修士論文

ライントレースカーに用いた 2 種類の白線追従ユニットの動作検証

Performance evaluation of two different types of Line detectors for Lane-Keeping model

報　告　者

学籍番号：1265044

氏名：笹川 純一

指　導　教　員

星野　孝総　准教授

令和 6 年 2 月 19 日

高知工科大学大学院工学研究科

基盤工学専攻電子・光システム工学コース

目次

第 1 章　序論

1.1　研究背景

近年、人工知能 (AI) の進化は驚異的であり、宇宙、医療、芸術、教育、産業など、その応用範囲はますます広がっている [1–5]。中でも組み込みシステムにおいては、制約のある環境での効率的な処理やリアルタイム性が求められており、マイクロコントローラ (マイコン) を用いて AI を組み込むことによる研究が注目を集めている [6, 7]。

従来の組み込みシステムは、特定の任務に焦点を当てたプログラムや制御アルゴリズムを実行するものであった。しかし、AI の導入により、システムは環境の変化に柔軟に対応し、学習や意思決定の能力を向上させることが期待される。

Internet of Things(IoT) の時代が到来しつつある中で、ネットワークを介して集まる情報を処理する人工知能 (AI) 技術、特に深層学習 (deep learning) の発展はなくてはならない存在であると言える [8]。中でも、従来の IoT における AI 活用は、端末機器がクラウド・サーバと通信を行う形が主流であった。端末機器からの情報をもとに、クラウド・サーバ内部の深層学習の結果として得られた AI モデルが推論を行い、結果のみを受け取って返す仕組みとなっている。この方式の欠点として、リアルタイム性の確保が難しく、ネットワークへの常時接続が不可欠であることが挙げられる。

このような課題に対処するため、新たなアプローチとして注目を集めているのがエッジコンピューティング技術である。AI モデルを組み込んだ端末機器で処理を完結させることで、リアルタイム性の向上やネットワークへの依存の軽減が可能である。

しかし、その一方で通常のマイコンであれば、メモリなどの計算資源が限られているため、そのまま深層学習を利用した AI モデルを実装しても十分なパフォーマンスを得られることはないだろう。AI に要求される処理のうち、非常に計算コストが高い学習と比べると推論のコストはそこまで高いものではない。また、専用の AI アクセラレータを備えたマイコンであれば、推論の処理を十分に行い、かつ、その推論情報をもとにしながら別の処理を行うことが可能だ。

本研究では組込みシステムに組み込む AI モデルとして畳み込みニューラルネットワーク (CNN : Convolutional Neural Network) を扱うことにした。CNN は深層学習の画像処理分野において様々なタスクに特化した手法が研究されており、物体認識、物体検出、画像生成など多岐にわたる。また、エッジデバイスでの運用を考えられた軽量なモデルがあることからも、CNN モデルを組込みシステムに組み込むことができると考えた。

西山による先行研究では類似性ファジィ推論を用いた白線追従制御器を開発し、コースの走行実験を行ってきた [9]。このシステムはフォトセンサを用いて白線の情報を読み取り、マイコン内で演算を行うものであった。今回、コースの情報をセンサではなくカメラを用いることによって深層学習モデルを組み込んだシステムを構築することにした。カメラはその性質上、単純なフォトセンサよりも処理が遅くなることは容易に考えられる。しかし、だからこそ画像処理や深層学習、機械学習への理解が深められ、深層学習モデルを組み込んだライントレースカーを試してみることは十分に意義のあることであると考えられる。

西山のライントレースカーを再現し、比較の対象とする。その際、アルゴリズムを多少変更して走

行挙動の変化について調べた。フォトセンサを用いた白線の検出は、白線検出の一点に絞れば高速であり最適であると考えられる。それに対してカメラでの白線検出は、白線以外にも多くのものが映像に現れ、白線を抽出するだけでも多くの処理を必要とする。それでも、ライントレースに関する新たな知見を得られることを期待し、カメラを使用した走行がどれだけフォトセンサを使用した走行に迫れるかを実現しようとするものである。

カメラによる白線検出の方法として 2 種類を選択し、1 つは機械学習、もう 1 つは白線のベクトル抽出を組み込んだ。どちらが優れているのか? それぞれの長所と短所に関しても実験を通して考えていきたい。

1.2 目的

本研究の目的はカメラを用いた白線追従走行器 (ライントレースカー) を設計、製作しコースを走行するシステムを開発することである。その過程として、カメラから得られる情報として画像とベクトルの 2 種類を別々の方法で取得し、それぞれのライントレースカーの回路の設計と製作を行い、コースの走行を目指す。

画像データから走行のための制御量を計算するメインチップとして Maix bit を使用した。Maix bit は MicroPython によるプログラミングが可能で、ESP32 のような開発ボードとほとんどサイズが同じであることから電子工作においては扱いやすいものである。特に深層学習モデルを高速で扱えるアクセラレータを搭載していることから、ライントレースの処理を AI にさせることができるのではないかと考え、実装していくことにした。

ベクトル情報を使用する方法としてはメインのチップには ESP32 を使用し、カメラモジュールに Pixy2 を使用している。このライントレースカーに関しては回路の設計、製作だけではなく、ボディパーツを 3 D プリンタで出力することでキットを使わないオリジナルなものの開発を目標とした。また、西山のフォトセンサを用いたライントレースカーを今回制作したライントレースカーに実装して、両者の比較をより直接的なものにした。

これらを実現するためには扱うマイコンへの理解や深層学習の手段や方法について深く知る必要があり、その技術を身に付けることも目的の一つである。

1.3 研究の概要

本研究では、ライントレースカーにおける情報取得をカメラで行い、画像データからモータ制御を行う方法と、ベクトル情報から制御を行う 2 つの方法を試した。本稿の構成は以下の通りである。第 2 章では本研究のベースとなる先行研究について述べる。第 3 章では本研究で使用したマイコンやモジュールについて使用方法や特徴を述べる。第 4 章では深層学習モデルを用いたライントレースカーの制作および実験結果について述べ、第 5 章ではラインのベクトル情報を用いたライントレースカーの設計と製作および実験結果について述べる。最後に第 6 章にまとめを述べ、今後の課題と展望について述べる。

第 2 章　先行研究

本研究の内容紹介前に、ベースとなる所属研究室の先行研究として、西山のライントレースカー [9] について述べる。

西山はライン類似度ファジィ推論を用いたライントレースカーを開発し、その動作検証を行った。このライントレースカーはラインの情報を 8 個のフォトセンサで取得し、その情報をもとに演算を行うものである。ファジイ推論は、数値化が難しい性質のものや具体的な数値との結びつけが難しい問題に対して有効であり、ルールに熟練者の知識を組み込んで反映させることができる特徴をもつ。しかし、通常のファジイ制御では、制御量の調整にメンバーシップ関数の形状変更やファジィルールの調整が必要である [10, 11]。これには熟練者が実際の物理現象に近いメンバーシップ関数を調整する必要があるが、最適なメンバーシップ関数の設計には膨大な検証とチューニングが必要となる問題があった。この問題に対処するため西山はメンバーシップ関数に類似度演算を適用する類似度ファジィ推論法を使用し、ライントレースカーの制御を行った。

走行させるライントレースカーの外観を図 2.1 に示す。

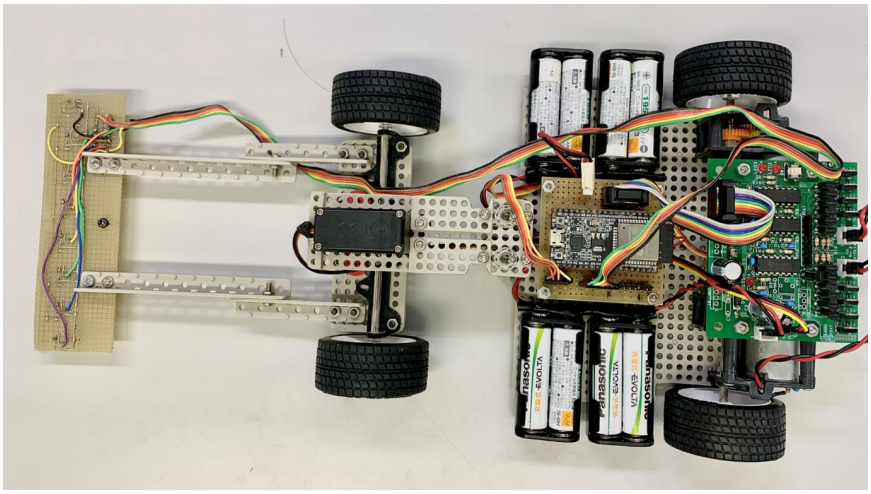

図 2.1　類似度ファジィ推論を用いたライントレースカーの外観

従来のファジィ推論法ではメンバーシップ関数を用いて適応度 μ を導出し、μ を重みとした加重平均より制御量を決定していた。西山は適応度の導出において Overlap coefficient を改良した類似度計算を用いた。Overlap coefficient を用いたときの適応度 μ の導出の式を式 2.1 に示す。

$$\mu_i = \frac{\sum_{a=1}^{8} \min(SR_{ia}, S_a)}{\min(\sum_{a=1}^{8} SR_{ia}, \sum_{a=1}^{8} S_{ia})} \tag{2.1}$$

センサ入力を S、センサルールを SR とし、類似度演算の推論結果を μ とする。Overlap coefficient では | A ∩ B | を分子に用いるが、センサ値の場合は数値であるため集合ではなく min 演算を用いて計算している。i はルールの数であり本実験では 5 ルールを使用する。a はセンサの数を表しているため、1〜8 までの合計値で計算する。

次に式 2.2 に示すように各ルール i に対する推論結果 μ_i を重みとする操舵角ルール AR_i の加重平均より操舵角制御量 SP を決定する。

$$SP = \frac{\sum_{i=1}^{5} \mu_i \cdot AR_i}{\sum_{i=1}^{5} \mu_i} \tag{2.2}$$

左右モータ制御量の決定については、式 2.2 における操舵角ルール AR_i をそれぞれ右モータルール RMR_i、左モータルール LMR_i に置き換えて計算することによって導出する。また、AR_i、RMR_i、LMR_i の設定のために差分進化法 (DE:Differential Evolution) によって最適化を行った。

次にシステム構成について述べる。フォトセンサを使用し、類似度ファジィ推論を用いたライントレースカーのシステム構成を図 2.2 に示す。

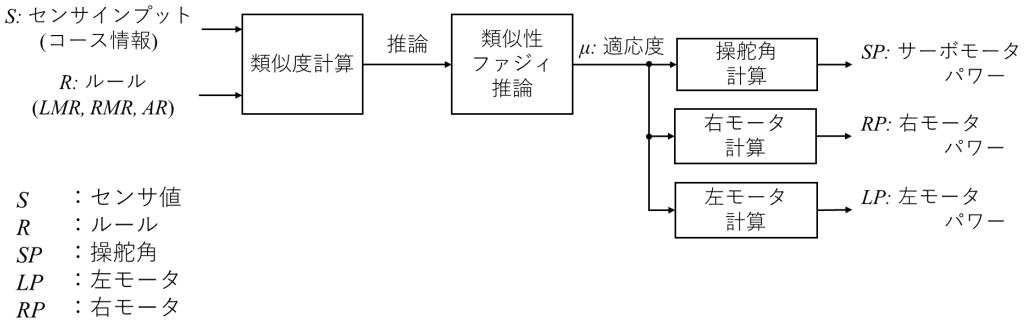

図 2.2　類似度ファジィ推論を用いたライントレースカーのシステム構成

類似度ファジィ推論を用いたライントレースカーは 8 個のアナログ入力のフォトリフレクタによってコース情報を取得し、過去の走行情報から作成したルールとの類似度を導出することで推論を行う。推論結果と操舵角、右モータ、左モータそれぞれのルールをもとにして式 2.2 の計算を行うことでそれぞれの制御量を決定する。メインの CPU には ESP32 を使用し、計算された値は逐一 WiFi で PC に送信してデータを取得できるようになっている。

次にコースについて述べる。

コースは図 2.3 に示すような Japan Micom Car Rally 規定のコースを用いた。このコースは直線 600 mm × 4 枚、カーブ半径 R450 mm × 16 枚を用いた全長 10 m、幅 300 mm のコースとなっている。コース中央に幅 20 mm の白いセンターラインがあり、その両端に 10 mm の灰色ライン、またコースの両端にも 10 mm の白線が施されている。このコースを 2 周走行させて走行時間、走行速度、サンプリング数、処理速度について比較を行った。

走行ルールを 5 ルールとし、推論方法を変更したときの走行データを表 2.1 に示す。

推論方法として、Overlap coefficient(min)、Overlap coefficient の分母を変更したもの (max)、jaccard、dice、簡略化ファジィ推論の 5 つ選択した。max、jaccard、dice について、推論方法の算出方法は以下に示す。

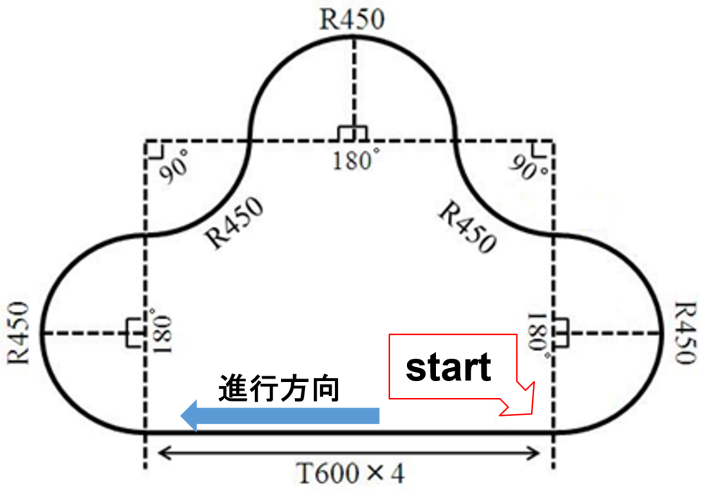

図 2.3　走行コース

表 2.1　2 周走行結果

	min	max	jaccard	dice	Fuzzy
走行時間 (s/2lap)	15.42	14.54	15.07	15.42	15.71
走行速度 (m/s)	1.29	1.38	1.33	1.29	1.27
サンプリング数	22540	21261	22026	22517	22591
処理速度 (us)	684	684	684	685	695

Overlap coeffient(分母変更)

