

修士論文

カラーQRコードの実装とその応用に
関する研究

(Implementation of color QR codes and a study on detectable
illustration generation methods using image generation AI)

池田 新

立命館大学大学院
理工学研究科電子システム専攻

2026年3月

内容梗概

近年、技術的背景や、権利的背景、社会的背景から、QRコード(Quick Response code)が広く普及している。QRコードは、高速読み取りが可能な二次元コードの一種であり、従来使用してきたバーコードに比べて、記録可能な情報量が多く、誤り訂正能力が強いかつ高速な読み取りが可能という技術的な優位性があった。また、自動車部品のトレーサビリティ管理のために開発されたという背景がありながらも、特許権を行使しないと宣言されたことで世界中での特許フリーな使用が可能となっている。さらには、スマートフォンのような、高性能かつ多機能なデバイスが世界的に普及したこと、広告媒体や、決済手法としてQRコードの利用がなされている。しかし、現状のQRコードは単一の色の濃淡で表現されており、その色については、符号化する情報量増加の余地が残されている。そこで本研究では、RGBを用いたカラーQRコードシステムの実装を行い、その有用性について検討した。本研究で扱うカラーQRコードは、RGBの単色カラー画像に変換したQRコードを重ね合わせることにより作成し、計8色(赤、青、緑、シアン、マゼンタ、イエロー、白、黒)のカラー画像となっている。具体的には、カラーQRコードの検出アプリの実装、従来のQRとの検出距離の比較実験を行った。また、近年技術向上が進んでいる画像生成AIを使用したQRコードの検出が可能なイラスト生成AIの実装方法について検討を行った。本研究では、撮影距離約20cmの近距離において安定した検出が可能な読み取りアプリを実装した。さらに、カラーQRコードを入力画像としたQRコード検出可能な画像の生成を目的として、画像の生成環境を構築し、既存のControlNetを用いた画像生成を通して、生成手法の検討を行った。

目 次

第1章 序論	1
1.1 研究背景	1
1.2 QR コードについて	2
1.3 本研究の目的	5
第2章 カラー QR コードに関する先行研究	7
2.1 ステゴパネル	7
2.2 三重大学 寺田らによる研究	9
2.3 Henryk Blasinski による研究	11
第3章 カラー QR コードエンコード/デコードシステムの開発	13
3.1 エンコード方法	13
3.2 色空間について	15
3.3 デコード方法	16
3.4 読み取りアプリの開発	20
第4章 検出性能の比較実験	23
4.1 実験手法	23
4.2 実験結果	28
4.3 考察	29
第5章 カラー QR コードを用いた AI イラストの生成	31
5.1 イラストに見える QR コード	31
5.2 カラー QR コード入力によるアート QR コード生成の利点	35
5.3 イラスト生成 AI の原理	36
5.3.1 GAN	37
5.3.2 潜在拡散モデル	37
5.4 使用したソフトとモデル	38
5.4.1 VAE (variational auto encoder)	40
5.4.2 チェックポイント (ckpt)	40
5.4.3 controlnet	40
5.4.4 K サンプラー	40
5.4.5 LoRA	40

5.5 実験内容	41
5.6 実験結果	41
5.6.1 各 controlnet による出力画像への影響	41
5.6.2 3 種類の controlnet を使用した画像生成	44
5.7 考察	45
第 6 章 結論	47
6.1 まとめ	47
6.2 今後の展望	47
6.2.1 カラー QR コードの読み取りシステムについて	47
6.2.2 カラー QR コード入力によるアート QR コードの生成について	48
参考文献	48
謝辞	57
発表論文リスト	59

図 目 次

1.1 電子機器の普及率の推移	2
1.2 QR コードの構造	3
1.3 QR コードのバージョン	4
1.4 フайнダーパターンの原理	5
2.1 肉眼で観測したステゴパネル	7
2.2 カメラで観測したステゴパネル	8
2.3 フリッカーノイズが入ったカメラ画像	8
2.4 ステゴパネルの点滅周波数と文字表現の仕組み	9
2.5 カラーサンプル画像の元データ	10
2.6 学習用カラー QR コードの元データ (パターン A)	10
2.7 学習用カラー QR コードの元データ (パターン B)	10
2.8 使用色 8 色のカラー QR コード	11
2.9 使用色 64 色のカラー QR コード	11
2.10 Blasinski による色干渉除去アルゴリズムの開発	12
3.1 加法混色	14
3.2 カラー QR コードのエンコード方法	14
3.3 RGB 色空間における色の表現例	15
3.4 HSV 色空間での色の表現方法	16
3.5 カラー QR コードのデコード方法	17
3.6 デコード時のフローチャート	18
3.7 膨張処理のイメージ	19
3.8 収縮処理のイメージ	19
3.9 読み取りアプリの web 画面表示	21
3.10 QR コード不検出時の web 画面表示	22
4.1 実験に使用したカラー QR コード画像	24
4.2 実験時の画像撮影環境	25
4.3 距離 20 cm から撮影したときの画像	25
4.4 距離 40 cm から撮影したときの画像	26
4.5 距離 60 cm から撮影したときの画像	26

4.6	距離 80 cm から撮影したときの画像	27
4.7	比較対象とするモノクロ QR コードの画像	27
4.8	膨張処理前の画像	29
4.9	膨張処理後の画像	30
5.1	生成 AI の市場規模予測	31
5.2	アート QR コードの例 1	32
5.3	アート QR コードの例 2	33
5.4	アート QR コード生成の流れ	34
5.5	QRr コードの調整に使用した QR コード生成サイト	34
5.6	アート QR コード生成手法 A	35
5.7	アート QR コード生成手法 B	36
5.8	カラーアート QR コードの仕様の例	36
5.9	GAN による生成のイメージ	37
5.10	GAN による生成のイメージ	38
5.11	comfyUI の操作画面	39
5.12	画像生成に使用した入力画像	42
5.13	brightness の強度による比較	42
5.14	qrcode_monster_v1 の強度による比較	43
5.15	qrcode_monster_v2 の強度による比較	43
5.16	T2Iadapter_color の強度による比較	43
5.17	3 種類の controlnet を使用した生成結果 (1)	44
5.17	3 種類の controlnet を使用した生成結果 (2)	45
6.1	カラー QR コードの検出率を向上させる改善案	48

表 目 次

4.1	カラー QR コードサンプルに使用した文字列	24
4.2	色抽出処理のみを行った際の検出率の比較 [%]	28
4.3	膨張処理時にカーネルサイズ 5×5 を適用した際の検出率 [%]	28
4.4	膨張処理時にカーネルサイズ 4×4 を適用した際の検出率 [%]	28
4.5	膨張処理時にカーネルサイズ 3×3 を適用した際の検出率 [%]	28
4.6	膨張処理時にカーネルサイズ 2×2 を適用した際の検出率 [%]	29
5.1	使用したプロンプト	35

第1章 序論

1.1 研究背景

近年、高速かつ安定した読み取りが可能な2次元コードであるQRコード(Quick Response code)が世界的に使用されている。このような技術の普及には、大きく分けて三つの背景がある。それは、技術的背景、権利的背景、社会的背景である。

始めに、技術的背景について述べる。これはQRコードが持つ技術的な優位性に由来するものであり、従来使用してきたバーコードと比較して、QRコードは記録可能な情報量が多く、誤り訂正能力が強い、かつ高速で安定した読み取りが可能というようにメリットが複数存在している。

次に、権利的な背景について述べる。QRコードは1994年に特許出願がなされたにもかかわらず、誰でも自由に使用可能である。QRコードは自動車部品のトレーサビリティ管理のためにデンソーの一事業部(現デンソーウェーブ)によって開発された?。しかしながら、デンソーはその仕様をオープンにし、特許の自由な使用を許可した。これにより、QRコード決済や、航空券等のチケット管理など、本来開発者が想定していなかったような用途でも使われ、社会に広く浸透していった。

最後に社会的な背景について述べる。これはQRコードの読み取りを可能にする電子機器、特にスマートフォンの普及に由来する。半導体製造技術の向上により、高性能かつ低消費電力な電子デバイスが安価に手に入るようになったことで、2010年代からスマートフォンが急速に普及してきた。図1.1は、日本における電子機器の普及率の推移を示したものである。総務省の情報通信白書によると、日本においてスマートフォンの保有率は2010年で10%程度に留まっていたものが、2021年には90%近くに達している[1]¹。このスマートフォンの普及は、誰もがQRコードを読み取り、情報を取得する事ができるという土台を作り、QRコードの普及を強く後押しした。

¹総務省 令和4年版 情報通信白書 <https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r04/html/nd238110.html>
より引用

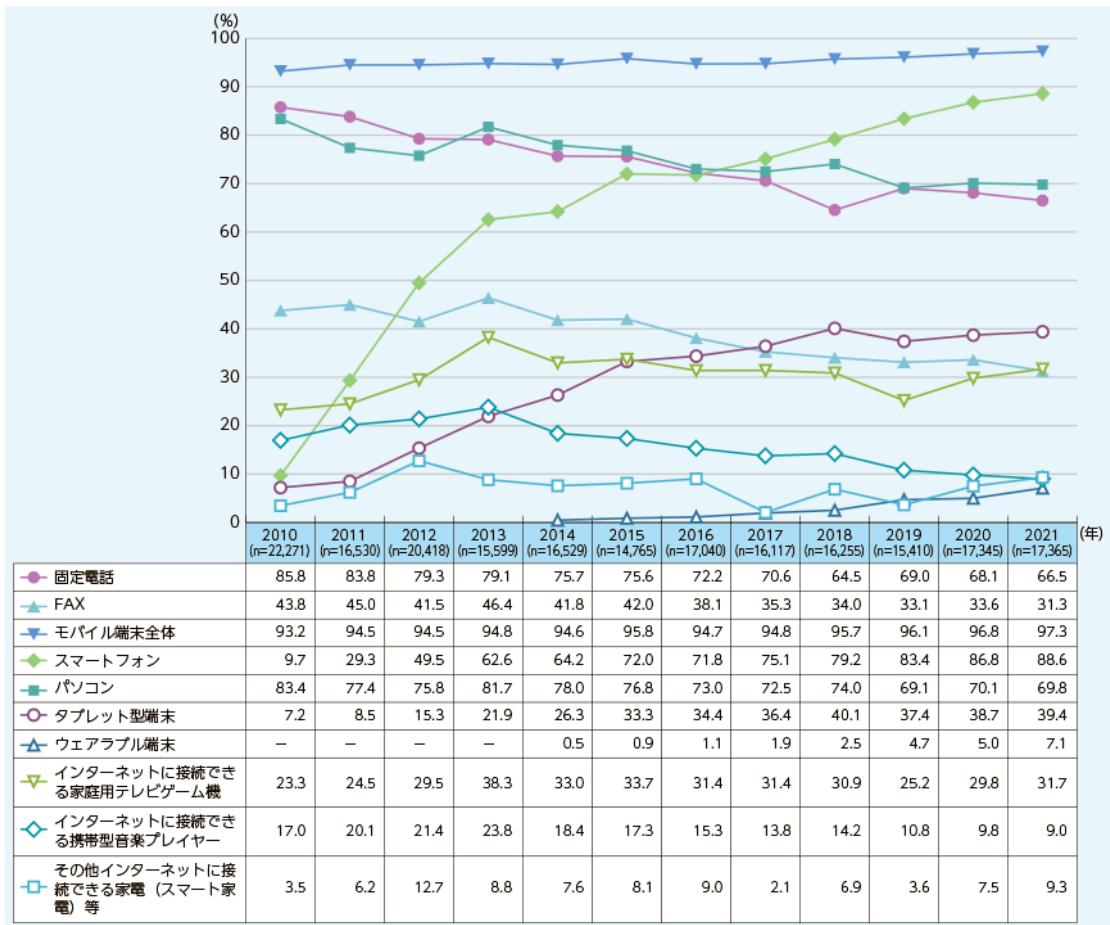


図 1.1: 電子機器の普及率の推移

1.2 QR コードについて

QR コード (Quick Response code) は、1994 年に株式会社デンソーの一事業部 (現株式会社デンソーウェーブ) によって開発された、高速かつ安定した読み取りが可能なマトリクス型の 2 次元コードの一種である。本コードの特徴的な形状は囲碁が元となっている。同様に広く普及しているバーコードと比較して、縦と横に情報を格納していることから、より大容量のデータをエンコードすることができる、なおかつ情報密度の高い印字をすることができる。ここで QR コードの構造について述べる。図 1.2 は QR コードの構造を簡易的に表したものである。コードを構成する白黒の正方形はセルと呼ばれる。このセルの組み合わせにより情報が表され、QR コード内部には、切り出しシンボル (図 1.2 中②)、タイミングパターン (図 1.2 中③)、アライメントパターン (図 1.2 中④)、フォーマット情報 (図 1.2 中

