卒業論文

ＦＰＧＡボードを用いた液晶用ガラスの欠損検出画像処理の高速化（Ⅲ）

氏名：松山　圭輔
学籍番号：2260070087-5
指導教員：山崎　勝弘　教授
提出日：平成 23 年 2 月 18 日

立命館大学　理工学部　電子情報デザイン学科

内容梗概

FPGA ボードとは、回路構成の書き換えができる集積回路であり、CPU によるソフトウェア処理と比べて高速な処理を実行させることができる。信号処理部分にパソコンのような大きさを必要としないため、検査装置などの小型化に適している。本論文では、使用する画像処理アルゴリズムの説明、CPU ソフトウェアによる画像処理を実行、比較・考察、FPGA ボードに実装するためのモジュールの説明と今後の課題について述べる。

目次

図目次

表目次

１．はじめに

検査装置、制御装置や計測装置は、一般にセンサーなどからの入力信号に対して何らかの演算処理を行い、その結果に基づいて動作する。画像処理システムならば画像センサーの信号を処理して被写体の形状などの良否判定を行う。

このような装置の多くは、パソコン上で動作する CPU ソフトウェアによって信号処理を行っている。近年の CPU の処理能力は目を見張るほどの急成長を遂げているとはいえ、信号処理の内容によっては十分とはいえないものもある。CPU によるソフトウェア処理でも十分ではあるが、瞬時に結果が必要だったりする場合などでは CPU によるソフトウェア処理では十分な性能や速度が得られないことも少なくない。また、パソコンを用いてソフトウェア処理を行う場合、パソコンのサイズが装置の小型化の妨げになっている。

FPGA（Field Programmable Gate Array）ボードは回路構成の書き換えができる集積回路であり、CPU によるソフトウェア処理に比べて高速な処理が可能である。そして、FPGA ボードで信号処理を行えば、信号処理部分にパソコンのような大きさを必要とせず、装置の小型化を図ることができる。

本研究では、FPGA ボードを用いて、液晶用ガラス欠損検出の画像処理の高速化を行う。液晶用ガラスの欠損検出の画像処理の高速化をテーマにした理由として、現代の市場が背景にある。液晶用ガラスは、今主流の液晶テレビや一人一台になりつつあるパソコン、今後ますます普及する事が予測されるスマートフォンなどに使われており、今後もその需要の高さが伺える。その大きな需要に対応するために液晶用ガラスの生産スピードも向上が望まれる。どんな製品にも言えることだが、生産すると同時に必要なのが検品である。製造した製品すべてが優良品であるわけがなく、中には傷のついたものも含まれている。そのちょっとした傷は、その後製品に大きな影響を与えかねない。そういった問題を回避するためにも検品は重要であり、正確に検品を行なう必要がある。しかし液晶用ガラスの検品では大量の画像データを処理するため、高速に処理できるシステムが必要である。

そこで、精度の高い検品を行うことのできる、かつ高速に処理するのに適した FPGA ボードを本研究で使用することに決めた。

本論文では、２章で液晶用ガラス欠損検出のためのアルゴリズムを述べ、3 章で CPU によるソフトウェア処理について述べ、４章では FPGA ボードによる処理について述べ、5 章では考察と比較を述べる。

２．液晶用ガラス欠損検出のための画像処理アルゴリズム

２．１　TDI（Time Delay Integration）

TDI（Time Delay Integration）とは、同じ画像の画素をずらして撮影を繰り返し、その共通部分を重ね合わせ平均値を取ることで、ノイズの影響が小さい画像を得る手法である。例として、図１に画像の画素を数値化したものを示す。

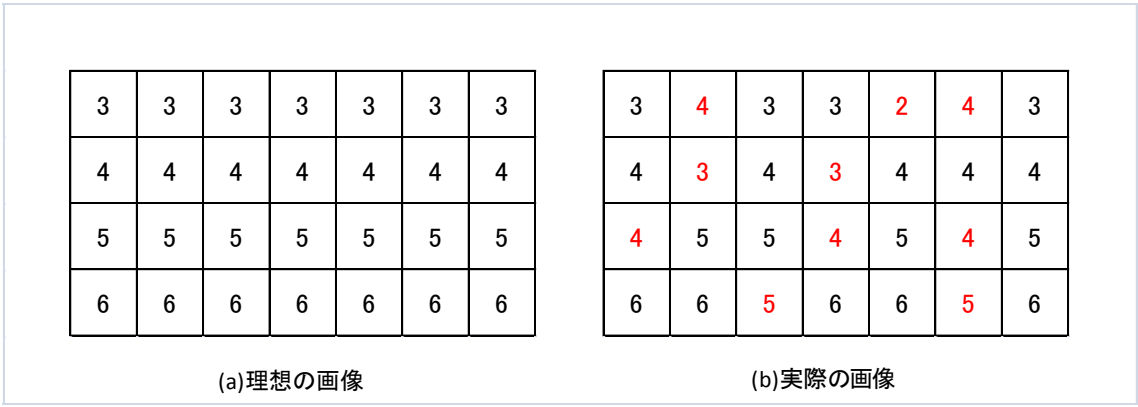

図１：画素の数値化

図１の左の画像が理想の画像なのだが、実際は右の画像のようにノイズ（赤い数字の部分）が入ってしまう。このノイズの部分をなるべく左の画像の状態に近づけるために画素の平均化を行う。画素の平均化のために、撮影した画像の画素を１画素ずつずらした画像を複数枚用意し、共通部分を重ねる。その過程を図２に示す。

