平成 24 年度　修士論文

イメージ付きカラー化二次元コードの構成と読み取り特性の研究

平成 25 年 2 月 7 日

サムレットウィット　ダムリ

福岡工業大学　大学院工学研究科

情報通信工学専攻　若原研究室

イメージ付きカラー化二次元コードの構成と読み取り特性の研究

サムレットウィット　ダムリ

福岡工業大学　大学院工学研究科　情報通信工学専攻

あらまし　現在，インターネットや携帯電話の普及に伴い QR コードが世界中に広まっており，各種用途に QR コードが使われている．二次元コードの代表である QR コードは白黒 2 値のドット表示であるためデザイン性が良くないので，ドットをカラー化して見栄えを良くする方法が用いられる．しかし，ドットの色を変更したり見栄えを良くするためにイメージを重畳したりするとイメージ損傷になり QR コードの読み取り特性に影響を及ぼす．また，紙に印刷したりしてディスプレイに表示した QR コードの場合には，カメラで撮影する際の照明条件および表示条件によって，QR コードの読み取り特性が異なる．

本論文では，通常の QR コードにドット絵などのイメージを重畳し白黒 2 値からカラー化二次元コードを作成するカラー化 QR エディタを構成するとともに，QR リーダとして現状のスマートフォンなどに用いられている QR デコーダをそのまま用いた場合に，カラー化する際のパラメータによってどの程度正常に認識できるのか実験により読み取り特性を評価する．

キーワード　二次元コード，QR コード，誤り訂正能力，カラー化，イメージ

2013 年 3 月 10 日

Configuration and Reading Characteristics of Two-Dimensional Color Code with Images

Damri SAMRETWIT

Graduate School of Information and Communication Engineering
Fukuoka Institute of Technology

Abstract In recent years, QR code is widely used in the accessing to the Internet by cellular phone. As the design of this code is a simply black and white dotted symbol, it is very simple and insipid. In order to improve this weakness, a new color two-dimensional QR editor with images has been developed.

In this paper, we have studied the structure of the QR code, the way of multiplexing the dotted images and colorization method. In order to improve the QR reading characteristics, the several image multiplexing methods are proposed. The experiments are tried and the reading characteristics of the two-dimensional colorized code are measured by using the existing common QR decoder. The colorization method of the two-dimensional codes is evaluated.

Keyword Two-dimensional code, QR code, Error correction capability, Colorization, Image

March 10th, 2013

目次

第 1 章　序論

1.1 QR コードの概要

図 1.1 QR コード

QR コード[1] （QR コード® は，株式会社デンソーウェーブの登録商標である）は二次元コード[2]の一種であり，Quick Response コードの略である．名前の通り高速で読み取れることを目的にデンソーウェーブ（開発当時は株式会社デンソーの一部門）が開発したものである．QR コードは，縦，横二次元方向に白と黒の明暗パターン情報を持つことで，一次元のバーコードよりも記録できる情報量を飛躍的に増加させたコードである．

QR コードの主な仕様を表 1.1 に示す．QR コードのシンボルを構成する最小の単位のセルをモジュールと言い，モジュールの大きさは型番（バージョン）によって決定される．このシンボルのバージョン（型番）には， 1 型～40 型までの 40 種類がある．1 型は 21×21 モジュール，2 型は 25×25 モジュール，最大 40 型は 177×177 モジュールというように，型番が一つ上がる度に一辺につき 4 モジュールずつ増加する．

QR コードには，汚れや破損などがあっても正しく読み取れるように誤り訂正符号を用いており，データコード語に対して修正するために付けられる誤り訂正能力として，下記の 4 レベルがある．

レベル L：コード語の約 7%が復元可能
レベル M：コード語の約 15%が復元可能
レベル Q：コード語の約 25%が復元可能
レベル H：コード語の約 30%が復元可能

さらに，QR コードを読み取り易くするために行う処理としてマスク処理があり,そのパターンは 8 種類用意されている．その中で白である“明”モジュール数と黒である”暗”のモジュール数を均一化し，画像の高速読み取り処理の障害となるパターンの発生が抑えられるマスクを採用することになっている．マスク処理では，符号化領域のビットパターンとマスク処理パターンを XOR（排他的論理和）する．

図 1.2 は，白と黒の正方形の格子状に配置されているバージョン 3 の QR コードの例を示す．QR コードには，すべての方向の高速読み取りを有効にするために，3 つのコーナーに配置されたシンボ

ルの位置を示すための 3 つのファインダーパターン（位置検出パターン）がある．

従来のバーコードは，せいぜい英数字を 20 桁程度の情報量しか表せなかったが，QR コードは，次に述べるように英数字だけでなくバーコードの数十倍から数百倍の情報量を扱う事ができる．すなわち，QR コードは，数字・英字・漢字・カナ・ひらがな・記号・バイナリ・制御コード等あらゆるデータを扱う事が可能である．データ量は 7,089 文字(数字)まで 1 つのコードで表現できる．図 1.3 に示すように，QR コードの収納可能文字数である．

表 1.1 QR コードの主な仕様

	仕様	
誤り訂正符号	RS コード (Reed-Solomon)	データコード語
		誤り訂正コード語
	BCH コード	形式情報
		型番情報
モード	数字	10 ビット/数字
	英数字	11 ビット/文字
	8 ビットバイト	バイナリ 8
	漢字	13 ビット/文字
バージョン（型番）	1	21×21 モジュール
	2	25×25 モジュール
	.	.
	40	177×177 モジュール
誤り訂正レベル	L	約 7%
	M	約 15%
	Q	約 25%
	H	約 30%
位置検出パターン	1:1:3:1:1	3 同心の正方形: 7x7, 5x5, 3x3 モジュール
位置合せパターン	1:1:1:1:1	3 同心の正方形: 5x5, 3x3, 1x1 モジュール

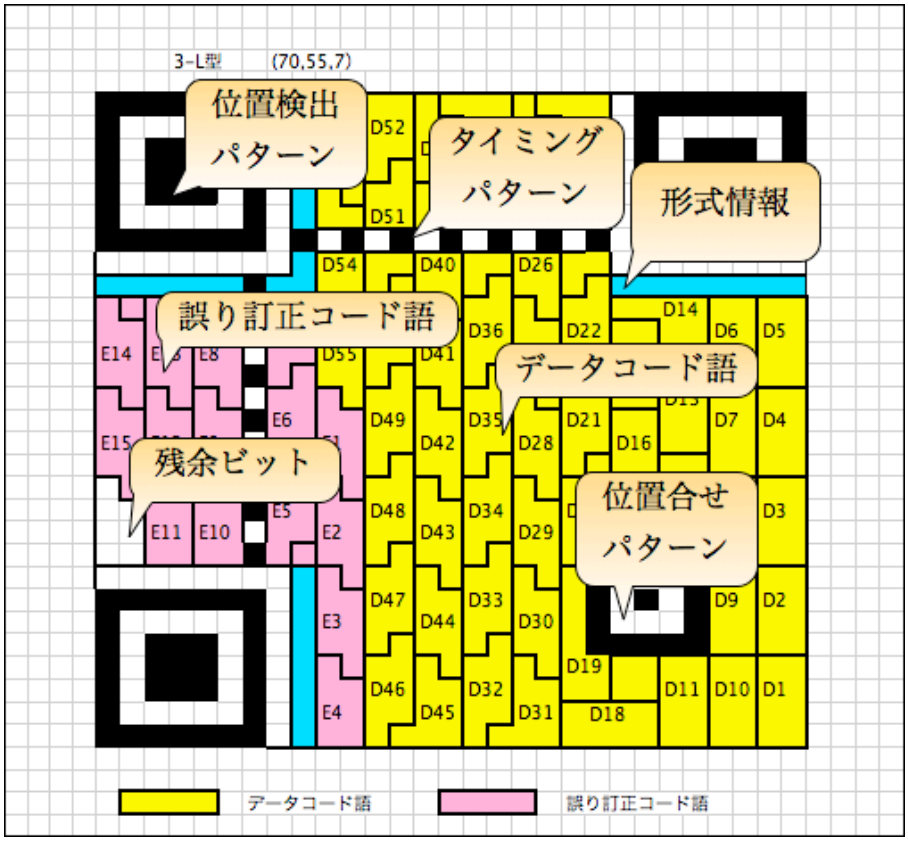


図 1.2 QR コードの例（バージョン３）

表 1.2 QR コードの表示可能文字数

QR コードの収納可能文字数	
数字のみ	最大 7,089 文字
英数字	最大 4,296 文字
バイナリ(8 ビット)	最大 2,953 バイト
漢字，全角カナ	最大 1,817 文字

1.2 QR コードの問題点

現状の QR コードは，白黒のドットで表示しているため，ただ目で見るだけでは内容は全然理解できない．また，この QR コードが示すサイトの名前，人の名前，広告をはじめ，どんな内容も分からない．見栄えも良くないので無視される事もある．これを解決する方法の 1 つは，図 1.3 に示すように絵などを QR コードの上に重畳する事である．しかし，QR コードの上に無条件にイメージを重畳するとその QR コードは読み取れなくなる．

図 1.3 イメージを重畳した QR コード

1.3 解決策

1.2 の問題点で述べた解決策として，QR コード作成ソフトを利用し，図 1.3 に示すように，イメージを重畳した QR コードであり，または絵のドットを描いて QR コードを作成し，Gimp[3]などのフリーの画像処理ソフトを利用して各ドットをカラー化してデザイン性を良くする方法がある．ただし，各ドットの色を変更するとイメージ損傷になり読み取り特性に影響を受ける．このため，読み取り特性の実験を行って，既存の携帯などに搭載されている QR デコーダで読み取れる色にカラー化するとともに，イラストなどのイメージを挿入する二次元コードを作成し，その読み取り特性を評価する必要がある．このため，NetBeans[4]を利用し，Java[5]言語でカラー二次元コードエディタのプロトタイプを作成する事により，実際にカラー二次元コードを作成し，既存の QR デコーダを利用して読み取り実験を行って，カラードットの認識特性を評価する．

1.4 各種二次元コードの開発状況

近年，QR コードのデザイン性を改善するため新しいタイプの二次元コード（QR コードの仕様に準拠していない）が開発されており，特にイメージや画像を重ね合わせたものが多く，携帯電話などの QR リーダ（QR デコーダ）をそのまま用いて簡単に読みとれる．例えば，QRForest[6]は，図 1.4 に示すように，ユーザが位置形状，データ領域の形状，配置形状と重ね合わせた画像を変更することができる二次元コードである．

図 1.4 イメージを重ね合わせた QRForest

その他の二次元コードの新しいタイプとして，メロディや画像を保存することができる着信メロディ付きの QR コードにイメージを合成できる QR-JAM[7]，DesignQR[8]などがあり，図 1.5 と図 1.6 に示す．

図 1.5 QR-JAM

図 1.6 DesignQR

さらに誤り訂正用の RS(Reed-Solomon)符号を非組織符号化法[9]により構成した新しいイメージ付きの二次元コードが開発されており，図 1.7 に示すように画像は任意の位置に多重化される．また，アニメーション QR コード[10]は，図 1.8 に示すように，最適化アルゴリズムを使用して設計されている．

図 1.7 非組織符号化 QR コード

図 1.8 アニメーション QR コード

第 2 章　研究の目的

現在，インターネットや携帯電話の普及に伴い QR コードが世界中に広まっており，各種用途に QR コードが使われている． 二次元コードの代表である QR コードは白黒 2 値のドット表示であるためデザイン性が良くないので，ドットをカラー化して見栄えを良くする方法が用いられる．しかし，ドットの色を変更したり見栄えを良くするためにイメージを重畳したりすると QR コードにとってはイメージ損傷になり，その読み取り特性に影響を及ぼす．また，紙に印刷したりディスプレイに表示したりした QR コードの場合には，カメラで撮影する際の照明条件および表示条件によっても QR コードの読み取り特性が異なる．

本論文では，通常の QR コードにドット絵などのイメージを重畳し白黒 2 値からカラー化してイメージ付き二次元コードを構成するカラー化二次元コードエディタを開発する．また，QR デコーダとして現状のスマートフォンなどに用いられている QR デコーダをそのまま用いた場合に，カラー化する際のパラメータによってどの程度正常に認識できるのか実験により読み取り特性を評価する．

第 3 章　QREditor の概要

QREditor[11]とは, 福岡工業大学の情報工学部情報通信工学科の山元研究室の越智祐樹により 2008 年度卒業研究として開発された二次元コードエディタである．白黒だけの作成の二次元コードエディタである．誤り訂正や型番やマスクを選択が可能であり，普通の QR コードを作成する場合はデータを入力し，交換ボタンをクリックすると QR コードの画像が出て来る．

図 3.1 白黒エディタ

二次元コードエディタでドット絵を描く場合は, エディタを描くボタンをクリックするとエディタ画面が出て来る．図 3.2 に示すように，QR エディタの画面である．この QR エディタでは，データを入力し，バージョンを選択してデータコードのシミュレーションが出来る. データコード語が入っていない中央部に自由に黒ドットを描いたり消したりする事ができる．　描き終わって OK ボタンをクリックするとその入力したデータや描いた絵のデータを計算して QR コードの画面に送り，エディタ二次元コードの作成ができる．

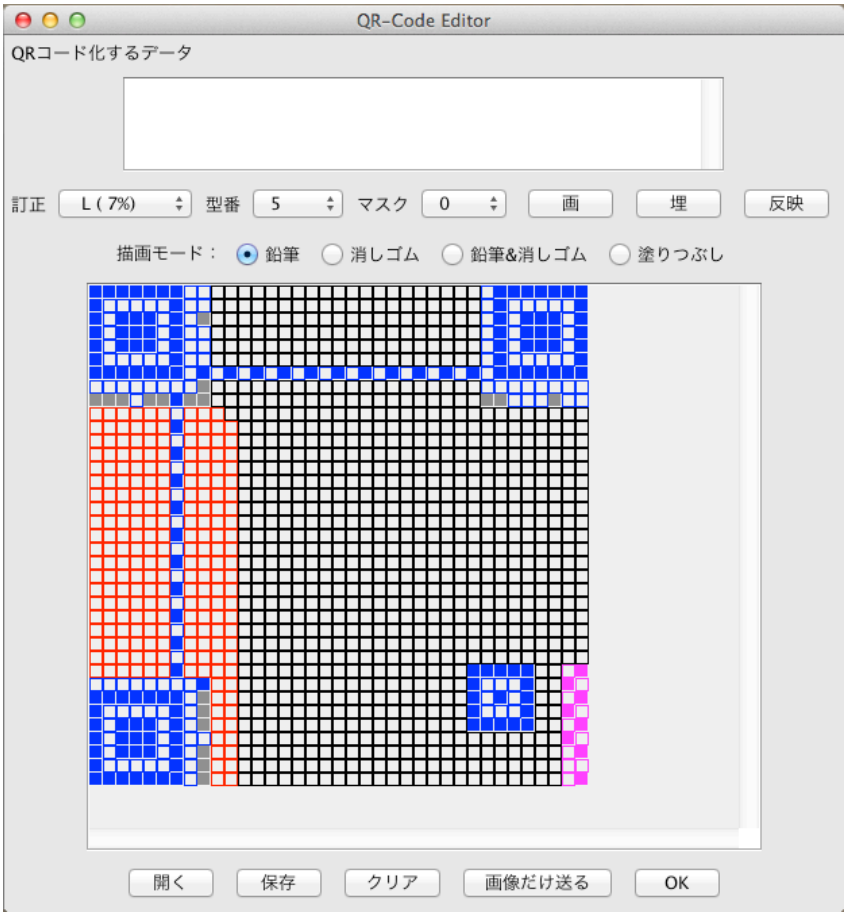


図 3.2 QR エディタの画面

第 4 章　二次元コードのイメージ付きカラー化手法

4.1 QR コードへの画像の挿入

QR コードの構造は, 図 1.2 に示したように, 機能パターン領域と符号化領域に分けられる. まず, 機能パターン領域は，位置検出パターン，位置合わせパターン，タイミングパターン，クワイエットゾーンで構成される．符号化領域は，形式情報（誤り訂正レベル，マスクパターン参照子，誤り訂正ビットなどを元にして計算し全体を 15 ビットで表す）及び型番情報，データおよび誤り訂正コード語で構成される．この符号化領域は，データを符号化した時の 0 と 1 の組み合わせで絵柄が決定する．その中でも実データの入っていない埋め草コード部の絵柄は，埋め込むデータを保持したまま自由に操作することが可能である．埋め草コードはデータビット列の終端パターンの後にある最終コード語の空の位置を埋める目的で使用するデータではないゼロのビットである. QR コードの作成手順を以下に示す.

1. 誤り訂正レベルの決定

誤り訂正レベルを利用者が選択する．誤り訂正レベルは低くすればするほど埋め草コード部が大きくなり，操作できる部分は増える.

2. 型番の決定

通常の QR コードを生成する手順とは違い，絵柄を元に埋め草コードを取得するため，型番の自働決定はできないので，利用者が決定する．型番は基本的に大きくなるほど操作できる部分は広がるが，位置合わせパターンの数を考えると一概にそうとは言えない.

