画像中の指先検出

秋田コアビジネスカレッジ 高度ITエンジニア科
鈴木 良太

要旨

指先には、その人の指の大きさ・色といった個性があり、私たちはそれらの影響を受けずに指先を検出できる方法をとる必要がある。また、背景のもの・太陽の光といった外部環境からの影響も受けてはならない。太っている人・やせている人、黒人・白人・黄色人種、公園・学校、自室・グラウンド、これらどの組み合わせでも完璧に動くものでなければ誰も使うことはできない。だから私たちは、だれでも100%使える仕組みを考えなければならない。

1. 序論

C++を勉強したいという、直接関係のない理由から指の検出が必要になったのは、C++で開発したいと思ったものが3D設計アプリだったからだ。私は普段、机として組み立て式のラックを使用している。このラックは、部品さえあれば自分の好きなように組み立てることができ、組み立て方次第では机にも棚にもなる。しかし、好きなようにといっても限界がある。部品の種類とその形は決まっているため、どのように組み立てれば自分の思った通りになるのかを考えながら組み立てなければならない。そうすると、自分の作りたかったものはできなかった、または思っていたよりも良くはなかったということがあるだろう。すると今度は組み立てたものを分解し、別のアプローチで組み立てをする。それを自分が気に入る形になるまで繰り返すのはとても非効率だ。そこで考えたのが、画面のなかで設計をするというアイディアだ。
KINECTを使用し、自分の手で操作することができればより自分の世界を再現できるだろうと考えた。しかし、それには指の動きや形といった情報を判断するシステムが必要だったが、KINECTにはそんな機能はない。だから私は、自分で作ることにした。指を検出し、直感的な操作を可能にするために。

1.1. 先行研究

まずは、指先を検出するにはどうすればいいのかを考える。人が、これは指だと判断することは、あまりにも当たり前のことすぎて想像しにくい。無理矢理理由をつけるならば、背景と色が違う、手から突出しているなどだろう。前者は、色というキーワードが入っている時点で判定に使用したくない材料だ。しかし後者ならどうだろう。人の指は、基本的に手から大きく突出している。これが最も多くの人に共通する指の特徴だろう。だから私は、この材料を使用して指先を検出することにした。

しかし、どうやってこれを判断させることができるのか。手から突出していると一目でわかるのは人だからで、コンピューターにこれを判断させることは難しい。ただひたすら突出しているものを指とするならば、背景に映る木の枝や鳥のくちばしなども指として検出してしまうだろう。

そこで私は、スペインのチームが開発したKINECTで指を検出するアプリケーションの論文を参考にすることにした。論文によると、彼らも手の色では不完全だとし、別のアプローチをかけている。具体的な方法としては、まず手の輪郭を座標として抽出し、その輪郭を基に手のひらの座標を検出する。そして手の輪郭と手のひらの位置から指先の座標を検出するようだ。

これで大きな流れをつかむことはできたが、問題はどうやって手の輪郭の座標を手に入れるのかだ。しかし、これに

ついてはインターネットで調べるとすぐに答えが見つかった。OpenCVというライブラリを使用すれば簡単に輪郭の座標を抽出できるらしい。私はこのライブラリを活用し、指の検出に挑んだ。

2. システム概要

このシステムを開発するには、これらの手順を踏む必要がある。

1. 手の輪郭の抽出
2. 余計な輪郭の除去
3. ノイズ除去
4. 手のひらの検出
5. 指先の検出

ここでは、これらの目標を達成するための方法を説明する。

2.1. 手の輪郭の抽出

はじめに、最初からKINECTからの映像に対して試行錯誤すると時間がかかると判断し、まずはこの画像に対して行うことにした。

OpenCVにとって輪郭の抽出はとても簡単だ。以下に、ここで使用するOpenCVの関数を記す。

関数名	内容
imread	画像を読み込む。
cvtColor	画像の色空間を変換する。画像のグレースケール化に使用。
Threshold	配列の要素に対して，ある定数での閾値処理を行う。画像の二値化に使用。
findContours	二値画像から輪郭を抽出する。

関数findContoursを使用するには二値画像をつくる必要がある。そのため、関数cvtColorと関数thresholdが必要となる。関数findContoursをRGB画像に使用するためには、必要となる手順だ。これにより、輪郭を抽出することには成功した。