Overlap coefficient 演算は式 2.1 である。この演算において、センサ入力 S は白線を検出した場合 8 つのセンサのうち隣り合う 2 つが最大値となる 1 に近い値を示し、残りが 0 となる。また、センサルール SR は 8 つのセンサのうち隣り合う 2 つの値を最大値として設定しているため、センサ入力 S とセンサルール SR を比べると 8 つのパラメータのうち隣り合うどこか 2 つの値を参照する。また、Overlap coefficient 演算の分母は最小値をとるためセンサ値を参照するが、分母を最大値に変更するとセンサが反応した部分のルールの値 (1) を参照する。

Overlap coefficient 演算分母を変更したもの (max) を 2.3 に示す。

$$\mu_i = \frac{\sum_{a=1}^{8} \min(SR_{ia}, S_a)}{\max(\sum_{a=1}^{8} SR_{ia}, \sum_{a=1}^{8} S_{ia})} \tag{2.3}$$

この演算手法での適応度 μ を Overlap Coefficient 演算での適応度と比較すると、分母に max を使用することで適応度がわずかに低くなる。

Jaccard

Jaccard 演算は 2 つの集合に含まれている要素のうち共通要素が占める割合を表しており、分子にセンサ入力 S とセンサルール SR の共通要素を示し、分子にセンサ入力 S とセンサルール SR の和集合をとることで式 2.4 となる。

$$\mu_i = \frac{\sum_{a=1}^{8} \min(SR_{ia}, S_a)}{\sum_{a=1}^{8} SR_{ia} + \sum_{a=1}^{8} S_{ia} - \sum_{a=1}^{8} \min(SR_{ia}, S_a)} \tag{2.4}$$

Dice

Dice 演算は Jaccard 演算の分母を 2 つの要素の和集合から 2 つの要素数の平均をとることで計算できる。よって、Dice 演算は 2 つの集合の平均要素数と共通要素数の割合を表しており、式 2.5 として計算できる。

$$\mu_i = \frac{2 \cdot \sum_{a=1}^{8} \min(SR_{ia}, S_a)}{\sum_{a=1}^{8} SR_{ia} + \sum_{a=1}^{8} S_{ia}} \tag{2.5}$$

以上の計算方法を用いてフォトリフレクタをセンサとしたライントレースカーが、1 周あたりにどの程度のラップを刻むのかを改めてプログラムして書き込んで測定してカメラ走行に対する基準とした。表 2.1 は西山の手法と推論方法を変更した手法でコースを走行させたときの結果を示しており、西山の手法は min の列に相当する。推論方法を変更することによってわずかな変化はあったが、極端な違いはなく、1 周約 7 秒〜8 秒での走行していた。

走行速度が最も速いものは Overlap coefficient 演算の分母を変更したものであった。これは適応度の計算において分母に max を用いると分母に min を用いたときに比べてわずかに適応度が大きくなる。そのため出力値がわずかに大きくなるため走行速度が速くなったと考えられる。

走行データを図 2.4 に示す。

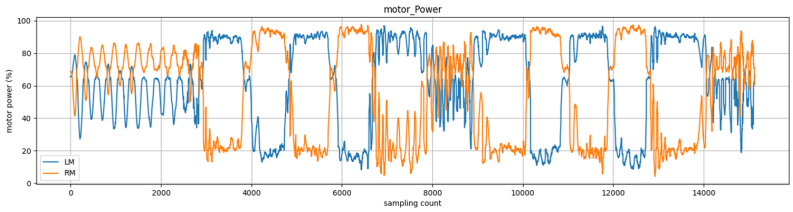

図 2.4　Overlap coefficient(max) のモータ出力

出力を見ると、はじめの直線部分において細かな揺れが発生していることが分かる。これは、直線部において少しでも斜めに進行すると、隣り合うセンサが何度も反応し、それぞれが逆方向に向こうとするために発生してしまうものである。この揺れは目視でも確認できるものであり、フォトセンサユニットを取り付けているヘッド部分が左右に振れながら走行していた。

本研究では、フォトセンサを用いたときの走行データを以上のものとし、カメラを使用した時の走行がどうなるのかを次章以降で調べていく。

第 3 章　開発環境

3.1　Maix bit

3.1.1　Maix bit の特徴

Maix bit は RISC-V アーキテクチャの CPU と深層学習アクセラレータ KPU を内蔵するワンチップ IC ”K210” を搭載したマイコンボードである。[8] 特徴として約 11 FPS と高速な物体認識性能と、平均 180 mA の低消費電力性能を両立している。信号を ON/OFF する GPIO や、PWM 信号出力、シリアルインターフェース機能を持つ I/O 端子を備えているため、カメラを利用したラインントレースカーのメインの制御用マイコンとして利用した。Maix bit の外観を図 3.1 に示す。

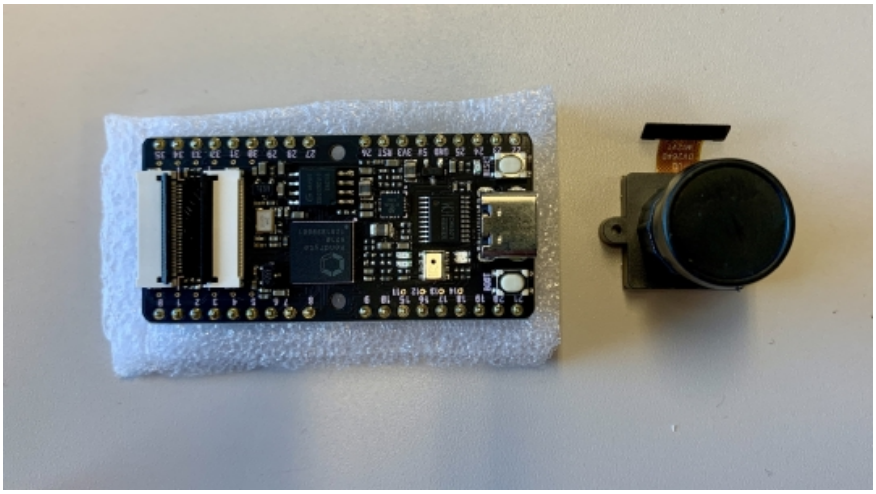

図 3.1　Maix bit の外観

サイズは 52.3 × 25.4 mm と小型であり、ブレッドボードに挿せる形状であることから拡張性が高く、電子工作にも向いているマイコンボードであると言える。また、図 3.1 右にあるカメラユニットと映像確認用のディスプレイユニットも同梱されており、画像処理や機械学習を使ったシステムを組み込みやすくなっている。

搭載されている Kendryte K210 の最大の特徴は Knowledge Processing Unit(KPU) という、深層学習のためのアクセラレータを持つことである [12]。これによって、畳み込みニューラルネットワークなどのディープラーニングのアルゴリズムを高速に実行することができる。なお、KPU で畳み込みニューラルネットワークを演算するためには、事前に学習データを用意しておく必要がある。

Maix bit でのプログラミングを行うためには専用の統合開発環境の MaixPy Ide を使用し、MicroPython での記述を行う。

3.1.2　畳み込みニューラルネットワークについて

畳み込みニューラルネットワーク (CNN : Convolutional Neural Network) は近年研究が熱心に行われている深層学習手法の一つで、主に画像認識分野で優れた性能を発揮している。2 次元データである画像は横の並びだけではなく縦の並びが重要であり、これを分類するためには 1 次元に並べるだ

けでは分類するのは難しいものとなっている。畳み込みニューラルネットワークのアルゴリズムはニューラルネットワークが 2 次元の画像の形状を保持したまま処理できるよう拡張されたものであり、その中心となる仕組みは畳み込み層とプーリング層からなる特徴抽出にある。

畳み込み層は、畳み込みフィルタを用いて手前にある層の特徴抽出を行う層であり、画像に対する処理としては画像より小さいサイズのフィルタをスライドさせながら掛け合わせることによって行われる。

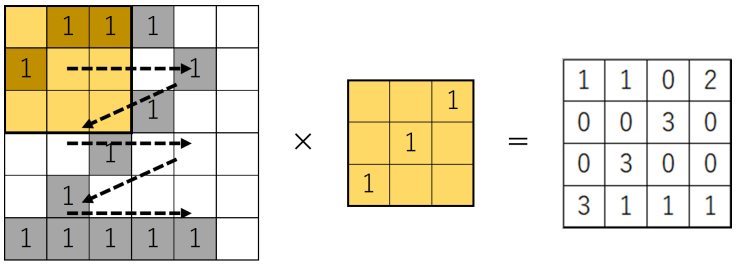

図 3.2　畳み込み処理

CNN における畳み込み処理の例を図 3.2 に示す。画像左に入力画像、真ん中の黄色の配列が特徴を抽出するためのフィルタ、右が畳み込み処理を行った後の値となる。畳み込みの処理は以下のようになる。

入力画像として 6 × 6 の白黒画像として数字の 2 の画像を入力する。次にフィルタを入力画像の左上の部分に合わせ、重なっている 9 個のセル各々に対して、それらのセルの数値を掛け算する。そして、この 9 個の掛け算の結果を合計する。この例では画像とフィルタの両方が 1 となっている部分が 1 つしかないため合計値は 1 となる。次にこのフィルタを 1 セル分右にずらし同様の処理を行う。フィルタを順にずらしていき、右まで移動したら 1 セル分下げて右にずらしていく処理を繰り返す。

以上の畳み込みの処理によって得られた行列は、元の画像の各領域部分にどの程度フィルタと類似している画像が存在していたかを表している。

CNN では畳み込み層の後にプーリング層と呼ばれる領域を圧縮する処理を行う層を加える。

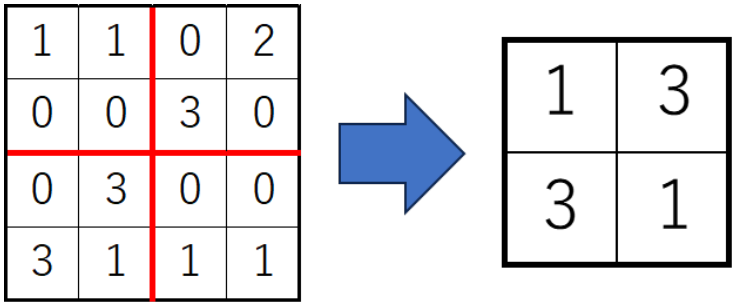

図 3.3　プーリング層の処理

図 3.3 にプーリング層の処理の例を示す。この例では 4 × 4 の畳み込まれた行列を赤線のように 2 × 2 の領域の行列で区切り、最大値プーリングでは各領域の中の最大値をその領域の値とする。

CNN の場合、学習の対象はフィルタとなる。フィルタを使って畳み込みの処理を行った結果は入力画像をそのフィルタで特徴づけたデータになっている。そしてこのデータを通常のニューラルネッ

トワークに渡し、教師データとの誤差から誤差逆伝播法を用いて、ニューラルネットワークのパラメータが更新され、それと同時にフィルタ自体も更新される。

3.1.3 Maix bit での CNN モデルの使用方法

次に、Maix bit で CNN モデルを組み込む際の使用方法を述べる。Maix bit に CNN モデルを組み込むには専用の学習モデル変換ツール「NNCase」使用する必要があるが、文献 [12] にある方法ではうまくいかなかったため、動作した方法を記述する。文献 [12] の方法では、kmodel フォーマットに変換できないか、変換できても Kendryte K210 にロードすることができず利用できなかった。試行錯誤の末、以下に示す方法でうまく学習データを利用可能となった。

初めに CNN モデルを作成する。モデルの作成には TensorFlow と Keras を使用する。TensorFlow は、Google が開発しているディープラーニングや機械学習を開発するためのソフトウェアライブラリで、Keras は TensorFlow 上で動作するディープラーニングを使いやすくするライブラリである。これらを利用することで比較的短いソースコードでニューラルネットワークを構築することができる。TensorFlow v2.12.0、Keras v2.12.0 を使用し、学習データは手書き数字のデータセットの MNIST を使用して 0〜9 までの数字の分類モデルを作成した。

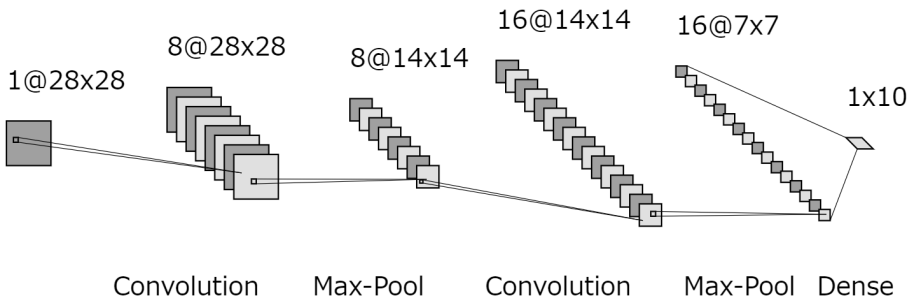

図 3.4　手書き数字のモデル

図 3.4 のように文献 [12] の CNN モデルを作成した。このモデルは畳み込み層 (Convolution) とプーリング層 (Max-Pool) を 2 回かけた構造となっている。モデルの最後には全結合層 (Dense) をかけており、抽出した特徴から 0〜9 までの数字それぞれの確率を出力する。

TensorFlow では組み込み機器や IoT 端末などでモデルを動作させるために、TensorFlow Lite というコンパクトなディープラーニングのモデル形式を持つ。TensorFlow Lite 向けに作成した学習データは、Kendryte が提供する NNCase を使用することで、Kendryte K210 向けの学習モデルのフォーマット kmodel へ変換する。

モデルを変換するプログラムをプログラム 3.1 に示す。

プログラム 3.1　モデル変換のプログラム

```
1 #TensorFlow -> TensorFlowLite 形式に変換
2 converter = tf.lite.TFLiteConverter.from_keras_model(model)
3 tflite_model = converter.convert()
4 tflite = os.path.join(model_dir, 'model.tflite')
5 open(tflite, 'wb').write(tflite_model)
```

```
6
7  #TensorFlowLite->kmodel 形式に変換
8  import subprocess
9  kmodelv2 = os.path.join(model_dir, model_dir_name + '.kmodel')
10 subprocess.run(['./ncc_win_x86_64_v2_beta4/ncc.exe', 'compile', tflite, kmodelv2, '-i
       ', 'tflite', '-o', 'kmodel', '--dataset', img_path])
```

