実験項目	実験 C4 Processing3 による画像処理 およびヒューマンインタフェースの実現
校名　科名 学年　番号	熊本高等専門学校　　人間情報システム工学科 3 年　　42 号
氏名	山口惺司
班名　回数	4 班　　1 回目
実験年月日 建物　部屋名	2023 年　12 月　14 日　木曜 2023 年　12 月　21 日　木曜 2023 年　1 月 11 日　木曜 3 号棟　　2 階　HIPC 室
共同実験者名	

1. 概要

Processing は手軽に CG や画像処理、情報可視化、インタラクション・システムなどを実装できるプログラミング言語である。Java 言語を基にしており Windows、MacOS、Linux、Android などの各種 OS 上で動作できる。またライブラリを利用することでカメラや音声、Kinect センサなどのデバイスや ネットワークからのデータ入出力なども行える。マニュアルは http://processing.org/reference/で参照 できる。 ヒューマンインタフェースとは、人間がコンピュータを操作するときに使う機器のことであり、代表 的なものとしてマウスやキーボード、ディスプレイなどが挙げられる。近年は wii リモコンや Kinect セ ンサ、スマートホンなど、センサや画像処理で新しいヒューマンインタフェースを提供する例が増えて いる。これらはマウスやキーボードに比べてより直感的な操作方法をユーザに提供し、より強いユーザ エクスペリエンス(UX)を持ったシステムを実現している。 そこで本実験ではまず Processing3 を用いて画像処理プログラミングの基礎を学ぶ。そして USB カ メラの映像を解析して指で画像を動かす簡易システムを実装する。さらにその仕組みを応用したシステ ムを開発する。

2. 課題

1. 課題 1（画像表示）

   内容：

   本課題では Processing で画像ファイルを表示するプログラムを作る。

   画像を 2 つ用意し、マウスの位置に絵を表示させる。また、”1”キーもしくは”2”キーで表示する画像が切り替わるようにする。

   ソースコード：

```
// 大域変数
PImage mountain, sea
  ; // 画像ファイルを扱う変数を作る
int mode = 1;

// 初期設定
void setup () {
  size (640, 480, P3D);
  mountain = loadImage("mountain.png"); //「test.png」という画像ファイルを読み込む
  sea = loadImage("sea.png");
  imageMode(CENTER);
}
// メインルーチン
void draw() {
```

```
  background (160, 255, 255); // 背景を水色にする
  if (mode == 1) {
    image(mountain, mouseX, mouseY, 300, 300); // 画像を x=100, y=50 の位置に描く
  } else if (mode == 2) {
    image(sea, mouseX, mouseY, 300, 300);
  }
  fill(0);
}

void keyPressed() {
  if (key == '1') {
    mode = 1;
  } else if (key == '2') {
    mode = 2;
  }
}
```

実行例：

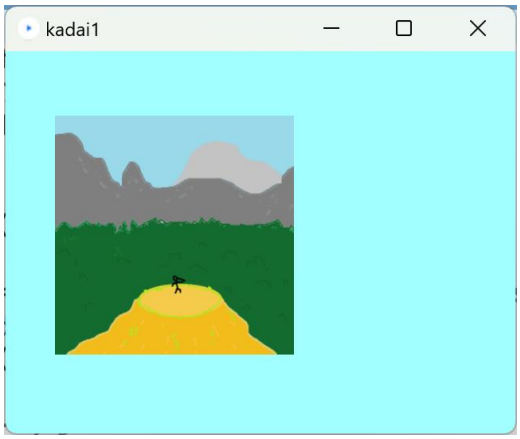

図 1-1 実行例 1

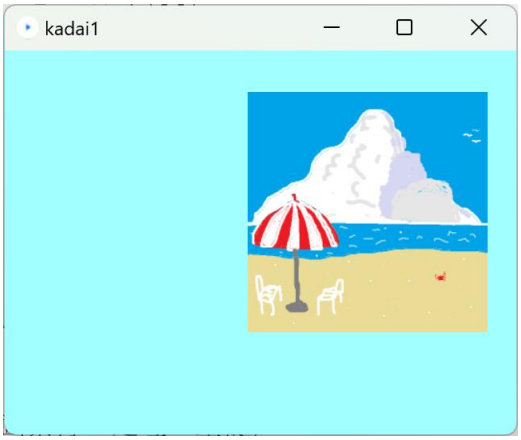

図 1-2 実行例 2

2. 課題２（画像処理１：画素毎の処理の基礎）

内容：

本課題では画像データを書き換える方法を学ぶ。 まず Processing では画像の変数（説明上 img とおく）から以下のデータを取り扱える。

・img.width ・・・画像 img の横幅。

・img.height ・・・画像 img の縦幅。

・img.pixels[i] ・・・画像 img 内の i 番目の色データ（一次元配列）。座標(x,y)の色データは img.pixels[y * img.width + x]で表される。

表 2-1： 画像 img と配列 img.pixels[]の対応関係（例：横幅 4、縦幅 3 の画像）

pixels[0]	pixels[1]	pixels[2]	pixels[3]
pixels[4]	pixels[5]	pixels[6]	pixels[7]
pixels[8]	pixels[9]	pixels[10]	pixels[11]