2.2. 余計な輪郭の削除

手の輪郭を抽出することには成功した。しかし、画像左上にもうひとつ別の輪郭ができてしまった。これを残したまま次のステップに移ることはできない。

では、どうすれば余計な輪郭を除去できるのか。まずは二値画像を確認する。

見てのとおり、輪郭を抽出している部分には、白い塊ができている。私がほしいのは、手の形をした塊だけ。左上の塊はゴミでしかない。そこで私は、これを塗りつぶしてしまおうと考えた。

では、どうやってこの一塊をピンポイントに塗りつぶすか。それはOpenCVが輪郭の座標をどのように管理しているか、これを調べた結果解決した。

OpenCVが抽出した座標は構造体を含む二次元配列に保存されている。一次元目は輪郭A、輪郭Bといった、抽出した輪郭をグループごとに分けている。そして二次元目には構造体が入っていて、その構造体にx軸、y軸の情報が入っていた。ここで重要なのは一次元目。グループごとに分かれているならばとれる手段がある。先の画像を見てみると、明らかに手の輪郭のほうが大きいのがわかる。それはカメラに写したいものが手であることから、手がカメラに一番近い位置にあるからだ。仮にこの画像の背景がもっと複雑であったとしても、手の輪郭を超えることはないだろう。つまり、各配列の中に入っている構造体の数を数え、その数が一番多いものが手の輪郭であろうと判断する。そして、そうでない輪郭をつくりだす塊を塗りつぶせばいいと考えた。

ならば、どうやって塗りつぶすか。これはさすがにOpenCVも自動でやってはくれない。そのため、自分で塗りつぶしのアルゴリズムを考える必要があった。まったく調べずにつくった私のアルゴリズムは、四近傍と呼ばれるものらしい。上下左右のいずれかの方向に塗っていくというものだ。このアルゴリズムに関しては、探せばいくらでも見つかるためここには記述しない。

こうして塗りつぶした結果、画像内に手型の塊のみを残すことに成功した。

2.3. ノイズ除去

前の手順でつくった画像を見ると、手のなかに黒い塊がいくつもできている。おそらくこれは手のしわによるものだろう。しかし、これをこのままにしておくことはできない。なぜなら、次の手順である手のひらの検出に悪影響を及ぼすからだ。

なぜ悪影響を及ぼすのかを説明するために、まずは手のひらの検出方法を説明する。手のひらを検出するには、輪郭の座標と手の座標が必要となる。手の座標というのは、手を構成している白い塊の一画素の座標という意味だ。まず手の一画素を決める。その一画素と輪郭を形成している点の座標との距離を一つずつ測っていく。測ったなかで最小の距離と一画素の座標を配列に入れ、次の一画素へと移る。これを、手を構成している白い塊すべての画素で行い、それぞれの最小距離と一画素の座標を配列に入れていく。距離を測り終えたら、距離が入った配列のなかから、距離が最大のものを取り出す。こうして取り出した一画素の座標が手のひらである。

では、どうやって手の一画素を決めるのか。私が出した答えは、画像左上の(0, 0)の座標にある一画素からほぼ総当たりで調べるというものだ。ほぼ総当たりというのは、ある程度の数、例えば10ずつスキップして計測していく方法をとったからだ。そうしないと、時間がかかりすぎる。もっと効率的な方法はなかったのかと思うかもしれないが、まったく思いつかなかった。左上からほぼ総当たりで画素にアクセスしていき、その一画素の色が白だったら手だと判断する。これが確実に手の一画素を決められる方法だった。

しかし、手のなかに黒い部分が残っているとどうだろう。スキップした先がその黒い部分だった場合、さらにスキップされることになる。そこが手のひら付近でなければいいが、手のひら付近だった場合、手のひらの位置が大幅にずれる。これでは検出する意味がなくなってしまうため、黒い塊をなくさなければならない。

具体的な方法としては、画像をぼかしてから二値化するというものだ。画像をぼかすことによって、手のしわを検出しないようにする。以下に、ここで使用する関数を記す。

関数名	内容
GaussianBlur	ガウシアンフィルタを用いて画像の平滑化を行う。

ガウシアンフィルタとは、ガウス関数をもちいて画像をぼかす処理である。ガウス関数についての説明は省略する。もとの画像を関数GaussianBlurでぼかしてから、グレースケール化、二値化という手順を踏むことで、黒い塊(ノイズ)を除去することができた。

