Spring 4-17-2013

Distributed Multi-object Tracking with Multi-camera Systems Composed of Overlapping and Non-overlapping Cameras

Youlu Wang
University of Nebraska-Lincoln, youlu.wang@huskers.unl.edu

DISTRIBUTED MULTI-OBJECT TRACKING WITH MULTI-CAMERA SYSTEMS COMPOSED OF OVERLAPPING AND NON-OVERLAPPING CAMERAS

by

Youlu Wang

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Doctor of Philosophy

Major: Electrical Engineering

Under the Supervision of Professors Sina Balkır and Senem Velipasalar

Lincoln, Nebraska

May, 2013

DISTRIBUTED MULTI-OBJECT TRACKING WITH MULTI-CAMERA SYSTEMS COMPOSED OF OVERLAPPING AND NON-OVERLAPPING CAMERAS

Youlu Wang, Ph.D.

University of Nebraska, 2013

Adviser: Sina Balkır and Senem Velipasalar

Multiple cameras have been used to improve the coverage and accuracy of visual surveillance systems. Nowadays, there are estimated 30 million surveillance cameras deployed in the United States. The large amount of video data generated by cameras necessitate automatic activity analysis, and automatic object detection and tracking are essential steps before any activity/event analysis. Most work on automatic tracking of objects across multiple camera views has considered systems that rely on a back-end server to process video inputs from multiple cameras. In this dissertation, we propose distributed camera systems in peer-to-peer communication. Each camera in the proposed systems performs object detection and tracking individually and only exchanges a small amount of data for consistent labeling. With the lightweight and robust algorithms running in each camera, the systems are capable of tracking multiple objects in a real-time manner.

The cameras in the system may have overlapping or non-overlapping views. With partially overlapping views, the object labels can be handed off between cameras based on geometric relations. Most camera systems with overlapping views attach cameras to PCs and communicate via Ethernet, which hinders the flexibility and scalability. With the advances in VLSI technology, smart cameras have been introduced. A smart camera not only captures images, but also includes a processor, memory and communication interface making it a stand-alone unit. We first present a wireless embedded smart camera system for cooperative object tracking and detection of composite events. Each camera is a CITRIC mote

consisting of a camera board and a wireless mote. All the processing is performed on camera boards. Power consumption of the proposed system is analyzed based on the measurements of operating currents for different scenarios.

On the other hand, in wide-area tracking applications, it is not always realistic to assume that all the cameras in the system have overlapping fields of view. Tracking across non-overlapping views present more challenges due to lack of spatial continuity. To address this problem, we present another distributed camera system based on a probabilistic Petri Net framework. We combine appearance features of objects as well as the travel-time evidence for target matching and consistent labeling across disjoint camera views. Multiple features are combined by adaptive weights, which are assigned based on the reliability of the features and updated online. We employ a probabilistic Petri Net to account for the uncertainties of the vision algorithms and to incorporate the available domain knowledge.

Synchronization is another important problem for multi-camera systems, because it is essential to have the precise relevance between the video data captured by different cameras. We present a computationally efficient and robust method for temporally calibrating video sequences from unsynchronized cameras. As opposed to expensive hardware-based synchronization methods, our algorithm is solely based on video processing. This algorithm is to match and align the object trajectories using the Longest Consecutive Common Subsequence, and thus to recover the frame offset between video sequences.

With the increasing number of cameras in the system, cost and flexibility are important factors to consider. The cost of each camera node increases with the increasing resolution of the image sensor. A possible way of employing low-cost low-resolution sensors to achieve higher resolution images is presented. In this system, four embedded cameras with low-resolution customized sensors are tiled in different arrangements. With the customized CMOS imager, we perform edge and motion detection on the focal plane, then stitch the four edge images together to get a higher-resolution edge map.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my advisor Professor Senem Velipasalar for her patience, wisdom, enthusiasm, perseverance, encouragement and understanding. She always conveys a strong motivation and passion in regard to research and academia. She sets a high standard as a researcher and a professor, that I look up to but may never be able to reach. She teaches me why "research" is called "research" — because we have to keep failing and "re"-search for new solutions. And every time when I fail, she encourages me not to give up and helps me to find a new way out. It would not have been possible to accomplish this dissertation without her tremendous help and guidance throughout the whole process. As both being female researchers, she always understands my concerns and supports my needs without any hesitation. I cannot find enough words to express my thankfulness.

I would like to thank Professors Mustafa Cenk Gursoy, Mehmet Can Vuran and Michael W. Hoffman for serving on my doctoral committee, and for their generous comments and suggestions to make my dissertation much better. I also thank them for the impressive and enjoyable lectures they teach. Their lectures have been great resources to broaden my mind and inspire my research.

I want to show my special thanks to Professor Sina Balkır for being the co-chair of my committee, for granting me this great opportunity of being a student in UNL at the beginning and helping me to finish my PhD study at the end. It is the best of luck to have the kind support of him.

I share the credit of my work with my fellow graduate students Mauricio Casares, Alvaro Pinto, Li He and Zhe Zhang. They have been working closely with me, and share all their time and thoughts unselfishly. I could not have achieved any of the accomplishments without their efforts. I also thank all my friends in Lincoln. Their friendship brings me a lot of joy and laughter, making my life in Lincoln unforgettable forever.

Last but not the least, I would like to thank my family — my grandfather Xin Wang, my grandmother Qihua Xie, my father Xuan Wang, my mother Guanzhen Li, my husband Xinwang Zhang and my son Shi Wang — for their unconditional love. Although my grandfather has leaved us and didn't have the chance to see this dissertation, I am sure he would be happy for what I have achieved today. This dissertation is dedicated to them.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

1.1 Background

Cameras are widely employed in military and commercial applications, and public transportation scenarios for purposes of surveillance, statistics gathering, and traffic flow monitoring. Nowadays, there are estimated 30 million surveillance cameras deployed in the United States. The scale and complexity of camera systems have been continuously increasing to have better coverage and accuracy. The large amount of video data generated by multi-camera systems necessitates automatic activity analysis.

1.1.1 Overview of Object Tracking

Instead of viewing the recorded videos and detecting objects by human eyes, automatically detecting foreground objects is the first step of automatic video analysis. Existing methods for foreground object detection can be generally classified into two categories: temporal difference methods[42, 43], and background subtraction methods[3, 47, 39, 14, 19, 38, 30, 40]. Temporal difference methods subtract two consecutive frames and then apply a threshold to the output. The pixels with a difference higher than the threshold are considered foreground pixels. These methods do not need to deal with the problem caused by the background changing over time. However, they cannot detect all the pixels of a moving object, as the overlapping part of the objects will be removed. On the other hand, background subtraction methods build a model of the background and subtract this from the current frame to detect the foreground pixels in the scene. The background model is usually required to be updated over time and adapt to changes in the environment. Most of the state-of-the-art tracking algorithms for fixed cameras employ the background subtraction methods [9].

After the foreground pixels are detected, they are represented in a binary frame, wherein the 1s indicate the foreground pixels and 0s indicate the background pixels. The foreground pixels need to be grouped into blobs with a certain connected component analysis method. Each blob corresponds to an object. For each detected object, a label is assigned and a tracker with a suitable representation of the object is formed. The purpose of the tracker is to generate the trajectory of the object by locating its position in every frame. The representation of the object normally contains the location of the object and the descriptors of some features, such as color, shape, size and texture. Then additional analysis can be performed based on the object trajectories to recognize their behaviors. In visual surveillance, this analysis often refers to detecting suspicious activities or events of interest [9].

Object tracking with multi-camera systems can be desired for varying scenarios, such as monitoring smaller areas with overlapping views, or tracking objects across wide areas with disjoint camera views. Using multiple cameras with overlapping views fuses the information of objects from different angles. This helps to resolve tracking difficulties caused by occlusion and crowdedness, and thus enhance the accuracy. With partially overlapping views, the consistent labels of the tracked objects can be handed off from one camera to another based on the geometric relations. Thus, a larger area can be monitored with multiple cooperative cameras. On the other hand, in wide-area tracking applications, it is not always realistic to assume that all cameras will have overlapping fields of view. Tracking across non-overlapping views presents more challenges due to lack of spatial continuity, and the difficulty of recovering geometric relations. To re-identify the objects and achieve consistent labeling across disjoint views, more features and more complex models are needed.

1.1.2 Object Tracking with Distributed Cameras

Most work on autonomous tracking of objects across multiple camera views has considered systems that rely on a back-end server to process video inputs from multiple cameras. Yuan et al. [48], Collins et al. [7, 8], Nguyen et al. [28], Lo et al. [26] and Krumm et al. [23] present systems where a server/controller performs the coordination and integration of the data from individual nodes. But, these systems have a bandwidth scaling problem, since the central server can quickly become overloaded with the aggregate sum of messages/requests from the nodes. Also, the server is a single point of failure for the whole system. In addition, server-based systems are not practical in many realistic environments, and have high installation costs. These problems of server-based systems necessitate the use of peer-to-peer (P2P) systems, where individual nodes communicate with each other without going through a centralized server.

More recently, multi-camera systems communicating in a P2P fashion have been introduced. Each camera node has its own processing power, and is able to detect and track objects by itself. Camera nodes cooperate to solve the consistent labeling problems, by exchanging object labels, and retrieve the locations of the occluded objects. Thus, the amount of data that need to be exchanged can be reduced significantly. Also each node has the ability to initiate a request, produce a reply, and make its own decisions. This removes the necessity of a central server and decreases the required communication bandwidth. Therefore, these systems are usually capable of performing object tracking in real time.

In most distributed multi-camera systems, each camera is attached to a different CPU and communication is performed over wired links [1, 15, 44]. These systems are assumed to be wall-powered and have bulky sizes. These affect the flexibility of camera installation as well as mobility, and incur significant costs, especially when more cameras are desired nowadays for wider areas and more complicated scenarios.

1.1.3 Object Tracking with Embedded Smart Cameras

With the advances in VLSI technology and embedded computing, smart cameras have been introduced, and it has now become viable to install many spatially-distributed cameras interconnected by wireless links. A smart camera not only captures images, but also includes a processor, memory and communication interface making it a stand-alone unit. Yet, many system- and algorithm-wise challenges remain to be addressed to have operational wireless smart-camera networks (Wi-SCaNs).

Embedded smart cameras have limited processing power, memory, energy and bandwidth. Although many methods have been introduced for robust foreground object detection and tracking, much less attention has been paid to the memory requirement and the portability of these algorithms to an embedded processor. Due to limited resources, most of the embedded smart camera systems [6, 20, 33] use relatively simple and sometimes less robust methods such as temporal difference and running average. Robust and feasible algorithms which require less memory, less computation and are optimized for the hardware architecture, need to be developed.

Another challenge is related to the wireless communication and data exchange between embedded cameras. Frequent transfer of large-sized data consumes more energy and incurs more communication delay. In many systems, communication is 100 to 1000 times more expensive in energy than computation [34]. Unlike wall-powered multiple cameras connected to CPUs, and communicating via Ethernet, wireless smart cameras have much less memory for storage, limited bandwidth and limited power supply. Due to these constraints, it is not viable to transfer or save every frame or every object trajectory. The tracking algorithms in each camera should be able to process and abstract the raw data as much as possible, and should only require minimal amount of information from other cameras.

Moreover, instead of transferring or saving every frame or every trajectory, there should be a mechanism to detect events of interest. Events of interest can be defined beforehand, and simpler events can be combined in a sequence to define semantically higher-level and composite events. Moreover, event scenarios can span multiple camera views, which make the definition of more complex events possible. Cameras communicate with each other about the portions of a scenario to detect an event that spans different camera views.

1.2 Related Work

1.2.1 Related Work on Multi-camera Multi-object Tracking

1.2.1.1 Multi-camera Tracking with Overlapping Camera Views

In a multi-camera setup, usually every single camera has the ability of tracking the objects individually. Cameras collaborate with each other to track objects consistently for longer periods of time, or resolve merge/split problems caused by objects interacting. Object tracking with partially overlapping camera views has been researched extensively in the last decade [44, 100, 76, 95, 79, 77, 69, 62, 71, 68, 8, 10, 71].

With partially overlapping camera views, the geometric relationship between the cameras can be recovered and utilized as an important cue. Converting all coordinates into a common 3D coordinate system is a popular approach to relate the objects across multiple cameras [76, 77, 8, 10, 23, 62, 119]. This approach requires the cameras to be fully calibrated, which is expensive and sometimes inconvenient. With all objects moving in a common 3D space, a tracking algorithm similar to 2D tracking can be adopted, such as Kalman filter [10, 76] or particle filter [77]. Most of the proposed systems have some degree of distributed processing, wherein each camera has the ability of object detection/tracking. But at the end, they still need a central processing unit to integrate the simplified data from the sensors, convert them into the common 3D space and make the decisions.

Blanco et al. [77] argue that 3D tracking based on partially erroneous 2D tracks are likely to fail when handling multiple-people interaction. To address this problem, they propose a Bayesian framework for combining 2D low-level cues from multiple cameras directly into the 3D world through 3D Particle Filters, instead of combining the tracking results from each camera. Dockstader et al. [10] propose a Kalman filter-based approach in 3D space, targeted at resolving the problem of occlusion and human interacting. The corrected state vectors from each view provide input observations to a Bayesian belief network, in the central processor. Then, a layer of Kalman filtering is employed to update the 3D state estimates. Collins et al. [8] also adopt a distributed-processing and central-decision-making framework. The central control unit uses a 3D geometric site model to integrate symbolic object trajectory information accumulated by each sensor node, and presents the results to the user on a map-based graphical user interface. The feasibility of real-time processing is demonstrated.

Another useful and reasonable assumption for most tracking scenarios is that all of the objects moving on the same planar ground. With the common ground plane assumption, a homography matrix between every two adjacent cameras can be computed, which is easier than full calibration [44, 79, 69]. Khan and Shah [69] use a planar homography constraint that combines foreground likelihood information from different views to resolve occlusions and determine ground plane locations of people. The homography constraint indicates that only the pixels of people's feet (on the ground) will consistently warp to foreground regions in every view. The field-of-view (FOV) lines [95] is also introduced by Khan and Shah so that the labels can be handed off when the objects enter other cameras' fields of view. In this way, the consistent labeling is achieved. Calderara et al. [79] propose a method to detect the FOV lines automatically. Kayumbi et al. [24] propose a registration algorithm based a statistical homography estimation. Then, a mosaic scene is generated with the registration of the trajectories from multiple camera views.

There are other works that use feature matching approaches to avoid camera calibration. Moller et al. [100] propose a calibration-free method that use color histogram matching based on the mean shift[12] tracking algorithm. But a coarse knowledge of the transfer points between two camera views is still required. Cai et al. [68] employ multivariate normal distributions to model the features, such as location, intensity, and geometric features. The correspondences are established using a set of feature points in a Bayesian probability framework. Chang et al. [16] also use Bayesian networks to fuse multiple features for matching subjects between consecutive frames and between multiple camera views. They divide the features into two groups: geometry-based modalities and recognition-based modalities. The former includes epipolar geometry, homography and landmark modalities; the latter includes apparent height and apparent color.

1.2.1.2 Object Tracking across Non-overlapping Camera Views

In wide-area tracking and wide-area surveillance applications, it is not always realistic to assume that all the cameras in the system will have overlapping fields of view. Tracking across disjoint camera views is a more challenging problem due to lack of spatial continuity, and thus having blind regions. In this case, recovering geometric relations may become difficult or infeasible in some scenarios. Feature-based matching is commonly used to solve the object re-identification problem. The cues that are used for object matching typically include appearance features, spatio-temporal evidence or the combination of these two types of information.

Color is one of the most commonly used appearance features. Color information is often represented by color histograms in the RGB or HSV color spaces. HSV is more robust to illumination changes due to its inherent properties. In the HSV color space, the luminance information is placed in the V channel and the chromaticity information is placed in the H (hue) and S (saturation) channels. The separation of the brightness information from the chromaticity reduces the effect of illumination change across difference camera views. In the RGB color space, each channel of Red, Green and Blue stores the brightness information and color information, which makes RGB histogram more vulnerable to different light condition or camera characteristics. To reduce this effect, Porikli [109] proposed a cross correlation model function for pair-wise inter-camera color calibration. The correlation matrix is computed from 1D RGB color histograms, and the model function is obtained from a minimum cost path traced within the matrix. The minimum cost path, which represents a mapping from one camera's color histogram to that of the other, is obtained by dynamic programming. This method could be computationally expensive. A more efficient way to map the color histograms from one camera to another is calculating the Brightness Transfer Function (BTF). Javed et al. [92] proposed a subspace-based BTF using probabilistic PCA to calculate the subspace of BTFs for a set of known correspondences. Their method relies on a large number of training data with a good range of clothing colors to give an accurate mean BTF (MBTF). Prosser et al. [111] proposed to use cumulative BTF (CBTF) instead of MBTF, which makes use of the available color information from a very sparse training set. A comparison of these two different BTFs can be found in [86], which demonstrates similar behaviors of the two methods when the simple association problem needs to be solved. Their experiments also show that appearance matching relying exclusively on color is not reliable when the scenario is more complicated than simple association, such as new object detection. Cheng et al. [81] proposed to cluster color into a subset of "major colors", named Major Color Spectrum Histogram Representation (MCSHR). The illumination variations are compensated by a cumulative histogram equalization. Again, only examples of simple object association are shown in their work. Jeong and Jaynes [93] use UY channels to build a 2D Gaussian Mixture Model and Affine Transformation to find the warping function (color transfer function) between the two models.

In addition to color information, some other appearance features can be combined for object re-identification. For example, height is used together with MCSHR for people tracking [96]. Texture or edge features are also useful for object matching. Cohen et al. [83] use a covariance matrix-based function integrating color and texture features (gradients) to represent each blob. This method requires a lot of data to be saved for post-processing: each blob's data for all blobs in all frames need to be saved. Then, the blobs are clustered into trajectories based on the appearance similarity. Cai et al. [78] present a human appearance model by using the region signatures centered at points on the edges of the human objects. The region signatures include the domain color representation and geometric constraints. Their proposed matching method is sequence-to-sequence matching, not frame-to-frame. Similar to [83], their algorithm is computationally expensive, and not intended to be used for real-time processing.

Spatio-temporal information is another important evidence to be considered for object re-identification. One way of using spatio-temporal constraints is predicting the objects' positions when they are in the blind region. With the assumption of linear motion model, a Kalman filter or a similar mechanism is employed [102, 82]. The positions of the objects could also be inferred based on a common ground assumption, which allows the warping between the cameras' views using a homography matrix [94]. In [108], expanded triangulation with motion constrains, which assumes linear motion of the objects, is employed for inferring the positions of the objects. The algorithm was only applied to the applications with small gaps between the cameras.

Another category of research also uses spatio-temporal information, but focuses on recovery of camera network topology, and not the object tracking. Rahimi et al. [112] recover the calibration parameters of the cameras and the targets' trajectories using MAP estimation. Huang et al. [88] use the transition time as the only evidence to infer the traffic flow status across non-overlapping views. A Gaussian Mixture Model (GMM) of the transition time is built without identifying the object correspondences explicitly. Niu et al. [104] use the appearance model to measure the similarity between disappearing and reappearing trajectories, then detect the possible link between the disjoint views, and estimate the transition time by the weighted cross correlated model. Finally, the non-overlapping network topology is recovered based on the estimated mutual information. Makris et al. [98] build up transition probability models based on transition time between the exits and entries. The topology of the camera networks is recovered by finding the maxima of the cross correlation functions.

To achieve more robust tracking results, spatio-temporal evidence is often combined with multiple appearance features. Javed et al. [91] combine color and travel time in a Bayesian formulation for object association. The best match is found by maximizing the posteriori. Kang et al. [94] use a spatio-temporal Joint Probability Data Association Filter (JPDAF) to formulate a joint probability model encoding objects' appearance and motion. Two non-overlapping camera views are warped in the reference of a moving camera view and merged into a mosaic. Thus, the object's motion can be inferred when it is in the gap between two stationary cameras. Chilgunde et al. [82] use position and size changes for object matching. With the assumption of constant velocity model, Kalman filter is used to predict the positions in the blind region. Monari et al. [102] intend to track objects in both overlapping and non-overlapping camera networks. They use 3D positions combined with CIE color space features to perform object association. The 3D positions in the blind region are predicted by a Kalman filter.

Huang and Russell [89] use multiple features for vehicle matching in a Bayesian formulation. Different from most of the related work, an association matrix is employed for finding the best assignments for multiple objects, which travel close to each other. They use multiple features, including lane information, size, color and travel time, to identify vehicles in a traffic application with a 2-camera setup. By adding additional elements of transition probabilities, the possibility of new and missing vehicles is also considered.

1.2.2 Related Work on Embedded Smart Cameras

Common computing platforms for smart cameras are FPGAs, digital signal processors (DSPs), and/or general purpose microprocessors [34]. Different smart camera systems have been introduced recently. Fleck et al. [18] present a network of smart cameras for tracking multiple people. They use commercial IP-based cameras, which consist of a CCD image sensor, a Xilinx FPGA for low-level image processing and a Motorola PowerPC CPU. The system uses color-based particle filters for tracking, but handoff of the objects is based upon a centralized model of the observed scene. Quaritsch et al. [32] employ smart cameras with multiple DSPs for data processing and a mobile agent framework for handling the handoff between cameras. Bramberger et al. [2] present another smart camera architecture developed from common off-the-shelf components, including a CMOS image sensor, multiple Texas Instruments TMS320C64x DSPs for image processing and an Intel XScale IXP425 for network processing. They provide two IP-based external communication: wired Ethernet and wireless GSM/GPRS. While this high-end platform provides sufficient capabilities for image processing, it requires an average power consumption of 35 W.

Wired or IP-based cameras have powerful processing capabilities and relatively high bandwidth for communication. However, they have high power consumption and are larger in size. Many embedded vision platforms, designed for wireless sensor networks, have been developed more recently [36, 20, 33, 17, 22, 13, 25]. The MeshEye platform [20] integrates two low-resolution image sensors and one VGA image sensor. It uses an ARM7 microcontroller with 55 MHz speed, and has 64 KB RAM and 256 KB flash memory. The Cyclops platform [33] is developed as a sister board for the Mica2 and MicaZ sensor boards, and has a 7.3 MHz-processor. However, in both of these platforms the processing power is still limited. The platform introduced by Kleihorst et al. [22] has an 84 MHz XETAL-II SIMD processor. It has higher resolution but has 128 KB of memory. The CMUcam2 [36] is a low-cost embedded camera with 75 MHz RISC processor and 384 KB SRAM. Due to the limited memory and processing power, only low-level image processing can be performed. The image processing algorithm cannot be modified after deployment since it is integrated in the firmware of the processor. Panoptes platform [17], which hosts a 206 MHz processor and 64 MB of RAM, is developed to generate medium-resolution video at high frame rates. It uses a USB web camera as a video sensor and 802.11 for wireless communications. This platform can perform more sophisticated processes in this high-end architecture, but the high energy consumption of the node limits the lifetime of a wireless application or necessitates wall-powered operation. Rinner et al. [35] presented a comparison of various smart camera platforms.

1.2.3 Related Work on Event Detection

Most of the previous work on event detection focused on detecting a finite set of specific and predefined events [54, 19, 57, 61, 63, 65, 70]. Stringa and Regazzoni [65], and Sacchi and Regazzoni [63] present surveillance systems for the detection of abandoned objects. The system proposed by Haritaoglu et al. [19] can recognize events such as depositing/removing an object or exchanging bags. Rota and Thonnat [61] use two sets of a priori information for video sequence interpretation: contextual information and predefined scenarios. Medioni et al. [57] analyze a set of predefined scenarios in video streams obtained from an airborne moving platform. Watanabe et al. [70] introduce a system for detecting events in which a person enters or leaves a room and/or an object appears or disappears. In addition to predefined events, research community has worked on unusual event or abnormal behavior detection [50, 52, 55, 59, 60, 38, 66, 72, 73].

However, an event detection system should be generic enough to detect broad range of events by giving users the flexibility to customize their own events with varying complexity. In other words, event definitions should not be predefined and hard-coded into the system, nor should they be limited in number. The system introduced by Black et al. [51] supports various SQL activity queries such as returning objects that have followed a certain path over a specific time interval. Yet, it does not discuss specification and detection of more complex events. Other approaches have been introduced that use event description or programming languages to enter the events of interest to the system [53, 64, 67]. Ivanov and Bobick [56] use a parser, which requires the interaction structure described to it in terms of stochastic context free grammar. Nevatia et al. [27, 58] introduce an event ontology for video event representation. The work in [27] mostly focuses on the event representation and markup languages but not the actual recognition of those events. As stated in [58], the definitions in their event representation language are similar to the function definitions of a computer programming language. These methods require familiarity with programming languages and, thus, event specification may require expert intervention. Moreover, although these methods provide some ability to define customized events, they remain limited in terms of event complexity. They mostly focus on detecting events on a single camera view, i.e., event definitions do not span multiple camera views. Also, the focus has not been on performing the event detection across the fields of view of multiple embedded smart cameras.

1.3 Contributions and Dissertation Outline

The novel contribution in this dissertation is divided into three parts. The first part presents a wireless embedded smart camera system for cooperative object tracking and detection of composite, semantically high-level and user-defined events spanning multiple partially overlapping camera views. The second part presents a probabilistic Petri-net based framework for object tracking across disjoint camera views, which utilizes multiple features and fuses them with adaptively updated weights. The third part includes other applications related to multi-camera systems. An efficient and robust algorithm for temporal calibration of unsynchronized cameras is proposed. In addition, a tiled low-cost low-power embedded system is presented, with the ability of focal plane image processing.

In the proposed wireless embedded smart camera system, each camera node has the ability to perform multi-object tracking individually. They only exchange data with the neighbors for the purpose of consistent labeling and event detection. Each camera node is a CITRIC mote [6] that consists of a camera board with a microprocessor, and a wireless mote. Lightweight and robust foreground detection and tracking algorithms are implemented and run on the microprocessor of the camera board. Chapter 2 describes the algorithms that run on each smart camera board for object detection and tracking, including the background subtraction algorithm designed for embedded cameras, a fast connected component labeling method and a lightweight tracking algorithm.

In Chapter 3, the approaches for cooperative object tracking and composite event detection are described. The cameras have partially overlapping fields of view. They exchange data in a P2P manner over wireless links to track objects with consistent labels, to update locations of occluded or lost objects, and also to inform other cameras about the occurrence of a primitive event in a composite event scenario. Even if an object is totally occluded in one camera view, its location can still be updated from other cameras. The protocols of peer-to-peer communications are explained in detail.

To address limited energy, limited memory and bandwidth issues, we detect *events of interest* so that interesting and important video portions and trajectories can be determined. In the presented system, events of interest can be defined beforehand by users, and primitive events can be combined in a time sequence to define composite, spatio-temporal and semantically higher-level events. Event scenarios can span multiple camera views. The complexity of event scenarios can be increased by increasing the number of primitive events, and/or the number of camera views they span.

After an event is detected, that portion of the live video can be saved or transferred. Another functionality provided is the ability to record the last portion of an event scenario from different camera views if possible. When a camera detects that a defined event scenario is occurring, it determines the other cameras that can see this region, if there is any. Then, it can send out a *Record* message addressed to those cameras so that they can start recording as well. This provides multiple views of the event of interest and, thus, additional information. Multiple real-time experiments are performed with two and three camera setups. Many different event scenarios are detected, which are composed of multiple primitives spanning different camera views.

Moreover, since energy is limited for embedded smart cameras, power consumption analysis of the camera systems is essential. The energy consumption and performance of the proposed system during different parts of processing a frame and during different message exchanges between camera nodes are analyzed, and presented in Chapter 4. The energy consumption analysis when tracking different numbers of objects, and when tracking different-sized objects are also presented. In addition, a more efficient blob forming algorithm is implemented, and compared it with the previous version to show the significant improvement in the processing time and, thus, energy consumption. To calculate the power consumption, the currents drawn by the embedded smart camera board for different scenarios are measured. We also compared the operating currents when transmitting and receiving different-sized packets. The results provide additional insight in terms of computation versus communication tradeoff and how to efficiently place the cameras in the scene. They also demonstrate and emphasize the importance of carefully designing a communication protocol and implementing lightweight algorithms in these resource-constrained environments.

In Chapters 5, 6 and 7, multiple object tracking with non-overlapping views are explored. In Chapter 5, we present a real-time distributed system with non-overlapping camera views. Although many methods have been developed that focus on building statistical or non-statistical models for object matching, much less attention has been paid to designing and implementing algorithms for real-time applications, and distributed processing. In this system, each camera is connected to a PC and the PCs communicate with each other through TCP/IP. The tracking algorithms are inherited from the previous system, and we combine multiple features to match objects across non-overlapping views. This is our first prototype system of real-time distributed object tracking with disjoint views.

In Chapter 6, a more sophisticated approach for object matching is proposed to improve the robustness of the multi-feature algorithm. Each feature is modeled more accurately and the weight of each feature is assigned adaptively based on their reliability. A common method to associate objects across disjoint camera views is using Bayesian formulation or maximum a posteriori (MAP) estimation. This type of algorithms normally find the best match by finding a best path through the graphic model or finding the object that maximizes the a posteriori probability. But as stated in Section 1.2.1, most of the methods only work for simple object association but have difficulty in distinguishing the new objects from the already observed ones. To account for this problem, we adopt a threshold-based method to match the "seen-before" objects as well as detect "never-seen-before" objects. A weighted sum of the similarity scores of multiple features is the criterion for object matching. The weights of features are learned automatically during training based on the reliability of each feature. If the similarity score obtained for a feature is in accordance with the overall matching outcome, this feature is considered to be reliable. To adapt to changes in the environment, these reliability values are updated online using the data from matched objects.

In Chapter 7, a distributed camera system for object tracking across disjoint camera views is presented. We incorporate *domain knowledge* to account for the information related to the environment and the system setup. Our system is capable of processing more complicated object tracking or event detection tasks with incorporating the domain knowledge, compared to the related work that only solves the object association problem. Considering the uncertainties caused by vision algorithms, a probabilistic result is preferred to a deterministic one. To incorporate the uncertainties of each stage (foreground detection, tracking and object matching) in a proper way, we employ a probabilistic Petri Net (pPN) based approach. In our system, the tracking process within a single camera and object matching across adjacent cameras are modeled by the pPN and a score of each object's tracking and matching result is yielded as the output of the pPN. Another advantage of employing the pPN is that the domain knowledge can be efficiently incorporated into the algorithm. When a rich set of domain knowledge is available, the pPN also helps to implement and control the work flow.

The proposed approach can be generalized to various surveillance applications involving disjoint camera views, such as indoor human tracking or outdoor human/vehicle tracking. In Chapter 7, we first present the wide-area tracking of vehicles as an example. This example shows how we fuse multiple features, train the parameters, and handle blind regions and "never-seen-before" objects. Then, a similar approach together with a different set of domain knowledge is employed for tracking people in another example with a disjoint camera setup. This example is more challenging, because unlike vehicles moving in certain lanes in fixed directions, people's routes are more diverse. These different examples and results illustrate how our framework can be applied to different scenarios with different domain knowledge. We also present the pPN for each scenario, where the domain knowledge is incorporated in the work flow.

In Chapter 8, a frame-level temporal calibration approach of unsynchronized cameras is presented. Temporal calibration is essential for all multi-camera systems. Instead of hardware-based synchronization, image processing-based recovery of the time offset is an easier and less expensive alternative. The proposed approach is based on finding the longest consecutive common subsequence (LCCS) between the corresponding trajectories from two camera views. Since this approach avoids the exhaustive search among all the trajectory points, the efficiency is improved significantly. Then, the offset between the two cameras can be recovered by finding the time difference between the two matched trajectories. A robust confidence check step is performed to select the most reliable offset.

Chapter 9 presents our work on image processing on the focal plane with customized camera sensors. In a large sensor network, the cost of each node becomes an important factor. Camera sensors with high resolution have larger silicon areas, more complex designs and thus higher costs. In this chapter, a possible way of employing low-cost low-resolution sensors to obtain higher resolution images is presented. The frames from four low resolution embedded smart cameras are tiled in two different arrangements. Edge and motion detection are performed on the focal plane, and the results can be tiled to a larger frame in the same way.

1.4 Publications

The above work has been published in prestigious and peer-reviewed journals and conference proceedings. The publications are listed below:

Peer-reviewed Published Journal Papers:

[J1] Youlu Wang, Senem Velipasalar, Mustafa Cenk Gursoy, "Distributed Wide-Area Multi-Object Tracking with Non-Overlapping Camera Views," *Springer Int'l Journal on Multimedia Tools and Applications*, pp. 1–33, Nov. 2012 (DOI 10.1007/s11042-012-1267-x).

[J2] Youlu Wang, Senem Velipasalar, Mauricio Casares, "Cooperative Object Tracking and Composite Event Detection With Wireless Embedded Smart Cameras," *IEEE Trans. on Image Processing*, vol. 19, no. 10, pp. 2614–2633, Oct. 2010.

Peer-reviewed Published Conference Papers:

[C1] Youlu Wang, Senem Velipasalar, Mustafa Cenk Gursoy, "Wide-area Multi-Object Tracking with Non-Overlapping Camera Views," *Proc. of the IEEE Int'l Conf. on Multimedia and Expo*, pp. 1–6, July 2011.

[C2] Youlu Wang, Li He, Senem Velipasalar, "Real-time Distributed Tracking with Non-Overlapping Cameras," *Proc. of the IEEE Int'l Conf. on Image Processing*, pp. 697–700, Sept. 2010.

[C3] Youlu Wang, Mauricio Casares, Senem Velipasalar, "Cooperative Object Tracking and Event Detection with Wireless Smart Cameras," *Proc. of the IEEE Int'l Conf. on Advanced Video and Signal Based Surveillance*, pp. 394–399, Sept. 2009.

[C4] Youlu Wang, Senem Velipasalar, Mauricio Casares, "Detection of Composite Events Spanning Multiple Camera Views with Wireless Embedded Smart Cameras," *Proc. of the ACM/IEEE Int'l Conf. on Distributed Smart Cameras*, pp. 1–8, Aug. 2009.

[C5] Youlu Wang, Senem Velipasalar, "Frame-level Temporal Calibration of Unsynchronized Cameras by Using Longest Consecutive Common Subsequence," *Proc.of the IEEE Int'l Conf. on Acoustics, Speech and Signal Processing*, pp. 813–816, Apr. 2009.

Our work on the wireless embedded smart camera system, described in Chapter 2, Chapter 3 and Chapter 4, is published in part in [J2], [C4] and [C3]. The real-time object tracking system with non-overalpping camera views, that is presented in Chapter 5, is published in [C2]. [J1] and [C1] include the multi-feature object matching algorithm and Petri-Net based framework for object tracking across disjoint views, that are described in Chapter 6 and Chapter 7, respectively. [C5] presents the work on frame-level temporal calibration in Chapter 8.

Part II

Object Tracking and Event Detection with Wireless Embedded Smart Cameras

Chapter 2

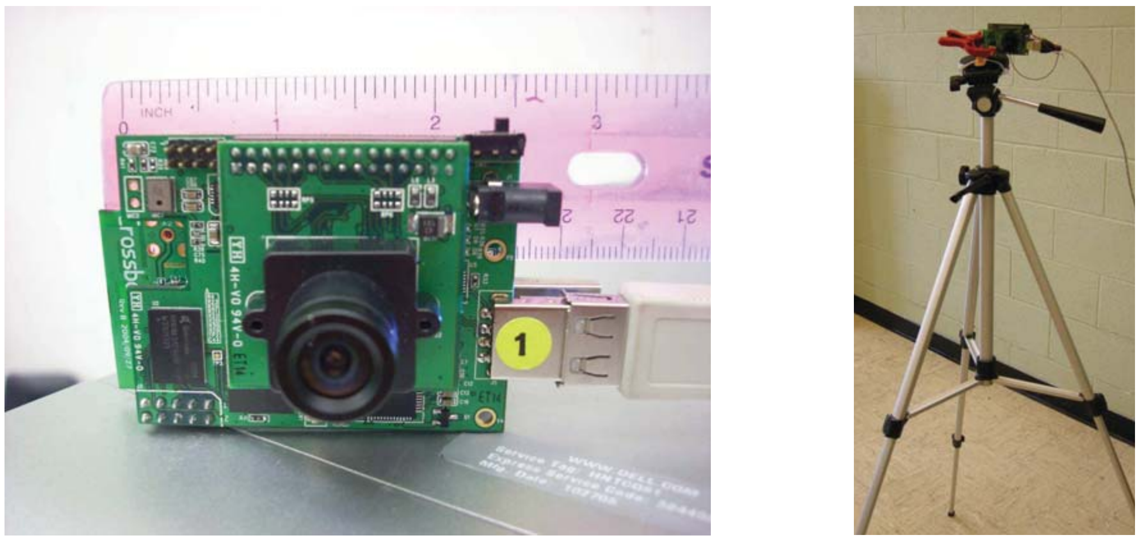
Embedded Smart Cameras and Lightweight Vision Algorithms

Due to the limited processing power and limited memory of the embedded smart cameras, it is critical to design lightweight computer vision algorithms that require less computation and less memory, and consume less power. We designed and implemented lightweight algorithms on our smart camera boards. All the processing, which includes foreground detection, morphological operations, connected component labeling, blob forming, object tracking and event detection, is done onboard on the microprocessor of the smart camera unit. With the attached wireless motes, the camera nodes communicate with each other in a peer-to-peer manner, which removes the necessity of a central controller.

In this chapter, we firstly introduce the embedded camera boards and the attached wireless motes that are employed in our system. Then, the algorithms running on each individual camera are described.

2.1 The Wireless Embedded Smart Camera Platform

The wireless embedded smart camera platform employed in our system is a CITRIC mote [6]. It consists of a camera board and a wireless mote, and is shown in Figure 2.1. The camera board captures video frames by a CMOS image sensor, and then processes them. An embedded Linux system runs on the camera board. Each camera board connects to a wireless mote via a serial port.



(a) (b)

Figure 2.1: The wireless embedded smart camera platform employed in the proposed system.

2.1.1 CITRIC: The Camera Board

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.

The image sensor of the camera board is an 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 from Marvell with a maximum speed of 624 MHz, 256 KB of internal SRAM and a wireless MMX coprocessor to accelerate multimedia operations. It is capable of working in low voltage and low frequency, as low as 0.85 V and 13 MHz, to achieve low power consumption. The typical CPU frequencies that the CITRIC platform supports are $208, 312, 416, 520$ MHz. Besides the internal memory of the microprocessor, the PXA270 is connected to 64 MB of SDRAM and 16 MB of NOR FLASH. 64 MB is the largest size of the Single Data Rate (SDR) mobile SDRAM components natively supported by the PXA270 currently available in the market [6].

All of our experiments were run in real-time with QVGA (320×240) resolution. All the algorithms run on the embedded Linux system ported onto the PXA270 microprocessor. The embedded Linux system includes the JPEG compression library. Since we only store detected events of interest, with this compressing functionality, 64 MB SDRAM provides enough space for our experiments. All the programming data and saved results are transferred by the UART port of the PXA270. A USB-to-UART bridge controller is connected between the PXA270 UART port and USB port on a PC. The camera board can be powered by a USB port from a PC, or four AA batteries.

2.1.2 TelosB: The Wireless Mote

The wireless mote connected to the camera board is a TelosB mote from Crossbow Technology. The TelosB uses a Texas Instruments MSP430 microcontroller and Chipcon CC2420 IEEE 802.15.4-compliant radio, both for low-power operation [6].

The Texas Instruments MSP430 MCU operates at 8MHz with 10KB RAM. The TelosB is a commercial off-the-shelf mote loaded with TinyOS/NesC and multi-hopping communication protocols. Thus, we can easily utilize them to perform wireless communication and exchange data between camera nodes. Since the maximum data rate of the 802.15.4 is 250kbps, it is not viable to transfer whole video frames between camera nodes. Also, due to high power consumption of wireless communication and small buffer size of the mote, transferring large-sized packets should be avoided. We need to buffer and transfer as few and as small-sized packets as possible. We designed and implemented our algorithms and the communication protocol by taking this fact into account.

We focus on the lightweight algorithms, their energy requirement, P2P event detection and the application-layer protocol, and use the preloaded lower layer protocols in the TelosB mote. When TelosB is idle, no serial communication is performed between the camera board and the wireless mote. When the camera needs necessary information from other cameras, and needs to exchange data, only then it performs serial communication with the wireless mote to send and receive packets.

2.2 Foreground Detection

Many methods have been introduced for background subtraction and foreground object detection [14, 19, 21, 30, 31, 38, 47, 49]. However, most of these methods have been developed and tested on PCs instead of embedded smart cameras, and much less attention has been paid to the memory requirement and the portability of these algorithms to an embedded platform. Lighting variations and non-static backgrounds make the foreground detection problem even more challenging, since we are interested only in salient motion in tracking applications. We need to separate cases of uninteresting motion, such as swaying trees and water fountains, from the salient motion regions. The necessity of handling these challenging cases increases the algorithm complexity, and thus memory requirements. However, due to resource constraints, most of the embedded smart camera systems [6, 20, 33] use relatively simpler and sometimes less robust methods, such as temporal difference and running average, for foreground detection. The outputs are not robust enough for reliable tracking.

An efficient algorithm for salient foreground detection is proposed in [5]. This algorithm is designed for embedded systems, and takes into account the memory requirements as well as the computational complexity. It is highly robust against lighting variations and non-static backgrounds including scenes with swaying trees, water fountains and rain. It provides better or comparable foreground detection results, and requires the least amount of memory when compared with the state-of-the-art background subtraction algorithms. In addition, this algorithm avoids floating point computations. This provides additional advantage when running it on embedded smart cameras, since most of the microprocessors do not integrate a floating point unit. We implemented both this algorithm and the adaptive Mixture of Gaussians (MoG) [38] on our smart camera board to compare their performances. The MoG algorithm runs at 1.6 frames per second (fps), and our lightweight algorithm runs at 12.5 fps, when there is one foreground object in the scene.

This algorithm employs a temporal difference method until a complete background model is built. It differentiates between salient and non-salient motion based on the history of a pixel's location, and by considering neighborhood information. At each frame, each pixel is classified either as a background or a foreground pixel, and its state is set to be 0 or 1, respectively. For a pixel at location (i, j), a counter $h(i, j)$ holds the number of changes in the state of this pixel during the last 100 frames, i. e. the counter $h(i, j)$ keeps the number of times a pixel's state changes from 0 to 1 or vice versa. The stability of a pixel at location (i, j) is determined by this counter $h(i, j)$. The motivation is that the lower the value of $h(i, j)$, the more stable and reliable that location is, or vice versa. Thus, rather than saving many values for each pixel location, such as averages for three color values, multiple Gaussian distribution means and variances, multiple codewords with multiple entries, only the h counter and background model need to be saved.

This method selectively updates the background model with an automatically adaptive rate. If a pixel location is determined to be consistently reliable, the value of this pixel is incorporated to the background model with a higher weight. Also, the number of memory accesses and instructions are adaptive, and are decreased even more depending upon the amount of activity in the scene and on a pixels history. The algorithm requires 6.25-byte memory for the data saved for each pixel, whereas original mixture of Gaussians [38], Eigenbackground [29] and Codebook [21] methods require 32, 28 and 91 bytes per pixel, respectively. We imported this algorithm to our embedded smart camera boards to perform foreground detection.

2.3 Fast Blob Forming and Connected Component Labeling

After performing foreground detection, a binary image is obtained in which white and black pixels represent the foreground and background pixels, respectively. This binary image usually contains some white pixels that do not correspond to salient motions, but are caused by sensing errors, changing lighting conditions, non-salient motions or other interferences, instead. These pixels will be referred to as noise pixels. To remove noise pixels from foreground and then group the foreground pixels into blobs, the conventional method is performing morphological operations followed by a connected component labeling algorithm.

