レーンキーピングアシストシステムの実装開発

沼野 剛志　　笠井 正三郎†

Development of Lane Keeping Assistance System Implementation

Tsuyoshi NUMANO　　Shozaburo KASAI†

ABSTRACT

If we drive a car for a long time on the road such as expressways, the driver might mistake an acknowledgement and a judgment in the operating. As a result, we might cause a traffic accident. According to National Highway Traffic Safety Administration, 37% of road fatalities in the motorway in the United States is assumed to be the one by the road deviating. Then, we are paying attention to the lane keeping assistance system. The system recognizes traffic lane by the camera attached on the front of the car. If the car shifts from the vicinity of the center of the running lane, it assists maintenance of the traffic lane with addition the steering force to the steering wheel. We develop the lane keeping assistance system. In the previous research, the lane keeping assistance system is developed on the computer with image processing software. In this research, we try to make this system with hardware. We design the hardware that is run by FPGA. This system is set up CMOS camera and FPGA that is high integration, high speed, and low cost. We aim to develop the lane keeping assistance system.

Keywords : lane keeping assistance system, image processing, FPGA

1. 序論

近年, 交通事故の発生率は減少傾向にあるが, 依然として高い水準に変わりはない. 警視庁の統計 (平成 22 年)[(1)] によると, 交通事故の発生原因は, 前方不注意や安全不確認の要素のみで全体の約 80 %以上を占めている. つまり, 交通事故のほとんどは, 運転者の認知や判断ミスによって引き起こされていると考えられる. そこで近年, 自動車の運転の支援を行うレーンキーピングアシストシステムが注目されている. これは, 車の前方部に取り付けたカメラで車線を検出することによって, 自動車が車線から逸脱するのを自動的に防ぐもので, 運転者を支援するシステムの 1 つである. 長時間運転するドライバーに対して, 運転操作における疲労を緩和すると共に, 車線逸脱の注意喚起を行うことを目的としている.

先の研究[(2)]では, カメラから取り込んだ画像を処理し, 車の走行経路を予測するプログラムを開発するところまで進んでいた. しかし, メモリや算術演算を多用するアルゴリズムでは, リアルタイム処理に不向きなことと, ハードウェア上に直接実装することが困難であるという課題が残った. そこで, 本研究ではハードウェアに実装するための新たなアルゴリズムを開発し, それらをハードウェアに組み込むことでリアルタイム処理を実現させた. また, 小型車輪型ロボットにそれらを搭載し, レーンキーピングアシストシステムの実証を行った. そして, ほとんどの高速道路で使用可能なレーンキーピングアシストシステムを開発し, 所期の目標を達成することが出来たので報告する.

*電気電子工学専攻 2 年
†電子工学科　教授

2. 画像処理と走行線予測

本研究では, カメラから得られる画像を使用して道路上の車線を認識することでレーンキーピングアシストシステムを実現する. まず最初に, 車線認識を行うために必要な画像処理と認識された車線からの走行線予測について簡単に説明する.

2.1　2 値化　道路には様々な色が存在するが, 抽出する車線は白やオレンジである. 取得した画像を 2 値化し, 道路を「黒」, 車線を「白」とすることで, データの量を減らし, 処理を行いやすくする. カメラから得られる入力画像を図 1 に, それを 2 値化した画像を図 2 に示す.

図 1　取り込み画像

図 2　2 値化画像

2.2　車線検出　2 値化した画像で車線は白の幅のある直線となることに着目し, 車線を検出する. 画像の中心から左右にスキャンして行き, 横方向に幅を持った白画素を車線と認識する. 車線付近の拡大画像を図 3 に示す. 図 3 では車線の内側からグレーで示している部分のように, 3 画素の幅を持つものを車線と認識している. この操作を車線の存在する領域で繰り返し行い, 車線を検出する. 車線が破線になっている部分に関しては, 破線のうち空白部分の車線間の幅が, 他の車線間の幅と大きく異なることに着目して, 無視することにした.