∴(x, y)の色データは img.pixels[y * img.width + x]である。

これを利用し、上下反転及び左右反転させた画像を図 2 のように表示する。

図 2-1 結果例

ソースコード：

```
PImage inImg; // 入力画像
PImage outImg0, outImg1, outImg2; // 出力画像

// 初期設定
void setup() {
  size (600, 600, P3D);
  // 元画像を読み込む
  inImg = loadImage("test.png");
  textFont(createFont("Century", 30));
  textAlign(LEFT, TOP);
}
// メインルーチン
void draw() {
  int x = 0, y = 0;
  int p1, p2;
  background(0);
  // 左右反転した画像 outImg を作る
  outImg0 = createImage(inImg.width, inImg.height, RGB);
  outImg1 = createImage(inImg.width, inImg.height, RGB);
  outImg2 = createImage(inImg.width, inImg.height, RGB);
  for (y = 0; y < inImg.height; y++) {
```

```
    for (x = 0; x < inImg.width; x++) {
      p1 = y * inImg.width + x;
      p2 = y * inImg.width + (inImg.width - 1 - x);
      outImg0.pixels[p1] = inImg.pixels[p2]; //入力画像の画素 p2 を出力画像の画素 p1 に配置
    }
  }

  for (y = 0; y < inImg.height; y++) {
    for (x = 0; x < inImg.width; x++) {
      p1 = y * inImg.width + x;
      p2 = (inImg.height - y - 1) * inImg.width + x;
      outImg1.pixels[p1] = inImg.pixels[p2]; //入力画像の画素 p2 を出力画像の画素 p1 に配置
    }
  }

  for (y = 0; y < inImg.height; y++) {
    for (x = 0; x < inImg.width; x++) {
      p1 = y * outImg1.width + x;
      p2 = y * outImg1.width + (outImg1.width - 1 - x);
      outImg2.pixels[p1] = outImg1.pixels[p2]; //入力画像の画素 p2 を出力画像の画素 p1 に配置
    }
  }

  // 画像を表示する
  image(inImg, 0, 0, 300, 300);
  image(outImg0, 300, 0, 300, 300);
  image(outImg1, 0, 300, 300, 300);
  image(outImg2, 300, 300, 300, 300);
}
```

実行例：

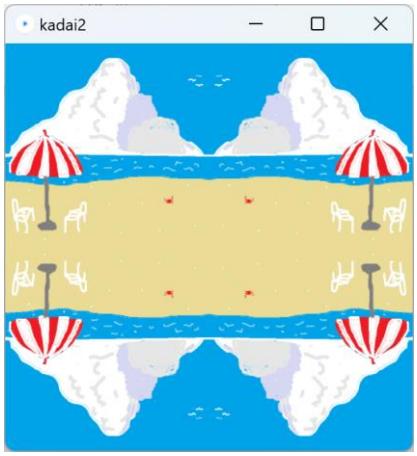

図 2-2 出力画像

3. 課題３（画像処理２：画素毎の色の変更）

内容：

各画素の色は 3 原色(RGB)の混合で表されている。ある画素 pixels[i]の各原色の数値(0～255)を取り 出すには以下のようにする。

・float r = red(img.pixels[i]); ・・・赤色の数値を取り出す

・float g = green(img.pixels[i]); ・・・緑色の数値を取り出す

・float b = blue(img.pixels[i]); ・・・青色の数値を取り出す

また、画素そのものは color 型の変数として定義されている。ある画素の色を変えるには以下の要領 で行う。

```
color c = color(0, 255, 0); // 緑色のデータを作る ※color(赤の濃さ, 緑の濃さ, 青の濃さ);
img.pixels[i] = c;
```

または

```
img.pixels[i] = color(0, 255, 0);
```

以上のことを基に、マウスの x 座標値に応じて濃さが変わる（マウスを右にずらすほど緑色が減る）プログラムを作れ。(実行例を図 3-1、3-2 に示す。)

図 3-1 入力画像例

図 3-2 出力画像例

ソースコード：

```
PImage inImg, outImg;

void setup() {
  size(1000, 600);
  inImg = loadImage("fukei.png");
  imageMode(CENTER);
}

void draw() {
  background(255);
  outImg = createImage(inImg.width, inImg.height, RGB);
  float r = 0, g = 0, b = 0;
  int x=0, y=0;
  for (y = 0; y < inImg.height; y++) {
    for (x = 0; x < inImg.width; x++) {
      r = red(inImg.pixels[y * inImg.width + x]);
      g = green(inImg.pixels[y * inImg.width + x]);
      b = blue(inImg.pixels[y * inImg.width + x]);
      color c = color(r, g-mouseX*255/width, b);
      outImg.pixels[y * inImg.width + x] = c;
    }
  }



  image(outImg, width/2, height/2, 600, 400);
}
```

