

目次

第 1 章 序論	1
第 2 章 QR コード	2
2.1 QR コードの概要	2
2.2 RS 符号	4
2.2.1 QR コード上の RS 符号	6
第 3 章 Aesthetic QR コード	8
3.1 目的画像の二値化手法	8
3.2 ランダム法	10
3.3 色変換手法	11
第 4 章 数式処理システム Maple を用いた Aesthetic QR コードの実装	13
4.1 数式処理システム Maple ^[8]	13
4.2 実験環境	13
4.3 Maple による QR コードの生成法の実装	14
4.4 Aesthetic QR コードの生成に関する実験	16
第 5 章 結論	18
謝辞	19
参考文献	20
付録 A プログラムリスト	21

第1章 序論

QR コード^[1]は、1994年に株式会社デンソーウェーブが開発した二次元バーコードであり、食品や製造業の在庫管理など多方面の分野で利用されている。一般的なQRコードは、白と黒の正方形のモジュールで構成されておりデザイン性を考慮していない。一方で広告、サービス業界ではデザイン性を考慮したQRコードが求められている。デザイン性を考慮したQRコードでは、一定のルールからQRコードを変更することによって、QRコード上にロゴ画像（以後、目的画像と述べる）を埋め込んだものがある。このようなQRコードをAesthetic QRコード^[2]という。

Aesthetic QRコードの研究は大きく三種類に分けることができる：

1. QRコードの一部に目的画像を埋め込む方法^[3],
2. 画像のヒストグラムを考慮して目的画像を埋め込む方法^[4],
3. QRコードに利用されているRS符号中のpadding codewordsと呼ばれる領域(以下、埋め草コード語)を考慮して目的画像を埋め込む方法^[2].

上で述べたAesthetic QRコードの中で、1と2は実装が行われ広く利用されている。しかし3については、視覚的に目的画像に近いAesthetic QRコードが作れるが、作成ツールが存在しない。そこで、本研究では、3番目の手法に分類される目的画像に近いAesthetic QRコードを自動生成するソフトウェアの開発について提案する。Aesthetic QRコードを自動生成する手法として、本研究では、Kuribayashiらの論文^[2]で提案されているRandom Methodのアルゴリズムを用いる。QRコードで用いられるRS符号の計算は代数拡大体上で計算されるため、本研究では数式処理システムMapleを用いてAesthetic QRコードを生成するソフトウェアを開発した。

以下、第2章ではQRコードを構成するReed-Solomon符号とQRコードの概要について述べ、第3章では本研究で用いるKuribayashiらの論文^[2]のRandom Method, Color Translationについて述べる。第4章では数式処理MapleによるAesthetic QRコードの実装とその結果について述べる。第5章では結論と今後の課題について述べる。

第2章 QRコード

この章では Aesthetic QR コードの構成要素である Reed-Solomon 符号（以下，RS 符号）と QR コードについて述べる。

2.1 QRコードの概要

QR コードの構成要素の最小単位は白と黒で表されるモジュールであり，白のモジュールは0のビット値を表し，黒のモジュールは1のビット値を表す．QR コードのサイズはバージョンによって決定され，そのバージョン (v) は1～40である．1型は， 21×21 モジュール，2型は， 25×25 モジュール，というように，型番が一つ上がるごとに一辺につき4モジュールずつ増加し，40型は， 177×177 モジュールとなる．したがって，バージョン v は $(17 + 4v) \times (17 + 4v)$ モジュールである．

図 2.1 に QR コードの構成要素を表す．

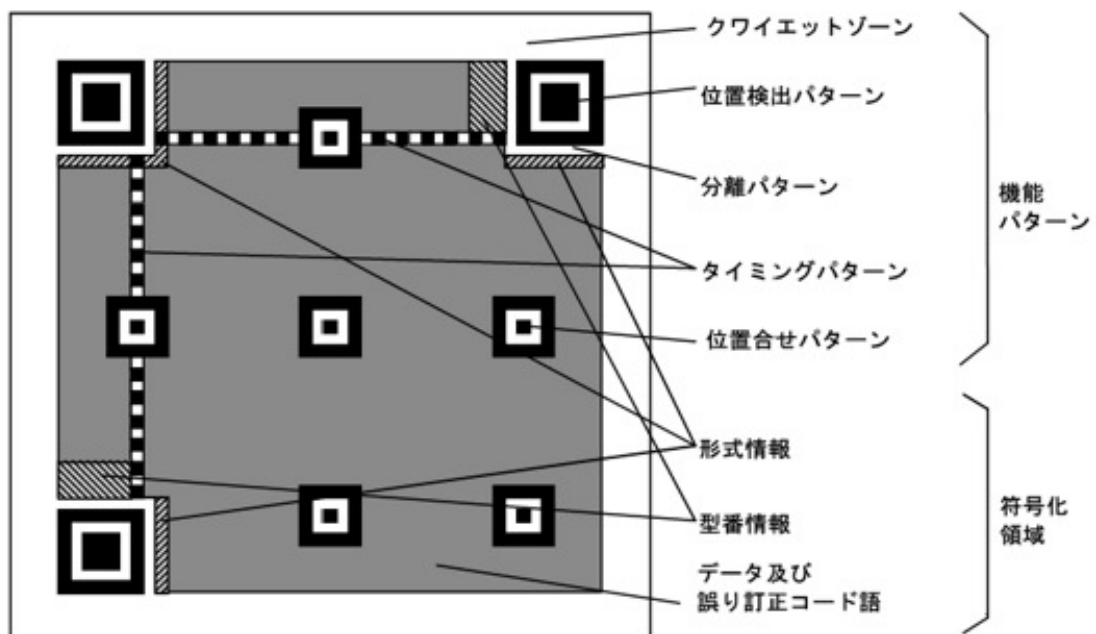


図 2.1 QRコードシンボルの構造^[9]

QRコードは、QRコード上にある符号化されたデータを正確に認識するために機能パターンを持っている。機能パターンは主に3つの構成要素から成り立っており、それぞれ位置検出パターン、位置合わせパターン、タイミングパターンと呼ばれる。

位置検出パターンはQRコードの左上、左下、右上の角にある3つの正方形のブロックである。それらの境目を明白にするために、形式情報との間に白いモジュールを置く。これを分離パターンと呼ぶ。

位置合わせパターンは小さな正方形のブロックで、位置検出パターンの垂直・平行座標に関係する位置に置く。バージョンによっては付加しない場合もあり、バージョン1には存在しない。

タイミングパターンは左上の位置検出パターンから右上の位置検出パターンへと、左上の位置検出パターンから左下の位置検出パターンへの白黒が交互に並ぶ2つのラインのことである。

QRコードのデータビットは、QRコードの右下から始まり、2モジュール幅の列上に配置する。列が最上部に達すると、次の2モジュール列は右端から始まり、下方向へ続く。現在の列が端に達すると、次の2モジュールの列に移動して方向を変更する。データビットは機能パターン（位置検出パターン、タイミングパターン、位置合わせパターン）の位置では、次のモジュールへ配置される。

上方向のデータビットの配置は図2.2に、下方向のデータビットの配置は図2.3に示す。

0	1
2	3
4	5
6	7

図 2.2 上方向のビット配列

6	7
4	5
2	3
0	1

図 2.3 下方向のビット配列

またデータの格納方法にも種類があり、英数字モードや8ビットバイトモードなどがある。

QRコードは、誤り訂正符号としてRS符号を使用し、その能力はL、M、Q、Hの4つのレベルに昇順で分類される。各誤り訂正レベルはQRコード内の全シンボルの約7%、約15%、約25%、約30%までのシンボルを訂正することができる。それぞれを表2.1に示す。

表 2.1 誤り訂正レベル

レベル	L	M	Q	H
誤り訂正能力	約 7%	約 15%	約 25%	約 30%

2.2 RS 符号

RS 符号とは符号理論における誤り訂正符号の一つである. その高い誤り訂正能力から, QR コードなどに応用されている.

