Real-Time Motion Tracking System using Omni-Directional PTZ Camera

Application Software Research and Development Report

Director. Kiyeon Lee
Researcher. Joonseok Lee
Researcher. Hyunwoong Shin
Researcher. Sunghwan Kim
Researcher. Jeongwoo Choi

2006. 8. 31

Small and Medium Business Administration

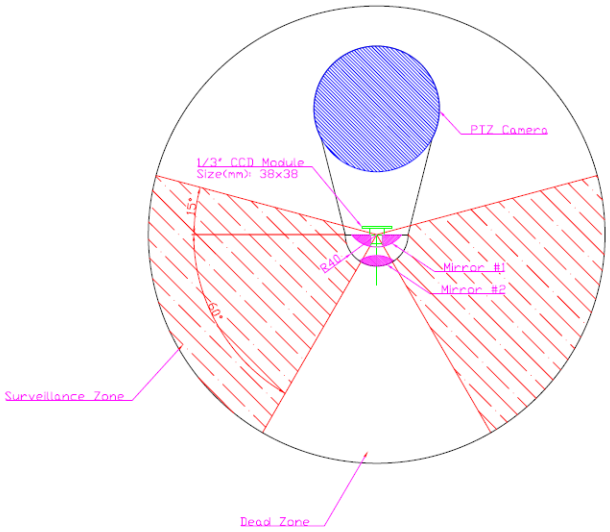
Table of Contents

I. Introduction

1. Program Overview

This program aims for intelligently tracking moving objects using a wide-angle lens and Pan-tilt-zoom (PTZ) camera.

The overall structure of this system is shown in Figure 1. We use a wide-angle lens which can see all over the areas under the camera, except for a little dead zone right under the camera. The sensor camera (CCD module) sends an image of its observing range to this program. Then, it detects from the continuous set of images and selects a target to track. The coordinate of target is sent to PTZ camera, then it rotates to capture the target. One cycle of tracking finishes when this program takes its snapshots.



[Figure 1] System Structure

Overall flow of this program is shown in Figure 2. The input image to the program is captured 7.5 times in a second (7.5 fps) by CCD module (we use the term sensor camera for this, interchangeably.) We first convert the image into gray-colored one and conduct some preprocessing. When the image is prepared, we apply block-based image difference algorithm in order to find out moving objects. Discovered objects are assumed as people when they are big enough, or ignored otherwise. (Size filtering) After detection process, we compare them with other objects in moving objects list, in the respect of color, location, and moving speed. We distinguish each moving object by these characteristics. When distinguishing process finishes, we select one of them as an actual target for tracking, by using our multi-object tracking algorithm. We calculate center point of the target and move the PTZ camera to the point. We take four to five snapshots for each target. This whole process repeats for each frame.

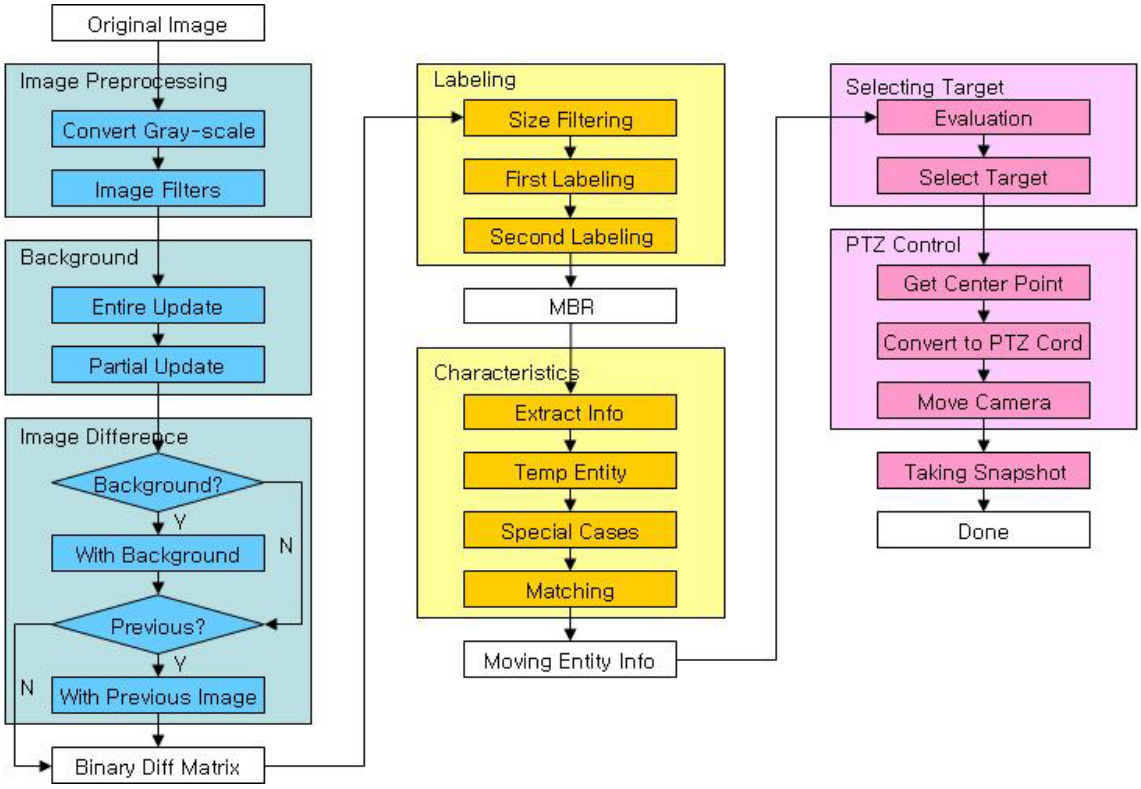
In the next chapter, we explain how this program detects moving objects. how it distinguishes each object, how it chooses the target for tracking, and how it locates PTZ camera to the exact point of moving object, in detail.

2. User Interface

The screen of this program is shown in Figure 3. Menu includes:

Camera: connect to a network camera.

Open: test the program with saved images.

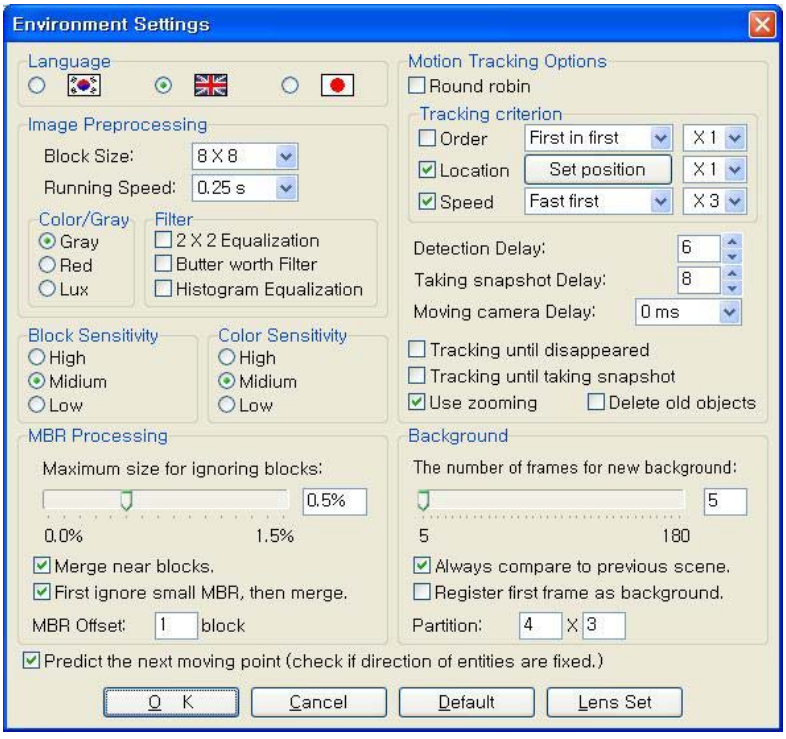


[Figure 2] Overall Flow

[Figure 3] Program User Interface

Next: open the next image when testing with saved images.
Auto: open next images automatically, with a specific time interval.
Background: register the current image as background manually.
Debug: see some debug information instead of taking snapshots.
Environment: open customizing window.
Information: see program information.
Exit: exit the program.

