Chapter 2.

Real-Time Distance-Scanning System

2.1 System Overview

Because it is tedious and inefficient to take several pieces of pictures from space and proceed off-line processing for image mosaicing, in this chapter we present an efficient panorama mosaicing system by using a video camera, and the developed system is called real-time distance-scanning system. In this system, the manifold projection surveyed in subsection 1.3.1 is employed for modeling it, and the winner-update strategy for fast block matching is used to speed up the estimation of camera motion to achieve real-time performance.

A flowchart of the proposed real-time distance-scanning system is described in Figure 2.1. It starts with the capture of a color image of the scene from a video sequence by using USB or IEEE-1394 cameras. All of the captured color images are transformed from RGB model to YIQ model, and only the luminance plane, Y-plane, is used for the following procedures. First, the Y-plane of the first captured frame is saved in a buffer and then go back to capture the next frame. The Y-planes of the other captured frames are also saved in a buffer for motion estimation. If the captured frame is not the first frame in the video sequence, then the central vertical area of the Y-plane of this frame is divided into several blocks. The Sobel operator [4] is applied on all of the divided blocks to calculate the individual edge intensity inside, and only the block whose edge intensity is greater than a threshold is chosen to estimate the motion vector. In the stage of motion estimation, a fast block matching technique called the winner-update strategy proposed by Chen, Hung, and Fuh [2] is used in this research. Finally, all of the estimated motion vectors are separated into three groups, and only the mean vector of the middle group is considered as the final estimated motion vector between this captured image and its previous captured image in order to eliminate the affection of outliers. In the image mosaicing step, the width of a cut stripe and its paste position are both decided by the estimated motion vector. Details of each step are introduced in the following subsections.

