Title: SCALABLE CLUSTERED CAMERA SYSTEM AND METHOD FOR MULTIPLE OBJECT TRACKING

Abstract: Embodiments of the invention are directed to a Scalable Clustered Camera System (SCCS), which is a peer-to-peer multi-camera system for multiple object tracking. Instead of transferring control of tracking jobs from one camera to another, each camera in the presented system performs its own tracking, keeping its own tracks for each target object, which provides fault tolerance. A fast and robust tracking algorithm is described to perform tracking on each camera view, while maintaining consistent labeling. In addition, a novel communication protocol is introduced, which can handle the problems caused by communication delays and different processor loads and speeds, and incorporates variable synchronization capabilities, so as to allow flexibility with accuracy tradeoffs.
Published: without international search report and to be republished upon receipt of that report

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.
SCALABLE CLUSTERED CAMERA SYSTEM AND METHOD FOR MULTIPLE OBJECT TRACKING

CROSS REFERENCE TO RELATED APPLICATION

[001] This application claims the benefit of U.S. Provisional Patent Application No. 60/814,446, the contents of which is hereby incorporated by reference herein.

FIELD OF THE INVENTION

[002] The invention relates to the tracking of objects and, more particularly, the tracking of objects with multiple cameras.

BACKGROUND OF THE INVENTION

[003] Reliable and efficient tracking of objects by multiple cameras is an important and challenging problem which finds wide-ranging application areas. Most existing systems assume that data from multiple cameras is processed on a single processing unit or by a centralized server. However, these approaches are neither scalable nor fault-tolerant. We propose multi-camera algorithms that operate on peer-to-peer computing systems where a different processing unit is used to process each camera, and the processing units communicate with each other directly, eliminating the need for a central server. Peer-to-peer vision systems require co-design of image processing and distributed computing algorithms as well as sophisticated communication protocols, which should be carefully designed and verified to avoid deadlocks and other problems.

SUMMARY OF THE INVENTION

[004] Embodiments of the invention are directed to a Scalable Clustered Camera System (SCCS), which is a peer-to-peer multi-camera system for multiple object tracking. Instead of transferring control of tracking jobs from one camera to another, each camera in the presented system performs its own tracking, keeping its own tracks for each target object,
which provides fault tolerance. A fast and robust tracking algorithm is proposed to perform tracking on each camera view, while maintaining consistent labeling. In addition, a novel communication protocol is introduced, which can handle the problems caused by communication delays and different processor loads and speeds, and incorporates variable synchronization capabilities, so as to allow flexibility with accuracy tradeoffs. This protocol was exhaustively verified by using the SPIN verification tool. The success of the proposed system is demonstrated on different scenarios captured by multiple cameras placed in different setups. Also simulation and verification results for the protocol are presented.

BRIEF DESCRIPTION OF THE DRAWINGS

[005] Embodiments of the invention will be more readily understood from the detailed description of exemplary embodiments presented below considered in conjunction with the attached drawings, of which:

[006] FIG. 1 shows the total number of messages that need to be sent, for T=1,5,10,20, with the server-based scenario and SCCS, respectively; (b) is the rotated version of (a);

[007] FIG. 2 shows corresponding locations of recovered FOV lines;

[008] FIGs. 3-5 show recovered FOV lines for different video sequences and camera setups;

[009] FIGs. 6-7 show and examples of successfully resolving a merge;

[010] FIG. 8 shows communication between two cameras;

[011] FIG. 9 shows camera states at the synchronization point;

[012] FIG. 10 shows message totals needed for different scenarios;

[013] FIG. 11 shows the different number of states reached for verification;

[014] FIG. 12 shows different camera locations;

[015] FIG. 13 shows processing times;
FIG. 14 shows waiting times;

FIG. 15 shows average accuracy;

FIGs. 16-17 show an exemplary camera setups;

FIGs. 18-20 show lost camera examples;

FIGs. 21-22 show resolving;

FIG. 23 shows accuracy results; and

FIG. 24 shows requests for synch rates.

It is to be understood that the attached drawings are for purposes of illustrating the concepts of the invention and may not be to scale.

DETAILED DESCRIPTION OF THE INVENTION

The following references, some of which are referred to in this application, are hereby incorporated by reference herein.


L. INTRODUCTION

Embodiments of the invention are directed to a distributed smart camera system that uses a peer-to-peer communication protocol to transfer data between multiple
cameras. In a *smart camera* system, each camera is attached to a computing component, in this case different CPUs. Distributed camera systems communicating in peer-to-peer fashion can be scaled up to very large systems, unlike server-based designs that are limited by the server capacity.

[0028] Reliable and efficient tracking of objects by multiple cameras is an important and challenging problem which finds wide-ranging application areas such as video surveillance, indexing and compression, gathering statistics from videos, traffic flow monitoring, and smart rooms. Due to the inherent limitations of a single visual sensor, such as limited field of view and delays due to panning and tilting, a collaboration of multiple cameras makes manifest an inevitable trend. Multiple cameras can enhance the capability of vision applications, providing fault-tolerance and robustness for issues such as target occlusion. Examples of multi-camera systems can be found in [1] -[17].

[0029] Yet, using multiple cameras to track multiple objects poses additional challenges. One of these challenges is the *consistent labeling* problem, i.e. establishing correspondences between moving objects in different views. Multi-camera systems, rather than treating each camera individually, compare features and trajectories from different cameras in order to obtain history of the object movements, and handle the loss of the target objects which may be caused by occlusion or errors in the background subtraction (BGS) algorithms.

[0030] Different approaches have been taken to solve the consistent labeling problem. Kelly et al. [9] assume that all cameras are fully calibrated to construct a 3D environment and objects are tracked as a group of voxels, whereas Cai and Aggarwal [4] assume that only the neighboring cameras are calibrated; they switch cameras when one camera loses view of the target, and can handle limited occlusion. Bayesian networks are used by Chang and Gong [5] to do feature matching, and establish correspondences. They use camera calibration
information, landmark modalities and apparent height and color to derive additional constraints. Utsumi et al. [15] employ feature matching as well. Yet, calibrating cameras fully is expensive and impractical to install by the end user as it requires some expert intervention. In addition, relying on feature matching can cause problems as the features can be seen differently by different cameras due to lighting variations.

Lee et al. [12] assume that the intrinsic camera parameters are known and use the centroids of the tracked objects to estimate a homography and to align the scene’s ground plane across multiple views.

Observation intervals and transition times of objects across cameras are used for tracking in [10]. Khan and Shah [11] presented a method which uses field of view (FOV) lines and does not require camera calibration. However, due to the way the lines are recovered, they may not be localized reliably, and if there is dense traffic around the FOV line, the method can result in inconsistent labels. Funiak et al. [19] proposed a distributed calibration algorithm for camera networks.

Although many groups have developed methods to combine data from multiple cameras, much less attention has been paid to the computational efficiency and scalability of these methods. Many existing systems assume that multiple cameras are processed on a single CPU or by a centralized server. However, these are not scalable approaches. Chang and Gong [5] propose a multi-camera system which is implemented on an SGI workstation. For a single CPU system, the amount of processing necessary to track multiple objects on multiple camera views can be excessive for real-time performance. Furthermore, scalability is debilitated as each additional camera imposes greater performance requirements.