3. マスクの決定

マスクは，評価を行い，最適なマスクの選択をすることが望ましいが，今回の研究では，自動で評価を行う部分を省略しているため，利用者が選択できるようにしている.

4. 入力データの決定

QR コードに埋め込むデータを決定する．データが少ないほど埋め草コード部は大きくなる.

5. 埋め草コード部分の位置を求める

1.～5. で決定された情報を元に，シンボルの埋め草コードが入る部分を機能パターンの位置やインタリーブ配置を考慮して求める. 図 4.1 の黒枠で囲っている部分は, 型番 5, 誤り訂正レベル L の 5-L 型シンボルでの入力データ「http://www.fit.ac.jp/」の時の埋め草コード部の位置を示している.

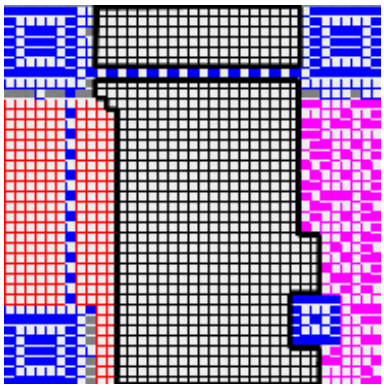


図 4.1 埋め草コード部

6. 埋め草コード部分の絵柄を決定する

埋め草コード部のデータを自由に操作して，出力したい絵柄になるようにする．図 3.2 はその例である．

7. 埋め草コード部分にマスクを適用する

6. で決定した埋め草コードはそのままでは使用できない．何故なら，QR コードではマスク処理が必須なので，マスクをかけると絵柄が大きく変わるためである．なので，4.1.6 で決定したデータにマスク処理したデータを埋め草コードとする．これで QR コード生成時にマスク処理をしたとき，元通り復号される．

8. QR コードの生成

1.〜4. で決定した設定と，7. の埋め草コードを使用して QR コードを生成する．

図 7 に示すように Image Multiplexing QR Editor（以降 QREditor と表示）のプログラム[12]を使用して QR コードを作成する．このプログラムでは，QR コードのマスクの代わりに任意のイメージ情報を挿入する事ができる特徴を有する．このため QR コードの冗長エリアに自由にドット絵などイメージをデザインする事が可能になる．図 3.2 には，ドット絵で「NORTH」の文字を記述しており，通常の QR リーダで読み取ることが出来る．

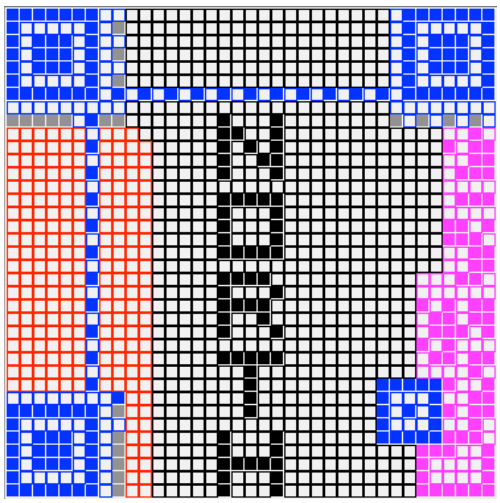


図 4.2　QR Editor によるドット絵挿入

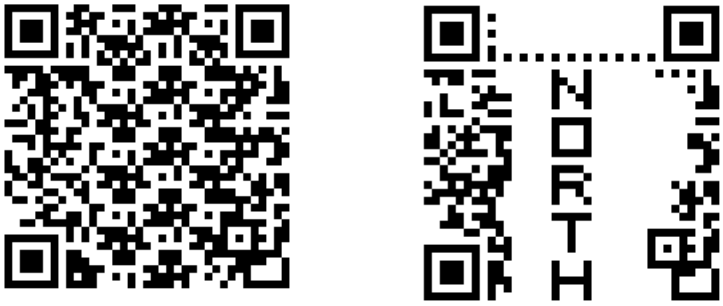


図 4.3　QR Editor から作成したコード

冗長エリアの中にイメージを入れると二次元コードは読み取れなくなる．つまり，冗長エリアであってもイメージを入れるとそのイメージも損傷になるという事である．

この原因は，人間の目で見るとその冗長エリアの中は単にデータがないと想定されるが，実は空白と見えていてもダミーデータが入っていることに起因している．図 4.3 に示すように冗長エリアの情報を見ているのはそのデータとマスクを排他的論理和(Exclusive OR)するものであり白い四角に見られるという事である．

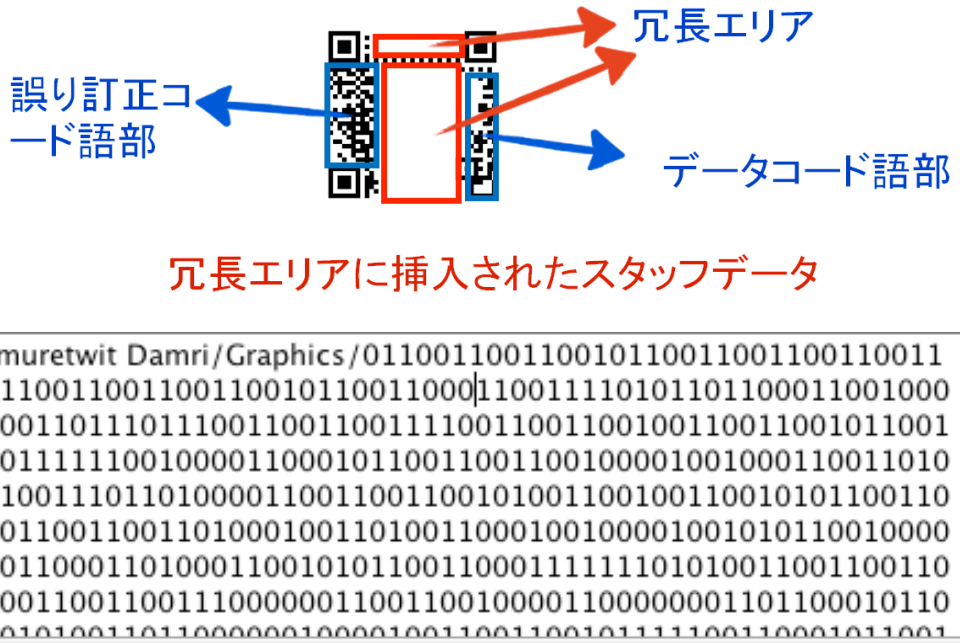


図 4.4 二次元コードの冗長エリアの中のデータ

4.2「暗い色の四角」と「明るい色の四角」のルール

二次元コードにイメージ損傷を作成しても RS コードに影響されない方法もある．“デザイン二次元コード”[13]によると，“符号化と復号では要となる処理に差異がある”である．また，同文献に“復号器は即応符号を構成する小正方形を「暗い色の四角」と「明るい色の四角」の二つに大別して認識する．黒い四角，群青色の四角，深緑色の四角などはいずれも「暗い色の四角」の暗ドットとして認識される．同様に，白い四角，淡いピンク色の四角，黄色の四角などはいずれも「明るい色の四角」を明ドットとして認識される．”とも述べられている．図 4.5 に示す例では，それぞれ赤と青

であるため「暗い色の四角」に相当し，暗ドットとして読みとりが可能である．以下，「暗い色の四角」と「明るい色の四角」のルールと呼ぶ．

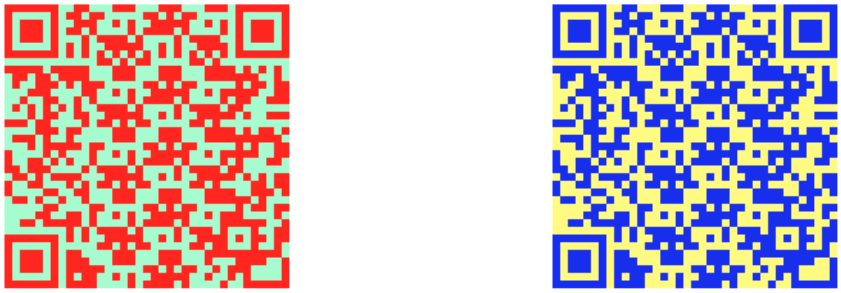
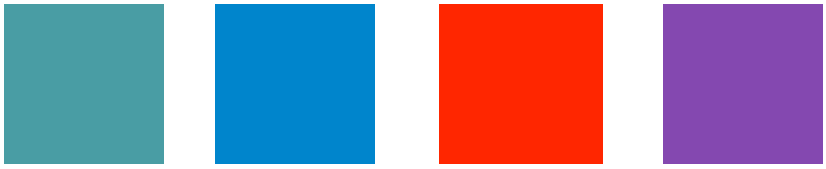
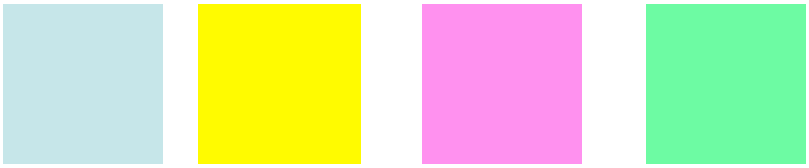


図 4.5 「暗い色の四角」の例（赤と青）
QR デコーダは，黒：暗い色の四角と白：明るい色の四角の二つに識別して認識する．



暗ドット（黒）：「暗い色の四角」



明ドット（白）：「明るい色の四角」
図 4.6　暗ドットと明ドット

QR 符号の符号化と復号では，白黒により明暗の処理を行っている．既存の QR デコーダでは，「暗い色の四角」である暗い緑，青，赤，むらさきを黒と識別し，「明るい色の四角」である薄い青，黄，ピンク，薄い緑を白と識別する．これを利用すると，二次元コードにイメージ情報を重畳して損傷させても誤り訂正符号に影響させずに読み取りが可能になる．

4.3 色の三属性

色の見え方は照明条件や表示条件によって変化するが，色の違いとその明るさや鮮やかさにより明暗を区別している点で共通する．これは，色相（H），彩度（S），明度（B）に対応しており，色の三属性[14]と呼ばれている．

白や灰色，黒のグレースケールは，明度で区別され，色相を含まず彩度が 0 である．これらの色は無彩色である．グレースケール以外の色は三属性すべてを持つ有彩色である[15]．しかしながら実際には，白や黒，グレーであっても通常は幾らかの彩度を有する[16]ので，いわゆる白や黒，グレーを，色の三属性を一つしか持たない色とするのは不適切である．

1. 色相は赤，黄，緑，青といった色の様相の相違である．特定の波長が際立っていることによる

変化であり，際立った波長の範囲によって，定性的に記述できる．ただし，常に同じ波長が同じ色に見える訳ではない．この総体を順序立てて円環にして並べたものが色相環である．色相の変化を示す例を 3 つ挙げる（5.3 節の実験 1 参照）．

2. 彩度は色の鮮やかさを意味する．物体の分光反射率が平坦になる程，彩度は低くなる．また，色相によって彩度が高いときの明度は異なる．彩度の変化を示す例を 3 つ挙げる（5.3 節の実験 2 参照）．
3. 明度は色の明るさを意味する．明度の高低は，物体の反射率との相関性が高い．光の明暗に関して明るさ (brightness, luminousity) があるが，同様の知覚内容を指していると言える．5.3 節の実験 2 に読み取り実験結果を示す．

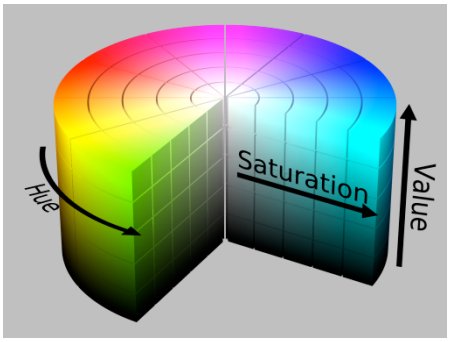


図 4.7 色の三属性[14]

4.4 レイアウト及びデザイン

前節の白黒 QR エディタ（QREditor）をカラー化するため，Java 言語を用いてカラーQREditor を開発する．Java 開発環境としては NetBeans を用いて開発した．また，画像編集処理ソフトとしては GIMP を用いてカラー編集処理を行った．これらを付録 A　ソフトウエアに示す．

デザイン性はカラーQREditor を用いてデータを入力し，各ドットをプロットしドット絵を描くことが出来る．ユーザは誤り訂正レベルや型番情報やマスクを自由に選択ができる．なお，自動的に行う場合には最低限の型番で選択されるので，ドット絵を描く場合には手動で選択するのが良い．変換ボタンをクリックすると，入力しているデータを二次元コードに変換し，コードを作成する．作成する二次元コードの画像のサイズの選択ボタンと白黒エディタのアルゴリズムに迷惑を掛けずに入力したデータを消すためのクリアボタンを追加した．エディタボタンをクリックするとエディタ画面が現れる．（図 4.8 カラーエディタ）

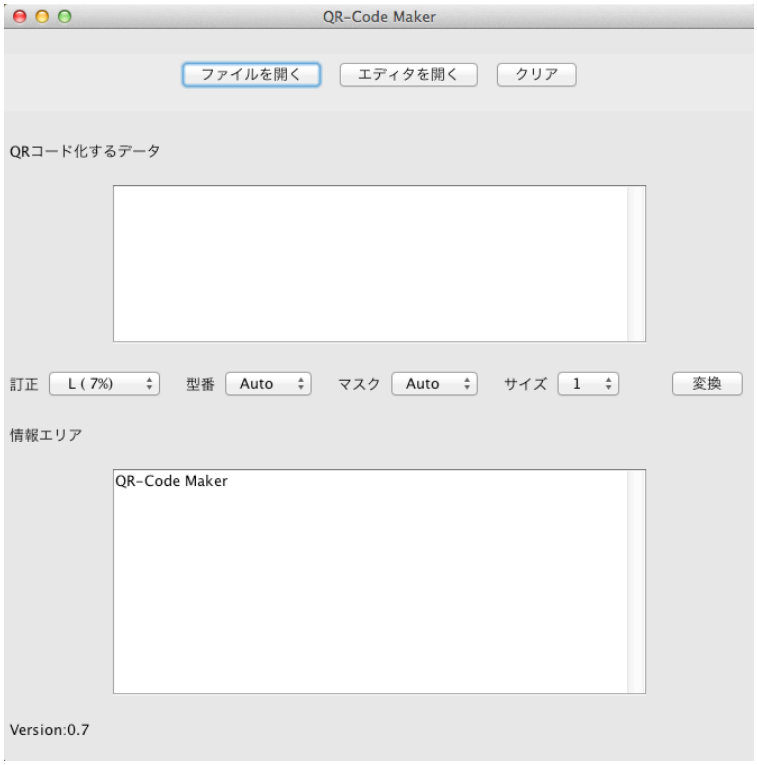


図 4.8 QREditor の画面

4.5　白黒 QREditor の機能追加によるカラーQREditor の構成

カラーQR エディタのユーザインタフェースはほぼ白黒 QREditor のままで，カラー化のボタンを追加してドットの描き方を編集できるようにした．このプログラムの特徴は，各ドットの色を個別にカラー化でき，最初に現れたエディタの画面では白黒のドットを設定している．カラー化のボタンをクリックすると色の選択画面が現れる．鉛筆の設定に色を選びこの画面に入っていない範囲に自由に描く事ができる.領域&背景の設定に入っている範囲や誤り訂正コード語や他のパターン背景の色の変更もできる．消しゴムにより描いたドットを背景色に設定する．図 4.9 に，カラー化のドットを描いて作成したカラードット絵二次元コードの画像である.

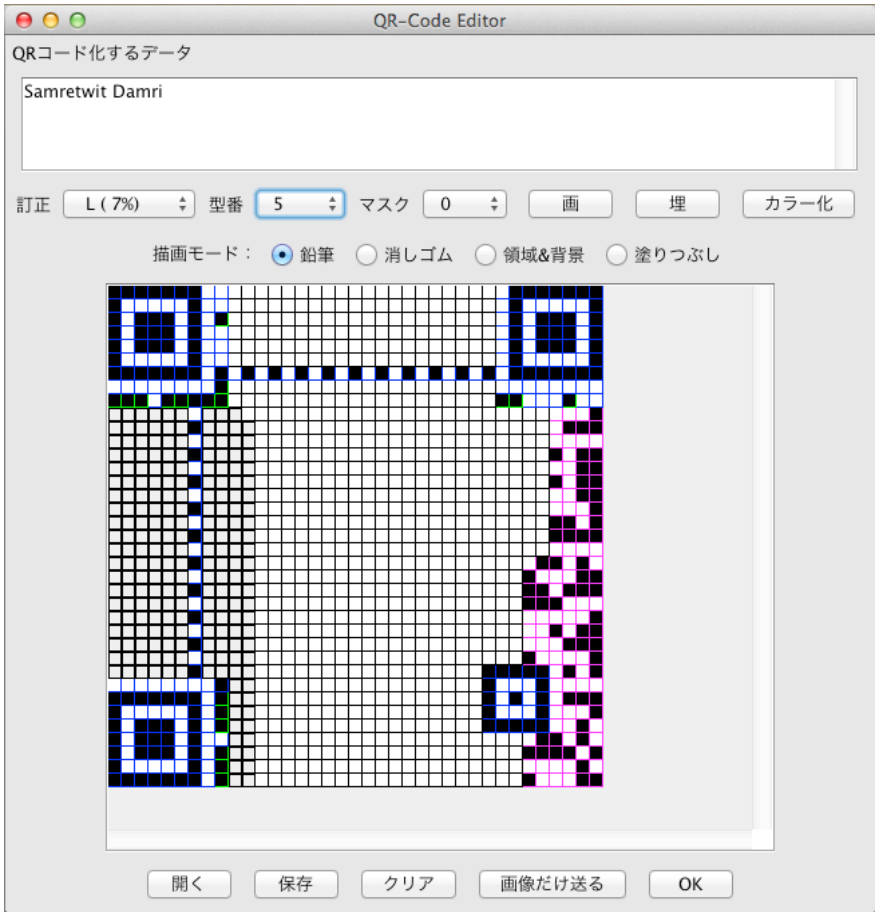


図 4.9 カラーQREditor のドット編集の画面

4.6 カラーバーの追加

カラーバーは, 選択できる色を RGB 値や HSB 値により設定する. 画面の下では選ばれた色の RGB 値を表示している．さらにその選んだ色はどのくらいの明るさが教えている．カラー調整を RGB のバーで自由に設定できるが, 輝度値の値によっては “明”か“暗”の設定を判断できない色もあるので, 確実に選択できる色として 50%以下の輝度値は“暗”, 90%以上の輝度値は“明”と設定することとした. これらの中間の値では QR デコーダのアルゴリズムにより“明”または“暗”の判定が難しかった．図 4.10 に試作したカラーQREditor で作成したカラードット絵付二次元コードの例を示す.

