# RaspberryPi を用いたミカン果実に対する画像認識システムの構築

三重大学工学部工学研究科技術部
吉田俊一
yoshida@eng.mie-u.ac.jp

## 1. はじめに

私が活動を支援している研究室において、ミカンなどの果実を収穫するロボットハンドの開発を企図することとなった。（図１） その過程において強く要望されたのが収穫したミカンのヘタを短く切断する機構の開発であった。みかんを箱に入れて運搬する際にヘタが長く飛び出していると相互に傷つけてしまい、商品としての価値を失ってしまう。そこでカメラからの映像をもとにヘタの位置を認識・推定し、モータなどを用いた機構により姿勢を変化させてハサミを用いて適正な長さにヘタを切断する機構を開発することになった。

しかしながら研究室において画像処理は初めての挑戦であり、ノウハウのない分野であった。技術職員として、学生に指導する必要性及び画像認識技術はかねてより興味がある分野であったので、開発環境を整え実際に USB カメラを用いた姿勢角推定システムを構築した。

本報告では、まだ開発途上ではあるものの、その内容について報告する。

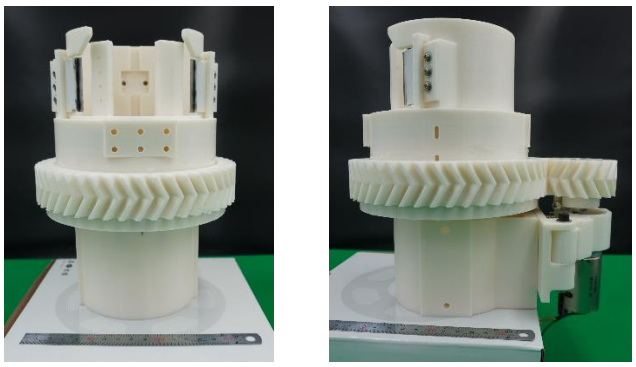

図１　開発中のみかん収穫ハンド

## 2. 画像処理ライブラリ　OpenCV

筆者が過去にカメラ関連技術に触れたのは、15 年以上昔に大学衛星プロジェクトを支援していた際、カメラモジュールと FPGA を接続し映像データを取得したいというニーズがあったときである。当時は技術的なハードルが高く、大学内での開発を断念して外部企業へ協力を依頼することとなった。その際の経験により画像処理は大変な技術であると印象付けられてきた。

この度改めて調べると、現在では当時に比べて遥かに技術的障壁が下がり、取り扱いが容易になっていた。その中心にあったのが画像処理ライブラリの OpenCV である。

OpenCV は PC に接続されたカメラやファイルの取り扱いや画像への数学的処理や機械学習に用いることができる関数群が整備されており、C++や Python 等の開発言語で用いることができる。

OpenCV は画像や映像の扱いは極めて抽象化されており、従来 C++で行うように各画素データや、メモリの確保、大きさなどは意識せずに取り扱うことができることに驚いた。

今回は後述の RaspberryPi 上に Python の開発環境を構築し、そこに OpenCV をインストールして実験を行った。

## 3. ハードウェア

今回は将来的にロボットへの組み込みなどを考えて、ハードウェアとして RaspberryPi を使用することとした。RaspberryPi は 1.5GHz 駆動の CPU を搭載して、OS としてカスタマイズした Linux を採用し

たミニ PC であって、Python の開発環境を容易に構築できる。OS はマイクロ SD カードにインストールされ、今回は純正の Raspberry Pi OS を使用した。

またデジタル入出力端子（GPIO）を備え、モータ駆動用 IC モジュールやデジタル通信型センサーを接続可能という特徴がある。カメラによる特徴抽出は PC であっても可能であるが将来的にはモータやセンサを接続したいことから GPIO 利用を念頭に RaspberryPi を選択した。