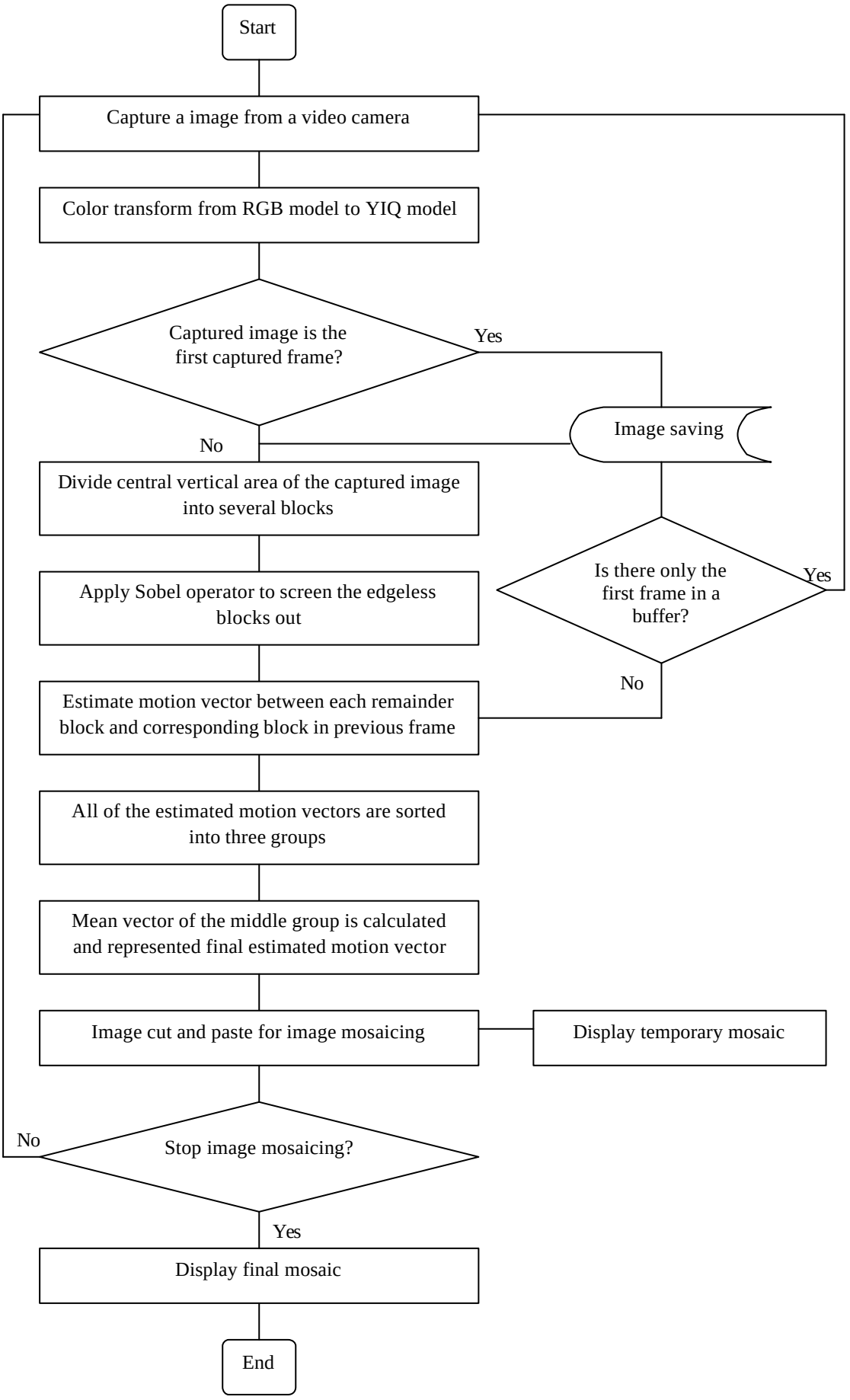

Figure 2.1 Flowchart of the proposed real-time distance-scanning system.

2.2 Image Acquisition and Pre-Processing

In this research, Logitech Quick Pro 3000 USB camera and Sony VFW-500 IEEE-1394 camera are our input devices. The video frame rate of these two equipments is 30 Hz, which limits the maximum frame rate and the minimum latency time of this system.

After capturing the color image from a video camera, some image pre-processing operations will be performed first. All of the captured color images are transformed from RGB model to YIQ model by using the following formula [4]:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}, \qquad \text{(Eq. 2.1)}$$

where the *Y*-component is the NTSC luminance and is considered as a gray-scale image in this research, whereas *I* and *Q* preserve color information of the image. All of the transformed *Y*-component images are carried into follow-up operations and saved in a buffer for later motion-estimation use. Besides, the first acquired image is selected as the reference frame for this real-time distance-scanning system.

If the captured image at time *t*, denoted as I_t, is not the first acquired image, it is necessary to estimate the camera motion between the acquired images I_t and I_{t-1} for image mosaicing.

In this research, we assume that the camera motion consists translations in both *x* and *y* directions. Before estimating the motion vector of the camera, a selected central vertical region of the captured image I_t is divided into two column block sets with total *n* squared blocks, as shown in Figure 2.2. Because only vertical stripes closest to the center of images are combined to build a panoramic mosaic, it is a better choice to use central vertical region to estimate camera motion. On the other hand, the maximum number of the divided blocks, *n*, is decided by the selected block size and the image resolution. For example, given a 320 $\times$ 240 image and the selected block size is 32×32 pixels, then the central vertical region can be separated into two column block sets and each column block set has 7 blocks. Figure 2.2 illustrates 14 divided blocks of this example.

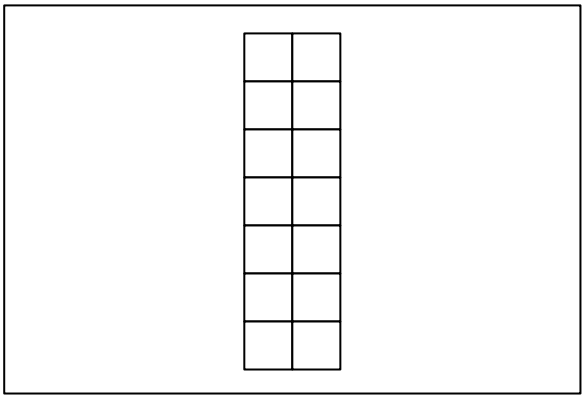

Figure 2.2 A 320×240 image can be separated into 14 blocks whose size is 32×32 pixels.

The fact that more edge intensity in a block, more accurate estimated motion vector can be. The Sobel operator [4] is applied to these n blocks to calculate the edge intensity in each block and a threshold is selected to screen the edgeless blocks out. Finally, only n' ($n' \leq n$) blocks containing more edge inside are remained.

An extremely awful situation is happened when $n' = 0$. That is, all of these n blocks are edgeless. In this case, a block closest to the center of the captured image is enforced to execute the follow-up motion estimation procedure and n' is set to 1.

