カメラ画像を用いた自己位置推定による自律走行ロボットの製作

著者　吉田　暉　　指導教員　斉藤　徹

1. はじめに

現在人工知能技術として進んでいるものとして自動運転技術がある。既に運転サポート、アシスト機能としての提供が始まっており、世界の自動車メーカーがしのぎを削って開発を進めている。こういった自動運転技術には画像処理技術が不可欠である。最近は画像処理が可能な小型コンピュータが安価に利用できることから、学生が小規模な自動走行の実験ができる環境を構築することが出来る。本研究は画像処理実験環境の構築及び画像処理について理解を深めるため、カメラ情報を基に自己位置推定を行い、自律走行するロボットを製作することを目標とする。今回は自己位置推定プログラムの製作を行い、動作検証を行った。

2.　基本事項

2.1 研究内容

本研究では縦 2m、横 2m のマーカーが描かれたコースを、指定されたスタートとゴールの間を自己位置推定しながら走行するロボットの製作を目標とした。車体は 20cm×20cm の中に収まるような大きさを想定し、左右 2 つに直流モータとコース位置を把握するためのカメラ、画像処理を行う小型コンピュータを搭載する。これ以外の光センサーや接触センサーは持たないものとする。

システムにはカメラを用いた視野画像及び、予め用意した地図画像が入力される。画像の詳細は 3.1 にて説明する。また地図画像上での初期値の座標及び角度も与えられる。入力された 2 つの画像を基に自己位置推定を行い、車体があると思われる座標と車体の方向を出力する。ここで出力される座標は地図画像上での座標である。その座標を基にゴールまでの距離及び角度を算出し、ロボットを走行させる。開発には様々な画像処理の機能がある OpenCV を使い、学生が自己位置推定などの処理を簡単に実験できる環境を提供することを目指すこととした。このため当初は RaspberryPi による車体構築を目指したが、小型コンピュータの性能を考慮し、デスクトップパソコンで検証を行った。本研究ではロボット上で自己位置推定を行うことが目標であるが、ロボットの製作及びモータ制御に時間がかかると考えた。そこで今回はデスクトップパソコン上に簡単な開発環境を作成し USB カメラを用いて自己位置推定を制御することを目標とした。

2.2　RaspberryPi

RaspberryPi は大きさが約 85mm×53mm の小型コンピュータである。一般的な小型コンピュータは OS を搭載することができないが RaspberryPi は Raspbian と呼ばれる専用の OS を搭載することができる。また他の小型コンピュータとくらべて性能が高いため、複雑な処理を行わせることも可能であり、カメラモジュール等も販売されている。本研究で行う自己位置推定は複雑な処理であり、カメラを用いるため、RaspberryPi が最適あると判断し採用した。

2.3　OpenCV

OpenCV はオープンソースのコンピュータビジョンライブラリである。このライブラリは C,C++ で書かれているが、それ以外にも Python、Ruby、MATLAB 等でも用意されている。OpenCV は計算効率を優先し、リアルタイムアプリケーションに重点をおいて設計されている。ライブラリには工場の製品検査、画像診断、セキュリティ、ユーザインタフェース、カメラキャリブレーション、ステレオビジョン、ロボット工学を含む、多くの領域にわたる 500 以上の関数が用意されている。

3 自己位置推定

3.1 使用する画像

カメラからリアルタイムで撮影される視野画像は高さ 16cm、俯角 45° の 640pixel×480pixel の画像を使用する。次にコース全体の地図画像は白いテーブル上の 2m×2m の範囲を高さ 2.5m から撮影し、大きさが約 1000pixel×1000pixel の画像を使用する。しかし、視野画像は透視変換を行うので、実際に使われる画像の大きさは 350pixel×300pixel 程度になる。地図画像には、簡単なテンプレートマッチングで自己位置推定ができるように、色がついたアルファベットを複数敷き詰めることとし、車体の視野画像に常に 1 つ以上のアル