図 3 拡大画像

2.3 走行線予測 上記の方法で検出された車線より, 道路の中心座標を求め, 走行線を予測する. 走行線の予測には最小二乗法を用いた. 最小二乗法とは, 測定値とモデル関数から得られる理論値の差の二乗和が最小となるようなモデルのパラメータを決定する手法のことである. 中心点の座標が $(x, y) = (x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$ というデータが与えられたとき, 求めたい一次方程式の式を $y = ax + b$ とすると各パラメータ a, b は次式 (1), (2) で求められる[(2)]. ここでの n は, 抽出した点の数である.

$$a = \frac{n\sum_{k=1}^{n} x_k y_k \quad \sum_{k=1}^{n} x_k \sum_{k=1}^{n} y_k}{n\sum_{k=1}^{n} {x_k}^2 \quad \left(\sum_{k=1}^{n} x_k\right)^2} \tag{1}$$

$$b = \frac{\sum_{k=1}^{n} {x_k}^2 \sum_{k=1}^{n} y_k \quad \sum_{k=1}^{n} x_k y_k \sum_{k=1}^{n} x_k}{n\sum_{k=1}^{n} {x_k}^2 \quad \left(\sum_{k=1}^{n} x_k\right)^2} \tag{2}$$

走行線を表示した画像を図 4 に示す. 画像サイズは 320 × 240 で行った. 検出された道路の中心座標の個数より $n = 14$ とし, 予測される走行線は式 (3) のように求まった.

$$y = \quad 6.32x + 8.62 \times 10^2 \tag{3}$$

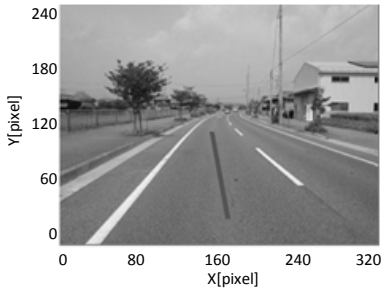

図 4 走行線を示した画像

3. ハードウェアによる画像処理

3.1 実装に必要なこと 画像処理を行うハードウェアに FPGA を使用することで, 高速処理・低コスト化・低消費電力化を実現する事が出来る. しかしその一方で, 小数点を含む乗除算や, メモリを多用した演算が困難になると言うデメリットも存在する. そのため, ソフトウェア上でシミュレーションを行ったアルゴリズムをハードウェアにそのまま組み込む事が出来ない. そこで, ハードウェアに適したアルゴリズムを開発した. また, 今回用いたハードウェアではメモリの制約があるため, 画像のサイズは 320 × 240 として設計した.

3.2 道路画像の分割 カメラから得られた道路画像は, 奥に行くほど車線の間隔が狭く, また, 白線の幅も狭くなっている. その特徴を用いて, 道路画像を図 5 のように, 4 つのラインで評価することにした. 4 つのラインで, それぞれそのラインに適した幅を持つものを車線と認識する. 4 つのラインだけで道路画像を評価できれば, ハードウェアのメモリを節約することだけでなく, 後の演算回数を減らすことができるので, 処理速度の向上が期待できる.

ラインと車線幅の決定は, 高速道路で撮影した画像を 20 枚程度比較し, その平均値を用いて行った. 撮影方法は, 車のダッシュボード (地面から垂直に 1.7m の距離) に取り付けたカメラを地面に対して水平に対して 5 °下向きに傾けたカメラより撮影を行った.

表 1 に, 今回用いたライン位置と車線幅を示す. それぞれのラインで, そのラインに適した幅を持つものを車線と認識する.