以下に, RS 符号の各用語について説明する.

1. 符号多項式

符号長 n の線形符号 C の任意の符号語をベクトル表現

$$u = (u_0, u_1, u_2, \dots, u_{n-1}) \quad (2.1)$$

としたとき式 2.1 の多項式表現は

$$u(x) = u_0 + u_1x + u_2x^2 + \dots + u_{n-1}x^{n-1} \quad (2.2)$$

である. ここで変数 x^i は単に記号 u_i の位置を示すだけである. 式 2.2 のようにある符号語に対応する多項式を特に符号多項式と呼ぶ^[5].

2. 生成多項式

ある情報記号と対応する多項式 (以下, 情報多項式) $q(x)$ からこれの誤り訂正を行う符号語 $u(x)$ を生成することを考えた際

$$u(x) = q(x)g(x) \quad (2.3)$$

と表される $g(x)$ を生成多項式と呼ぶ^[5].

3. 体

体とは代数学においてある性質を満たした集合である. 体の性質の中でも最も特徴的な点は, 元 (体の要素をこのように呼ぶ) の四則演算は結果も元になる (体の中で閉じている) という点である. 例えば実数は体であり, 実数を用いた四則演算は計算結果が全て実数になる. しかし自然数は体ではなく, 例えば $1 - 2$ の演算結果は負の値となりこれは自然数ではないのでこれは体とは言えない.

4. ガロア体 (有限体)

体の中でも元が有限なものをガロア体 (Galois field) と呼び、有限体とも呼ばれる。元の数が q のガロア体を $GF(q)$ で表し、元の数 は素数、あるいは素数のべき乗である必要がある^[5]。例えば $GF(2)$ の元は一般的に 0 と 1 である。計算例として以下に $GF(2)$ 上の加算減算の計算結果を示す。

表 2.2 $GF(2)$ 上の加算結果

入力 1	入力 2	出力
0	0	0
0	1	1
1	0	1
1	1	0

表 2.3 $GF(2)$ 上の減算結果

入力 1	入力 2	出力
0	0	0
0	1	1
1	0	1
1	1	0

5. 拡大体, 原始元, 原始多項式

ガロア体 $GF(p)$ 上の既約多項式 (これ以上因数分解できない多項式) $g(x)$ を選び、その根 ($g(x) = 0$ となるような x の値) を α とする。この α を $GF(p)$ の元に追加することで新たな体が生成でき、そうしてできた新たな体を拡大体と呼ぶ。この時の α を原始元といい、この既約多項式は原始多項式という^[5]。拡大体の例として複素数が挙げられる。複素数は実数の拡大体であり実数上の既約多項式 $x^2 + 1$ の根を i として実数の元に追加したものである。

拡大体 $GF(2^m)$ 上の RS 符号について考える。拡大体 $GF(2^m)$ の原始元を α とするとき、 $\alpha, \alpha^2, \alpha^3, \dots, \alpha^{2^t}$ を根として持つ $GF(2^m)$ 上の生成多項式 $g(x)$ により生成される符号を t 重誤り訂正 RS 符号と呼ぶ。

QR コード上の生成多項式^{[6][7]} は $n - k$ 次多項式であり、これを $g(x)$ とする。

$$g(x) = (x - 1)(x - \alpha) \cdots (x - \alpha^{n-k-1}) = \prod_{i=0}^{n-k-1} (x - \alpha^i) \quad (2.4)$$

式 2.4 の $g(x)$ を展開した多項式を

$$g(x) = g_1 x^{n-k} + g_2 x^{n-k-1} + \cdots + g_{n-k+1}, \quad g_1 = 1 \quad (2.5)$$

とする。その係数列 $g_1 = 1, g_2, \dots, g_{n-k+1}$ から定まる $k \times n$ 行列

$$G_0 = \begin{bmatrix} 1 & g_2 & \cdots & g_{n-k-1} & 0 & \cdots & \cdots & 0 \\ 0 & 1 & g_2 & \cdots & g_{n-k-1} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & g_2 & \cdots & g_{n-k-1} & 0 \\ 0 & \cdots & \cdots & 0 & 1 & g_2 & \cdots & g_{n-k-1} \end{bmatrix}$$

を生成行列と呼ぶ。

一般的に QR コードでは情報記号と検査記号を分けた組織符号が用いられる。行列 G_0 を使って組織符号を計算するためには、行列 G_0 を”掃き出し法”によって $G = [I_k, P]$ (I_k は $k \times k$ 単位行列) という標準形に変形する。

データ $v = (v_1, v_2, \dots, v_k)$ に対する符号語 u は標準形の生成行列 $G = [I_k, P]$ により、

$$u = vG \quad (2.6)$$

として表現することができ、この u が QR コード上の RS 符号である。

2.2.1 QR コード上の RS 符号

以下に QR コード上での RS 符号について示す。

RS 符号の各シンボルは拡大体 $GF(2^8)$ を使用し、原始多項式は $x^8 + x^4 + x^3 + x^2 + 1$ を使用する。この原始多項式の原始元 α を用いて計算を行う。

RS 符号の情報長は固定されているため、QR コードに入れる情報が少ない場合は埋め草コード語が追加される。例えばバージョン 1 の QR コードを構成する RS 符号の長さは $n = 26$ シンボルとなっているが、そのうちの情報記号の長さは 16 シンボルである。QR コードに入れる文字列が $k = 16$ シンボル未満で表現される場合、情報記号のシンボル数を 16 シンボルに合わせるため、埋め草コード語という情報を持たないシンボルを付加する必要がある。主に QR コードに書き込む文字列を RS 符号化したものを含むシンボル (以下、データと述べる) の個数を \hat{k} と表すとき、データを表す RS 符号のシンボルは

$$\alpha_1, \dots, \alpha_{\hat{k}} \quad (2.7)$$

である。 $\hat{k} < k$ の時、RS 符号のデータを表すシンボルを k 個に合わせるため、埋め草コード語を付加する。埋め草コード語は

$$\alpha_{\hat{k}+1}, \dots, \alpha_k \quad (2.8)$$

と表す.

これにより, RS 符号の情報記号のシンボルは

$$\alpha_1, \dots, \alpha_{\hat{k}}, \alpha_{\hat{k}+1}, \dots, \alpha_k$$

と表せる.

QR コード上の RS 符号の全体図を図 2.4 に示す.