TensorFlow Lite コンバータで、Keras の学習モデルから、TensorFlow Lite の学習モデルに変換する。TensorFlow Lite の学習モデルの拡張子は"tflite" である。次に NNCase で TensorFlow Lite の学習モデルを KPU の学習モデルに変換する。KPU の学習データの拡張子は"kmodel" である。

NNCase は Python への組み込み関数ではなく外部のプログラムなので、外部のプログラムを呼び出す subprocess で起動させる必要がある。

3.2 ESP32

ESP32 は中国の Espressif Systems 社が開発した 32 ビットマイコンで、チップ上に Wi-Fi 通信モジュールが搭載されていることから、電子工作だけでなく IoT 端末用のマイコンとしても使用される [13]。本研究ではベクトル情報を用いたラントレースカーの制御用マイコンとして採用した。ESP32 の外観を図 3.5 に示す。

図 3.5　ESP32 の外観

ESP32-devkitC は開発環境として ArduinoIDE や Micropython によるプログラミングが可能となっている。ArduinoIDE は C++ 風の言語での記述ができ、センサやアクチュエータを制御するライブラリや通信用ライブラリなどが豊富に提供されており、扱いやすい。ESP32 の動作電圧は 3.3 V であるが電池などの電源を利用する場合は 3.7 V から 6.0 V の入力が必要のため外部の電源を使用しない場合 3 本または 4 本の電池が必要となる。

CPU は Tensilica 社の XtensaLX6 によるデュアルコア構成で処理を行い、動作周波数は 240 MHz で動作する。プログラムメモリとして 480 kbyte の ROM を持ち、データメモリとして 520 kbyte の SRAM を持つ。

アナログ入力機能として、12 bit A/D コンバータを 2 チャンネル搭載しており、アナログ入力に対応したピンに割り振ることができる。12 bit もの高い分解能を持ちながら、サンプリング速度は 10 ns 程と高速で処理が可能である。

シリアル通信機能として、I^2C や SPI などに対応している。どちらも 2 チャンネルずつの接続が可能となっており、通信用のピンであればどのピンにもリマップ可能となっている。

無線通信機能として Wi-Fi 通信機能と Bluetooth Low Energy(BLE) を備えている。Wi-Fi 機能はステーションモード (STA モード) とアクセスポイントモード (AP モード) があり、STA モードは主に PC や他の機器への送信を行い、AP モードでは PC やスマホなどからアクセスする場合に使用する。

最大の特徴としては ESP デバイス間で無線通信が行える ESP-NOW である。ESP-NOW は、Espressuif 社が開発した通信仕様で、Espressif 社の Wi-Fi デバイス間でデータを送受信するための独自技術となっている [14]。インターネット経由ではなく直接通信ができること、他の通信方式に比べて非常に高速に送受信ができる。テキストやセンサーデータなどをやり取りするのに有効な手段である一方で、一度に通信できるデータ量は 250 バイトになっているため、画像や音声などのマルチメディア的な用途には適さない。また、通信は 1 対 1 のみとなっており、複数デバイスに同じデータを共有するような使い方にも向かない。本実験においては走行中の走行データを ESP-NOW によってリアルタイムに送信し、走行データを取得するようにしている。

この方法の最大のメリットはインターネット環境が不要であるため、無線通信に場所を選ばないことである。とはいえ、ライン走行のメインルーチンの中にデータ送信ルーチンを入れるのは処理能力の関係で難しく、それぞれ別スレッドを立てることで走行制御と通信を両立し、独立して動作することができる。

3.3 Pixy2

Pixy2 は高速イメージングセンサによってオブジェクトを検出し追尾できるモジュールとなっている。ラインを検出する機能が優れており、交差点や道路標識をも検出することが可能となっている。Maix bit と大きく違う点として Pixy2 はライン追跡アルゴリズムを 60 FPS で動作させることができることにある。図 3.6 に Pixy2 の外観を示す。

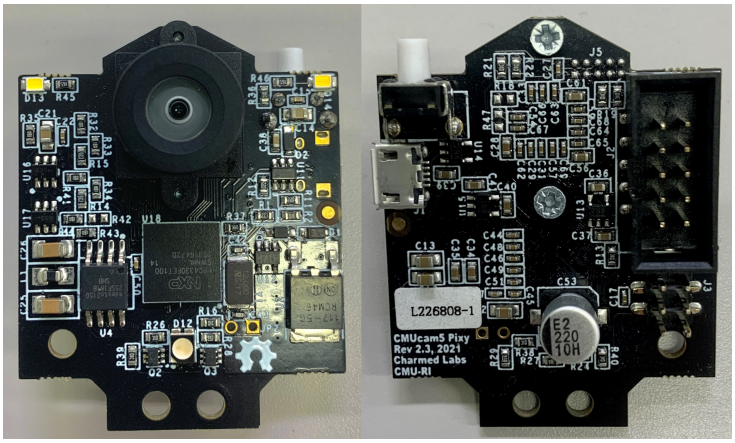

図 3.6　Pixy2 の外観

Pixy2 は前面にカメラを持ち、背面に USB コネクタ、ボタン、接続用コネクタを持つ。レンズ視野は水平 80 度、垂直 40 度で解像度は 1296 × 976 Pixel である。

通信方式として、SPI、I^2C、UART および USB と多様なインターフェースが用意されているため制御用のマイコンへのアクセスが容易である。本研究ではライン追跡を Pixy2 で行い、取得された情報を ESP32 へ通信してモータの制御をするようにした。

Pixy2 の設定と取得する映像の確認には PixyMon を使用する。PixyMon の使用方法としては Pixy2 とケーブルで接続してソフトを起動するだけで Pixy2 が撮影できている映像を確認することができる。

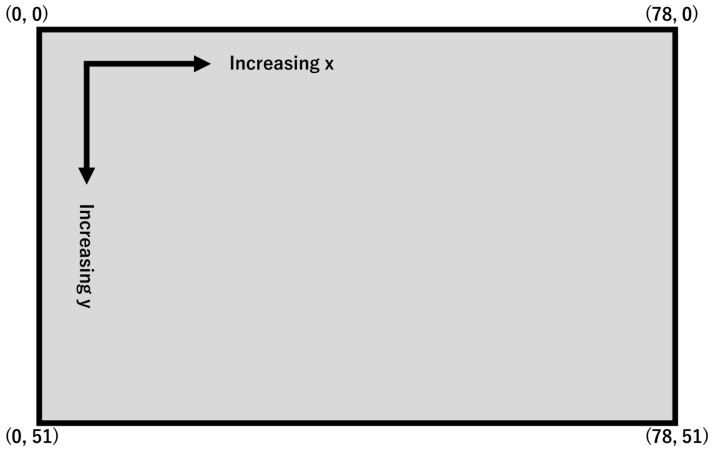

図 3.7　Pixy2 での描画空間

図 3.7 に Pixy2 の描画空間を示す。実際に撮影された映像は縦横の 2 次元空間に転写されており、画面左上を基準とし、横軸 (x 方向) に 78 pixel、縦軸 (y 方向) に 51 pixel の描画空間を持つ。

ESP32 とは SPI 通信によって通信を行い、Pixy2 からは取得されたベクトル情報 (x0,y0),(x1,y1) が送信され、ESP32 からはプログラムの実行時に Pixy2 の動作設定を送信することで Pixy2 のモードを切り替えることができる。このモード切替によって幅の認識程度や認識精度が変化する。

第 4 章　CNN を使用した白線追従制御器の制作

4.1　概要

カメラを使用した白線認識システムを構築し走行することを考えた。元となる市販のライントレースカーのメインとなる制御用マイコンとフォトセンサユニットを変更しファジィ推論による走行が可能か確認した後に、カメラを取り付けて走行中の画像と走行データを取得した。取得した画像を基に走行中のラインの状態を判別する CNN モデルを作り、取得した走行データに合わせて各パラメータを調整し、シミュレーションを行った後に実際のコースでの走行を行った。

4.2　CNN を行うための準備

4.2.1　開発用ボードの変更

白線認識部をカメラユニットに置き換える試みとして機械学習と画像処理を高速で行える Maix bit マイコンに変更した。Maix bit はカメラ、マイク、ディスプレイなどの機能を備えた小型の AI デバイスであり、Tensorflow で作成できるモデルを用いて機械学習アルゴリズムを実行することができる [12]。また、ESP32 マイコンのように I/O ピンが直接外部に出ている開発用ボードのため、モータとサーボへのアクセスと出力が容易であると考えられる。

4.2.2　フォトセンサユニットの変更

メインの CPU を Maix bit マイコンに変更し、走行時のコース状況を保存するためにまずフォトセンサを使って走行させた。ファジィ推論を使用した機体は AD コンバータを内蔵した ESP32 マイコンを使用し、マルチプレクサによる入力信号の切り替えによってセンサから情報を取得していた。しかし、Maix bit マイコンには AD コンバータが内蔵されていないため、ESP32 に搭載されているものと分解能が同じ 12 bit AD コンバータの MCP3208 をフォトセンサユニットに組み込んだ。このとき、センサ情報の取得には SPI 通信を使用した。

4.2.3　写真撮影用走行器

マイコンとフォトセンサを変更した走行器を図 4.1 に示す。

ボディとモータドライバには株式会社日立ドキュメントソリューションズ製のマイコンカー制作キットをベースとして、フォトセンサユニットとマイコンを変更した。Maix bit マイコンを用いたライントレースカーにおいてフォトセンサユニットでの走行を確認できたため、ボディ中間部にカメラを設置して走行中のコースのラインの撮影を始めた。この時、ESP32 では C 言語でプログラムを記述していたが、Maix bit では AI を処理するために MicroPython で記述している。フォトセンサを用いた走行においても MicroPython での記述を行っているが、極端に遅くなることはなく、無事にコースを完走することができた。

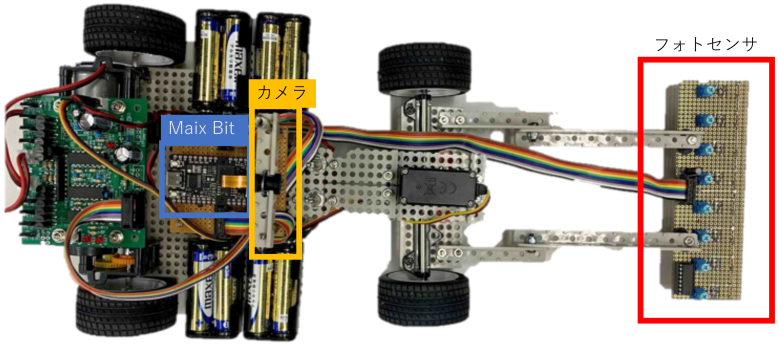

図 4.1　コース撮影用走行器の外観

4.3　Maix bit へ搭載する CNN モデルの設計

初めに、走行中のラインの状態を判別するためのモデルの作成をし、CNN による状態判別が可能か確認した。ラインの状態を判別するためには走行中のラインの画像と、その画像がどの状態なのかを始めに設定しなければならないため、最初に走行中の画像を取得するようにした。

画像の収集には Maix bit に取り付けられる OV2640 カメラモジュールを使用した。取得された画像は図 4.2 のような 320 x 240 ピクセルのカラー画像になる。

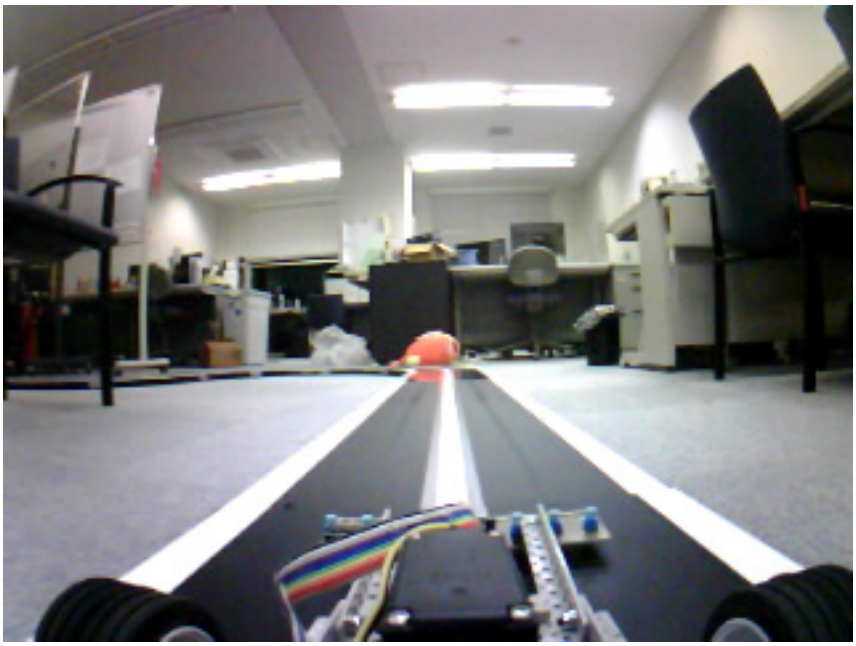

図 4.2　取得された画像

Maix bit マイコンに搭載できる容量には限界があり、走行中の推論時間も短くしたいため、学習を行う前処理として図 4.3 のような 40 x 40 ピクセルのグレースケール画像にして学習を行うようにした。

作成する CNN モデルは文献 [12] を参考に 2 層の畳み込み層とプーリング層からなる CNN モデ

図 4.3　前処理後

ルを構築し、直線、右カーブ、左カーブの 3 種類の状態を分類できるように学習を行った。また、このとき各画像に対する状態の設定は手動で確認して設定するようにした。

本実験では学習用画像として 3 種類合わせて合計 1200 枚撮影し、そのうち 100 枚ほどを検証用データとして使用した。学習結果を図 4.4 に示す。

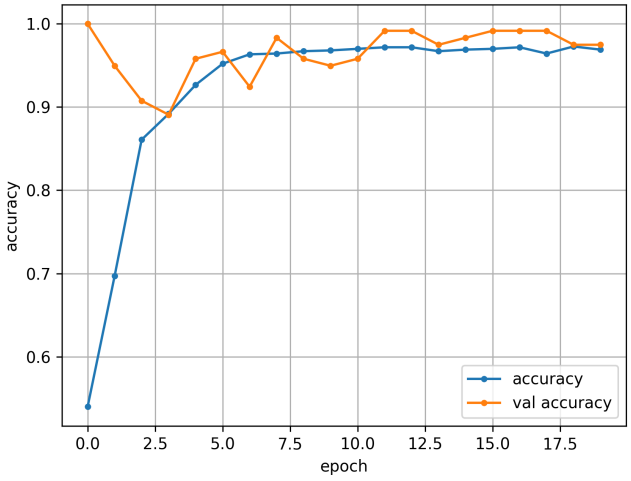

図 4.4　学習結果の正解率

accuracy は学習用データから求めた正解率を示し、val_accuracy は検証用データから求めた正解率を表している。横軸には学習回数を示し、縦軸に正解率を示す。20 回の学習回数で、学習用データと検証用データに対する精度はどちらも 90 % 以上となっていた。

