〔非公開〕

ＴＲ－Ｍ－００３０

映像データベースに関する研究

－動きベクトルを用いたカメラワークの検出－

宮崎　仁　Hitoshi MIYAZAKI
井上　誠喜　Seiki INOUE

１９９８．２．２５

ＡＴＲ知能映像通信研究所

映像データベースに関する研究
— 動きベクトルを用いたカメラワークの検出 —

1998 年 2 月 25 日

ATR 知能映像通信研究所 第 3 研究室
豊橋技術科学大学 実務訓練生

宮崎 仁

1　研究の背景・目的

映像から特徴を取りだしデータベース化することによって、短時間で映像の内容を知ることができる。本実習では、取り出す映像の特徴としてカメラワークを検出することを目的とした。また、その手法として、ブロックマッチングにより求めた動きベクトルの特徴を調べることにした。

2　画像処理の基礎

今回の実務訓練で初めて画像処理を行うので、まず、画像処理についての基礎的なことを学ぶ必要があった。そこで、既存の画像処理プログラムを用いて2値化、輪郭抽出、雑音除去などの画像処理を勉強した。また、そのプログラムを参考にして、簡単な画像処理プログラムを作成した。

以下に示すのは、今回作成した画像処理の結果例である。

図 1: 原画像

図 2: 左右対称な画像

図 3: 中央に鏡を置いた画像

図 4: 濃度値を反転させた画像

図 5: 解像度を落とした画像

図 6: 階調度を落とした画像

図 7: 点描写した画像（1 ピクセル）

図 8: 点描写した画像（13 ピクセル）

3　カメラワークの検出

本研究では、動画像からカメラワークの検出を行うため、各フレーム間の動きベクトルを利用することにした。以下に、それぞれの過程について説明を行う。

3.1　動きベクトルの生成

この章では、動きベクトルを求める手法について説明を行う。

本実習では、動きベクトルを求めるのにブロックマッチングと呼ばれる手法を用いた。これは、図9のように、前フレーム f_{n-1} のブロックを上下左右に動かし、現フレーム f_n との差分を総和 D

$$D = \sum_i \sum_j |f_n(x_i, y_i) - f_{n-1}(x_i + dx, y_j + dy)|$$

が最小になる方向 (dx, dy) を動きベクトルとして求める方法である。ブロックの大きさは、8画素 ×8画素、ブロックを動かす最大量は、x方向 y方向ともに8画素である。また、本実習では、動きベクトルを求める画像として G画像を用いることにした。また、各フレーム間隔は、2とした。

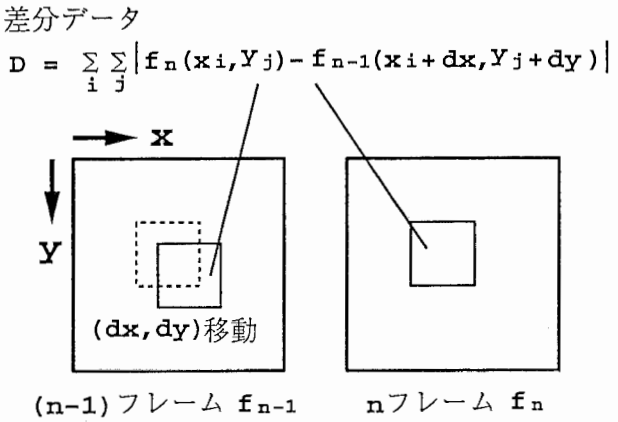

図 9: 動きベクトルの求め方（D が最小となる dx、dy を求める）

3.2 カメラワークの判定

この章では、動きベクトルからカメラワークを判定する方法について説明を行う。

3.2.1 パン、チルトの判定

次の図13に示すのは、パンしている動画像の2フレーム（図11、図12）に対してブロックマッチングにより動きベクトルを求めた結果である。図13から分かるようにカメラワークが、パン、チルトである場合、動きベクトルはほぼ一定方向に向いている。よって、動きベクトルの角度を求め、標準偏差を調べることでパン、チルトを判定することにした。

また、角度の平均によりパン、チルトの方向も求めることができる。本実習では、図10に示すように、パン、チルトを8つの方向に判別することにした。

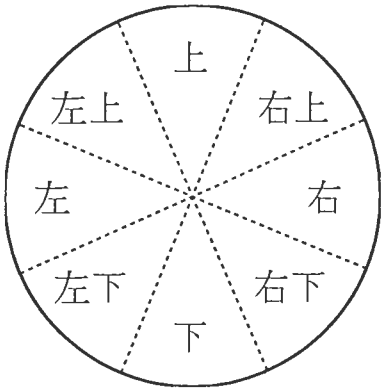

図 10: パン、チルトの方向

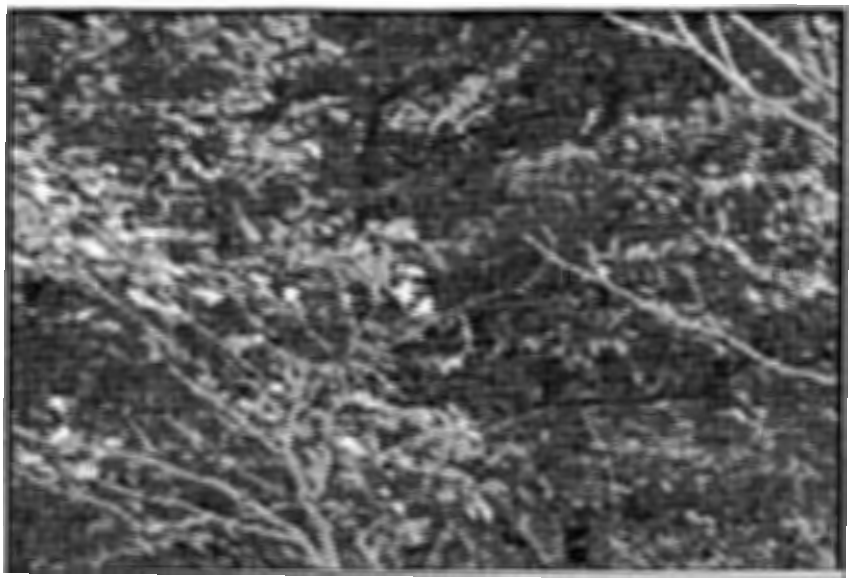

図 11: (n-2) フレームの画像（パン）

図 12: n フレームの画像（パン）

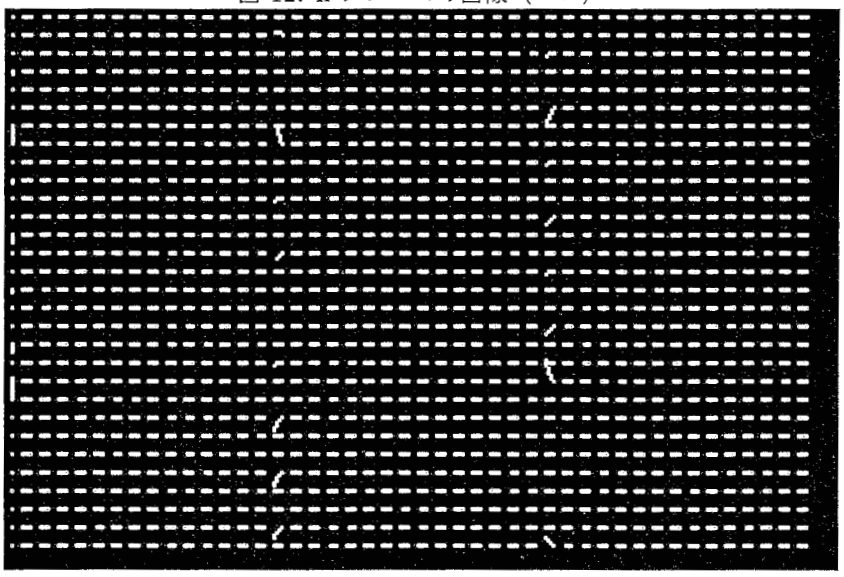

図 13: 動きベクトル（パン）

3.2.2 ズームの判定

図18に、カメラワークがズームの場合の動画フレーム（図16、図17）から求めた動きベクトルを示す。図18から分かるように、動きベクトルは焦点を中心に向かうようなベクトルである。よって、図14に示すように、すべてのヘクトルに対して、動きベクトルの向きにそれぞれのブロックの中心を通る直線を延ばしていけば、焦点部分でもっとも多く直線が交わるはずである。したがって、本実習では全画素値が0の画像に濃度値1の直線をベクトルの向きに描き、すべての動きベクトルに対して同様な直線を足し合わせていくことを行った。それにより、画像中の最大濃度値を求めることで、もっとも直線の重なりが多かった部分を求めることができる。本実習では、その値を用いてズームの判定を行うことにした。

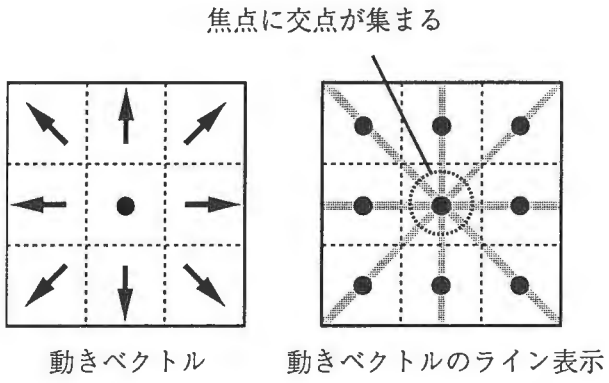

図 14: 動きベクトルのライン表示（ズーム判定）

3.2.3 ズームイン、ズームアウトの判定

図15で示すように、全画素が１２７の画像上に、ブロックの中央から動きベクトルに対して正方向に関しては、濃度値1を引く直線を描き、負方向に対しては、濃度値1を足すような直線を描く。それによって、描かれる画像は、ズームインであれば焦点部分の濃度値が高くなり、逆にズームアウトであれば焦点部分の濃度値が低くなる。したがって、焦点部分の濃度値が１２７より大きいか小さいかによって、ズームイン、ズームアウトの判定が行えるのである。図19、図20に示したものは、ズームイン、ズームアウトの時の処理結果である。ただし、この画像は、視覚的にズームイン、ズームアウトがわかりやすいように、加える濃度値を5にしたものである。

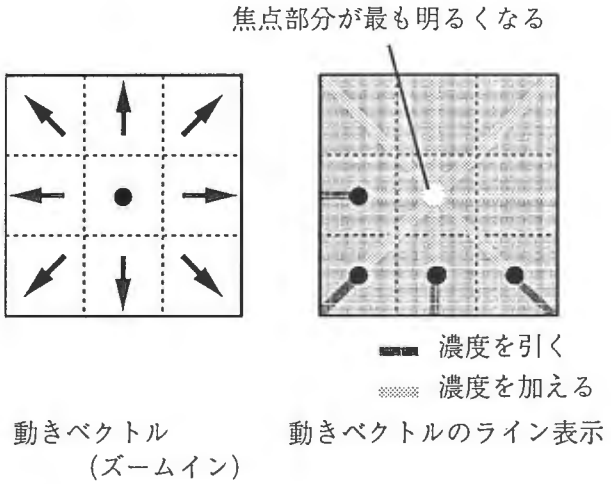

図 15: 動きベクトルのライン表示（ズームイン、アウト判定）

図 16: (n-2) フレームの画像（ズーム）

図 17: n フレームの画像（ズーム）

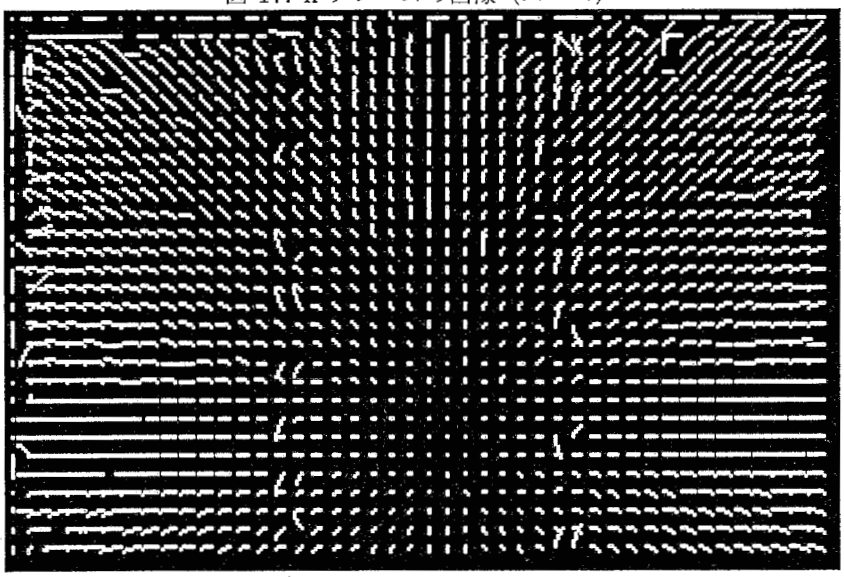

図 18: 動きベクトル（ズーム）

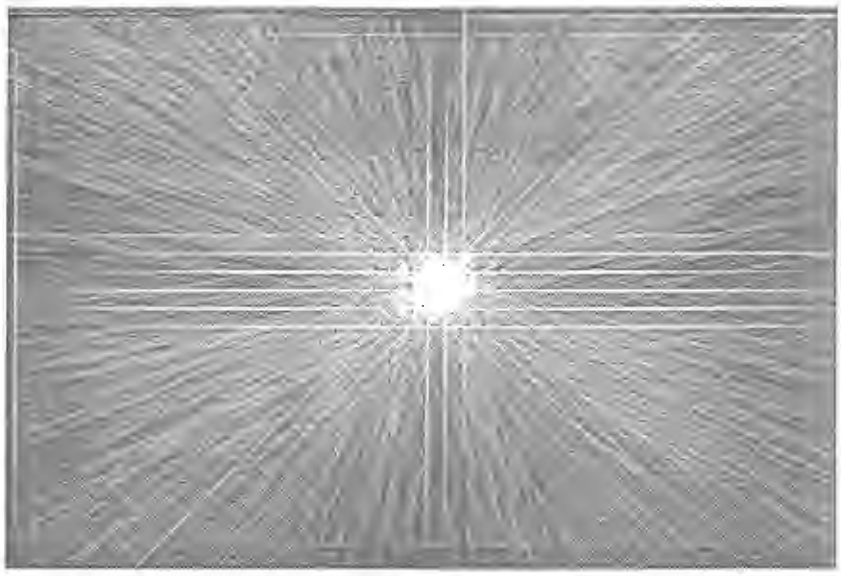

図 19: 動きベクトルのライン表示（ズームイン）

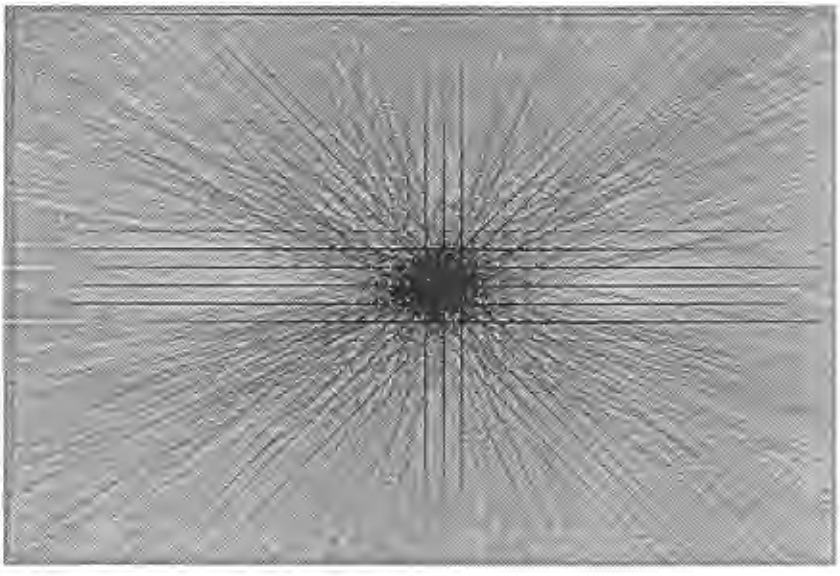

図 20: 動きベクトルのライン表示（ズームアウト）

3.2.4 ズーム時のパン判定

パンしながらのズームの場合、図21のように、焦点が中央から移動する。したがって、ズーム時のパンの判定は、焦点の位置をみることで分かる。本実習では、図22に示すように、画像を9つの領域に別け、焦点がどの位置にあるのかによって、パンとその方向を見分けることにした。また、焦点の位置は、ズームインとズームアウトでは、上下左右反対になるので、アウトかインかによって判定を変更した。

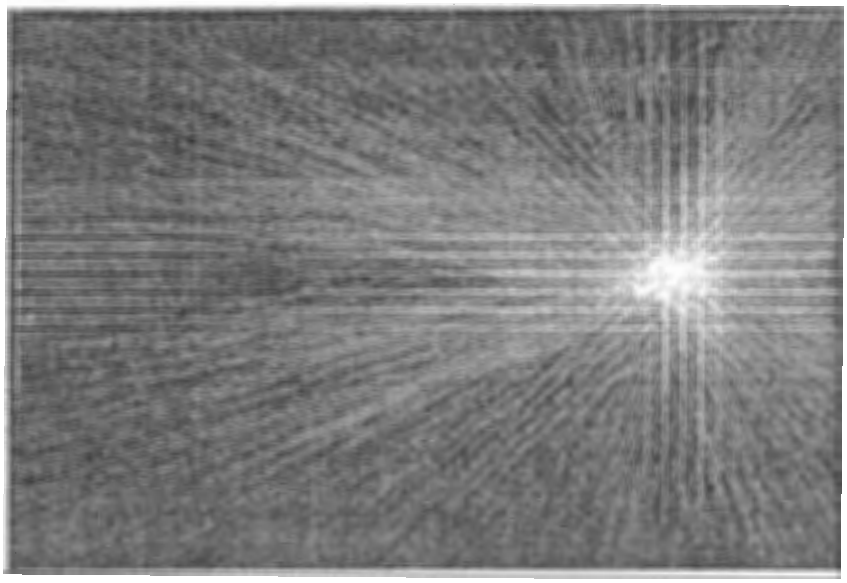

図 21: 動きベクトルのライン表示（ズームイン & パンライト）

左上	上	右上
左		右
左下	下	右下

ズームイン

右下	下	左下
右		左
右上	上	左上

ズームアウト

図 22: ズーム時のパン方向

4　実験結果

ビデオカメラで撮影した風景画像を対象に、パン、チルト、ズームの判定を行ってみた。図23、図24は、パン、チルトグラフに関しては動きベクトルの標準偏差を、

$$\text{パラメータ} = (90 - \text{標準偏差}) \times 100/90$$

の演算を行ったものを、ズームグラフに関しては、前章でのべた動きベクトル直線の最大交差値を

$$\text{パラメータ} = \text{最大交差値} \times 100/170$$

の演算をしたものをグラフにまとめたものである。これは、グラフの値が高いほど、カメラワークが起こっている確率が高いことを示していると考えられる。また、グラフ中央のバーは、実際の映像を私自身がカメラワークを判別した結果である。ちなみに、このグラフを求めるのにかかった計算時間は、半日程度であった。