First, we implemented classic morphological operations using a 5×5 DISK shape structuring element [37] on the microprocessor of the smart cameras. An *opening* operation is performed, followed by a *closing* operation to remove the noise and fill the holes. Then, we perform connected component labeling using union-find structure [37]. The binary image is searched row-by-row three times to form foreground blobs. In Chapter 4, the energy consumption when using this multi-pass connected component labeling algorithm is presented.

It is shown that this algorithm runs very slow and consumes more energy on the camera boards.

To reduce the processing time and, thus, the energy consumption, we designed and implemented another algorithm that uses a combined and more efficient approach to accomplish noise removal and blob forming in a single pass. At the beginning, all pixels in the binary frame are marked as *unvisited*. The algorithm starts searching through every pixel in the binary image. Once an unvisited foreground pixel is reached, a search is performed around this pixel to grow a blob until no white pixels remain connected to the previously found ones. Every searched pixel is then marked as *visited*. A threshold is predefined for the minimum blob size. If the number of pixels in a blob is smaller than the threshold, it is removed from the foreground to eliminate noise pixels, by setting all the pixels in the blob to 0.

If a blobs size is greater than the size threshold, and it is the first blob formed in this frame, this blob is saved. Then, the search continues to find next unvisited foreground pixel to form new blobs. Once a new blob is formed, the distance between this blob and each of the saved blobs is calculated. A distance threshold is employed to determine if this new blob is a fragment of a bigger blob, and if it should be grouped together with one of the previously found blobs. If the calculated distance between the new blob and one of the saved blobs is smaller than the threshold, this new blob is grouped together with the saved one and their pixels get the same label. The previous steps are repeated until no unvisited foreground pixels remain. Thus, noise removal and connected component labeling are accomplished in a single pass. In Chapter 4, a comparison of the currents drawn and energy consumption between the classic multi-pass approach and our single-pass efficient approach is presented. The processing time and energy consumption are significantly reduced using the proposed approach.

2.4 Object Tracking Algorithm

Tracking multiple objects becomes more challenging when tracking needs to be performed on an embedded smart camera with limited processing power, energy and memory. In [44], a P2P multi-camera system is presented wherein each camera is attached to a different CPU. This system employs efficient and robust algorithms for tracking and consistent labeling. Each camera performs its own tracking and keeps its own trajectories for each target object, which provides fault tolerance. And this system is also fully distributed by removing the necessity for a central controller. Although it is developed in PCs, it is feasible to run in the wireless embedded cameras because of its efficiency and sparse message traffic. We started with this algorithm, optimized it, and implemented it on the microprocessor of our camera boards.

After performing foreground detection and connected component analysis, a rectangular bounding box is formed around each foreground blob. When a new foreground blob is detected within the camera view, a new tracker is created, and the intensity histogram of the foreground object is built and saved as the *model histogram* of the tracker. The tracker also holds the coordinates of the bounding box of this object, and a label that will be used during tracking.

At each frame, the trackers are matched to detected foreground blobs by using a computationally efficient blob tracker which uses a matching criterion based on the bounding box intersection and the Bhattacharyya coefficient [12]. The Bhattacharya coefficient is derived from the sample data by using:

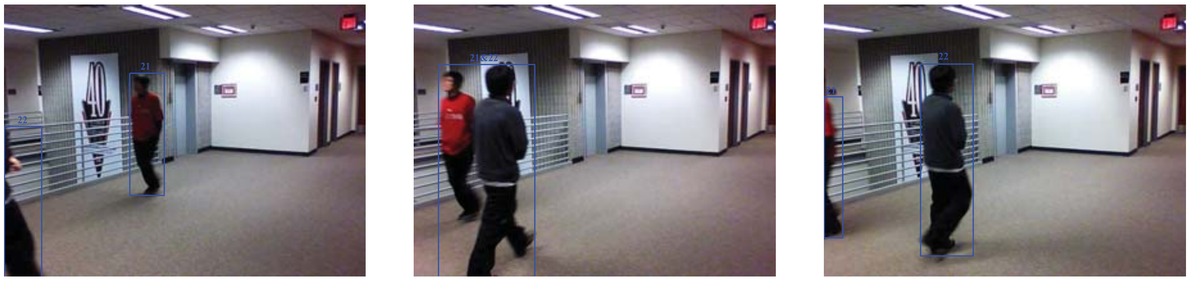
$$\hat{\rho}(y) \equiv \rho[\hat{\mathbf{p}}(\mathbf{y}), \hat{\mathbf{q}}] = \sum_{u=1}^{m} \sqrt{\hat{p}_u(\mathbf{y}), \hat{q}_u} \tag{2.1}$$

where $\hat{\mathbf{q}} = \{\hat{q}_u\}_{u=1\ldots m}$, and $\hat{\mathbf{p}}(\mathbf{y}) = \{\hat{p}_u(\mathbf{y})\}_{u=1\ldots m}$ are the probabilities estimated from the

m-bin histogram of the model in the tracker and the candidate blobs, respectively. These probabilities are estimated by normalizing the intensity histogram of the blob or the model histogram of the tracker. If the bounding box of a foreground blob intersects with that of the tracker, the Bhattacharya coefficient between the model histogram of the tracker and the histogram of the foreground blob is calculated by using Eq. (2.1). The tracker is assigned to the foreground blob which results in the highest Bhattacharya coefficient and whose resultant Bhattacharya coefficient is higher than a threshold. Thus the bounding box of the tracker is updated using the coordinates of the matched blob. The Bhattacharya coefficient with which the tracker is matched to its object is called the *similarity coefficient*. If the similarity coefficient is greater than a predefined distribution update threshold, the model histogram of the tracker is updated to be the intensity histogram of the foreground blob to which it is matched.

Based on this matching criterion, if objects merge, multiple trackers are matched to one foreground blob, as shown in Figure 2.2 (b). The trackers that are matched to the same foreground blob are put into a merge state, and in this state their model histograms are not updated. Here we use a variable to record the merge state, where 1 indicates the tracker is in merge state. Their bounding boxes are updated by the coordinates of the merged blob. When objects split from each other, trackers are matched to their objects based on the bounding box intersection and Bhattacharya coefficient mentioned above. The variable for merge state is reset to 0.

Figure 2.2 shows an example of resolving a merge. In Figure 2.2 (a), there are two objects in the view with labels 21 and 22, respectively. They are detected to be merging into one blob as shown in Figure 2.2 (b). In the merge state, both of the labels 21 and 22 are displayed on the blob. They split later and are matched to their correct trackers in Figure 2.2 (c). However, there may be some unfavorable cases that they are not matched to their correct trackers after they split. This may happen if their appearances are very similar.



(a) (b) (c)

Figure 2.2: Example of resolving a merge.

Then an additional checking step need to be performed to differentiate them. The details of the additional checking step can be found in [44].

One advantage of this algorithm is that it requires very little memory and computation for keeping and matching the trackers. Table 2.1 shows the data that are contained in a tracker. In a tracker, we just need an integer that represents the label, an integer that indicates if the object is in merge state, four integers $(x_{min}, x_{max}, y_{min}, y_{max})$ for the coordinates of the bounding box, and a 32-bin model histogram. Although the normalized histogram should contain 32 fractions, we scale it by 10000 and round it into integers for faster processing. Thus, there are totally 38 integers saved in a tracker. For a 32-bit microprocessor, the size of an integer is 4 bytes, and thus, the size of a tracker will be 152 bytes.

Another advantage is that this tracking algorithm allows for sparse message traffic by handling the cases of merging and splitting within a single camera view without sending request messages to other cameras. The cameras only need to request additional information from other cameras in consistent labeling and lost labeling scenarios. This will be introduced in the following chapter.

Table 2.1: Data contained in a tracker

Tracker	
label	4 *bytes*
merge state	4 *bytes*
x_{min}	4 *bytes*
x_{max}	4 *bytes*
y_{min}	4 *bytes*
y_{max}	4 *bytes*
model histogram	32×4 *bytes*

2.5 Conclusions

In this chapter, the embedded smart cameras with the attached wireless motes have been introduced. These CITRIC cameras are equipped with the state-of-the-art low-power microprocessors and can be powered by batteries. An embedded Linux system runs on the camera boards. The wireless motes are 802.15.4-compliant and also have very low-power consumption. They have the maximum data rate of 250 kbps.

Lightweight and robust algorithms are designed and implemented on the embedded cameras. The foreground detection algorithm separates the non-salient motions from the salient motions by taking into account the stability and reliability of the pixels. As opposed to most other background subtraction algorithms that require to save many variables for each pixel, very little memory is required by this algorithm. Moreover, this algorithm avoids complex computations and floating point processing, which makes it suitable for running embedded platforms.

After the foreground pixels are found, an efficient single-pass approach for connected component labeling is employed. This approach reduces the processing time and energy consumption compared to the classic approach based on the morphological operations.

Every moving object is assigned a unique label and a tracker is built to contain the descriptors of this object. In each incoming frame, a matching process is performed between the trackers and the blobs. The matched trackers are updated based on the information of the corresponding blobs. If there are unmatched blobs, new trackers are created for them. This algorithm can also successfully resolve the merge/split problem of the objects without any input from other cameras, which helps to reduce the communication load.

Chapter 3

Cooperative Object Tracking and Event Detection

In order to solve the consistent labeling problem, we employ the Field of View (FOV) lines. When a new object enters the scene, the camera determines if this object is already in other cameras' view by checking the relation between the object's location and the FOV lines of other cameras. If it is determined that this object is already being tracked by other cameras, this camera sends a request message addressed to those cameras to request the label of the object.

A camera also uses the FOV lines to determine to which cameras the request messages should be addressed. For instance, if the camera loses an object due to occlusion, it first figures out which other camera(s) can see this object by using the FOV lines. It then sends a request to retrieve and update the location of the occluded object.

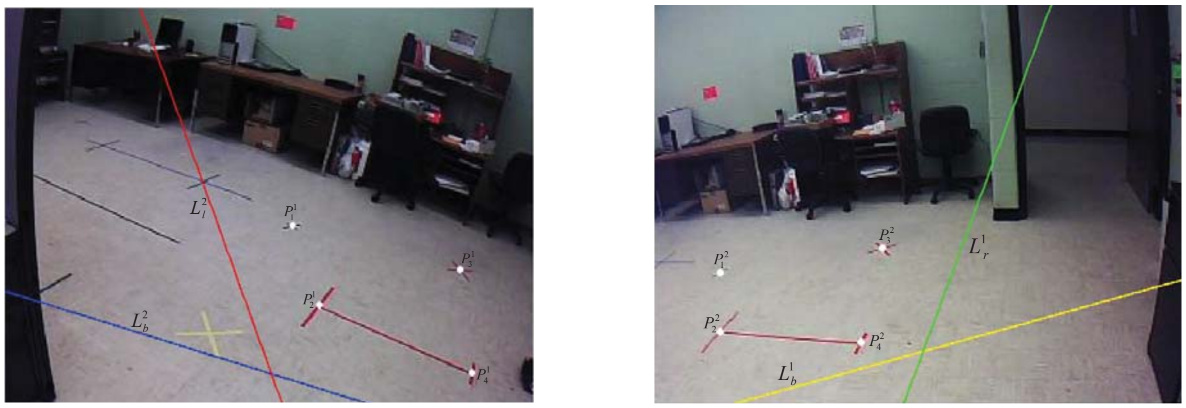
We name the above cases of communication as *New_Label* case and *Lost_Label* case, respectively. Other than these types of communications, we also define composite events spanning multiple camera views, with primitive events that are defined on different views. Informing other cameras of the occurrences of the primitives is another function fulfilled by the P2P wireless communications.

In this chapter, the approaches for recovering the FOV lines and achieving consistent labeling are described. Also, the details of the composite event definition and detection are explained. The application layer communication protocol that is designed for this system is presented.

3.1 Consistent Labeling

3.1.1 Recovery of FOV Lines

The FOV lines were introduced by Khan and Shah [95]. We recover the FOV lines off-line as described in [44]. As stated in Chapter 1.2.1, homography is a commonly-used constraint for consistent labeling, with the assumption of a common ground plane. Given four pairs of corresponding points on the same plane, the homography matrix can be computed using the Direct Linear Transformation (DLT) algorithm [142].



(a) Camera 1 (b) Camera 2

Figure 3.1: Recovery of the FOV lines

Figure. 3.1 shows an example of the corresponding points that are chosen for homography estimation in a pair of camera views. These points are denoted as $\mathbf{p}^1 = \{p_1^1, p_2^1, p_3^1, p_4^1\}$ and $\mathbf{p}^2 = \{p_1^2, p_2^2, p_3^2, p_4^2\}$ in Camera 1 and Camera 2, respectively. For DLT computation, the homogeneous coordinates are used. The homogeneous coordinates of the points are in the form of $\vec{\mathbf{p}}_k^i = (x_k^i, y_k^i, 1)^T$, where $i \in \{1, 2\}$, $k \in \{1, \dots, 4\}$. The computed homography between the ground planes in Camera 1 and Camera 2 is denoted as H, which is a 3×3 matrix. With H, any point $\mathbf{p}_a^1$ on the ground plane of Camera 1 and its corresponding point $\mathbf{p}_a^2$ in Camera 2 satisfy:

$$\vec{\mathbf{p}}_a^2 \cong H\vec{\mathbf{p}}_a^1 \tag{3.1}$$

$$\vec{\mathbf{p}}_a^1 \cong H^{-1}\vec{\mathbf{p}}_a^2 \tag{3.2}$$

In Eq.(3.1), to convert the homogeneous coordinates $\vec{\mathbf{p}}_a^2$ to 2D coordinates $\mathbf{p}_a^2$, $\vec{\mathbf{p}}_a^2$ needs to be normalized so that its third entry equals to 1. Then the first two entries of the normalized $\vec{\mathbf{p}}_a^2$ is $\mathbf{p}_a^2$. The same operation is needed to retrieve $\mathbf{p}_a^1$ from Eq.(3.2).

Given that two points define a straight line, the FOV lines of Camera 1 in Camera 2 can be determined by converting two points on each boundary of one camera to the corresponding points in the view of the other camera, using Eq. (3.1) and Eq. (3.2). A boundary of a camera's view is denotes as s, where $s \in \{l, r, t, b\}$. Since H is the homography for the ground plane, only the boundaries on the ground need to be converted. And four boundaries are not necessarily all visible in the view. Figure 3.1 shows the recovered FOV lines in the two camera views. Only two boundaries of each camera have their correspondences in the other view. In Figure 3.1 (a), the red line (L_l^2) and the blue line (L_b^2) correspond to the left boundary and bottom boundary of the ground plane in Camera 2, respectively. And in Figure 3.1 (b), the green line (L_r^1) and the yellow line (L_b^1) correspond to the right boundary and bottom boundary of the ground plane in Camera 1, respectively.

3.1.2 Checking the Visibility of Objects

When a new foreground object enters a camera's FOV, the camera first checks if this object can be seen by any other cameras by employing the FOV lines. The midpoint of the bottom line of an object's bounding box is considered as its location. The location (x_o^i, y_o^i) of an object in Camera i is

$$\begin{aligned} x_o^i &= \frac{x_{o,min}^i + x_{o,max}^i}{2} \\ y_o^i &= y_{o,max}^i \end{aligned}$$

where $x_{o,min}^i$ and $x_{o,max}^i$ are the minimum and maximum x coordinates, and $y_{o,max}^i$ is the maximum y coordinate of the bounding box. To check if this location is in the FOV of Camera j, one of the four points that are used for homography computation will be employed. Let us assume that all the FOV lines of Camera j in Camera i have the form as $y = sx + c$. If this point lies on the visible side of all the FOV lines, then for every FOV line, $sign(y_o^i - sx_o^i - c) = sign(y_k^i - sx_k^i - c)$, which means these two points are on the same side of this line, where (x_k^i, y_k^i) are the coordinates of any one of the points in $\mathbf{p}^i$. Then, it is deduced that this object is visible by Camera j. In this case, a request for a label addressed to Camera j will be sent out by wireless communication to achieve consistent labeling. This message will include the coordinates of this object (x_o^i, y_o^i). Meanwhile, Camera i will assign a temporary label to this object, waiting for its correct label to be sent back. When Camera j receives this request, it converts the received coordinates (x_o^i, y_o^i) into its own coordinates (x_o^j, y_o^j), using Eq. (3.1) or Eq. (3.2), accordingly. Then Camera j checks all the trackers it has, finds the closest tracker whose distance from the object is within a threshold, and returns the label of this tracker to the requester. Camera i will then replace the temporary label by the received label.

As stated previously, a camera also uses a similar approach to retrieve the location of a lost or occluded object. When an object is occluded in the scene, the camera checks if its last location is in the FOV of another camera. If it is, it sends out a request message containing the label of the lost object to that camera. When the other camera receives the label, it finds the label in its trackers, and sends back the coordinates of the tracker to the requester. The requesting camera needs to convert the received coordinates into its own coordinates using the homography matrix, before updating the location of the tracker.

Details of the packets that are sent out for the *New_Label* and *Lost_ Label* requests and replies are described in Section 3.3.

3.2 Composite and Spatio-temporal Event Detection

Object tracking is widely employed in visual surveillance systems. However, object tracking by itself is not sufficient for most applications. Tracking results should be analyzed to detect occurrences of events of interest. For instance, in the surveillance scenarios, the main interest is detecting instances of events such as objects entering a prohibited region or a person entering through an exit-only door. Detection of events of interest is especially important in wireless smart camera systems, since it is not possible to transfer or save every frame or every object trajectory.

We present a wireless embedded smart camera system that can detect composite, spatio-temporal and semantically higher-level events. Event scenarios are defined beforehand and they can span multiple camera views. More complicated events can be built by using simpler basic building blocks [27]. We define semantically higher-level events by using the building blocks, which are henceforth called the primitive events or simply primitives. Primitive events are connected to each other by a sequence operator, since the most fundamental relation among component events is one of sequence [27]. The current primitive events in the presented system are motion detection (MD), tripwire crossing (TW) and abandoned object (AO). MD primitive is defined by specifying a rectangular region of interest (ROI) on a camera view. If motion is detected in this region, it will be concluded that MD primitive has occurred. TW primitive is defined by specifying a line, and a direction on a camera view. If an object crosses this line in the specified direction, it will be concluded that TW primitive has occurred. AO primitive is defined by specifying a rectangular region and the waiting time before considering the object abandoned.

Multiple primitives can be defined on one or more camera views and can be connected to each other by a sequence operator to define higher-level events spanning different camera views. The desired time interval between each primitive can also be specified. For instance, let the event scenario of interest be detecting a person entering into the scene in the first camera view, and *then* intruding a region defined in the second camera view in a time interval of m seconds. The entry of a person can be detected by defining a TW primitive (E_1) at the entrance watched by the first camera. The intrusion of the prohibited region can be detected by defining a MD primitive (E_2) on the second camera view. These events are then connected by a time sequence operator so that E_1 happens first and then E_2 happens in less than m seconds. The complexity of event scenarios can be increased by increasing the number of primitives on a camera view, and/or the number of camera views they span.

In the proposed system, different cameras have partially overlapping fields of view, but the primitive events can be defined in the non-overlapping regions. Thus, cameras communicate with each other about the portions of a scenario to detect an event that spans different camera views.

The definition of each primitive event, for instance the TW or the ROI, is saved on the camera that is responsible for detecting this primitive. Each camera also has an array (`CamID`) containing the camera IDs in the same order as their primitives are in the defined composite event scenario. The first camera in this array is responsible for detecting the first primitive event in the sequence. If the first primitive event occurs in its view, this camera will send a

Primitive_Occurred message addressed to the next camera in the `CamID` array. This message includes the label of the object performing this event. The details of communication and packet contents are described in Section 3.3. After receiving this message, the next camera in the `CamID` list will be checking if the next primitive event is performed by the same object. The cameras in the `CamID` list will only detect a primitive and inform the next camera in the list, when they are informed of the occurrence of the previous primitive. When all of the primitive events in the defined scenario are performed by the same object, the entire scenario occurs and is detected.

Let an event scenario be composed of three primitive events, E_1, E_2 and E_3. Let E_1, E_2 and E_3 be defined in the views of the first, second and third camera, respectively. Also, let all events be defined in the non-overlapping regions of the FOVs, i.e. only the first camera can see the region where E_1 is defined, only the second camera can see the region where E_2 is defined, and only the third camera can see the region where E_3 is defined. When the first camera detects that E_1 has occurred, it will broadcast a message addressed to the second camera to inform the occurrence of E_1. This message also contains the label L of the object involved in E_1 so that the camera which is responsible for detecting E_2 can check if the object involved in E_2 has the same label L. If E_2 occurs in the second camera view within a pre-defined time interval after it receives the message from the first camera, and if the object performing E_2 has label L, the second camera will broadcast a message addressed to the third camera. If the third camera detects that E_3 has occurred, and the label of the object involved is L, it will declare that the defined event scenario has occurred.

In many cases, the second camera will receive the message, informing the occurrence of a primitive, and including the label L of the object, before this object actually enters its own FOV. Thus, when it receives the *Primitive_Occurred* message, it will save the object's label L. When a new object enters its view, FOV lines will be used to determine which camera can see this object, and a message will be sent to that camera to retrieve the label of the object. If the retrieved label is L, then this is the object that should be performing the following primitives.

Another important point to note is the potential race conditions when detecting a composite event. Consider a scenario where the first primitive is detected on the first camera view, and it broadcasts the *Primitive_Occurred* message containing label L of the object. Let this object enter the FOV of the second camera, which is responsible with detecting the second primitive in the sequence. This object will be given a temporary label T until the second camera gets the correct label for this object from another camera. If this object performs the second primitive before the correct label is received, i.e., while it still has the temporary label T, this might cause the compound event to be missed. To avoid this, the second camera can save the instances of primitive events performed by objects with temporary labels. Once the correct label for an object with temporary label T is received, the camera can associate it with the primitive event detected previously.

In an embedded smart camera, it is not possible to save or transfer all the captured frames or every object trajectory due to limited resources. Thus, by detecting events of interest, we can save only those portions of video where the defined event scenario occurs, and/or we can save or transfer only the trajectories involved in an event scenario. For instance, when an object is detected entering a prohibited region, only the frames where the object is crossing the ROI are saved on the corresponding camera board. Some example scenarios, and the saved frames are presented in Section 4.2.

3.3 Communication between Cameras

The embedded smart cameras in our system communicate in a P2P manner over wireless links. Compared to server-based approaches, this provides important advantages in terms of bandwidth. Also, thanks to the P2P communication, cameras do not need to send the state of each tracker to a centralized location at each frame. This decreases the number of messages that need to be sent around significantly.

Since sending large-sized packets requires more energy, and incurs more communication delay, it is very important to carefully design when to communicate and what to communicate, and to employ algorithms that do not require transfer of large data between cameras. In the following subsections, we describe how the decisions about when to communicate, with whom to communicate and what to communicate are made. We also describe the details of the message packets sent between the cameras.

3.3.1 Packet Formats

We use the default packet formats provided by CITRIC developers for serial and wireless communication. Serial packet format, shown in Figure 3.2, contains two 1-byte delimiters, a 3-byte serial packet header, a 2-byte footer and an Active Message (AM) packet. The AM Packet shown in Figure 3.2 is the packet format for wireless communication with a 7-byte header and no footer. When a TelosB receives a serial packet from the camera board, it can just easily take the AM packet and send it out, without further encapsulating. And each AM packet payload is filled by a Camera Board (CB) packet, which contains a 3-byte header and the CB payload. The first byte of the CB header indicates the type of this packet and is divided into 2 halves — values $[0 \sim 127]$ are restricted types used by the CITRIC API, while values $[128 \sim 255]$ are user values available for application specific data. We define our application message types in this field. Every type of the message that is described in Section 3.3.5 is assigned a unique number in the range of $[128 \sim 255]$.The other two bytes in the CB header are the originating source mote ID on a multi-hop network. This will be equal to the source ID in the AM packet header on a single hop network.

Once a packet is sent, TelosB will wait for an acknowledgement from the receiver. If

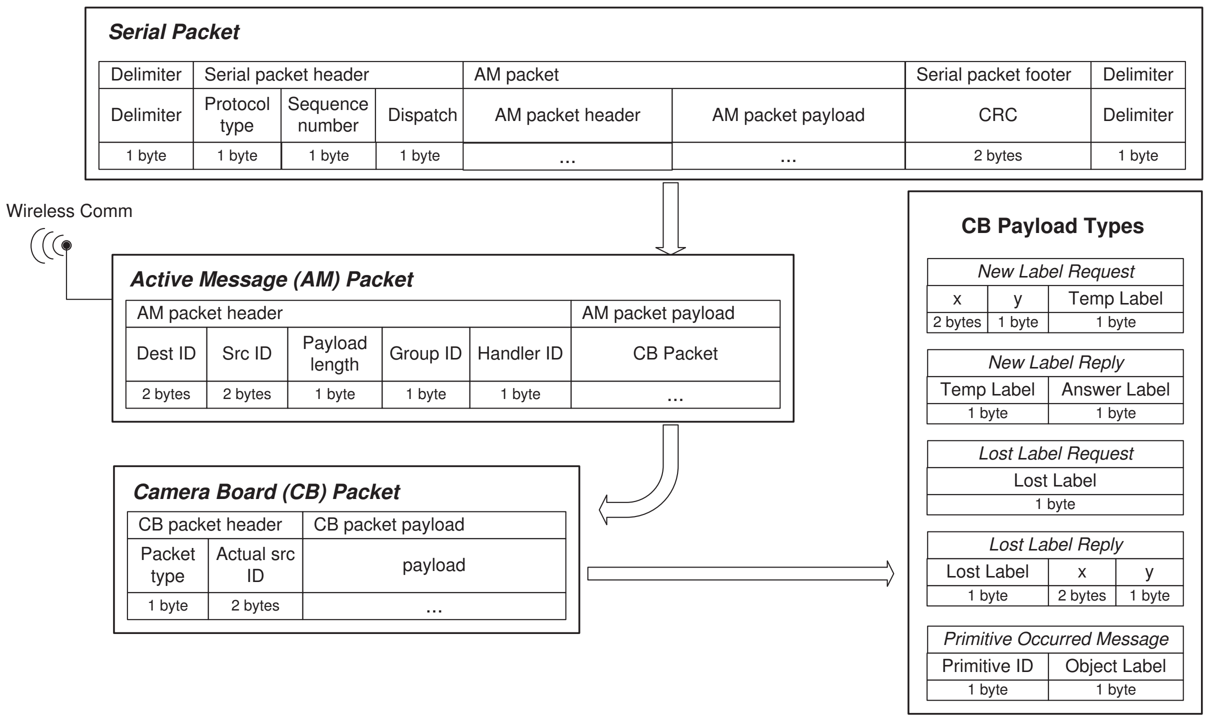


Figure 3.2: Packet Format of Wireless and Serial Communication.

TelosB does not receive an acknowledgement packet, it will send the data packet again after a certain amount of time. The number of retries and the time interval between each retry can be set by the users. The default number of retries is 20 and the time interval between retries is 200 ms.

3.3.2 Flow of Processing and Communication

The camera board and the TelosB mote perform their tasks in parallel and communicate via a serial port. The TelosB mote and the camera board draw current from the same power supply instead of TelosB having its own separate power supply. When TelosB is not transmitting a packet, it is in idle mode. The drawn current increases when the TelosB transmits or receives a packet. The effect of wireless communication on the power consumption will be analyzed in Chapter 4.

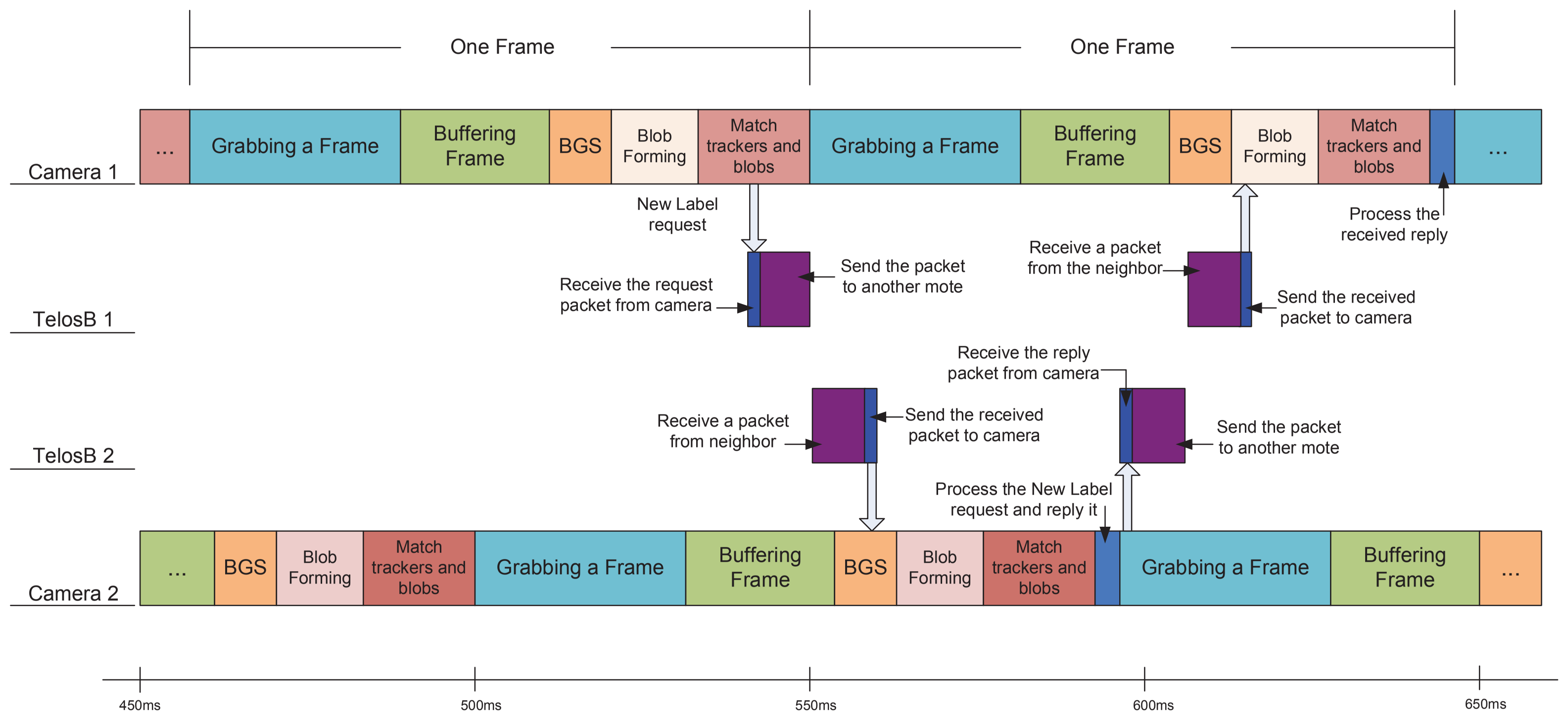


Figure 3.3: The process of request sending and reply receiving between two cameras.

Thanks to the advantages of the multi-threading scheme in embedded Linux systems, a separate thread is employed for communicating with the wireless mote. An incoming queue and an outgoing queue are shared between the main thread and the communication thread. If the camera needs to send a packet, it wraps the message in the CB packet and puts it in the outgoing queue, waiting for the communication thread to send it out. The communication thread keeps polling the serial port. If there is a packet comes in, it takes the message out of the AM packet and puts in the incoming queue for the main thread to process.

Figure 3.3 illustrates the process of sending a request and receiving a reply between two camera nodes. During the processing of a frame, if the camera board needs to send a request message, it puts the message in the outgoing queue. The message will be sent over the serial port to the wireless mote immediately. For instance, Camera 1 sends a request packet while processing the current frame, and its wireless mote transmits it immediately addressed to Camera 2. When the packet is received, the message will be put in the incoming queue of Camera 2, since the main thread of Camera 2 is still busy with processing the current frame. After finishing the processing of the current frame, the main thread gets the request message, and then sends a reply. The reply will be transmitted immediately by the wireless mote. The main thread of Camera 1 will receive the reply after it finishes processing of the current frame.

In our wireless embedded smart camera system, each camera node is in the single-hop communication range of the others. Thus, each message exchange is performed in single-hop. In our experiments, using the previously described message exchange process, Camera 1 sends out a request during the processing of the current frame, and will receive the reply by the end of the next frame. Thus, the time interval between sending the request and receiving the reply is less than 100 ms. This alleviates coherent data transfer problem. However, in the cases of multihop communication and unbalanced workload on different nodes, a synchronization mechanism would be necessary.

3.3.3 When to Communicate

A camera needs to communicate with other cameras when 1) a new object appears in its FOV, 2) a tracker cannot be matched to its target object, and 3) a primitive event that is part of a pre-defined composite event scenario occurs in its field of view. These three cases will be referred to as *New_Label*, *Lost_Label* and *Primitive_Occurred* cases, respectively. These cases have been explained previously and are summarized here as the following.

When a new object is detected in the current camera view, it is possible that this object is being tracked by other cameras. If this is the case, the camera will issue a *New_Label* request addressed to those cameras to require the existing label of this object, and to maintain consistent labeling.

If a tracker cannot be matched to its object due to occlusion or failure of the background subtraction, then the camera will send a *Lost_Label* request to obtain and update the location of the object from the other cameras that can see the same object.

Also, as described in Section 3.2, we define composite and semantically high-level events as a sequence of primitive events, and these primitives can be defined on different camera views. When a camera detects a primitive event, it sends a *Primitive_Occurred* message addressed to the next camera in the sequence to let it know about the occurrence of this primitive event, and the label of the object performing the primitive.

3.3.4 With Whom to Communicate

In the *New_Label* and *Lost_Label* cases, before sending the request, the current camera checks the visibility of the target by other cameras by employing the FOV lines. If it is deduced that this object is visible by another camera, the ID of that camera will be included in the request message. This way, when a camera receives a broadcasted message, it will drop the message if the target ID in the message does not match its own ID.

In the *Primitive_Occurred* case, since the primitives and their sequence are all pre-defined, we already know which primitive is defined on which camera view. When a primitive event occurs in one camera, it will address the message to the next camera in the sequence of events.

3.3.5 What to Communicate

Small-sized packets are exchanged between cameras to reduce power consumption and delay. The contents of messages for different scenarios are described in the following. As aforementioned, each type of message is assigned a unique type ID, which will be inserted into the CB header (Figure 3.2) in the packet.

1) New_Label Request

When a new object appears in the current camera view, a tracker is created for it. If it is determined by using FOV lines that another camera can see this object, a temporary label is assigned to the object and a request message addressed to that camera is created. The ID of the camera to which this message is addressed is inserted into the destination ID in the AM packet header. The AM packet for this message has the following format:

$$AM\_header \quad CB\_header \quad x \quad y \quad Tmp\_label$$

where x and y are the coordinates of the object in the current camera view. *Tmp_label* is the temporary label assigned to this newly found object. When a reply is received, this temporary label is replaced by the received label. In this case, we need 7 bytes for the AM header, 3 bytes for the CB header, 2 bytes for x, 1 byte for y (the width of the frame is greater than 256 and the height of the frame is less than 256), and 1 byte for the *Tmp_label*. Thus, we only use 14 bytes for a *New_Label* request packet.

2) New_Label Reply

When a camera node receives a packet that is addressed to itself, and if this packet is for a *New_Label* request, the camera node will calculate the object's corresponding location in its own view by using the received coordinates and the homography matrix calculated off-line. Then, it will find the distance of the closest tracker to the calculated location. If this distance is smaller than a threshold, it will send the label of this tracker as reply in the following packet form:

$$AM\_header \quad CB\_header \quad Tmp\_label \quad Ans\_label$$

where *Tmp_label* is the temporary label the requesting camera is using, and *Ans_label* is the reply label. A unique number indicating that this packet is for a *New_Label* reply is also assigned in the CB header. Destination ID is the source ID found in the received packet. In this case, we only use 12 bytes in the *New_Label* reply packet.

3) Lost_Label Request

For a tracker that cannot be matched to its object, a camera that can see the most recent location of this tracker is found by using the FOV lines. Then, a *Lost_Label* request packet is formed, which has the following format:

$$AM\_header \quad CB\_header \quad Lost\_label$$

where *Lost_label* is the label of the tracker which could not be matched to an object. We only use 11 bytes for a *Lost_Label* request.

4) Lost_Label Reply

When a camera node receives a packet that is addressed to itself, and if this packet is for a *Lost_Label* request, the camera node sends the current location of the tracker, whose label is the same as the *Lost_Label* entry of the request message, as reply. The reply packet has the following format:

$$AM\_header \quad CB\_header \quad Lost\_label \quad x \quad y$$

where *Lost_label* is the label of the tracker received from the requester, and x and y are the coordinates of the object. In this case, we use 14 bytes in the *Lost_Label* reply packet.

When the requesting camera receives the reply, it calculates the corresponding location of the object in its own view, and updates the tracker's location.

In our experiments, when the object is partly or fully occluded, we sent the *Lost_Label* request every frame to see and show the continuous update of the objects location. This is for visualization purposes only. Sending the request every frame is not very efficient in terms of energy consumption, since transmitting and receiving a message increase the operating current, as shown in Chapter 4. Instead of sending the request every frame, when we detect a new blob (reappearing after occlusion), we can retrieve its label using the *New_Label* request.

5) Primitive_Occurred Message

As described previously, composite and semantically higher level events of interest are defined beforehand by connecting primitive events in a sequence. If a defined primitive event is detected in a camera node, this node sends out a *Primitive_Occurred* message to inform the next camera node in the defined event sequence. Thus, it does not need any replies. When the first primitive event occurs in the view on which it was defined, this camera sends a message addressed to the camera that is responsible to detect the next primitive event in the sequence. Once the second primitive occurs, and if the third primitive is defined on a different camera view, the second camera sends a message addressed to that camera. The form of the packet for the *Primitive_Occurred* message is:

$$AM\_header \quad CB\_header \quad Prim\_ID \quad Obj\_label$$

where *Prim_ID* is the order number of the primitive event in the sequence, and *Obj_label* is the label of the object performing this event. In this case, we only use 12 bytes for the *Primitive_Occurred* message.

In the current version, when a camera receives a *Primitive_Occurred* message, it resets a counter, and starts counting the frames to determine if the time interval criterion is satisfied between two consecutive primitive events.

3.4 Conclusions

We presented a wireless embedded smart camera system for cooperative object tracking and detection of composite, semantically high-level events spanning multiple camera views. With sharing a common planar ground across the partially overlapping camera views, the FOV lines of the neighboring cameras in the current camera view can be recovered using the homography matrices. If an object is deduced visible in another camera view, the current camera can either send a *New_Label* request addressed to that camera to retrieve the label for a newly detected object, or send a *Lost_Label* request to retrieve the updated location of an occluded object.

The presented embedded smart camera system can detect composite event scenarios spanning multiple camera views. Semantically higher level events can be defined by connecting primitive events in a time sequence. The complexity of event scenarios can be increased by increasing the number of primitives on a camera view, and/or the number of camera views they span.

Cameras exchange data in a P2P manner over wireless links to track objects with consistent labels, to update locations of occluded or lost objects, and also to inform other cameras about the occurrence of a primitive event in a composite event scenario. The cameras only need to send 14 bytes for *New_Label* request, 12 bytes for *New_Label* reply, 11 bytes for *Lost_Label* request, 14 bytes for *Lost_Label* reply and 12 bytes for *Primitive_Occurred* message.

Chapter 4

Power Analysis and Experimental Results

Power Consumption is critical for embedded cameras powered by batteries. From the perspective of algorithms, well-designed algorithms that are optimized for the hardware architecture help to reduce the power consumption significantly. To analyze the factors that influence the power consumption of our system, and demonstrate the efficiency of the proposed algorithms, we performed experiments in different tracking scenarios and estimated the average power consumptions by measuring the operating currents. The results also provide additional insight in terms of computation versus communication tradeoff and careful camera placement, and demonstrate and emphasize the importance of carefully designing a communication protocol in these resource-constrained environments.

In the second part of this chapter, to demonstrate the successfulness of the algorithms that are proposed in Chapter 2 and Chapter 3, we performed experiments for each scenario including *New_Label*, *Lost_Label* and *Primitive_Occurred* cases. Multiple examples of composite event detection involving different numbers of cameras and different numbers of primitives are presented.

4.1 Power Consumption and Performance Analysis

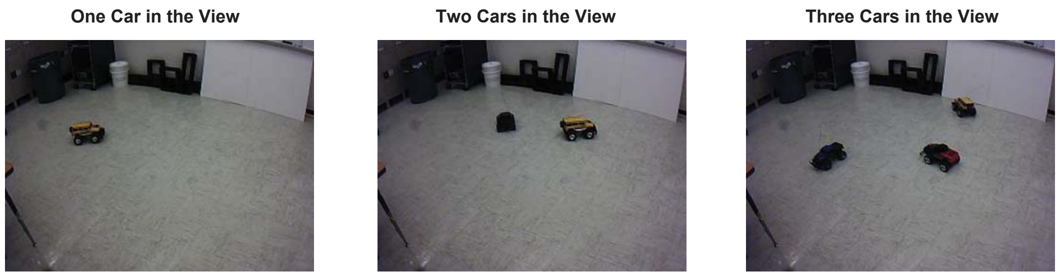
We performed a detailed analysis of the energy consumption and performance of the presented system during different parts of processing a frame, when tracking different number of objects, when tracking different-sized objects, and when transmitting and receiving message packets. We also compared the energy requirement when transmitting different-sized packets.

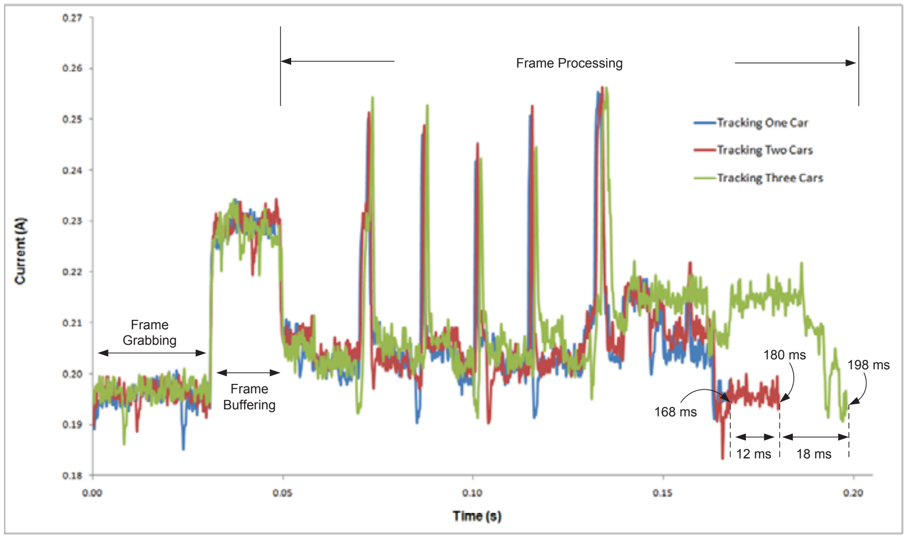
For this analysis, we measured the operating current of the embedded smart camera for different scenarios, which are listed in the following. To measure the currents, we used a precise oscilloscope and a 1Ω resistor configuration placed at the input of the supply source. We then computed the energy consumption of the proposed system during different tasks based upon the measured operating currents.