以上の結果から、作成したモデルは走行中のラインの状態の判別が十分に可能と判断したため、

Maix bit にモデルを組み込み、今度は走行中のラインを 500 枚取得して推論を行った。推論結果を表 4.1 に示す。

表 4.1　走行中の画像の推論結果

	直線	左カーブ	右カーブ
取得枚数 (枚)	114	81	306
分類失敗 (枚)	2	4	4
正解率 (%)	0.98	0.95	0.99

直線クラス、左カーブクラス、右カーブクラスの 3 つのクラスの分類を行った。全体で 500 枚撮影し、そのうち各クラスに分類された画像の枚数を取得枚数に示し、取得枚数のうち目視で分類失敗と判断したものを分類失敗に示す。3 つのクラスの中で右カーブクラスに対する正解率が 99 % と最も高く、左カーブクラスに対する正解率が最も低い結果となった。

4.4　走行シミュレーション

操舵角制御量の算出には CNN の推論結果を用いた。CNN モデルの推論結果は走行中に撮影された画像を処理した後、現在のラインの状態を 3 つのクラス (直線、右カーブ、左カーブ) のいずれかに分類する予測を行う。これらの予測に基づいて機体のステアリングを適切に調節し、検出された直線に沿った軌道を維持するための操舵角制御量が導出される。作成したモデルの構造は下の図 4.5 のようにした。

直線、右カーブ、左カーブの 3 つのクラスを分類するモデルを作成した。2 つの畳み込み層とドロップアウト層からなるモデルを構築し、ドロップアウト層のパラメータは 0.5 に設定した。

制御量の決定は次のようにした。クラス i となる確率を P_i，操舵角ルールを SR_i とし、それぞれの重み付き平均から操舵角制御量 SP を式 4.1 のように重み付き平均によって導出する。

$$SP = \frac{\sum_{n=1}^{i}\{P_n \cdot SR_n\}}{\sum_{n=1}^{i}\{P_n\}} \tag{4.1}$$

左右モータの制御量 LMP と RMP についても同様に，クラス i となる確率を P_i を重みとする左右のモータルール LMR，RMR の重み付き平均から導出する．

$$LMP = \frac{\sum_{n=1}^{i}\{P_n \cdot LMR_n\}}{\sum_{n=1}^{i}\{P_n\}} \tag{4.2}$$

$$RMP = \frac{\sum_{n=1}^{i}\{P_n \cdot RMR_n\}}{\sum_{n=1}^{i}\{P_n\}} \tag{4.3}$$

走行挙動を検証するため、マイコンに実装する前に CNN の結果をもとにシミュレーションを行った。画像取得時の操舵角と左右サーボ制御値を記録し、推論結果から算出した制御量との対応関係をグラフで評価する。さらに、教師データの画像をシミュレーション用に変更した。その結果、カメラに走行時の白線以外に風景が映りこむことが分かった。

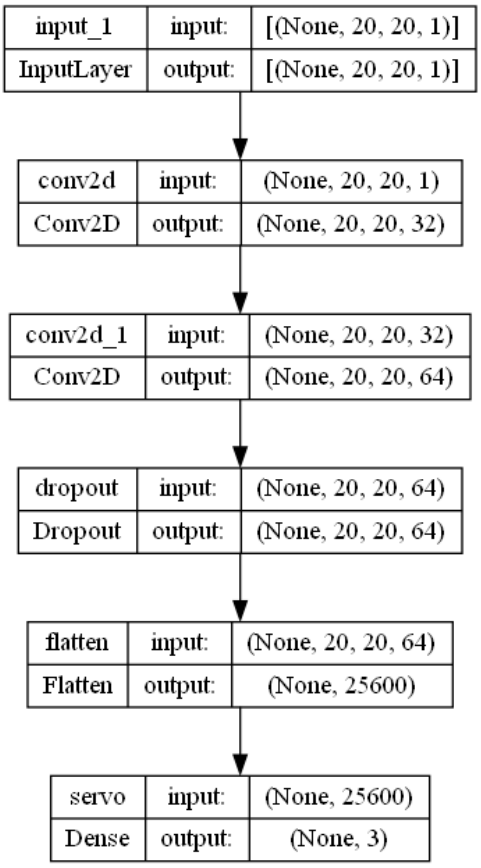

図 4.5　今回使用した CNN のモデル構造

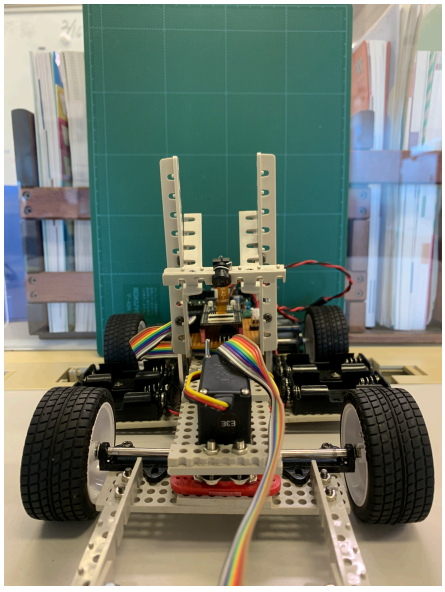

図 4.6　カメラ位置変更前

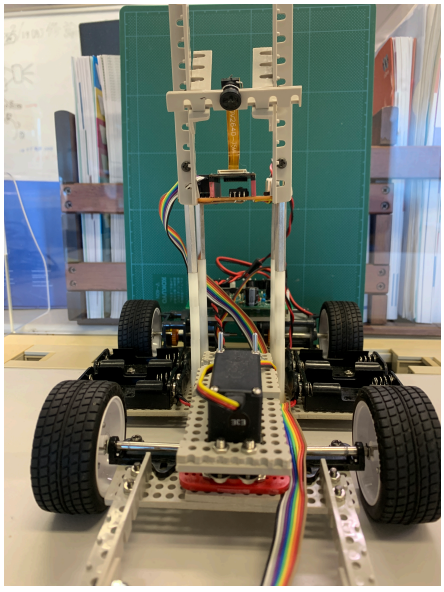

図 4.7　カメラ位置変更後

カメラの取り付け位置を高くし、図 4.6 から図 4.7 のように変更した。カメラユニットはケーブルの関係で基板近くにしか取り付けられないため基板下の支柱を 40 mm × 1 本から 40 mm × 3 本に変更した。画角を調整することで、図 4.2 で観察したような無関係な風景によるノイズを除去した図 4.8 の画像を得た。

図 4.8　カメラ位置を高くした時のコース画像

以上の画角の画像を教師用データとし、走行時のコースの画像を 8000 枚撮影し、そのうち 800 枚を検証用データとして学習を行った。このとき、コースの分類は左カーブ、直線、右カーブの 3 クラスとした。

次に、式 4.1 から式 4.3 におけるそれぞれのルールのパラメータの調整方法について説明する。それぞれの制御量に対して調整するルールは 3 つのため、9 つ分のルールを調整する必要があり、これらを人の手で 1 から調整するのは難しく、時間がかかると判断した。そのため、本実験では差分進化法 (DE : Defferential Evolution) によって各ルールのパラメータ調整を行った。

Differential Evolution(DE) は、1995 年に K.Price と R.Storn らによって提案された進化的アルゴリズムで、個体群が世代交代を繰り返しながら進化して最適解を探索していく [20] [21] 。DE の特徴としては制御パラメータの数が少なく設計が容易であることが挙げられる。

DE によるパラメータの最適化のために、まず Maix bit マイコンに CNN モデルを搭載し、走行中のラインの判別を行いながら各クラスの推論確率、操舵角度、左モータ制御量、右モータ制御量の走行データを取得する。ただし、走行データを取得するためには、白線に沿って走行させなければならない。このために、カメラとフォトセンサの両方を搭載した状態で、カメラで走行ラインを撮影して CNN での分類を実施するが、モータ制御はすべてフォトセンサからの出力によって行い、完全にコースを走行できている状態で画像データを取得した。走行中に動作した CNN モデルの推論結果の変化を図 4.9 に示す。

横軸にサンプル数、縦軸にサンプルごとの各クラスの推論確率を示しており、色はそれぞれ緑が直線、赤が左カーブ、青が右カーブを示している。図 4.9 を見ると、推論確率の変化は 3 クラスのうち 1 つが大きく反応する結果となった。走行コースに照らし合わせて確認すると、直線、右、左、右と

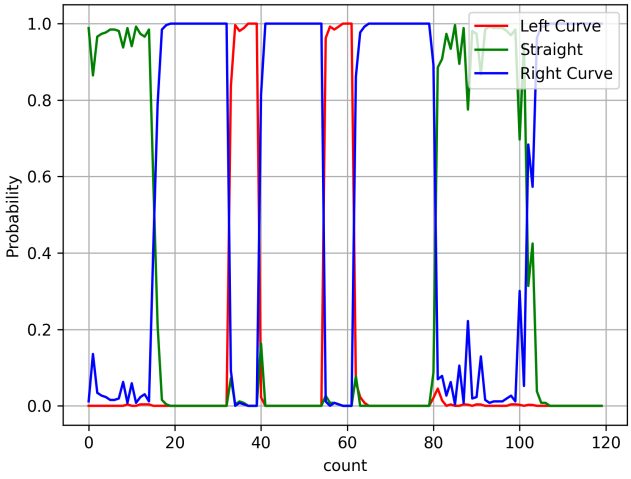

図 4.9　推論確率の推移

変化しており、正しくコースを認識できていることが分かった。

次にフォトセンサを使って走行させたときの走行時の操舵角とモータの出力の変化を図 4.10 と図 4.11 に示す。

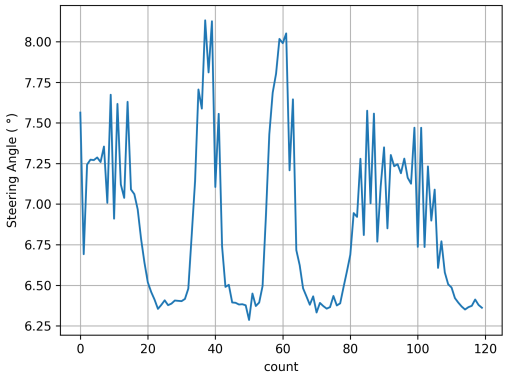

図 4.10　フォトセンサ走行の操舵角出力

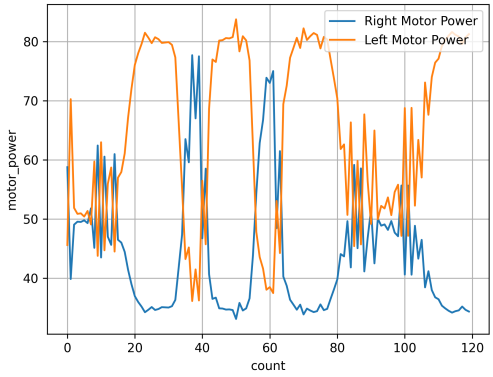

図 4.11　フォトセンサ出力の左右モーターパワー出力

以上の推測された確率を使用して DE によるパラメータの最適化を行った。最適化するパラメータは式 4.1〜4.3 の SR_n、LMR_n、RMR_n の 9 つのパラメータである。各サンプルごとに SP、LMP、RMP を計算し、走行データの誤差を二乗平均平方根誤差（RMSE:Root Mean Squared Error）により算出する。最終的に RMSE が最小となるパラメータが最適な走行パラメータとして出力される。最適化されたパラメータを使用した時のシミュレーション結果は図 4.12 のようになった。

青色のグラフに走行時の操舵角の制御量を示し、オレンジ色のグラフにシミュレーションによって導出した操舵角の制御量 (model_predict) を示している。図 4.10 と図 4.11 に対応した位置におけるカメラデータの CNN 分類をふまえて、制御量パラメータを決定し、フォトセンサでの走行時のサー

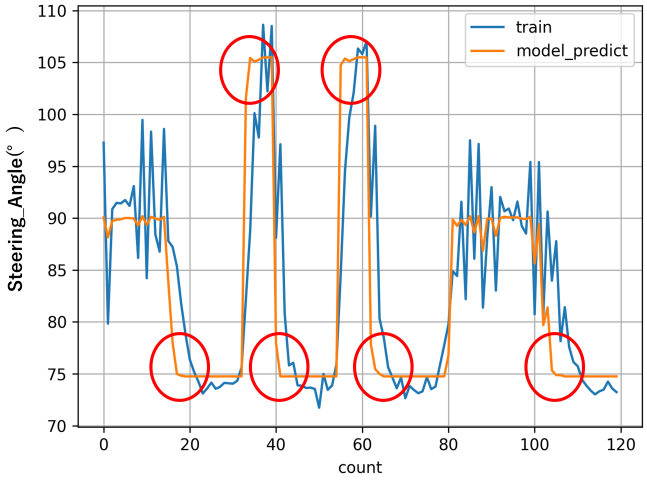

図 4.12　サーボ制御量のシミュレーション結果

ボ変位 (train) に重ねて描いた。直線、右カーブ、左カーブのサーボ操舵角が実際の走行パターンにある程度一致しており、また安定した挙動に見える。また、count0 〜15 の直線部においては走行する上で見られていた揺れが抑えられているように見えた。しかし、直線からカーブに差し掛かる部分や、カーブとカーブの切り替わりのタイミング (赤色〇印) では必要以上に角度がついており、走行データに追従できていない部分があった。

4.5　追従精度の向上

次に、教師用データへの追従精度を高め、走行時の挙動の安定化を図ることを考えた。追従がうまくできていない部分はコースの分類が変わるタイミングで発生しており，その際の画像は図 4.13 や図 4.14 のようになっていた.