In order to increase processing power, and handle multiple video streams, distributed systems have been employed instead of using a single CPU. In a distributed
system, different CPUs are used to process inputs from different cameras. Yet, most existing distributed multi-camera systems use a centralized server/control center [2], [6], [13], [20]. Nguyen et al. [13] propose a system using multiple cameras where all the local processing results are sent to a main controller. Collins et al. [6] introduced the VSAM system, where all resulting object hypotheses from all sensors are transmitted at every frame back to a central operator control unit, which is responsible for integrating symbolic object trajectory information accumulated by each of the processing units together with a 3D geometric site model. Using a central server simplifies some problems, such as video synchronization and communication between the algorithms handling the various cameras. But, server-based multi-camera systems have a bandwidth scaling problem, since the central server can quickly become overloaded with the aggregate sum of requests from an increased number of nodes. In addition, server-based systems are not practical in many realistic environments, and have high installation costs. Besides the algorithm development, hardware design and resource management has also been considered for parallel processing. Watlington and Bove [21] proposed a data-flow model and use a distributed resource manager to support parallelism for media processing.

[0035] The aforementioned problems of server-based systems necessitate the use of peer-to-peer systems, where individual nodes communicate with each other without going through a centralized server. Several important issues need to be addressed when designing peer-to-peer systems. First, there may be significant communication delays between processing nodes. This necessitates the design of tracking algorithms requiring relatively little inter-process communication and a small number of messages between the nodes. This requires a careful design and choice of when to trigger the data transfer, what data to send in what fashion, and to whom to send this data. Another important issue and challenge is maintaining consistency for data across cameras as well as operations upon the data without
use of a centralized server. Also, even if the cameras and input video sequences are synchronized, communication and processing delays pose a serious problem. Depending on the amount of processing each processor has to do, one processor can run faster/slower than the other. Thus, when a processor receives a request, it may be ahead/behind compared to the requester. These issues mandate efficient and sophisticated communication protocols for peer-to-peer systems.

[0036] These protocols find use in real-time systems, which tend to have stringent requirements for proper system functionality. Hence, the protocol design for these systems necessitates transcending typical qualitative analysis using simulation and instead, requires verification. The protocol must be checked to ensure it does not cause unacceptable issues such as deadlocks and process starvation, and has correctness properties such as the system eventually reaching specified operating states.

[0037] Atushi et al. [1] use multiple cameras attached to different PCs connected to a network. They use calibrated cameras and track the objects in world-coordinates, sending message packets between stations. Ellis [7] also uses a network of calibrated cameras. Bramberger et al. [3] introduce a distributed embedded smart camera system consisting of loosely coupled cameras. They use predefined migration regions to handover the tracking process from one camera to the other. But, these methods do not discuss the type and details of communication, and how to address the communication and processing delay issues.

[0038] Verification of communication protocols is a rich topic, particularly for security and cryptographic systems. Karlof et al. [24] analyzed the security properties of two cryptographic protocols and discovered several potential weaknesses in voting systems. Evans and Schneider [25] verified time-dependent authentication properties of security protocols. Vanackere [26] modeled cryptographic protocols as a finite number of processes interacting with a hostile environment and proposed a protocol analyzer TRUST for
verification. Finally, a burgeoning body of work exists pertaining to the formal verification of networked multimedia systems. Bowman et al. [27] described multimedia stream as a timed automata, and verified the satisfaction of quality of service QoS properties including throughput and end-to-end latency. Sun et al. [28] proposed a testing method for verifying QoS functions in distributed multimedia systems where media streams are modeled as a set of timed automata.

[0039] Our previous work [16], [18] introduced some of the tools necessary towards building a peer-to-peer camera system. The work presented in [16] performs multi-camera tracking, and information exchange between cameras. However, it was implemented on a single CPU in a sequential manner, and the tracking algorithm used required more data transfer. This paper introduces SCCS, the Scalable Clustered Camera System, together with its communication protocol and its verification results. SCCS is a scalable peer-to-peer multi-camera system for multi-object tracking. It is a smart camera system wherein each camera is attached to a computing component, in this case different CPUs. In this article, if not specified otherwise, a peer-to-peer smart camera system is referred when the term multi-camera system is used.

[0040] A computationally efficient and robust tracking algorithm is introduced to perform tracking on each camera view, while maintaining consistent labeling. Instead of transferring control of tracking jobs from one camera to another, each camera in SCCS performs its own tracking and keeps its own tracks for each target object, thus providing fault tolerance. Cameras can communicate with each other to resolve partial/complete occlusions, and to maintain consistent labeling. In addition, if the location of an object cannot be determined at some frame reliably due to the errors resulted from BGS, the track of that object is robustly updated from other cameras. If tracking is performed independently instead on different camera views, there are two possibilities when an object in the view of a camera,
$C^j$, is completely occluded by a static background item for a long time: 1) the current location can be estimated from the speed and direction of the previous movement, but this requires several assumptions, such as steady movement, and no direction change; 2) this object is lost in $C^j$ but is continued to be tracked in the other views where it is visible. However, we do not want to lose the object in any of the views since we cannot guarantee that the target will always be visible in multiple cameras. Thus, it is always more advantageous to keep all the tracks updated in all the views. SCCS updates trajectories without interruption and without any need for an estimation of the moving speed and direction, even if the object is totally invisible to that camera. Our tracking algorithm deals with the merge/split cases on a single camera view without sending requests to other nodes in the system. Thus, it provides coarse object localization with sparse message traffic.

[0041] In addition, a novel communication protocol is introduced, which coordinates multiple tracking components across the distributed system, and handles the processing and communication delay issues. The decisions about when and with whom to communicate are made such that the frequency and size of transmitted messages are kept small. This protocol incorporates variable synchronization capabilities, so as to allow flexibility with accuracy tradeoffs. Non-blocking sends and receives are used for message communication, since for each camera it is not possible to predict when and how many messages will be received from other cameras. Moreover, the type of data that is transferred between the nodes can be changed, depending on the application and what is available, and our protocol remains valid and can still be employed. For instance, when full calibration of all the cameras is tolerated, the 3D world coordinates of the objects can be transferred between the nodes. We verified this communication protocol with success by using the SPIN verification tool.

[0042] We present experimental results which demonstrate the success of the proposed peer-to-peer multi-camera tracking system, with a minimum accuracy of 94.2% and
90% for new_label and lost_label cases, respectively, with a high frequency of synchronization. We also present the results obtained after exhaustively verifying the presented communication protocol with different communication scenarios.

[0043] The rest of the paper is organized as follows: Section II compares SCCS with a server-based scenario. Section III describes the computer vision algorithms in general. More specifically, recovery of field of view (FOV) lines is described in Section III-A.1, and the tracking algorithm is introduced in Section III-B. The communication protocol is explained in Section IV, and its verification and obtained results are described in Section V. Section VI presents the experimental results obtained with several different video sequences with varying difficulty, and Section VII concludes the paper.

[0044] II. A SERVER-BASED SYSTEM SCENARIO VERSUS SCCS

[0045] As stated before, server-based multi-camera systems have a bandwidth scaling problem, and are limited by the server capacity. In order to illustrate the excessive number of messages and the load a server needs to handle, and compare these to SCCS, we will introduce a server-based system scenario in this section. In this server-based system, the nodes keep the server updated by sending it messages for each tracker in their FOV. To make a fair comparison between this scenario and the communication protocol used by SCCS, we assume that these messages are sent at the synchronization points, which will be defined in Section IV-D. Due to different processing rates of the distinct processors coupled with communication delays, a server keeps the received data buffered to provide consistent data transfer between the nodes. However, this is not a practical approach since the buffer size may need to be very large. Thus, we designed this scenario so that the server does not need a buffer. The nodes are required to wait at each synchronization point until they receive an overall done message from the server. At this point, each node needs to send a message for each tracker. These messages also indicate if the node has a request from any of the other
nodes or not. Then, the server handles all these messages, determines the replies for each request, if there were any, and sends the replies to the corresponding nodes. The nodes update their trackers after receiving the replies, and acknowledge the server that they are done. After receiving a done message from all the nodes, the server sends an overall done message to the nodes so that nodes can move on. Based on this scenario, the total number of messages that go through the server can be determined by using:

$$M_{server} = S \times 2 \times N + \sum_{i=1}^{N} E_i + S \times \sum_{i=1}^{N} T_i$$  \hspace{1cm} (1)$$

where $S$ is the number of synchronization points, $N$ is the number of nodes/cameras and $E_i$ is the total number of events that will trigger requests in the view of camera $C_i$. $T_i$ is the total number of trackers in the view of $C_i$, and in this formula, without loss of generality, it is assumed that, for camera $C_i$, $T_i$ remains the same during the video. Whereas, for SCCS this number is equal to:

$$M_{SCCS} = S \times (2 \times N - 1) + 2 \times (N - 1) \times \sum_{i=1}^{N} E_i$$  \hspace{1cm} (2)$$

