of a software system is also very important. Thus, we also discuss the software architecture of the implementation and present how it performs.

In Chapter 4, we use several real world video sequence to test our proposed framework and see how well it works. We finally conclude the thesis in Chapter 6 and discuss future work.

Chapter 2

Probabilistic Visual Tracking and Object Detection

The proposed framework consists of two major parts: a visual tracking algorithm and an object detection algorithm. There are two categories of methods that solve the visual tracking problem: deterministic and probabilistic. Deterministic approaches are usually formulated as an optimization problem. However, deterministic approaches are usually vulnerable to variable settings, such as illumination changes, target occlusion and merge. On the other hand, probabilistic approaches tackle the tracking problem from a statistical perspective: they try to find the optimum solution in the statistical sense. These approaches are usually more reliable than the deterministic ones.

An object detection algorithm is the other major part of the framework. If we consider an image as a set of features, then the detection algorithm extracts these features. Extracting the feature set usually involves intensity patterns, texture details and shape information [18]. For the detection algorithm, both generative and discriminative approaches can be used [20] [21] [22]. Generative approaches usually

build a Bayesian model and solve an optimization problem on this model; discriminative approaches use deterministic machine learning methods to determine whether a feature set belongs to an object.

In this chapter we will discuss particle filter and histogram of oriented gradient (HOG) detection algorithm in detail.

2.1 Overview of Bayesian Filtering

Filtering is the process of extracting information about a quantity of interest at a certain moment using the data measured up to and including the moment. Bayesian filtering is based on the well-known Bayesian theorem, which describes the fundamental probability law governing the process of logical inference. Bayesian approach to statistical inference has become an important branch in statistics and is widely applied to statistical decision, detection and estimation, pattern recognition and machine learning [23] [24]. Bayesian theory was also applied to filtering framework. One of the earliest work on Bayesian estimation can be found in [25].

In Bayesian filtering framework, a system is described using state-space model. The state-space approach focuses on the *state vector*. The state vector is an aggregation of all the states that are necessary for describing the system. In visual tracking problems, the state vector may include, for example, the position and the velocity of the target. However, not all the system states are accessible: in tracking problems the system states are what we want to know. To solve a dynamic system using Bayesian filtering framework, two models are required: dynamic model and measurement model [26]. Dynamic model describes how the system evolves with time, and measurement model relates the noisy measurement to the states.

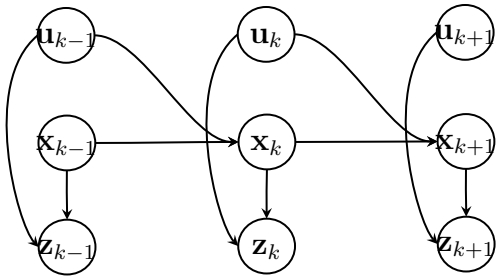


Figure 2.1: State-space model represented using a graphical model.

2.1.1 Formulation of Bayesian Tracking

To give a formal definition of Bayesian tracking, we need to first define the two models. We first define the *dynamic model* describing the evolution of system state with time.

$$\mathbf{x}_k = \mathbf{f_k}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{v}_{k-1}) \tag{2.1}$$

where $\mathbf{x}_{k-1}$ and $\mathbf{u}_{k-1}$ are the system state and system input at time instance $k-1$, respectively; $\mathbf{v}_{k-1}$ is an i.i.d. noise sequence; $\mathbf{f_k}$ is function of the system state $\mathbf{x}_{k-1}$ and noise process $\mathbf{v}_{k-1}$ and it may be a non-linear function [26].

Since in most cases we have no direct access to the real system states, a *measurement model* or *observation model* is needed to perform inference on the system states. The output of the measurement model is the observation that is directly accessible. The measurement model defines how the system states are reflected the observation.

$$\mathbf{z}_k = \mathbf{h}_k(\mathbf{x}_k, \mathbf{u}_k, \mathbf{n}_k) \tag{2.2}$$

where $\mathbf{z}_k$ represents the measurement of the system at time instance k, $\mathbf{h}_k$ may also be a non-linear function and $\mathbf{n}_k$ is an i.i.d. measurement noise. Fig. 2.1 shows a graphical model representation of state-space model of a dynamic system.

The essence of tracking problems is to estimate the current system state using

all the observations obtained up to that time. Given the initial probability $p(\mathbf{x}_0)$, transition density $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ and likelihood $p(\mathbf{z}_k|\mathbf{x}_k)$, the objective is to estimate the optimal current state $\mathbf{x}_k$ using observations up to time k, $\mathbf{z}_{0:k} = \{\mathbf{z}_i, i = 1, 2, ..., k\}$; this is equivalent to estimating the posterior density $p(\mathbf{x}_k|\mathbf{z}_{1:k})$.

The posterior density can be calculated recursively by using Bayesian rule. We make two assumptions when performing the recursive calculation:

1. The state transition is a one-order Markov process, that is $p(\mathbf{x}_k|\mathbf{x}_{0:k-1}) = p(\mathbf{x}_k|\mathbf{x}_{k-1})$;

2. The current observation is only determined by the current state, that is $p(\mathbf{z}_{1:k-1}|\mathbf{x}_k) = p(\mathbf{z}_{1:k-1})$.

Using Bayesian rule, we start from the posterior density:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_{1:k})p(\mathbf{x}_k)}{p(\mathbf{z}_{1:k})} \tag{2.3}$$

$$= \frac{p(\mathbf{y}_k, \mathbf{y}_{1:k-1})p(\mathbf{x}_k)}{p(\mathbf{y}_k, \mathbf{y}_{1:k-1})} \tag{2.4}$$

$$= \frac{p(\mathbf{z}_k|\mathbf{z}_{1:k-1}, \mathbf{x}_k)p(\mathbf{z}_{1:k-1}|\mathbf{x}_k)p(\mathbf{x}_k)}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})p(\mathbf{z}_{1:k-1})} \tag{2.5}$$

$$= \frac{p(\mathbf{z}_k|\mathbf{z}_{1:k-1}, \mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})p(\mathbf{y}_{1:k-1})p(\mathbf{x}_k)}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})p(\mathbf{z}_{1:k-1})p(\mathbf{x}_k)} \tag{2.6}$$

$$= \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})} \tag{2.7}$$

To be more clear, we give names to the terms in (2.7):

1. *Prior:* $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$ contains the prior information. It can be calculated using Chapman-Kolmogorov equation:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})d\mathbf{x}_{k-1} \tag{2.8}$$

2. *Likelihood:* The term $p(\mathbf{z}_k|\mathbf{x}_k)$ is the likelihood of observing the measurement $\mathbf{z}_k$ when the system is in state $\mathbf{x}_k$. It determines the measurement noise model.

3. *Evidence:* The normalizing constant $p(\mathbf{y}_n|\mathbf{y}_{1:k-1})$ is called evidence. It can be calculated as

$$p(\mathbf{z}_k|\mathbf{y}_{1:k-1}) = \int p(\mathbf{z}_k|\mathbf{x}_n)p(\mathbf{x}_n|\mathbf{z}_{1:k-1})d\mathbf{x}_k \tag{2.9}$$

The essence of Bayesian filtering or inference is to calculate the exact solution or approximate solution of the above three terms. Although we have the recursive expressions for them, it is generally difficult to obtain exact solutions. Exact solutions can be obtained only when the posterior density is Gaussian for every time step, which is known as *Kalman filtering* [27].

2.2 Monte Carlo Sampling

For most real world applications, the integrals in (2.8) and (2.9) are intractable. Thus, it is impossible to obtain exact solutions for the system. Instead, we have to use approximate algorithms to get approximate solutions.

Monte Carlo sampling denotes a class of the approximate algorithms. They concentrate on sampling from a given distribution and, using sampling and estimation methods to obtain approximate solutions to mathematical problems.

Monte Carlo sampling is the heart of particle filters and is discussed here. Examples of Monte Carlo calculation and optimization can be found in [28] [29].

2.2.1 Problems to Solve

Monte Carlo sampling concentrates on solving two fundamental problems [30]:

- Generate samples $\{\mathbf{x}^{(i)}\}_{i=1}^{N_p}$ from a given distribution $p(\mathbf{x})$, where $p(\mathbf{x})$ is called the target distribution.

- Estimate the expectation of a function $f(\mathbf{x})$ under this probability distribution $p(\mathbf{x})$:

$$E[f] = \int f(\mathbf{x})p(\mathbf{x})\, d\,\mathbf{x} \tag{2.10}$$

To solve the first problem, we use a uniform distribution as the base and perform transformation on it to obtain the target distribution. To solve the second problem, we try to obtain a set of samples $\mathbf{x}^{(i)}$ (where $i = 1, ..., N_p$) drawn independently from the target distribution $p(\mathbf{x})$. When the number of samples is large enough, the expectation in (2.10) can be approximated as

$$\hat{f} = \frac{1}{L}\sum_{i=1}^{N_p} f(\mathbf{x}^{(i)}) \tag{2.11}$$

We encounter difficulties when solving the second problem. First, the samples $\mathbf{x}^{(i)}$ may not be independent, which makes the effective sample size much smaller than the apparent sample size. Second, for many practical problems it is hard to directly sample from the target distribution $p(\mathbf{x})$. The later difficulty motivates the use of proposal distributions.

2.2.2 Basics of Sampling Methods

We start with a pseudo-number generator. The pseudo-number generator is assumed to generate numbers that are uniformly distributed over $(0, 1)$. Thus, we need to transform a uniform distribution to a given distribution that we are interested in.

Assume that x is uniformly distributed over $(0, 1)$ and y is a function of x such

that $y = f(x)$. From statistics we know that the distribution of y is given by

$$p(y) = p(x)\|\frac{d\,x}{d\,y}\| \tag{2.12}$$

According to the assumption that x is uniformly distributed, we have $p(x) = 1$ in the above equation. By integrating over y, we have

$$x = h(y) = \int_{-\infty}^{y} p(\hat{y})\,d\,\hat{y} \Longrightarrow y = h^{-1}(x) \tag{2.13}$$

By choosing a function according to (2.13), we get an arbitrary distribution from uniform distribution.

2.2.3 Importance Sampling

The goal is to sample a distribution in the region of importance, in order to achieve computational efficiency. This is important for high dimensional problems where the region of interest is relatively small compared to the whole data space.

The basic idea of importance sampling is to use a proposal distribution $q(\mathbf{x})$ to approximate the target distribution $p(\mathbf{x})$ [31] [32]. The support of the proposal distribution is assumed to cover that of the target distribution.

Assume, again, that the target distribution that we are interested in is $p(\mathbf{x})$ and the proposal distribution is $q(\mathbf{x})$. Then we can express the expectation in (2.10) using finite sum over samples $\mathbf{x}^{(i)}$ drawn from $q(\mathbf{x})$

$$E[f] = \int f(\mathbf{x})p(\mathbf{x})\,d\,\mathbf{x} = \int f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}q(\mathbf{x})\,d\,\mathbf{x} \simeq \frac{1}{N_p}\sum_{i=1}^{N_p}\frac{p(\mathbf{x}^{(i)})}{q(\mathbf{x}^{(i)})}f(\mathbf{x}^{(i)}) \tag{2.14}$$

By defining the *importance weight* as $W(\mathbf{x}^{(i)}) = \frac{p(\mathbf{x}^{(i)})}{q(\mathbf{x}^{(i)})}$, we can write the expecta-

tion as

$$E[f] = \frac{1}{N_p} \sum_{i=1}^{N_p} W(\mathbf{x}^{(i)}) f(\mathbf{x}^{(i)}) \tag{2.15}$$

We normalize the importance weight and have the following expression for the expectation

$$E[f] = \frac{1}{N_p} \sum_{i=1}^{N_p} \tilde{W}(\mathbf{x}^{(i)}) f(\mathbf{x}^{(i)}) \tag{2.16}$$

where $\tilde{W}$ is the *normalized importance weight* and is defined as

$$\tilde{W}(\mathbf{x}^{(i)}) = \frac{W(\mathbf{x}^{(i)})}{\sum_{i=1}^{N_p} W(\mathbf{x}^{(i)})} \tag{2.17}$$

The performance of importance sampling is crucially affected by the choice of the proposal distribution $q(\mathbf{x})$. If $p(\mathbf{x})f(\mathbf{x})$ is strongly varying and has a significant proportion of its mass concentrated over relatively small regions of $\mathbf{x}$ space, then a large portion of the importance weights $\{W(\mathbf{x}^{(i)})\}$ may be relatively insignificant, resulting a much smaller effective sample size compared to the sample size N_p. This is a major drawback of importance sampling. To alleviate this drawback, resampling must be applied according to certain rules.