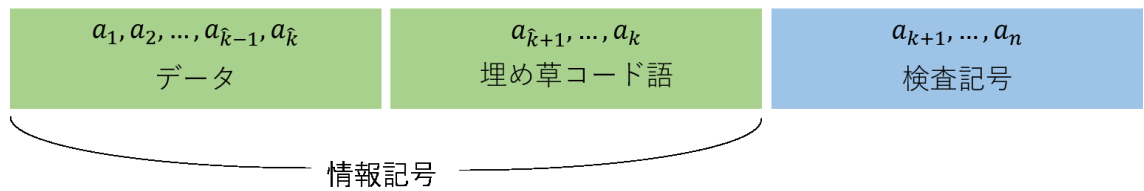


図 2.4 QR コード上の RS 符号の全体図

第3章 Aesthetic QR コード

本章では，Kuribayashi らの論文で提案された^[2] のランダム法 (Random-Method), 色変換手法 (Color Translation) について述べる．ランダム法は QR コード上に配置された二値行列が埋め込む目的画像のモジュールパターンと類似した Aesthetic QR コードを生成するアルゴリズムである．色変換手法はカラー画像に対する Aesthetic QR コードを生成するためのアルゴリズムである．Aesthetic QR コードの例を図 3.1 に示す．このように Aesthetic QR コードはデザイン性が考慮されている．



図 3.1 Aesthetic QR コード例^[2]

3.1 目的画像の二値化手法

目的画像はカラー画像でも二値画像でもよい．カラー画像や二値画像を用いる際に，目的画像と QR コードとの距離を測るために，目的画像を二値化する必要がある．ここでの距離にはハミング距離を用い，最もハミング距離の小さい QR コードを用いて Aesthetic QR コードを作る．目的画像を二値化画像にする際に閾値を指定するが，その閾値を Algorithm 1 に示す閾値計算手法で決定する．

閾値を用いて目的画像から目的画像の二値行列を作成するアルゴリズムを Algorithm 2 に示す．

Algorithm 1 論文^[2]の閾値計算手法

入力: サイズ $L \times L$ の目的画像

出力: 閾値 \bar{Y}

- 方法: 1. 入力画像の大きさは, QR コードのバージョン v と同じサイズに予め変更しておく. RGB 色成分は YUV 色成分に変換され, 輝度 (Y) 成分 $Y_{i,j}$ ($1 \leq i, j \leq L$) が得られる.
2. その中心の正方形 (元の画像サイズの $\frac{1}{4}$) の値の平均 \bar{Y} を計算する.

$$\bar{Y} = \frac{4}{L^2} \sum_{i=\frac{L}{4}}^{\frac{3L}{4}} \sum_{j=\frac{L}{4}}^{\frac{3L}{4}} Y_{i,j} \quad (3.1)$$

Algorithm 2 論文^[2]の目的画像に対する二値行列の生成

入力: サイズ $L \times L$ の目的画像の輝度 (Y) 成分 $Y_{i,j}$ ($1 \leq i, j \leq L$), 閾値 \bar{Y}

出力: 二値行列 $B_{i,j}$

- 方法: 1. 目的画像を二値化する際, 二値行列 $B_{i,j}$ は, 以下の規則によって決定される.

$$B_{i,j} = \begin{cases} 1 & Y_{i,j} > \bar{Y} \\ 0 & otherwise \end{cases} \quad (3.2)$$

3.2 ランダム法

ランダム法とは QR コードのシンボルの順番を規則に従って変えることで、目的画像を二値化した画像との距離が小さい QR コードを生成する方法である。まず、Aesthetic QR コードの元となる QR コードを生成するための生成行列 G を求める。Random Method ではこの G を用いて、掃き出し法により QR コードを生成する。ただし、単位行列の列ベクトルの位置は情報記号の最初の \hat{k} 列を除いてランダムに決定される。次に、QR コードと目的画像の画像サイズを等しいものとして、QR コードの 1 モジュールと目的画像の 1 画素を対応させ、目的画像と QR コードがどれだけ近いものか比較する。ただし、QR コードに書き込むデータのシンボルを除いて比較する。具体的には、目的画像の二値化を行ったものに QR コードのデータ (RS 符号中の情報記号の中のデータにあたる箇所) を書き加えたものを用意し、それと生成した QR コードとのハミング距離を取る。以上を N 回繰り返してハミング距離が最小となる QR コードを求め、それを使って Aesthetic QR コードを作成する。

本研究では QR コードの中でバージョン 1 の QR コードを用いた。バージョン 1 の QR コードを作成する手順を Algorithm3 に示す。

Algorithm 3 論文^[2]のランダム法を用いたバージョン 1 の Aesthetic QR コード

入力 (試行回数 N): バージョン 1 の QR コードに入るシンボル長 16 の文字列, サイズ 21×21 の目的画像

出力: サイズ 21×21 のバージョン 1 の Aesthetic QR コード

- 方法:
1. 目的画像の画素値を QR コードのモジュールに割り当て、Algorithm2 で決まった閾値 \bar{Y} でモジュールを二値化し、二値行列 $B_{i,j}$ を作成する。
 2. $B_{i,j}$ に所定のマスクパターンを作用させる。
 3. 式 (2.8) の α_t の位置 t ($\hat{k} + 1 \leq t \leq n$) を変化させることにより、RS 符号を計算する。以後この手順を N 回繰り返すことにより、 $B_{i,j}$ とのハミング距離が最小となる RS 符号を見つける。一定の試行回数終了後、QR コード上の RS 符号は、ハミング距離が最小の RS 符号に置き換える。
 4. マスク処理前である QR コードの各モジュールに対して、所定のマスクパターンを適用し、Aesthetic QR コードとして出力する。
-

3.3 色変換手法

色変換手法とは, Algorithm3 までで生成した Aesthetic QR コードに色を追加するアルゴリズムである. 例えば, Algorithm3 までで生成した Aesthetic QR コード内の黒の画素があったとし, この画素をそのまま同じ位置にある目的画像の画素の色に変換してしまうと, 目的画像の輝度値によっては本来 1 と読み取るはずの画素が 0 と判別されてしまうなどして, Aesthetic QR コードとして読み込まれないという問題が生じる可能性がある. そのため, そのまま目的画像の色に変換するのではなく, 画素ごとに輝度値を変更する必要がある. 例えば, Aesthetic QR コードとして 1 と読み取る画素は輝度値を変更してある程度暗く, 反対に 0 と読み取る画素はある程度明るくする必要がある. このアルゴリズムはその輝度値を変更するためのアルゴリズムである.

具体的にはまず, 目的画像の平均輝度値を求めこれを \bar{Y} とする. 次に閾値 ϵ (本実験では実験的に 0.25 とする) を定める. ϵ とはある画素を平均輝度値からどれだけ離れた値に変更するかという値である.

色変換手法を適用するアルゴリズムを以下に示す.