[0046] The more detailed discussion of how (2) is obtained will be given in Section IV-D. As seen from (2), the total number of messages sent around by SCCS is independent of the number of trackers in each camera view, since the communication is done in a peer-to-peer manner. This fact can also be seen in Fig. 1(a) and 1(c). These figures were obtained by setting $E_i=20$ and $T_i = T, \forall i$, where $T \in \{1, 5, 10, 20\}$. In addition, equations (1) and (2), and Fig. 1 show that the server-based system does not scale well.

[0047] III. MULTI-CAMERA MULTI-OBJECT TRACKING
A. Field of View (FOV) Lines

FOV lines have been introduced by Khan and Shah [11] to solve the consistent labeling problem. They show that when FOV lines are recovered, the consistent labeling problem can be solved successfully. The 3D FOV lines of camera $C^i$ are denoted by $L^i_s$ [11], where $s \in \{r, l, t, b\}$ correspond to one of the sides of the image plane. The projections of the 3D FOV lines of camera $C^i$ onto the image plane of camera $C^j$ will result in 2D lines denoted by $L^i_s^j$, and called the FOV lines.

I) Recovery of Field of View Lines: The FOV lines are recovered by observing moving objects in different views and using entry/exit events by Khan and Shah [11]. However, there needs to be enough traffic across a particular FOV line to be able to recover it. In addition, the method in [11] relies on the performance of the BGS algorithm. Depending on the size of the objects, they may not be detected instantly or entirely, which affects the location of the FOV lines. Since FOV lines will play an important role in consistent labeling and in our communication protocol later, it is necessary to recover all of them in a robust way. Moreover, the precision in locating the FOV lines is very important, especially in crowded scenes, for consistent labeling.

In this application, embodiments are described that are directed to a robust and reliable way of recovering the FOV lines, which does not rely on the object movement in the scene or the performance of the BGS algorithm. This way, all visible FOV lines in a view can be recovered at the beginning, even if there is no traffic at the corresponding region. In addition, there is no need to know the intrinsic or extrinsic camera parameters. As in [11], it is assumed that the scene ground is planar, and a homography is estimated to recover the FOV lines.
With reference to Figure 2, \( p_{h_1}^{(i)} \) and \( p_{h_2}^{(i)} \) are the corresponding locations of \( p_{h_1}^{(i)} \) and \( p_{h_2}^{(i)} \), respectively, and the recovered FOV line passing through \( p_{h_1}^{(i)} \) and \( p_{h_2}^{(i)} \) is shown with a dashed line in the view of \( C^j \).

The inputs to the proposed system are four pairs of corresponding points (chosen off-line on the ground plane) in two different camera views.

These points in the views of \( C^i \) and \( C^j \) are denoted by \( P^{(i)} = \{ p_1^{(i)}, ..., p_s^{(i)} \} \) and \( P^{(j)} = \{ p_1^{(j)}, ..., p_s^{(j)} \} \) respectively (See Fig. 2). Let \( \overrightarrow{p}_k^{(i)} = (x_k^{(i)}, y_k^{(i)}, 1)^T \) (\( k \in \{1, ..., 4\} \)) denote the homogeneous coordinates of the input point \( p_k^{(i)} = (x_k^{(i)}, y_k^{(i)}) \). The, a homograph (H) is estimated from \( \{ p_1^{(i)}, ..., p_s^{(i)} \} \) and \( P^{(j)} = \{ p_1^{(j)}, ..., p_s^{(j)} \} \) by using a Direct Linear Transform (DLT), as described by Hartley and Zisserman [30].

The image of the camera view whose FOV lines will be recovered on the other view, is called the field image. After the homography is estimated, the system finds two points on one of the boundaries of the fieldimage, so that each of them is in general position with the four input points. Then it checks with the user that these boundary points are coplanar with the four input points. Let’s denote the two points found on the image boundary \( s \) of the camera \( C^j \) by \( \bar{p}_{s_1}^{(i)} = (x_{s_1}^{(i)}, y_{s_1}^{(i)}) \) and \( \bar{p}_{s_2}^{(i)} = (x_{s_2}^{(i)}, y_{s_2}^{(i)}) \) where \( s \in \{r, l, t, b\} \) correspond to one of the sides of the image plane (See Fig. 2). The corresponding locations of \( (x_{s_1}^{(i)}, y_{s_1}^{(i)}) \) and \( (x_{s_2}^{(i)}, y_{s_2}^{(i)}) \) on the view of camera \( C^j \) are denoted by \( \bar{p}_{s_1}^{(j)} = (x_{s_1}^{(j)}, y_{s_1}^{(j)}) \) and \( \bar{p}_{s_2}^{(j)} = (x_{s_2}^{(j)}, y_{s_2}^{(j)}) \), and computed by using:
\[ H^{(2)} = H^{(1)} \]

where \( n \in \{1,2\} \), \( \vec{p}^{(i)}_{s_n} = (x_n^{(i)}, y_n^{(i)}, 1)^T \) and \( \vec{p}^{(j)}_{s_n} \) denotes the homogeneous coordinates of \( \vec{p}^{(j)}_{s_n} = (x_n^{(j)}, y_n^{(j)}) \). \( x_n^{(j)} \) and \( y_n^{(j)} \) are obtained by normalizing \( \vec{p}^{(j)}_{s_n} \) so that its third entry is equal to 1.

[0057] Once \( p^{(j)}_{s_{(i)}} \) and \( p^{(j)}_{s_{(2)}} \) are obtained on the other view, the line going through these points defines the FOV line corresponding to the image boundary \( s \) of the camera \( C^j \) (See Fig. 2). Two points are found on each boundary of interest and the FOV line corresponding to that boundary is recovered similarly.

[0058] Figures 3 to 5 show the recovered FOV lines for different video sequences and camera setups. Although there was no traffic along the right boundary of Fig. 3b, the FOV line corresponding to it is successfully recovered as shown in Fig. 3a.

[0059] Fig. 3. (a) -(b) and (c) -(d) show the recovered FOV lines for two different camera setups. The shaded regions are outside the FOV of the other camera.

[0060] Fig. 4. (a), (b) and (c) show the recovered FOV lines. The shaded regions are outside the FOV of the other cameras.

[0061] Fig. 5. (a),(b) and (c) show the recovered FOV lines. The shaded regions are outside the FOV of the other cameras.