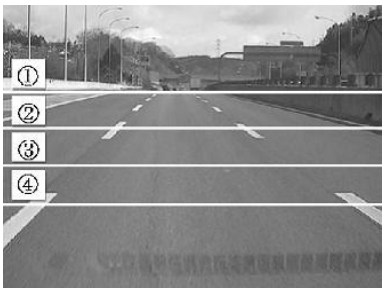

図 5 道路画像の分割

表 1 ライン位置と車線, 白線幅

ライン	ライン位置	車線間幅	白線幅
①	70	40〜60	6
②	120	70〜120	8
③	160	150〜190	10
④	200	270〜320	12

3.3 FPGA による車線検出 FPGA による車線の検出は, 4 つのラインを評価することで行う. 4 つのラインをそれぞれカメラからデータが得られる順にリアルタイムで走査して行き, そのラインに適した幅を持つものを車線と認識する. この手法では, ラスタスキャンしながらリアルタイムに車線を検出することができる. 図 6 に 4 つのラインで車線を検出し, 中心点を求めた画像を示す. 処理結果が確認しやすいように, 走査したラインを青, 中心点を赤でディスプレイに表示するように設計した. 車線の破線部分などのように, 車線が存在しない場合, もしくは表 1 に示している条件と一致しない場合, 車線が急激に変化しないと仮定して, 前回取得に成功した値を中心点としている.

図 6 中心点を求めた画像

また, 図 7 のように, 中心付近の狭い個所にノイズが集中している場合でも, 道路の中心点が急激に変化しないと仮定しているため, 前回取得に成功した値を中心点とすることで, 検出精度の向上を図った. しかし図 8 のように, 中心付近に分散したノイズが乗ると, 表 1 に示している条件と一致してしまい, ノイズを誤って車線として検出してしまう.

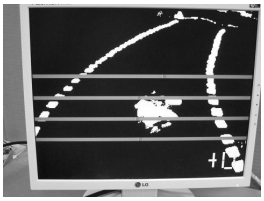

図 7 ノイズが集中している

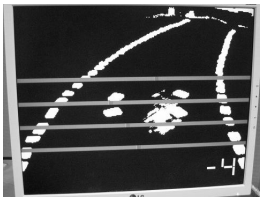

図 8 ノイズが横方向に分散

3.4 FPGA による走行線予測 道路を 4 つのラインで評価することによって, 4 つの車線の位置が検出される. 車線の位置情報により, 車線間の中心座標をそれぞれ得ることができる. 走行線の傾きを決定するため, その中でも, ライン 2 とライン 3 の 2 点間の差分を取ることとした. まず, 画像データの縦の軸を y 軸, 横の軸を x 軸とする. 2 点間で評価する場合, 予測される走行線の傾き a は次式 (4) で表される.

$$a = \frac{\Delta y}{\Delta x} \tag{4}$$

ここで, Δy は, ライン 2 とライン 3 の y 軸に対する画素の差であり, この値は表 1 より既知である. また, Δx は x 軸におけるライン 2 とライン 3 の車線の中心の画素の差とする. ここで, Δx が求まれば, 走行線の傾き a が求まる. しかし, 先にも述べたように, FPGA 上で小数点を含む乗除算を行うことはできるだけ避けたい. そこで, あらかじめ走行線の傾きを求めておき, 最低限の処理でデータを評価することを考える.

1 から画像の最大幅である 320 まで変化させたときの Δx と既知の Δy より a を求めておく. ライン 2 とライン 3 の x 軸におけるその差 Δx を求め, その結果より求まる a に一番近い値を出力とする. 今回の設計では, 出力に 4 ビット使用できるので, 1 ビット目を符号ビットとし, 残りの 3 ビットを用いて, 走行線の傾きを 8 段階で評価した. 8 段階で評価した結果を表 2 に示す. また, 表 2 に対応した 8 段階の傾きを 図 9 に示す. 1 ビット目の符号ビットは, 右に傾いていれば ”1” , 左に傾いていれば ”0” を出力する. 例を挙げると, ライン 2 とライン 3 の差 Δx が ”-18” だった場合, 表 2 より, 番号は③, 傾き a は ”-2” で, 出力される 4 ビットは ”0011” となる. また, 求まった走行線の予測データは FPGA から約 16.7ms 毎にパラレルに 4 ビット出力され, 走行制御用の SH-2 マイコンでそれらを取り込むように設計した.