Algorithm 4 論文 [2] の色変換手法

入力: algorithm3 で生成した Aesthetic QR コード

入力: 目的画像

出力: カラー画像に対する Aesthetic QR コード

方法: 1. 目的画像を入力 of Aesthetic QR コードの大きさに変更する,
 2. 大きさを変更した目的画像の平均輝度値を求める,
 3. 大きさを変更した目的画像の画素 $\beta_{i,j}$ の輝度値 $Y_{i,j}$ を, 以下の方法にしたがって
 新しい輝度値 $Y'_{i,j}$ に変更する.

if $\beta_{i,j} = 1$, then

$$Y'_{i,j} = \begin{cases} Y_{i,j} & Y_{i,j} > \bar{Y} + \epsilon \\ \bar{Y} + \epsilon & otherwise \end{cases} \quad (3.3)$$

otherwise

$$Y'_{i,j} = \begin{cases} Y_{i,j} & Y_{i,j} < \bar{Y} - \epsilon \\ \bar{Y} - \epsilon & otherwise \end{cases} \quad (3.4)$$

第4章 数式処理システム Maple を用いた Aesthetic QR コードの実装

4.1 数式処理システム Maple^[8]

Maple は数式を正確に誤差なく計算するためのシステムである．数式の計算には記号計算や数値計算，グラフ描画などが含まれる．Maple は 1985 年にカナダのウォータールー大学で開発が始められた．現在，世界中で使用されており，科学技術計算や工学問題や教育などに応用されている．主に計算可能な数式としては以下のものがあげられる．

- 多倍長整数演算
- 多項式演算
- 行列ベクトル演算
- 代数体上での計算 (ガロア体の計算)
- 数値計算

また，Maple で定義されたプログラミング言語があり，その言語を使って新しい数学関数を定義したり，ユーザーインターフェースを作成したりすることができる．

4.2 実験環境

実験に使用した PC 環境，言語を以下に示す．

- ソフトウェア実装環境
 - CPU : Intel(R)Core i5 7500 3.4GHz
 - OS : Windows 10 pro
 - 実装 RAM : 16.0GB

- 開発環境

- Maple : Maple 2021.1

実験に使用した各パラメータを以下に示す.

- QR コード

- RS 符号 : (26,16) 符号
- 入力文字 : tahara
- バージョン (v) : 1
- マスクパターン : 001
- 誤り訂正レベル : M

QR コードのバージョン 1 の構造を図 4.1 に示す.

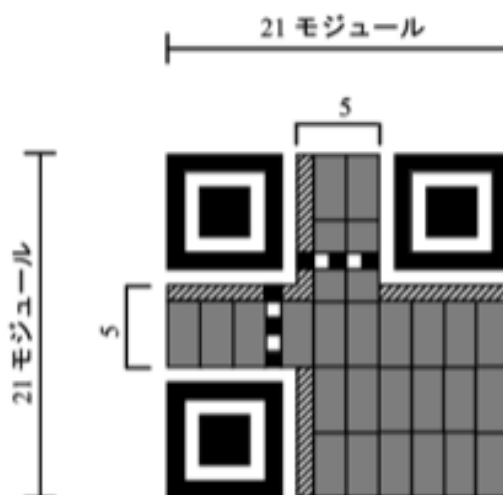


図 4.1 QR コードのバージョン 1 の構造

4.3 Maple による QR コードの生成法の実装

本研究では, RS 符号を生成する際に必要な有限体の実装を行うために, Aesthetic QR コードの生成に数式処理 Maple(以下, Maple) を用いた. 以下に Maple において, QR コード中の RS 符号を生成する方法を示す.

拡大体 $GF(2^8)$, 原始多項式 $x^8 + x^4 + x^3 + x^2 + 1$ とその原始元 α は以下のように表される. Maple において原始多項式は GF 関数の第 3 引数で定義される. GF はガロア体を定義するための関数である.

```
G8 := GF(2, 8, alpha^8+alpha^4+alpha^3+alpha^2+1):
a := G8:-ConvertIn(alpha):
```

図 4.2 Maple 上でのガロア体の実装

Algorithm1～3 の実装を Maple 上で行い，ソースコードを付録 A に示す．Aesthetic QR コードを生成するための関数は `gen_AestheticQRcode` である．`gen_AestheticQRcode` は，Algorithm2 で生成した QR コードと，目的画像を入力として，Aesthetic QR コードを生成する．結果を図 4.4 に示す．

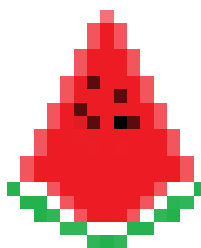
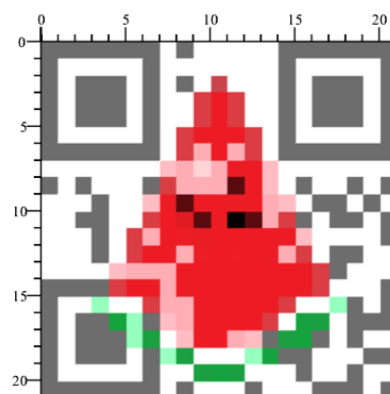


図 4.3 目的画像

```
> #関数名 : gen_AestheticQRcode
#入力1 : QRコードの情報が入ったベクトル
#入力2 : 目的画像を示すパス
#出力 : AestheticQRcodeの情報が格納された行列

AestheticQRcode := gen_AestheticQRcode(AestheticQRcode_Vec, image_path):
Preview(AestheticQRcode);
```

図 4.4 関数 `AestheticQRcode` の入力と出力

4.4 Aesthetic QR コードの生成に関する実験

Aesthetic QR コードは以下の2つの値で評価を行う。

- 目的画像の二値化行列 $B_{i,j}$ と Algorithm2 で得られた QR コードの二値表列のハミング距離
- 一つの Aesthetic QR コードの生成に要した時間

ハミング距離は Kuribayashi らの論文^[2]で使われている。

2つの符号語つの符号語 $a = (a_1, a_2, \dots, a_n)$ と $b = (b_1, b_2, \dots, b_n)$ で対応するビット (桁) で値 (0 または 1) が異なっているビット (桁) の数をハミング距離と言い、記号で $d(a, b)$ と書く。その中でも一番ハミング距離が小さいものを最小ハミング距離と呼ぶ。

ハミング距離は2つの符号語 $a = (a_1, a_2, \dots, a_n)$ と $b = (b_1, b_2, \dots, b_n)$ に対して以下の式で定義される。

$$d(a, b) = \sum_{i=1}^n (a_i + b_i) \pmod{2} \quad (4.1)$$

本実験では QR コードに挿入する画像は以下の目的画像 (図 4.5) を使用した。図 4.6~4.10 はランダム法の試行回数が $N = 1, 10, 100, 1000, 10000$ の場合の結果 (Aesthetic QR コード) を表す。実際にソフトウェアによって生成した画像が図 4.6~ 図 4.10 である。

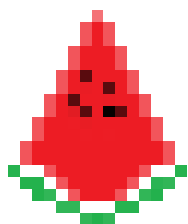


図 4.5 目的画像

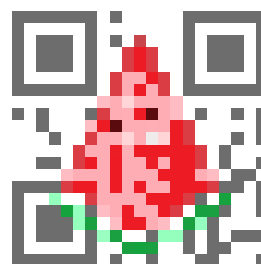


図 4.6 $N = 1$

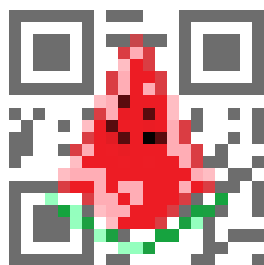


図 4.7 $N = 10$

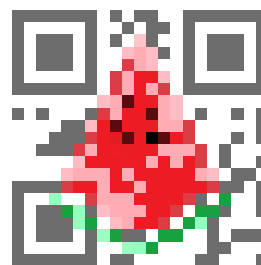
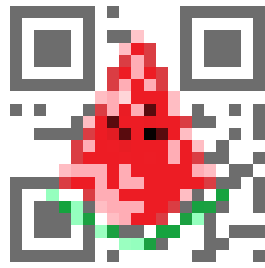
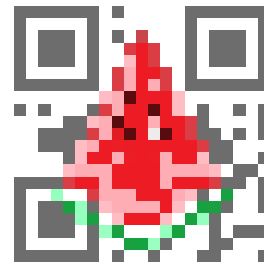


