レーンキーピングアシストシステムの開発

益田 祐次* 笠井正三郎†

The Development of Lane Keeping Assistance System

Yuji MASUDA* 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, he might cause a traffic accident. And so, we are paying attention the lane keeping assist system. The system recognizes traffic lane by the camera in 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. First of all, we extract white line from the road image, and we forecast running line. Next, we design the hardware that uses FPGA. The system sets up CMOS camera and FPGA that is high integration, high speed, and low cost. We test circuit that displays check and display an image taken from CMOS camera. Finally, we will develop the lane keeping assist system.

Keywords : lane keeping assistance system, image processing, FPGA

1. 序論

交通事故の発生には様々な要因が関わっている．このうち, 人の身体能力や心身状態に関わる人的要因は, 交通事故の原因の 90 %を占めるとも言われており, 交通事故防止対策の検討対象となることも多い. また, 自動車運転作業は, 認知・判断・操作という一連の作業から構成されており, 交通事故の原因についても, 認知ミス, 判断ミスあるいは操作ミスに分類して論じられている[(1)]. 運転者の認知・判断ミスを防ぎ, 交通事故を減らすことは急務であり, 近年, 自動車の運転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. レーンキーピングアシストシステムは, 車の前方部に設置したカメラにより道路上の白線を抽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ある.

本研究では, まずは走行線を決めるために必要な基準線として, 白線を画像処理により抽出するアルゴリズムを検討する. さらに, 車載用に小型化し, 実動作に対応できる様に処理速度をあげるためにハードウェアで実現させる. 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.

2. 画像の前処理

2.1 概要 道路画像から白線を抽出するための基本的な手法を確立するため, カメラで撮影した静止画像 (320×240 の JPEG) をパソコンに取り込み, OpenCV を使用しソフトウェア処理により, 撮影画像から白線を抽出する[(2)]. 図 1 に走行中の自動車から撮影した写真を示す. この画像より白線を抽出するため以下の処理を行う.

図 1 取り込み画像

2.2 2 値化処理 一般のカラー画像では, 1 つの画像から多くのことが分かる反面, 情報が多すぎて処理が複雑になるという欠点を持っている. この欠点は画像が大きくなるほど顕著になってくる. その膨大な情報の中で欲しい情報だけを抜き出してくる作業の 1 つが 2 値化である. 2 値化処理は多くの画素の色情報を整理し分類する処理である. 画素の値がある基準値（閾値）よりも大きいか小さいかで分類し閾値より大きいときは 1, 小さいときは 0 とすることで画像の色情報を 1 と 0 の 2 値に分けて画像を単純化する[(3)(4)].

道路には様々な色が存在するが, 抽出する白線は白（あるいはオレンジ）である. 取得した画像を 2 値化し, 道路を「黒」, 白線を「白」にする. 色は 0〜255 の 256 段階で表現されている. 閾値を半分の 128 とし, 2 値化した画像を図 2 に示す.

光の反射や道路の凸凹により, 白（オレンジ）線以外までもが白になってしまっている. そこでこの画像で白（オレンジ）線を「白」と判断するのに適している値 150 とし, 2 値化した画像を図 3 に示す. 光の反射もほぼなく, 道路が「黒」, 白線が「白」になっている.

*電気電子工学専攻 2 年
†電子工学科 教授

図 2 2 値化画像 (閾値 128)

図 3 2 値化画像 (閾値 150)

2.3 ノイズ除去処理 最適な閾値で 2 値化を行っても, 道路には凸凹や光のあたり方などによる点や穴といったノイズが残っている. これを膨脹と収縮を繰り返し行うことにより, ノイズを除去する. 膨張とは近傍に 1 つでも白があれば白にする処理 (図 4) で, 収縮とは近傍に 1 つでも黒があれば黒にする処理 (図 5) である. ノイズ除去処理を行った画像を図 6 に示す. この処理により画像中心部の白点や白線中の黒点が除去されていることがわかる.