[0062] Another way of recovering the FOV lines is using the projective invariants in \( P^2 \) [18]. On the projective
\[ J_a = \begin{vmatrix} M_{1a}^{(1)} & M_{2a}^{(1)} & M_{3a}^{(1)} \\ M_{1a}^{(2)} & M_{2a}^{(2)} & M_{3a}^{(2)} \end{vmatrix} = \begin{vmatrix} M_{1a}^{(3)} & M_{2a}^{(3)} & M_{3a}^{(3)} \\ M_{1a}^{(4)} & M_{2a}^{(4)} & M_{3a}^{(4)} \end{vmatrix} = \begin{vmatrix} M_{1a}^{(5)} & M_{2a}^{(5)} & M_{3a}^{(5)} \end{vmatrix}, \tag{4} \]

\[ J_b = \begin{vmatrix} M_{1b}^{(1)} & M_{2b}^{(1)} & M_{3b}^{(1)} \\ M_{1b}^{(2)} & M_{2b}^{(2)} & M_{3b}^{(2)} \end{vmatrix} = \begin{vmatrix} M_{1b}^{(3)} & M_{2b}^{(3)} & M_{3b}^{(3)} \\ M_{1b}^{(4)} & M_{2b}^{(4)} & M_{3b}^{(4)} \end{vmatrix} = \begin{vmatrix} M_{1b}^{(5)} & M_{2b}^{(5)} & M_{3b}^{(5)} \end{vmatrix}, \tag{5} \]

[0063] Where \( M_{abc}^{(i)} \) \( \{a, b, c\} \in \{1, \ldots, 5\} \), denotes the determinant of the matrix \( M_{abc}^{(i)} \) for image \( I \), whose columns are the homogeneous coordinates of the points \( p_a^{(i)} \), \( p_b^{(i)} \) and \( p_c^{(i)} \).

[0064] With five coplanar points, one (\( p_{s_i}^{(i)} \)) on the boundary \( s \), and four that are input to the system (\( \{ p_1^{(i)}, \ldots, p_4^{(i)} \} \)), two independent \( P^2 \) invariants can be calculated by using equations (4) and (5). Then the points in \( \{ p_1^{(i)}, \ldots, p_4^{(i)} \} \) are inserted in (4) and (5), and these equations are rewritten to solve for the corresponding point of \( p_{s_i}^{(i)} \) on the view of \( C_i^j \). The same steps are repeated for the boundary point \( p_{s_j}^{(i)} \), and the line passing through the calculated corresponding points on the view of \( C_i^j \) is recovered as the FOV line corresponding to the boundary \( s \) of the view of camera \( C^j \).

[0065] 2) Checking Object Visibility: As stated previously, \( L_i^j \) denotes the projection of the 3D FOV line \( L_i^{s} \) onto the view of \( C_i^j \) and is represented by the equation of the line, which is written as: \( y = Sx+C \). Henceforth, a point \( p_m^{(i)} = (x_m^{(i)}, y_m^{(i)}) \) will be considered on the visible side of \( L_i^{s} \), if \( \text{sign}(y_m^{(i)} - Sx_m^{(i)} - C) \), where \( (x_a^{(i)}, y_a^{(i)}) \) are the coordinates of the \( p_a^{(i)} \), which is any one of the input points in \( P_i^{(i)} \).

[0066] When an object \( O_j^{(i)} \) enters the view of \( C_i^j \), BGS is applied first and a boundary box around the foreground object is obtained. Then, its visibility in the view of \( C_i^j \) is checked by employing \( L_i^{s} \). The midpoint (\( p_m^{(i)} \)) of the bottom line of the bounding box of the object...
is used as its location. If this point lies on the visible side of $L_j^s\alpha$ for all $s \in \{r,l,t,b\}$, then it is deduced that $O^{(j)}$ is visible by $C^i$ (See Fig. 2).

B. The tracking algorithm: Coarse object localization with sparse message traffic

Our proposed tracking algorithm allows for sparse message traffic by handling merge and split cases within a single camera view without sending request messages to other cameras.

First, foreground objects are segmented from the background in each camera view by using the BGS algorithm presented by Stauffer and Grimson [23], which employs adaptive background mixture models to model the background and to segment the foreground objects. Then, connected component analysis is performed, which results in foreground blobs. When a new foreground blob is detected within the camera view, a new tracker is created, and a mask for the tracker is built where the foreground pixels from this blob and background pixels are set to be 1 and 0 respectively. The box surrounding the foreground pixels of the mask is called the bounding box. Then, the color histogram of the blob is learned from the input image, and is saved as the model histogram of the tracker.

At each frame, the trackers are matched to detected foreground blobs by using a computationally efficient blob tracker which uses a matching criteria based on the boundary box intersection and the Bhattacharya coefficient $\rho(y)[22]$ defined by

$$
\rho(y) \equiv \rho[p(y), q] = \int \sqrt[p(y)]{q_{zd}z} dz. \tag{6}
$$

In (6), $z$ is the feature representing the color of the target model and is assumed to have a density function $q_z$ while $p_z(y)$ represents the color distribution of the
candidate foreground blob centered at location \( y \). The Bhattacharya coefficient is derived from the sample data by using:

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

(7)

where \( \hat{q} = \{ \hat{q}_u \}_{u=1}^{m} \) and \( \hat{p} = \{ \hat{p}_u \}_{u=1}^{m} \) are the discrete densities estimated from the \( m \)-bin histogram of the model and the candidate blobs respectively. These densities are estimated by using the color information at the nonzero pixel locations of the masks. If the bounding box of a foreground blob intersects with that of the current model mask of the tracker, the Bhattacharya coefficient between the model histogram of the tracker and the color histogram of the foreground blob is calculated by using (6). The tracker is assigned to the foreground blob which results in the highest Bhattacharya coefficient, and the mask, and thus the bounding box, of the tracker are updated. The Bhattacharya coefficient with which the tracker is matched to its object is called the \textit{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 color histogram of the foreground blob to which it is matched.

Based on this matching criteria, when objects merge, multiple trackers are matched to one foreground blob, and the labels of all matched trackers are displayed on this blob, as shown in Figures 6, 7, 21 and 22. The masks of the trackers are then updated in the previously discussed fashion. The trackers that are matched to the same foreground blob are put into a \textit{merge state}, and in this state their model histogram is \textit{not} updated. When objects split from each other, trackers are matched to their targets based on the boundary box intersection and Bhattacharya coefficient criteria mentioned above.
With reference to Fig. 6, an example of successfully resolving a merge is shown: (a)(b)(c) and (a')(b')(c') show the original images, and the tracked objects with their labels, respectively.

There may be rare but unfavorable cases where a foreground object, appearing after the split of merged objects, may not be matched to its tracker. We deal with these cases as follows: denote two trackers by $T_1$ and $T_2$, and their target objects by $O_1$ and $O_2$ respectively. When these objects merge, $O_{1\cup2}$ is formed, and $T_1$ and $T_2$ are both matched to $O_{1\cup2}$. After $O_1$ and $O_2$ split, $B_{T_iO_j}$ are calculated, where $\{i, j\}\in\{1, 2\}$, and $B_{T_iO_j}$ denotes the Bhattacharyya coefficient calculated between the histograms of $T_i$ and $O_j$. Based on $B_{T_iO_j}$, both $T_1$ and $T_2$ can still be matched to $O_2$, for instance, and stay in the \textit{merge state}. Denote the similarity coefficient of $T_i$ by $S_{T_i}$. Thus, in this case, $S_{T_1} = B_{12}$ and $S_{T_2} = B_{22}$. This can happen because the model distributions of the trackers are not updated during the \textit{merge state}, and there may be changes in the color of $O_1$ during and after the merge. Another reason may be $O_1$ and $O_2$ having similar colors from the outset. When this occurs, $O_1$ is compared against the trackers which are in the \textit{merge state}, and intersect with the bounding box of $O_1$. That is, it is compared against $T_1$ and $T_2$, and $B_{T_iO_1}$ and $B_{T_jO_1}$ are calculated. Then, $O_1$ is assigned to the tracker $T_{i\ast}$, where:

$$i^\ast = \arg\min_{i\in\{1, 2\}} (S_{T_i} - B_{T_iO_1}).$$ \hfill (8)

If a foreground blob cannot be matched to any of the trackers, and if there are trackers in the \textit{merge state}, the unmatched object is compared against those trackers by using the logic in (8), which is also applicable if there are more than two trackers in the \textit{merge state} as shown in Figures 7 and 21.
An example of resolving the merge of multiple objects is shown in FIG. 7.