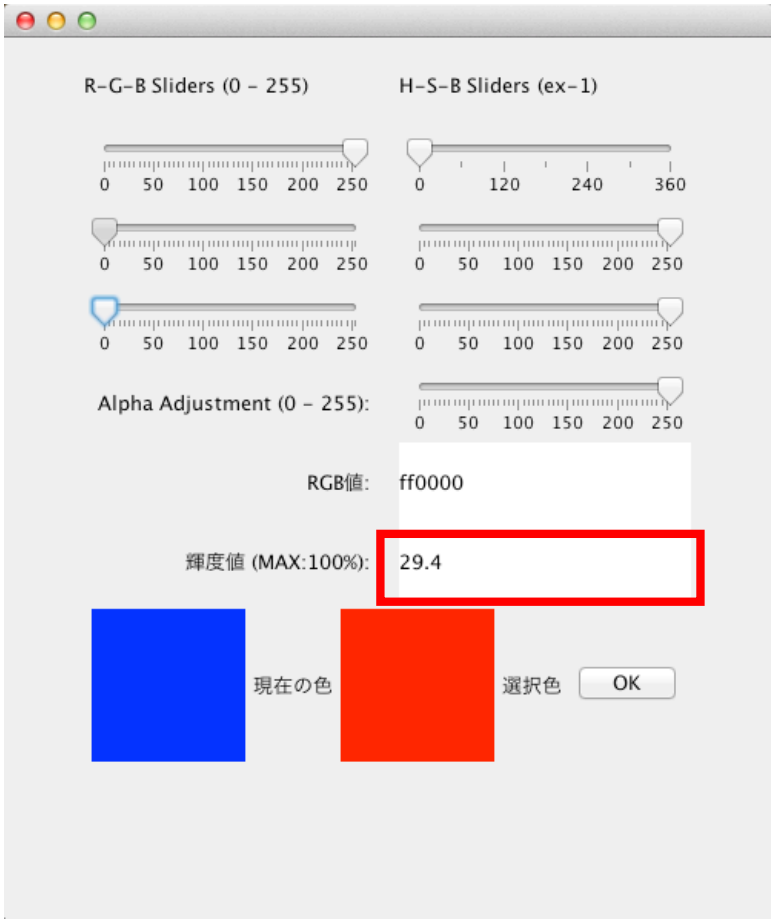


図 4.10　RGB と HSB の三属性のバー

インターフェースではその真ん中の輝度値になった状態で“OK”ボタンをクリックすると注意メッセージが表れる．かなり暗い設定の 50-60%の範囲の輝度値の“暗”の色ではさらに暗く設定するようメッセージが表示される．かなり明るい設定の 80-90%の範囲の輝度値の“明”の色はさらに明るく設定するようメッセージが表示される．

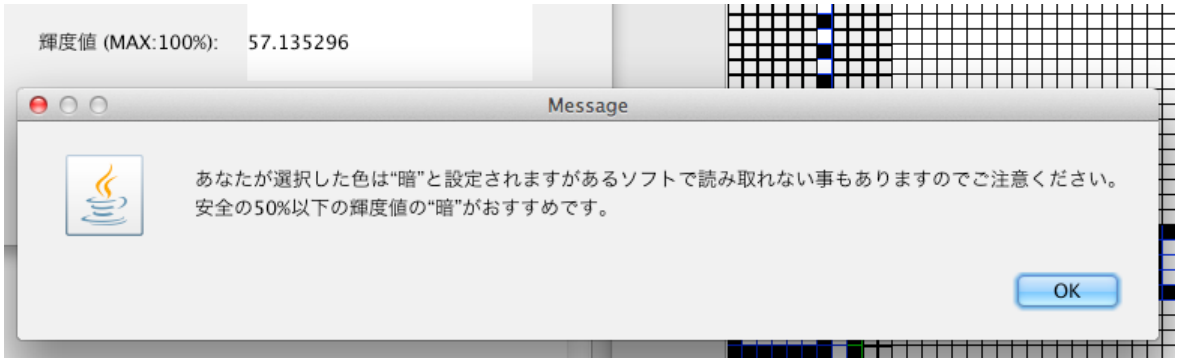


図 4.11 色をさらに暗く設定するための注意メッセージ

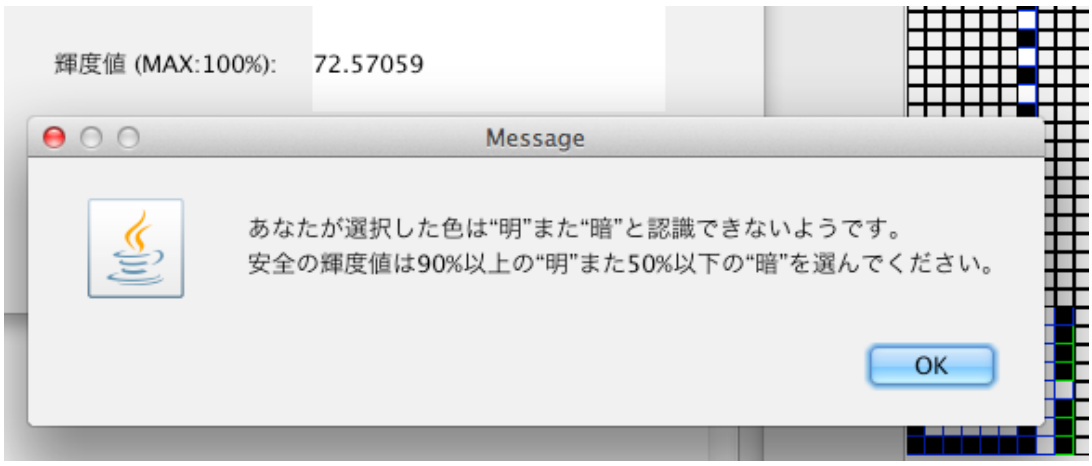


図 4.12 色をさらに明るくするよう注意メッセージ

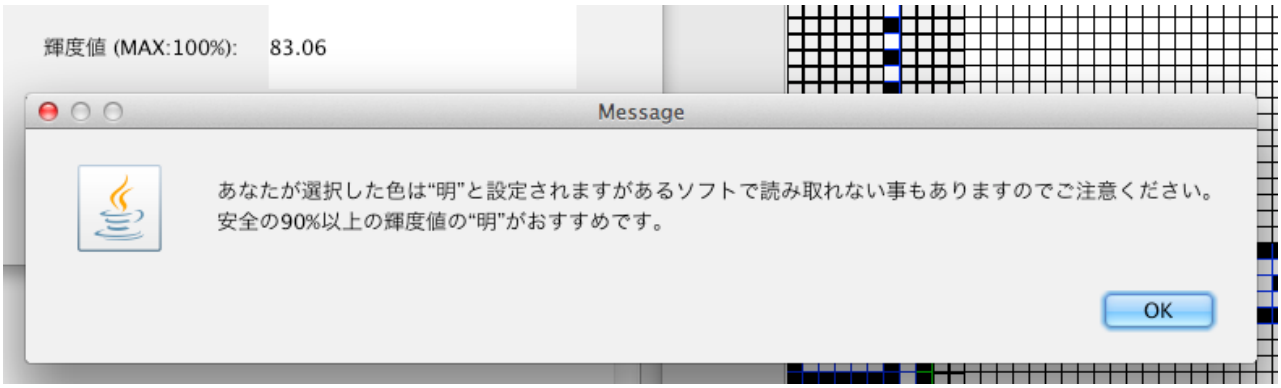


図 4.13 明暗を認識できない注意メッセージ

第 5 章　実験

5.1 実験環境

1　表示装置

実験時に利用した，表示装置は以下の通りである．

1)　MacBook Air(11 インチ)

2　実験期間

実験を行った期間は，以下の通りである．

1)　2011 年 7 月～2013 年 1 月

3　環境条件

実験時に利用した，環境条件である．

1)　通常照明(蛍光灯)

4　環境モニタリング条件

実験時に利用した，環境モニタリング条件である．

1)　80%輝度

5　QR 読み取り装置（QR リーダ）

実験時に利用した，QR 読み取り装置である．

1)　iPhone 4

6　画像編集プログラム

実験時に利用した画像編集プログラムである．

1)　Gimp 2.8.0

7　読み取りプログラム（QR デコーダ）

実験時に利用したカラー二次元コードを読み取るプログラムである．

1)　i-nigma[17]

2)　QRReader for iPhone[18]

3)　Best Barcode Scanner[19]

8　符号化データ

実験を行った際の符号化データである．

1)　英字 15 文字

9　型番(バージョン)

実験を行った際の型番(バージョン)である．

1)　5

10　誤り訂正レベル

実験を行った際の誤り訂正レベルである．

1)　L，M，Q，H のうちの 1 レベル

本研究では，二次元コードを開発するため，各種カラー化評価実験を行って誤り訂正の両力を確認するとともに色の 3 属性を用いドットの色の認識特性を評価した．その得られた結果を利用して

QREditor のソフトを改造してカラー化し，カラーQREditor を開発した．その実験の内容を以下に説明する．

5.2 QR コードにイメージを重畳した二次元コードの実験

イメージとして大きさの異なる 7 種類の静止画を用意し，QR コード上に重畳する．イメージの配置場所は左端，中央，右端の 3 種類とし，図 5.1 に配置条件を示す．同図で 7 種類の大きさの絵として，5x4 セル，7x5 セル，9x6 セル，10x8 セル，12x9 セル，15x11 セル，15x13 セルを準備し，QR コードはバージョン 3（29x29 セル）の二次元コードを用いる．

(a)5*4セル　(b)7*5セル　(c)9*6セル　(d)10*8セル

(a)12*9セル　(b)14*10セル　(c)15*11セル　(d)15*13セル

図 5.1 重畳するイメージの大きさ

また，イメージを重畳する位置についても，その配置場所により読み取り特性が変わるものと思われ，図 5.2 に示すように，左上，中央，左中央の 3 ヶ所の位置に配置した場合の読み取り特性を測定することとした．

(a)左上　(b)中央

(c)左中央　(d)右下

図 5.2 イメージの重畳位置

表 5.1 イメージを中央に重畳した場合の結果

	5*4	7*5	9*6	10*8	12*9	14*10	15*11	15*13
H	O	O	O	O	O	O	X	X
Q	O	O	O	O	O	X	X	x
M	O	O	O	O	X	X	X	X
L	O	O	X	X	X	X	X	X

表 5.2 イメージを左中央に重畳した場合の結果

	5*4	7*5	9*6	10*8	12*9	14*10	15*11	15*13
H	O	O	O	O	O	O	X	X
Q	O	O	O	O	O	X	X	x
M	O	O	O	O	X	X	X	X
L	O	O	X	X	X	X	X	X

表 5.3 イメージを右下に重畳した場合の結果

	5*4	7*5	9*6	10*8	12*9	14*10	15*11	15*13
H	O	O	O	O	O	O	X	X
Q	O	O	O	O	O	X	X	x
M	O	O	O	O	X	X	X	X
L	O	O	X	X	X	X	X	X

5.2.1 読み取り実験結果と考察

イメージを重畳して QR コードとして読み込み，読み取り実験の結果を表 5.1，表 5.2，表 5.3 に示す．○は読み取り成功，X は失敗である．なお,イメージを左上，すなわち位置検出パターンと合致する場合にはすべて読み取れなかった．表 5.1 と表 5.2 の結果から，レベル H では，中央の配置で 20.10%，左中央で 28.10%，右下で 15.60%，レベル Q では中央の配置で 15.60%，左中央で 23.40%，右下で 15.60%, ，レベル M では中央の配置で 11.90%，左中央で 20.10%，右下で 15.60%, ，レベル L では中央の配置で 5.00%，左中央で 7.80%，右下で 7.80%，の面積比率（全体は 675 セルで算出）で欠如しても読み取れることがわかった．

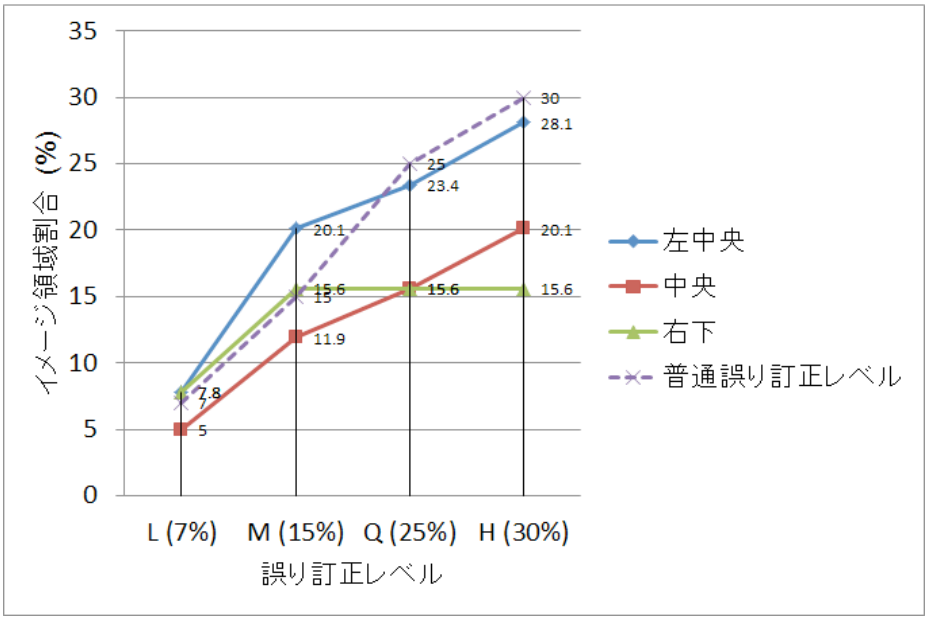


図 5.3　イメージを QR コードの上に入れた結果

図 5.3 によると色々な位置にイメージを重畳した割合を示す．割合はイメージのサイズ対全ての QR コードのセルの面積比である．点線は普通の誤り訂正割合，青線は左中央に重畳した割合，赤線は中央に重畳した割合，緑色線は右下に重畳した割合である．左中央位置は読み取り特性が最大の場所である．

5.2.2 画像を左中央に重畳した場合の結果

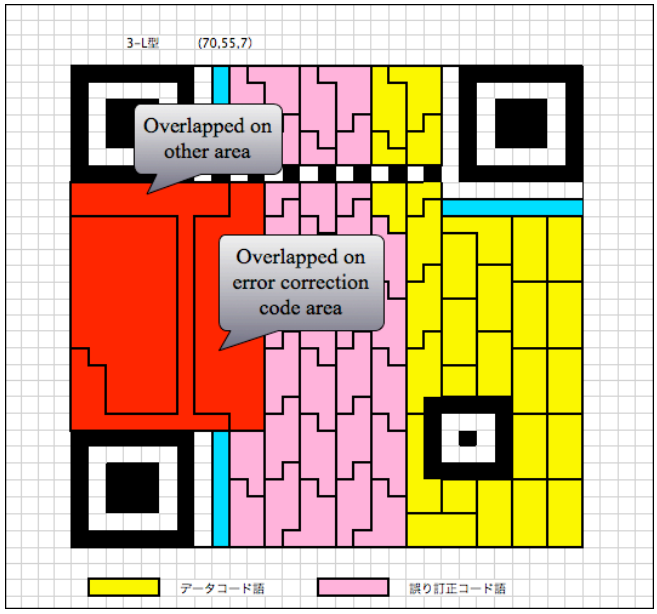


図 5.4 画像を左中央に重畳

赤色領域はイメージを重畳した位置．この位置は残余ビット，タイミングパターン，型番情報を重ねたため，誤り特性は影響を受ける．しかし，誤り訂正の能力により読み取りの特性が最も良い．残りのビット，タイミングパターン，型番情報は訂正アルゴリズムに影響されるビットがない事が分かった．