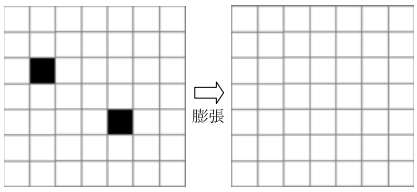

図 4 膨張処理

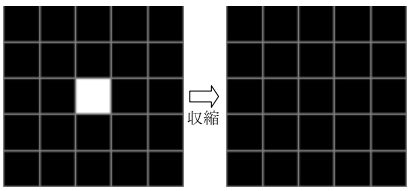

図 5 収縮処理

図 6 ノイズ除去処理を行った画像

3. 白線の抽出

白線の抽出には, 道路と白線で色の変化する部分をエッジとして抽出する手法, 白線を幅のある白画像としてテンプレートマッチングで抽出する手法, 最後にこれらを組み合わせた方法で抽出を行う.

3.1 エッジ強調による抽出 Sobel オペレータを用いて画像から白線の向きである垂直方向のエッジを抽出する. Sobel オペレータとは局所積和演算でグラジエントの強度（微分値）を求める方法で, 急激に色が変化する部分をエッジとして抽出する. あるピクセルの周辺ピクセルの濃度の微分を求め, それを新たな濃度値とする. つまり, 周辺のピクセルとの濃度値の差分を加算することで, 計算対象のピクセルの濃度値を強弱させる. この差分は 2 次元に対して行われるので, 垂直方向の係数行列 V($= v_{ij}$) と対象となるピクセルとその周辺からなる 3×3 の行列 I($= i_{ij}$) について各要素毎の積を足し合わせた値の絶対値になる.

$$差分 = \left| \sum_{i,j=1}^{3} v_{ij} \times i_{ij} \right|$$

垂直方向の係数行列 V を以下に示す.

$$V = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Sobel オペレータを用いた計算例を図 7 に示す. これを用いて道路と白線で色の変化する部分をエッジとして抽出する.

ここでの問題点は道路と白線以外のエッジも抽出してしまう点である. 道路以外の風景にエッジが存在すると白線ではない部分までが白線と判断されてしまう. そこで, 複数のサンプル画像により画像中にある白線の位置を仮定し, その範囲でエッジを抽出する.

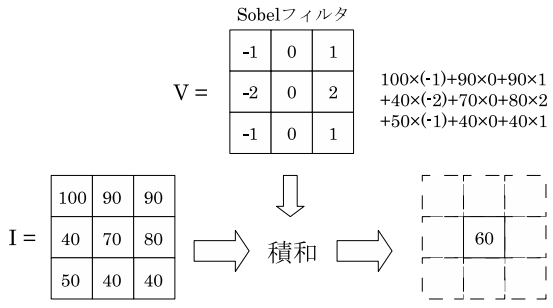

図 7 Sobel 画像

3.2 テンプレートマッチングによる抽出 テンプレートマッチングとは, あらかじめ用意した特定のテンプレート画像と入力画像で重なっている部分を探す方法である. テンプレート画像には, 白の正方形画像用いる. 2 値化した画像では白線は白の幅のある直線であることに注目する. 複数のサンプル画像により白線の幅を仮定し, その範囲内で最適であろう 7×7 画素の正方形をテンプレート画像として用いた. 画像の下方中心部から左右にマッチングさせていき, 一致した座標を記録していく.

初めて一致したところから白線の部分までは連続して一致すると予想できる．その一致した座標を記録していき同じ y 座標での一致した最初と最後の座標から白線の幅は算出できる．これにより y 座標がわかると白線の幅と白線の中心の座標が算出できる．

3.3　マッチング改善法による抽出　今まで述べてきたものは 1 ピクセルに対して毎回演算処理をしており，処理が多くなってしまうことや一致した座標を全て記録しておくため多くのメモリを使用するなどの問題点がある．そこで，道路と白線で色の変化する部分をエッジとして抽出する手法とラスター操作により白線以外のものを抽出しないようにするテンプレートマッチングの良い点を用いたマッチング改善法を行う．これは道路から白線に変わる場所つまりエッジの部分を下方中心部から左右に探していく．図 8 は画像の右側白線の左端を拡大した図である．これを用いて詳しく説明する．