2.2.4 Sampling-Importance Resampling

Sampling-importance-resampling is motivated by a collection of computationally intensive methods, called bootstrap, that are based on resampling from the observed data [33]. The goal of resampling is to eliminate the samples with small importance weights.

The generic sampling-importance-resampling procedure includes the following steps:

1. Draw L samples $\{\mathbf{x}^{(i)}\}_{i=1}^{N_p}$ from the proposal distribution $q(\mathbf{x})$;

2. Calculate the importance weight $W^{(i)}$ for each sample $\mathbf{x}^{(i)}$;

3. Normalize the importance weight to obtain $\tilde{W}^{(i)}$;

4. Resample with replacement N times from the sample set $\{\mathbf{x}^{(i)}\}_{i=1}^{N_p}$, where the probability of resampling from each sample $\mathbf{x}^i$ is proportional to $\tilde{W}^{(i)}$.

The resampling step can be performed after each importance sampling, or when necessary. To determine whether resampling is necessary, both dynamic and deterministic strategies can be employed. Under the deterministic framework, resampling step is taken out for every k importance sampling steps. Under dynamic framework, the variance of the sample weights is monitored. Resampling is performed only when the variance is above certain value. One commonly used method for determining whether resampling is necessary is calculating the effective sample size $\hat{N}_{eff}$, defined as

$$\hat{N}_{eff} = \frac{1}{\sum_{i=1}^{N_p}(\tilde{W}_n^{(i)})^2} \tag{2.18}$$

A threshold N_T is usually setup for $\hat{N}_{eff}$. If $\hat{N}_{eff} < N_T$, then resampling step is performed; otherwise no resampling step is performed. The advantage of dynamic resampling framework is that it preserves desired performance while controls computational complexity to certain extent.

2.3 Particle Filters

Particle filters are also known as sequential Monte Carlo estimation. They work as follows: The state space is partitioned into many parts and particles are filled in according to certain probability measure; higher probability means denser particles. The particle system evolves with time according to the dynamic model defined by an

evolving density function. Since the density function can be represented by point-mass histogram, we can obtain a set of particles representing the evolving density function by randomly sampling from the state space.

The core components in a particle filter are the sampling algorithm and the re-sampling algorithm. Resampling is performed to restore the particle filter from degeneration. Different types of particle filters can be obtained by combining different sampling methods and timing of resampling.

2.3.1 Sequential Importance Sampling (SIS) Filter

To utilize sampling methods, we use a tuple, $\{\mathbf{x}_{0:k}^{(i)}, w_k^{(i)}\}$, to characterize the posterior density $p(\mathbf{x}_{0:k}|\mathbf{z}_{1:k})$. $\{\mathbf{x}_{0:k}^{(i)}, i = 0, ..., N_p\}$ is a set of samples and $\{w_k^{(i)}, i = 0, ..., N_p\}$ is the associated normalized weights. The posterior density can be expressed as

$$p(\mathbf{x}_{0:k}|\mathbf{z}_{1:k}) \approx \sum_{i=1}^{N_s} w_k^{(i)} \delta(\mathbf{x}_{0:k} - \mathbf{x}_{0:k}^{(i)}) \tag{2.19}$$

The weights $\{w^{(i)}\}$ are then selected according to the principle of importance sampling: if it is hard to sampling from a given density $p(\mathbf{x})$ but it is easy to evaluation a proposed density $q(\mathbf{x})$, then we can use samples from $q(\mathbf{x})$ to approximate that from $p(\mathbf{x})$. If we sample $\{\mathbf{x}_{0:k}^{(i)}, i = 0, ..., N_s\}$ from the proposed density $q(\mathbf{x})$, then the weights in (2.19) can be written as

$$w_k^{(i)} \propto \frac{p(\mathbf{x}_{0:k}^{(i)}|\mathbf{z}_{1:k})}{q(\mathbf{x}_{0:k}^{(i)}|\mathbf{z}_{1:k})} \tag{2.20}$$

To calculate the importance weights in (2.20), we first assume that the proposed

density can be factorized in the following way

$$q(\mathbf{x}_{0:k}|\mathbf{z}_{1:k}) = q(\mathbf{x}_k|\mathbf{x}_{0:k-1}, \mathbf{z}_{1:k})q(\mathbf{x}_{0:k-1}|\mathbf{z}_{1:k-1}) \tag{2.21}$$

We can express $p(\mathbf{x}_{0:k}|\mathbf{z}_{1:k})$ in terms of $p(\mathbf{x}_{0:k-1}|\mathbf{z}_{1:k-1})$, $p(\mathbf{z}_k|\mathbf{x}_k)$ and $p(\mathbf{x}_k|\mathbf{x}_{k-1})$:

$$p(\mathbf{x}_{0:k}|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|(\mathbf{x}_k|\mathbf{x}_{k-1}))}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})} p(\mathbf{x}_{0:k-1}|\mathbf{z}_{1:k-1}) \tag{2.22}$$

$$\propto p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{0:k-1}|\mathbf{z}_{1:k-1}) \tag{2.23}$$

Then, we obtain the weight update equation:

$$w_k^{(i)} \propto \frac{p(\mathbf{z}_k|\mathbf{x}_k^{(i)})p(\mathbf{x}_k^{(i)}|\mathbf{x}_{k-1}^{(i)})p(\mathbf{x}_{0:k-1}^{(i)}|\mathbf{z}_{1:k-1})}{q(\mathbf{x}_{\mathbf{k}}^{(\mathbf{i})}|\mathbf{x}_{0:k-1}^{i}, \mathbf{z}_{1:k})q(\mathbf{x}_{0:k-1}^{(i)}|\mathbf{z}_{1:k-1})} \tag{2.24}$$

$$= w_{k-1}^{(i)} \frac{p(\mathbf{z}_k|\mathbf{x}_k^{(i)})p(\mathbf{x}_k^{(i)}|\mathbf{x}_{k-1}^{(i)})}{q(\mathbf{x}_k^i|\mathbf{x}_{0:k-1}^{(i)}, \mathbf{z}_{1:k})} \tag{2.25}$$

$$= w_{k-1}^{(i)} \frac{p(\mathbf{z}_k|\mathbf{x}_k^{(i)})p(\mathbf{x}_k^{(i)}|\mathbf{x}_{k-1}^{(i)})}{q(\mathbf{x}_k^i|\mathbf{x}_{k-1}^{(i)}, \mathbf{z}_k)} \tag{2.26}$$

where we assume that $q(\mathbf{x}_k|\mathbf{x}_{0:k-1}, \mathbf{z}_{1:k}) = q(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{z}_k)$ for the last line. This assumption is useful when only one filtered estimation of $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ is required for each time step. Then the posterior density can be expressed as

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) \approx \sum_{i=1}^{N_p} w_k^i \delta(\mathbf{x}_k - \mathbf{x}_k^{(i)}) \tag{2.27}$$

Using the above derivation, the SIS filtering algorithm can be written as the following pseudo-code. The algorithm recursively propagate the importance weight and the initial samples.

Input: Initial particles and weights $\{\mathbf{x}_{k-1}^{(i)}, w_{k-1}^{(i)}\}_{i=1}^{N_p}$ and observations $\mathbf{z}_k$
Output: Propagated particles and weights $\{\mathbf{x}_{k-1}^{(i)}, w_{k-1}^{(i)}\}_{i=1}^{N_p}$

```
begin
    foreach i = 1 : N_p do
        Draw x_k^(i) ~ q(x_k | x_{k-1}^(i), z_k)
        Calculate the importance weights w_k^(i) according to (2.26)
        Normalize the obtained weights
    end
end
```

Algorithm 1: The SIS particle filter

2.3.2 Sampling-importance Resampling (SIR) Filter

The SIS particle filter is simple to implement but has a common problem known as degeneracy. As time evolves, more and more particles will have negligible weights. This is described in [34] that the variance of the importance weights can only increase. To alleviate degeneracy, resampling should be performed. It is worth noting that although resampling can alleviate degeneracy, but cannot eliminate it. Meanwhile, resampling brings new problems. For example, since the particles with larger weights are statistically selected many times, the diversity of the particles decrease and the resulting resampled particles include many duplicated ones.

The basic idea of resampling is to eliminate the particles with small importance weight and replace them with the particles with larger importance weights. There exist many resampling methods such as stratified sampling, residual sampling [35] and systematic resampling [36]. Systematic resampling is the algorithm used in [37] and preferred by the authors of [26] and will be used in our proposed framework. Algorithm 2 shows the systematic resampling algorithm.

By combining the resampling algorithm and the SIS filtering algorithm, we obtain the generic SIR filtering algorithm, as shown in Algorithm 3.

Input: A set of particles with weights $\{\mathbf{x}_{k-1}^{(i)}, w_{k-1}^{(i)}\}_{i=1}^{N_p}$
Output: Resampled set of particles $\{\mathbf{x}^{*(i)}_{k-1}, w_{k-1}^{*(i)}\}_{i=1}^{N_p}$

begin
  Let $c_1 = 0$
  foreach $i = 2 : N_p$ **do**
    $c_i = c_{i-1} + w_k^{(i)}$
  end
  Set $i = 1$
  Draw $u_1 \sim \mathcal{U}(0, \frac{1}{N_p})$
  foreach $j = 1 : N_p$ **do**
    $u_j = u_1 + N_p^{-1}(j-1)$
    while $u_j > c_j$ **do**
      $i = i + 1$
    end
    $\mathbf{x}^{*(j)}_k = \mathbf{x}_k^{(i)}$
    $w_k^{*(j)} = \frac{1}{N_p}$
  end
end

Algorithm 2: The resampling algorithm

2.4 Histogram of Oriented Gradients

Histogram of Oriented Gradients (HOG) is an effective feature descriptor for object detection. It is first introduced by Dalal and Triggs [18]. The descriptor counts the occurrences of gradient orientation in localized portions of an image. HOG is similar to edge orientation histograms and scale-invariant feature transform descriptors (SIFT) [38]; it differs from these methods in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy. The HOG detection algorithm combines the HOG descriptor with machine learning algorithms to detect a certain class of objects in a given image.

Input: Particles from previous time step $\{\mathbf{x}_{k-1}^{(i)}, w_{k-1}^{(i)}\}_{i=1}^{N_p}$ and current observation $\mathbf{z}_k$
Output: Updated particles $\{\mathbf{x}_k^{(i)}, w_k^{(i)}\}_{i=1}^{N_p}$

begin
  foreach $i = 1 : N_p$ **do**
    Draw $\mathbf{x}_k^{(i)} \sim q(\mathbf{x}_k|\mathbf{x}_{k-1}^{(i)}, \mathbf{z}_k)$
    Calculate the importance weights $w_k^{(i)}$ according to (2.26)
    Normalize the obtained weights
  end
  $t = \sum_{i=1}^{N_p} w_k^{(i)}$
  foreach $i = 1 : N_p$ **do**
    $w_k^{(i)} = \frac{w_k^{(i)}}{t}$
  end
  Calculate $\hat{N_{eff}}$ using (2.18)
  if $\hat{N_{eff}} \geq N_T$ **then**
    Resample using Algorithm 2 to obtain a new set of particles $\{\mathbf{x}*_{k-1}^{(i)}, w_{k-1}^{*(i)}\}_{i=1}^{N_p}$
  end
end

Algorithm 3: The SIR particle filter

2.4.1 Rectangular HOG Descriptor

There are four types of HOG descriptors: rectangular HOG (R-HOG), circular HOG (C-HOG), bar HOG and center-surrounded HOG. R-HOG is the simplest HOG descriptor among the four and is used for object detection in the proposed framework; detailed discussion about the other three HOG descriptors can be found in [39].

HOG descriptors are based on evaluating a dense grid of well-normalized local histograms of image gradient orientations over the image windows. The idea of the method is that the distribution of local intensity gradient or edge directions can characterize local object appearance and shape quite well, even without precise knowledge of the corresponding gradient or edge positions [39]. To calculate HOG descriptors, an image is first divided into small spatial regions called *cells*. Larger spatial regions

called *block* are used to normalize the calculated histograms.

Overlapping square or rectangular grids of cells are used for calculating R-HOG. The descriptor blocks are calculated over dense uniformly sampled grids and are usually overlapped; each block is normalized independently. Using square R-HOG as an example, the orientation histogram is calculated on a grid of $\varsigma \times \varsigma$ of cells of $\eta \times \eta$ pixels, each of which contains β orientation bins [39].