5.2.3 画像を中央に重畳した場合の結果

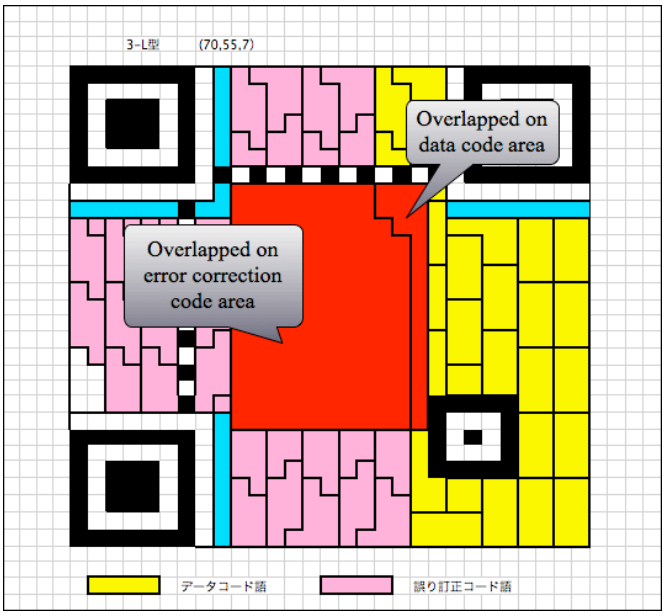


図 5.5 画像を中央に重畳

この位置はデータコード領域，誤り訂正領域が重なっているため，この位置は他のパターンにあまり影響されない．しかし，実験によるとレベル L と M で画像領域割合は最低である．

5.2.4 画像を右下に重畳した場合の結果

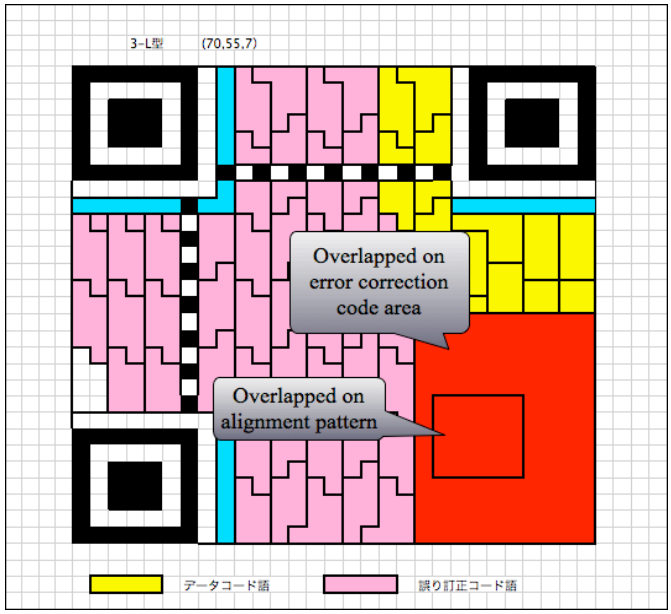


図 5.6 画像を右下に重畳

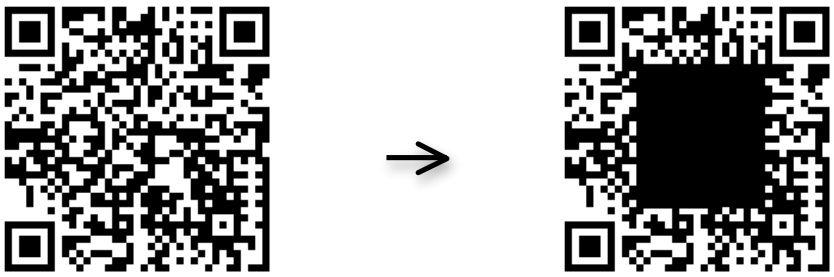
この位置はデータコード領域を位置合わせパターン領域に重ねたため，実験によるとレベル L と M でこの画像領域の割合は中央位置の画像領域の割合より高い．しかし，レベル Q と H では 15.60% 以上上がる事はなかった．位置合わせパターンは訂正アルゴリズムに影響されるビットがある可能性がある．

5.3 QR コードに損傷実験

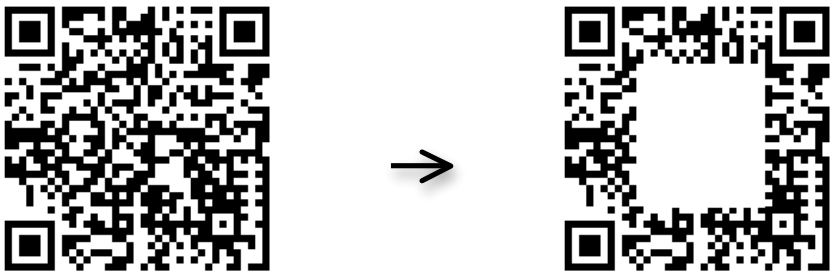
損傷された QR コードの例を図 5.7 に示す．この実験では，二次元コードの黒として識別したドットを白ドットに入り替えたり，白ドットを黒ドットに入り替えたりして，どの位の色が変更されたらそのドットが読み取れなくなるか評価する実験である．すなわち，色が変更されたドットが損傷になり読み取り特性が劣化するという事を確認する．

損傷された QR コードとして読み込み実験の結果を図 5.8 の読み取り特性のグラフに示す．一番高い誤り回復（レベル H）の割合は通常の誤り訂正レベルより少し（0，01%-3.80%）小さくなっている．実験通りに RS ブロックの全て 1072 セルから回復の割合のレベル L は 6.99%，レベル M は 12.30%，レベル Q は 21.20%，レベル H は 28.60%であった．

しかし，実際には損傷されたセルは回復（訂正）割合より小さくなり読み取れない可能性がある．それは損傷されたセルの位置や読み取る QR リーダのソフト次第である．



（ａ）損傷なし　　（ｂ）中心部へ暗の損傷



（ｃ）損傷なし　　（ｄ）中心部へ明の損傷

図 5.7 損傷された QR コードの例

この手法は，イメージや汚れなどの損傷エリアの面積の全体に対する比率を QR コードの誤り訂正の能力以下の面積に対応させる事である．図 5.9 と図 5.10 はその例であり，同図に示すように，読み込めるレベル H の誤り訂正を選んで 256(23.80%)セルのハートと 95(8.80%)セルのスターを入れている QR コードであり，両者とも読み込みが可能である．

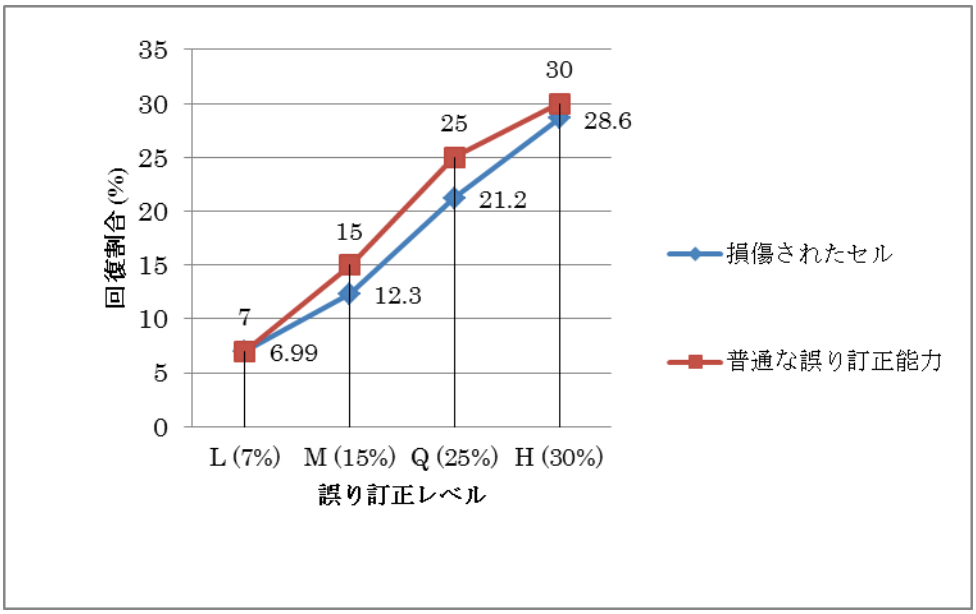


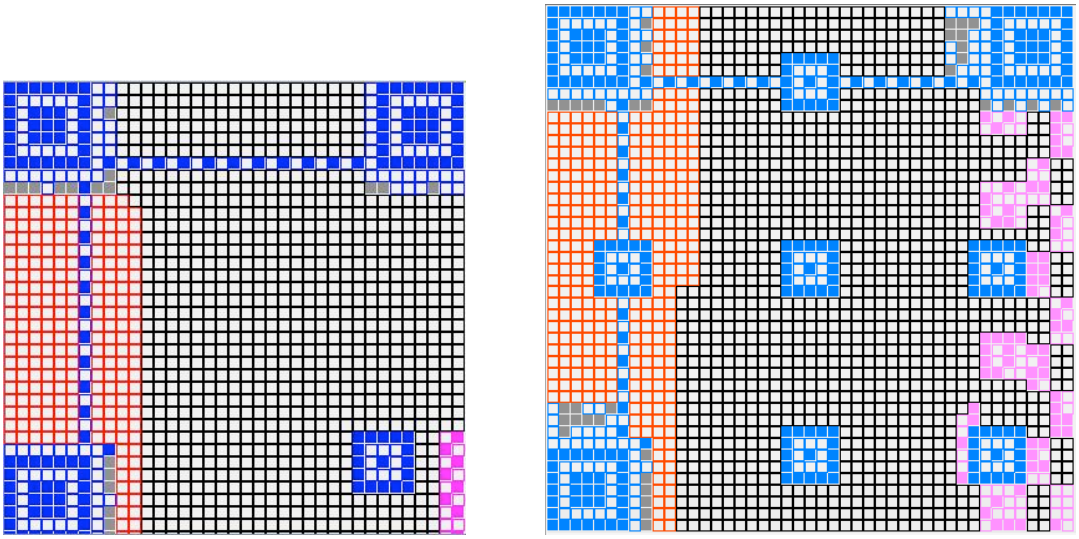
図 5.8 読み取り特性

図 5.9　256 セルのイメージ

図 5.10　95 セルのイメージ

5.4 冗長エリアに画像の挿入実験

中央の冗長エリアがある二次元コードを作成してその冗長エリアにイメージを入れる方法である．この方法は冗長エリアが小さい誤り訂正レベル M，Q，H の二次元コードではなく，レベル L を使用するのが良い．さらに，図 5.11 に示すように，バージョン 6 以上であれば RS ブロック数が 2 以上になりブロック毎にデータコード語が分離され，またバージョンが 7 以上では位置合わせパターンが 6 個以上になるので，図 5.12 に示すようにバージョン 5 レベル L の QR コードを使用するのが望ましい．



バージョン 5　　バージョン 7

図 5.11 QR コードバージョン 5 と 7 レベル L

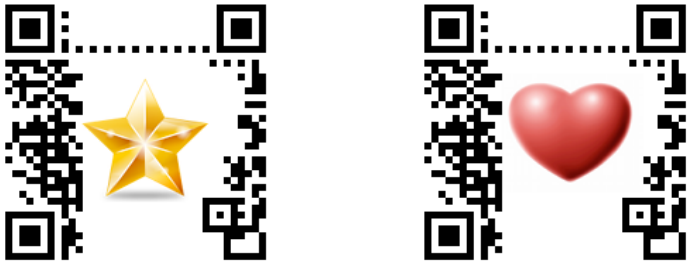
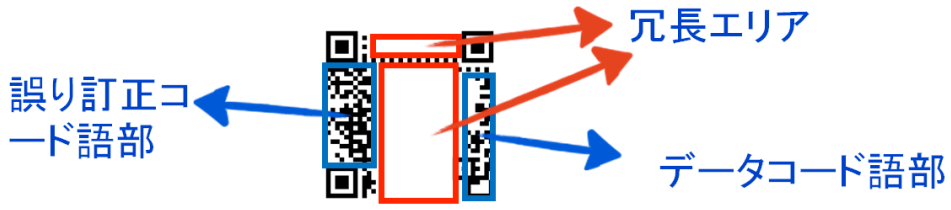


図 5.12 冗長エリアにイメージを重ね合わせた損傷
（いずれも読み取り失敗）

中央の冗長エリアの中にイメージを挿入すると二次元コードは読み込まなくなる．つまり，冗長エリアであってもイメージを入れるとそのイメージも損傷になるという事である．

この原因は，人間の目で見るとその冗長エリアの中は単にデータがないと想定されるが，実は空白と見えていてもダミーデータが入っていることによる．図 4.13 に示すように冗長エリアの情報はそのデータとマスク情報を排他的論理和(Exclusive OR)するものであり，白い四角すなわち明ドットに判定されるという事である．



冗長エリアに挿入されたスタッフデータ

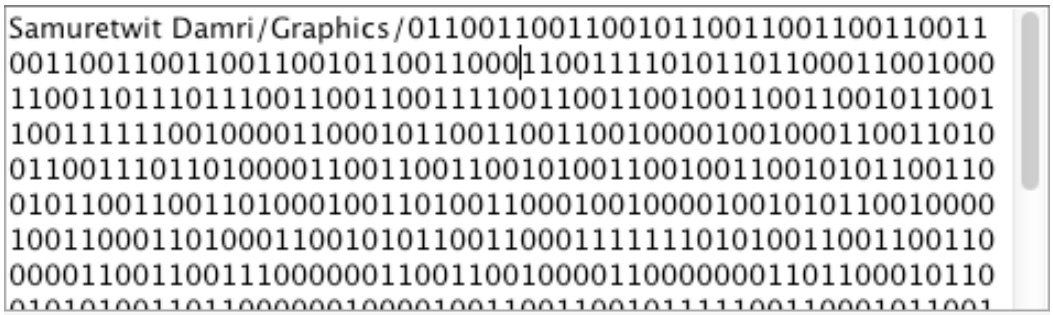


図 5.13 二次元コードの冗長エリアの中のデータ

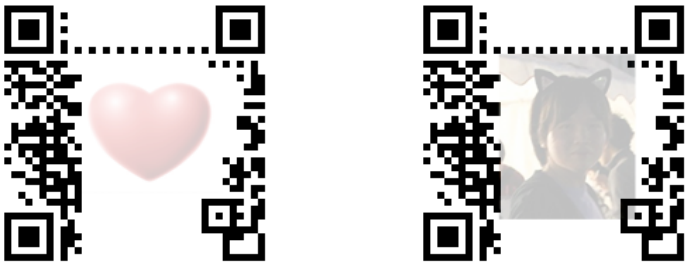


図 5.14　明るくしたイメージ

しかし，「暗い色の四角」と「明るい色の四角」のルールに基づいて図 5.14 に示すように冗長エリアの部分に入っているイメージの色を全て明るくすると，その二次元コードは正しく読み取りが可能なようになる．

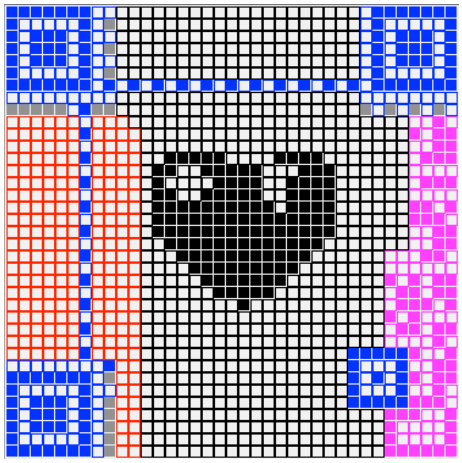


図 5.15　　QR コードのダミーデータ

（a）冗長エリアに重畳

（b）確認

図 5.16　ダミーデータが入っている二次元コード

もう一つの方法は「暗い色の四角」と「明るい色の四角のルール」に基づいて図 5.15 に示すようにイメージの姿が似ているダミーデータが入っている二次元コードを作成してダミーデータの所にそのイメージを入れる事であり，その二次元コードが読み込めるようになる．図 5.16 にその例を示す．しかし，この手法はダミーデータとの位置合わせが難しく失敗の可能性が高い．

5.5 カラーイラスト二次元コードの作成実験

次に，カラーのドット絵を重畳するカラーイラスト二次元コードを作成方法について述べる．まず，図 5.17 に示すようにイラスト二次元コードを作成する．「暗い色の四角」と「明るい色の四角」のルールに基づいて，作成した二次元コードのドットの色を自由に変えてカラーイラスト二次元コードを作成できる．