実行例：

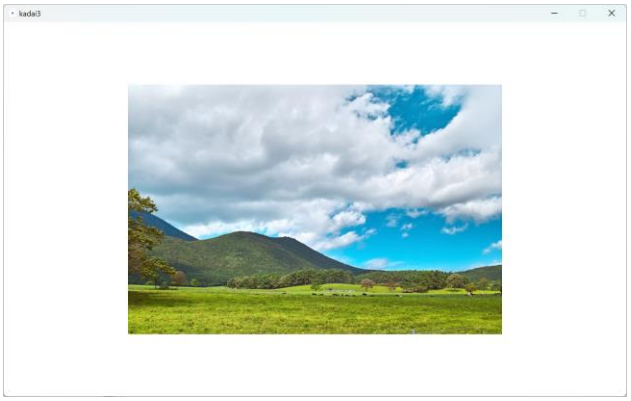

図 3-3 入力画像

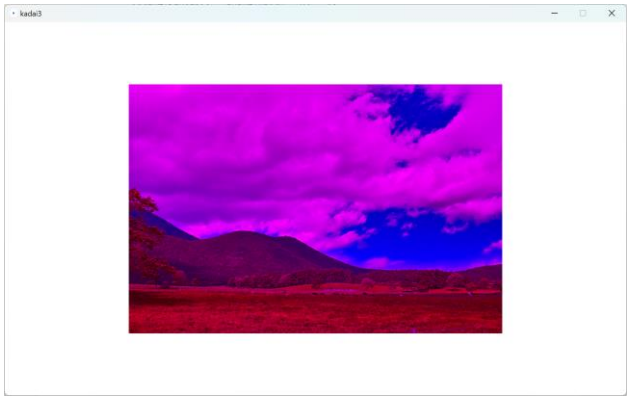

図 3-4 出力画像

4. 課題４（画像処理３：二値化）

内容：

現在ほとんどの画像はカラー画像であり、色の値は三原色(R、G、B)の各値を指定して作られる。一方、 画像を利用した計算機処理の場合、処理速度・省メモリ・用途等の諸観点から、最初にカラー画像を白 黒濃淡または二値のモノクロ画像に変換する場合がある。例えば文字認識プログラムでは、文字領域を 黒、それ以外の領域を白色にすることで認識対象物が明確になる。このようにカラー画像を二値画像に 変換する処理を二値化という。

二値化は以下の規則で行う。

【考え方】

座標(x, y)の色の明るさ L(x, y)が閾値 S 以上ならば白色、そうでなければ黒色にする

【明るさ L(x, y)の求め方】

単純な方法の一つとして「輝度」を使う方法がある。RGB の値から輝度 L を求める式を次式に示す。

$$L(x, y) = 0.2126R(x, y) + 0.7152G(x, y) + 0.0722B(x, y)$$

上式は R、G、B の重み付き平均を取る式になっている。通常の平均であれば１３ $R(x, y)$ + １３ $G(x, y)$ + １３ $B(x, y)$ で求められるが、上式の場合、各項にかかる定数は、人が明るい色と感じやすい色ほど値が大きくなっ ている（そして平均の機能を失わないよう、定数を合計すると 1 になるよう値が選ばれている)。例え ば緑色と青色とでは緑色の方がより明るく感じられると言われていることから、緑色の係数が大きい。 なお上式の係数は HDTV の場合のものであり、形式によって値は多少異なる。

【閾値 S の求め方】

単純には、対象画像の輝度の平均値を S とする方法が考えられる。この方法は白色の領域と黒色の領 域を概ね半数程度ずつに分けるときには有効であろう。しかし画像によっては、例えば大半が白色でごく一部を黒色にした方が良いなど、偏りのある構図の場合がある。このことを考えると、閾値 S は人が 調整できることが望ましい。 以上のことを基に、輝度に基づいて画像を二値化するプログラムを作れ。その際の閾値は、マウスの x 座標値を用いてユーザが変更できるようにすること。(実行例を図 4 に示す。)

図 4 二値化の例

ソースコード：

```
PImage inImg, outImg;

void setup() {
  size(1200, 600);
  inImg = loadImage("fukei.png");
  imageMode(CENTER);
}

void draw() {
  background(255);
  outImg = createImage(inImg.width, inImg.height, RGB);
  float r = 0, g = 0, b = 0;
  float l = 0;
  int x=0, y=0;
  for (y = 0; y < inImg.height; y++) {
    for (x = 0; x < inImg.width; x++) {
      r = red(inImg.pixels[y * inImg.width + x]);
      g = green(inImg.pixels[y * inImg.width + x]);
      b = blue(inImg.pixels[y * inImg.width + x]);
      l = 0.2126 * r + 0.7152 * g + 0.0722 * b;
      if(l > mouseX*255/width){
        outImg.pixels[y * inImg.width + x] = color(255);
      } else {
        outImg.pixels[y * inImg.width + x] = color(0);
      }
    }
  }
  image(inImg, width/2-300, height/2, 600, 400);
  image(outImg, width/2+300, height/2, 600, 400);
}
```

実行例：

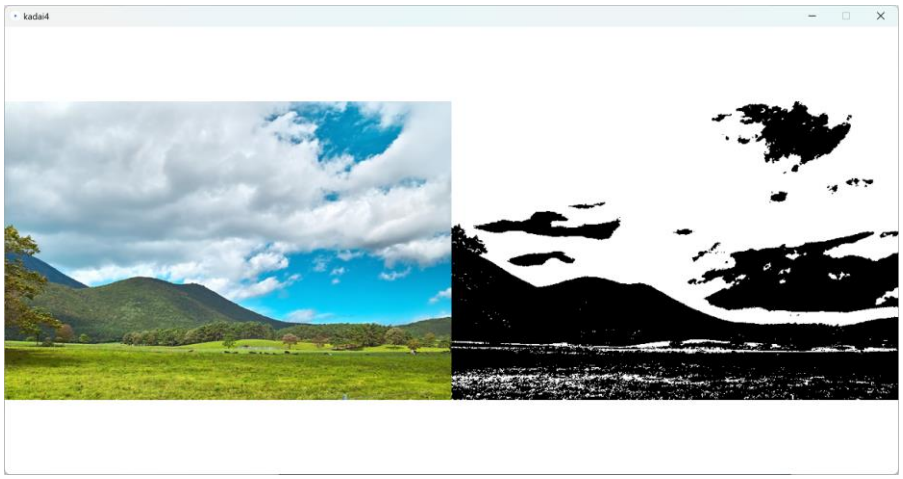

図 4-2 二値化前の画像(左)と二値化後の画像(右)