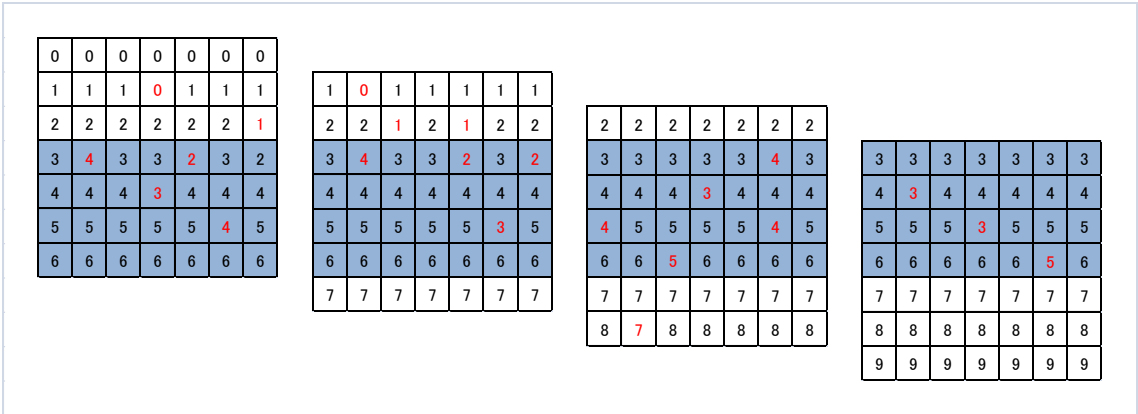

図２：画素の平均化

図２を見て、共通部分は間の４行が共通しているので、この４行の数値をそれぞれ平均して出す。

3	3	3	3	3	3	3
4	4	4	4	4	4	4
5	5	5	5	5	5	5
6	6	6	6	6	6	6

(a)理想の画像

3	3.5	3	3	2.5	3.25	2.5
4	3.75	4	3.5	4	4	4
4.75	5	5	4.5	5	4	5
6	6	5.75	6	6	5.75	6

(b)平均化を行った画像

図 3：TDI

図 3 の(b)を見ると、図 1 の(b)と比べて理想の画像である図 3 の(a)により近付いたことが分かる。このように、画像を重ね合わせ、平均値をとることによりノイズを減らすことができる。画像を読み込んで前処理として TDI を行うことにより、よりスムーズに画像処理ができる。

２．２　ラプラシアンフィルタ

ラプラシアンフィルタとは、画像中に含まれる物体の輪郭部分（エッジ）を抽出するフィルタである。画像に対して 2 次微分をするフィルタで、2 変数関数 f(x,y)のラプラシアンは、偏微分を使って(1)の式で表せる。

$$L(x,y) = \frac{\partial^2}{\partial \mathrm{x}^2} f(x,y) + \frac{\partial^2}{\partial \mathrm{y}^2} f(x,y) \qquad (1)$$

ディジタル画像のラプラシアン L(x,y)は、ラディアントをもう一度微分したもので、(2)のような式で表わされる。

$$\begin{aligned} L(x,y) = {} & 4f(x,y) \\ & - \{f(x,y-1) + f(x,y+1) \\ & + f(x-1,y) + f(x+1,y)\} \end{aligned} \qquad (2)$$

ラプラシアンの計算に用いる微分オペレータを(3)、(4)に示す。

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad (3)$$

(3)の係数は 4 近傍ラプラシアンフィルタの時のマスクパターンとして使い、上下両隣の 4 近傍の画素の差分を取っている。

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad (4)$$

(4)の係数は 8 近傍ラプラシアンフィルタの時のマスクパターンとして使い、上下両隣＋斜め方向の画素の差分を取っている。
図 4 に例として、8 近傍ラプラシアンフィルタを用いた時の処理の流れを示す。

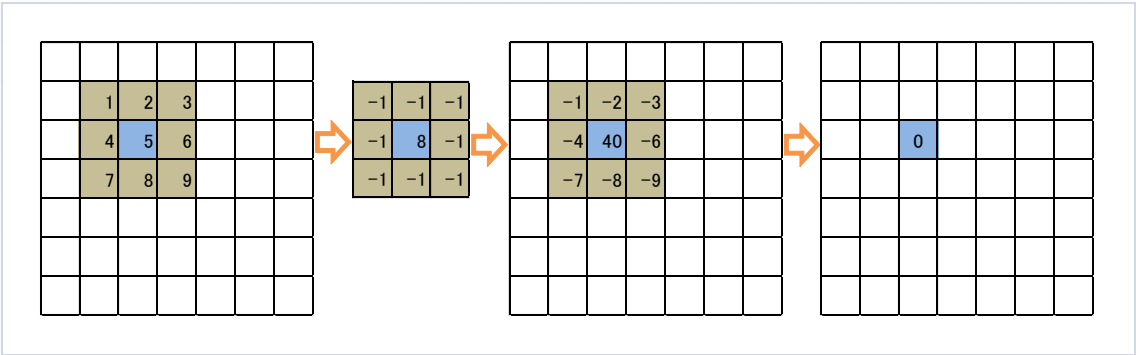

図 4：ラプラシアンフィルタ

本研究では 8 近傍ラプラシアンフィルタをマスクパターンとして使用する。

２．３　ラベリング

ラベリングとは、つながっている画素（連結成分）に同じラベル（番号）をつけ、異なった連結部分には異なった番号を付ける処理のことであり、2 値画像処理では非常に重要な処理である。図 5 にラベリング処理の過程を示す。