図 4.8 $N = 100$

図 4.9 $N = 1000$ 図 4.10 $N = 10000$

また、表 4.1 に各試行回数における実験で得られたハミング距離、RS 符号の生成にかかった時間を表す。

表 4.1 結果のまとめ

試行回数 (N)	1	10	100	1000	10000
最小ハミング距離	83	63	51	47	41
時間 (秒)	0.08	0.81	8.01	80.39	771.17

表 4.1 を見ると、試行回数を増やしていくたびに最小ハミング距離が小さくなっており、目的画像に近い Aesthetic QR コードが生成されていることが確認できた。しかし、試行回数 10000 回の所を見ると RS 符号が生成するのにかかる時間が約 13 分もかかっており、Maple 上の実装について計算時間の短縮が検討課題である。

第 5 章 結論

本研究では，数式処理 Maple 上での Aesthetic QR コードを自動生成するソフトウェアを開発した．目的画像に近い Aesthetic QR コードを自動生成する方法について，Kuribayashi らの手法を数式処理システム Maple 上に実装した．その結果，任意の目的画像に対する Aesthetic QR コードを生成することができた．しかし，今後の課題として

- 生成速度を短縮するアルゴリズムの検討
- 生成可能な Aesthetic QR コードのバージョンの追加

などが挙げられる．

謝辞

本研究に際して，日々，様々なご指導をいただきました高橋寛教授，甲斐博准教授，王森レイ講師に心より感謝致します．そして，本研究に際してご審査いただきました遠藤慶一准教授，宇戸寿幸准教授に感謝の意を表します．最後に日頃から助言や励ましをいただきました諸先輩方，並びに同研究室の皆様に深く御礼を申し上げます．

参考文献

- [1] QR code.com. <http://www.qrcode.com/en>.
- [2] M. Kuribayashi and M. Morii "Aesthetic QR Code Based on Modified Systematic Encoding Function", IEICE transactions on information and systems ,VOL.E100-D, NO.1,pp.42-51,2017.
- [3] DENSO WAVE フレーム QR
<https://www.denso-wave.com/ja/system/qr/product/frame.html>
- [4] Visualead. <http://www.visualead.com>.
- [5] 汐崎陽, 情報・符号理論の基礎, 2011 年
- [6] 池田和興, 例題が語る符号理論, 共立出版, 2007 年
- [7] J. Justesen and T. Hoholdt "A Course In Error-Correction Codes", European Mathematical Society Publishing House, 2004.
- [8] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt, First Leaves: A Tutorial Introduction to Maple V, Springer-Verlag, 1992.
- [9] JIS X0510. 情報技術－自動認識及びデータ取得技術－QRコード バーコードシンボル体系仕様
<http://www.jisc.go.jp/app/pager?id=2738494>.

付 録A プログラムリスト

以下のファイルは本研究のために作成したプログラムファイルである．以下にソースコードを示す．

[A.QRcode_Naoya]

```

1 restart;
2 isFinderPattern := proc(version, x, y)
3   local size;
4   size := QRcodeSize(version);
5   return x <= 7 and y <= 7 or x <= 7 and size - 8 <= y or size - 8 <= x and y <= 7;
6 end proc;
7
8 isTimingPattern := proc(version, x, y)
9   return not isFinderPattern(version, x, y) and (x = 6 or y = 6);
10 end proc;
11
12 isFunctionPattern := proc(version, x, y)
13   return isFinderPattern(version, x, y) or isTimingPattern(version, x, y);
14 end proc;
15
16 dataWritable := proc(version, x, y)
17   local size;
18   size := QRcodeSize(version);
19   if isFunctionPattern(version, x, y) then
20     return false;
21   end if;
22   if x <= 8 and y <= 8 then
23     return false;
24   end if;
25   if x = 8 and size - 8 <= y then
26     return false;
27   end if;
28   if y = 8 and size - 8 <= x then
29     return false;
30   end if;
31   if 7 <= version then
32     if x < 6 and size - 11 <= y and y < size - 8 then
33       return false;
34     end if;
35     if y < 6 and size - 11 <= x and x < size - 8 then
36       return false;
37     end if;

```

```
38   end if;
39   if x = 8 and y = 4*version + 9 then
40     return false;
41   end if;
42   return true;
43 end proc;
44
45 PosProceed := proc(version, xref::uneval, yref::uneval)
46   local size, x, y, relx, rely; x := eval(xref); y := eval(yref);
47   size := QRcodeSize(version);
48   ASSERT(x <> 6);
49   relx := x; rely := y;
50   if 6 < x then
51     relx := relx - 1;
52   end if;
53   if relx mod 2 = 0 then
54     if iquo(relx, 2) mod 2 = 0 then
55       if rely = size - 1 then
56         if relx = 0 then
57           x := size - 1;
58         else x := x - 1;
59         end if;
60       else x := x + 1; y := y + 1;
61     end if;
62     else if rely = 0 then
63       if x = 7 then
64         x := x - 2;
65       else x := x - 1;
66       end if;
67     else x := x + 1;
68     y := y - 1;
69     end if;
70   end if;
71   else x := x - 1;
72 end if;
73 xref := x; yref := y;
74 end proc;
75
76 PosNext := proc(version, x_::uneval, y_::uneval)
77   local size, x, y; x := eval(x_); y := eval(y_);
78   size := QRcodeSize(version);
79   do
80     break;
81     if x = 0 and y = size - 1;
82     PosProceed(version, x, y);
83     break;
84     if dataWritable(version, x, y);
85   end do;
86   x_ := x; y_ := y;
87 end proc;
88
```

```

89 setVec := proc(x, y, exp, in_Vec, version)
90   local Vec, M; M := QRcodeSize(version);
91   Vec := in_Vec;
92   if evalb(exp) then
93     Vec[M*y + x + 1] := "Black";
94   else
95     Vec[M*y + x + 1] := "White";
96   end if;
97 end proc;
98
99 fill := proc(centerX, centerY, halfWidth, exp, Vec, version)
100   local x, y, size;
101   size := QRcodeSize(version);
102   for y from max(centerY - halfWidth, 0) to min(centerY + halfWidth, size - 1) do
103     for x from max(centerX - halfWidth, 0) to min(centerX + halfWidth, size - 1) do
104       setVec(x, y, exp, Vec, version);
105     end do;
106   end do;
107 end proc;
108
109 PlaceFinderPattern := proc(Vec, version)
110   local size;
111   #global QRcodeVersion;
112   size := QRcodeSize(version);
113
114   #左上
115   fill(3, 3, 4, false, Vec, version):
116   fill(3, 3, 3, true, Vec, version):
117   fill(3, 3, 2, false, Vec, version):
118   fill(3, 3, 1, true, Vec, version):
119
120   #右上
121   fill(size - 1 - 3, 3, 4, false, Vec, version):
122   fill(size - 1 - 3, 3, 3, true, Vec, version):
123   fill(size - 1 - 3, 3, 2, false, Vec, version):
124   fill(size - 1 - 3, 3, 1, true, Vec, version):
125
126   #左下
127   fill(3, size - 1 - 3, 4, false, Vec, version):
128   fill(3, size - 1 - 3, 3, true, Vec, version):
129   fill(3, size - 1 - 3, 2, false, Vec, version):
130   fill(3, size - 1 - 3, 1, true, Vec, version):
131
132 end proc:
133
134 PlaceTimingPattern := proc(Vec, version)
135   local i, size;
136   size := QRcodeSize(version);
137   for i from 8 to size - 9 do
138     setVec(i, 6, i mod 2 = 0, Vec, version);
139   end do;