5. 課題５（画像処理４：ぼかし / 移動平均によるノイズ除去処理）

内容：

画像に混入したノイズを除去したいという要求は少なからずある。ここでは移動平均を使い、ごま塩 を振りかけたようなノイズを低減する実験を行う。(※ なお、移動平均はあくまでもノイズの低減であり、除去ではない。精度はメディアんフィルタの方が良い場合が多い。)

【基本的な考え方】

ごま塩状のノイズはその周囲にある色と著しく異なっている。また、元の色はその周囲の色に近い色 であった可能性がある。例えば青空の写真に黒いごま塩状のノイズがあるとき、ノイズの入った箇所の 元の色は周囲の青色と同色である可能性が高い。 そこで色の平均を使ってノイズを低減させることを考える。今ノイズが座標(x, y)にあるとき、(x, y)を中心とする 3 画素×3 画素の領域の色の平均値を求め、その色を(x, y)に上書きする。青空の写真 であれば、黒いノイズが 1 つ混じっていたとしても平均値は青色なので、黒色成分をある程度減らし、 元の青色に近づけることができる。これを定式化すると以下のようになる。

$$R'(x,y) = \frac{1}{9}\sum_{a=-1}^{1}\sum_{b=-1}^{1} R(x+a, y+b)$$

$$G'(x,y) = \frac{1}{9}\sum_{a=-1}^{1}\sum_{b=-1}^{1} G(x+a, y+b)$$

$$B'(x,y) = \frac{1}{9}\sum_{a=-1}^{1}\sum_{b=-1}^{1} B(x+a, y+b)$$

ここで R(x, y)、G(x, y)、B(x, y)はそれぞれ座標(x, y)の赤色、緑色、青色の値である。このように、 周囲の値を参照して一点の値を更新するような計算のことを「畳み込み」という。また、上式によるノ イズ除去のことを「平均平滑化」「移動平均」と呼ぶ。 上式の計算はすべての画素に対して行う。すなわちノイズの無い画素に対しても上式を適用する。こ れは、どれがノイズでどれがノイズでないかが分からないためである。ところがすべての画素で移動平 均をとる

と、画像全体がぼけるという現象が起きる。画像をぼかさずノイズ除去する実験は今回は行わ ないが、基本的な方法として以下の方法がある。 ・ 平均値ではなく中央値を使う。興味のある学生は「メディアンフィルタ」で検索するとよい。 ・ 移動平均による処理を行った後で、鮮鋭化（アンシャープ）処理を行う。

以上のことを用いて移動平均によるノイズ除去プログラムを作れ。

ソースコード：

```
PImage inImg, outImg;

void setup() {
  size(1200, 600, P3D);
  inImg = loadImage("noise+5000.png");
  imageMode(CENTER);
}

void draw() {
  background(255);
  outImg = createImage(inImg.width, inImg.height, RGB);
  int x=0, y=0;
  for (y = 0; y < inImg.height; y++) {
    for (x = 0; x < inImg.width; x++) {
      float newR = 0, newG = 0, newB = 0;
      float N = 0;
      for (int b = -1; b <= 1; b++) {
        for (int a = -1; a <= 1; a++) {
          if (x+a > 0 && x+a < inImg.width && y+b > 0 && y+b < inImg.height) {
            newR += red(inImg.pixels[(y+b) * inImg.width + (x+a)]);
            newG += green(inImg.pixels[(y+b) * inImg.width + (x+a)]);
            newB += blue(inImg.pixels[(y+b) * inImg.width + (x+a)]);
            N++;
          }
        }
      }

      color c = color(newR/N, newG/N, newB/N);
      outImg.pixels[y * inImg.width + x] = c;
    }
  }
  image(inImg, width/2-300, height/2, 600, 400);
  image(outImg, width/2+300, height/2, 600, 400);
```

```
}
```