2.3 Motion Estimation

In this system, a fast block matching technique, winner-update strategy [2], is adopted to estimate motion vectors of these n' blocks. The concept and algorithm of the winner-update strategy are reviewed in this subsection. This fast algorithm is very helpful for achieving real-time distance-scanning.

In this system, the motion vector is estimated by using block matching technique which minimizes the matching error. The matching error between each block at position (x, y) in the current image, I_t, and the candidate block at position $(x+u, y+v)$ in the previous captured image, I_{t-1}, is usually defined as SAD (Sum of Absolute Difference):

$$SAD_{(x,y)}(u,v) \equiv \sum_{j=0}^{B-1}\sum_{i=0}^{B-1}\left|I_t(x+i,y+j) - I_{t-1}(x+u+i,y+v+j)\right|, \qquad \text{(Eq. 2.2)}$$

where the block size is $B \times B$. The best estimation of the motion vector, $(\hat{u},\hat{v})$, is defined as:

$$(\hat{u},\hat{v}) \equiv \arg\min_{(u,v)\subset S} SAD_{(x,y)}(u,v), \qquad \text{(Eq. 2.3)}$$

where S is the search region specified in this work.

The exhaustive search (or full search) requires a large amount of computation and is not practical for real-time distance-scanning. By employing the winner-update strategy, the efficiency of block matching can be strengthened.

2.3.1 Concept of Winner-Update Strategy [2]

We illustrate the concept of winner-update strategy with a simple game of poker cards. Suppose there are five players in a poker game and each player is dealt four cards. The sum of the values of his cards is the penalty score of each player and the player with the minimum penalty score is the winner in this poker game. The basic idea is that each player does not have to calculate the summation of all four cards in his hand for determining the winner. If the intermediate summation for one player is greater than the total penalty score of the winner, then there is no chance for this player to win the game. Thus we can stop calculating his penalty score to save computation time. Figure 2.3 is an example for clarifying this idea.

Of course, we do not know the penalty score of the winner in advance. But the following winner-update strategy can help us to deal with this problem. First, all cards but the first cards of players are laid the faces down. The value of the first card is the lower bound of the penalty score for each player. Only the temporary winner who has the minimum lower bound is permitted to turn up the face of the next card and update intermediate lower bound of the penalty score. A new temporary winner is then selected again and the process is iterated until the temporary winner has no card to turn up the face and becomes the final winner. The executing procedures of the example given in Figure 2.3 are listed step by step in Table 2.1.

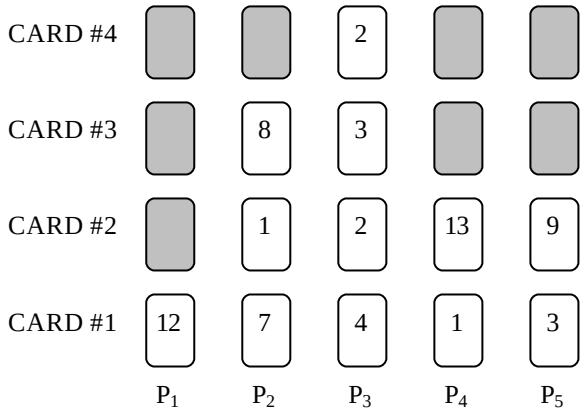

Figure 2.3 A simple game for illustrating the concept of winner-update strategy.

Table 2.1 Step by step calculations of the penalty score of each player.
The boldface score numbers are the temporary winners after each iteration.

operation	P_1	P_2	P_3	P_4	P_5
initialize	12	7	4	**1**	3
turn up CARD #2 of P_4	12	7	4	**14**	3
turn up CARD #2 of P_5	12	7	4	14	**12**
turn up CARD #2 of P_3	12	7	**6**	14	12
turn up CARD #3 of P_3	12	7	**9**	14	12
turn up CARD #2 of P_2	12	**8**	9	14	12
turn up CARD #3 of P_2	12	**16**	9	14	12
turn up CARD #4 of P_3	12	16	**11**	14	12
choose P_3 as the winner	12	16	**11**	14	12

This strategy is a special case of the branch-and-bound strategy with one branch for each node except the root node, as shown in Figure 2.4.

2.3.2 Winner-Update Algorithm for Block Matching [2]

The lower bounds derived from Minkowski's inequality can be used in the winner-update algorithm for fast block matching. This kind of lower bound is proposed by Lee and Chen in their BSP (Block Sum Pyramid) algorithm [14] and the number of the lower bounds for each search position is $\log_2 B+1$, where the block size is $B \times B$.