1 つの正方形が 1 画素である．斜線の画素では黒から白に変わっている最初の画素である．つまりエッジである．この画素に注目し，左右 2 画素の色を見て，左 2 画素が「黒」，右 2 画素が「白」であると，斜線の画素を右側白線の左端と判断する．同様にして，左右の白線の両端座標を取得することができ，白線の位置と中心座標を得ることができる．全ての中心座標を結び白線を抽出する．図 9 に左右の白線を抽出した結果を示す．赤で示した線が，画像処理によって白線と判断した線である．画像の白線上に赤線が表示されており，白線の抽出ができているのがわかる．

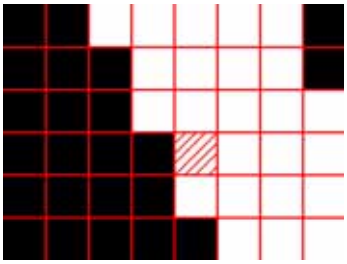

図 8　拡大画像

4.　走行線予測

4.1　最小二乗法を用いた走行線予測　白線の抽出が行えたため，それらを用いて走行線を表示する．左右の白線の中心座標を算出し，それらに最小二乗法を用いて直線をひき，それを走行線とする．最小二乗法を用いることにより走行線の方向を出すことができるため，先のフレームでの目標が求めることができる．図 10 に走行線を青で表示した画像を示す．

4.2　動画への対応　静止画では走行線を表示することができたため，静止画の連続である動画で試みた．実際に高速道路を走行中に車のダッシュボードからビデオカメラ (日本ビクター製：GR-HD1) で撮影した動画を入力として実行した．動画から静止画を入力し，それに白線抽出処理を行い，走行線の表示までを行う．入力に使用した動画は SD モードで録画したもので，解像度 320×240，フレームレート 60Hz である．動画でも問題なく走行線の表示に成功した．

図 9　左右白線の抽出画像

図 10　走行線を表示した画像

5.　システム設計

5.1　概要　前章までの画像処理は，コンピュータ上で動作するソフトウェアで実現してきた．しかし画像処理アルゴリズムをハードウェアで実装することにより，ソフトウェア処理よりも，小型・高速かつ低消費電力で行うことができる．近年，ハードウェアの進歩は目を見張るものがあり，高集積・多機能・高速・低価格なプログラマブル・デバイスである FPGA を用いることが可能となった．白線の抽出では市販のディジタルカメラで撮影した画像及び動画を用いて行ったため，ハードウェアに画像を取得するためのカメラが必要である．システム全体の構成を図 11 に示す．カメラから画像を読み込み，読み込んだ画像データに対して白線抽出処理を行い，走行線の予測を行うハードウェアを設計する．また，取得した画像及び処理後の画像を示すためにディスプレイに転送を行う[5]．

信号処理システムはカメラ制御モジュール，画像処理モジュール，データ転送モジュール から構成される．カメラ制御モジュールは，CMOS イメージ・センサからの画素値を読み取る．その画素値は，FPGA で構成される画像処理モジュールに転送され，画像処理を行う．画像処理後のデータは，データ転送モジュールにより FPGA 外部に送り出され，ディスプレイに表示する．

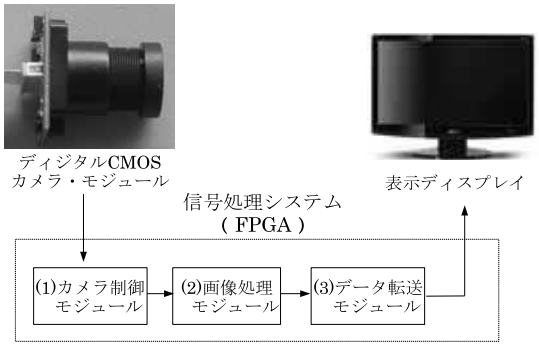

図 11　画像処理システムの構成

本研究では, FPGA として Xilinx 社の Spartan-3E を用いて画像処理回路を実現している. FPGA 開発ツールには設計入力・論理合成・配置配線・タイミング解析・FPGA への回路データのダウンロードなど, FPGA 開発の一連の作業を行える Xilinx 社の「ISE WePACK 9.1i」を使用する.

