International Journal of Computational Intelligence in Control
Vol. 13 no. 3 (December, 2021)

Received: 25th May 2021 *Revised: 20th July 2021* *Accepted: 15th August 2021*

Neighbor Node Determination Algorithm for Efficient Human Tracking

H. Kakiuchi[1], T. Kawamura[2], T. Sasama[2] and K. Sugahara[2] and L. Wilkerson[3]

[1]Melco Power Systems Co., Ltd., 1-1-2 Wadasaki-cho, Hyogo-ku, Kobe 652-8555 Japan
E-mail: Kakiuchi.Hiroto@zs.MitsubishiElectric.co.jp

[2]Tottori University, 4-101, Minami, Koyama-cho, Tottori-shi, Tottori 680-8552 Japan
E-mail: kawamura, sasama, sugahara@ike.tottori-u.ac.jp

[3]Lonnie Wilkerson, Southern Illinois University, Carbondale, IL. 62901, *E-mail: kitnyx@gmail.com*

Abstract: *Much recent research is concerned with overcoming limitations of existing video surveillance systems, particularly for use in automatic human tracking systems. This paper presents an algorithm which determines the position of neighbors in a system of video cameras. By utilizing this deployed position and the view distance of video cameras, this algorithm also determines the interrelationship cameras in such a network must have in an automatic human tracking system. The system is enhanced by a video monitoring system utilizing mobile agent technologies. Mobile agents are suitable for distributed processing and parallel processing, since they can monitor their own behavior and run on distributed computers. Multiple mobile agents in the system can track numerous people using information gathered from several neighboring video cameras at the same time. Modifications to installed cameras usually complicate processing due to changes in the deployed position and view distance; additionally, complications can arise when the deployed position and/or the view distance of video cameras vary due to other circumstances. Therefore, a robust computation not influenced by these circumstances is needed, and this algorithm was developed to solve these concerns.*

Keywords: *adjacency matrix, human tracking, mobile agent, neighbor node.*

1. INTRODUCTION

Video surveillance systems are seeing widespread use in the remote monitoring of people. Principally, the video surveillance system is used as a security system because of its ability to track a particular person. If the function of the video surveillance system is extended to track numerous people, the demands of the system are extended in various ways. Two common examples of such uses are to search for lost children and to gather/analyze consumers' route pattern for marketing research of a retail establishment. Such video surveillance systems are referred to as "automatic human tracking systems" in this paper. Our aim is to show how automatic human tracking systems can be improved by resolving some of the problems of conventional video surveillance system.

Currently existing video surveillance systems have many limitations to their capabilities. In one case, systems have difficulty isolating a number of people located at different position at the same time and track those people automatically. In another, the number of possible targeted people is limited by the extent of users' involvement in manually switching the view from one video camera to another. Although approaches do exist to increase the efficiency of identifying and tracking particular people in a system comprised of numerous surveillance positions, these approaches demand an increase in the workload of the user since it demands users to identify the target.

Some researchers have suggested solutions to the above problems. The first approach was to use an active camera (Terashita, Ukita, & Kidode, 2009; Kawaguchi, Shimada, Arita, & Taniguchi, 2008) to track a person automatically, thus the camera moves in a synchronized motion along with the projected movement of the targeted person. Since a method for correcting blurring image (Yin & Hussain, 2008) is proposed, the active camera is available. This approach is capable of locating and tracking small number of people, but improvements must be made to facilitate the locating and tracking of larger numbers of people. Another common approach was to position the camera efficiently at strategic surveillance locations (Erdem & Sclaroff, 2006). This is not possible in some situations due to the number of cameras that would be necessary for full coverage, and in such cases this approach is not feasible due to limited resources. A

third approach involved the implementation of sensors to efficiently track a target with multiple cameras (Yao *et al.*, 2008a; Yao *et al.*, 2008b). This solution also meets with resource and local restrictions such as installation barriers and the amount of area to be monitored.

A better approach to identify and track numerous targeted people at the same time involves image processing and installation of video cameras at any designated location. However, the concern then becomes the appropriateness of using a single server when locating numerous people, since the image processing increases server load. As such, a new type of system that is capable of more efficiently identifying and locating people must be developed. In this proposed system utilizing mobile agent technologies, the ratio of mobile agents and tracked targets is directly proportional (Lange & Oshima, 1999; Gray, Cybenko, Kotz, Peterson, & Rus, 2002; Motomura, Kawamura, & Sugahara, 2005; Kawamura, Motomura, & Sugahara, 2005). According to many studies, an agent-based approach is appropriate for distributed systems and parallel processing (Cabri, Leonardi, & Zambonelli, 2000; Valetto, Kaiseri, & Gaurav, 2001; Jennings, 2001), since mobile agents can transfer copies of themselves to other servers in the system. By working cooperatively, such a multi-agent system would be effective (Monticolov, Hilaire, Gomes, & Koukam, 2008). With distributed processing, mobile agent technologies are more effective and efficient than conventional video surveillance systems, assuming that a large number of servers with video camera are installed. If one mobile agent can track one person, then multiple mobile agents can track numerous people at the same time, and the server balances the load process of the operating mobile agent on each server with a camera. A video surveillance system enhanced with mobile agent technologies is called "Automatic Human Tracking System" (Kakiuchi, Hamada, Kawamura, Shimizu & Sugahara, 2008; Hamada, Iwasaki, Kakiuchi, Kawamura, & Sugahara, 2008). In such a system, a mobile agent tracks a person captured by a video camera and a server process the data. The video camera and the server are treated as a single entity since the video camera and the server are deployed at the surveillance position. Upon initialization of a person as a target to track, a mobile agent is generated for that particular person. After verifying the features of the person (Ishida *et al.*, 2008), the mobile agent tracks the movement of the person by utilizing the neighbor camera/server location information.

Without knowing a neighbor camera/server location, the mobile agent is not able to track the person, and there will be a need to deploy new duplicate mobile agents on the servers, which is not efficient. Though it is difficult to determine the neighbor server in IP network, it is possible to search the neighbor server by defining the relationship between IP address and physical position of the video camera in the system configuration file. Neighbor camera/servers are called "neighbor camera node/nodes" in this paper. The neighbor camera node location information is determined by the location and view distance of the video camera. The neighbor camera nodes differ by the difference in view distance and view overlap even if video camera's locations are the same. Therefore, it is necessary to update information about camera locations whenever the number of installed cameras is modified or the view distance of camera is changed. The neighbor camera location information is determined utilizing the "neighbor node determination algorithm" in this paper. This algorithm resolves the problems introduced above.

2. AUTOMATIC HUMAN TRACKING SYSTEM

2.1. System Features

The features of the system are shown in Fig. 1. A graphical user interface was developed in order to improve maintainability of the system configuration. The functions of the GUI are to create/edit graphical representation of a building layout, to deploy video cameras on the graphical layout, to monitor mobile agents, and to create data for simulation. The GUI utilizes the algorithm to determine neighbor nodes to compute the adjacency of video cameras, thus information is displayed graphically and the maintainability is improved. The algorithm is utilized not only to calculate the neighboring video cameras, but also to determine the mobile agent's next destination accurately. The algorithm is the subject of this paper; it contributes to the reliability of the system by using minimal computing resources. Bypass methods, which can improve the robustness of the automatic human tracking system, are currently being researched utilizing the algorithm. Since the mobile agents utilize these methods, continuous tracking of the target people can be ensured; thus, the reliability and persistency of the system is improved. Lost target re-detection and acquisition methods are currently being researched utilizing the algorithm, the results of which will improve the robustness of systems as well as further increase reliability and persistency. The goal of these re-detection methods will be to ensure continuous tracking of target people by re-detecting if the mobile agents lose track of the people. OSGi (OSGi Alliance, 2008) and mobile agent technologies are adopted to improve the system scalability. OSGi is a framework developed using the Java Language and server software including web server functions. In the OSGi server, the OSGi framework

manages/controls software which is treated as a single component called a "bundle". A mobile agent server is such a bundle on the OSGi server. In the simulator, an image processor runs on the OSGi server utilizing mock information of a target person. The simulator is tested to improve the reliability and the scalability.