Assuming that the size of the matching block is $2^K \times 2^K$, we can construct K + 1

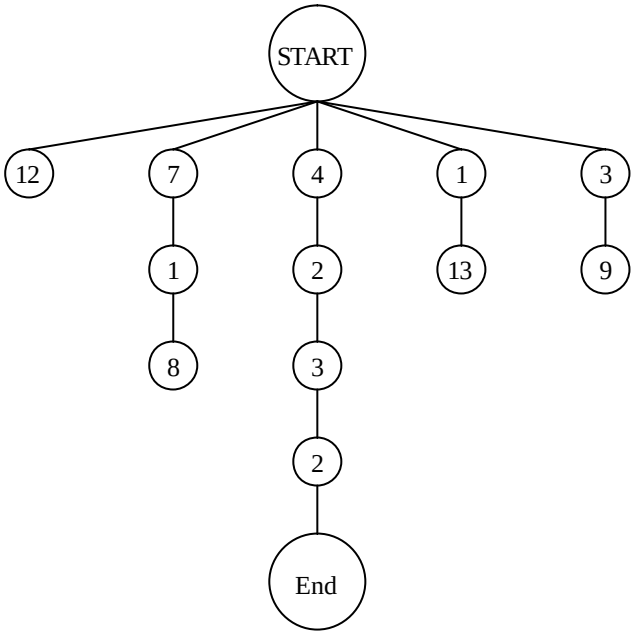

Figure 2.4 The winner-update strategy is a special case of the branch-and-bound strategy with only one branch for each node except the root node, "START". This figure illustrates the search procedures as Table 2.1.

levels of images by using Eq. 2.4. Each pixel, $I_t^l(x,y)$, at level l and at time t can be calculated by summing up its four corresponding pixels at level $l + 1$.

$$I_t^l(x,y) = I_t^{l+1}(x,y) + I_t^{l+1}(x+2^{K-l-1}, y) + I_t^{l+1}(x, y+2^{K-l-1}) + I_t^{l+1}(x+2^{K-l-1}, y+2^{K-l-1}), \quad \text{(Eq. 2.4)}$$

where l is the level number from 0 to $K - 1$ and the original image at time t is I_t^K. We can construct a block-sum pyramid for all the positions, $I_t^l(x,y)$, from level $l = 0$ to $K - 1$ by using Eq. 2.4. A four-level example of the block sum pyramid is illustrated in Figure 2.5.

Let LSAD (Layered Sum of Absolute Difference) of each level be the matching error between the template block at position (x, y) in image I_t^l and the candidate block at position $(x+u, y+v)$ in image I_{t-1}^l. LSAD is defined by Eq. 2.5:

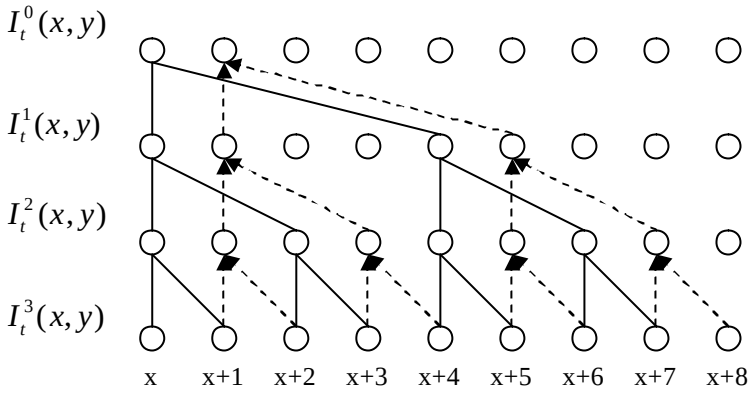

Figure 2.5 The construction of four-level sum pyramid images in one dimension. Two sum pyramid images at position x and $x + 1$ are described in solid line and dash line, respectively.

$$LSAD^{l}_{(x,y)}(u,v) \equiv \sum_{j=0}^{2^{l}-1}\sum_{i=0}^{2^{l}-1}\left|I_t^{l}(x+2^{K-l}i, y+2^{K-l}j) - I^{l}_{t-1}(x+u+2^{K-l}i, y+v+2^{K-l}j)\right|,$$

(Eq. 2.5)

where $l = 0, \ldots, K$. By using the Minkowsky's inequality, the following inequalities can be derived [2]:

$$SAD(u,v) = LSAD^{K}(u,v) \geq LSAD^{K-1}(u,v) \geq \cdots \geq LSAD^{1}(u,v) \geq LSAD^{0}(u,v). \quad \text{(Eq. 2.6)}$$

As a result, LSADs can be used as lower bounds of SAD and we call this kind of lower bounds as the pyramidal lower bounds. The computational saving, for the most part, is enormous. However, if all of the candidate blocks are very similar to the template block, it wastes of time to calculate and compare all of $LSAD^{l}(u,v)$. Fortunately, this condition occurs rarely in practice.

