サイコロの目の画像認識

山田 佳奈　(指導教員：粕川 正充)

1　はじめに

近年多くのカメラが身近に存在している。カメラから得られた映像を単に記録するだけではなく、映像に対して画像処理を施すことによって、文字や物体の認識、芸術性や視覚効果の付加など、より高度な作業が可能になる。身近にある高度な画像処理を行うアプリは、例えば、Google 翻訳、Zaim、SNOW などがあり、どれも非常に興味深い。そんな中、電動ダイスカップというサイコロを自動で振る機械の存在を知った。透明のドーム状の入れ物にサイコロが複数個入っていて、ボタンを押すとそれらのサイコロを同時に振ることができるものである。これの目の数を認識できないかと思い、本研究に取り組んだ。

2　研究準備

2.1　研究環境

以下の環境で研究を行った。

- CPU：AMD ryzen7 1700 3.4GHz
- Memory DDR4：2666MHz, 16GB
- OS：Ubuntu16.04LTS
- GPU：NVIDIA Geforce1080ti(Memory：11GB)
- Software：CUDA 10.0,python3.5.2,OpenCV3.0

2.2　画像データセットの作成

データセットに使用するサイコロの画像は、実際にサイコロ２個を振った数秒間の動画を iPhone で撮影し、それを取り込んだ。取り込んだ動画データは mov 形式だったため、ffmpeg を用いて mp4 形式に変換し、フレームごとに静止画として保存した。静止画は 121 枚になったので、完全にサイコロが止まっている状態から 10 フレーム前までを実験対象とした。これらの変換、静止画保存にも OpenCV と python を用いてプログラミングをした。

最初は真上からの撮影したが、照明の反射でサイコロの目が読み取りづらくなってしまう場合があったため、斜め上から撮影するようにした。また、背景を考慮せずにそのまま撮影した場合、二値化した際にうまくサイコロの上面だけを切り出すことができなかったので、背景を黒いものにして撮影を行うようにした。

3　研究内容

まずはサイコロを含む範囲で画像をトリミングした。撮影位置がほぼ同じなので、緑の部分を含んだ真ん中の辺りで自分で座標を設定した。トリミングした画像の中から 1 枚選択し、カラー画像 (図 1) とグレースケールと二値化画像 (図 2) を取得した。カラー画像の場合は一旦、グレースケール画像に変換してから二値化処理を行う場合と、一定のルールに基づいて一気に二値化を行う場合があるが、今回は OpenCV を用いたので、一旦グレースケール変換を行ったあとに二値化を行うという前者の方法となる。二値化をする目的としては、対象物の形状を知りたい場合などに、二値化フィルターによる輪郭強調が利用できるが、背景との境界がハッキリすることで、ラベリングや画像検出処理を行う上で処理しやすい画像を生成することができるからである。

次にサイコロの上面２面の重心を求め、その点を中心にさらにトリミングした。この際の座標設定も自分で与えている。二値化した画像をこの座標でトリミングする時に射影変換も行い、歪んだ形が正方形に近づくように整えた。今回は背景を黒にして撮影しており、サイコロの白い部分を関心のある特徴として取り出してしまうので、サイコロの目を取り出すために、これの白黒反転した画像を dst.png として保存した (図 3)。また、この時にサイコロの上面以外の余白がないように気をつけて座標を設定した。

図 1: カラー

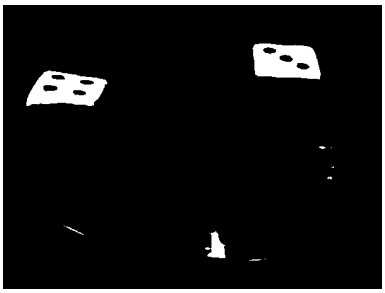

図 2: 二値化

図 3: dst.png

次に行う操作についてだが、OpenCV3.0 より、画像処理の中でも非常に重要な処理のひとつである、ラベリングが利用できるようになったのでこれを利用した。ラベリングとは、二値化した画像内の連続した点の集まりごとに、個別の番号を割り当てる処理のことで、ラベリングにより、画像内にいくつのオブジェクトが存在しているのか、そしてそのオブジェクトがど

のような特徴を持ち、画像内のどこに存在しているのかを知ることができる。ラベリング処理を施し、それぞれのサイコロの外接矩形の左上座標、重心の座標、面積を求めた (図 4)。