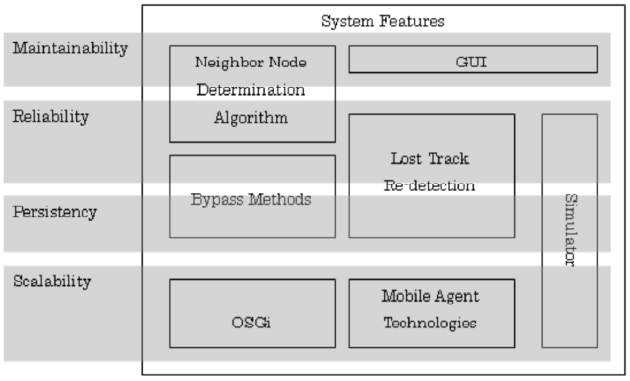

Figure 1: System Features

2.2. System Architecture and Process Flow

The system configuration of the automatic human tracking system is shown in Fig. 2. It is assumed that the system is installed in a given building. Before a person is granted access inside the building, the person's information is registered in the system. Through a camera an image of the person's face and body is captured and registered into the system. Any person who is not registered or not recognized by the system is not allowed to roam inside the building.

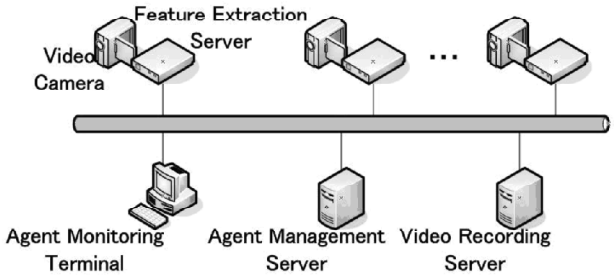

Figure 2: System Configuration

This system is composed of an agent monitoring terminal, agent management server, video recording server and feature extraction server with video camera. The agent monitoring terminal is used for registering the target person's information, retrieving and displaying the information of the initiated mobile agents, and displaying video of the target entity. The agent management server records mobile agents' tracking information history, and provides the information to the agent monitoring terminal. The video recording server records all video images and provides the images to the agent monitoring terminal via request. The feature extraction server along with the video camera analyzes the entity image and extracts the feature information from the image.

A mobile agent tracks a target entity using the feature information and the neighbor nodes information. The number of mobile agents is in direct proportion to the number of the target entities. A mobile agent is initialized at the agent monitoring terminal and launched into the feature extraction server. The mobile agent extracts the features of a captured entity and compares it with the features already stored by the agent. If the features are equivalent, the entity is located by the mobile agent.

The system architecture is shown in Fig. 3. The GUI is operated only on the agent monitoring terminal. The GUI is able to register images of the entities and monitor the status of all the mobile agents. The mobile agent server is executed on the feature extraction server and allows the mobile agents to execute. The Feature extraction function is able to extract features of the captured entities, which is then utilized in the tracking of those entities as mobile agents. OSGi S/W acts as a mediator for the different software, allowing the components to utilize each other. The Agent information manager manages all mobile agent information and provides the information to the agent monitoring terminal. The Video recording S/W records all video, and provides the video movie to agent monitoring terminal. Each PC is equipped with an Intel Pentium IV 2.0 GHz processor and 1 GB memory. The system has an imposed condition requirement that maximum execution time of feature judgment is 1 second and maximum execution time of mobile agent transfer is 200 milliseconds.

The processing flow of the proposed system is shown in Fig. 4. First, a system user selects an entity on the screen of the agent monitoring terminal, and extracts the feature information of the entity to be tracked. Next, the feature information is used to generate a mobile agent per target which is registered into the agent management server.

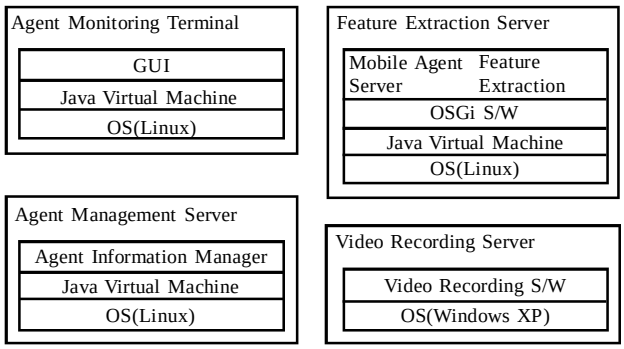

Figure 3: System Architecture

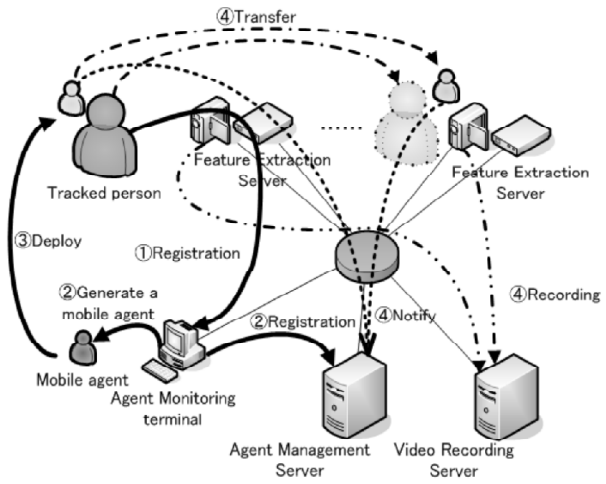

Figure 4: Process Flow

Then the mobile agent is launched from the terminal to the first feature extraction server. When the mobile agent catches the target entity on the feature extraction server, the mobile agent transmits information such as the video camera number, the discovery time, and the mobile agent identifier to the agent management server. Finally the mobile agent deploys a copy of itself to the neighbor feature extraction servers and waits for the person to appear. If the mobile agent identifies the person, the mobile agent notifies the agent management server of the information, removes the original and other copy agents, and deploys the copy of itself to the neighbor feature extraction servers again. Continuous tracking is realized by repeating the above flow.

2.3. Simulator and Graphical User Interface

A simulator is currently being developed in Java Language. The simulator consists of 3 functions, an image processing simulator, an editor for the creation of target simulation routes and a simulation feature data creator. The simulator tools are shown in Fig. 5. The genuine image processing function is also currently being developed with a feature extraction method based on SIFT (Lowe, 2004; Cui, Hasler, Thormaehlen, & Seidel, 2009). Since it is difficult to place many cameras, the image processing simulator is performed on the feature extraction server instead of a genuine image processing function. In addition, this simulator changes a target entity's feature to a walking target entity by using a simulation agent. The simulation agent (Pettre, Simeon, & Laumond, 2002) is also a mobile agent that simulates the movement of a target entity and changes the target entity features as shown in Fig. 6. The movement of the target entity is digitized by the editor of route simulation and the target entity features are digitized by the simulation feature data creator. If the system executes multiple simulation agents, numerous target entities are able to be simulated.

The graphical user interface is shown in Fig. 7. An agent's movement path information is displayed on the left side area of the GUI. A map of running agents and each agent's status are displayed on the center side area. A user sets the configuration, the IP address of the feature extraction server, the accuracy level of feature extraction server, etc on the right side. The GUI can simulate various layouts of the feature extraction server, set the accuracy level of the feature extraction server, and confirm the movement of the agents. Tests of the generator confirmed accuracy of the system with data from 20 cameras. Currently the simulator needs more improvement especially when using more than 20 cameras. This simulator will be adding other necessary functions. And the simulator will also be utilized to verify the performance of the system. As such, the simulator is utilized in the examination.