As stated previously, this algorithm provides coarser object localization and decreases the message traffic by not sending a request message each time a merge or split occurs. If the exact location of an object in the blob formed after the merge is required, we propose another algorithm that can be used at the expense of more message traffic: When a tracker is in the \textit{merge state}, other nodes that can see its most recent location can be determined as described in III-A.2, and a request message can be sent to these nodes to retrieve the location of the tracker in the \textit{merge state}. If the current location of the tracker is not visible by any of the other cameras, then the mean-shift tracking [22] can be activated. The mean-shift tracking algorithm aims to minimize the distance between a given target distribution and the candidate distribution in the current frame. The similarity between two distributions is expressed as a metric based on the Bhattacharya coefficient. Given the distribution of the target model and the estimated location of the target in the previous frame, an optimization is performed to find a new location and increase the value of the Bhattacharya coefficient.

The mean-shift tracking is error-prone since it can be distracted by the background. It is also computationally more expensive. Thus, when a tracker is in the \textit{merge state}, it is preferable to send messages to other nodes, and request the location of this tracker, if its most recent location is in their FOV. Thus, this algorithm requires additional message traffic. We proposed this second algorithm as an alternative if the exact location of a tracker in the \textit{merge state} is required. The experiments presented in Section VI, were performed by using the first proposed algorithm as the tracking component of the SCCS.

IV. INTER-CAMERA COMMUNICATION PROTOCOL

One issue that needs to be addressed when using peer-to-peer systems is that communication is expensive and there may be significant communication delays between
processing nodes. Also, the number of messages that are sent between the nodes should be
decreased to save power and increase speed. Another issue is maintaining consistency for
data across cameras as well as operations upon the data without use of a centralized server. In
addition, even if the cameras and input video sequences are synchronized, communication
and processing delays pose a serious problem. The processors will have different amounts of
processing to do, and may also run at different processing rates. This, coupled with potential
network delays, causes one processor to be ahead of/behind the others during execution. Thus
when a processor receives a request, it may be ahead/behind compared to the requestor.
Hence, system synchronization becomes very important to ensure the transfer of coherent
vision data between cameras. These aforementioned issues mandate an efficient and
sophisticated communication protocol.

As mentioned before, the SCCS protocol utilizes point-to-point
communication, as opposed to some previous approaches that require a central message
processing server. Our approach offers a latency advantage, and the nodes do not need to
send the state of the trackers to a server at every single frame. This decreases the number of
messages considerably as will be discussed in Section IV-D.1. Moreover, this design is more
scalable, since for a central server implementation, the server quickly becomes overloaded
with the aggregate sum of messages and requests from an increased number of nodes.

In this section, a communication protocol is introduced which can handle
communication and processing delays and hence maintain consistent data transfer across
multiple cameras. This protocol is designed by determining the answers to these questions:

a. When to communicate - determining the events which will require the
transfer of data from other cameras. These events will henceforth be referred to as request
events.
b. With whom to communicate - determining the cameras to which requests should be sent.

c. What to communicate - choosing the data to be transferred between the cameras.

d. How to communicate - designing the manner in which the messages are sent, and determining the points during execution at which data transfers should be made. The protocol is designed so that the number of messages that are sent between the nodes is decreased, and the process synchronization issue is addressed.

In the following, we refer to camera \(C^i\) and \(C^j\) for a requesting and replying camera node, respectively. The block diagram in Fig. 8 illustrates the concepts discussed in this section. It should be noted that, at some point during execution, each camera node can act as the requesting or replying node. The implementation of the proposed system consists of a parallel computing cluster, with communication between the cameras performed by the Message Passing Interface (MPI) library [32]. In this work, the use of MPI is illustrative but not mandatory since it, like other libraries, provides well-defined communication operations including blocking and non-blocking send and receive, broadcast, and gathering. MPI is also well-defined for inter-and intra-group communication and can be utilized to manage large camera groups. We take advantage of the proven usefulness of this library, and treat it as a transparent interface between the camera nodes. With reference to Fig. 8, communication between two cameras is shown.

A. When to communicate - Request Events

A camera will need information from the other cameras when: a) a new object appears in its FOV, or b) a tracker cannot be matched to its target object. These events are called request events, and are referred to as new_label and lost_label events, respectively. If
one of these events occurs within a camera’s FOV, the processor processing that camera needs to communicate with the other processors.

[0091] In the new_label case, when a new object is detected in the current camera view, it is possible that this object was already being tracked by other cameras. If this is the case, the camera will issue a new_label request to those cameras to receive the existing label of this object, and to maintain consistent labeling.

[0092] Camera $C^i$ could also need information from another node when a tracker in $C^i$ cannot be matched to its target object, and this is called the lost_label case. This may occur, for instance, if the target object is occluded in the scene or cannot be detected as a foreground object at some frame due to the failure of the BGS algorithm. In this case, a lost_label request will be sent to the appropriate node to retrieve and update the object location.

[0093] Another scenario where communication between the cameras may become necessary is when trackers are merged and the location of each merged object is required. However, if the exact location of the object is not required, and coarser localization is tolerated, then the tracking algorithm introduced in Section III-B can be used to handle the merge/split within single camera view without sending request messages to the other nodes.

[0094] B. With whom to communicate

[0095] The proposed protocol is designed such that rather than sending requests to every single node in the system, requests are sent to the processors who can provide the answers for them. This is achieved by employing the FOV lines.

[0096] When a request needs to be made for an object $O^{(j)}$ in the view of $C^j$, the visibility of this object by camera $C^j$ is checked using the FOV lines as described in Section III-A.2. If it is deduced that the object is visible by $C^j$, a request message targeted for node $i$ is created and the id of the target processor, which is $i$ in this case, is inserted into this message. Similarly, a list of messages for all the cameras that can see this object is created.
C. What to communicate

The presented protocol sends minimal amounts of data between different nodes. Messages consist of 256-byte packets, with character command tags, integers and floats for track labels and coordinates, respectively, and integers for camera id numbers. Clearly, this is significantly less than the amount of data inherent in transferring streams of video or even image data and features.

Messages that are sent between the processors, processing the camera inputs, are classified into four categories: 1) New label request messages, 2) Lost label request messages, 3) New label reply messages, and 4) Lost label reply messages. As stated, all these messages consist of 256-byte packets.

1) New label request case: If a foreground object viewed by camera $C^i$ cannot be matched to any existing tracker, a new tracker is created for it, all the cameras that can see this object are found by using the FOV lines, and a list of cameras to communicate is formed. A request message is created to be sent to the cameras in this list. The format of this message is:

```
Cmd_tag Target_id Curr_id Side x y Curr_label.
```

In this case, $Cmd_tag$ is a string that holds $NEW_LABEL_REQ$ indicating that this is a request message for the new label case. $Target_id$ and $Curr_id$ are integers. $Target_id$ is the id of the node to which this message is addressed, and $Curr_id$ is the id of the node that processes the input of the camera which needs the label information. For instance, $Curr_id$ is $i$ in this case. These id numbers are assigned to the nodes by MPI at the beginning of the execution. $Side$ is another string which holds information about the side of the image from which the object entered the scene. Thus, it can be right, left, top, bottom, or middle. The next two entities in the message, $x$ and $y$, are doubles representing the coordinates of the location $(p_m^{(i)})$ of the object in the coordinate system of $C^i$. Finally,
\textit{Curr\_label} is an integer holding the temporary label given to this object by $C^i$. The importance and benefit of using this temporary label will be clarified in Sections IV-D and VI.