4.1.1 Operating Currents while Tracking Different Number of Objects

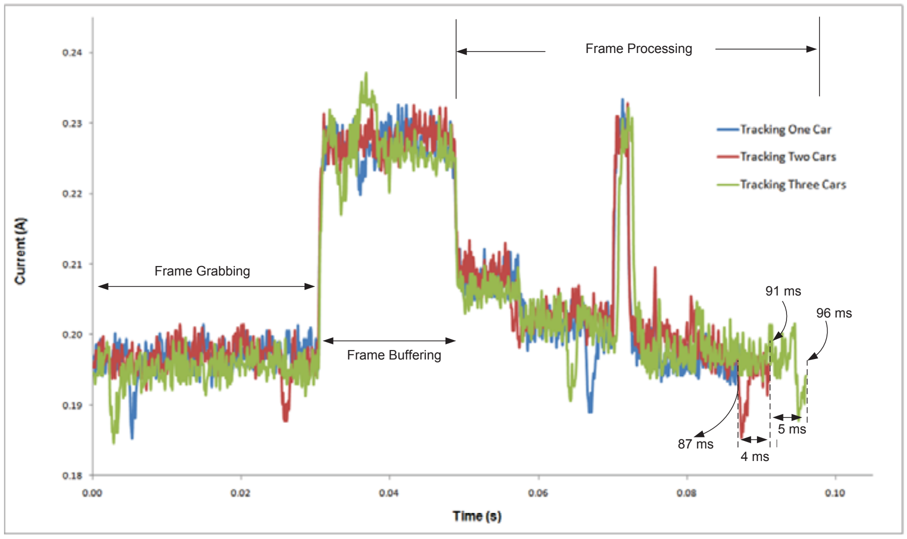
The energy consumption of the camera board depends highly upon the amount of activity. Different number of objects in the scene causes some variations in the operating current of the camera board, and more importantly in the processing time. Amount of current and processing time also depend upon the algorithms used.We measured the operating current when there were different number of objects in the scene. We used three remote-controlled cars, and employed two different algorithms, described in Chapter 2.3, for blob forming.

The blue, red, and green plots in Figure 4.1 (a) are the currents drawn during the processing of one frame when tracking one, two and three cars, respectively, and when using the multiple pass connected component labeling algorithm. For the one-car case, the processing of the frame takes 168 ms. This number includes the time needed for frame capturing, foreground detection, connected component labeling and tracking. When there





(a)



(b)

Figure 4.1: Amount of the current drawn by the camera board over time while tracking one, two and three remote-controlled cars and when using: (a) multi-pass connected component labeling algorithm and (b) single-pass blob forming algorithm.

Table 4.1: Power and energy consumption for different scenarios when using the multi-pass connected component labeling

	One-Car	*Two-Cars*	*Three-Cars*	*One Bigger Car*	*Two Smaller Cars*
Current (mA)	202	202	204	204	202
Power Consumption (W)	1.171	1.171	1.182	1.182	1.171
Time (msec)	168	180	198	207	180
Engery (J)	0.197	0.211	0.234	0.245	0.211

Table 4.2: Power and energy consumption for different scenarios when using the single-pass blob forming algorithm

	One-Car	*Two-Cars*	*Three-Cars*	*One Bigger Car*	*Two Smaller Cars*
Current (mA)	201	201	199	198	201
Power Consumption (W)	1.166	1.166	1.154	1.149	1.166
Time (msec)	87	91	96	99	91
Engery (J)	0.101	0.106	0.111	0.114	0.106

are two cars in the scene, it takes 12 ms more to finish processing of one frame (red plot). When three cars are tracked, it takes 18 ms longer for the camera to finish processing one frame compared to the two-car case.

From Figure 4.1 (a), we can see that when embedded smart cameras track multiple objects with comparable sizes, they consume more energy with increasing number of objects. The energy required for processing a frame containing one car is $E = P \times T = V_{board} \times I \times T = (6 - 202mA \times 1\Omega) \times 202mA \times 168ms = 0.197J$, where V_{board} is the voltage across the camera board, and 6 volt is the power supply we used in our experiments. Then V_{board} is equal to the power supply minus the voltage across the 1Ω resistor. Similarly, the energy required for processing frames containing two and three cars are $0.211J$ and $0.234J$, respectively. These values were obtained when TelosB was not attached to the camera board. We also measured the operating currents with TelosB attached. All operating currents and computed energy values are summarized in Table 4.1.

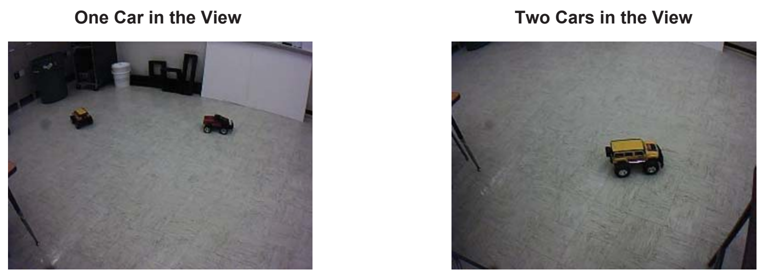
We also performed a similar experiment when running the more efficient, one-pass blob forming algorithm. Figure 4.1 (b) shows the obtained operating currents while tracking different number of objects. As can be seen, the processing time of each frame decreases significantly. Specifically, it decreases to 87, 91, and 96 ms for tracking one, two and three cars, respectively. This, in turn, provides significant savings in the energy consumption. The decrease in energy consumption can be seen by comparing Tables 4.1 and 4.2.

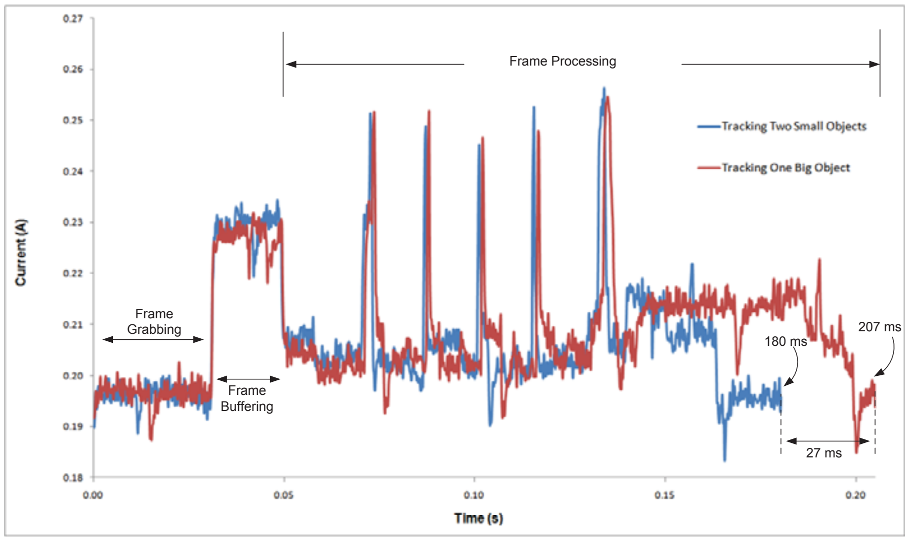
4.1.2 Operating Currents while Tracking Different-Sized Objects

We also analyzed the effect of the size of the tracked objects on the operating current of the embedded camera board. We compared the measured operating currents when tracking two smaller objects and when tracking one larger object. We placed the camera closer to the scene to capture a larger view of one of the cars, and measured the operating current for one frame by using the oscilloscope.

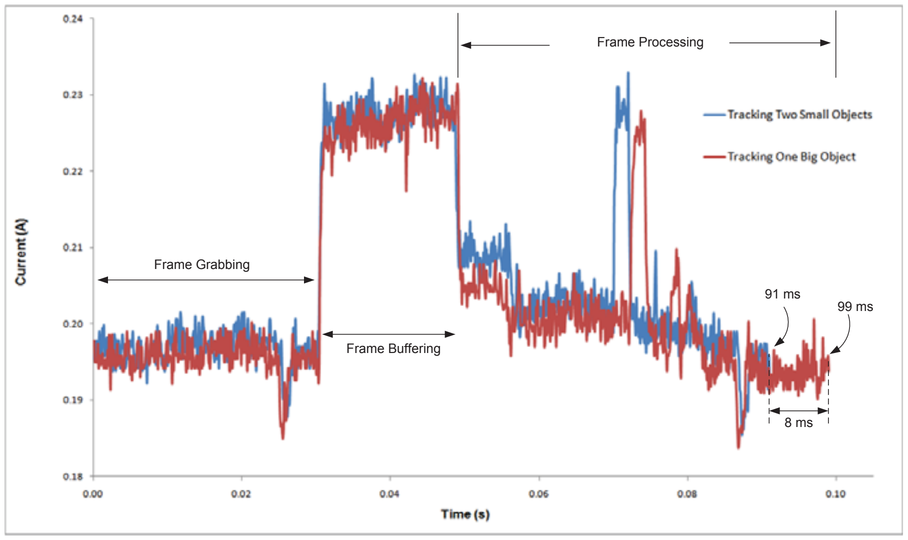
Figure 4.2 (a) shows the current amounts drawn by the camera board when using the multi-pass connected component labeling algorithm. Due to the nature of the algorithm, the number of foreground pixels in the image will have an influence on the speed of building connected components. Thus, it is expected to have increased energy consumption. As seen in Figure 4.2 (a), when tracking one larger-sized car, it takes 27 ms more to finish the processing of one frame.

Figure 4.2 (b) shows the current amounts drawn by the camera board when using the more efficient blob forming algorithm. As expected, compared to the multi-pass algorithm, the processing time increases less with increasing number of foreground pixels. When tracking one larger-sized car, it takes 8 ms more to finish the processing of one frame, compared to tracking two smaller-sized cars. In this experiment, two smaller-sized cars occupy 390 and





(a)



(b)

Figure 4.2: Operating currents of the camera board while tracking one larger car and two smaller cars, and when using: (a) multi-pass connected component labeling algorithm and (b) single-pass blob forming algorithm.

604 pixels in the frame. In the case, where camera is set up closer to the scene, the larger-sized car occupies 1944 pixels. The computed energy values for two different algorithms are listed in Tables 4.1 and 4.2.

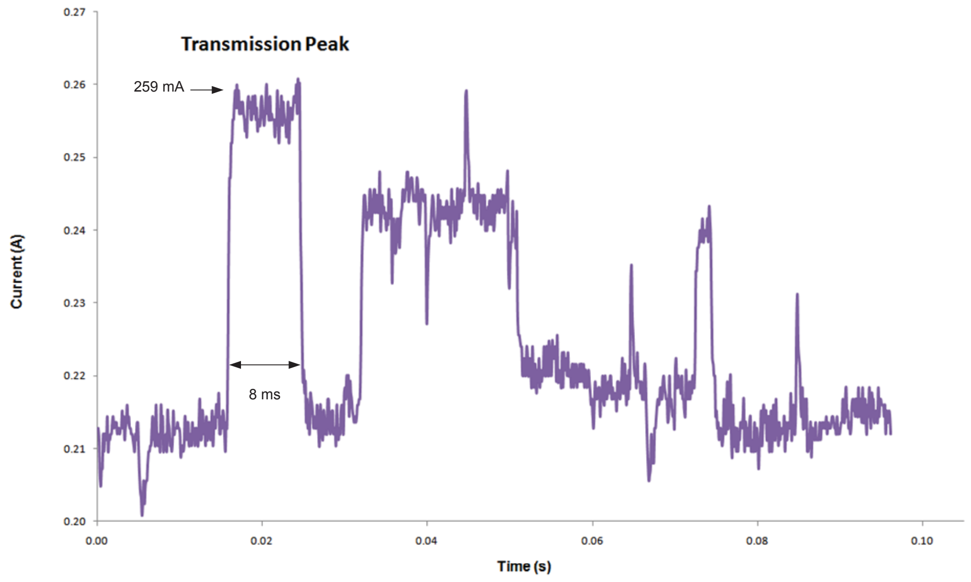
This analysis also provides additional insight on how to efficiently place the resource-constrained cameras in the scene. Since tracking more of smaller objects consume less energy, it may be preferable to install the cameras further from the scene depending upon the application.

4.1.3 Operating Currents while Tracking One Car with Communication

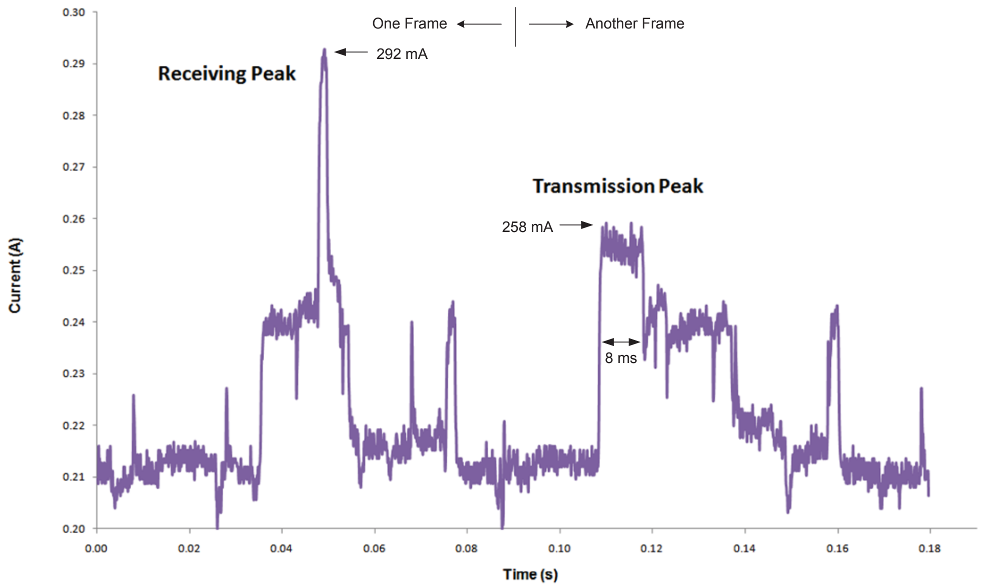
To be able to capture the instances of a camera communicating wirelessly, we designed an experiment in which we force the camera to send a new label request to the other camera every five frames. Thus, the camera also receives the new label reply every five frames. Figure 4.3 (a) and (b) show peaks in the drawn current caused by transmitting and receiving, respectively. In this experiment, the camera is tracking one car in the scene, and the single-pass blob forming algorithm is used.

As stated previously, once a packet is sent to other wireless nodes, TelosB will wait for an acknowledgement from the receiver. If TelosB does not receive an acknowledgement packet, it will send the data packet again after a certain amount of time. The default number of retries is 20 and the time interval between retries is 200 ms. There is a delay between the time the camera sends its request to TelosB and the time TelosB transmits. This time delay is measured to be 10 ms. The peak in the operating current caused by transmitting a packet is marked in Figure 4.3 (a).

The peaks caused by receiving a new label request packet and transmitting a new label reply packet are shown in Figure 4.3 (b). We can see that with data exchange, there are ap-



(a)



(b)

Figure 4.3: Operating current of the camera when (a) transmitting a new label request, and (b) when receiving a new label request and transmitting a new label reply.

parent peaks in the operating current caused by the wireless communication. The additional energy consumption caused by transmitting and receiving a packet is analyzed in Section 4.1.4.

We also measured the average operating current for different parts of the processing of a frame, i.e., we measured the average operating current for grabbing a frame, buffering the frame, and for performing foreground detection and tracking on the frame. Then, we calculated the average consumed power during these different portions as described in Section 4.1.4.

4.1.4 Power and Energy Consumption of the Embedded Smart Cameras

In order to calculate the energy consumption, we use

$$E = P \times t = V \times I \times t = (6 - I \times 1\Omega) \times I \times t \tag{4.1}$$

where 6 volt is the voltage supply used, and $I \times 1\Omega$ is the voltage drop across the 1Ω resistor. First, we computed the energy consumption while tracking one, two and three cars, and when using the multi-pass connected component labeling and single-pass blob forming algorithms. In order to observe the current drawn only by the camera board, we first measured the currents when the TelosB is not attached. The measured current values, and the computed energy consumption when using the two different blob forming algorithms are listed in Tables 4.1 and 4.2. As can be seen, when there are more objects in the scene, or there are larger-sized objects in the scene, the energy consumption increases. Also, morphological operations followed by the multi-pass connected component labeling algorithm consume significantly more energy than the single-pass blob forming algorithm.

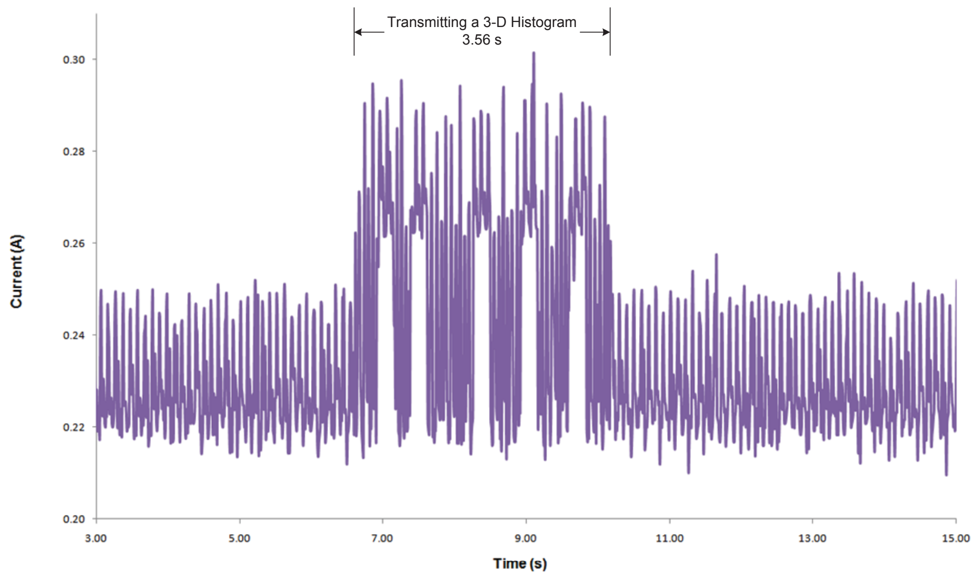
We also measured the drawn current, when the TelsoB is attached to the camera board.

According to our measurements, TelosB causes an increase of 25 to 28 mA in the drawn current.

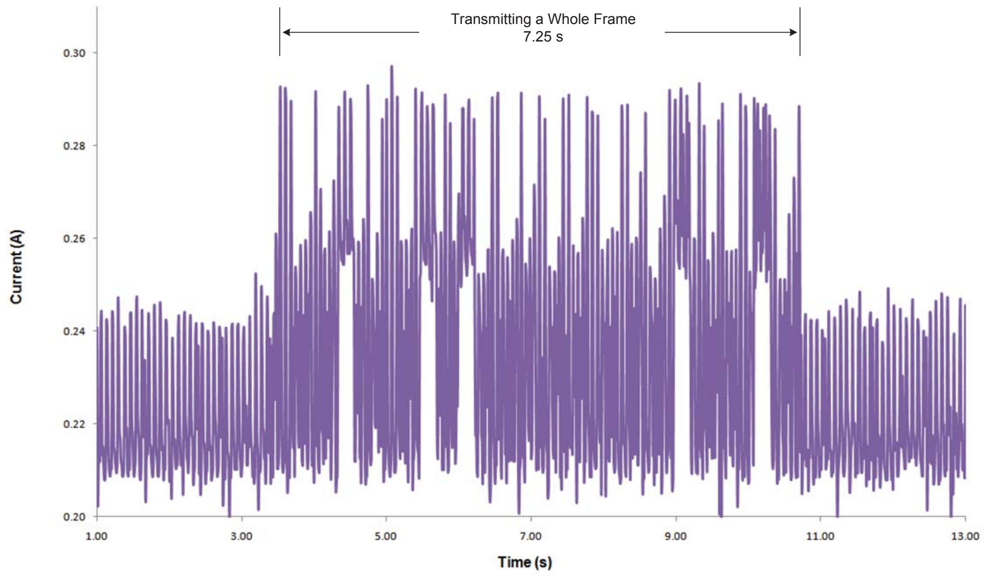
We also analyzed the additional power and energy consumption caused by transmitting and receiving packets. In Figure 4.3 (a), the packet is being transmitted while the camera is grabbing a frame. The average current while transmitting a packet is 256 mA, and transmission takes 8 ms. The average current when grabbing a frame is 213 mA. Thus, the additional power consumption caused by transmitting a new label request packet is $(6 - 0.256) \times 0.256 - (6 - 0.213) \times 0.213 = 0.24W$, and the additional energy consumption is $0.24 \times 8ms = 1.92mJ$.

In Figure 4.3 (b), the packet is being received while the camera is buffering a frame. The average operating current while receiving a packet is 285 mA, and it takes 4 ms. The average operating current when buffering a frame is 241 mA. Thus, the additional power consumption caused by receiving a new label request packet is $(6 - 0.285) \times 0.285 - (6 - 0.241) \times 0.241 = 0.24W$, and the additional energy consumption is $0.24 \times 4ms = 0.96mJ$.

In another experiment, we measured the operating current drawn over time when transmitting a packet containing color histogram information for a tracked object, and when transmitting a whole image. The size of the 3-D histogram payload is 4096 B, and the size of the whole JPEG image payload is 10.5 KB. As can be seen in Figure 4.4, transmitting a 3-D histogram is completed in 3.56 s, and transmitting a whole JPEG image is completed in 7.25 s. Thus, transmitting a whole image or other large-sized data packets incur significant energy consumption and delay. When the 3-D histogram is transmitted, it causes an additional 0.641 J of energy consumption. Transmitting the whole image incurs an additional 1.131 J consumption. Compared to these, transmitting a new label request message in our application layer protocol takes significantly less time (8 ms) and consumes significantly less energy (1.92 mJ). Hence, it is very important to design algorithms that require transfer of small-sized packets between nodes. It is also very important to design efficient protocols to



(a)



(b)

Figure 4.4: Current amounts drawn over time when transmitting (a) a packet containing color histogram of a tracked object (b) a whole image.

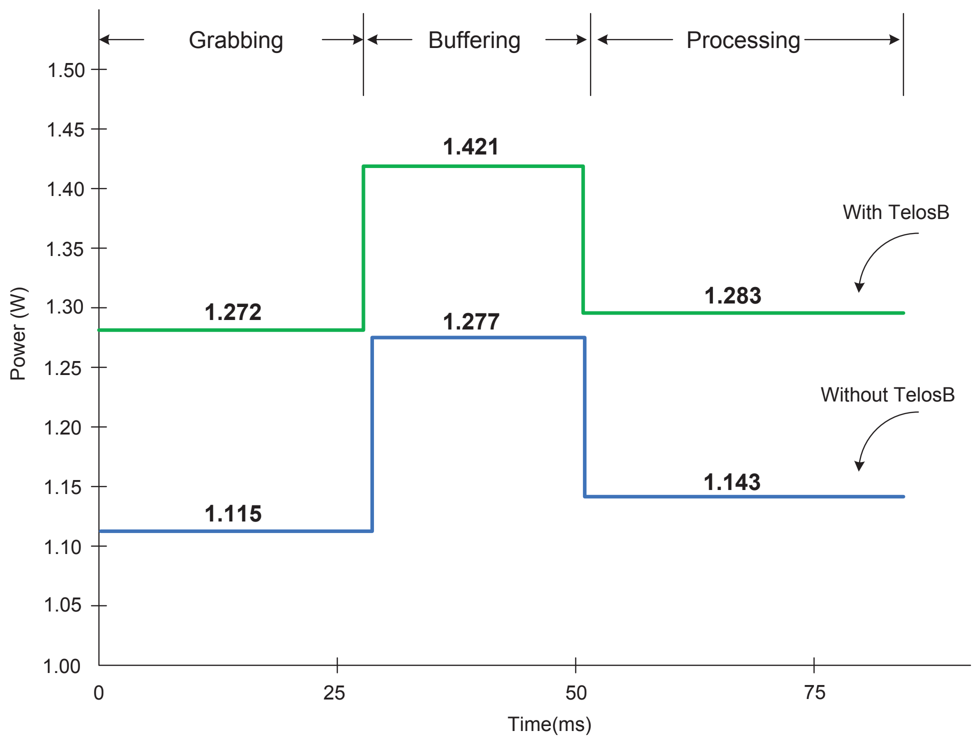


Figure 4.5: Average power consumed by the camera during different portions of processing a frame(with and without TelosB)

decrease the message traffic.

We also calculated the average consumed power during different parts of processing a frame. These parts are grabbing a frame, buffering the frame, and performing vision processing, i.e., foreground detection and tracking. The obtained values are displayed in Figure 4.5.

We performed another experiment, and based upon the measurements of operating currents during wireless communication and tracking multiple objects, we calculated the average power consumption for different parts of processing a frame while tracking two cars and transmitting or receiving a packet. Figure 4.6 shows the obtained results. When a car enters into the FOV of the camera, it transmits a New label request message, and during the transmission, the average power consumption is 1.481 W as seen in Figure 4.6. The

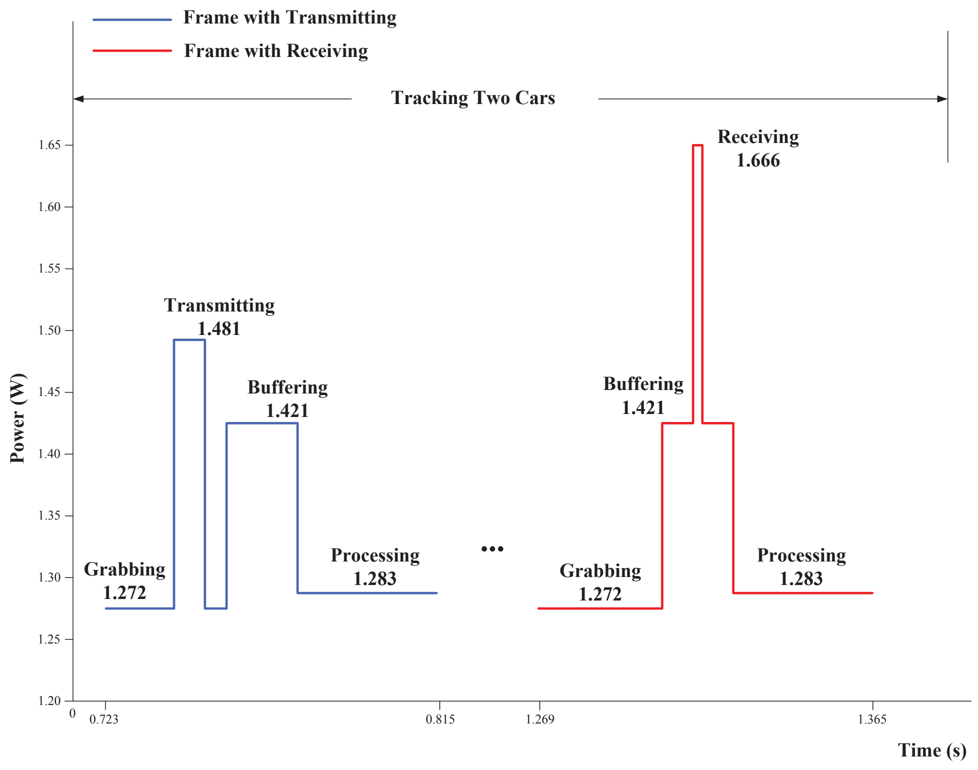


Figure 4.6: Average power consumed by the camera during different portions of processing a frame while tracking two cars and transmitting or receiving a packet

average power consumption during foreground detection, blob forming and tracking is 1.283 W. When the camera receives the reply message, the average power consumption is 1.666 W.

These results demonstrate the importance of carefully designing a communication protocol, implementing lightweight algorithms, and configuring camera placements in these resource-constrained environments. Deciding what data to send and when to send is very important since communication is expensive, and even sending or receiving a 14-B packet causes jumps in the power consumption. Camera placements also make a difference since tracking one larger object consumes more energy than tracking multiple smaller objects as shown in our experiments.

4.2 Object Tracking and Event Detection Experimental Results

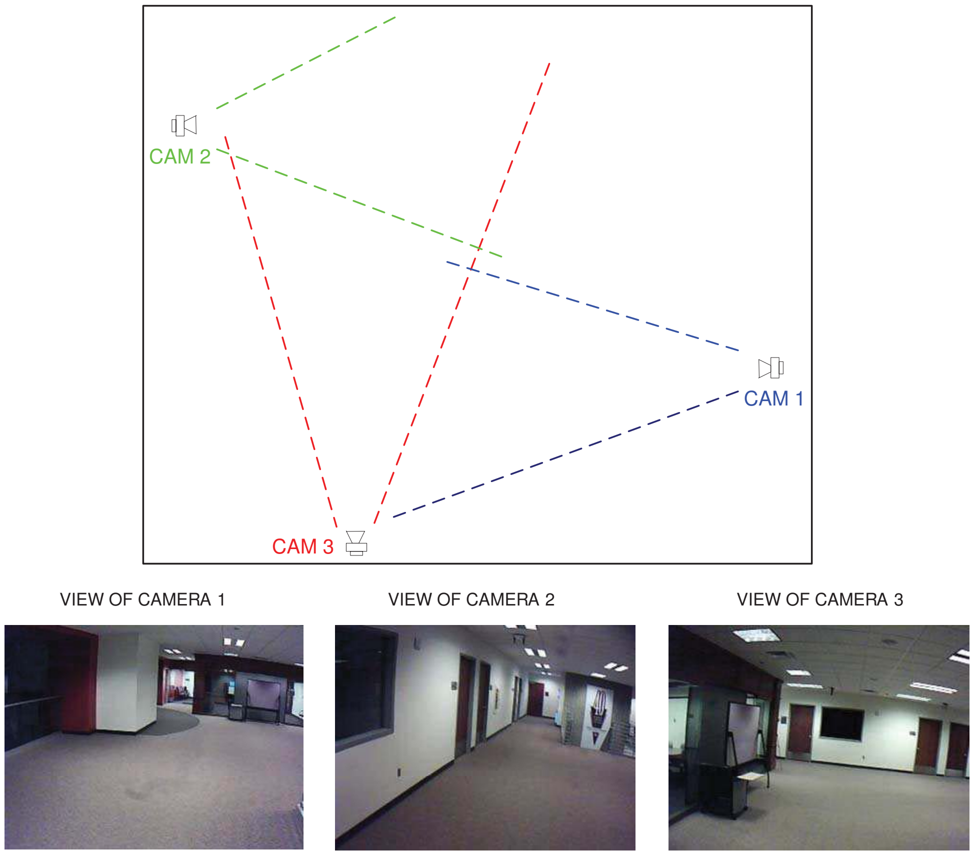
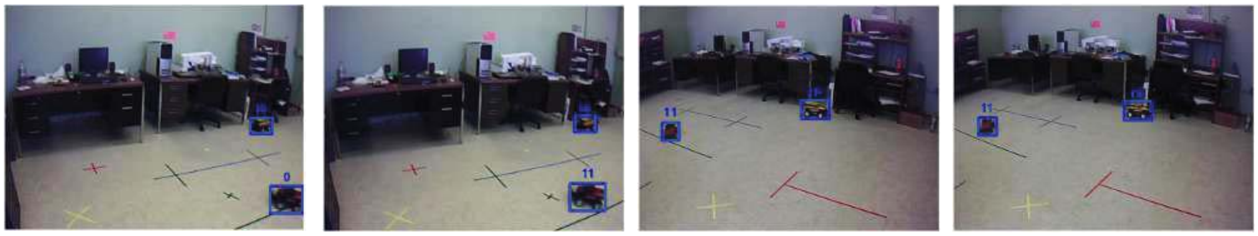


Figure 4.7: The three-camera setup.

We performed different sets of experiments for different scenarios with the presented wireless embedded smart camera system. Experiments were carried out by tracking people as well as remote-controlled cars. Below, we present results of tracking with consistent labels, updating locations of lost/occluded objects, and detecting composite events spanning multiple camera views.

In the first set of experiments, we used two CITRIC cameras with partially overlapping fields of view, and tracked remote-controlled cars. Then, we set up three CITRIC cameras



(a) Camera 1 (b) Camera 1 (c) Camera 2 (d) Camera 2

Figure 4.8: Two cars being tracked across different camera views with consistent labels.

to track people and detect events of interest. Event scenarios were defined so that they span over three different camera views.

The configuration of the camera positions is as shown in Figure 4.7. Camera 1 and Camera 3, and Camera 2 and Camera 3 have overlapping fields of view; whereas the fields of view of Camera 1 and Camera 2 do not overlap. Thus, we only compute the homography matrix between Camera 1 and Camera 3, and between Camera 2 and Camera 3. If an object moves from the view of Camera 1 into the view of Camera 3 and then Camera 2, Camera 3 needs to act as a bridge, and transfer the label that it receives from Camera 1 to Camera 2.

4.2.1 Consistent Labeling

We first performed experiments where multiple cars are tracked with consistent labels across different camera views. As seen in Figure 4.8 (a), when a car enters the FOV of the first camera it is assigned a temporary label 0. Then, when the camera receives the correct label from the other camera, the temporary label is changed to the correct one (11) as shown in Figure 4.8 (b). Figure 4.8 (c) and (d) show the second camera tracking the object.

Figures 4.10, 4.11 and 4.12 show key frames from different composite event detection experiments. These figures also include several examples of successful label transfer, and show cars being tracked with consistent labels across different camera views. As seen in Figures 4.10 (b), 4.11 (d), 4.12 (c), 4.12 (g) and 4.12 (i), when a car enters the FOV of a camera, it is first assigned a temporary label 0. Then, this camera sends a New_Label request message to the other camera if it determines that the other camera can see the same object. When the requesting camera receives the correct label from the other camera, the temporary label is replaced by the correct label, as shown in Figures 4.10 (c), 4.11 (e), 4.12 (d), 4.12 (h) and 4.12 (j).

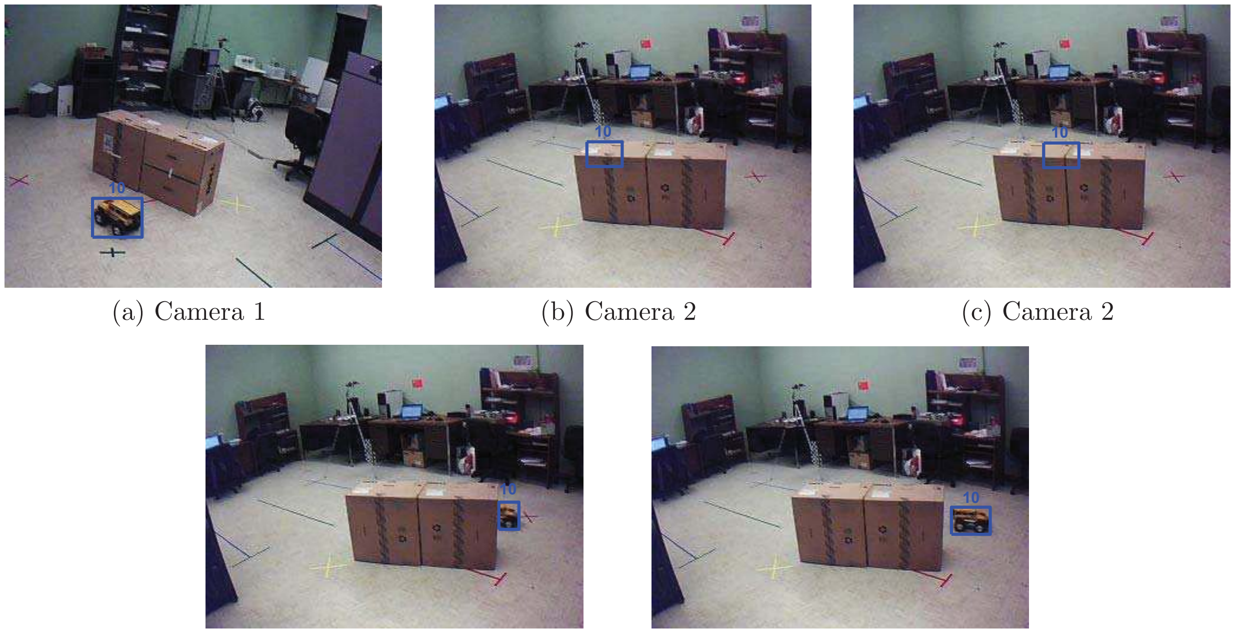
Figures 4.13 and 4.14 show multiple people being tracked with consistent labels by three embedded smart cameras. As seen in Figures 4.13 (k), 4.14 (h) and 4.14 (j), when a person enters the FOV of a camera, it is first assigned a temporary label 0. Then, this camera sends a New_Label request message to the other camera(s) if it determines that the other camera(s) can see this person. When the requesting camera receives the correct label from the other camera, the temporary label is replaced by the correct label, as shown in Figures 4.13 (l), 4.14 (i) and 4.14 (k).

4.2.2 Updating the Location of a Lost Object

For this experiment, we used two cameras, and placed an occluding structure in the scene as seen in Figure 4.9. The first camera can see and track the car, whereas the second camera loses it at some point since it is occluded by the box. Figure 4.9 (a) shows an example frame from the first camera view. When the second camera loses the object, it can still update its location by exchanging data with the first camera as seen in Figures 4.9 (b) and (c). In Figure 4.9 (d) and (e), the car reappears and the tracker is locked back to its object.

4.2.3 Event Detection Experiments

We defined different composite and spatio-temporal event scenarios spanning two and three different camera views. The composite event scenarios consist of two, three or four primitives connected in sequence. The definition of each primitive event is saved as a structure, which



(a) Camera 1 (b) Camera 2 (c) Camera 2

(d) Camera 2 (e) Camera 2

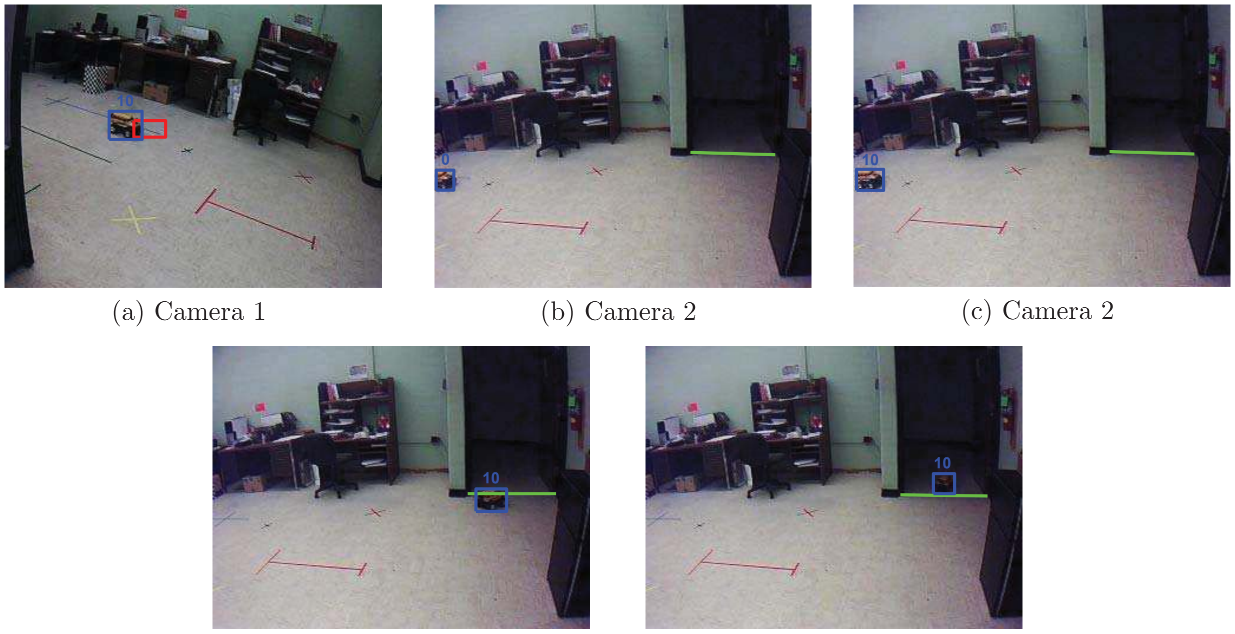
Figure 4.9: A car being occluded in the second camera view.

contains the primitive ID, the type of the primitive (MD, TW or AO), the ID of the camera that is responsible of detecting this primitive, and the parameters of the primitive event (such as point coordinates defining the ROI, location and direction of the defined tripwire etc).

We performed event detection experiments by tracking people as well as remote-controlled cars. The maximum time interval between the primitive events was set to be 100 frames in all the experiments.

1) Event Scenario Composed of Two Primitive Events Spanning Two Camera Views

The first event scenario of interest is detecting a car going through a region of interest on the first camera view, *and then* exiting the scene on the second camera view. Thus, this event spans two different camera views. This scenario was defined as a sequence of two

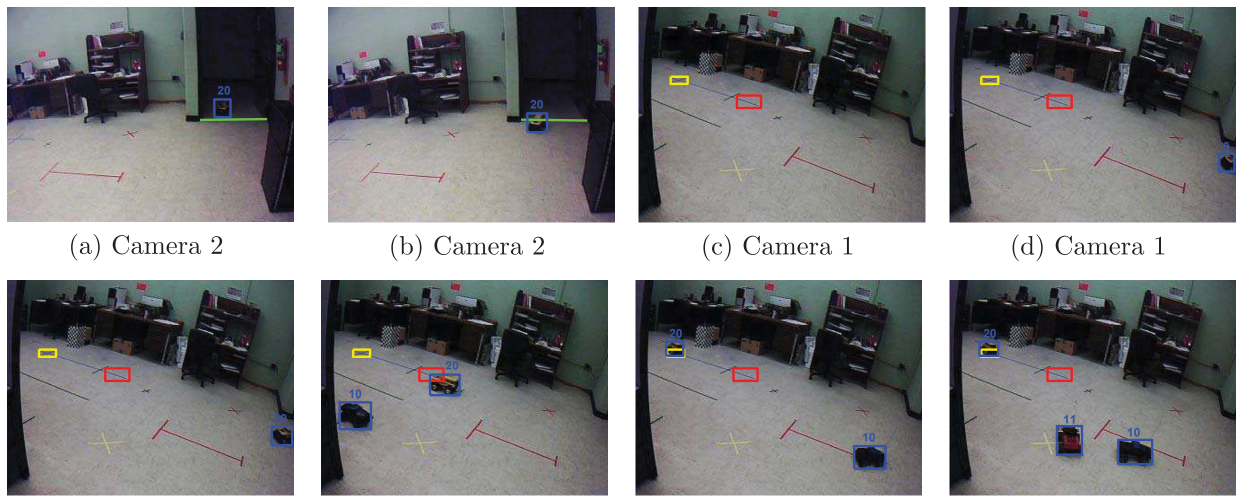


(a) Camera 1 (b) Camera 2 (c) Camera 2

(d) Camera 2 (e) Camera 2

Figure 4.10: An event scenario composed of two primitive events spanning two different camera views.

primitive events, namely MD in the first camera view, and TW in the second camera view. Figure 4.10 (a) shows the first primitive event being detected on the first camera view. When this primitive is detected, the first camera sends a *Primitive_Occurred* message addressed to the second camera, and this message includes the label 10, which is the label of the object performing the event. At this time, the second camera cannot see this object yet, and saves this label. In Figure 4.10 (b) a new car just enters into the FOV of the second camera, and is assigned a temporary label 0. The second camera sends a new_label request to the first camera, and receives the correct label 10 from the first camera as seen in Figure 4.10 (c). The second primitive event is detected on the second camera view as shown in Figure 4.10 (d) and Figure 4.10 (e). Since the label of the object performing the second primitive event is the same as the label in the received *Primitive_occurred* message, and the time interval between these two primitive events is less than the specified interval, the overall event scenario occurs,



(a) Camera 2 (b) Camera 2 (c) Camera 1 (d) Camera 1

(e) Camera 1 (f) Camera 1 (g) Camera 1 (h) Camera 1

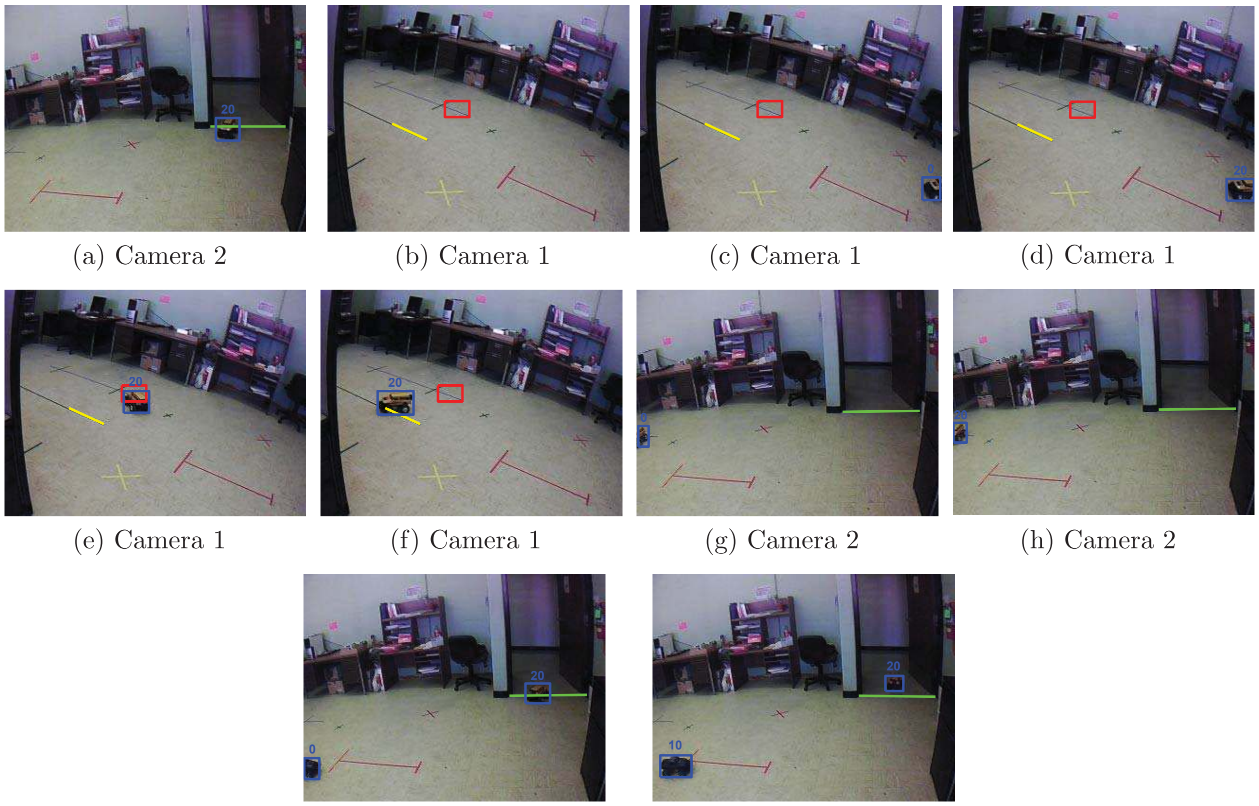
Figure 4.11: An event scenario composed of three primitive events spanning two different camera views.

and is successfully detected.