⑤) が存在している.²



図 1.2: QR コードの構造³

各構造の要素について、それぞれ述べる。

切り出しシンボル : 位置検出パターン、もしくはファインダーパターンとも呼ばれる。QR コードの 3 つの角に配置されており、「黒・白・黒・白・黒」の 1:1:3:1:1 の比率を持つ正方形パターンをもつ。QR コードの存在検出に使われ、画像中から QR コード領域を切り出す基準となる。

タイミングパターン : 切り出しシンボル同士を結ぶ黒と白が交互に並んだ直線状のパターンであり、最小セルの間隔の推定に関わる。

アライメントパターン : 切り出しシンボルより小型の中央に黒点を持つ正方形パターンであり、QR コードのバージョンが上がるほどに QR コード中に含まれる数が増加する。局所的な歪みの補正に使用される。

²QR コードってどういう仕組み?種類や歴史、使用時の注意点などを解説 | KDDI トビラ (<https://time-space.kddi.com/ict-keywords/20190425/2624>) より引用

³QR コードってどういう仕組み?種類や歴史、使用時の注意点などを解説 | KDDI トビラ (<https://time-space.kddi.com/ict-keywords/20190425/2624>) より引用

フォーマット情報 :QR コードの誤り訂正に関する情報を含む。誤り訂正方式の判定にかかる役割を持ち、正しいデータの復元を可能にする。

次に、QR コードの仕様により定められているバージョンについて述べる。QR コードは縦横を構成するセル数毎にバージョンが決められている(図 1.3)。バージョンは 1 から 40 まで設定されており、例えば、バージョン 1 では 21×21 セル、バージョン 40 では 177×177 セルとなっている。最小サイズであるバージョン 1 では、数字で 17 文字、英数字で 10 文字、漢字で 4 文字、バイナリで 7 ビットのデータ表現が可能である。これに対し、最大サイズであるバージョン 40 では、数字で 7,089 文字、英数字で 4,296 文字、漢字で 1,817 文字、バイナリで 2,953 ビットと大容量のデータ表現が可能である。また、QR コードには 3 つの角に配置されている特徴的なパターンがある。このパターンをファインダーパターンもしくは位置検出パターンと呼ぶ。デコード時にはこのパターンを利用することで、QR コードの位置特定を行う。具体的には、3 つの位置検出パターンを基準として、QR コードの向きや回転状態を検出し、正しい方向に補正する。その後、位置検出パターン間の距離を測定することでコードのサイズを認識し、3 つのパターンの相対的な配置を基に、QR コード全体の四角形領域を特定する。

図 1.4 に示すように、A、B、C のいずれの位置においても、白黒部分の幅の比率は 1:1:3:1:1 となっている。この比率は QR コードが回転した場合でも保持されるため、位置検出パターンの検出結果、及びそれらの位置関係から回転角度を認識することができる。その結果、 360° いずれの方向からでも読み取りが可能となり、処理の効率化が実現されている。



図 1.3: QR コードのバージョン⁴

⁴ [2] より引用

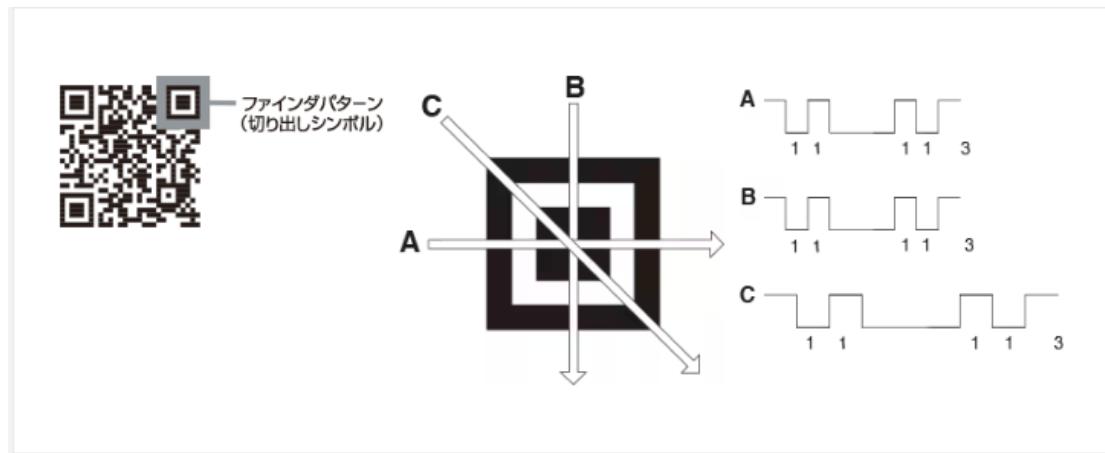


図 1.4: フайнダパターンの原理⁵

1.3 本研究の目的

本研究の目的は大きく二つに分けられる。一つ目は、QRコードのカラー化によって、面積当たりの情報量増加である。二つ目は、カラーQRコードの技術を用いて広告媒体等で他のデザインを損なわないような二次元コードを作成することである。

⁵ [3] より引用

第2章 カラーQRコードに関する先行研究

本章では、カラーQRコードに関する先行研究について述べる。

2.1 ステゴパネル

本研究のアイデアの元となった、肉眼では観認不可能な情報表示LEDパネルであるステゴパネルについて述べる。ステゴパネル(Stego-panel)とは、画像や音声などのデータに秘密情報を埋め込む情報秘匿技術の一種であるステガノグラフィー(Steganography)と、照明パネル(Panel)を組み合わせた造語である。ステゴパネルは、カメラで撮影することで、肉眼では観認できない情報を取得することが可能となる(図2.1及び、2.2)[4][5][6][7][8][9]。

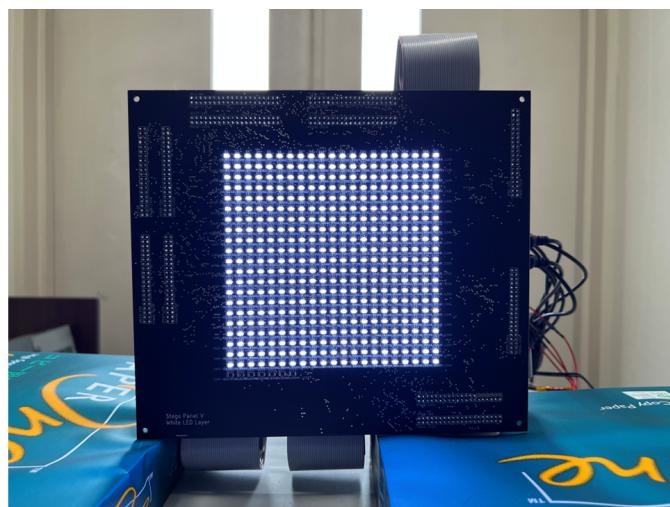


図 2.1: 肉眼で観測したステゴパネル

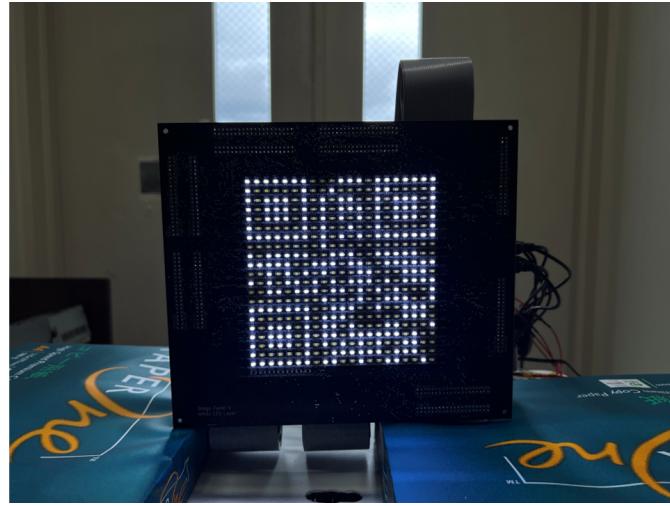


図 2.2: カメラで観測したステゴパネル

ステゴパネルの原理は、2種類の点灯周波数を使用したLEDの点灯制御である。肉眼ではおよそ 60 Hz 以下の点滅しか認識することはできない。反対に、60 Hz 以上の点滅は、肉眼には常時点灯していると視認される。このような現象は残像現象と呼ばれる。一方で、カメラによる撮影では、露光時間を制御することにより、肉眼に比べて瞬間の光を観測することができる。従って、カメラで高速点滅している照明やデジタルサイネージを撮影したときに、画像中に縞模様のノイズが表示される(図 2.3)。このような、ちらつき(Flicker)に由来するノイズはフリッカーノイズと呼ばれる。

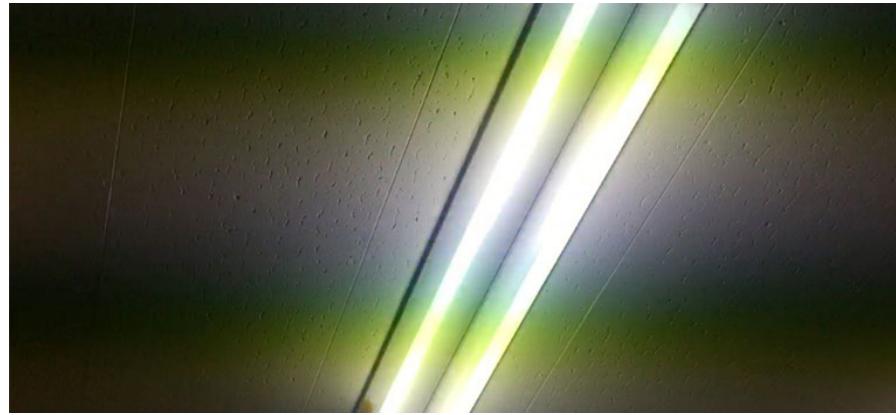


図 2.3: フリッカーノイズが入ったカメラ画像

図 2.4は、ステゴパネルの点滅周波数と文字表現の仕組みを簡易的に表したものである。ここでは、LED の点滅周波数を 20 Hz と 100 Hz と仮定している。ステゴパネルでは、文字情報部分と背景部分で違う点滅周波数を使用することで情報の秘匿を実現している。