図23、24から、パンしている時、パングラフの値は高くなっており、また、ズームしている時は、ズームグラフの値が高くなっている。したがって、実際のカメラワークとグラフは、対応が一致している事が分かる。（パン、チルトグラフの中で、パンではないのに数値が高くなっている部分があるが、これは、手振れが起きている状態であるためである。）

また、この映像はズームインしながらパンしている部分も含まれており、その部分のグラフの特徴を見ると、ただのズームの時と比べて、ズームに関しては低くなることがわかり、パンに関しては逆に高くなっていることが分かる。

しかしながら、パンやズーム中に急激に値が変動している部分が目立つことが分かる。この原因としては、1つには、フレーム間でまるで動きのない画像、つまり、動きベクトルがすべて0であるものがあることにある。本実習では、角度を求める際、動きベクトルが0であるときは、そのベクトルは考慮しないものとして省くようにしてある。もし、すべての動きベクトルが0である場合、角度の標準偏差は90とし、パングラフでは、0の値を取るようにしている。よって、パンの途中にまるで動きのない画像があった場合、その変動は急激になるのである。また、ズームに関しても、ほぼ同様なことが言える。

また、もう1つ値が急激に変動する原因としては、動きベクトルがランダムな方向に向いてしまっていることにある。

前者のような動きのない画像が含まれてしまう原因は不明であるが、後者が起こる原因としてまず考えられるのは、画像のブロックのある領域が特徴に少ない画像である、もしくは、同じような模様のある画像である場合が考えられる。

また、画像の動きが、速すぎてブロックの移動量範囲を越えてしまっているなどが考えられる。

だが、本実習では、その原因を特定し、改善するには至らなかった。

次に、パンに関しては、グラフでの80（標準偏差では18度）で、ズームに関しては、グラフでの60（最大交差値で102）を閾値として、パン、ズーム、および、それらの方向の判定を行ってみた。結果、当然、パンやズーム途中にうまく判定できない部分はあるものの、ほぼ正しくズームやパンの判定が行われていた。また、判定された部分に関しては、パンの方向、ズームイン、アウトは正しく判定されていた。

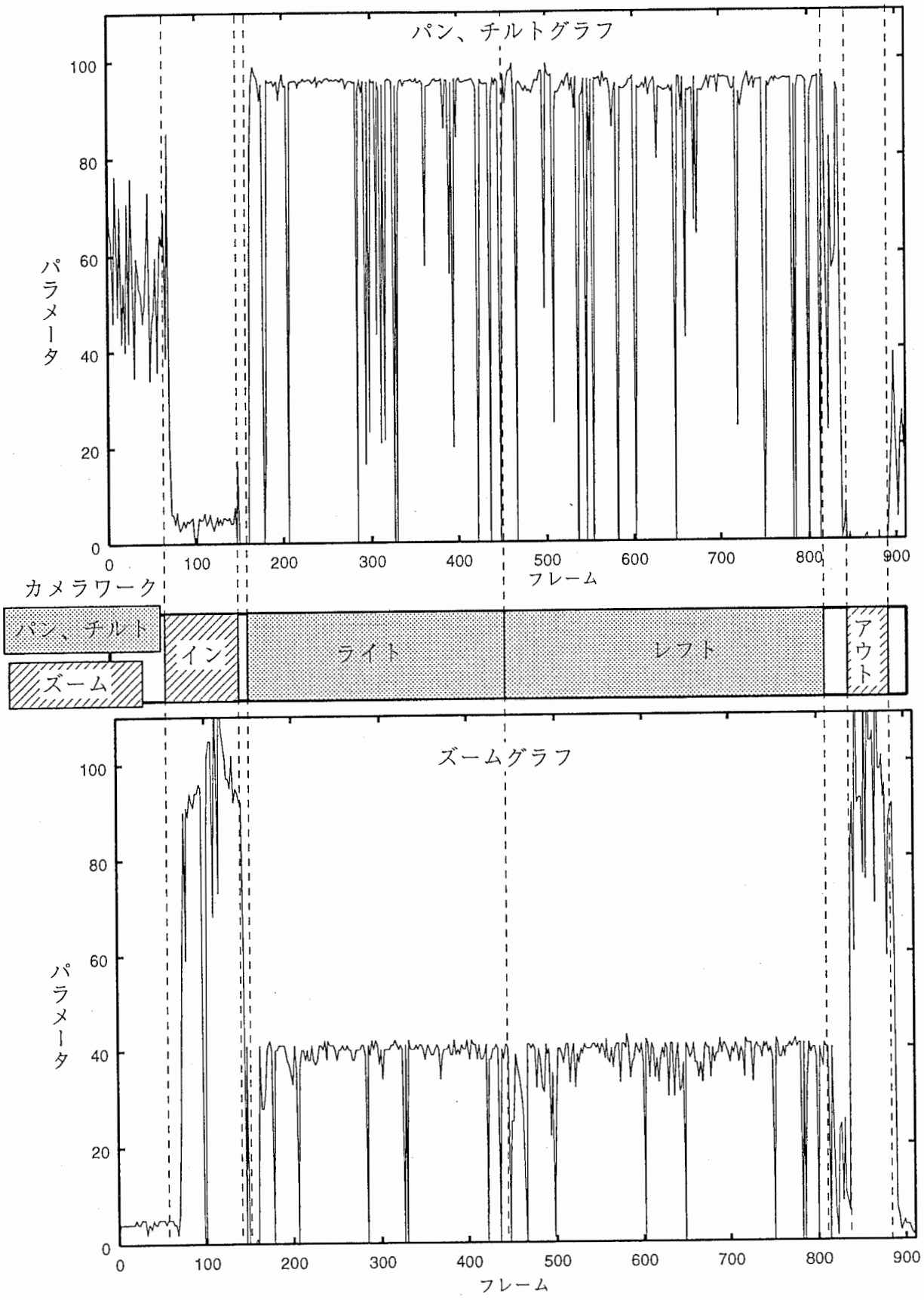

図 23: カメラワークの判定（0 フレームから 910 フレームまで）

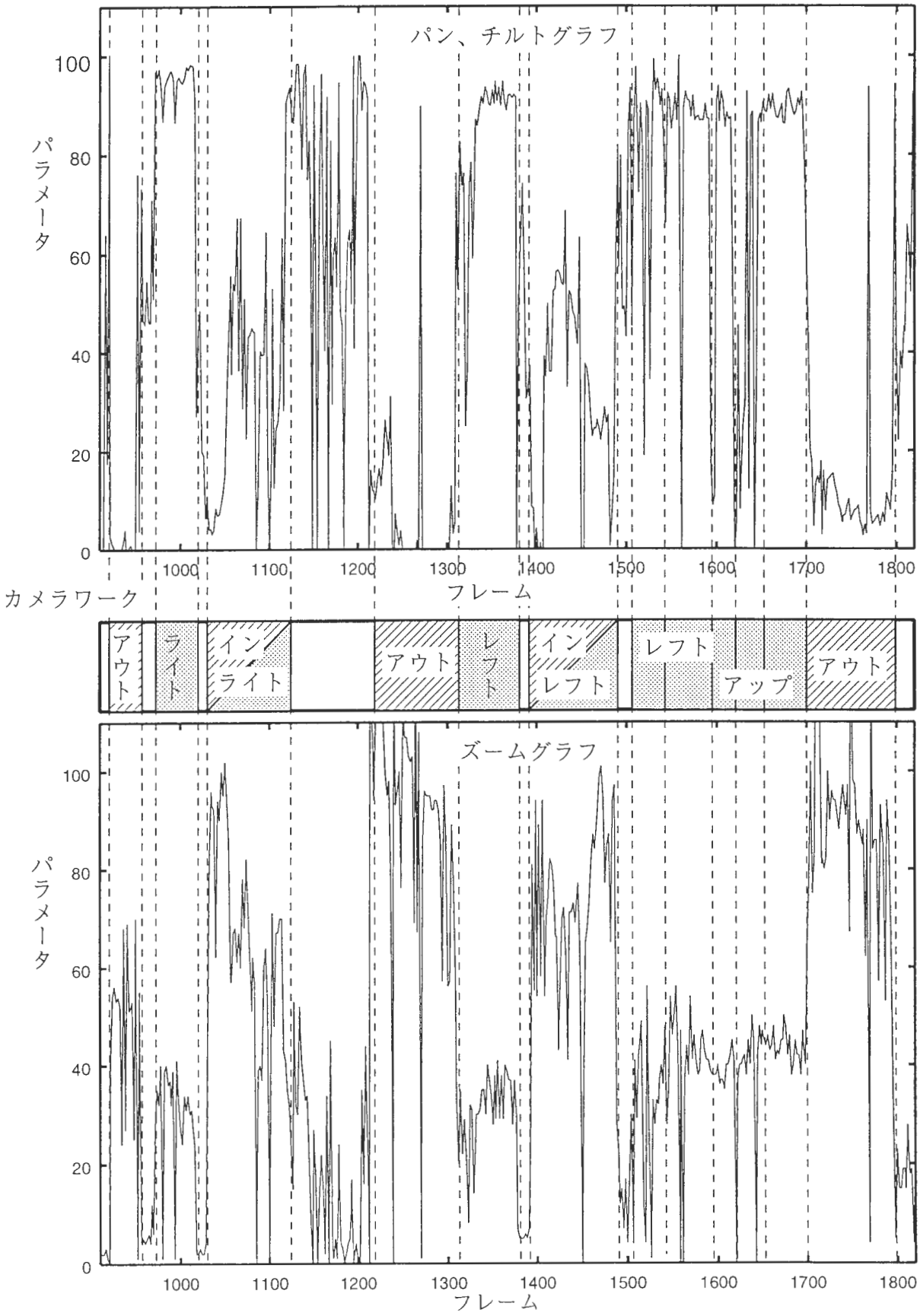

図 24: カメラワークの判定（910 フレームから 1820 フレームまで）

4.1 まとめ

本実習では、ブロックマッチングによる動きベクトルの特性を調べることで、カメラワークの検出を行う研究をした。結果、ビデオカメラで撮影した風景画像に対しては、グラフにばらつきがあるもののパン、チルト、ズームを検出することができた。今後の課題としては、空や壁のような特徴の少ない画像領域における処理、また、動きのあるものをとらえた画像に対する処理などが挙げられる。

5 謝辞

実務訓練を行うにあたって、多大なご指導を頂いた、井上誠喜室長をはじめ知能映像通信研究所第3研究室の方々に心から感謝いたします。また、充実した実務訓練の場を与えてくださった知能映像通信研究所の方々にこの場をお借りして厚く御礼を申し上げます。

参考文献

[1] 八木 伸行 ・ 井上 誠喜 ・ 林 正樹 ・ 中須 秀輔 ・ 三谷 公二 ・ 奥井 誠人 ・ 鈴木 正一 ・ 金次 保明 共著,
"C言語で学ぶ 実践画像処理", オーム社, 1992.

[2] 八木 伸行 ・ 井上 誠喜 ・ 林 正樹 ・ 奥井 誠人 ・ 合志 清一 共著,
"C言語で学ぶ 実践ディジタル映像処理", オーム社, 1995.

ユーザーズ ガイド

1998 年 2 月 25 日

ATR 知能映像通信研究所 第 3 研究室
豊橋技術科学大学 実務訓練生

宮崎 仁

1 プログラム概要

本実習で、作成したプログラムは、 /home/miyazaki/Kadai/ に格納されています。各プログラムの概要は、表に示す通りです。

表 1: プログラムの概要

ソースプログラム	概要
CVideo.c	メインプログラム
DispX.c	ウィンドウ表示
Dither.c	画像の量子化
image_operate.c	画像の読み込み、書き込みなど
Mvec.c	動きベクトルを求める
DrawMvec.c	動きベクトルを描画する
motionvec_operata.c	パン判定に用いるパラメータを求める
recognize_zoom.c	ズーム判定に用いるパラメータを求める
recognize_camera_work.c	カメラワークの判定を行う

2 使用のために

本実習で作成したプログラムは、 ppm(raw) 形式の画像を対象としています。

また、読み込む画像は、 /raid5-5/miyazaki/ の下に格納された画像を読み込むようにしています。画像ファイルは、たとえば、 filename25.rppm のような、ファイル名フレーム番号.拡張子の形になっています。このプログラムを使用されるときは、 image_operate.c ファイルに定義されている read_image_ppm 関数中のファイルオープン部分

```
/* ファイルオープン */
  sprintf(fname, "/raid5-5/miyazaki/%s.rppm", filename);
  if((fp = fopen(fname, "rb")) == NULL){
    printf("Cannot open file.\n");
    exit(0);
  }
```

の sprintf 文を画像ファイルのあるディレクトリに変更してください。また、必要があれば、拡張子の変更も行ってください。

同様に、画像の書き込みに関しても、 /Imagedata/PPM/ の下に行うようにしています。したがって、そのディレクトリをつくるか、プログラムファイル image_operate.c に定義されている write_image_ppm 関数中のファイルオープン部分

```
 /* ファイルオープン */
  sprintf(fname, "Imagedata/PPM/%s.rppm", filename);
  if((fp = fopen(fname, "wb")) == NULL){
    fprintf(stderr, "cannot open file: %s\n", fname);
    exit(0);
  }
```

を変更してください。

このプログラムでは、画像を表示するウィンドウを実行時に 3 つ開きます。ウィンドウには、それぞれ"IN"、"OUT"、"WORK" という名前がついています。

また、ブロックマッチングにおける、最大移動量は"mc.h" に入っています。変更したいときは、その中の定義を変更してください。

カメラワークの判定を行う際の閾値を、"recognize.h" で宣言している。必要であれば、変更してください。

3 使用方法

プログラムの使用方法について説明をします。まず、CVideo というコマンドを入力すると以下のようなメニューが表示されます。

```
### Video Signal Processing Menu (Color) ###
  1: Read Image File
  2: Write Image File
  8: Copy Image
  9: Clear Image
 15: Recognize Camera Work
 16: Movie
  0: END
Process Number ?
```

次に、実行したい処理をメニュー左の番号で指定してください。以下に、それぞれについての説明を行います。

3.1 メニュー1：ファイル読み込み

ファイルから画像データを読み込み、ウィンドウに表示する処理を行う際に、このメニューを選びます。
コマンドを実行すると、以下の文が表示されます。

```
Filename (read) ?
```

ここでは、読み込むファイル名を入力してください。例えば、

```
Filename (read) ? sample100
```

というように、打ち込んでください。
ファイル名を入力すると、次に以下のようなことを聞いてきます。

```
Which window (1:IN, 2:OUT, 3:WORK) ?
```

ここでは、読み込んだ画像をどのウィンドウに表示するかを番号で指定してください。
以上の入力を行うと、指定したウィンドウに画像が表示されます。

3.2 メニュー2：ファイルへの書き込み

ウィンドウに表示された画像を、ファイルに格納を行う際に、このメニューを選んでください。
コマンドを入力すると以下のようなメッセージが表示されます。

```
Which window (1:IN, 2:OUT, 3:WORK) ?
```

ここでは、出力したい画像が表示されているウィンドウを、番号で指定してください。次に、

```
Filename (write) ?
```

というメッセージが表示されます。ここで、出力するファイル名を入力してください。
以上の操作を行うことで、画像をファイルに格納することができます。

3.3 メニュー8：画像のコピー

ウィンドウに表示された画像を、他のウィンドウにコピーする際にこのメニューを選んでください。
このコマンドを実行すると、以下のようなメッセージが表示されます。

```
1:  IN --> OUT
2:  IN --> WORK
3:  OUT --> IN
4:  OUT --> WORK
5:  WORK --> IN
6:  WORK --> OUT
```

上のメニューでは、左側にコピー元、右側がコピーされるウィンドウを示しています。希望する処理を番号で選んでください。

番号を選ぶと、画像がコピーされ、ウィンドウに表示されます。

3.4 メニュー９：画像のクリア

ウィンドウに表示された画像を、初期化する時に、このメニューを選択します。

メニューを選択すると、以下のメッセージが表示されます。

```
1:  Clear IN
2:  Clear OUT
3:  Clear WORK
```

ここでは、初期化したい画像を番号で選んでください。

番号を選ぶと、指定された画像が初期化されます。

3.5 メニュー１５：カメラワークの判別

動画像から、カメラワークを判別する際にこのメニューを選びます。ただし、連続した動画のフレームが２枚以上ないと実行できません。

このメニューを選択すると、以下のようなメッセージが表示されます。

```
Filename (read) ?
Frame Start Number ?
Frame End Number ?
Frame Interval ?
```

それぞれメッセージにしたがって、読み込むファイル名、はじめのフレーム番号、終りのフレーム番号、フレーム間隔の順に入力してください。

たとえば、以下のように入力します。

```
Filename (read) ? sample
Frame Start Number ? 1116
Frame End Number ? 1118
Frame Interval ? 2
```

すべて入力し終わると、計算をはじめます。ウィンドウ"WORK" に表示される画像は、前フレーム画像で、"IN" に表示される画像が現フレームの画像です。

しばらくすると、動きベクトルの表示が"WORK" に、動きベクトルのライン表示が"OUT" に行われます。また、動きベクトルの平均値、角度の平均値、標準偏差、動きベクトルのライン表示の最大値などが表示されます。また、カメラワークを判定した結果が表示されます。

ここで、補足として、プログラムファイル CVideo.c のマクロ定義部分

```
/* ブロック分割数の指定時に宣言 */
/* #define  PRACTICABLE_BLOCK */
```

のコメントをはずすと、ブロックマッチングのブロックの個数を指定することができるようになります。そのときには、さらに次のようなメッセージが表示されます。

```
Ready [y] or Cange Block_Number ?[s]
```