2.4.2 Overall Processing Flow

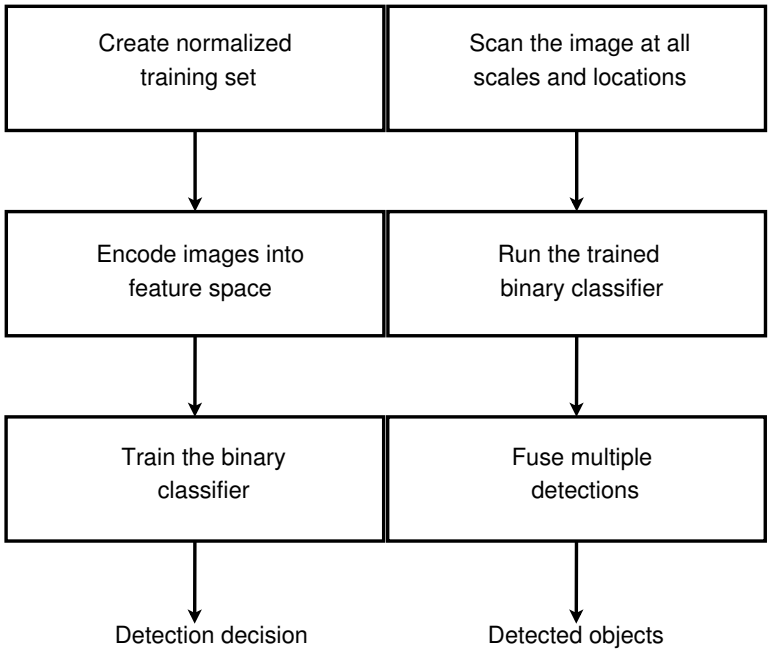
HOG detection algorithm consists of two phases: *learning phase* and *detection phase*. In the learning phase, the HOG features of known objects are used to train a binary classifier; in the detection phase, the trained classifier is used to perform a dense multi-scale scan for preliminary object decisions at each location of the test image and these preliminary decisions are then used to get the final detection decision. The processing flow of the two phases are shown in Fig. 2.2.

2.4.3 Feature Extraction

In order to train the binary classifier and perform detection, features must be extracted. The feature extraction process contains five major steps:

- *Normalization*: global image normalization/equalization is performed to reduce the influence of illumination effects. Gamma (power law) compression, either computing the square root or the log of each color channel, is used for this step [39].

- *Computing the gradient*: first order image gradients are computed in this step. The locally dominant color channel that provides color invariance is used to perform the computation. These gradients provide information about contour,



(a) Learning process (b) Detection process

Figure 2.2: The overall processing flow of HOG detection algorithm

silhouette and certain texture information; they also resist illumination variations.

- *Encoding*: The encoding is designed to be sensitive to local image content while it still provides resistance to small changes in pose or appearance. The image window is divided into smaller cells. For each cell accumulated local 1-D histogram of gradient or edge orientations over all the pixels in the cell is calculated. This cell-level 1-D histogram is the basic form of orientation histogram. The range of gradient angle is divided into a predefined number of bins. The gradient magnitudes of the pixels in a cell are used to vote into the orientation histogram.

- *Normalizing the histogram*: the gradient histogram calculated in the previous step is normalized for further processing. Normalization provides better invariance to illumination, shadowing, and edge contrast. The accumulated local histogram energy over a group of local cells, known as a block, are computed to normalize each cell in the block. There are overlaps among different blocks, that is a cell is shared by several blocks; but each block has a different normalizing constant. Therefore, a cell appears several times in the normalized histogram but with different normalizing constant. This normalized histogram is known as the histogram of oriented gradient (HOG).

2.4.4 The Classifier

Extracted HOG features are then used to train a classifier. A linear SVM is used in [39] and will be used in our proposed framework as well. The SVM can be trained using SVMLight [40]. With calculated SVM coefficients it becomes easy to perform classification on observed HOG features.

2.4.5 Multi-scale Localization

To detect and precisely localize any objects that appear in the image, the detector scans the image with a window at all positions and scales; the classifier is then executed in each window and multiple overlapping detections are fused to obtain the final detections. The following steps are used to perform the multi-scale localization:

- *Computing scales*: the scales are computed according to the normalized window size used for training the classifier and the image size.

- *Rescaling*: bilinear interpolation is used to rescale the image at each scale.

- *Classification*: features are extracted on each scaled image and a dense scan is performed; the detection results are stored in a list.
- *Non-maximum suppression*: each detection is first represented in 3-D position and scale space; a mean shift vector is iteratively calculated for each point until a convergence mode is reached and the list of all modes give the final results.
- *Computing the bounding box*: for each mode the bounding box is calculated according to the final center and scale.

2.5 Conclusion

In this chapter, we have discussed particle filters and HOG object detection algorithm. These are the two major parts in our proposed framework. With the knowledge of the two major components, we can proceed to the details of the proposed framework and how it is implemented.

Chapter 3

Detection-assisted Tracking by Mobile Cameras

Now we have the preliminary knowledge for developing and implementing the detection-assisted tracking framework. The goal of this framework is to perform robust visual tracking using mobile cameras and without invoking object detection at every frame. We first introduce the system architecture and then discuss each component in more detail. Then, we describe the implementation on PC platform. Since smart cameras are becoming more popular and they allow mobility, it is very important to detect objects without relying on static background subtraction. Thus, we have implemented the proposed on embedded smart cameras and describe our initial results in Chapter 4.

3.1 System Architecture

Fig. 3.1 shows the architecture of the framework. The detector and the tracker manager form the heart of the system. The detector provides observation to the

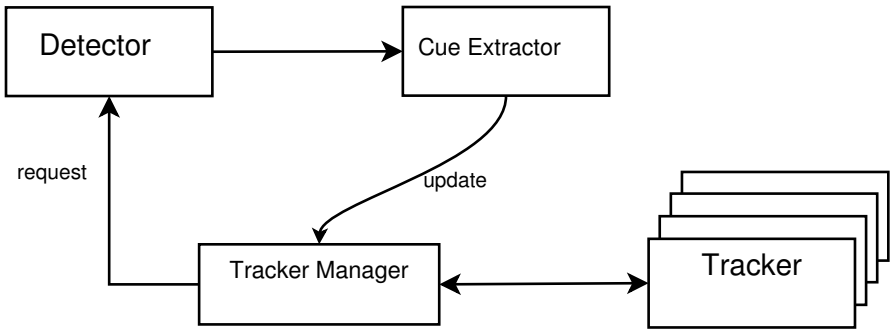


Figure 3.1: Overview of the system structure

rest of the system. Cue extractor extracts various visual cues from the observations and trackers use the extracted cues to perform visual tracking. Since the framework is designed to track multiple objects, there is a tracker manager for maintaining all trackers. The tracker manager creates a new tracker when there is a new observation.

In the initialization stage, the detector is invoked to provide the initial observations. The tracker manager creates trackers according to the initial observation. The tracker manager is also responsible for maintaining the states of trackers such as whether they need correction. For each frame, tracker manager checks each existing tracker to see whether it needs new observation and report this to the system. Then, the system invokes object detector for the next frame according to this information and provides the new observation to the tracker manager. The observation is provided as a list of regions of interest (ROI). Tracker manager then dispatches these observations to the trackers that need correction.

3.2 Detector

In the implementation, we use the standard histogram of oriented gradient (HOG) human detector. HOG human detector was first introduced in [18]. As discussed in

Chapter 2, the basic idea of HOG is that local visual features can be characterized well by the distribution of local intensity gradients or edge directions. An image is divided into cells and 1-D histograms of gradient direction are calculated. The calculated histograms are then normalized over cells to get the HOG. A support vector machine (SVM) is then trained to classify human and non-human regions.

3.3 The Tracker

There are various choices for trackers. The simplest tracker can be implemented by performing simple data association. For each region detected by the detector, a simple feature, such as color histogram, is calculated. Then the features are associated across frames by maximizing the similarity between two regions in two different frames. This method is simple, however, it is vulnerable to changes in environment settings such as lighting and it is difficult to handle complex scenarios such as occlusion.

Instead of using the simple data association approach, we use particle filter-based trackers in our implementation. As discussed in Chapter 2, particle filter is a probabilistic tracking method; instead of setting fixed threshold for associating data, it uses the prediction-correction approach. The tracker is continuously corrected by newly available observations. It is more robust to environmental changes and can handle occlusions better. Each tracker corresponds to a target in the scene so that the system can track multiple targets.

As discussed in Chapter 2, there are many types of particle filters. SIR (sampling-importance resampling) particle filter is one of the simple yet powerful particle filters. In a standard SIR particle filter [37], the resampling stage is performed for every time step. However, it is not necessary to perform resampling stage so frequently. In our implementation, a slight modification is done. Instead of perform resampling for each

time step, we perform resampling according to a certain predefined criterion, which will be discussed below.

One of the important steps in implementing particle filter is the calculation of the importance weight. According to the discussion in Chapter 2, the importance weight is calculated as

$$w_k^i = w_{k-1}^i p(\mathbf{z}_k|\mathbf{x}_k^i) \tag{3.1}$$

where $\mathbf{x}_k$ is the particle filter state at time step k and $\mathbf{z}_k$ is the observation at time step k. The term $p(\mathbf{z}_k|\mathbf{x}_k^i)$ is in fact the likelihood; it can be calculated by comparing the state of particles with the observations.

At the initialization stage, all the particle weights are set to $w = \frac{1}{N_p}$ where N_p is the number of particles. To decide when to perform resampling, we define the number of effective particles, N_{eff} [35]. However, it is impossible to calculate the exact solution for N_{eff}, we use $\hat{N_{\text{eff}}}$ to approximate N_{eff}; $\hat{N_{\text{eff}}}$ is calculates as

$$\hat{N}_{\text{eff}} = \frac{1}{\sum_{i=1}^{N_p}(w_k^i)^2} \tag{3.2}$$

Then a predefined ratio, p_{eff}, is used to decide whether resampling should be performed. If $\hat{N}_{\text{eff}} \leq p_{\text{eff}} \times N_p$, then resampling is performed; here $0 \leq p_{\text{eff}} \leq 1$.

A tracker is not only a particle filter in the proposed framework. It contains the tracking algorithm and other necessary algorithms such as fusing algorithms for combining multiple visual cues and classification for selecting the best observation. Particle filter is an instance of tracking algorithm and is used in our implementation.

3.3.1 The Observation Model

The observation model is one of the core components in particle filter tracking. The observation model defines how the measurements are related to the system states. The cue extractor in Fig. 3.1 is responsible for generating the observation model. Many features, such as color or intensity histogram, shape or texture of the target, can be selected as visual cues for the observation model. Color histogram is a commonly used visual cue for many vision applications. It is relatively easy to calculate yet provides details of the object. Therefore, we use color histogram for the observation model in our framework.

Representation Different from intensity histograms, we have to consider all the three components in a color histogram. 3-D color histograms are used for the observation model. Each color component has a range of $[0, 255]$; then this region is equally divided into N sub-regions. Then, the 3-D color histogram contains a N-D vector for each color component, ($\mathbf{r}$, $\mathbf{g}$ and $\mathbf{b}$). Each component of the N-D vector represents the number of pixels whose corresponding color value falls into the corresponding range. We use $\mathbf{p} = \{p_i\}_{i=1}^{N}$ to denote the color histogram.

Similarity A similarity measure for two color histograms is necessary to calculate the importance weights. A lot of similarity measures exist, such as Kullback-Leibler distance, sum of square distance, etc. The Bhattacharya coefficient is the one commonly used to measure similarity between two histograms and is defined as:

$$B = B[\mathbf{p}, \mathbf{q}] = \sum_{i=1}^{N} \sqrt{p_i q_i} \tag{3.3}$$

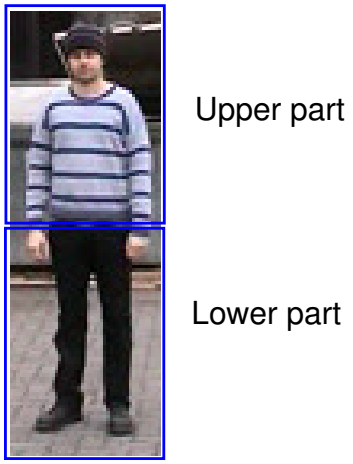


Figure 3.2: The divided histogram. For a detected object, it is divided into two parts and the overall similarity is calculated on the two parts.

Divided Histogram To make color histogram a more reliable descriptor, we equally divide the target region into two parts and then combine the similarity of the two parts, as shown in Fig. 3.2. This is based on the fact that the color appearance is usually consistent for shirts and pants, respectively. Assume that the similarities of the two parts are B_1 and B_2 respectively. Then the overall similarity is calculated using

$$B = \alpha_1 \times B_1 + \alpha_2 \times B_2 \tag{3.4}$$

In our implementation, we consider the two parts to have the same importance and therefore set $\alpha_1 = \alpha_2 = 0.5$. The overall similarity B is used to measure the similarity between two histograms.

