Webカメラの映像からの目検出ソフトの作成

３年Ｃ組　稲益　秀成
指導教員　米田　隆恒

１．要約

今回、私は、コンピュータに搭載されているWebカメラを用いて利用者の顔を撮影し、その映像から利用者の目線を検出する方法を研究した。

キーワード　目、目線、Webカメラ、ポインティングデバイス、アイトラッキング

２．研究の背景と目的

人間は他人の目を見ると、その目線と目線の先にあるものから、その人が何を見ているのかをおおまかに知ることができる。そこで私は、人間の目は視覚情報を得るという感覚器官としての機能だけでなく、自分の視覚情報を他人に知らせる情報伝達器官としての機能も持っているのではないかと考えた。ここで今日幅広く普及しているコンピュータやスマートフォンなどが人間の目線を情報として取得できれば、これらの電子機器の使い方が広がるのではないだろうか。例えば目線でマウスを動かしてハンズフリーで操作するなど、目線をポインタ代わりに使うことも考えられる。

現在、コンピュータのマウス操作に加えて、スマートフォンやタブレットPCなど、ディスプレイを指でタッチして操作するデバイスが増えてきているが、この操作方法には問題点がある。それは、指でタッチ操作している間は指でディスプレイが隠されてしまうという点である。文字入力時のカーソルの位置を指定するとき、従来のマウス操作ではディスプレイを見ながら操作でき、カーソルをおきたい場所とカーソルが重なってしまっても、カーソルが小さいため、その場所がカーソルで隠れてしまうことはなかった。しかし、タッチ操作でこれと同じことをしようとすると、カーソルをおきたい場所がどうしても指で隠れてしまうので、カーソルをおきたい場所を見ながらそこに正確にカーソルを移動させるのは難しい。

「画面を見ること」自体が操作方法になればこの問題点を解決することができる。

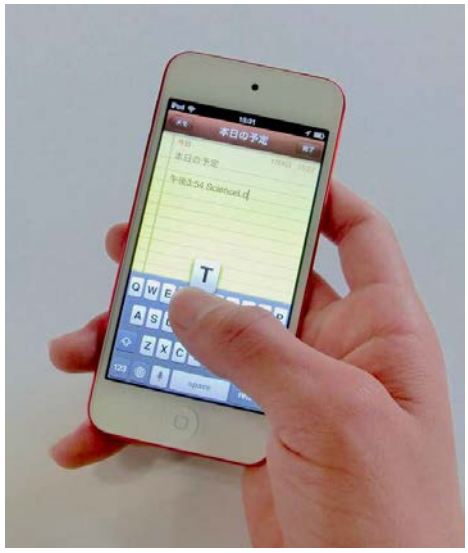

図１　タッチ操作のデバイスの例

そこで、私はコンピュータが利用者の目線を得る方法について考えていくことにした。そのため本論文では、Web カメラの映像から、黒目の位置を検出する方法について報告する。

３．研究内容

今回、コンピュータの利用者の目線を取得するために Web カメラを用いた。

図２　一般的な Web カメラの例

図３　スマートフォンなどに
搭載されている Web カメラ(内蔵カメラ)

Web カメラとは、図２のような Skype や FaceTime など、インターネットでテレビ電話を楽しむときに相手にリアルタイムで送信するときに自分の顔を撮影するカメラのことで、図３のように最近のコンピュータやスマートフォンや、タブレット端末などには同じ用途のカメラが標準で内蔵されている。

これを利用することで新たな装置を追加しなくても、目線の検出ができ、手軽に使用することができるという利点がある。

Web カメラの映像から利用者の目線の動きを検出する流れとしては、

（１）Web カメラの映像から利用者の顔部分のみを切り出す。
（２）切り出した顔の範囲から目の位置を検出する。
（３）目の位置から、目の中心を計算し、その動きから目線を計算する。