5.2 VGA ディジタル CMOS カメラ・モジュール 画像を取得するために使用したカメラ・モジュールは, カラー CMOS イメージ・センサ「OV7640」(米国 OmniVision Technologies 社) にレンズ系を付け, カメラ・モジュールとして商品化されたものである. 画素数は 640×480 ピクセルである. ディジタル画像データの出力フォーマットは, YUV(4:2:2), RGB(4:2:2) などが選択できる. 画像フレーム・レートは VGA のとき 30 フレーム/秒, QVGA のとき 60 フレーム/秒である. FPC ケーブル接続用のコネクタが実装されており, 0.5mm ピッチの FPC ケーブルを接続することにより画像データを取り出すことができる.

5.3 FPGA(Spartan-3E) 本研究で使用した FPGA は「Spartan-3E ファミリ」の「XC3S250E」であり, 大量生産される民生機器でも使われている低コスト FPGA である. XC3S250E は, 66 本のユーザ I/O と 25 万ゲート相当の論理ブロックを搭載している.

6. ハードウェア製作

ハードウェア構成のブロックを図 12 に示す. 以下, 主となる処理について説明する.

6.1 カメラ・インターフェース回路 FPGA の内部にあるカメラ・インターフェース回路は, CMOS イメージ・センサが出力した画像データをメモリ (VRAM) に書き込み, そのデータを読み出して D-A コンバータへ出力する. カメラ・モジュールの画像信号を FPGA でビット数や同期信号を付加するなどの加工をし, D-A コンバータへ出力する. その構成を図 13 に示す. カメラ・モジュールへは基準となるクロックを出力する必要があり, このクロックに同期してピクセル・クロック及び画像データが出力される. CMOS イメージ・センサからは 640×480 ピクセルの画像が 30 フレーム/秒で出力される. これをディスプレイで表示しようとする場合は, 60 フレーム/秒でアナログ RGB コネクタへ出力する必要がある. これはフレーム・レートを合わせるために, 同じ画像を 2 回出力することにした. この処理を実現するため, 1 フレームの画像すべてをフレーム・

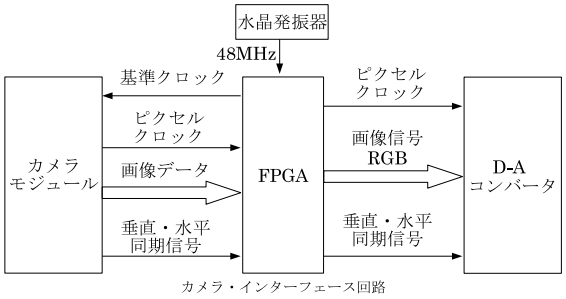

図 12 ハードウェア構成のブロック図

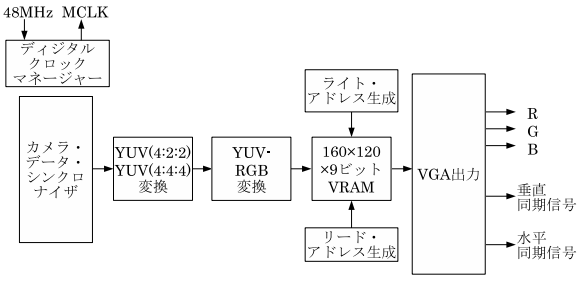

図 13 カメラ・インターフェース回路

メモリ (VRAM) へ保存する必要がある. 今回の回路では使用できるメモリは FPGA の内部の RAM しかない. RAM として使用できるサイズは,

$18 \times 1,024 \times 12 \div 8 = 27,648$ バイト

である. 640×480 ピクセルの画像を RGB 各 8 ビットで保存しようとした場合, 必要なメモリサイズは,

$640 \times 480 \times 3 = 921,600$ バイト

となる. すべてのメモリブロックを使用しても収まらない. そこで画像の解像度を 4 分の 1 に落とし, 解像度 160×120 で色のデータ RGB を各 3 ビットで保存することにした. カメラインターフェース回路を構成する主な回路について, 以下に説明する.