実際にこの手法で道路画像を撮影し, 出力した結果を図 10 に, また, 図 11 にこのとき予測されている走行線の傾きを示す. 図 10 の右下に黄色で表示されている数字が, 表 2 の番号を表しており, 数字の前についている符号がプラスかマイナスで, どちらに傾いているかを示している. つまり, この場合 ”+1” と表示されているので, 傾き a は ”+6” であり, 出力される 4 ビットは”1001”となる. 入力画像の車線の傾きに対して, 近い値の走行線の傾きが予測されていることが分かる.

この手法を用いると, 乗除算を用いず, カメラからのデータをリアルタイムに処理し, 走行線の傾きを決定することができる.

表 2 2 点間の差と傾きの対応

番号	傾き a	2 点間の差 Δx	3 ビット出力
⓪	12〜	$0 \leqq \Delta x < 4$	000
①	6	$4 \leqq \Delta x < 7$	001
②	3	$7 \leqq \Delta x < 13$	010
③	2	$13 \leqq \Delta x < 20$	011
④	1.5	$20 \leqq \Delta x < 27$	100
⑤	1	$27 \leqq \Delta x < 40$	101
⑥	0.75	$40 \leqq \Delta x < 53$	110
⑦	0〜0.75	$53 \leqq \Delta x < 320$	111

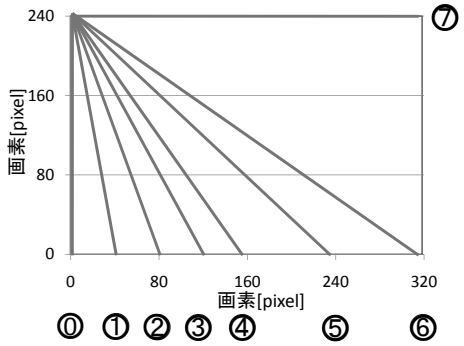

図 9 8 段階の傾き

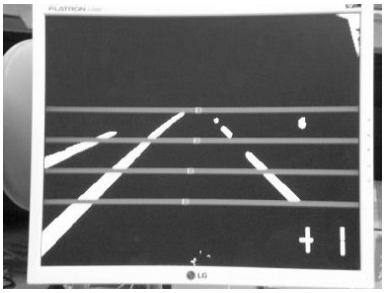

図 10 道路画像のディスプレイ出力

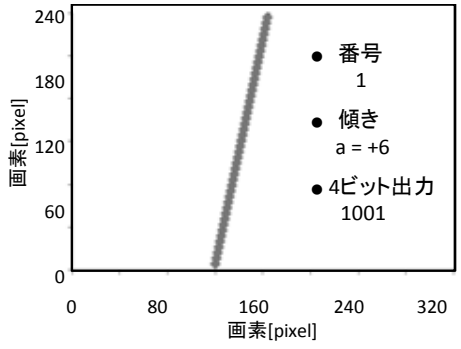

図 11 予測される走行線

4. システム

実際にハードウェアにそれらを組み込むためのシステムを構築する．システム全体の構成を図 12 に示す．カメラから取得した画像データを FPGA で処理し, 3.4 のアルゴリズムで走行線を予測する．予測された走行線のデータ (傾き量 3bit, 傾きの符号 1bit) を FPGA からマイコンに送信し, マイコンが走行線に応じた出力をモーターに送ることで走行制御を行う．画像処理と走行制御の処理を分けることで, 効率化を図った．開発したシステムの流れを以下に示す．

1. カメラで道路を撮影し, 道路画像データを FPGA へ転送.
2. FPGA による 2 値化, 車線検出, 走行線の予測処理.
3. 走行線のデータを SH-2 マイコンへ 4 ビットで出力.
4. SH-2 マイコンでデータを受信し, モータで走行制御.