2) Event Scenario Composed of Three Primitive Events Spanning Two Camera Views

The second event scenario of interest is detecting a car entering the scene in the second camera view, *and then* going through a region of interest in the first camera view, *and then* parking in a region defined on the first camera view. This scenario was defined as a sequence of three primitive events, namely TW in the second camera view, followed by MD in the first camera view, followed by AO in the first camera view.

Figure 4.11 (a) and (b) shows the first primitive event being detected on the second camera view. When this primitive is detected, the second camera sends a *Primitive_Occurred* message addressed to the first camera, and this message includes the label 20, which is the label of the object performing the event. At this time [Figure 4.11 (c)], the first camera cannot see this object yet, and saves this label. In Figure 4.11 (d), a new car just enters into the FOV of the first camera, and is assigned a temporary label 0. The first camera sends a



(a) Camera 2 (b) Camera 1 (c) Camera 1 (d) Camera 1

(e) Camera 1 (f) Camera 1 (g) Camera 2 (h) Camera 2

(i) Camera 2 (j) Camera 2

Figure 4.12: An event scenario composed of four primitive events spanning two different camera views.

new label request to the second camera, and receives the correct label 20 as seen in Figure 4.11 (e). The second primitive event is detected on the first camera view as shown in Figure 4.11 (f). The third primitive event is detected on the first camera view as shown in Figure 4.11 (g) and (h). Since the label of the object performing the second and third primitive events in the first camera view is the same as the label 20 in the *Primitive_Occurred* message received from the second camera, and the time intervals between primitive event pairs are less than the specified intervals, the overall event scenario occurs, and is successfully detected.

3) Event Scenario Composed of Four Primitive Events Spanning Two Camera Views

The event scenario of interest is detecting a car entering the scene in the second camera view, *and then* going through a region of interest in the first camera view, *and then* crossing a line in the first camera view, *and then* exiting the scene in the second camera view. This scenario was defined as a sequence of four primitive events, namely TW in the second camera view, followed by MD in the first camera view, followed by TW in the first camera view, followed by another TW in the second camera view. Figure 4.12 shows the key frames, and the detection of each primitive event. This scenario is interesting since the object leaves the view of the second camera, is continued to be tracked in the first camera, and then reenters into the FOV of the second camera. Since the first and last primitives are defined on the second camera view, correct label exchange, and exchanges of *Primitive_Occurred* messages are essential. When the second camera detects the first primitive (Figure 4.12 (a)) it sends a *Primitive_Occurred* message addressed to the first camera, and this message includes the label 20, which is the label of the object crossing the tripwire. At this time, the first camera cannot see this object yet (Figure 4.12 (b)), and saves this label. The label of a new object entering into the view of the first camera is received from the second camera as seen in Figure 4.12 (d). The second and third primitive events are detected on the first camera view as shown in Figures 4.12 (e) and 4.12 (f), respectively. At this time this car is no longer visible in the second camera view. The label of a new object entering into the view of the second camera is received from the first camera as seen in Figure 4.12 (h). The fourth primitive event is detected on the second camera view as shown in Figures 4.12 (i) and 4.12 (j). Since the labels of the objects performing all the primitive events are the same, and the time intervals between primitive event pairs are less than the specified intervals, the overall event scenario occurs, and is successfully detected.

4) Event Scenario Composed of Three Primitive Events Spanning Three Camera Views

The defined event scenario is detecting a person crossing a tripwire in the second camera view, *and then* going through a region of interest in the third camera view, *and then* going through another region of interest in the first camera view. Figure 4.13 shows the key frames, and detection of each primitive event. An interesting point to note is that, in this experiment, two different people perform the defined sequence of events at different times, i.e. the defined scenario occurs twice. The system can successfully detect the occurrences of both instances.

Figure 4.13 (a) shows the first primitive event detected with Object 20 in the second camera view, and then in Figure 4.13 (b) and (c) the third camera sees Object 20 and retrieves the correct label from the second camera. Then Object 20 enters the first camera and its label retrieved from the third camera Figure 4.13 (d) and(e). Object 20 is observed with the second primitive event in Figure 4.13 (f) and the third primitive event in Figure 4.13 (j).

Meanwhile, another object labeled 21 is detected with the first primitive event in the second camera view, as in Figure 4.13 (g). Then it is correctly labeled in the third camera [Figure 4.13 (h-i)], and in the first camera [Figure 4.13 (k-l)]. The second primitive event of Object 21 is detected in the third camera view in Figure 4.13 (m), and the third primitive event is detected in the first camera view in Figure 4.13 (n).

5) Event Scenario Composed of Four Primitive Events Spanning Three Camera Views

The defined event scenario is detecting a person crossing a tripwire in the second view, *and then* going through a region of interest in the third camera view, *and then* going through

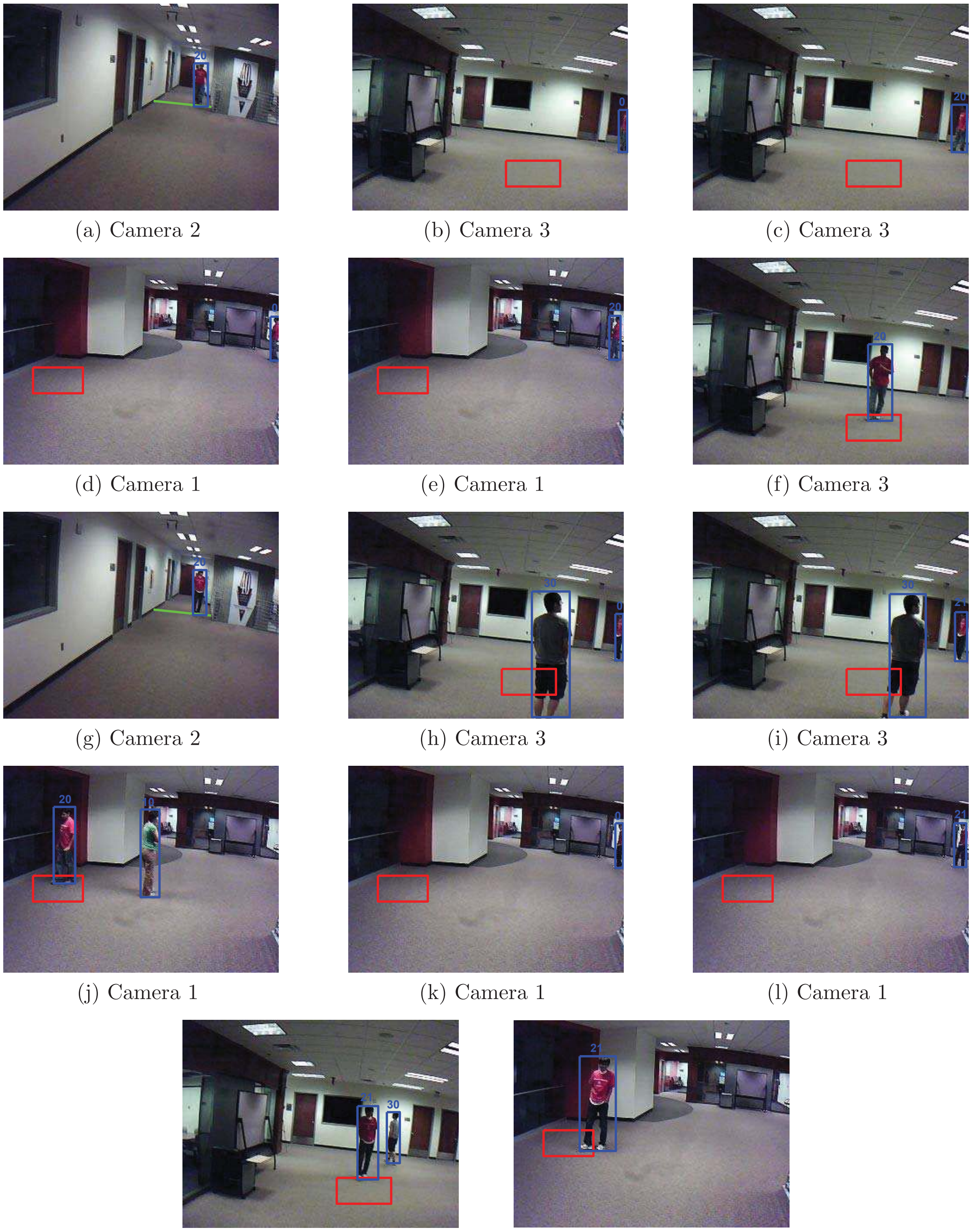
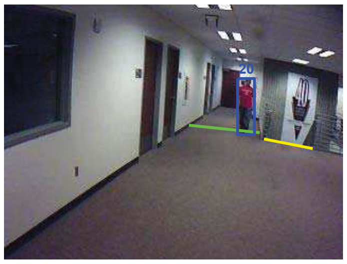
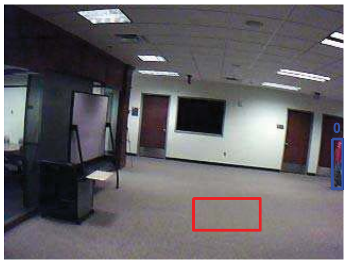
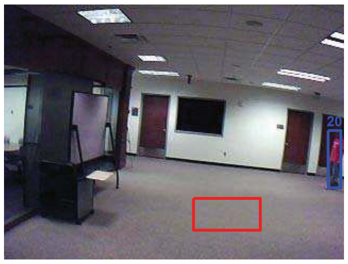
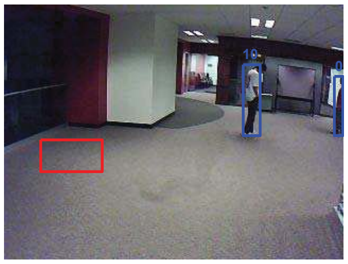

(a) Camera 2 (b) Camera 3 (c) Camera 3
(d) Camera 1 (e) Camera 1 (f) Camera 3
(g) Camera 2 (h) Camera 3 (i) Camera 3
(j) Camera 1 (k) Camera 1 (l) Camera 1
(m) Camera 3 (n) Camera 1

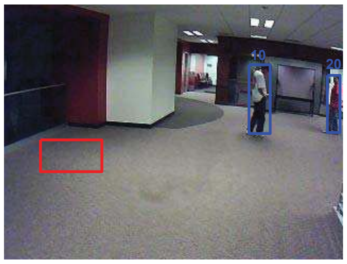
Figure 4.13: An event scenario composed of three primitive events spanning three different camera views.

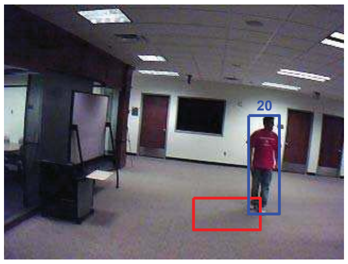

(a) Camera 2

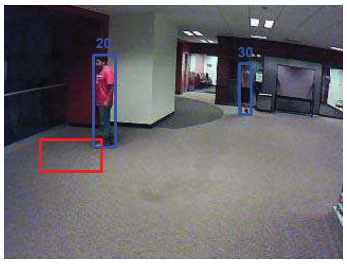

(b) Camera 3

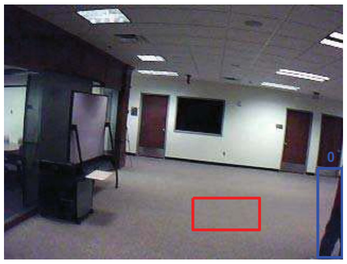

(c) Camera 3

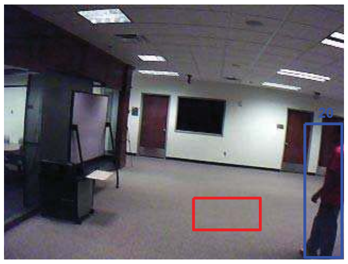

(d) Camera 1

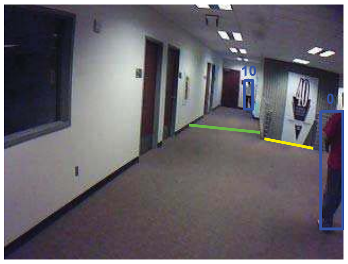

(e) Camera 1

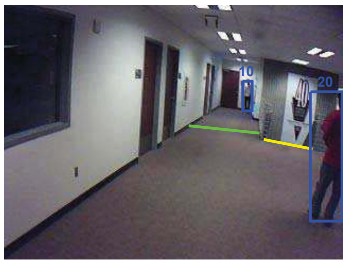

(f) Camera 3


(g) Camera 1


(h) Camera 3


(i) Camera 3


(j) Camera 2


(k) Camera 2

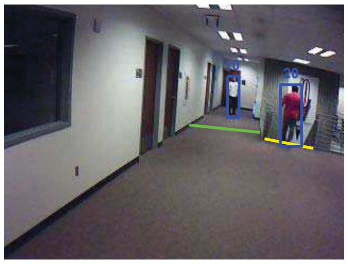

(l) Camera 2

Figure 4.14: An event scenario composed of four primitive events spanning three different camera views.

another region of interest in the first camera view, *and then* finally leaving the scene by crossing a tripwire in the second camera view. Figure 4.14 shows the key frames, and the detection of the each primitive event.

In Figure 4.14 (a) the first primitive event is detected with Object 20 in the second camera view. The third camera sees Object 20 and retrieves the correct label from the second camera [Figure 4.14 (b-c)] and the first camera sees Object 20 and retrieves the correct label from the third camera [Figure 4.14 (d-e)]. The second primitive event is detected with Object 20 in the third camera view as shown in Figure 4.14 (f). The third primitive event is detected with Object 20 in the first camera view in Figure 4.14 (g). Later on, Object 20 return to the third camera view in Figure 4.14 (h-i) and then return to the second camera view in Figure 4.14 (j-k). Finally, the fourth primitive event is detected in the second camera view with Object 20 in Figure 4.14 (l).

4.3 Conclusions

We measured the operating currents of the cameras for different scenarios and evaluated the power consumption of the system. We analyzed the power consumption during different parts of the processing and during different message exchanges between camera nodes. We also evaluated the power consumption when tracking different number of objects, and when tracking different-sized objects. Since power is a limited resource for embedded smart cameras, this analysis is very important. Additional to the power consumption, we also analyzed the energy consumption, because efficient algorithms also reduce the processing time. Thus, the processing time for one frame will be shorter, which results in a lower energy consumption for one frame.

We also presented examples of consistent labeling and updating the location of occluded or lost objects from other cameras. We showed examples of detecting composite and spatio-temporal event scenarios spanning multiple camera views. Examples include event scenarios that are composed of two, three and four primitive events spanning two or three different camera views. Experiments were performed by tracking remote-controlled cars as well as multiple people. The results show the success of the proposed system.

Part III

Distributed Object Tracking with Non-overlapping Camera Views

Chapter 5

Real-time Distributed Tracking with Non-Overlapping Camera Views

5.1 Introduction

In previous chapters, we proposed a wireless smart embedded camera system for real-time object tracking. The cameras in the system are assumed to have partially overlapped views and share a common planar ground, thus the homography matrix can be used to recover the FOV lines and maintain the consistent labeling.

With the success of object tracking with overlapping views, it is natural to attempt to extend the framework to non-overlapping views. In Chapter 1, the related work of object tracking with non-overlapping views has been reviewed. Although, methods have been developed that focus on building statistical or non-statistical models for object matching, much less attention has been paid to designing and implementing algorithms for real-time applications, and distributed processing. In this chapter, we propose a real-time, distributed system for multi-object tracking with non-overlapping cameras.

In our system, each camera is connected to a PC and the PCs communicate with each other through TCP/IP. Similar to previously introduced embedded camera system, each camera performs multi-object tracking individually, and exchanges data in a P2P manner. We combine multiple features to match objects across non-overlapping views. These features are side of entry, color histogram, height, moving direction, speed and travel time. These are extracted and transmitted to neighboring cameras at different points in time while the object is being tracked in the current camera view. In the next camera, similarity scores will be computed for each feature, and an overall similarity score will be obtained by taking a weighted sum of the individual feature similarities. The system is first trained to learn several parameters including camera placements, ratio of heights on different camera views, average traveling time information from one camera view to the other and a threshold for the overall similarity score.

This is our first prototype system that our tracking algorithms are extended to the scenarios with non-overlapping views. The focus of this work is to build up a system for successful real-time object tracking. A more sophisticated and more robust framework will be introduced in Chapter 6 and Chapter 7.

5.2 Object Tracking across Non-overlapping Cameras

Firstly, like the previously proposed wireless embedded camera system, every camera in the system performs the background subtraction, blob forming and tracking individually, and inherits the same algorithms as described in Chapter 2. As there are no other cameras share the view, each camera has to resolve the occlusion problem by it own. This will be discussed in the more sophisticated framework presented in Chapter 7. Here, the only problem that needs to be solved is the consistent labeling. Since it is difficult to retrieve the geometric relation between the cameras, we have to find other cues to re-identify the objects. This problem is also referred to as *Object Re-identification* problem.

We combine similarities of multiple features to match objects across non-overlapping views. These features are side of entry, color histogram, height, moving direction, speed and travel time. The system is first trained to learn several parameters which are listed below:

Camera Configuration: The IDs of a camera's right and left neighbors should be known beforehand, or are learned during training. These IDs are saved in two separate lists for the right and left sides.

Object's height ratio in two cameras: An average ratio, R_H, for the objects' heights at the entry locations of two different camera views is learned in the training stage.

Ratio of travel times: In the training stage, the amount of time it takes for an object to go through the first camera's view and through the blind region is measured and saved as variables T_C and T_B, respectively. Then the ratio R_T is calculated by $R_T = T_B/T_C$. The average for R_T is found for different objects.

Threshold for similarity: We combine multiple features by calculating a weighted sum of the similarity score of each feature. A proper threshold for the overall similarity is learned during training stage.

In Section 5.2.1, we explain how we use each evidence to match objects during testing; we describe the work flow of a camera, and explain the communication between cameras in details in Section 5.2.2.

5.2.1 Weighted Matching Criteria

Figure 5.1 shows an example of the work flow of a camera in the testing stage. As can be seen, this camera sends object's height, color histogram, speed/travel times at different points in time. The next camera receives these data and saves them in a structure as a candidate, with the label of this object.

When a new tracker is created in the next camera, it tries to find a match among the N received candidates. Each candidate, $i \in \{1 \ldots N\}$, has K different features that are

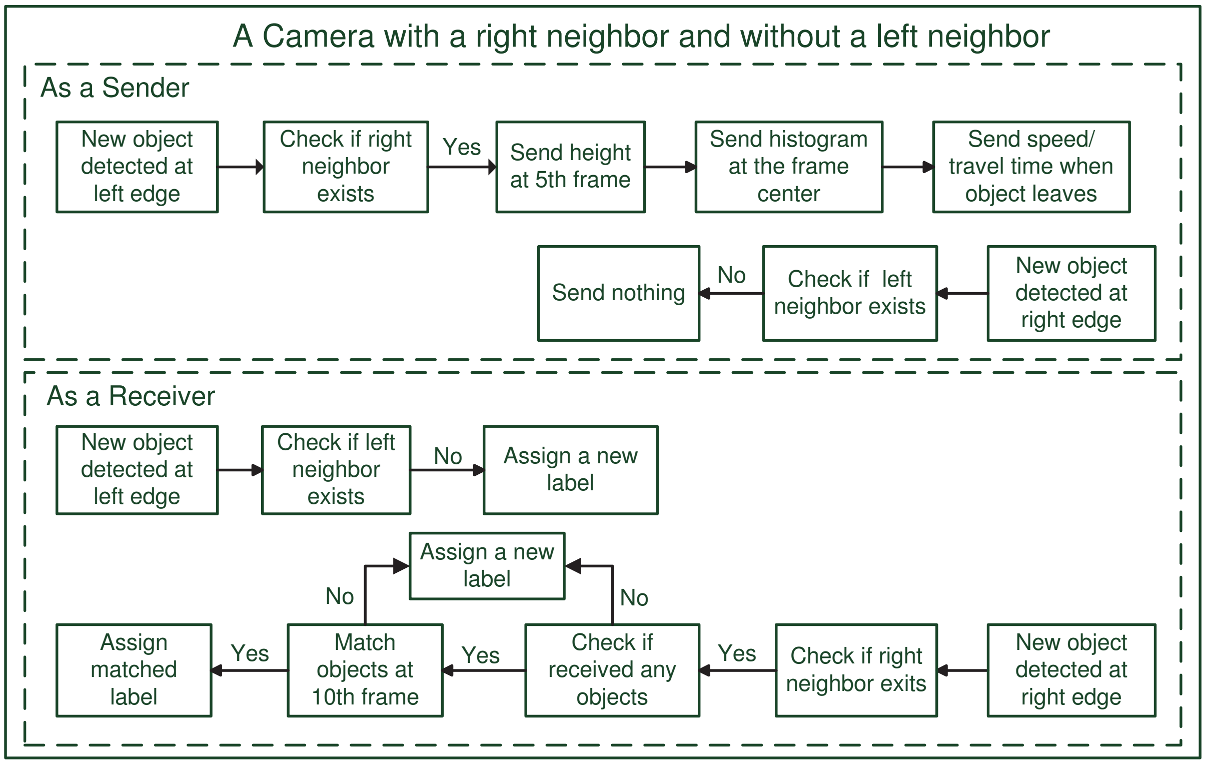

A Camera with a right neighbor and without a left neighbor
As a Sender
New object detected at left edge
Check if right neighbor exists
Yes
Send height at 5th frame
Send histogram at the frame center
Send speed/ travel time when object leaves
Send nothing
No
Check if left neighbor exists
New object detected at right edge
As a Receiver
New object detected at left edge
Check if left neighbor exists
No
Assign a new label
Assign a new label
No
No
Assign matched label
Yes
Match objects at 10th frame
Yes
Check if received any objects
Yes
Check if right neighbor exits
New object detected at right edge

Figure 5.1: The work flow of object matching in one camera

described below. For each feature, $j \in \{1 \dots K\}$, a similarity score s_j^i is calculated and is given a weight w_j. To combine multiple features, an overall similarity score is calculated. The best matching candidate object O is found by

$$O = \arg\max_{i \in N} \sum_{i \in N, j \in K} (w_j s_j^i) \tag{5.1}$$

If the overall similarity score of the object O is greater than a pre-defined threshold, then the candidate object O is matched to this tracker, and the tracker is assigned the label of O.

Entry location and moving direction From checking the side of an object's entry and its moving direction, a camera knows two types of information: 1) which camera view(s) may this object come from and 2) which camera(s) could possibly see this object when it leaves this camera's view. By knowing the former, the current camera checks if it has already received any object's information from the neighbor who may have been tracking this object previously. By knowing the second type of information, the current camera will send the feature data to another camera who will possibly track this object next. The entry side information is for judging if this object needs to be matched to the received candidates from the previous camera, or if the current camera should send any data to the next camera. It doesn't contribute to the overall similarity score.

Color histogram The similarity of color histograms of the newly detected object and of the received candidates is calculated using (2.1). Since similarity between color histograms is a main evidence for appearance similarity, it is given the highest weight among all the criteria. The color histogram is built and transmitted to the next camera when the object is in a good position in the view, such as with a better resolution or when it is not occluded. For instance, if the object is in the merge state, the color histogram will not be transmitted until the object gets out of the merge.

Height If h_1 and h_2 are the object's heights in the first camera and second camera views, respectively, and R_H is the ratio learned in the training stage, the similarity s_H between two heights is calculated by:

$$s_H = 1 - \left| \frac{h_2 - h_1 * R_H}{h_1 * R_H} \right| \tag{5.2}$$

Speed and travel time As shown in Figure 5.1, the current camera sends the object's speed v_1 and also the travel time t_c when this object leaves its FOV. The next camera records the time it receives this speed/travel time packet as t_{rcv}. When this object enters the next camera view, the time it is detected, t_{det}, is also recorded. Then the travel time of this object in the blind region, t_b, is calculated by $t_b = t_{det} - t_{rcv}$. Then the similarity s_T of the travel time is computed by:

$$s_T = 1 - \left| \frac{t_b - t_c \cdot R_T}{t_c \cdot R_T} \right| \tag{5.3}$$

where R_T is the ratio of the travel time in blind region to the travel time in the first camera learned during training. Then the similarity s_V between speeds v_1 and v_2 is calculated by:

$$s_V = 1 - \left| \frac{v_1 - v_2}{v_1} \right| \tag{5.4}$$

5.2.2 Communication and Work Flow between Cameras

Communication between camera nodes is implemented via TCP/IP. Communication is performed in a separate thread in parallel and share memory with frame processing. Thus, even though we transmit relatively larger data sets such as 3-D histograms, the transmission is finished before the object arrives. It does not create latency for the real-time tracking. Every camera node is given a unique ID and maintains two lists containing the IDs of its right and left neighbors.

There are three types of packets for transferring color histogram, object height and speed/travel time. For each type of packet, the packet header contains a synchronization word, node ID, packet type, object label and the payload length. The payload, which may be the color histogram, object height or speed/travel time, follows the packet header.

For matching objects across multi-camera views, each camera node needs to receive information from the previous camera as well as send information to the next camera. In other words, a camera node acts as a *sender* and a *receiver* at the same time.

Figure 5.1 shows an example of the work flow that one camera performs in our real-time system. In this example, the camera shown is the left one in a two-camera configuration, i.e., this camera has no neighbors on its left, and has one right neighbor. If more cameras are added to the system, the cameras in between just need to combine both the left camera's tasks and the right camera's task in one camera.

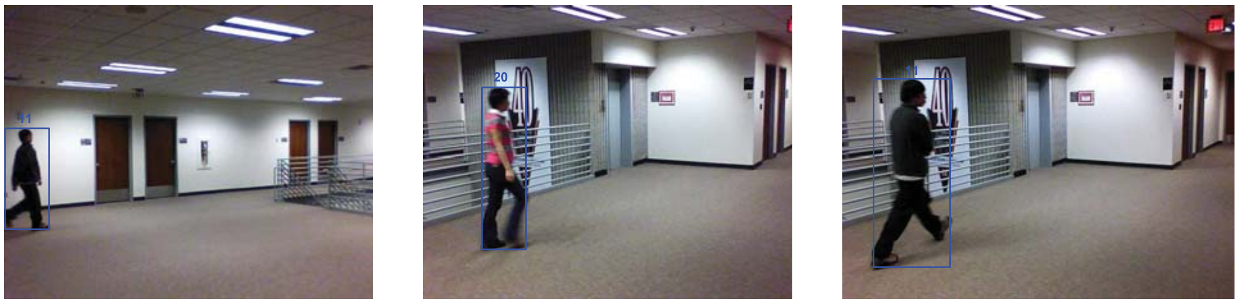
As a Receiver When there is a new object detected on one side of the view, the camera will check if there is a neighbor camera on this side. If no, a new label is assigned to the object immediately. If there is a neighbor on that side, this object will be assigned a temporary label 0 and start pending for the matching process until n^{th} frame. n is a small number such as 5 or 10 depending on the frame rate. The reason for waiting n frames is to be able to see the full shape of the object, and obtain a more accurate color histogram. If the object is in the merge state, the camera will wait until merge is resolved. After performing the matching process at the n^{th} frame, if it finds a match in the received candidates, the matched object's label will be assigned. If no match is found, a new label will be assigned.

As a Sender When there is a new object detected on one side of the view, the camera will check if there is a neighbor camera on the other side. If not, the camera sends nothing about this object. If there is, this camera will first send the object's height at the m^{th} frame after the object is detected. m is also a small integer such as 5. Then, the color histogram of the object is sent when it arrives the center region of the frame. And as we stated in Sec 5.2.1, the speed and travel time will be sent when the object leaves the view. These different sets of data will be saved in an object structure in the receiving camera, and will be used in the matching process.

5.3 Experimental Results

We performed different experiments with a setup consisting of two non-overlapping cameras. The weights of color histogram, height, travel time and speed are 0.5, 0.2, 0.2 and 0.1, respectively. The threshold of the overall similarity for object matching is 0.8. The height of the object is sent at the 5^{th} frame. The object matching is performed at frame $n = 10$.

Figure 5.2 shows an experiment during which two people enter from the left side of the right camera at different times. The person with label 11 was being tracked by the left camera, as seen in Figure 5.2 (a). The features of this person was sent to the right camera



(a) (b) (c)

Figure 5.2: Two people enter the view of the right camera from the same side (a) Person 11 is being tracked by the left camera (b) another person enters the right camera's view before Person 11 does, and gets the label 20 (c) Person 11 enters the right camera's view and is assigned the correct label.

according to the flow chart shown in Figure 5.1. After person with label 11 leaves the FOV of the left camera, another person enters the right cameras view before the person with label 11 does (Figure 5.2 (b)). Since new person enters at the left side and the right camera has received some candidate data, this person goes through the matching process and cannot be matched to the received candidate data. Thus, it is assigned a new label 20. Then, the person who has left the first cameras view, enters the view of the right camera, and is assigned the correct label 11.

Figure 5.3 shows an experiment during which two objects with similar appearance are tracked by the left camera. Object with labels 11 and 12 are tracked as seen in Figure 5.3 (a-b) and (c-d). Thus, the right camera receives the data for two candidates before they arrive. Later in Figure 5.3 (e) Object 11 enters the view of the right camera, and is assigned a temporary label 0. Then, this object is matched to the correct candidate and is assigned the correct label 11 as shown in Figure 5.3 (f).

Figure 5.4 shows another experiment during which two people enter the right camera view from opposite sides. Person 11 is tracked in the left camera first, as seen in Figure5.4

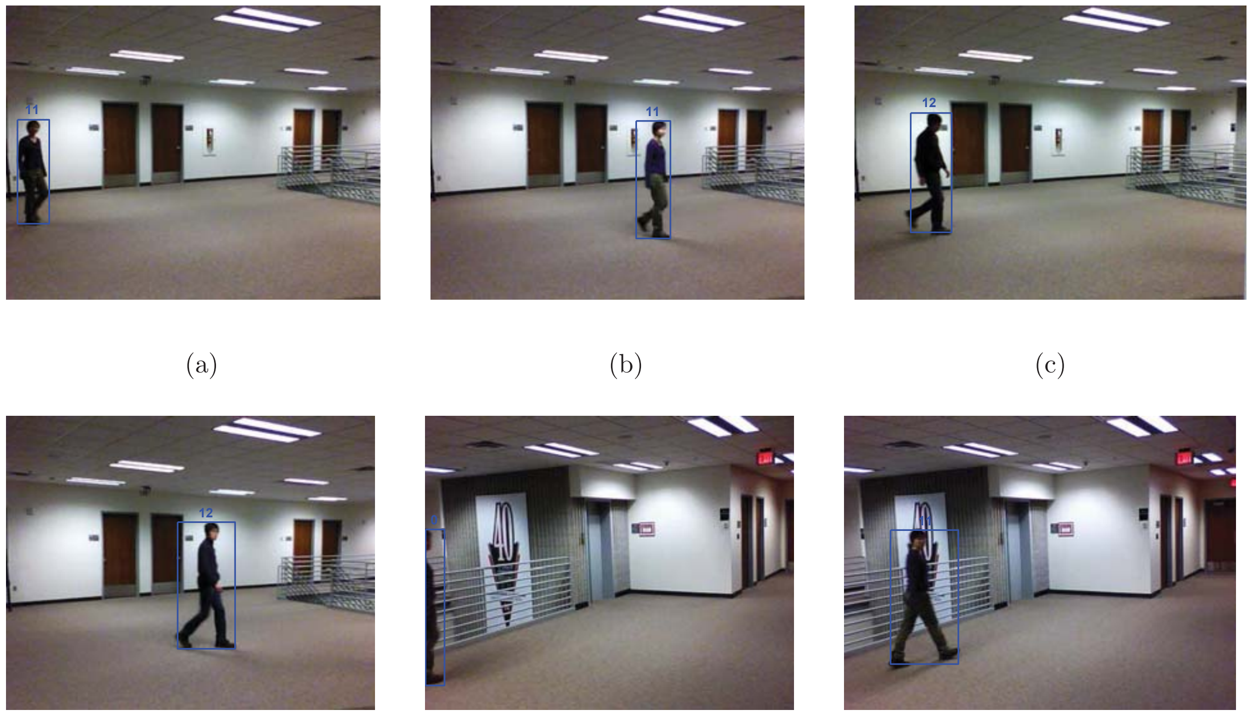
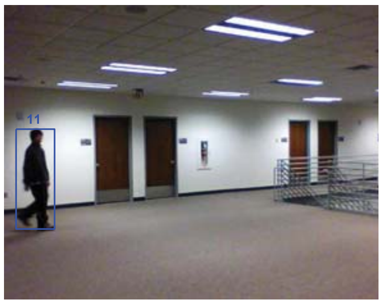
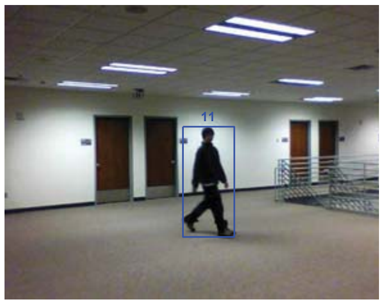


Figure 5.3: Two people tracked by the left camera and one tracked by the right camera (a-b),(c-d) People with labels 11 and 12 are tracked by the left camera; (e) a person enters the right camera view and is assigned a temporary label 0; (f) new person is assigned the correct label 11.

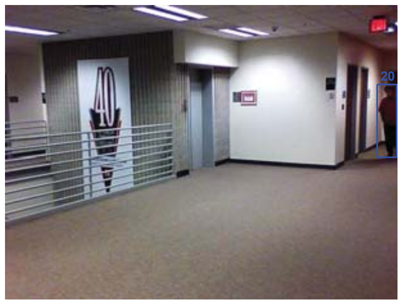
(a-b). Then, as seen in Figure5.4 (c), a person enters the right cameras view from the right side. Right camera checks the side of entry, and assigns it a new label 20 since there is no neighbor camera on the right. Then, the person who has left the left cameras view , enters the view of the right camera, and is assigned a temporary label 0 first. Two people merge and stay merged from frame 5 to 7. At the 10^{th} frame, after people split, matching process is performed, and the person who left the other camera, is assigned the correct label 11.



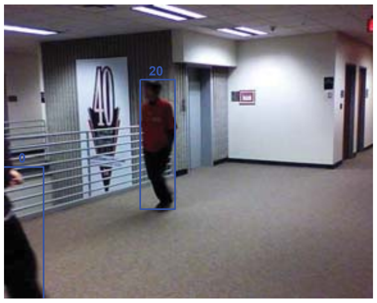
(a)



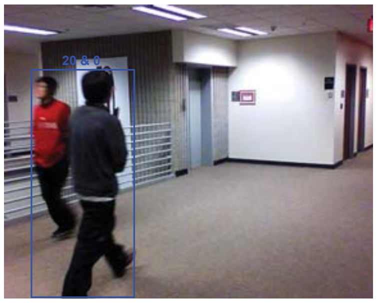
(b)



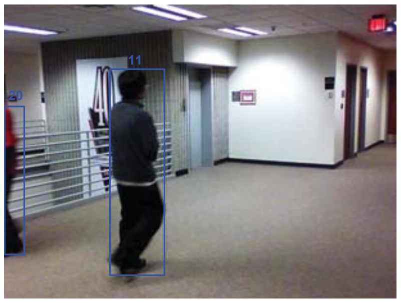
(c)



(d)



(e)



(f)

Figure 5.4: Two people enter at the opposite side of the right camera (a-b) Person 11 is being tracked by the left camera; (c) another person enters the right cameras view at the right side and is assigned label 20; (d) person 11 enters the right cameras view and is assigned a temporary label 0; (e) person 20 and person 0 merge during frames 5 to 7; (f) after the split, person 0 is assigned the correct label 11 by matching process.

5.4 Conclusions

We presented a distributed real-time system for tracking across non-overlapping camera views. Each camera attaches to a PC, performs multi-object tracking, and exchanges data in a P2P manner via TCP/IP. With the difficulty of recovering the geometric relations of object correspondences, multiple features are employed and combined by a weighted sum of the similarities. These features are side of entry, color histogram, height, moving direction, speed and travel time. The system is first trained to learn several parameters including camera placements, ratio of heights on different camera views, average traveling time information from one camera view to the other and a threshold for the overall similarity score. Real-time experiments with a two-camera setup are performed.

Chapter 6

A More Robust Algorithm for Object Re-identification

We presented a distributed tracking system wherein each camera performs object tracking individually and employs multiple features for object re-identification in a real-time manner in last chapter. Although this system is a good step toward a real-time object tracking system for non-overlapping views, the algorithm for object matching is not robust enough. Just a few features are selected and combined by weights. The weights are determined empirically and do not adapt to the environment changes.

In this chapter, a more robust algorithm is proposed for object matching. More features are utilized to improve the robustness and modeled more accurately. The weights are learned based on the reliability of the features, and updated adaptively over time.

When an object leaves a camera's view, this camera creates and sends a message packet containing the appearance features, exit time and the label of this object. Appearance features, travel-time evidence and the matching procedure are described in detail below. A scenario of wide-area vehicle tracking is presented as an example.

6.1 Multi-feature Object Matching

6.1.1 Color Histogram

In last chapter, 3D RGB histogram was employed as the color descriptor. However, RGB histogram is sensitive to illumination change as every channel contains the brightness information. On the contrary, HSV color space is more robust due to its separation of the brightness from the chromaticity. In this chapter, we performed a comparison between RGB and HSV color models, and elected 3D HSV histogram. Each bin in the histogram corresponds to an (H,S,V) range. The comparison results are shown in Chapter 7.4.1.

Due to the light reflection and reliability of foreground segmentation, the color of the foreground pixels on the edges of the objects is less reliable than that of the pixels near the center. If an object's bounding box is centered at y with the height h and the width w, the foreground pixels near to the center are assigned higher weights in the color histogram [80]. The weight assigned to each foreground pixel is

$$\omega(r) = \begin{cases} 1 - r^2 & \text{if } r < 1 \\ 0 & \text{otherwise} \end{cases}$$

where $r = |x - y|/a$, $|x - y|$ is the distance between the pixel x and the center y of the bounding box, and $a = \sqrt{h^2 + w^2}/2$ is the maximum distance from a point inside the bounding box to the center. Then, the histogram of the object is computed by

$$\hat{p}_u(y) = K \sum_{u=1}^{m} \omega\left(\frac{|x - y|}{a}\right) \bullet \delta(b(x) - u), \quad u = 1, \ldots, m \tag{6.1}$$

where m is the size of the histogram, δ is the Kronecker delta function, $b(x)$ denotes the color histogram index of pixel x, and K is the normalization factor defined as

$$K = \frac{1}{\sum_{u=1}^{m} \omega(\frac{|x-y|}{a})}$$

It is used to ensure that

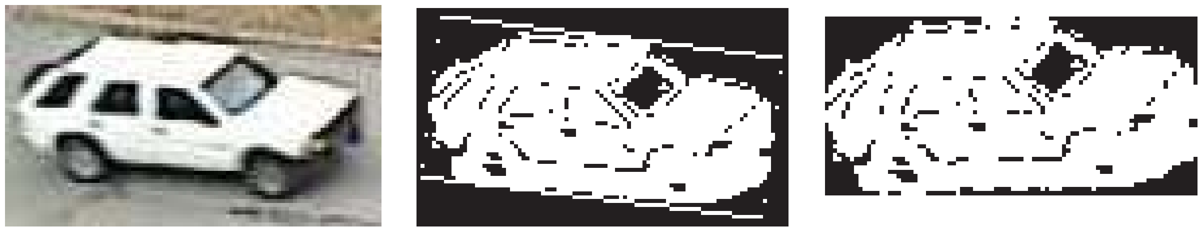
$$\sum_{u=1}^{m} \hat{p}_u(y) = 1$$

The similarity score s_C between two histograms is calculated by (2.1).

6.1.2 Texture

Texture is an important characteristic for the analysis of image properties. A wide variety of measures for discriminating textures have been proposed. In [84], Histogram of Oriented Gradients (HOG) descriptor is proposed for human detection. The local object textures are modeled by calculating the distribution of the local intensity gradients and the edge directions. HOG is one of the best features for capturing edge and shape information. One weakness of HOG is that image regions with different contents may lead to a similar gradient histogram, due to the nature of the histogram. LBP is considered as a complementary feature for HOG and has been combined with HOG for human detection [116, 87]. LBP is originally proposed by Ojala et al. [106]. It is invariant to monotonic gray-level changes and can be computed efficiently. For the human detection application, a sliding window method is normally employed. But in our case, since a foreground blob is already formed for each object, there is no need for searching the whole frame with sliding windows of different sizes. Thus, computation of texture features is not a time-consuming task in our system. The foreground blob is divided into a fixed number of cells to form the HOG and LBP descriptors.

One factor that influences the texture descriptors is the different camera angles in different



(a) The object (b) Foreground pixels (c) Rotated pixels

Figure 6.1: Example of the angle correction for texture descriptors.

views. To address this problem in vehicle tracking, we rotate the foreground pixels by the angle of the lane before building the HOG and LBP descriptors. Figure 6.1 shows an example of the angle correction. Figures 6.1 (a) and (b) show the color image of the object and the segmented foreground pixels, respectively. The bounding box is also drawn along the direction of the lane in Fig. 6.1 (b). If the angle between the lane and x-axis is denoted by θ, the foreground image (Fig. 6.1 (b)) is warped by the rotation matrix

$$\begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

Figure 6.1 (c) shows the rotated foreground pixels.