今回は USB カメラの解像度を 1920x1080 ピクセルに設定して使用する。この環境でカメラで撮影した映像に対して各種の画像処理を適用してみた。

開発環境として Python3 をインストールした。Python は機械学習等に用いられることが多いインタプリタ言語である。そのため、コンパイラ言語である C++に比べると実行速度に劣るが、デバッグの容易さが評価されており、現在は主流の開発言語になっている。Python 及び OpenCV はインターネットに接続した状態で Git コマンドを用いてインストール可能である。

将来想定しているのは図 2 のようなシステムであり、センサによってミカンが置かれたことを検知したらカメラによって姿勢を推定、それに基づいてローラを回転させてヘタを上向きにし、ハサミで余分な箇所を切断する、というものである。。

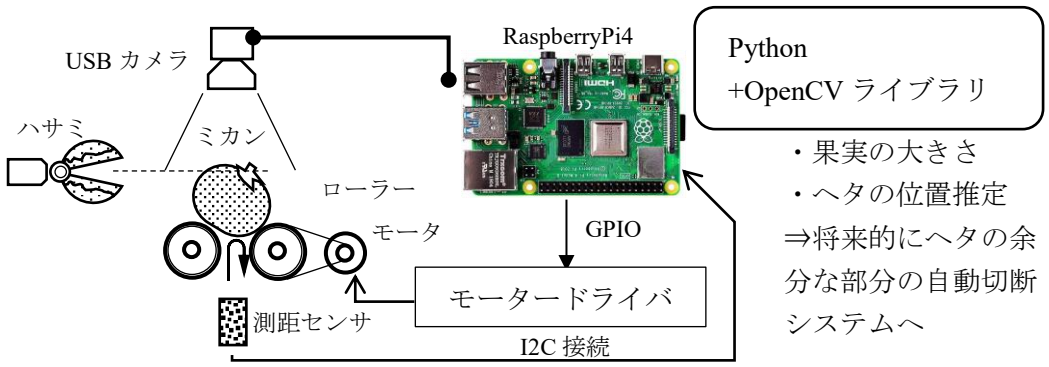


図 2　想定しているミカン果実の剪定システム

## 4. ミカンを対象とした画像認識

### 4.1　ミカン果実の姿勢推定手法

ミカンを収穫したあとに、一つづつローラーのついた台の上に載せて回転させて任意の姿勢をとらせる機構を考える。その際にカメラ画像からの情報でミカン果実の姿勢を推定する必要がある。

そのためには(1) ミカン果実の大きさ　(2)ヘタの位置を検出　の２つが必要であると考えた。

画像中にあるミカンを円として認識し、検出した円の半径 r と中心座標を求める。（図 3）

次にヘタの位置を検出する必要がある。方法として、色の違いによる方法を考えたが果実の成熟度合いによって色が異なるため、誤検出の要因になると考えた。今回はテンプレート画像を用意して合致度を見るテンプレートマッチングを採用した。これにより、画像中のヘタがある座標が判明し、それにより検出した円の中心座標とのズレ量（dx,dy）を求めることができる。

半径 r とズレ量（dx,dy）から姿勢角 θ を求める。

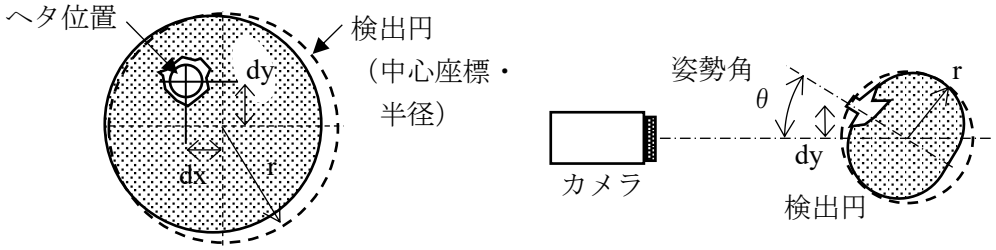


図 3　ミカン姿勢の推定

### 4.2　ミカンの寸法測定

画像認識には cv2.HoughCircles 関数を用いる。これは画像のなかにある円形状を検出するものである。パラメータが不適切であると極めて多数の円を検出してしまう。なかには全く円ではない形状を検出してしまうこともあるので、検出する円の直径を制限、検出パラメータの調整及び、映像中心エリア以外の検出円は破棄する、などの処置を行った。cv2.HoughCircles 関数で、円を検出するために直接関係するパラメータは３つあり、(a)検出器の解像度　(b)エッジ検出しきい値の上限、下限とされている。これらのパラメータを調整して、ミカンを安定して円として検出できる値を探した。