である。

［１］顔検出をしない方法

目を検出するときに、最初に顔を認識し、そこから目を検出した方が検出精度は上がるが、顔を正確に認識するには多くの処理を必要とし、そこからさらに目を検出しようとすると、プログラム処理が多く、すごく重くなってしまうのではないかと考えた。あくまで、マウスやタッチパネルに代わるものをつくりたいので、このポインティングデバイスを使うことでコンピュータのパフォーマンスが悪くなってしまってはいけない。そこで、Web カメラの映像から、まず顔を検出するのではなく、目を最初に検出できれば、顔検出の処理がなくなるので、プログラム処理が少なく、軽くできると考え、目だけにみられる外観の特徴を探してみた。そして、目の外観の特徴として、「黒い」、「丸い」、「２つ横に並んでいる」

という特徴を挙げた。しかし、これだけでは顔の中だけでも、眉毛、鼻の穴など同じような外観のものがあり、とくに鼻の穴は黒目とよく似ており誤認識をしてしまう。よって黒目と鼻の穴を分けられることがまず重要だと考えた。この二つを判別するために、目の特徴である白目も認識に使う方法を考えた。黒目の横には白目があるのに対し、鼻の穴の周りには白色はない。つまり、白色と黒色が隣り合っている部分を探せば、そこは鼻の穴ではなく、目である可能性が高いだろうと考えた。そこから、最初に白色と黒色を検出し、それが隣り合っている場所を検出するというプログラムを作成した。

［１］の結果

思っていた通り、黒色と白色が隣り合っている場所を探すプログラム処理は重くはなかった。しかしながら、図４のように顔だけでなく、背景にもたくさん反応してしまった。ここから、背景と目を判別するのは、逆にたくさんの処理が必要になり、結果的に処理が重くなってしまう。よって、黒色と白色のみを検出することで目を検出することは難しいと考え、先に顔の範囲を検出してから、この処理を行う方が良いと考えた。

図４　［１］の実行結果

［２］顔の範囲の検出

［１］の方法で作成したプログラムでは目の認識の誤差が多く、実用には難しいだろうという結果になってしまった。そこで、プログラムの処理は増えるが、顔を認識してから目を検出する方法をとることにした。

まず有名な顔検出方法として、Viola－Jones 法というものがある。

図５　Viola－Jones 法での探索窓の例

（http://makematics.com/research/viola-jones/ より引用）

これは、顔検出を行いたい映像に図５のような探索窓を順番に動かしていき、その探索窓の領域をあらかじめ作成しておいた識別器から、顔か顔でないかを判定するという方法である。しかしこの方法では、左上から順番に１ピクセルごとに見ていかなければいけないので、処理が重いだろうと推察した。１枚の画像から顔を正確に検出するのには問題はないが、Web カメラの映像からリアルタイムで顔を検出し、さらにそこから目を検出するには、この Viola－Jones 法は少し処理が重そうだと思い、他の軽い処理方法で顔を認識できなければならないと考えた。

そして、軽い処理で顔を検出する方法として、映像の中心付近にある肌の色を検出する方法を用いた。

Web カメラは利用者の顔を撮影するためのものなので、映像の中央付近に顔が来るようなカメラ位置に設置されている。よって、映像の中心付近の肌の色を検出すればそこは顔、または首であるのが普通なので、この方法を用いることにした。

肌の色を検出するには、コンピュータで普段色を値で表現するときにつかう RGB ではなく、色相の値を用いることにした。これは、色相の値は RGB に比べて、陰などの影響や照明による色の変化が小さいと考えたからである。

また、顔の位置が移動しても追尾できるように、最初のフレームでは映像の中央が肌の色検出の基本となる点（以降、「基準点」と記す）だが、それ以降のフレームでは前のフレームでの顔の位置の中心を基準点とするようにした。

基準点から顔の範囲を検出していく方法としては、処理が少なくなるように、肌の色の範囲を全部調べていくのではなく、基準点から上下左右といったように方向でみていき、それでできた十字に外接する四角形を顔の範囲とすることにした（図６）。