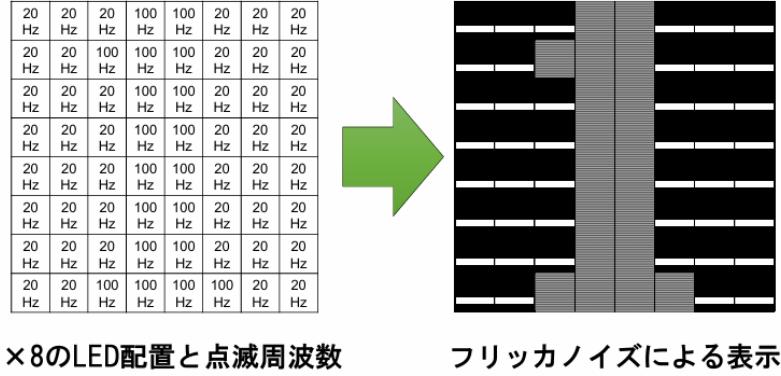


図 2.4: ステゴパネルの点滅周波数と文字表現の仕組み

このように、ステゴパネルとは、周波数の違いを利用して異なる情報を表示する技術である。これに対し、パネルに表示する色に応じて、異なる情報を表示できるのではないかというアイデアが生じた。この発案をもとにして、本研究内容であるカラー QR コードのシステムが想起された。

2.2 三重大学 寺田らによる研究

論文 [10] では、実用化されているカラーコードをまとめたうえで、それらに使用されている色数の少なさを課題としている。その解決策として、正確な色認識技術を提案し、ニューラルネットワークを用いた色の補完技術を構築している。更には、色補完技術の評価対象としてカラー QR コードを挙げ、色の認識技術に対する実験を行いその有効性を確認している。

当論文では、カラー QR コードを用いた認識実験として、2種類の実験を行っている。

- A. ニューラルネットワークに対する入力画像として、カラーサンプル画像(図 2.5)を使用した実験
- B. ニューラルネットワークに対する入力画像として、学習用からー QR コード(図 2.6, 及び 2.7)を使用した実験

各実験では、日付を変えて元データを撮影した画像をニューラルネットワークに学習させている。これにより作成した色認識技術と従来手法である閾値による色認識術との比較評価を行っている。比較対象には、使用色 8 色のカラー QR コード(図 2.8)と、使用色 64 色のカラー QR コードを使用している(図 2.9)。



図 2.5: カラーサンプル画像の元データ

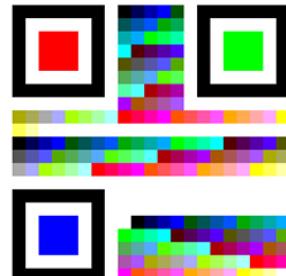


図 2.6: 学習用カラー QR コードの元データ (パターン A)

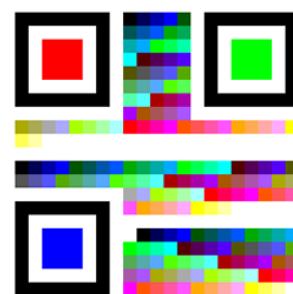


図 2.7: 学習用カラー QR コードの元データ (パターン B)

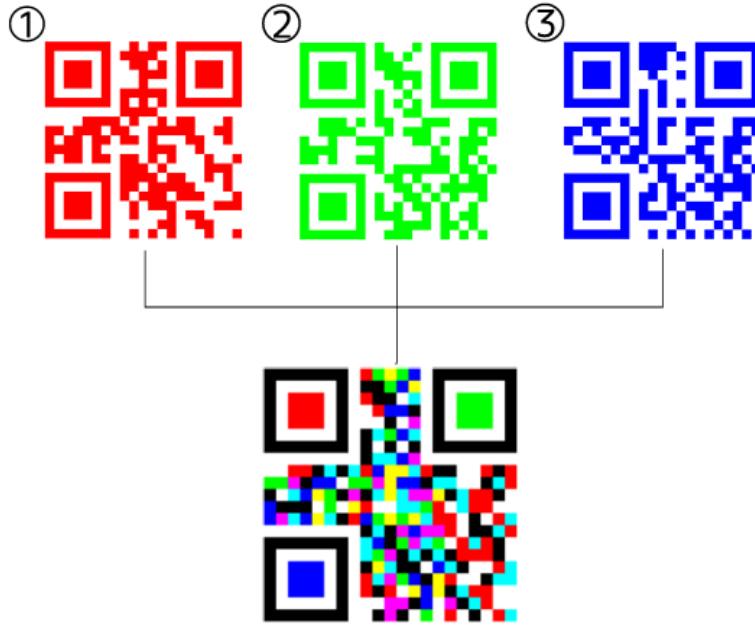


図 2.8: 使用色 8 色のカラー QR コード

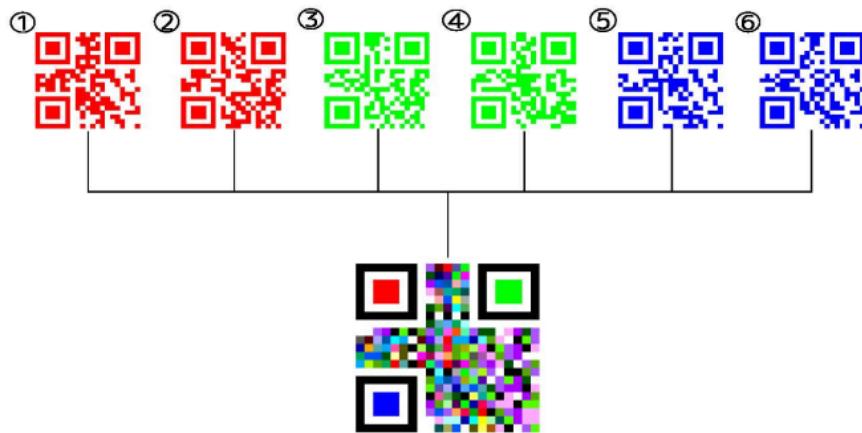


図 2.9: 使用色 64 色のカラー QR コード

2.3 Henryk Blasinski による研究

QR コードのカラー化に関する研究は、日本のみならず、海外でも行われてきた。論文 [11] では、カラー QR コード実装において、大きな課題となる色干渉 (color interference) について述べている。また、この課題に対し、印刷色材チャネルと撮影時の色チャネル間のチャネル間干渉の影響を軽減する事を目的として、印刷過程と撮影過程の物理的に動機付けられた数学モデルに基づいて、干渉除去アルゴ

リズムの開発を行っている(図 2.10)。性能を評価した実験の結果として、提案フレームワークは色干渉の影響をうまく克服し、対応する誤り訂正方式と併用することで、各色チャネルに対して低いビットエラー率と高いデコード率を実現することが示されたとしている。

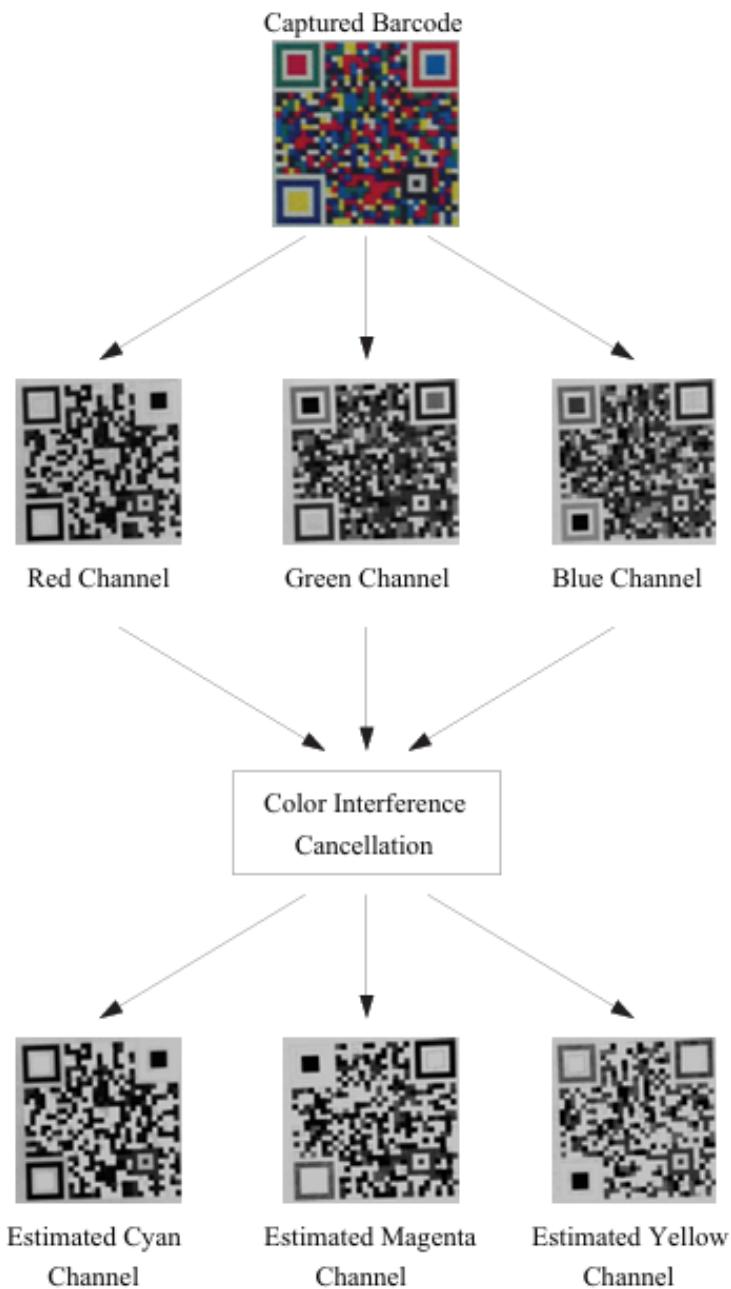


図 2.10: Blasinski による色干渉除去アルゴリズムの開発

第3章 カラーQRコードエンコード/ デコードシステムの開発

本章では、カラーQRコードの読み取りシステム開発について述べる。はじめに、3.1、及び3.2節で、エンコード方法とデコード方法を述べ。3.3節で読み取りアプリの開発について述べる。

3.1 エンコード方法

カラーQRコードの読み取りシステムに使用するエンコード方法を示す。はじめに、任意の文字列が一般のQRコード生成して、仕様によりモノクロのQRコードが作成される。同様に、3つの文字列をもとに3種類のモノクロQRコードを用意する。この3種類のモノクロQRコードを単色のカラー画像に変換する。本システムでは、赤(R), 緑(G), 青(B)の三色に変換している。その後、3種類の単色カラー画像を加法混色(図3.1)により重ね合わせることで、カラーQRコードが作成される(図3.2)。カラーQRコードは、赤、緑、青色の各色が画素値として0か255のどちらかの値をとる。つまり、 $2 \times 2 \times 2 = 2^3 = 8$ 通りの色で表される。ここで8通りの色とは、白、黒、赤、緑、青、シアン、マゼンタ、イエローの8種類となる。画素値を(R, G, B)で表記すると、各色はそれぞれ、白(255, 255, 255)、黒(0, 0, 0)、赤(255, 0, 0)、緑(0, 255, 0)、青(0, 0, 255)、シアン(0, 255, 255)、マゼンタ(255, 0, 255)、イエロー(255, 255, 0)である。