Determining the Best Observation It is possible that the detector returns more than one candidate target regions. Trackers are responsible for choosing the best observation when more than one ROI is returned. The following two criteria are used to determine this and a similarity score is calculated using these two criteria for each

ROI. The comparison is made between a tracker and each candidate target region. The candidate that is most similar to the tracker is selected as the new observation and is used to update the tracker.

1. Color histogram: it measures the similarity of appearance; this is the most intuitive criterion. The color of a detected region must match that of the tracker in order to become a valid candidate.

2. Distance in image: the distance between a detected region and the tracker. This is used to eliminate false detections. This is based on the fact that the movement of a person is continuous. If a detected region has a very similar appearance but is far from the tracker, then it will not be considered as a valid candidate.

To describe how the similarity score is calculated, we first define the operator []:

$$[x] = \begin{cases} 1 & x \neq 0 \\ 0 & x = 0 \end{cases} \tag{3.5}$$

Let d_i denote the distance in image space and s_i the similarity of the color histograms; they are for the i^{th} candidate ROI and the tracker, respectively. Let w_d be the score for measuring the distance d_i and d_{th} denote the predefined threshold for distance. Then, we define the following expression for w_d.

$$w_d = (1 - \frac{d_i}{d_{th}} \times d_i)[d_i \leq d_{th}] \tag{3.6}$$

Let w_s denote the score for measuring the similarity of color histograms and s_{th} denote the corresponding threshold. We define w_s as follows.

$$w_s = s_i \times [s_i \geq s_{th}] \tag{3.7}$$

Let β_s and β_d be the weights associated with color histogram and distance, respectively. The best candidate ROI will then be calculated by solving the following optimization problem, where the superscript i indicates the i^{th} candidate, β_s and β_d satisfy that $0 \leq \beta_s \leq 1$, $0 \leq \beta_d \leq 1$ and $\beta_s + \beta_d = 1$.

$$C = \underset{i}{\operatorname{argmax}}(\beta_s^{(i)} w_s^{(i)} + \beta_d^{(i)} w_d^{(i)}) \tag{3.8}$$

3.3.2 The Dynamic Model

Besides the observation model, particle filter also needs a dynamic model to work. The dynamic model describes how the system states transit. Many system variables can be selected for the space model. In our implementation we use the position of the target in the image space, $\mathbf{x} = \{x, y\}$.

A simple dynamic model is used to propagate the system state. We assume that the position of the target is offset by a random value at each time step. Since both the target and the camera may be moving, it may not be suitable to assume a constant velocity model. Therefore, we use the random offset model to describe the movement of the target. The dynamic model is defined in (3.9).

$$(x, y)_t = (x, y)_{t-1} + (\Delta x, \Delta y) \tag{3.9}$$

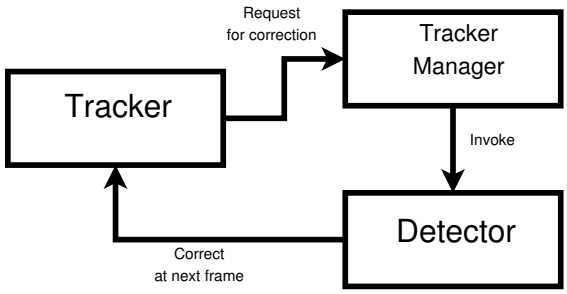


Figure 3.3: The procedure of correcting trackers. A tracker is aware of whether it needs correction; it sends request to the tracker manager. Then the tracker manager invoke the detector at the next frame.

where Δx and Δy are the increment of position and satisfy

$$p(x_k|x_{k-1}) \sim \mathcal{N}(\mu_x, \sigma_x) \tag{3.10}$$

$$p(y_k|y_{k-1}) \sim \mathcal{N}(\mu_y, \sigma_y) \tag{3.11}$$

The values of μ and σ are set by experiments to achieve the best performance.

3.3.3 Correcting Trackers

If the tracker can always track the target accurately, then it is not necessary to use a detector. However, as discussed in Chapter 2, particle filters degenerate with time, i.e. particle filters become less and less accurate as time goes. Therefore, we must have certain means for correcting trackers and putting them into good states so that they can perform tracking accurately. In our framework, we use HOG detector for this purpose. At each frame the trackers check themselves for degeneracy. If any of the trackers finds that it has degenerated, then it reports to the tracker manager and the detector will be invoked at the next frame. Fig. 3.3 shows the correction procedure.

3.3.4 The Timing of Correction

The detector is only invoked to assist the trackers when needed. We use the degeneracy state of the particle filter for determining whether the detector should be activated. Two different criteria are discussed in this section: the absolute number of resampling and the rate of resampling. For the absolute number of resampling each tracker maintains the number of resampling performed; for the rate of resampling each tracker maintains the number of frames between two resampling stages.

Absolute Number of Resampling Recall that we use the number of effective particles, N_{eff}, to describe the degree of degeneracy of a particle filter. Once the ratio $\frac{N_{\text{eff}}}{N_p}$ reaches the predefined threshold p_{eff}, then resampling will be performed.

Although resampling is aimed at alleviating degeneracy, it cannot completely eliminate degeneracy. When the particle filters degenerate, the performance becomes worse. Each tracker maintains the number of resampling performed, $N_{\text{resampling}}$, to monitor the degeneracy. Since resampling occurs when the degeneracy is no longer acceptable, $N_{\text{resampling}}$ is a good criterion for monitoring degeneracy. When $N_{\text{resampling}}$ is above a predefined threshold N_{th}, then the tracker will report that it needs correction.

Rate of Resampling Another criterion for determining whether detector should be invoked is the rate of resampling. We measure it as *frames per resampling*, i.e. the number of frames between two samplings. This is a local feature of video sequence describing how fast the tracker is degenerating at this moment. If the tracker is degenerating fast enough, then we invoke the detector to correct the tracker.

3.4 Implementation

In this section we will discuss implementation of the proposed framework in detail. The framework has been implemented and tested on PC platform. An initial attempt of implementing this framework on embedded smart camera platform has also been made. We will first describe the PC implementation and then the attempt on CITRIC smart camera platform.

3.4.1 PC Platform

The implementation on PC platform is done using C++, Qt and OpenCV 2.2; Qt is used for user interface and OpenCV is for common image processing tasks. The basic idea of implementing the framework is to utilize publicly available libraries whenever possible and to make the implementation extensible.

3.4.1.1 Implementation Overview

Fig. 3.4 is an overview of the implementation on PC platform. As shown in the figure, the *Pipeline* is top level component; it contains all the functional parts, such as `VideoCapture`, *ObjectDetector* and *TrackerManager*, and necessary glue logic that combine all the components together. The *Pipeline* exposes a very interface: `Start` and `Stop` whose functions are as the name suggest. A utility function called `TakeSnapshot` is also provided for saving screenshots to disk files.

VideoCapture is the input component; it supports input video data from both disk files and video devices such as webcam. We directly use `cv::VideoCapture` class from OpenCV for *VideoCapture* component. *VideoCapture* outputs a serial of

frames in the form of OpenCV's matrix objects, `cv::Mat`. The frame sequence is then sent to the *Detector* for further processing.

ObjectDetector is the one of the two core components in the framework. It employs HOG-based human detection algorithm to detect humans in an image. Since OpenCV provides a class called `cv::HOGDescriptor`, we use this class to implement our object detector. `cv::HOGDescriptor` contains a default SVM trained for detecting humans; we find it performs well on our data so we keep the default SVM. *ObjectDetector* takes as input an object of `cv::Mat` containing a frame and returns a list of possible regions corresponding to humans. The detected regions are directly sent to *TrackerManager*. The detector's only responsibility is to discover any possible regions; it does not associate the detected regions to corresponding trackers. *ObjectDetector* also provides a simple interface containing only two functions: `Init` and `Run`, for initializing the detector and running detection on a frame.

TrackerManager manages all the trackers in the system and is responsible for associating detected regions to trackers. *TrackerManager* can be considered as the top level interface of trackers; it maintains a list of all trackers and contains glue logic for managing the trackers. *TrackerManager* takes a `cv::Mat` object containing a frame and a list of detected regions as input and directly outputs the tracking results. It is also responsible for determining whether any tracker needs correction. In the common operation, *Pipeline* queries *TrackerManager* for the need of correction at each frame. *TrackerManager* and trackers are discussed in detail below.

VideoOutput outputs processed video data to a specified disc file. It also outputs processed data to screen. OpenCV provides `cv::VideoWriter` class for archiving

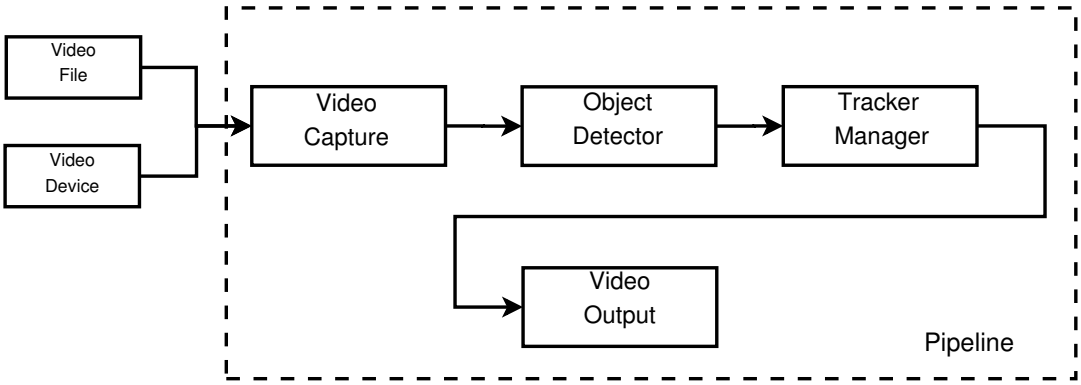


Figure 3.4: Overview of implementation on PC platform. The *Pipeline* is the top level component; it contains all the functional components and necessary glue logic. *VideoCapture* is responsible for input video data, supporting both video file and video devices. *TrackerManager* is responsible for managing trackers. *VideoOutput* is responsible to output processed video data, to both screen and disk files.

video data to disk files; we use this class to perform the same task. Every processed frame is encapsulated in a `cv::Mat` object and is sent to this component.

3.4.1.2 TrackerManager and Trackers

The *TrackerManager* maintains all the trackers and related control logic. The manager maintains a list of trackers. *TrackerManager* is the interface through which trackers interact with the other components in the framework; it is responsible for the following tasks:

- Creating new trackers
- Associating newly detected regions to the correct tracker
- Determining whether any tracker needs correction.

Trackers are self-contained components in the implementation. To make the system flexible, all trackers share a common interface defined in `TrackerBase` and

are derived from this interface. The `TrackerBase` interface supports the following operations:

- *Run*: perform tracking task on the provided frame;
- *DrawParticles*: draw the bounding box of the target in the frame;
- *NeedCorrection*: determines whether the tracker itself needs correction;
- *CorrectTracker*: performs the actual correction for the tracker.

At each frame, the tracker manager polls all the trackers that it maintains and checks whether there is any tracker that needs correction. If there is any, then the tracker manager notifies *Pipeline* and the detector will be invoked. Upon receiving the detected regions, the tracker manager passes the regions to the trackers; the trackers will then pick the best regions.

The main part of trackers is a Bayesian filter. In the implementation, a common interface for Bayesian filter, `BayesianFilter`, is defined. `BayesianFilter` provides a similar set of functions as `TrackerBase` except that it also outputs the expected position of the target. Any other Bayesian filters are expected to use `BayesianFilter` interface.

We have implemented a more concrete class `ParticleFilter` that implements `BayesianFilter` interface. As described in Chapter 2, different types of particle filters can be obtained by using different sampling methods. Therefore, `ParticleFilter` also serves as the foundation of particle filters. However, `ParticleFilter` interface adds `Resample` to `BayesianFilter` since particle filters needs resampling to alleviate degeneration while some Bayesian filters, such as Kalman filter, do not need this step.