ここで、ブロックの個数を変更する場合は"s" を、変更しない場合は"y" を入力します。

“s" と入力した場合、さらに以下のようなメッセージが表示されます。

```
Plese set Block Number.
```

ここで、ブロックの個数を x 方向、y 方向の順に入力してください。例えば、以下のように入力します。

```
45 30
```

すると、計算をはじめます。

3.6 メニュー１６：動画の再生

動画のフレームを連続的に表示したいとき、このメニューを選んでください。
このメニューを選ぶと以下のようなメッセージが表示されます。

```
Filename (read) ?
Frame Start Number ?
Frame End Number ?
Frame Interval ?
```

ここでは、メニュー１５のカメラワークの判別と同様に、読み込むファイル名、はじめのフレーム番号、終りのフレーム番号、フレーム間隔を入力します。
すると、指定された範囲のフレーム画像が連続的に表示されます。

3.7 バックグラウンド

上で、説明した中で、メニュー１５のカメラワークの判別は、たくさんのフレームで実行すると、長い計算時間が必要となります。そこで、バックグラウンドで実行できるようになっています。
その方法は、以下のように"CVideo" のあとに、オプションをつけることです。

```
CVideo -f sample -s 0 -e 1800 -i 2 > comment &
```

オプションのそれぞれの意味は、表2に示す通りです。

表 2: オプション

オプション	意味
f	読み込むファイル名
s	はじめのフレーム番号
e	終りのフレーム番号
i	フレーム間隔
h	ヘルプ

コマンドの後ろにつけた comment ファイルをみることで、計算がどこまで進んだかをチェックすることができます。
また、上で示したような実行の場合、その実行結果は、下の図3ようなファイルに格納されています。ファイル名の後ろの方にある数字は、はじめのフレーム番号、終りのフレーム番号、フレーム間隔です。

表 3: 格納ファイル

ファイル名	内容
sample_0_1820_2.data	計算の全結果
sample_standstill_0_1820_2.data	動きベクトルの平均
sample_pan_0_1820_2.data	パンパラメータ
sample_zoom_0_1820_2.data	ズームパラメータ
sample_work_0_1820_2.data	カメラワーク判定結果

プログラムリスト

1998年2月25日

ATR知能映像通信研究所 第3研究室
豊橋技術科学大学 実務訓練生

宮崎 仁

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "params.h"

/*--------------------------------------------------------------------------*/
/*                        線対象な画像を作成する                            */
/*--------------------------------------------------------------------------*/
line_symmetry(image_r_in, image_g_in, image_b_in,
              image_r_out, image_g_out, image_b_out)
     unsigned char image_r_in[Y_SIZE][X_SIZE];    /* 入力画像（赤）*/
     unsigned char image_g_in[Y_SIZE][X_SIZE];    /* 入力画像（緑）*/
     unsigned char image_b_in[Y_SIZE][X_SIZE];    /* 入力画像（青）*/
     unsigned char image_r_out[Y_SIZE][X_SIZE];   /* 出力画像（赤）*/
     unsigned char image_g_out[Y_SIZE][X_SIZE];   /* 出力画像（緑）*/
     unsigned char image_b_out[Y_SIZE][X_SIZE];   /* 出力画像（青）*/
{
  int i,j,k;

  for(i = 0; i < Y_SIZE; i++){
    for(j = 0; j < X_SIZE; j++){
      image_r_out[i][j] = image_r_in[i][X_SIZE - j];
      image_g_out[i][j] = image_g_in[i][X_SIZE - j];
      image_b_out[i][j] = image_b_in[i][X_SIZE - j];
    }
  }
}

/*--------------------------------------------------------------------------*/
/*                        階調度を下げた画像を作成する                      */
/*--------------------------------------------------------------------------*/
kaityoudo(image_r_in, image_g_in, image_b_in,
              image_r_out, image_g_out, image_b_out, number)
     unsigned char image_r_in[Y_SIZE][X_SIZE];    /* 入力画像（赤）*/
     unsigned char image_g_in[Y_SIZE][X_SIZE];    /* 入力画像（緑）*/
     unsigned char image_b_in[Y_SIZE][X_SIZE];    /* 入力画像（青）*/
     unsigned char image_r_out[Y_SIZE][X_SIZE];   /* 出力画像（赤）*/
     unsigned char image_g_out[Y_SIZE][X_SIZE];   /* 出力画像（緑）*/
     unsigned char image_b_out[Y_SIZE][X_SIZE];   /* 出力画像（青）*/
     int      number;                             /* 落とす階調度 */
{
  int i,j;

  for(i = 0; i < Y_SIZE; i++){
    for(j = 0; j < X_SIZE; j++){
      if((image_r_in[i][j] % number) >= (number / 2)){
        image_r_out[i][j] =
          (unsigned char) (((int)(image_r_in[i][j] / number) + 1) * number);
      }
      else{
        image_r_out[i][j] =
          (unsigned char) ((int)(image_r_in[i][j] / number) * number);
      }
      if((image_g_in[i][j] % number) >= (number / 2)){
        image_g_out[i][j] =
          (unsigned char) (((int)(image_g_in[i][j] / number) + 1) * number);
      }
      else{
        image_g_out[i][j] =
          (unsigned char) ((int)(image_g_in[i][j] / number) * number);
      }
      if((image_b_in[i][j] % number) >= (number / 2)){
        image_b_out[i][j] =
          (unsigned char) (((int)(image_b_in[i][j] / number) + 1) * number);
      }
      else{
```



```
        image_b_out[i][j] =
          (unsigned char) ((int)(image_b_in[i][j] / number) * number);
      }
    }
  }
}

/*--------------------------------------------------------------------------*/
/*                        濃度値を反転させた画像を作成する                  */
/*--------------------------------------------------------------------------*/
complementary_color(image_r_in, image_g_in, image_b_in,
              image_r_out, image_g_out, image_b_out)
     unsigned char image_r_in[Y_SIZE][X_SIZE];     /* 入力画像（赤）*/
     unsigned char image_g_in[Y_SIZE][X_SIZE];     /* 入力画像（緑）*/
     unsigned char image_b_in[Y_SIZE][X_SIZE];     /* 入力画像（青）*/
     unsigned char image_r_out[Y_SIZE][X_SIZE];    /* 出力画像（赤）*/
     unsigned char image_g_out[Y_SIZE][X_SIZE];    /* 出力画像（緑）*/
     unsigned char image_b_out[Y_SIZE][X_SIZE];    /* 出力画像（青）*/
{
  int i,j;

  for(i = 0; i < Y_SIZE; i++){
    for(j = 0; j < X_SIZE; j++){
      image_r_out[i][j] = HIGH - image_r_in[i][j];
      image_g_out[i][j] = HIGH - image_g_in[i][j];
      image_b_out[i][j] = HIGH - image_b_in[i][j];
    }
  }
}

/*--------------------------------------------------------------------------*/
/*                        中央に鏡を置いた画像を作成する                    */
/*--------------------------------------------------------------------------*/
mirror(image_r_in, image_g_in, image_b_in,
              image_r_out, image_g_out, image_b_out, mode)
     unsigned char image_r_in[Y_SIZE][X_SIZE];    /* 入力画像（赤）*/
     unsigned char image_g_in[Y_SIZE][X_SIZE];    /* 入力画像（緑）*/
     unsigned char image_b_in[Y_SIZE][X_SIZE];    /* 入力画像（青）*/
     unsigned char image_r_out[Y_SIZE][X_SIZE];   /* 出力画像（赤）*/
     unsigned char image_g_out[Y_SIZE][X_SIZE];   /* 入力画像（緑）*/
     unsigned char image_b_out[Y_SIZE][X_SIZE];   /* 出力画像（青）*/
     int       mode;                              /* 右半分か、左半分かを選ぶ */
{
  int i,j;

  if(mode == 1){
    for(i = 0; i < Y_SIZE; i++){
      for(j = 0; j <= X_SIZE/2; j++){
        image_r_out[i][j] = image_r_in[i][j];
        image_g_out[i][j] = image_g_in[i][j];
        image_b_out[i][j] = image_b_in[i][j];
      }
      for(j = X_SIZE/2 + 1; j < X_SIZE; j++){
        image_r_out[i][j] = image_r_in[i][X_SIZE - j];
        image_g_out[i][j] = image_g_in[i][X_SIZE - j];
        image_b_out[i][j] = image_b_in[i][X_SIZE - j];
      }
    }
  }

  if(mode == 2){
    for(i = 0; i < Y_SIZE; i++){
      for(j = X_SIZE/2 - 1; j >= 0; j--){
        image_r_out[i][j] = image_r_in[i][X_SIZE - j];
        image_g_out[i][j] = image_g_in[i][X_SIZE - j];
        image_b_out[i][j] = image_b_in[i][X_SIZE - j];
```

```
      }
      for(j = X_SIZE/2; j < X_SIZE; j++){
        image_r_out[i][j] = image_r_in[i][j];
        image_g_out[i][j] = image_g_in[i][j];
        image_b_out[i][j] = image_b_in[i][j];
      }
    }
  }
}

/*------------------------------------------------------------------------------*/
/*                    解像度を落とした画像を作成する                            */
/*------------------------------------------------------------------------------*/
mosaic(image_r_in, image_g_in, image_b_in,
       image_r_out, image_g_out, image_b_out, size)
     unsigned char image_r_in[Y_SIZE][X_SIZE];    /* 入力画像（赤）*/
     unsigned char image_g_in[Y_SIZE][X_SIZE];    /* 入力画像（緑）*/
     unsigned char image_b_in[Y_SIZE][X_SIZE];    /* 入力画像（青）*/
     unsigned char image_r_out[Y_SIZE][X_SIZE];   /* 出力画像（赤）*/
     unsigned char image_g_out[Y_SIZE][X_SIZE];   /* 出力画像（緑）*/
     unsigned char image_b_out[Y_SIZE][X_SIZE];   /* 出力画像（青）*/
     int      size;
{
  int i,j,k,l;
  int sum_r, sum_g, sum_b, div;

  div = size * size;
  for(i = 0; i < Y_SIZE; i += size){
    for(j = 0; j < X_SIZE; j += size){
      sum_r = 0;  sum_g = 0;  sum_b = 0;
      for(k = 0; k < size; k++){
        for(l = 0; l < size; l++){
          if(((i + k) < Y_SIZE) || ((j + l) < X_SIZE)){
            sum_r += image_r_in[i + k][j + l];
            sum_g += image_g_in[i + k][j + l];
            sum_b += image_b_in[i + k][j + l];
          }
        }
      }

      sum_r /= div;  sum_g /= div;  sum_b /= div;
      for(k = 0; k < size; k++){
        for(l = 0; l < size; l++){
          if(((i + k) < Y_SIZE) || ((j + l) < X_SIZE)){
            image_r_out[i + k][j + l] = (unsigned char) sum_r;
            image_g_out[i + k][j + l] = (unsigned char) sum_g;
            image_b_out[i + k][j + l] = (unsigned char) sum_b;
          }
        }
      }
    }
  }
}

/*------------------------------------------------------------------------------*/
/*                  点描写をした画像を作成する（13ピクセル）                    */
/*------------------------------------------------------------------------------*/
#define RANDOMIZE() srand((unsigned int) time(NULL))
#define RANDOM(x)   (rand() % (x))

speck_picture(image_r_in, image_g_in, image_b_in,
              image_r_out, image_g_out, image_b_out, number)
     unsigned char image_r_in[Y_SIZE][X_SIZE];    /* 入力画像（赤）*/
     unsigned char image_g_in[Y_SIZE][X_SIZE];    /* 入力画像（緑）*/
     unsigned char image_b_in[Y_SIZE][X_SIZE];    /* 入力画像（青）*/
     unsigned char image_r_out[Y_SIZE][X_SIZE];   /* 出力画像（赤）*/
```



```
     unsigned char image_g_out[Y_SIZE][X_SIZE];   /* 出力画像（緑）*/
     unsigned char image_b_out[Y_SIZE][X_SIZE];   /* 出力画像（青）*/
     int   number;                                /* 点の数 */
{
  int i,j;
  int x_point, y_point;

  for(i = 0; i < Y_SIZE; i++){
    for(j = 0; j < X_SIZE; j++){
      image_r_out[i][j] = 0;
      image_g_out[i][j] = 0;
      image_b_out[i][j] = 0;
    }
  }

  for(i = 0; i < number; i++){
    x_point = RANDOM(X_SIZE);
    y_point = RANDOM(Y_SIZE);
    speck_picture_sub(image_r_in, image_r_out, x_point, y_point);
    speck_picture_sub(image_g_in, image_g_out, x_point, y_point);
    speck_picture_sub(image_b_in, image_b_out, x_point, y_point);
  }
}

/* 指定された位置に13画素の点をその平均値で描く */
speck_picture_sub(image_in, image_out, x_point, y_point)
     unsigned char image_in[Y_SIZE][X_SIZE];    /* 入力画像 */
     unsigned char image_out[Y_SIZE][X_SIZE];   /* 出力画像 */
     int    x_point;                            /* x位置 */
     int    y_point;                            /* y位置 */
{
  int i,j,k;
  int pick_pixel[13];
  int sum;

  /* 入力画像から点（13ピクセル）の濃度値を取り出す */
  pick_pixel[0] = image_in[y_point - 2][x_point];
  pick_pixel[1] = image_in[y_point + 2][x_point];
  pick_pixel[2] = image_in[y_point][x_point - 2];
  pick_pixel[3] = image_in[y_point][x_point + 2];
  pick_pixel[4] = image_in[y_point - 1][x_point - 1];
  pick_pixel[5] = image_in[y_point - 1][x_point];
  pick_pixel[6] = image_in[y_point - 1][x_point + 1];
  pick_pixel[7] = image_in[y_point][x_point - 1];
  pick_pixel[8] = image_in[y_point][x_point];
  pick_pixel[9] = image_in[y_point][x_point + 1];
  pick_pixel[10] = image_in[y_point + 1][x_point - 1];
  pick_pixel[11] = image_in[y_point + 1][x_point];
  pick_pixel[12] = image_in[y_point + 1][x_point + 1];

  /* 平均値を求める */
  sum = 0;
  for(k = 0; k < 13; k++){
    sum += pick_pixel[k];
  }
  sum /= 13;

  if((sum * 1.1) < LEVEL) {
    image_out[y_point - 2][x_point] =
      image_out[y_point + 2][x_point] =
      image_out[y_point][x_point - 2] =
      image_out[y_point][x_point + 2]
      = (unsigned char) (sum * 1.1);
  }
  else{
    image_out[y_point - 2][x_point] =
```

```
      image_out[y_point + 2][x_point] =
      image_out[y_point][x_point - 2] =
      image_out[y_point][x_point + 2]
      = HIGH;
  }
  image_out[y_point - 1][x_point -1] =
    image_out[y_point - 1][x_point] =
    image_out[y_point][x_point - 1] =
    image_out[y_point][x_point + 1]
    = image_out[y_point + 1][x_point - 1] =
    image_out[y_point + 1][x_point] =
    image_out[y_point + 1][x_point + 1] =(unsigned char) sum;
  image_out[y_point][x_point] = (unsigned char) (sum / 1.1);

}

/*--------------------------------------------------------------------------*/
/*                  点描写をした画像を作成する（1ピクセル）                 */
/*--------------------------------------------------------------------------*/
speck_picture2(image_r_in, image_g_in, image_b_in,
               image_r_out, image_g_out, image_b_out, number)
     unsigned char image_r_in[Y_SIZE][X_SIZE];    /* 入力画像（赤）*/
     unsigned char image_g_in[Y_SIZE][X_SIZE];    /* 入力画像（緑）*/
     unsigned char image_b_in[Y_SIZE][X_SIZE];    /* 入力画像（青）*/
     unsigned char image_r_out[Y_SIZE][X_SIZE];   /* 出力画像（赤）*/
     unsigned char image_g_out[Y_SIZE][X_SIZE];   /* 出力画像（緑）*/
     unsigned char image_b_out[Y_SIZE][X_SIZE];   /* 出力画像（青）*/
     int    number;                               /* 点の数 */
{
  int i,j,m;
  int x_point, y_point;

  for(i = 0; i < Y_SIZE; i++){
    for(j = 0; j < X_SIZE; j++){
      image_r_out[i][j] = 0;
      image_g_out[i][j] = 0;
      image_b_out[i][j] = 0;
    }
  }

  for(i = 0; i < number; i++){
    x_point = RANDOM(X_SIZE);
    y_point = RANDOM(Y_SIZE);
    image_r_out[y_point][x_point] = image_r_in[y_point][x_point];
    image_g_out[y_point][x_point] = image_g_in[y_point][x_point];
    image_b_out[y_point][x_point] = image_b_in[y_point][x_point];
  }
}
```

```
#define X_SIZE_EXT  256   /* 画像の横方向の大きさ以上で最小の２のべき乗の数 */
#define Y_SIZE_EXT  128   /* 画像の縦方向の大きさ以上で最小の２のべき乗の数 */
#define X_SIZE_PWR      8       /* X_SIZE_EXT = 2 ** X_SIZE_PWR */
#define Y_SIZE_PWR      7       /* Y_SIZE_EXT = 2 ** Y_SIZE_PWR */
#define BIT             8       /* 量子化のビット数 */

#define HIGH          255       /* 最大濃度値  */
#define LOW             0       /* 最小濃度値  */
#define MAX_TAP        64       /* フィルタの最大タップ数（水平，垂直とも）*/

#define GRAY    1       /* モノクローモード */
#define COLOR   2       /* カラーモード     */
                        /* ppm画像タイプ */

#define IN      1               /* 入力画像用ウィンドウ          */
#define OUT     2               /* 出力画像用ウィンドウ          */
#define WORK    3               /* ワーク画像用ウィンドウ        */