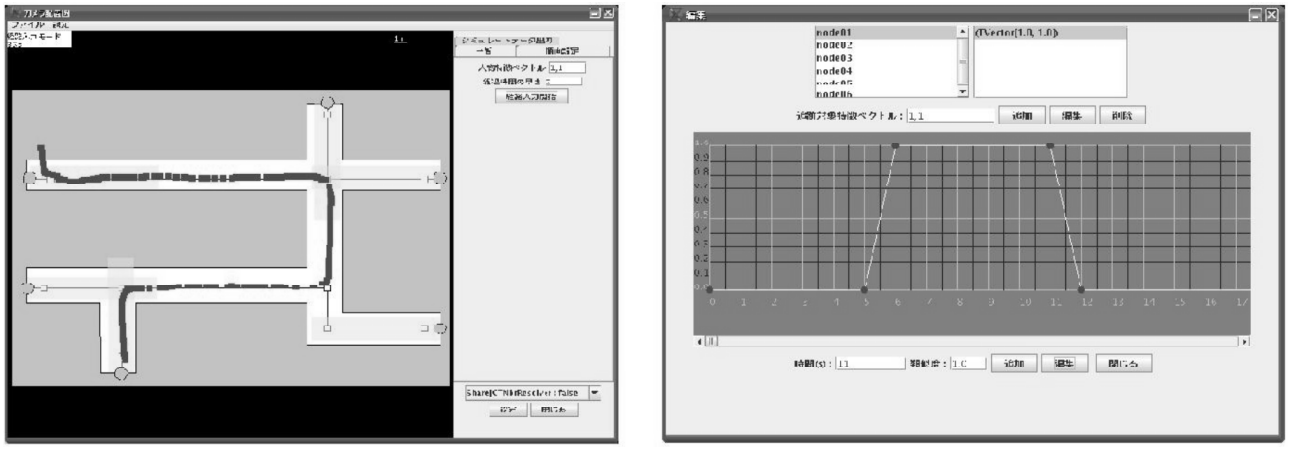

Figure 5: Editor of Simulation Route to Follow and Creator of Simulation Feature Data

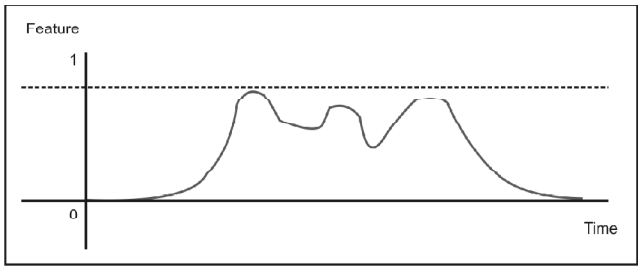

Figure 6: Feature Data

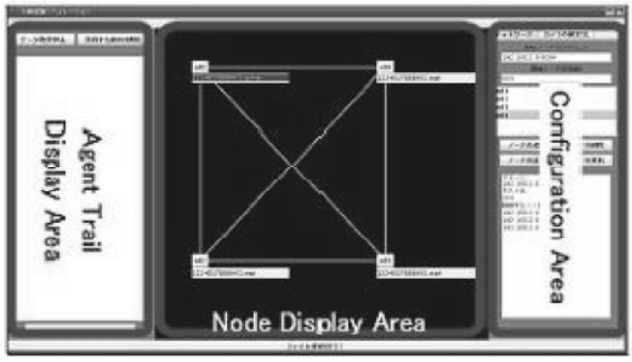

Figure 7: Graphical User Interface

3. NEIGHBOR NODE DETERMINATION ALGORITHM

3.1. Problem with Determining Neighbor Video Cameras

If a mobile agent tracks a target entity, the mobile agent has to know the deployed location of the video cameras in the system. However the abilities of the neighbor cameras are also determined by their view distances. A problem caused by a difference in the view distances can occur. This problem occurs when there is a difference in expected overlap of a view or an interrupt of view.

A scenario in which a neighbor video camera's location is influenced by view distance is shown in Fig. 8. The left side figures of Fig. 8 show 4 diagrams portraying a floor plan with 4 video cameras each, considering the view distances of each video camera are different and assuming that the target entity to be tracked moves from the location of video camera A to video camera D. The right side figures of Fig. 8 show neighbors of each video camera with arrows.

The neighbor of video camera A in object a-1 of Fig. 8 is video camera B but not C and not D as the arrows in object a-2 show. In object a-1 of Fig. 8, video camera C and D are also not considered neighbors of video camera A, because video camera B blocks the view of video camera C and D. And the target entity can be captured at an earlier time on video camera B.

But in the case of object b-1 of Fig. 8, the neighbors of video camera A are video camera B and C but not camera D as the arrows in object b-2 of Fig. 8 show. In the case of object c-1 of Fig. 8, the neighbors of video camera A are all video cameras as the arrows in object c-2 of Fig. 8 show. Thus neighbor video camera's location indicates the difference in view distances of video cameras. The case of object d-1 in Fig. 8 is more complicated.

The neighbors of video camera A in object d-1 of Fig. 8 are video camera B, C, and D as the arrows in object d-2 of Fig. 8 show. And video camera B is not considered the neighbor of video camera C. It is because video camera A exists as a neighbor between video camera B and C. When it is assumed that a target entity moves from A to D, the target entity is sure to be captured by video camera A, B, A, and C in that order.

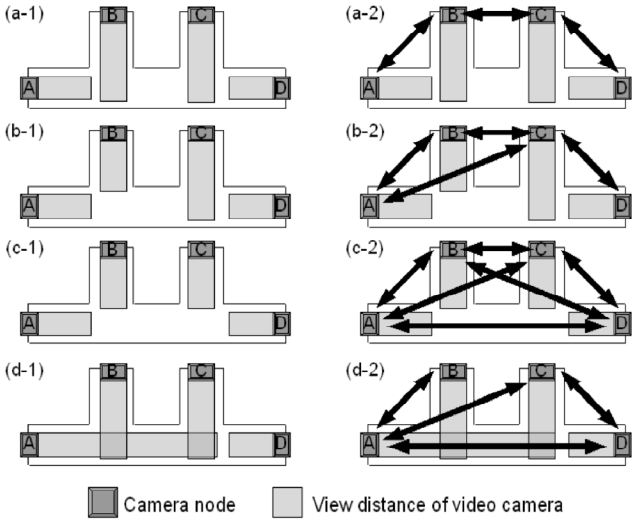

Figure 8: Example that View Distance Influences

This scenario indicates that the definition of "neighbor" cannot be determined clearly because the determination of the neighbor definition is influenced by the change of view distance and it becomes more complicated as the number of video cameras increases.

3.2. Overview of the Algorithm

The developed algorithm can easily determine the neighbor video camera's location without regard to the influence of view distances and any modification of the information of the currently installed cameras. The modification information is set in the system to compute neighbor video cameras on the diagram, which is expressed as a graph. Nodes are used to compute neighbor video camera's information in this algorithm. The nodes are defined as follows:

Camera Node: the location of video camera is labeled as camera node and the nodes are defined as $\mathbf{A} = \{a_1, a_2, \ldots, a_p\}$. This node is also a server with video camera.

Non-camera Node: the nodes are defined as $\mathbf{V} = \{v_1, v_2, \ldots, v_q\}$. The conditions of a non-camera node are stated below:

- Either of crossover, corner, terminal of passage.
- The position where a video camera is installed.
- The end point of the view distance of a video camera.

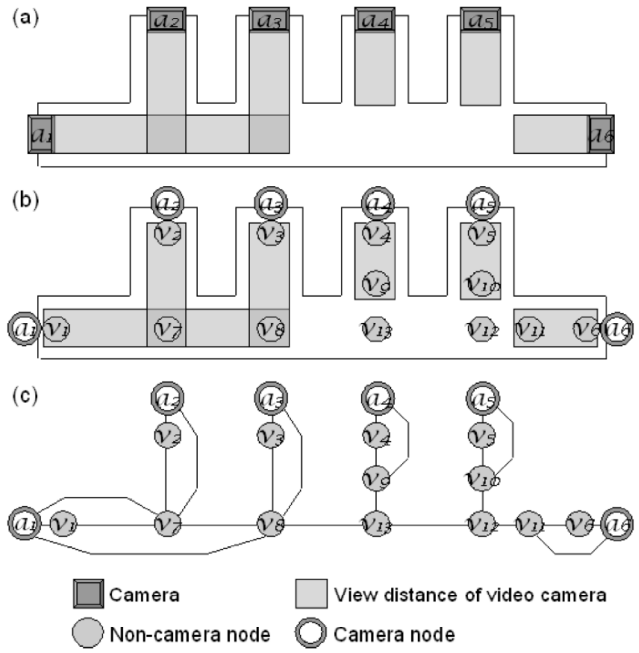

Figure 9: Figure that Sets Non-camera Nodes

In addition, the point where the above conditions are overlapped is treated as one node. When the view distance of the video camera reaches a non-camera node, the non-camera node is defined as the neighbor of the camera node. When two non-camera nodes are next to each other on a course, those nodes are specified as neighbors. Fig. 9 shows an example of these definitions applied and shows the view distances of the video cameras.