2.3.3 Robust Motion Vector Estimation by Using Projected Median

There are n' remainder blocks after eliminating edgeless blocks and the winner-update strategy is performed to estimate individual motion vector of each remainder block. However, outliers of estimated motion vectors may be included by

simply using block matching for motion estimation. In order to remove outliers, all of these *n'* estimated motion vectors are projected onto *x* axis and *y* axis, respectively, and then the projected points in each axis are sorted from the smallest to the largest. The symbols, dx_i and dy_i, $i = 1, \ldots, n'$, are used to denote the projected points from the smallest to the largest in both *x* axis and *y* axis, respectively, and the three groups of the projected points in *x* axis are composed by $dx_1 \sim dx_{\lfloor n'/3 \rfloor}$, $dx_{\lceil n'/3 \rceil} \sim dx_{\lfloor 2n'/3 \rfloor}$, and $dx_{\lceil 2n'/3 \rceil} \sim dx_{n'}$, and in *y* axis are composed by $dy_1 \sim dy_{\lfloor n'/3 \rfloor}$, $dy_{\lceil n'/3 \rceil} \sim dy_{\lfloor 2n'/3 \rfloor}$, and $dy_{\lceil 2n'/3 \rceil} \sim dy_{n'}$, respectively.

Clearly, outliers are usually lying in the first and the third groups. Thus, only the projected points in the middle group are adopted to calculate the projected mean vector, (*Dx*, *Dy*), in *x* and *y* axes by using Eq. 2.7:

$$Dx = \frac{\sum_{i=\lceil n'/3 \rceil}^{\lfloor 2n'/3 \rfloor} dx_i}{\lfloor 2n'/3 \rfloor - \lfloor n'/3 \rfloor}$$
$$Dy = \frac{\sum_{i=\lceil n'/3 \rceil}^{\lfloor 2n'/3 \rfloor} dy_i}{\lfloor 2n'/3 \rfloor - \lfloor n'/3 \rfloor} \qquad \text{(Eq. 2.7)}$$

Finally, the projected mean vector, (*Dx*, *Dy*), is considered as the refined motion vector between image I_t and the previous captured image I_{t-1}.

For example, if *n'* = 9, we estimate individual motion vector of each block by using winner-update strategy block matching technique, and these *n'* estimated motion vectors are projected onto *x* and *y* axes as shown in Figure 2.6. In each axis, the projected points of estimated motion vectors are separated into three groups, R_1, R_2, *and* R_3, from the smallest to the largest as illustrated by red, blue, and green lines in Figure 2.6, respectively. Only the projected points fall in region R_2 are selected to calculate the projected mean vector (*Dx*, *Dy*). That is,

$$Dx = \frac{\sum_{i=4}^{6} dx_i}{3} \quad \text{and} \quad Dy = \frac{\sum_{i=4}^{6} dy_i}{3},$$

and the red point is the refined motion vector, (*Dx*, *Dy*), in this example.