#define MAXSIZE 3000

/*-------------------------------------------------------------------------*/
/*                          IMAGE構造体の定義                              */
/*-------------------------------------------------------------------------*/
typedef struct img {
  char type;          /* 画像タイプ */
  unsigned int xsize; /* xサイズ */
  unsigned int ysize; /* yサイズ */
  unsigned int depth;
  unsigned char *rdata;       /* r成分 */
  unsigned char *gdata;       /* g成分 */
  unsigned char *bdata;       /* b成分 */
} IMAGE;
```

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>
#include        "params.h"
#include        "recognize.h"

/* ブロック分割数の指定時に宣言 */
/* #define  PRACTICABLE_BLOCK */
main(argc, argv)
     int     argc;
     char   *argv[];
{
  static int command = -1;           /* メニューコマンド */
  char    source[80];                /* 入力ファイル名 */
  char    destin[80];                /* 出力ファイル名 */
  char    fname[80];                 /* 入力ファイル名（フレーム番号つき）*/
  char    c[50];                     /* ブロック個数を変更時のコマンド用 */
  char    pro_standstill_fname[80];  /* スタンドパラメータ格納ファイル */
  char    pro_pan_fname[80];         /* パンパラメータ格納ファイル */
  char    pro_zoom_fname[80];        /* ズームパラメータ格納ファイル */
  char    rec_work_fname[80];        /* カメラワーク格納ファイル */
  char    fname_current[80], fname_past[80]; /* 現在、過去の画像ファイル */
  char    frame_num[80];             /* フレーム番号 */
  int     i, j, k;
  int     n;                      /* 画像出力ウィンドウの指定 */
  int     block_number_x;         /* ブロックの個数（x方向）*/
  int     block_number_y;         /* ブロックの個数（y方向）*/
  int     init_display_flag;      /* ディスプレイが初期化チェック用フラグ */
  int     set_flag;
  IMAGE   data;                   /* ファイルから読み込んだ画像データ */

  int  start_frame, end_frame;    /* 読み込むフレーム番号（最初と最後）*/
  int  interval_frame;            /* 読み込むフレームの間隔 */

  unsigned char *image_in[3];    /* 入力画像配列  */
  unsigned char *image_out[3];   /* 出力画像配列  */
  unsigned char *image_work[3]; /* ワーク画像配列 */

  char            *vx;           /* 動きベクトルのｘ成分 */
  char            *vy;           /* 動きベクトルのｙ成分 */

  double v_average;                  /* 動きベクトルの平均値 */
  double *v_angle;                   /* 動きベクトルの角度 */
  double angle_average;              /* 角度の平均値 */
  double angle_standard_deviation;   /* 角度の標準偏差 */
  int  camera_work_cur;                 /* カメラワーク(現フレーム) */
  int  camera_work_pas;              /* カメラワーク（前フレーム）*/
  int  count;                        /* 最大濃度値の数 */
  int  max;                          /* 濃度の最大値 */
  int  value[MAXSIZE];          /* ズームイン、アウト判定における最大濃度値 */
  int  position_x[MAXSIZE];          /* 最大濃度値の位置（x座標）*/
  int  position_y[MAXSIZE];          /* 最大濃度値の位置（y座標）*/

  FILE     *fp;                        /* ファイル読み込み用ファイルポインタ */
  FILE     *probability_standstill;    /* スタンドパラメータ出力用ポインタ */
  FILE     *probability_pan;           /* パンパラメータ出力用ポインタ */
  FILE     *probability_zoom;          /* ズームパラメータ出力用ポインタ */
  FILE     *recognize_work;            /* カメラワーク出力用ポインタ */

  /* 初期値 */
  block_number_x = 45;
  block_number_y = 30;
  init_display_flag = 0;
  sprintf(source, "%s", "sample");
  start_frame = 0;
  end_frame = 100;
```



```
  interval_frame = 2;
  camera_work_pas = -1 ;

  /* バックグラウンド用 */
  if(argc > 1){
    for(i = 1; i < argc; i++){
      if(argv[i][0] == '-'){
        switch(argv[i][1]){
        case 'f':
          sscanf(argv[i+1], "%s", source);
          i++;
          break;
        case 's':
          sscanf(argv[i+1], "%d", &start_frame);
          i++;
          break;
        case 'e':
          sscanf(argv[i+1], "%d", &end_frame);
          i++;
          break;
        case 'i':
          sscanf(argv[i+1], "%d", &interval_frame);
          i++;
          break;
        case 'h':
          printf("Recognize Camera Work from Video\n");
          printf("CVideo -> print Menu");
          printf("CVideo -option value\n");
          printf("-f filename : read filename\n");
          printf("-s number   : start frame number\n");
          printf("-e number   : end frame number\n");
          printf("-i number   : interval value\n");
          printf("-h          : help\n");
          exit(0);
        }
      }
    }

    /* 各種ファイルのオープン */
    sprintf(pro_standstill_fname,
            "Imagedata/Framedata/%s_standstill_%d_%d_%d.data",
            source, start_frame, end_frame, interval_frame);
    if((probability_standstill = fopen(pro_standstill_fname, "wb")) == NULL){
      fprintf(stderr, "cannot open file: %s\n", pro_standstill_fname);
      exit(0);
    }
    sprintf(pro_pan_fname, "Imagedata/Framedata/%s_pan_%d_%d_%d.data",
            source, start_frame, end_frame, interval_frame);
    if((probability_pan = fopen(pro_pan_fname, "wb")) == NULL){
      fprintf(stderr, "cannot open file: %s\n", pro_pan_fname);
      exit(0);
    }
    sprintf(pro_zoom_fname, "Imagedata/Framedata/%s_zoom_%d_%d_%d.data",
            source, start_frame, end_frame, interval_frame);
    if((probability_zoom = fopen(pro_zoom_fname, "wb")) == NULL){
      fprintf(stderr, "cannot open file: %s\n", pro_zoom_fname);
      exit(0);
    }
    sprintf(rec_work_fname, "Imagedata/Framedata/%s_work_%d_%d_%d.data",
            source, start_frame, end_frame, interval_frame);
    if((recognize_work = fopen(rec_work_fname, "wb")) == NULL){
      fprintf(stderr, "cannot open file: %s\n", rec_work_fname);
      exit(0);
    }
    sprintf(fname, "Imagedata/Framedata/%s_%d_%d_%d.data",
            source, start_frame, end_frame, interval_frame);
```

```
if((fp = fopen(fname, "wb")) == NULL){
  fprintf(stderr, "cannot open file: %s\n", fname);
  exit(0);
}

for(j = start_frame; j <= (end_frame - interval_frame);
    j += interval_frame){
  /* 現在のフレーム画像の読み込み */
  sprintf(fname_current, "%s%d", source, j + interval_frame);
  read_image_ppm(fname_current, &data);

  /* ディスプレイの初期化 */
  if(init_display_flag == 0){
    color_image_init(image_in, image_out, image_work,
                     data.xsize, data.ysize);
    init_display_flag = 1;
  }
  color_image_read(data, image_in);

  /* 過去のフレーム画像の読み込み */
  sprintf(fname_past, "%s%d", source, j);
  read_image_ppm(fname_past,&data);
  color_image_read(data, image_work);

  /* 領域確保 */
  vx = (char *) malloc((size_t) block_number_x * block_number_y);
  vy = (char *) malloc((size_t) block_number_x * block_number_y);
  if (vx == NULL || vy == NULL) {
    printf("Memory not enough. \n");
    exit(-1);
  }

  v_angle =
    (double *)malloc(block_number_x * block_number_y * sizeof(double));
  if (v_angle == NULL) {
    printf("Memory not enough for v_angle. \n");
    exit(0);
  }

  printf("    IN:      Current frame %s. \n", fname_current);
  printf("    WORK:    Previous frame %s. \n\n", fname_past);

  /* 動きベクトルを求める */
  draw_mvec(image_in, image_work, image_out,
            data.xsize, data.ysize, vx, vy,
            block_number_x, block_number_y);

  /* 動きベクトルの２乗和平均、角度、角度平均、角度の標準偏差を求める */
  power_average_motion_vector(vx, vy, block_number_x, block_number_y,
                              &v_average);
  angle_motion_vector (vx, vy, block_number_x, block_number_y, v_angle);
  average_angle_motion_vector(v_angle, block_number_x, block_number_y,
                              &angle_average);
  standard_deviation_angle_motion_vector(v_angle, angle_average,
                                         block_number_x, block_number_y,
                                         &angle_standard_deviation);

  /* 動きベクトルのライン表示により、交点の最大値を求める */
  find_point_intersection_max(vx, vy, data.xsize, data.ysize,
                              block_number_x, block_number_y, &max,
                              position_x, position_y, &count);
  printf("count : %d\n", count);
  for(i = 0; i < count; i++){
    find_value_position(vx, vy, data.xsize, data.ysize, block_number_x,
                        block_number_y, position_x[i], position_y[i],
                        &value[i]);
```



```
  }

  /* パラメータにより、カメラワークを判定する */
  recognize_camera_work(v_average, angle_average,
                        angle_standard_deviation, max, position_x,
                        position_y, value, count, data.xsize,
                        data.ysize, &camera_work_cur);

  /* 結果のファイル出力 */
  fprintf(fp, "####### %s  -  %s #######\n", fname_past, fname_current);
  fprintf(fp, "vx_average : %lf\n", v_average);
  fprintf(fp, "angle_average : %lf\n", angle_average);
  fprintf(fp, "angle_standard_deviation : %lf\n",
          angle_standard_deviation);
  fprintf(fp, "Max Value : %d\n", max);
  for(i = 0; i < count; i++){
    fprintf(fp, "Position x,y : %d %d\n",
            position_x[i], position_y[i]);
    fprintf(fp, "Value : %d\n\n", value[i]);
  }
  if(camera_work_cur != camera_work_pas){
    if(camera_work_cur & STANDSTILL){
      fprintf(fp, "Camera_Work : StandStill\n");
      fprintf(recognize_work, "%d\n\tStandStill\n",j);
    }
    if(camera_work_cur == NO_RECOGNIZE){
      fprintf(fp, "Camera_work : No Recognize\n");
      fprintf(recognize_work, "%d\n\tNo Recognize\n",j);
    }
    if(camera_work_cur & PAN_UP){
      fprintf(fp, "Camera_work : Pan Up\n");
      fprintf(recognize_work, "%d\n\tPan Up\n",j);
    }
    if(camera_work_cur & PAN_DOWN){
      fprintf(fp, "Camera_work : Pan Down\n");
      fprintf(recognize_work, "%d\n\tPan Down\n",j);
    }
    if(camera_work_cur & PAN_RIGHT){
      fprintf(fp, "Camera_work : Pan Right\n");
      fprintf(recognize_work, "%d\n\tPan Right\n",j);
    }
    if(camera_work_cur & PAN_LEFT){
      fprintf(fp, "Camera_work : Pan Left\n");
      fprintf(recognize_work, "%d\n\tPan Left\n",j);
    }
    if(camera_work_cur & PAN_RIGHT_UP){
      fprintf(fp, "Camera_work : Pan Right_Up\n");
      fprintf(recognize_work, "%d\n\tPan Right\n",j);
    }
    if(camera_work_cur & PAN_RIGHT_DOWN){
      fprintf(fp, "Camera_work : Pan Right_Down\n");
      fprintf(recognize_work, "%d\n\tPan RightDown\n",j);
    }
    if(camera_work_cur & PAN_LEFT_UP){
      fprintf(fp, "Camera_work : Pan Left_Up\n");
      fprintf(recognize_work, "%d\n\tPan LeftUp\n",j);
    }
    if(camera_work_cur & PAN_LEFT_DOWN){
      fprintf(fp, "Camera_work : Pan Left_Down\n");
      fprintf(recognize_work, "%d\n\tPan LeftDown\n",j);
    }
    if(camera_work_cur & ZOOM_IN){
      fprintf(fp, "Camera_work : Zoom_In\n");
      fprintf(recognize_work, "%d\n\tZoom In\n",j);
    }
    if(camera_work_cur & ZOOM_OUT){
```

```
          fprintf(fp, "Camera_work : Zoom_Out\n");
          fprintf(recognize_work, "%d\n\tZoom Out\n",j);
        }
        camera_work_pas = camera_work_cur;
      }
      if(j == end_frame - interval_frame){
        fprintf(recognize_work, "%d\n",j);
      }

      fprintf(probability_standstill, "%d %lf\n",
              j, (8 - v_average) * 100 / 8);
      fprintf(probability_pan, "%d %lf\n",
              j, (90 - angle_standard_deviation) * 100 / 90);
      fprintf(probability_zoom, "%d %lf\n", j, (double)(max * 100 / 170));

      /* 領域の解放 */
      free(data.rdata);
      free(data.gdata);
      free(data.bdata);
      free(vx);
      free(vy);
      free(v_angle);
    }

    /* ファイルクローズ */
    fclose(fp);
    fclose(probability_standstill);
    fclose(probability_pan);
    fclose(probability_zoom);
    fclose(recognize_work);
  }

  /* メニューコマンドモード */
  else{
    while (command) {
      printf("### Video Signal Processing Menu (Color) ###\n");
      printf("  1: Read Image File\n");
      printf("  2: Write Image File\n");
      printf("  8: Copy Image\n");
      printf("  9: Clear Image\n");
      printf(" 15: Recognize Camera Work \n");
      printf(" 16: Movie\n");
      printf("  0: END\n");
      printf("Process Number ?   ");
      scanf("%d", &command);
      switch (command) {
      /* 画像の読み込み、および ディスプレイ表示 */
      case 1:
        /* 画像の読み込み */
        printf("  Filename (read) ? ");
        scanf("%s", source);
        printf("  Which window (1:IN, 2:OUT, 3:WORK) ? ");
        scanf("%d", &n);
        read_image_ppm(source,&data);

        /* ディスプレイの初期化 */
        if(init_display_flag == 0){
          color_image_init(image_in, image_out, image_work,
                           data.xsize, data.ysize);
          InitCDisplay(argc, argv, data.xsize, data.ysize);
          init_display_flag = 1;
        }

        /* ディスプレイ表示 */
        switch (n) {
        case 1:
```



```
          color_image_read(data, image_in);
          DisplayCImage(image_in[0], image_in[1],
                        image_in[2], IN, data.xsize, data.ysize);
          break;
        case 2:
          color_image_read(data, image_out);
          DisplayCImage(image_out[0], image_out[1],
                        image_out[2], OUT, data.xsize, data.ysize);
          break;
        case 3:
          color_image_read(data, image_work);
          DisplayCImage(image_work[0], image_work[1],
                        image_work[2], WORK, data.xsize, data.ysize);
          break;
        default:
          break;
        }
        break;

        /* 画像のファイル保存 */
      case 2:
        printf("  Which window (1:IN, 2:OUT, 3:WORK) ?   ");
        scanf("%d", &n);
        printf("  Filename (write) ? ");
        scanf("%s", destin);
        switch (n) {
        case 1:
          write_image_ppm(destin, data, image_in);
          break;
        case 2:
          write_image_ppm(destin, data, image_out);
          break;
        case 3:
          write_image_ppm(destin, data, image_work);
          break;
        default:
          break;
        }
        break;

        /* 画像のコピー */
      case 8:
        printf("  1:  IN --> OUT   \n");
        printf("  2:  IN --> WORK   \n");
        printf("  3:  OUT --> IN   \n");
        printf("  4:  OUT --> WORK   \n");
        printf("  5:  WORK --> IN   \n");
        printf("  6:  WORK --> OUT   ");
        scanf("%d", &n);
        switch (n) {
        case 1:
          for (i = 0; i < 3; i++) {
            image_copy(image_in[i], image_out[i], data.xsize, data.ysize);
          }
          DisplayCImage(image_out[0], image_out[1], image_out[2],
                        OUT, data.xsize, data.ysize);
          break;
        case 2:
          for (i = 0; i < 3; i++) {
            image_copy(image_in[i], image_work[i], data.xsize, data.ysize);
          }
          DisplayCImage(image_work[0], image_work[1], image_work[2],
                        WORK, data.xsize, data.ysize);
          break;
        case 3:
          for (i = 0; i < 3; i++) {
```

```
          image_copy(image_out[i], image_in[i], data.xsize, data.ysize);
        }
        DisplayCImage(image_in[0], image_in[1], image_in[2],
                      IN, data.xsize, data.ysize);
        break;
      case 4:
        for (i = 0; i < 3; i++) {
          image_copy(image_out[i], image_work[i], data.xsize, data.ysize);
        }
        DisplayCImage(image_work[0], image_work[1], image_work[2],
                      WORK, data.xsize, data.ysize);
        break;
      case 5:
        for (i = 0; i < 3; i++) {
          image_copy(image_work[i], image_in[i], data.xsize, data.ysize);
        }
        DisplayCImage(image_in[0], image_in[1], image_in[2],
                      IN, data.xsize, data.ysize);
        break;
      case 6:
        for (i = 0; i < 3; i++) {
          image_copy(image_work[i], image_out[i], data.xsize, data.ysize);
        }
        DisplayCImage(image_out[0], image_out[1], image_out[2],
                      OUT, data.xsize, data.ysize);
        break;
      default:
        break;
      }
      break;

      /* 画像のクリア */
    case 9:
      printf("  1:  Clear IN   \n");
      printf("  2:  Clear OUT   \n");
      printf("  3:  Clear WORK   ");
      scanf("%d", &n);
      switch (n) {
      case 1:
        for (i = 0; i < 3; i++) {
        image_clear(image_in[i], data.xsize, data.ysize);
      }
        DisplayCImage(image_in[0], image_in[1], image_in[2],
                      IN, data.xsize, data.ysize);
        break;
      case 2:
      for (i = 0; i < 3; i++) {
        image_clear(image_out[i], data.xsize, data.ysize);
      }
        DisplayCImage(image_out[0], image_out[1], image_out[2],
                      OUT, data.xsize, data.ysize);
      break;
      case 3:
        for (i = 0; i < 3; i++) {
          image_clear(image_work[i], data.xsize, data.ysize);
        }
        DisplayCImage(image_work[0], image_work[1], image_work[2],
                      WORK, data.xsize, data.ysize);
        break;
      default:
        break;
      }
      break;

      /* カメラワークのパラメータを求める、および 判定 */
    case 15:
```



```
      /* 読み込むファイル名、フレーム番号の入力 */
      printf("  Filename (read) ? ");
      scanf("%s", source);
      printf("  Frame Start Number ? ");
      scanf("%d", &start_frame);
      printf("  Frame End Number ? ");
      scanf("%d", &end_frame);
      printf("  Frame Interval ? ");
      scanf("%d", &interval_frame);

      for(j = start_frame; j <= (end_frame - interval_frame);
          j += interval_frame){

        /* 現在のフレーム画像の読み込み */
        /* sprintf(frame_num, "0000%d", j + interval_frame);
           while(strlen(frame_num) > 5){
           k = 0;
           while(frame_num[k] != '\0'){
           frame_num[k] = frame_num[k+1];
           k++;
           }
           } */
        sprintf(fname_current, "%s%d", source, j + interval_frame);
        read_image_ppm(fname_current, &data);
        if(init_display_flag == 0){
          color_image_init(image_in, image_out, image_work,
                           data.xsize, data.ysize);
          InitCDisplay(argc, argv, data.xsize, data.ysize);
          init_display_flag = 1;
        }
        color_image_read(data, image_in);
        DisplayCImage(image_in[0], image_in[1],
                      image_in[2], IN, data.xsize, data.ysize);

        /* 過去のフレームの画像の読み込み */
        /* sprintf(frame_num, "0000%d", j);
           while(strlen(frame_num) > 5){
           k = 0;
           while(frame_num[k] != '\0'){
           frame_num[k] = frame_num[k+1];
          k++;
          }
          } */
        sprintf(fname_past, "%s%d", source, j);
        read_image_ppm(fname_past,&data);
        color_image_read(data, image_work);
        DisplayCImage(image_work[0], image_work[1],
                      image_work[2], WORK, data.xsize, data.ysize);
        printf("    IN:            Current frame %s. \n", fname_current);
        printf("    WORK:    Previous frame %s. \n", fname_past);
        printf("    Block_Size:  %d (%d pixel)   %d (%d pixel).\n",
               block_number_x, data.xsize / block_number_x, block_number_y,
               data.ysize / block_number_y);

#ifdef PRACTICABLE_BLOCK
        /* ブロックの個数変更モード */
        printf("  Ready [y] or Cange Block_Number ?[s]  ");
        scanf("%s", c);
        if (c[0] == 's' || c[0] == 'S'){
          printf("Plese set Block Number.\n");

          scanf("%d %d",&block_number_x, &block_number_y);
          printf("    Block_Size:  %d (%d pixel)   %d (%d pixel).\n",
                 block_number_x, data.xsize / block_number_x,
                 block_number_y, data.ysize / block_number_y);
        }
```

```
        else if (c[0] != 'y' && c[0] != 'Y'){
          break;
        }
#endif
        /* 領域確保 */
        vx = (char *) malloc((size_t) block_number_x * block_number_y);
        vy = (char *) malloc((size_t) block_number_x * block_number_y);
        if (vx == NULL || vy == NULL) {
          printf("Memory not enough. \n");
          exit(-1);
        }

        v_angle =
          (double *)malloc(block_number_x * block_number_y * sizeof(double));
        if (v_angle == NULL) {
          printf("Memory not enough for v_angle. \n");
          exit(0);
        }

        /* 動きベクトル、２乗和平均、角度、角度の標準偏差を求める */
        draw_mvec(image_in, image_work, image_out,
                  data.xsize, data.ysize, vx, vy,
                  block_number_x, block_number_y);
        DisplayCImage(image_out[0], image_out[1], image_out[2],
                      WORK, data.xsize, data.ysize);
        power_average_motion_vector(vx, vy, block_number_x, block_number_y,
                                    &v_average);
        angle_motion_vector (vx, vy, block_number_x,
                             block_number_y, v_angle);
        average_angle_motion_vector(v_angle, block_number_x, block_number_y,
                                    &angle_average);
        standard_deviation_angle_motion_vector(v_angle, angle_average,
                                               block_number_x,
                                               block_number_y,
                                               &angle_standard_deviation);

        /* 動きベクトルのライン表示、および交点の最大濃度値を求める */
        draw_mvec_line_fullsize(image_out,vx, vy,
                                data.xsize, data.ysize,
                                block_number_x, block_number_y, ZOOM_OUT_IN);
        DisplayCImage(image_out[0], image_out[1], image_out[2],
                      OUT, data.xsize, data.ysize);
        find_point_intersection_max(vx, vy, data.xsize, data.ysize,
                                    block_number_x, block_number_y, &max,
                                    position_x, position_y, &count);
        printf("count : %d\n", count);
        for(i = 0; i < count; i++){
          find_value_position(vx, vy, data.xsize, data.ysize,
                              block_number_x, block_number_y,
                              position_x[i], position_y[i], &value[i]);
        }

        /* カメラワークの判定を行う */
        recognize_camera_work(v_average, angle_average,
                              angle_standard_deviation, max, position_x,
                              position_y, value, count, data.xsize,
                              data.ysize, &camera_work_cur);

        /* 結果をディスプレイ表示する */
        printf("v_average : %lf\n", v_average);
        printf("angle_average : %lf\n", angle_average);
        printf("angle_standard_deviation : %lf\n", angle_standard_deviation);
        printf("Max Value : %d\n", max);
        for(i = 0; i < count; i++){
          printf("Position x,y : %d %d\n", position_x[i], position_y[i]);
          printf("Value : %d\n\n", value[i]);
```



```
        }
        if(camera_work_cur & STANDSTILL){
          printf("Camera_work : StandStill\n");
        }
        if(camera_work_cur == NO_RECOGNIZE){
          printf("Camera_work : No Recognize\n");
        }
        if(camera_work_cur & PAN_UP){
          printf("Camera_work : Pan Up\n");
        }
        if(camera_work_cur & PAN_DOWN){
          printf("Camera_work : Pan Down\n");
        }
        if(camera_work_cur & PAN_RIGHT){
          printf("Camera_work : Pan Right\n");
        }
        if(camera_work_cur & PAN_LEFT){
          printf("Camera_work : Pan Left\n");
        }
        if(camera_work_cur & PAN_RIGHT_UP){
          printf("Camera_work : Pan Right_Up\n");
        }
        if(camera_work_cur & PAN_RIGHT_DOWN){
          printf("Camera_work : Pan Right_Down\n");
      }
        if(camera_work_cur & PAN_LEFT_UP){
          printf("Camera_work : Pan Left_Up\n");
        }
        if(camera_work_cur & PAN_LEFT_DOWN){
          printf("Camera_work : Pan Left_Down\n");
        }
        if(camera_work_cur & ZOOM_IN){
        printf("Camera_work : Zoom_In\n");
      }
        if(camera_work_cur & ZOOM_OUT){
          printf("Camera_work : Zoom_Out\n");
        }

        /* 領域の解放 */
        free(data.rdata);
        free(data.gdata);
        free(data.bdata);
        free(vx);
        free(vy);
        free(v_angle);
      }
      fclose(fp);
      break;

      /* 画像の連続表示（ムービー）*/
    case 16:
      /* 読み込む画像ファイル名、フレーム番号など入力 */
      printf("  Filename (read) ? ");
      scanf("%s", source);
      printf("  Frame Start Number ? ");
      scanf("%d", &start_frame);
      printf("  Frame End Number ? ");
      scanf("%d", &end_frame);
      printf("  Frame Interval ? ");
      scanf("%d", &interval_frame);

      for(j = start_frame; j <= (end_frame - interval_frame);
          j += interval_frame){
        /* 画像の読み込み、およびディスプレイ表示 */
        /* sprintf(frame_num, "0000%d", j + interval_frame);
           while(strlen(frame_num) > 5){
```

```
            k = 0;
            while(frame_num[k] != '\0'){
            frame_num[k] = frame_num[k+1];
            k++;
            }
            } */
          sprintf(fname_current, "%s%d", source, j);
          read_image_ppm(fname_current, &data);
          if(init_display_flag == 0){
            color_image_init(image_in, image_out, image_work,
                             data.xsize, data.ysize);
          InitCDisplay(argc, argv, data.xsize, data.ysize);
            init_display_flag = 1;
          }
          color_image_read(data, image_out);
          DisplayCImage(image_out[0], image_out[1],
                        image_out[2], OUT, data.xsize, data.ysize);
          printf("    Current frame %d. \n", j);

          /* 領域の解放 */
          free(data.rdata);
          free(data.gdata);
          free(data.bdata);
        }
        break;

      case 0:
        printf("  Bye-bye ........\n");
        break;
      default:
        printf("  Not defined number\n");
        break;
      }
    }
  }
  EndDisplay();
}
```