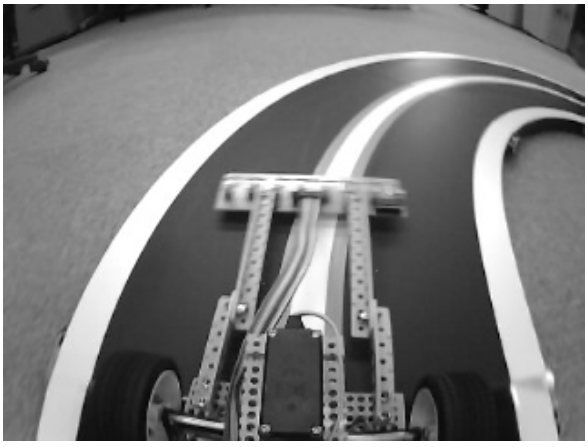

図 4.13　直線から右カーブへの切り替わり

図 4.14　左カーブから右カーブへの切り替わり

これらの画像はどちらのクラスに属しているかの判別がはっきりと分かれてしまうために制御量の

導出の際に滑らかな推移にならず、追従ができていないと考えた。図 4.9 を見るとそれぞれのクラスとの切り替わりのタイミングで推論確率が大きく変化し、いずれかのクラスの確率がほとんど 100 % になっている。制御量は推論確率とパラメータとの重み付き平均をとるようにしているため、いずれかの確率が 100 % に近づくとその他のクラスのパラメータが影響しない。そのため、切り替わりのタイミングでは右カーブに入る手前の位置で右カーブと判断され、想定より早めに操舵を大きく切ってしまうためコースアウトする結果となってしまった。

この問題を解決するために切り替わりのタイミングに該当する画像をどちらにも属するようにオーバーラップを行ったデータセットを作成し、学習を行った。図 4.15 にオーバーラップを行ったデータセットで作成したモデルを使用した場合の確率の変化を示す。画像をオーバーラップさせることによって特定のクラスにのみ高い確率を示すのではなく、滑らかに変化を得られると考えた。

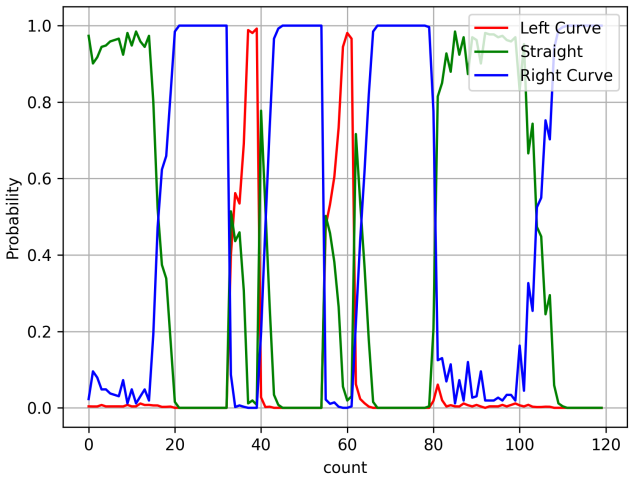

図 4.15　オーバーラップを考慮した推論結果

図 4.9 と比較すると特定のクラスが大きく反応している点は変わらないが、急激に反応して 100 % になるのではなく、滑らかに変化するようになっていた。

次に、このモデルを使用してシミュレーションを行った結果を図 4.16 に示す。

図 4.12 と比べると、図 4.12 の赤〇印に見えた急な切り返しがなくなり，追従の精度が上がっているように見える。特にカーブ部分は元々のデータセットを使用したモデルに比べて、左右どちらも走行データへの追従精度が向上していることが分かる。また、直線部分はフォトセンサ走行にはあった揺れが抑えられた挙動に見える。

以上のチューニングされたパラメータを使用し、実装実験を行った。

4.6　実装実験

シミュレーションにおける追従精度の向上と走行挙動の安定を確認できたため、構築したモデルを Maix bit に搭載して実際のコースを走行させる。実装実験を行った際の白線追従走行器のシステム構成を図 4.17 に示す。

初めにカメラによってコースラインの画像を撮影して前処理を行った後に CNN によるクラス分類

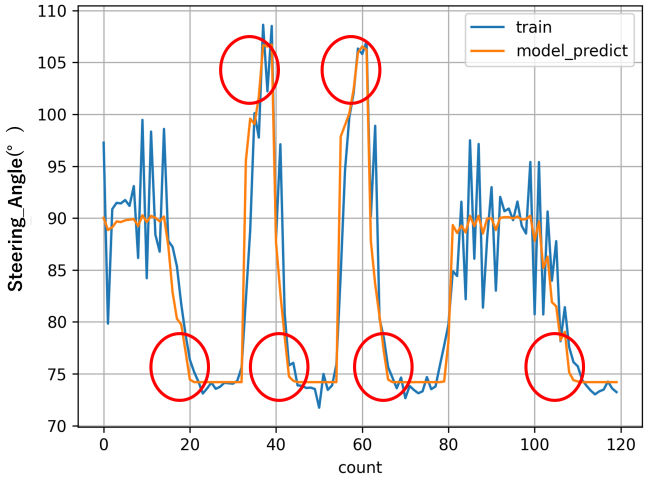

図 4.16　サーボ制御量のシミュレーション結果

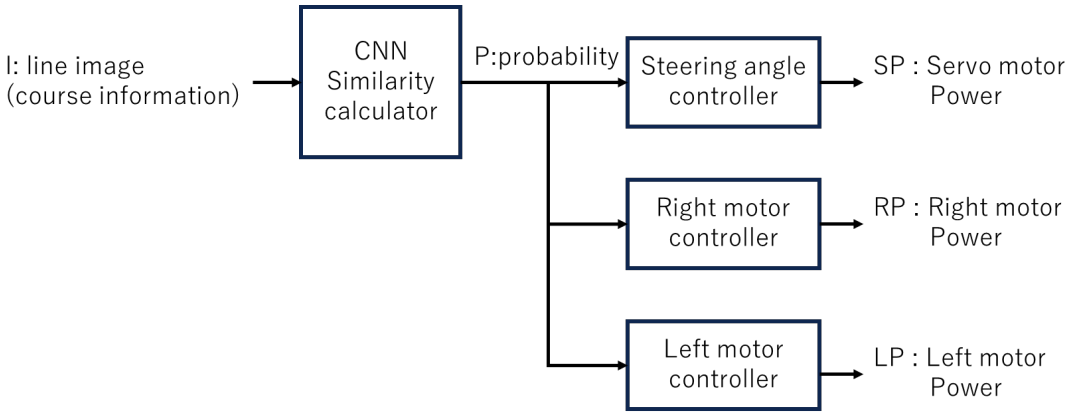

図 4.17　CNN を使用した白線追従制御器のシステム構成

の推測を行う。推測によって出力されるそれぞれの確率とあらかじめ設定した操舵角、右カーブ、左カーブのそれぞれのパラメータとの加重平均より制御量を決定し、PWM 信号を出力することで走行させるようにした。走行中の走行データは無線通信ができないため、Maix bit に取り付けられる micro SD カードに時々刻々と書き込むことによって取得するようにした。、

次に走行するコースの詳細を図 4.18 に示す。

コースは類似度ファジィ推論での走行と同じ Japan Micom Car Rally 規定のコースを用いた。スタートの位置を直線部分が一番長くなる図の位置に設定し、1 周以上の走行を行った。本実験では CNN の処理速度を考慮しモータの出力を半分に抑えた。

走行結果を図 4.19 と図 4.20 に示す。

図 4.19 に走行時の操舵角の推移を示し、図 4.20 に左右モータの走行データを示す。どちらも横軸はサンプル数を示し、操舵角走行データの縦軸は操舵角度 (°) を示し、左右モータ走行データの縦軸はモータ出力 (%) である。操舵角度は 90 ° のときまっすぐ進み、90 ° より小さいと右方向に向き、90 ° より大きいと左方向を向くようになっている。

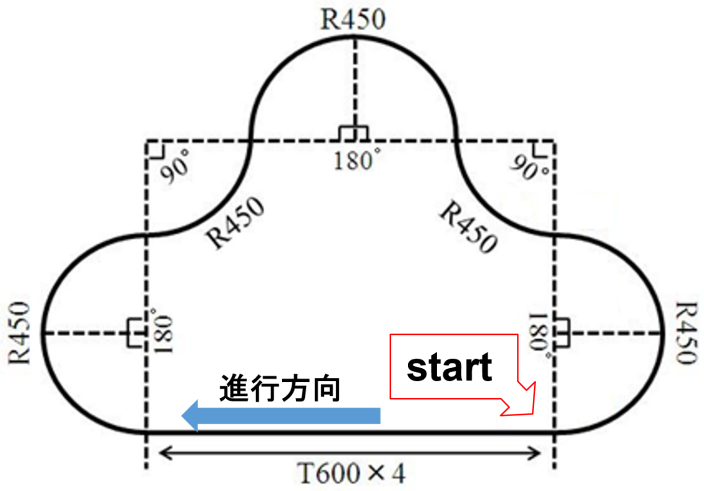

図 4.18　走行コース

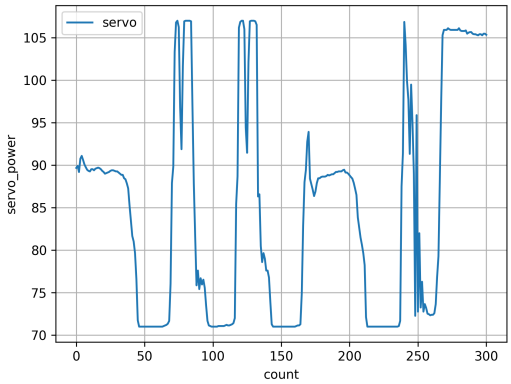

図 4.19　操舵角

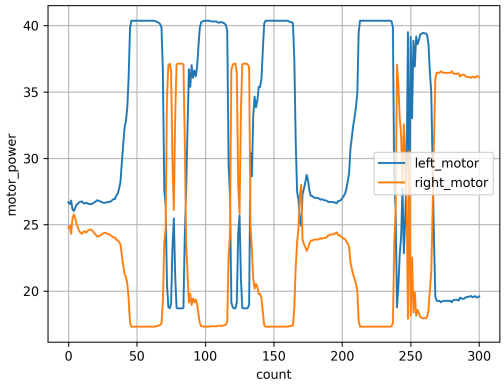

図 4.20　左右モータ

図 4.20 において青色のグラフに左モータの出力を示し、オレンジのグラフに右モータの出力を示している。モータについては出力 0 % で停止、100 % で最大出力となり、出力値が大きくなると加速、出力値が小さくなると減速する。

本実験での走行では 1 周走行した後に 2 周目に入ると直線部と初めの右カーブまでは走行できていたが、左カーブに入るときにうまく曲がり切れずコースアウトしてしまった。操舵角を見るとフォトセンサ走行のときに比べて count 0〜40、count 160〜200 の直線部分での揺れが抑えられていた。1 周目の count 40〜60、count 90〜120 にあたる右カーブ部分では揺れも見られず一定の角度を保ったまま走行できていることが確認できた。左カーブ部分では一定の出力ではなく、直線部に近い出力が確認でき揺れが発生していた。また、カーブが終わって直線部に戻るときにも少し揺れが発生していた。

一周走行時の走行時間、サンプリング数、サンプリングごとの処理速度を表 4.2 にまとめた。

走行時間は 1 周で 23.4 s かかっていた。これはシステム構築の際にモータの制御量を抑えたためである。実行時間は画像の取得、CNN での推論、制御量の計算と出力までの処理時間を測定した。

表 4.2　走行時間、走行速度、サンプリング数、処理速度

	CNN 走行
走行時間 (s/1lap)	23.4
走行速度 (m/s)	0.43
サンプリング数 (個/lap)	172
実行時間 (ms)	135.3

測定には time ライブラリを使用した。

フォトセンサを使用した時の走行時間は 2 周で約 15 秒であり、CNN を使用したライントレースカーが 1 周するまでに 2 周するほど速い。特に実行時間に大きく差があり、フォトセンサでの走行は 1 サンプリングあたり約 680 us に対し CNN での走行は 135 ms と約 200 倍になっている。これは、フォトセンサでの走行はフォトセンサからのデータ取得と AD コンバータによる変換のみであるのに対し、CNN での走行はカメラの待ち時間と CNN での推論に多くの時間をとられているためである。特にカメラを使用するとカメラの性能に左右され、今回使用したモジュールは 11 FPS で動作するため 91 ms よりも処理速度を速くすることは困難である。

4.7　考察

2 周目にコースアウトしたことについて考察する。本実験では 2 周目の初めの左カーブでコースアウトし、その部分は図 4.19 における count 240〜260 となっている。コースアウト後の count260 以降は車体がライン上に存在しない画像が入力されることになるが、左カーブと推測される確率が高かったため左に曲がろうとしていた。グラフを見ると 2 周目の右カーブまでは 1 周目と同じ動作をしているが、左カーブに侵入してすぐに出力が安定しなくなり、右カーブと判定されていることからラインの外に向かって走行したと考えた。

この方法では個々の事例に対して部分的にチューンアップすることが難しく、操舵角や左右のモータの出力を微調整するしかない。その結果、最初の 1 周目の走行データも変わってしまい、最適解を得るのに途方もない時間がかかってしまうのが欠点である。そのため、これ以上の調整は行わず、別の制御法を試してみることにした。また、カメラ撮影時のフレーム速度が遅く、これはコース判定の CNN 時間も含んでいるため、これ以上の速度向上が困難であると思われたことも制御方法を変更した 1 つの理由である。

第 5 章　ラインのベクトル情報を用いたライントレースカーの製作

5.1　CNN を使用したライントレースカーとの変更点

Pixy2 カメラユニットはほぼリアルタイムにラインのベクトルを検出する機能がある。この機能は 60 FPS で動作することができ、この機能を使用することで CNN より高速処理が可能であると考えて実験してみることにした。

まず、Pixy2 カメラユニットを使用するためのライントレースカーを新たに作成した。本実験ではライントレースカーのボディを 3D プリンタによって出力し、メイン基板部分やモータードライバー制御基板を設計し、ライントレースカーを製作した。製作したものは以下の通りである。

5.1.1　ライントレースカーのボディの作成

本実験で使用したラントレースカーのボディ部分は 3D プリンタによって出力した。まず、ボディ部分の 3D データを作成する。本実験では図 5.1 のような 3D データを高知工科大学の山本利水先生より提供していただいた。