The algorithm accomplishes this using an adjacency matrix. Two kinds of adjacency matrix are used by the algorithm. One is an adjacency matrix **X** made from camera nodes' locations as rows and non-camera nodes' locations as columns. Another one is as adjacency matrix **Y** made from non-camera nodes' location as rows and columns. The neighbor information for video cameras is calculated from the connection information of non-camera nodes by using adjacency matrix **X** and **Y**. However, neighbor information can be miscalculated in one particular condition. When the non-camera nodes cross each other and the view distance of two or more video cameras overlap, it is necessary to break the connection between those nodes. This is explained by the definition of adjacency matrix **Y**.

Below is the algorithm to determine neighbor nodes:

(i) Set camera nodes and non-camera nodes on the diagram as shown in object (b) of Fig. 9.

(ii) Transform the diagram to a graph as shown in object (c) of Fig. 9.

(iii) Generate an adjacency matrix **X** from camera node locations and non-camera node locations on the graph, and generate an adjacency matrix **Y** from non-camera node locations on the graph. Adjacency matrix **X** indicates that rows are camera nodes and columns are non-camera nodes. Adjacency matrix **Y** indicates that rows and columns are non-camera nodes, which results in adjacency matrix **Y** resolving an overlap problem of view distances between video cameras.

(iv) Calculate adjacency matrix **X'** and **Y'** by excluding unnecessary non-camera nodes from adjacency matrix **X** and **Y**.

(v) Calculate neighbor's location matrix by multiplying adjacency matrix and transposed matrix $\mathbf{X'^T}$. This neighbor's location matrix is the neighbour's node information.

An unnecessary non-camera node is a non-camera node which has no camera node as a neighbor. Adjacency matrix **X**' and **Y**' are computed without unnecessary nodes, and using the procedure shown later. There are reasons why it might be better to include the unnecessary nodes in the diagram from the beginning as we have done. Since the risk of committing an error will be higher as the diagram becomes larger, we include the unnecessary nodes from the beginning and remove them at the end.

3.3. Fundamental of the Algorithm

The object (a) of Fig. 9 is expressed in the object (c) of Fig. 9 as a graph. In the object (c) of Fig. 9, the adjoining nodes are connected with a line. Table 1 is the adjacency matrix **X** which is a table representation based on the object (c) of Fig. 9, and Table 2 is the temporary adjacency matrix **Y** which is a table representation based on the object (c) of Fig. 9 before satisfying all conditions. Table 3 is the adjacency matrix **Y** whose conditions are satisfied after all.

Table 1
Adjacency Matrix X with Camera Nodes and Non-camera Nodes

X	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13
a1	1	0	0	0	0	0	1	1	0	0	0	0	0
a2	0	1	0	0	0	0	1	0	0	0	0	0	0
a3	0	0	1	0	0	0	0	1	0	0	0	0	0
a4	0	0	0	1	0	0	0	0	1	0	0	0	0
a5	0	0	0	0	1	0	0	0	0	1	0	0	0
a6	0	0	0	0	0	1	0	0	0	0	1	0	0

Table 1 is the adjacency matrix **X** which consists of p rows and q columns, in which $|\mathbf{A}| = p$ and $|\mathbf{V}| = q$. Element x_{ij} is defined as (1) in adjacency matrix **X**.

$$x = \begin{cases} 1 & \text{There is the line which links camera node } a_i \text{ and non-camera node } v_j. \\ 0 & \text{There is no link} \end{cases} \quad (1)$$

Table 2
Adjacency Matrix Y with Two Non-camera Nodes

Y	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13
v1	0	0	0	0	0	0	1	0	0	0	0	0	0
v2	0	0	0	0	0	0	1	0	0	0	0	0	0
v3	0	0	0	0	0	0	0	1	0	0	0	0	0
v4	0	0	0	0	0	0	0	0	1	0	0	0	0
v5	0	0	0	0	0	0	0	0	0	1	0	0	0
v6	0	0	0	0	0	0	0	0	0	0	1	0	0
v7	1	1	0	0	0	0	0	1	0	0	0	0	0
v8	0	0	1	0	0	0	1	0	0	0	0	0	1
v9	0	0	0	1	0	0	0	0	0	0	0	0	1
v10	0	0	0	0	1	0	0	0	0	0	0	1	0
v11	0	0	0	0	0	1	0	0	0	0	0	1	0
v12	0	0	0	0	0	0	0	0	0	1	1	0	1
v13	0	0	0	0	0	0	0	1	1	0	0	1	0

Table 2 is the adjacency matrix **Y** which consists of q rows q columns, in which $|\mathbf{V}| = q$. Element y_{ij} is defined as (2) in adjacency matrix **Y** before satisfying all conditions.

$$y_{ij} = \begin{cases} 1 & \text{There is the line which links two non-camera node } v_i \text{ and } v_j \\ 0 & \text{There is no link} \end{cases} \quad (2)$$

Consider the problem of d-1 in Fig. 8 and Fig. 9. Video cameras a_2 and a_3 are in a situation in which they have the concern with overlapped view in Fig. 9. Examination shows that the non-camera nodes v_7 and v_8 are adjacent to more than one camera. The summation of column v_7 in **X** and the summation of column v_8 in **X** is larger than 1, if the adjacency matrix **X** is used. If the summation is larger than 1, then the number of cameras around the non-camera node is more than 1. In addition, if the adjacency matrix **Y** is used, $y_{78} = y_{87} = 1$ also indicates that v_7 is a neighbor of v_8. In this case, the neighbor relationship of non-camera nodes, v_7 and v_8, can be separated to resolve the overlapped view problem. Non-camera nodes, v_7 and v_8, can be separated if it is assumed $y_{78} = y_{87} = 0$.

If the conditions (3) are satisfied, then element y_{ij} and element y_{ji} are replaced as $y_{ij} = y_{ji} = 0$.

$$y_{ij} = y_{ji} = 1, \sum_{n=1}^{m} x_{ni} > 1, \sum_{n=1}^{m} x_{nj} > 1 \quad (3)$$

Table 3
Adjacency Matrix Y with two Non-camera Nodes and the Condition

Y	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13
v1	0	0	0	0	0	0	1	0	0	0	0	0	0
v2	0	0	0	0	0	0	1	0	0	0	0	0	0
v3	0	0	0	0	0	0	0	1	0	0	0	0	0
v4	0	0	0	0	0	0	0	0	1	0	0	0	0
v5	0	0	0	0	0	0	0	0	0	1	0	0	0
v6	0	0	0	0	0	0	0	0	0	0	1	0	0
v7	1	1	0	0	0	0	0	0	0	0	0	0	0
v8	0	0	1	0	0	0	0	0	0	0	0	0	1
v9	0	0	0	1	0	0	0	0	0	0	0	0	1
v10	0	0	0	0	1	0	0	0	0	0	0	1	0
v11	0	0	0	0	0	1	0	0	0	0	0	1	0
v12	0	0	0	0	0	0	0	0	0	1	1	0	1
v13	0	0	0	0	0	0	0	1	1	0	0	1	0

Table 3 is the adjacency matrix **Y** that satisfies the conditions (3) and resolves the problem of d-1 in Fig. 8. The summation of column v_7 and column v_8 in adjacency matrix **X** are larger than 1, therefore y_{78} and y_{87} are replaced by 0.

Matrix **Z** is a transposed matrix **X**. It is possible to consider that x_{ij} indicates a neighbor condition for camera node a_i to non-camera node v_j, and z_{ji} indicates a neighbor condition for non-camera node v_j to camera node a_i.

Next, considering the case as shown in Fig. 10, the camera node a_i is a neighbor to camera node a_j via non-camera node v_n. The relationship between camera node a_i and non-camera node v_n is treated as $x_{in} = 1$ and the relationship between non-camera node v_n and camera node a_i is treated as $z_{nj} = 1$. In addition, considering camera node a_i can reach camera node a_j via non-camera node v_n, then $x_{in} \times z_{nj} = 1$ can be derived. Therefore, the arithmetic expression indicates the relationship between camera a_i and camera a_j, and it is possible to state that camera node a_i is a neighbor to camera node a_j. If it is assumed that the element b_{ij} of adjacency matrix **B** indicates the relationship between camera node a_i and camera node a_j, then the arithmetic expression via non-camera node v_n ($n = 1 \ldots m$) is possible to be expressed as the expression (4). However, if $i = j$, it makes $b_{ij} = 0$,

because neighbor relationships as $i = j$ indicate a camera as neighbor to itself.