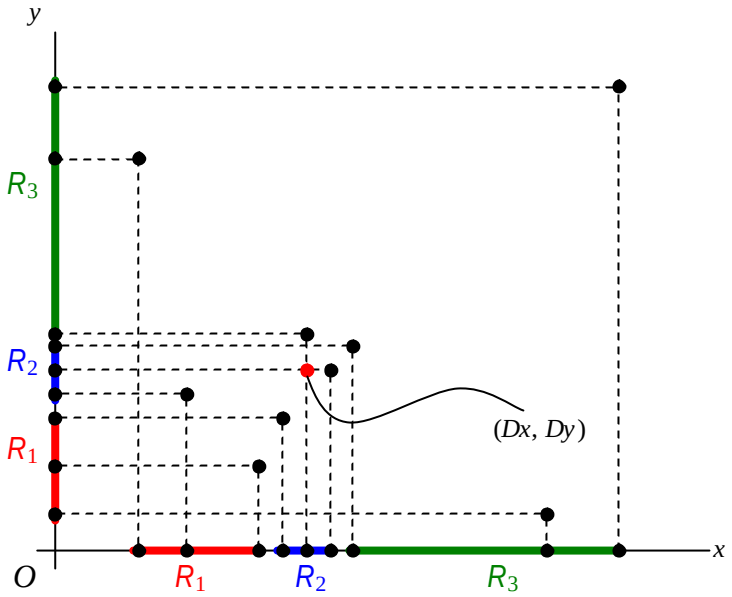

Figure 2.6 The estimated motion vectors are projected onto *x* axis and *y* axis, and the projected points are divided into three groups, R_1, R_2, *and* R_3, respectively. The projected mean vector, (*Dx*, *Dy*), is calculated from projected points in region R_2, and is described by red point.

2.4 Image Mosaicing

An initial point (x_0, y_0) on panoramic mosaic is selected as the starting position for reconstructing mosaics in this system. After estimating motion vectors, each vertical stripe with width *w* is cut from center of the image I_t and is pasted onto panoramic mosaic with an accumulated position $(x_0 + \sum_t Dx_t, y_0 + \sum_t Dy_t)$ by covering previous stripes, as shown in Figure 2.7. Thus, the actual width of the vertical stripe cut from image I_{t-1} is Dx_t because the rest region (vertical stripe with width $w - Dx_t$) is covered by the next pasted stripe of image I_t. The basic concept of image mosicing in this system is illustrated in Figure 2.7.

The selection of width *w* has some major considerations. First, if a wider *w* is selected, some unnecessary executive time is wasted because a large region of stripe is covered by next stripe. Second, if *w* is smaller than an arbitrary estimated Dx_t, then the constructed panoramic mosaic is separated in this pasting position. That is, the *w* must be larger than the allowed moving speed of the camera. In our

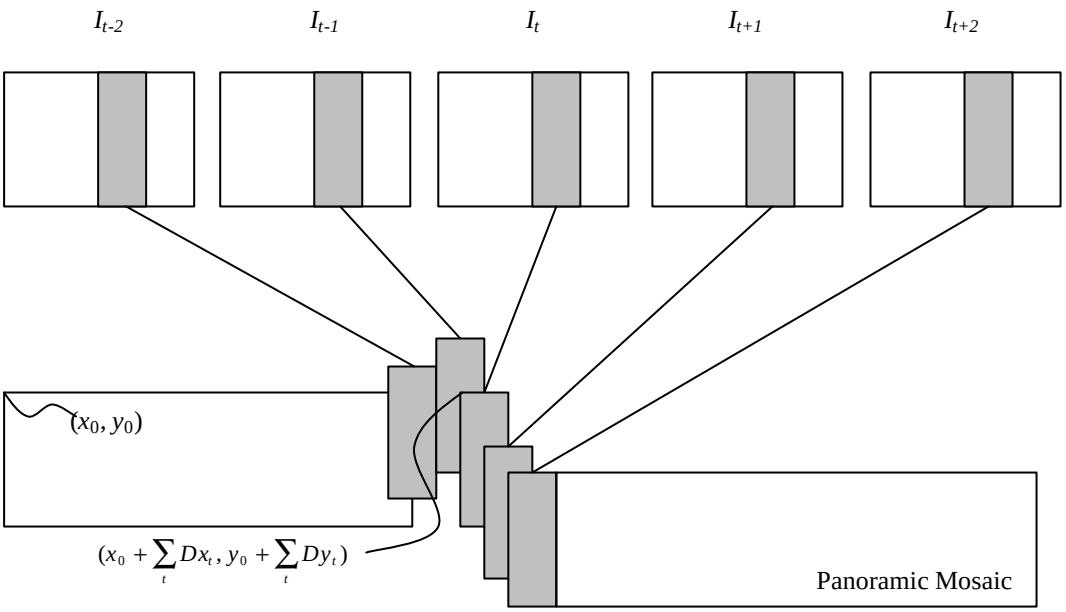

Figure 2.7 Using the selected stripes of the captured images to construct panoramic mosaics.

implementation, w is set to 50 pixels.