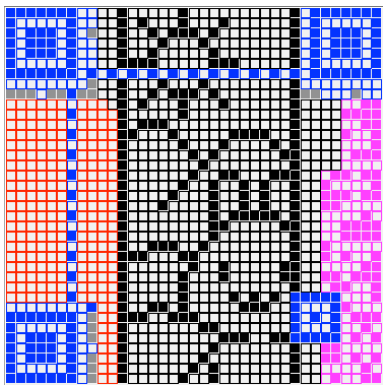


図 5.17 イラスト二次元コード

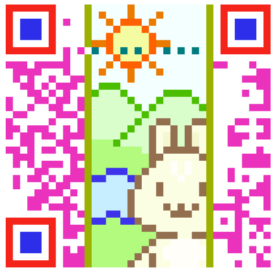


図 5.18　白黒→カラーイラスト二次元コード

二次元コードのバージョンを 5 以上に選び，(2)に述べた理由で RS ブロック数が 2 以上でブロックが複数に分離されバラバラになって位置合せパターンが邪魔しているので，二次元コードのバージョンを 5 のように大きい冗長エリアの誤り訂正レベル L を選ぶのが良い．

この手法により，デザインを描くのに多少時間がかかっても二次元コードの上に自由にデザインできる上に「暗い色の四角」と「明るい色の四角」のルールを利用して誤り訂正の影響を受ける事はない．また，この方法で作成した出力は見栄えも良い．図 5.18 にその例を示す．

5.6 色相の変化による読み取り実験

同色では，暗ドットと明ドットを入れ替えても読み取れるようになる．

それらの色の色相の値を変更させると色の RGB の値が変わって，ドットの色が変わり赤，緑，青の色になる．図 5.19 に示すように，全色では明度と彩度の値はそのまま 100%にしておくと，それらの色は暗でも明でも認識できるようになる．

次に，明度と彩度の 100%の値でも色相の値を変更させると，色も変化し色の濃さも変わる．図 5.20 に，色相の値を変更し黄色から青まで濃度の順に並べたものを示す．この順番では，黄色とシアンは一番明るい色であって暗い色としては使えない，また，緑は黄色より暗く，ピンクと赤では青色より明るく，青は一番暗い色に識別される．RGB から見ると RGB の色の 2 つを混合したら，色相の値が変化し色の濃さも明るくなる．すなわち，RGB の値の多対多関係は色相である．さらに，RGB3 色を全て混合したら，結果は白になる．

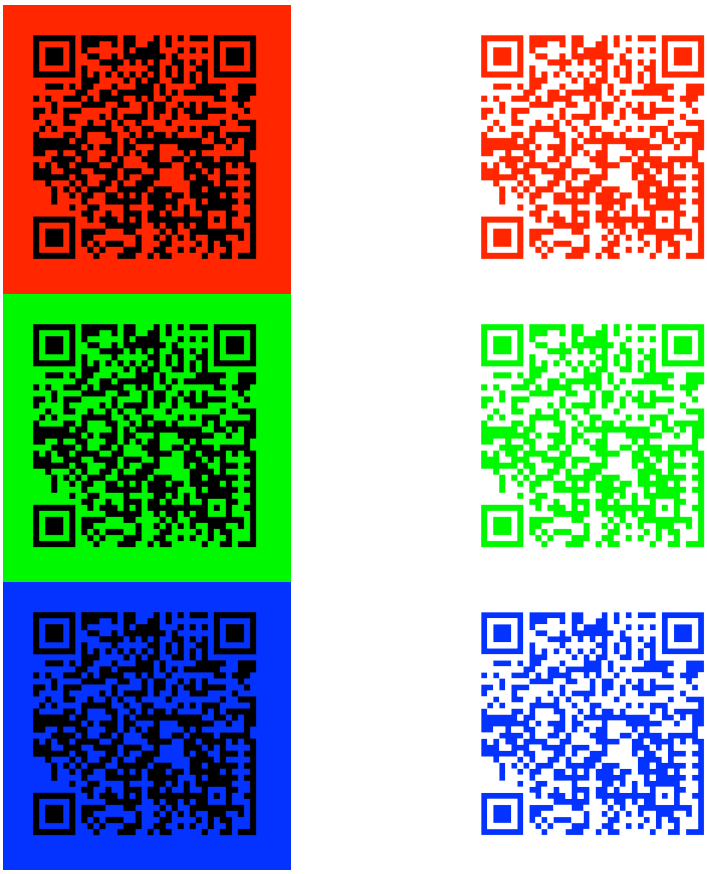


図 5.19 ドットの色相を変化させた QR コード

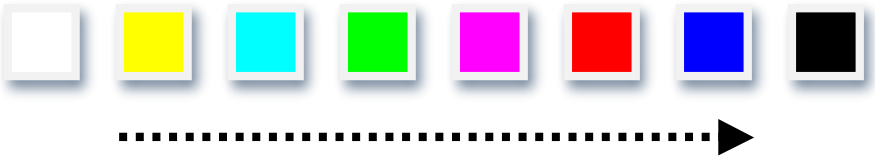


図 5.20 カラー化の色相濃度の変化の例

色の濃さの順番が明確になることにより，ドットの白黒の色に代わりカラー化する事が可能になる．図 5.21 に示すように，黒ドットに代わり濃さが一番高い青ドットを入れ，白ドットに代わり濃さが青より薄いピンクやシアンドットを入れると読み取り可能になるが，明暗ドットの場合に比べると色の濃さの差が小さく，なかなか読み取り難い．一方，図 5.22 に示すように，明ドットに代わりピンクドットに入れ替え，暗ドットに代わり濃さが一番薄い黄色ドットやピンクドットと濃さがあまり違わない赤ドットに入れ替えると読み取れなくなる．

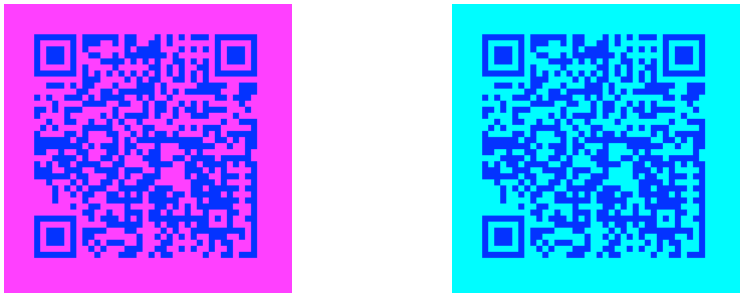


図 5.21　色相を変化させ読み取り可能な例

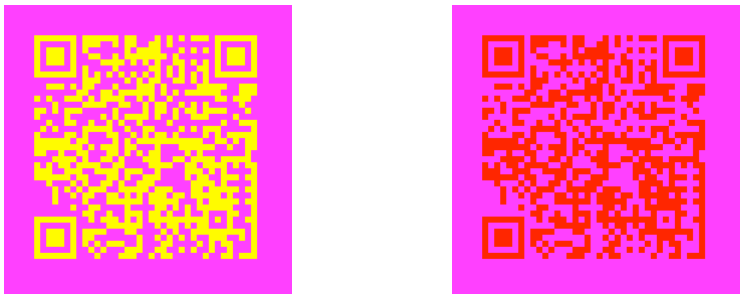


図 5.22　色相変化で読み取り不可能な例

5.7 明度の変化による読み取り実験

実験 2 は，明度の値を変更させた読み取り実験である．それぞれの色の値を 0%まで低くすると色は黒くなる．すなわち，ドットの色の明度の値を低くするに従ってそのドットの色は黒になる．この実験では，それぞれの色の明度の値をどれくらい低くしたら，色が暗く認識されて読み取りが可能か評価する．図 5.23 に示すように，青や緑の明度の値を変更して濃さの差が異なって，濃い青や緑が暗に認識され薄い青や緑が明に認識されて読み取れる QR コードである．

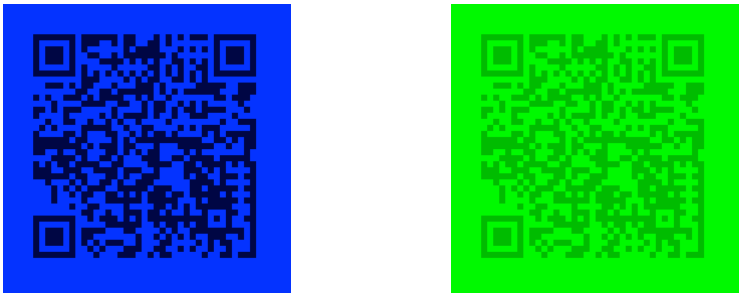


図 5.23　明度の変化による読み取り可能な二次元コード(QRRader for iPhone で読み取り)

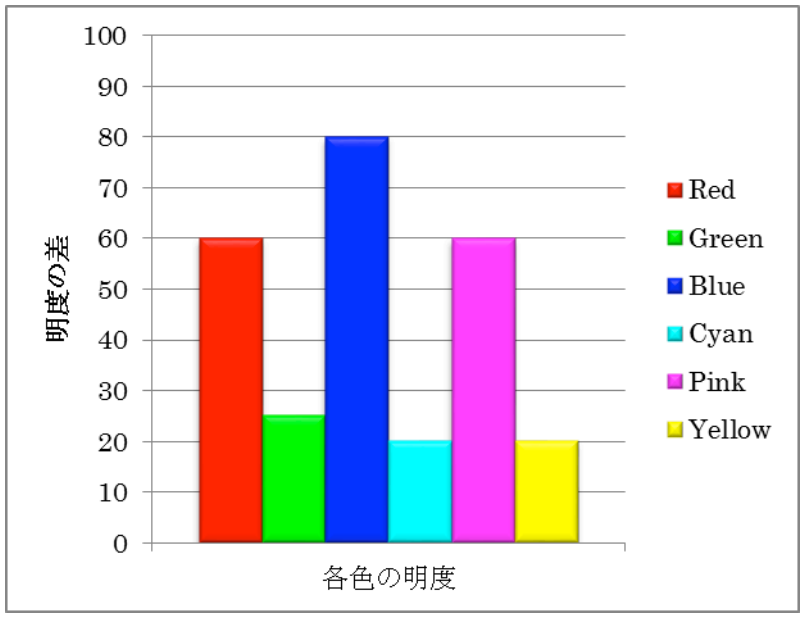


図 5.24　明度の変化に対する各色の読み取り特性

図 5.24 に示すように，明暗の代わりに各色の薄い色と暗い色を入れて，各色の明度を変化させ薄い色を“明”に認識され暗い色を“暗”に認識され正しく読み取った明度差の限界のグラフである. 赤の場合は，色の明度の差が 60.00%以上であると正しく“明”と“暗”に認識される．緑の場合は 25.00%以上であり，青では 80.00%以上，シアンでは 20.00%以上，ピンクでは 60.00%以上，黄色では 20.00%以上であれば正しく読み取れる事が分かった.

この実験の結果，どの色でも明度の値を落としてその色の濃さを変化させると暗くなり，色の明度の値の差が十分な値になったら正しく読み取れる事が明らかになった.

5.8 彩度の変化による読み取り実験

実験 3 では，彩度の値を変更する実験である．明度の場合と反対にそれぞれの色の値は 0.00%になると色が明るくなり，彩度の値を低くするとドットの色が明るくなる．この実験では，それぞれの色の彩度の値をどこまで低くしたら，色が明るく認識されるか評価する．図 5.25 に示すように，ドットは青と赤の彩度の 100.00%の値で青と赤の背景部分の“明”の部分を彩度の値を低くし, 読み取り可能な限界の QR コードである.

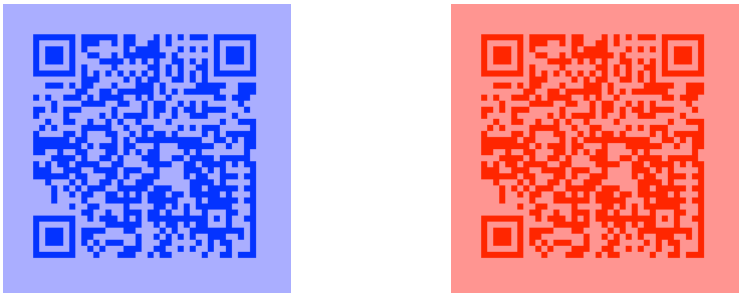


図 5.25　彩度の変化によるカラー化の例

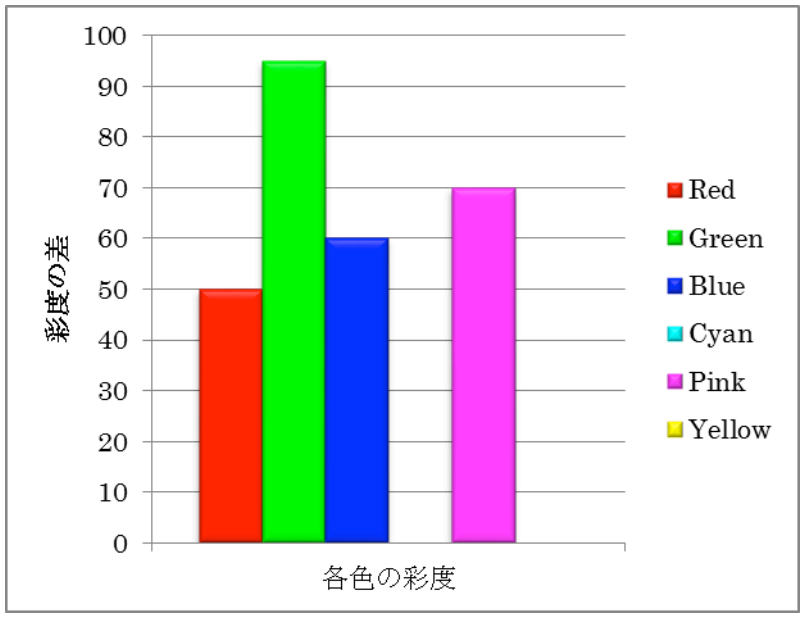


図 5.26　彩度の変化に対する読み取り特性

図 5.26 に示すように，QR コードの明暗の代わりに各色の薄い色と暗い色に入れ替えて，各色の彩度の差を変化させ，薄い色が“明”に認識され暗い色が“暗”に認識されて正しく読み取られた限界のグラフである．赤の場合には色の彩度の差が 50%以上にしたら薄い色と暗い色に認識される．緑の場合には 95%以上，青の場合には 60%以上，ピンクの場合には 70%以上であれば正しく読み取られることがわかった．しかし,シアンと黄色は彩度の値を変更して 100%の差に設定しても“明”と“暗”の認識できない事が分かった．

この実験の結果，どの色でも色の濃さを明るくする際に，“明”と“暗”で彩度の値を低くすると色の濃さが明るくなり，色の彩度の値の差が十分な値になったら，値が低い色のドットは“明”と認識される．

以上の 3 実験の結果は，通常の QR リーダーでどの色でも“明”と“暗”のいずれかに識別されるので，各ドットの周辺の環境，濃さの状況次第でそのドットの色が“明”か“暗”に認識される事がある．

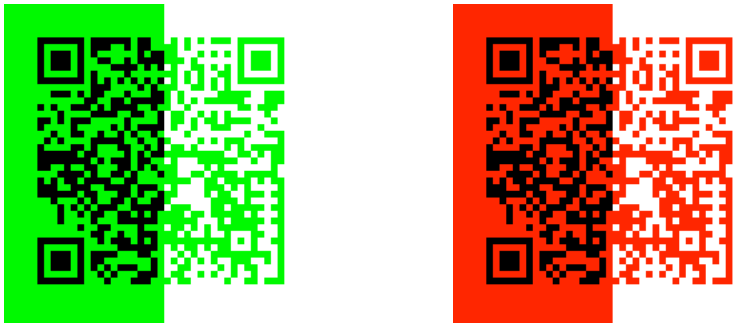


図 5.27　背景の半分のエリアの色を変化させる例

具体的には図 5.27 の左図に示すように，同じ緑の色が左半分のエリアでは“明”に，右半分のエリアでは“暗”に識別され，正しく読み取ることが可能である．一方，同右図のように同じ赤であっても，“明”または“暗”に識別され正しく読み取ることが可能である．

5.9 イメージ付きカラー化二次元コードの読み取り特性の評価実験

前述したように，既存の QR デコーダがどのような色を“暗”と判定し，どのような色を“明”と判定するのか明らかになっていない．このため,GIMP でカラー化する際に，色相，彩度および明度の三属性を変化させ,読み取り実験を行った. QR デコーダとしては, スマートフォン（iphone4 の QRReader for iPhone)を用いた．色の三属性の変化範囲としては，図 5.20 に示したように色相の値を 0 の赤から 360 の赤まで変化させると同時に彩度も 0 から 100 まで変化させて読み取りが可能か記録した結果である．なお，用いた符号データは英字 15 文字，QR コードの型番は 5，誤り訂正レベルは H の 30% で PC（MacBookAir）上に表示させた．図 5.28 は彩度を変化させ“暗”ドットを入れ替えた例である．