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <X11/Xlib.h>
#include        <X11/Xutil.h>
#include        "params.h"

#define         X_IN_POS        470     /* 入力画像の横方向表示位置     */
#define         Y_IN_POS        40      /* 入力画像の縦方向表示位置     */
#define         X_OUT_POS       470     /* 出力画像の横方向表示位置     */
#define         Y_OUT_POS       205     /* 出力画像の縦方向表示位置     */
#define         X_WORK_POS      470     /* ワーク画像の横方向表示位置   */
#define         Y_WORK_POS      370     /* ワーク画像の縦方向表示位置   */

#define GLEVEL  128     /* パレットを使用してモノクロ画像を表示する時の階調数*/
#define CLEVELR 6 /* パレットを使用してカラー画像を表示する時の階調数（Ｒ）  */
#define CLEVELG 6 /* パレットを使用してカラー画像を表示する時の階調数（Ｇ）  */
#define CLEVELB 6 /* パレットを使用してカラー画像を表示する時の階調数（Ｂ）  */

typedef struct _ximage {        /* 画像用構造体         */
  Window  win;
  XImage *im;
  Pixmap  pix;
}       XIMAGE;

Display *d;                     /* 表示画面用ポインタ           */
Visual *vis;                    /* 表示画面の情報               */
GC      gc;                     /* 描画環境                     */
int     depth;                  /* 画素当たりのビット数  */
int     pal_mode;               /* パレットモード               */
int     p_table[256];           /* パレット番号書き込み用テーブル       */

XIMAGE  ximage_in;              /* 入力画像の構造体             */
XIMAGE  ximage_out;             /* 出力画像の構造体             */
XIMAGE  ximage_work;            /* ワーク画像の構造体           */

InitDisplay(argc, argv, xsize, ysize)
/* 画面表示の初期化（モノクロ）  */
  int     argc;                 /* コマンドラインからの引数の数  */
  char    *argv[];              /* コマンドラインからの引数      */
  int     xsize;                /* 画像横サイズ */
  int     ysize;                /* 画像縦サイズ */
{
  d = XOpenDisplay(NULL);
  if (d == NULL) {
    printf("cannot open Display. \n");
    exit(0);
  }
  vis = DefaultVisual(d, 0);
  depth = DefaultDepth(d, 0);
  gc = XCreateGC(d, DefaultRootWindow(d), 0, 0);
  if (vis->class == PseudoColor && depth == 8) {
    pal_mode = GRAY;
    InitPal();
  }
  InitWindows(xsize, ysize);
}

InitCDisplay(argc, argv, xsize, ysize)
/* 画面表示の初期化（カラー）  */
  int     argc;                 /* コマンドラインからの引数の数  */
  char    *argv[];              /* コマンドラインからの引数      */
  int     xsize;                /* 画像横サイズ */
  int     ysize;                /* 画像縦サイズ */
{
  d = XOpenDisplay(NULL);
  if (d == NULL) {
```



```
    printf("cannot open Display. \n");
    exit(0);
  }
  vis = DefaultVisual(d, 0);
  depth = DefaultDepth(d, 0);
  gc = XCreateGC(d, DefaultRootWindow(d), 0, 0);
  if (vis->class == PseudoColor && depth == 8) {
    pal_mode = COLOR;
    InitPal();
  }
  InitWindows(xsize, ysize);
}

EndDisplay()
/* 画面表示の終了 */
{
}

DisplayImage(image, position, xsize, ysize)
/* 画像の表示（モノクロ）  */
  unsigned char *image;         /* 画像データ           */
  int     position;             /* 表示ウィンドウ       */
  int   xsize;                  /* 画像横サイズ */
  int   ysize;                  /* 画像縦サイズ */
{
  switch (position) {
  case 1:                       /* 入力画像用ウィンドウ                 */
    ColorDisplay(ximage_in, image, image, image, xsize, ysize, "IN");
    break;
  case 2:                       /* 出力画像用ウィンドウ                 */
    ColorDisplay(ximage_out, image, image, image, xsize, ysize, "OUT");
    break;
  case 3:                       /* ワーク画像用ウィンドウ               */
    ColorDisplay(ximage_work, image, image, image, xsize, ysize, "WORK");
    break;
  }
}

DisplayCImage(image_r, image_g, image_b, position, xsize, ysize)
/* 画像の表示（カラー）  */
  unsigned char *image_r;       /* 画像データ（Ｒ）             */
  unsigned char *image_g;       /* 画像データ（Ｇ）             */
  unsigned char *image_b;       /* 画像データ（Ｂ）             */
  int     position;             /* 表示ウィンドウ               */
  int     xsize;                /* 画像横サイズ */
  int     ysize;                /* 画像縦サイズ */
{
  switch (position) {
  case 1:                       /* 入力画像用ウィンドウ                 */
    ColorDisplay(ximage_in, image_r, image_g, image_b, xsize, ysize, "IN");
    break;
  case 2:                       /* 出力画像用ウィンドウ                 */
    ColorDisplay(ximage_out, image_r, image_g, image_b, xsize, ysize, "OUT");
    break;
  case 3:                       /* ワーク画像用ウィンドウ               */
    ColorDisplay(ximage_work, image_r, image_g, image_b, xsize, ysize, "WORK");
    break;
  }
}

InitPal()
/* パレットの初期化      */
{
  Colormap cmap;
  Status  result;
  XColor  xcol;
```

```
  int     i, npal;

  cmap = DefaultColormap(d, 0);
  if (pal_mode == GRAY) {
    for (i = 0; i < GLEVEL; i++) {
      xcol.red = (i * HIGH / (GLEVEL - 1) << 8) & 0xff00;
      xcol.green = (i * HIGH / (GLEVEL - 1) << 8) & 0xff00;
      xcol.blue = (i * HIGH / (GLEVEL - 1) << 8) & 0xff00;
      result = XAllocColor(d, cmap, &xcol);
      if (result)
        p_table[i] = xcol.pixel;
      else {
        printf("Palette not enough. \n");
        exit(0);
      }
    }
  } else {
    npal = CLEVELR * CLEVELG * CLEVELB;
    for (i = 0; i < npal; i++) {
      xcol.red = (((i / CLEVELG / CLEVELB) % CLEVELR) * HIGH / (CLEVELR - 1) << 8) &
0xff00;
      xcol.green = (((i / CLEVELB) % CLEVELG) * HIGH / (CLEVELG - 1) << 8) & 0xff00;
      xcol.blue = ((i % CLEVELB) * HIGH / (CLEVELB - 1) << 8) & 0xff00;
      result = XAllocColor(d, cmap, &xcol);
      if (result)
        p_table[i] = xcol.pixel;
      else {
        printf("Palette not enough. \n");
        exit(0);
      }
    }
  }
}

InitWindows(xsize, ysize)
/* ３つのウィンドウを初期化する  */
     int     xsize;                /* 画像横サイズ */
     int     ysize;                /* 画像縦サイズ */
{
  unsigned char *image;

  image = (unsigned char *) calloc(xsize * ysize, 1);
  if (image == NULL) {
    printf("Memory not enouph in InitWindows. \n");
    exit(0);
  }

  InitWindow(&ximage_in, X_IN_POS, Y_IN_POS, xsize, ysize);
  XMapWindow(d, ximage_in.win);
  DisplayImage(image, xsize, ysize, 1);
  InitWindow(&ximage_out, X_OUT_POS, Y_OUT_POS, xsize, ysize);
  XMapWindow(d, ximage_out.win);
  DisplayImage(image, xsize, ysize, 2);
  InitWindow(&ximage_work, X_WORK_POS, Y_WORK_POS, xsize, ysize);
  XMapWindow(d, ximage_work.win);
  DisplayImage(image, xsize, ysize, 3);
  free(image);
}

InitWindow(ximage, posi_x, posi_y, xsize, ysize)
/* ウィンドウを初期化する */
  XIMAGE *ximage;               /* 画像の構造体          */
  int     posi_x;               /* ウィンドウの横方向表示位置          */
  int     posi_y;               /* ウィンドウの縦方向表示位置          */
  int     xsize;                /* 画像横サイズ */
  int     ysize;                /* 画像縦サイズ */
```



```
{
  ximage->win = XCreateSimpleWindow(d, DefaultRootWindow(d),
    posi_x, posi_y, xsize, ysize, 0,
        WhitePixel(d, 0), 0x000000ff);
  if (vis->class == PseudoColor && depth == 8) {
    ximage->im = XCreateImage(d, DefaultVisual(d, 0), 8,
                              ZPixmap, 0, 0, xsize, ysize, 8, 0);
    (ximage->im)->data = (char *) calloc(1, xsize * ysize);
    if ((ximage->im)->data == NULL) {
      printf("Memory not enough. \n");
      exit(0);
    }
    ximage->pix = XCreatePixmap(d, ximage->win, xsize, ysize, 8);
  } else if (vis->class == TrueColor || vis->class == DirectColor) {
    ximage->im = XCreateImage(d, DefaultVisual(d, 0), depth,
                              ZPixmap, 0, 0, xsize, ysize, 32, 0);
    (ximage->im)->data = (char *) calloc(1, xsize * ysize * 4);
    if ((ximage->im)->data == NULL) {
      printf("Memory not enough. \n");
      exit(0);
    }
    ximage->pix = XCreatePixmap(d, ximage->win, xsize, ysize, depth);
  } else {
    printf("cannot support your X-Window. \n");
    exit(0);
  }
  XPutImage(d, ximage->pix, gc, ximage->im, 0, 0, 0, 0, xsize, ysize);
  XSetWindowBackgroundPixmap(d, ximage->win, ximage->pix);
  XFlush(d);
}

ColorDisplay(ximage, imager, imageg, imageb, xsize, ysize, wtitle)
/* ウィンドウにカラー画像を表示する      */
  XIMAGE  ximage;               /* 画像構造体 */
  unsigned char *imager;        /* 画像データ（Ｒ）*/
  unsigned char *imageg;        /* 画像データ（Ｇ）*/
  unsigned char *imageb;        /* 画像データ（Ｂ）*/
  int     xsize;                /* 画像横サイズ */
  int     ysize;                /* 画像縦サイズ */
  char    wtitle[];             /* ウィンドウのタイトル  */
{
  int    *pd;
  unsigned char *pdc;
  int     i, j;
  unsigned char *psr0, *psg0, *psb0;
  unsigned char *psr, *psg, *psb;

  XStoreName(d, ximage.win, wtitle);
  if (vis->class == PseudoColor && depth == 8) {
    pdc = (unsigned char *) ximage.im->data;
    if (pal_mode == GRAY) {
      psg = imageg;
      for (i = 0; i < xsize * ysize; i++) {
        j = *psg++ * (GLEVEL - 1) / HIGH;
        *pdc++ = p_table[j];
      }
    } else {
      psr0 = psr = (unsigned char *) malloc((size_t) xsize * ysize);
      psg0 = psg = (unsigned char *) malloc((size_t) xsize * ysize);
      psb0 = psb = (unsigned char *) malloc((size_t) xsize * ysize);
      if (psr == NULL || psg == NULL || psb == NULL) {
        printf("Memory not enough in ColorDisplay. \n");
        exit(0);
      }
      dither(imager, psr, CLEVELR, xsize, ysize);
      dither(imageg, psg, CLEVELG, xsize, ysize);
```

