H-068

USB カメラ映像を用いた指先によるマウス操作
Mouse Operation by the Finger using USB Camera Image

長戸 陽太 † 本田 郁二 †
Yota Nagato Ikuji Honda

概 要

USB カメラを利用し、安価かつ手軽にタッチパネルのようなマウスの操作を実現する手法を提案する。

PC のディスプレイを指先でなぞり、これを USB カメラで撮影する。その映像に対して様々な画像処理を行うことにより指先の位置の検出を行い、その指す位置までマウスポインタを移動させる。これらの処理を連続して行うことでマウス操作を実現する。

本稿では処理をリアルタイムで、かつ正確に行うための工夫と、今後の展望について述べる。

1. 原理

本手法は大まかに、以下の流れで処理を行う。ただし、1, 2 は前処理として一度だけ行い、それ以降は 3〜6 を繰り返す。

1. 射影変換行列の計算, ディスプレイ領域の認識
2. 肌色の保存
3. 肌色領域の検出
4. 輪郭線追跡
5. 指先座標の取得
6. マウスポインタの移動

以下では上記の各処理について詳細を述べる。ここでは右手での利用を前提として説明を行うが、実際は左手用に切り替える事も可能である。

1.1 射影変換行列の計算, ディスプレイ領域の認識

レンズによるわずかな歪みなどを無視すれば、USB カメラで撮影した映像内のディスプレイ上の各点 (x, y) と、実際のディスプレイ上の座標 (u, v) は次式による射影変換が可能である。

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \sim \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (1)$$

($\sim$ は同値関係を示し、定数倍の違いを許して等しい事を意味する。) そこで、射影変換行列 $H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix}$ を計算することでカメラの映像と実際のディスプレイの位置の対応付けを行う。

キャリブレーション用画像 (図 1) をディスプレイに全画面表示し、その時のカメラの映像をキャプチャする (図 2(a))。次に、画面を全て黒くし、カメラの映像を取得する (図 2(b))。これらの絶対差分を計算し、明度ヒストグラムの谷を閾値として二値化を行う (図 2(c))。

図 1: キャリブレーション用画像

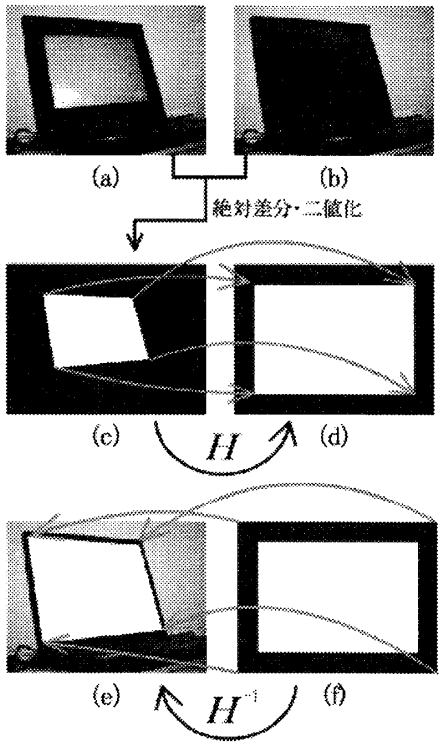

図 2: 射影変換行列の算出・ディスプレイ領域の認識

得られた二値画像の四角形の四隅と、図 1 の画像の内側の白い長方形の四隅が対応するので、これらの座標を元に連立方程式を解き、射影変換行列 H を求める (図 2(c), (d))。

次に、求めた射影変換行列の逆行列を用いて、ディスプレイの 4 隅の座標をカメラ映像内の座標に変換し、その 4 点を頂点とする四角形を求め、これをディスプレイ領域 D として保存する (図 2(f)→(e))。

単純な白、黒の全画面表示の差分を取るのではなく、図 1 の画像を用いるのは、ディスプレイの端は反射光な

†慶応義塾大学大学院 理工学研究科 総合デザイン工学専攻

どの影響でうまく四隅の座標を検出できない事が多いためである。また、画面の端に指がかかってしまった場合などにもある程度対応可能である。

1.2 肌色の保存

照明等の環境や、ユーザ毎の肌の色の違いなどによる影響を抑えるため、使用時に実際にカメラでユーザの手を撮影する事により基準となる肌色を決定する。このとき、手のしわ等で基準となる肌色の彩度が低くなってしまう事を防ぐため、数カ所の色の平均値を求め、それを保存する。

1.3 肌色領域の検出

カメラの映像と同サイズの二次元配列 $f(x,y)$ を用意し、全ての要素を 0 で初期化する。次に、以下の手続きを行うことで、$f(x,y)=1$(肌色画素), $f(x,y)=0$(その他) に各画素を分類する。