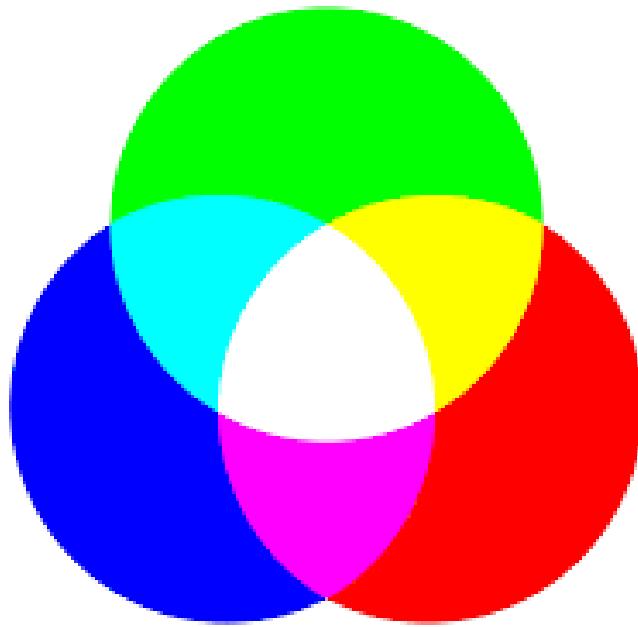


図 3.1: 加法混色

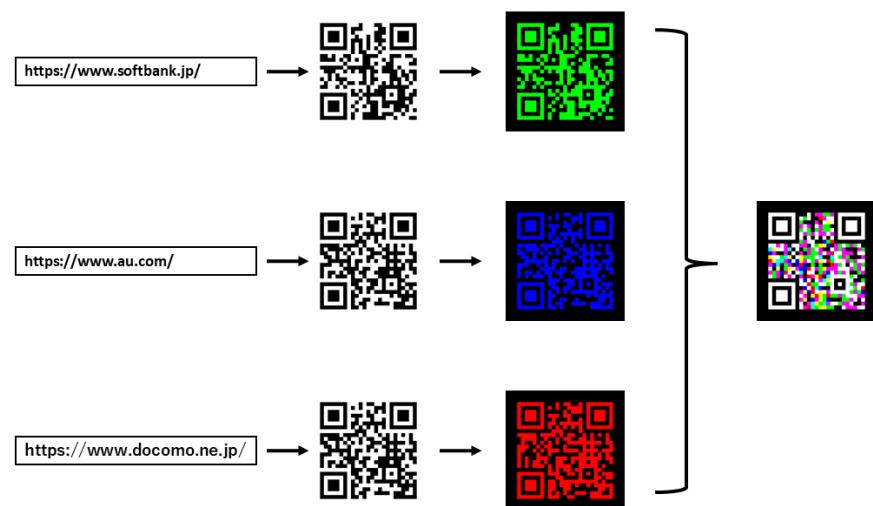


図 3.2: カラー QR コードのエンコード方法

3.2 色空間について

コンピュータ上で画像は、色の数値の組み合わせによる色空間により表現されている。最も主要な色空間としてRGB色空間 [12] が挙げられる。RGB色空間ではR（赤）、G（緑）、B（青）に0～255までの値が格納され、その値の組み合わせにより色が表現される。例として $(R, G, B) = (255, 165, 0)$ のとき、橙色となり、 $(R, G, B) = (255, 165, 0)$ のとき、灰色が表現される（図3.3）。また、別の色空間としてHSV色空間が挙げられる。HSV色空間 [13] とは、色を色相（H）、彩度（S）、明度（V）の3成分で表現する色空間のことである。HSVとは、Hue(色相)、Saturation(彩度)、Value(明度)の頭文字に由来している。HSV色空間を用いた色の表現例が図3.4である。RGB色空間がデバイス依存の色表現であるのに対して、HSV色空間は人間の色知覚に近い直感的な表現を目的として提案された。HSV色空間を使用する利点として、より正確な色抽出が可能な点が挙げられる。HSV色空間において、色の種類はH成分によって決定されるため、照明等による明るさの影響を抑えたロバスト性の高い色抽出が可能になる。

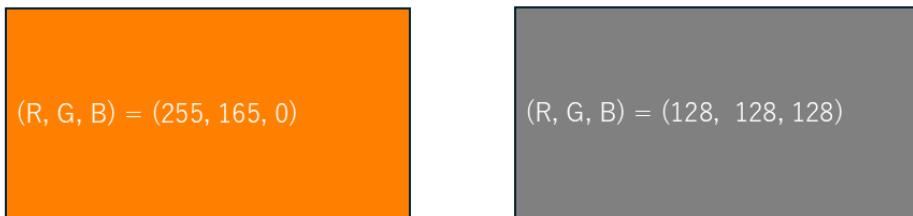


図 3.3: RGB 色空間における色の表現例

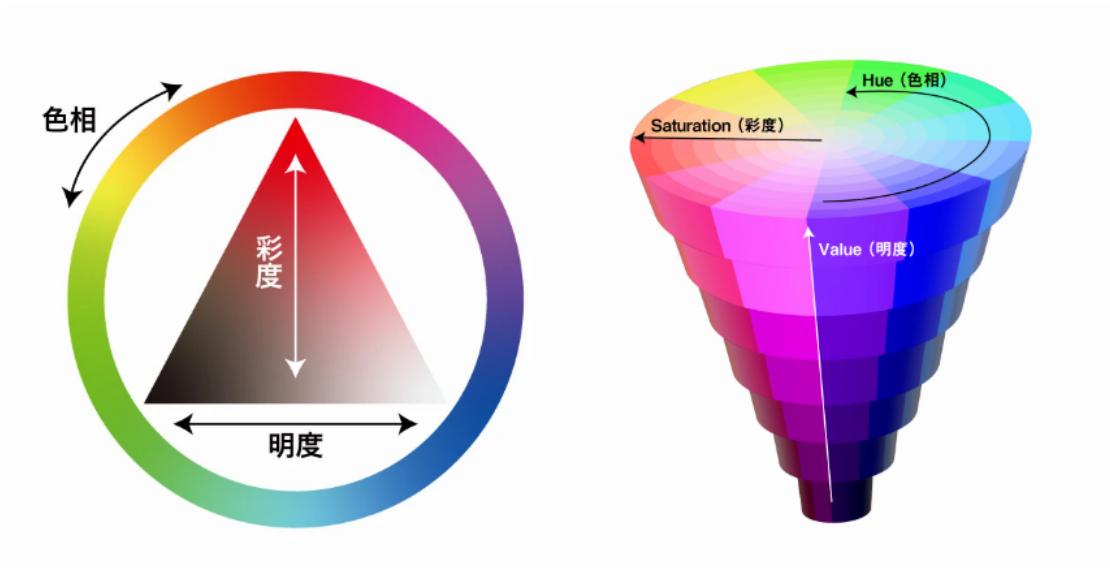


図 3.4: HSV 色空間での色の表現方法¹

3.3 デコード方法

図 3.5は、カラー QR コードの読み取りシステムにおけるデコード方法を示したものある。はじめに、カラー QR コードに対して、画像処理による色の抽出を行う。デコードのフローチャートが図 3.6である。

¹321web—色の三属性「色相」「明度」「彩度」とは?【HSB/HSV】(<https://321web.link/color-attribute/>) より引用

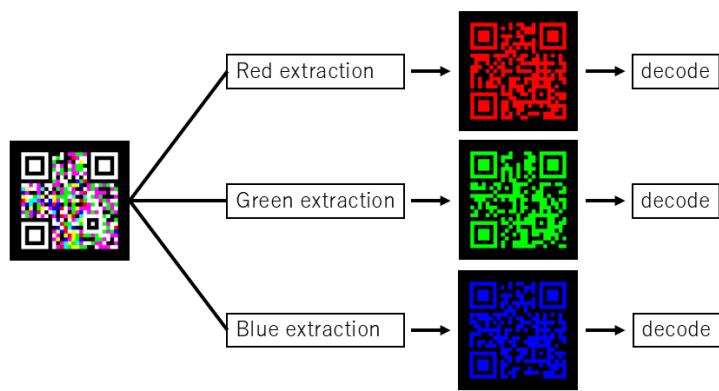


図 3.5: カラー QR コードのデコード方法

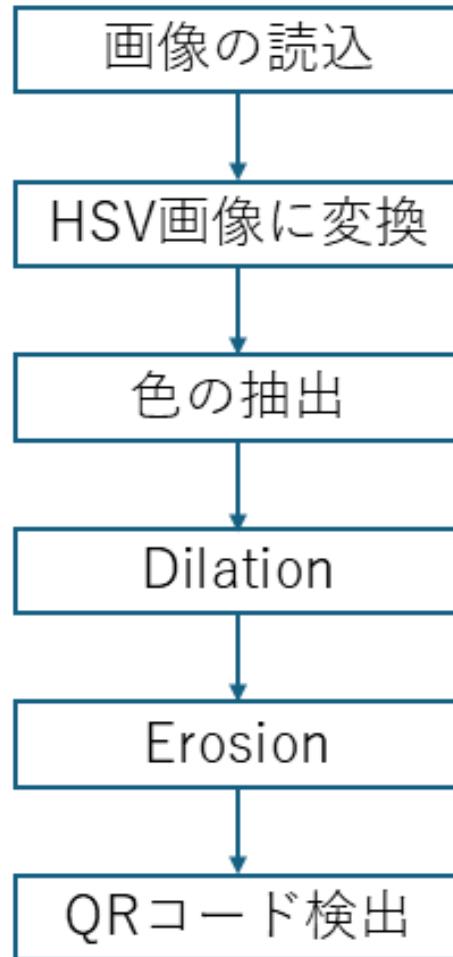


図 3.6: デコード時のフローチャート

ここで、膨張 (dilation) 処理と収縮 (erosion) 処理について詳しく述べる。膨張処理とは、画像中の指定個所を太くする処理のことである。例えば、白色の箇所に対して膨張処理を行う場合、対象とする構造要素 (カーネル) に対して、周囲も白色に塗りつぶすような処理が行われる。膨張処理の処理イメージが図 3.7 である。

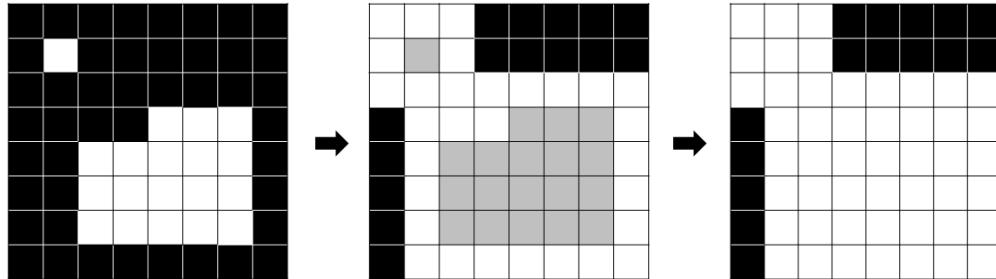


図 3.7: 膨張処理のイメージ

次に、収縮処理とは、画像中の指定個所を細くするような処理のことである。例えば、白色の箇所に対して収縮処理を行う場合、対象とする構造要素(カーネル)に対して、周囲の黒い画素に浸食されるような処理が行われる。収縮処理の処理イメージが図 3.8 である。

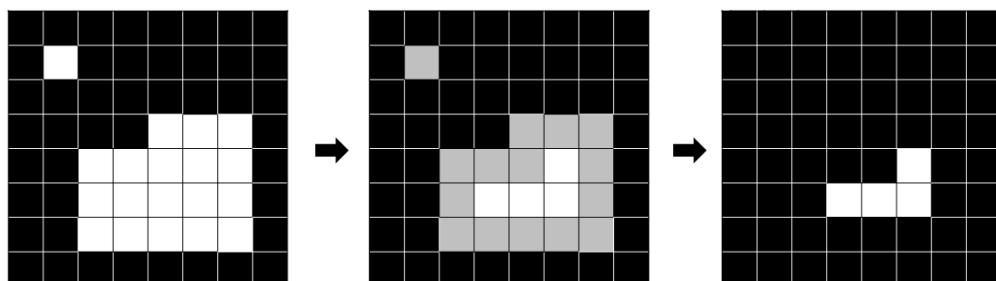


図 3.8: 収縮処理のイメージ

このように画像の形状を整えるような処理を総称してモルフォロジー処理と呼

ばれる。また、収縮の後に膨張を行う処理をオープニング(opening)処理、膨張の後に収縮を行う処理をクロージング(closing)処理と呼ぶ。本システムでは、ノイズ除去を目的として、クロージング処理を適用している。