ディジタル・クロック・マネージャ (DCM) DCM は外部から入力されたクロック信号を分周することができる. 水晶発振器から入力された 48MHz のクロックから, 基準となる 24MHz のクロックを生成し, カメラ・モジュールの MCLK に出力する. カメラ・モジュールは, MCLK を基準として PCLK を出力する. 画像データ, 垂直同期信号及び水平同期信号は PCLK に同期して出力する. DCM は出力したクロックをフィードバックし, 外部から入力されたクロックと比較することにより, これらの時間的ずれを補正する.

YUV-RGB 変換 パソコンのディスプレイ信号は RGB の 3 原色データで表現されている. 人間の目が色の変化よりも明るさの変化に敏感な性質を利用して, 輝度情報により多くのデータ量を割り当てるのが YUV フォーマットである. 輝度信号 (Y) と, 輝度信号と青色成分の差 (U), 輝度信号と赤色成分の差 (V) の 3 つの情報で色を表す形式である. YUV のフォーマットでは, 輝度と色のデータが分離されており, このままでは人間にとって正しい色彩で画像を表現できない. 従って, 輝度と色のデータを人間が知覚することのできる光の 3 原色 (RGB) に変換する.

YUV から RGB への変換は, 以下の式により表される.

R = 1.164(Y-16) + 1.596(V-128)

G = 1.164(Y-16) - 0.392(U-128) - 0.813(V-128)

B = 1.164(Y-16) + 2.017(U-128)

固定小数点で計算するため, YUV の係数を 4096 (12 ビット左シフト), 定数項を 16 倍 (4 ビット左シフト) し, 各項をまとめ, 係数を 16 ビットの 16 進表示にすると以下の通りになる.

R = 0x12a0×Y + 0x18d5×V + 0xf211

G = 0x12a0×Y + 0xf9ba×U + 0xf2fe×V + 0x087a

B = 0x12a0×Y + 0x2046×U + 0xeeb3

符号なし 8 ビット × 符号つき 16 ビットの演算を行う乗算器を 9 個用意し, YUV と各係数を乗算する．これらの結果に, 定数項を 256 倍（8 ビット左シフト）した値を加算する．その結果を 12 ビット右シフトして 8 ビットの RGB を求めている．

ライトアドレス生成　カメラモジュールが出力する垂直同期信号及び水平同期信号から, 入力データの有効画像をカウントし VRAM へ書き込むアドレスを生成する．横方向, 縦方向ともに 3 ピクセル間引きして VRAM へ書き込んでいる．

160×120×9 ビット VRAM　1 フレームを格納する RAM. RGB 各 3 ビットをひとまとめにして書き込み, 読み出しを行う．デュアルポート RAM になっているため, 書き込みと読み出しにはそれぞれ別のクロックを使用することができる．

リード・アドレス生成　出力に使用するクロックをカウントし, VRAM から読み出すアドレスを生成する．書き込みとは逆に, VRAM から読み出された 1 ピクセルのデータを, 横方向, 縦方向ともに 4 ピクセルに拡大する．

VGA 出力　出力に使用するクロックをカウントした値から, アナログ RGB のコネクタへ出力する水平・垂直同期信号を生成する．RGB 各 5 ビットずつ設定する必要があるため, VRAM から読み出した RGB 各 3 ビットと下位 2 ビットに 0 を設定したデータを出力する．

6.2　アナログ RGB 表示出力回路　アナログ RGB 信号は, パソコン表示装置の標準インターフェース信号として使われている．本研究では 15 ピン・シュリンク D サブ・コネクタを用いて RGB 信号を出力する．アナログ RGB 出力信号は, 3 原色アナログ画像信号及び水平同期信号, 垂直同期信号の 5 本で構成されている．この 5 種類の信号でパソコン用表示装置にカラー画像を表示する．FPGA から出力されるディジタルの画像データを 3 チャンネル 8 ビット高速ビデオ D-A コンバータ「ADV7125」でアナログ画像信号に変換する．FPGA 基板のピン数の制約により, RGB 各 6 ビットとする．64×64 ピクセルのタイルを並べている．水平・垂直カウンタの重みのある下から数えて第 6 ビット目の内容が変化するごとに色を変化させている．水平カウンタの第 6 ビット目と垂直カウンタの第 6 ビット目とで排他的論理輪をとって, その結果で色を決定する．市松模様を表示させた結果を図 14 に示す．