今回の試みでは、まずミカンの大きさを測定することを目的とし、そして精度がどの程度得られるのかを検討することとした。しかしミカンの大きさを変えて揃えることが困難であったため、測定対象として大きさを変えた円を紙に印刷して使用した。

cv2.HoughCircles 関数でパラメータが不適切である事例を図 4 に示す。ミカンの画像ではないが、全く円が無い箇所にも検出されている様子がわかる。

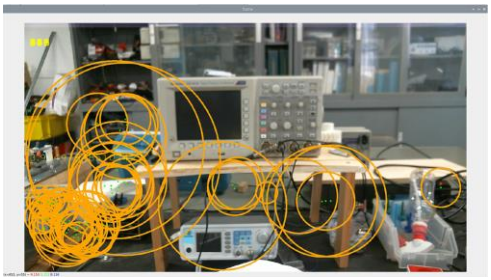

図 4　cv2.HoughCircles 関数での検出例

今回はこのような事例を防ぐため、パラメータの調整及び、検出する円の直径を制限、映像中心エリア以外の検出円は破棄する、などの処置を行った。

サンプルとした円の直径は 42mm,52mm,63mm,73mm,84mm である。それぞれ 30 回ほど検出された円の半径を記録して、それを実際の寸法に換算して集計した。その結果としては全体として実際の円よりも大きめに検出される。この原因については判然としなかったが、おそらくエッジ検出アルゴリズムによるものと考えられる。（図 5，図 6）

検出に必要な時間としては、およそ 0.9～1.0 秒であった。

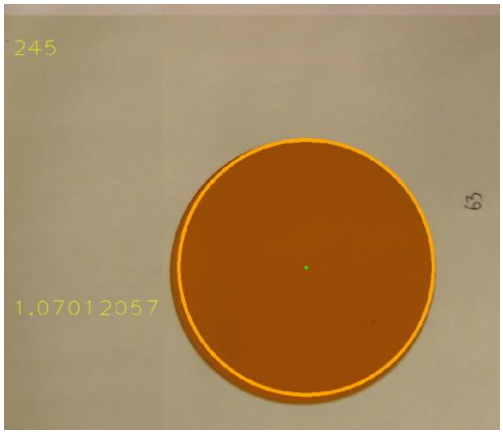


図 5　OpenCV により円を検出している様子

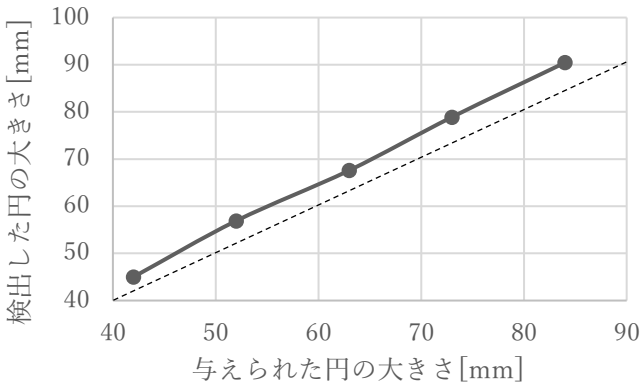


図 6　OpenCV により検出された円の大きさ

### 4.3　ミカンのヘタの位置推定

次に取得した映像上にミカンのヘタが存在する位置を求める必要がある。

そのためには cv2.matchTemplate 関数を使用する。この関数はテンプレートとなる画像が取得画像の上にあるのか、取得画像上の位置ごとの適合度を求める関数となっている。そしてその適合度が任意のしきい値を越えた場所にヘタがあるとした。(図 7,8)

