

かわさきマシンのための画像認識アルゴリズム

2003年11月24日
篠原 隆之(OB)

手動操縦型ロボットの大会であるかわさきに自動操縦型ロボットで参加しよう、という中上氏の挑戦に乗せられて、かわさきマシンに搭載するための画像認識プログラムを開発しました(当時私は学生でした)。そのプログラムは最初「しりこん・ちつぶ」(第7回)に搭載され、そのまま「しりこん・あいりす」(第8回)「まるちぷらい」(第10回)とアルゴリズムを変えることなく受け継がれています。

ここではその処理の内容とアルゴリズムについて解説します。

1. 要求の定義

「らびすらすり」型の展開マシンがとる攻撃方法はおおざっぱに言えば、リングの反対側にいる敵に狙いを定めて一気にアームを展開して攻撃しておしまい、というものです。このシーケンスでマシンに要求される自律性は、

- ・敵に狙いを定めること
- ・アームを展開するトリガを引くこと

の2点だけです。

そこで、マシンに搭載したカメラの画像から敵の機体を検出し、方向と距離を推定することにしました。方向と距離が分かることにより、アームの展開方向を調整し、敵が射程距離まで近づいて移動が安定した瞬間を見定めてトリガを引く、という機能が実現します。

2. 入力画像の特徴

実際にマシンに搭載したカメラのサンプル画像が図1です。カメラはマシンの高い位置に光軸が水平になるように固定してあるので、画像の下半分に敵の機体とリングが見えます。撮影したのは練習走行会なのでリング上の障害物の配置などが本番とは異なりますが、ここからいろいろなことが分かります。

- ・下半分の画像には、敵機、障害物、白線、ロープ、ロープ越しの背景が見える。
- ・会場の照明が反射して、リング上の障害物は部分的に光って見える。
- ・リング上の白線や2本のロープが白い横線として見える。
- ・背景には人が見える。
- ・敵機はごちゃごちゃしたかたまりのように見える。

3. アルゴリズム

入力画像は256階調(8 bits/pixel)のグレースケール画像です。画像の下半分を切り出して以下の処理を行います。

3-1. 水平方向の高周波成分を抽出する

画像を縦1ピクセル×横8ピクセルの領域に分割して、各領域毎に高周波成分の強さを算出します。

入力: input[0]~input[7]

出力: output

$$\begin{aligned}w1 &= \text{input}[0] - \text{input}[1] + \text{input}[2] - \text{input}[3] \\ &\quad + \text{input}[4] - \text{input}[5] + \text{input}[6] - \text{input}[7] \\ w2 &= \text{input}[0] - \text{input}[1] + \text{input}[2] - \text{input}[3] \\ &\quad - \text{input}[4] + \text{input}[5] - \text{input}[6] + \text{input}[7] \\ \text{output} &= \min(\text{abs}(w1 / 4), \text{abs}(w2 / 4))\end{aligned}$$

これを「ごちゃごちゃ度」と呼んでいます。具体的には、隣り合う画素の値を交互に足し引きすることで計算しています。w1とw2はちょっとだけ符号の付け方が異なっていて、絶対値の小さい方を採用することで計算の精度を向上させています。

この処理の出力が図2です。outputの値の大きさを明るさで示しています。

3-2. 垂直方向の差分をとる

前項で得られたごちゃごちゃ度の画像の各画素について、垂直方向の差分の絶対値を計算します。

入力: input(x,y), input(x,y+1)

出力: output(x,y)

$$\text{output}(x,y) = \text{abs}(\text{input}(x,y) - \text{input}(x,y+1))$$

これによって、垂直方向に変化の少ない縦方向のエッジ(リング上のポールや背景)を除去し、縦にも横にもごちゃごちゃしている敵機を浮かび上がらせます。

この処理の出力が図3です。



図1. マシン搭載カメラのサンプル画像

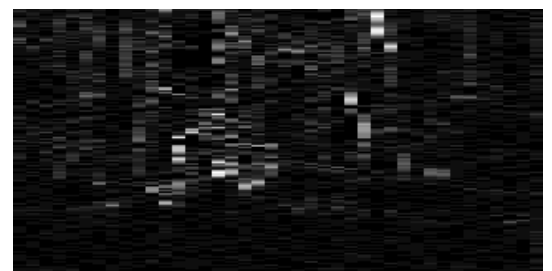


図2. 水平方向の高周波成分を抽出した結果

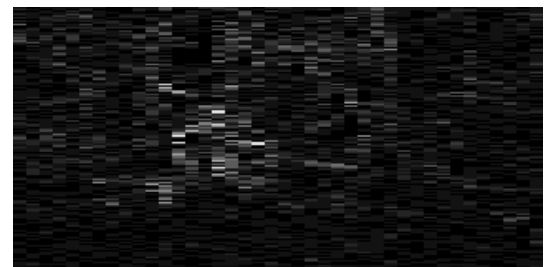


図3. 垂直方向の差分をとった結果

3-3. 垂直方向に膨張させる

前項の出力では、敵機が見えている領域の内部に値の大きい画素と小さい画素(ギャップ)が混在しているため、かたまりとして抽出できません。そこでギャップを埋めるために縦方向に膨張させます。

入力: $input(x,y), input(x,y+1)$

出力: $output(x,y)$

$output(x,y) = \max(input(x,y), input(x,y+1))$

この計算を5回実行することで5ピクセル以内のギャップを埋めることができます。

3-4. 垂直方向に収縮させる

ここまでの処理では、白線や照明の反射のように縦方向に輝度の変化が大きいものがポツポツと残ってしまう傾向があります。それらはある程度の高さを持つ敵機とは異なり、縦方向の広がり(大きさ)が小さいという特徴を持ちます。それを除去するために縦方向に収縮させます。