```
      dither(imageb, psb, CLEVELB, xsize, ysize);
      for (i = 0; i < xsize * ysize; i++) {
        j = (int) *psr++ * CLEVELG * CLEVELB
          + (int) *psg++ * CLEVELB
          + (int) *psb++;
        *pdc++ = p_table[j];
      }
      free(psr0);
      free(psg0);
      free(psb0);
    }
  }
  if (vis->class == TrueColor || vis->class == DirectColor) {
    pd = (int *) ximage.im->data;
    psg = imageg;
    if (vis->blue_mask == 255) {
      psr = imager;
      psb = imageb;
    } else {
      psb = imager;
      psr = imageb;
    }
    for (i = 0; i < xsize * ysize; i++) {
      /*
       * 希望の色にならない時は, psr, ps
       * g, psbの順番を変更してみてください
       */
      *pd++ = ((int) *psb++ & 0xff)
        + ((int) *psg++ << 8 & 0xff00)
        + ((int) *psr++ << 16 & 0xff0000);
    }
  }
  XPutImage(d, ximage.pix, gc, ximage.im, 0, 0, 0, 0, xsize, ysize);
  XCopyArea(d, ximage.pix, ximage.win, gc, 0, 0, xsize, ysize, 0, 0);
  XRaiseWindow(d, ximage.win);
  XFlush(d);
}
```

```
#include "params.h"

static int *error, *error1,
        *error2;                /*   誤差配列     */

dither(image_in, image_out, nq, xsize, ysize)
/*   平均誤差最小ディザ法により画像を量子化する.    */
  unsigned char *image_in;      /* 入力画像配列          */
  unsigned char *image_out;     /* 出力画像配列          */
  int     nq;                   /* 階調数 */
  int     xsize;
  int     ysize;
{
  int     i, j;
  int     d;

  /* 領域確保 */
  image_in = (unsigned char *) malloc(xsize * ysize * sizeof(unsigned char));
  image_out = (unsigned char *) malloc(xsize * ysize * sizeof(unsigned char));
  error = (int *) malloc(xsize * sizeof(int));
  error1 = (int *) malloc(xsize * sizeof(int));
  error2 = (int *) malloc(xsize * sizeof(int));

  for (i = 0; i < xsize; i++) {
    d_quantize((int) image_in[i], &image_out[i], nq);
    error1[i] = (int) image_in[i] - d_iquantize(image_out[i], nq);
  }
  for (i = 0; i < xsize; i++) {
    d_quantize((int) image_in[xsize * i], &image_out[xsize * i], nq);
    error2[i] =
      (int) image_in[xsize * i] - d_iquantize(image_out[xsize * i], nq);
  }
  for (i = 2; i < ysize; i++) {
    d_quantize((int) image_in[xsize * i], &image_out[xsize * i], nq);
    error[0] =
      (int) image_in[xsize * i] - d_iquantize(image_out[xsize * i], nq);
    d_quantize((int) image_in[xsize * i + 1], &image_out[xsize * i + 1], nq);
    error[1] =
      (int)image_in[xsize * i + 1] - d_iquantize(image_out[xsize * i + 1], nq);
    for (j = 2; j < xsize - 2; j++) {
      d = (error1[j - 2] + error1[j - 1] * 3 + error1[j] * 5
           + error1[j + 1] * 3 + error1[j + 2]
           + error2[j - 2] * 3 + error2[j - 1] * 5 + error2[j] * 7
           + error2[j + 1] * 5 + error2[j + 2] * 3
           + error[j - 2] * 5 + error[j - 1] * 7) / 48;
      d_quantize((int)image_in[xsize * i + j] + d,
                 &image_out[xsize * i + j], nq);
      error[j] = (int) image_in[xsize * i + j] + d
        - d_iquantize(image_out[xsize * i + j], nq);
    }
    d_quantize((int) image_in[i * xsize + xsize - 2],
               &image_out[i * xsize + xsize - 2], nq);
    error[xsize - 2] =
      (int) image_in[i * xsize + xsize - 2]
      - d_iquantize(image_out[i * xsize + xsize - 2], nq);
    d_quantize((int) image_in[i * xsize + xsize - 1],
               &image_out[i * xsize + xsize - 1], nq);
    error[xsize - 1] =
      (int) image_in[i * xsize + xsize - 1]
      - d_iquantize(image_out[i * xsize + xsize - 1], nq);
    for (j = 0; j < xsize; j++) {
      error1[j] = error2[j];
      error2[j] = error[j];
    }
  }
```



```
  free(image_in);
  free(image_out);
  free(error);
  free(error1);
  free(error2);
}

d_quantize(in, pout, nq)
/* 量子化する     */
  int     in;                   /* 入力          */
  unsigned char *pout;          /* 出力          */
  int     nq;                   /* 階調数        */
{
  *pout = (float) in *(nq - 1) / HIGH + 0.5;
  if (*pout < 0)
    *pout = 0;
  if (*pout > nq - 1)
    *pout = nq - 1;
}

d_iquantize(in, nq)
/* 量子化データを画像データに変換する     */
  unsigned char in;             /* 入力          */
  int     nq;                   /* 階調数        */
{
  return ((int) in * HIGH / (nq - 1));
}
```

```
#include        <stdio.h>
#include        <stdlib.h>
#include        "params.h"
#include        "mc.h"

/*------------------------------------------------------------------------*/
/*              img構造体からデータを画像配列に読み込む                   */
/*------------------------------------------------------------------------*/
color_image_read(data, image)
     IMAGE    data;                  /* 画像データ */
     unsigned char *image[3];        /* 画像配列 */
{
  image_copy(data.rdata, image[0], data.xsize, data.ysize);
  image_copy(data.gdata, image[1], data.xsize, data.ysize);
  image_copy(data.bdata, image[2], data.xsize, data.ysize);
}

/*------------------------------------------------------------------------*/
/*              ppm(raw)形式のファイルをimg構造体に読み込む                */
/*------------------------------------------------------------------------*/
read_image_ppm(filename, data)
     char      *filename;            /* 画像ファイル名 */
     IMAGE     *data;                /* 画像データ */
{
  FILE          *fp;                /* 画像ファイル用ファイルポインタ */
  char           fname[256];        /* 画像ファイル名 */
  char           buf[512];          /* ファイル読み込み用配列 */
  int            size;              /* 画像サイズ */
  unsigned long  i = 0;

  /* ファイルオープン */
  sprintf(fname, "/raid5-5/miyazaki/%s.rppm", filename);
  if((fp = fopen(fname, "rb")) == NULL){
    printf("Cannot open file.\n");
    exit(0);
  }

  /* 画像形式（モノラル : "P5"、カラー : "P6"）を読み込む */
  fgets(buf, 512, fp);
  if(strncmp("P6", buf, 2) == 0) data->type = COLOR;
  else if(strncmp("P5", buf, 2) == 0) data->type = GRAY;
  else {
    puts("unknown file type");
    exit(0);
  }

  do{
    fgets(buf, 512, fp);
  }while(buf[0] == '#');

  /* データサイズと階調度を読み込む */
  sscanf(buf, "%d %d", &(data->xsize), &(data->ysize));
  fgets(buf, 512, fp);
  sscanf(buf, "%d", &(data->depth));

  /* 領域確保 */
  size = data->xsize * data->ysize;
  data->rdata = (unsigned char *)malloc(size * sizeof(unsigned char));
  if(data->rdata == NULL) {
    printf("Memory not enough red. \n");
    exit(0);
  }
  if(data->type == COLOR){
    data->gdata = (unsigned char *)malloc(size * sizeof(unsigned char));
    data->bdata = (unsigned char *)malloc(size * sizeof(unsigned char));
    if ((data->gdata == NULL) || (data->bdata == NULL)) {
```



```
      printf("Memory not enough green bule. \n");
      exit(0);
    }

    /* 画像データを読み込む */
    while(i < size){
      data->rdata[i] = fgetc(fp);
      data->gdata[i] = fgetc(fp);
      data->bdata[i] = fgetc(fp);
      i++;
    }
  }
  else {
    fread(data->rdata, size, 1, fp);
  }

  /* ファイルを閉じる */
  fclose(fp);
}

/*------------------------------------------------------------------------*/
/*       img構造体の中身をppm(raw)形式のファイルに書き出す                */
/*------------------------------------------------------------------------*/
write_image_ppm(filename, data, image)
     char      *filename;            /* 出力ファイル名 */
     IMAGE     data;                 /* 入力画像データ */
     unsigned  char *image[3];       /* 出力画像 */
{
  FILE      *fp;                    /* ファイルポインタ */
  char       fname[256];            /* 出力ファイル名 */
  int        size;                  /* 画像サイズ */
  int        j,i=0;

  /* ファイルオープン */
  sprintf(fname, "Imagedata/PPM/%s.rppm", filename);
  if((fp = fopen(fname, "wb")) == NULL){
    fprintf(stderr, "cannot open file: %s\n", fname);
    exit(0);
  }

  /* 画像形式の出力 */
  if(data.type == COLOR) {
    fprintf(fp,"P6\n");
  }
  else {
    fprintf(fp,"P5\n");
  }

  /* 画像サイズ、階調度出力 */
  fprintf(fp,"# \n");
  fprintf(fp,"%d %d\n", data.xsize, data.ysize);
  fprintf(fp,"%d\n", data.depth);
  size = data.xsize * data.ysize;

  /* 画像データの出力 */
  if(data.type == GRAY)
    fwrite(image[0], size, 1, fp);
  else while(i < size){
    for(j = 0; j < 3; j++) {
      fputc(image[j][i], fp);
    }
    i++;
  }

  /* ファイルクローズ */
  fclose(fp);
```

```
}

/*------------------------------------------------------------------------*/
/*                カラー画像データ領域の確保                              */
/*------------------------------------------------------------------------*/
color_image_init(image_in, image_out, image_work, xsize, ysize)

     unsigned char    *image_in[3];       /* 入力画像 */
     unsigned char    *image_out[3];      /* 出力画像 */
     unsigned char    *image_work[3];     /* ワーク画像 */
     int               xsize;             /* 画像横サイズ */
     int               ysize;             /* 画像縦サイズ */
{
  int     i;
  int     size;                          /* サイズ */

  /* 領域確保 */
  size = xsize * ysize;
  for (i = 0; i < 3; i++) {
    image_in[i] = (unsigned char *) malloc(size * sizeof(unsigned char));
    image_out[i] = (unsigned char *) malloc(size * sizeof(unsigned char));
    image_work[i] = (unsigned char *) malloc(size * sizeof(unsigned char));

    if ((image_in[i] == NULL) || (image_out[i] == NULL) ||
        (image_work[i] == NULL)) {
      printf("Memory not enough in color_image_init. \n");
      exit(0);
    }
  }
}

/*------------------------------------------------------------------------*/
/*                モノクロ画像データをコピーする                          */
/*------------------------------------------------------------------------*/
image_copy(image_in, image_out, xsize, ysize)
     unsigned char    *image_in;         /* 入力画像配列 */
     unsigned char    *image_out;        /* 出力画像配列 */
     int               xsize;            /* 画像横サイズ */
     int               ysize;            /* 画像縦サイズ */
{
  long    i;

  for (i = 0; i < (long) ysize * xsize; i++)
    image_out[i] = image_in[i];
}

/*------------------------------------------------------------------------*/
/*                モノクロ画像データをクリアする                          */
/*------------------------------------------------------------------------*/
image_clear(image, xsize, ysize)
     unsigned char   *image;             /* 画像データ */
     int              xsize;             /* 画像横サイズ */
     int              ysize;             /* 画像縦サイズ */
{
  long    i;

  for (i = 0; i < (long) ysize * xsize; i++)
    image[i] = 0;
}

/*------------------------------------------------------------------------*/
/*                モノクロ画像データを塗り潰す                            */
/*------------------------------------------------------------------------*/
image_background(image, xsize, ysize, val)
     unsigned char    *image;            /* 画像データ */
     int               xsize;            /* 画像横サイズ */
```



```
     int               ysize;            /* 画像縦サイズ */
     int               val;              /* 書き込む値 */
{
  long i;
  for(i = 0; i < (long) ysize * xsize; i++){
    image[i] = val;
  }
}
```

```
#define DX_MAX          8           /* ブロックを動かす最大量（x方向） */
#define DY_MAX          8           /* ブロックを動かす最大量（y方向） */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "params.h"
#include "mc.h"

/*--------------------------------------------------------------------------*/
/*                          動きベクトルを求める                            */
/*--------------------------------------------------------------------------*/
draw_mvec(image_pr, image_ct, image_rt, xsize, ysize, vx, vy,
          block_number_x, block_number_y)
     unsigned char *image_pr[3];        /* 前フレームの画像       */
     unsigned char *image_ct[3];        /* 現フレームの画像       */
     unsigned char *image_rt[3];        /* 出力画像               */
     int      xsize;                    /* 画像横サイズ */
     int      ysize;                    /* 画像縦サイズ */
     char     *vx;                      /* 動きベクトル (x方向) */
     char     *vy;                      /* 動きベクトル (y方向) */
     int      block_number_x;           /* ブロックの個数 (x方向) */
     int      block_number_y;           /* ブロックの個数 (y方向) */
{
  int i;

  mvec(image_pr[1], image_ct[1], vx, vy, xsize, ysize,
       block_number_x, block_number_y);
  for(i = 0; i < 3; i++){
    /* 現在のフレーム画像を出力画像にコピーする */
    /* image_copy(image_ct[i], image_rt[i], xsize, ysize); */
    /* 出力画像をクリアする */
    image_clear(image_rt[i], xsize, ysize);
  }
  for(i = 0; i < 3; i++){
    mvec_line(image_rt[i], vx, vy, xsize, ysize,
              block_number_x, block_number_y);
  }
}

/*--------------------------------------------------------------------------*/
/*                          動きベクトルを描画する                          */
/*--------------------------------------------------------------------------*/
mvec_line(image, vx, vy, xsize, ysize, block_number_x, block_number_y)
     unsigned char  *image;             /* 画像 */
     signed char    *vx;                /* 動きベクトルのｘ方向成分 */
     signed char    *vy;                /* 動きベクトルのｙ方向成分 */
     int            xsize;              /* 画像横サイズ */
     int            ysize;              /* 画像縦サイズ */
     int            block_number_x;     /* ブロックの個数 (x方向) */
     int            block_number_y;     /* ブロックの個数 (y方向) */
{
  int     i, j;
  int     xs, ys;           /* 始点の位置 */
  int     xe, ye;           /* 終点の位置 */

  for (i = 0; i < block_number_y; i++) {
    for (j = 0; j < block_number_x; j++) {
      /* 始点をブロックの中央に設定する */
      xs = j * (xsize / block_number_x) + (xsize / block_number_x) / 2;
      ys = i * (ysize / block_number_y) + (ysize / block_number_y) / 2;

      /* 終点を始点から動きベクトルを足した位置に設定する */
      xe = xs + vx[i * block_number_x + j];
      ye = ys + vy[i * block_number_y + j];

      /* 終点が画像からハズレたら飛ばす */
      if (xe < 0 || ye < 0)
        continue;
```



```
      if (xe >= xsize || ye >= ysize)
        continue;

      /* 始点から終点へ線を描く */
      line_image(image, xs, ys, xe, ye, 255, xsize, ysize);
    }
  }
}

/*--------------------------------------------------------------------------*/
/*                              線を引く                                    */
/*--------------------------------------------------------------------------*/
line_image(image, xs, ys, xe, ye, val, xsize, ysize)
     unsigned char *image;              /* 画像 */
     int      xs, ys;                   /* 始点 */
     int      xe, ye;                   /* 終点 */
     int      val;                      /* 書き込む値 */
     int      xsize;                    /* 画像横サイズ */
     int      ysize;                    /* 画像縦サイズ */
{
  int     x, y, x1, x2, y1, y2, i;
  float   dx, dy;

  if (abs(xs-xe) >= abs(ys-ye)){
    if(xe >= xs){
      x1 = xs;
      x2 = xe;
      y1 = ys;
      y2 = ye;
    }
    else {
      x1 = xe;
      x2 = xs;
      y1 = ye;
      y2 = ys;
    }
    if (x2 == x1){
      dy = 0;
    }
    else{
      dy = (float) (y2 - y1) / (x2 - x1);
    }
    for (x = x1; x <= x2; x++) {
      y = (int) (y1 + dy * (x - x1));
      image[y * xsize + x] = (unsigned char) val;
    }
  }
  else {
    if(ye >= ys){
      y1 = ys;
      y2 = ye;
      x1 = xs;
      x2 = xe;
    }
    else{
      y1 = ye;
      y2 = ys;
      x1 = xe;
      x2 = xs;
    }
    if (y2 == y1)
      dx = 0;
    else
      dx = (float) (x2 - x1) / (y2 - y1);
    for (y = y1; y <= y2; y++) {
      x = (int) (x1 + dx * (y - y1));
```

```
      image[y * xsize + x] = (unsigned char) val;
    }
  }
}
```

```
#include "params.h"
#include "mc.h"

/*--------------------------------------------------------------------------*/
/*                          動きベクトルを求める                            */
/*--------------------------------------------------------------------------*/
mvec(image_pr, image_ct, vx, vy, xsize, ysize, block_number_x, block_number_y)
     unsigned char  *image_pr;        /* 前フレームの画像 */
     unsigned char  *image_ct;        /* 現フレームの画像 */
     char           *vx;              /* 動きベクトルのｘ方向成分 */
     char           *vy;              /* 動きベクトルのｙ方向成分 */
     int             xsize;           /* 画像横サイズ */
     int             ysize;           /* 画像縦サイズ */
     int             block_number_x;  /* ブロックの個数（x方向）*/
     int             block_number_y;  /* ブロックの個数（y方向）*/
{
  int     i, j;
  int     dx, dy;                     /* ブロックの移動量 */
  int     ddx, ddy;                   /* 差分が最小時の移動量 */
  int     x, y;                       /* 比較する画素の位置座標 */
  int     xs, ys;                     /* 現フレームの画素配列番号 */
  int     xd, yd;                     /* 過去フレームの画素配列番号 */
  float   d, min;                     /* 差分の総和、最小値 */