[00103] 2) \textit{Lost label request case}: When a tracker in $C^i$ cannot be matched to its target object, this is called the \textit{lost\_label} case. For every tracker that cannot find its match in the current frame, the cameras that can see the most recent location of its object are determined by using the FOV lines. Then, a \textit{lost\_label} request message is created to be sent to the appropriate nodes to retrieve the updated object location. The format of a \textit{lost\_label} message is:

[00104] $\text{Cmd\_tag\ Target\_id\ Curr\_id\ Lost\_label\ x\ y.}$

[00105] In this case, $\text{Cmd\_tag}$ is a string that holds \textit{LOST\_LABEL\_REQ} indicating that this is a request message for the \textit{lost\_label} case. $\text{Target\_id}$ and $\text{Curr\_id}$ are integers. $\text{Target\_id}$ is the id of the node to which this message is addressed, and $\text{Curr\_id}$ is the id of the node that processes the input of the camera that needs the location information. $\text{Lost\_label}$ is another integer which holds the label of the tracker which could not be matched to its target object. Finally, $x$ and $y$ are doubles which are the coordinates of the latest location ($p_m^{(i)}$) of the tracker in the coordinate system of $C^i$.

[00106] 3) \textit{New label reply case}: If node $j$ receives a message, and the $\text{Cmd\_tag}$ of this message holds \textit{NEW\_LABEL\_REQ}, then node $j$ needs to send back a reply message. The format of this message is:

[00107] $\text{Cmd\_tag\ Temp\_label\ Answer\_label\ Min\_pnt\_dist.}$

[00108] In this case, $\text{Cmd\_tag}$ is a string that holds \textit{NEW\_LABEL\_REP} indicating that this is a reply message to a \textit{new\_label} request. $\text{Temp\_label}$ and $\text{Answer\_label}$ are integers. $\text{Temp\_label}$ is the temporary label given to a new object by the requesting camera, and $\text{Answer\_label}$ is the label given to the same object by the replying camera. Finally,
Min_pnt_dist is the distance between the corresponding location of the sent point and the current location of the object. As stated in Section IV-C.1, the NEW_LABEL_REQ request message has information about the requester id, side, and object coordinates in the requester’s coordinate system. Let \( p_m^{(i)} = (x,y) \) denote the point sent by node \( i \). When camera node \( j \) receives this message from node \( i \), the corresponding location of \( p_m^{(i)} \) in the view of \( C_j \) is calculated by using (3) as described in Section III-A.1, and this corresponding (j) location is denoted by \( p_m^{(j)} \). If the received Side information is not middle, the FOV line corresponding to this side of the requester camera view is found. Then, the label of the object that has crossed this line, and is closest to the \( p_m^{(j)} \) is sent back as the Answer_label. If, on the other hand, the received Side information is middle, then it means that this object appeared in the middle of the scene, for instance from inside of a building. In this case, the FOV lines cannot be used, and the label of the object that is closest to the \( p_m^{(j)} \) is sent back as the Answer_label. The Min_pnt_dist that is included in the reply message is the distance between \( p_m^{(j)} \) and the location of the object that is closest \( p_m^{(j)} \).

The proposed protocol also handles the case where the labels received from different cameras do not match. In this case, the label is chosen so that Min_pnt_dist is the smallest among all the reply messages. 4) Lost label reply case: If node \( j \) receives a message from node \( i \), and the Cmd_tag of this message holds LOST_LABEL_REQ, then node \( j \) needs to send back a lost_label reply message to node \( i \). The format of this message is:

\[\text{Cmd_tag Lost_label x_reply y_reply.}\]

In this case, \( \text{Cmd_tag} \) is a string that holds LOST_LABEL_REP indicating that this is a reply message to a lost_label request. Lost_label is an integer which is the label of the tracker in \( C_j \) that could not be matched to its target object. When node \( j \) receives a lost_label request, it sends back the coordinates of the current location of the tracker with the
label $Lost_{label}$ as $x_{reply}$ and $y_{reply}$. These coordinates are floats, and they are in the coordinate system of $C_i^j$. When a reply message is received by node $i$, the corresponding point of the received location is calculated on the view of $C_i^j$ as described in Section III-A.1, and the location of the tracker is updated.

D. How to communicate

The steps so far provide an efficient protocol both by reducing the number of times a message must be sent as well as the message size. This part of the protocol addresses the issue of handling the communication and processing delays without using a centralized server. This process will henceforth be called the system synchronization. It should be noted that system synchronization is different from camera or input video synchronization as mentioned above.

The SCCS protocol utilizes non-blocking send and receive primitives for message communication. This effectively allows for a camera node to make its requests, noting the requests it made, and then continuing its processing, with the expectation that the requestee will issue a reply message at some point later in execution. This is in contrast to blocking communication where the execution is blocked until a reply is received for a request. With blocking communication, the potential for parallel processing is reduced, as a camera node may be stuck waiting for its reply, while the processing program will likely require stochastic checks for messages. It is very difficult for each camera to predict when and how many messages will be received from other cameras. In the non-blocking case, checks for messages can take place in a deterministic fashion. Another possible problem with blocking communication is the increased potential for deadlocks. This can be seen by considering the situation where both cameras are making requests at or near simultaneous instances, as neither can process the other node’s request while each waits for a reply.
System synchronization ensures the transfer of coherent vision data between cameras. To the best of our knowledge, existing systems do not discuss how to handle communication and processing delays without using blocking communications. Even if the cameras are synchronized or time-stamp information is available, communication and processing delays pose a problem for peer-to-peer camera systems. For instance, if camera $C^i$ sends a message to camera $C^j$ asking for information, it incurs a communication delay. When camera $C^j$ receives this message, it could be on a frame behind camera $C^j$ depending on the amount of processing its processor has to do, or it can be ahead of $C^j$ due to the communication delay. As a result, the data received may not correspond to the data appropriate to the requesting camera’s time frame. To alleviate this and achieve system synchronization, our protocol provides synchronization points, where all nodes are required to wait until every node has reached the same point. These points are determined based on a synchronization rate which will henceforth be called synch rate. Synchronization points occur every synch rate frames.

Between two synchronization points, each camera focuses on performing its local tracking tasks, saving the requests that it will make at the next synchronization point. When a new object appears in a camera view, a new label request message is created for this object, and the object is assigned a temporary label. Since a camera node does not send the saved requests, and thus cannot receive a reply until the next synchronization point, the new object is tracked with this temporary label until receiving a reply back. Once a reply is received, the label of this object is updated.

Typical units of synchronization rate are time-stamp information for live camera input, or specific frame number for a recorded video. Henceforth, to be consistent, we refer to the number of video frames between each synchronization point when we use the terms synchronization rate or synchronization interval. There is no deterministic
communication pattern for vision systems, so it is expected that the camera processors will frequently have to probe for incoming request messages. Although the penalty of probing is smaller than that of a send or receive operation, it is still necessary to decrease the number of probes because of power constraints. In order to decrease the amount of probing, we make each camera probe only when it finishes its local tasks and reaches a synchronization point.

[00118] With reference to Fig. 9, there are shown camera states at the synchronization point. Fig. 9 shows a diagram of the system synchronization mechanism. This figure illustrates the camera states at the synchronization point. In the first state, the camera finishes its local tracking, and the processor sends out all of its saved requests. Then, the camera enters the second state and begins to probe to see if a done message has been received from the previous camera. If not, this node probes for incoming requests from the other nodes and replies to them while waiting for the replies to its own requests. When the done message is received from the previous camera the camera enters the third state. When all of its own requests are fulfilled, it sends out a done message to the next camera. In the fourth state, each camera node still processes requests from other cameras, and keeps probing for the overall done message. Once it is received, a new cycle starts and the node returns back to the first state.