入力: $input(x,y), input(x,y+1)$

出力: $output(x,y)$

$output(x,y) = \min(input(x,y), input(x,y+1))$

この計算を10回実行します。前項の処理で5回膨張しているため、10回の収縮によって5ピクセル以内の高さに収まるゴミが除去されることになります。

この処理の出力が図4です。

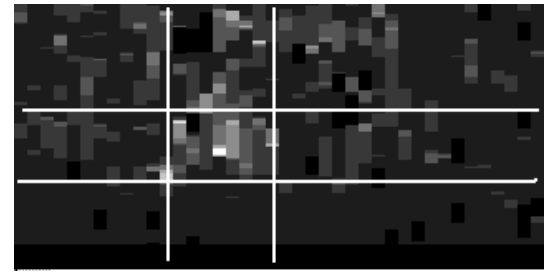


図4. 垂直方向の膨張と収縮を実行し、敵機位置を推定した結果

3-5. 移動平均が最大になる領域を探索

ここまでの処理で敵機の存在確率のようなものを求めたので、ここから敵機の位置を推定します。

まず各画素の値を垂直方向に合計します。

入力: $input(x,y), 0 \leq y < h$

出力: $sum_y(x)$

$sum_y(x) = input(x,0) + \dots + input(x,h-1)$

次に水平方向の幅 we の移動平均が最大になる位置 xe を求めます。

入力: $sum_y(x), 0 \leq x < w$

出力: xe

$value(x) = sum_y(x) + sum_y(x+1) + \dots + sum_y(x+we-1)$

の値が最大となる $x = xe$ を見つける。

次に画像から $xe \leq x < xe+we$ の領域を切り出して、水平方向に合計します。

入力: $input(x,y), xe \leq x < xe+we$

出力: $sum_x(y)$

$sum_x(y) = input(xe,y) + \dots + input(xe+we-1,y)$

最後に垂直方向の高さ he の移動平均が最大になる位置 ye を求めます。

入力: $sum_x(y), 0 \leq y < h$

出力: ye

$value(y) = sum_x(y) + sum_x(y+1) + \dots + sum_x(y+he-1)$

の値が最大となる $y = ye$ を見つける。

以上の結果 xe, ye から、画像中での敵機の位置が決まります。

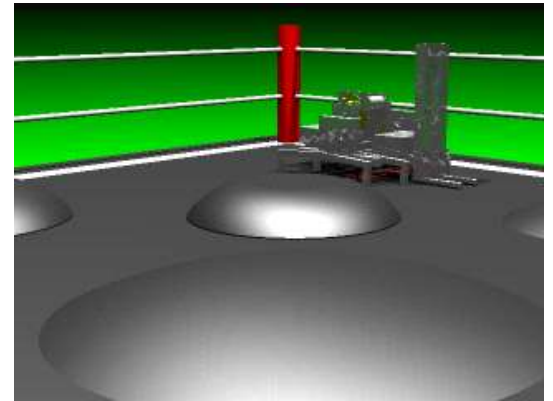


図5. POV-Rayを使用したシミュレーション画像

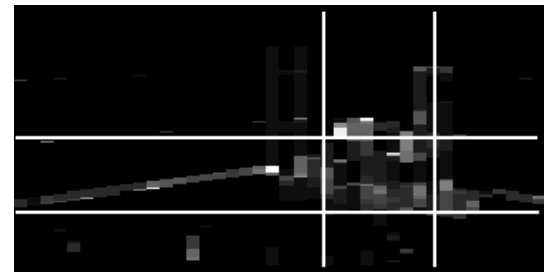


図6. シミュレーション画像(図5)から敵機位置を推定した結果

3-6. 逆透視変換

画像中での敵機位置から実際の敵機の方角と位置を求めます。x座標から方向を計算し、y座標から距離を求めます。原理的には距離を求めることは不可能なので、ここでは検出された座標がリング上の高さ0の点であるという仮定を使って距離を計算します。(式は省略)

y座標の値は敵機の高さや視覚的な特徴によって上下に変動しやすいので、距離の精度はあまり高くありません。一方x座標から求まる敵機の方角の精度は、敵機の高さがある程度知れているという前提があるため、距離の精度に比べたら高いと言えます。

4. まとめ

かわさきマシンはごちゃごちゃしている、という仮定がこのアルゴリズムの前提になっています。実際にこのアルゴリズムで敵機(あるいは何か別のもの)をトラッキングしている様子を見たことがある方も多いと思いますが、条件が良ければわりと安定して敵機を捕捉することができます。この手法は画像の輝度のパターンとその空間的広がりが手がかりとして用いるため、照明の明るさや敵機のカラーリングなどの影響を受けにくいという特徴があります。カメラのフラッシュなどの外乱を受けて大きくずれることもありますが、検出結果の履歴に対してフィルタを適用することで軽減できます。

5. あれこれ

この報告でざっくり省かせて頂いたことを最後に紹介しておきます。

実は、リング上の白線をハフ変換で抽出してリング内での自機の位置を推定するプログラムも開発しましたが、カメラの解像度の低さや取り付け角度の不安定さにより実用的な精度が出ないことがわかったので実機では使用しませんでした。

また、POV-Rayというレイトレーシングソフトを使ってカメラの画像をシミュレーションしてソフトのテストに使用しました(図5、図6)。照明の反射が再現できています。カメラの取り付け位置の検討にも役立ちました(ちょっと記憶が不鮮明ですが)。

このアルゴリズムを開発したのは3年も前ですが、今回初めて文書化しました。何かの役に立てば幸いです。ちょっと工夫すればかわさき以外にも応用できるかもしれません。