実行例：

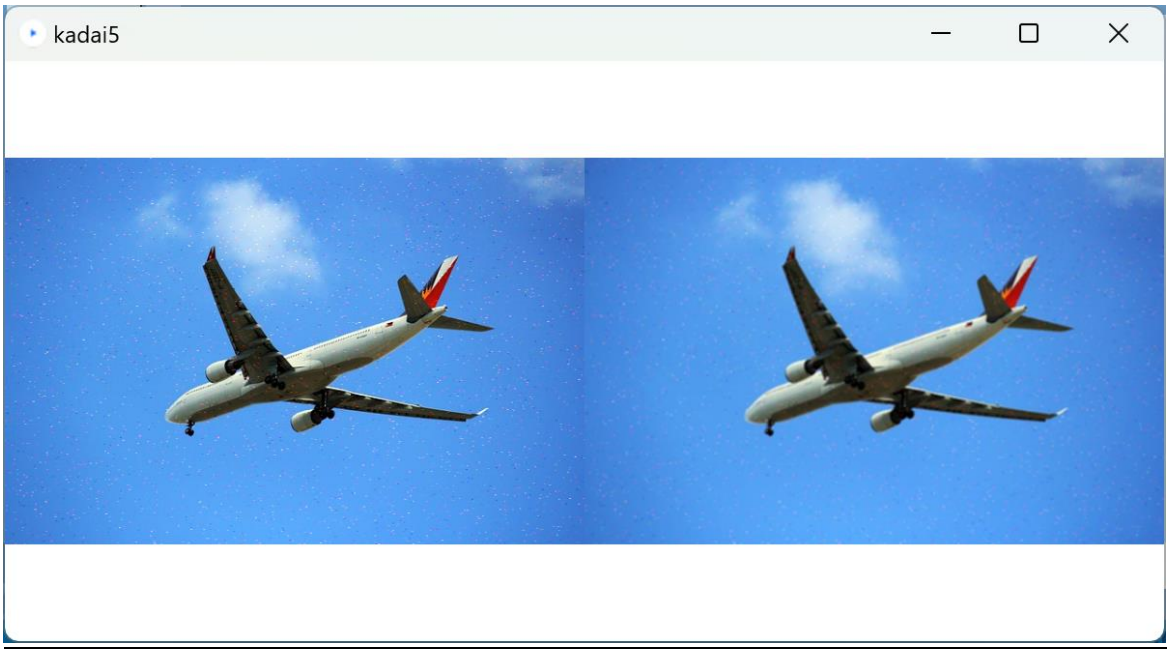

図 5 ノイズ除去前の画像(左)とノイズ除去後の画像(右)

6. 課題６（カメラ映像の描画）

内容：

カメラを用いて、カメラの元映像、水平反転させた映像、垂直反転させた映像、水平＆垂直反転させた映像を出力せよ。

ソースコード：

```
import processing.video.*; // カメラを表す変数
Capture cam;
PImage cam1, cam2, cam3;

void setup() {
  size(640, 480);
  String[] settings = Capture.list(); // 設定リストを取得
  if (settings.length == 0) {
    println("There are no cameras available for capture.");
    exit();
  }
  cam = new Capture(this, settings[0]); // 0 番目の設定を有効化
  cam.start(); // 撮影開始
  printArray(settings);
}
```

```
void draw() {
  if (cam.available() == true) cam.read(); // 現在のカメラ画像を取得

  int x = 0, y = 0;
  int p1, p2;

  cam1 = createImage(cam.width, cam.height, RGB);
  cam2 = createImage(cam.width, cam.height, RGB);
  cam3 = createImage(cam.width, cam.height, RGB);

  for (y = 0; y < cam.height; y++) {
    for (x = 0; x < cam.width; x++) {
      p1 = y * cam.width + x;
      p2 = y * cam.width + (cam.width - 1 - x);
      cam1.pixels[p1] = cam.pixels[p2];
    }
  }

  for (y = 0; y < cam.height; y++) {
    for (x = 0; x < cam.width; x++) {
      p1 = y * cam.width + x;
      p2 = (cam.height - y - 1) * cam.width + x;
      cam2.pixels[p1] = cam.pixels[p2];
    }
  }

  for (y = 0; y < cam2.height; y++) {
    for (x = 0; x < cam2.width; x++) {
      p1 = y * cam2.width + x;
      p2 = y * cam2.width + (cam2.width - 1 - x);
      cam3.pixels[p1] = cam2.pixels[p2];
    }
  }

  // 画像を表示する
  image(cam,0, 0, width/2, height/2);
  image(cam1, width/2, 0, width/2, height/2);
  image(cam2, 0, height/2, width/2, height/2);
  image(cam3, width/2, height/2, width/2, height/2);
}
```