On the right side of the screen, we can see the result of motion detection from the sensor camera. Detected objects are assigned a number and different color. Among them, the tracking target is marked with ★ sign. Detailed information about the tracking target is shown on the top-right panel. (Object number, the number of frames, pan, tilt, zoom) You can see the zoomed image of the tracking target on the big screen. You also can see a panorama image on the bottom, with a yellow box showing current location of the camera. Snapshots are listed on the right-bottom side. Those snapshots are saved in the sub-directory named "snapshots" as jpg images.



[Figure 4] Customizing Options

Figure 4 shows the screen for customizing. This program provides Korean, English, and Japanese, with various options. You can customize block size, gray image mode, and kinds of image filters for preprocessing. Sensitivity of image difference algorithm is determined by user's selection on block sensitivity and color-difference

sensitivity. For MBR processing, minimum size of block and block merging option can be customized. For motion tracking, you can specify criteria for choosing tracking target as well as degree of zoom. For background image, the number of frames, division size, and whether comparing with the previous image can be determined by users.

3. Development and Test Environment

CPU: Intel Pentium IV 2.6GHz
RAM: 512MB
Network: 100Mbps LAN
Programming Language: Delphi 7
Operating System: Microsoft Windows XP/2000/2003

II. Implementation Details

1. Block-based Motion Detection Algorithm

(1) Image Preprocessing

We apply image preprocessing for the input image. This is for simple and faster image difference operation as well as for better tolerance into the image. Preprocessing includes converting into gray image and image filtering.

1) Converting into gray image

Windows system expresses colors by using RGB color, summing components of red, green, and blue. Beside this, HSL and CMYK also use similar way of expressing colors. Therefore, it takes comparatively long time to check how two colors are similar. In order to simplify this problem, we convert images into gray-scale. When images are converted into gray-scale, we can deal with them with only one value. Many methods for converting to gray image have suggested, but we implemented the following three.

① Standard gray-scale

We can get gray value of each pixel by

$$Gray = 0.3R + 0.6G + 0.1B$$

where R, G, B denote red, green, and blue component, respectively.

② Red component

We may use red component as a representative of the color. Skin colors generally have more component in red, so this improves ability to distinguish people.

$$Gray = R$$

③ Luminance

When the color is expressed as HSL color, H, S, L means hue, saturation, and luminance, respectively. We may use this luminance value as a representative. This method reflects R, G, B values without any weight, unlike the standard gray-scale.

$$Gray = L = \frac{R + G + B}{3}$$

[Figure 5] Gray Converting result

(Original image, Standard Gray-scale, Luminance, Red Component, in that order.)

2) Image Filtering

Image filtering means giving some changes to the original one generally. In this program, we primarily use image filters for removing noise on the input image. We implement the following three filters, and they can be applied by any combination.

① 2 X 2 Equalization

In this method, a 2 X 2 block is used as an unit. Color of this block is assigned by averaging colors in the original image. This is helpful to remove abrupt color changes, generally caused by sudden change in light or small dust, in a specific pixel, by putting average with adjacent pixels.

② Butterworth Filter

This filter removes sudden change of color in the respect time. For each frame, every pixel is converted to "color of current frame + that of previous frame input + that of previous frame after the filter applied." Thus, sudden change in color can be discarded by giving average with previous values.

③ Histogram Equalization

This is a method in image processing of contrast adjustment using the image's histogram. When the light repeats being light and dark, this filter removes the effect by redistributing the luminance histogram. Therefore, color change effect caused by fluorescent lamp can be removed with this filter.

Now, we finished preprocessing and are ready to apply motion detection algorithm. All of the options discussed in this subsection can be customized by users.

(2) Background Image

In each frame, it is general that only some small part of the image changes compared to its previous one, because of the short time interval. We can say that the unchanged part as background. If we can distinguish background and moving objects, it would be easier to detect and track them.

In this program, we compare each frame with background image in order to get moving objects. Therefore, it is important to have pure background without any moving objects on it. For this, we use initialization process, entire update, and partial update of background.

1) Initialization

In the initial phase, we have no background. In this phase, we cannot apply image difference between background and current frame. So, we only apply it between current frame and previous frame. When we can judge there is no moving object on the screen, we register the current screen as background. If we cannot find any moving object within the last n frames, background is registered. If any moving object is found, counting starts from 0 again. This n value can be customized, but default is 120 frames (about 16 seconds).

2) Entire update

When no moving object has found since the last 120 frames (default, customizable), current scene is registered as background. This periodic update applies small change on the image such as gradual brightness change.

3) Partial update

It is almost impossible to update the whole background when there is at least one moving object in the screen, because it requires long time. Partial update allows us to renew background for areas with no moving object, even though we have in some part. Background is divided into 2 X 2 or 4 X 3, and each part is registered as background if there is no moving object for the last $\frac{n}{2}$ frames. However, this work should require harsher condition than entire update, due to its relatively shorter time interval. We allow background update only if no pixel in the division is changed.

(3) Block-based Image Difference Method

We implement motion detection by using image difference method, comparing colors of two images pixel by pixel. It requires a lot of computation time, however, when we compare pixel by pixel, so we introduce a block consisting of several pixels. We use a block size of 4 X 4, 8 X 8, and 16 X 16. Each block is marked as "changed block" when the number of changed pixel in the block exceeds some threshold. We call how much the color changed as "color sensitivity," while how many pixels changed (defined as ratio) as "block sensitivity." These two can be customized. For changed block, we assign 1 on a binary difference matrix, and 0 otherwise. Table 1 is an example.

0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	1	1	0	0	0
0	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0

[Table 1] Binary Difference Matrix Example

Image difference can be classified as two categories: image difference between current frame and previous frame, and between current frame and background. This can be set in Environment menu, but we have no choice but to use only the former when we have no background. This is summarized in Table 2.