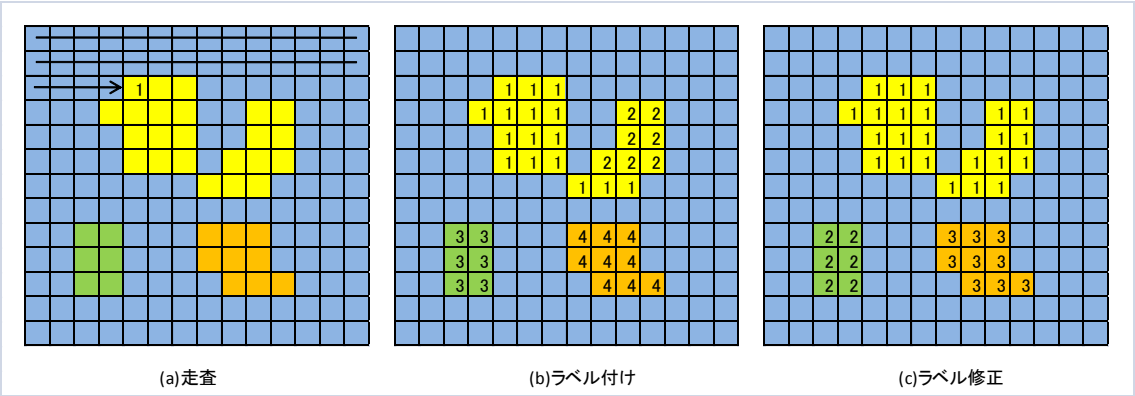

図 5：ラベリング

まず、図 5 の(a)のように左上から走査を開始し、ラベルが付けられていない画素を見つけたら新しいラベルを付ける。この時、上、左、左上、右上の画素を見て、違うラベルが付けられていた場合はそのラベルと同じラベルを、今つけようとしている画素につける。
左上から右下まで操作が完了し、全てのラベル付けが終わった画像が図 5 の(b)である。ここで注意すべきなのは、ラベル 1 とラベル 2 が連結している点である。ラベリングとは先に言ったように、連結画素にはすべて同じラベルが付けられていなければならないのだが、連結部分で違うラベルが付けられている。しかしラベルを付ける際に上、左、左上、右上の 4 つの画素を見てはいるが、4 つの画素の中にラベルが複数存在した場合に何か特別な処理をしろという命令はいれていないので何回試してみてもラベル 1 とラベル 2 が連結しているということに変わりはない。そこで、ルックアップテーブルを使う。ルックアップテーブルを使うことによって、ラベルを付ける際に注目する上、左、左上、右上の画素のラベルが複数存在した場合に、最小のラベルを割り振ることができる。
この処理により個々の連結成分に分離することができ、各連結成分の特徴を調べることができる。

３．CPU ソフトウェアによる演算と実験

３．１　Cプログラミングによる記述

本研究では使用する画像処理アルゴリズムの理解をより深めるために、まず C プログラミングによる記述を行う。CPU ソフトウェアによる演算処理と FPGA ボードによる演算処理の違いについての比較は 5 章で行う。

画像処理の流れを図 6 に示す。

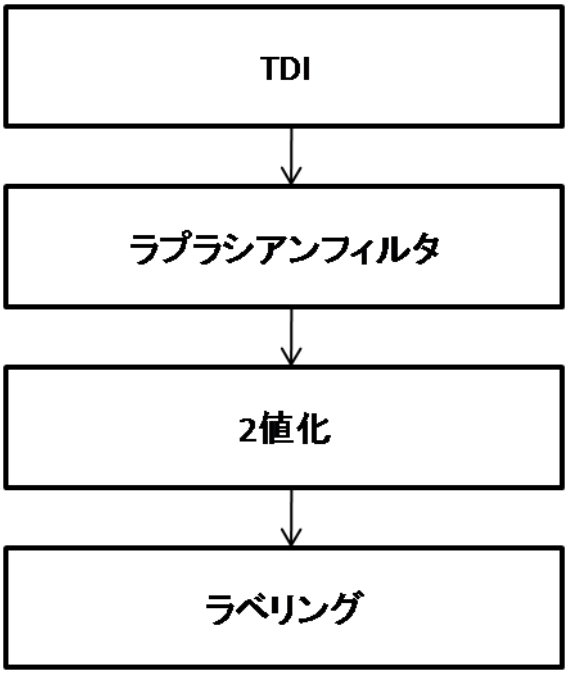

図 6：画像処理手順

まず、元画像に対して TDI を複数回行い、ノイズの除去された画像にラプラシアンフィルタをかける。エッジのかかった画像を見やすくするために 2 値化を行い、最後にラベリングを行い欠損の検出を行う。

３．２　実験環境

C プログラミングで記述したものを実行させる環境を下に示す。

・マシン名　　：Platini
・機種　　：OPTIFLEX960
・CPU(Hz)　　：Core2Quad　2.66G
・RAM（GB）　：4.0

３．３　実験結果

画像処理を行う元画像と、元画像に TDI 処理だけを行ったものを図 7 に示す。それぞれの画像は重ね合わせた画像の枚数を変えてある。

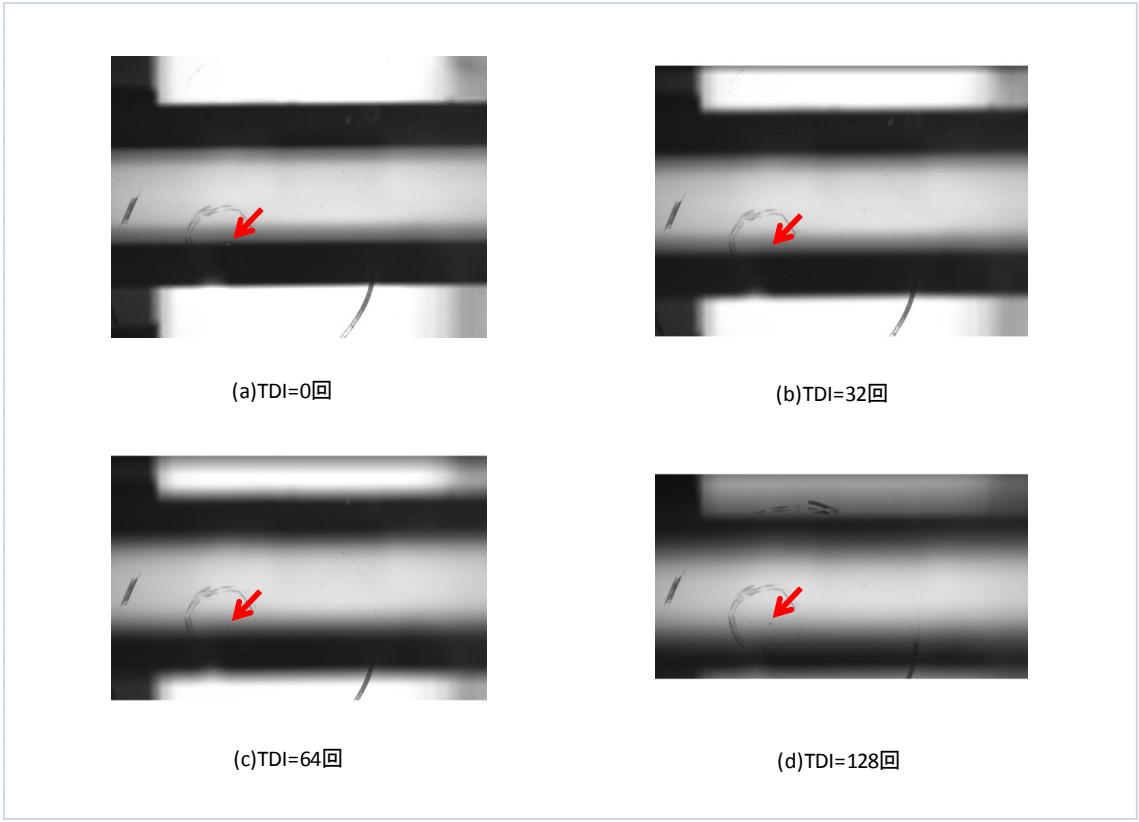

図 7：TDI の実行結果

表 1：TDI 処理実行時間

TDI(回数)	32	64	128
処理時間(sec)	0.62	1.481	2.469

図 7 の(a)と(d)を比較すると分かりやすいが、明らかに画像の縦幅がせまくなっている。これは、画像を重ね合わせ平均値を取る際に、画像の重ね合わせる枚数が多いほど、画像全体の共通部分が少なくなるためである。