また、色の抽出処理に関しては、OpenCVによるマスクの生成を使用している。図??は、カラーQRコードの読み取りアプリの一部である。ここでは、赤色に対する色の抽出処理を行っている。ここで、取得された画像は、cv2.cvtColorにより、RGB画像から、HSV画像に変換される。次に、np.arrayにより、各色の範囲が定義される。赤色が抽出される範囲には黄色や、マゼンタも含まれるため、それぞれ定義がなされる。さらに、cv2.inRangeにより、抽出したペアツ同士のマスクが生成される。最後に、cv2.bitwise_orを用いて、別々に抽出した白、赤、黄、マゼンタの範囲が結合される。ここで定義された範囲を黒に塗りつぶし、他の領域を白に塗りつぶすことで、カラーQRコードから個別のQRコードが抽出される。

```

80
81 # BGRからHSVに変換
82 hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
83
84 # 赤要素の抽出
85 if i == 0:
86     # 白色範囲
87     lower_white = np.array([0, 0, 100])
88     upper_white = np.array([180, 60, 255])
89
90     # 赤色範囲 ([0-180]の範囲で設定)
91     lower_red1 = np.array([0, 80, 100])
92     upper_red1 = np.array([15, 255, 255])
93     lower_red2 = np.array([165, 80, 100])
94     upper_red2 = np.array([180, 255, 255])
95
96     # 黄色範囲
97     lower_yellow = np.array([15, 80, 100])
98     upper_yellow = np.array([45, 255, 255])
99
100    # マゼンタ範囲
101    lower_magenta = np.array([135, 80, 100])
102    upper_magenta = np.array([165, 255, 255])
103
104    # 各色のマスクを作成
105    mask_white = cv2.inRange(hsv_image, lower_white, upper_white)
106    mask_red1 = cv2.inRange(hsv_image, lower_red1, upper_red1)
107    mask_red2 = cv2.inRange(hsv_image, lower_red2, upper_red2)
108    mask_yellow = cv2.inRange(hsv_image, lower_yellow, upper_yellow)
109    mask_magenta = cv2.inRange(hsv_image, lower_magenta, upper_magenta)
110
111    # 赤色の2つのマスクを結合
112    mask_red = cv2.bitwise_or(mask_red1, mask_red2)
113
114    # 白、赤、黄、マゼンタのマスクを結合して黒にする領域を指定
115    mask_to_black = cv2.bitwise_or(mask_white, mask_red)
116    mask_to_black = cv2.bitwise_or(mask_to_black, mask_yellow)
117    mask_to_black = cv2.bitwise_or(mask_to_black, mask_magenta)
118
119    # すべての領域を白に塗りつぶし
120    result = np.ones_like(image) * 255
121
122    # マスクで指定された領域を黒に変換
123    result[mask_to_black > 0] = [0, 0, 0]
124

```

図 3.9: 赤色に対する色の抽出処理

3.4 読み取りアプリの開発

前述のエンコード、デコード方法を使用して、読み取りアプリの開発を行った。開発にあたり Python 製の軽量 Web フレームワークである Flask を使用し、画像保存時のデータベースとしては、SQLite を使用した。また、バックエンド側の画像処理には、OpenCV を使用した。さらに、QR コードの認識には、ZBar を Python から利用するライブラリである pyzbar を使用した。ここで、Zbar とは、バーコードや QR コードなどの 2 次元コードを読み取る為の、フリーかつオープンソースで利用できるライブラリのことである。

開発したアプリにスマートフォン端末からアクセスした際の表示画面が図 3.9 である。本システムの実装はノート PC にローカルのサーバーを立てることで実装を行った。従って、スマートフォンをローカルサーバーと同一の Wi-Fi に接続するなど、同一のネットワーク内のみで動作検証が可能である。ユーザーは”ファイルを選択”ボタンからカメラによる画像撮影が可能である。次に、画面中央の各色の抽出ボタンを選択可能である。各ボタンを押すことで、抽出する色の種類を選択することが可能であり、バックエンド側での画像処理が実行される。QR コードが検出された際には、検出先の URL へ遷移される。また、QR コード不検出の場合には、画面中央にポップアップが表示される(図 3.10)。

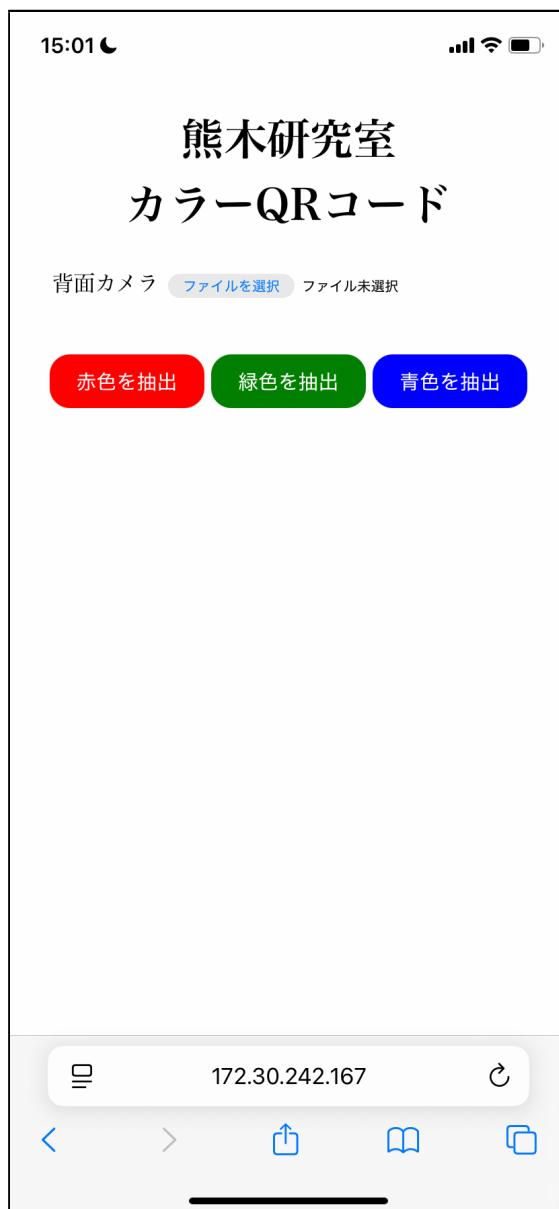


図 3.10: 読み取りアプリの web 画面表示

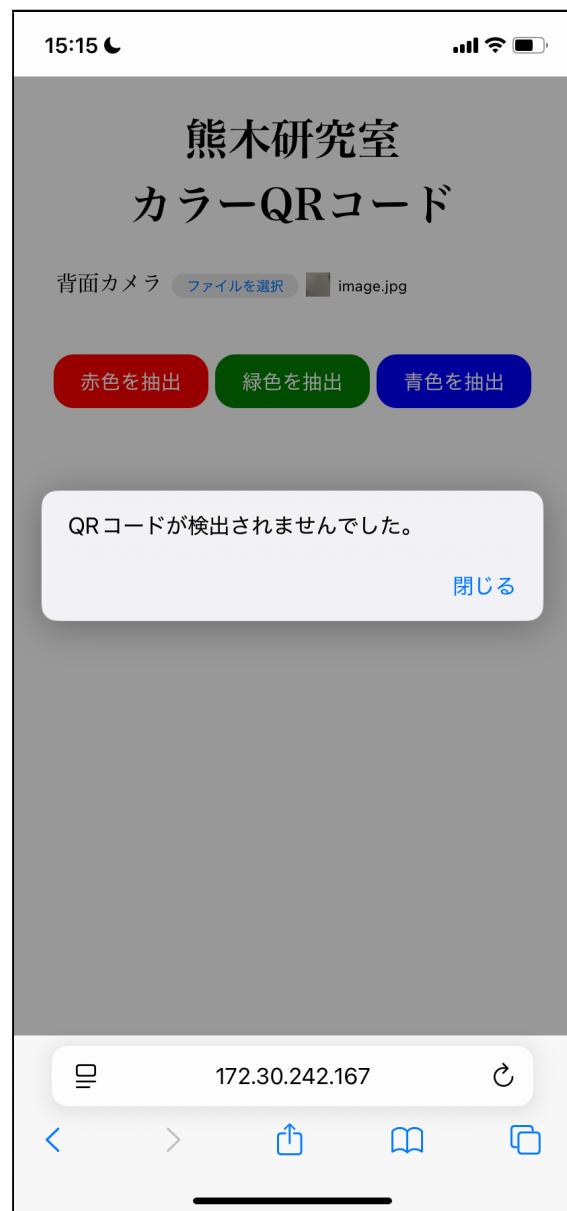


図 3.11: QR コード不検出時の web 画面表示

第4章 検出性能の比較実験

本章では、従来のモノクロ QR コードとカラー QR コードの検出率による比較実験について述べる。はじめに、4.1 節で実験手法について述べ、4.2 節で実験結果について述べる。また、4.3 節では実験に対する考察を述べる。

4.1 実験手法

実験に際して、10種類のカラー QR コード画像を用意した。これは、カラー QR コードの種類による検出への影響を調べる為である。用意した10種類のカラー QR コードが図 4.1 である。これらのカラー QR コードは、ランダムに生成した10文字の半角英数字を3種類用意し、先頭にサンプル番号を追加した計11文字の半角英数字をエンコードすることで作成している（表 4.1）ここで、サンプル番号とは sample2 であれば、数字の 2 を表す。また、画像の撮影には、apple 社製のスマートフォン端末である iPhone XR を使用した。搭載されている背面カメラの画素数は 1200 万画素である¹。

撮影距離について、20 cm, 40 cm, 60 cm, 80 cm の4種類から撮影を行った。撮影時の実験環境を図 4.2 に示す。

¹iPhone XR - 技術仕様—<https://support.apple.com/ja-jp/111868> より引用

表 4.1: カラー QR コードサンプルに使用した文字列

	赤	緑	青
sample0	0bu2rh5g1aa	0xox7ljd0t4	0warg1fs2qj
sample1	1ajauukb8mj	1sjw40ypu0l	1klos4n5aq6
sample2	2arwtbe1h2x	2tndp3aqedx	2ptohyekclg
sample3	3apa5k0sxwa	3viyx8b2ndm	3nmrc7u0tb4
sample4	4deeqtvrucr	4wf5i1dsksi	4usyhe4fa31
sample5	5hjfteyeqim	5sf11bex4a3	5juyka6nm5g
sample6	6aekcjml32b	6i4d2i0q5le	6b3e3up784p
sample7	7cnk4rjdsuy	7frelcmy6lh	7e3qbglmuru
sample8	8anykgc3hhr	8y3mt7irher	8mia1e7bbxr
sample9	9sa1vx44ni4	9xiicrmnow8	9vjkso2scsd