  /* ブロックループ */
  for (i = 0; i < block_number_y; i++) {
    printf("   Line %d\r", i);
    for (j = 0; j < block_number_x; j++) {
      /* ブロック移動量変化ループ */
      for (dy = -DY_MAX; dy <= DY_MAX; dy++) {
        for (dx = -DX_MAX; dx <= DX_MAX; dx++) {
          d = 0;
          /* ブロック中の画素ループ */
          for (y = 0; y < (ysize / block_number_y); y++) {
            /* 現フレームと過去フレームの比較画素番号を求める */
            yd = i * (ysize / block_number_y) + y;
            ys = yd + dy;
            if (ys < 0)
              ys = 0;
            if (ys > ysize - 1)
              ys = ysize - 1;
            for (x = 0; x < (xsize / block_number_x); x++) {
              xd = j * (xsize / block_number_x) + x;
              xs = xd + dx;
              if (xs < 0)
                xs = 0;
              if (xs > xsize - 1)
                xs = xsize - 1;
              /* 差分を求める */
              d += abs(image_ct[yd * xsize + xd] - image_pr[ys * xsize + xs]);
            }
          }

          /* 最初に求めた値を初期値として代入しておく */
          if (dx == -DX_MAX && dy == -DY_MAX) {
            min = d;
            ddx = dx;
            ddy = dy;
          }
          /* 差分の最小値とそのときのブロックの移動量を求める */
          if (d < min) {
            min = d;
            ddx = dx;
            ddy = dy;
          }
        }
```



```
      }
      /* 差分が最小な時の移動量を動きベクトルとして代入する */
      vx[i * block_number_x + j] = ddx;
      vy[i * block_number_x + j] = ddy;
    }
  }
}
```

```
#include  <stdio.h>
#include  <math.h>
#include  "params.h"

#define RAD_DEG(x) ((x) * 180 / 3.14159265358979323846)  /* ラジアン → 度 */
#define ERROR_ANGLE   -360

/*-------------------------------------------------------------------------*/
/*                    メッセージ出力用マクロ定義                           */
/*-------------------------------------------------------------------------*/
/* #define PRINT_MEASSEGE_MOTIONVEC_OPERATE */
#ifdef  PRINT_MEASSEGE_MOTIONVEC_OPERATE
#define  mesprint(x)          printf("%s", x)
#define  valprintf(x, y, z)   printf("%s : %lf%s", x, y, z)
#define  valprintd(x, y, z)   printf("%s : %d%s", x, y, z)
#else
#define  mesprint(x)         /* */
#define  valprintf(x, y, z)  /* */
#define  valprintd(x, y, z)  /* */
#endif

/*-------------------------------------------------------------------------*/
/*                    動きベクトルの２乗和平均を求める                     */
/*-------------------------------------------------------------------------*/
power_average_motion_vector(vx, vy, block_number_x, block_number_y, v_average)
     signed char    *vx;                /* 動きベクトルのｘ成分 */
     signed char    *vy;                /* 動きベクトルのｙ成分 */
     int            block_number_x;     /* ブロックの個数（ｘ成分）*/
     int            block_number_y;     /* ブロックの個数（ｙ成分）*/
     double         *v_average;         /* 動きベクトルの平均 */
{
  int       i;
  int       vx_power_sum, vy_power_sum;      /* 角度の２乗和 */
  double    vx_average, vy_average;          /* 角度の平均 */

  mesprint("In function of power_average_motion_vector\n");

  vx_power_sum = 0.0;
  vy_power_sum = 0.0;

  for(i = 0; i < block_number_x * block_number_y; i++){
    vx_power_sum += vx[i] * vx[i];
    vy_power_sum += vy[i] * vy[i];
    valprintd("vx", vx[i], "\t");
    valprintd("vy", vy[i], "\n");
    valprintd("vx_power_sum", vx_power_sum, "\t");
    valprintd("vy_power_sum", vy_power_sum, "\n");
  }

  vx_average = (double) vx_power_sum /
    (double) (block_number_x * block_number_y);
  vy_average = (double) vy_power_sum /
    (double) (block_number_x * block_number_y);

  valprintf("vx_average(power)", vx_average, "\t");
  valprintf("vy_average(power)", vy_average, "\n");

  *v_average = sqrt(vx_average + vy_average);

  valprintf("v_average", *v_average, "\n");
  mesprint("In function of power_average_motion_vector\n\n");
}

/*-------------------------------------------------------------------------*/
/*                    動きベクトルから角度を求める                         */
/*-------------------------------------------------------------------------*/
```



```
angle_motion_vector(vx, vy, block_number_x, block_number_y, v_angle)
     signed char    *vx;                /* 動きベクトルのｘ成分 */
     signed char    *vy;                /* 動きベクトルのｙ成分 */
     int            block_number_x;     /* ブロックの個数（ｘ成分）*/
     int            block_number_y;     /* ブロックの個数（ｙ成分）*/
     double         *v_angle;           /* 動きベクトルの角度 */
{
  int i;

  mesprint("In function of angle_motion_vector\n");

  /* ベクトルから角度を求める */
  for(i = 0; i < block_number_x * block_number_y; i++){
    valprintd("vx", vx[i], "\t");
    valprintd("vy", vy[i], "\n");
    /* ０ベクトルの時、エラーをいれる */
    if((vx[i] == 0) && (vy[i] == 0)){
      v_angle[i] = ERROR_ANGLE;
    }
    else{
      v_angle[i] = atan2(vy[i], vx[i]);
      v_angle[i] = RAD_DEG(v_angle[i]);
    }
    valprintf("v_angle(-180-0-180)", v_angle[i], "\n");
  }

  mesprint("Out function of angle_motion_vector\n\n");
}

/*-------------------------------------------------------------------------*/
/*                    動きベクトルの角度平均を求める                       */
/*-------------------------------------------------------------------------*/
average_angle_motion_vector(v_angle, block_number_x, block_number_y,
                            angle_average)
     double   *v_angle;               /* 動きベクトルの角度 */
     int      block_number_x;         /* ブロックの個数（ｘ方向）*/
     int      block_number_y;         /* ブロックの個数（ｙ方向）*/
     double   *angle_average;         /*       角度平均          */
{
  int i;
  double angle_sum;                  /* 角度の合計 */
  double angle_power_sum;            /* 角度の２乗和 */
  double angle_power_average;        /* 角度の２乗和平均 */
  int  count;                        /* 計算された角度の数 */

  mesprint("In function of average_angle_motion_vector\n");

  angle_sum = 0.0;
  angle_power_sum = 0.0;

  count = 0;
  for(i = 0; i < block_number_x * block_number_y; i++){
    if(v_angle[i] == ERROR_ANGLE){
      continue;
    }
    angle_power_sum += v_angle[i] * v_angle[i];
    count++;
  }
  angle_power_average = angle_power_sum / count;
  angle_power_average = sqrt(angle_power_average);

  valprintf("angle_power_average", angle_power_average, "\n");

  count = 0;
  for(i = 0; i < block_number_x * block_number_y; i++){
    valprintf("v_angle", v_angle[i], "\n");
```

```
    if(v_angle[i] == ERROR_ANGLE){
      continue;
    }
    if((angle_power_average > 90) && (v_angle[i] < 0)){
      angle_sum += v_angle[i] + 360;
    }
    else{
      angle_sum += v_angle[i];
    }
    count++;
    valprintf("angle_sum", angle_sum, "\n");
  }

  if(count == 0){
    *angle_average = 0.0;
  }
  else{
    *angle_average = angle_sum / count;
    if(*angle_average > 180){
      *angle_average -= 360;
    }
  }
  valprintf("angle_average", *angle_average, "\n");

  mesprint("Out function of average_angle_motion_vector\n");
}

/*--------------------------------------------------------------------------*/
/*                  動きベクトルの角度標準偏差を求める                      */
/*--------------------------------------------------------------------------*/
standard_deviation_angle_motion_vector(v_angle, angle_average, block_number_x,
                                       block_number_y,
                                       angle_standard_deviation)
     double     *v_angle;                    /* 動きベクトルの角度 */
     double      angle_average;              /* 動きベクトルの角度平均 */
     int         block_number_x;             /* ブロックの個数（x方向）*/
     int         block_number_y;             /* ブロックの個数（y方向）*/
     double     *angle_standard_deviation;   /* 角度の標準偏差 */
{
  int i;
  double     angle_subtraction;        /* 角度と角度平均との差 */
  double     angle_subtraction_sum;    /* 上述の和 */
  int        count;                    /* 合計された数 */

  mesprint("In function of standard_deviation_angle_motion_vector\n");

  angle_subtraction_sum = 0.0;
  count = 0;
  for(i = 0; i < block_number_x * block_number_y; i++){
    valprintf("v_angle", v_angle[i], "\t");
    valprintf("angle_average", angle_average, "\n");
    /* 角度が不定なら、計算からはずす */
    if(v_angle[i] == ERROR_ANGLE){
      continue;
    }
    if(v_angle[i] >= angle_average){
      angle_subtraction = v_angle[i] - angle_average;
    }
    else{
      angle_subtraction = angle_average - v_angle[i];
    }
    if(angle_subtraction > 180){
      angle_subtraction = 360 - angle_subtraction;
    }
    count++;
    valprintf("angle_subtraction", angle_subtraction, "\n");
```



```
    angle_subtraction_sum += angle_subtraction;
    valprintf("angle_subtraction_sum", angle_subtraction_sum, "\n\n");
  }

  /* 角度がすべて不定だったら、偏差を90にする */
  if(count == 0){
    *angle_standard_deviation = 90;
  }
  else{
    *angle_standard_deviation = angle_subtraction_sum / count;
  }
  valprintf("angle_standard_deviation", *angle_standard_deviation, "\n");

  mesprint("Out function of standard_deviation_angle_motion_vector\n\n");
}

/*--------------------------------------------------------------------------*/
/*                              TEST_MAIN                                   */
/*--------------------------------------------------------------------------*/
/* #define TEST_MAIN_MOVEVEC_OPERATE */
#ifdef  TEST_MAIN_MOVEVEC_OPERATE
main(argc, argv)
     int   argc;
     char  **argv;
{
  int       *vx_dummy;                  /* 動きベクトルx成分（入力用）*/
  int       *vy_dummy;                  /* 動きベクトルy成分（入力用）*/
  char      *vx;                        /* 動きベクトルx成分（引き数用）*/
  char      *vy;                        /* 動きベクトルy成分（引き数用）*/
  double     vx_average;                /* 動きベクトルの平均（x成分）*/
  double     vy_average;                /* 動きベクトルの平均（y成分）*/
  double    *v_angle;                   /* 動きベクトルの角度 */
  double     angle_average;             /* 動きベクトルの平均角度 */
  double     standard_deviation_angle;  /* 動きベクトルの角度の標準偏差 */
  int        block_number_x;            /* ブロックの個数（x方向）*/
  int        block_number_y;            /* ブロックの個数（y方向）*/
  int        i, j;

  printf("TEST : power_average_motion_vector & angle_motion_vector\n");
  printf("Input original datasize -> block_xsize block_ysize\n");
  scanf("%d %d", &block_number_x, &block_numbe_y);

  /* 領域確保 */
  vx_dummy = (int *) malloc (block_number_x * block_number_y * sizeof(int));
  vy_dummy = (int *) malloc (block_number_x * block_number_y * sizeof(int));
  if((vx_dummy == NULL) || (vy_dummy == NULL)){
    printf("Memory not enough for vx_dummy, vy_dummy\n");
    exit(0);
  }

  vx = (char *) malloc(block_number_x * block_number_y * sizeof(char));
  vy = (char *) malloc(block_number_x * block_number_y * sizeof(char));
  if((vx == NULL) || (vy == NULL)){
    printf("Memory not enough for vx, vy\n");
    exit(0);
  }

  v_angle = (double *)malloc(block_number_x * block_number_y * sizeof(double));
  if (v_angle == NULL) {
    printf("Memory not enough for v_angle. \n");
    exit(0);
  }

  /* 動きベクトルの入力 */
  printf("Input original data\n");
  for(i = 0; i < block_number_y; i++){
```

```
    for(j = 0; j < block_number_x; j++){
      printf("vx[%d][%d] vy[%d][%d] : ", i, j, i, j);
      scanf("%d %d", &vx_dummy[i * block_number_x + j],
            &vy_dummy[i * block_number_x + j]);
      vx[i * block_number_x + j] = vx_dummy[i * block_number_x + j];
      vy[i * block_number_x + j] = vy_dummy[i * block_number_x + j];
    }
  }
  /* 動きベクトルの絶対値平均 */
  power_average_motion_vector(vx, vy, block_number_x, block_number_y,
                              &vx_average, &vy_average);
  printf("average x,y: %lf %lf\n\n", vx_average, vy_average);

  /* 動きベクトルの角度 → 角度平均 → 標準偏差 */
  angle_motion_vector (vx, vy, block_number_x, block_number_y, v_angle);
  average_angle_motion_vector(v_angle, block_number_x, block_number_y,
                              &angle_average);
  standard_deviation_angle_motion_vector(v_angle, angle_average,
                                         block_number_x,block_number_y,
                                         &standard_deviation_angle);
  printf("angle_average : %lf\n", angle_average);
  printf("angle_standard_deviation : %lf\n", standard_deviation_angle);
}
#endif
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "params.h"
#include "mc.h"
#include "recognize.h"

/*------------------------------------------------------------------------------*/
/*                    メッセージ出力用マクロ定義                                */
/*------------------------------------------------------------------------------*/
/* #define PRINT_MEASSEGE_RECOGNIZE_ZOOM */
#ifdef  PRINT_MEASSEGE_RECOGNIZE_ZOOM
#define  mesprint(x)           printf("%s", x)
#define  valprintf(x, y, z)    printf("%s : %lf%s", x, y, z)
#define  valprintd(x, y, z)    printf("%s : %d%s", x, y, z)
#else
#define  mesprint(x)         /* */
#define  valprintf(x, y, z)  /* */
#define  valprintd(x, y, z)  /* */
#endif

/*------------------------------------------------------------------------------*/
/*              最大画素値 および その位置を求める関数                          */
/*------------------------------------------------------------------------------*/
find_point_intersection_max(vx, vy, xsize, ysize, block_number_x,
                            block_number_y, max, position_x, position_y, count)
     char    *vx;                    /* 動きベクトルのｘ方向成分 */
     char    *vy;                    /* 動きベクトルのｙ方向成分 */
     int      xsize;                 /* 画像横サイズ */
     int      ysize;                 /* 画像縦サイズ */
     int      block_number_x;        /* ブロックの個数（x方向）*/
     int      block_number_y;        /* ブロックの個数（y方向）*/
     int     *max;                   /* 最大画素値 */
     int      position_x[MAXSIZE];   /* 最大濃度位置（ｘ座標）*/
     int      position_y[MAXSIZE];   /* 最大濃度位置（ｙ座標）*/
     int     *count;                 /* 上述の位置数 */
{
  int             i;
  int             val;              /* 書き込む濃度値 */
  int             size;             /* 画像サイズ */
  unsigned char   *image;           /* 画像 */

  mesprint("In function of find_point_intersection_max.\n");

  val = 1;

  /* 領域確保 */
  size = xsize * ysize;
  image = (unsigned char *) malloc(size * sizeof(unsigned char));

  if (image == NULL) {
    printf("Memory not enough in find_point_intersection_max. \n");
    exit(0);
  }
  mesprint("Clear image.\n");
  /* 画像の初期化 */
  image_clear(image, xsize, ysize);

  /* 動きベクトルのライン表示 */
  mvec_line_fullsize(image, vx, vy, xsize, ysize,
                     block_number_x, block_number_y, val, ZOOM);
  /* 最大濃度値を求める */
  find_intersection_max_value_sub(image, xsize, ysize, max, position_x,
                                  position_y, count);
  free(image);
  mesprint("Out function of find_point_intersection_max.\n");
```



```
}

find_intersection_max_value_sub(image, xsize, ysize, max,
                                position_x, position_y, count)
     unsigned char   *image;                    /* 画像 */
     int              xsize;                    /* 画像横サイズ */
     int              ysize;                    /* 画像縦サイズ */
     int             *max;                      /* 最大濃度値 */
     int              position_x[MAXSIZE];      /* 最大濃度位置（ｘ座標）*/
     int              position_y[MAXSIZE];      /* 最大濃度位置（ｙ座標）*/
     int             *count;                    /* 最大濃度位置数 */
{
  int i;

  mesprint("In function of find_intersection_max_value_sub.\n");
  /* 最大濃度値を求める */
  *max = 0;
  for(i = 0; i < xsize * ysize; i++){
    if((int)(image[i]) > *max){
      *max = (int)(image[i]);
      valprintd("max", *max, "\n");
    }
  }

  /* 最大濃度位置を求める */
  *count = 0;
  /* 最大濃度値が０であれば、終了する */
  if(*max == 0){
    return;
  }
  for(i = 0; i < xsize * ysize; i++){
    if((int)(image[i]) == *max){
      position_x[*count] = i % xsize;
      position_y[*count] = i / xsize;
      (*count)++;
      valprintd("x, y", i % xsize, ", ");
      valprintd("", (int)(i / xsize), "\n");
    }
  }

  mesprint("Out function of find_intersection_max_value_sub.\n");
}

/*------------------------------------------------------------------------------*/
/*          最大濃度位置の１２７背景表示での濃度値を求める関数                  */
/*------------------------------------------------------------------------------*/
find_value_position(vx, vy, xsize, ysize, block_number_x, block_number_y,
                    position_x, position_y, value)
     char     *vx;                     /* 動きベクトル（ｘ成分）*/
     char     *vy;                     /* 動きベクトル（ｙ成分）*/
     int       xsize;                  /* 画像の横サイズ */
     int       ysize;                  /* 画像の縦サイズ */
     int       block_number_x;         /* ブロックの個数（ｘ方向）*/
     int       block_number_y;         /* ブロックの個数（ｙ方向）*/
     int       position_x;             /* 画素の位置（ｘ座標）*/
     int       position_y;             /* 画素の位置（ｙ座標）*/
     int      *value;                  /* 上述位置の濃度値 */
{
  int             size;             /* 画像の画素数 */
  int             val;              /* 動きベクトル描画に加える値 */
  unsigned char  *image;            /* 画像 */

  mesprint("In function of find_value_position.\n");

  val = 1;
```

```
  /* 領域確保 */
  size = xsize * ysize;
  image = (unsigned char *) malloc(size * sizeof(unsigned char));

  if (image == NULL) {
    printf("Memory not enough in find_value_position. \n");
    exit(0);
  }
  mesprint("Clear image.\n");
  /* 画像の127初期化 */
  image_background(image, xsize, ysize, 127);

