JPHACKS におけるアイトラッキング技術の開発

ずっとエイリアンでいいのに

2020 年 11 月 7 日

1　はじめに

JPHACKS1 日目, 我々はある決断をした.

「漫画のコマを見ると音がなるサービス」この開発には, 正確なアイトラッキング技術が必要であった. 我々は当時, アイトラッキングの技術は発展しているしこの部分は問題ないだろうと楽観視していた.

しかし, 現実はそんなに甘くなかった. 既存の Web カメラを用いたアイトラッキング技術では, 求める精度には及ばなかった. 我々は決断を迫られた. アイデアを変えるか, それともこのまま続けるか.

確かに, 既存のアイトラッキングの精度はそれほど高くはない. しかしこれはハッカソンだ, そこまで要求はされないだろう. その考えが頭によぎった. そう, これが 2 日間のハッカソンならそれで良かったのだ.

我々には 1 週間の時間が与えられた. 既存の技術を使うなら 2 日あれば終わってしまう. 我々にはそれ以上が求められた...

よかろう, それなら, アイトラッキングを自作してやるよ.

2　先行研究

アイトラッキングについての知識は一切無かったため, 既存技術の調査を行った. 以下はその代表的なものである.

2.1　角膜反射法

非常に精度の高いアイトラッキングシステムとして, 角膜反射法を用いたものがある. これはカメラ位置から赤外線を飛ばし黒目に投影された点と黒目の位置 (正確には瞳孔とプルキニエ像) の位置関係によって視線を推定するという技術である. (図 1)

精度が高く多くのアイトラッキングシステムに組み込まれているが, カメラ位置から赤外線を発

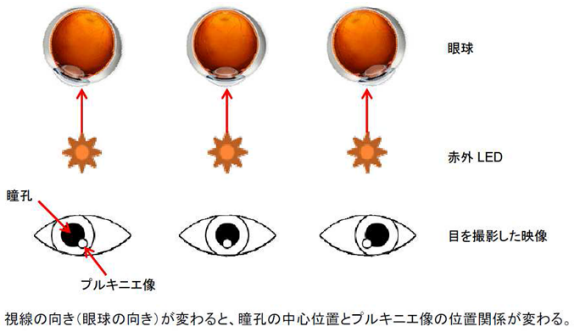

図 1　角膜反射法

する必要がある為, 通常のウェブカメラではなく専用のセンサーが必要となる. 本開発では, 汎用的なアイトラッキングシステムとして PC 付属のウェブカメラを利用したい. その為, 角膜反射法を利用する事ができない事になる.

2.2　WebGazer

ウェブカメラを用いたアイトラッキングシステムはいくつか存在する. その内の一つが WebGazer である. WebGazer はオープンソースのプロジェクトでブラウザ上で実行可能なデモも公開されている.

他にもウェブカメラを用いたアイトラッキングシステムは存在するが, ブラウザ上で動作する物は少ない. また, ブラウザ上で動作しても WebGazer と精度は似ている印象を受けた. WebGazer は論文も公開しており, 詳細な実験結果等を確認可能である. [2]

WebGazer のアイトラッキング手法を簡潔にまとめると, 下記のようになる.

1. カメラ画像から目の画像を抽出
2. 抽出した画像を前処理
3. 目の画像からすくリー座標を予測

これらは次の章で詳しく説明を加える.

3　独自実装

既存システムでは私たちの求める精度は得られなかった. そこで我々は WebGazer をベースに, より精度の高いアイトラッキングを開発することにした.

コードの拡張性を増やす既存のライブラリを用いず, まずは WebGazer の必要部分のみを簡潔に再現することにした. いわば車輪の再発明であるが, WebGazer のコードは数々の実験によって膨れ上がっており, またコードのミスも散見された. 従って情報を絞り拡張性を高めるこの作業には大きな意味があったといえる.

以下は WebGazer の論文を読み, それを元に実際に我々が実装した手法である.

3.1 Facemesh による目の画像抽出

Facemesh と呼ばれる顔認識を行いキーポイントを抽出する事ができるライブラリを利用し, ウェブカメラの情報から目の位置を推定, 画像を切り抜いた. Facemesh はブラウザ上で動作する程軽いため, サーバーとの不要な通信も避ける事ができた.

3.2 目の画像の前処理

3.2.1 解像度

Web ブラウザ上で動作させる為, 重い処理はなるべく避けたい. 画像の解像度は計算量に多きな影響を与えるため, WebGazer では片目を縦 6 ×横 10 に分割しており, 我々もそれに習う事としに習う事とした.

3.2.2 グレースケール化

目の色の情報は視線推定には関係が薄い. 計算量の削減のためにグレースケールに変換する.