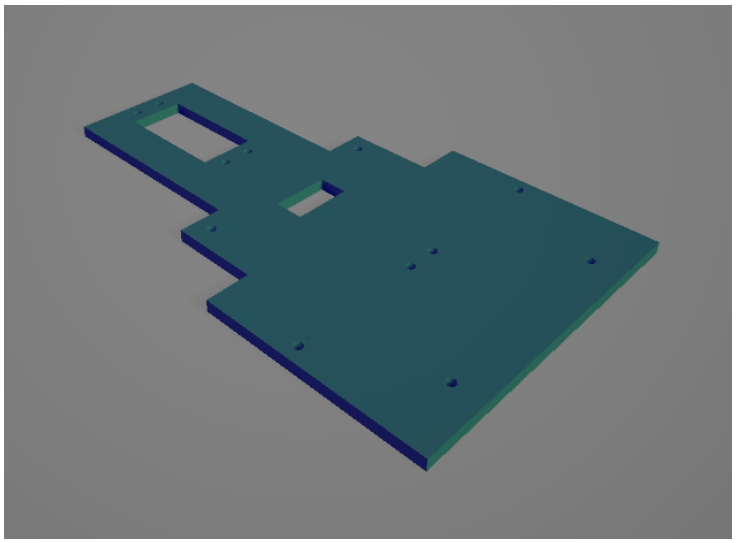

図 5.1　ライントレースカーのボディパーツ

3 D データを「123D Design」によって画面内に出力した [22]。

作製したボディの大きい穴は、サーボモータを取り付けるための穴と左右のモータとの信号線を通すための穴であり、小さい穴は基板とモータをボディパーツに固定するための直径 3 mm のねじ穴となっている。注意点として、3D プリンタで出力したとき、ほとんどの場合で設計した時の採寸よりも小さくなってしまう。ボディにあけられている穴は実際のサーボモータやねじのサイズに合わせて設定されているため、出力されたものをそのまま使おうとすると穴に収まらない。そのため出力されたものはその都度やすりで削るなどして調整する必要があり、今回はサーボモータ用の穴をやすりで広げ、取付用のねじ穴にドリルを通して実寸に合うよう調整した。

5.1.2　制御基板の設計と製作

次にサーボモータと左右のモータを制御するための基板を作成した。本実験では制御用基板をメインの制御用基板とモータの制御用基板に分けることにした。メインの制御用基板は以前までのラインントレースカーキットに搭載できるサイズにしておき、モータードライバー基板とケーブルで接続して動作するようにすることで、マイコンの交換が容易にできると考えた。

設計した基板の回路図を図 5.2 に示す。

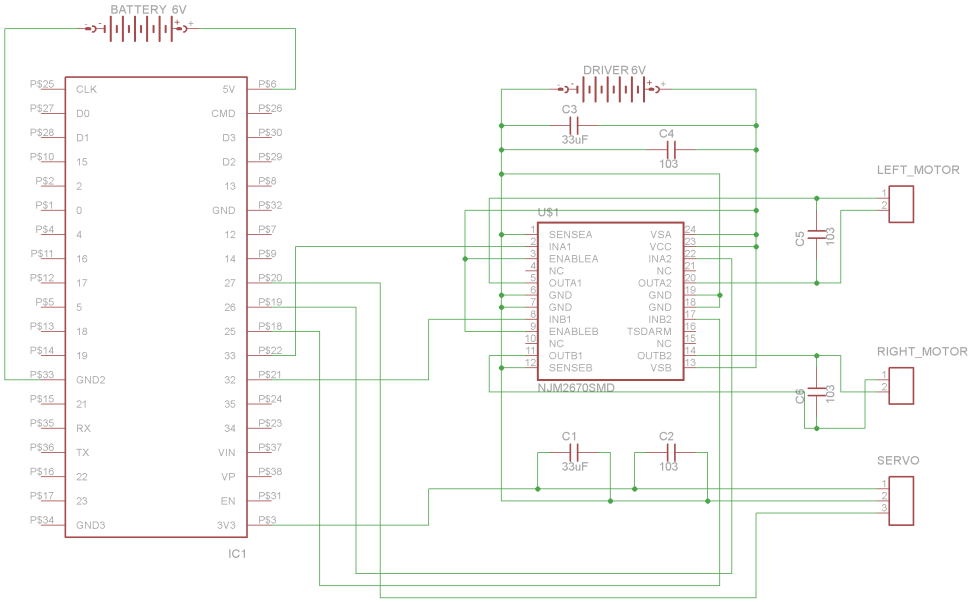

図 5.2　設計した回路図

図 5.2 において左側にある IC が ESP32、中央にある IC がモータドライバーの NJM2670 である。右側にあるものがモータであり、上から左モータ、右モータ、操舵用サーボモータとなっている。

また、回路内にはノイズの低減のためのコンデンサをいくつか設置してある。ノイズの除去を行っていない場合、電源-グランド間において大きくノイズが発生しており、サーボモータに影響して制御が効かなくなっていたために設置する必要があった。

左右モータ出力端子の間、電源の両端、サーボモータの電源とグランド端子の間にそれぞれ設置しており、モータにはセラミックコンデンサのみを使用し、それ以外にはセラミックコンデンサと積層コンデンサを使用した。更にサーボモータへのノイズの影響が少なくなるように電源をモータの電池からとるのではなくマイコンの 3V3 端子から電源を供給するようにした。

NJM2670 は汎用のデュアル H ブリッジドライブ IC であり、一つの IC で 2 つのモータを制御することができることからライントレースカーのモータードライバーとして採用した。大きな特徴として、H ブリッジ、サーマルシャットダウン回路、アラーム出力から構成されている。アラーム出力によってアプリケーションの問題を検出することができ、マイクロプロセッサで監視することによりシステムの信頼性を大幅に向上させることができる。従って、マイクロプロセッサで駆動する 2 相ステッピングモーター・アプリケーションに適している。

5.2　ベクトルの現れ方

完成した機体を用いてコースの特徴的な位置に置いた時のベクトルの現れ方を調べた。ベクトルの現れ方を図 5.3 に示す。

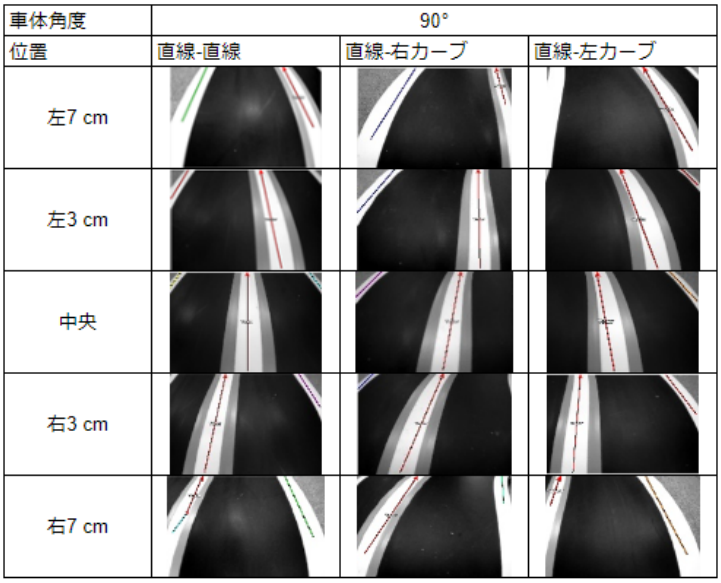

図 5.3　コースの位置による見え方の違い

車体の角度をコースと並行の状態 (90°) にし、直線位置とそれぞれのカーブに入る位置でのベクトルの見え方を調べた。それぞれの部分において車体に取り付けたカメラの位置が、中央、左右 3 cm、レーンから落ちるぎりぎりの左右 7 cm になるように設置してコース上のベクトルを取得した。どの位置においても取得されるベクトルは、ほとんどが中央と外側の線の 2 本を取得しており、直線の中央にいる場合には真ん中と左右の 3 本のベクトルを取得していた。直線部分では、車体が右側にあると真ん中の白線から得られるベクトルは 90 ° よりも大きくなり、車体が左側にあると 90 ° よりも小さくなっていた。左カーブ直前の位置において車体が左 3 cm でベクトルは 90 ° に近くなり、右カーブ直前の位置では車体が右 3 cm の位置でベクトルが 90 ° に近くなっていた。

次に車体の角度を変更した時のベクトルの見え方を図 5.4 に示す。

図 5.4 では車体の角度を 75 ° に変更して車体角度 90 ° と同じ位置でのコース上に見えるベクトルを取得した。車体を左に傾けているためカメラ位置が右 7 cm の位置になるよう設置すると後輪がコースからはみ出してしまうため取得できない位置があった。また、右カーブに入る位置で左 7 cm にいる場合コースの外側の線のみ取得し、内側の線は見えなくなっていた。

次に車体の角度を 105 ° に変更した時の見え方を図 5.5 に示す。

図 5.5 では車体の角度を 105 ° に変更し、車体角度 90 ° と同じ位置でのコース上に見えるベクトルを取得した。車体を右に傾けているため、車体角度 75 ° と同様に左 7 cm の位置になるように設置すると後輪がコースからはみ出してしまい、ベクトルの取得ができない状態となってしまった。

図 5.3〜図 5.5 においてそれぞれの車体角度におけるベクトルの見え方を確認すると、図 5.4 の赤枠で示した車体角度 75 ° -直線-左 7 cm の外側のベクトルと、図 5.5 の赤枠で示した車体角度 105

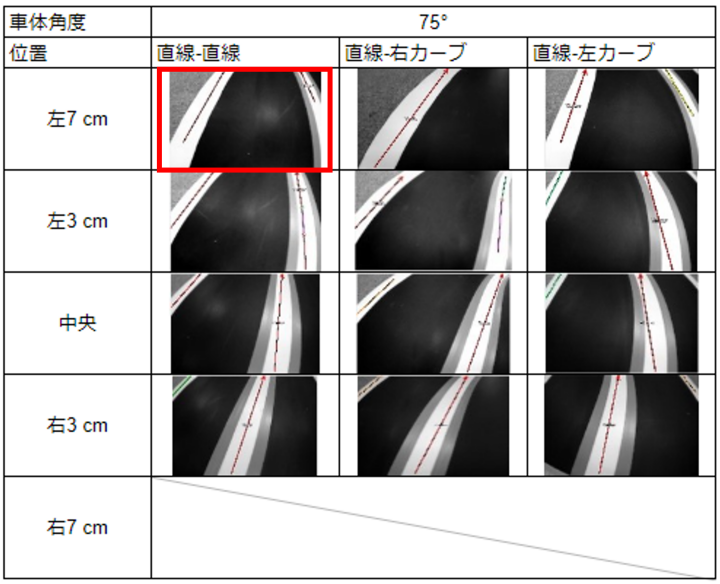

図 5.4　車体の角度を 75 ° に変更

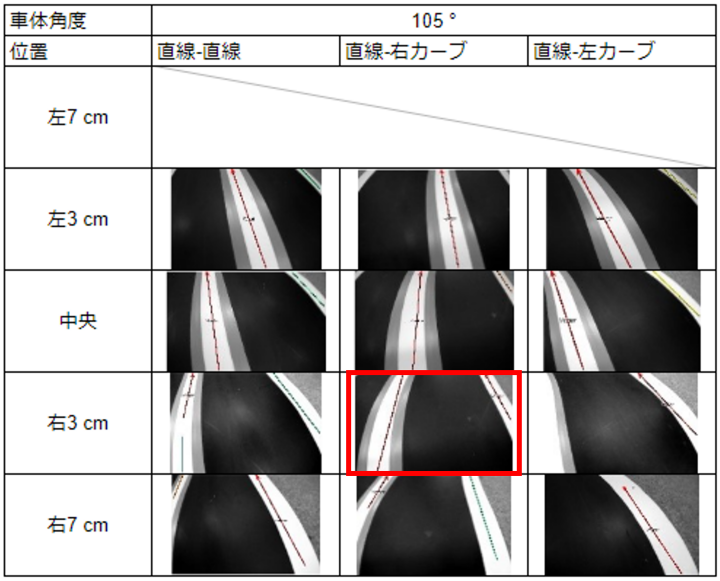

図 5.5　車体の角度を 105 ° に変更

° -右カーブ-右 3 cm の中央のベクトルのように外側のベクトルと中央のベクトルが同じように見えることがあることが分かった。車体が左側にあるときは左外側にあるラインがあたかも中央のラインにあるかのように表示されることがあり、その逆も同様であった。

　このことが単純にベクトルから車体の位置や車体角を決められない大きな要因であり、本研究において最も困難な部分になった。

5.3 サポートベクターマシンによるベクトルの分類

サポートベクターマシン (Support Vector Machine : SVM) とは、機械学習における分類と回帰の問題を取り扱う手法の一つである [23, 24]。1960 年代からある手法だが、1992 年にこの手法がカーネルトリックという考え方により拡張され、現在でも認識性能が高い手法の一つとして広く利用されている。

本実験では線形サポートベクターマシンを使用してコースのクラスの分類を試した。線形サポートベクターマシンの本質は、直線や平面のような「まっすぐ」で「平ら」な境界によって分類を行うことである。

はじめに、分類を行うためのデータを用意する。SVM を使った走行器のコースはオーバルのコースでの走行を目標として開発を行った。Pixy2 から ESP32 に送信されるデータはベクトルの始点と終点にあたる (x0,y0)、(x1,y1) の 4 つの値である。ベクトルの角度に関しては ESP32 内の atan2 関数によって算出した。

直線、右カーブ、左カーブを分類するために、はじめにサークル状のコースで左回りのみできるプログラムと右回りのみできるプログラムでそれぞれ走行させ、次に直線のみ走行できるプログラムで直線を走行させ、データを取得していった。コース中央の白線を取得するようにするために表 5.1 に示した値で Pixy2 の設定を行った。

表 5.1　Pixy2 設定

Configure	value
Edge threshold	40
Maximum line width	250
Minimum line width	0
White line	ON
Camera brightness	60

特徴量としてベクトルの始点の x 座標 (x0) と、ベクトルの角度 (deg) の 2 つを入力し、直線、左カーブ、右カーブの 3 クラスを分類する 2 特徴量 3 クラスの分類器を作成した。作成した SVM の表示結果を図 5.6 に示す。

青色の点が直線を示し、緑色が右カーブの値を示し、黄色が左カーブの時の値を示す。また、赤の点はサポートベクトルを表し、各領域の境界に最も近い点である。図 5.6 を見ると、線形サポートベクターマシンサポートによる境界がきちんと作成されており、境界線を使うことによって分類が可能であると判断した。このときの境界は直線と右カーブの境界が $deg = -0.6x0 + 126$ となり、直線と左カーブの境界が $deg = -0.475x0 + 98$ となった。この境界条件に基づき、それぞれのクラスに判定されたときにそれぞれの走行パターンで動作させるようにした。