彩度の 40　　彩度の 60　　彩度の 90

図 5.28 “暗“の部分を入れ替えた例

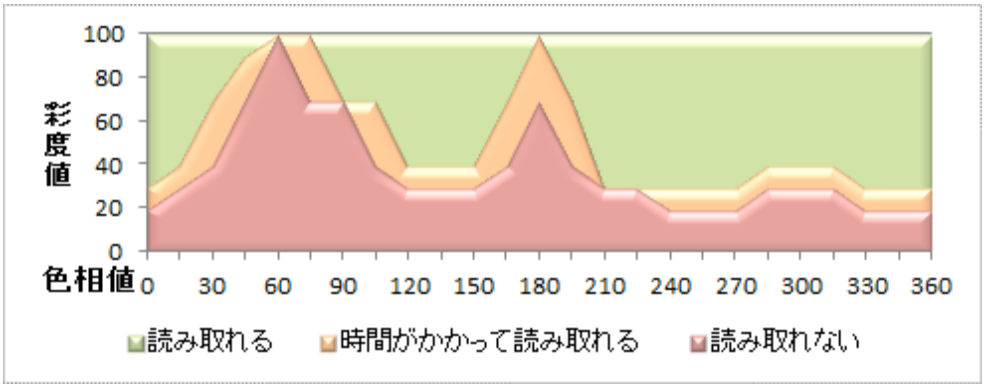


図5.29 彩度と色相を変化させた時の読み取り特性 (QRReader for iPhone)

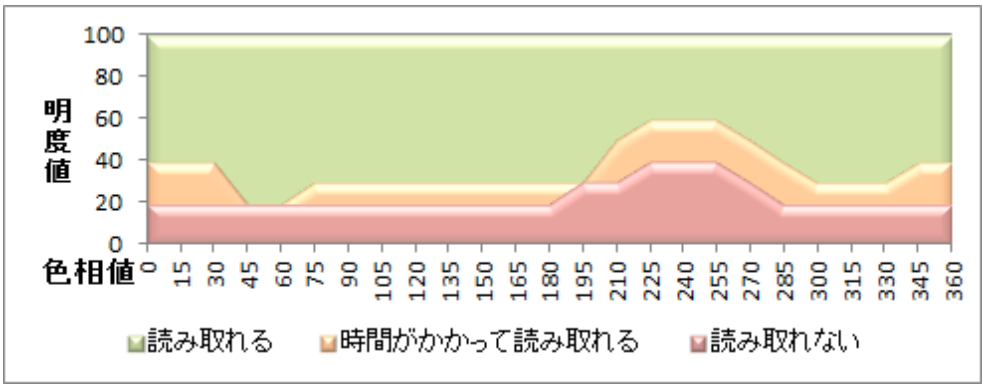


図5.30 明度と色相を変化させた時の読み取り特性 (QRReader for iPhone)

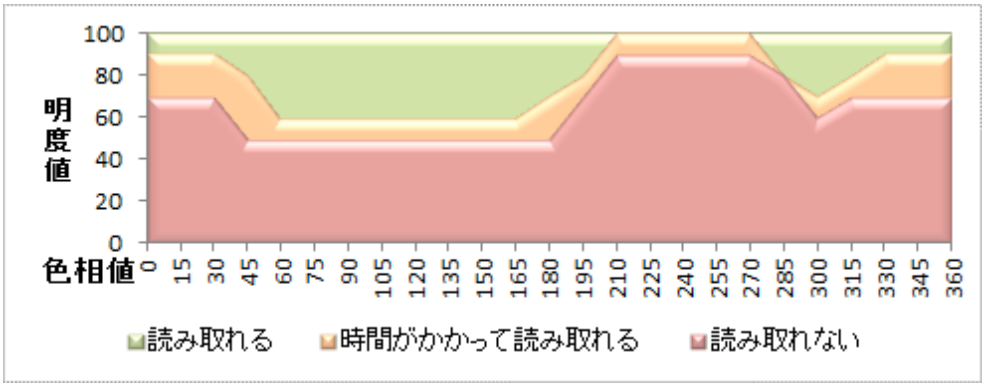


図5.31 明度と色相を変化させた時の読み取り特性 (Best Barcode Reader)

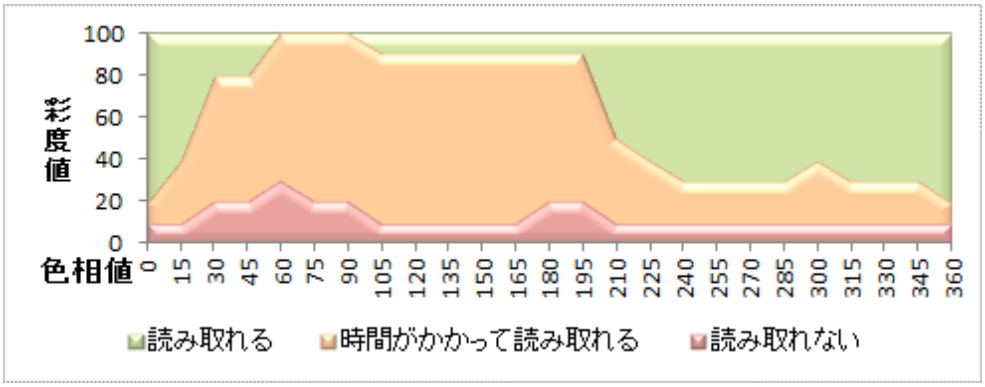


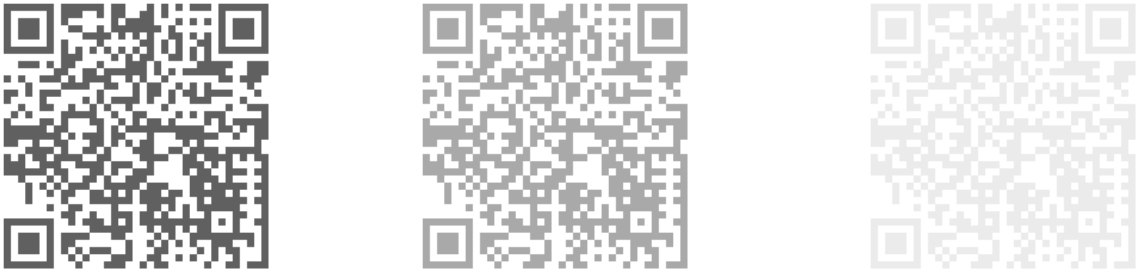
図5.32 彩度と色相を変化させた時の読み取り特性 (Best Barcode Reader)

図5.29で色相値を変化すれば色が変わり, 彩度値が小さいほど色が明るくなって読み取り難くなる. 色相の値が60の場合は黄色に該当し，明度が高いので彩度値を変化しても読み取れない．色相の値が180の色はシアンであり明度も高いので読み取れても時間を要する．このように明るい色には注意が必要である．図5.32で明度値が低いほど色が暗く読み取り難くなることがわかる．色相の値が0と360の色は赤であり明度が高いので明度値を減らして，読み取れない事が大きい．色相の値が240の色は青であり赤よりも暗いので読み取る範囲がさらに大きい．つまり赤と青はかなり明度が低いので明の部分に入れ替えると読み取り難くなる.

5.10 モノクロ輝度値の変換式による実験

5.9 のため，RGB から見ると，それぞれの明るさが持っているので，それらの色の明るさを検討する．色を認識する場合，「暗い色の四角」と「明るい色の四角」のルールに従って色の濃さを判定

すると考えられるためカラーの RGB 値をモノクロに変換してその輝度値で判定することとし，平均値を求める．実験では，カラーをモノクロ化して，その RGB 値をグレー化し，“暗“の場合の明るさを黒と認識し，”明”の明るさを白と認識する．図 5.33 は，グレー化して“暗“のドットを明度値 30, 60, 90 のドットに入れ替えて作成した二次元コードの例である．一方, 図 5.34 はグレー化して“明“のドットを明度値 30，60，90 のドットに入れ替えて作成した二次元コードの例である．



明度値 30　　明度値 60　　明度値 90

図 5.33　“暗“の部分を入れ替えた例

明度値 30　　明度値 60　　明度値 90

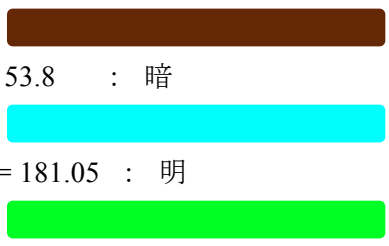
図 5.34　“明“の部分を入れ替えた例

この読み取った結果で，RGB のモノクロ輝度値が 200 以上の値であれば完全に“明“と認識され，160 以上の場合はほぼ“明“と認識された．一方，RGB のモノクロ平均値が 50 以下の場合は完全に“暗“と識別され, 80 以下の場合はほぼ“暗“と認識され, 80 から 160 の間は“明“または“暗“と認識された．

RBG のモノクロ輝度値としては，下記の(1)式を計算して求める．

$$X = 0.294R + 0.615G + 0.091B \qquad (1)$$

$$0 < R, G, B < 255$$



0.294X170 + 0.615X40 + 0.091X0 = 53.8　　：　暗

0.294X0 + 0.615X255 + 0.091X255 = 181.05　：　明

0.294X0 + 0.615X255 + 0.091X0 = 158.1　　：　両方

図 5.35　(1)式によるモノクロ輝度値の計算の例

R，G，B の値は，GIMP の RBG の 0 から 255 までの値であり，X は輝度値であり出力の明るさである．この数式で RBG を入力して求めた X の値が 160 以上の場合は“明“と認識され，80 以下の値の場合は“暗“と認識され，X が 80 と 160 の間の場合は“明“または“暗“と認識される．図 5.35 は，モノクロ平均値の計算の例である．

色の明るさが分かりやすいように RGB の値から色相の値に変換し，輝度の明るさとの関係を求めて図 5.36 に示す．

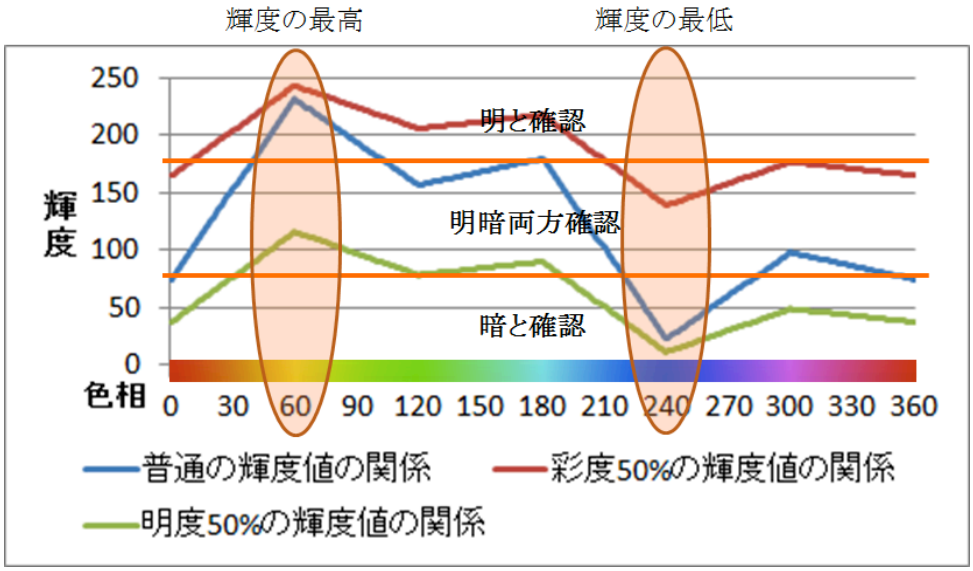


図 5.36　色相変化によるモノクロ化輝度値と輝度の関係

色相の値が 60 の場合は RGB の赤と緑色が混色した黄色に相当し，最も明るい．一方，色相の値が 180 の場合は RGB の緑色と青を混色したシアンに相当し，次に明るい．最も暗いのは色相の値が 240 の青色であり，次は色相の値が 0 と 360 の赤である．80 と 160 の間の値である 158 の緑色や 94 のピンク色は，周囲の環境により“明“または”暗“と認識される．

色の明るさを分かりやすいように RGB 値から色相の値に変換し，輝度の明るさとの関係を求めて図に示す．このグラフでは彩度値の 100%と明度値の 100%の普通の輝度値と彩度 50%の輝度値と明度 50%の輝度値の関係に別れている．彩度 50%ではかなり明と認識され，明度 50%ではかなり暗と認識される．つまり，彩度を落とすと輝度が増え，明度を落とすと輝度が落ちる．色相値が 60 の場合は RGB の赤と緑色が混色した黄色に相当し，最も明るい．一方，色相値が 180 の場合は RGB の緑色と青を混色したシアンに相当し，次に明るい．最も暗いのは色相値が 240 の青色であり，次は色相値が 0 と 360 の赤である．80 と 160 の間の値である色は，周囲の環境により“明“または”暗“と認識される．

5.11 カラーQREditor プロトタイプの作成

実験から得られた色の輝度値や計算手法結果によりカラー化ドット絵付二次元コードエディタ（カラーQREditor）のプロトタイプを試作し，読み取り実験を行った．

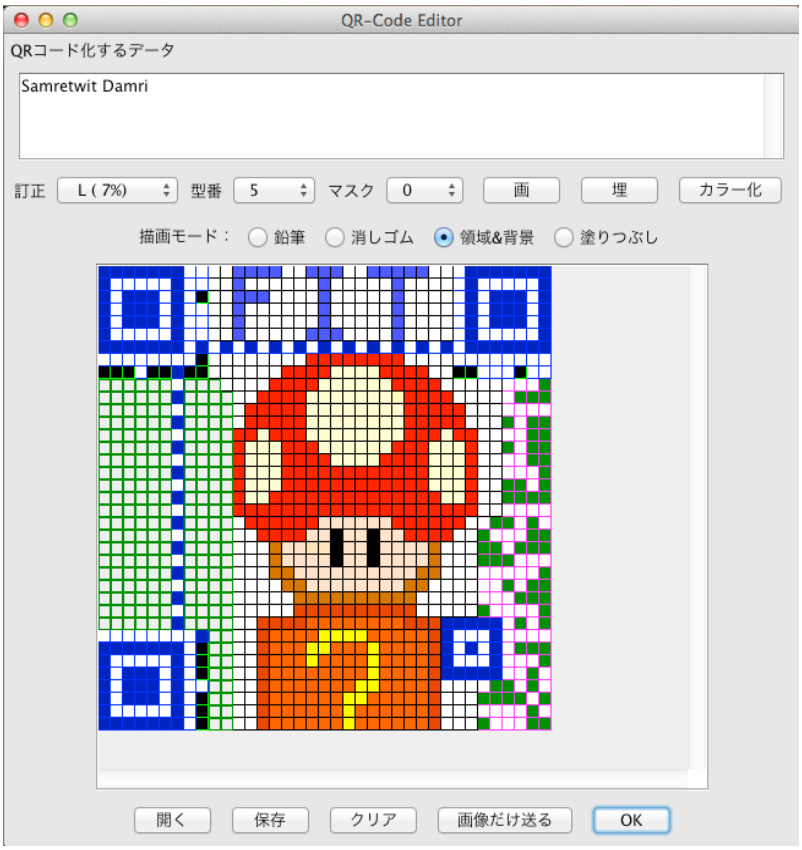


図 5.42 ドット編集の例

図 5.43 作成したゲームキャラクターのカラードット二次元コード

作成した例を図 5.42，図 5.43 に示す，プロトタイプを用いて描いた絵のシミュレーションとその二次元コードである．このカラー二次元コードは実験で利用した“i-nigma”，“QRReader for iPhone”および“Best Barcode Reader”の QR デコーダで読み取る事は可能であった．しかし，QR デコーダのカラーの認識アルゴリズムはそれぞれ異なっているのでこれ以外の QR デコーダアルゴリズムや誤り訂正能力などの設定によっては QR デコーダ次第で読み取れない可能性もある．

図 5.44 に示すように，研究員や他人の頼まれてこのソフトで作成してみてもらったカラー二次元コードである．

図 5.44　作成してみてもらったカラー二次元コード

第 6 章　考察

6.1 全体の利用状況への考察

1.　照明条件

本研究の実験は，通常照明(蛍光灯)での部屋内で行っていた．同じディスプレイやモニタの状態によっては部屋の条件が変わりその影響を受けて結果が少し変化した．しかし，読み取り特性に関してはあまり影響を受けないので，結果はほとんど変わらなかった．その後，昼間の屋外で同じ状態で実験したところ，太陽の光が輝き過ぎてモニタの画面がよく見えず読み取りが不可能になった．これは周りの明るさが強過ぎて液晶画面が読み取れなくなったせいである．印刷紙では実験を行っていないが，紙には明るさの影響が少ないので室内の読み取り特性とほぼ同じであると思われる．

2.　モニタリング条件

実験では MacBook Air 2010 の表示装置を利用して行った．このノートパソコンの表示の明るさを変更したりして, 実験を行った. 普通の明るさの 80%の明るさで行い正確に読み取ることが出来た．しかし，画面の明るさを変更するとその結果は変わり，明るさを落とせば落とすほど二次元コードの画像は見難くなって来て，最後は見えなくて読み取れないようになった．つまり，同じ二次元コードの画像でも画面の明るさを暗くすると，デコーダは認識できず読み取れなくなる事が分かった．

3.　読み取るプログラム条件

この実験では iPhone 用の 3 つの読み取りプログラムを利用して実験を行った．しかし，それぞれ少し異なる結果が出た. RGB や HSB の三属性を変えて読み取れるかどうかはソフトのアルゴリズム次第である．“QRReader for iPhone”は黄色やシアンをかなり“明”と認識する．しかし，“Best Barcode Reader”はこの 2 つの色の認識は曖昧であって，“明”と認識する事もあり，“暗”と認識する事もあった．つまり，あるソフトを利用してカラードット二次元コードを読み取れても他のソフトでは読み取れない可能性がある．

6.2 ユーザの意見に対する考察

この実験でユーザ利用実験を行い，ユーザの反応はかなり良かった．カラードットの二次元コードはあまり見えないからである．しかし，冗長エリア以外にドット絵を描く事が出来るのかや編集する事はできるかとよく聞かれた．冗長エリア以外にはドット絵などを描くことが出来ず，冗長エリア以外のエリアにはデータコード語や誤り訂正ワード語など重要な情報が入っているので，編集する事は出来ない．データコード語は読み取に必要な情報が入っており，誤り訂正コード語は誤りを訂正する情報が入っている．また，位置検出パターンや位置合せパターンは読み取り角度を補正するために使われるので，編集してはいけない．これ以外の冗長な部分はある程度の誤り訂正を利用して，少しは編集できる．しかし，誤り訂正能力が低下することになるのであまり良くない．さらに，本プログラムを日本語，タイ語をはじめ，英語以外の言語で利用できることが要望されている．

第 7 章　結論

1. まとめ

カラードット絵二次元コードエディタ（カラーQREditor）のプロトタイプを試作し，二次元コードのデータが入っていない冗長エリアを利用してイメージを重畳し，色相，彩度，明度を変化させて QR コードの読み取り実験を行った. その結果, カラーバーの RGB 値または HSB 値を変化させ, 自由に読み取り可能なカラードット絵を描く事が可能であることも確認した.