  /* 動きベクトルのライン表示 */
  mvec_line_fullsize(image, vx, vy, xsize, ysize,
                     block_number_x, block_number_y, val, ZOOM_OUT_IN);
  *value = (int) image[xsize * position_y + position_x];

  free(image);
  mesprint("Out function of find_value_position.\n");
}

/*--------------------------------------------------------------------------*/
/*          動きベクトルをライン表示する関数（ディスプレイ出力用）          */
/*--------------------------------------------------------------------------*/
draw_mvec_line_fullsize(image, vx, vy, xsize, ysize, block_number_x,
                        block_number_y, mode)
     unsigned char  *image[3];          /* 画像 */
     char           *vx;                /* 動きベクトルのｘ方向成分 */
     char           *vy;                /* 動きベクトルのｙ方向成分 */
     int             xsize;             /* 画像横サイズ */
     int             ysize;             /* 画像縦サイズ */
     int             block_number_x;    /* ブロックの個数（x方向）*/
     int             block_number_y;    /* ブロックの個数（y方向）*/
     int             mode;              /* モード（背景０、１２７）*/
{
  int i;

  mesprint("In function of draw_mvec_line_fullsize.\n");
  mesprint("Clear image.\n");

  /* 画像の初期化 */
  if(mode == ZOOM){
    for(i = 0; i < 3; i++){
      image_clear(image[i], xsize, ysize);
    }

    /* 動きベクトルのラインを描く */
    mvec_line_fullsize(image[0], vx, vy, xsize, ysize,
                       block_number_x, block_number_y, 5, ZOOM_OUT_IN);
  }

  /* 画像の127初期化 */
  if(mode == ZOOM_OUT_IN){
    for(i = 0; i < 3; i++){
      image_background(image[i], xsize, ysize, 127);
    }

  /* 動きベクトルのラインを描く */

    mvec_line_fullsize(image[0], vx, vy, xsize, ysize,
                       block_number_x, block_number_y, 5, ZOOM_OUT_IN);
  }
  /* 画像のコピー */
  image_copy(image[0], image[1], xsize, ysize);
  image_copy(image[0], image[2], xsize, ysize);
  mesprint("Out function of draw_mvec_line_fullsize.\n");
```



```
}

/*--------------------------------------------------------------------------*/
/*              動きベクトルをライン表示する関数（演算用）                  */
/*--------------------------------------------------------------------------*/
mvec_line_fullsize(image, vx, vy, xsize, ysize, block_number_x,
                   block_number_y, val, mode)
     unsigned char  *image;             /* 画像  */
     signed char    *vx;                /* 動きベクトルのｘ方向成分 */
     signed char    *vy;                /* 動きベクトルのｙ方向成分 */
     int             xsize;             /* 画像横サイズ */
     int             ysize;             /* 画像縦サイズ */
     int             block_number_x;    /* ブロックの個数（x方向）*/
     int             block_number_y;    /* ブロックの個数（y方向）*/
     int             val;               /* 書き込む値 */
     int             mode;              /* モード（背景０、１２７）*/
{
  int            i, j;
  int            xs, ys;               /* 始点 */
  int            xc, yc;               /* ブロックの中央点 */
  int            xe, ye;               /* 終点 */
  signed char    dummy_vx, dummy_vy;   /* 動きベクトル（ダミー）*/

  mesprint("In function of mvec_line_fullsize.\n");
  valprintd("draw value", val, "\n");

  for (i = 0; i < block_number_y; i++) {
    for (j = 0; j < block_number_x; j++) {
      /* 始点、終点、中央点をブロックの中央に設定しておく */
      xs = j * (xsize / block_number_x) + (xsize / block_number_x) / 2;
      ys = i * (ysize / block_number_y) + (ysize / block_number_y) / 2;
      xc = xs;
      yc = ys;
      xe = xs;
      ye = ys;

      valprintd("Motion Vector vx,vy", vx[i * block_number_x + j], ",");
      valprintd("", vy[i * block_number_x + j], "\n");
      /* 動きベクトルが0の時は、次に進む */
      if((vx[i * block_number_x + j] == 0)
         && (vy[i * block_number_x + j] == 0)){
        continue;
      }

      /* 始点を動きベクトル方向に延ばしていき、画像からはみでたら終了 */
      while(1){
        xs -= vx[i * block_number_x + j];
        ys -= vy[i * block_number_x + j];
        if (xs < 0 || ys < 0 || xs >= xsize || ys >= ysize){
          xs += vx[i * block_number_x + j];
          ys += vy[i * block_number_x + j];
          break;
        }
      }

      /* 終点を動きベクトルとは逆方向に延ばしていき、画像からはみでたら終了 */
      while(1){
        xe += vx[i * block_number_x + j];
        ye += vy[i * block_number_x + j];
        if (xe < 0 || ye < 0 || xe >= xsize || ye >= ysize){
          xe -= vx[i * block_number_x + j];
          ye -= vy[i * block_number_x + j];
          break;
        }
      }
```

```
      valprintd("Start point : x,y", xs, ",");
      valprintd("", ys, "\n");
      valprintd("Center point : x,y", xs, ",");
      valprintd("", ys, "\n");
      valprintd("End point : x,y", xe, ",");
      valprintd("", ye, "\n");

      if(mode == ZOOM){
        line_image_add(image, xs, ys, xe, ye, val, xsize, ysize);
      }
      if(mode == ZOOM_OUT_IN){
        line_image_add(image, xs, ys, xc, yc, val, xsize, ysize);
        line_image_add(image, xc, yc, xe, ye, -val, xsize, ysize);
      }
    }
  }
  mesprint("Out function of mvec_line_fullsize.\n");
}


/*-------------------------------------------------------------------------*/
/*                              線を引く                                   */
/*-------------------------------------------------------------------------*/
line_image_add(image, xs, ys, xe, ye, val, xsize, ysize)

     unsigned char   *image;          /* 画像 */
     int              xs, ys;         /* 始点 */
     int              xe, ye;         /* 終点 */
     int              val;            /* 書き込む値 */
     int              xsize;          /* 画像横サイズ */
     int              ysize;          /* 画像縦サイズ */
{
  int     x, y, x1, x2, y1, y2, i;
  int     dummy;
  float   dx, dy;

  mesprint("In function line_image_add\n");

  if (abs(xs-xe) >= abs(ys-ye)){
    if(xe >= xs){
      x1 = xs;
      x2 = xe;
      y1 = ys;
      y2 = ye;
    }
    else {
      x1 = xe;
      x2 = xs;
      y1 = ye;
      y2 = ys;
    }
    if (x2 == x1){
      dy = 0;
    }
    else{
      dy = (float) (y2 - y1) / (x2 - x1);
    }
    for (x = x1; x <= x2; x++) {
      y = (int) (y1 + dy * (x - x1));
      dummy = image[y * xsize + x];
      dummy += val;
      if(dummy > 255){
        image[y * xsize + x] = 255;
      }
      else if(dummy < 0){
        image[y * xsize + x] = 0;
```



```
      }
      else{
        image[y * xsize + x] = (unsigned char) dummy;
      }
    }
  }
  else {
    if(ye >= ys){
      y1 = ys;
      y2 = ye;
      x1 = xs;
      x2 = xe;
    }
    else{
      y1 = ye;
      y2 = ys;
      x1 = xe;
      x2 = xs;
    }
    if (y2 == y1)
      dx = 0;
    else
      dx = (float) (x2 - x1) / (y2 - y1);
    for (y = y1; y <= y2; y++) {
      x = (int) (x1 + dx * (y - y1));
      dummy = image[y * xsize + x];
      dummy += val;
      if(dummy > 255){
        image[y * xsize + x] = 255;
      }
      else if(dummy < 0){
        image[y * xsize + x] = 0;
      }
      else{
        image[y * xsize + x] = (unsigned char) dummy;
      }
    }
  }
  mesprint("Out function line_image_add\n");
}

/*-------------------------------------------------------------------------*/
/*                              TEST_MAIN                                  */
/*-------------------------------------------------------------------------*/

/* #define TEST_MAIN_DRAW_MVEC_LINE */
/* #define TEST_LINE_IMAGE_ADD  */
/* #define TEST_DRAW_MVEC_LINE_FULLSIZE */

#ifdef TEST_MAIN_DRAW_MVEC_LINE
main(argc, argv)
     int     argc;
     char  **argv;
{
  int xs, ys;                          /* 始点 */
  int xe, ye;                          /* 終点 */
  unsigned char *image_in[3];          /* 入力画像 */
  unsigned char *image_out[3];         /* 出力画像 */
  unsigned char *image_work[3];        /* ワーク画像 */
  int xsize = 200;                     /* 画像横サイズ */
  int ysize = 200;                     /* 画像縦サイズ */
  double vx[MAXSIZE], vy[MAXSIZE];     /* 動きベクトル */
  int i;

  color_image_init(image_in, image_out, image_work, xsize, ysize);
  InitCDisplay(argc, argv, xsize, ysize);
```

```
  for(i = 0; i < 3; i++){
    image_clear(image_out[i], xsize, ysize);
  }
#ifdef TEST_LINE_IMAGE_ADD
  while(1){
    printf("Test : line_image_add\n");
    printf("--------------------------\n");
    printf("Input  start x,y -> xs ys\n");
    scanf("%d %d", &xs, &ys);
    printf("Input  end   x,y -> xe ye\n");
    scanf("%d %d", &xe, &ye);
    line_image_add(image_out[0], xs, ys, xe, ye, 100, xsize, ysize);
    image_copy(image_out[0], image_out[1], xsize, ysize);
    image_copy(image_out[0], image_out[2], xsize, ysize);
    DisplayCImage(image_out[0], image_out[1], image_out[2], OUT, xsize, ysize);
  }
#endif
}
#endif
```

```
#define    STANDSTILL          0x1       /* カメラワークを示す定数 */
#define    PAN_UP              0x2
#define    PAN_DOWN            0x4
#define    PAN_RIGHT           0x8
#define    PAN_LEFT            0x10
#define    PAN_RIGHT_UP        0x20
#define    PAN_RIGHT_DOWN      0x40
#define    PAN_LEFT_UP         0x80
#define    PAN_LEFT_DOWN       0x100
#define    ZOOM_IN             0x200
#define    ZOOM_OUT            0x400
#define    NO_RECOGNIZE        0x0

#define    STANDSTILL_LIMIT       1.1   /* カメラワークの閾値 */
#define    PAN_LIMIT             18.0
#define    ZOOM_LIMIT             102

#define    OK   1
#define    NO   0

#define    LEFT   20           /* ズーム時のパン、チルトの方向 */
#define    RIGHT  21
#define    UP     22
#define    DOWN   23

#define    ZOOM    1          /* ライン表示のモード */
#define    ZOOM_OUT_IN  2
```

```
#include <stdio.h>
#include <math.h>
#include "params.h"
#include "recognize.h"

/*------------------------------------------------------------------------------*/
/*                    メッセージ出力用マクロ定義                                */
/*------------------------------------------------------------------------------*/
/* #define PRINT_MEASSEGE_RECOGNIZE_CAMERA_WORK */
#ifdef  PRINT_MEASSEGE_RECOGNIZE_CAMERA_WORK
#define  mesprint(x)   printf("%s",x)
#define  valprint(x,y,z) printf("%s : %lf%s", x,y,z)
#else
#define  mesprint(x)      /* */
#define  valprint(x,y,z)  /* */
#endif

/*------------------------------------------------------------------------------*/
/*                  動きベクトルからカメラワークを識別する                      */
/*------------------------------------------------------------------------------*/
recognize_camera_work(v_average, angle_average,
                      angle_standard_deviation, max, position_x,
                      position_y, value, count, xsize, ysize, camera_work)
     double  v_average;                  /* 動きベクトルの平均 */
     double  angle_average;              /* 角度平均 */
     double  angle_standard_deviation;   /* 角度の標準偏差 */
     int     max;                        /* 濃度の最大値 */
     int     position_x[MAXSIZE];        /* 最大濃度値の位置（x成分） */
     int     position_y[MAXSIZE];        /* 最大濃度値の位置（y成分） */
     int     value[MAXSIZE];    /* 最大濃度値の位置の１２７背景による濃度値 */
     int     *camera_work;               /* カメラワーク */
{
  /* 認識不能 */
  *camera_work = NO_RECOGNIZE;

  /* カメラ固定かどうかのチェック */
  /* recognize_motion_frame_standstill(v_average, camera_work); */

  /* パンかどうかのチェック */
  recognize_motion_frame_pan(angle_average,
                             angle_standard_deviation, camera_work);

  /* ズームかどうかのチェック*/
  recognize_motion_frame_zoom(max, position_x, position_y, value, count,
                              xsize, ysize, camera_work);
}

/*------------------------------------------------------------------------------*/
/*                  カメラ固定の状態かどうかを調べる                            */
/*------------------------------------------------------------------------------*/
recognize_motion_frame_standstill(v_average, camera_work)
     double     v_average;              /* 動きベクトルの平均 */
     int        *camera_work;           /* カメラワーク */
{
  /* ２乗和平均により、カメラ固定かどうかを判定する */
  if(v_average < STANDSTILL_LIMIT){
    *camera_work += STANDSTILL;
  }
}

/*------------------------------------------------------------------------------*/
/*                  パン,チルトかどうかの判定をする                             */
/*------------------------------------------------------------------------------*/
recognize_motion_frame_pan (angle_average,
                            angle_standard_deviation, camera_work)
     double   angle_average;             /* 角度平均 */
```



```
     double   angle_standard_deviation;  /* 角度の標準偏差 */
     int      *camera_work;              /* カメラワーク */
{
  /* 角度の標準偏差から、パンかどうかを判定し、方向も求める */
  if(angle_standard_deviation <= PAN_LIMIT){
    /* パンが左であるかどうかの判定 */
    if((angle_average > -22.5) && (angle_average <= 22.5)){
      *camera_work += PAN_LEFT;
    }
    /* パンが左上であるかどうかの判定 */
    else if((angle_average > 22.5) && (angle_average <= 67.5)){
      *camera_work += PAN_LEFT_UP;
    }
    /* チルトが上であるかどうかを判定 */
    else if((angle_average > 67.5) && (angle_average <= 112.5)){
      *camera_work += PAN_UP;
    }
    /* パンが右上であるかどうかを判定 */
    else if((angle_average > 112.5) && (angle_average <= 157.5)){
      *camera_work += PAN_RIGHT_UP;
    }
    /* パンが右であるかどうかを判定 */
    else if(((angle_average > 157.5) && (angle_average <= 180)) ||
            ((angle_average <= 157.5) && (angle_average > -180))) {
      *camera_work += PAN_RIGHT;
    }
    /* パンが左下であるかどうかを判定 */
    else if((angle_average <= -22.5) && (angle_average > -67.5)){
      *camera_work += PAN_LEFT_DOWN;
    }
    /* チルトがしたであるかどうかを判定 */
    else if((angle_average <= -67.5) && (angle_average > -112.5)){
      *camera_work += PAN_DOWN;
    }
    /* パンが右下であるかどうかの判定 */
   else if((angle_average <= -112.5) && (angle_average > -157.5)){
      *camera_work += PAN_RIGHT_DOWN;
    }
  }
}

/*------------------------------------------------------------------------------*/
/*                  ズームかどうかを判定する                                    */
/*------------------------------------------------------------------------------*/
recognize_motion_frame_zoom(max, position_x, position_y, value, count, xsize,
                            ysize, camera_work)
     int      max;                       /* 濃度の最大値 */
     int      position_x[MAXSIZE];       /* 最大値の位置（x成分）*/
     int      position_y[MAXSIZE];       /* 最大値の位置（y成分）*/
     int      value[MAXSIZE];            /* １２７背景表示での最大値 */
     int      count;                     /* 最大濃度位置の数 */
     int      *camera_work;              /* カメラワーク */
{
  int   position_x_sum;             /* 最大濃度位置の和（x座標）*/
  int   position_y_sum;             /* 最大濃度位置の和（y座標）*/
  int   position_x_average;         /* 最大濃度位置の平均（x座標）*/
  int   position_y_average;         /* 最大濃度位置の平均（y座標）*/
  int   i;

  /* 濃度の最大値からズームかどうかの判定 */
  if(max > ZOOM_LIMIT){
    /* 位置平均を求める */
    position_x_sum = 0;
    position_y_sum = 0;
    for(i = 0; i < count; i++){
      position_x_sum += position_x[i];
```

```
    position_y_sum += position_y[i];
  }
  position_x_average = position_x_sum / count;
  position_y_average = position_y_sum / count;

  /* ズームインかどうかの判定 */
  if(value[0] > 127){
    *camera_work += ZOOM_IN;
    /* 最大濃度の位置からパンであるかどうか判定 */
    if(position_x_average < xsize / 3){
      if(position_y_average < ysize / 3){
        *camera_work += PAN_LEFT_UP;
      }
      else if(position_y_average < 2 * ysize / 3){
        *camera_work += PAN_LEFT;
      }
      else{
        *camera_work += PAN_LEFT_DOWN;
      }
    }
    else if(position_x_average < 2 * xsize / 3){
      if(position_y_average < ysize / 3){
        *camera_work += PAN_UP;
      }
      if((position_y_average >= 2 * ysize / 3)
         && (position_y_average < ysize)){
        *camera_work += PAN_DOWN;
      }
    }
    else{
      if(position_y_average < ysize / 3){
        *camera_work += PAN_RIGHT_UP;
      }
      else if(position_y_average < 2 * ysize / 3){
        *camera_work += PAN_RIGHT;
      }
      else{
        *camera_work += PAN_RIGHT_DOWN;
      }
    }
  }
  /* ズームアウトの場合 */
  else{
    *camera_work += ZOOM_OUT;
    if(position_x_average < xsize / 3){
      if(position_y_average < ysize / 3){
        *camera_work += PAN_RIGHT_DOWN;
      }
      else if(position_y_average < 2 * ysize / 3){
        *camera_work += PAN_RIGHT;
      }
      else{
        *camera_work += PAN_RIGHT_UP;
      }
    }
    else if(position_x_average < 2 * xsize / 3){
      if(position_y_average < ysize / 3){
        *camera_work += PAN_DOWN;
      }
      if((position_y_average >= 2 * ysize / 3)
         && (position_y_average < ysize)){
        *camera_work += PAN_UP;
      }
    }
    else{
      if(position_y_average < ysize / 3){
```



```
        *camera_work += PAN_LEFT_DOWN;
      }
      else if(position_y_average < 2 * ysize / 3){
        *camera_work += PAN_LEFT;
      }
      else{
        *camera_work += PAN_LEFT_UP;
      }
    }
  }
}
```