ファベットが写るような間隔とした。

3.2　処理の流れ

まず視野画像及び地図画像は前述した物が与えられる。次に視野画像からアルファベットが写った部分のみを切り出し、地図画像から初期位置周辺の画像を切り出す。切り出しが終わったら地図画像と視野画像を用いてテンプレートマッチングを行う。この時、視野画像は数度ずつ回転させながらマッチングを行うことで車体の方向を推定する。マッチングした結果には最も類似度が高い座標及び、角度が出力される。今回は実験結果が正しいか判断するため、地図画像上に描画させる。

3.3　画像の回転

入力された画像を回転させるにはアフィン変換を用いる。しかし、そのまま回転させると画像の一部が見切れたり、黒色に塗りつぶされる部分が発生してしまう。テンプレートマッチングにおいて、黒色に塗りつぶされた部分もソース画像としてマッチングを行うため、期待した通りの結果が得られない可能性がある。そこで、予め用意した背景画像を使用し、背景画像の中心に回転させた力画像を置くことで、黒色に塗りつぶされる場所が発生せず､かつ画像が見切れることがなくなる。背景画像は一定の大きさではなく、入力された画像にあわせて大きさが変動するようにしている。また背景画像の色は白いテーブルと同じ色になるように調整している。

3.4　尺度の調整

テンプレートマッチングを行うにあたって、視野画像と地図画像の尺度が同じになるように視野画像を縮小させる必要がある。ほぼ同じ大きさになった状態で視野画像を回転させると地図画像よりも大きくなり、プログラムは正しく動作しない。そこで地図画像からアルファベットを切り出した際に、周辺が少し写るように大きく切り出している。こうすることで、視野画像が地図画像よりも大きくなることはないので、エラーを回避することができると考えた。

4　実行結果

地図画像を図 1、視野画像を図 2 として実験を行った。実際にマッチングに使用した画像を図 3、マッチングを行った結果を図 4 に示す。

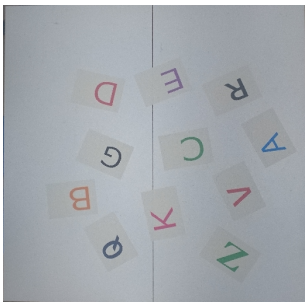

図 1　地図画像

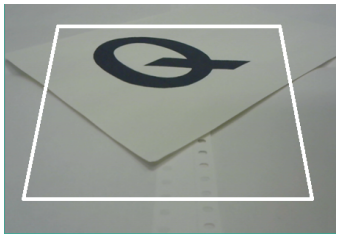

図 2　カメラ画像

図 3　透視変換後

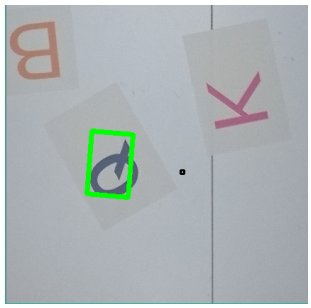

図 4　実行結果

図 2 の白色の四角形は透視変換をする際に使用した画像の領域を表している。図 2 と図 3 を比べると真上から見た画像に近くなっていることがわかる。また図 4 で描画されている四角形は図 3 の画像と同じ大きさで描画されている。図 4 を見ると同じ｢Q｣の場所に四角形が描画できている。また、四角形の右側の黒色の小さい点は車体の位置を予測しており、実際にあった場所とほぼ一致している。しかし、角度が若干ずれているので。今後は角度の精度を上げる必要がある。

5　今後の課題

今回は 1 つのマーカーが完全に写っている状態の時を想定して実験を行ったが、撮影する場所によっては一部しか映らない物が多くなるため、画像が一部しかなくても正確に自己位置推定ができるようにする必要がある。また、実験結果で述べた通り､角度がまだ正確に検出できていないため、角度の精度を上げる方法を考える必要がある。今回は処理速度の関係上、初期角度の±45°を 1°ずつ回転させている。しかし、この時の 1 回の処理にかかる時間は約 3 秒であり、小型コンピュータで動作させるには致命的な遅さである。初期角度の±5°を回転させると 1 回あたり約 0.5 秒とぎりぎり運用できる段階となる。今後は角度の精度を向上させるとともに、処理速度を大きく上げる方法を考える必要がある。