6.1.2.1 Histogram of Oriented Gradients

A foreground blob is divided into n cells. For each cell, an m-bin HOG is built as described in [84]. Each bin in the HOG corresponds to an orientation spanning. The combination of n HOGs forms the HOG descriptor, with the size of $m * n$ bins. An important step described in [84] is the local normalization, which helps to reduce the impacts of the illumination and contrast variations. The cells are grouped into blocks and block-based normalization is performed.

6.1.2.2 Local Binary Patterns

A Local Binary Pattern (LBP) operator is obtained by thresholding the neighborhood of each pixel with the center pixel value and considering the result as a binary number [107]. An LBP is usually denoted by $LBP_{P,R}$, where (P,R) indicates the pixel neighborhood with P sampling points in a circle of radius R. The binary pattern is transformed into a decimal number. Figure 6.2 illustrates an example of computing the binary pattern and its decimal value.

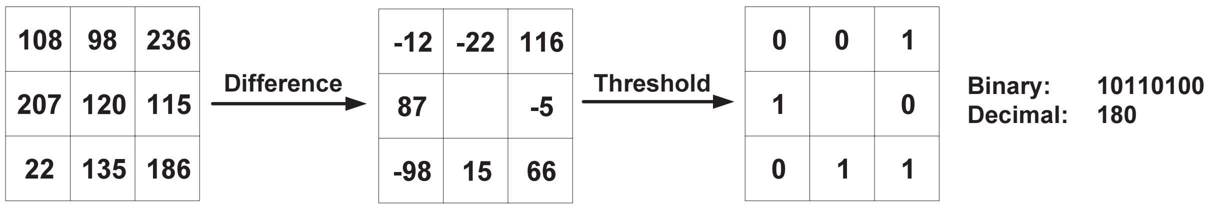


Figure 6.2: An example of LBP operator.

To reduce the length of the feature vector and implement a simple rotation-invariant descriptor, an extension to the original operator has been introduced called *uniform patterns* [107]. A local binary pattern is called uniform if the binary pattern contains at most two bitwise transitions from 0 to 1 or vice versa when the bit pattern is considered circular. When computing the LBP histogram, each uniform pattern is assigned a separate bin and all nonuniform patterns are put in a single bin. By using uniform patterns, the number of bins of the LBP histogram is reduced to 59. As suggested in [107], the best performance is achieved by the (8,1) neighborhood using uniform patterns.

6.1.2.3 Texture Similarity

HOG and LBP are treated as two separate features for object matching. If two HOG descriptors denoted as A and B, the cosine similarity of HOG, s_H, is calculated by

$$s_H = cos(\theta) = \frac{A \times B}{\|A\|\|B\|} \tag{6.2}$$

The similarity score s_L of the LBP descriptors is also calculated by (6.2).

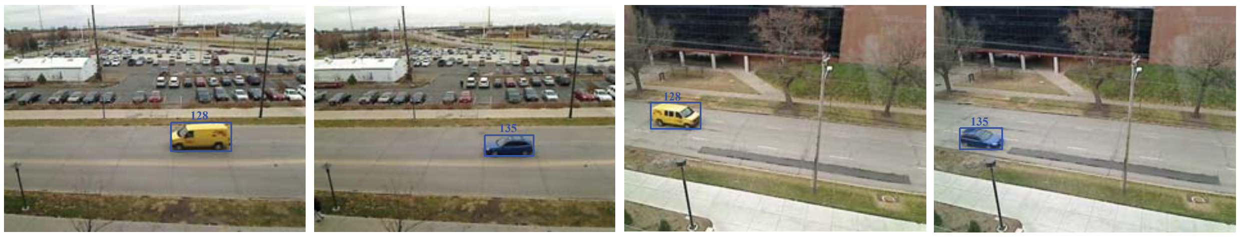
6.1.3 Aspect Ratio and Size

The aspect ratio is a useful feature that can be used to differentiate the compact cars and large-sized vehicles in a traffic application. If the aspect ratios of the two objects are a_1 and a_2 respectively, the similarity score is calculated by

$$s_A = 1 - \left| \frac{a_1 - a_2}{a_1} \right| \tag{6.3}$$

The object size is also used as an appearance feature. It is the number of pixels in the detected foreground blob. The size of an object varies depending on its distance from the camera. Figure 6.3 shows an example, where the cars coming from Camera 1 travel in different lanes in the view of Camera 3. Let the sizes of the two objects be s_1 and s_2 in the previous camera and current camera, respectively. Also, let f_l denote the size ratio of the objects, i.e. $f_l = s_1/s_2$. f_l will be different depending on the lane the vehicle is traveling in. The closer the object is to the camera, the bigger s_2 is, and thus the smaller f_l is. The f_l for different lanes are learned during training. The similarity score is then calculated by

$$s_S = 1 - \left| \frac{s_1 - s_2 * f_l}{s_1} \right| \tag{6.4}$$



(a) Camera 1 (b) Camera 1 (c) Camera 3 (d) Camera 3

Figure 6.3: Example of the different size factors of the cars in different lanes.

6.1.4 Travel Times

In traffic flow, travel times of the cars across the blind regions are dependent on the traffic conditions and traffic lights. The travel time of a car is the difference between the time the car enters the current camera view and the time the car exits the previous camera view. Based on the domain knowledge, either a single Gaussian model or a Gaussian Mixture Model (GMM) can be employed. If some traffic conditions are present in the blind region, such as intersections and traffic lights, a single distribution is not accurate to model the travel times. Thus, a GMM becomes necessary.

The parameters of the GMM are estimated in the training stage by using an Expectation-Maximization(EM) algorithm. These parameters are the number of Gaussians (K), the mean μ and the variance σ^2 for each Gaussian distribution. Given N different transition times t_n ($n \in \{1 \ldots N\}$), the parameters of the j^{th} distribution are calculated by two steps at $(i+1)^{th}$ iteration until the convergence:

1. E step:

$$p(j \mid t_n) = \frac{\omega_j^{(i)} P(t_n \mid j; \mu_j^{(i)}, \sigma_j^{(i)})}{\sum_{k=1}^{K} \omega_k^{(i)} p(t_n \mid k; \mu_k^{(i)}, \sigma_k^{(i)})} \tag{6.5}$$

2. M step:

$$\omega_j^{(i+1)} = \frac{1}{N} \sum_{n=1}^{N} p(j \mid t_n) \tag{6.6}$$

$$\mu_j^{(i+1)} = \frac{\sum_{n=1}^{N} t_n p(j \mid t_n)}{\sum_{n=1}^{N} p(j \mid t_n)} \tag{6.7}$$

$$(\sigma_j^2)^{(i+1)} = \frac{\sum_{n=1}^{N} (t_n - \mu_j^{(i+1)})^2 p(j \mid t_n)}{\sum_{n=1}^{N} p(j \mid t_n)} \tag{6.8}$$

For calculating the similarity score of the travel times between a candidate and detected object, the travel time t is obtained by

$$t = t_{o,e} - t_{c,l} \tag{6.9}$$

where $t_{o,e}$ is the time the object enters the current camera view, and $t_{c,l}$ is the time the candidate leaves the previous camera view. Similar to the clustering or classification problems that adopt a GMM, a Gaussian distribution that yields the highest probability is selected as the distribution that this time value belongs to. Considering the various factors that may influence the travel time, a travel time that falls into the range of mean $\pm$ one standard deviation does not get penalized, i.e. it will have a score of 1. Otherwise, the similarity score of travel time is calculated by

$$s_{TT} = e^{-\frac{(x-\mu)^2}{2\sigma^2}} / e^{-\frac{(\mu+\sigma-\mu)^2}{2\sigma^2}} = e^{-\frac{(x-\mu)^2-\sigma^2}{2\sigma^2}} \tag{6.10}$$

where μ and σ are the mean and the standard deviation of the Gaussian distribution component that this object belongs to.

If the domain knowledge in the blind region is simple, i.e. there are no intersections or traffic lights in the blind region, and vehicles move continuously, a single Gaussian distribution is adopted. The similarity of the travel time is also evaluated by (6.10).

6.1.5 Weighted Matching Criteria

When a new object o enters into the view of the current camera from side S, we first check if there are candidate packages sent by the neighboring camera(s) on side S. If there are, the current camera tries to find a match among the C received candidates. Each candidate, $c \in \{1 \dots C\}$, has different features that are described above, namely color histogram (CH), HOG (H), LBP (L), aspect ratio (A), size (S) and travel time (TT). For each feature $j \in F = \{CH, H, L, A, S, TT\}$, a similarity score $S_j(c, o)$ is calculated and is given a weight w_j. To combine multiple features, an overall similarity score is calculated. The best matching candidate $\hat{O}(o)$ for object o is found by

$$\hat{O}(o) = \arg \max_{c \in \{1 \dots C\}} \sum_{c \in \{1 \dots C\}, j \in F} w_j S_j(c, o) \tag{6.11}$$

If the overall similarity score of the candidate $\hat{O}(o)$ is greater than a pre-defined threshold, then the candidate $\hat{O}(o)$ is matched to this tracker, and the tracker is assigned the label of $\hat{O}(o)$. Otherwise, a new label is assigned to this tracker.

With this method, not only object association problem can be addressed, but also new objects are differentiated from the already observed ones.

6.2 Adaptive Parameter Updating

6.2.1 Online Updating of Travel Time Models Using Confidence Score

For traffic scenarios, the travel times may change during different times of the day. The Gaussian models of travel time need to be updated adaptively with changing traffic conditions. If a GMM is employed and trained during the training stage, the mean and variance of each Gaussian are updated online by using the travel time of the matched object. Considering the existence of false positives, the matched object with a higher similarity score, which indicates higher confidence in matching, should have more impact on the travel time model; or vice versa. The mean and variance are updated by

$$\mu_{new} = (1-\lambda)\mu_{old} + \lambda t_{match} \tag{6.12}$$

$$\sigma^2_{new} = (1-\lambda)\sigma^2_{old} + \lambda(t_{match} - \mu_{new})^2 \tag{6.13}$$

where t_{match} is the travel time of the matched object; λ is the update parameter in the range of $[0, 1]$, which is defined as $\lambda = \alpha \cdot S$. α is a constant update factor which is a small number such as 0.05. S is the confidence factor, which is equal to the overall similarity of a matched pair. Thus, the matched objects with high similarity scores contribute more to the parameter updating.

6.2.2 Adaptive Weight Estimation

There are many different cues that could be used for object re-identification. An individual similarity for each cue can be evaluated, and then these similarities need to be combined for a final evaluation. Weighted sum is a simple but effective way to achieve the information integration. The problem is how to assign the weight to each cue. It is intuitive that a more reliable cue should be assigned a higher weight. Observers or algorithms must evaluate the degree of reliability of each cue and assign higher weights to the more reliable cues. The reliability is normally context sensitive and changes with the environment [90]. There have been some work to explore evaluating the reliability of the cues and combine them in a self-organized manner. [113] proposed a method to integrate five visual cues for face detection. In their work, each feature generates a two-dimensional salient map by comparing with a prototype that describes the appearance of the face. Then, different features are combined by a method called *Democratic Integration*. We apply a similar method in our system to evaluate the weights of the multiple features, and extend this method to an online adaptive weight estimation, which suits the real-time systems.

To find a match among the received candidates for an incoming object, an overall similarity score is calculated between the object and each candidate. The overall similarity score is the weighted sum of various features, and we evaluate the reliability of each feature as follows: if the similarity of a feature is in accordance with the result of the overall similarity score, this feature is considered as reliable. For a correctly matched pair of an object and the candidate, the higher the similarity of a single feature, the more reliable that feature is. On the contrary, if the similarity of a feature is low enough, it is considered having zero reliability. To compute the reliability quantitatively, the $q$uality of a feature is introduced. The similarity score of a feature for the best matched candidate is compared with the average score of this feature over all candidates. The higher the score (as compared to the average), the more it contributes to the running evaluation of the reliability of this feature. The quality of this feature is the difference between the similarity score and the average. On the other hand, if the similarity score is lower than the average, the quality is set to zero. The quality of each feature is calculated in this way, and then normalized over all the features. The reliability of a feature is the running average of the quality of this feature. Since the qualities are normalized, the reliability scores will add up to 1. The formulation of the method is described below.

For Object o, given C candidates, a similarity score $S_j(c,o)$ is calculated for each feature $j \in F = \{CH, H, L, A, S, TT\}$ of each candidate $c \in \{1 \dots C\}$. From the descriptions in Section 6.1, we know that $0 \leq S_j(c,o) \leq 1$. The overall similarity result of each object-candidate pair is

$$R(c,o) = \sum_{j \in F} r_j S_j(c,o) \tag{6.14}$$

where r_j is introduced as the reliability of the j^{th} feature, with $\sum_j r_j = 1$. Then the best match for Object o is found by

$$\hat{O}(o) = \arg \max_{c \in \{1 \ldots C\}} \{R(c, o)\} \tag{6.15}$$

When (6.11) is compared with (6.14) and (6.15), we can see that the weight w_j assigned to each feature is the reliability r_j of that feature.

To estimate r_j, the quality $\tilde{q}_j(o)$ is introduced, and $0 \leq \tilde{q}_j(o) \leq 1$. The quality $\tilde{q}_j(o)$ measures how successful the feature predicts the result or how much it agrees with it. It is calculated by

$$\tilde{q}_j(o) = \mathcal{R}(S_j(\hat{O}(o), o) - < S_j(c, o) >)) \tag{6.16}$$

where $< \ldots >$ denotes an average over all candidate similarity scores for Object o; $\mathcal{R}$ is the ramp function:

$$\mathcal{R}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$

In (6.16), the similarity score of each feature for the best matched candidate is compared to the average score of this feature over all candidates. If the score of this feature is greater than the average, the quality is the distance to that average; otherwise the quality is zero.

Normalized qualities $q_j(o)$ are computed by

$$q_j(o) = \frac{\tilde{q}_j(o)}{\sum_{k \in F} \tilde{q}_k(o)} \tag{6.17}$$

This definition ensures that

$$\sum_{j \in F} q_j(o) = 1 \tag{6.18}$$

Then the reliability r_j computes a running average of $q_j(o)$. Due to the normalization of

$q_j(o)$, r_j is also normalized. The sum of reliabilities over all features will be 1. Thus, the reliabilities can be used as weights. A feature with a normalized quality higher than its current reliability will tend to increase its reliability, and a feature with a normalized quality lower than its current reliability will have its reliability lowered [113].

An initial set of r_j can be estimated beforehand. With the system running, the environment conditions may change over time. For example, the light conditions may be different during different times of the day. Then the reliability of color information might change. When the color shift between two cameras is larger, a lower weight should be assigned. To adapt to the changes in the environment, an online weight updating is applied. Since r_j is the running average of $q_j(o)$ over all object o, it is easily updated by incorporating the results of newly matched objects. However, to reduce the impact of the false positives that may corrupt the reliability of features, only the matched objects with the overall similarities higher than a threshold are used to update the reliability scores.

6.3 Conclusions

Compared to the object re-identification method that is used in Chapter 5, a more robust algorithm combining multiple features with adaptive weights is proposed. These features include $3D$ color histograms, HOG and LBP descriptors, object sizes, aspect ratios and travel times. The travel times can be modeled by the GMM or a single Gaussian model depending on the domain knowledge in the blind region. The parameters of GMM are trained using the EM algorithm in the training stage and updated online taking the confidence score of the matched objects into account. The weight of each feature is estimated based on the reliability of each feature and also trained in the training stage. Like the parameters of the Gaussian models, the weights are also updated after every newly matched object is found, which makes this system adaptive to environment changes.

Chapter 7

A Petri Net-based Framework for Tracking and Object Matching

7.1 Introduction

In last chapter, we employ multiple features and update their parameters adaptively to improve the robustness of the tracking results. But we didn't consider that some information of the environment and the system setup may be helpful to simplify the matching process and improve the robustness further more. For example, if we know the camera topology and the map of the roads beforehand, we can exclude some candidates that are not possible appearing in a certain direction. Or, if we know there are intersections or traffic lights in the blind region, we can decide to use GMM over a single Gaussian to improve the accuracy of the travel time model.

We name this type of information *Domain Knowledge*. Domain knowledge includes the configuration of the cameras in the network, possible entrances/exits in or out of the camera views, occluding structures, e.g. columns in the scene, useful traffic rules, reasonable assumptions based on the environment conditions, etc.

There are several benefits of involving the domain knowledge. First of all, our system is capable of processing more complicated object tracking or event detection tasks. As aforementioned, the related work has focused on object tracking and object matching across disjoint camera views. Most of them only address the problem of object re-identification or association, i.e. the objects observed in the current/downstream camera must have already been seen/detected in the previous/upstream camera(s). Their algorithms and experiments only focus on finding the object correspondences in adjacent cameras. In other words, new objects coming from blind regions or observed objects disaappearing/leaving in the blind regions are not considered, except in the work by Huang and Russell [89]. In the related work that only solves the object association problem, they implicitly assume a simple domain knowledge, which is that there is only a small gap between two adjacent cameras and no entrances or exits exist in this gap. In our system, we consider more complicated scenarios that involve entrances/exits and intersections in the blind regions. In these cases, the objects that are detected in the downstream camera may be new objects coming from the blind region (i.e. they do not exist in the candidate lists received from the upstream camera(s)), and/or some of the objects in the candidate list will never show up in the downstream camera view. These scenarios make our application even more challenging and realistic.

In this chapter, we propose a distributed camera system for object tracking across non-overlapping views. Considering the uncertainties caused by vision algorithms, a probabilistic result is preferred to a deterministic one. To incorporate the uncertainties of each stage (foreground detection, tracking and object matching) in a proper way, we employ a pPN. In our system, every camera performs multi-object tracking individually and then object matching is performed if candidate data are received from the previous camera(s). The tracking process within a single camera and object matching across adjacent cameras are modeled by the pPN and a score of each object's tracking and matching result is yielded as the output of the pPN. In our example three-camera setup, vehicles travel from Camera 1 and 2 to Camera 3. Camera 3 maintains a pPN, which includes the transitions from other cameras or entrances from the blind region into Camera 3. Similarly, if there are more cameras in the network, each camera that has upstream adjacent cameras needs to maintain a pPN, which includes the possible transitions from the previous cameras to the current camera.

Another advantage of employing the pPN is that the domain knowledge can be efficiently incorporated into the algorithm. When a rich set of domain knowledge is available, the pPN also helps to implement and control the work flow.

The proposed approach can be generalized to various surveillance applications involving disjoint camera views, such as indoor human tracking or outdoor human/vehicle tracking. In this chapter, we first present the wide-area tracking of vehicles as an example. This example shows how we fuse multiple features, train the parameters, and handle blind regions and "never-seen-before" objects. Then, a similar approach together with a different set of domain knowledge is employed for tracking people in another example with a disjoint camera setup. This example is more challenging, because unlike vehicles moving in certain lanes in fixed directions, peoples routes are more diverse. In the traffic scenario, the upstream camera assumes that a car will not reappear after leaving the camera view. On the other hand, a person can always come back to the view. For such cases, we need to save object trackers in a list for a certain amount of time after objects leave the view. These different examples and results illustrate how our framework can be applied to different scenarios with different domain knowledge. We also present the pPN for each scenario, where the domain knowledge is incorporated in the work flow.

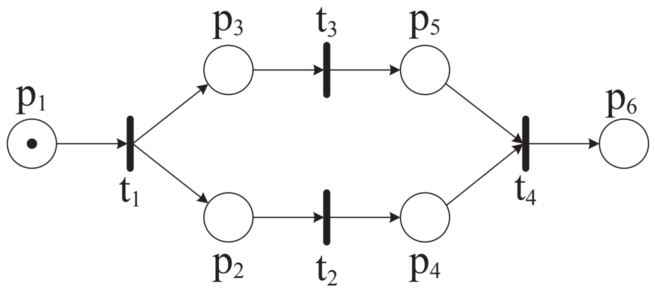
In the rest of this chapter, we first briefly review the definitions of Petri Net(PN) and probabilistic Petri Net(pPN). Then our pPN-based framework is explained using an example of wide-area vehicle tracking. Another people tracking example is also presented which employs a different set of domain knowledge and shows how the occlusion can be handled by the pPN. Experiments and comparisons with related work are performed. The results show that our system is not only able to track objects across disjoint cameras with high accuracy, and also distinguish the new objects from the already observed objects successfully. At last, a discussion about the scalability and information about how to collect domain knowledge is presented.

7.2 Petri Nets

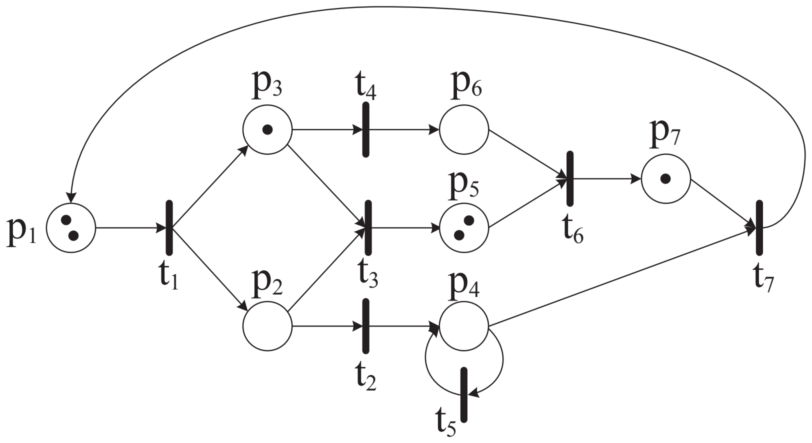
7.2.1 Definition of Petri Nets

A Petri Net (PN) is a graphic tool used for modeling the relations between the conditions and events in dynamic systems. As a graphic tool, Petri Nets can help to visualize the complex processes or systems similar to flow charts, block diagrams, and networks. In addition, It can also simulate the dynamic and concurrent activities of systems [85, 103]. Petri Nets have been widely used for years in many areas, such as manufacturing system modeling [101], production scheduling [105], sequence control [110], power system design [97], communication protocol modeling and analysis[114, 115], and software development[99], etc.

A Petri Net is a particular type of directed bipartite graph composed of places, transitions and directed arcs. Places are illustrated as circles and transitions as bars or boxes, as shown in Figure 7.1. The places and transitions are connected by arcs, where input arcs are from a place to a transition and output arcs are from a transition to a place. Figure 7.1 (a) consists of 6 places and 4 transitions. Each transition has input places and output places. For instance, t_1 has p_1 as its input place, and p_2 and p_3 are its output places. A Petri Net may contain self-loops where the input place and output place of a transition are the same. For instance, in Figure 7.1 (b), the input place and output place of t_5 are both p_4. In modeling a system with conditions and events, transitions can be considered as events. Input places



(a)



(b)

Figure 7.1: Examples of Petri Nets

and output places can represent the preconditions and postconditions of events [103].

The places may contain none or a positive number of tokens. The tokens are represented by black dots, as in the place p_1 in Figure 7.1 (a). At any given time instance, the distribution of tokens in places is called the Petri Net marking. For a Petri Net with m places, the marking is represented by a $m \times 1$ vector M. The elements of M, denoted as $M(p)$, are nonnegative integers indicating the number of tokens in place p, where $p \in \{1, \ldots, m\}$. A Petri Net containing tokens is called a marked Petri net. For instance, the marking of the Petri Net in Figure 7.1 (a) is $(1, 0, 0, 0, 0, 0)^T$. Figure 7.1 (b) shows another Petri Net with a marking $(2, 0, 1, 0, 2, 0, 1)^T$. A Petri Net is formally defined in Table 7.1 [117].

Table 7.1: Definition of a Petri Net

$PN = (P, T, I, O, M_0)$, where

1. $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places;
2. $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions, $P \cup T \neq \emptyset$, and $P \cap T = \emptyset$;
3. $I : (P \times T) \mapsto N$ is an input function which defines directed arcs from places to transitions, where N is a set of nonnegative integers;
4. $O : (P \times T) \mapsto N$ is an output function that defines directed arcs from transitions to places;
5. $M_0 : P \mapsto N$ is the initial marking.

Transitions are active components. If there are enough tokens in the input places, the transitions are enabled. Transitions are only allowed to fire if they are enabled. When a transition is enabled and the condition associated to this transition is satisfied, it fires. When the transition fires, it moves tokens from its input places to its output places, which may reflect the occurrence of events or execution of operations in a dynamic system.

7.2.2 Probabilistic Petri Nets

There have been various extensions of the Petri nets such as Colored PNs, Continuous PNs, Stochastic timed PNs and Fuzzy PNs. Albanese et al. [74] proposed the probabilistic Petri Net for modeling the uncertainty and inaccuracies in a visual surveillance system. Compared to the original PNs, a probability is attached to every arc pointing from a place to a transition in the pPN. A token is assigned the probability 1 at the initial place. When it moves to the next place, the probability is multiplied by the probability attached to the arc. After moving through the whole PN from the initial place to the end place, the final probability is the product of the probabilities attached to the arcs through which the token has passed.

Figure 7.2 is an example of a pPN modeling the car pickup activity in the parking lot that is presented in [74]. There are 8 places $p_0, \ldots, p_7$ and 6 transitions $t_0, \ldots, t_5$, where p_0

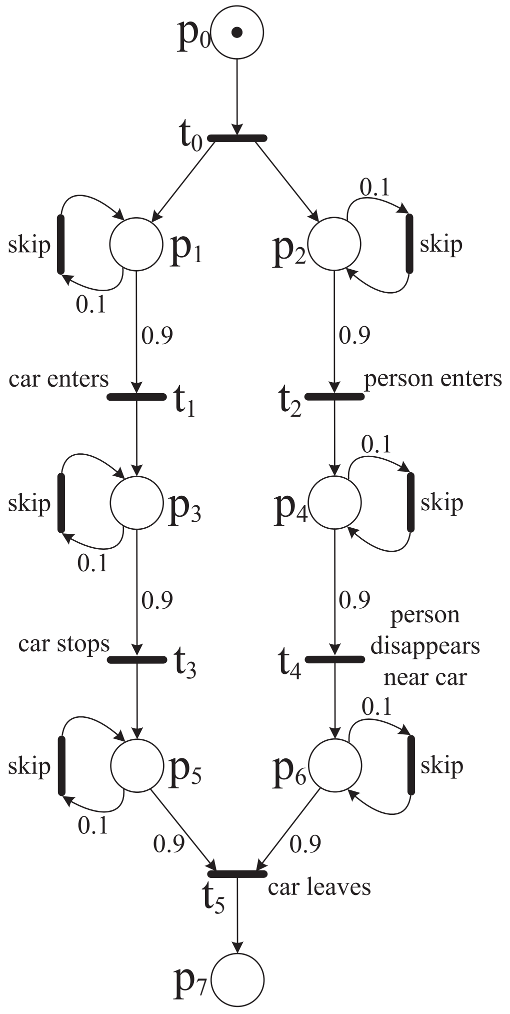


Figure 7.2: A probabilistic Petri Net for car pickup activity [74]

is the start node and p_7 is the end node. Transition t_0 is unconstrained. Whenever there is a token in p_0, t_0 is fired immediately and place a token in both p_1 and p_2. The detected objects are considered as tokens in this pPN. When a car enters the scene, t_1 is fired and the token is moved to p_3. Then the t_3 is enabled but it will not be fired until the attached condition is satisfied — *car stops*. When "*car stops*" is detected, the token is moved to p_5; and when the car leaves the parking lot, t_5 is fired and the token is moved to the end node p_7. Similarly, if the detected object is a person, it will be place in p_4 and go down the other path in the pPN.

In Figure 7.2, there are also 6 unnumbered transitions called *skip transitions*. The skip transitions are used to model the *away* deviations from the base activity pattern. Each deviation is penalized by a low probability, which controls how tolerant the model is to deviations from the base activity pattern.

A pPN is useful to model the activities of the interested objects/events in a visual surveillance system. It helps to handle the complexity of multi-objects performing activities concurrently. With the attached probabilities and skip transitions, the uncertainties and inaccuracies of the system are taken into account.

7.3 A Petri Net-based Framework for Tracking and Object Matching

We adopt a pPN-based approach to perform object tracking and consistent labeling on a camera. Figure 7.3 shows the graphical model of the steps employed by camera 3 in a three-camera setup. The camera configuration can be seen on the upper left-hand corner of the Figure 7.3. In this model, the uncertainties and inaccuracies could be created by the background subtraction, the tracking algorithm or the object matching process, and are modeled by probabilities p_b, p_t, and p_m, respectively. The *Tracking Box* and the *Matching Box* represent the processes of intra-camera tracking and inter-camera object matching, respectively. The intra-camera tracking algorithm is inherited from Chapter 2; and the inter-camera object matching approach is described in Chapter 6.

In Figure 7.3, an arc with no probability on it means that its probability is assumed to be 1. In order not to confuse the places with the parameter probability p, we use l to denote the places instead.

Once a new object is detected, it will be put in the START place l_0. Then, it immediately

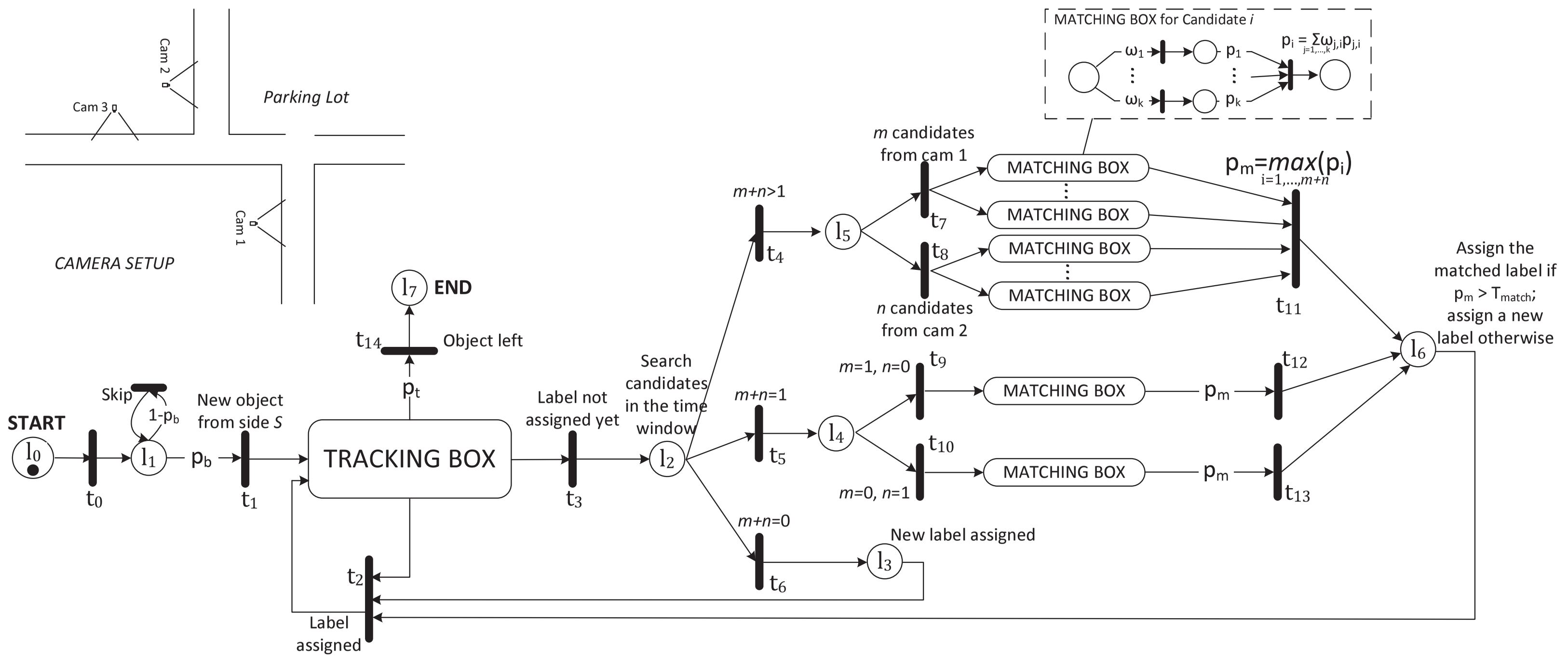


Figure 7.3: Probabilistic Petri Net for tracking and object matching.

moves into l_1, since there is no condition attached to t_0. From l_1 to t_1, a probability p_b is attached to the arc to model the reliability of the background subtraction. This probability is learned during training. Then, the object is moved into the *Tracking Box*, where a tracker for this object is created and updated every frame until it leaves the camera's view. If the object enters the view from side S and there are received packets from the neighboring camera(s) on this side, this object will be assigned a temporary label first, and moved into the *Matching Box* in an attempt to find a match from the received candidates which had left the view of the neighboring camera(s). A probability p_m will be attached to the token as the output of the Matching Box. p_m is the weighted overall similarity score if there is a matched candidate found. If no candidate package has been received from other camera(s), a new label will be assigned.

The tracking process is performed every frame and the probability p_t indicating the tracking confidence is updated every frame until the object leaves the camera's view. p_t accounts for the errors that may be caused by segmented objects and unresolved merges/splits, and is the product of the average similarity coefficient and a confidence measure. The confidence measure is based on the length of the trajectory. It is the ratio between the current trajectory length and the length of the road in the view. When the object leaves the view, the confidence measure of the trajectory length approaches 1.

Thus, for an object tracked by the system, the final probability of the tracking is:

$$p = p_b \cdot p_m \cdot p_t \tag{7.1}$$

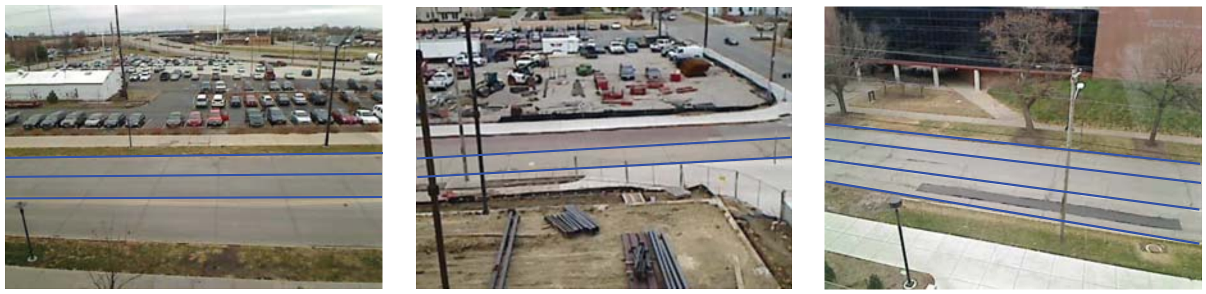
The topology of the camera setup is shown on the upper left-hand corner of Figure 7.3. Example images captured by these three cameras can be seen in Figure 7.4. Camera 3 is watching a one-way road. When a vehicle enters into the view of Camera 3, it may have come from the view of Camera 1, view of Camera 2, or from regions that are not watched by any cameras. In the latter case, camera 3 needs to detect this correctly, and assign a new label to the vehicle. Thus, some of the domain knowledge is also incorporated in the pPN implicitly.

7.4 Vehicle Tracking Experiments

We performed the wide-area vehicle tracking experiments with three disjoint cameras having the configuration shown in Figure 7.3. We used 1 hours of video data for training, and another 15 minutes of video data from three cameras for testing. Figure 7.4 shows the views of the three cameras. In Camera 1 and Camera 2, the viewed roads are two-way, but only the direction in which a car can travel towards Camera 3 was considered. Camera 3, on the other hand, watches a one-way road. The cars entering into the view of Camera 3 may come from Camera 1, Camera 2 or other blind regions that are not watched by any other camera. The distance between Camera 1 and Camera 3 is approximately 150 meters, with two intersections in the blind region. One of the intersections, which is close to Camera 2, has traffic lights. Due to these intersections, the travel times in the blind region vary significantly. The distance between Camera 2 and Camera 3 is approximately 20 meters. The cars that wait for the green light can still be seen in Camera 2's view. Thus, this intersection does not impact the travel time between Camera 2 and Camera 3 .

7.4.1 Comparison of RGB and HSV Color Model

To better address the color shifts and light changes between different camera views, we have performed several experiments to compare the RGB and HSV color models. We can generally divide the colors of vehicles into three major categories: light colors, such as white, gray, gold; bright colors, such as red, yellow, blue; and darker colors - such as black and dark blue.



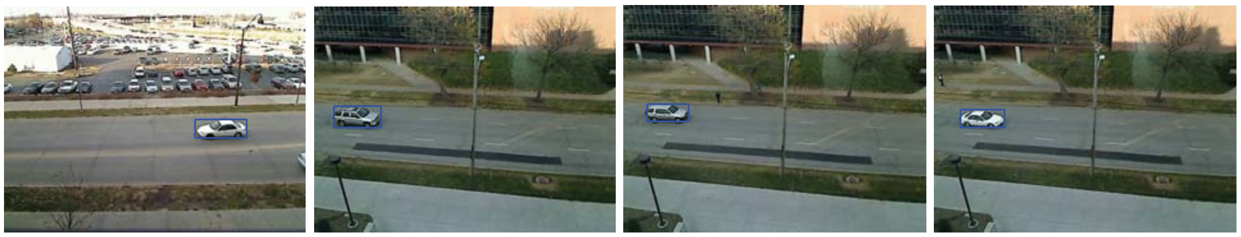
(a) Camera 1 (b) Camera 2 (c) Camera 3

Figure 7.4: Views of the three cameras.

In our experiments, we have used two different cameras (Camera 1 and Camera 3) to compare RGB and HSV histogram for different color categories. As seen in Figure 7.5 and Figure 7.6, Camera 3 has a noticeably darker view than Camera 1 does. Figure 7.5 shows an example scenario for lighter-colored cars. A white car is first seen by Camera 1 (Figure 7.5 (a)), and is compared with two gray cars (Figure 7.5 (b) and (c)) and itself (Figure 7.5 (d)) seen by Camera 3. Lighter colors are easily influenced by the light condition or the white balance settings. For this scenario, the similarity scores obtained by using 3D RGB histogram are 0.6601, 0.6834 and 0.6487 for cars (b), (c) and (d) respectively. On the other hand, when HSV histogram is used, the scores are 0.5324, 0.5117, 0.5784, respectively. In summary, the gray cars wrongly received higher scores than the white car when RGB histogram is used.

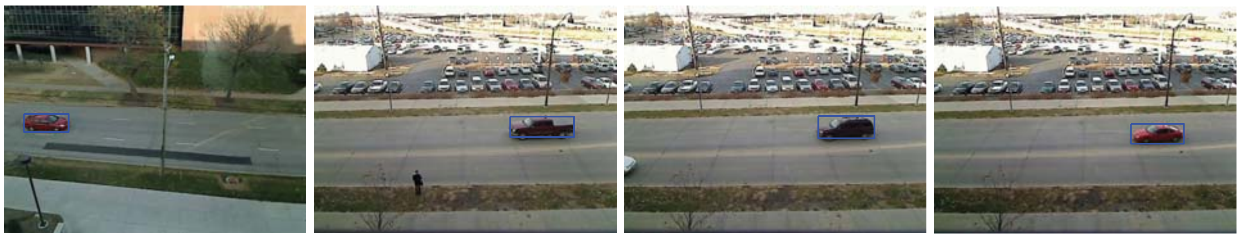
Figure 7.6 show another scenario involving red cars. In this case, using RGB histograms results in similarity scores of 0.5293,0.4393 and 0.8282, respectively, while employing HSV histograms gives scores of 0.5085, 0.3941 and 0.8356. HSV histograms perform relatively better by resulting in higher score for the same car, and lower scores for the different cars as compared to the RGB histograms.

We have adopted HSV histograms in this work.



(a) Camera 1 (b) Camera 3 (c) Camera 3 (d) Camera 3

Figure 7.5: A scenario involving white and light gray cars.



(a) Camera 3 (b) Camera 1 (c) Camera 1 (d) Camera 1

Figure 7.6: A scenario involving cars with red colors.

7.4.2 Training Stage

7.4.2.1 Domain Knowledge

The first type of domain knowledge that should be learned during training is the camera configuration. In Camera 1's view, as shown in Figure 7.4 (a), there are two lanes, and the view is close to an intersection. Based on the traffic rules, normally the cars that will turn left/right will move to the left/right lane. Thus, by detecting lanes, the cars on the right lane can be removed from the candidate list to be sent out.

Since there are two intersections in the blind region and one of them has the traffic lights, there are more than one possible distributions to represent the travel times. Thus, a GMM is built to model the travel times of the cars traveling from Camera 1's view to Camera 3's view. In our experiments, a GMM with three mixtures is trained. The means, variations and weights for each Gaussian distribution are $(22.83, 24.30, 0.464)$, $(39.37, 53.26, 0.427)$ and

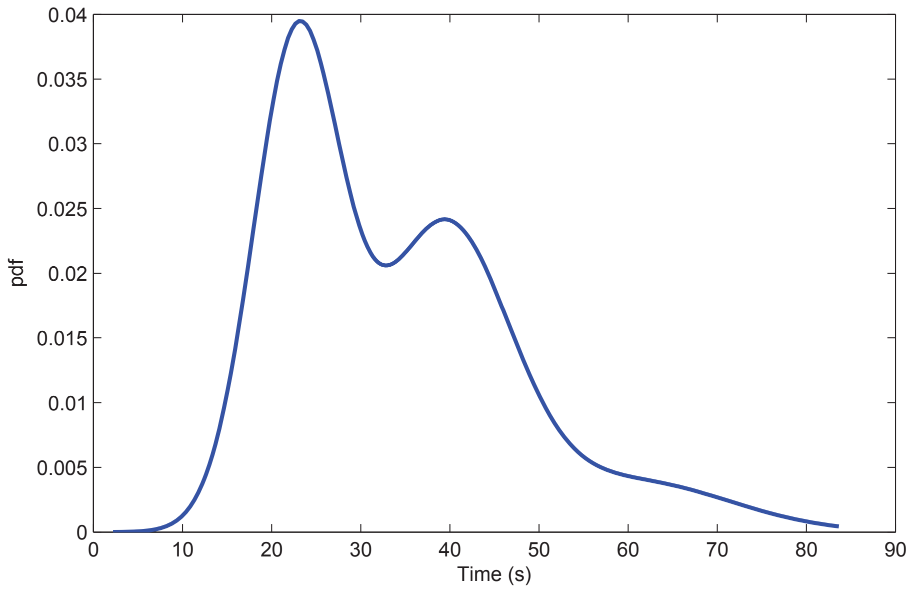


Figure 7.7: Gaussian Mixture Model of the travel time between Camera 1 and Camera 3.

$(60.36, 123.92, 0.109)$, respectively. The plot of the GMM is shown in Fig. 7.7.

Camera 2 watches only one lane. The cars on the lane may turn left or right. Since the cars are still in the view when they wait for the green light, there is no need to build a GMM for the travel times. Thus, a single Gaussian distribution is used to model the travel time.

Also, the angles of the lanes with respect to the x axis are learned in the training stage. This is used for correcting the texture descriptors.

7.4.2.2 Uncertainty

The reliability of background subtraction, p_b, is 0.997. p_t and p_m are calculated in the tracking box and the matching box for each object during the testing stage.

7.4.2.3 Weights of Features

The weights of the features are trained using the method described in Section 6.2.2 during the training stage. The results are shown in Figure 7.8.