```



```

140   for i from 8 to size - 9 do
141     setVec(6, i, i mod 2 = 0, Vec, version);
142   end do;
143 end proc;
144
145 PlaceAlwaysBlack := proc(in_Vec, version)
146   local size, Vec;
147   Vec := in_Vec; size := QRcodeSize(version);
148   Vec[size*(size - 8) + 9] := "Black";
149 end proc;
150
151 PlaceFunctionPattern := proc(Vec, version)
152   PlaceTimingPattern(Vec, version);
153   PlaceFinderPattern(Vec, version);
154   PlaceAlwaysBlack(Vec, version);
155 end proc;
156
157 PlaceFormatInfo := proc(Vec, version)
158   local Gxx, formatInfox, errCorCodex, result, maskPattern, deg, pos_x, pos_y, size, M, x
159     ; x := 'x'; Gxx := x^10 + x^8 + x^5 + x^4 + x^2 + x + 1;
160   formatInfox := 1;
161   errCorCodex := formatInfox*x^10;
162   errCorCodex := rem(errCorCodex, Gxx, x) mod 2;
163   result := formatInfox*x^10 + errCorCodex;
164   maskPattern := x^14 + x^12 + x^10 + x^4 + x;
165   result := (result + maskPattern) mod 2;
166   deg := 14;
167   for pos_x from 0 to 5 do setVec(pos_x, 8, coeff(result, x, deg) = 1, Vec, version);
168   deg := deg - 1;
169 end do;
170   setVec(7, 8, coeff(result, x, deg) = 1, Vec, version);
171   deg := deg - 1;
172   setVec(8, 8, coeff(result, x, deg) = 1, Vec, version);
173   deg := deg - 1;
174   setVec(8, 7, coeff(result, x, deg) = 1, Vec, version);
175   deg := deg - 1;
176   for pos_y from 5 by -1 to 0 do
177     setVec(8, pos_y, coeff(result, x, deg) = 1, Vec, version);
178     deg := deg - 1;
179   end do; deg := 14;
180   size := QRcodeSize(version);
181   for pos_y from size - 1 by -1 to size - 7 do
182     setVec(8, pos_y, coeff(result, x, deg) = 1, Vec, version);
183     deg := deg - 1;
184   end do;
185   for pos_x from size - 8 to size - 1 do
186     setVec(pos_x, 8, coeff(result, x, deg) = 1, Vec, version);
187     deg := deg - 1; end do;
188 end proc;
189
190 flip := proc(x, y, in_Vec, version) local size, Vec;

```

```

190  Vec := in_Vec;
191  size := QRcodeSize(version);
192  if Vec[x + 1 + size*y] = "White" then
193    Vec[x + 1 + size*y] := "Black";
194  elif Vec[x + 1 + size*y] = "Black" then
195    Vec[x + 1 + size*y] := "White";
196  end if;
197 end proc;
198
199 myMask := proc(maskPattern, in_Vec, version) local condition, size, x, y, Vec;
200   size := QRcodeSize(version);
201   for y from 0 to size - 1 do
202     for x from 0 to size - 1 do
203       condition := false;
204       if maskPattern = "000" then
205         condition := evalb((x + y) mod 2 = 0);
206       elif maskPattern = "001" then
207         condition := evalb(y mod 2 = 0);
208       elif maskPattern = "010" then
209         condition := evalb(x mod 3 = 0);
210       elif maskPattern = "011" then
211         condition := evalb((x + y) mod 3 = 0);
212       elif maskPattern = "100" then
213         condition := evalb((iquo(x, 3) + iquo(y, 2)) mod 2 = 0);
214       elif maskPattern = "101" then
215         condition := evalb(((x*y mod 3) + x*y) mod 2 = 0);
216       elif maskPattern = "110" then
217         condition := evalb((((x*y mod 3) + x*y) mod 2) mod 2 = 0);
218       elif maskPattern = "111" then
219         condition := evalb((((x*y mod 3) + x + y) mod 2) mod 2 = 0);
220       else print("マスクパターン値が異常です");
221       condition := false;
222     end if;
223     Vec := in_Vec;
224     if condition then
225       flip(x, y, Vec, version);
226     end if;
227   end do;
228 end do; end proc;
229
230 SetModule := proc(x, y, isBlack, in_Vec)
231   local Vec;
232   Vec := in_Vec;
233   if isBlack then Vec[x + M*y] := "Black";
234   else Vec[x + M*y] := "White";
235   end if;
236 end proc;
237
238 PlaceCode := proc(codePairs, Vec, version)
239   local x, y, index, i, size;
240   global QRcodeVersion;

```

```

241   size := QRcodeSize(version);
242   x := size - 1; y := size - 1;
243   for index to numelems(codePairs) do
244     for i to 8 do setVec(x, y, codePairs[index][i] = 1, Vec, version);
245       PosNext(QRcodeVersion, x, y);
246     end do;
247   end do;
248 end proc;
249
250 #関数 : gen_QRcode
251 #出力 : QRcodeの内容が入ったVec
252
253 gen_QRcode := proc(str,L,image_path,n,k,khat,QRcodeVersion,For_QRcode_binary_image)
254 local i,j;
255 local delta,myrand,myset,mylist;
256
257 #deltaを求める
258 delta := Vector(k):
259 for i from 1 to k-khat do
260   delta[i] := i;
261 end do:
262 randomize():
263 myrand := rand(k-khat+1..n):
264 myset := {}:
265 while (nops(myset)<>khat) do
266   myset := 'union'(myset, {myrand()});
267 end do:
268 mylist := convert(myset, list);
269
270 #mylist:=[13,14,15,16];####Debug
271 #mylist:=[9, 10, 14, 15, 19, 22, 24, 26];####Debug
272
273 for i from 1 to khat do
274   delta[i+k-khat] := mylist[i];
275 end do:
276
277 #Gの生成
278 local inv,l,tmp,G;
279 G := Matrix(k,n):
280 for i from 1 to k do
281   for j from 1 to n do
282     G[i, j] := G8:-input(0);
283   end do;
284 end do;
285 for i from 1 to k do
286   for j from 1 to 11 do
287     G[i,(i-1)+j] := G8:-'^'(a,GP[j]);
288   end do;
289 end do;
290 for i from 1 to k do
291   inv := G8:-inverse(G[i,delta[i]]):