3.2.3 ヒストグラム標準化

グレースケールでも画像によって光の加減が違う. 暗い画像や明るい画像が同等に扱われてしまうと, 問題が生じる可能性が高い. その為, 相対的な色の情報として捉えられるようにヒストグラム標準化というものを行う.

3.3 目の画像からスクリーン座標を推定

目の画像は前処理を経て縦 6 ×横 10 次元の画像が両目で, 合計 120 次元の情報となる. これを 2 次元のスクリーン座標に変換する必要がある. WebGazer ではこれを, リッジ回帰で実現していた.

しかし, tensorflow.js は逆関数や LU 分解が実装されておらず, リッジ回帰は非常に導入コストが高い. そこで我々はこの部分においては tensorflow.js を用いた学習によって, リッジ回帰と同等の内容を実現した.

3.4 キャリブレーション

上記内容を事前に得たラベル付きデータからリッジ回帰の係数を計算する. 大抵の場合, これはブラウザで各個人に合わせてサービス利用直前に行われる.

例えば画面に表示された点を目で追う事で点の位置を正解データ, その時の目の画像を入力とする. この作業はキャリブレーションと呼ばれる.

4　提案手法

我々はここでようやくスタートラインに立った. WebGazer の精度を超えるべく, 我々の考えた新しい手法を提案する. (もしかしたら既に行われている手法もあるかもしれないが, 少なくとも WebGazer にはない我々が考えた新しい手法である.)

4.1　CNN の利用による解像度の向上

4.1.1　概要

WebGazer では, 縦 6 ×横 10 次元の目の画像から 2 次元の座標を予測していた. これは明らかに解像度が低いと言える.

WebGazer では, 120 次元の画像データを 2 次元に変換するのにリッジ回帰を用いていた. しかしリッジ回帰は LU 分解を必要とし, これが $o(n^3)$: であるために画像の次元を増やせば爆発的に計算量が増えた. この為, 目の画像の解像度をあげられなかった.

我々は, この問題を解決するために CNN を導入することにした. CNN は並列処理かつオンラインで学習をできるため, 画像の次元数が増えても処理時間の増え方はそれほど大きくはない. これによって, 画像の解像度をあげる事が可能となる. 画像の解像度が上がればより細かい目の動きも反映ができ, 精度が上がるのではないかと考えた.

4.1.2　実験方法

ブラウザ上で CNN を使う為に tensorflow.js を利用して, WebGL をバックエンドに使用した. これによって, サーバーとの通信を必要とせずフロントエンドのみで学習を可能とした.

この条件で画像の解像度をあげて, 精度の変化を確認した.

4.1.3　実験結果

画像解像度を 2 倍にあげたところ, 学習時間は大きくは変わらなかった. しかし精度についても, 少なくとも悪くなってはいないが目立った向上はないように思えた.

4.2　キャリブレーション方法の変更

WebGazer ではマウスをクリックした位置をキャリブレーション用のデータとしていたが, これは手間であり, また人によって差が生じる. 従って, 基本的に自動で動く点を目で追いかける事でキャリブレーションを行うことにした. この点の動かし方についてもいくつかの手法を試した.

4.2.1　実験方法

1. 左上から右下に横方向に走査 ブラウザ左上から, 右に向かって点を動かし右端に着いたら少し下に動いてから左に移動. これを繰り返す.

2. 右下から左上に横方向に走査 ブラウザ右下から, 左に向かって点を動かし左端に着いたら少し上に動いてから右に移動. これを繰り返す.
3. ランダムに走査 ランダムな位置に点を表示
4. 全体的に走査 左上, 右下, 右上, 左下, 中上, 中下, 左中, 右中の順に走査

4.2.2 実験結果

1, 2 は上, 下に偏り, 3 は微妙, 4 を採用

4.3 補正

4.3.1 顔の向きをもとに補正

4.3.2 顔の位置をもとに補正

4.3.3 顔の大きさをもとに補正

4.3.4 中央値を利用

4.3.5 カルマンフィルター

5 実験・評価

5.1 実験設定

5.2 実験結果

5.3 考察

6 おわりに

その気になればアイトラッキングを開発する事も出来る. 諦める事を諦めた. 満足している. とても楽しかった. 漫画のコマについてもやった.

参考文献

[1] 目は口ほどに物を言う。「アイトラッキングの原理とアウトプット例」 https://techblog.nhn-techorus.com/archives/3435

[2] WebGazer: Scalable Webcam Eye Tracking Using User Interactions http://cs.brown.edu/people/apapouts/papers/ijcai2016webgazer.pdf