Distributed Realtime Interaction and Visualization System

Jan-Friso Evers-Senne, Jan-Michael Frahm, Felix Woelk, Jan Woetzel and Reinhard Koch

Christian-Albrechts-University of Kiel , Institute of Computer Science and Applied Mathematics
Hermann-Rodewald-Straße 3, 24098 Kiel, Germany
Email: {evers,jmf,woelk,jw,rk}@mip.informatik.uni-kiel.de

Abstract

A distributed realtime system for immersive visualization is presented which uses distributed interaction for control. We will focus on user tracking with fixed and pan-tilt-zoom cameras, synchronization of multiple interaction devices and distributed synchronized visualization. The system uses only standard hardware and standard network protocols. Furthermore for the realtime visualization we only use consumer graphics hardware.

1 Introduction

In the last years a lot of research was performed to design human computer interfaces which can be used in a more natural way [2, 6, 7]. Some of these systems have devices attached to the user's body to facilitate interaction. The most intuitive way to control interaction, however, is to exploit natural human actions like walking, pointing or gestures. We will describe a system that captures these natural actions with a multicamera system.

Every human computer interface has different tasks. These tasks are organized as follows:

1. Recording of user interaction with sensor devices like cameras, mouse, keyboard, tracking devices, microphones, etc.
2. Processing of the interaction data and fusion of information.
3. Updating of all affected data and displaying the new state of the application.

In the following we will discuss those tasks. User interaction processing is a problem on it's own and has been investigated in detail in the past but there is still work in progress. The problem of distributed interaction processing is also a current research topic [11, 9, 10].

The paper is organized as follows: At first we will give an overview of the different tasks of the system and how they work together. Then we will review the synchronisation protocol and extend it to incorporate interaction processing. Section 4 describes the interaction processing in detail with the user interface tasks for the user tracking, sensor integration and synchronized scene rendering. We will evaluate the system and conclude with some outlook for future work.

2 System overview

In this section we will give a brief overview of our system architecture and its tasks. The system enables the user to explore a virtual scene interactively with more immersive techniques than keyboard, mouse, headtracker or any other cable connected devices.

We will track the user with three cameras. One fixed camera is located at the ceiling and two pan-tilt-zoom cameras are mounted in front of the user (see fig. 1). The user can interact with a 3D scene that is displayed on a stereo backprojection wall by simply walking throughout the scene. This interaction feels very natural for the user, because there is no special tracking device fixed to the users body. Since the system is not restricted to a single user, an extra task in case of multiple users is to select the user in control.

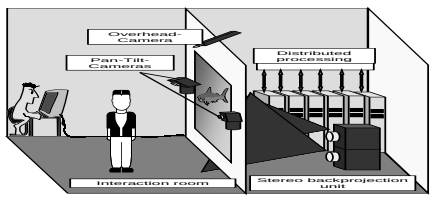


Figure 1: View of interaction area with the display and the cameras.

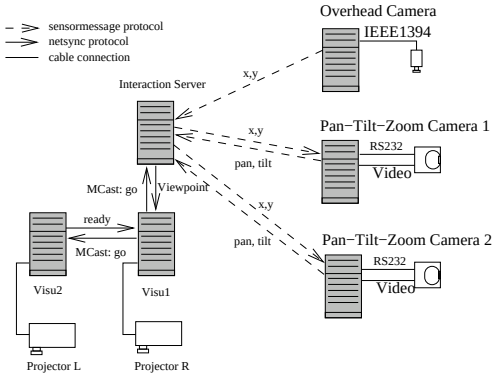


Figure 2: Components of the distributed system and their connections

The system architecture is as follows. The camera at the ceiling (overhead camera) locates the position of the user's feet on the floor. This sensor delivers a 2D position that can be used to initialize and confine the search range of two pan-tilt-zoom cameras facing the user. If the pan-tilt-zoom cameras have found the user's head by searching for skin colored blobs they will track the user's face. From the rotation angles of the pan-tilt cameras we will triangulate the user's 3D head position in space. This position is used to calculate the viewpoint for a virtual view of the scene. The viewpoint position is transmitted to two framewise synchronized visualization nodes for realtime stereo display.