実行例：

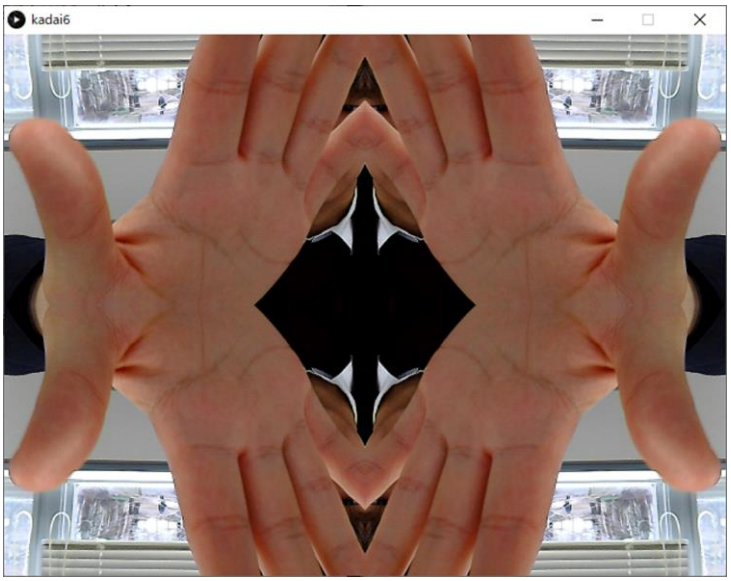

図 6 出力映像

7. 課題 7 （赤色領域の抽出）

内容：

カメラ映像の中にある赤色（とそれに近い色の）領域に外接する四角形を描くプログラムを作れ。

図 7-1 入力画像例

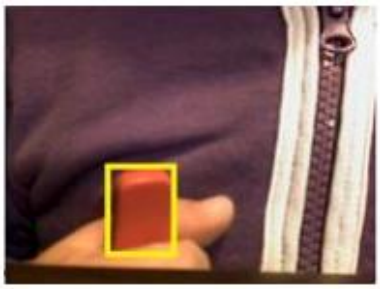

図 7-2 出力画像例

ソースコード：

```
import processing.video.*; // カメラを表す変数
Capture cam;
float Rt = 255, Gt = 0, Bt = 0;

void setup() {
  size(640, 480);
  String[] settings = Capture.list(); // 設定リストを取得
  if (settings.length == 0) {
    println("There are no cameras available for capture.");
    exit();
  }
  cam = new Capture(this, settings[0]); // 0 番目の設定を有効化
  cam.start(); // 撮影開始
```

```
  printArray(settings);
}

void draw() {
  if (cam.available() == true) cam.read(); // 現在のカメラ画像を取得
  image(cam, 0, 0);
  noFill();
  color t;
  t = color(Rt, Gt, Bt);
  int x, y, p1;
  float xmin = width, xmax = 0, ymin =height, ymax = 0;
  for (y = 0; y<cam.height; y++) {
    for (x = 0; x<cam.width; x++) {
      p1 = y * cam.width + x;
      float r = red(cam.pixels[p1]);
      float g = green(cam.pixels[p1]);
      float b = blue(cam.pixels[p1]);
      float diff = sqrt(pow(Rt-r, 2) + pow(Gt-g, 2) + pow(Bt-b, 2));
      if (diff <= 25) {
        if (x < xmin) {
          xmin = x;
        }
        if (x > xmax) {
          xmax = x;
        }
        if (y < ymin) {
          ymin = y;
        }
        if (y > ymax) {
          ymax = y;
        }
      }
    }
  }
  stroke(255, 255, 0);
  strokeWeight(5);
  rect(xmin, ymin, xmax-xmin, ymax-ymin);
}

void mousePressed() {
  loadPixels();
  int i = mouseY * width + mouseX;
```

```
  Rt = red(pixels[i]);
  Gt = green(pixels[i]);
  Bt = blue(pixels[i]);
}
```

実行例：

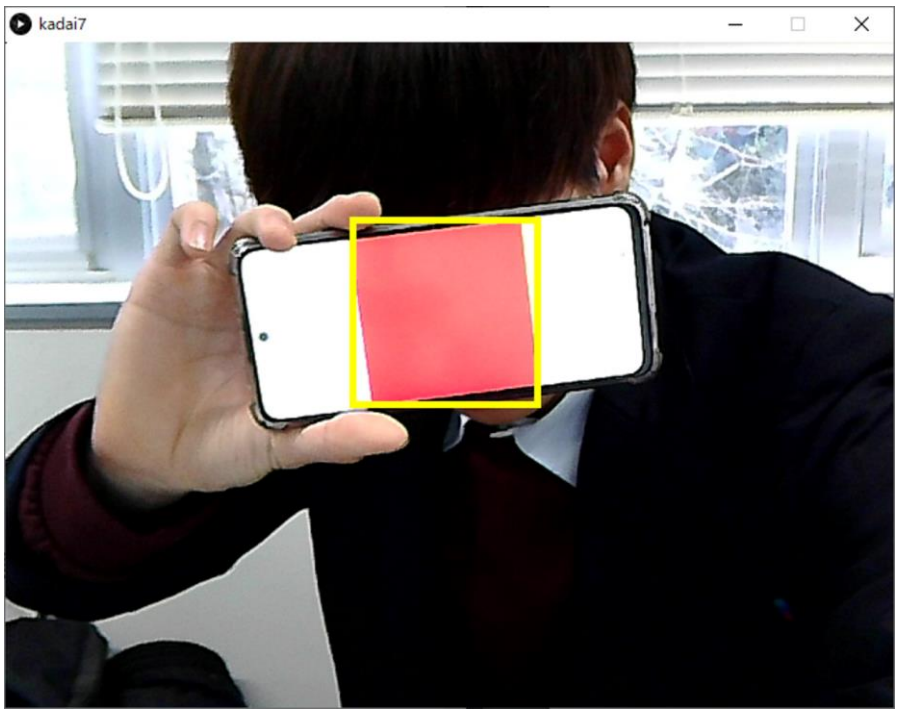

図 7-3 実行例

8. 課題８（赤色マーカーを用いたユーザインタフェース）

内容：

前問を応用すれば、赤色など特定の色マーカーを使ってあたかもマウスのような動きを実現できる。 このような技術はより直感的な操作をユーザに提供できるため、ヒューマンインタフェースやインタラ クション、拡張現実など幾つかの分野で研究開発されている。