試作したカラードット絵二次元コードエディタ（カラーQREditor）で作成した二次元コードを作成するとともにカラーイメージも重畳させることが出来，イメージを重畳したカラー二次元コードの読み取り特性を測定した.

1. イメージを重畳する位置および大きさによってその読み取り特性が変わるが，左中央の位置に重畳する場合が最適で，ほぼ誤り訂正能力に近い値の大きさのイメージまで重畳できることがわかった.
2. イメージ付きカラー化二次元コードは QR コードの冗長エリアにドット絵などイメージをデザインしてカラー化したものであり, 既存の QR デコーダをそのまま用いても，二次元コードの白ドットを明るい色に黒ドットを暗い色に変換する限り, 既存の QR デコーダで読み取り可能であることがわかった.
3. また，RGB の値からモノクロ輝度値に変換して実験することにより，イメージ付きカラー化二次元コードは，誤り訂正の影響を受ける事なく認識でき，白黒からカラー化することにより見栄えが良くなる.
4. RGB からモノクロ化する際の数式は，輝度信号を求める式と同じであり，この値が 90%以上の場合は“明”と認識され，50%以下の場合は“暗”に認識され，これらの中間値の場合は周辺の色の値により“明”または“暗“に認識される.
5. 色相の違いにより，黄色などはかなり“明”と認識され，青色などはかなり“暗”と認識されるさらに，ピンク色をはじめ，“明”と“暗”の両方に認識される色もある.

本システムの利用状況が 3 つのデコーダだけを利用して計算の結果は十分ではなかったが，プロトタイプはある程度カラー絵ドットの描く事ができるようになった．簡単で使いやすく，ドットのカラーをある程度を選択でき, その選んだ色の輝度値が見せてユーザーは“明”また“暗”を識別できる.

2. 今後の課題

残された課題として，以下のものが挙げられる.

- ユーザが利用しやすいように評価システムの形式を見直す.
- カラーの明暗の識別の計算を詳細化し，iPhone 以外の他の携帯の機種でも実験する.
- ユーザーがさらに楽しめるコンテンツを追加する.
- 日本語やタイ語などに拡張する.

参考文献

[1] QRCode http://www.qrcode.com/aboutqr.html

[2] 二次元コードシンボル QR コード 基本仕様(JISX0510) 日本規格協会 2004 年

[3] Gimp http://www.gimp.org/ http://en.wikipedia.org/wiki/GIMP

[4] NetBeans http://netbeans.org/ http://ja.wikipedia.org/wiki/NetBeans

[5] Java http://www.oracle.com/jp/technologies/java/overview/index.html http://ja.wikipedia.org/wiki/Java

[6] QRForest http://qrforest.com/

[7] QR-JAM http://staff.aist.go.jp/hagiwara.hagiwara/qrjam/index.html

[8] DesignQR http://d-qr.net/index.htm

[9] 藤田和謙, 栗林稔, 森井昌克, “QR コードへの画像埋め込みに関する検討と提案”, 信学技報 LOIS2010-51, pp.39-44, Jan.2011

[10] 小野智司, 森永健介, 中山茂: “最適化アルゴリズムを用いたアニメーション QR コードの作成”, 芸術科学会論文誌, Vol.8, No.1, pp.25-34 (2009).

[11] 越智祐樹 : “QR コード用ドット絵画像処理法の検討”, 信学技術 IEICE Technical Report MuMC2009-50(2009-11)

[12] 若原 俊彦, 山元 規靖, 越智 祐樹 ; ” QR コードの画像重畳評価法の検討 ”, 信学技報, vol. 110, no. 450, LOIS2010-65, pp. 1-5（2011 年 3 月）

[13] 萩原 学, “デザイン二次元コード”, 電子情報通信学会誌 Vol.94, No.4, PP.341-343 2011.

[14] 『色彩学概説』 千々岩 英彰 東京大学出版会

[15] 尾登誠一「3 色の世界を知る」『色彩楽のすすめ』岩波書店〈岩波アクティブ新書〉, 2004 年, 34 頁. ISBN 4-00-700101-4

[16] 大日精化工業株式会社 色彩用語解説 http://www.daicolor.co.jp/color/color_01.html

[17] i-nigma http://www.i-nigma.com/i-nigmahp.html

[18] QRReader for iPhone http://itunes.apple.com/us/app/qr-reader-for-iphone/id368494609?mt=8

[19] Best Barcode Reader https://itunes.apple.com/us/app/best-barcode-scanner-scan/id454087075?mt=8

発表文献

（１） Samretwit Damri，若原俊彦：”イメージを重畳した二次元コードの読取り特性の研究”，平成 23 年度第 19 回電子情報通信学会九州支部学生会講演会 D-54（2011 年 9 月）

（２） Damri Samretwit, Toshihiko Wakahara,;"A Study on the Superimposed Images of Two-Dimensional Code", Third International Conference on Intelligent Networking and Collaborative Systems (INCoS 2011) MNSA-2011 (2011 Nov.)

（３） サムレットウィット ダムリ・若原俊彦"二次元コードのイメージ損傷に対する読み取り特性改善の研究"，信学技報, vol. 111, no.470, LOIS2011- 86, pp.79 - 84（2012 年 3 月）

（４） サムレットウィット ダムリ，若原俊彦：”カラー二次元コードの表示条件に対する読み取り特性の検討”，平成２4 年度電子情報通信学会九州支部第 20 回学生会講演会 D - 52（2012 年 9 月）

（５） Toshihiko Wakahara, Damri Samretwit and Noriyasu Yamamoto,"A Study on the Reading Characteristics of Colored 2-Dimensional Code", Session:Information Networking and Wireless Communication Systems, 15th International Conference on Network-based Information Systems(NBiS2012) Sept. 2012

（６） サムレットウィット ダムリ,若原俊彦：”カラー化イメージ付き二次元コードの読み取り特性の検討",信学技報, vol.112 , no.308 , MOMUC2012-34, pp.13-18（2012 年 11 月）

（７） サムレットウィット ダムリ，若原俊彦：カラー化ドット絵付二次元コードエディタの構成，電子情報通信学会 2013 年総合大会　発表予定

謝辞

本研究を進めるに当たり，ご指導を頂いた修士論文指導教員の若原俊彦教授に感謝致します．また，卒業研究でお世話になった若原研究室の皆様に感謝します．

付録 A　ソフトウエア

1 Java

Java は汎用で同時にのクラスベースのオブジェクト指向のコンピュータ·プログラミング言語である．具体的に可能の限りいくつかの実装の依存関係として持つように設計されている．アプリケーション開発者は"一度書けばどこでも動く"ようにすることを意図している．1 つのプラットフォーム上で実行されるコードは，別で実行するには，再コンパイルする必要がないという意味である．Java アプリケーションは通常にバイトコード（クラスファイル）にコンパイルされる．バイトコードはどんな Java 仮想マシン上で実行することができる．Java は特にクライアント·サーバ，Web アプリケーションのため，使用されている最も一般的なプログラミング言語の一つである．

2 NetBeans

NetBeans は Java または PHP，C/C++，HTML5 をはじめ他の言語をは開発するための統合開発環境である．また，NetBeans は Java のデスクトップアプリケーションや他のアプリケーション·プラットフォーム·フレームワークである．NetBeans IDE は Java で書かれており，Windows，OS X，Linux，Solaris, および互換性のある JVM をサポートする他のプラットフォーム上で実行することができる．NetBeans プラットフォームはモジュールと呼ばれるコンポーネントのセットでアプリケーション開発されることができる．NetBeans プラットフォームに基づくアプリケーションは，サードパーティの開発者が拡張することができる．

3 GIMP

GIMP（ギンプ，ジンプ，GNU Image Manipulation Program）レタッチや画像編集のソフトであり，無料にオープンソースソフトウェアとして GPLv3 ライセンスの下でリリースされる．Microsoft Windows，Mac OS 及び Linux を含み，ほとんどのオペレーティングシステムに合わせた GIMP のバージョンがある．GIMP はレタッチや編集，フリーフォーム描画，リサイズ，トリミング，フォトモ

ンタージュ，異なる画像フォーマット間の変換や，その他の特殊なタスクに使用されるイメージのツールを持っている．GIF や MPEG ファイルなどのアニメーション画像がアニメーションのプラグインを使用して作成できる．

付録 B　プログラム

- QRCode.java に追加したコード

```
class QRCode extends JFrame implements ActionListener{
        JButton clearB;
        public static int editMode = 0;

        //サイズ
                sizeList = new JComboBox();
                sizeList.addItem("1");
                sizeList.addItem("2");
                sizeList.addItem("3");
                sizeList.addItem("4");
                sizeList.addItem("5");
                sizeList.addItem("6");
                sizeList.setLightWeightPopupEnabled(false); //重量コンポーネント化してプルダウンが表示されない不具合解消
                sizeList.addActionListener(this);
        public void actionPerformed(ActionEvent e) {
                //ボタン(クリア)のイベント
                if(obj==clearB || obj==Clear){
                                inputD.setEditable(true);
                                inputD.setText("");
                                editMode = 0;
                }
                if(obj==sizeList){
                                QRShow.blockWidth=sizeList.getSelectedIndex()+4;
                                QRShow.quiet=QRShow.blockWidth*5;
                }
        }
}
        class QRShow extends JFrame{
          ImageIcon icon = new ImageIcon("icon.png");                                                    //アイコンの設定
          public static GPanel[] Grap=new GPanel[QRCode.simbolNumMax]; //グラフィックパネルの型
          public static int blockWidth= 4;
          //ブロックの大きさを格納
```

```
  public static int quiet=blockWidth*5;                                                    //クワ
イエットゾーンの大きさを格納
  public int windowSizeX;
                        //ウインドウの大きさを格納
  public int windowSizeY;
                        //ウインドウの大きさを格納
  BufferedImage readImage=null;
  int imageX=0;
  int imageY=0;
  QRShow(int simbolNum,int[][] simbol,int simSize,BufferedImage readImage,int imageX,int imageY){
            //グラフィックパネルインスタンス生成
                        Grap[simbolNum]=new GPanel(simSize,simbol,readImage,imageX,imageY);
                        //サイズ調整
                                    windowSizeX=simSize*blockWidth+(quiet*2);
                                    windowSizeY=windowSizeX;
                                    Grap[simbolNum].setPreferredSize(new Dimension(windowSizeX, windowSizeY));
            //表示
                        setSize(windowSizeX+5,windowSizeY+25);
                        System.out.println("simSize="+simSize);
                        setIconImage(icon.getImage());
                        getContentPane().add(Grap[simbolNum],BorderLayout.CENTER);
                        setVisible(true);
  }
}
class GPanel extends JPanel{
  int[][] simbol;
  int simSize=0;
  BufferedImage readImage=null;
  int imageX=0;
  int imageY=0;

          public static Color NO_DATA_COLOR = Color.BLACK;
          public static Color BLACK_DATA_COLOR = Color.BLACK;
          public static Color DATA_COLOR = Color.BLACK;
          public static Color DATA_VER_COLOR = Color.BLACK;
          public static Color DATA_K_COLOR = Color.BLACK;
          public static Color BACKGROUND_COLOR = Color.WHITE;
```

```
        public static Color[][] simbolC;

  GPanel(int a,int[][] b,BufferedImage c,int d,int e){
            setBackground(Color.white);
            simSize=a;
            simbol=new int[simSize][];
            for(int i=0;i<simSize;i++){
                    simbol[i]=(int[])b[i].clone();
            }
            readImage=c;
            imageX=d;
            imageY=e;
  }
  public void paintComponent(Graphics g) {
  super.paintComponent(g);
            //QR コード描画
                    g.setColor(Color.BLACK);
                    for(int i=0;i<simSize;i++){
                            for(int j=0;j<simSize;j++){
                                    if(QRCode.editMode == 0){
                                        if(simbol[i][j]==QRCode.BLACK_DATA                                    ||
simbol[i][j]==QRCode.DATA_1 || simbol[i][j]==QRCode.DATA_VER || simbol[i][j]==QRCode.DATA_K){

  g.fillRect(j*QRShow.blockWidth+QRShow.quiet,i*QRShow.blockWidth+QRShow.quiet,QRShow.blockWidth,QRShow.
blockWidth);
                                        }
                                    }
                                    else{
                                        switch(simbol[i][j]){
                                                case QRCode.BLACK_DATA:
                                                    g.setColor(BLACK_DATA_COLOR);
                                                    break;
                                                case QRCode.DATA_1:
                                                    if(simbolC[i][j]==NO_DATA_COLOR){
                                                        g.setColor(NO_DATA_COLOR);
                                                    }
                                                    else {
                                                        g.setColor(simbolC[i][j]);
```

```
                                                    }
                                                    break;
                                                case QRCode.DATA_VER:
                                                    g.setColor(DATA_VER_COLOR);
                                                    break;
                                                case QRCode.DATA_K:
                                                    g.setColor(DATA_K_COLOR);
                                                    break;
                                                case QRCode.DATA_0:
                                                    if(simbolC[i][j]==NO_DATA_COLOR){
                                                        g.setColor(BACKGROUND_COLOR);
                                                    }
                                                    else{
                                                        g.setColor(simbolC[i][j]);
                                                    }
                                                    break;
                                                default:
                                                    g.setColor(BACKGROUND_COLOR);
                                                    break;
                                            }

        g.fillRect(j*QRShow.blockWidth+QRShow.quiet,i*QRShow.blockWidth+QRShow.quiet,QRShow.blockWidth,QRShow.bl
        ockWidth);
                                        }
                                    }
                                }
                            //読み込み画像表示
                                if(readImage!=null){
                                    Graphics2D g2 = (Graphics2D)g;
                                    g2.drawImage(readImage, imageX+QRShow.quiet, imageY+QRShow.quiet, this);
                                }
                        }
                    }
```