When the size of a video sequence is large, an image pyramid can be built for keeping the system real-time. Because our input camera devices can only allow 320 × 240 and 640 × 480 input video resolutions, a two-layer image pyramid is built, and the estimated motion vectors in the low resolution (320 × 240) layer are served as coarse estimations for initializing the search range for finding the motion vectors in the high resolution (640 × 480) layer. The computation cost can be lowered down by using such a coarse-to-fine strategy.

2.5 Examples of the Conducted Mosaics

By integrating the steps introduced above, a real-time distance-scanning system is built and panoramic mosaics can be constructed well for free camera motions and arbitrary scene structures. The developed real-time distance-scanning system was implemented on a Pentium III 1.2 GHz PC and the software was developed by using Microsoft Visual C++ 6.0. A window-based interface was designed for parameter setting as shown in Figure 2.8. The captured frame rate under these operation

environments is 14.90 Hz and the executed frame rate is 3.68 Hz. Figure 2.9 shows an experimental result of the real-time distance-scanning system.

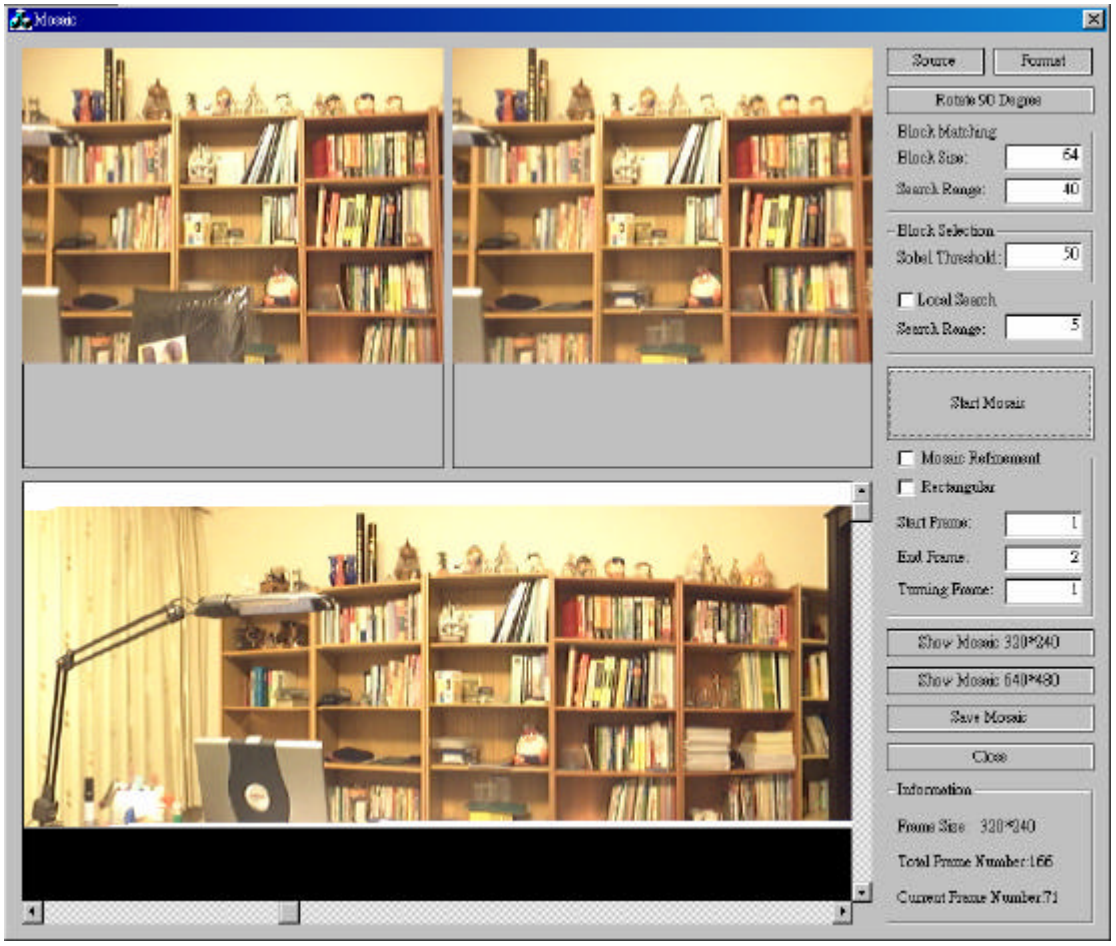

Figure 2.8 The GUI (Graphic User Interface) of the developed real-time distance-scanning system.

Figure 2.9 A constructed mosaic with intentional arbitrary camera motion.