All the modules of the system are computationally expensive and demand realtime requirements of at least 10-15 frames per second for tracking and 30 frames per second for visualisation. We have therefore distributed the computational load to different Linux client nodes. Currently each camera is attached to a separate node with a frame grabber, and the stereo display is splitted onto two nodes with fast OpenGL consumer graphics cards. These machines need a synchronization and intercontrol protocol to meet the requirements of the realtime interaction system and to fuse and distribute the inputs. This is handled by the interaction server. All machines are connected by standard Ethernet network interfaces. Figure 2 sketches the complete system and the connectivity with the interaction server.

3 Distributed synchronized visualization and interaction

In the following subsections we will describe the protocol for framewise visualization synchronization and synchronization of the interaction processing.

3.1 Synchronization for distributed visualization

Distributed rendering of a dynamic scene depends on the synchronization of graphics clients to assure a coherent visualization. For framewise synchronization of the graphics clients we need a common decision that all clients have rendered their views and simultaneously display their views after decision concurrently. To guarantee this framewise synchronization of the displays we use the protocol from [4].
This protocol uses a spanning tree architecture for graphics nodes to distribute a *ready-to-display* message to a specific node (root node) in the network, namely one of the graphics nodes. After receiving *ready-to-display* messages from all other visualization nodes the root node distributes a *display-immediately* message via IP-multicast to all clients (see fig. 3). This protocol is optimal with respect to the number of messages sent and therefore produces few collisions on the network. In [4] it was measured that the generated time shift between different displays is not noticeable by the user.

Furthermore it provides a frame counter to synchronize interaction devices. The protocol is also able to distribute payload information on each synchronization step to the graphics nodes. In the next section we will describe the extension of the protocol used for interaction processing in detail.

3.2 Control of visualization network by interaction

In many virtual environments user interaction has to be processed. The above mentioned protocol [4] for synchronization of graphics clients can be extended to handle such interaction information. We will describe this extension of the protocol in this section.

We assume that every type of interaction processing has one process (interaction server) which is able to transmit interaction processing information

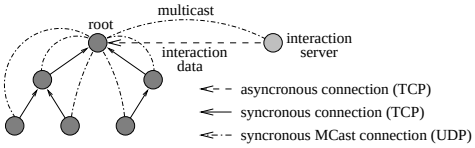


Figure 3: Architecture for visualization and interaction processing

to our visualization network (see fig. 3). This assumption is no constraint for interaction processing because there is only one interaction process, which is the interaction server itself. For distributed interaction processing one process has to collect the relevant interaction information, fuse the information and transmit it to the visualization network.

Tracking of user interaction does not depend on the frame rate of visualization because it can be continuous interaction, for example pointing to an object at the display, or discrete interactions like clicking, selecting, etc. Especially interaction processing does not need to be synchronous to the frame rate of visualization. For this reason we assume interaction processing as an asynchronous task. This assumption leads to a visualization network separated from the architecture for interaction processing. In this case it is possible to use any type of interaction processing algorithm. To correlate interaction events with the displayed context it is necessary to have a globally consistent time for visualization and interaction clients. For time synchronization of the client clocks NTP [8] could be used. For framewise synchronization the frame counter is an appropriate time stamp.

3.3 Interaction synchronization

The interaction server is a specific process not included in the visualization network. It communicates with the root of the visualization network and transmits all interaction data to the root. Furthermore, the interaction server receives the frame counters which are distributed via multicast from the root of the visualization network. These frame counters could be used to correlate the interaction events with the displayed context.

The root of the visualization tree provides interaction information to all graphic clients simultaneously through the IP-Multicast for synchronization. All graphic clients have to process this information in their context. In this way every client can modify its state depending on interaction information. If this processing produces relevant information for other graphic clients it can be submitted as payload with the *ready-to-display*-messages to the root and will be transmitted to all other clients with the next *synchronization*-message. The modified architecture for visualization and interaction is shown in figure 3.

The interaction processing adds a delay to the synchronization. In the worst case we will have a delay of two frames. The aspects of quality-of-service criterion for interaction are discussed in detail by Holloway [5]. If the visualization reaches frame rates of 30 or more frames per second, this interaction delay meets the quality-of-service criterion for most applications.

4 Interaction processing

In this section we will describe the different modules necessary for user interaction. User input is obtained by evaluation of different camera sensors. The acquired sensor data are fused in the interaction server and forwarded to the scene visualization for stereoscopic rendering. The processing is entirely distributed and scalable, hence new sensors can be added if available. A simple GUI is also integrated into the interaction server that lets the user control the scene using mouse and keyboard if no external sensor clients are available, or to modify additional parameters not supported by the external sensors. All techniques described in this section have strong realtime requirements of at least 10 frames per second, which influences the choice of algorithms.