図 7 の(a)と比べて(d)の 128 枚重ね合わせたものの方が鮮明に見える。これは平均値をとることで、散らばっていた強いノイズが周りの画素と同化するからである。

では次に、画像処理を行う元画像と、元画像に TDI を行いラプラシアンフィルタ（8 近傍）をかけたものを図 8 に示す。

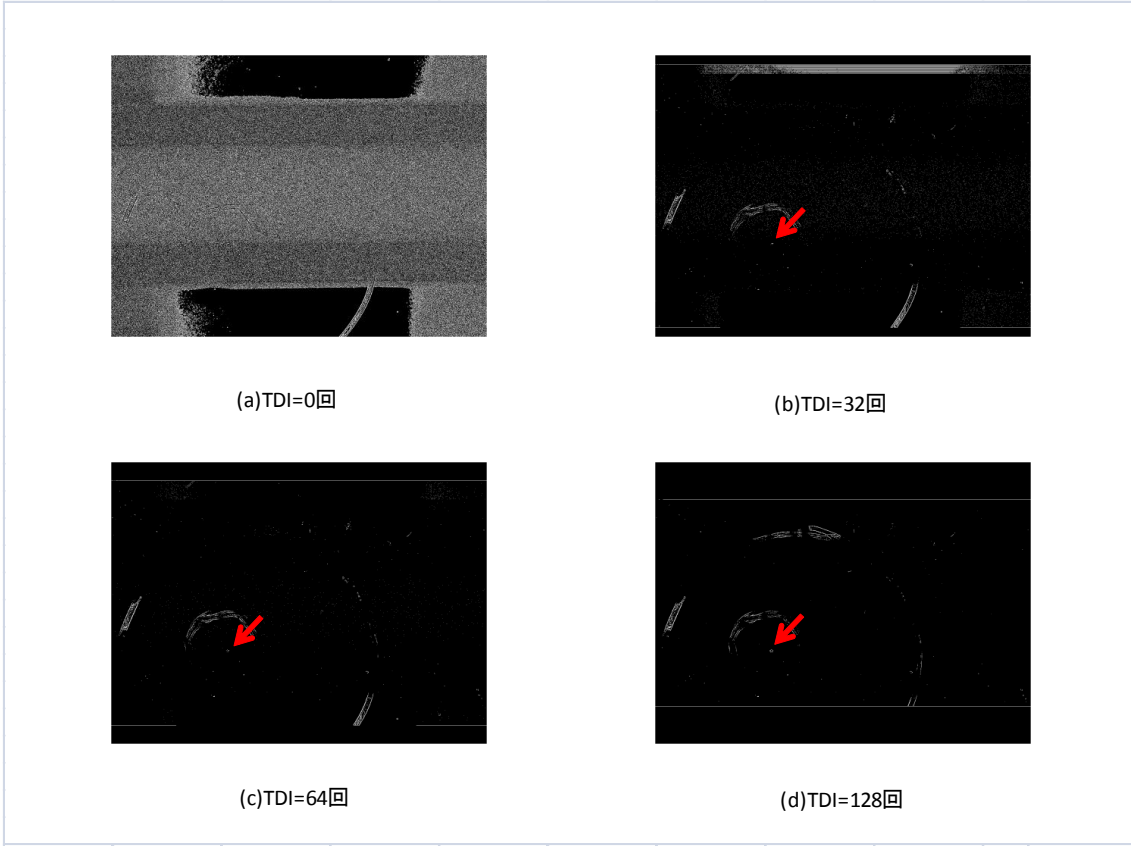

図 8：ラプラシアンフィルタの実行結果

表 2：ラプラシアンフィルタ処理実行時間

TDI(回数)	0	32	64	128
処理時間(sec)	0.14	0.14	0.125	0.125

図 8 の(a)の元画像をラプラシアンフィルタに通してみたが、ノイズがひどくてどこが傷かわからない状態である。それに比べて図 8 の(b)TDI32 回、(c)TDI64 回、(d)TDI128 回は綺麗に出力されている。見ての通り、TDI 処理を行う回数が多くなるにつれて画像のノイズが減っていっていることが見てわかる。同じ処理を行うにしても、ノイズを減らす TDI 処理の重要性が分かった。