```

```

292   for j from 1 to n do
293       G[i,j] := G8:-'*(inv, G[i,j]);
294   end do;
295   for l from 1 to i-1 do
296       tmp := G[l,delta[i]];
297       for j from 1 to n do
298           G[l,j] := G8:-'-(G[l,j], G8:-'*(tmp, G[i,j]));
299       end do;
300   end do;
301   for l from i+1 to k do
302       tmp := G[l,delta[i]];
303       for j from 1 to n do
304           G[l,j] := G8:-'-(G[l,j], G8:-'*(tmp, G[i,j]));
305       end do;
306   end do;
307 end do:
308
309
310 #FP_binの生成
311 local p,cnt,padding_location,code_num,size,x,y,Str_bin,FP_bin;
312 FP_bin := [0,1,0,0]:
313 FP_bin := [op(FP_bin),op(binarize(L))]:
314 Str_bin := Use_ToByteArray(str):
315 for i from 1 by 1 to L do
316     tmp := op(binarize(Str_bin[i])):
317     FP_bin := [op(FP_bin),tmp]:
318 end do:
319 FP_bin := [op(FP_bin),op([0,0,0,0])]:
320 #code_num: 何コード目かを示す
321 #deltaのうち埋め草コード部のみをpadding_locationに格納
322 padding_location := [];
323 for p from k-khat+1 by 1 to n do
324     if member(p,delta) then
325         padding_location := [op(padding_location),p];
326     end if;
327 end do;
328
329 size := QRcodeSize(QRcodeVersion);
330 x := size-1;
331 y := size-1;
332
333 cnt:=1;
334 code_num := floor((cnt-1)/8) + 1;
335 while code_num < n+1 do
336     if code_num > k-khat and member(code_num, padding_location) then
337         FP_bin := [op(FP_bin),floor(For_QRcode_binary_image[y+1,x+1]+1)mod 2];
338     end if;
339 cnt++;
340 PosNext(QRcodeVersion,x,y);
341 code_num := floor((cnt-1)/8) + 1;
342 end do;

```

```

343 FP_bin := Use_LengthSplit(FP_bin,8):
344
345 #FP_listを作成する
346 local FP_list,For_FP_list,dec;
347 FP_list := [];
348 for i from 1 by 1 to k do
349     For_FP_list := Use_Reverse(FP_bin[i]):
350     dec := 0:
351     for j from 1 by 1 to 8 do
352         dec := dec + For_FP_list[j] * (2 ^ (j-1));
353     end do:
354     FP_list := [op(FP_list),get_exp(dec)]:
355 end do:
356
357 #FP, Fの生成
358 local FP, F;
359 FP := Vector(k, FP_list):
360 F := Vector(k):
361 for i from 1 to k do
362     if (FP[i]>=0) then
363         F[i] := G8:-'^'(a, FP[i]);
364     else
365         F[i] := G8:-input(0);
366     end if;
367 end do;
368
369 #Cの生成
370 local C;
371 C := Vector(n);
372 for i from 1 to n do
373     C[i] := G8:-input(0);
374     for j from 1 to k do
375         C[i] := G8:-'+'(C[i], G8:-'*'(F[j], G[j, i]));
376     end do;
377 end do:
378
379 #Vecの生成
380 local maskPattern,Vec,M:
381 local c;
382 c := [];
383 for i to n do
384     c := [op(c), mybin(G8:-output(C[i]))];
385 end do;
386
387 M := QRcodeSize(QRcodeVersion);
388 maskPattern := "001":
389 Vec := Vector(M*M):
390 for i to M*M do
391     Vec[i] := "Pink";
392 end do:
393

```

```

394 PlaceCode(c,Vec,QRcodeVersion):
395 myMask(maskPattern,Vec,QRcodeVersion):
396 PlaceFormatInfo(Vec,QRcodeVersion):
397 PlaceFunctionPattern(Vec,QRcodeVersion):
398
399 return Vec:
400
401 end proc:
402 #関数名: Calculate_Hamming_distance
403 #入力1: 目的画像を二値化し、文字列を加えて書き換えたもの
404 #入力2: 生成したAestheticQRcodeの元となる情報(Vec)
405 #出力: 二つのハミング距離
406
407 Calculate_Hamming_distance := proc(For_Hamming_binary_image,QRcode_Vec,k,khat);
408 local i,j,Hamming_distance,cnt,code_num,QRcode_Size;
409
410 Hamming_distance := 0;
411 cnt:=1:
412 code_num := floor((cnt-1)/8) + 1:
413
414 QRcode_Size := sqrt(numelems(QRcode_Vec));
415 i := QRcode_Size-1:
416 j := QRcode_Size-1:
417 #while code_num < k-khat+1 do
418 while code_num < n+1 do
419     if code_num > k-khat then
420         if QRcode_Vec[j*21+(i+1)] = "Black" then
421             if For_Hamming_binary_image[i+1,j+1] = 1 then
422                 Hamming_distance++;
423             end if;
424         elif QRcode_Vec[j*21+(i+1)] = "White" then
425             if For_Hamming_binary_image[i+1,j+1] = 0 then
426                 Hamming_distance++;
427             end if;
428         end if;
429     end if;
430 #print(j*21+(i+1),i+1,j+1);
431     cnt++;
432     PosNext(1,i,j):
433     code_num := floor((cnt-1)/8) + 1:
434 end do:
435
436 return Hamming_distance:
437 end proc:
438 read "//wfs01/Users/e1848taha/Desktop/thesis/Maple/A_QRcode_Naoya_Function.mpl";
439 str := "Tahara":
440 #image_path := "./pic/Lenna.jpg":
441 #image_path := "./pic/mican.png":
442 #image_path := "./pic/flower.png":
443 image_path := "./pic/suika.png":
444 n:=26: k:=16: QRcodeVersion:=1:

```

```

445 with(StringTools):
446 L := Length(str):
447 khat := 16 - (1 + 1 + L):
448
449 G8 := GF(2, 8, alpha^8+alpha^4+alpha^3+alpha^2+1):
450 a := G8:-ConvertIn(alpha):
451 GP := Vector(n-k+1,[0,251,67,46,61,118,70,64,94,32,45]):
452
453 FP_bin := [0,1,0,0]:
454 FP_bin := [op(FP_bin),op(binarize(L))]:
455 Str_bin := ToByteArray(str):
456 for i from 1 by 1 to L do
457     tmp := op(binarize(Str_bin[i])):
458     FP_bin := [op(FP_bin),tmp]:
459 end do:
460 FP_bin := [op(FP_bin),op([0,0,0,0])]:
461 For_Hamming_binary_image := gen_resized_binary_image(image_path):
462 size := QRcodeSize(QRcodeVersion):
463 x := size-1:
464 y := size-1:
465
466 cnt:=1:
467 code_num := floor((cnt-1)/8) + 1:
468
469 while code_num < k-khat+1 do
470     For_Hamming_binary_image[y+1,x+1] := ((FP_bin[cnt] + 1) mod 2):
471     cnt++:
472     PosNext(QRcodeVersion,x,y):
473     code_num := floor((cnt-1)/8) + 1:
474 end do:
475
476 For_QRcode_binary_image := Array(For_Hamming_binary_image):
477
478 #目的画像にマスク処理
479 #マスクは001
480 for i from 1 by 1 to 21 do
481     for j from 1 by 1 to 21 do
482         if i+1 mod 2 = 0 then
483             For_QRcode_binary_image[i,j] := (floor(For_QRcode_binary_image[i,j]) + 1) mod 2 ;
484             end if;
485         end do;
486     end do;
487 with(ColorTools):
488 #Preview(Read(image_path));
489 #Preview(For_Hamming_binary_image);
490 #Preview(For_QRcode_binary_image);
491 N := 1:
492
493 sum_time := 0:
494
495 #for CNT from 1 by 1 to 10 do