図 4.1: 実験に使用したカラー QR コード画像

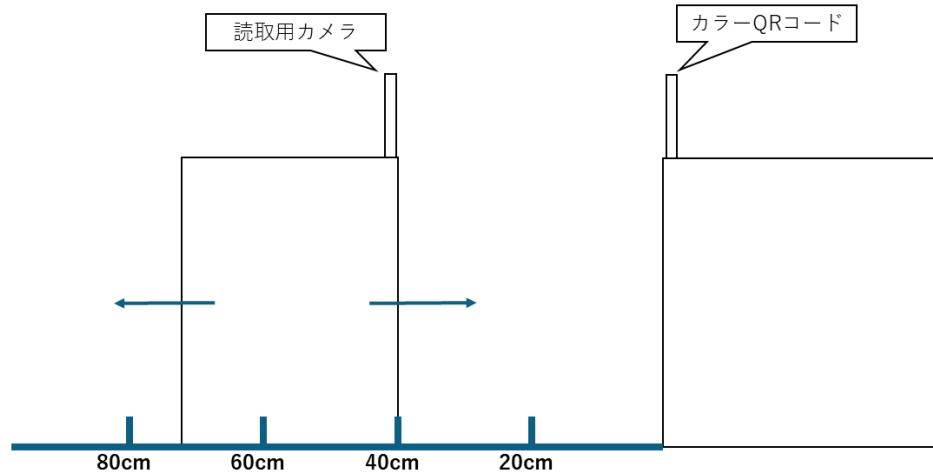


図 4.2: 実験時の画像撮影環境

撮影した画像を以下に示す。画像はそれぞれ、撮影距離 20cm, 撮影距離 40cm, 撮影距離 60cm, 撮影距離 80cm を表している。

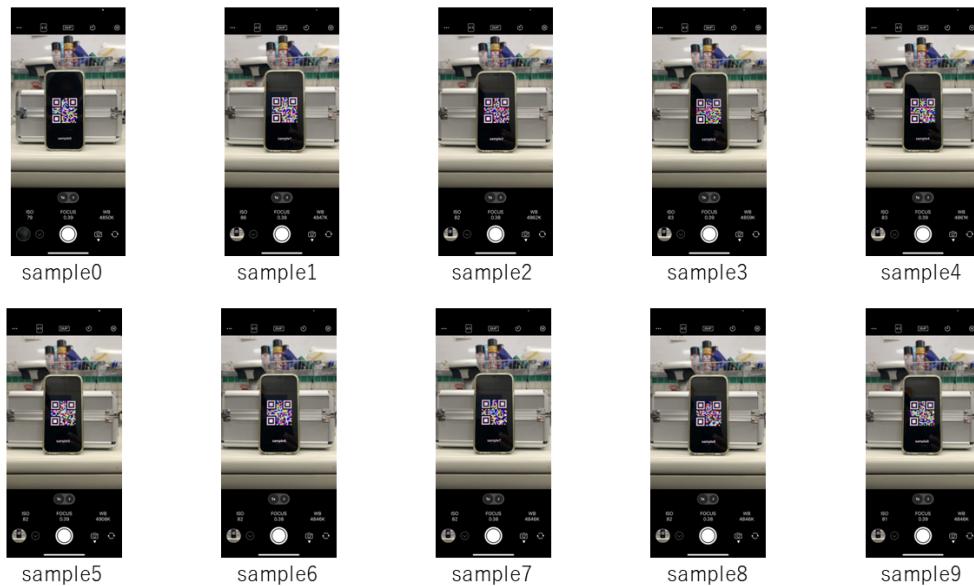


図 4.3: 距離 20 cm から撮影したときの画像

第 4 章

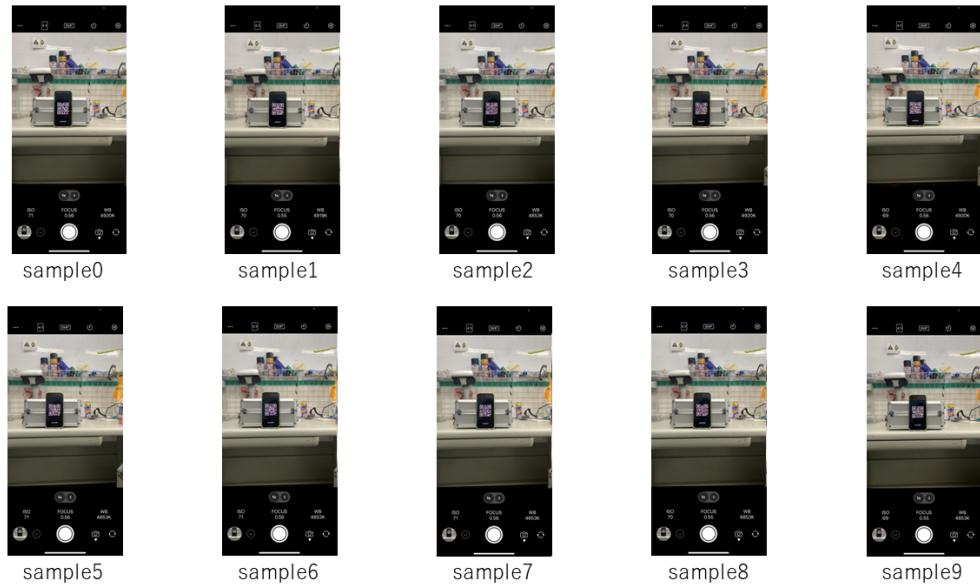


図 4.4: 距離 40 cm から撮影したときの画像

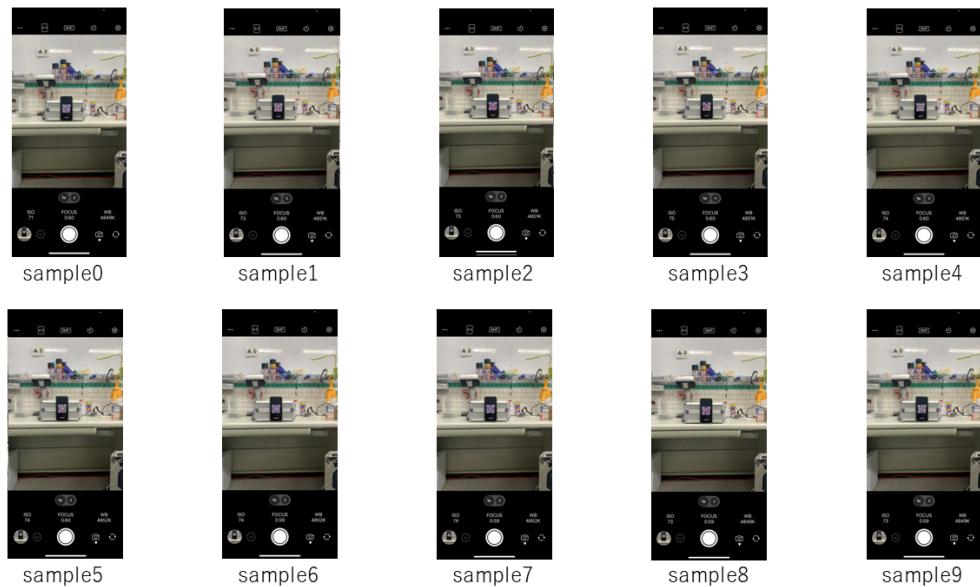


図 4.5: 距離 60 cm から撮影したときの画像

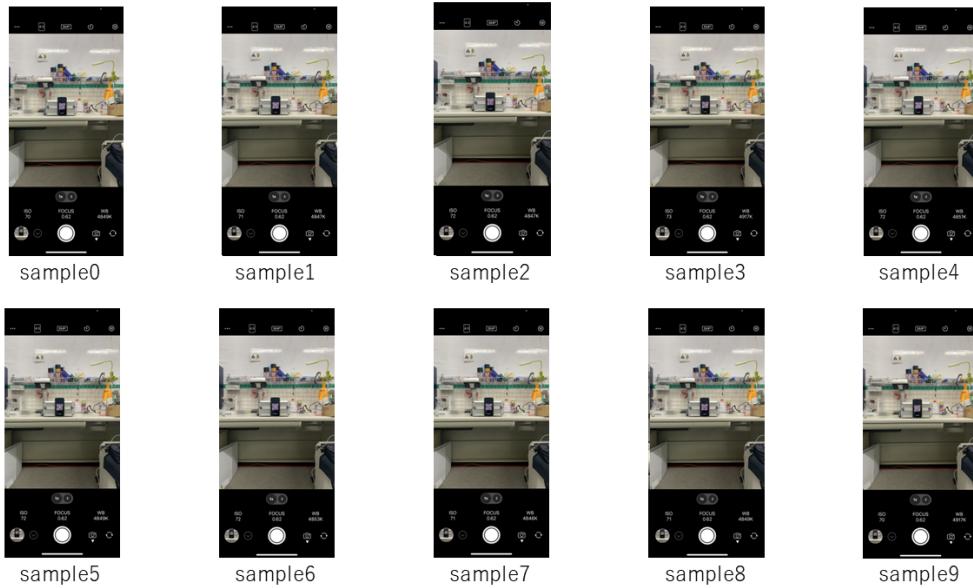


図 4.6: 距離 80 cm から撮影したときの画像

比較対象として、従来のモノクロ QR コードの撮影を行った。撮影された画像をまとめたものが図 4.7である。

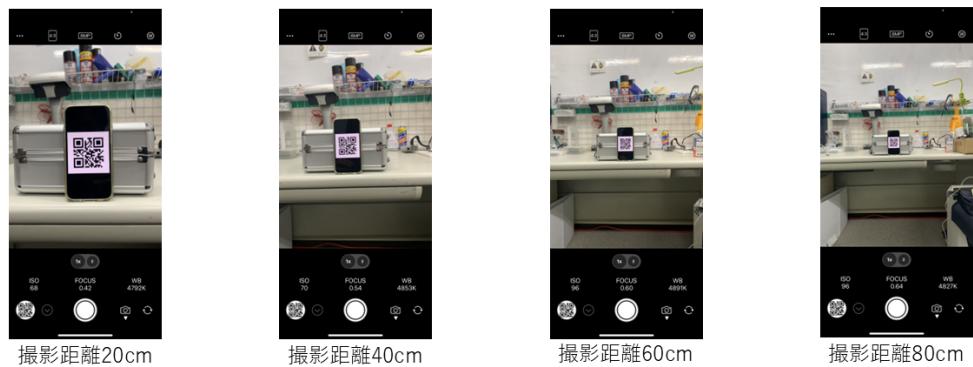


図 4.7: 比較対象とするモノクロ QR コードの画像

4.2 実験結果

はじめに、カラー QR コードに対して、色抽出処理のみを行って、その検出率の比較を行った。結果を表形式でまとめたものが表 4.2 である。

表 4.2: 色抽出処理のみを行った際の検出率の比較 [%]

	distance20cm	distance40cm	distance60cm	distance80cm
Blue	100%	20%	0%	0%
Green	100%	0%	0%	0%
Red	100%	40%	0%	0%

実験の結果を踏まえ、膨張処理を加えたうえで検出率の比較を行った。膨張処理に適用するカーネルサイズはそれぞれ、 5×5 , 4×4 , 3×3 , 2×2 を使用した。各カーネルサイズでの検出率をまとめたものが表 4.3, 4.4, 4.5, 及び 4.6 である。

表 4.3: 膨張処理時にカーネルサイズ 5×5 を適用した際の検出率 [%]

	distance20cm	distance40cm	distance60cm	distance80cm
Blue	100	0	0	0
Green	100	0	0	0
Red	100	0	0	0

表 4.4: 膨張処理時にカーネルサイズ 4×4 を適用した際の検出率 [%]

	distance20cm	distance40cm	distance60cm	distance80cm
Blue	100	0	0	0
Green	100	0	0	0
Red	100	0	0	0

表 4.5: 膨張処理時にカーネルサイズ 3×3 を適用した際の検出率 [%]

	distance20cm	distance40cm	distance60cm	distance80cm
Blue	100	20	0	0
Green	100	0	0	0
Red	100	40	0	0

表 4.6: 膨張処理時にカーネルサイズ 2×2 を適用した際の検出率 [%]

	distance20cm	distance40cm	distance60cm	distance80cm
Blue	100	20	0	0
Green	100	0	0	0
Red	100	40	0	0