次は、画像処理を行う元画像と、元画像に TDI 処理を行いラベリングをしたものを図 9 に示す。

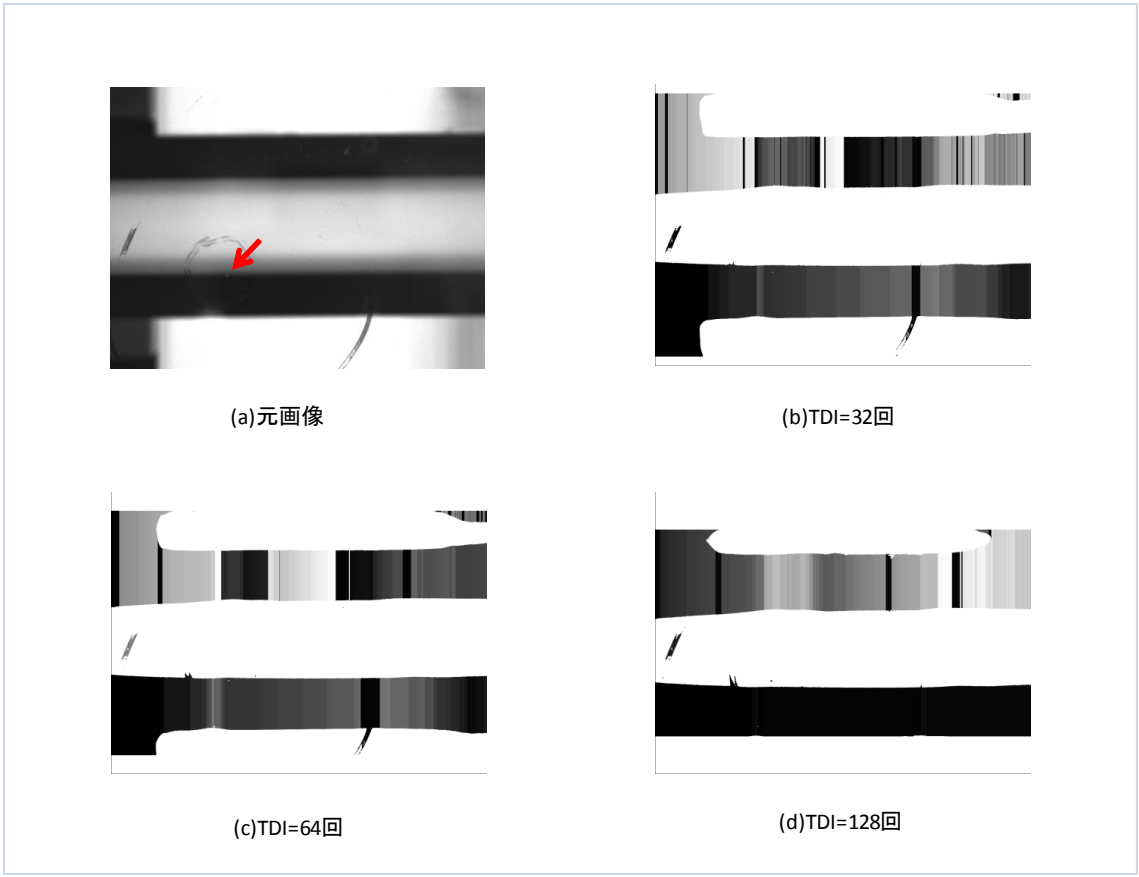

図 9：ラベリングの実行結果

表 3：ラベリング処理実行時間

TDI(回数)	32	64	128
処理時間(sec)	5.4422	2.844	2.063

図 9 の(a)は元画像そのままである。元画像にもラベリング処理を行いたかったが、ノイズがひどく計算量が多いためか、最後まで処理をすることができずエラーとなってしまったからである。しかし、図 9 の(b)、(c)、(d)を見てみても、画像がどういう状態なのかわからないものとなっている。図 8 の(b)、(c)、(d)と図 9 の(b)、(c)、(d)を見比べると、元は同じ画像なのに出力された画像が別物になっているのがわかる。これによって、TDI 処理でノイズを減らすだけでは満足の行く結果は得られないことが分かった。

最後に、画像処理を行う元画像と、元画像に TDI を行い、ラプラシアンフィルタをかけ、ラベリングをしたものを図 10 に示す。

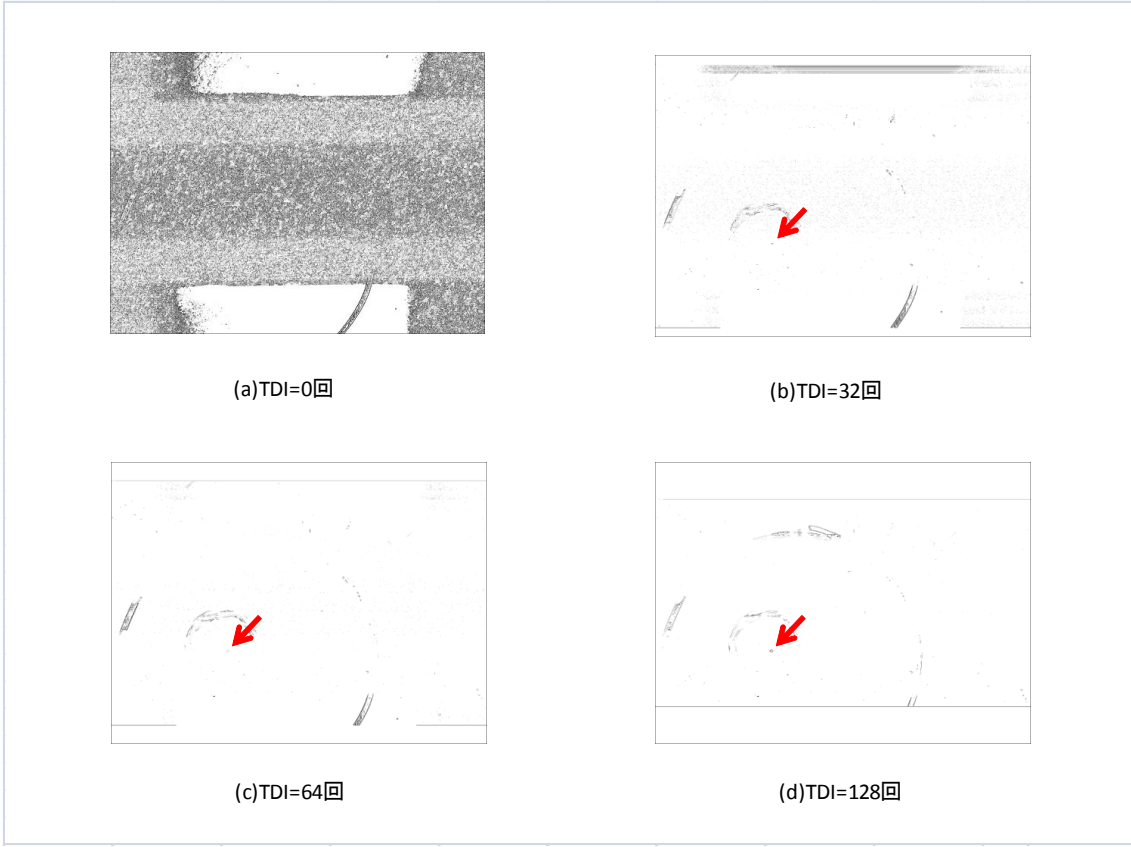

図 10：ラプラシアンフィルタ、ラベリングの実行結果

表 4：ラプラシアンフィルタ、ラベリング処理実行時間

TDI(回数)	0	32	64	128
処理時間(sec)	79.313	6.469	5.422	2.797

図 10 の(a)が元画像に一連の処理を行った画像で、(b)、(c)、(d)が TDI 処理を行った後に一連の処理を行った画像となる。見比べて分かるように、だいぶ傷を見やすく処理することができた。図 7、8、9 で行ったそれ単体の処理よりも、図 10 にあるすべての処理を行った画像の方がわかりやすいものとなっている。