Image Difference option	Only with background	With previous image
Background registered	Image difference with background image	Image difference with both, then union
No background	Image difference with previous image	Image difference with previous image

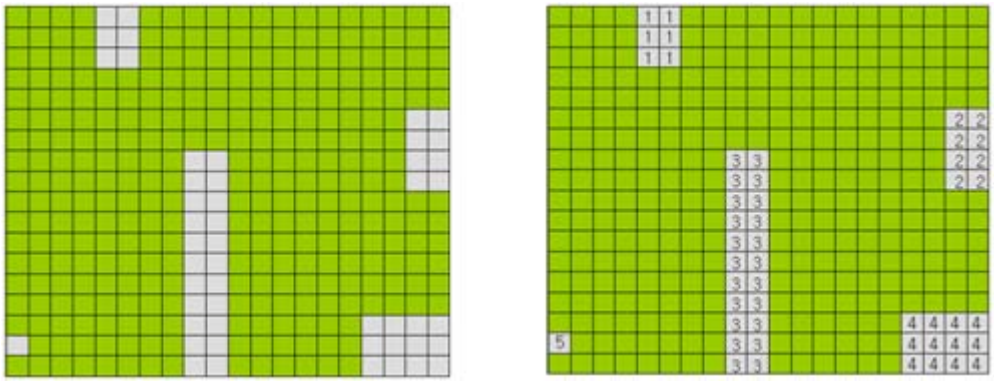
[Table 2] Image Difference Criteria

(4) Labeling

When we finish to build binary difference matrix, we have to distinguish each moving object discovered. By using labeling algorithm, we recognize each object from the binary difference matrix. Size filtering is applied then, in order to ignore non-human small objects. Human-sized objects are then assigned a serial number. Lastly, we set MBR (Moving object Boundary Rectangle; smallest rectangle fully containing the object) for each object and save information about the object. Detailed algorithm is as follows.

1) First Labeling

First Labeling refers a process to recognize adjacent blocks on the binary difference matrix. We first assign a serial number during scanning the matrix from top-left to right-bottom. We use Sequential Connected Blocks Algorithm Using 8-connectivity here. 8-connectivity blocks mean up, down, left, right, left-up, right-up, left-down, and right-down. If blcok [i, j] has binary value 1, we investigate upper block [i, j-1], left block [i-1, j], and left-up block [i-1, j-1]. Then, we assign the smallest object number among them to the block [i, j]. An example is shown in Figure 6.



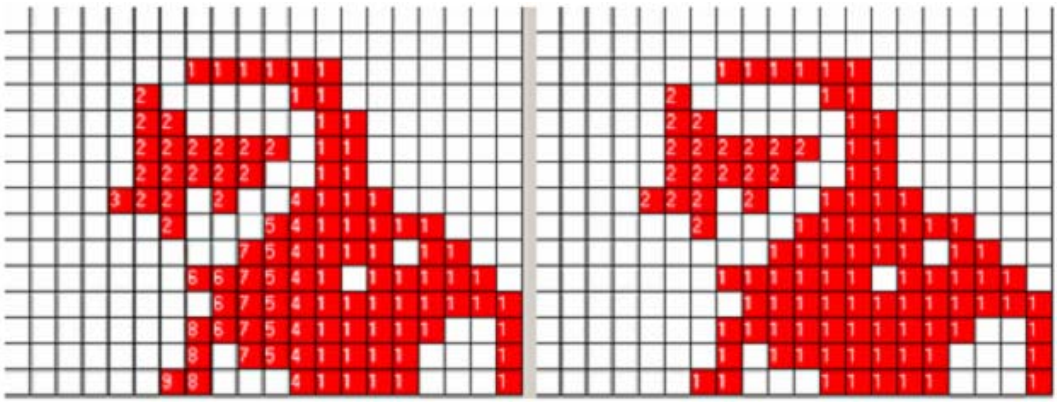
[Figure 6] First Labeling Example

2) Second Labeling

Even though we finish First Labeling, we may have some different label in the same object. Second Labeling aims to remove this inconsistency. In Figure 7, we can see two objects, but nine object numbers. We apply Sequential Connected Blocks Algorithm Using 8-connectivity again, and build a set of 2-tuple containing two adjacent blocks. For Figure 7, the result is:

{(1, 4), (4, 5), (5, 7), (7, 6), (6, 8), (8, 9)} {(2, 3)}

Using these sets, we assign smallest number for each object. That is, we assign number 1 to object 1, 4, 5, 6, 7, 8, 9, and number 2 to object 2 and 3. The result is shown in the right side of Figure 7.

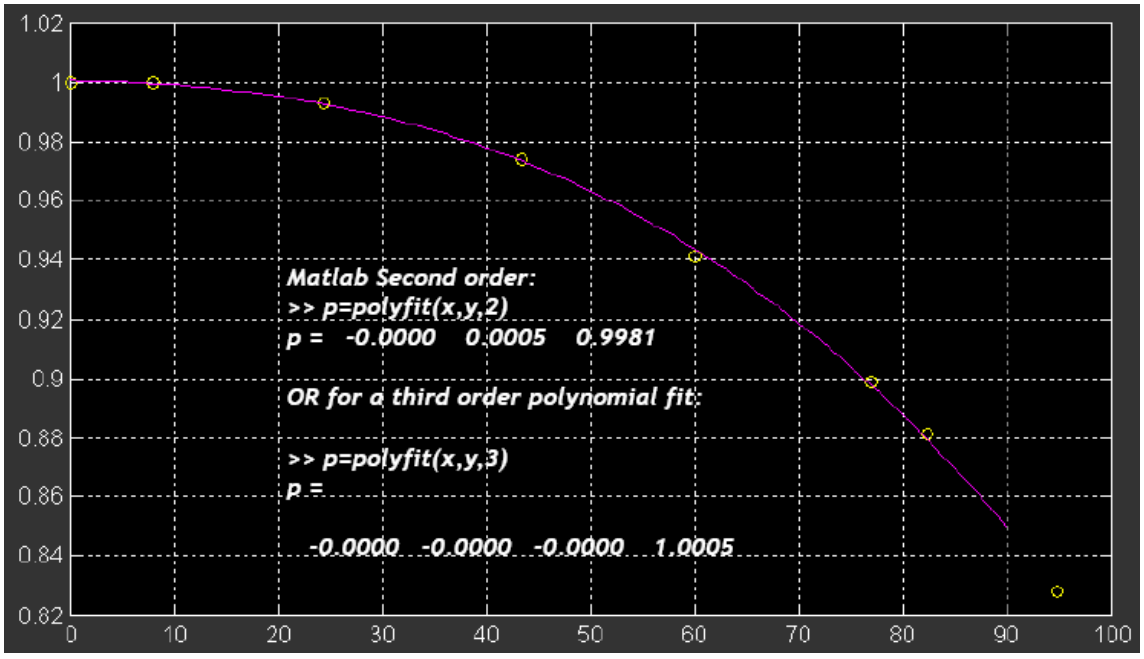


[Figure 7] Second Labeling Example

3) Size filtering

Moving objects discovered on screen can vary in size. Objects smaller than ordinary human size are generally not moving objects, but some rolling caused by a sudden noise or light change. Also, although they are actually moving objects, we can ignore them because they are not people. Therefore, we ignore objects smaller than some predefined threshold. Default value for this is 5 pixels, but customizable.

One thing should be considered here, though. Due to characteristics of wide-angle lens, objects on center will be shown much bigger than those on the outer ring in spite of same size. If the threshold is fixed, we may not be able to recognize people on outer boundary. Therefore, we adjust the threshold value to 4 pixels (80% of center), according to the data shown in Figure 8, which shows how much relative size gets smaller when approaching boundary.