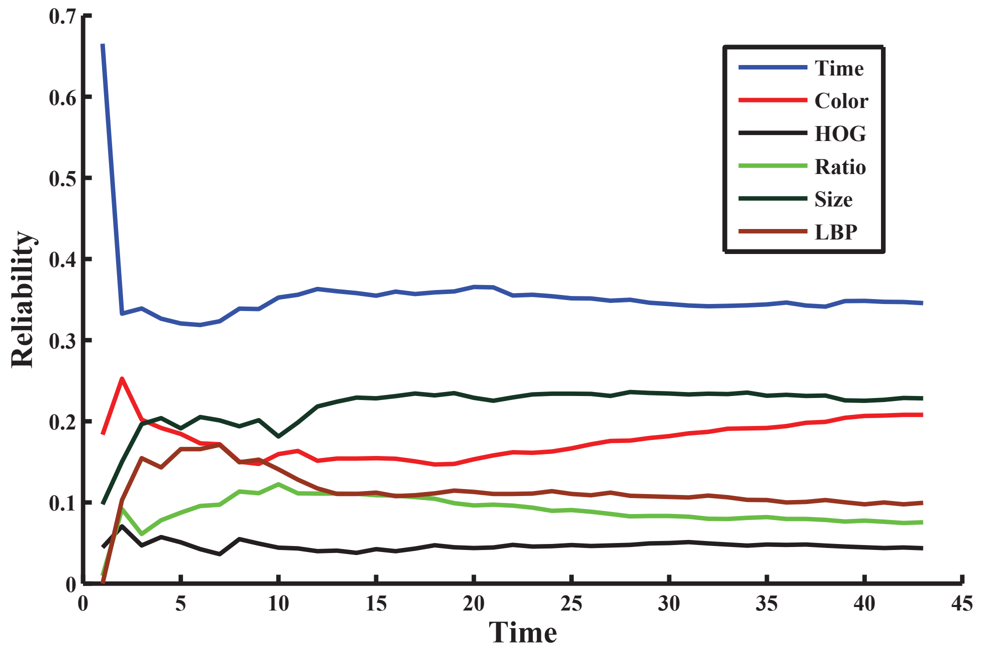
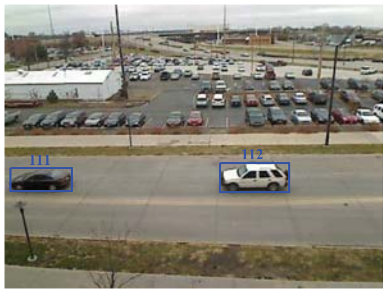


Figure 7.8: Results of weight training.

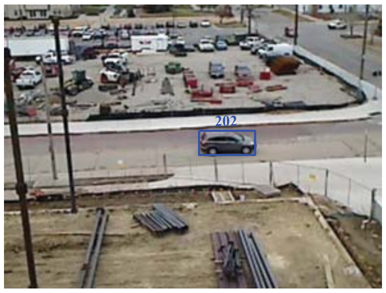
7.4.3 Testing Stage

In the testing stage, the weights for the color, HOG, LBP, travel time, size and aspect ratio are set to be 0.2079, 0.0435, 0.0993, 0.3456, 0.2283 and 0.0755 respectively. The threshold for the overall similarity score is 0.77. In Camera 1, the cars leaving the view from the right lane are not sent out as candidates to Camera 3. There are 34 candidate vehicles detected and sent out by Camera 1. In Camera 2, 18 candidate vehicles are detected and sent out. Camera 3 detects 55 cars entering its view from left. Among these 55 cars, 47 of them are assigned correct labels after the matching process, and a success rate of 85.45% is achieved. Our algorithms run on a PC with a 2.13-GHz Intel Core Duo processor and 4GB memory. It takes 31ms to perform background subtraction; and 16 $\sim$ 32ms for tracking algorithm, including feature extraction and object matching.

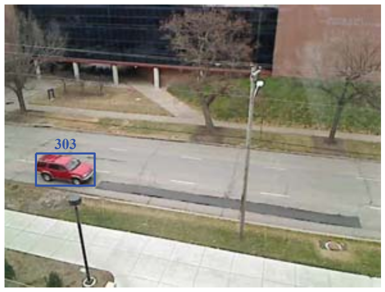
Figure 7.9 shows an example where three cars enter the view of Camera 3 consecutively.



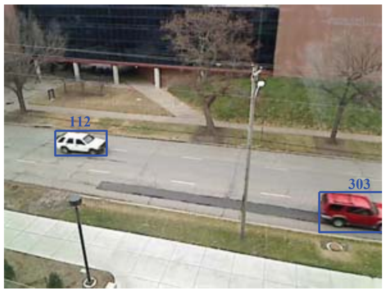
(a) Camera 1



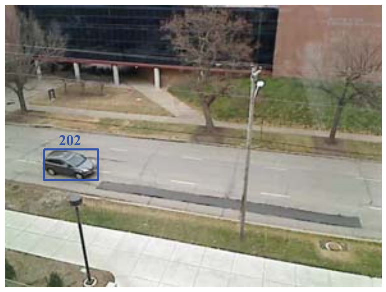
(b) Camera 2



(c) Camera 3

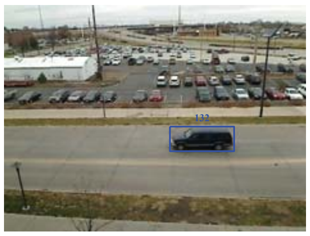


(d) Camera 3

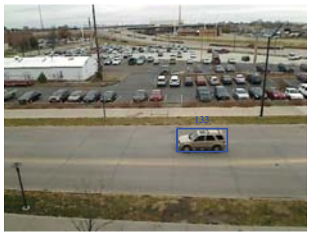


(e) Camera 3

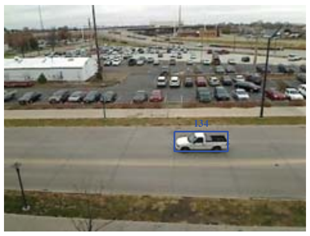
Figure 7.9: Example of matched cars.



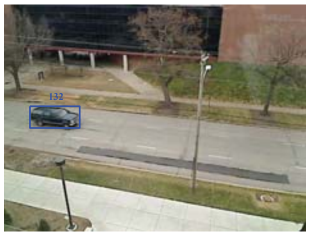
(a) Camera 1



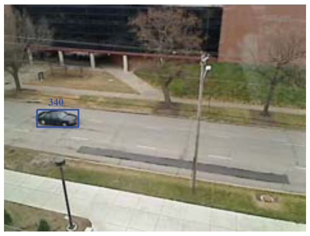
(b) Camera 1



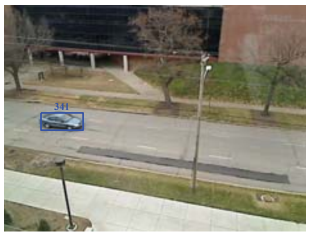
(c) Camera 1



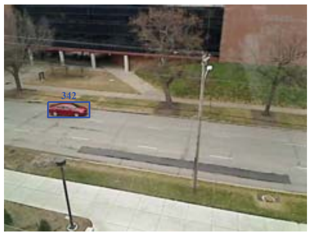
(d) Camera 3



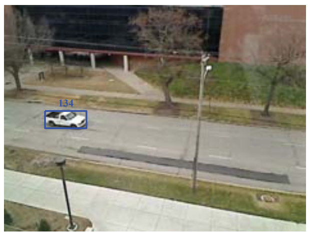
(e) Camera 3



(f) Camera 3

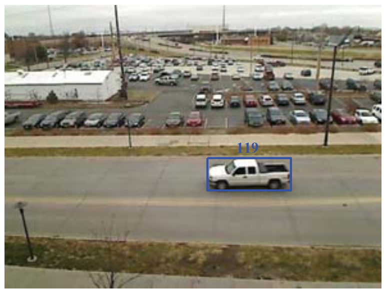


(g) Camera 3

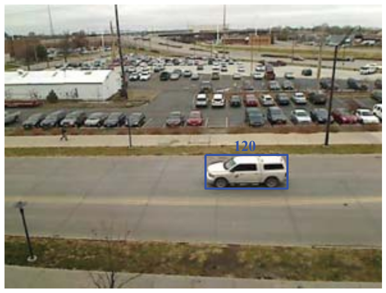


(h) Camera 3

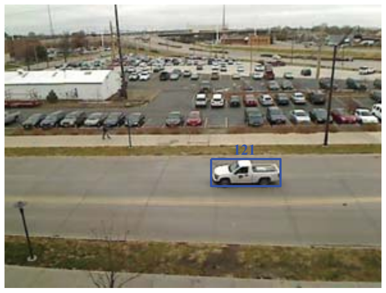
Figure 7.10: Example of handling new and left cars.



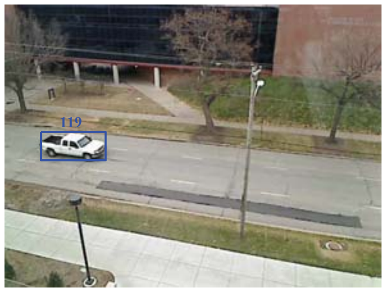
(a) Camera 1



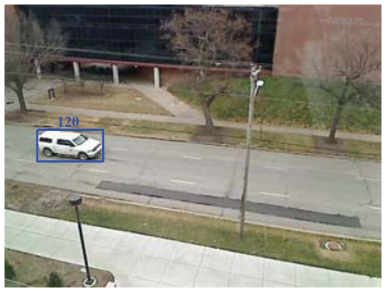
(b) Camera 1



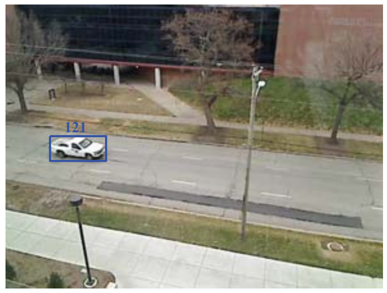
(c) Camera 1



(d) Camera 3



(e) Camera 3



(f) Camera 3

Figure 7.11: Example of matched cars with similar features.

Object 112 came from Camera 1, Object 202 came from Camera 2 and Object 303 came from the blind region (i.e. received a new label). They are all assigned correct labels. When the red car entered the view of Camera 3 as seen in Figure 7.9 (c), there were two candidates received from Camera 1 and one candidate from Camera 2. After going through the matching box, the maximum matching score was smaller than the matching threshold 0.77. Thus, this car is assigned a new label 303. In Figure 7.9 (d), a white car enters the scene. There were four received candidate packages in total, three from Camera 1 and one from Camera 2. After the matching process, the candidate with the label 112 has the maximum matching score of 0.8218, and the white car is assigned the correct label 112. The third car in Figure 7.9 (e) were compared with the same four candidates. The candidate with the label 202 resulted in the highest matching score of 0.8742, and the car was given the correct label of 202.

Figure 7.10 shows a scenario that involvs all of the following three cases: i) newly seen cars (i.e. cars that were not seen by any other cameras before); ii) reentering cars (i.e. cars leaving the previous cameras's view and entering the current camera's view); iii) disappearing cars (i.e. cars leaving the previous camera's view, and leaving the scene in the blind region. These cars are put in the candidate list, but they never enter the current camera's view). In this example scenario, there are three consecutive cars (Objects 132, 133 and 134 leaving the view of Camera 1. Object 132 enters the view of Camera 3 first and is correctly labeled (Figure 7.10 (d)). Then, three new cars 340, 341 and 342 enter the view of Camera 3 and are correctly assigned new labels (Figure 7.10 (e), (f) and (g)). Then, Object 134 arrives and consistently labeled (Figure 7.10 (h)). Object 133 that left the view of Camera 1 just leaves the scene and does not appear again in the view of Camera 3.

Figure 7.11 shows a more challenging example where three consecutive cars with the same color and similar texture enter the view of camera 3. The cars are matched to the correct candidates with matching scores of 0.8299, 0.8507 and 0.8144, respectively. The matching process performs well even if the objects have similar appearances.

7.4.4 Comparison with Other Work

As stated in Chapter 1.2.1, Huang and Russell's work [89] is one of the most cited publications of object identification/tracking across non-overlapping views; and also one of the few systems that take the new or left objects into account, not just simple association of already seen objects. Due to the similar application of traffic scenarios and use of multiple features, we present a comparison of our method with their approach. However, one thing to note is that, in [89], they only used two cameras (downstream and upstream), allowing possible exits or entrances, which is a simpler scenario than ours. We performed three sets of experiments to compare the performances of the two systems.

1) Object Association with 2 Cameras

This is the simplest scenario that involvs only 2 cameras (Camera 1 and Camera 3). In this scenario, new objects in Camera 3 and disappearing objects from Camera 1 are manually removed; i.e. the only remaining vehicles are the ones that leave Camera 1, and enter the view of Camera 3. In this experiment, Huang and Russell's algorithm achieved 100% recognition rate and our approach resulted in 93.33% accuracy. The advantage of their algorithm is that they use a group matching method instead of a one-by-one method. An association matrix is employed to find the best assignment of a small group of vehicles that are close to each other. Thus, a vehicle may not be assigned to the candidate that yields the highest score, but the final assignments yield the high group score.

2) Object Association with 3 Cameras

In this experiment, we used three cameras instead of two. Similar to the above scenario, the new and disappearing objects are still not involved, i.e. the vehicles detected in Camera 3 come from either Camera 1 or Camera 2. The algorithm in [89] achieved 70% recognition rate and our method achieved 86.05% accuracy. The algorithm in [89] assumes that the blind region does not involve intersections or other complicated traffic environments. Thus only a single univariate Gaussian distribution is used to model the travel time between the upstream and downstream cameras, which results in a decrease in the accuracy. On the other hand, we employ GMMs to model the travel time, which is capable of handling different travel time ranges caused by different traffic conditions. By taking the domain knowledge into account, and employing the algorithms accordingly, our system can handle more complicated and varying scenarios.

3) Object Identification with 3 Cameras

In the last set of comparison experiments, our 3-camera scenarios involving new, reentering and disappearing vehicles are tested. To address this problem, the algorithm in [89] adds additional elements into the association matrix to account for the new and diappearing objects, but requires a complicated training process. The prior probabilities of the disappearing and new vehicles are added to the association matrix. Since their algorithm is based on group matching of multiple cars, if an old car leaves and a new car enters, their appearance probabilities are replaced by the prior probabilities, and then the association matrix is discounted. Inherently, the group matching method will not handle the new/diappearing vehicle scenario very well. In this pair of tests, the performance of their algorithm dropped significantly to 23.85% and our accuracy rate was 85.45%. This result is somewhat consistent with what is claimed in the experimental results presented in [89], which is 100% accuracy with 14% coverage, and 50% accuracy with 80% coverage.

From above groups of experiments, we can see that our system achieves a consistent accuracy rate with increasing complexity of scenarios, thanks to use of domain knowledge and the reliable fusion of multi-features.

7.5 People Tracking Experiment Incorporating New Domain Knowledge

To illustrate the generalization of our proposed work to other scenarios, we tested it on the 3DPes dataset [75]. There are three cameras used in this dataset, and the camera setup is shown in the upper left corner of Figure 7.12. The views of three cameras are covering three different directions. Any object that comes from one of the directions may go to any of the other two directions. Each of these three cameras will keep a pPN to involve the domain

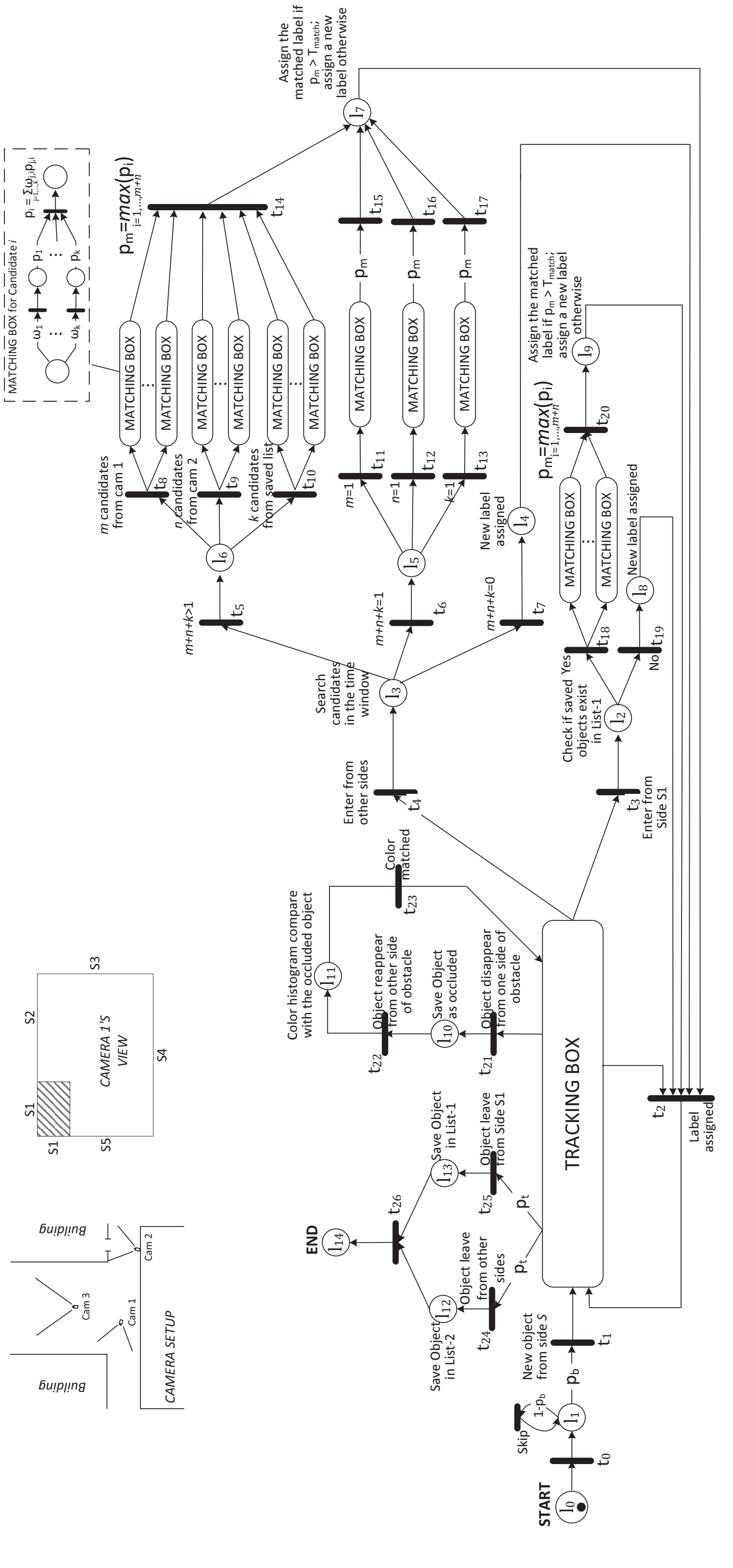
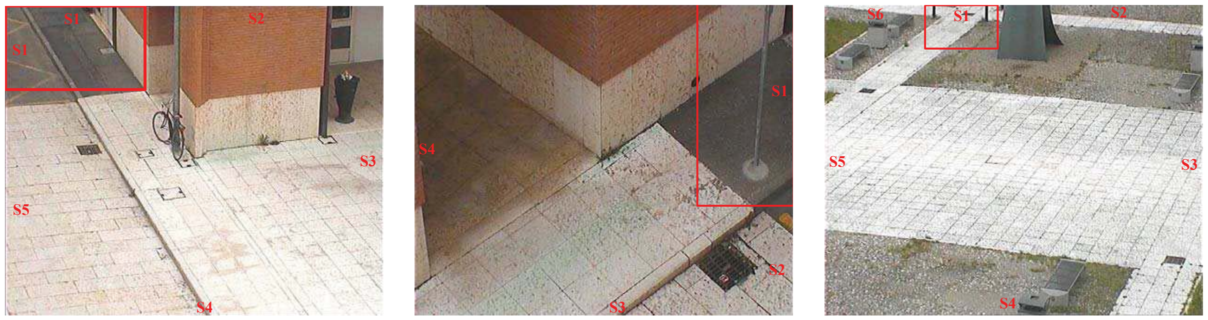

START
END
TRACKING BOX
MATCHING BOX
MATCHING BOX for Candidate i
Skip
New object from side S
Label assigned
Enter from other sides
Enter from Side S1
Search candidates in the time window
Check if saved objects exist in List-1
m candidates from cam 1
n candidates from cam 2
k candidates from saved list
New label assigned
Assign the matched label if $p_m > T_{match}$; assign a new label otherwise
$p_m = max_{i=1,\dots,m+n}(p_i)$
$m+n+k>1$
$m+n+k=1$
$m+n+k=0$
Yes
No
Object disappear from one side of obstacle
Save Object as occluded
Object reappear from other side of obstacle
Color histogram compare with the occluded object
Color matched
Object leave from Side S1
Save Object in List-1
Object leave from other sides
Save Object in List-2
CAMERA SETUP
CAMERA 1'S VIEW
Building
Cam 1
Cam 2
Cam 3
S1
S2
S3
S4
S5

Figure 7.12: Probabilistic Petri Net for people tracking and matching.

knowledge. Figure 7.12 shows the pPN that is kept by Camera 1. Since the three cameras have similar domain knowledge, the other two pPNs will be similar.



(a) Camera 1 (b) Camera 2 (c) Camera 3

Figure 7.13: Views of the three cameras.

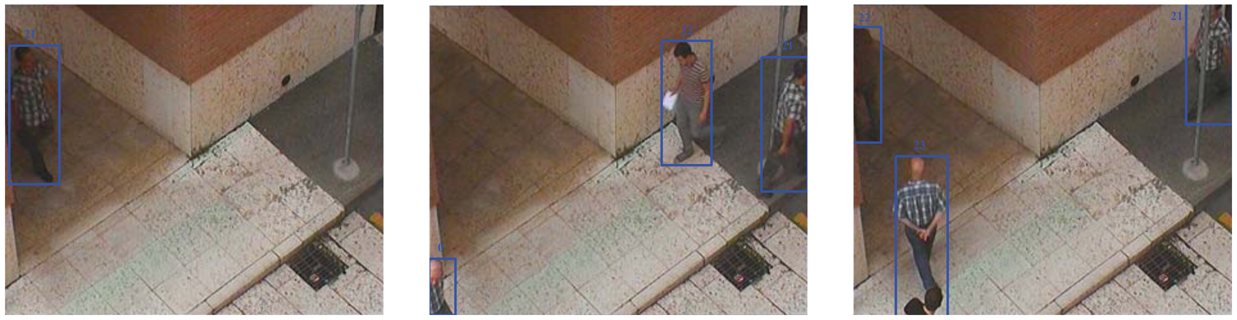
In this experiment, there are different kinds of domain knowledge compared to the above traffic scenario. First of all, people may enter the view from any side of the view, while vehicles always enter from a certain side of the view above. The domain knowledge that can be utilized here is that not every person from every side needs to be compared with the received candidates. When we know that there are no previous cameras from a specific direction, we can assign new labels to those people. Another difference between the people tracking and the vehicle tracking scenarios is that unlike vehicles moving in certain lanes in fixed directions, people's routes are more diverse. In the traffic scenario, the upstream camera assumes that a car will not reappear after leaving the camera view. On the other hand, a person can always come back to the view. There are two different cases in which a person reenters the camera's view: (i) a person leaves the current camera view, changes his/her mind in the blind region and reenters the current view.For such cases, we need to save object features in a list for a certain amount of time after objects leave the view. We call this list *saved object list*; (ii) a person leaves the current view, enters the next camera's view and comes back to the current view again. In this case, the label will be handed off from the current camera to the next, and then handed back to the current camera again.

Figure 7.13 shows example frames from three cameras. The notations used for the image sides are also marked on each view, where $S1$ denotes the side where never-seen-before objects enter. These are the objects that none of the three cameras has tracked previously. Red rectangles mark the regions where never-seen-before objects come in the scene. Any new objects detected in the red rectangles are considered as entering from side $S1$. Due to the possibility of a person coming back to the view after leaving from side $S1$, that person's features are saved in a separate list, which is denoted by *List-1* in Figure 7.12. When a new object is detected in a red rectangle, it will be compared with the objects saved in *List-1* first. If no match is found, the object will be assigned a new label. The objects saved in *List-1* will be removed after a certain amount of time. On the other hand, if an object is detected near the sides other than $S1$, this object needs to be compared with received candidates as well as the saved objects in *List-2*, which contains the objects that left from other sides. Figure 7.12 shows the pPN that incorporates the new domain knowledge.

Another new and important domain knowledge shown in Figure 7.12 is the information about the obstacles in the camera views. This kind of knowledge only involves a single camera. The location of the obstacles could be learned beforehand. Obstacles are referred to as the fixed structures in the background that occlude moving objects. If one moving object is occluded by another moving object, this is referred to as the merge/split case, which is handled by the algorithm described in Chapter 2. As stated in Chapter 2.4, when an object is not occluded, we check if the bounding boxes in the current frame and the previous frame intersect. If they intersect, we then compare the color histogram to determine if they are the same object. Here, with the domain knowledge that there is an obstacle in the view with a known location, if an object disappears from one side of the obstacle and another object appears on its other side, we consider their bounding boxes virtually intersecting. Then the color histograms are compared to determine if they are the same object, according to the method described in Chapter 2.4.

Besides the domain knowledge, some other features/assumptions may also be used in people tracking scenarios. For example, we can assume that people walk vertically. So no rotation is needed for texture matching. Also, we can use separate color histograms for torso and legs. On the other hand, size and aspect ratios may not work for people since people mostly have similar sizes and shapes from a far distance. Moreover, there are some new features that only apply to people but may be very useful, such as gait information. Since the main purpose here is to show an example of extending our framework to different scenarios, seeking different sets of features are out of scope of the current work.

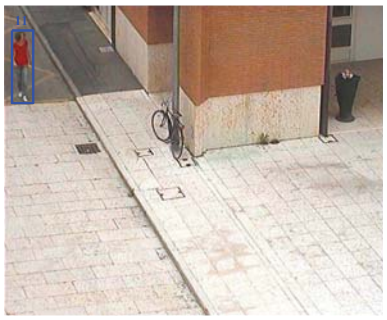
Since no training data is provided with the dataset, we set the parameters empirically, and only use the features of color, texture and travel time. Figure 7.14 shows an example where an object is matched to a saved one. Object 21 leaves Camera 2's view and comes back to the view shortly. Since his features were saved when he left, he is assigned the label 21 again when he returns.



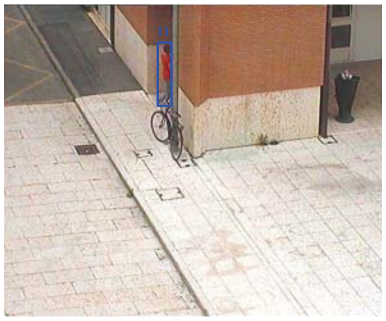
(a) Camera 2 (b) Camera 2 (c) Camera 2

Figure 7.14: An example of the saved object.

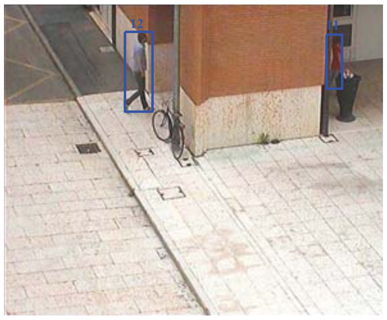
Figure 7.15 shows a person entering into the Camera 1's view and then moving to Camera 3's view later. This person enters from side $S1$ in Camera 1's view (Figure 7.15 (a)). Since there is no saved object yet, she is assigned a new label 11. Then she leaves from the right side of Camera 2 (Figure 7.15 (d)) and is correctly labeled in Camera 3 (Figure 7.15 (e)). Figure 7.15 also shows an example of accurately handling occlusion thanks to the incorporated



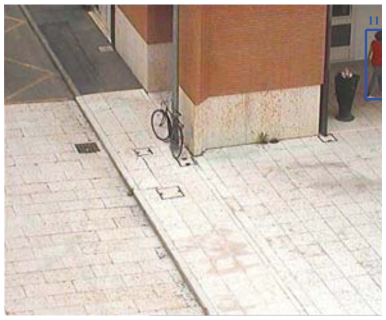
(a) Camera 1



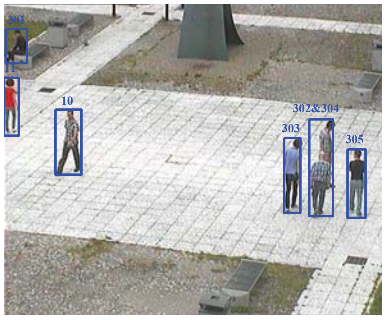
(b) Camera 1



(c) Camera 1



(d) Camera 1



(e) Camera 3

Figure 7.15: An example of people tracking and occlusion handling.

domain knowledge. Object 11 goes behind the wall (Figure 7.15 (b)), reappears (Figure 7.15 (c)) and gets the correct label.

7.6 Discussions of Scalability and Domain Knowledge

With the efficient algorithms we propose, this system will be suitable for real-time applications with wide-area non-overlapping cameras. In our distributed system, each camera node performs tracking individually and only exchanges the data of possible candidates in a peer-to-peer manner. Only one packet of feature data needs to be sent to the downstream camera(s) for an object when that object leaves the view. Thus, this system has the general advantages of distributed and peer-to-peer systems over server-based systems, including higher efficiency and scalability, low bandwidth requirements and no single point of failure.

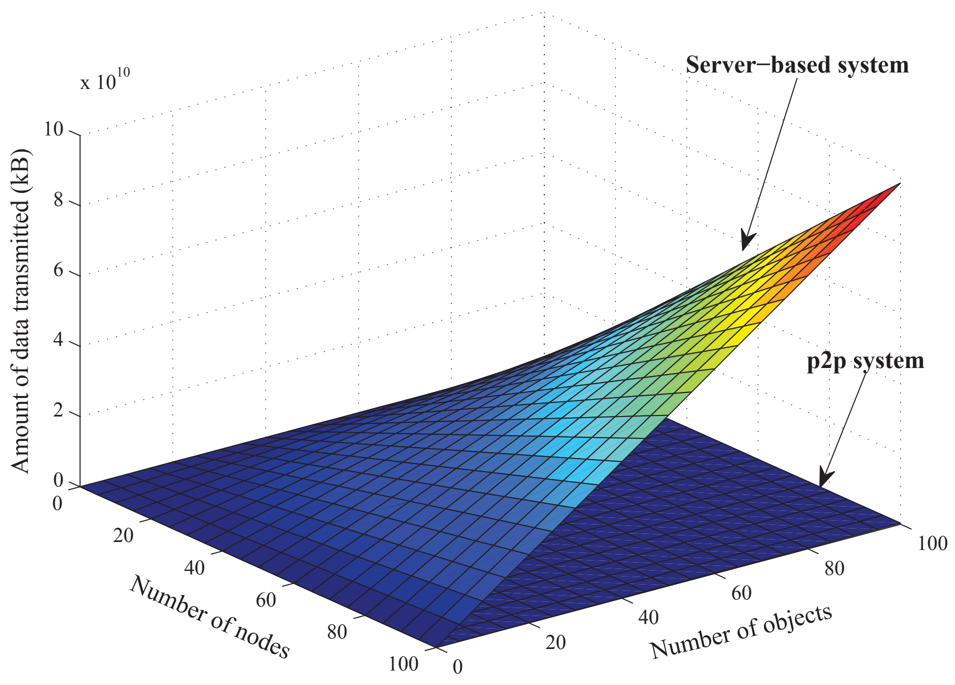


Figure 7.16: The Comparison of amount of data transmitted between a server-based system and our peer-to-peer system.

One feature of this system is that each downstream camera needs to maintain a graph incorporating the domain knowledge. Since we consider wide-area and non-overlapping camera settings, the system will involve fewer cameras compared to an entirely overlapping camera setup. Thus, each camera will have a limited number of predecessors and successors.

To show the scalability with the number of cameras and targets, we created a server-based scenario and compared it with our peer-to-peer system. In the server-based scenario, each camera only detects the foreground objects, and sends the information about trackers to the server at every frame. Thus, each camera node has to send every tracker at every frame to the server. The server needs to keep the received trackers in a buffer, and track the objects intensively. The amount of data that is received by the server in every minute can be computed by:

$$D_{server} = o \cdot c \cdot f \cdot S_T \tag{7.2}$$

where o is the average number of objects that are in the view, which indicates the crowdedness of the scene; c is the number of camera nodes in the system; f is the number of frames per minute; S_T is the size of the data packet for a tracker. From Eq. (7.2), we can see that the amount of data transmitted increases proportionally with o or c, but is also multiplied by a large number f , which significantly magnifies any increase in o and c.

On the other hand, the amount of data that need to be transmitted in our distributed peer-to-peer (p2p) system can be estimated by:

$$D_{p2p} = o \cdot c \cdot p \cdot \sum_{i=1}^{N} w_i n_i \cdot S_F \tag{7.3}$$

where n_i denotes the number of successors of a camera node, and $1 \leq n_i \leq N$. With our wide-area, non-overlapping and sparse camera deployment, N is assumed to be a single digit number. Each camera might have different number of successors, which depends on the camera deployment. w_i is the percentage of cameras having n_i-many successors. In addition, with the domain knowledge that we incorporated, we can discard some objects that exit the scene in a direction towards which no successors exist. Thus, we do not send packets for these objects. To account for this, we introduce a new parameter p in Eq. (7.3), which denotes the probability that we will send a packet for the object, i.e. the probability of objects going in the downstream camera direction. In some scenarios, p will be simply 1. When Eq. (7.3) is compared with Eq. (7.2), it can be seen that $\sum_{i=1}^{N} w_i n_i$ is much smaller than f, and could be further reduced by p. S_F is the size of the packet for feature data, and is comparable with S_T.

Figure 7.16 shows the amount of data that needs to be sent with different number of

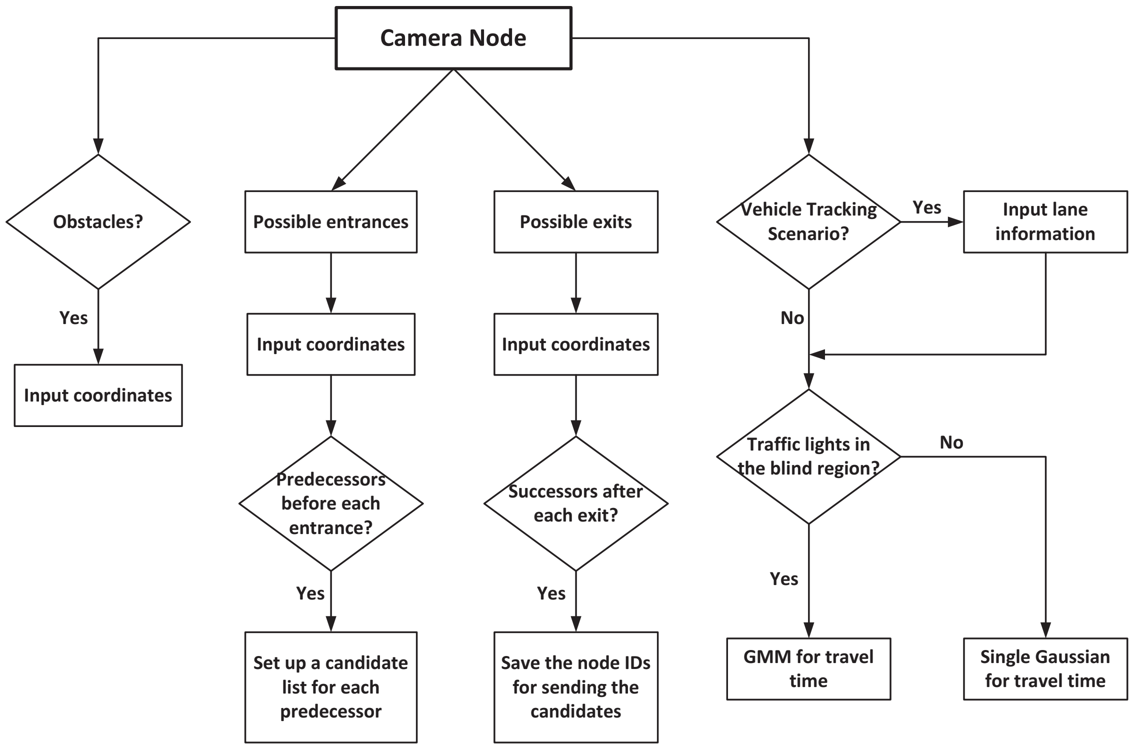

Camera Node
Obstacles?
Yes
Input coordinates
Possible entrances
Input coordinates
Predecessors before each entrance?
Yes
Set up a candidate list for each predecessor
Possible exits
Input coordinates
Successors after each exit?
Yes
Save the node IDs for sending the candidates
Vehicle Tracking Scenario?
Yes
Input lane information
No
Traffic lights in the blind region?
No
Yes
GMM for travel time
Single Gaussian for travel time

Figure 7.17: The flowchart for the users' input of domain knowledge.

camera nodes and different number of objects (varying levels of crowdedness). For this scenario, we assumed that the maximum number of successors for a camera (N) is 5. We assumed that n_i is uniformly distributed between 1 and 5, i.e. w_i is 0.2 for all n_i. In this case, p is assumed to be 1 meaning that we send information about every exiting object to the successors, which is a worst-case scenario in terms of number of messages. In our experiment videos, the frame rate is 15 frames per second. Thus, f is 900. As can be seen in Figure 7.16, compared to a server-based system, our p2p system is much more scalable with increasing number of nodes and increasing number of objects even when we assume that $p = 1$. More specifically, the amount of data transmitted does not increase significantly as the number of nodes and objects are increased.

Although this system is scalable and efficient, the performance will be affected if it is applied to a crowded scenario. Another factor that may affect the performance would be the travel times of objects. In our work, we assume that the objects are moving in a consistent way. Even if there are traffic lights in the blind region, the travel time can still be modeled. However, changes in the travel time in some completely random manner, for example when a person decides to chat with somebody in the blind region, might cause assigning a new label when the target enters the view of the next camera.

Another issue related to scalability is the collection of domain knowledge for each camera. The users only need to provide basic information to set up the domain knowledge for a particular camera, and a user interface can be built to receive this information. Figure 7.17 shows a flowchart for designing the user interface. The users need to input the location coordinates of possible entrances, exits, intersections and obstacles during system initialization. With a graphical user interface (GUI), the coordinates can be entered on the cameras view. If the system is developed for vehicle tracking scenarios, the lane information could also be entered as input.

7.7 Conclusions

We have presented a distributed wide-area multi-object tracking system composed of non-overlapping cameras. A probabilistic Petri Net-based approach has been used to account for the uncertainties of the vision algorithms and to incorporate the available domain knowledge. Multiple features are used for object matching across non-overlapping views and combined by adaptive weights, which make the system adapts to the environment changes.

We first presented wide-area tracking of vehicles, where we used three non-overlapping cameras. Our method achieved high accuracy. It handles complicated scenarios well by taking the domain knowledge into account. With the proposed method, not only already observed objects can be re-identified in the current camera, but also never-seen-before objects coming from blind regions can be handled.

By using different sets of available domain knowledge, the proposed work can be applied/extended to other scenarios. We have used 3DPes dataset to demonstrate how our method can be applied to a people tracking scenario. The domain knowledge helps to make the tracking and matching process more efficient and the results more robust. It also helps handling the occlusion of targets by fixed structures in a single camera view.

Although we only present the experimental results on three camera setups, the proposed approach is feasible for larger camera networks, thanks to distributed processing and p2p exchange of small amount of data. To collect the domain knowledge from users and set up the pPN for each downstream camera, a graphical user interface may be implemented.

Part IV

Other Work on Multi-camera Systems

Chapter 8

Frame-Level Temporal Calibration of Unsynchronized Cameras

8.1 Introduction

We present a method for temporal calibration of video sequences from unsynchronized cameras by using object trajectories. Temporal calibration identifies corresponding frames in video sequences captured by different cameras. A low-level method for temporal calibration is synchronization that forces cameras to capture the corresponding frames at the same time by having a master clock. A generic temporal calibration method that is based only on image information provides a solution for cameras without a common clock as well, and removes the need for special equipment and hardware.

Temporal calibration is very important for multi-camera systems, because the transfer of relevant data between cameras is essential. Hardware-based synchronization increases installation cost. An alternative way is to use image/video processing to align frames from the cameras and retrieve the frame offset.

Kuthirummal et al. [118] presented an approach in Fourier Domain, which requires at least seven stationary corresponding points in three views. Also, a point needs to be tracked over a number of frames in three views. Lee et al. [119] introduced a method to align the centroids of moving objects. However, centroid points are treated individually rather than as a part of a trajectory, which increases the combinatorial complexity. Moreover, accuracy can be affected by the height of the objects, thus by their distance to the cameras. Caspi et al. [121] also introduced a trajectory-based algorithm. It is assumed that the temporal offset between the two sequences is at most 25 frames. Tuytelaars and VanGool [122] proposed a method that can deal with moving cameras and general $3D$ scenes. However, this method requires tracking five corresponding points in two sequences, which are selected manually as a subset of a feature point set tracked through the video sequence. Velipasalar and Wolf [123] introduced a search algorithm to match and align trajectories obtained from different sequences. This method is robust to errors caused by background subtraction or location extraction. Yet, it performs an exhaustive type of search.

In this chapter, we describe a method based on finding the Longest Consecutive Common Subsequence (LCCS). Longest Common Subsequence (LCS) was proposed by Vlachos et al. [124] to find similar multi-dimensional trajectories, and was used by Buzan et al. [125] and Cheriyadat and Radke [126] for finding similar trajectories in video sequences. Both [125] and [126] focus on trajectory clustering in a single camera view. We present an LCCS-based algorithm with a customized similarity criterion, and employ it in a multi-camera application to find consecutive matching points as a part of our method. The proposed algorithm provides significant improvement in terms of computational complexity, and has comparable or better results with respect to the previous work described in [123].

8.2 LCS and LCCS

Longest Common Subsequence (LCS) is a classic problem in finding the maximal common characters in two sequences, in the same order but not necessarily consecutive [120]. For example, there are two sequences as follows:

$$THISISEXAMPLEFORLCS$$
$$THATISNOTEXAMPLE$$

The longest common subsequence of the above two sequences is:

$$THISEXAMPLE$$

The LCS is extended to Longest Consecutive Common Subsequence that all the characters in the longest common subsequence must be consecutive. It is also referred to as Longest Common Substring. In the above example, the LCCS will be:

$$EXAMPLE$$

LCS is well-known as an example of dynamic programming [120]. If the lengths of the two sequences are n and m respectively, the dynamic programming needs to create a $(n+1) \times (m+1)$ table to save the intermediate results. And the characters of the LCS can be found by tracing back the dynamic programming table.

In the video analysis applications, sequences with two-dimensional points are often compared instead of characters. In this case, two points that are close enough can be considered as "common characters". A 2D trajectory matching application for LCS is proposed in [124]. In [124], the purpose of using LCS is that LCS allows time stretch (the common points are not consecutive) and thus reduce the noise of the outliers. Their application is to find the similar trajectories that are created by different objects moving at different speeds.

However, in our application, we desire to match the two trajectories in two video sequences that are actually created by the same object. We do not need to consider the time stretch problem. Thus, LCCS is more desirable for this case. Moreover, the noise in our application is caused by the errors of the background subtraction and tracking algorithms. The noise points may make the LCCS algorithm fail if we keep searching the consecutive common points along the trajectories. To address this issue, we modify the LCCS algorithm and only search for a consecutive common subsequence with a certain length. The length of the already found consecutive common subsequence is the latest filled element in the dynamic programming table. Once it reaches the length threshold, LCCS searching stops. A simpler but effective method is employed instead to continue searching similar points. This LCCS-based algorithm is described as follows.