4.1 2D Foot tracking with overhead camera

The overhead camera mounted at the ceiling is equipped with a wide-angular lens. It is tilted to view the whole floor of the interaction area in front of the display. The significant radial lens distortion of the wide-angular lens is compensated with the inverse radial distortion function to avoid errors in the estimated floor position. Since the camera views the planar floor we can use four points on the floor to compute a homography as calibration that relates

ground floor scene coordinates and image coordinates.

The interaction area can be segmented into foreground, which is the user, and background, which is the empty interaction area. To acquire a background reference, a ground truth image is incorporated as mean image of a sequence of the interaction area without the user. The length n of this sequence depends on the noise of the camera. The required length n for a given fixed noise level $\sigma_{background}$ of the background image is given by

$$n = \frac{\sigma_{image}}{\sigma_{background}},$$

with σ_{image} the variance of the ergodic noise process of the camera images. To speed up processing and to reduce image noise we use a mean-filtered subsampled image of size 320x240 pixel.

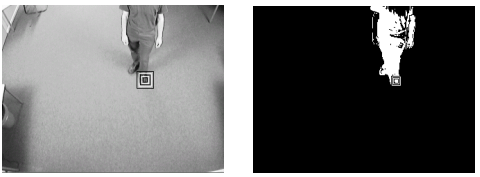
To compute the position of the user in the interaction area we apply two steps:

The first step is based on calculating the difference image between the current camera image and the background reference and uses a threshold ϑ to segment it into background and foreground (see fig. 4). To allow change of the lighting situation an adaptive threshold is used which is taking the following constraints into account:

- The adaptive threshold is limited to a maximum and minimum value as a confidental interval depending on image noise.
- It is assumed that the user's foot in the interaction area covers a ratio $u(x, y)$ of the image which only depends on the position on the floor.
- The threshold should adapt uniformly as the lighting condition changes smoothly and the user is moving slowly with respect to the frame rate of the camera.

The second step locates the user's feet on the floor. We exploit the fact that, due to the tilted viewing frustum of the camera, the feet are always visible in the camera even if the head of the user in the interaction area may not be visible. With respect to the viewing geometry of the cameras, the user's feet are always located on the bottom most part of the foreground. Since the light falls from the display onto the user, the shadows are always cast away from the screen and the camera can easily see the intersection of the feet with the human shadow.

We identify the user position with the bottom most foot position in the segmented camera image. The foot position is found by scanning the segmented image from the bottom right to the top left and search for the first occurrence of a block of the size $u(x, y)$ with at least 80 percent foreground pixels.



(a) *foot position* (b) *segmented image*

Figure 4: Segmentation of the overhead camera into foreground and background

The reliability of this pose estimation depends on the noise in the difference image. To avoid noise in the estimated position we only update the estimated position if the new position has a distance from the last estimated position in pixel greater than a given threshold ω. Normally $\omega = 2$ pixel is a good choice.

It can be assumed that the feet move on a plane, namely the floor, so a homography from the camera to the floor is applied to get the position of the user's foot on the floor in world coordinates. These 2D coordinates are submitted to the interaction server for further processing.

4.2 3D Head tracking with pan-tilt-zoom cameras

For the correct estimation of the users viewpoint it is not sufficient to know the 2D position of the user as given by the overhead camera. Instead we need to know the 3D position of the user's head. The missing unknown parameter is the height of the head above the floor. In the most simple approach this height could be assumed to be fixed. The drawback would be, that movements in vertical direction could not be recognized and therefore would not effect the virtual camera position, which is used for visualisation.

We use two pan-tilt-zoom (ptz) cameras positioned on either side of the screen to determine the user's head position. The position of the cameras can be calibrated with four points by a homography

Figure 5: **Left**: Picture of the user as seen by the pan-tilt-zoom camera. **Right**: The according binary image after segmentation.

between the interaction floor and the camera image plane. In addition, we have to consider the degrees of freedom (pan, tilt, zoom) of the cameras.

In order to obtain information for 3D triangulation two tasks have to be solved:

- The position of the user's head within the image has to be determined.
- If the user's head is not within the image, initialization angles have to be computed.