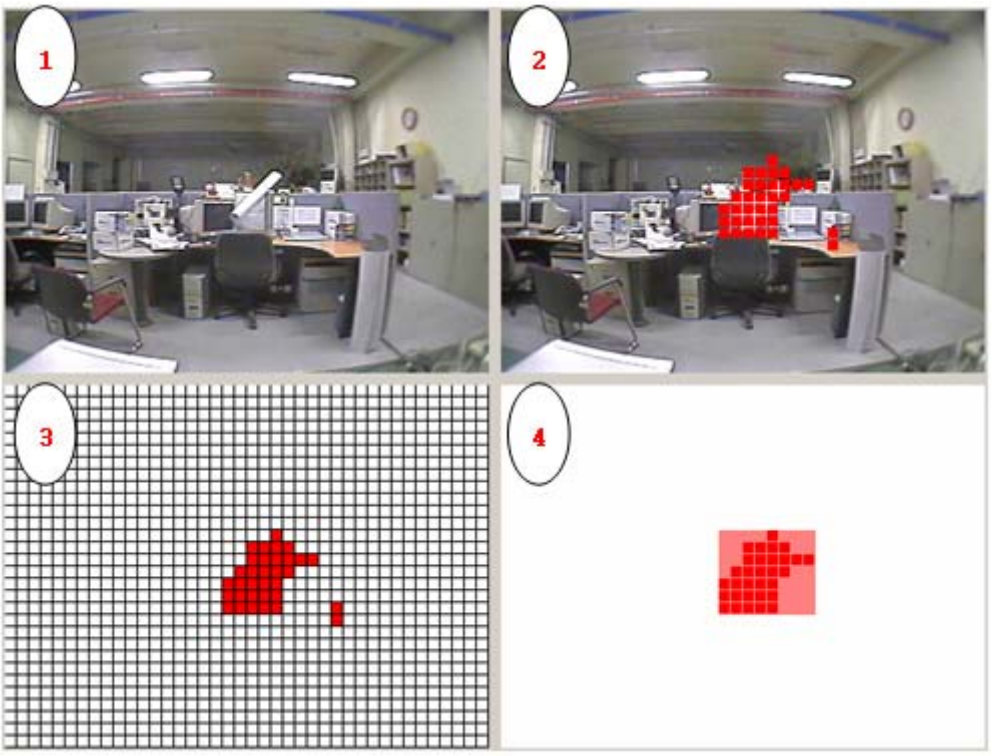
[Figure 8] Tilt angle (x-axis) and relative size (y-axis) on the wide-angle lens used in this system

4) Recognizing objects and building MBR

We assign a serial number to each moving object when we finish Second Labeling and size filtering. We use this number for tracking algorithm including comparison of characteristics, selection of tracking target, and calculation for tracking position. We store information on each object such as location, size (number of changed blocks), and color. This information is called "Characteristics," which will be used for distinguishing each object later.

① Location of the object: We store the location of MBR (the smallest rectangle containing the object).
② Changed blocks: We save coordination of every changed block. This will be used for extracting color and calculating center point of the object.
③ Color of the object: We extract color from center point of all blocks. Average of these color values is a representative color for the object. But color values from background color are excluded because they are not from the object.

Figure 9 shows the overall process. In (1), we can see a thrown paper on the air, without any filtering. In (2) and (3), changed blocks are shown. We can see MBR of this paper in (4). A small object with just two blocks is ignored in (4).



[Figure 9] MBR Example

2. Improved Motion Tracking Algorithm

When more than one moving objects are found, we have to choose one of them for tracking. For this, we first have to distinguish each object. We utilize several characteristics of objects for distinguishing and remembering each object. In this section, we discuss detailed implementation for discriminating each object.

(1) Overall Structure

When moving objects are discovered, we extract some characteristics about them. We store the extracted characteristics temporarily, and compare them with previous objects stored in moving objects list. For each object, we connect it to the most similar one, that is, the one has the most similar characteristics. We use an algorithm discussed in subsection (4) in order to find the best combination for all objects found. If a temporarily stored object finds its partner in stored list, it means they are actually the same object. In this case, we update characteristics with new data. If it could not find its former self, we conclude that it came from outside of the observing range. In this case, we add this into the moving object list.

(2) Characteristic Components

We use the following characteristics for recognizing each moving object:

① Proximity in Location Due to the restriction that human beings cannot move long distance within a second, the location should be similar if two are actually same object.
② Similarity in size Size can bs used a secondary criterion, if both location and color are similar. This can be various even for the same person, because posture or gesture can result in change in disclosed size to the camera.
③ Similarity in color When two objects are located nearly, this criterion is very important. We use RGB color with a 3-dimensional matrix of [0..255][0..255][0..255] for measuring distance of two colors.

We sum those three parameters with the following weight.

Color (40%), Size (15%), Location (45%)

Location is the most important criterion because human ability to move is restricted. (We have 7.5 frames in a second.)

Characteristic points are calculated as follows:

① Color: $x = \frac{(64-p)}{64} \times 100$, where $p = \sqrt{(R_x - R_y)^2 + (G_x - G_y)^2 + (B_x - B_y)^2}$, each variable denote RGB color of two points x and y. That is, a distance within a 3-dimensional color space. When the distance is larger than 64, $x = 0$.)

② Size: $y = \frac{(5-p)}{4} \times 100$, where p is a ratio of size between the two objects. When the ratio is larger than 4, $y = 0$.)

③ Location: $z = \frac{(60-p)}{60} \times 100$, where $p = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, distance between the two objects. When the distance exceeds 60, $z = 0$.)

By summing those three values with predefined weight factor, we get total point. Every moving object discovered each time calculates its similarity with every moving object in the previous objects list. An object with the highest similarity, but higher than 30, is determined as the same object. If this point is lower than 30, which means not so much similar, the discovered object is added as a newly introduced moving object to the list. We used aging strategy for updating characteristics in order to avoid sudden change of characteristic values. (new value 30% + previous value 70%) With this, we can preserve previous values with 30% right before, 21% the second one, 14.7% the thrid one, and 10.3% the fourth one.

(3) Special cases

Moving object is entirely possible that newly enter to the area, disappear from it, overlap each other, or separate again. We assume that at most one of this case can occur at once.

We keep track the number of moving objects in each frame. When it increased, advent or separation may have occurred. When it decreases, disappearance or overlap may have occurred. If no change in the number of objects, nothing may have happened. It is also possible that both advent and overlap or both disappearance and separation might have occurred at once. However, this would be extremely rare, so we ignore this case for simple analysis and implementation.

① When the number of objects increases: If there is a newly found object with more than 30 point with any previous object, we can think of this case as separation. If there is no such object, this can be shown as introduction of new object. In both cases, no special process is needed.

② When the number of objects decreases: In this case, we may have an object which existed in the last frame but not in the current one. For classifying this disappeared object, we first see whether it was located near the boundary of observing range. If yes, we consider this as natural disappearance. If it was located on center, it cannot disappear naturally due to the restriction of human movement ability. In this case, we check other objects near the disappeared one which may overlap it. We can use the same algorithm for this checking, but only one criterion location is used. When we find an overlapping object, we update the detailed data, preparing for the future use when

separation occurs. If nothing is found, it may be ignored due to size filtering. In this case, we do not update anything.

(4) Matching algorithm between two objects

This program repeats detecting, distinguishing, and adding moving objects, with a list of them. Therefore, we need an algorithm matching two objects; one from the previous database (called Moving Entity) and the other one from the current frame (called Temp Entity).