8.3 Trajectory Alignment Using LCCS

We detect, track and extract the location of each moving object, as described in more detail in the previous work [123], to form trajectory data. Let $L^c_{t_c}$ be the label of the ${t_c}^{th}$ trajectory on the view of camera c. Thus, $c \in \{1, 2\}$ and $t_c \in \{1, 2, \ldots, N_c\}$ where N_c is the number of trajectories in the sequence captured by the c^{th} camera. The trajectory data for label $L^c_{t_c}$ is in the following format:

$$L^c_{t_c} \rightarrow \left\{ \begin{array}{c} \left(F_1^{L^c_{t_c}}, x_{E_1}^{L^c_{t_c}}, y_{E_1}^{L^c_{t_c}}, x_{C_1}^{L^c_{t_c}}, y_{C_1}^{L^c_{t_c}}\right) \\ \left(F_2^{L^c_{t_c}}, x_{E_2}^{L^c_{t_c}}, y_{E_2}^{L^c_{t_c}}, x_{C_2}^{L^c_{t_c}}, y_{C_2}^{L^c_{t_c}}\right) \\ \vdots \\ \left(F_n^{L^c_{t_c}}, x_{E_n}^{L^c_{t_c}}, y_{E_n}^{L^c_{t_c}}, x_{C_n}^{L^c_{t_c}}, y_{C_n}^{L^c_{t_c}}\right) \end{array} \right\} \tag{8.1}$$

where $F_i^{L_{t_c}^c}$ is the frame number for the i^{th} point in the trajectory, $P_E(F_i^{L_{t_c}^c}) = (x_{E_i}^{L_{t_c}^c}, y_{E_i}^{L_{t_c}^c})$ is the extracted location of the foreground object at frame $F_i^{L_{t_c}^c}$ in the current view, and $P_C(F_i^{L_{t_c}^c}) = (x_{C_i}^{L_{t_c}^c}, y_{C_i}^{L_{t_c}^c})$ is the corresponding location of $P_E(F_i^{L_{t_c}^c})$ in the other view, calculated at frame $F_i^{L_{t_c}^c}$ by using an estimated homography [123].

8.3.1 LCCS-based Algorithm

Let $L_{t_1}^1$ and $L_{t_2}^2$ denote two trajectories containing n and m points, respectively, which are expressed as

$$\begin{aligned} L_{t_1}^1 = \Big\{&(F_1^{L_{t_1}^1}, x_{E_1}^{L_{t_1}^1}, y_{E_1}^{L_{t_1}^1}, x_{C_1}^{L_{t_1}^1}, y_{C_1}^{L_{t_1}^1}), \ldots, \\ &(F_n^{L_{t_1}^1}, x_{E_n}^{L_{t_1}^1}, y_{E_n}^{L_{t_1}^1}, x_{C_n}^{L_{t_1}^1}, y_{C_n}^{L_{t_1}^1})\Big\} \\ L_{t_2}^2 = \Big\{&(F_1^{L_{t_2}^2}, x_{E_1}^{L_{t_2}^2}, y_{E_1}^{L_{t_2}^2}, x_{C_1}^{L_{t_2}^2}, y_{C_1}^{L_{t_2}^2}), \ldots, \\ &(F_m^{L_{t_2}^2}, x_{E_m}^{L_{t_2}^2}, y_{E_m}^{L_{t_2}^2}, x_{C_m}^{L_{t_2}^2}, y_{C_m}^{L_{t_2}^2})\Big\} \end{aligned}$$

We define $Head\left(L_{t_1}^1\right)$ and $Head\left(L_{t_2}^2\right)$ as

$$\begin{aligned} Head\left(L_{t_1}^1\right) = \Big\{&(F_1^{L_{t_1}^1}, x_{E_1}^{L_{t_1}^1}, y_{E_1}^{L_{t_1}^1}, x_{C_1}^{L_{t_1}^1}, y_{C_1}^{L_{t_1}^1}), \ldots, \\ &(F_{n-1}^{L_{t_1}^1}, x_{E_{n-1}}^{L_{t_1}^1}, y_{E_{n-1}}^{L_{t_1}^1}, x_{C_{n-1}}^{L_{t_1}^1}, y_{C_{n-1}}^{L_{t_1}^1})\Big\} \\ Head\left(L_{t_2}^2\right) = \Big\{&(F_1^{L_{t_2}^2}, x_{E_1}^{L_{t_2}^2}, y_{E_1}^{L_{t_2}^2}, x_{C_1}^{L_{t_2}^2}, y_{C_1}^{L_{t_2}^2}), \ldots, \\ &(F_{m-1}^{L_{t_2}^2}, x_{E_{m-1}}^{L_{t_2}^2}, y_{E_{m-1}}^{L_{t_2}^2}, x_{C_{m-1}}^{L_{t_2}^2}, y_{C_{m-1}}^{L_{t_2}^2})\Big\} \end{aligned}$$

The Euclidean distance between the n^{th} extracted point in the first trajectory, and the m^{th} calculated point in the second trajectory is denoted by $d_{E_nC_m}$.

Definition 1 *Given a positive number ϵ, we define $LCCS_\epsilon\left(L_{t_1}^1, L_{t_2}^2\right)$ as follows:*

$$LCCS_{\epsilon}\left(L_{t_1}^{1}, L_{t_2}^{2}\right) = \begin{cases} 1 + LCCS_{\epsilon}\left(Head\left(L_{t_1}^{1}\right), Head\left(L_{t_2}^{2}\right)\right) \\ \qquad if\ d_{E_nC_m} < \epsilon\ and\ d_{C_nE_m} < \epsilon \\ \\ 0 \quad if\ L_{t_1}^{1} or\ L_{t_2}^{2} is\ empty\ or \\ d_{E_nC_m} \geq \epsilon\ or\ d_{C_nE_m} \geq \epsilon \end{cases}$$

The constant ϵ is the distance matching threshold. The points that are close in space are regarded as matching points. If the extracted location in the first view and the calculated corresponding location from the second view are close enough *and* the extracted location in the second view and the calculated corresponding location from the first view are close enough, then the matching score is increased by 1. LCCS searches all points in two trajectories sequentially and collects the LCCS score in a recursive way. LCCS only saves the number of consecutive matching points by resetting the LCCS score to 0 once the search meets an unmatched pair.

Since we assume that cameras have the same frame rate, we can find all matching points by LCCS without time stretching. However, there may be possible errors due to background subtraction and/or location extraction. Thus, there may be points in a trajectory, which make LCCS comparison fail, and reset the similarity score to 0. To avoid this, we introduce a positive integer, M, as a threshold for the number of matched points. We only need to find the first M matching points between the two trajectories. Then, we stop searching for matching points by LCCS once we have $LCCS_{\epsilon}\left(L_{t_1}^{1}, L_{t_2}^{2}\right) = M$. We continue to search the trajectory from the last matched point pair. For example, if i^{th} and j^{th} points in two trajectories are the M^{th} matched point pairs, then we stop LCCS at these points. We denote the number of matched points as N_{match}, and N_{match} is set to be M when LCCS-based search is stopped. Then, we compare the $(i+1)^{th}$ and $(j+1)^{th}$ points from the two trajectories, respectively. If they match, N_{match} will be increased by 1, and it will be $M+1$; if they do not match, we move on to the points $(i+2)$ and $(j+2)$ without increasing N_{match}. We continue this search until we reach the end of one of the trajectories.

Definition 2 *We define the similarity function S between two trajectories $L^1_{t_1}$ and $L^2_{t_2}$, given ϵ and M, as follows:*

$$S\left(\epsilon, M, L^1_{t_1}, L^2_{t_2}\right) = \frac{N_{match}}{min\left(n, m\right)}$$

We define the similarity function S by normalizing N_{match} by the minimum length of the two trajectories, thus $0 \leq S \leq 1$. This similarity function S is used as the main criteria to find the best matching trajectories.

Thus, for a trajectory $L^1_{t_1}$ in the first camera view, we calculate the value of S with every trajectory from the second camera. In other words, if there are N_2 trajectories in the second camera view, we perform N_2 many similarity computations. As described above, we have two groups of location coordinates for every point in each trajectory: extracted location and its calculated location in the other view. The distance between the points $P_C\left(F_i^{L^1_{t_1}}\right)$ and $P_E\left(F_j^{L^2_{t_2}}\right)$ is denoted by $d_{C_iE_j}$, where $t_1 \in \{1, 2, \ldots, N_1\}$ and $t_2 \in \{1, 2, \ldots, N_2\}$. With the given distance threshold ϵ, we consider two points matching with each other when $d_{C_iE_j} < \epsilon$ and $d_{E_iC_j} < \epsilon$ are both satisfied.

After computing the similarity scores between the $L^1_{t_1}$ and all the trajectories in the second camera, we pick the trajectory in second camera view, which gives the highest S value, as the match of the trajectory $L^1_{t_1}$. Then, we can easily obtain the frame offset from these two trajectories, since all matching point pairs have the same frame offset. The frame offset from $L^1_{t_1}$ is denoted by $O^{L^1_{t_1}}$, and is obtained by subtracting the frame numbers of any matched pair of points. Then, the two matched trajectories and their corresponding frame offset value are saved as the input of the confidence check step.

We perform the above steps for every trajectory in the first camera view to find their matching trajectory in the second view. The pseudo code for the proposed LCCS-based trajectory matching is presented in Table 1.

In Table 8.3.1, t_1' denotes the matching trajectory found for t_1. After we obtain candidate trajectory pairs, we obtain the median value S_{med} of their similarity scores. We keep the trajectory pairs whose similarity score is greater than S_{med}. This decreases the number of possible matches by half by removing the pairs with low scores. In addition to the computational aspects, this step is useful since a trajectory may not have a real match in the other camera view. This trajectory will have a low score, and will be removed with this step.

8.3.2 Confidence Check for the Frame Offsets

In this step, we perform a confidence check to find the most reliable frame offset value among the different offset values obtained from the matched trajectories. The confidence check is inherited from [123] and described below. Let Λ denote the set of the trajectory numbers on the current camera view, that are kept with their matched trajectories from the other view. In other words, the set Λ is built from the elements of $\{1, 2, ...N_1\}$ such that the S value calculated for the trajectories with labels $\{L^1_{t_1} : t_1 \in \Lambda\}$ and their matched trajectories is greater than S_{med}.

Let T^{match} be the saved data for the matched trajectories that are kept. T^{match} has the following format:

$$T^{match} = \left\{ \left(L^1_{t_1}, L^2_{t_1'}, O^{L^1_{t_1}} \right) : t_1 \in \Lambda \right\}$$

The confidence check is formulated as follows:

$$O^* = \operatorname*{argmin}_{O \in \{O^{L^1_{t_1}} : t_1 \in \Lambda\}} \frac{1}{|T^{match}|} \sum_{\substack{\tau \in \{L^1_{t_1} : \\ t_1 \in \Lambda\}}} \left(\frac{1}{|\tau|} \sum_{e=1}^{|\tau|} D(F^\tau_e, F^\tau_e + O) \right) \tag{8.2}$$

```
for every $L^1_{t_1}, t_1 \in \{1 \ldots N_1\}$
    $S_{max} = 0;$
    for every $L^2_{t_2}, t_2 \in \{1 \ldots N_2\}$
        $n = \text{length}(L^1_{t_1});$
        $m = \text{length}(L^2_{t_2});$
        set $table_{match} = [n+1]\,[m+1]$ all 0; k=1;
        while $k \le n * m$
            $i = floor\left((k-1)/m\right) + 1;$
            $j = k - (i-1) * m;$
            if $d_{C_j E_i} < \epsilon$ and $d_{E_j C_i} < \epsilon$
                $table_{match}\,[i+1]\,[j+1] = 1 + table_{match}\,[i]\,[j]\,;$
                if $table_{match}\,[i+1]\,[j+1] == M$
                    $i_{stop} = i;$
                    $j_{stop} = j;$
                    $N_{match} = M;$
                    break;
                else $k++;$
        set $i = i_{stop};$
        set $j = j_{stop};$
        while $i < n$ and $j < m$
            $i++;$
            $j++;$
            if $d_{C_j E_i} < \epsilon$ and $d_{E_j C_i} < \epsilon$
                $N_{match} = N_{match} + 1;$
            else $continue;$
        $S = N_{match}/min\,(n, m)\,;$
        $O^{L^1_{t_1}}_{L^2_{t_2}} = F^{L^2_{t_2}}_{j_{stop}} - F^{L^1_{t_1}}_{i_{stop}};$
        if $S > S_{max}$
            $S^{L^1_{t_1}}_{max} = S;\ t'_1 = t_2;\ O^{L^1_{t_1}} = O^{L^1_{t_1}}_{L^2_{t_2}};$
        save $\left(L^1_{t_1}, L^2_{t'_1}, S^{L^1_{t_1}}_{max}, O^{L^1_{t_1}}\right)$
```

Table 8.1: Pseudo code for the LCCS-based trajectory matching

The confidence check starts with a $L^1_{t_1}$, where $t_1 \in \Lambda$, and $O^{L^1_{t_1}}$ which is the frame offset candidate obtained from the corresponding trajectory pair. For all the track points of $L^1_{t_1}$, this offset candidate is added to their frame numbers. Then the points of a trajectory, which exist at the resulting frames, in the other camera are found. The point-wise distance $D(F^{\tau}_e, F^{\tau}_e{+}O)$ is calculated for each point pair, and the mean of the point-wise distance measures over the number of trajectory points is found. If there are multiple trajectories existing at the resulting frames in the other camera, the minimum of the mean point-wise distance measures obtained from these trajectories is used. The same process is repeated, again using $O^{L^1_{t_1}}$, for the track points of the next trajectory in Λ, and the overall mean of the point-wise distance measure over different trajectories is obtained for the offset $O^{L^1_{t_1}}$. All offset candidates are tried in this way, and the offset candidate that has the minimum overall mean of point-wise distance over all different trajectories is the best frame offset that we recover.

8.3.3 Comparison of the Proposed Method with the Previous Work

The previous method presented in [123], calculates the distance of *each* point in *each* trajectory of the first camera to the *each* point in *each* trajectory of the second camera. This exhaustive search involves four main loops, which results in $O\,(N_1{*}\,N_2{*}n{*}m{*}\,C)$ operations. N_1, N_2, n, m are the sizes of each nested loop. C is the number of operations inside the innermost loop.

As seen in Table 8.3.1, we set up three loops at the beginning, and the initial sizes of these loops are also $N_1, N_2, n * m$. However, in most cases, the loops are not executed completely. Once we find M many matching point pairs, all loops are broken. If the M matching points are at the beginning of the trajectories, we only need $M * m$ steps to find

Table 8.2: The frame offsets obtained after the confidence check with the proposed method and the previous work.

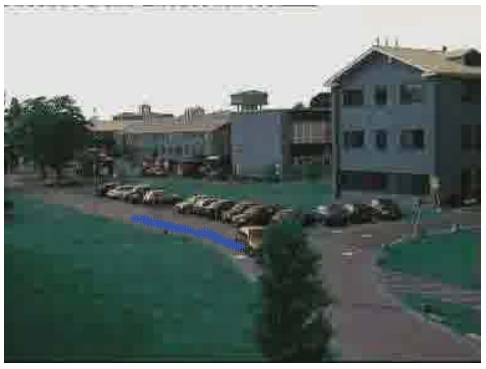
		Frame Offsets			
Video 1	Ground Truth	300	500	800	1000
	Previous Method	301	499	792	989
	Proposed Method	301	498	800	998
	Accuracy	99.67%	99.6%	100%	99.8%
Video 2	Ground Truth	300	500	800	1000
	Previous Method	300	500	807	1000
	Propose Method	299	495	795	999
	Accuracy	99.67%	99%	99.37%	99.9%

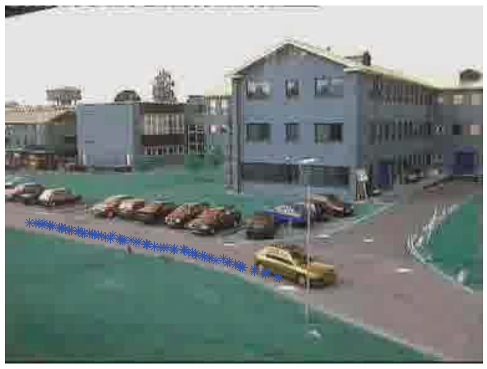
the first M matching pairs. There will be $L = min\,(n - i_{stop}, m - j_{stop})$ more steps after the LCCS stops. Thus, the number of operations becomes $O(N_1 * N_2 * (M * m * C + L))$. M is normally much smaller than n, which reduces the total number of operations approximately by M/n. In our experiments, $(M/n_{avg}) < 0.2$. Thus, the running time and complexity is reduced significantly compared to the previous work in [123].

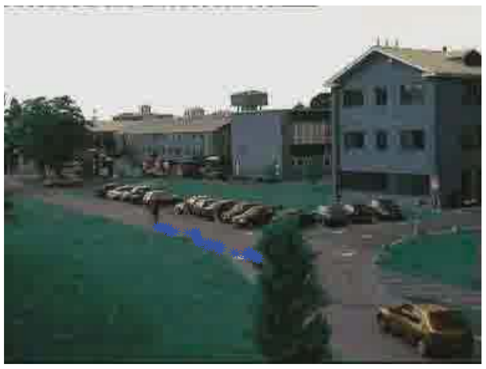
8.4 Experimental Results

The proposed algorithm is tested on the trajectory data obtained from the video sequences in the PETS2001 database. One of the two sequences of each video set is delayed by a known offset. In this way, the ground truth for the frame offset is known for each experiment. In our experiments, we use $\epsilon = 20$ and $M = 5$.

The examples of the matched trajectories from two cameras are shown in Figure 8.1. As seen in Figure 8.1(d), the algorithm is robust to errors of the background subtraction algorithm, which caused a zigzag-like trajectory. Table 8.2 shows the results obtained after


(a) Trajectory on the 1st view


(b) The match of the trajectory in (a)


(c) Trajectory on the 1st view

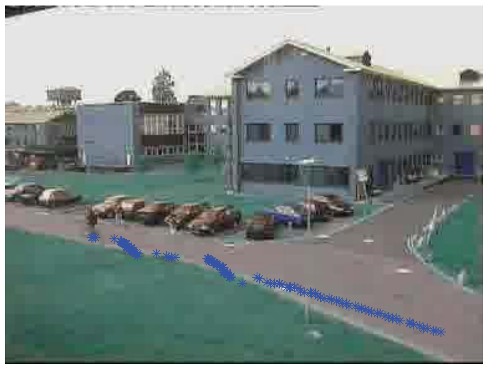

(d) The match of the trajectory in (c)

Figure 8.1: Examples of matched trajectories in two cameras

the confidence check step together with the ground truth. Results obtained with the proposed algorithm and the previous method in [123] are displayed together. The proposed method, which provides significant improvement in terms of computational complexity, has comparable or better results with respect to our previous work. If background subtraction and location extraction results are more accurate, better results can be achieved with the proposed method.

8.5 Conclusions

We presented a computationally efficient and robust algorithm to match and align object trajectories from unsynchronized cameras, and thus to recover the frame offset. This method employs LCCS during the trajectory matching. Compared to the previous work in [123], which performs an exhaustive search, the proposed algorithm reduces the operation time by a factor of M/n_{avg}, where $M = 5$ in the experiments, and n_{avg} is the average trajectory length. While providing significant improvement in terms of computational complexity, the proposed algorithm has comparable or better results with respect to our previous work. It is reliable and robust to possible errors due to background subtraction or location extraction. After performing the experiments with different frame offsets and different video sequences, an average accuracy rate of 99.63% is achieved.

Chapter 9

Edge And Motion Detection On Focal Plane

9.1 Introduction

As presented in Part II, a smart camera usually consists of a CCD or CMOS image sensor, an on-board processor, memory, communication interfaces and other supporting circuits. Smart cameras with embedded processors have become stand-alone units, and will play an increasingly important role in sensor networks. With wired or wireless interfaces, a smart camera can form a sensor node in a sensor network, acquiring images, processing data, and communicating with other sensor nodes.

In sensor networks, possibly with very large number of sensor nodes, cost of each node becomes an important factor. Moreover, when nodes are battery-operated, the power source is limited. Thus, it is very important to consider complexity, power consumption and cost when designing image sensor chips for embedded smart camera nodes. Complex structure in the image sensor will increase the silicon area, and will, in turn, increase the cost and power consumption. Meanwhile, the resolution provided by the camera should be high enough for the computer vision tasks. However, fabricating a higher-resolution sensor as a single chip requires more complex read-out circuitry and more area, and is much more expensive compared to lower-resolution sensor chips. Salas et al. [129] provided a detailed comparison of fabricating a higher-resolution sensor as a single chip and tiling lower-resolution embedded smart cameras in terms of bandwidth, clock frequency, area, power, cost and global computations. They showed that if multiple lower-resolution cameras are tiled, this will provide lower read-out bandwidth, lower read-out circuit complexity, higher robustness and lower costs compared to a single-chip high-resolution sensor.

We built a tiled, embedded smart camera system with four cameras, and tiled the cameras in two different combinations by placing them in 2×2 and 1×4 arrangements. We performed experiments by stitching the individual camera images both automatic and semi-automatic ways. By using two different camera placements and stitching, we obtained two different higher-resolution images from four cameras. This flexibility is another advantage provided by tiling multiple lower-resolution cameras instead of using a single-chip, higher-resolution camera.

Another challenge of wireless sensor networks is that bandwidth is limited, and transmitting data consumes energy. If the raw frame can be processed on the focal plane, it will significantly reduce the processing time and the size of transmitted data. As stated in previous chapters, moving objects are often of interest. In this case, we can transmit reduced-sized data by only sending information about the moving objects, such as their edge and color information. Therefore, edge and motion detection in the embedded smart cameras will find great use. However, smart camera nodes have limited processing power and memory. Thus, it is very important to carefully use and allocate the processing power in the microprocessor to different vision tasks.

Most of the previous work on edge detection has focused on implementing the algorithms in the embedded processor. An alternative way is to perform edge detection on the focal plane. Edge detection on focal plane provides the advantages of having higher speed and low power consumption. It also reduces the load on the embedded processor and spares the precious memory and processing power. Image sensors with focal plane edge detection have been fabricated [132][133]. In [133], a CMOS image sensor was introduced. The edges are obtained by comparing the values of two neighbor rows in the sensor array. Since it can only detect the difference between two rows, the edge strength in the horizontal direction will be lost. In [132] a CCD image sensor is used, and edges are detected in both horizontal and vertical directions. However, CCD sensors have relatively complex structure, higher power consumption and higher cost.

In our embedded smart cameras, we used a CMOS image sensor with focal-plane edge detection integrated. We can compute differences in the pixel values both in the horizontal and vertical directions by this imager, and combine these two difference arrays in the microprocessor to obtain an edge strength output. Then, we convert the edge images into binary images by applying a threshold.

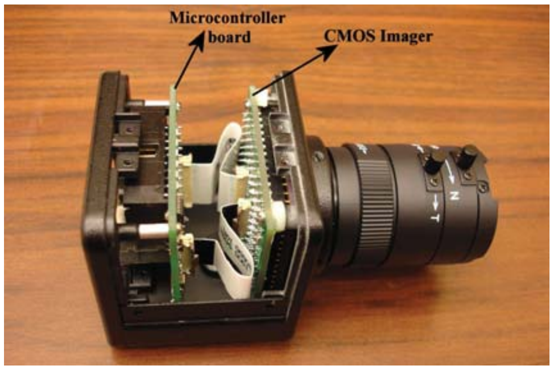
In addition to edge detection, our imager has the capability of performing motion detection on the focal plane. By subtracting the previous edge map from the edge map of the current frame, we detect the edges of moving objects.

In the remainder of this chapter, the employed low-resolution smart cameras are first introduced. Then edge detection on focal plane is explained in detail, from the structure to particular circuits. Two possible tiling combinations to obtain a higher-resolution image are shown, and our method to stitch individual views is discussed. Experimental results are presented and this chapter is concluded at the end.

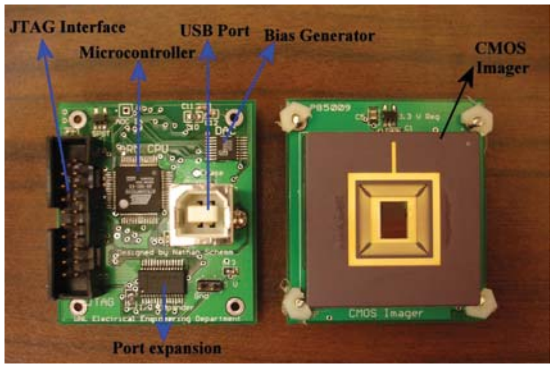
9.2 The Low-Resolution Embedded Smart Camera

The properties of our low-resolution embedded smart camera have been presented in [129]. Here, a brief summary is given for continuity and convenience.

Each of the cameras consists of a customized CMOS imager, an embedded microprocessor, interfaces and other supporting circuits. Figures 9.1 (a) and (b) are the photos of the existing smart camera and the CMOS imager used in this camera, respectively [3].



(a)



(b)

Figure 9.1: (a) the embedded smart camera, (b) the board and the CMOS imager used in this camera.

The standard CMOS processes offer the advantage of being able to integrate analog, digital, or mixed signal processing and computation circuits on the same silicon chip with the sensor [135][136]. The integration of photo-detectors and computational circuitry opens the possibility of performing image processing on the focal plane before the image is read out. However, the photo-detectors available in standard CMOS processes suffer from lower SNR, lower dynamic range and higher fixed pattern noise (FPN). In the CMOS imagers, Active Pixel Sensor technology is employed that increases the signal-to-noise ratio (SNR), and improves the dynamic range by integration of the active amplifier into each pixel. FPN is also greatly reduced by employing Correlated Double Sampling (CDS) techniques.

A 32-bit RISC ARM microcontroller (AT91SAM7S256), with 64 KB internal, high-speed

SRAM and 256 KB internal high-speed flash memory, is used in this camera. Due to the low resolution of the image sensor (80 × 44), this microcontroller has enough resources to read out the image data from the imager, store the necessary data and process them before transmission.

Thanks to the smart camera architecture, three hierarchical levels of processing are available as shown in Figure 9.2. At the sensor chip level, low-level vision operations such as edge and motion detection can be performed. These operations are carried out by analog circuits integrated on the focal plane, i.e. on the same chip the image sensor is on. The second processing level is formed by embedded controllers. Due to their programmability and computational power, they can perform higher level vision tasks like moving object segmentation. The third level of processing comes from the distributed computation across the tiles.

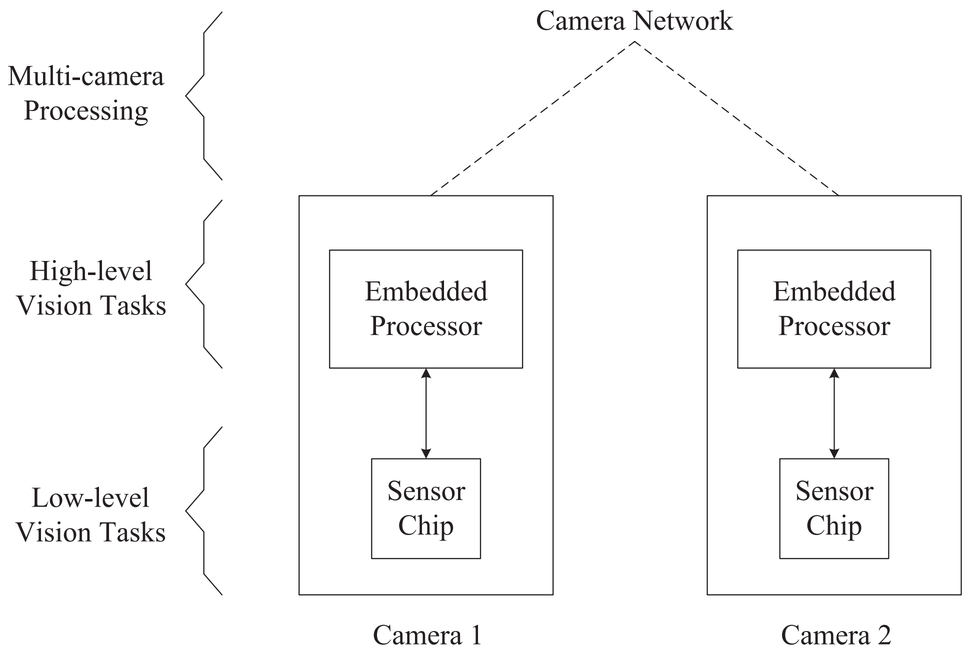


Figure 9.2: Three levels of processing obtained by the smart camera architecture [129].

9.3 Edge Detection on the Focal Plane

9.3.1 CMOS Imager Architecture

The CMOS imager adopts a column-level architecture. The advantage of column-parallel architecture is that it relaxes the speed constraints of the analog-to-digital converters. It has also the advantage of enabling a sequential conversion and readout [131].

The diagram of the column-level architecture is shown in Figure 9.3. There are two vertical shift registers. The first one controls the reset transistors in one row and the second one enables one row at a time for readout into the horizontal shift register. For each column of the sensor array, there is a column-level processor. The column-level processor reads the data from the horizontal shift register and then converts the analog signal to digital values.

A column-level processor is composed of an analog memory bank, a CDS circuit, a single-slope A/D and read-out logic. The featured structure of the memory bank and CDS circuit make edge and motion detection possible on the focal plane. The details will be explained next in Section 9.3.2.

9.3.2 Focal Plane Processing

The CMOS imager has a structure that allows edge detection and subtraction of two consecutive frames to be performed on the focal plane. The operation to perform is selected based on the values of three signals: $intra/inter$, W/N, and $odd/even$. These three signals can be controlled by the microprocessor. If $intra/inter = 0$, it means that the inter-frame mode is selected, and the current pixel values will be compared with the values from the previous frame. Thus, the difference frame will be obtained by this selection. If $intra/inter = 1$, then the pixel value will be compared with the west- or the north-neighbor, and edge detection on the focal plane will be performed. By using the signal W/N, one of two modes can be selected for the edge detection. If $W/N = 1$, the pixel value of the west-neighbor will be read and stored. If $W/N = 0$, then the pixel value of the north-neighbor will be stored. Thus, edge gradient will be computed in horizontal or vertical direction depending on the value of the W/N.

Another control signal is *odd/even*. The *odd/even* line alternates between 1 and 0 every time the row number changes. If the active row number is an odd number, *odd/even* is set to 1, otherwise it is reset to 0. We denote the current pixel value by X , the pixel value to be compared with X by $\hat{X}$. $\hat{X}$ can either be the value from the previous frame or the value of a neighboring pixel. Then, by alternating *odd/even* signal, X and $\hat{X}$ will be presented to the A/D converter in the right order.

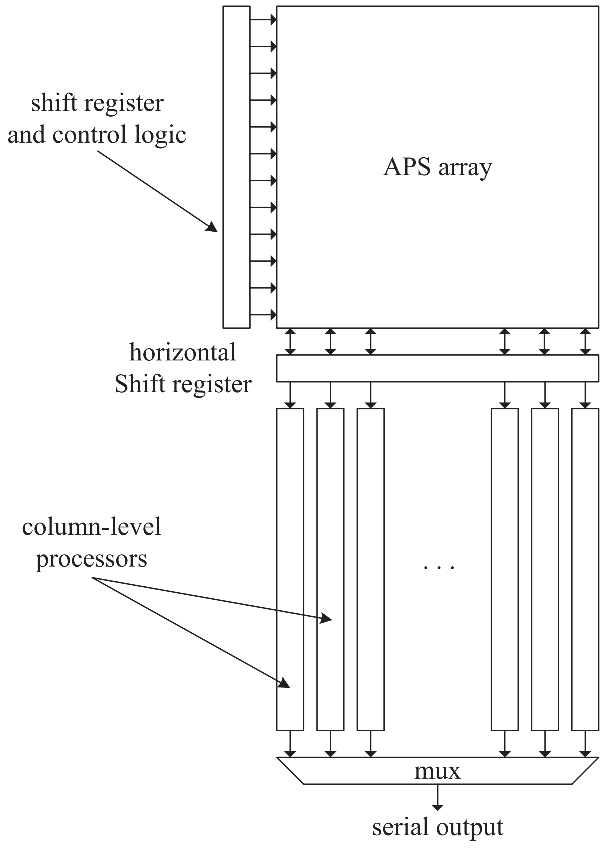


Figure 9.3: Diagram of the column-level architecture [131].

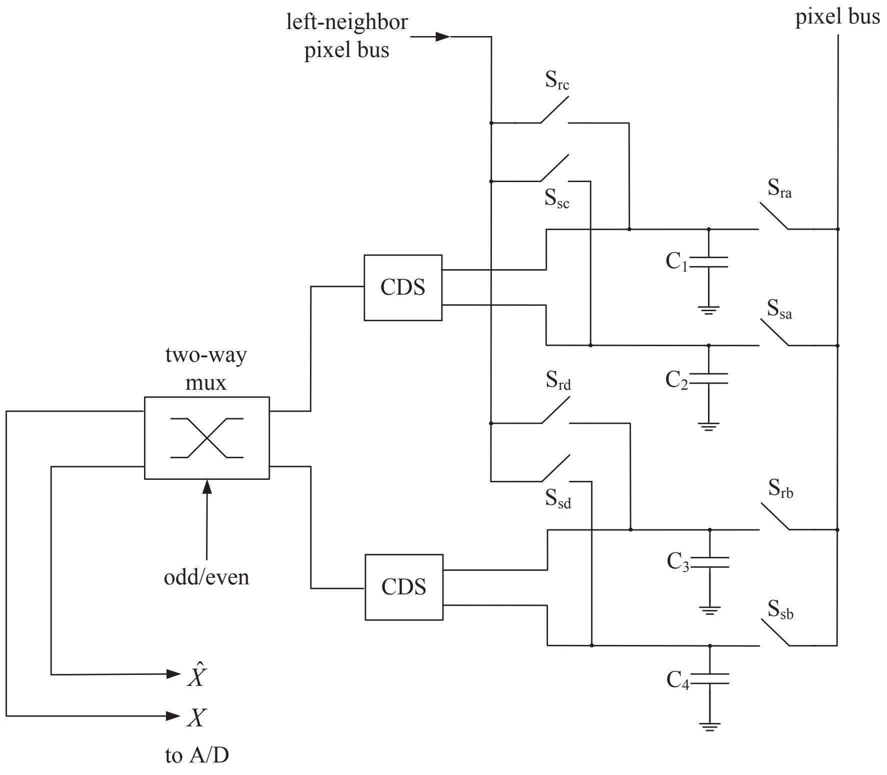


Figure 9.4: The diagram of the edge detection circuit [131].

Figure 9.4 shows the diagram of the edge detection circuit. It consists of four capacitors, eight switches, two correlated-double-sampling (CDS) blocks and a two-way MUX.

The capacitors C_1 to C_4 work as memory banks to store the voltages from the pixel values X and $\hat{X}$. The storage of the pixel value X is alternated between the capacitor pairs $C_1 - C_2$ and $C_3 - C_4$. This alternation is controlled by the eight switches (S_{ra}, S_{rb}, S_{rc}, S_{rd}, S_{sa}, S_{sb}, S_{sc} and S_{sd}). The odd/even line keeps track of the capacitor pair that currently stores the pixel value X. The other capacitor pair will be storing the value of $\hat{X}$. In different pairs of capacitors, C_1 and C_3 store the reset voltages, while C_2 and C_4 store the voltages of either X or $\hat{X}$. The reason we alternate the storage between the capacitor pairs is to be able to sample a new row of pixels while keeping the row sampled before, which after sampling the new row becomes the previous row. When the edge detection is being performed, the imager

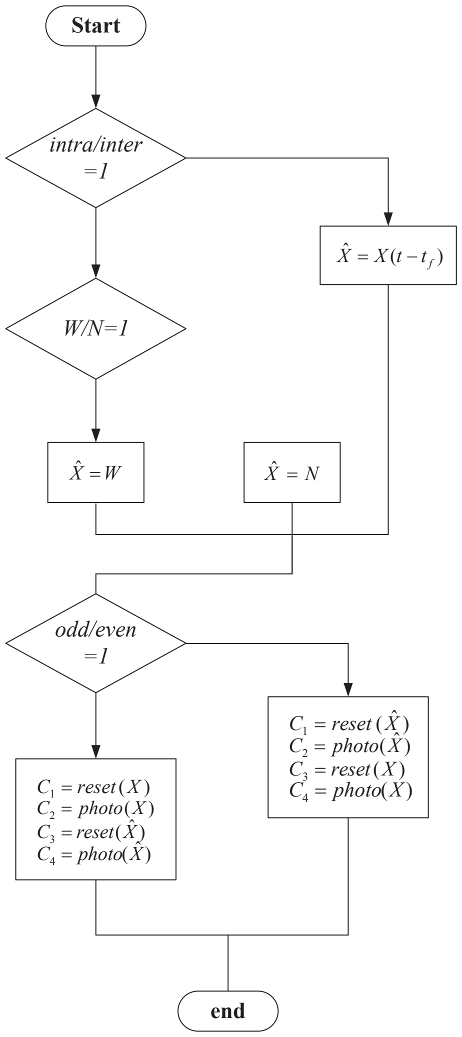


Figure 9.5: The flow diagram of pixel selection [131].

does not need to read each row twice, which makes the readout process more efficient.

The pixel sampling process is illustrated in Figure 9.5. In this figure, the notation $\hat{X} = X(t - t_f)$, means that the time difference between the current and previous frame is t_f. When an odd row is being read-out, the reset voltages of X and $\hat{X}$ are sampled in C_1 and C_3, respectively, and the photo-generated voltages are sampled in C_2 and C_4. Similarly, when an even row is read-out, C_2 and C_4 store the reset voltages, and C_1 and C_3 store the photo-generated voltages [131].

The CDS circuit is a two-transistor differential amplifier [137][138]. Neglecting second order effects, the output of the circuit is given by

$$V_{out} = V_{DD} - (Vr - Vs)$$

where Vr represents the pixel reset voltage and Vs is the photo-generated pixel voltage. The output of the CDS representing the difference between Vr and Vs, is the value of either X or $\hat{X}$. The function of the two-way multiplexer is to decide which output of two CDS circuits is X and which one is $\hat{X}$, and thus to guarantee that the values of X and $\hat{X}$ are presented to the A/D converter in the right order. Then, X and $\hat{X}$ are quantized and read out to the microcontroller.

9.3.3 Combination of Horizontal and Vertical Edge Strengths

Limited by the architecture presented above, either the horizontal edge gradient component or the vertical gradient component can be computed at a time. If $W/N = 1$, the pixel value of the west-neighbor is stored. If $W/N = 0$, then the pixel value of the north-neighbor is stored. Let E_w denote the difference between the current pixel value and its west-neighbor. Similarly, let E_n denote the difference between the current pixel value and its north-neighbor. Instead of computing edge strength in only one direction, we combine the gradient components to obtain the full edge strength. We read out E_w and E_n one at a time separately, and combine them in the microcontroller by using

$$E = \sqrt{(E_w^2 + E_n^2)} \tag{9.1}$$

where E represents the full edge strength. Then, we use a threshold T to build a binary edge map. When $E(i,j) > T$, we set the pixel value at location (i,j) to be 1, otherwise we

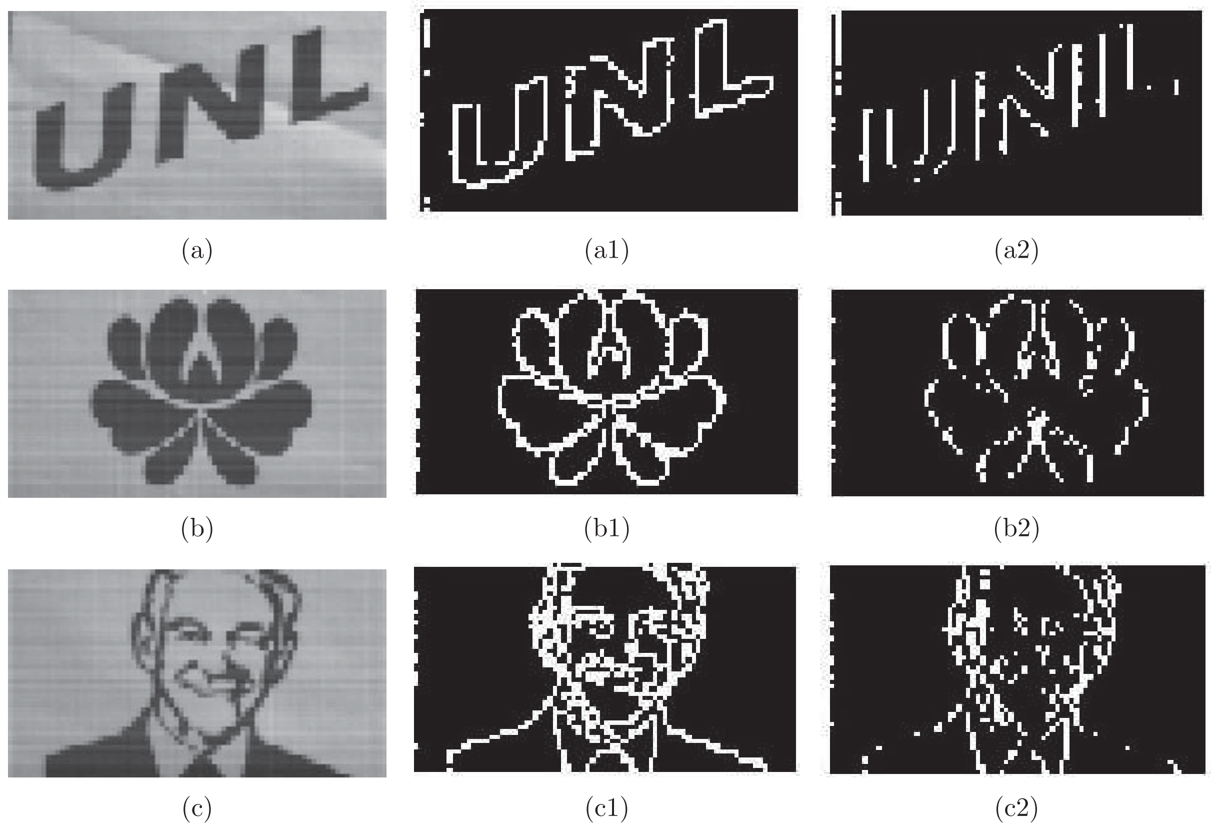


Figure 9.6: Comparison of the detected edges on the focal plane with the full edge strength calculation and west-edge strength only: (a)(b)(c) Original images, (a1)(b1)(c1) edges obtained with the proposed full-edge strength method, (a2)(b2)(c2) edges obtained with west-edge strength only.

set it to be 0.

In the middle column of Figure 9.6, we present three edge maps detected on the focal plane by using the proposed method. We also compare the edge maps obtained by the proposed method and by using horizontal direction (E_w) only. As can be seen by comparing the second and third columns of Figure 9.6, the proposed approach provides much better edge maps.

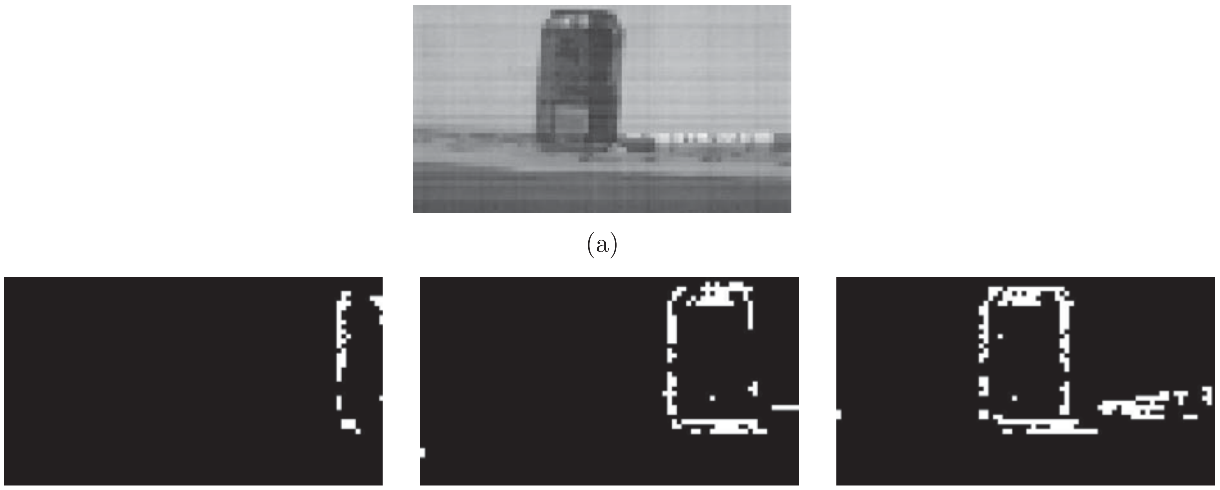


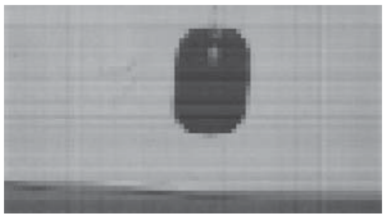
Figure 9.7: Edge-based detection of a moving bottle: (a) An example frame, (b)(c)(d) edges of the moving object at different instances.