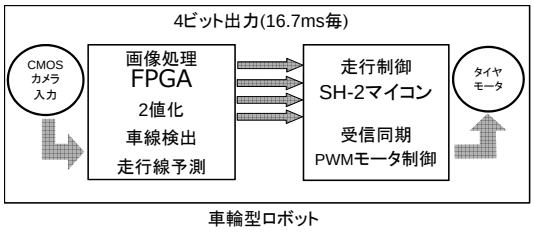

図 12 システムの構成

4.1 車輪ロボットの概要 車輪ロボット型ライントレーサ CQ-BBLTR-M (株式会社イーエスピー企画), 画像処理ボード CQ-SP3EDW2 (株式会社イーエスピー企画) と CMOS カメラ OV7640 (米国 OmniVision Technologies 社) を組み合わせて, カメラ搭載車輪型ロボットを製作した．実際に製作したマシンを図 13 に示す．車体のフロント部分に CMOS カメラを取り付け, 道路の情報を画像データとして読み取れるようにした．CMOS カメラから得られた画像データより車線を検出し, 車が車線から逸脱しないように制御する．FPGA と SH-2 マイコンの接続は, それぞれの I/O ピンを直接ケーブルで繋ぎ, 4 ビットの通信が行えるように設計した．

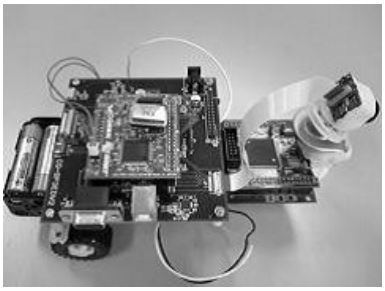

図 13 車輪型ロボット

4.2 FPGA から SH-2 マイコンへのデータ転送 今回の設計では, カメラ・モジュールから得られた画像を, VGA 出力でディスプレイに表示している．VGA 出力では, 1 枚の画像を 1 秒間に 60 回出力することができる．そこで, FPGA から SH-2 マイコンへのデータ転送は, 画像データが 1 回更新されるごとに, 4 ビットの信号が 1 回出力されるように設計した．つまり, FPGA で行った画像処理によって予測された走行線予測情報を, SH-2 マイコンへ 1 秒間に 60 回のペースで送信することが可能である．データ転送のタイミングについて詳しく説明するため, まず, 今回設計した VGA 画像信号タイミングの設計について述べる．

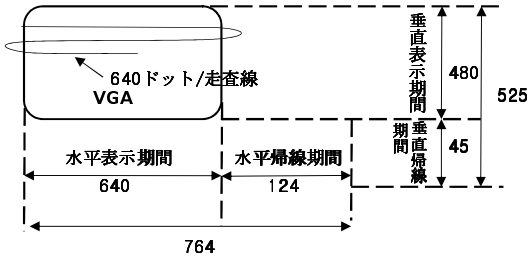

図 14 VGA 画面のタイミング設計

図 14 に VGA 画面のタイミング図を示す．CRT ディスプレイは陰極線によって作られる光のポイントを左右に走査（水平走査）することにより水平 1 ライン分の画像を表示する．光の点が左端から右端まで走査し, 再び左端まで帰ってくる期間を「全水平走査期間」と呼ぶ．そのうち画面表示期間は「水平表示期間」, 左端に帰る期間は「水平帰線期間」と呼ぶ．ピクセルクロックは液晶表示モジュールおよび CMOS カメラの標準値に合わせて 24MHz, FPGA の基本クロックはその 2 倍の 48MHz とする．今回の設計では CMOS カメラの仕様に合わせて

- ピクセルクロック数／ Hsync：764　(水平表示期間 640 ＋水平帰線期間 124)
- 水平走査線数／ 1 フレーム：525　(垂直表示期間 480+垂直帰線期間 45)