当初はテンプレート画像に実際のミカンのヘタ部分を切り抜いたものを使っていた。しかし角度がずれたり、大きさが異なると適合度が大きく下がってしまい、良好な結果を得られなかった。

そこで実際の画像ではなく、手書きで図形を組み合わせたヘタに見える画像を用意したところ、認識率が向上した。その理由についてははっきりとは判っていないが、抽象的な画像のほうがより平均的であって、どのような映像に対してもより近い結果が得られているものと推測している。

処理速度については、ピクセル数が多いこともあって一連の処理におよそ 1.5 秒ほどかかってしまう。

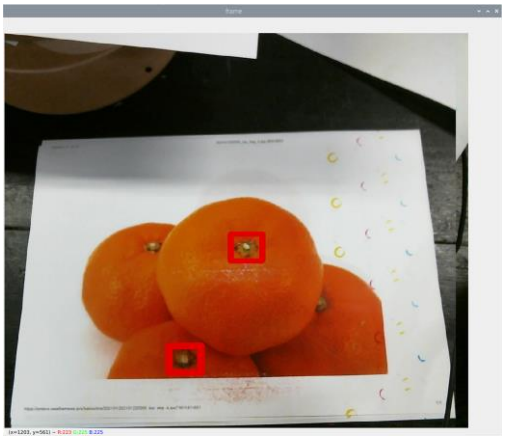

図 7　テンプレートマッチングによるヘタの検出

(a) 実際のミカンのヘタ画像

(b) 手描きのヘタ画像

図 8　テンプレート画像

### 4.4　ミカンの姿勢角検出

ミカンの寸法とヘタの位置を取得できたので、そこからミカンの姿勢角を求める。

ミカン果実の半径、および検出された円の中心と、ヘタの位置の差から果実姿勢を推定する。円検出とパターンマッチングのプログラムを組み合わせて作成した。

その結果を図に示す。

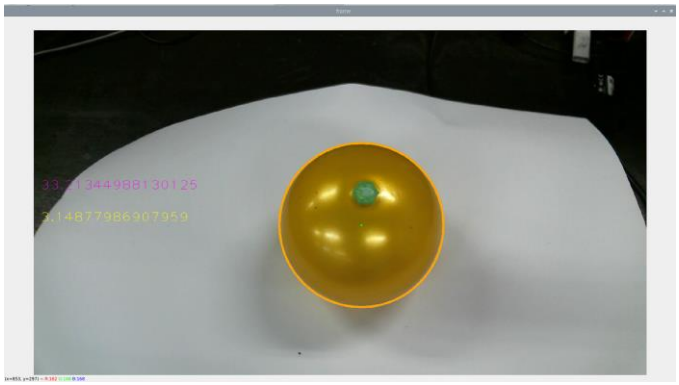

図 9　円検出とテンプレートマッチングを組み合わせた姿勢検出

この一連の処理におよそ 3.3 秒ほどかかる。実際にミカン果実の姿勢を整える機構を考えた場合には、2 回から 3 回ほどはやり直す必要が生じると考えており、処理が完了するまでにミカン果実一個に着き 20 秒ほどかかってしまうと考える。すると一日 8 時間稼働として 1400 個ほどの果実を処理することになるが、求められる処理速度には程遠い。

速度向上のためには、取得画像の解像度の見直しや、ハードウェアの変更などが必要になる。

## 5. まとめ

今回は未だ中途ではあるが、RaspberryPi と OpenCV を用いた画像認識を実施し、所定の成功を得た。画像認識を行うハードルは格段に下がっており、応用も容易であると感じた。しかしながら OpenCV ライブラリの中身や動作アルゴリズムまで理解しようとすると、資料が乏しく、学習の妨げになるとも感じる。（こうすると上手くいくが、その理由がわからない等）

また Python の処理速度向上が課題としてあげられる。Python はインタプリタ言語であるので、これを実行ファイル化することで高速化できる可能性がある。

今回はプログラム的に画像処理で終わってしまったが、次年度以降はこれにセンサーやステッピングモータを接続して、実際に姿勢を変更する機構を製作する予定である。