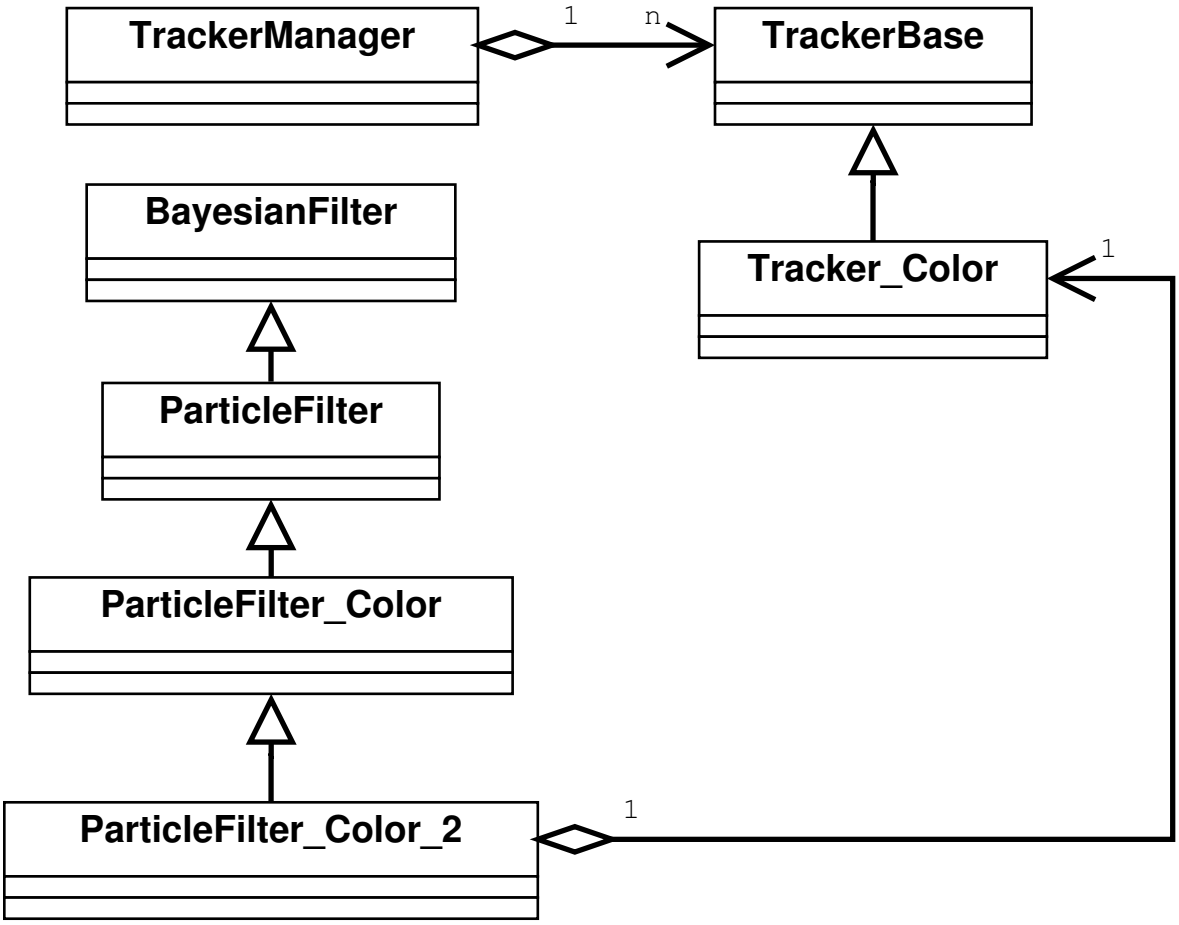


Figure 3.5: The relationship of *TrackerManager*, *Tracker* and *ParticleFilter*. It is easy to introduce new types of Bayesian filters and trackers to extend the system.

Fig. 3.5 summarizes the relationship among *TrackerManager*, *Tracker* and *ParticleFilter*. It is easy to extend the system deriving from the base classes. For example, a `KalmanFilter` class, which performs Kalman filtering, can be derived from `BayesianFilter` implementing required interface. To corporate the `KalmanFilter` into the system, we can define `Tracker_Kalman` that derives from `TrackerBase`.

3.4.2 CITRIC Platform

In addition to implementing the proposed framework on PC platform, we also implemented the HOG-based detector on the CITRIC platform. CITRIC is a Linux-powered embedded smart camera platform that was introduced in [41]. The platform is equipped with an XScale processor and a CMOS image sensor and runs a customized Linux operating system. Detailed information on the platform is provided in Section 4.1.1.

Due to the limited resources of the embedded camera, the training stage is performed on a PC. The size of each training sample is 64×128. A trained SVM vector is then ported onto the camera board.

For the HOG descriptor, unsigned orientations, spanning from 0 to 180 degree, in conjunction with 9 histogram bins are used to achieve the best performance. The size of the sliding window, the cells and the blocks are 64×128, 8×8 and 2×2, respectively. The overlapping between the blocks in the normalization step is 1. To handle the problem of different resolutions between the training samples and the test image, the test image is downsampled to multiple levels to search the possible positive detections. The final decision is made based on the distance between the HOG feature vector and the trained SVM vector.

The HOG calculation and classification are performed on the camera board. The frames are captured by the image sensor. After each frame is captured, it is searched through for the positive detections. Once a target object is found, a rectangle is drawn around the object. Then, this frame is saved in the camera.

3.5 Conclusion

In this chapter we have discussed the design and implementation of the proposed framework in detail. A full implementation of the proposed framework is done on PC platform using C++ and OpenCV 2.2. The proposed framework itself is extensible: any proper tracking and detection algorithms can be put in the places of tracker and detector. When implementing the framework, extensibility is also considered. As described in Section 3.4.1, it is easy to employ other types of Bayesian filters into the framework. We have also ported part of the framework onto the CITRIC embedded smart camera platforms. In the next chapter, we will look at the performance of the framework, on both PC and CITRIC platforms.

Chapter 4

Experiments and Evaluation on PC Platform

The experiments and evaluation are divided into two parts: on PC platform and on CITRIC platform. In this chapter we evaluate our implementation of the proposed framework on PC platform and on CITRIC platform in Chapter 5. The framework is extensible by plugging in different components. Although any object detection algorithm, such as detectors for cars or faces, can be used in our framework, we mainly focus on human detection in this thesis. We first describe how the experiments are performed and then present the experimental results.

4.1 Overview

For evaluating the PC implementation, we use the BoBoT dataset maintained by Bonn University [42]. The video sequences in BoBoT dataset are of size 320×240 and contain both human and objects. Since we mainly focus on human tracking, we only use the sequences that contain human. These sequences are recorded using a

Name	Size	Length (frames)	Features
Seq. D	320×240	1118	Moving camera, one target
Seq. F	320×240	432	Moving camera, occlusion, similar object
Seq. I	320×240	998	Moving camera, occlusion, many similar objects

Table 4.1: Summary of testing video sequence

moving camera and the trajectory of the camera is not known beforehand.

4.1.1 PC Platform

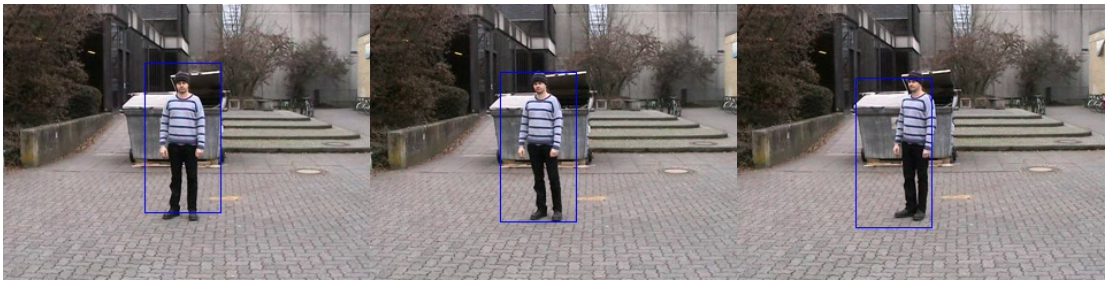
We use Ubuntu 10.10 for the PC platform. The computer is equipped with a dual core Intel E8600 CPU running at 3.33 GHz. The computer has 3G memory. The implementation uses OpenCV 2.2. We compiled OpenCV 2.2 from source on our computer. The platform also contains Qt4 and Boost library which are used in the implementation.

4.2 Evaluation of PC Implementation

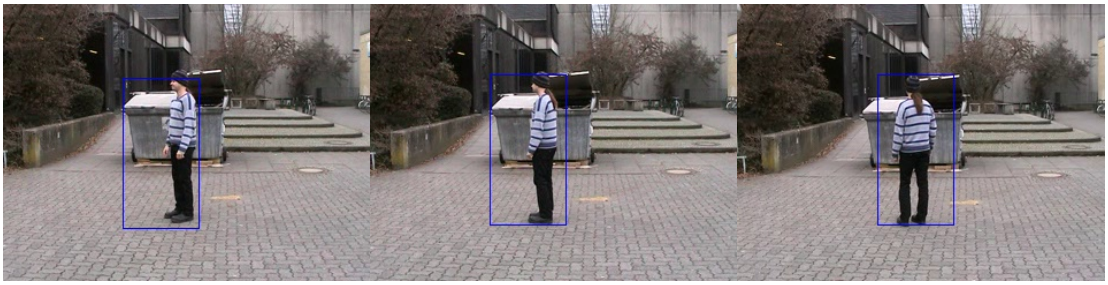
We first present the tracking results of the PC implementation. We use *Seq. D*, *Seq. F* and *Seq. I* for the experiments and the video sequences are all of size 320×240. Table 4.1 summarizes the information about the three testing video sequences.

4.2.1 Tracking without Correction

Fig. 4.1 shows the tracker performance with increasing number of particle filter re-sampling on video *Seq.D*. We can observe from the figure that the performance of the tracker is degrading. The degeneracy of the tracker is not quite severe since for this part of the video, the person and the camera are not performing significant movement.



(a) Frame 0 (b) Frame 10 (c) Frame 20



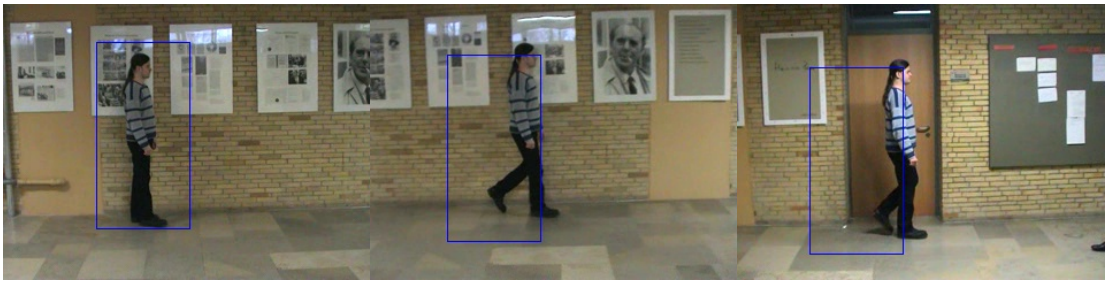
(d) Frame 30 (e) Frame 40 (f) Frame 50

Figure 4.1: The performance of the tracker with increasing number of resampling. The video is recorded by a moving camera. Figures (a) - (f) correspond the resampling count of 0, 10, 20, 30, 40 and 50 where (a) shows the initial detection. The performance degrades when resampling count increases.

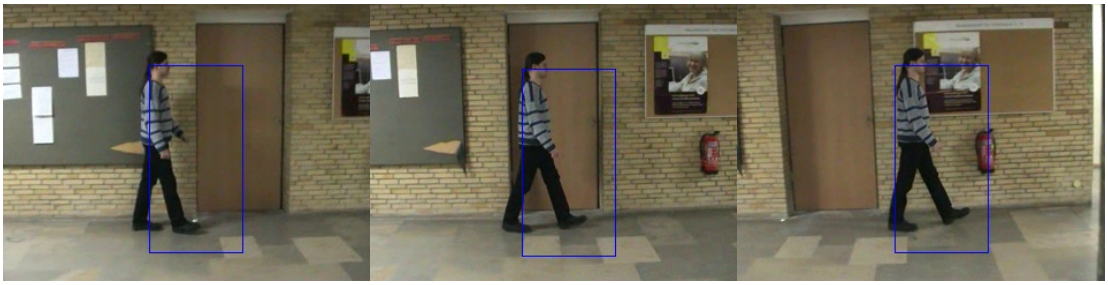
Fig. 4.2 shows the performance of the tracker *Seq. F*. From (a) to (f), the number of resampling increases from 0 to more than 170. In this video sequence, the person is walking and the camera is following the person. The degeneracy of the tracker becomes obvious under such significant movements.