For this, we devise an algorithm as follows. This can be metaphored as a couple matching between male and female. Assume that we have m guys and n girls, and we have to match them with the highest harmony. We consider Mr. and Miss. Right as much as possible, but when competition occurs, the loser may choose the second choice. We also have the minimum cutline (30 point) for matching. If they have no partner with more than 30 point, they will remain as a single. In this system, we can think of this matching as a function Temp Entity → Moving Entity. Domain is Temp Entity, so it should connect to at most and at least one Moving Entity. Now, each Temp Entity competes each other to get better Moving Entity. When two Temp Entities compete for one Moving Entity, the one with higher similarity wins. The loser should try to find other partners. With this strategy, we would choose only one even if the same person is shown as two. When two objects approach and one Temp Entity takes a Moving Entity, the other Temp Entity gives up this, and go to other Moving Entity. This algorithm is shown below:

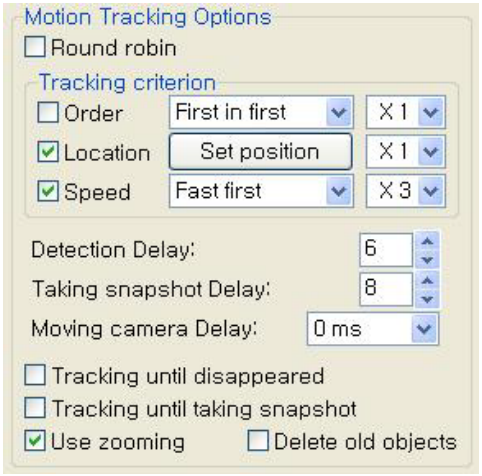
```
finished := false;

while not (finished) do
begin
    for i=1 to TempEntity.length do
    begin
        if (wantToCouple[i]) then // there is a partner with similarity larger than 30
        begin
            if (max[i] > 30) then // it has similarity larger than 30 with any single
            begin
                if (now[i] = 0) then // this is single as well
                begin
                    now[i] := getBestPartner(i);
                    guy := getHerPartner(now[i]); // current spouse of the partner

                    if (guy > 0) then // there is a spouse of the partner
                    begin
                        now[guy] := 0; // take away
                        end[guy] := false;
                    end;

                    end[i] := 0;
                end
                else
                    end[i] := true; // give up, when there is no partner
            end
            else // want to remain single: add
            begin
                registerNewEntity (tempEntity[i]);
                end[i] := true;
            end;
        end;
    end;

    // finishing check:
    finished = true;
    for i=1 to TempEntity.length do
    begin
        if (end[i] = false) then
            finished := false;
    end;
end;
```

3. Multi-object Tracking Algorithm

When we discover several moving objects and finish distinction of them, we have to choose one object for actual tracking. This algorithm contains users' choice as well as technical aspects. In this section, we discuss various kinds of options for motion tracking and its implementation in detail.

(1) Criteria and Options

Figure 10 shows options for selecting tracking target. When "Round Robin" on the top-most is selected, all moving objects caught by sensor will be tracked sequentially, while other options are disabled. We track each ordinary object for 45 frames, but 20 frames for one suddenly appeared and disappeared again. When this time period expires, this process repeats from the sequence number 1 again.



[Figure 10] Tracking Options

We can choose three kinds of tracking criteria when we do not use round robin option. This includes the order that object enters to the observing area, distance between predefined location (a cashbox, for example), and speed of the object. Users may use arbitrary combination of these criteria with some weight factors if needed. We provide 1x, 2x, 3x, and 5x of weight factors.

① Order: We can set priority to older (low sequence numbered) or newer (high sequence numbered) object. For example, we may track the oldest one (which means the object entered first) until it disappears from the observing area. When the object disappears, then we try to find other objects for tracking. In this case, we will continue to track the object when we find the oldest one again in the observing area. (Vice versa, for tracking newest one option.)

② Location: If we set this option, the program tracks an object on the nearest position from the predefined location. This option will be useful if we want to observe some important location such as cashbox. When anyone approaches to the predefined location, it will be tracked with first priority. If other options are not used, we stop tracking when no one is near from the predefined location.

③ Speed: We can give some priority on faster or slower objects. This option will be useful when we want to catch escaping person with fast speed. This speed is calculated by the length of movement vector, the arrow from the previous position to the current position. We also use aging strategy here, with 70% of history of the vector and 30% of current information.

Among those three options, points from ones that user set to use will be summed. For the order, the first one gets 30 points, the second one 20 points, and the third one 10 points. Other objects will not get any point for this. For location, the point is proportional to the proximity with the predefined location. When the distance is longer than 100, no point will be assigned. Otherwise, the object gets (100 - distance) points. For speed, we also grant points linearly proportional to the size of movement vector, when it is smaller than 100. If the length of vector is larger than 100, we judge that it has some errors, because it is impossible that human beings move in that fast speed. We eventually select the object with highest weighted sum of points from the three criteria.

(2) Switching Targets

After 45 frames for tracking one object, we switch to other object. We evaluate all the objects for selecting the next target. We exclude the one we were tracking right before, even though it is still the most favorable target. In this case, we choose the second-highest-pointed object.

However, we may need to exceed 45 frames in some cases. For example, when we were not able to take snapshots within the 45 frames, or when the user sets to track the first object forever until it disappears. In order to fulfill this need, we provide the following options:

① Tracking until it disappears: An option for tracking continuously until when it disappears. When an object is selected as a target once, it will be tracked regardless of its priority after that time.

② Tracking until taking snapshots successfully: Snapshots are captured once per 10 frames. If the camera is moving when taking a snapshot, it is possible not to be able to take a vivid image. If all 4 or 5 pictures within 45 frames are not vivid, tracking time will be extended until getting a clear image with this option.

Following two extra options are provided:

① Necessary time for deciding moving object: Necessary time for an object to be admitted as a moving object by the program. Defined as the number of frames.

② Snapshot delay: Time interval we wait for taking a snapshot.

③ Camera movement delay option: It is unavoidable for the camera to take some delay due to network traffic or hardware movement. This can vary user by user, so we need to set duration for camera control command differently. We provide 0ms, 150ms, 300ms, 550ms, and 800ms. Because we have one frame per each 133ms, each duration corresponds to everytime, once in twice, once in three times, once in four times, and once in six times.

(3) Taking Snapshots

While tracking moving objects, we store snapshots periodically. Specification is as follows:

* Size: 320 × 240 pixels
* Interval: 10fps, once in each 1.3 second
* Form: Jpeg (.jpg)
* File name: xxx-hhmmss-yyy.jpg

xxx: serial number of the target

hhmmss: time (hour-minute-second)

yyy: time stamp (mili-second)

* At most 15 snapshots are shown on the screen.

Figure 11 shows snapshot-taking example:

[Figure 11] Snapshot Example

4. PTZ Camera Control

When we finish selecting a target, we then need to find its coordination and to send it to the camera. In this section, we discuss calculating center point of the target, correction method for falling behind effect, converting coordinate from sensor to PTZ camera, and panorama image implementation. This part is specific to our sensor CCD module and PTZ camera.

(1) Center point Calculation and Correction

We are storing four vertex of MBR as location of each object. When we command camera to track an object, we have to one point, not a range of MBR. So, we need to calculate the center point. This can be calculated simply by averaging left and right for x-axis and top and bottom for y-axis, but this is imprecise because not every block in MBR is actually part of the object. Therefore, we calculate the center of gravity of each object. We saved list of blocks which were changed, so we can consider these blocks as part of the moving object. By calculating average of x-axis and y-axis of these blocks, we can get center of gravity of this object. This is more precise than simple average of every block.