9.3.4 Motion Detection in the Microcontroller

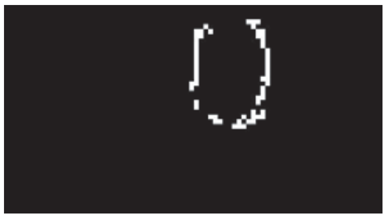
With the structure presented above, our imager has the capability of motion detection on the focal plane. This can be achieved by subtracting the consecutive frames. However, the microcontroller has a slow processing cycle, and our current capacitors are too small to hold the previous pixel value for longer periods of time. As future work, the microcontroller can be replace by an FPGA to realize this functionality.

In order to illustrate what is possible on the focal plane, we implemented subtraction of two consecutive edge maps in the microcontroller. This provides an edge-based motion detection. Let E_t and E_{t-1} denote the current and previous edge maps respectively. We compute $E_{diff} = E_t - E_{t-1}$. Since edge maps are binary images, E_{diff} can be -1, 0 or 1. We only keep the locations with value 1 in the output to get the current edges of the moving object.

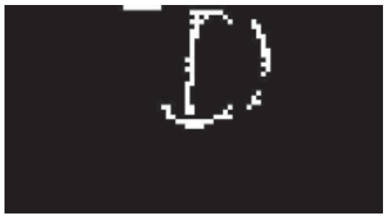
Figures 9.7 and 9.8 show the edges of the moving objects detected by our method.



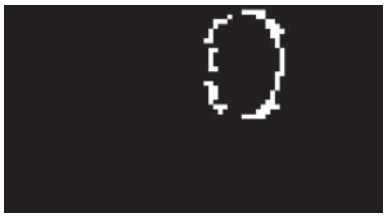
(a)



(b)



(c)



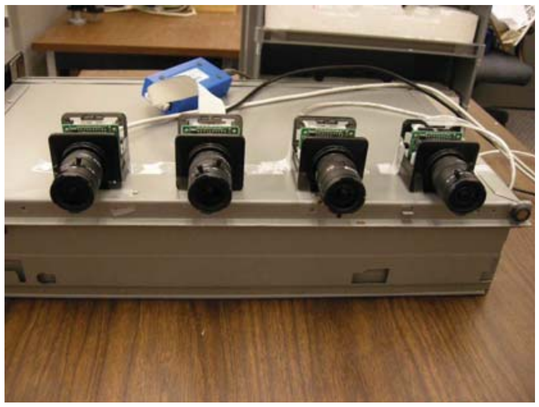
(d)

Figure 9.8: Edge-based detection of a moving mouse:a) An example frame, (b)(c)(d) edges of the moving mouse at different instances.

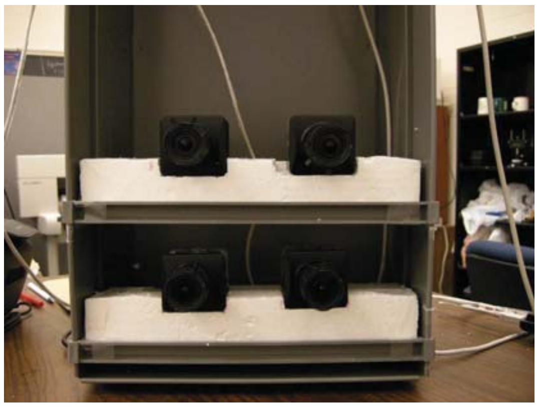
9.4 Tiling of the Multiple Embedded Smart Cameras

9.4.1 Two Different Tiling Arrangements

Advantages of tiling multiple low-resolution embedded smart cameras instead of fabricating a higher-resolution sensor as a single chip are discussed and demonstrated in [129]. These



(a)



(b)

Figure 9.9: (a) 1×4 array and (b) 2×2 array of our smart cameras.

advantages include lower read-out bandwidth, lower read-out circuit complexity, lower power consumption, lower costs and higher robustness.

Another additional advantage of tiling multiple cameras is that they can be arranged in multiple ways, and thus different higher resolution images can be obtained.

We tiled four smart cameras in two different ways: 2×2 tiling arrangement and 1×4 tiling arrangement. We then stitched individual camera images to obtain two different higher-resolution images. If there are more cameras available, such as 16 or 32 cameras, more tiling combinations become possible, and different resolution images can be obtained.

After stitching the original images, we stitch the edge maps, detected on the focal plane, by using the same corresponding point locations and offsets, and obtain a higher-resolution edge map.

Figure 9.9 shows the photos of the two ways we arrange our four embedded cameras. In both arrangements, the cameras are placed so that their optical axes are parallel to each other, and the distances between neighboring cameras in the horizontal direction are the same.

9.4.2 Stitching Algorithm for Calibrated Cameras

Most of the previous work about stitching or mosaicking is based on Scale Invariant Feature Transform (SIFT) [139] and RANSAC [140]. RANSAC takes the potential point matches found by SIFT, and filters out the outliers by using a constraint, such as homography or fundamental matrix constraint. This method is mostly suitable for relatively high-resolution images, since finding the corresponding points becomes challenging otherwise. In our case, due to the low resolution of images, we are not guaranteed to obtain eight or more corresponding point pairs necessary to compute the fundamental matrix.

We implemented an alternative way to obtain the fundamental matrix and to pick the

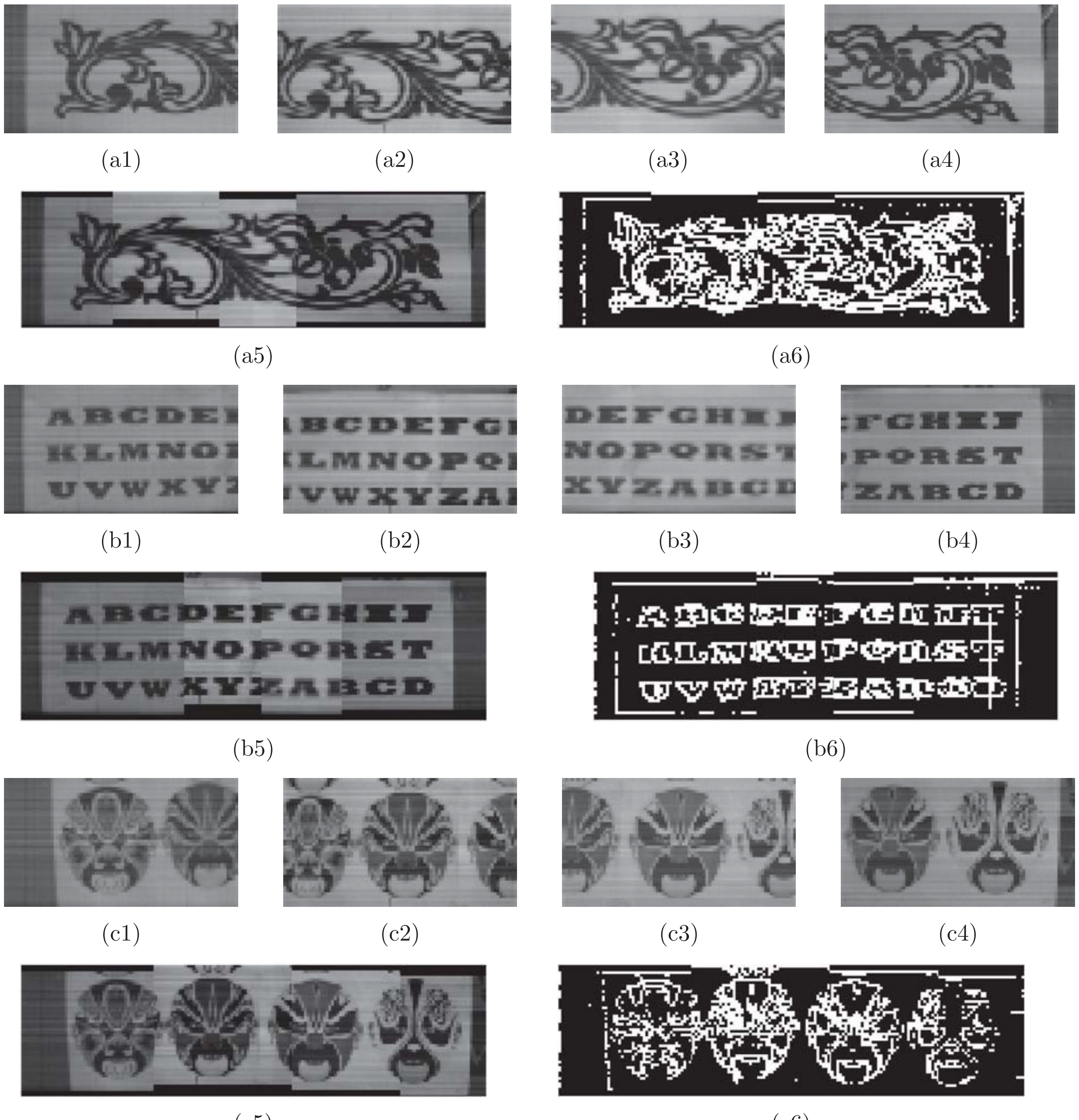


Figure 9.10: Automatically stitched higher-resolution images and edge maps obtained on the focal plane.

best corresponding point pair. First, we calibrate our cameras and obtain the camera matrix P for each camera. P includes both intrinsic and extrinsic parameters of a camera. Let P,

P' and e, e' denote the two camera matrices and two epipoles in a stereo system. The fundamental matrix F can be computed by

$$F = [e']_x P' P^+ \tag{9.2}$$

where P^+ is the pseudo-inverse of P:

$$P^+ = P^T (PP^T)^{-1} \tag{9.3}$$

and $[e']_x$ is the corresponding skew-symmetric matrix of e', defined as [142]:

$$[e']_x = \begin{bmatrix} 0 & -e'(3) & e'(2) \\ e'(3) & 0 & -e'(1) \\ -e'(2) & e'(1) & 0 \end{bmatrix} \tag{9.4}$$

We then use the SIFT algorithm [143] to find matched points in the image pairs. Since we already computed the fundamental matrix by Eq. (9.2), we do not need many matched points at this step. Let x and x' denote two points on two images, respectively. If x' is the corresponding point of x, then

$$x'^T F x = 0 \tag{9.5}$$

We take the point pairs, which are the output of the SIFT algorithm, and find the pair for which $x'^T F x$ is minimum. Our cameras are placed so that their optical axes are parallel to each other. Thus, one corresponding point pair is sufficient to stitch these camera images. By using the point pair for which $x'^T F x$ is minimum, we adjust the position of the left, right or top/bottom images, and stitch them to one another. We stitch the edge maps in the same way, by using the coordinates of the best matched points.

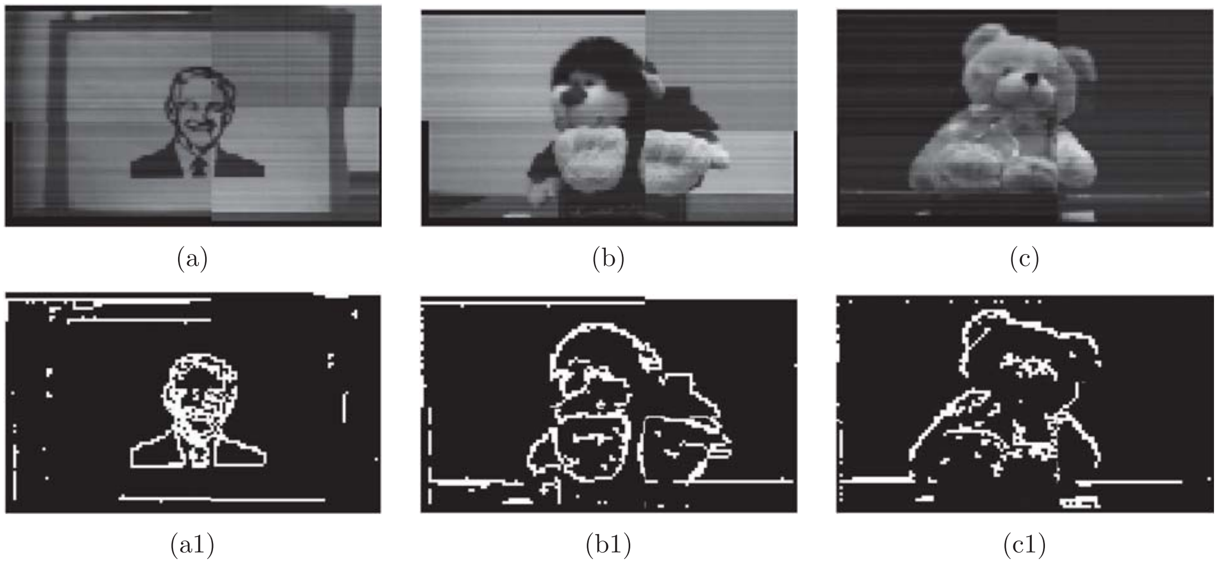


Figure 9.11: Higher-resolution images and edge maps obtained in the focal plane with the 2×2 camera setup.

9.5 Experimental Results

The resolution of our embedded smart cameras is 80×44. To be able to obtain corresponding points on different camera views, cameras are placed so that around one third of their fields of view overlap. The final stitched images we obtained are around 170×50 for the 1×4 camera placement, and 125×80 for the 2×2 camera placement.

We first set up our cameras as shown in Figure 9.9 (a) in 1×4 arrangement. We stitched the individual camera views as described in Section 9.4.2. Figure 9.10 shows the automatically stitched, higher-resolution images and edge maps for three different cases. Figures 9.10 (a1)-(a4), (b1)-(b4) and (c1)-(c4) are the images seen by each camera. Figures 9.10 (a5),(b5) and (c5) are the automatically stitched, higher-resolution images, and Figures 9.10 (a6),(b6) and (c6) are the higher-resolution edge maps.

We then set up our cameras as shown in Figure 9.9 (b) in 2×2 arrangement. We automatically stitched the two images in the upper and lower rows among themselves. Figure

9.11 shows examples of stitched images obtained with the 2×2 camera placement.

However, the stitching algorithm is based on the assumption that the cameras are places in a perfectly parallel direction. In practical, there might be minor disparity of the cameras positions, which makes the output image might not be perfectly smooth. Due to the low resolution of the smart cameras, there might be a small inaccuracy in the camera matrices and the fundamental matrix. This inaccuracy might cause a mismatch when selecting a best matching pair of corresponding points. In this situation, user's input could be an alternative way to select one best matching pair from the SIFT matching points.

9.6 Conclusions

We presented a customized CMOS imager with edge detection on the focal plane in this chapter. With the featured structure of the CMOS imager, full edge strength maps are obtained on the focal plane. Then, a simple frame differencing is performed based on the edge maps in the microprocessor to detect moving objects. With the featured CMOS imager, we built up four low-resolution embedded smart cameras. These four cameras are arranged in two different position combinations to obtain two different higher-resolution images. We also stitched the edge maps detected on the focal plane in the same way.

Bibliography

[1] N. Atsushi, K. Hirokazu, H. Shinsaku and I. Seiji, "Tracking multiple people using distributed vision systems," *Proc. of IEEE Int'l Conf. on Robotics and Automation*, pp. 2974–2981, 2002.

[2] M. Bramberger, A. Doblander, A. Maier, B. Rinner and H. Schwabach, "Distributed embedded smart cameras for surveillance applications," *IEEE Computer*, vol. 39, pp. 68–75, 2006.

[3] M. Casares and S. Velipasalar, "Light-weight salient foreground detection for embedded smart cameras," *Proc. of ACM/IEEE Int'l Conf.on Distributed Smart Cameras*, 2008.

[4] M. Casares and S. Velipasalar, "Light-weight salient foreground detection with adaptive memory requirement," *Proc. of the IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing*, 2009.

[5] M. Casares, S. Velipasalar and A. Pinto, "Light-weight salient foreground detection for embedded smart cameras," *Computer Vision and Image Understanding*, vol. 114, no. 11, pp. 1223–1237, 2010.

[6] P. Chen, P. Ahammad, C. Boyer, S. Huang, L. Lin, E. Lobaton, M. Meingast, S. Oh, S. Wang, P. Yan, A. Yang, C. Yeo, L. C. Chang, J. D. Tygar and S. S. Sastry, "Citric:

A low-bandwidth wireless camera network platform," *Proc. of ACM/IEEE Int'l Conf. on Distributed Smart Cameras*, 2008.

[7] R. T. Collins, A. J. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, O. Hasegawa, P. Burt and L. Wixson, "A system for video surveillance and monitoring: VSAM final report," *Technical report CMU-RI-TR-00-12, Robotics Institute*, Carnegie Mellon University, 2000.

[8] R. T. Collins, A. J. Lipton, H. Fujiyoshi and T. Kanade, "Algorithms for cooperative multisensor surveillance," *Proc. of the IEEE*, vol. 89, no. 10, pp. 1456–1477, 2001.

[9] A. Yilmaz, O. Javed and M. Shah, "Object tracking: A survey," *ACM Computing Surveys*, vol. 38, no. 4, 2006.

[10] S. Dockstader and A. M. Tecklp, "Multiple camera tracking of interacting and occluded human motion," *Proc. of the IEEE*, pp. 1441–1445, vol. 89, no. 10, 2001.

[11] H. Yang, L. Shao, F. Zheng, L. Wang and Z. Song, "Recent advances and trends in visual tracking: A review," *Neurocomputing*, vol. 74, no. 18, pp. 3823–3831, 2011.

[12] D. Comaniciu, V. Ramesh and P. Meer, "Real-time tracking of non-rigid objects using mean shift," *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, vol. 2, pp. 142–149, 2000.

[13] I. Downes, L. B. Rad and H. Aghajan, "Development of a mote for wireless image sensor networks," *Proc. of Cognitive systems with Interactive Sensors*, 2006.

[14] A. Elgammal, D. Harwood and L. Davis, "Non-parametric model for background subtraction," *Proc. of European Conf. on Computer Vision*, pp. 751–767, 2000.

[15] T. Ellis, "Multi-camera video vurveillance," *Proc. of Int'l Carnahan Conf. on Security Technology*, pp. 228–233, 2002.

[16] T. H. Chang and S. Gong, "Tracking multiple people with a multi-camera system," *Proc. of IEEE Workshop on Multi-Object Tracking*, 2001.

[17] Wu-chi Feng, Wu-chang Feng and M. L. Baillif, "Panoptes: Scalable low-power video sensor networking technologies," *Proc. of ACM Conf. on Multimedia*, pp. 562–571, 2003.

[18] S. Fleck, F. Busch, P. Biber and W. Strasser, "3D Surveillance–A distributed network of smart cameras for real-time tracking and its visualization in 3D," *Proc. of Conf. on Computer Vision and Pattern Recognition Workshop*, pp. 118–118, 2006.

[19] I. Haritaoglu, D. Harwood and L. S. Davis, "W^4: Real-time surveillance of people and their activities," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 809–830, 2000.

[20] S. Hengstler, D. Prashanth, S. Fong and H. Aghajan, "Mesheye: A hybrid-resolution smart camera mote for applications in distributed intelligent surveillance," *Proc. of Int'l Symposium on Information Processing in Sensor Networks*, pp. 360–369, 2007.

[21] K. Kim, T. H. Chalidabhongse, D. Harwood and L. Davis, "Real-time foreground-background segmentation using codebook model," *Journal of Real-time Imaging*, vol. 11, pp. 172–185, 2005.

[22] R. Kleihorst, A. Abbo, B. Schueler and A. Danilin, "Camera mote with a high-performance parallel processor for real-time frame-based video processing," *Proc. of ACM/IEEE Int'l Conf. on Distributed Smart Cameras*, pp. 106–116, 2007.

[23] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale and S. Shafer, "Multi-camera multi-person tracking for EasyLiving," *Proc. of IEEE Int'l Workshop on Visual Surveillance*, pp. 3–10, 2000.

[24] G. Kayumbi and A. Cavallaro, "Robust homography-based trajectory transformation for multi-camera scene analysis," *Proc. ACM/IEEE Int'l Conf. on Distributed Smart Cameras*, pp. 59–66, 2007.

[25] P. Kulkarni, D. Ganesan and P. Shenoy, "The case for multi-tier camera sensor network," *Proc. of ACM Workshop on Network and Operating System Support for Digital Audio and Video*, 2005.

[26] B. P. Lai Lo, J. Sun and S. A. Velastin, "Fusing visual and audio information in a distributed intelligent surveillance system for public transport systems," *Acta Automatica Sinica*, pp. 393–407, 2003.

[27] R. Nevatia, J. Hobbs and B. Bolles, "An ontology for video event representation," *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition Workshops*, 2004.

[28] K. Nguyen, G. Yeung, S. Ghiasi and M. Sarrafzadeh, "A general framework for tracking objects in a multi-camera environment", *Proc. of the Int'l Workshop on Digital and Computational Video*, pp. 200–204, 2002.

[29] N. Oliver, B. Rosario and A. Pentland, "A bayesian computer vision system for modeling human interactions," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, pp. 831–834, 2000.

[30] I. Pavlidis, V. Morellas, P. Tsiamyrtzis and S. Harp, "Urban surveillance systems: from the laboratory to the commercial world," *Proc. of the IEEE*, vol. 89(10), pp. 1478–1497, 2001.

[31] M. Piccardi, "Background subtraction techniques: a review," *Proc. of the IEEE Int'l Conf. on Systems, Man and Cybernetics*, vol. 4, pp. 3099–3104, 2004.

[32] M. Quaritsch, M. Kreuzthaler, B. Rinner, H. Bischof and B. Strobl, "Autonomous multicamera tracking on embedded smart cameras," *EURASIP Journal on Embedded Systems*, 2007.

[33] M. Rahimi, R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin and M. Srivastava, "Cyclops: In situ image sensing and interpretation in wireless sensor networks," *Proc. of the Int'l Conf. on Embedded Networked Sensor Systems*, pp. 192–204, 2005.

[34] B. Rinner and W. Wolf, "An introduction to distributed smart cameras," *Proc. of the IEEE*, vol. 96, no. 10, pp. 1565–1575, 2008.

[35] B. Rinner, T. Winkler, W. Schriebl, M. Quaritsch and W. Wolf, "The evolution from single to pervasive smart cameras," *Proc. of ACM/IEEE Int'l Conf. on Distributed Smart Cameras*, 2008.

[36] A. Rowe, C. Rosenberg and I. Nourbakhsh, "A second generation low cost embedded color vision system," *Proc. of IEEE Embedded Computer Vision Workshop in conjunction with IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, vol. 3, pp. 136, 2005.

[37] L. Shapiro and G. Stockman, "Computor Vision," *Prentice Hall*, 2001.

[38] C. Stauffer and W. E. L. Grimson, "Learning patterns of activity using real-time tracking," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 747–757, 2000.

[39] T. Kanade, R. T. Collins, A. J. Lipton, P. Burt and L. Wixson, "Advances in cooperative multi-sensor video surveillance," *Proc. of DARPA Image Understanding Workshop*, pp. 3–24, 1998.

[40] N. T. Nguyen, S. Venkatesh, G. West and H. H. Bui, "Multiple camera coordination in a surveillance system," *ACTA Automation Sinica*, vol. 29, no. 3, pp. 408–422, 2003.

[41] T. Teixeira, D. Lymberopoulos, E. Culurciello, Y. Aloimonos and A. Savvides, "A lightweight camera sensor network operating on symbolic information," *Proc. of 1st Workshop on Distributed Smart Cameras*, 2006.

[42] C. H. Anderson, P. J. Burt and G. S. V. D. Wal, "Change detection and tracking using pyramid tranform techniques," *Proc. of SPIE Intelligent Robots and Computer Vision*, vol. 579, pp. 72–78, 1985.

[43] P. L. Rosin and T. Ellis, "Image difference threshold strategies and shadow detection," *Proc. of British Machine Vision Conf.*, pp. 347–356, 1995.

[44] S. Velipasalar, J. Schlessman, C. Chen, W. Wolf and J. Singh, "A scalable clustered camera system for multiple object tracking," *EURASIP Journal on Image and Video Processing*, 542808, 2008.

[45] Y. Wang, M. Casares and S. Velipasalar, "Cooperative object Tracking and Event Detection with Wireless Smart Cameras," *Proc. of IEEE Int'l Conf. on Advanced Video and Signal Based Surveillance*, 2009.

[46] Y. Wang, M. Casares and S. Velipasalar,"Detection of composite events spanning multiple camera views with wireless embedded smart cameras," *Proc. of ACM/IEEE Int'l Conf. on Distributed Smart Cameras*, 2009.

[47] C. R. Wren, A. Azarbayejani, T. Darrell and A. P. Pentland, "Pfinder: Real-time tracking of the human body," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 19(7), pp. 780–785, 1997.

[48] X. Yuan, Z. Sun, Y. Varol and G. Bebis, “A distributed visual surveillance system,” *Proc. of IEEE Conf. on Advanced Video and Signal Based Surveillance*, pp. 199–204, 2003.

[49] Z. Zivkovic, “Improved adaptive gausian mixture model for background subtraction,” *Proc. of Int'l Conf. on Pattern Recognition*, pp. 28–31, 2004.

[50] A. Adam, E. Rivlin, I. Shimshoni and D. Reinitz, “Robust real-time unusual event detection using multiple fixed-location monitors,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 30, no. 3, pp. 555–560, 2008.

[51] J. Black, T. Ellis and D. Makris, “A hierarchical database for visual surveillance applications,” *Proc. of IEEE Conf. on Multimedia & Expo*, vol. 3, pp. 1571–1574, 2004.

[52] O. Boiman and M. Irani, “Detecting irregularities in images and in video,” *Proc. of IEEE Int'l Conf. on Computer Vision*, pp. 462–469, 2005.

[53] M. Borg, D. Thirde, J. Ferryman, F. Fusier, V. Valentin, F. Bremond and M. Thonnat, “Video surveillance for aircraft activity monitoring,” *Proc. of IEEE Conf. on Advanced Video and Signal Based Surveillance*, pp. 16–21, 2005.

[54] F. Cupillard, A. Avanzi, F. Bremond and M. Thonnat, “Video understanding for metro surveillance,” *Proc. of IEEE Conf. on Networking, Sensing & Control*, pp. 186–191, 2004.

[55] H. Dee and D. Hogg, “Detecting inexplicable behaviour,” *Proc. of British Machine Vision Conf.*, pp. 477–486, 2004.

[56] Y. A. Ivanov and A. F. Bobick, “Recognition of multi-agent interaction in video surveillance,” *Proc. of IEEE Conf. on Computer Vision*, pp. 169–176, 1999.

[57] G. Medioni, I. Cohen, F. Bremond, S. Hongeng and R. Nevatia, “Event detection and analysis from video streams,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 23, no. 8, pp. 873–889, 2001.

[58] R. Nevatia, T. Zhao and S. Hongeng, “Hierarchical language-based representation of events in video streams,” *Proc. of IEEE CVPR Workshop on Event Mining*, vol. 4, pp. 39–39, 2003.

[59] J. Owens and A. Hunter, “Application of the self-organising map to trajectory classification,” *Proc. of IEEE Int'l Workshop on Visual Surveillance*, pp. 77–83, 2000.

[60] F. Porikli and T. Haga, “Event detection by eigenvector decomposition using object and frame features,” *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 462–469, 2004.

[61] N. Rota and M. Thonnat, “Video sequence interpretation for visual surveillance,” *Proc. of IEEE Int'l Workshop on Visual Surveillance*, pp. 59–68, 2000.

[62] A. Mittal and L. Davis, “M_2Tracker: A multi-view approach to segmenting and tracking people in a cluttered scene,” *Int'l Journal of Computer Vision*, vol. 51, no. 3, pp. 189–203, 2003.

[63] C. Sacchi and C. S. Regazzoni, “A distributed surveillance system for detection of abandoned objects in unmanned railway environments,” *IEEE Trans. on Vehicular Technology*, vol. 49, no. 5, pp. 2013–2026, 2000.

[64] V. D. Shet, D. Harwood and L. S. Davis, “VidMAP: Video monitoring of activity with Prolog,” *Proc. of IEEE Conf. on Advanced Video and Signal Based Surveillance*, pp. 224–229, 2005.

[65] E. Stringa and C. S. Regazzoni, “Real-time video-shot detection for scene surveillance applications,” *IEEE Trans. on Image Processing*, vol. 9, no. 1, pp. 69–79, 2000.

[66] N. Vaswani, A. Chowdhury and R. Chellappa, “Activity recognition using the dynamics of the configuration of interacting objects,” *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 633–640, 2003.

[67] V. T. Vu, F. Bremond and M. Thonnat, “Automatic video interpretation: a novel algorithm for temporal scenario recognition,” *Proc. of Int’l Joint Conf. o Artificial Intelligence*, pp. 9–15, 2003.

[68] Q. Cai, J. K. Aggarwal, “Tracking human motion in structured environments using a distributed-camera system,” *IEEE Tran. on Pattern Analysis and Machine Intelligence*, vol. 21, no. 11, pp. 1241–1247, 1999.

[69] S. Khan and M. Shah, “A multiview approach to tracking people in crowded scenes using a planar homography constraint,” *Proc. of European Conf. on Computer Vision*, pp. 133–146, 2006.

[70] H. Watanabe, H. Tanahashi, Y. Satoh, Y. Niwa and K. Yamamoto, “Event detection for a visual surveillance system using stereo omni-directional system,” *Proc. of Int’l Conf. on Knowledge-Based Intelligent Information and Engineering Systems*, pp. 890–896, 2003.

[71] Y. Jo and J. Han, “A new approach to camera hand-off without camera calibration for the general scene with non-planar ground,” *Proc. of ACM int’l Workshop on Video Surveillance and Sensor Networks*, pp. 195–202, 2006.

[72] T. Xiang and S. Gong, “Beyond tracking: Modelling activity and understanding behaviour,” *Int’l Journal of Computer Vision*, vol. 67, no. 1, pp. 21–51, 2006.

[73] H. Zhong, J. Shi and M. Visontai, "Detecting unusual activity in video," *Proc. of IEEE Conf. on Pattern Recogniton*, pp. 28–31, 2004.

[74] M. Albanese, R. Chellappa, V. Moscato and A. Picariello, "A constrained probabilistic petri net framework for human activity detection in video," *IEEE. Trans. on Multimedia*, vol. 10, no. 8, pp. 1429–1443, 2008.

[75] D. Baltieri, R. Vezzani and R. Cucchiara, "3dpes: 3d people dataset for surveillance and forensics," *Proc. of 1st Int'l ACM Workshop on Multimedia access to 3D Human Objects*, pp. 59–64, 2011.

[76] J. Black, T. Ellis and P. Rosin, "Multi view image surveillance and tracking," *Proc. of Workshop on Motion and Video Computing*, pp. 169–174, 2002.

[77] C. del Blanco, R. Mohedano, N. Garcia, L. Salgado and F. Jaureguizar, "Color-based 3d particle filtering for robust tracking in heterogeneous environments," *Proc. of 2nd ACM/IEEE Int'l Conf. on Distributed Smart Cameras*, 2008.

[78] Y. Cai, K. Huang and T. Tan, "Human appearance matching across multiple non-overlapping cameras," *Proc. of Int'l Conf. on Pattern Recognition*, 2008.

[79] S. Calderara, A. Prati, R. Vezzani and R. Cucchiara, "Consistent labeling for multi-camera object tracking," *Image Analysis and Processing*, vol. 3617, pp. 1206–1214, 2005.

[80] X. Cao, C. Wu, J. Lan, P. Yan and X. Li, "Vehicle detection and motion analysis in low-altitude airborne video under urban environment," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 21, no. 10, pp. 1522–1533, 2011.

[81] E. Cheng, C. Madden and M. Piccardi, "Mitigating the effects of variable illumination for tracking across disjoint camera views," *Proc. of IEEE Int'l Conf. on Video and Signal Based Surveillance*, 2006.

[82] A. Chilgunde, P. Kumar, S. Ranganath and W. Huang, "Multi-camera target tracking in blind regions of cameras with non-overlapping fields of view," *Proc. of the British Machine Vision Conference*, 2004.

[83] I. Cohen, Y. Ma and B. Miller, "Associating moving objects across non-overlapping cameras: A query-by-example approach," *Proc. of IEEE Conf. on Technologies for Homeland Security*, pp. 566–571, 2008.

[84] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *Proc. of IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, vol. 1, pp. 886–893, 2005.

[85] R. David and H. Alla, "Petri nets for modeling of dynamic systems: a survey," *Automatica*, vol. 30, no. 2, pp. 175–202, 1994.

[86] T. D'Orazio, P. Mazzeo and P. Spagnolo, "Color brightness transfer function evaluation for non overlapping multi camera tracking," *Proc. of ACM/IEEE Int'l Conf. on Distributed Smart Cameras*, 2009.

[87] P. Geismann and A. Knoll, "Speeding up hog and lbp features for pedestrian detection by multiresolution techniques," *Proc. of Int'l Conf. on Advances in Visual Computing*, pp. 243–252, 2010.

[88] C. Huang, W. Chiu, S. Wang and J. Chuang, "Probabilistic modeling of dynamic traffic flow across non-overlapping camera views," *Proc. of Int'l Conf. on Pattern Recognition*, 2010.

[89] T. Huang and S. Russell, "Object identification: a bayesian analysis with application to traffic surveillance," *Artificial Intelligence*, vol. 103, no. 1–2, pp. 77–93, 1998.

[90] R. A. Jacobs, "What determines visual cue reliability?," *Trends in Cognitive Sciences*, vol. 6, no. 8, pp. 345–350, 2002.

[91] O. Javed, Z. Rasheed, K. Shafique and M. Shah, "Tracking across multiple cameras with disjoint views," *Proc. of IEEE Int'l Conf. on Computer Vision*, vol. 2, pp. 952–957, 2003.

[92] O. Javed, K. Shafique and M. Shah, "Appearance modeling for tracking in multiple non-overlapping cameras," *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, vol. 2, pp. 26–33, 2005.

[93] K. Jeong and C. Jaynes, "Object matching in disjoint cameras using a color transfer approach," *Machine Vision and Applications*, vol. 19, no. 5, pp. 443–455, 2008.

[94] J. Kang, I. Cohan and G. Medioni, "Persistent objects tracking across multiple non-overlapping cameras," *Proc. of IEEE Workshop on Motion and Video Computing*, 2005.

[95] S. Khan and M. Shah, "Consistent labeling of tracked objects in multiple cameras with overlapping fields of view," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 25, no. 10, pp. 1355–1360, 2003.

[96] C. Madden and M. Piccardi, "A framework for track matching across disjoint cameras using robust shape and appearance features," *Proc. of IEEE Conf. on Advanced Video and Signal Based Surveillance*, pp. 188–193, 2007.

[97] Z. Lin, F. Wen, C.Y. Chung and K.P. Wong, "A survey on the applications of Petri Net theory in Power Systems," *IEEE Power Engineering Society General Meeting*, 2006.

[98] D. Makris, T. Ellis and J. Black, "Bridging the gaps between cameras," *Proc. of IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, 2004.

[99] M. V. Iordache and P. J. Antsaklis, "Petri nets and programming: A survey," *American Control Conference*, pp.4994–4999, 2009.

[100] B. Moller, T. Plotz and G. Fink, "Calibration-free camera hand-over for fast and reliable person tracking in multi-camera setups," *Proc. of Int'l Conf. on Pattern Recognition*, 2008.

[101] Z.W. Li, M.C. Zhou and N.Q. Wu, "A survey and comparison of Petri Net-based deadlock preventio policies for flexible manuafacturing systems," *IEEE Trans on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, vol. 38, no. 2, 2008.

[102] E. Monari, J. Maerker and K. Kroschel, "A robust and efficient approach for human tracking in multi-camera systems," *Proc. of IEEE Int'l Conf. on Advanced Video and Signal Based Surveillance*, pp. 134–139, 2009.

[103] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[104] C. Niu and E. Grimson, "Recovering non-overlapping network topology using far-field vehicle tracking data," *Proc. of Int'l Conf. on Pattern Recognition*, 2006.

[105] G. Tuncel and G. M. Bayhan, "Applications of Petri nets in production scheduling: A review," *The Int'l Journal of Advanced Manufacturing Technology*, vol. 34, no. 7–8, pp. 762–773, 2007.

[106] T. Ojala, M. Pietikainen and D. Harwood, "A comparative study of texture measures with classification based on feature distributions," *Pattern Recognition*, vol. 29, no. 1, pp. 51–59, 1996.

[107] T. Ojala, M. Pietikainen and T. Maenpaa, “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[108] R. Pflugfelder and H. Bischof, “Tracking across non-overlapping views via geometry,” *Proc. of Int’l Conf. on Pattern Recognition*, 2008.

[109] F. Porikli, “Inter-camera color calibration by correlation model function,” *Proc. of IEEE Int’l Conf. on Image Processing*, 2003.

[110] L. Ferrarini, “An incremental approach to logic controller design with Petri nets,” *IEEE Trans. on Systems, Man and Cybernetics*, vol. 22, no. 3, pp. 461–473, 1992.

[111] B. Prosser, S. Gong and T. Xiang, “Multi-camera matching using bi-directional cumulative brightness transfer functions,” *British Machine Vision Conference*, 2008.

[112] A. Rahimi, B. Dunagan and T. Darrell, “Simultaneous calibration and tracking with a network of non-overlapping sensors,” *Proc. of IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, vol. 1, pp. I–187–I–194, 2004.

[113] J. Triesch and C. von der Malsburg, “Democratic integration: Self-organized integration of adaptive cues,” *Neural Computation*, vol. 13, no. 9, pp. 2049–2074, 2001.

[114] B. Berthomieu and M. Diaz, “Modeling and verification of time dependent systems,” *IEEE Trans. on Software Engineering*, vol. 17, no. 3, pp. 259–273, 1991.

[115] J. Billington, G. R. Wheeler and M. C. Wilbur-Ham, “PROTEAN: A high-level Petri net tool for the specification and verification of communication protocols,” *IEEE Trans. on Software Engineering*, vol. 14, no. 3, pp. 301–316, 1988.

[116] X. Wang, T. Han and S. Yan, “An hog-lbp human detector with partial occlusion handling,” *Proc. of Int’l Conf. on Computer Vision*, pp. 32–38, 2009.

[117] R. Zurawski and M. Zhou, “Petri nets and industrial applications: a tutorial,” *IEEE Trans. on Industrial Electronics*, vol. 41, no. 6, 1994.

[118] S. Kuthirummal, C.V. Jawahar, and P.J. Narayanan, “Video frame alignment in multiple views,” *IEEE Int’l Conf. on Image Processing*, vol. 3, pp. 357–360, 2002.

[119] L. Lee, R. Romano, and G. Stein, “Monitoring activities from multiple video streams: Establishing a common coordinate frame,” *IEEE Trans. on PAMI*, pp. 758–767, 2000.

[120] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, “Introduction to Algorithms,” *Massachusetts Institute of Technology*, 2009.

[121] Y. Caspi, D. Simakov, and M. Irani, “Feature-based sequence-to-sequence matching,” *VAMODS workshop*, 2002.

[122] T. Tuytelaars and L. Van Gool, “Synchronizing video sequences,” *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, vol. 1, pp. 762–768, 2004.

[123] S. Velipasalar and W. Wolf, “Frame-level temporal calibration of video sequences from unsynchronized cameras,” *Machine Vision and Applications*, vol. 19, no. 5–6, pp. 395–409, 2008.

[124] M. Vlachos, G. Kollios, and D. Gunopulos, “Discovering similar multidimensional trajectories,” *IEEE Int’l Conf. on Data Engineering*, pp. 673–684, 2002.

[125] D. Buzan, S. Sclaroff, and G. Kollios, “Extraction and clustering of motion trajectories in video,” *IEEE 17th Int’l Conf. on Pattern Recognition*, vol. 2, pp. 521–524, 2004.

[126] A. M. Cheriyadat and R. Radke, “Automatically determining dominant motions in crowded scenes by clustering partial feature trajectories,” *First ACM/IEEE Int’l Conf. on Distributed Smart Cameras*, pp. 52–58, 2007.

[127] T. Teixeira, E. Culurciello, J. H. Park, D. Lymberopoulos, A. Barton-Sweeney, and A. Savvides, “Address-event imagers for sensor networks: evaluation and modeling,” *Proc. of Int’l Conf. on Information Processing in Sensor Networks*, pp. 458–466, 2006.

[128] P. J. Shin, J. Park, and A.C. Kak, “A QOS evaluation testbed for MAC protocols for wireless camera networks,” *Proc. of ACM/IEEE Int’l Conf. on Distributed Smart Cameras*, pp. 235–242, 2007.

[129] W.D. Leon-Salas, S. Velipasalar, N. Schemm, and S. Balkir, “A low-cost, tiled embedded smart camera system for computer vision applications,” *Proc. of ACM/IEEE Int’l Conf. on Distributed Smart Cameras*, vol. 29, no. 3, pp. 125–131, 2007.

[130] W.D. Leon-Salas, S. Balkir, K. Sayood, N. Schemm, and M.W. Hoffman, “A low-complexity circuit for on-sensor concurrent A/D conversion and compression,” *IEEE Sensors Journal*, vol. 7, no. 9, pp. 1317–1325, 2007.

[131] W.D. Leon-Salas, “Focal plane video compression: A dissertation,” *ETD collection for University of Nebraska - Lincoln*, 2006.

[132] L. D. McIlrath, “A CCD/CMOS focal-plane array edge detection processor implementing the multiscale veto algorithm,” *IEEE Journal of Solid-State Circuits*, vol. 31, no. 9, pp. 1239–1247, 1996.

[133] M. Tabet and R. Hornsey, “CMOS image sensor camera with focal plane edge detection,” *Proc. of Canadian Conf. on Electrical and Computer Engineering*, vol. 2, pp. 1129–1133, 2001.

[134] Atmel Corporation, “Atmel AT91SAM datasheet,” `http://www.atmel.com/dyn/resources/prod_documents/6175s.pdf`.

[135] M. Schanz, W. Brockherde, R. Hauschild, B. J. Hosticka, and M. Schwarz, "Smart CMOS image sensor arrays," *IEEE Trans. on Electron Devices*, vol. 44, no. 10, pp. 1699–1705, 1997.

[136] R. Forchheimer, K. Chen, C. Svensson, and A. Odmark, "Single-chip image sensors with a digital processor array," *Journal of VLSI Signal Processing*, vol. 5, pp. 121–131, 1993.

[137] M. G. Johnson, "An input-free VT extractor circuit using a two-transistor differential amplifier," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 6, pp. 704–705, 1993.

[138] X. L. Jin and J. Chen, "Novel principle and realization of simple low-power low-noise correlated double sampling for CMOS pixel readout circuit," *IEEE Proc. of Conf. on Electron Devices and Solid-State Circuits*, pp. 113–116, 2003.

[139] D.G. Lowe, "Object recognition from local scale-invariant features," *Proc. of IEEE Int'l Conf. on Computer Vision*, vol. 2, pp. 1150–1157, 1999.

[140] M.A. Fischler and R.C. Bolles, "Random Sample Consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Comm. of the ACM*, vol. 24, pp. 381–395, 1981.

[141] B. Wilburn, N. Joshi, V. Vaish, E-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy, "High performance imaging using large camera arrays," *ACM Trans. on Graphics*, vol. 24, no. 3, pp. 765–776, 2005.

[142] R. Hartley, and A. Zisserman, "Multiple view geometry in computer vision," *Cambridge University Press*, second edition, 2003.

[143] D. Lowe, "Demo Software: SIFT Keypoint Detector," `http://www.cs.ubc.ca/~lowe/keypoints/`.

[144] M. Brown and D.G. Lowe, "Recognising panoramas," *Proc. of IEEE Int'l Conf. on Computer Vision*, vol. 2, pp. 1218–1225, 2003.