For the first task, we use a color based algorithm instead of face recognition which utilises a self-adapting bounding box in the HSV color space. This choice is motivated by the fact that proper face recognition is often computationally expensive. The self-adaption form the detected blobs in the previous images makes it feasible to use the color based face detection in our dark room. After segmentation according to the bounding box and employing morphological operations one or several binary blobs are determined from the image (see fig 5). The mean hue and saturation of the biggest blob is calculated and used as the new center of the bounding box in the next processing step. To avoid the bounding box from moving to a complete different color if no appropriate blob is found (e.g. if there is no head in the image at all), some upper bounds for the distance between the new bounding box and the initial user given values are used. This approach assumes a background color which significantly differs from color of human skin. We use dark blue curtains to separate the interaction area. Obviously this is a "single user" algorithm. As soon as more than one user appears in the image, it is no longer determined which user is been tracked. However, just changing the initial values determining the bounding box, any other color (e.g. a specific marker on the *control user*) or all skin coloured blobs can be tracked.

Since the maximal field of view of the two cameras does not allow us to view the whole interaction area at once, an initial guess for the pan and tilt angles is needed. The position of the users feet - as determined by the overhead camera - is used for initialization.

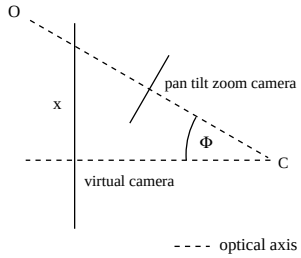


Figure 6: Relationship between the virtual and the real camera: The projection centers of the two cameras coincide with the center of rotation of the pan-tilt-zoom camera in the point C. Any Object O on the optical axis of the real camera projects to the pixel position $(x = \cos\phi, y = \cos\theta)^T$ in the virtual camera, where ϕ and θ are the pan and tilt angles of the real camera. For simplicity only the one dimensional case is shown in this figure.

This leaves us with the following problem: Given a position on the floor of the interaction area, calculate the according pan and tilt angles in order to initially find the users head.

As a solution the concept of the virtual camera is used:

A virtual camera with unit focal length is defined for each pan tilt zoom camera. The center of projection of the virtual camera coincides with the center of projection of the pan-tilt-zoom camera (being the center of rotation at the same time) and its optical axis coincides with the axis of the real camera if the angles are both zero. Thus the pixel on the optical axis of the pan-tilt-zoom camera will project to a location $(\cos\phi, \cos\theta)^T$ in the virtual camera, with ϕ and θ being the pan and tilt angles (see Fig. 6).

Given a position on the floor of the interaction area, the homography is used to determine the pixel position in the virtual camera. The according pan angle is then calculated as the $\arccos$ of the x position of the virtual camera. For the initial tilt angle a fixed mean body height is assumed.

It is not neccessary to find the user's head exactly (in the image center) at first guess. It suffices if the user's head is visible somewhere in the image. The exact pan and tilt angles are computed from the deviation of the user's head position to the

image center. As an additional advantage the pan and tilt angles can be used for triangulation if the user's head is exactly centered in both pan-tilt-zoom camera images. If one camera is not able to recognize the face, it sets an invalid flag. These data are sent to the interaction server that computes the final 3D head position based on the available estimates from all cameras.

4.3 Sensor fusion with interaction server

The interaction server acts as a central communication platform between one or more sensors and the visualization network.

The interaction server is connected to the root node of the visualization network which uses the netsync protocol to control visualization. The interaction server receives multicast messages containing the frame counter of the current render cycle. In addition the interaction server sends interaction data like viewpoint updates asynchronously to the root node.

The interaction server accepts TCP connections from arbitrary sensor clients, each implementing a specialized interaction device. These clients send their data, marked with the frame counter of the time it was acquired, asynchronously to the interaction server. The frame counter of the sensor data is used to maintain data consistency. The fusion of the data from multiple sensors is done in the interaction server to assure a coherent status.

If new data from any sensor client arrives, the current state (in this case position of the user's head) is recalculated based on the known data of other sensor clients and the newly received data.

The sensor clients are also allowed to request data from other sensors. These requests are handled by the interaction server to avoid network traffic between individual sensor clients and again to assure coherence. For example the sensor client controlling the pan-tilt-zoom cameras for face tracking uses this mechanism to initially query the 2D user position from the overhead camera.