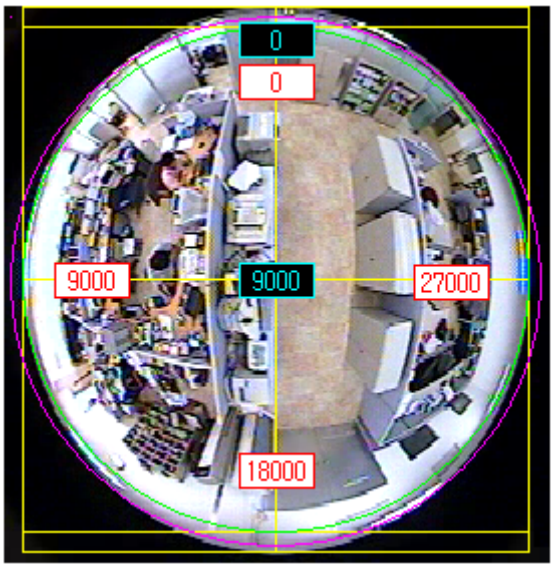
We have to consider one thing here, however. Moving camera is a mechanical operation, so this takes considerable time compared to electronic calculation. Therefore, it is entirely possible that the object is not there when the camera arrives to that position. So, we suggest the following process in order to solve this problem.

Beside the calculation of coordination, we estimate its possible location using its average moving speed and direction. We give this estimated value, not the current location, to PTZ camera. For each frame, the target calculates its estimated location after some duration using the movement vector and its history of movement. We use its history 40% and current information 60%. This allows us to gradually update direction and speed of the object when they change.

(2) Mapping from Sensor to PTZ coordination

Next work is sending coordination of the center to PTZ camera for tracking. However, we are using Cartesian coordinate on sensor image, while we need actual Pan and Tilt value, which use Polar coordinate, on PTZ camera. Therefore, we need to convert given Cartesian coordinate into Polar coordinate, and multiply a constant. This part can be different system by system.

In our system, range of Pan value is between 0 and 35999, denoting 0° to 360° times 100. For tilt, it ranges from 0 to 9000, but this is not simple to calculate. Even though Tilt value is expressed as angle(ϕ) of Polar coordinate, we know only the distance(r) because the input image is flat. Thus, we have to draw a formula between the distance value and Tilt value. We estimate this by using the following formula for calculating Pan and Tilt. Figure 12 shows Pan and Tilt coordination on the input image from the sensor. White colored one is Pan, while Black one is Tilt.



[Figure 12] Pan, Tilt coordination

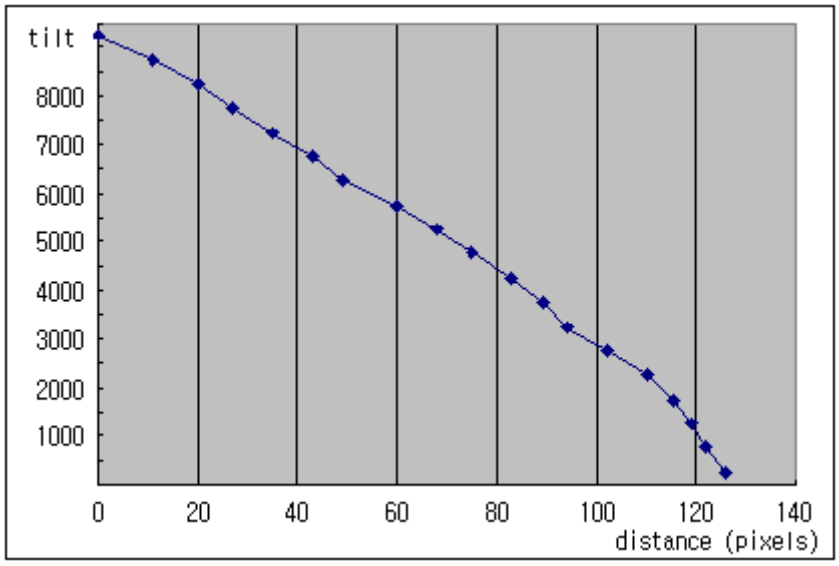
- Cartesian coordination → Spherical coordination: $r = \sqrt{(x^2 + y^2)}$, $\theta = \tan^{-1}\frac{y}{x}$
- Spherical coordination → Cartesian coordination: $x = r\cos\theta$, $y = r\sin\theta$

Note that we use 100θ for Pan value.

Quadrant 1: $pan = \tan^{-1}\frac{y}{x} \times \frac{18000}{3.14} + 27000$ Quadrant 2: $pan = \tan^{-1}\frac{x}{y} \times \frac{18000}{3.14}$

Quadrant 3: $pan = \tan^{-1}\frac{y}{x} \times \frac{18000}{3.14} + 9000$ Quadrant 4: $pan = \tan^{-1}\frac{x}{y} \times \frac{18000}{3.14} + 18000$

$tilt = -69.48 \times \sqrt{(x^2 + y^2)} + 9429.17$ (Gathered by linear regression. See Figure 13.)



[Figure 13] Tilt values for each distance

We assume that the image from sensor is a perfect circle. In fact, however, this is an ellipse whose vertical axis is a little bit longer than its horizontal axis by 20 pixels. So we cannot avoid some errors. We map by assuming a circle and using longer axis as radius, so Tilt values of shorter axis will have some errors. In order to correct this, we subtract $\sin x$ from 90° and 270° part, shorter axis direction. However, we need a graph inverse of $\sin x$, because errors get larger when approaching to shorter axis. So, we use $\sin^2 x$, and draw the following equation:

$$tilt = -69.48\sqrt{(x^2+y^2)} + 9429.17 - 200\sin(pan \times \frac{3.141592}{18000})^2$$

The first and second term is previous Tilt formula, and the last term is correction part with $\sin^2 x$ form. We multiply 200 in order to move the camera downward about 2° when Tilt is 90°. (These values are adjusted by experiment.)

Lastly, we discuss about Zoom. Our PTZ camera can optically zoom 26x at most, but it is appropriate to zoom about 2 to 4 times for taking human-sized objects. We classify moving objects into the following three categories according to its moving speed:

① Moving within Pan 3° and Tilt 2° (Type 1): We can take a vivid image even with a high zoom. In this case, we zoom up to 4x according to the distance from the center.

② Moving within Pan 6° and Tilt 3.5° (Type 2): We can take a vivid image only when we give low zoom. In this case, we give up to 2x zoom only when it is too far.

③ Moving faster than Type 1 or 2 (Type 3): We do not zoom at all.

Refer to Table 3, for detailed specification:

Distance (pixels)	0 ~ 85	86~115	116~
Type 1	**2x**	**3x**	**4x**
Type 2	1x	1x	**2x**
Type 3	1x	1x	1x

[Table 3] Zoom Criteria

(3) Panorama Image

We provide a rectangular panorama image. In this image, horizontal axis and vertical axis correspond to Pan and Tilt, respectively. Figure 14 shows its example:

[Figure 14] Panorama Image Example

We can get a panorama image with the following algorithm. Horizontal axis corresponds to Pan and vertical axis to Tilt, regardless of camera used, because this is only about the image. We also assume that the image from the sensor is a perfect circle, although actually not. We convert this original image from the sensor, which can be seen as a Polar coordinate, into Cartesian coordinate. We can use the following formula:

$$r = \sqrt{x^2 + y^2} \qquad \theta = \tan^{-1}\frac{y}{x}$$

Then, we map the r and θ values to their corresponding points on the panorama image by multiplying its size. This conversion runs only once. For every frame, each point will find its color with the pre-calculated coordinates from the sensor image. Therefore, this takes O(1) time.