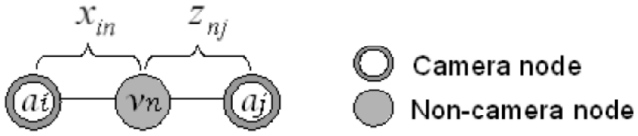

Figure 10: Two Camera Nodes via one Non-camera Node

$$b_{ij} = \sum_{n=1}^{m} x_{in} z_{nj} \quad \begin{matrix} a_i \text{ is adjacent to } a_j. \\ a_i \text{ is not adjacent to } a_j. \end{matrix} \qquad (4)$$

Considering the case in Fig. 11, camera node a_i is a neighbor to non-camera node v_j via non-camera node v_n. The relationship between camera node a_i and non-camera node v_n is treated as $x_{in} = 1$, and the relationship between non-camera node v_n and non-camera node v_j is treated as $y_{jn} = 1$. y_{jn}, as an element of transposed matrix **Y** can be represented as $ynj = yjn = 1$. Additionally, considering camera node a_i can reach non-camera node v_j via non-camera node v_n, $x_{in} \times y_{nj} = x_{in} \times y_{jn} = 1$ can be derived. Therefore, the arithmetic expression specifies the relationship between camera node a_i and non-camera node v_j, and it is possible to define camera node a_i as a neighbor to non-camera node v_j. If it is assumed that adjacency matrix element c_{ij} indicates a relationship between camera node a_i and non-camera node v_j, then the arithmetic expression via non-camera node v_n ($n = 1,\ldots,m$) can be expressed as the expression (5).

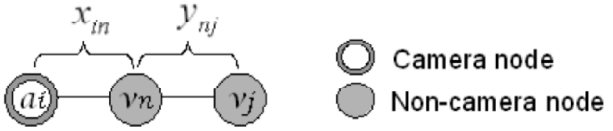

Figure 11: Camera Node and Non-camera Node via One Non-camera Node

$$c_{ij} = \sum_{n=1}^{m} x_{in} y_{nj} \quad \begin{cases} \geq 1 & a_i \text{ is adjacent to } v_j. \\ = 0 & a_i \text{ is not adjacent to } v_j. \end{cases} \qquad (5)$$

Considering the case on Fig. 12, the camera node a_i is a neighbor to camera node a_j via two non-camera nodes, v_n and v_m. If it is assumed that the element of adjacency matrix **D** is d_{ij}, it is possible to derive the expression (6) applying the result of Fig. 11.

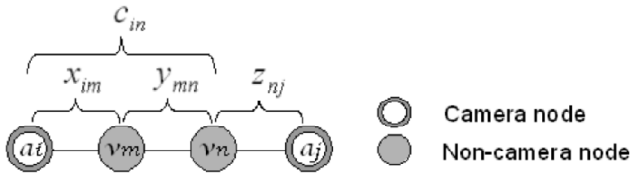

Figure 12: Two Camera Nodes via two Non-camera Nodes

$$d_{ij} = \sum_{n=1}^{m} c_{in} z_{nj} \quad \begin{cases} \geq 1 & a_i \text{ is adjacent to } a_j. \\ = 0 & a_i \text{ is not adjacent to } a_j. \end{cases} \qquad (6)$$

From the above results, when adjacency matrix **E** is calculated via n or more nodes, it can use the expression (7).

$$E = X(Y)^{n-1} X^T \quad \begin{cases} \geq 1 & a_i \text{ is adjacent to } a_j. \\ = 0 & a_i \text{ is not adjacent to } a_j. \end{cases} \qquad (7)$$

When the value of n on the expression (7) is considered, it is difficult to decide whether or not the non-camera nodes are n, especially when they are in a loop, as shown in Fig. 13. If a loop of non-camera nodes does not exist, the problem will not occur. In this case, adjacency matrixes **X'** and **Y'** are computed so that a diagram may be constituted from camera nodes and non-camera nodes without unnecessary non-camera nodes. Fig. 14 illustrates the graph re-calculated from Fig. 9.

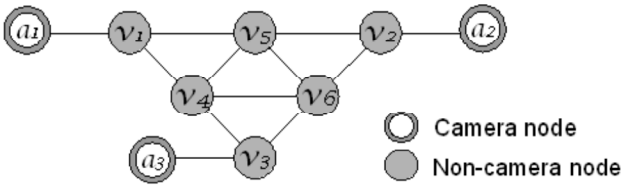

Figure 13: Graph that Non-camera Nodes are Looped

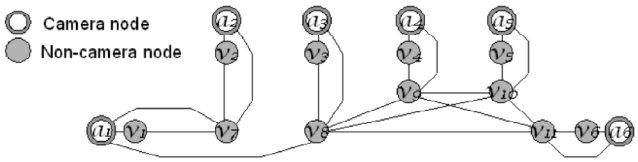

Figure 14: Graph without Unnecessary Non-camera Nodes

3.4. Elimination of Unnecessary Non-Camera Node

By considering the adjacency matrix **X'**, **Y'** computed without unnecessary non-camera nodes, unnecessary non-camera nodes can be seen to be nodes that are not connected to any camera nodes. The matrix is calculated by the following procedure.

In the case of the adjacency matrix **X**, the procedure is stated below:

(i) Search for unnecessary non-camera node v_n in which camera node is not a neighbor.

(ii) Remove the column of the node v_n.

The adjacency matrix **X'** is computed by the adjacency matrix **X** without the unnecessary nodes.

When the data on Table 1 is considered as an example, the unnecessary non-camera nodes can be identified as highlighted in Table 4. The columns v_{12} and

v_{13} represent the unnecessary non-camera nodes, and adjacency matrix **X'** become as Table 5.

Table 4
Unnecessary Nodes in Adjacency Matrix X

X	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13
a1	1	0	0	0	0	0	1	1	0	0	0	0	0
a2	0	1	0	0	0	0	1	0	0	0	0	0	0
a3	0	0	1	0	0	0	0	1	0	0	0	0	0
a4	0	0	0	1	0	0	0	0	1	0	0	0	0
a5	0	0	0	0	1	0	0	0	0	1	0	0	0
a6	0	0	0	0	0	1	0	0	0	0	1	0	0

Table 5
Adjacency Matrix X'

X'	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11
a1	1	0	0	0	0	0	1	1	0	0	0
a2	0	1	0	0	0	0	1	0	0	0	0
a3	0	0	1	0	0	0	0	1	0	0	0
a4	0	0	0	1	0	0	0	0	1	0	0
a5	0	0	0	0	1	0	0	0	0	1	0
a6	0	0	0	0	0	1	0	0	0	0	1

In the case of adjacency matrix **Y**, non-camera node v_n is extracted to compute adjacency matrix **X'**. The procedure is stated below:

(i) Determine the column of unnecessary non-camera node v_n from the adjacency matrix **X**.

(ii) Perform an OR operation on the column of unnecessary non-camera node v_n and the columns of all the neighbor nodes of v_n on adjacency matrix **Y**.

(iii) Perform an OR operation on the row of unnecessary non-camera node v_n and the rows of all the neighbor nodes of v_n on adjacency matrix **Y**.

(iv) Remove the row and column of the unnecessary non-camera node v_n from the adjacency matrix **Y**.

The adjacency matrix **Y'** is computed from the adjacency matrix **Y** without the unnecessary non-camera nodes. When Table 3 is considered as an example, the unnecessary non-camera nodes can be identified as highlighted in Table 6. The inherited result is shown on Table 7. The rows and columns of v_{13} and v_{14} represent the unnecessary non-camera nodes, and the adjacency matrix **Y'** becomes Table 8. It makes y'_{ij} ($i = j$) to 0, because neighbor relation as $i = j$ indicates neighbor to itself.