The concept of sensor fusion with the interaction server is general. As an example we will describe 3D head position estimation from our multi camera setup. Three cameras are used to observe the interaction area: one overhead camera and two pan-tilt-zoom cameras. Therefore we achieve a more robust pose estimation in the case of one user and we are also able to select the control user in case of multiple users.

For a 3D tracking of the user the overhead camera and one pan-tilt-zoom camera is sufficient. Alternatively we can also use only the two pan-tilt-zoom cameras to estimate the 3D position of the users head. Knowing the position of the two pan-tilt-zoom cameras, the interaction server receives the orientation from each pan-tilt-zoom camera and calculates the intersection of the two rays originating from the cameras. In the general case these two rays do not intersect due to the problems of wide base line stereo position estimation. Therefore the point which has the minimimal distance to each ray is used as users head position.

Now we can use every pair of two cameras to compute additional pose estimations. It is then possible to compute a mean value of the positions as a more robust estimation of the users head position.

If only the overhead camera is available it is possible to track the users position in 2D on the floor. In addition we can extend the foot position received from the overhead camera to a 3D position by assuming a fixed height of the head above the floor.

If both pan-tilt-zoom cameras are tracking and the overhead camera does not work properly, this does not affect the user tracking, as long as there is only one single user in the interaction area.

If there is more than one face visible for the pan-tilt-zoom cameras, it has to be decided which one should be tracked. In this case pan and tilt angles for all faces seen from both pan-tilt-zoom cameras can be used to compute a 3D position for each visible face. The one which is the nearest to the display is chosen to be the *control user*. Using this control user's position, the matching blob in the overhead cameras image is selected and the two redundant positions from overhead camera and one pan-tilt-zoom camera are calculated. Finally, the mean value of all three positions is taken as the user position.

The calculated 3D user position in world coordinates is transformed to a coordinate system appropriate to the visualization subsystem and transmitted to the root node of the visualization subsystem.

4.4 Synchronized 3D scene rendering

The users head position which has been computed by the interaction server is now sent to the stereo

Figure 7: A view of the interaction area with a user wearing polarized glasses. The pan-tilt-zoom cameras can be seen on the left and right side of the display.

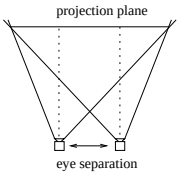


Figure 8: "parallel axis asymmetric frustum perspective projection": The cameras are separated and the frustums are distorted asymmetrically.

visualization subsystem. The visualization is rendered onto a $3m \times 2m$ stereo backprojection system. Two projectors equipped with polarization filters produce stereoscopic images visible with polarized glasses. Each projector is driven by one standard Linux PC equipped with modern consumer 3D accelerator graphics hardware. Figure 7 shows the view of the interaction area with display and cameras.

A method called "parallel axis asymmetric frustum perspective projection" as described in [1] is used to generate the stereo views for the projection system. Assuming the distance between the viewers eyes is approximatly 7 cm, the virtual camera for the left eye is moved 3.5 cm to the left, and the virtual camera for the right eye is moved 3.5 cm to the right. Now both viewing frustums must be distorted to view the same projection plane like shown in figure 8.

The user can explore the scene by walking around the interaction area. These movements are captured by the previously described system which modifies the position and orientation of the virtual camera. Moving forward lets the virtual camera move in the same direction, which lets the user approach the scene. Moving backward, the virtual camera is retracted, so the user sees the scene from an increasing distance. Moving sideways lets the virtual camera move on a circular trajectory in the opposite direction, which is equivalent to rotating the scene on a virtual turntable around an axis in front of the user. This method allows the user to inspect objects using natural movements. To look at the scene more from the right, some steps to the right let the scene rotate clockwise and vice-versa. We don't use the direction where the user looks at to control the viewpoint to enable the user to look around at the displayed scene. The area of exploration is limited to the size of the interaction area, so that additional controls have to be used to navigate in larger scenes.

The two graphics nodes are synchronized as described in section 3.1. In this case the binary synchronization tree is degraded to two connected nodes. The root node listens for a connection from the interaction server. Once connected, the interaction server sends updates of the data describing the virtual camera (position and orientation of the user) to the root node, which distributes this update with the next synchronization cycle. Each of the nodes receives this data and recalculates and resets the parameters of its virtual camera accordingly.