We also provide a manual tracking using this panorama image. By disabling automatic motion tracking and by clicking a point on the panorama image, PTZ camera moves to the point. Pan and Tilt value will be shown on the top-most panel. You may also move the camera by giving exact Pan and Tilt coordination on the panel and press the "Move" button.

On the panorama image, you can see a yellow box, denoting the position it is currently tracking.

III. Conclusion

In this report, we propose an efficient real-time motion tracking method using a wide-angle lens and PTZ camera. We implement fast and noise-tolerant motion detection by introducing blocks and applying filters. With a simple, but strong block-based preprocessing and background update, we dramatically improve tracking speed and accuracy. For an environment with multi-objects, we also make it possible to distinguish each moving object with its characteristics and to track one of them according to the user's criteria.

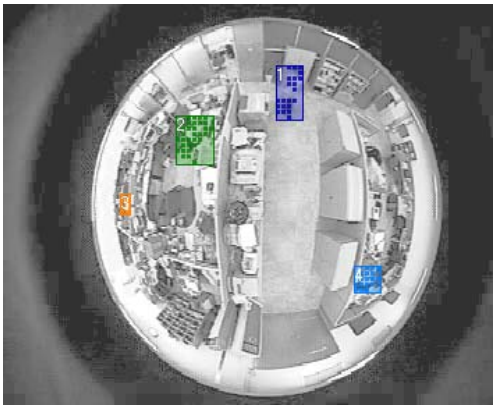
In this chapter, we discuss our achievement on this project, by dividing it into four categories. Each experiment is the average from results of ten repetitive experiment.

① Block-based motion detection algorithm: previous level 30% → objective 95%

Figure 15 shows an example of motion detection with four people. We can see the person 1 is recognized as one object although blocks in it are not connected. Also, we can catch relatively small object like number 3 and 4, by applying distance correction.

Detection operates precisely in almost every case, but it fails to detect some when it stops moving and registered as background. We test and estimate this error with the following experiment.

Experiment: Save every detection result of 300 consecutive frames. Regardless of the number of MBR (this is important for other criteria, but ignore here because we do not consider distinguishing), we check whether it is discovered or not. In Table 4, we provide part of this table. (White background color means precise operating, while yellow means errors.) In this table, A and B means accurate detection, but F means missing the object. For continuous F, we can consider it as permanent disappearance, so it's okay. For the yellow-colored case, however, we did not detect the object in spite of its presence.



[Figure 15] Detecting moving objects

Result: For all the 300 frames,

— Precise detection for existing objects (White A, B): 825 times

— Precise ignorance for non-existing objects (White F): 539 times

— Missing detection for existing objects (Yellow F): 67 times

→ Error rate: $\frac{67}{825+539+67} \times 100 = 4.682\%$

Accuracy: $100 - 4.682 = 95.318\%$

According to the result, we confirm that we achieved our objective, 95% of accuracy. Error cases include wrong update for background image, hiding effect of objects resulting in smaller size, or too small size when it locates on boundary.

② Distinguishing objects using characteristics

: previous level 0% → objective 70%

We conducted the following experiment for checking accuracy of distinguishing moving objects.

Experiment: We also store motion detection result of 300 consecutive frames, and calculate the ratio of frames which were distinguished correctly. In Table 4, objects marked as F are excluded from this experiment because they are not observed in that frame. "A" means that it holds the same sequence number (succeeded in distinction), while "B" means that it changed its number although it was detected (failed in distinction).

Result: For all the 300 frames,

— Succeeded in distinction (A): 576 times

— Failed in distinction (B): 249 times

→ Accuracy: $\frac{576}{576+249} \times 100 = 69.818\%$

We can conclude that distinguishing each object works correctly about 70% of accuracy. Failing to distinguish objects may be caused by overlapping each other, hidden by obstacles, or unexpected change in direction. When we improve these special cases, it may show much better performance.

Frame	1	2	3	4
27	A	A	F	A
28	A	A	F	A
29	A	A	F	A
30	A	A	F	A
31	A	A	F	A
32	A	A	F	A
33	A	A	F	A
34	A	A	F	A
35	A	A	F	A
36	A	A	F	A
37	A	A	F	A
38	A	A	F	A
39	A	A	F	A
40	A	A	F	A
41	A	A	F	A
42	A	A	F	A
43	A	A	F	A
44	A	A	F	A
45	A	B	F	A
46	A	A	F	A
47	A	A	F	A
48	A	F	F	A
49	A	F	F	A
50	B	F	B	A
51	B	B	B	F
52	F	A	A	F
53	B	A	A	F
54	B	A	A	F

[Table 4] Test result for motion detection and distinguishing

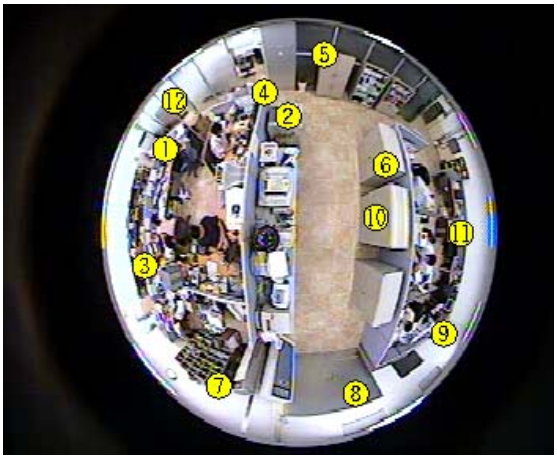
③ Multi-object tracking algorithm: previous level 0% → objective 60%

This program implements basic strategy for multi-object motion tracking, but we may need to provide more options for users to customize themselves. At least, options which already implemented work correctly. When we want to observe all objects equally, we may use "Round Robin" option. In this case, we can see all objects perfectly. However, it can fail to track some objects staying in the observing area shortly. On the other hand, when we set to track objects near some point, we may not track all objects.

This part is difficult to conduct an experiment, as well as highly depends on motion detection (subsection ①) and characteristic distinction (subsection ②). Assuming that multi-object tracking algorithm works perfect, we can estimate overall accuracy using result of the previous two experiments. It gives $95.318 \times 69.818 \div 100 = 66.549\%$. Thus, we have achieved our objective 60%. (We can judge 100% success in tracking only by seeing actual animation, so we skip this here. We will show this on the conference.)

④ Controlling PTZ camera: previous error rate 30% → objective 10%

Figure 16 shows some points with special landmarks in the observing area. Point 1 is center of monitor, and point 2 is center of copying machine. Point 3 and 4 are turning points of partitions. Point 5, 6, and 7 are boundary of cabinets. Point 8 is the name of factory, and point 9 is center of window. Point 10 is a handle of cabinet, and point 11 is a file of documents. Point 12 is center of a chair in the room. Figure 17 lists the screen of these points, with x2 zoom. We can see that targets are located on center of the screen.



[Figure 16] Pan/Tilt test location

[Figure 17] Pan/Tilt Test Result

In order to calculate this more precisely, we conducted the following experiment.

Pan Error Estimate: We may have no error for the case of Pan, if the sensor image is a perfect circle. Unfortunately, it is an ellipse whose vertical axis is a slightly longer than its horizontal axis. We assume that this is a circle, and due to this assumption, we have some error. We conducted an experiment in order to estimate this error.