とした．これにより繰り返し周波数は

$$\frac{\frac{24.0\times10^6}{764}}{525} = 59.8$$

となる．つまり, 1 秒間の間に, 画像データが約 60 回更新される．先ほど述べたように FPGA から SH-2 マイコンへのデータ転送も, このタイミングで行う．そのため, 以下の時間間隔で走行線予測信号が FPGA から出力される．

$$\frac{1}{24.0\times10^6}\times764\times525 = 16.7\text{ms}$$

この出力タイミングが具体的にどのような数値か考える．高速道路を自動車が 80.0km/h の速さで走っているとする．16.7ms というタイミングで出力を行うとすると

$$\frac{80.0\text{km/h}}{36.0\times10^2\text{s}} = 22.2\text{m/s}$$
$$22.2\text{m/s}\times16.7\times10^{-3}\text{s} = 37.1\times10^{-2}\text{m}$$

つまり, 自動車が約 37cm 進む毎に走行線予測信号が出力される．SH-2 マイコン側では, 上記のタイミングで出力されている信号を, 10ms 毎にサンプリングし, データを取得している．

5. 実機実験

実際にアルゴリズムを FPGA に組み込み, 実機実験を行った. 道路のモデルを CMOS カメラで撮影し, ディスプレイに出力した様子を図 15 に示す. 奥にあるディスプレイにはカメラから得られた画像データが出力されている.

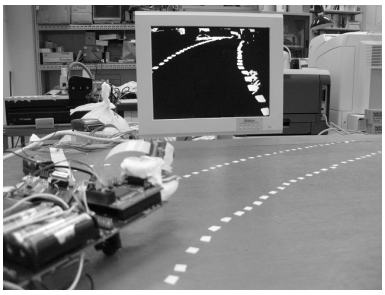

図 15 動作の確認

5.1 実験環境 本研究で用いている実験環境について述べる. 表 3 で高速道路における実スケールと実験環境を比較する. 車線横幅や, 道路横幅については車輪型ロボットの大きさに比例したスケールとした. 車線縦幅と車線間隔は, 高速道路を 80km/h で走行している場合を想定して設計している. なぜなら, 今回使用したアルゴリズムでは, 4 つのラインごとに車線を検出するため, 車線における破線部分の空白が発生するタイミングを考慮しなければならない. そのため, ロボットの進行速度と実際の車の速度を比較し, 破線が起こるタイミングを優先させて, 車線縦幅と, 破線部分の車線間隔を決定した.

高速道路を 80km/h で走行すると, 車線幅である 8m を通り過ぎるのにかかる秒数は 0.36s である. ロボットの進行速度は現在, 20mm/s なので

$$0.36\text{s} \times 0.02\text{m/s} = 7.20 \times 10^{-3}\text{m}$$

となる. そのため, 車線縦幅は 7.2mm とした. 同じように車線間隔も計算して 10.8mm としている. 表 3 に今回用いた実験環境と, 実際の高速道路のスケールを示す.

表 3 高速道路におけるスケールと実験環境

	実スケール [m]	実験環境 [mm]	比率
車線横幅	0.2	6	3/100
道路横幅	3.5	105	3/100
車横幅	2.5	75	3/100
車線縦幅	8	7.2	9/1000
車線間隔	12	10.8	9/1000
(速度)	(80km/h)	(20mm/s)	9/1000

また, 高速道路におけるカーブに対応するため, 仮想道路内のカーブの曲率半径は 4.5m とした, これを実スケールに直すと, 約 150m の曲率半径になる. 日本に存在する高速道路での最小の曲率半径は, 名阪国道の天理東 IC から福住 IC にかけての 150m である. すなわち, この環境で正しく制御できれば, 国内のどの高速道路でも実用可能であると考えられる.