そこで本課題では色マーカーでマウスを代替するプログラムを簡易的に実装する。

（課題）

前問の外接四角形の中心座標を求め、その位置に画像を表示するプログラムを作れ。

図 8-1 入力画像例

図 8-2 出力画像例

ソースコード：

```
import processing.video.*; // カメラを表す変数
Capture cam;
float Rt = 255, Gt = 0, Bt = 0;
PImage ocarina;
float size = 0;

void setup() {
  size(640, 480);
  String[] settings = Capture.list(); // 設定リストを取得
  if (settings.length == 0) {
    println("There are no cameras available for capture.");
    exit();
  }
  cam = new Capture(this, settings[0]); // 0 番目の設定を有効化
  cam.start(); // 撮影開始
  printArray(settings);

  ocarina = loadImage("ocarina.png");
  imageMode(CENTER);
}

void draw() {
  if (cam.available() == true) cam.read(); // 現在のカメラ画像を取得
  image(cam, width/2, height/2);
  noFill();
  int x, y, p1;
  float xmin = width, xmax = 0, ymin =height, ymax = 0;
  for (y = 0; y<cam.height; y++) {
    for (x = 0; x<cam.width; x++) {
      p1 = y * cam.width + x;
      float r = red(cam.pixels[p1]);
      float g = green(cam.pixels[p1]);
```

```
      float b = blue(cam.pixels[p1]);
      float diff = sqrt(pow(Rt-r, 2) + pow(Gt-g, 2) + pow(Bt-b, 2));
      if (diff <= 25) {
        if (x < xmin) {
          xmin = x;
        }
        if (x > xmax) {
          xmax = x;
        }
        if (y < ymin) {
          ymin = y;
        }
        if (y > ymax) {
          ymax = y;
        }
        size = xmax-xmin;
      }
    }
  }
  stroke(255, 255, 0);
  strokeWeight(5);
  float cx = (xmax+xmin)/2;
  float cy = (ymax+ymin)/2;
  image(ocarina, cx, cy, size, size);
  size = 0;
}

void mousePressed() {
  loadPixels();
  int i = mouseY * width + mouseX;
  Rt = red(pixels[i]);
  Gt = green(pixels[i]);
  Bt = blue(pixels[i]);
}
```

実行例：

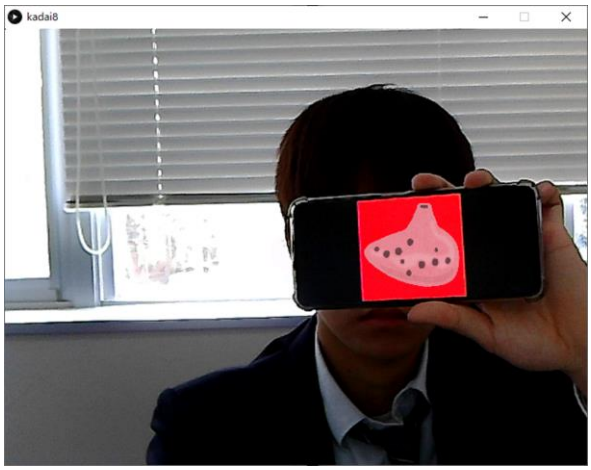

図 8-3 実行例 1

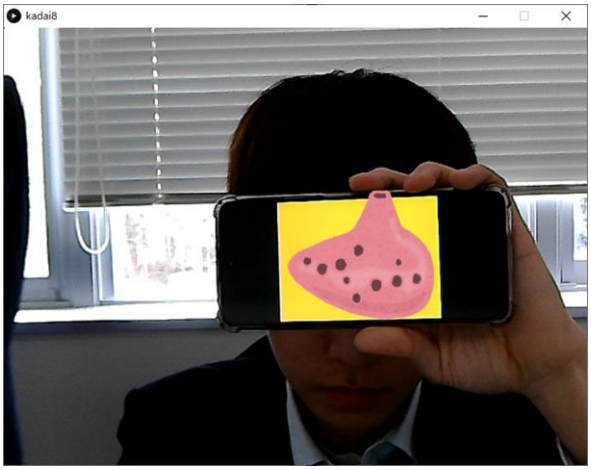

図 8-4 実行例 2

9. 課題 9 （赤色マーカーを用いたユーザインタフェースの発展）

内容：

赤色マーカーの距離によって、表示される画像が変化するプログラムを作成した。

距離が遠い順に、たまご→ひよこ→ニワトリ→チキン、となっている。

ソースコード：

```
import processing.video.*; // カメラを表す変数
Capture cam;
float Rt = 255, Gt = 0, Bt = 0;
PImage egg, hiyoko, niwatori, chicken;
float size = 0;

void setup() {
  size(640, 480);
  String[] settings = Capture.list(); // 設定リストを取得
  if (settings.length == 0) {
    println("There are no cameras available for capture.");
    exit();
  }
  cam = new Capture(this, settings[1]); // 0 番目の設定を有効化
  cam.start(); // 撮影開始
  printArray(settings);

  egg = loadImage("egg.png");
  hiyoko = loadImage("hiyoko.png");
  niwatori = loadImage("niwatori.png");
  chicken = loadImage("chicken.png");
```

```
  imageMode(CENTER);
}

void draw() {
  if (cam.available() == true) cam.read(); // 現在のカメラ画像を取得
  image(cam, width/2, height/2);
  noFill();
  int x, y, p1;
  float xmin = width, xmax = 0, ymin =height, ymax = 0;
  for (y = 0; y<cam.height; y++) {
    for (x = 0; x<cam.width; x++) {
      p1 = y * cam.width + x;
      float r = red(cam.pixels[p1]);
      float g = green(cam.pixels[p1]);
      float b = blue(cam.pixels[p1]);
      float diff = sqrt(pow(Rt-r, 2) + pow(Gt-g, 2) + pow(Bt-b, 2));
      if (diff <= 30) {
        if (x < xmin) {
          xmin = x;
        }
        if (x > xmax) {
          xmax = x;
        }
        if (y < ymin) {
          ymin = y;
        }
        if (y > ymax) {
          ymax = y;
        }
        size = xmax-xmin;
      }
    }
  }
  stroke(255, 255, 0);
  strokeWeight(5);
  float cx = (xmax+xmin)/2;
  float cy = (ymax+ymin)/2;

  if (size <= 100){
    image(egg, cx, cy, size, size);
  } else if(size <= 150){
    image(hiyoko, cx, cy, size*0.8, size);
```

```
  } else if(size <= 200){
    image(niwatori, cx, cy, size, size);
  } else {
    image(chicken, cx, cy, size, size);
  }
  size = 0;
}

void mousePressed() {
  loadPixels();
  int i = mouseY * width + mouseX;
  Rt = red(pixels[i]);
  Gt = green(pixels[i]);
  Bt = blue(pixels[i]);
}
```