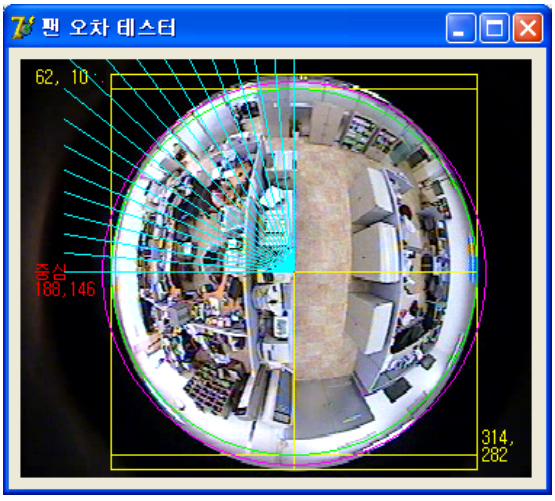
On a sensor image like Figure 18, we find intersections of each 5° angle line and both circle and ellipse. When we subtract coordinate of the center from these values and calculate $\frac{y}{x}$, this is $\tan\theta$. We can calculate pan value (100θ) from this by using the formula $\tan^{-1}\frac{y}{x}=\theta$. This result is summarized in Table 5.

Average of absolute value of errors in Table 5 gives 13.61. Because we use only one quadrant [0..9000], the overall error rate is given by

$$\frac{13.61}{9000}=0.15\%,$$

and accuracy is

$$100-0.15=99.85\%.$$



[Figure 18] Pan Error Tester

Angle	Circle crd.		Ellipse crd.		Circle $\frac{y}{x}$	Ellipse $\frac{y}{x}$	Circle pan $(100\tan\theta)$	Ellipse pan $(100\tan\theta)$	Error
	x	y	x	y					
0	-126	0	-126	0	0.00	0.00	0.00	0.00	0.00
5	-125	-10	-125	-10	0.08	0.08	457.39	457.39	0.00
10	-124	-20	-124	-20	0.16	0.16	916.23	916.23	0.00
15	-122	-31	-122	-31	0.25	0.25	1425.70	1425.70	0.00
20	-119	-41	-120	-42	0.34	0.35	1901.08	1929.01	-27.93
25	-115	-51	-116	-51	0.44	0.44	2391.63	2373.29	18.33
30	-111	-61	-112	-62	0.55	0.55	2879.10	2896.77	-17.67
35	-105	-70	-107	-71	0.67	0.66	3369.01	3356.63	12.37
40	-98	-78	-101	-81	0.80	0.80	3851.69	3872.89	-21.20
45	-91	-86	-95	-90	0.95	0.95	4338.19	4345.19	-6.99
50	-83	-94	-87	-98	1.13	1.13	4855.62	4840.28	15.34
55	-75	-101	-78	-106	1.35	1.36	5340.34	5365.26	-24.92
60	-65	-107	-69	-113	1.65	1.64	5872.23	5859.10	13.13
65	-55	-113	-59	-120	2.05	2.03	6404.66	6381.82	22.84
70	-45	-118	-48	-126	2.62	2.63	6912.54	6914.56	-2.02
75	-34	-121	-36	-130	3.56	3.61	7430.51	7452.14	-21.63
80	-23	-124	-25	-133	5.39	5.32	7949.20	7935.44	13.76
85	-12	-125	-12	-135	10.42	11.25	8451.64	8492.04	-40.40
90	0	-126	0	-136	∞	∞	9000.00	9000.00	0.00

[Table 5] Pan Error Estimate

Tilt Error Estimate: For Tilt value, we use a regression test as in Figure 13. Points in Figure 13 means tilt coordination from center of the screen, so we can get a general formula for tilt which is given by

$$tilt = -69.48 \times \sqrt{(x^2 + y^2)} + 9429.17.$$

Therefore, we can figure out error rate by calculating its correlation coefficient r and determination coefficient r^2, which are given by

$$r = \frac{S_{(xy)}}{\sqrt{S_{(xx)}S_{(yy)}}}, \quad r^2 = \frac{S_{(xy)}^2}{S_{(xx)}S_{(yy)}}$$

where, $S_{(xy)} = \sum_i x_i y_i - \frac{\sum_i x_i \sum_i y_i}{n}$, $S_{(xx)} = \sum_i x_i^2 - \frac{(\sum_i x_i)^2}{n}$, $S_{(yy)} = \sum_i y_i^2 - \frac{(\sum_i y_i)^2}{n}$

Correlation coefficient and determination coefficient for the Tilt formula are

$$r = -0.9951,$$
$$r^2 = 0.9902.$$

Determination coefficient ranges in $0 \leqq r^2 \leqq 1$. When it approaches to 1, it means the regression line is precise and useful. Because determination coefficient for this experiment is 0.9902, we can say that accuracy of Tilt is about 99%.

Considering accuracy of both Pan and Tilt, we can conclude that error rate for both Pan and Tilt is about 1%. This means we excessively accomplished our objective, 90% of accuracy.

Table 6 shows overall achievement of this research, including previous level of our company, objective of this research, and actual achievement of this project.

Evaluation Criteria	Previous Level	Objective	Achieved
Block-based motion detection algorithm	30%	95%	**95.3%**
Distinguishing objects using characteristics	0%	70%	**69.8%**
Multi-object tracking algorithm	0%	60%	**66.5%**
Controlling PTZ camera	70%	90%	**99.4%**

[Table 6] Project Evaluation Criteria

We propose that we achieved our objective in all over the criteria. User interface has also improved by implementing various kinds of options, manual tracking, panorama images, and taking snapshots. In the future, taking an animation rather than snapshots may be more useful.

Reference

<Image Difference, Motion Detection, Preprocessing>

[1] Kiyeon Lee, A Study on Real-time Object Tracking System using Omni-directional Vision, 2005, Seoul National University of Technology

[2] Young Su Yang, Motion Detection using Improved Image Difference Analysis, 2000, Injae University

[3] Seonghee Ryu, Real-time Implementation of Moving Object Detection and Tracking for Unmanned Observing System, 1999, Suwon University Graduate School

[4] Yuri Kang, Motion tracking system with Improved block-basd Image processing, 2001, Korea University

[5] Hao Zhang, Butterworth Filtering and Implicit Fairing of Irregular Meshes, Univ. of Toronto

<Distinguishing objects using characteristics>

[6] Yunho Kim et el., Extracting Geometrical Characteristics of Moving objects, 1991, Mokwon University Department of Electrical and Electronic Engineering

[7] Jong-myeon Jeong et el., Movement Analysis for Motion Tracking, 1998, Hanyang University

[8] Yoonchang Lee, Implementation of Object Tracking System using Characteristics, 2001, Jeonnam University Computer Science Department

<Image Filtering, Panorama Image>

[9] Sang-Hun Jeong, Geometrical revision of Images from Wide-angle Lens, 2003, Seoul National University of Technology

[10] Mikyeong Kim, Panorama VR Objects Creation using Onmi-directional camera and Java3D, 2003, Busan University

[11] Steve Roberts, Digital Signal Processing B4 Option lectures

[12] Kenji Tanaka et el., A method for panoramic stereo image acqusition, Univ. of Tokyo

[13] Joshua Gluckman et el., Real-Time Omnidirectional and Panoramic Stereo, Department of Computer Science, Univ. of Columbia

* Note that some of references were written in Korea, so English translation of titles may be different.