2.4. 手のひらの検出

手のひらの検出方法は、前の手順で説明したとおりだ。以下の画像がその結果である。

2.5. 指先の検出

先の手順でようやく準備が整った。指先を検出するには、輪郭の中から3つの座標を選ぶことから始まる。次の画像を見てイメージしてほしい。

点A、点B、点Cがある。点Bが指先の頂点にあたる。点BA間、点BC間をそれぞれ直線で結び、その時にできた角度θを求める。角度θは、内積の定義を使って求める。こうして得た角度θが、あらかじめ決められた数値以下であれば、それは指先である。以降、決められた数値のことを定数αとする。

点BA間の直線と点BC間の直線の長さは、固定である。なぜなら、輪郭は手首付近もとられている。例えば、点Aを親指の先、点Bを小指先、点Cを手の付け根というとり方も可能となる。これでは、ほぼすべての点を指先として検出してしまうため、直線の長さは固定しなければならない。

しかし、これでもまだ十分ではない。各指には、いくつもの指先が検出されている。これは、定数α以下の角度θをもつ点が、指先付近にはいくつか存在するからだ。それぞれ点一つ分だけ移動するだけの指先もあるだろう。これを各指一つだけにしなければならない。なぜなら、指と指の間にも指先を検出している。これを防ぐために必要なのだ。具体的な方法は、指先と手のひらの距離を使用する。指先の点Bと手のひらの距離をそれぞれ測り、保存しておく。すべての点との距離を測り終えたら、その中から距離が離れている上位五つのみを残す。こうして残ったものが指先である。しかし、一つの指に複数の指先を検出してしまうとどうだろう。指先は、最も距離があいている中指に集中してしまう。これでは結果としておかしいため、各指に指先は一つというルールを徹底しなければならない。

ではどうやって一つにするのか。三つの点がつくる角度θがほぼ直線だった場合、そこは指の横側ということになる。三つの点は、輪郭という川を流れるように移動していく。急なカーブになると角度は鋭くなり、直線になると角度は平らになる。つまり、角度θがほぼ直線になった時点で、そこはもう指先ではないと判断し、それまでに検出した指先の中から角度が最小のもののみを残せばいい。角度が小さければ小さいほど指の頂点に近いため、最適な一点を残せる。

よって、五つの指先を検出することに成功した。

3. KINECTへの移植

静止画では、成功した。次はKINECTから送られてくるデータでも正確に動作
るかだ。しかし、結論から言うとそれはできなかった。

理由は、背景だ。OpenCVで二値化する際に、背景と手が一体化してしまうことが多い。この問題の対応策はある。KINECTの深度カメラを使用することだ。しかし、私にはそれをする時間がなかったため、この問題は未解決のまま終わった。

4. 結論

人の指を検出する方法は掴んだ。しかし、外部環境に左右されては誰も使ってくれない。使用したKINECTが古いという問題も大きかったが、それでも不可能ではなかった。背景という問題とその対応策に気づいていながら完成させることができなかったことは残念である。

しかし、成果もある。OpenCVという便利なライブラリを知ることができ、二値化や内積の定義といった幅広い知識を手に入れることができた。

5. この開発の今後

私は、これ以上この開発を続けるつもりはない。C++という言語を使用して基礎を知るという目的は達成したため、後付けでついたこの開発を続ける意味はない。

もし、この続きが見たいのであれば、自分で続きをつくってほしい。

参考文献

F.Trapero Cerezo (2016) 「3D Hand and Finger Recognition using Kinect」<> 2016年10月8日アクセス.

西田五郎 (2014) 「OpenCVで輪郭抽出から隣接領域の切り出し(その1)輪郭抽出まで」, <http://independence-sys.net/main/?p=1619> 2016年アクセス.

「OpenCV 2.2 C++ リファレンス」,<http://opencv.jp/opencv-2svn/cpp/> 2016年アクセス.

hmichu (2016) 「[OpenCV][Mat]画素へのアクセススピード比較」,<>2016年アクセス.