4.2.2 Tracking with Detector

We ran the experiment on video *Seq. F* again, this time with the assistance of the detector. The results are shown in Fig. 4.3. In all the experiments, the threshold N_{th} is set to 30. This video sequence has 452 frames in total and the detector is invoked only 81 times. It is also worth noting that the detector may be invoked more



(a) Frame 0 (b) Frame 10 (c) Frame 20



(d) Frame 30 (e) Frame 40 (f) Frame 50

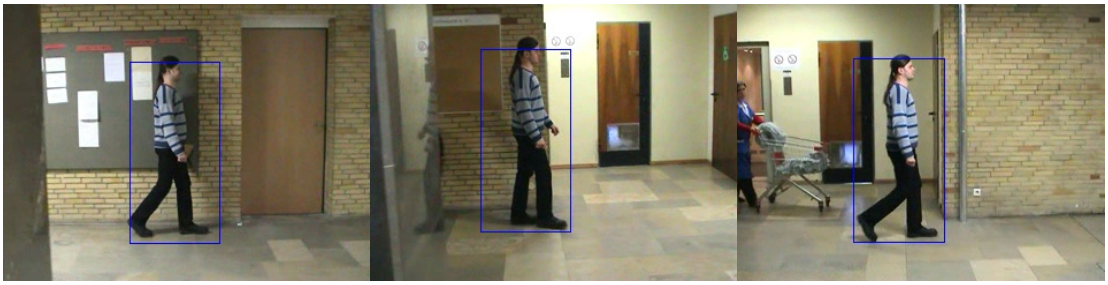
Figure 4.2: The performance of the tracker with increasing number of resampling on another video sequence. In this video sequence both the person and the camera are moving significantly: the person is walking down a hallway and the camera is following the person. From (a) - (f), the number of resampling increases from 0 to more than 170. The bounding box output of the tracker is deviating from the target.

than once for one correction request. This is due the observation selection scheme described in Section 3.3.1. If the tracker determines that no desired observation is provided, then the tracker keeps on requesting new observation. From the figures we can see that the tracking is much more accurate.

4.2.3 Handling Occlusion

We also tested the ability of handling occlusions and existence of similar targets using *Seq. F*. Fig. 4.4 shows the scenario where occlusion occurs. The person is occluded by the pillar in the middle of the video. The results show that our framework is able

(a) (b) (c)



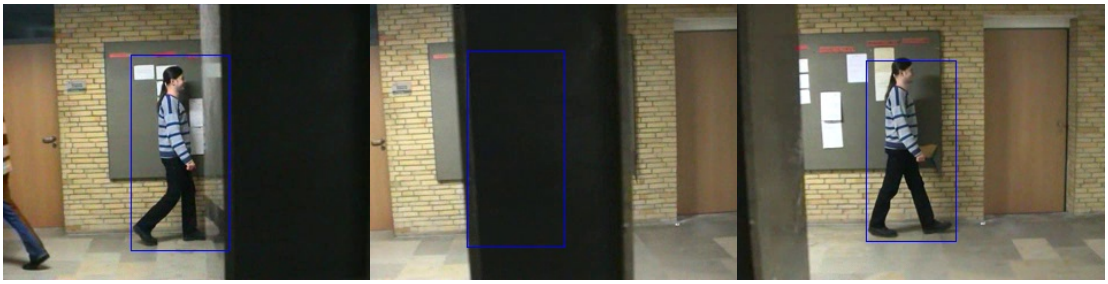
(d) (e) (f)

Figure 4.3: These figures show the performance of the tracker with the assistance of the detector. This is the same video sequence used in Fig. 4.2. Figures (a) - (f) show the bounding box output of the tracker throughout the whole video sequence. The threshold of the resampling count for correcting the tracker is set to 30. Better localization of the target is achieved with the help of the detector.

to handle occlusions.

4.2.4 Handling Similar Objects

Fig. 4.5 shows a scenario where similar objects coexist in the scene. When the detector is invoked, two objects are detected and both ROIs are sent to the tracker. The tracker discarded the ROI of the disturbing object according to the selection scheme. These results show that the proposed framework is able to handle situations where similar objects exist.



(a) (b) (c)

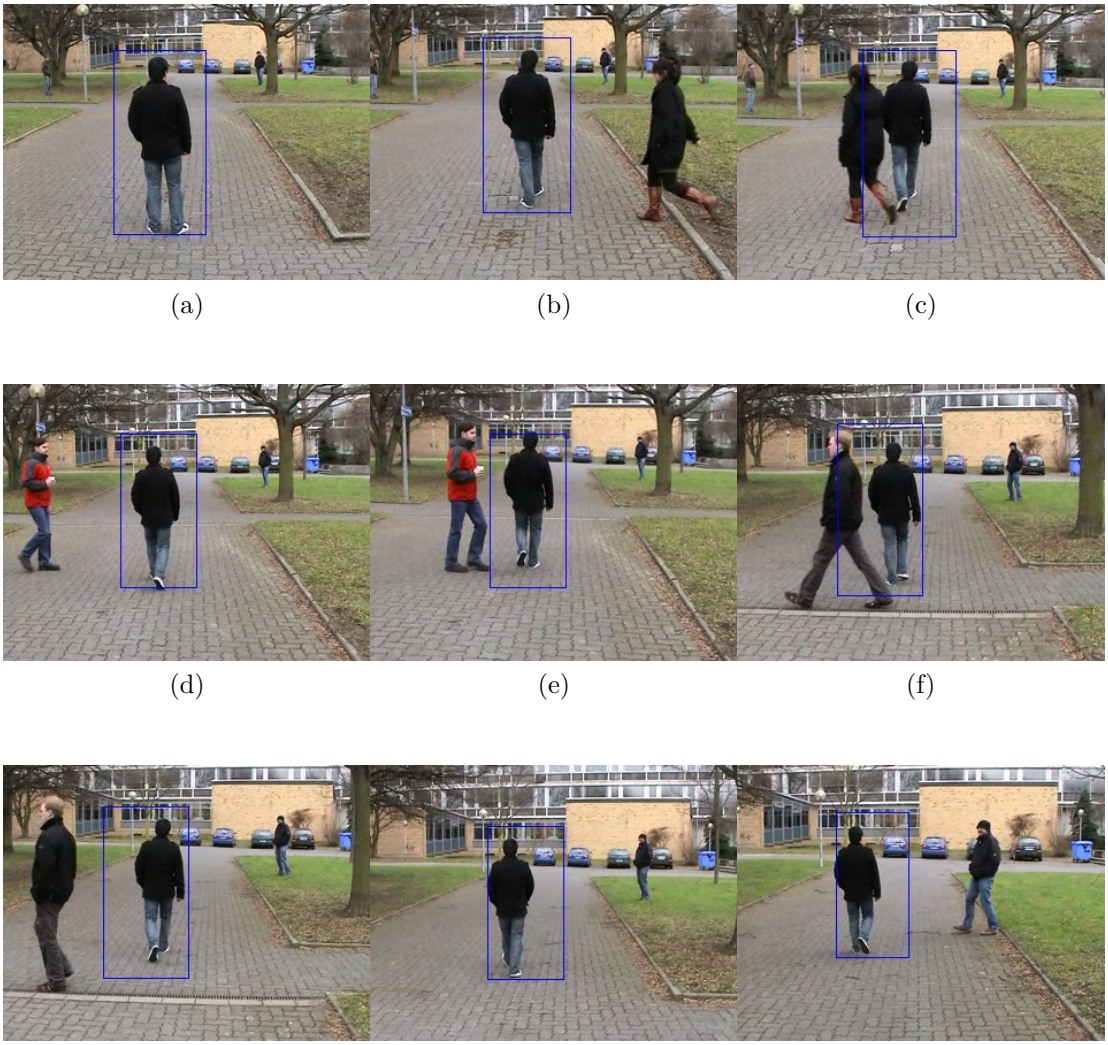
Figure 4.4: A scenario with occlusion, where the detector correction is turned on. (a) shows the instance when a new detection is performed.

(a) (b) (c)

Figure 4.5: A scenario where similar objects coexist. The detector correction is turned on. A person wearing similar clothes appears in the middle of the video. Our framework is able to handel situations where similar objects coexist.

4.2.5 Handling Crowded Scenes

We also tested the proposed method on a more challenging (*Seq. I*) where both the camera and the target are moving forward. There are multiple similar objects in the scene and they sometimes occlude the tracked target of tracking. The tracker correction stage is performed multiple times during the presence of the distracting objects, and the tracker is able to choose the correct observations. This sequence has 1016 frames in total and the detector is invoked 26 times. The results are shown in Fig. 4.6. We can see from the results that our proposed framework is able to handle



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 4.6: Results on a challenging scenario where both the target and the camera are moving forward and multiple similar objects exist.

such complicated scenarios without performing the object detection at every frame.

4.3 Conclusion

In this chapter we demonstrated the implementation of the proposed framework on PC platform. Unlike other detection based tracking framework, the proposed framework uses detection as an assistant to the tracking algorithm. To handle the existence of multiple object when the detector is activated, we also developed a scheme to help the tracker select the correct observation. The results show that our proposed framework is able to handle complicated scenarios such as occlusion and existence of multiple similar objects.

Chapter 5

Experiments and Evaluation on CITRIC Platform

In this chapter we present the performance of the HOG-based human detector running on static and mobile embedded smart cameras. We show the outputs and report the processing times on four different scenarios. One thing to note is that target detection and tracking from videos captured by mobile cameras is a challenging and computationally expensive task even for powerful computer platforms. Yet, it needs to be performed on the embedded platforms, as the next step of having operational *mobile* embedded smart cameras. The presented results are promising, and provide insight on the capabilities and limitations.

5.1 Overview

For evaluating the implementation on CITRIC platform, we use realtime data, i.e. we use CITRIC to capture video sequences and process them online, then the results are kept in the camera. A CITRIC camera is mounted on a remote-controlled car so

Figure 5.1: The CITRIC camera is mounted on a remote-controlled car.

that we obtain a mobile platform; this is shown in Fig. 5.1.

5.1.1 CITRIC Platform

The CITRIC platform is a Linux-enabled embedded platform. The camera board is composed of an image sensor, a fixed-point microprocessor, external memories and other supporting circuits. The camera is capable of operating at 15 frames per second (fps) in VGA and lower resolutions [41]. Fig. 5.2 shows the camera board.

The image sensor of the camera board is a Omni Vision OV9655, which is a low voltage SXGA CMOS image sensor and designed to perform well in low-light conditions. It supports image sizes SXGA (1280×1024), VGA (640×480), and any size scaling down from VGA. The microprocessor PXA270 is a fixed-point processor

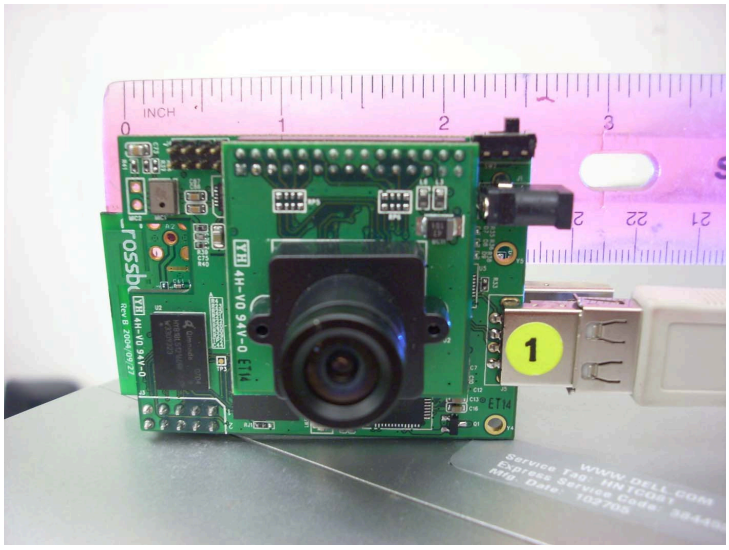


Figure 5.2: The CITRIC camera board.

with a maximum speed of 624MHz and 256KB of internal SRAM. It is capable of working in low voltage and low frequency, as low as 0.85V and 13MHz, to achieve low power consumption. Besides the internal memory of the microprocessor, the PXA270 is connected to a 64MB of SDRAM and 16MB of NOR FLASH. A USB-to-UART bridge controller is connected between the PXA270 UART port and the USB port on a personal computer. The camera board can be powered by the USB port from a personal computer, or four AA batteries.

We have run all of our experiments in QVGA (320×240) resolution. The algorithms run on the embedded Linux system imported onto the microprocessor. The frames of interest can be saved in JPEG format on the SDRAM.

5.2 Evaluation of Embedded Implementation

As an intial attempt, to implement the framework on embedded smart cameras, we have ported the core component, the detector, to the CITRIC camera. Since this is

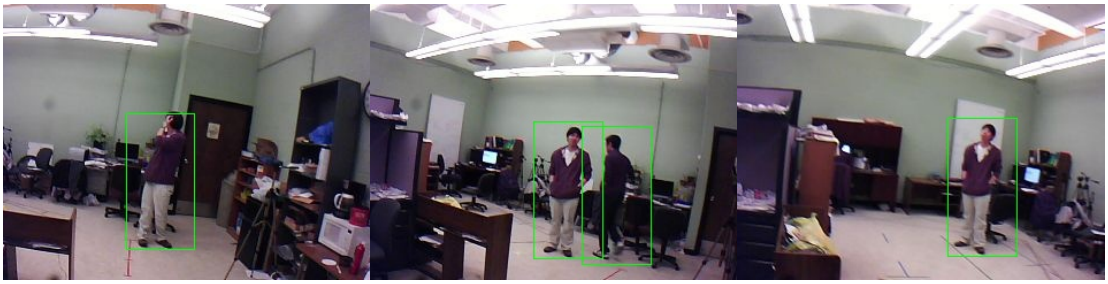
the preliminary stage of our work, we still keep the floating point calculations in the program to achieve the a performance as on PC. Thus, the processing speed of each frame on the embedded camera is much slower than in a PC, as expected. One of our future works is optimizing our program by implementing float point calculations by fixed point, or making the program more memory efficient to save from both memory accessing time and power.