これに対し、従来のモノクロの QR コードは、距離 80 cm からでも読み取りが可能であった。

4.3 考察

実験の結果として、膨張処理の追加による検出率の向上は見られなかった。これは、画像処理の対象となる QR コード部分の解像度の低さが問題であると考えられる。膨張処理前の画像と膨張処理後の画像は図 4.8、及び図 4.9 のとおりである。モルフォロジー処理に使用したカーネルサイズが対象とする QR コード領域に対して大きかったことで、想定したような色抽出後の画像補正がなされなかつたと考えられる。

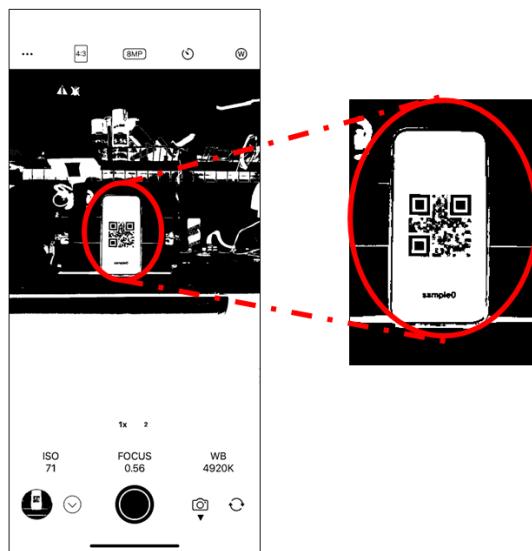


図 4.8: 膨張処理前の画像

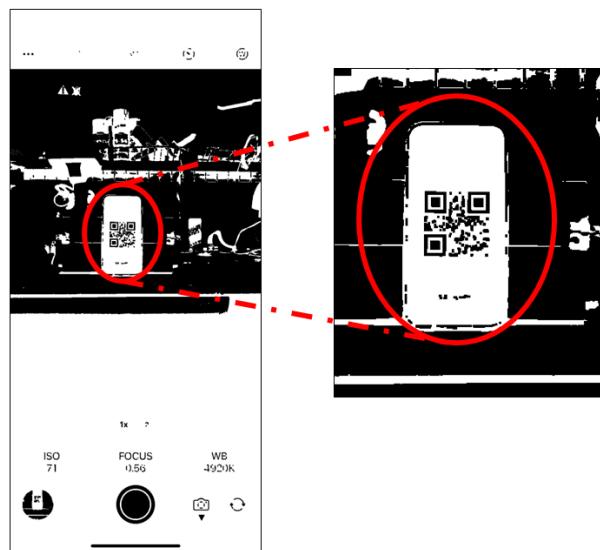


図 4.9: 膨張処理後の画像

また、一般に使用されている QR コードでは、同一サイズでも撮影距離 80cm から検出できることから、本実験で使用したコード検出システムは既存のものに大きく劣ることが分かった。

第5章 カラーQRコードを用いたAIイラストの生成

本章では、イラストに見えるQRコード画像について触れ、その技術の根幹となるイラスト生成AIの原理について述べる。また、カラーQRコードを使用した類似画像の生成手法について検討を行った。

5.1 イラストに見えるQRコード

近年、イラスト生成AI技術が急速に発展している。FORTUNE BUSINESS INSIGHTSの生成AI市場分析 [14]によると、世界の生成AI市場規模は2023年に438億7,000万米ドルと評価されていると述べている。また、この市場規模は年々増加し、年平均成長率は39.6%にまでのぼると推定している(図5.1)。

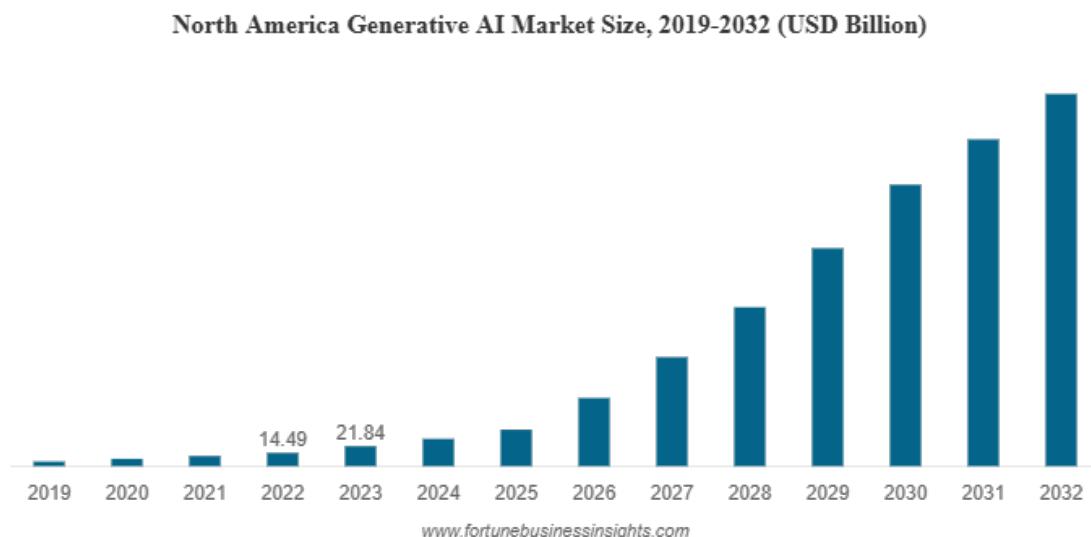


図5.1: 生成AIの市場規模予測

⁰Fortune Business Insights — 生成AI市場規模、シェア&業界分析、モデル別（生成官民ネットワークまたはGANSおよび変圧器ベースのモデル）、業界対アプリケーション、地域予測、2024-2032別—SSより引用

一方，このデータは生成 AI 全般を含めたものであり，イラスト生成 AI に限ったものではない。しかしながら，生成 AI の発展に伴い，イラスト生成 AI の技術進歩，また，社会への普及や，その市場規模も高まっていくと予測される。特にイラスト生成 AI の活用例として QR コードを入力画像として取り込むことで，イラストのように見えるが，実際に機能する二次元コードの生成が行われている。このような QR コードはアート QR コードと呼ばれている。その例が，図 5.2，及び 5.3 である。実際に QR コードリーダーでこれらの読み取りを行うと，それぞれ，(<https://qrbtf.com>) と，(https://note.com/st_ai) が得られる。

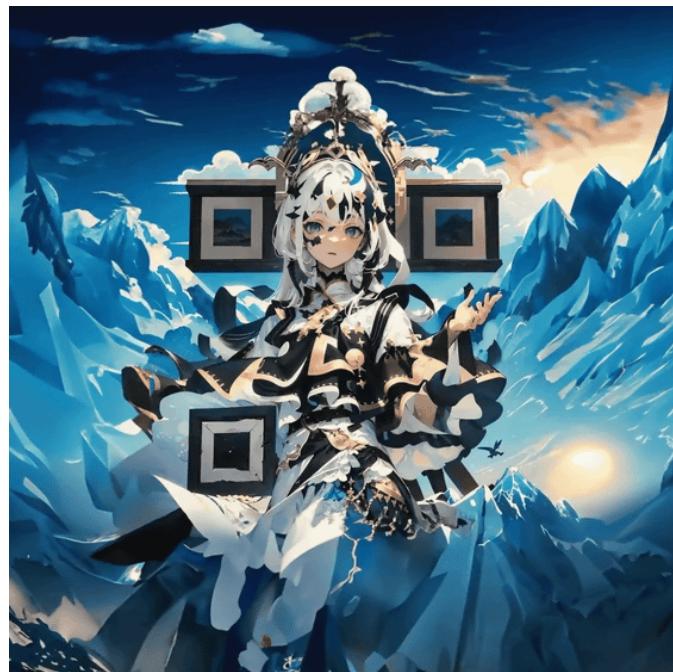


図 5.2: アート QR コードの例 1



図 5.3: アート QR コードの例 2

ここで, comfyUI を使用してアート QR コードを生成するための流れについて述べる。始めに, アート QR コードにするための元画像となる QR コードを用意する。次に QR コードのセルに対して調整を行い, QR コードセルの結びつきを弱める。最後に, イラスト生成 AI に対して入力画像として与える(図 5.4)。[15]によると, 特に, QR コードの調整処理が重要であるとしている。図 5.4 に記載のアート QR コードは, Anthony Fu's QR Toolkit¹を使用して, QR コードの調整を行った。Anthony Fu's QR Toolkit は web 上で公開されている QR コード生成サイトであり, 通常の QR コード生成に加え, 誤り訂正レベルの指定や, ピクセルスタイルの変更, 色の指定等を行うことができる(図 5.5)。また, アート QR コード生成時に使用したプロンプトを 5.1 に示す。