5.2 実機の動作 図 16 に, 出力される走行線に対して, 車輪型ロボットを仮想道路内で制御した様子を示す. 図 16 では, カメラから得られた道路画像と, その画像より予測された走行線データがリアルタイムにディスプレイに表示される. また, 走行線のデータは SH-2 マイコンに送信され, そのデータを元に車輪型ロボットが制御される. 今回, 制御方法には, 比例制御 (P 制御) と PID 制御を用いた.

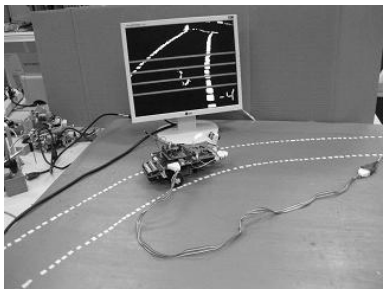

図 16 仮想道路上での動作

5.2.1 比例制御による動作 比例制御では, 走行線の傾きが小さければ小さいほど, つまり, 車線からのずれが大きいほど, 左右のモーターの回転数の差が大きくなるように設計した. 制御の目標値は表 2 で示した番号⓪に収束するように設定している. 動作を撮影した動画より, ディスプレイに表示される走行線の値を 0.5s ごとにサンプリングし, 実際の道路上での走行位置に置き換えたものを図 17 に示す. また, カーブ終端付近の拡大図を図 18 に示す. 比例制御を用いた手法では, 走行位置がかなり車線に近づいてしまっていることが分かる. 目標値からのズレより計算した誤差率は 58.7%であった. しかし, 走行線はディスプレイに表示されているように, リアルタイムで予測できている. つまり画像処理に関して言えば, 4 つのラインでも, 十分に適応が可能である.

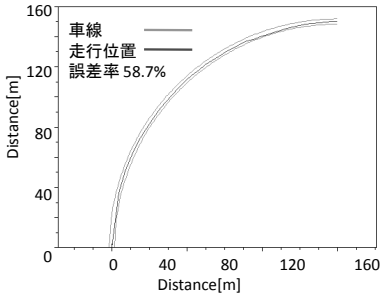

図 17 比例制御による動作の軌跡

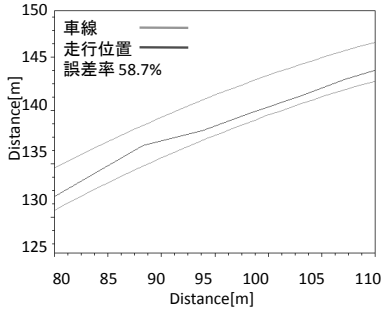

図 18 比例制御による動作の軌跡 (拡大図)

5.2.2　PID 制御による動作　図 19 に今回設計した PID 制御のブロック図を示す. ここで, Kp は比例要素, Ki は積分要素, Kd は微分要素である. それぞれのパラメータの値は実験を行いながら適宜調節を行った. 今回の実験で用いたパラメータは以下の通りである.

$$Kp = 0.8 \quad Ki = 0.01 \quad Kd = 0.3$$

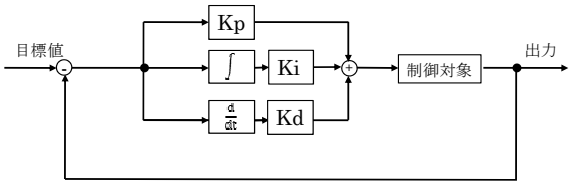

図 19　PID 制御のブロック図

PID 制御では比例要素に加え, 定常偏差を抑えるための積分要素, 応答速度を向上させるための微分要素を加えている. PID 制御を行った結果を図 20 に示す. また, カーブ終端付近の拡大図を図 21 に示す. 細かい範囲でジグザグに進んでいるが, 比例制御ほど大きな変化が無いことがわかる. また, PID 制御での誤差率は 47.1%であった.