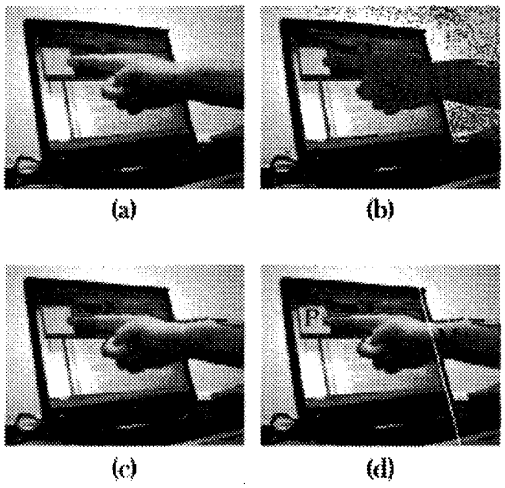

図 3: キャプチャから指先の検出まで

キャプチャした画像 (図 3(a)) をラスタスキャンし、各注目画素の色相を取得する。これが保存した肌色の色相から一定の閾値以内に有る場合、その画素 (i,j) を肌色画素として記憶する (配列の値を $f(i,j)=1$ とする)。このとき、明度が極端に高い画素や低い画素は色相が不安定なため、処理をスキップする。

次に、再びラスタスキャンを行い、注目画素 (i,j) が $f(i,j)=1$ で、かつ 四近傍の値全てが 0 の場合、$f(i,j)=0$ とする。これにより孤立点 (ノイズ) が除去される。

最後に、ディスプレイ領域の外部 $\bar{D}$ において、初期フレームとの明度の変化がある閾値を超えない部分は手領域ではないとみなし、配列の値を $f(i,j)=0$ とする。

図 3(a) に対し以上の処理を行った結果を図 3(b) に示す。

1.4 輪郭線追跡及びラベル付け

以上の処理を行うと、肌色に近い色を持った領域のみを取り出せるが、手の領域以外にも、不要な肌色領域が残ってしまうという問題が発生する。この不要な肌色領域を除去するためには、領域全体に対しラベリング処理を行い、最も手領域らしいもの以外を除去する方法が考えられるが、この方法では計算量が多く、処理に時間がかかってしまう。そこで我々は処理を高速に行うために以下の方法を採ることとする。

ラスタスキャンを行い、肌色画素の輪郭を構成する点 ($f(i-n,j)=0$ かつ $f(i,j)=1$) に到達したら時計回りに輪郭線を追跡しつつラベル付けを行う。追跡の開始点に到達したら再びラスタスキャンを行い、まだラベル付けされていない肌色画素の輪郭を探索する。この処理を全領域に対して行う事で肌色領域の外側輪郭のみを追跡することが可能である。

次に得られた肌色領域の輪郭のうちディスプレイ領域 D との共有部分の長さが最長となるものを手領域とみなし、それ以外は除去する。

図 3(b) から手領域の輪郭のみを抽出した結果を図 3(c) に示す。

1.5 指先の検出

ディスプレイの右上、右下の二点を通る直線と検出された手領域の輪郭とが交わる点の座標を取得し、それらの中点 M を取得する。

手領域の輪郭を構成する画素のうち、M からの距離が最も遠い画素を指先の画素 P とみなし座標を記憶する。

M, P の検出結果を図 3(d) に示す。

1.6 マウスポインタの移動

以上の処理により検出された指先の座標は映像内における座標であるので、射影変換行列 H を用いて実際のディスプレイの座標に射影変換する。最後に求めた座標にマウスポインタを移動する。

2. 実験

実行時の応答速度を計測するため、100 回マウスポインタを移動するのにかかる時間を 5 回計測し、その平均値を計算した。使用した PC は PentiumM 1300MHz のノートパソコン、使用言語は Visual C#である。

その結果、マウスポインタを移動するためにかかる時間は 2.38 回/秒であった。

3. まとめ・今後の展望

本論文では USB カメラの映像を元に指先でマウス操作を行う方法を提案した。

現状では処理にかかる速度が十分でないため、さらなる計算量の削減が必要である。改善案として、2 フレーム目以降の指先検出の際に前フレームの指先の座標を中心とする、ある一定の範囲に計算範囲を限定することや、ラスタスキャン時に全画素を探索するのではなく、数画素おきに探索を行うなどの方法が考えられる。

また、現在はクリックが実装されていないため実用性に欠けるので、ディスプレイへのタッチを判定する事が今後の課題である。判定の方法として、複数のカメラによるステレオ法を用いて、三次元的に指先の位置を検出する方法などを現在検討している。

参考文献

[1] ディジタル画像処理 (CG-ARTS 協会, 2006)