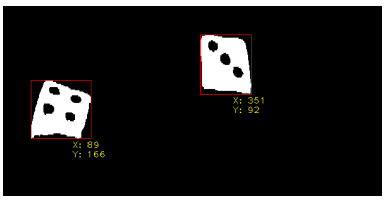

図 4: 左上の角の座標

それぞれのサイコロを射影変換し、同様にラベリング処理を用いて目の数を認識した (図 5)(図 6)。

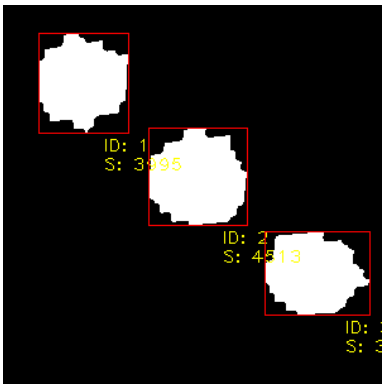

図 5: サイコロの目 (dice1)

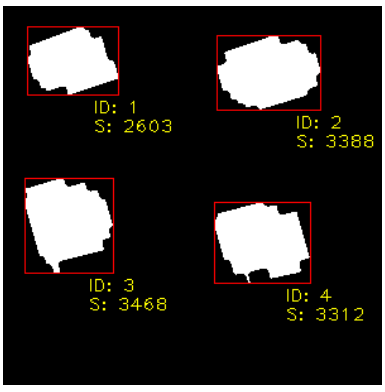

図 6: サイコロの目 (dice2)

以上のような操作で調べてみると、サイコロが完全に止まっている状態から 4 フレーム前まではサイコロの目を両方とも認識できたが、それより前のフレームの画像だとぶれてしまっていて、片方しか認識できなかったり、両方とも認識できなかったりした。今回のフレームレートは 22fps だったため、約 1.8 秒前の画像までは画像認識できたことが分かった。

4 今後の課題

トリミングする際の座標設定を今回は全て自分で調べて設定した。ここの操作を自動化できるようになればより多くのデータを高速に処理できるだろう。また、ぶれた画像は認識することができなかったので、フィルターをかけるなどの工夫も試みる。またそもそも今回は iPhone で撮影した際のデータ形式が MOV 形式で、フレームレートが 22fps だったので、フレームレートが 60fps くらいの別素材を使いたいと考えている。またラベリング処理を行った際に求めた面積や重心座標を活用して、座標設定やサイコロの目の認識に役立てるようにプログラムを改善したいと考えている。今回は 1 つの動画に対しての処理しか行えなかったが、複数の動画を 1 度に処理できるようにするなどの改善点もあげられる。また、今回はラベリング処理を行うということで OpenCV3.0 を使ったが、新しくリリースされた OpenCV4.0 には G-API というものが実装されており、こちらは画像処理をグラフでとらえてノードとノードをつなげて処理しようというものだそうだが、実験的 API ではあるが今後 OpenCV4.0 を使ってプログラムを改善することも考えられる。また今回は使わなかった手法であるが、マッチング手法を用いて画像認識を行うことも考えている。マッチングの手法には、大まかに「領域ベースマッチング」と「特徴ベースマッチング」があるが、特徴ベースマッチングが対象画像への回転や変形といった影響を受けづらいマッチング手法であるようなので、プログラムの改善に役立てたい。

参考文献

[1] ダイナミックに踊り出す電動サイコロ「Boogie Dice」, https://bouncy.news/4612

[2] 【Python × OpenCV】歩行者検知でやってること, http://nobutobook.blogspot.com/2016/11/python-opencv.html

[3] 機械学習でサイコロの目をカウントするモデルを作成する, http://kyudy.hatenablog.com/entry/2018/04/17/225201

[4] Tensorflow はじめました 実践! 最新 Google マシンラーニング, 株式会社 R & D, 2016, ISBN9784802090889 , 有山圭二

[5] 画像処理入門講座 : OpenCV と Python で始める画像処理 , https://postd.cc/image-processing-101/

[6] 詳解 OpenCV 3 ―コンピュータビジョンライブラリを使った画像処理・認識 , Gary Bradski,Adrian Kaehler(著), 松田晃一, 小沼千絵, 永田雅人, 花形理 (訳)

[7] 実践ＯｐｅｎＣＶ ２．４ ｆｏｒ Ｐｙｔｈｏｎ 映像処理&解析 /カットシステム/桑井博之

[8] Python+OpenCV でラベリング, http://okkah.hateblo.jp/entry/2018/08/02/163045

[9] OpenCV による画像処理入門 改訂第 2 版 , 小枝正直／上田悦子／中村恭之・著