```

```
496 min_Hamming_distance := 21*21;
497 st := time();
498
499 for i from 1 by 1 to N do
500   QRcode_Vec := gen_QRcode(str,L,image_path,n,k,khat,QRcodeVersion,
      For_QRcode_binary_image):
501   result_Hamming_distance := Calculate_Hamming_distance(For_Hamming_binary_image,
      QRcode_Vec,k,khat):
502   if min_Hamming_distance > result_Hamming_distance then
503     AestheticQRcode_Vec := QRcode_Vec;
504     min_Hamming_distance := result_Hamming_distance;
505   end if:
506   #if N>100 then
507     # if (i mod (N/10))=0 then
508       #   print((i/(N/100)),"%",min_Hamming_distance);
509     # end if;
510   #end if;
511 #print(result_Hamming_distance);
512 end do:
513 fin := time()-st:
514
515 #sum_time := sum_time + fin;
516 #print(fin);
517 #end do:
518
519 #avarage_time := sum_time/10;
520
521 print("time",fin);
522 print(min_Hamming_distance);
523 Preview(gen_AestheticQRcode(AestheticQRcode_Vec,image_path));
524 Write(write_path,gen_AestheticQRcode(AestheticQRcode_Vec,image_path)):
```


[A_QRcode_Naoya_Function]

```

1  with(ImageTools):
2  QRcodeSize := proc(QRcodeVersion)
3      return QRcodeVersion*4+17:
4  end proc:
5  #関数 : cal_avarage_luminance
6  #入力 1 : 平均輝度値を取得したいグレースケール画像(正方形)
7  #出力 : 入力 1 の平均輝度値
8
9  cal_avarage_luminance := proc(gray_image);
10 local i,j,L,s,g,avarage_luminance;
11
12 L := Height(gray_image);
13 s := round(L/4);
14 g := round(L*3/4);
15
16 avarage_luminance := 0;
17 for i from s by 1 to g do
18     for j from s by 1 to g do
19         avarage_luminance := avarage_luminance + gray_image[j][i];
20     end do;
21 end do;
22 avarage_luminance := avarage_luminance/((g - s)^2);
23
24 return avarage_luminance;
25 end proc:
26 #関数 : gen_resized_image
27 #入力 1 : 元画像(正方形)
28 #入力 2 : サイズ変更後の画像の縦横の長さ
29 #出力 : サイズを変更した画像
30
31 gen_resized_image := proc(image,L);
32 local resized_L,resized_image;
33
34 resized_L := L/Width(image);
35 resized_image := Scale(image,resized_L);
36
37 return resized_image;
38 end proc:
39 #関数 : gen_resized_binary_image
40 #入力 : 画像のパス
41 #出力 : 元画像を21*21に縮小し、グレースケールにしてから二値化した画像
42
43 gen_resized_binary_image := proc(image_path);
44 local i,j,image,resized_gray_image,avarage_luminance,resized_binary_image;
45
46 image := Read(image_path);
47 resized_gray_image := RGBtoGray(gen_resized_image(image,21));
48 avarage_luminance := cal_avarage_luminance(resized_gray_image);
49
50 resized_binary_image := Create(21,21,1);

```

```
51 for i from 1 by 1 to 21 do
52   for j from 1 by 1 to 21 do
53     if (resized_gray_image[i,j] >= avarage_luminance) then
54       resized_binary_image[i,j] := 1;
55     end if;
56   end do;
57 end do;
58
59 return resized_binary_image;
60 end proc:
61 with(StringTools):
62 Use_ToByteArray := proc(str):
63   return ToByteArray(str);
64 end proc:
65 with(ListTools):
66 #関数名 : binarize
67 #入力1 : 10進数
68 #出力 : 入力を2進数にし、8ビット化したリスト
69
70 binarize := proc(dec):
71   local bin, i;
72   bin := Reverse(convert(dec,base,2));
73   for i from 1 by 1 to 8-nops(bin) do
74     bin := [0,op(bin)];
75   end do;
76 end proc:
77 Use_LengthSplit := proc(list,n):
78   return LengthSplit(list,n);
79 end proc:
80 Use_Reverse := proc(list):
81   return Reverse(list);
82 end proc:
83 mybin := proc(a)
84   local i, t, r;
85   t := a;
86   r := [-1, -1, -1, -1, -1, -1, -1, -1];
87   for i from 1 to 8 do
88     r[i] := t mod 2;
89     t := iquo(t, 2);
90   end do;
91   r := Reverse(r);
92   return r;
93 end proc:
94
95
96
97 #関数名 : get_exp
98 #入力1 : 2の8乗のガロア体上の多項式表現
99 #出力 : 多項式表現の元となるアルファの冪
100
101 get_exp := proc(dec):
```

```

102 local exp, fin, pol, G8, a;
103 G8 := GF(2, 8, alpha^8+alpha^4+alpha^3+alpha^2+1);
104 a := G8:-ConvertIn(alpha);
105 exp := 0;
106 pol := G8:-input(dec);
107 fin := G8:-'^(a,0);
108 if dec = 0 then
109     exp := -1;
110 else
111     while pol<>fin do
112         pol := G8:-'/(pol,a);
113         exp := exp+1;
114     end do;
115 end if;
116 return exp;
117 end proc;
118
119 #関数名 : gen_AestheticQRcode
120 #入力 1 : Vec(生成した QR コードが格納されたベクトル)
121 #入力 2 : image_path(元画像のパス)
122 #出力 : AestheticQRcodeが格納された二次元リスト
123
124 gen_AestheticQRcode := proc(QRcode_vec, image_path);
125 local i, j, image, resize_height, resize_width, resize_image, AestheticQRcode;
126 local resize_gray_image, avarage_luminance, epsilon;
127
128 image := Read(image_path);
129 resize_height := 21/Height(image);
130 resize_width := 21/Width(image);
131 resize_image := Scale(image, resize_height, resize_width);
132 resize_gray_image := RGBtoGray(resize_image);
133
134 avarage_luminance := 0;
135 for i from round(21/4) by 1 to round(21*3/4) do
136     for j from round(21/4) by 1 to round(21*3/4) do
137         avarage_luminance := avarage_luminance + resize_gray_image[i][j];
138     end do;
139 end do;
140 avarage_luminance := avarage_luminance/((round(21*3/4) - round(21/4))^2);
141
142 AestheticQRcode := RGBtoYUV(resize_image);
143 epsilon := 0.25;
144
145 for i from 0 by 1 to 21-1 do
146     for j from 1 by 1 to 21 do
147         if QRcode_vec[i*21+j] = "Black" then
148             if AestheticQRcode[i+1,j,1] >= avarage_luminance - epsilon then
149                 AestheticQRcode[i+1,j,1] := avarage_luminance - epsilon;
150             end if;
151         elif QRcode_vec[i*21+j] = "White" then
152             if AestheticQRcode[i+1,j,1] <= avarage_luminance + epsilon then

```

```
153         AestheticQRcode[i+1,j,1] := avarage_luminance + epsilon;
154     end if;
155     else
156         #print("エラー!");
157     end if;
158 end do;
159 end do;
160
161 AestheticQRcode := YUVtoRGB(AestheticQRcode);
162 return AestheticQRcode;
163 end proc;
```