5.4 SVM による走行実装実験

SVM を使用した時の走行コースを図 5.7 に示す。

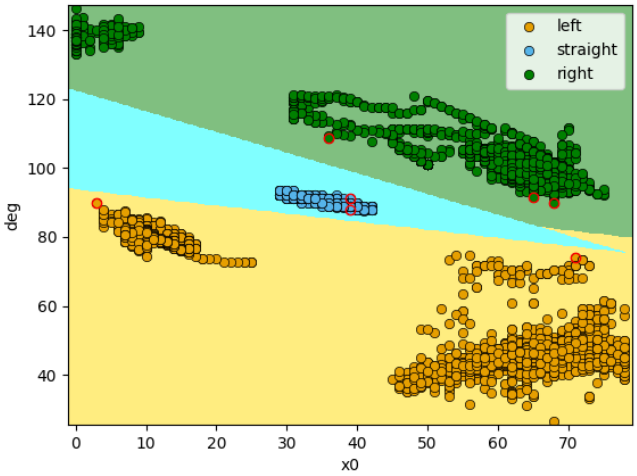

図 5.6　x0 と deg の値から SVM によって分けられた境界

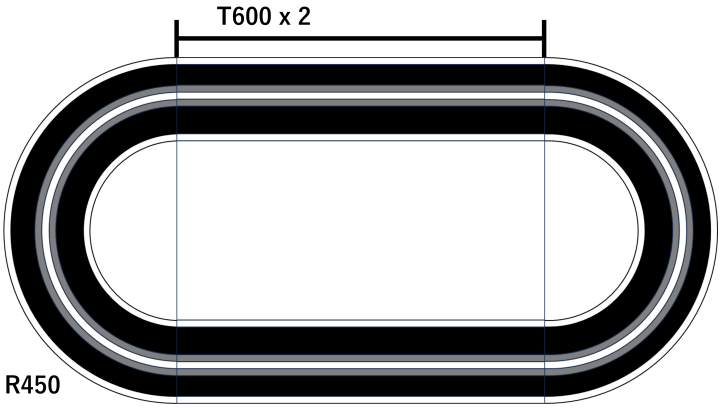

図 5.7　SVM を使用した時のコース

CNN による走行コースと同様に Japan Micom Car Rally のコースを使用し、オーバルのコースに変更して右回りでの走行実験を行った。このコースは、直線 600 mm × 4 枚、カーブ半径 450 mm × 8 枚を用いており、全長約 5.2 m となる。

SVM による境界によって直線と右カーブを分類し、取得されたベクトルの角度に応じてサーボと左右モータの制御量を決定するプログラムで走行を行った。直線と右カーブ分類時における角度に応じた制御量の設定は表 5.2 と表 5.3 のようにした。

直線部分では車体角度が 90 ° でサーボモータの角度が 90 ° になることを基準として、図 5.3 で車体が左側にあるとベクトルが 90 ° よりも小さくなることから、90 ° を下回った時にはわずかに右に向くようにし、反対に 90 ° を超えた場合には車体が右側にあるためわずかに左に向いて進むよう、サーボモータと左右のモータ出力を調整した。右カーブ判定部分では 90 ° を下回った場合には車体が左側にあるためまっすぐ進ませるようにした。90 ° 以上のベクトルが取得された場合はその区分に応じて右に曲げる度合いとモータの出力を変化させるようにした。

走行結果を図 5.8 と図 5.9 に示す。

車体角度 (°)	サーボ (°)	左モータ (%)	右モータ (%)
～80	94	44	34
81～90	92	39	39
90	90	39	39
91～100	88	39	39
101～110	86	39	39

表 5.2　直線判定時の設定

車体角度 (°)	サーボ (°)	左モータ (%)	右モータ (%)
～90	90	39	39
91～110	97	44	34
111～130	98	44	34
131～	103	48	34

表 5.3　右カーブ判定時の設定

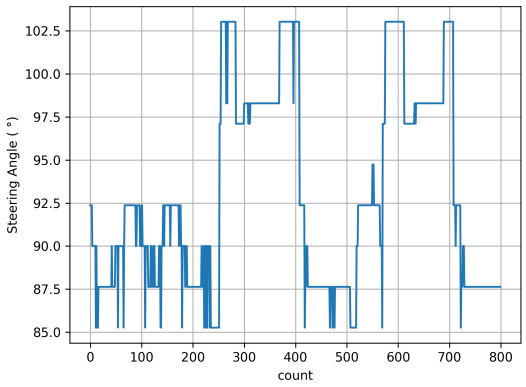

図 5.8　SVM 走行時の操舵角走行データ

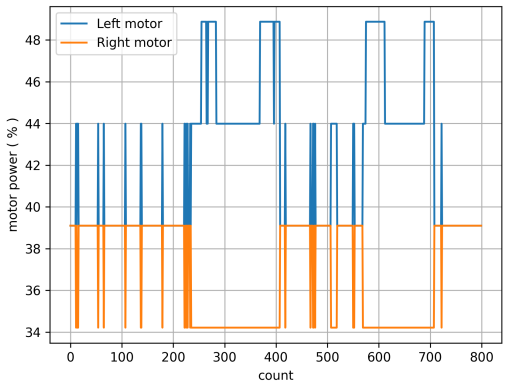

図 5.9　左右モータ走行データ

どちらも横軸にサンプル数を示しており、操舵角走行データの縦軸は操舵角度を示しモータ走行データの縦軸にはモータの出力を示している。

走行挙動を見ると count 0 ～ 250 あたりの直線部での揺れを確認できた。これは直線部のプログラムの制御として取得されたベクトルの角度に応じて出力を変化させているため 90 ° ちょうどの時はまっすぐ進むが、少しのずれで左右のどちらか斜めに進むことになってしまうために逆方向に進もうと働き、それが繰り返されてしまうために起こってしまう。この辺りに課題がある。

count 250 ～ 400 あたりまでは右カーブを走行しているときの走行データとなっている。走行データを見ると初めに大きく右に振り右を向いたまま走行した後、最後にもう一度大きく右に操舵を切っていることが分かる。これは右カーブ時に中央の白線からベクトルを取得するのではなく、外側の白線からベクトルを取得しているため大きく舵を切っていると考えられる。これも課題として残る。

右回りでの走行は連続で数周できたが、数周走行できる場合もあれば 1 周でコースアウトすることもあり、余り安定していなかった。また、左回りでの走行に関していえば 1 周できる時もあれば初めのカーブでコースアウトしてしまうことがあり、右回りよりも安定性に欠けていた。ベクトル情報から SVM で 3 クラスに分類する方法では、直線やカーブに入った直後の蛇行運動が激しく、適当な補正方法が見つからなかった。

5.5　走行手法の変更

5.5.1　2 特徴量 5 クラス分類

SVM による境界の分類から判定するライントレースカーの走行が確認できたため、次に分類するクラスを増やしたうえで滑らかな走行ができないか試すことにした。

初めに 2 特徴量 5 クラスでの SVM による分類を行い、それぞれのクラスにおいて重心となる位置を算出した。

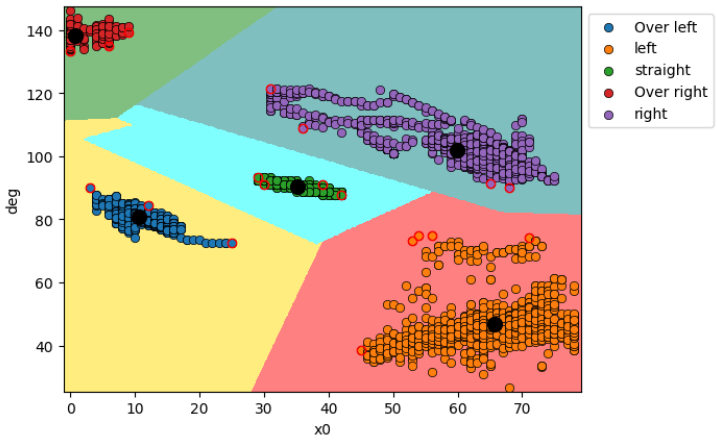

図 5.10　SVM による 5 クラス分類

走行データは図 5.6 の右回りのデータと左回りで 1 周成功した時のデータを組み合わせたものを使用し、特徴量として横軸にベクトルの始点の x0 を取り、縦軸にベクトルの角度 deg を選択した。Over left と Over right は x0 が小さいため外側の白線を認識した時のベクトルを表し、left、straight、right は中央の白線を取得したときのベクトルを表している。これは図 5.6 において左カーブや右カーブにそれぞれ 2 つの塊が存在していたことを意識したものである。SVM の境界を見ると、3 クラス分類の境界のなかで右または左と分類される領域を分割するような境界が作られていた。

また、各領域における黒の点はそれぞれのクラス内の走行データの重心位置を示している。

5.5.2　制御量の決定

本実験ではこの各クラスの重心を利用し、入力された点と重心の距離に応じてサーボと左右モータの制御量を決定するようにする。まず初めに検出されたベクトルの x0 と deg の位置と 5 クラスの各重心との距離をユークリッド距離によって計算する。入力点を $(x0,\ deg)$、重心点を $(M_x,\ M_{deg})$ とすると距離 d_i は下のようになる。

$$d_i = \sqrt{(x_0 - M_x)^2 + (deg - M_{deg})^2} \tag{5.1}$$

次に計算した各重心点との距離のうち距離の近いものを残し、残りを無視するようにした。パターンとしては距離の近いもの 1 つだけ残すパターンから距離の近いもの 5 つを使うパターンを設定した。

次に制御量の決定方法について説明する。制御量の決定には加重平均によって計算を行う。まず初めにそれぞれの距離 d_i の逆数を計算し、距離の近いものの値を大きくした。また、このとき距離を無視するパターンでは逆数をとった後に無視する値を 0 にした。

次に加重平均を計算するためのパラメータを DE によってチューニングを行った。チューニング元のデータとして図 5.8 のデータを使用し、サンプルごとの x0 と deg の値から距離 d_i の逆数 5 つを計算し、サーボ、左モータ、右モータとそれぞれ加重平均をとって制御量の出力とするため、パラメータを計 15 個チューニングする。各世代において DE によるパラメータと d_i の逆数を重みとした加重平均を計算し、サンプルごとの走行データとの誤差を RMSE で計算する。このとき、d_i は正規化した。最後に RMSE の値が最小となり、DE の収束が確認できたパラメータを走行パラメータとし、シミュレーションを行った。

シミュレーション結果のうちサーボモータの出力を図 5.11〜5.15 に示す。

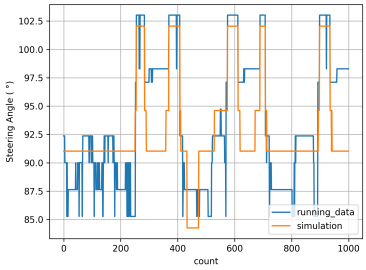

図 5.11　keep1

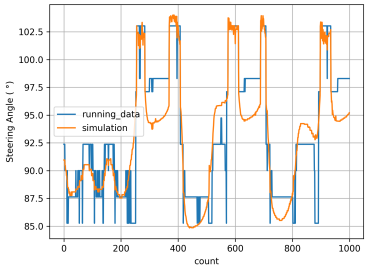

図 5.12　keep2

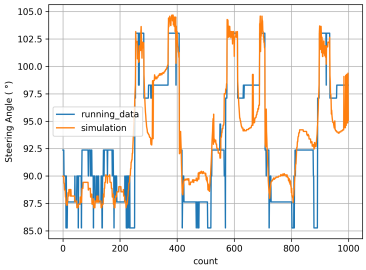

図 5.13　keep3

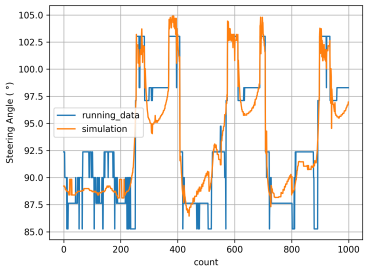

図 5.14　keep4

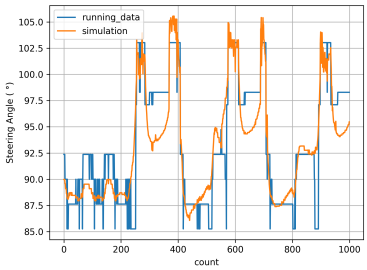

図 5.15　keep5

図 5.11〜5.15 は距離 d_i の計算後に距離の近いものを残す処理のパターンを示しており、keep1 が最も距離の近いもののみを残すパターン、keep5 がすべての距離の情報を使用した時のパターンとなっている。keep2、keep3、keep4 はそれぞれ 2 つ残すパターン、3 つ残すパターン、4 つ残すパターンを示す。

keep1 を見ると、このパターンは距離の最も近いものを残すため、正規化した場合距離の最も近いクラスの重みが 1 となり、残りは 0 となる。そのため、5 つのパラメータのうち選択されたクラスに最も適した値が出力されるようになるため、グラフのように段階的な出力となっていた。keep2〜keep5 は元の走行データに比べて出力が滑らかに変化しており、また、keep1 に比べて元の走行データに対する追従性が良くなっているように見える。

次に、DE によるパラメータの最適化において収束したときの RMSE の値を表 5.4 にまとめた。

表 5.4　収束したときの RMSE

Parttern	RMSE
keep1	3.59
keep2	2.61
keep3	2.65
keep4	2.28
keep5	2.65

keep1 から keep5 において最も RMSE の値が大きいのは keep1 であり、最も RMSE の値が小さいのは keep4 であった。keep1 の RMSE が最も大きかった原因として、直線部での変化がない点や、カーブの途中での出力の差が大きい事から出力の変化が段階的なものになってしまうためと考えられる。残りの keep2、keep3、keep5 に関しては RMSE がほとんど同じ結果となった。

5.5.3　実装実験と修正

それぞれのパターンにおけるパラメータを決定した後で、ライントレースカーに実装し、走行実験を行った。コースは SVM による走行と同じく図 5.7 のオーバルのコースを使用した。

keep1〜keep5 のパターンにおいて走行実験を行ったが、一周できないパターンがいくつかあった。走行できたのは keep1、keep2 のみで、keep3、keep4、keep5 パターンはカーブを曲がり切れず、外側にコースアウトしてしまった。走行できた keep1 と keep2 のサーボモータ出力を図 5.16 と図 5.17 に示す。