To improve the processing speed of the current algorithm, we explore adding some reasonable assumptions based on our applications. For example, we may only need to watch an ROI in the view, instead of searching the whole frame. If the ROI is much smaller than the whole frame, the processing time of each frame decreases significantly.

The specification of the CITRIC platform is described in Section 4.1.1 and is repeated here: the microprocessor on the camera mote is PXA270, a fixed-point processor from Marvell with a maximum speed of 624MHz and 256KB of internal SRAM. In order to achieve good performance, we use the floating point operations in the HOG detection algorithm. Since the processor of CITRIC camera mote does not contain a floating point processor, the detector is expected to run slowly on the mote. We can expect better performance on more recent embedded processors.

5.2.1 Running on Whole Frame

In the first scenario, we performed human detection on the whole frames captured by the camera mote. The camera mote was held by a person who was moving around. Fig. 5.3 shows the detection result. As can be seen, even though the background is complex and continuously changing, the detector can successfully detect the person(s) in the scene. In Fig. 5.3b, two people in the scene are close to each other, and the



(a) (b) (c)

Figure 5.3: The detection results when the camera was held by a person who was moving around. The detector can handle cases where two people are close to each other, and is robust to illumination changes.

detector is able to correctly detect both of them. In addition, in Fig. 5.3a through Fig. 5.3c, we can observe certain changes in illumination. he HOG-based detector is robust to these changes, and shadow effects. It takes 37 seconds to perform detection on a frame of size 320×240. As mentioned above, having no hardware support for floating point operations contributes to this.

5.2.2 Accelerating by Reducing Frame Size

In order to see the effect of the size of area that is processed, we implemented the second scenario. In this experiment, the camera is static and observes the door to detect when someone passes through the door. Since the door only occupies a relatively small portion of the whole frame, we cropped a whole frame into a smaller image. Fig. 5.4 shows the detection results for such a scenario. The size of the cropped portion is 80×170. It takes around 4.5 seconds to perform the detection on the microprocessor, which is much faster than processing the whole frame. Therefore, by carefully choosing the region of interest (ROI), it is possible to decrease the processing time. However, such an assumption will limit the application to static cameras since

(a) (b) (c)

(d) (e) (f)

Figure 5.4: Detection results when the camera is fixed and only a portion of a frame is sent to the detector. The detector can correctly handle such conditions and the running time is reduced significantly.

the ROI will change as the camera moves.

5.2.3 Moving the CITRIC Camera

We have two different scenarios for mobile camera setting. In first scenario, the CITRIC camera is mounted on a remote-controlled car, and the car is driven around in a room. Fig. 5.1 shows the setup used in our experiments. In this case, the camera is close to the ground since it was directly mounted on the car, resulting in an oblique

view of people. No assumption is made on how the car could move, i.e. the car can be freely driven around the room. Such a scenario is of interest for robotic applications where a camera-equipped robot or vehicle roam around a space to detect targets.

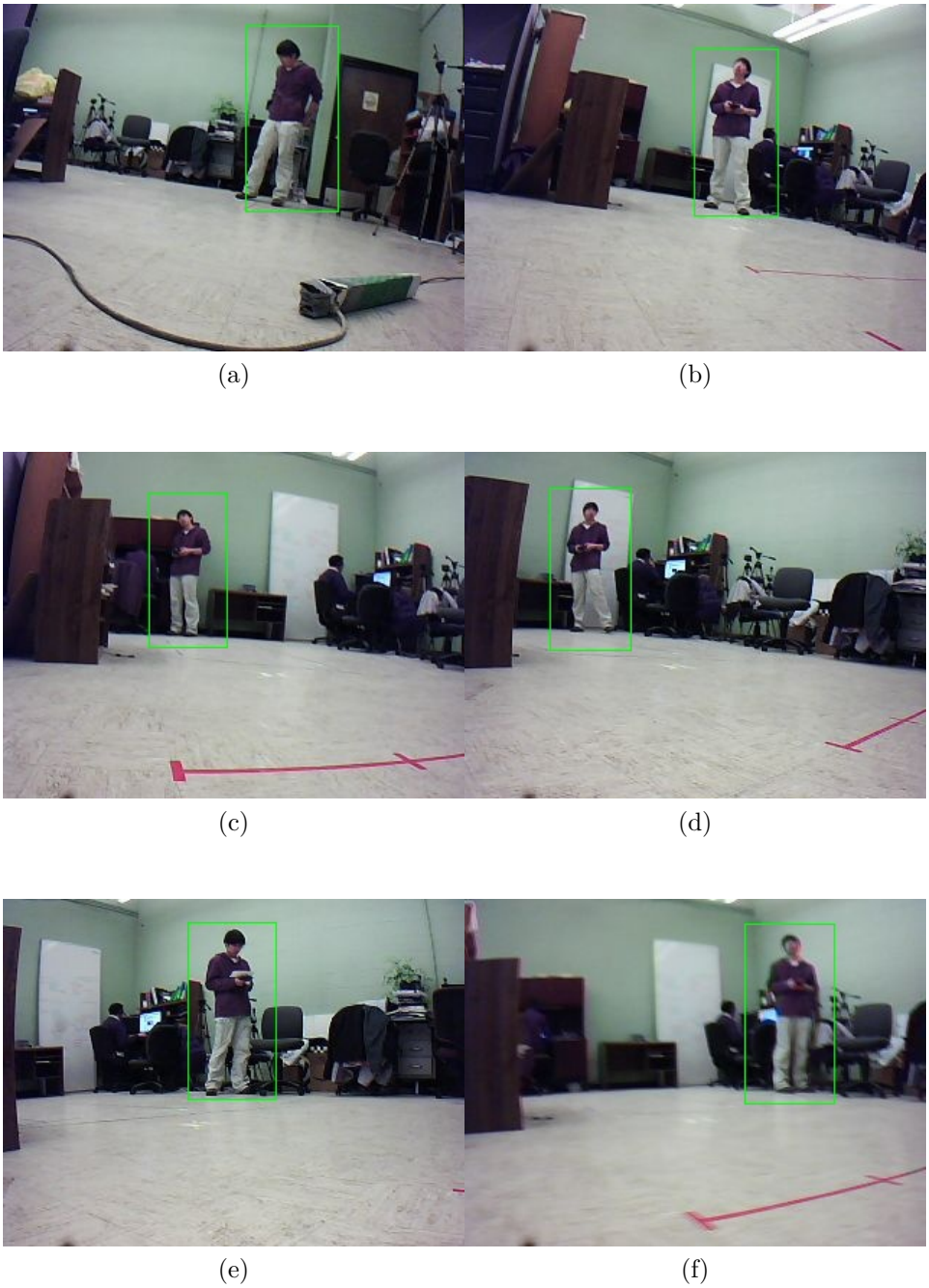
Since camera can move around in any direction, we did not crop the frames. Thus, the frame size for processing is 320×240, and it takes around 37 seconds to process a frame.

Fig. 5.5 shows the detection results. We can see that the detector performs well in this scenario despite the cluttered background. Also note that even though the image in Fig. 5.5f is blurred due to camera movement, we can still successfully detect the person.

In this second scenario, we kept the camera mounted on the car but used a tripod to lift the camera, and thus have a better field of view. The car is driven along a hallway. To be able to perform cropping and see its effect, two assumptions are made: (a) the car moves along the hallway along almost a straight line, (b) A person does not get too close to the camera. Since the camera is moving, we do not crop the frame in the horizontal direction, i.e. keep the full width of the frame. Under the above assumptions, we can crop the frame vertically. We have cropped 60 rows from the top of each frame. Thus, the size of the cropped image is 320×180. It takes 20 seconds to process a frame. Since we are using the full width of the frame, the processing time is longer than the second scenario. Fig. 5.6 shows the detection results for this scenario. As can be seen, the detector can handle the changes in both color and illumination.

5.2.4 Adaptive Frame Cropping

From previous experiments, we can see that the performance can be significantly improved by reducing the frame size. Inspired by this, we designed this scenario.

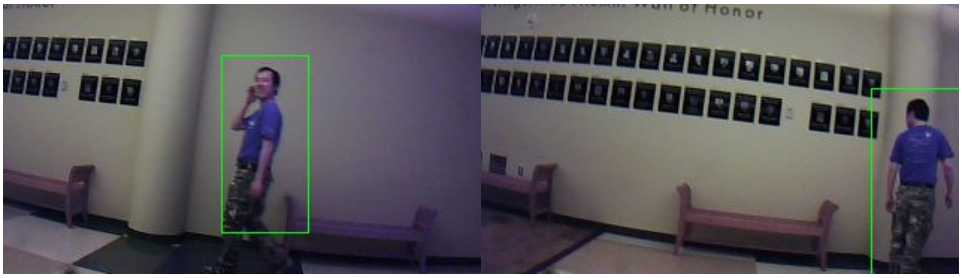


(a) (b)

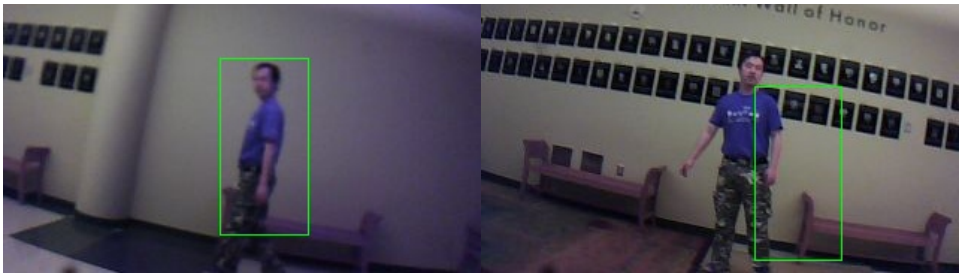
(c) (d)

(e) (f)

Figure 5.5: Detection results when the camera is mounted on a remote-controlled car, which was driven around in the room.



(a) (b)



(c) (d)

(e) (f)

Figure 5.6: The camera is mounted on a remote-controlled car and is driven along a hallway. The image is cropped vertically to reduce the processing time.

In this case, we dynamically determine the frame size. Frames are only cropped vertically, i.e. we keep the full width of frames. We assume that a person always walks on the ground. This yields the fact that a person only appears in a certain portion of a frame. Therefore, the idea is to estimate the region where a person may appear. We start by detecting people in the whole frame. If we have successful detections in following frames, then we record the boundary of each detected person as shown in Fig. 5.7a. Within each frame i, the minimum value of *Upper*, $U_{min}^{(i)}$ and the maximum value of *Lower*, $L_{max}^{(i)}$, are evaluated across all the detected persons. Each $U_{min}^{(i)}$ and $L_{max}^{(i)}$ are compared across frames to get the global minimum and maximum, U_{min} and L_{max}, which define the region where targets may appear. This is expressed in the following equations:

$$U_{min}^{(i)} = \min_k(y_k^{(i)}) \tag{5.1}$$

$$L_{max}^{(i)} = \max_k(y_k^{(i)} + h_k) \tag{5.2}$$

$$U_{min} = \min_i(U_{min}^{(i)}) \tag{5.3}$$

$$L_{max} = \max_i(L_{max}^{(i)}) \tag{5.4}$$

To handle boundary conditions, we pad 10 pixels to $Upper_{min}$ and $Lower_{max}$ for the cropping. However, it is possible that the target moves outside the boundary but is still inside the field of view of the camera. To handle such a condition, it is necessary to revert to processing the whole frame. We use a simple criterion for this experiment: if we detect nothing within 10 successive frames, then we revert to processing the whole frame on 11^{th} frame. Once we revert to the default operation, the above procedure restarts. The procedure for adaptive cropping is shown in Fig. 5.7b.