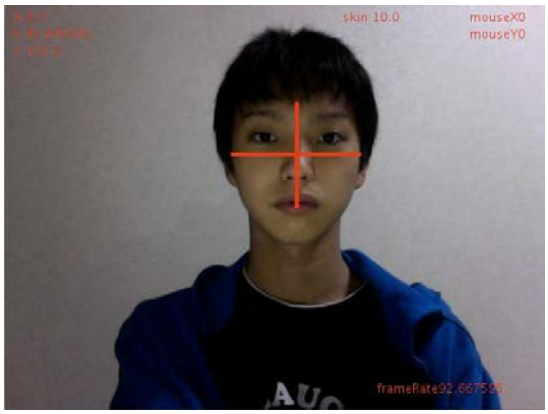

図６　基準点から上下左右方向に
調べている様子

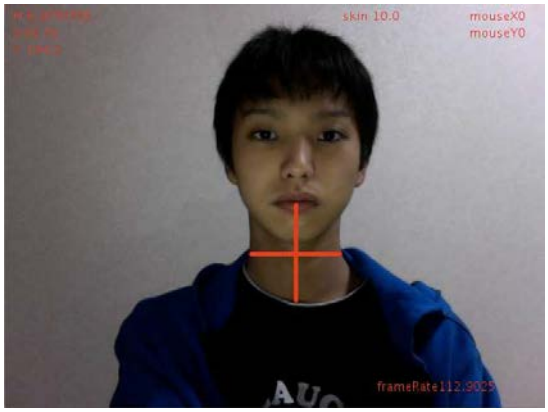

図７　首を顔だと誤認識している様子

しかし、基準点から上下左右の４方向だけでは、基準点がもし首にきてしまった時に図７のようにロで肌の色検出がとまってしまい、目のある顔まで範囲選択できないので、上下左右だけでなく、上方向の端からさらに左右に調べ、その左右それぞれ端までの半分の地点からさらに上方向に調べ、

その左右それぞれの分の上方向の端の２点の中間地点からさらに上方向へ調べるようにした。

［２］の結果

上下左右だけでなく、さらに上方向に何回か肌の色検出の処理を増やしたことで、もし基本となる点が首にきてしまっていても、口や目を超えて、額のところまで範囲をのばすことができるようになり、かなり認識精度がよくなった。また、プログラム処理がとても軽いのも良かった。

しかし、顔に陰ができてしまっていたり、全体的に暗いときはWebカメラからの映像にノイズが出てしまったりしてしまう。このノイズのせいで、色相が大きく乱れてしまい肌の色の検出が安定しなくなってしまうことがあった。

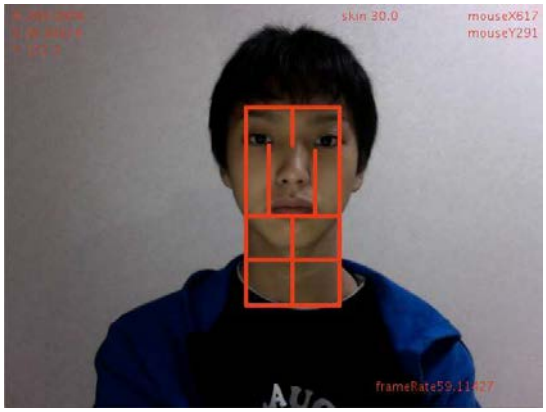

図８　［２］の実行結果

［３］目の位置の検出

［２］から、顔の範囲を検出することができたので、この顔の範囲で［１］で考えていた目の検出方法を使うことにした。その結果が図９である。

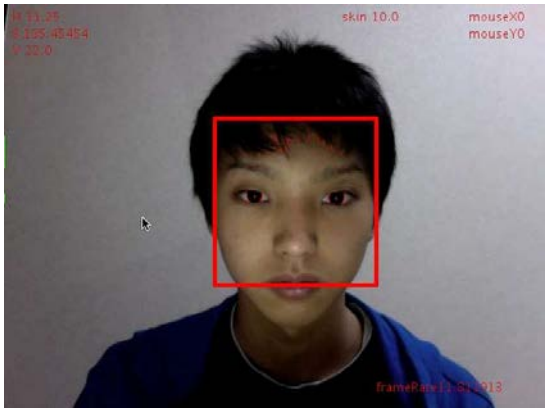

図９　黒色と白色が隣り合っている部分を赤で示している様子

白色と黒色が隣り合っている部分を赤色で表している。そして、［２］で検出した顔の範囲を横１列ずつで見ていき、黒色と白色がとなりあっている点がその列に１個以上８個以下あるところに目があるとする。コンピュータを操作するときは普通２つの目は横方向に並んでいるので、その列には黒色と白色がとなりあう点が４個あるはずである。８個以下とした理由は、黒目にディスプレイの映像が反射してしまい、目の中心部分が白くなってしまうことがあり、その場合、黒色と白色がとなりあう点は８個あることになってしまうからである。