[00119] The done messages in our protocol are sent by using a ring type of message routing to reduce the number of messages. Thus, each node receives a done message only from its previous neighbor node and passes that message to the next adjacent node when it finishes its own local operations and has received replies to all its requests for that cycle. However, based on the protocol, all the cameras need to make sure that all the others already have finished their tasks before starting the next interval. Thus, a single pass of the done message is insufficient. If we have N cameras ($C^i$, i = 0, ..., N-1), a single pass of the done message will be from $C^0$ to $C^1$, $C^1$ to $C^2$ and so on. In this case, $C^{i-1}$ will not know whether $C^i$
has finished its task since it will only receive done messages from $C_{i-2}$. Thus, a second ring pass or a broadcast of an overall done message will be needed. In the current implementation, the overall done message is broadcasted from the first camera in the ring since the message is the same for every camera.

[00120] In this protocol, the synchronization rate can be set by the end user depending on the system specification. Different synchronization rates are desirable in different system setups. For instance, for densely overlapped cameras, it is necessary to have a shorter synchronization interval because an object can be seen by several cameras at the same time, and each camera may need to communicate with other cameras frequently. On the other hand, for loosely overlapped cameras, the synchronization interval can be longer since the probability for communication is lower and as a result, excess communication due to superfluous synchronization points is eliminated.

[00121] 1) Comparison of the number of messages for a server-based scenario and for SCCS: We will continue the discussion in Section II, and compare the server-based system scenario, introduced in Section II, with SCCS in terms of the total number of messages, and the message loads on the server and the SCCS nodes.

[00122] Based on the SCCS protocol discussed in Section IV, the total number of messages that are sent around by SCCS can be determined by using (2). It should be noted that, when calculating $M_{SCCS}$, this equation considers the worst possible scenario, where it is assumed that all the cameras in the system have overlapping FOVs, and all the events happen in the overlapping region. In this case, in SCCS, N nodes will send $N - 1$ request messages to the other nodes for $E_i$ events and will receive $N - 1$ reply iN messages, hence the $2 \times (N - 1) \times \sum_{i=1}^{N} E_i$ term. At each synchronization point, each node will send a done message to its next
neighbor in the ring, and the first node will send an overall done message to N - 1 nodes, hence the S × (2 × N - 1) term.

[00123] Figures 10(a) and 10(c) show the total number of messages sent in the server-based system and SCCS for T = 3 and T = 15, respectively, where T_i, in (2), is set to be T, ∀i ∈ 1,..., N. As can be seen, even with the worst case assumptions for SCCS, the total number of messages sent is less than that of the server-based system. Another very important point to note is that this is the total number of messages. For the server-based system all of these messages go through the server. Whereas, in SCCS, one ordinary node i has to send only S + 2 × (N - 1) × E_i messages and receive 2 × S + (N - 1) × E_i messages. The node j sending the overall done message has to send 2 × (N - 1) × E_j + S × N messages, and receive (N - 1) × E_j + S messages. These numbers are plotted in Figures 10(b) and 10(d) for T = 3 and T = 15, respectively. As can be seen, the number of messages sent or received by the server is much larger than the number of messages sent or received by any node of the SCCS.

[00124] V. VERIFICATION OF THE PROTOCOL

[00125] Communicating between nodes in a peer-to-peer fashion and eliminating the use of a centralized server decreases the number of messages sent around, and provides scalability and latency advantages. However, this requires a sophisticated communication protocol which finds use in real-time systems having stringent requirements for proper system functionality. Hence, the protocol design for these systems necessitates transcending typical qualitative analysis using simulation; and instead, requires verification. The protocol must be checked to ensure that it does not cause unacceptable issues such as deadlocks and process starvation, and has correctness properties such as the system eventually reaching specified operating states. Formal verification methods of protocols can be derived from
treating the individual nodes of a system as finite state automata. These then emulate communication with each other through the abstraction of a channel.

[00126] *SPIN* is a powerful software tool used for the formal verification of distributed software systems. It can analyze the logical consistency of concurrent systems, specifically of data communication protocols. A system is described in a modeling language called *Promela* (Process Meta Language). Communication via message channels can be defined to be synchronous or asynchronous. Given a *Promela* model, *SPIN* can either perform random simulations of the system’s execution or it can perform exhaustive verification of correctness properties [29]. It goes through all possible system states, enabling designers to discover potential flaws while developing protocols. This tool was used to analyze and verify the communication protocol used in SCCS and described in Section IV.

[00127] To analyze and verify the communication protocol of the SCCS, we first described our system by using *Promela*. We modeled three different scenarios: (a) a 2-processor system with full communication, where full communication means every processor in the system can send requests and replies to each other, (b) a 3-processor system, where the first processor can communicate with the second and third, the second processor can only communicate with the first, and the third one only replies to incoming requests, and (c) a 3-processor system with full communication. The reason of modeling scenario (b) is clarified below.

[00128] After modeling different scenarios, we first performed random simulations. With random simulation, every run may produce a different type of execution. In all the simulations of all three scenarios, all the processors of the model terminated properly. However, each random simulation goes through one possible set of states. Thus, an exhaustive search of the state space is needed to guarantee that the protocol is error-free. We performed exhaustive verification of the three different scenarios with different
synchronization rates. We also inserted an assertion into the model to ensure that a processor starts a new synchronization interval only if every processor in the system has sent a done message at the synchronization point. All of our three scenarios have been verified exhaustively with no errors. Table I shows the results obtained, where the synch rate is 1, and there are 4 synchronization points. (a), (b) and (c) correspond to the scenarios described above. As can be seen in the table, when three processors are used with full communication, the number of states becomes very high compared to other scenarios, thus the search requires more memory. Scenario (b) was modeled so that we can compare scenario (c) to (b), and see the increase in the number of states and memory requirement. The total memory usage in the table is the “total actual memory usage” output of the SPIN verification. This is the amount after the compression performed by SPIN, and includes the memory used for a hash table of states.

<table>
<thead>
<tr>
<th>No. of States</th>
<th>State-vector Size (bytes)</th>
<th>Total Memory Usage (MB)</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Proc. (a)</td>
<td>12259</td>
<td>496</td>
<td>3.017</td>
</tr>
<tr>
<td>3 Proc. (b)</td>
<td>19569</td>
<td>1252</td>
<td>3.217</td>
</tr>
<tr>
<td>3 Proc. (c)</td>
<td>5846880</td>
<td>1252</td>
<td>146.417</td>
</tr>
</tbody>
</table>

Fig. 11 shows the number of states reached with the three scenarios, and with different number of synchronization points. For the 3-processor and full communication scenario, the number of states increases very fast with increasing number of communication points. Since the memory requirement increases with the number of states, the scenario (c) requires the most amount of memory for verification. In addition, when the synch rate is
increased, the number of states increases for the same number of synchronization points, as the requests of the local trackers are saved until the next synchronization.

[00130] Fig. 11. Number of states reached during the verification of different communication scenarios.

[00131] point, and then sent out. These results illustrate that, as is well known in the field of communications, verification of complicated protocols is not a straightforward task. Also, careful modeling of the large systems having many possible states is very important for exhaustive verification.

[00132] VI. EXPERIMENTAL RESULTS

[00133] A. Camera Setups

[00134] We have implemented SCCS on Linux using PC platforms and Ethernet. This section describes the results of experiments on a 3-camera 3-CPU system. Different types of experiments with different camera setups and video sequences of varying difficulty have been performed by using SCCS and the proposed communication protocol.

[00135] Fig. 12 shows the two different camera setups and two types of environment states used for the indoor experiments. We formed different environment states by placing or removing occluding structures, for instance a large box in our case, into the environment. As shown in Figures 12.(a1) and 12.(b1), we placed three cameras in two different configurations in a room. Figures 12.(a1) versus 12.(a2) and 12.(b1) versus 12.(b2) illustrate the two different environment states, i.e. scenes with or without an occluding box. As seen in Figures 12.(a3), 12.(b3) and 12.(b4), three remotely controlled cars/trucks have been used to experiment with various occlusion, merge and split cases. We also captured different video sequences by operating one, two or three cars at a time.