The camera is again mounted on the top of the remote-controlled car. The de-

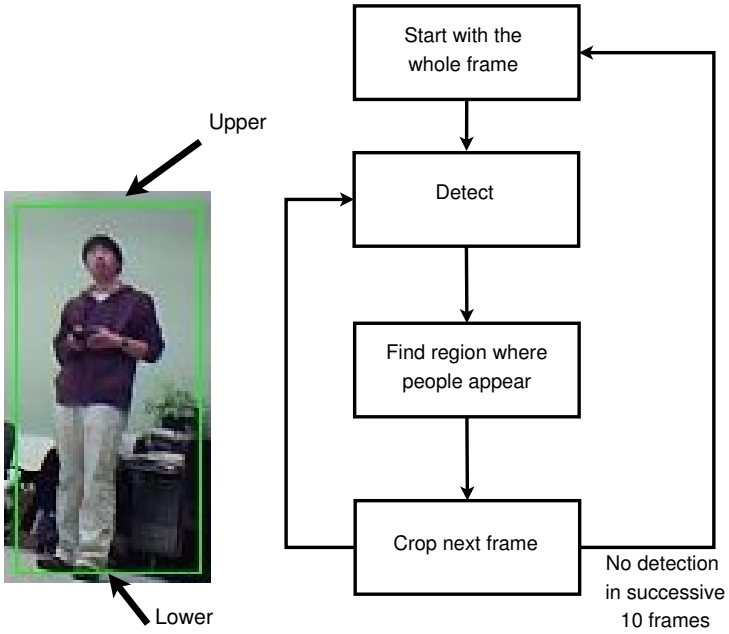
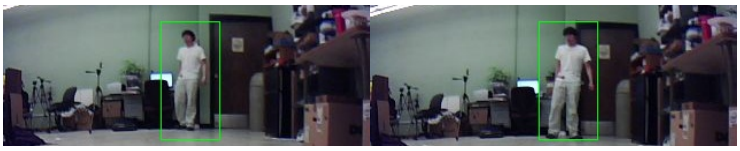


Figure 5.7: The boundary of a detected object and flow for automatic cropping. The boundary of a detected object is defined as Fig. 5.7a; note that in the image coordinate, the values of Upper is smaller than that of Lower. Fig. 5.7b shows the procedure for automatically crop a frame.

tection results are shown in Fig. 5.8. The camera starts with processing the whole frame. Fig. 5.8b, Fig. 5.8c and Fig. 5.8d shows that the camera has detected a person and started estimating the boundaries. Fig. 5.8e shows the first frame when the camera reverts to the default mode due to the lost of target. In this frame, the camera is working in default mode and detects a person. Then, the camera estimates the boundaries and crops the next frame, as shown in Fig. 5.8f. The average processing time for each frame is around 6 seconds.

Table 5.1 summarizes the performance as well as camera settings of the different processing methods used on CITRIC platform.

(a) Frame 0 (b) Frame 6



(c) Frame 14 (d) Frame 15

(e) Frame 62 (f) Frame 70

Figure 5.8: The camera is automatically cropping the frame. In Fig. 5.8a the camera starts with processing the whole frame. In Fig. 5.8b, the boundaries are estimated using the detected target and following frames are cropped. In Fig. 5.8c and Fig. 5.8d estimated boundaries are updated using the detection results. In Fig. 5.8e the camera reverts to the default mode due to lost of target. When the camera detects a person again, it restarts the adaptive cropping flow, as shown in Fig. 5.8f.

Processing Method	Performance (sec/frame)	Camera Setting
Whole frame	37	Static
Frame cropped to 80×170	4.5	Static
Frame cropped to 320×180	20	Moving
Adaptive frame cropping	6	Moving

Table 5.1: Summary of performance on CITRIC platform

5.3 Conclusion

Towards the goal of performing object detection and tracking with mobile embedded smart cameras, we have ported the HOG-based human detector onto the CITRIC camera mote that combines a camera sensor with a microprocessor. HOG-based detectors allow us to detect foreground objects with moving cameras, and are much more robust towards illumination changes, shadows and image blur. Ability to detect objects with moving cameras has application in different areas including robotics, surveillance and smart driving systems. We have provided output images and reported processing times when using static and mobile cameras for different scenarios. The presented results are very promising, and provide insight on the capabilities and limitations of these embedded platforms.

Chapter 6

Conclusion and Future Work

In this thesis we presented the detection-assisted tracking framework and its performance. Unlike other detection based tracking framework, the proposed framework uses detection as an assistant to the tracking algorithm. To handle the existence of multiple object when the detector is activated, we also developed a scheme to help the tracker select the correct observation. The results show that our proposed framework is able to handle complicated scenarios such as occlusion and existence of multiple similar objects.

The framework can be easily extended by plugging in components. Current results show that our framework is capable of handling complex scenes. We can employ even more sophisticated tracking algorithms and further process the detector output to handle more challenging scenarios.

Towards the goal of performing object detection and tracking with mobile embedded smart cameras, we have ported the HOG-based human detector onto the embedded smart camera platform — CITRIC. HOG-based detectors allow us to detect foreground objects with moving cameras, and are much more robust towards illumination changes, shadows and image blur. Ability to detect objects with moving cameras

has application in different areas including robotics, surveillance and smart driving systems. Due to the nature of the HOG-based algorithm, the speed of the microprocessor, and not having hardware support for floating point operations, processing a 320 image takes around 37 seconds. Depending on the application, we can crop input images in different ways, and thus decrease the processing time significantly. The presented results are very promising, and provide insight on the capabilities and limitations of these embedded platforms.

The proposed framework can be improved on embedded platforms in two directions. The first direction is to use more powerful embedded processors. Driven by the development of VLSI technologies, more and more processors are equipped with hardware floating point support; some of them are even equipped with GPUs. We can use these processors to build a new hardware platform. The second direction is to tailor the detector according to specific applications. In this thesis, we keep floating point operations and use the full algorithm for accuracy. In fact, the algorithm can be simplified according to the application and fixed point operation can be used.

Bibliography

[1] W. Niu, J. Long, D. Han, and Y.-F. Wang, "Human activity detection and recognition for video surveillance," in *Proc. of IEEE International Conference on Multimedia and Expo*, 2004. 1.1.1

[2] C. Rougier, J. Meunier, S.-A. A, and J. Rousseau, "Robust video surveillance for fall detection based on human shape deformation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21(5), pp. 611 – 622, 2011. 1.1.1

[3] S. Denman, C. Fookes, J. Cook, C. Davoren, A. Mamic, G. Farquharson, D. Chen, B. Chen, and S. Sridharan, "Multi-view intelligent vehicle surveillance system," in *Proc. of IEEE International Conference on Video and Signal Based Surveillance*, 2006. 1.1.1

[4] S. Park and C. S. G. Lee, "A global and local robot tracking and control strategy using multisensory inputs," in *Proc. of IEEE International Conference on Robotics and Automation*, 1994. 1.1.1

[5] H. Lang, Y. Wang, and W. de Silva Clarence, "Vision based object identification and tracking for mobile robot visual servo control," in *Proc. of IEEE International Conference on Control and Automation*, 2010. 1.1.1

[6] G. Shin and J. Chun, "Vision-based multimodal human computer interface based on parallel tracking of eye and hand motion," in *Proc. of International Conference on Convergence Information Technology*, 2007. 1.1.1

[7] V. I. Pavlovic, R. Sharma, and T. S. Huang, "Visual interpretation of hand gestures for human-computer interaction: a review," *IEEE Transactions on Pattern Analysis and Machine Intelligence,*, vol. 19(7), 1997. 1.1.1

[8] H. Aghajan and A. Cavallaro, Eds., *Multi-Camera Networks Principles and Applications*. Academic Press, 2009. 1.1.3

[9] Y. Wang, L. He, and S. Velipasalar, "Realtime distributed tracking with non-overlapping cameras," in *Proc. of IEEE International Conference on Image Processing*, 2010. 1.2

[10] Y. Wang, S. Velipasalar, and M. Casares, "Cooperative object tracking and composite event detection with wireless embedded smart cameras," *IEEE Transactions on Image Processing*, vol. 19, pp. 2614 – 2633, 2010. 1.2

[11] S. Velipasalar and et al., "A scalable clustered camera system for multiple object tracking," *EURASIP Journal on Image and Video Processing*, 2008. 1.2

[12] M. Casares and S. Velipasalar, "Light-weight salient foreground detection for embedded smart cameras," in *Proc. of the ACM/IEEE Intl Conf. on Distributed Smart Cameras*, 2008. 1.2

[13] B. Leibe, K. Schindler, and L. V. Gool, "Coupled detection and trajectory estimation for multi-object tracking," in *IEEE 11th International Conference on Computer Vision*, 2007. 1.2

[14] C. Huang, B. Wu, and R. Nevatia, "Robust object tracking by hierarchical association of detection responses," in *Proc. of the 10th European Conference on Computer Vision: Part II*, 2008. 1.2

[15] L. Zhang, Y. Li, and R. Nevatia, "Global data association for multi-object tracking using network flows," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, 2008. 1.2

[16] M. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. V. Gool, "Robust tracking-by-detection using a detector confidence particle filter," in *Proc. of IEEE 12th International Conference on Computer Vision*, 2009. 1.2

[17] Y. Li, C. Huang, and R. Nevatia, "Learning to associate: Hybridboosted multi-target tracker for crowded scene," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009. 1.2

[18] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005. 1.2, 2, 2.4, 3.2

[19] L. He, Y. Wang, S. Velipasalar, and M. C. Gursoy, "Human detection using mobile embedded smart cameras," in *Proc. of International Conference on Distributed Smart Cameras (to appear)*, 2011. 1.2

[20] J. Wen, H. Gong, X. Zhang, and W. Hu, "Generative model for abandoned object detection," in *Proc. of IEEE International Conference on Image Processing (ICIP)*, 2009. 2

[21] I. Fasel, B. Fortenberry, and J. Movellan, "A generative framework for real time object detection and classification," *Computer Vision and Image Understanding*, vol. 98(1), 2005. 2

[22] T. Mita, T. Kaneko, B. Stenger, and O. Hori, "Discriminative feature co-occurrence selection for object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30(7), 2008. 2

[23] J. O. Berger, *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag, 1985. 2.1

[24] T. Michaeli and Y. C. Eldar, "Hidden relationships: Bayesian estimation with partial knowledge," *IEEE Transactions on Signal Processing*, vol. 59(5), 2011. 2.1

[25] Y. C. Ho and R. C. K. Lee, "A bayesian approach to problems in stochastic estimation and control," *IEEE Transactions on Automatic Control*, vol. 9, pp. 333–339, 1964. 2.1

[26] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear-non-gaussian bayesian tracking," *IEEE Transactions on Signal Processing*, 2003. 2.1, 2.1.1, 2.3.2

[27] D. Simon, *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley-Interscience, 2006. 2.1.1

[28] M. O. Rabin, "Probabilistic algorithm for testing primality," *Journal of Number Theory*, vol. 12(1), 1980. 2.2

[29] J. C. Spall, *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Wiley-Interscience, 2003. 2.2

[30] D. J. C. MacKay, *Learning in Graphical Models.* The MIT Press, 1998, ch. Introduction to Monte Carlo Methods. 2.2.1

[31] ——, *Information Theory, Inference and Learning Algorithms.* Cambridge University Press, 2003. 2.2.3

[32] C. M. Bishop, *Pattern Recognition and Machine Learning.* Springer, 2007. 2.2.3

[33] A. C. Davison and D. V. Hinkley, *Bootstrap Methods and their Application.* Cambridge University Press, 1997. 2.2.4

[34] A. Doucet, "On sequential monte carlo methods for bayesian filtering," Department of Engineering, Univ. Cambridge, UK, Tech. Rep., 1998. 2.3.2

[35] J. S. Liu and R. Chen, "Sequential monte carlo methods for dynamical systems," *Journal of the American Statistical Association*, vol. vol. 93, pp. 1032 – 1044, 1998. 2.3.2, 3.3

[36] G. Kitagawa, "Monte carlo filter and smoother for non-gaussian non-linear state space models," *Journal of Computational and Graphical Statistics*, vol. 5, pp. 1–25, 1996. 2.3.2

[37] M. Isard and A. Blake, "Condensation — conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29(1), pp. 5–28, 1998. 2.3.2, 3.3

[38] D. G. Lowe, "Object recognition from local scale-invariant features," in *IEEE International Conference on Computer Vision*, 1999. 2.4

[39] N. Dalal, "Finding people in images and videos," Ph.D. dissertation, LInstitue National Polytechnique de Grenoble, 2006. 2.4.1, 2.4.3, 2.4.4

[40] T. Joachims, *Advances in Kernel Methods - Support Vector Learning*. The MIT Press, 2009, ch. Making large-scale svm learning practical. 2.4.4

[41] P. Chen and et al., "Citric: A low-bandwidth wireless camera network platform," in *Proc. of the ACM/IEEE International Conference on Distributed Smart Cameras*, 2008. 3.4.2, 5.1.1

[42] D. A. Klein, D. Schulz, S. Frintrop, and A. B. Cremers, "Adaptive real-time video-tracking for arbitrary objects," in *International Conference on Intelligent Robots and Systems (IROS)*, 2010. 4.1