Table 6
Unnecessary Nodes in Adjacency Matrix Y

Y	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13
v1	0	0	0	0	0	0	1	0	0	0	0	0	0
v2	0	0	0	0	0	0	1	0	0	0	0	0	0
v3	0	0	0	0	0	0	0	1	0	0	0	0	0
v4	0	0	0	0	0	0	0	0	1	0	0	0	0
v5	0	0	0	0	0	0	0	0	0	1	0	0	0
v6	0	0	0	0	0	0	0	0	0	0	1	0	0
v7	1	1	0	0	0	0	0	0	0	0	0	0	0
v8	0	0	1	0	0	0	0	0	0	0	0	0	1
v9	0	0	0	1	0	0	0	0	0	0	0	0	1
v10	0	0	0	0	1	0	0	0	0	0	0	1	0
v11	0	0	0	0	0	1	0	0	0	0	0	1	0
v12	0	0	0	0	0	0	0	0	0	1	1	0	1
v13	0	0	0	0	0	0	0	1	1	0	0	1	0

Table 7
Merged Result from Table 6

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13
v1	0	0	0	0	0	0	1	0	0	0	0	0	0
v2	0	0	0	0	0	0	1	0	0	0	0	0	0
v3	0	0	0	0	0	0	0	1	0	0	0	0	0
v4	0	0	0	0	0	0	0	0	1	0	0	0	0
v5	0	0	0	0	0	0	0	0	0	1	0	0	0
v6	0	0	0	0	0	0	0	0	0	0	1	0	0
v7	1	1	0	0	0	0	0	0	0	0	0	0	0
v8	0	0	1	0	0	0	0	1	1	1	1	1	1
v9	0	0	0	1	0	0	0	1	1	1	1	1	1
v10	0	0	0	0	1	0	0	1	1	1	1	1	1
v11	0	0	0	0	0	1	0	1	1	1	1	1	1
v12	0	0	0	0	0	0	0	1	1	1	1	0	1
v13	0	0	0	0	0	0	0	1	1	1	1	1	1

Table 8
Adjacency Matrix Y'

Y'	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11
v1	0	0	0	0	0	0	1	0	0	0	0
v2	0	0	0	0	0	0	1	0	0	0	0
v3	0	0	0	0	0	0	0	1	0	0	0
v4	0	0	0	0	0	0	0	0	1	0	0
v5	0	0	0	0	0	0	0	0	0	1	0
v6	0	0	0	0	0	0	0	0	0	0	1
v7	1	1	0	0	0	0	0	0	0	0	0
v8	0	0	1	0	0	0	0	1	1	1	1
v9	0	0	0	1	0	0	0	1	1	1	1
v10	0	0	0	0	1	0	0	1	1	1	1
v11	0	0	0	0	0	1	0	1	1	1	1

3.5. Expressions for Neighbor Nodes

It is possible to derive neighbor node matrix **E** using expression (8) from the results of the preceding paragraphs. However, if $i = j$ then the value of e_{ij} in matrix E is replaced with 0 because neighbor relation $i = j$ indicates it is a neighbor itself. The matrix **E** represents information on the neighbor nodes.

$$E = X'Y'X' \begin{cases} \geq 1 & a_i \text{ is adjacent to } a_j. \\ = 0 & a_i \text{ is not adjacent to } a_j. \end{cases} \quad (8)$$

4. EXAMINATION

The above algorithm was tested on the diagram as shown on Fig. 15. Three kinds of view patterns were applied to Fig. 15 in this test. Their patterns represent information on which view is crossed, which view is crossed and blocked, and finally which view is blocked.

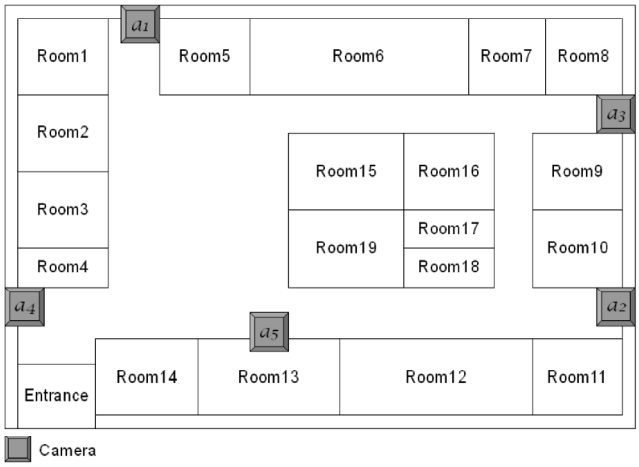

Figure 15: Base Diagram for Examination

4.1. Pattern with which View is Crossed

Fig. 16 represents a pattern in which the views for camera nodes, a_4 and a_5, are crossed at non-camera node v_8. Fig. 17 is a graph of Fig. 16 which is based on the algorithm.

Table 9 is the calculation results for the pattern of Fig. 16 by the algorithm. Each element shows a number of route pattern which a mobile agent can reach from a certain camera to neighbor cameras after eliminating

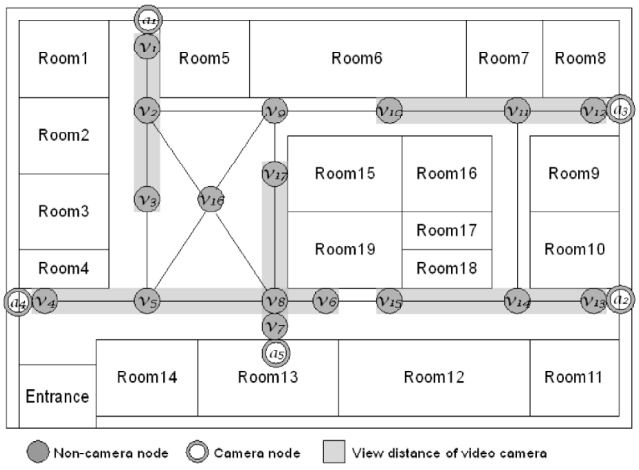

Figure 16: Pattern with which View is Crossed

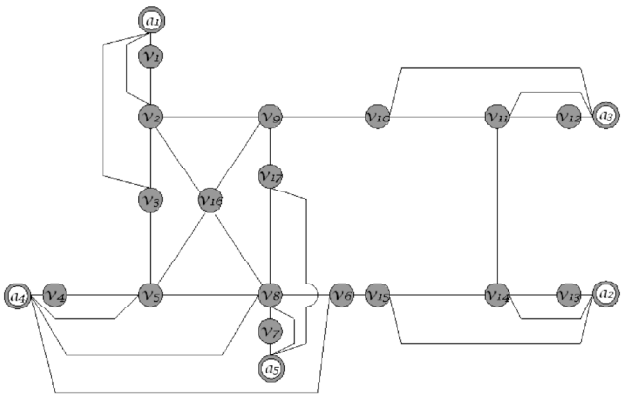

Figure 17: Graph of Fig. 16

unnecessary nodes. Arrows of Fig. 18 show adjacency of video cameras from the result.

Table 9
Calculated Adjacency Matrix E for Fig. 16

E	a1	a2	a3	a4	a5
a1	0	0	1	3	2
a2	0	0	1	1	0
a3	1	1	0	2	2
a4	3	1	2	0	5
a5	2	0	2	5	0

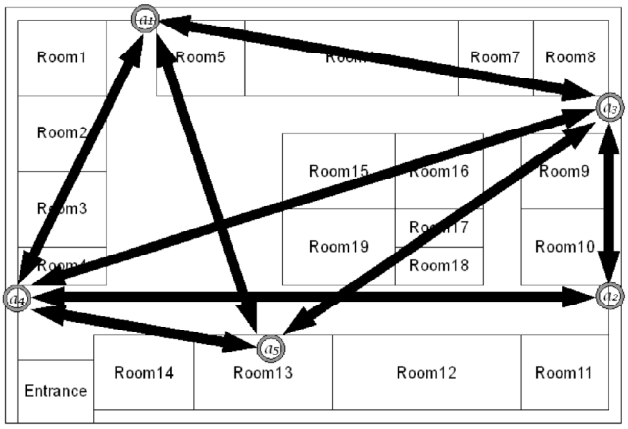

Figure 18: Adjacency of Fig. 16

4.2. Pattern with which View is Crossed and Blocked

Fig. 19 shows a pattern in which views for camera nodes, a_4 and a_5, are crossed at non-camera node v_8 and the view for camera node a_3 is blocked by the view for camera node a_5 at non-camera node v_9. Fig. 20 is a graph of Fig. 19 which is based on the algorithm.