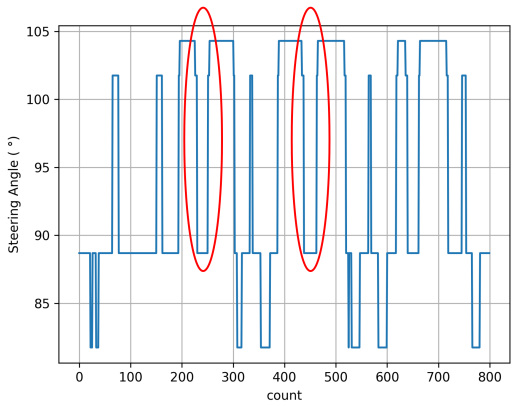

図 5.16　keep1 サーボモータ出力

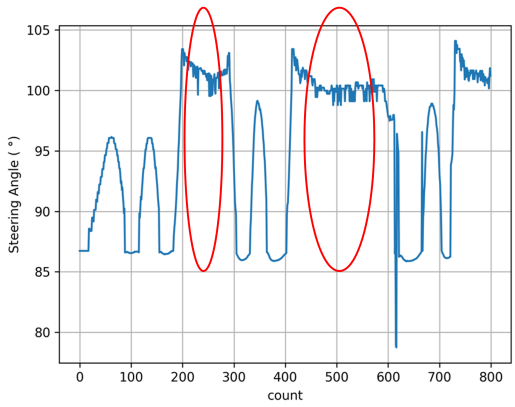

図 5.17　keep2 サーボモータ出力

走行挙動を見ると、シミュレーションのように keep1 は段階的な出力をしており、keep2 は滑らかに出力が変化していた。どちらも直線部分において揺れが起きており、keep1 の方が大きく揺れていた。

この二つを比べて大きく異なっている点は、赤枠で囲んだカーブを走行している際の出力にある。keep1 では走行判定を右、直線、右と変化させて曲がっているのに対して、keep2 では初めに少し大きく右に振れているが、ほとんど一定の出力を保ったままカーブを走行できていることが分かる。

しかし、keep2 において 1 回目のカーブに対して 2 回目のカーブで走行時間が延びていた。これは 2 回目のカーブでコースぎりぎりの大回りをしているために起きていた。また、このまま走行を続けているとコースを 3 周をしたあたりでコースから脱線していた。

以上の方法で右回りの走行が確認できたため次に左回りでの走行を試した。初めに同じプログラムを使用して左回りの走行を行ったがどれもうまくいかなかった。そのため、DE にかけるデータを変更し、左回りで走行するデータの量を増やすことで左回りでの安定化を図った。これらのチューニングを繰り返すことにより keep5 で右回りと左回りを同じプログラムで制御することができた。

keep5 でのサーボモータの出力を図 5.18 と図 5.19 に示す。

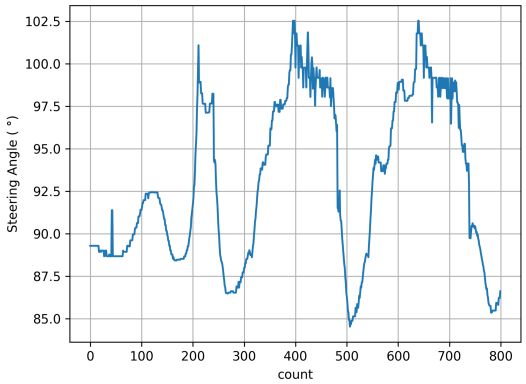

図 5.18　右回りでの keep5 サーボモータ出力

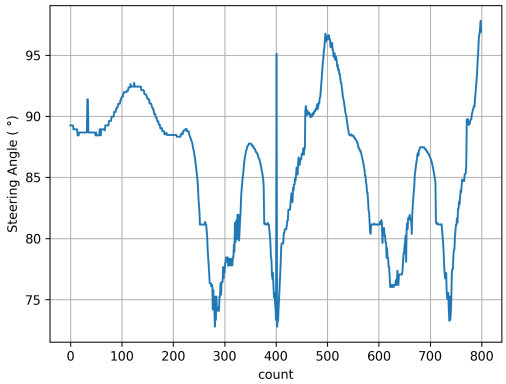

図 5.19　左回りでの keep5 サーボモータ出力

走行挙動を見ると、最初の右カーブは count 200 〜250 に見られるように初めに大きく右に曲がった後、緩やかにカーブしながら走行している。count 250 〜300 はカーブを抜けた後、直線部に侵入する時の動作であり、これを見ると直線部に入った瞬間は一度左に少し曲がって修正を行っているように見える。その後は同様の動作を繰り返している。

左回りの挙動を見ると、count 200 〜400 が左カーブを走行しているときの挙動であり、初めに大きく振れた後、まっすぐ進み、最後にもう一度左に大きく振れることで曲がっていた。

一周走行時の走行時間、サンプリング数、サンプリングごとの処理速度を表 5.5 にまとめた。

表 5.5　走行時間、走行速度、サンプリング数、処理速度

	SVM keep5 走行
走行時間 (s)	13.93
走行速度 (m/s)	0.38
サンプリング数 (個/lap)	875
実行時間 (ms)	15.9

全長約 5.2 m のオーバルのコースにおいて 13.93 秒で 1 周していた。

表 4.2 の CNN を使用したライントレースカーの走行データと比較すると、実行時間に大きく差があり、CNN を使用したライントレースカーのサンプリングごとの実行時間は 135.3 ms であり、対して Pixy2 を使用した場合は 15.9 ms と大幅に実行時間を短縮できていた。

Pixy2 は 60 FPS で動作しており、実行時間を見ると、画像の更新のタイミングで処理が完了していることが確認できる。CNN を使用した走行は推論に時間をとられているため、実行時間が Pixy2 に対して大幅に増加していると考えた。

最後に、オーバルのコースではなく図 4.18 の左右のカーブが混在する競技用コースでの走行を試した。しかし、完走することができず、右カーブまでは走行できたが右カーブから左カーブに侵入する際にコースを捉えられずコースアウトしてしまった。これについてはパラメータの調整で制御するか、分類方法を変更するかの結論は出せなかった。

5.6 考察

keep5 の走行挙動についての考察を行う。本実験での最終的なライントレースカーの走行挙動は、カーブにおいて初めに大きく曲がり、次にまっすぐ進み、最後にもう一度大きく曲がるような挙動をしていた。これはチューニングに用いた走行データの走行挙動が、同様の挙動をしているためだと考えられる。初めにチューニングしたパラメータを利用して走行データをとりつつ、何度もチューニングを行うことで同じプログラムで左右どちらも走行できるようにはなったが、初めの SVM での走行挙動の影響を強く受け、カーブの挙動の修正はできなかった。SVM を使用した走行では、カーブにおいてコース外側の白線を取得したことによって大きく曲がる行動をとるため起こってしまうものであった。これは左右どちらのカーブにおいても起こっており、原因としては一定の出力のままカーブを曲がり切るようなパラメータや条件を見つけることができなかったためである。最適なパラメータや条件を探すことは現状では難しく、これを今後の課題としたい。

第 6 章　まとめ

本研究の目的はカメラを用いた白線追従走行器を設計、製作し、コースを走行するシステムを開発することであった。CNN を利用する方法とラインのベクトル情報を利用する方法の 2 種類のライントレースカーを製作し、それぞれ別のコースを周回できるシステムを開発することができた。

使用したマイコンが情報の少ないものであったため、初めは参考とした書籍にあるようなプログラムを動かすだけでも苦労し、バージョンの違いや環境の違いに苦しめられた。

CNN を利用したライントレースカーは画像処理から始まり、CNN モデルの構築や画像のパターン変更を試し、競技用コースでの走行ができた。中でも判別の難しい画像を複数クラスに属するオーバーラップは推論における分類結果をどれか一つに絞るものではなく、ある程度の冗長性を持たせることに成功した。そのため、出力において急激な変化によるコースアウトを防ぎ、滑らかな走行挙動を実現することができたと考えられる。

ベクトル情報を用いたライントレースカーはコース上の白線をベクトルとして取得し、初めに直線のみ、右カーブのみ、左カーブのみの 3 つの走行データから SVM による境界を作成し、ベクトルを分類しながら制御量を決定する方法で走行を試した。この方法では右回りに関しては数周走行できたが、左回りでは 1 周までしか走行できなかった。

次に、この走行データをもとにしてチューニングを行い、5 クラスに分類した時のそれぞれのクラスの重心までの距離から制御量を決定する手法で走行を試した。この方法では 5 つのパターンを設定し、走行データを何度も取りチューニングを行うことで左右どちら周りでも走行できるものが完成した。ただ、カーブにおける挙動は一定の角度をもって曲がるのではなく、2 回大きく曲がることによって走行するものであった。この修正には、初めにとった 3 クラス分類の走行できれいにカーブを曲がれるデータを取れれば良かったと考えたが、最適なパラメータを見つけることを今後の課題とする。また、CNN を利用したライントレースカーでは競技用のコースを走行できたがベクトル情報を用いたライントレースカーでは走行できなかったため、心残りである。

今回、2 種類の方法でラインの情報を取得するライントレースカーを制作したが、想像よりも難しく、一筋縄ではいかなかった。初めて触れる分野が多く、うまくいかないことが多々あり、プログラムの問題なのか回路の問題なのか判別することが難しかった。特にモータを接続したときに発生するノイズの処理に苦労し、コンデンサを設置したりサーボの電源とモータの電源を別々のものを使うようにするなど様々な工夫を試みてある程度抑えることに成功した。

以上のことからカメラを用いた白線追従器の開発の目的は達成できたといえるが、少し安定性に欠けるものであったため、今後は改良を進めていく必要がある。

謝辞

今回の修士論文を執筆するにあたり、丁寧で熱心なご指導とご教示を賜りました高知工科大学システム工学群電子・光工学教室星野孝総先生並びに綿森道夫先生に、心からの感謝を申し上げます。また、ライントレースカーのボディパーツのデータの提供だけでなく、回路のノイズの除去に関して多くの知識をいただき指導してくださった同大学教育講師山本利水先生に、深く感謝します。本システムの開発にあたり、高知工科大学システム工学群 Soft Intelligent System on a Chip 研究室の皆様には、日頃から様々な意見を頂きありがとうございます。最後になりましたが、大学生活 6 年間を支えてくれた家族に深く感謝いたします。

参考文献

[1] David J Atkinson. Autonomy technology challenges of europa and titan exploration missions. In Artificial Intelligence, Robotics and Automation in Space, volume 440, page 175, 1999.

[2] Kunihiko Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural networks, 1(2):119–130, 1988.

[3] 松原仁 and 川村秀憲. 人工知能による文学創作 (小特集 創造性・芸術性における ai の可能性). 電子情報通信学会誌= The journal of the Institute of Electronics, Information and Communication Engineers, 102(3):240–246, 2019.

[4] 新井紀子 and 松崎拓也. ロボットは東大に入れるか?: 国立情報学研究所 「人工頭脳」 プロジェクト (¡ 特集¿ ロボットは東大に入れるか?). 人工知能, 27(5):463–469, 2012.

[5] Morikazu Takegaki and Suguru Arimoto. A new feedback method for dynamic control of manipulators. 1981.

[6] 橘優理子; 竹内佑太朗; 瀬川典久. 組み込みシステムとカメラを用いた COVID-19 のための身体的距離モニタリングシステムの提案. In: 人工知能学会全国大会論文集 第 35 回 (2021). 一般社団法人 人工知能学会, 2021. p. 3E2OS5b04-3E2OS5b04.

[7] 高橋慶行; 町田晃平. エッジコンピューティングを用いた生産ラインへの AI 組み込みに関する研究. 群馬県立産業技術センター研究報告= Reports of Gunma Industrial Technology Center, 2020, 5-8.

[8] 砂川寛行、トランジスタ技術 2019-11 月号、(社)CQ 出版社、東京、2019

[9] 西山由華, 四宮友貴, 山本利水, 星野孝総. 類似性ファジィ推論を用いた白線追従制御器の推論ルール数の検証. In: 日本知能情報ファジィ学会 ファジィ システム シンポジウム 講演論文集 第 36 回ファジィシステムシンポジウム. 日本知能情報ファジィ学会, 2020. p. 351-354.

[10] E.H. Mamdani, “Application of fuzzy algorithms for control of simple dynamic plant, Proceedings of the institution of electrical engineers,” vol.121, no.12, pp.1585-1588, 1974.

[11] T. Takagi, and M. Sugeno, “Fuzzy identification of systems and its applications to modeling and control,” IEEE transactions on systems, man, and cybernetics, vol.SMC-15, no.1, pp.116-132,1985.

[12] aNo 研、M5 シリーズで楽しむロボット開発、(社) 株式会社 シーアンドアール研究所、新潟、2020

[13] 下島健彦、IoT 開発スタートブック ESP32 でクラウドにつなげる電子工作をはじめよう!、(社) 株式会社技術評論社、東京、2019

[14] 大澤佳樹、アイデアをカタチにする!M5Stack 入門&実践ガイド [M5Stack Basic/M5StickC 対応]、(社) 株式会社技術評論社、東京、2022

[15] 後閑哲也、基礎入門 EAGLE によるプリント基板製作の素、(社) 株式会社技術評論社、東京、2009

[16] 中野雄大、WiFi に出力する電池駆動方式の信号処理端末の設計と製作、2021

[17] 青木直史、C 言語ではじめる音のプログラミング-サウンドエフェクトの信号処理-、(社) 株式会社オーム社、東京、2008
[18] 岩田利王、dsPIC 基板で始めるディジタル信号処理、(社)CQ 出版株式会社、東京、2009
[19] 雑誌、Interface2016 年 6 月号体感!全集 CD 付き!音声信号処理、東京、2016
[20] Rainer Storn, Kenneth Price, et al. Differential evolution-a simple and effecient adaptive schheme for global optimization over continuous spaces: technical report tr-95-012. International Computer Science, Berkeley, California, 1995.
[21] Rainer Storn and Kenneth Price. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization, 11(4):341-359, 1997.
[22] nekosan、はじめての「123D Design」、(社) 株式会社工学社、東京、2014
[23] 金丸隆志、RaspberryPi ではじめる機械学習、(社) 株式会社講談社、東京、2019
[24] CORTES, Corinna; VAPNIK, Vladimir. Support-vector networks. Machine learning, 1995, 20: 273-297.