４．Verilog HDL による記述と FPGA ボードを用いた実験

４．１　Verilog HDL による記述

3 章では CPU ソフトウェアによる画像処理を行った。4 章では FPGA ボードによる処理演算を行っていく。画像処理手順は、3 章で示した図 6 と同じ手順である。

４．２　シミュレーション環境と FPGA ボードでの実験環境

Verilog HDL で記述したものをシミュレーションさせる環境は、3 章で記述したマシンと同様である。FPGA ボード上に実装するにあたり、開発環境として Xilinx 社の EDK を使用した。また FPGA ボードには、同じく Xilinx 社の MicroBlaze を内蔵した Spartan3A Starter Kit Board を使用する。

４．３　画像処理モジュール

Verilog HDL での記述に先駆けて、画像処理全体のモジュール図を構成した。図 11 に画像処理全体のモジュール構成を示す。

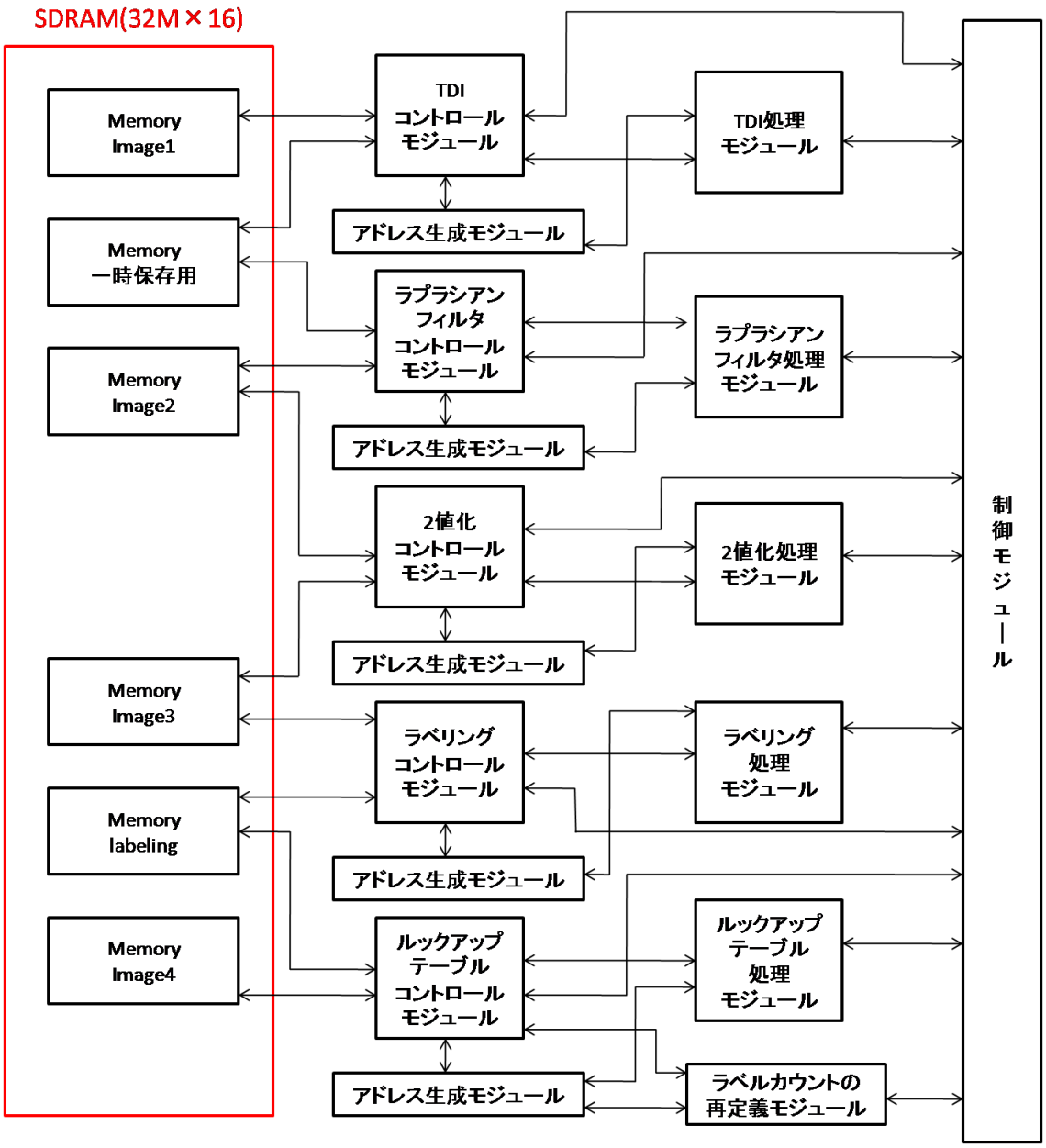

図 11：全体のモジュール構成

画像は、画像データが大きいため 32M×16 個の合計 512M 格納できる SDRAM に格納する。画像処理を行う各モジュールに対して制御を行うモジュールをつけ、さらに全体を制御するモジュールをつける。図 11 の TDI 処理モジュール、ラプララシアンフィルタ処理、2 値化処理モジュール、ラベリング処理モジュールはそれぞれ図 13、図 15、図 16、図 17 で具体的に示す。