[00136] First, the processing times of a single processor system and a distributed multi-camera system incorporating peer-to-peer communication were compared. Fig. 13
shows the speedup attained using our system relative to a uniprocessor implementation for two cases: processing input from two cameras and from three cameras. In the figure, processing times are normalized with respect to the uniprocessor case processing inputs from three cameras which takes the longest processing time. As can be seen, the uniprocessor approach does not scale very well as processing the input from three cameras takes

Fig. 12. (a1) and (b1) show the locations of the cameras for the first and second camera setups, respectively; (a2) and (b2) show the environment states for the lost label experiments. The photographs of the first and second camera setups are displayed in (a3), (b3) and (b4).

3.57 times as long compared to processing input from two cameras. Whereas, in our case, processing inputs from three cameras by using three CPUs takes only 1.18 times as long compared to processing inputs from two cameras by using two CPUs. Hence, it is demonstrated that the execution time required is maintained, without significant increase, while adding the beneficial functionality of an additional camera. In addition, our approach provides 3.37x and 10.2x speedups for processing inputs from two and three cameras, respectively, compared to a uniprocessor system.

With reference to Fig. 13, there is shown a comparison of the processing times required for processing inputs from two and three cameras by a uniprocessor system and by SCCS. 2Proc-2Cam and 3Proc-3Cam denote the times required by SCCS.

B. Waiting time experiments

In this set of experiments, we measured the average elapsed time between the instance an event occurs and the next synchronization point, where the reply of the request corresponding to this event is received.

Henceforth, this elapsed time will be referred to as waiting time. For instance, if the synch rate is 10, then the synchronization points will be located at frames 1,11,21,...,
281,291,301... and so on. If a new object appears in a camera’s FOV at frame 282, then the waiting time will be 9 frames, as the next synchronization point will be at frame 291.

[00143] Figure 14 shows the average waiting time for experiments performed with different video sequences with different synch rate values. As can be seen, even when the synch rate is 60 frames, the average Synchronization Rate (frames) Synchronization Rate (frames) Synchronization Rate (frames)

[00145] Fig. 14. Waiting times for different videos and environment setups; (a), (b) and (c) show the waiting times for the videos captured with indoor setup 1, indoor setup 2 and for the PETS video, respectively.

[00146] C. Accuracy of the data transfer

[00147] In this set of experiments, we measured the accuracy of the data transfer and data updates. This accuracy is determined by the following formula:

\[
\text{data\_transfer\_accuracy} = \frac{\text{#correct\_updates}}{\text{total\_requests}} \times 100
\] (9)

[00148] where #correct_updates represents the number of times a new_label or lost_label request is correctly fulfilled and the corresponding tracker is correctly updated (its label or its location). The determined accuracy values are shown in Fig. 15. As can be seen, for a synch rate of 1, the system achieves a minimum of 94.2% accuracy for the new label requests/updates on both indoor and outdoor videos. For the lost_label requests, a minimum of 90% accuracy is achieved for both indoor and outdoor videos with a synch rate of 1. Further, even with allowing the processors to operate up to 2 seconds without communication, a minimum of 90% accuracy is still attained for new_label requests with indoor sequences, while 90.9% accuracy is obtained for the outdoor sequence. Again, with allowing the processors to operate up to 2 seconds without communication, a level of 80% or...
higher accuracy is attained for lost_label requests with indoor sequences, while 60% accuracy is obtained for the outdoor sequence.

[00149] Figures 16 and 17 show examples of receiving the label of a new tracker from the other nodes, and updating the label of the tracker in the current view accordingly. For Fig. 16, the synch_rate is 10. As Average Accuracy (%). The average accuracy of the data transfer for indoor [(a)(b)] and outdoor (c) sequences can be seen in Fig. 16.(b1), when the car first appears in the view of camera 2, it is given a temporary label of 52, and is tracked with this label until the next synchronization point. Then, the correct label is received from the other nodes in the system and the label of the tracker in the view of camera 2 is updated to be 51 as seen in Fig. 16.(b3). Fig. 17 is another example for a synch_rate of 60 for the second camera setup. Again, the label of the tracker, created at frame 1468 and given a temporary label of 56, is updated successfully at frame 1501 from the other nodes in the system.

[00150] Figures 18, 19 and 20 show examples of updating the location of a tracker, whose target object is lost, from the other nodes. For Fig. 18, the synch_rate is 5, and the views of the three cameras are as seen in Figures 16(a1), 16(b1) and 16(c1). As seen in Figures 18.(a1) through (a10), the location of the car behind the box is updated every 5 frames from the other nodes, until it reappears. Fig. 19 is another example for a synch_rate of 1 for the second camera setup. The location of the tracker is updated from the other nodes at every frame. Figures 19.(a1) through (a5) show some example images. Fig. 20 shows an example, where the location of people occluded in an outdoor sequence is updated.

[00151] Figures 21 and 22 show examples of SCCS dealing with the merge/split cases on a single camera view for indoor and outdoor videos, respectively. The accuracy of giving the correct labels to objects after they split is displayed in Fig. 23.

[00152] Figure 24 shows the number of new_label and lost_label requests for different synchronization rates for the video captured by the first camera setup with the box placed in
the environment. As expected, with a synch_rate of 1, a lost_label request is sent at each frame as long as the car is occluded behind the box. Thus, the number of lost_label requests is highest for the synch_rate of 1, and decreases with increasing synch_rate.

[00153] VII. CONCLUSIONS

[00154] This paper has presented the Scalable Clustered Camera System, which is a peer-to-peer multi-camera system for multiple object tracking. Each camera is connected a CPU, and individual nodes communicate with each other directly eliminating the need for a centralized server. Instead of transferring control of tracking jobs from one camera to another, each camera in the presented system keeps its own tracks for each target object, which provides fault tolerance. A fast and robust tracking algorithm was proposed to perform tracking on each camera view, while maintaining consistent labeling.

[00155] Peer-to-peer systems require sophisticated communication protocols that can handle communication and processing delays. These protocols need to be evaluated and verified against potential deadlocks, and their correctness properties need to be checked. We introduced a novel communication protocol designed for peer-to-peer vision systems, which can handle the communication and processing delays. The reasons of processing delays include heterogenous processor, different loads at different processors and instruction and task scheduling within the node’s processing unit. The protocol presented in this paper incorporates variable synchronization capabilities. Moreover, compared to server-based systems, decreases the number of messages that a single node has to handle as well as the total number of messages that need to be sent considerably. We then analyzed and exhaustively verified this protocol, without any errors or redundancies, by using the SPIN verification tool.

[00156] Video sequences with varying levels of difficulty have been captured by using different camera setups and environment states. Different experiments were performed to
obtain the speed up provided by SCCS, to measure average data transfer accuracy and average waiting time. Experimental results demonstrate the success of the SCCS, with high data transfer accuracy rates.

[00157] It is to be understood that the exemplary embodiments are merely illustrative of the invention and that many variations of the above-described embodiments may be devised by one skilled in the art without departing from the scope of the invention.
CLAIMS

What is claimed is:

1. A method for tracking one or more objects with a plurality of cameras, the method comprising the steps of:
   finishing, by a first camera, its local tracking;
   sending, by a processor, all of its saved requests;
   probing, by the first camera, to determine if a done message has been received from a second camera;
   if no done message is received, probing for requests from other cameras;
   replying to the requests from the other cameras;
   upon receipt of a done message from a second camera, determining if the first camera's done requests are fulfilled; and
   sending a done message to a third camera.
FIG. 9

FIG. 10
FIG. 11

FIG. 12