両者の誤差を比較した図を図 22 に示す. 縦軸は道路の中心からのずれを表している. PID 制御は比例制御（P 制御）と比較して, 誤差率が 11.6%改善された. 性能の評価としては, 一部の点で設計範囲を超える出力が見られるが, 比例制御より高い精度で制御できていることが分かる.

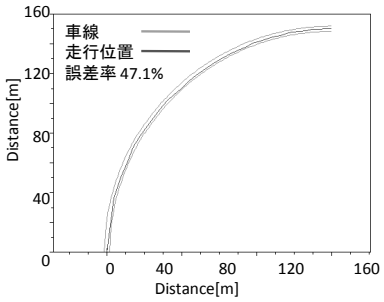

図 20　PID 制御による動作の軌跡

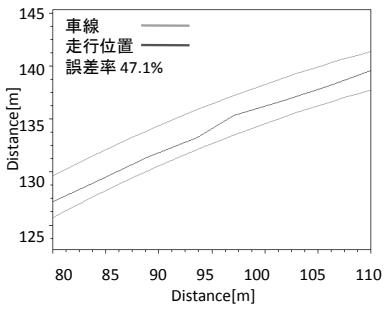

図 21　PID 制御による動作の軌跡 (拡大図)

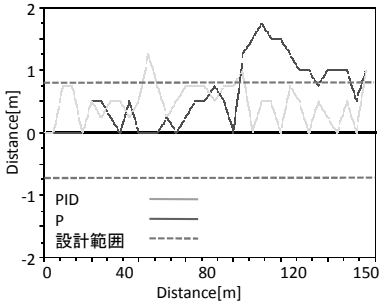

図 22　誤差による比較

動作開始時点で比例制御側の精度が高くなっているが, これは制御設計の問題ではなく, 画像処理部において車線を誤認識していると考えられる. この問題は, 2 値化を行う部分で発生しているため, 今後は 2 値化のしきい値決定手法に微分ヒストグラム法などを用いることで改善することが必要である. また, 今回は車の横幅を 2.5m, 道路の横幅を 3.5m として設計したので, 許容できる範囲を実際の道路で考えると, どちらの制御法に関しても車体の一部が車線からはみ出してしまっていることになる. この原因としては, 予測された走行線の分解能が 3 ビットしか無いため, 画像処理の部分で細かい判定が行えていないことが大きな要因として挙げられる. 今後, 制約の少ないハードウェアで実現できれば, この部分の誤差を低減することが出来ると考えられる.

6.　結論

本研究では次の 4 項目を実施した.

1. システムの構築と実機の製作
2. アルゴリズムのハードウェア化
3. FPGA での車線検出, 走行線予測
4. マイコンでの簡単な車輪型ロボット制御

画像処理を行う部分と, 走行制御を行う部分とを, 二つに分けてシステムを構築した. FPGA で走行線を予測することに関しては, ソフトウェア上でシミュレーションを行ったように求めることができない代わりに, 新たなアルゴリズムを開発した. そのアルゴリズムを実際に組み込み, 車線を検出し, 走行線を予測することができた. また, 予測された走行線より, 2 つの制御手法で車輪型ロボットを動作させるところに至る. 今後の課題として, FPGA での画像処理に関しては, 車線の抽出精度が信頼できるレベルではないのでさらなる検討が必要である. 他にも, FPGA での走行線予測に関しては, ハードウェアの都合上, ライン 1 とライン 4 を用いることができていない. そのため, 今回設計したハードウェアだけにこだわらず, 順序処理を行える別のハードウェアでシステムを開発していくことも検討していく.

参考文献

(1) 警視庁: 警視庁ホームページ警視庁の統計平成 22 年 http://www.keishicho.metro.tokyo.jp/toukei/bunsyo/toukei22/k_tokei22.htm, 2011

(2) 益田祐次: 「レーンキーピングアシストシステムの開発」, 神戸高専研究紀要第 48 号 pp.69-74, 2010