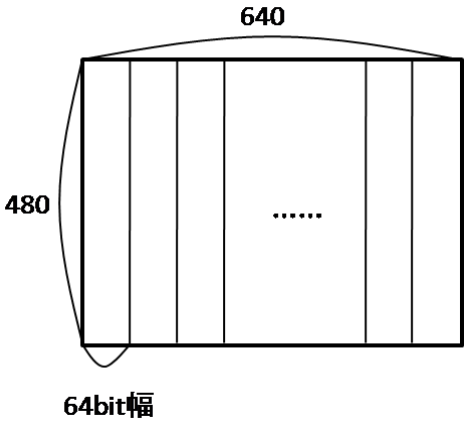

図 12：640×480 の画像

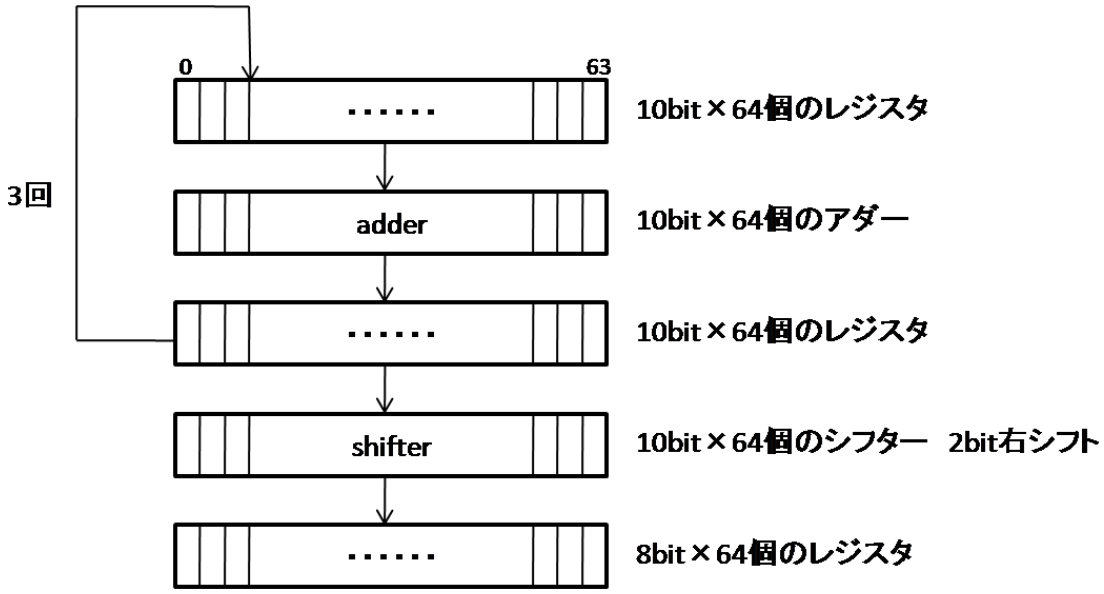

図 13：TDI 処理モジュール

例として、図 12 の横 640 縦 480 の画像を 4 枚用意して処理の内容を追っていく。まず、元画像を縦に 64bit 幅に区切り、1 行目の各値をそれぞれ図 13 の一番上のレジスタに格納する。次に、アダーにさきほど格納したものをそれぞれ入れ加算する。そしてこの処理を TDI する(回数－1)回行う、今回は 3 回行い、3 回の加算が終了したら 2bit シフトさせ除算を行い、平均化された値をレジスタに格納する。1 行目の平均化が終わったら 2 行目以降も 1 行目と同様の処理を行う。1 列目の処理が終わったら 2 列目以降も処理を行い、画像全体の平均化を行っていく。

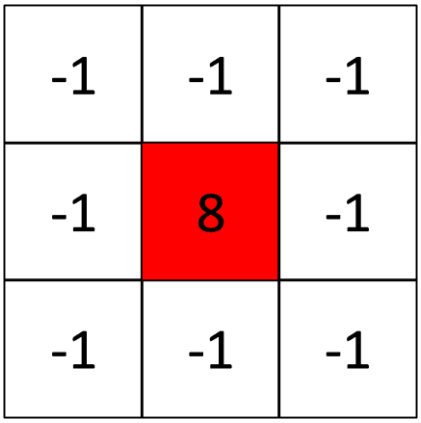

図 14：マスクパターン

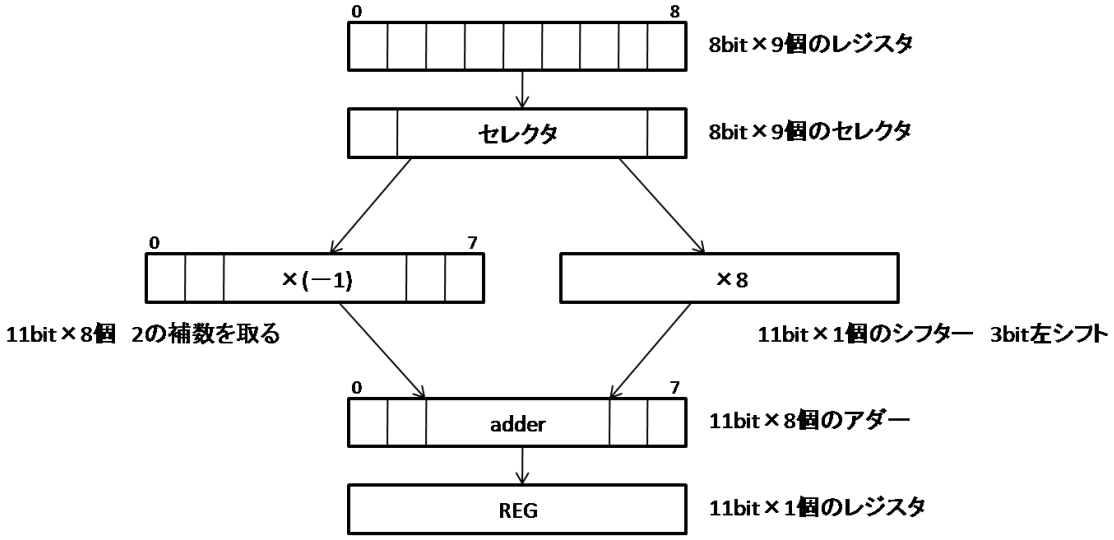

図 15：ラプラシアンフィルタ処理モジュール

ラプラシアンフィルタ処理は、画像の左上から右に向かって処理を行い、端まで行ったら次の行に移り、画像の右下までの順で処理を行う。図 14 が今回使用するマスクパターンである。これを使ってラプラシアン処理モジュールの説明を行う。まず、平均化された画像の 3 行 3 列分の画像データをそれぞれレジスタに格納する。そして、セレクタで(−1)をかける画素か 8 をかける画素かを分けて乗算を行う。最後に乗算された値すべてを加算し、レジスタに格納する。