顔の中心から上で、目があると予想される列が集まっている所の、一番上側と下側の中間の位置にある列を、目がある列とする。そして、その列にある黒色と白色が隣り合っている所と、黒い部分を探していき、映像に映っている目の中心を特定できると考えた。

［３］の結果

［２］の方法で顔の範囲を特定し、その領域内で［１］の処理をすることで、安定して

目の位置を検出することができた。また、先に横方向の列だけをみて、目の中心がある一列を見つけてから２つの目を検出しているので、とても軽く、［２］の顔検出の処理と合わせても、目的であるポインティングデバイスとして実用可能だと思えるくらいの速度が出た。

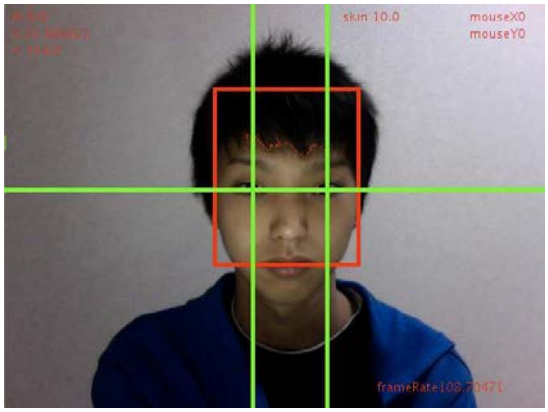

図 10　［３］の実行結果

［４］輪郭を使う目の中心の検出

映像中の目の位置の検出までは完成したが、今回の研究の目的である「目線をコンピュータが認識する」ためには、正確に目線を検出する必要がある。映像中での黒目の位置から目線を検出するためには、目の中心の位置を検出しなければならない。

目は球体つまり立体であり、Web カメラの映像は平面的である。したがって、本当の目の中心を計算することはとても難しい。よって、出来る限り精度が良くかつ処理が軽い方法を考えた。

黒目全体のピクセルすべての平均を用いた方が、より正確かもしれないが、それは処理が多くなってしまうだろう。そこで今回用いたのは、黒目の輪郭のピクセル位置の平均を用いるという方法である。

輪郭を取得する方法としては、

（１）輪郭を取得するために、黒目と白目の境目を検出する。

（２）［３］で取得したおおまかな目の位置から、輪郭追跡の開始点を決める。

（３）その点から、周りの８ピクセルを調べていき、隣の輪郭を発見する。

（４）開始点の隣の点の周りの８ピクセルを（３）と同様に調べ、さらに隣のピクセルを探す処理をする。

（５）（４）と同様の処理を続ける。

という流れである。

そして開始点に戻ってきたとき、（５）の処理を終了し、得られた輪郭のピクセルの平均の位置を計算し、それを目の中心とするという方法である。

［４］の結果

強い光源がカメラのある方向にあると、黒目にその光源が反射してしまい、輪郭抽出が失敗してしまうことがあった。

もし、強い光源がディスプレイだけならば、その光源は黒目の中央で反射するので、黒目の輪郭には影響しないが、部屋の灯り等は黒目の輪郭付近で反射してしまうことが多く、そのせいで正しく輪郭を抽出することができなくなってしまった。

よって、この輪郭を用いて中央を検出する方法を用いるのをあきらめた。

図 11　［４］の結果

［５］重心を使う目の中心の検出

輪郭を用いての中心の計算は難しいということがわかった。そして、黒目の範囲が正確に検出できていなくても、ある程度の精度で目の中心を計算する方法としては、黒目の重心を計算する方法をとった。