Table 10 is the calculation results for the pattern of Fig. 19 by the algorithm. Each element shows a number

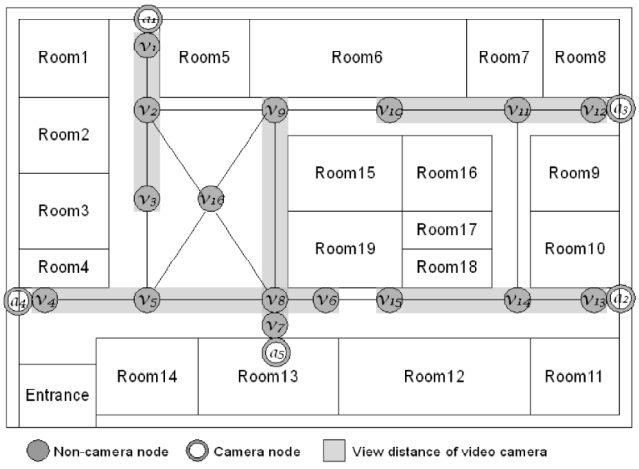

Figure 19: Pattern with which View is Crossed and Blocked

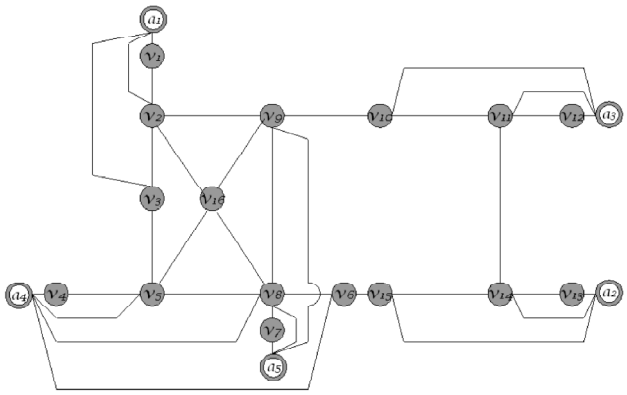

Figure 20: Graph of Fig. 19

of a route pattern which a mobile agent can reach from a certain camera to neighbor cameras after eliminating unnecessary nodes. Arrows of Fig. 21 show adjacency of video cameras from the result.

Table 10
Calculated Adjacency Matrix E for Fig. 19

E	a_1	a_2	a_3	a_4	a_5
a_1	0	0	0	3	2
a_2	0	0	1	1	0
a_3	0	1	0	0	1
a_4	3	1	0	0	4
a_5	2	0	1	4	0

4.3. Pattern with which View is Blocked

Fig. 22 shows a pattern in which the view for camera node a_3 is blocked by the view for camera node a_5 at non-camera node v_9. Fig. 23 is a graph of Fig. 22 which is based on the algorithm.

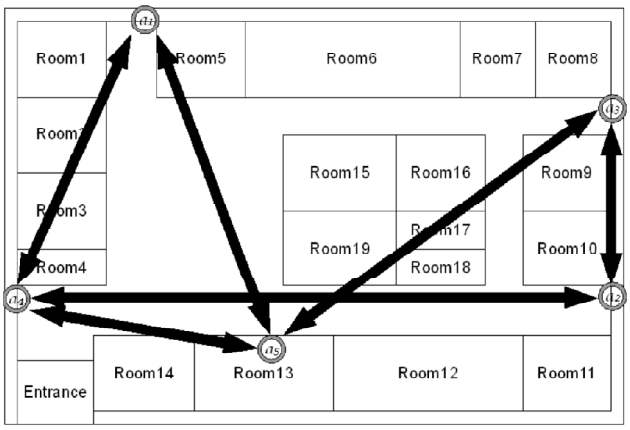

Figure 21: Adjacency of Fig. 19

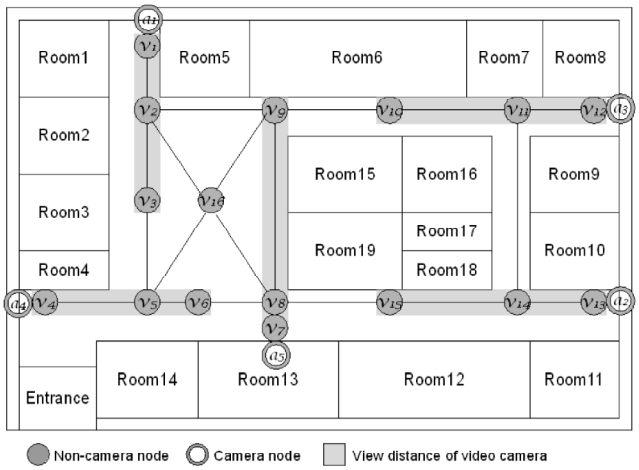

Figure 22: Pattern with which View is Blocked

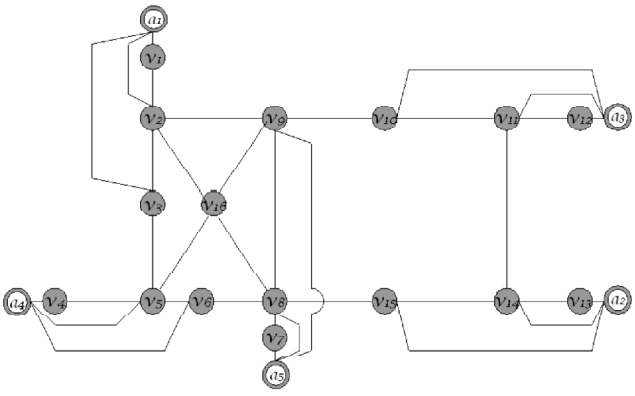

Figure 23: Graph of Fig. 22

Table 11 is the calculation results for the pattern of Fig. 22 by the algorithm. Each element shows a number of route patterns which a mobile agent can reach from a certain camera to neighbor cameras after eliminating unnecessary nodes. Arrows of Fig. 24 show adjacency of video cameras from the result.

Table 11
Calculated Adjacency Matrix E for Fig. 22

E	a1	a2	a3	a4	a5
a1	0	0	0	2	2
a2	0	0	1	0	1
a3	0	1	0	0	1
a4	2	0	0	0	3
a5	2	1	1	3	0

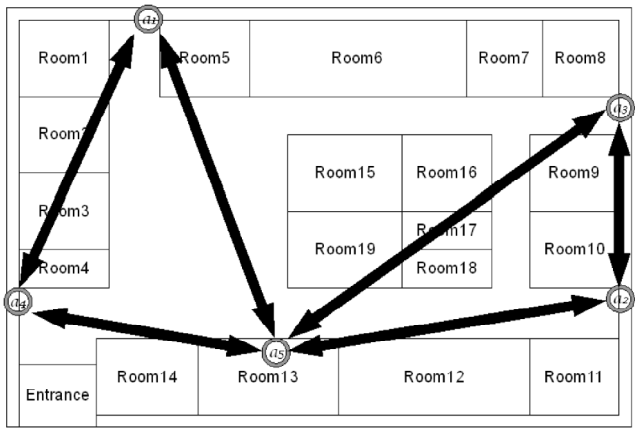

Figure 24: Adjacency of Fig. 22

5. CONCLUSION

As shown, this algorithm can easily determine the neighbor nodes, even when the view distance of the video camera is changed and modification to the number of installed video camera occurs. According to experimental tests, the algorithm can operate with an execution time of less than 10 milliseconds for a 5×16 adjacency matrix **X** and a 16×16 adjacency matrix **Y**.

Results of computing the time efficiency of the algorithm with other sized matrixes is shown in Fig. 25. The graph shows the mean value of the computing time as executed 5 times respectively and is almost linearly proportional to the increase in the size of the matrix. This result does not change when calculating off-line, but when used for real time on-line, we will verify the computing speed and improve the algorithm if necessary.

This algorithm can also determine the neighbor nodes accurately in the case of the complicated diagram as shown in Fig. 26. In addition, the mobile agents can efficiently and accurately decide to which server they should transfer by utilizing the neighbor node information determined by this algorithm. This research contributes to previous work by describing a way to implement a method which can efficiently track a target entity using mobile agent.