- QREditor.java に追加したコード

```
class QREditor extends JFrame implements ActionListener,KeyListener{
        public static Color[][] simbolC;
        QREditor(){}
```

```
            simbolC=new Color[simSize][simSize];
                    //カラー化ボタン
                colorB=new JButton("カラー化");
                colorB.addActionListener(this);
                colorB.addKeyListener(this);
        }
}
public void actionPerformed(ActionEvent e) {
                    //ボタン(カラー化)のイベント
                        if(obj==colorB){
                                tc = new ColorValueSliderControl();
                                tc.setLocation(0, 0);
                        }
            //ボタン(OK)のイベント
                        if(obj==okB){
                                data_out();
                                QRCode.lv=lv;
                                QRCode.encodeMode=encodeMode;
                                QRCode.ver=ver;
                                QRCode.inputD.setText(outstr);
                                QRCode.verList.setSelectedIndex(ver);
                                QRCode.lvList.setSelectedIndex(lv);
                                QRCode.maskList.setSelectedIndex(mask+1);

                                    GPanel.NO_DATA_COLOR = EGPanel.NO_DATA_COLOR;
                                    GPanel.BLACK_DATA_COLOR = EGPanel.BLACK_DATA_COLOR;
                                    GPanel.DATA_COLOR = EGPanel.DATA_COLOR;
                                    GPanel.DATA_VER_COLOR = EGPanel.DATA_VER_COLOR;
                                    GPanel.DATA_K_COLOR = EGPanel.DATA_K_COLOR;
                                    GPanel.BACKGROUND_COLOR = EGPanel.BACKGROUND_COLOR;

                                    GPanel.simbolC = QREditor.simbolC;

                                    QRCode.editMode = 1;
                                    QRCode.inputD.setEditable(false);

                                if(readImage!=null && EGPanel.imgVisible){
                                        QRCode.readImage=readImageBuf;
```

```
            QRCode.imageX=(int)(EGPanel.imageX/(float)(EGPanel.blockSize/QRShow.blockWidth));

            QRCode.imageY=(int)(EGPanel.imageY/(float)(EGPanel.blockSize/QRShow.blockWidth));
                                    }
                                                        }
                    //コンボボックス(マスク)のイベント
                            if(obj==maskList){
                                    mask=maskList.getSelectedIndex();
                                    init();
                            }
                    //ラジオボタン(鉛筆)のイベント
                            if(obj==RadioB_pencil){
                                    paintMode=0;
                            }
                    //ラジオボタン(消しゴム)のイベント
                            if(obj==RadioB_eraser){
                                    paintMode=1;
                            }
                    //ラジオボタン(鉛筆&消しゴム)のイベント
                            if(obj==RadioB_both){
                                    paintMode=2;
                            }
                    //ラジオボタン(塗りつぶし)のイベント
                            if(obj==RadioB_fill){
                                    paintMode=3;
                            }
        }
public void init (){
                    simbolC = new Color[simSize][simSize];
}
public void init (){
                    simbolC = new Color[simSize][simSize];
}
public void setBit(int simSize,int[] haitiBit,int[][] simbol){
            if(nowX>=0 && nowX<=simSize-1 && nowY>=0 && nowY<=simSize-1){
            //データをそれぞれ配置
                    if(simbol[nowY][nowX]==0){
```

```
                        if(haitiBit[tmp[1]]==DATA_E1){
                                simbol[nowY][nowX]=DATA_E1;
                        }else if(haitiBit[tmp[1]]==DATA_E0){
                                simbol[nowY][nowX]=DATA_E0;
                        }else if(haitiBit[tmp[1]]==DATA_0){
                                simbol[nowY][nowX]=DATA_0;
                            simbolC[nowY][nowX]=Color.WHITE;
                        }
                        tmp[0]=1;
                        tmp[1]++;
                }
        }
}
class EGPanel extends JPanel{
        //DataBase からデータ取得
                public static final int NO_DATA = DataBase.NO_DATA;
        //定数(データ無)
                public static final int BLACK_DATA = DataBase.BLACK_DATA;                //定数(黒データ)
                public static final int WHITE_DATA = DataBase.WHITE_DATA;                //定数(白データ)
                public static final int K_DATA = DataBase.K_DATA;                        //定数
(形式，型番予定)
                public static final int DATA_1 = DataBase.DATA_1;                        //定数
(コードデータ 1)
                public static final int DATA_0 = DataBase.DATA_0;                        //定数
(コードデータ 0)
                public static final int DATA_VER = DataBase.DATA_VER;                    //定数
(型番情報)
                public static final int DATA_K = DataBase.DATA_K;                        //定数
(形式情報)
                public static final int DATA_E1 = DataBase.DATA_E1;                      //定数
(エディタでの仮コードデータ 1)
                public static final int DATA_E0 = DataBase.DATA_E0;                      //定数
(エディタでの仮コードデータ 0)
                public static final String BR = DataBase.BR;                             //定数
(改行コード)
        //ドットのカラー
                public static Color rgbValue = Color.BLACK;
                public static double briV = 0;
```

```
        public static Color NO_DATA_COLOR = Color.BLACK;
        public static Color BLACK_DATA_COLOR = Color.BLACK;
        public static Color K_DATA_COLOR = Color.GRAY;
        public static Color DATA_COLOR = Color.BLACK;
        public static Color DATA_VER_COLOR = Color.BLACK;
        public static Color DATA_K_COLOR = Color.BLACK;
        public static Color DATA_E_COLOR = Color.BLACK;
        public static Color BACKGROUND_COLOR = Color.WHITE;
//その他
            static int quiet=1;
            static int blockWidth=9;
            static int blockSize=blockWidth+quiet;
            static int imageX=0;
            static int imageY=0;
            static boolean imgVisible=true;
                    //画像ファイルの表示切替
            int nowX=-1;
            int nowY=-1;
        static int regBlack = 60;
        static int regWhite = 80;
  EGPanel(){

  }
  public void paintComponent(Graphics g) {
  super.paintComponent(g);
          //QR コード外枠描画
                  g.setColor(Color.black);
                  for(int i=0;i<QREditor.simSize;i++){
                          for(int j=0;j<QREditor.simSize;j++){
                                  switch(QREditor.simbol[j][i]){
                                          case NO_DATA:
                                                  //データ未定義
                                          QREditor.simbolC[j][i] = NO_DATA_COLOR;
                                                  g.setColor(NO_DATA_COLOR);

  g.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                                  break;
                                          case BLACK_DATA:
```

```
                                        //黒データ
                                    g.setColor(Color.BLUE);

g.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                        g.setColor(BLACK_DATA_COLOR);

g.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                        break;
                                    case WHITE_DATA:
                                        //白データ
                                    g.setColor(Color.BLUE);

g.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                        g.setColor(BACKGROUND_COLOR);

g.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                        break;
                                    case K_DATA:
                                        //形式，型番情報が入る予定地
                                        g.setColor(K_DATA_COLOR);

g.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                        break;
                                    case DATA_1:
                                        //データ 1
                                    g.setColor(Color.BLACK);

g.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                        g.setColor(QREditor.simbolC[j][i]);

g.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                        break;
                                    case DATA_0:
                                        //データ 0
                                        g.setColor(Color.BLACK);

g.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                        g.setColor(QREditor.simbolC[j][i]);
```

```
g.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                        //System.out.println(QREditor.simbolC[j][i].getRed());
                                        break;
                                        case DATA_VER:
                                                //型番情報
                                        g.setColor(Color.GREEN);

g.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                                g.setColor(DATA_VER_COLOR);

g.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                                break;
                                        case DATA_K:
                                                //型番情報
                                        g.setColor(Color.GRAY);

g.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                                g.setColor(DATA_K_COLOR);

g.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                                break;
                                        case DATA_E1:
                                                //エディタ固定データ 1
                                        QREditor.simbolC[j][i] = DATA_E_COLOR;
                                        g.setColor(Color.MAGENTA);

g.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                                g.setColor(DATA_E_COLOR);

g.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                                break;
                                        case DATA_E0:
                                                //エディタ固定データ 0
                                                QREditor.simbolC[j][i]                  =
BACKGROUND_COLOR;
                                        g.setColor(Color.MAGENTA);
```

```
g.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                                    g.setColor(BACKGROUND_COLOR);

g.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                                                        break;
                                                default:
                                                        //エラー
                                                        System.out.println("エラー：　シンボルのデータがおかしいです.　："+QREditor.simbol[i][j]);
                                                        QRCode.info.setText(QRCode.info.getText()+"エラー：　シンボルのデータがおかしいです.　："+QREditor.simbol[i][j]+BR);
                                                }
                                        }
                                }
                        //読み込み画像表示
                                if(imgVisible && QREditor.readImage!=null){
                                        Graphics2D g2 = (Graphics2D)g;
                                        g2.drawImage(QREditor.readImage, imageX, imageY, this);
                                }
        }
        public void Clicked(int x,int y,int paintMode){
                if(x>=0 && y>=0 && x<=QREditor.simSize*(blockWidth+quiet) && y<=QREditor.simSize*(blockWidth+quiet)){
                                x=(int)x/(blockWidth+quiet);
                                y=(int)y/(blockWidth+quiet);
                                if(nowX!=x || nowY!=y){
                                        nowX=x;
                                        nowY=y;
                                        if((QREditor.simbol[y][x]==DATA_0||QREditor.simbol[y][x]==DATA_1) && paintMode==1){
                                                //消しゴム&両方
                                                QREditor.simbol[y][x]=DATA_0;
                                            QREditor.simbolC[y][x] = BACKGROUND_COLOR;
                                        }else   if((QREditor.simbol[y][x]==DATA_0||QREditor.simbol[y][x]==DATA_1)   && paintMode==0){
                                                //鉛筆&両方
                                               if(briV<regBlack)
                                               {
```

```
                    QREditor.simbol[y][x]=DATA_1;
                    QREditor.simbolC[y][x] = rgbValue;
                }else if(briV>regWhite)
                {
                    QREditor.simbol[y][x]=DATA_0;
                    QREditor.simbolC[y][x] = rgbValue;
                }
                  }else if(QREditor.simbol[y][x]==DATA_0 && paintMode==3){
                        //塗りつぶし
                  }else if(QREditor.simbol[y][x]==NO_DATA&&briV<regBlack && paintMode==2){
                        NO_DATA_COLOR = rgbValue;
                  }
                else if(QREditor.simbol[y][x]==BLACK_DATA&&briV<regBlack && paintMode==2){
                        BLACK_DATA_COLOR = rgbValue;
                  }
                else if(QREditor.simbol[y][x]==K_DATA && paintMode==2){
                        //未定義
                  }
                else if(QREditor.simbol[y][x]==DATA_VER&&briV<regBlack && paintMode==2){
                        DATA_VER_COLOR = rgbValue;
                  }
                else if(QREditor.simbol[y][x]==DATA_K){
                        DATA_K_COLOR = rgbValue;
                  }
                else if(QREditor.simbol[y][x]==DATA_E1&&briV<regBlack && paintMode==2){
                        DATA_E_COLOR = rgbValue;
                  }
                else
if((QREditor.simbol[y][x]==WHITE_DATA||QREditor.simbol[y][x]==DATA_E0)&&briV>regWhite && paintMode==2){
                    BACKGROUND_COLOR = rgbValue;
                  }
                        //未定義
                repaint();
            }
        }else{
            System.out.println("枠外です");
        }
    }
```

```
        public void Released(){
            nowX=-1;
            nowY=-1;
        }
        static void bitTurn(int x,int y){
            if(x>=0 && x<QREditor.simSize && y>=0 && y<QREditor.simSize){
                if(QREditor.simbol[y][x]==DATA_0){
                    QREditor.simbol[y][x]=DATA_1;
                }else if(QREditor.simbol[y][x]==DATA_1){
                    QREditor.simbol[y][x]=DATA_0;
                }
            }
        }
}
```

- 追加した ColorValueSliderControl ファイル

```
import java.awt.BorderLayout;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;

import javax.swing.*;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.border.TitledBorder;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class ColorValueSliderControl extends JFrame {
   public ColorValueSliderControl() {
      getContentPane().add(new TColor());
      //setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(500, 600);
```

```
        setVisible(true);
    }
}

class TColor extends JPanel implements ActionListener {

    public static final double RV = 0.294;
    public static final double GV = 0.615;
    public static final double BV = 0.091;

    TColor.DrawingCanvas canvas = new TColor.DrawingCanvas();
    TColor.DrawingCanvas currentC = new TColor.DrawingCanvas();
    JLabel rgbValue = new JLabel("000000");
    JLabel bValue = new JLabel("0");
    JButton okB;

    JSlider sliderR, sliderG, sliderB, sliderH, sliderS, sliderA,
            sliderAlpha;

    public TColor() {
        sliderR = getSlider(0, 255, 0, 50, 5);
        sliderG = getSlider(0, 255, 0, 50, 5);
        sliderB = getSlider(0, 255, 0, 50, 5);
        sliderH = getSlider(0, 360, 0, 120, 60);
        sliderS = getSlider(0, 255, 0, 50, 5);
        sliderA = getSlider(0, 255, 0, 50, 5);
        sliderAlpha = getSlider(0, 255, 255, 50, 5);

        JPanel panel = new JPanel();
        //OK ボタン
        okB=new JButton("OK");
        okB.addActionListener(this);

        panel.setLayout(new GridLayout(7, 2, 15, 0));

        panel.add(new JLabel("R-G-B Sliders (0 - 255)"));
        panel.add(new JLabel("H-S-B Sliders (ex-1)"));
        panel.add(sliderR);
```

```
    panel.add(sliderH);
    panel.add(sliderG);
    panel.add(sliderS);
    panel.add(sliderB);
    panel.add(sliderA);

    panel.add(new JLabel("Alpha Adjustment (0 - 255): ", JLabel.RIGHT));
    panel.add(sliderAlpha);

    panel.add(new JLabel("RGB 値: ", JLabel.RIGHT));

    rgbValue.setBackground(Color.white);
    rgbValue.setForeground(Color.black);
    rgbValue.setOpaque(true);
    panel.add(rgbValue);

    panel.add(new JLabel("輝度値  (MAX:100%): ", JLabel.RIGHT));

    bValue.setBackground(Color.white);
    bValue.setForeground(Color.black);
    bValue.setOpaque(true);
    panel.add(bValue);

    add(panel, BorderLayout.SOUTH);
    add(currentC);
    add(new JLabel("現在の色"));
    add(canvas);
    add(new JLabel("選択色"));
    add(okB);
}
//////////////////////////////////////////////イベントメソッド//////////////////////////////////////
        public void actionPerformed(ActionEvent e) {
            Object obj = e.getSource();
       //ボタン(OK)のイベント
                                if(obj==okB){
                                double briT;
                                currentC.setBackground(canvas.color);
                                currentC.repaint();
```

```
                                EGPanel.rgbValue = canvas.color;
                                briT = Double.parseDouble(bValue.getText());
                                EGPanel.briV = briT;
                                if(briT>50&&briT<90){
                                    if(briT>80){
                                        JOptionPane.showMessageDialog(null, "あなたが選択した色は“明”と設定されます"
                                                + "があるソフトで読み取れない事もありますのでご注意ください．¥n"
                                                + "安全の 90%以上の輝度値の“明”がおすすめです．");
                                    }else if(briT<60){
                                        JOptionPane.showMessageDialog(null, "あなたが選択した色は“暗”と設定されます"
                                                + "があるソフトで読み取れない事もありますのでご注意ください．¥n"
                                                + "安全の 50%以下の輝度値の“暗”がおすすめです．");
                                    }else{
                                        JOptionPane.showMessageDialog(null, "あなたが選択した色は“明”また“暗”と認識"
                                                + "できないようです．¥n"
                                                + "安全の輝度値は 90%以上の“明”また 50%以下の“暗”を選んでください.
");
                                    }
                                }
                                }
            }

    public JSlider getSlider(int min, int max, int init, int mjrTkSp, int mnrTkSp) {
        JSlider slider = new JSlider(JSlider.HORIZONTAL, min, max, init);
        slider.setPaintTicks(true);
        slider.setMajorTickSpacing(mjrTkSp);
        slider.setMinorTickSpacing(mnrTkSp);
        slider.setPaintLabels(true);
        slider.addChangeListener(new TColor.SliderListener());
        return slider;
    }

    class DrawingCanvas extends Canvas {
        Color color;
        int redValue, greenValue, blueValue;
        int alphaValue = 255;
        float[] hsbValues = new float[3];
```

```
    public DrawingCanvas() {
        setSize(100, 100);
        color = new Color(0, 0, 0);
        setBackgroundColor();
    }

    public void setBackgroundColor() {
        color = new Color(redValue, greenValue, blueValue, alphaValue);
        setBackground(color);
    }
}

class SliderListener implements ChangeListener {
    public void stateChanged(ChangeEvent e) {
        JSlider slider = (JSlider) e.getSource();
        if (slider == sliderAlpha) {
            canvas.alphaValue = slider.getValue();
            canvas.setBackgroundColor();
        } else if (slider == sliderR) {
            canvas.redValue = slider.getValue();
            displayRGBColor();
        } else if (slider == sliderG) {
            canvas.greenValue = slider.getValue();
            displayRGBColor();
        } else if (slider == sliderB) {
            canvas.blueValue = slider.getValue();
            displayRGBColor();
        } else if (slider == sliderH) {
            canvas.hsbValues[0] = (float) (slider.getValue() / 360.0);
            displayHSBColor();
        } else if (slider == sliderS) {
            canvas.hsbValues[1] = (float) (slider.getValue() / 255.0);
            displayHSBColor();
        } else if (slider == sliderA) {
            canvas.hsbValues[2] = (float) (slider.getValue() / 255.0);
            displayHSBColor();
        }
        canvas.repaint();
```

```
    }

    public void displayRGBColor() {
        canvas.setBackgroundColor();
        Color.RGBtoHSB(canvas.redValue, canvas.greenValue, canvas.blueValue,canvas.hsbValues);
        sliderH.setValue((int) (canvas.hsbValues[0] * 360));
        sliderS.setValue((int) (canvas.hsbValues[1] * 255));
        sliderA.setValue((int) (canvas.hsbValues[2] * 255));
        double briV = (((double)canvas.redValue*RV)+((double)canvas.greenValue*GV) + ((double)canvas.blueValue*BV))/2.55;
        rgbValue.setText(Integer.toString(canvas.color.getRGB() & 0xffffff, 16));

        bValue.setText(Float.toString((float)briV));
    }

    public void displayHSBColor() {
        canvas.color = Color.getHSBColor(canvas.hsbValues[0],
                canvas.hsbValues[1], canvas.hsbValues[2]);
        canvas.redValue = canvas.color.getRed();
        canvas.greenValue = canvas.color.getGreen();
        canvas.blueValue = canvas.color.getBlue();

        sliderR.setValue(canvas.redValue);
        sliderG.setValue(canvas.greenValue);
        sliderB.setValue(canvas.blueValue);

        canvas.color = new Color(canvas.redValue, canvas.greenValue,
                canvas.blueValue, canvas.alphaValue);
        canvas.setBackground(canvas.color);
    }
  }
}
```