図 16：2 値化処理モジュール

図 15 のラプラシアンフィルタ処理モジュールで生成された画像データを、閾値を用いて白(255)か黒(0)の二つに分ける。

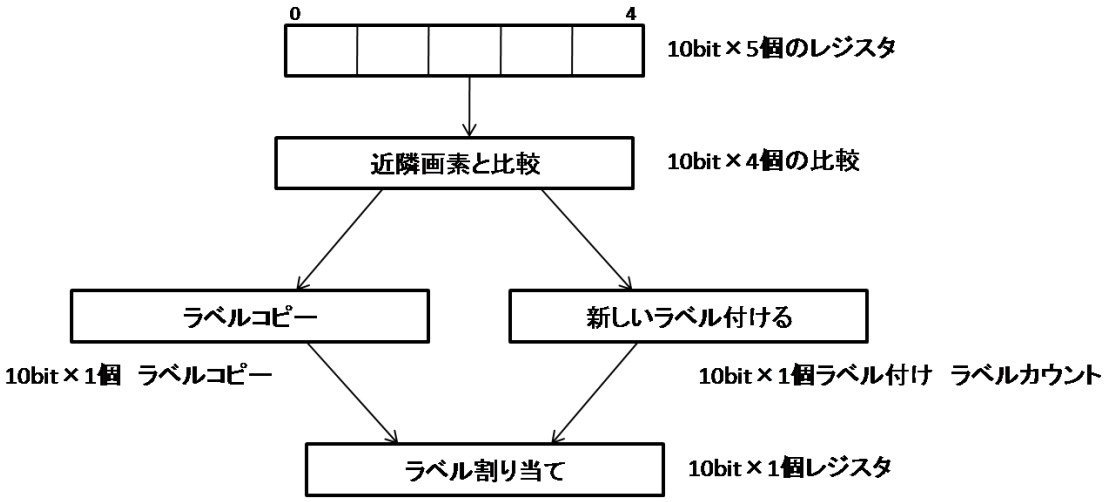

図 17：ラベリング処理モジュール

図 17 の一番上のレジスタ数が 5 個なのは、ラベル付けを行う際にラベルを付ける画素、上、左、右、下の 5 つのデータが必要なためである。

５．　考察

CPU ソフトウェアによる処理演算で、TDI 処理を行うと行わないとでは後々の処理結果に多大な影響を及ぼすことが分かった。TDI 処理を行う最低限の回数として 128 回が理想なのではないかと思う。精度の観点から見れば TDI＝64 回でも問題はないのだが、時間の観点から見れば TDI＝64 回と TDI＝128 回とでは倍近く処理時間が違う。本研究では、画像処理の高速化を狙っているので、TDI 処理は 128 回が必要最低限の回数であると思う。128 回以上の処理は本研究では行っていないが、その理由として精度的な観点から見ればこれ以上ノイズを除去する必要も見られないし、枚数が増えると TDI 処理にも時間がかかるようになるからである。今回使用する画像処理アルゴリズムの中で、時間のかかりそうな TDI 処理やラベリング処理を FPGA ボードで行うことにより、より速く処理できる。これにより、CPU ソフトウェアで処理するよりも FPGA ボードで処理した方が処理時間を速くできることが考えられる。

ラプラシアンフィルタについては、単純な積和の計算しか行っていないが、画像サイズが大きくなればなるほど、高速に処理できる FPGA ボードの方が処理時間を小さくできると思われる。ラベリングにかかる時間は、TDI の回数に比例していることが分かる。つまり、TDI の回数が少なければノイズが多いので処理時間が増え、TDI の回数が多ければノイズが少ないので処理時間は減少する。表 1 の TDI 処理実行時間から、TDI の回数が多い時の処理時間は、画像処理全体を高速化させる上で無視できない数字となっている。しかし、TDI の回数を少なくするとラベリングにかかる時間も増えるので、読み込む画像によって TDI 処理の回数を変える必要がある。本論文で使用したガラス基板の欠損画像の場合は、先に言ったように TDI＝128 回処理させるのが妥当であると言える。

６．おわりに

本論文では、株式会社ケーデーイーとの産学協同研究として、TDI を用いた画像処理の実験と CPU ソフトウェアによる画像処理演算、FPGA ボード上に実装させるためのモジュールを設計した。本研究を通して、画像処理の原理・仕組みをソフトウェアの観点から理解することができた。

今後の課題として、ソフトウェア面では本研究で設計したモジュールを Verilog HDL で記述し、シミュレーションを行い速度の向上を目指すことがあげられる。ハードウェア面では FPGA ボードの更なる理解と、本研究で設計したモジュールを FPGA ボードに実装することがあげられる。

謝辞

本研究の機会を与えてくださり、ご指導を頂きました山崎勝弘教授に深く感謝いたします。また、本研究に関して研究に必要なデータを頂いた株式会社ケーデーイーの西田氏と三宅氏をはじめ、様々な面で貴重な助言や励ましをくださった研究室の皆様に深く感謝します。

参考文献

[1]小林優：入門 Verilog　HDL 記述ハードウェア記述言語の速習＆実践、CQ 出版、2009 年

[2]大川内隆郎、大原竜夫：かんたん C 言語、株式会社技術評論社、2010 年

[3]井上誠揆喜：C 言語で学ぶ実践画像処理、オーム社、2008 年

[4]貴家仁志：ディジタル画像処理　画像処理信号入門、昭晃堂、2008 年

[5]内村圭一、上瀧剛共：実践画像処理入門、培風館、2007 年

[6]松崎祐樹：マハラノビス距離を用いたガラス検査用画像判別の実現、平成 18 年度情報処理学会関西支部　支部大会講演論文集　C-09　p.187-190、2006 年

[7]松崎祐樹：マハラノビス距離を用いた画像判別とラベリングの高速化の実現、立命館大学理工学部電子情報デザイン学科卒業論文、2006 年