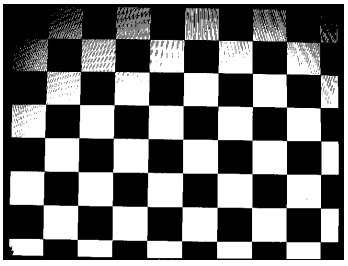

図 14　市松模様の表示

6.3　実行結果　製作したハードウェアでカラー CMOS カメラから取り込んだ画像を VGA 表示した画面を図 15 に示す．VGA の解像度を 4 分の 1 に間引いて表示しているため, 画像の粗さが目立つ．また, メモリ容量の制約から D-A コンバータへの出力も RGB 各 3 ビットしか使用していないため, 色の表現にも限界がある．しかし, カメラからの画像入力ができ, ディスプレイに表示することには成功している．

図 15　回路で表示した画面

6.4　2 値化　カラー CMOS カメラから取り込んだ画像の 2 値化を行う．RGB 形式の画像に 2 値化処理を行いディスプレイに表示する．パソコンのディスプレイに図 16 の静止画を表示する．その画面を CMOS カメラで撮影し, 2 値化処理を行いディスプレイに表示する．ディスプレイに表示した画面を図 17 に示す．白線以外にも原因不明のノイズがあり, 白線抽出処理には適切ではない．

次に YUV-RGB 変換を行う前の輝度（Y）データのみを使い, 2 値化を行った画面を図 18 に示す．こちらは道路上には特にノイズといったものがなく 2 値化できている．しかし, 輝度データのみであるため, いきなり光が強く当たったりすると真っ白になってしまったりするといった欠点がある．これらのどちらを用いるか今後検討を行い, 改善していく必要がある．

図 16　2 値化処理に用いる入力静止画

図 17　2 値化を行った画面 (RGB 形式)

図 18　2 値化を行った画面 (輝度データのみ)

7.　結論

静止画を用いて白線の抽出をするため OpenCV を用い, 静止画像から白線の抽出をするアルゴリズムを決定した. 続いてそれらが動画でも対応できるかを確認し, 動画での白線抽出に成功した. さらに白線抽出をすることにより車が走る走行線を予測するアルゴリズムを決定した. OpenCV でレーンキープ走行ができることが確認できたため, 処理速度が速いハードウェアで実行できるようにシステムの設計を行った. ディスプレイに表示でき, カメラからの画像を出力できるようにした. 白線抽出の前処理の段階である 2 値化を行った. まだ適切な 2 値化方法が決定していないので, 今後検討していく.

今回設計した回路でのカメラ画像では画像は粗く, 解像度も小さくしていた. 今後はメモリを節約し, 解像度をあげ精度をあげていく. また, 2 値化処理において最適な 2 値化処理を決めていく. さらに 2 値化画像から白線を抽出し, 走行線を算出するプログラムを組みこんでいく.

参考文献

(1) 西田泰: 『医学と工学からみた交通安全対策』日本交通医学工学研究会, Vol.2007, p27.

(2) 奈良先端科学技術大学院大学 OpenCV プログラミングブック制作チーム: ”OpenCV プログラミングブック”, 株式会社毎日コミュニケーションズ, 2007.

(3) JAE S.LIM: ”TWO-DIMENSIONAL SIGNAL AND IMAGE PROCESSING”, PRENTICE HALL, 1989.

(4) E.R.DAVIES: ”MACHINE VISION”, MORGAN KAUFMANN, 2004.

(5) 鳥海 佳孝, 田原 迫仁治, 横溝 憲治 共著: ”実用 HDL サンプル記述集”, CQ 出版社, 2002.