こうすることで、［４］に比べて誤差を減らすことができる。

［５］の結果

黒目の重心を計算することで、たとえ目の中心や輪郭付近が照明で反射してしまっても、目の中心を少ない誤差で計算することができるようになった。

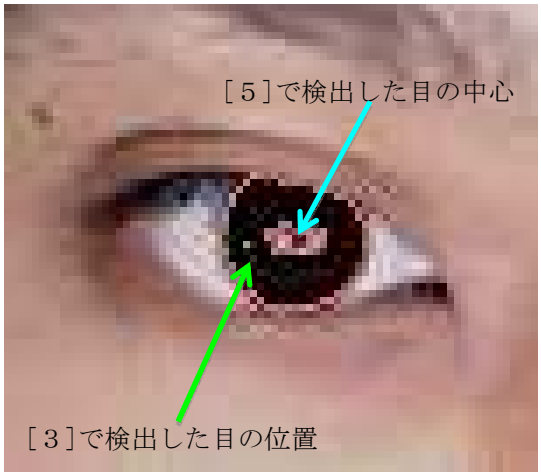

図 12　重心を計算している様子

５．考察

今回、コンピュータで利用者の目線を取得するために、Web カメラで利用者を撮影し、その映像から目線を検出する方法を研究した。

映像から顔の範囲の検出、そこから目の位置の検出までの処理はとても軽く、このアプリケーションをバックグラウンドで起動させておいても問題ないものができた。そして、重心を計算することで、目の中心の位置を検出することができた。

問題点は、Web カメラの映像にはノイズがあるために、肌の色による顔の範囲の検出が安定しなくなってしまうことがあるという点である。顔の範囲選択が安定していないので、全体の処理も安定しなくなってしまっている。これは精度がよく、かつ処理の軽い方法を考える必要がある。

６．今後の展望

今回検出した目の中心を目線とし、それをコンピュータのマウス操作などに活用できるようにしていくことが次の課題である。

しかし、今回このアプリケーションは Processing を用いて制作したが、この言語ではアプリケーションからマウスの位置を変更するなど、複雑な動作をさせることは難しい。Mac ならば Objective-C といった、その OS に特化した言語で開発しなければいけないので別の言語を勉強する必要がある。

これが完成すると、まずは目線でマウス操作ができるようになり、現在あるコンピュータに何もデバイスを追加せずに、今まで以上にコンピュータの操作が簡単になるだろう。

また、体の不自由な人でも目線の動きだけでコンピュータを操作できるようになるので、例えば、画面に「あいうえお…」といったような文字の表を表示しておき、それを一定時間見ることでその文字を選択したことにすれば、文章入力はもちろんだが、その文字を音声で出力することで、体が不自由な人でも会話することができるようになる。このような社会貢献につながると考えられる。

さらに、別の考え方として、利用者がコンピュータを使っているときに、ディスプレイのどこを見ているかがわかると、その利用者の興味がどこにあるかがわかることになり、例えばインターネットブラウザでは、そこからその利用者の興味がある内容を知ることができたり、逆にどのような内容が読んでもらえるかを今まで以上に調べることができたりするようになり、うまく使えば、Web サイトの質がよくなるようになると考えられる。

７．参考文献

[1] メディアンフィルター
http://www. gifu-nct.ac.jp/elec/yamada/iwata/median/index.html

[2] Delphi 応用編
http://www2s.biglobe.ne.jp/~aks-lab/delphi_part3_1.html

[3] コンピュータのセカイ- 今そこにあるミライ
http://news.mynavi.jp/series/computer_vision/010/index.html

[4] Processing Reference
http://processing.org/reference/

[5] RGB から HSV への変換と復元
http://hooktail.org/computer/index.php?RGB%A4%AB%A4%E9HSV%A4%D8%A4%CE%CA%D1%B4%B9%A4%C8%C9%FC%B8%B5

[6] C 言語による輪郭追跡処理について
http://homepage2.nifty.com/tsugu/sotuken/binedge/

[7] 力のつりあい
http://www.kdcnet.ac.jp/college/buturi/kougi/buturiko/mechanics/mechan5/mechan5.htm#center

８．謝辞

今回の研究にあたり、サイエンス研究会物理班顧問の米田隆恒先生、藤野智美先生には、研究に関する助言や指導をしていただきました。また、同研究会のメンバーには研究のサポートをしていただきました。この場をかりてお礼申し上げます。