実行例：

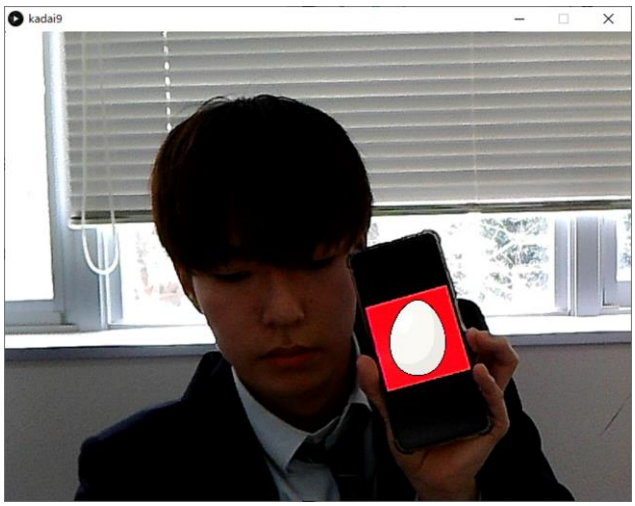

図 9-1 実行例 1

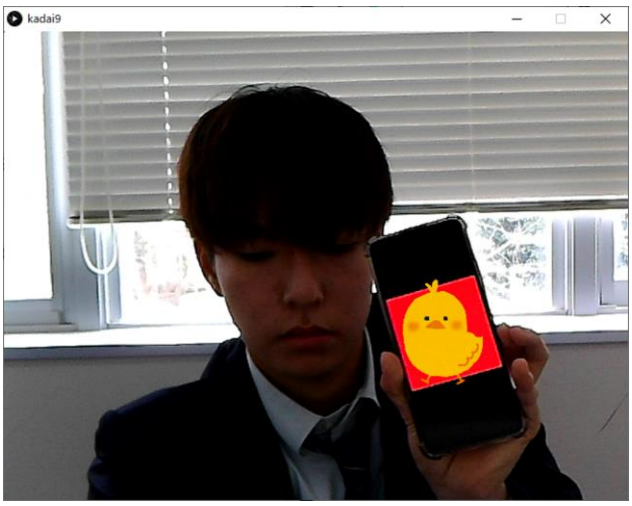

図 9-2 実行例 2

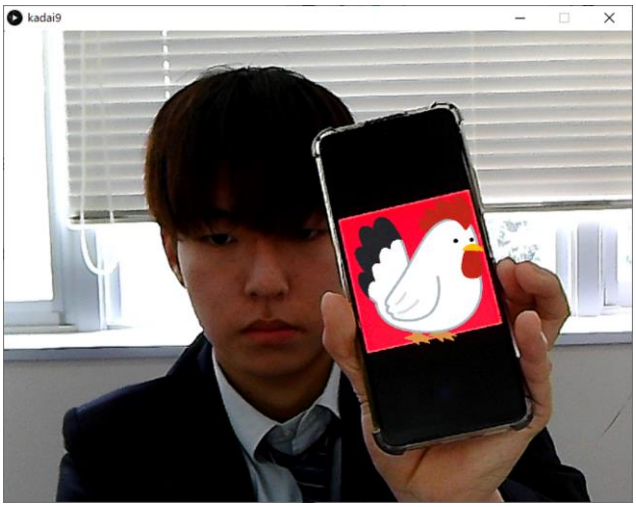

図 9-3 実行例 3

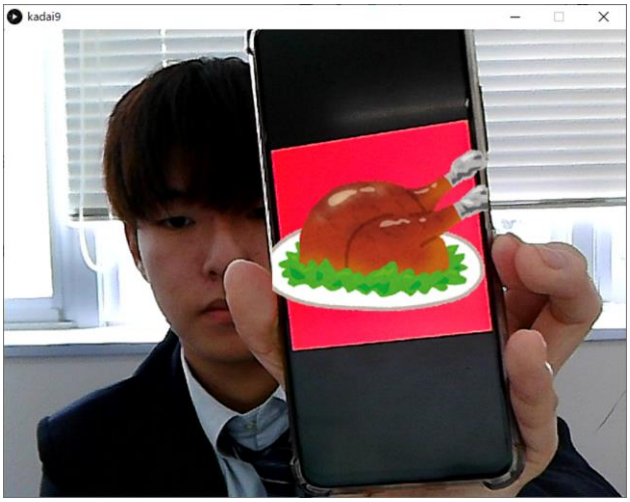

図 9-4 実行例 4

3. 感想

今回の実験で画像処理のやり方を主に学び、どうやったら赤色マーカーを読み取り易くするか、どうやったら課題 8 を発展させることができるか、など工夫する点がとても多く、大変だったが楽しかった。

画像処理ができるとプログラミングの幅も広がるので、これからも勉強していきたい。