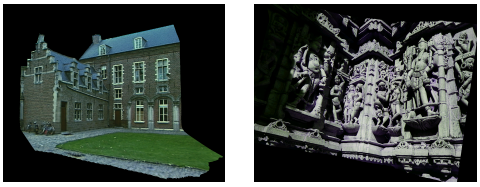
5 Evaluation

We have evaluated the system under a variety of scenarios. We use two Linux PCs with an Athlon 1GHz CPU and GeForce3 graphics engine for the visualization. Each camera is also supported by a PC with an Athlon 1GHz CPU. The pan-tilt-zoom cameras have a frame rate of 25Hz with resolution 360×288 and the overhead camera has a frame rate of 30Hz with resolution 640×480.

The algorithm used to estimate the 2D position from the overhead camera needs 10 ms to estimate the position of the user on the floor. Each pan-tilt-zoom camera needs 35ms to detect the face of the user. To compute and transmit all these data to the visualization network we need approximately 3ms. Therefore the amount of time for these computations is small enough to perform a complete user position estimation at full frame rate of the cameras.

We tested the system with different virtual scene models. The performance for two of these models in our framework will be discussed here. The first model is the *castle* shown in figure 5 (a). It contains 6131 triangles and a 1.3 MB texture. As second

model we choose the *temple* model shown in figure 5(b). This model contains 48893 triangles and a 5.8MB texture.



(a) *castle* model (b) *temple* model

Figure 9: Screen shots of virtual scene models

Hardware accelerated full scene anti aliasing (2x2) was used to enhance the quality of the rendered images. The *temple* model is rendered with 34,3 frames per second, the *castle* is rendered with 97 frames per second. This difference in rendering time is caused by the number of triangles because these consumer graphics engines are optimized for texture mapping but not for geometry. In both cases the interaction to explore the models is very natural and there are no noticeable delays in interaction for the user.

6 Future work

The described system is currently under development, hence there are many tasks for future work.

One problem of the position estimation by the overhead camera is that the estimated foot position switches between position of the right and the left foot if the user is walking. To avoid such uncertainties we will implement a kalman filter which tracks the left and the right foot separately. Then it will be possible to estimate the position on the floor by any linear combination of those positions. Another approach to overcome this problem is the use of a human model.

The color based face tracking of the pan-tilt cameras also has to be improved because still there are problems with skin colored clothes. A refined approach will use also structural information of the human face, like presented in [3].

To extend the navigation in the scene it is required that the user can control the system with gestures to push the degrees of freedom in navigation like long distance movements. We are looking at humanoid models to incorporate gesture navigation by pointing.

References

[1] Paul Bourke. 3D Stereo Rendering Using OpenGL (and GLUT).

[2] T. Darrel, P. Maes, B. Blumenberg, and A. Pentland. A novel enviroment for situated vision and behavior. In *Proc. CVPR-94 Workshop for Visual Behaviors*.

[3] S. Z. Li et. al. Statistical learning of multi-view face detection. In *Proceeding of ECCV 2002*.

[4] J.-M. Frahm, J.-F. Evers-Senne, and R. Koch. Network protocol for interaction and scalable distributed visualization. In *IEEE Proc. of 3D Data Processing Visualization Transmission*, 2002.

[5] Richard Lee Holloway. Registration errors in augmented reality systems. Technical Report TR95-016, 1, 1995.

[6] R. E. Kahn and M.J Swain. Understanding people pointing: The perseus system. In *Proc. IEEE Int. Symp. on Computer Vision 95*.

[7] M. Kolesnik and T. Kuleßa. Detecting, tracking and interpretation of a pointing gesture by an overhead view camera. In B.Radig, editor, *LNCS: Pattern Recognition*, 2001.

[8] Network Time Synchronization Project. NTP. www.ntp.org.

[9] S. C. A. Thomopoulos, R. Viswanathan, and D. K. Bougoulias. Optimal distributed decision fusion. In *IEEE Trans. Aerospace Elect. Syst.*, volume 25, pages 761–765, September 1989.

[10] J. N. Tsistsiklis. Decentralized detection. In *Advances in Statistical Signal Processing, Signal Detection*, volume 2, 1993.

[11] R. Viswanathan and P. K. Varshney. Distributed detection with multiple sensors: Part I - fundamentals. In *Proceedings of the IEEE*, volume 85, pages 54–63, 1997.