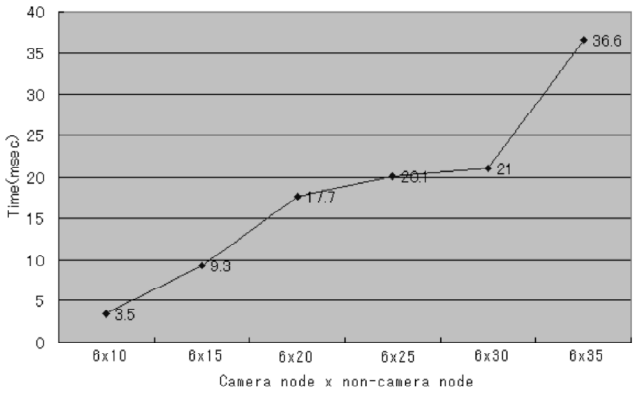

Figure 25: Calculation Speed of the Algorithm

The simulator can also be used effective to simulate a target entity. In other examinations, mobile agents were able to simultaneously track numerous entities in the automatic human tracking system with the simulator mentioned previously. However, at the present, the simulator needs more improvement especially when using more than 20 cameras. We are striving hard to improve the this.

An agent's transfer rate is from 600 milliseconds to 700 milliseconds using HTTP protocol. This result is slightly poor when considering the speed of the target. If TCP protocol is applied to the agent transfer, the rate was less than 100 milliseconds and it is enough to satisfy the concern. However, HTTP is better than TCP in that HTTP is convenient, flexible, and native on the Internet. Therefore, the agent transfer rate on the HTTP will be improved.

When it is assumed that agents transfer efficiently, highly precise tracking is realizable by utilizing the information that indicates the movement direction of a tracked person on video camera. In future research we will aim at the establishment of the precise tracking

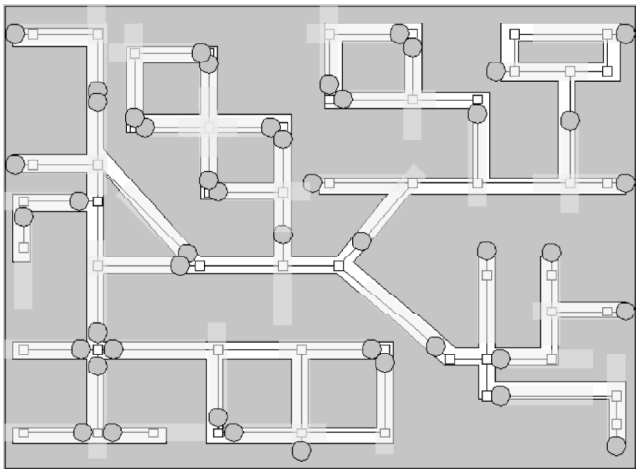

Figure 26: Complicated Diagram

technique using the algorithm adding the information of movement direction.

ACKNOWLEDGEMENTS

This work was supported in Melco Power Systems Co., Ltd. and in Tottori University. The authors would also like to acknowledge Mr. Yasuo Matsumiya, Mr. Kiyoshi Tsutsui, Mr. Masayuki Shibamoto, Mr. Mitsuo Hoshino, Mr. Kozo Tanigawa, Mr. Tappei Yotsumoto, Mr. Michio Kojima, Mr. Hiroyuki Adachi, Mr. Kensuke Fukumoto, Ms. Mayu Tanaka, Mr. Hubert Ambales, Mr. Yusuke Hamada, and Mr. Shinya Iwasaki.

REFERENCES

[1] Cabri, G., Leonardi, L., & Zambonelli, F. (2000), Mobile-Agent Coordination Models for Internet Applications, *Computer*, **33**: 2, 82–89.

[2] Cui, Y., Hasler, N., Thormaehlen, T., & Seidel, H. P. (2009), Scale Invariant Feature Transform with Irregular Orientation Histogram Binning, *Proceedings of the International Conference on Image Analysis and Recognition*, ICIAR 2009, Halifax, Canada: Springer.

[3] Erdem, U. M., & Sclaroff, S. (2006), Automated Camera Layout to Satisfy Task-specific and Floor Plan-specific Coverage Requirements, *CVIO2006*, **103**(3), 156–169.

[4] Gray, R. S., Cybenko, G., Kotz, D., Peterson, R. A., & Rus, D. (2002), D'Agents: Applications and Performance of a Mobile-agent System, *Software: Practice and Experience*, **32**: 6, 543–573.

[5] Hamada, Y., Iwasaki, S., Kakiuchi, H., Kawamura, T., & Sugahara, K. (2008), Pursuit Methods for Automatic Human Tracking System based on Mobile Agent Technologies, *Proceedings of the 59th Chugoku branch union convention of the Institute of Electrical Engineers of Japan and Information Processing Society of Japan*, Tottori, Japan, 486.

[6] Ishida, N., Hamada, Y., Kakiuchi, H., Shimizu, T., Kawamura, T., & Sugahara, K. (2008), Feature Extraction Method for Automatic Human Tracking System based on Mobile Agent Technologies, *Proceedings of the 59th Chugoku branch union convention of the Institute of Electrical Engineers of Japan and Information Processing Society of Japan*, Tottori, Japan, 418.

[7] Jennings, N. R. (2001), An Agent-based Approach for Building Complex Software Systems, *Communications of the ACM* , **44**: 4, 35–41.

[8] Kakiuchi, H., Hamada, Y., Kawamura, T., Shimizu, T. & Sugahara, K. (2008), To Realize Automatic Human Tracking System based on Mobile Agent Technologies, *Proceedings of the 59th Chugoku branch union convention of the Institute of Electrical Engineers of Japan and Information Processing Society of Japan*, Tottori, Japan, 485.

[9] Kawaguchi, Y., Shimada, A., Arita, D., & Taniguchi, R. (2008), Object Trajectory Acquisition with an Active Camera for Wide Area Scene Surveillance, *IPSJ SIG Technical Report*, 2008–CVIM–163, 1306–1311.

[10] Kawamura, T., Motomura, S., & Sugahara, K. (2005), Implementation of a Logicbased Multi Agent Framework on Java Environment, *Proceedings of IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems (Henry Hexmoor (eds.))*, Waltham, Massachusetts, USA, 486–491.

[11] Lange, D. B., & Oshima, M. (1999), Seven Good Reasons for Mobile Agents, *Communications of the ACM*, **42**: 3, 88–89.

[12] Lowe, D. G. (2004), Distinctive Image Features from Scale-Invariant Keypoints, *International Journal of Computer Vision* **60**(2), 91–110.

[13] Monticolov, D., Hilaire, V., Gomes, S., & Koukam, A. (2008) A Multi-agent System for Building Project Memories to Facilitate Design Process, *Integrated Computer-Aided Engineering*, **15**: 1, 3–20.

[14] Motomura, S., Kawamura, T., & Sugahara, K. (2005), Maglog: A Mobile Agent Framework for Distributed Models, *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, Phoenix, Arizona, USA, 414–420.

[15] OSGi Alliance (2008), OSGi Alliance Specifications OSGi Service Platform Release 1, *http://www.osgi.org/Specifications/HomePage*.

[16] Pettre, J., Simeon, T., & Laumond, J. P. (2002), Planning Human Walk in Virtual Environments, *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems, Lausanne*, Switzerland, 3048–3053.

[17] Terashita, K., Ukita, N., & Kidode, M. (2009), Efficiency Improvement of Probabilistic-Topological Calibration of Widely Distributed Active Cameras, *IPSJ SIG Technical Report*, 2009–CVIM–166, 241–248.

[18] Valetto, G., Kaiser, G., & Gaurav S. K. (2001), A Mobile Agent Approach to Process-based Dynamic Adaptation of Complex Software Systems, *Lecture Notes in Computer Science*, **2077**, 102–116.

[19] Yao, Y., Chen, C. H., Abidi, B., Page, D., Koschan, A., & Abidi, M. (2008a), Sensor Planning for Automated and Persistent Object Tracking with Multiple Cameras, *CVPR2008*.

[20] Yao, Y., Chen, C. H., Abidi, B., Page, D., Koschan, A., & Abidi, M. (2008b), Sensor Planning for PTZ Cameras Using the Probability of Camera Overload, *ICPR2008*.

[21] Yin, H., & Hussain, I. (2008), Independent Component Analysis and Nongaussianity for Blind Image Deconvolution and Deblurring, *Integrated Computer-Aided Engineering*, **15**: 3, 219–228.