¹<https://qrcode.antfu.me/#verify>

第 5 章

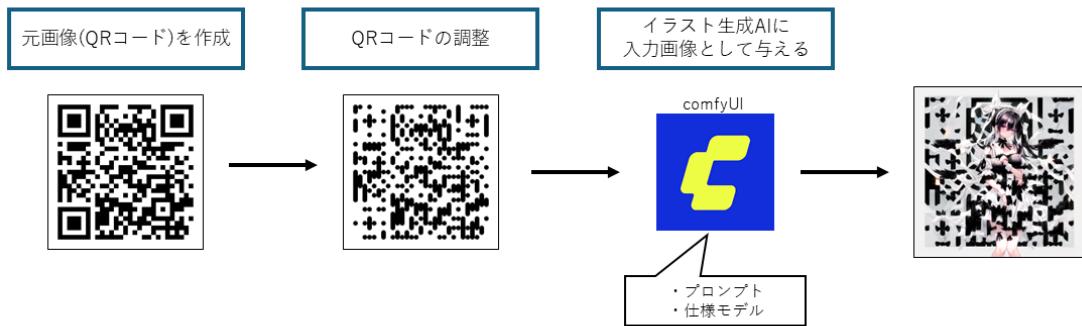


図 5.4: アート QR コード生成の流れ

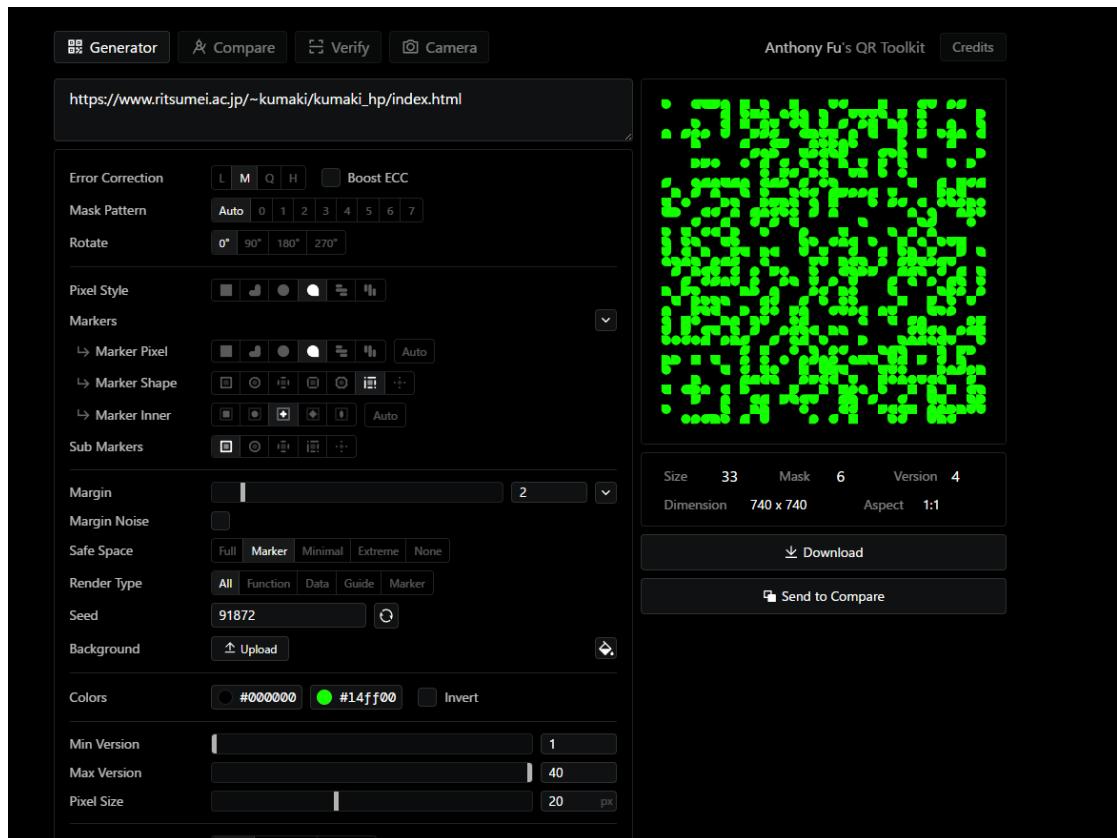


図 5.5: QRr コードの調整に使用した QR コード生成サイト

表 5.1: 使用したプロンプト

positive prompt	negative prompt
(1girl:1.3), solo, long_hair, breasts, looking_at_viewer, blush, black_hair, dress, bow, ribbon, closed_mouth, cleavage, bare_shoulders, medium_breasts, standing, purple_eyes, hair_ribbon, short_sleeves, frills, hairband, black_dress, wrist_cuffs, feet_out_of_frame, white_bow, white_ribbon, (black_ribbon:1.3)	embedding:EasyNegative, embedding:bad_prompt_version2-neg, embedding:verybadimagenegative_v1.3, (Multiple girls:1.3)

5.2 カラー QR コード入力によるアート QR コード生成の利点

現在のアート QR コード生成は、入力画像をモノクロの QR コードとするものにとどまっている。本節では、入力画像をカラー QR コードに拡張することを目標として得られる利点について述べる。

カラー QR コードに対応したカラーアート QR コード生成について 2 種類の方法が挙げられる。

- A. 3 種類の重ね合わせを行ったカラー QR コードを入力画像に使用する方法(図 5.6)
- B. 単色カラーの QR コードを入力画像として使用し、読み取りにカラー QR コード検出のシステムを使用する方法(図 5.7)

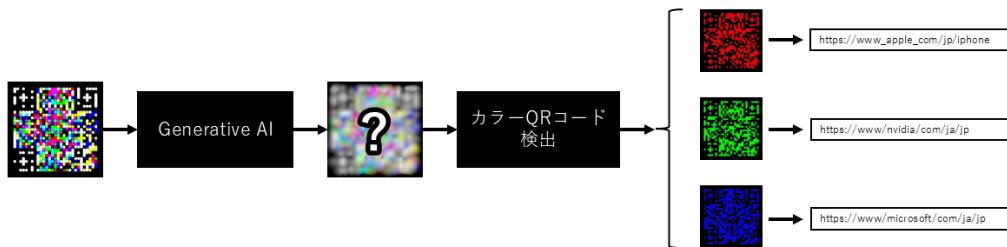


図 5.6: アート QR コード生成手法 A



図 5.7: アート QR コード生成手法 B

A の方法による利点として、情報量の増加が挙げられる。また、従来のアート QR コードと比較して、よりカラフルな画像の生成が行われると考えられる。

B の方法による利点として、よりイラストに近い形での画像生成が可能になることが挙げられる。例として、緑色単色 QR コードを使用して、カラーアート QR コードを生成する事を考える。この時、デコード時に緑色が抽出されることから、画素値でいう赤 (R) と青 (B) の値は任意の値をとれる。また閾値を使用して色の抽出を行う場合、QR コードの背景となる部分 (情報のないところ) では、赤 (R) と青 (B) の値は任意の値、緑 (G) は 128 以下の値で任意の値をとることができ (図 5.8)。

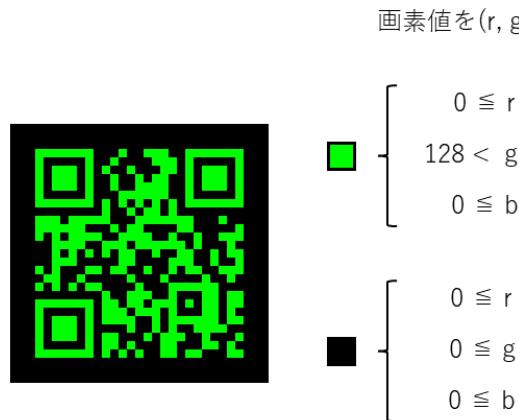


図 5.8: カラーアート QR コードの仕様の例

5.3 イラスト生成 AI の原理

代表的な生成モデルである GAN (敵対的生成ネットワーク) と Diffusion model (潜在拡散モデル) について述べる。

5.3.1 GAN

GANとは、Generative Adversarial Networkの頭文字である[16]。GANでは、生成モデル(Generator)と識別モデル(Discriminator)という2つのニューラルネットワークが互いに競いながら学習を行う構造をとる。生成モデルが実在する画像データに近い偽画像を生成し、一方で識別モデルは入力された画像が実が増加生成が増加を判別する役割を担う。

学習過程では、生成モデルは識別モデルを欺くような画像を生成しようとし、識別モデルはその判別精度を高めようとする。このような敵対的学习を繰り返すことで、より高品質な画像を生成することを目指すモデルである。GANによるイラスト生成の仕組みを表したもののが図5.9である。

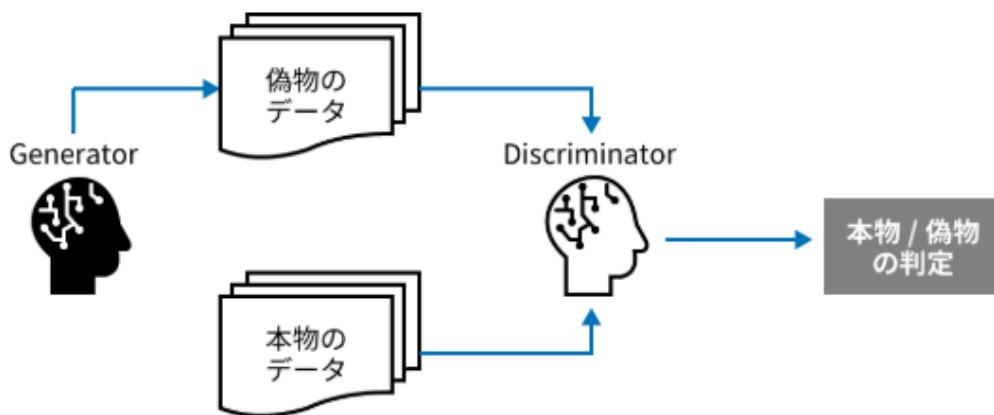


図5.9: GANによる生成のイメージ²

5.3.2 潜在拡散モデル

Diffusion（拡散モデル）は、近年の画像生成AIにおいて主流となっている生成モデルである[17]。拡散モデルは、画像に徐々にノイズを加えていく順拡散過程と、そのノイズを段階的に除去する逆拡散過程を学習することで、画像生成を実現する仕組みを特徴とする。学習段階では、実画像に対してランダムノイズを加え、そのノイズ成分を推定・除去する方法をモデルが学習する。

生成段階では、完全なノイズ画像を初期状態とし、学習済みモデルによってノイズを少しづつ取り除くことで、最終的に意味のある鮮明な画像を生成する。この過程は、ぼやけた画像に段階的に情報を付加し、徐々に明瞭な画像を完成させ

²Financial Engineering Group—敵対的生成ネットワーク
—https://www.feg.co.jp/analysis_tech/gan/より引用

る作業に例えられる。こうした逐次的かつ確率的な生成プロセスにより、拡散モデルは高い表現力と安定した学習特性を実現している。

また、GAN と比較して学習が安定しやすく、モード崩壊が起こりにくい点も拡散モデルの大きな利点である。この特性から、Stable Diffusion や DALL·E といった多くの先進的な画像生成 AI に採用されており、高品質かつ多様性のある画像生成が可能となっている。本研究で使用しているイラスト生成ツールである ComfyUI は、Stable Diffusion を基盤とした画像生成ツールである。ここで、Stable Diffusion は、潜在拡散モデルに基づく生成手法である。潜在拡散モデルによるイラスト生成の仕組みを表したもののが図 5.10 である。

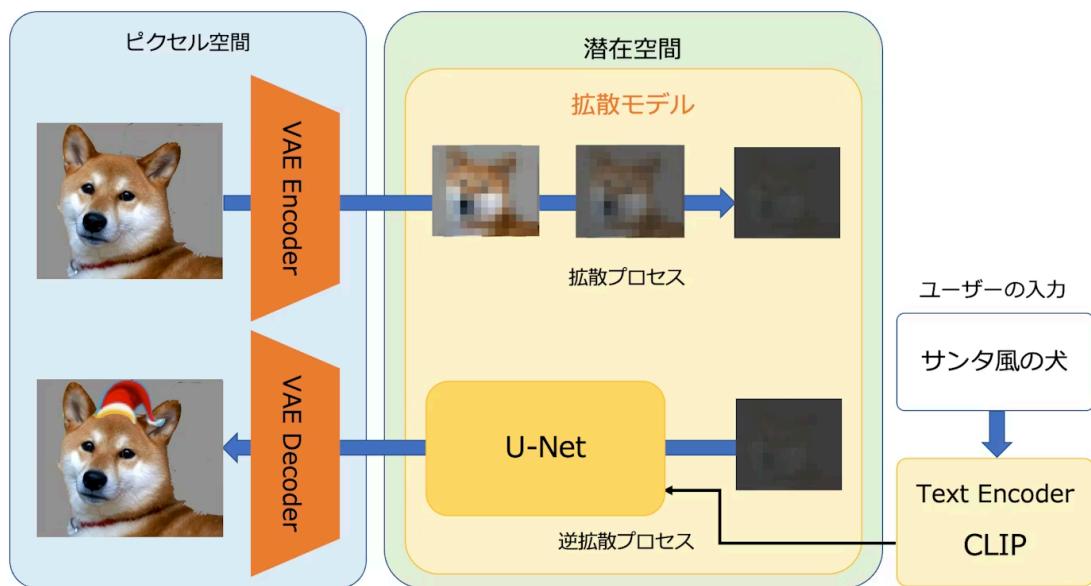


図 5.10: 潜在拡散モデルによる生成のイメージ³

5.4 使用したソフトとモデル

カラー QR コードを使用したイラスト生成を行うため、comfyUI⁴を使用した。ここで、comfyUI とは、イラスト生成 AI を独自にカスタマイズできるツールの一種である。本ツールでは、ノードと呼ばれる素子同士をつなぎ合わせることで、仕様モデルの選択や、モデルの強度の変更などの画像生成におけるカスタマイズをノーコードで行うことができる。本ツールの使用にあたって、一般的な QR コードをイラスト改変するワークフローが公開されていた web サイト [15] [18] を参考にした。comfyUI の操作画面が図 5.11 である。

³@ps010 (cyumizou) in Supership 株式会社—図で見てわかる！画像生成 AI 「Stable Diffusion」の仕組み <https://qiita.com/ps010/items/ea4e8ddef4de62d1ab1> より引用

⁴<https://www.comfy.org/>

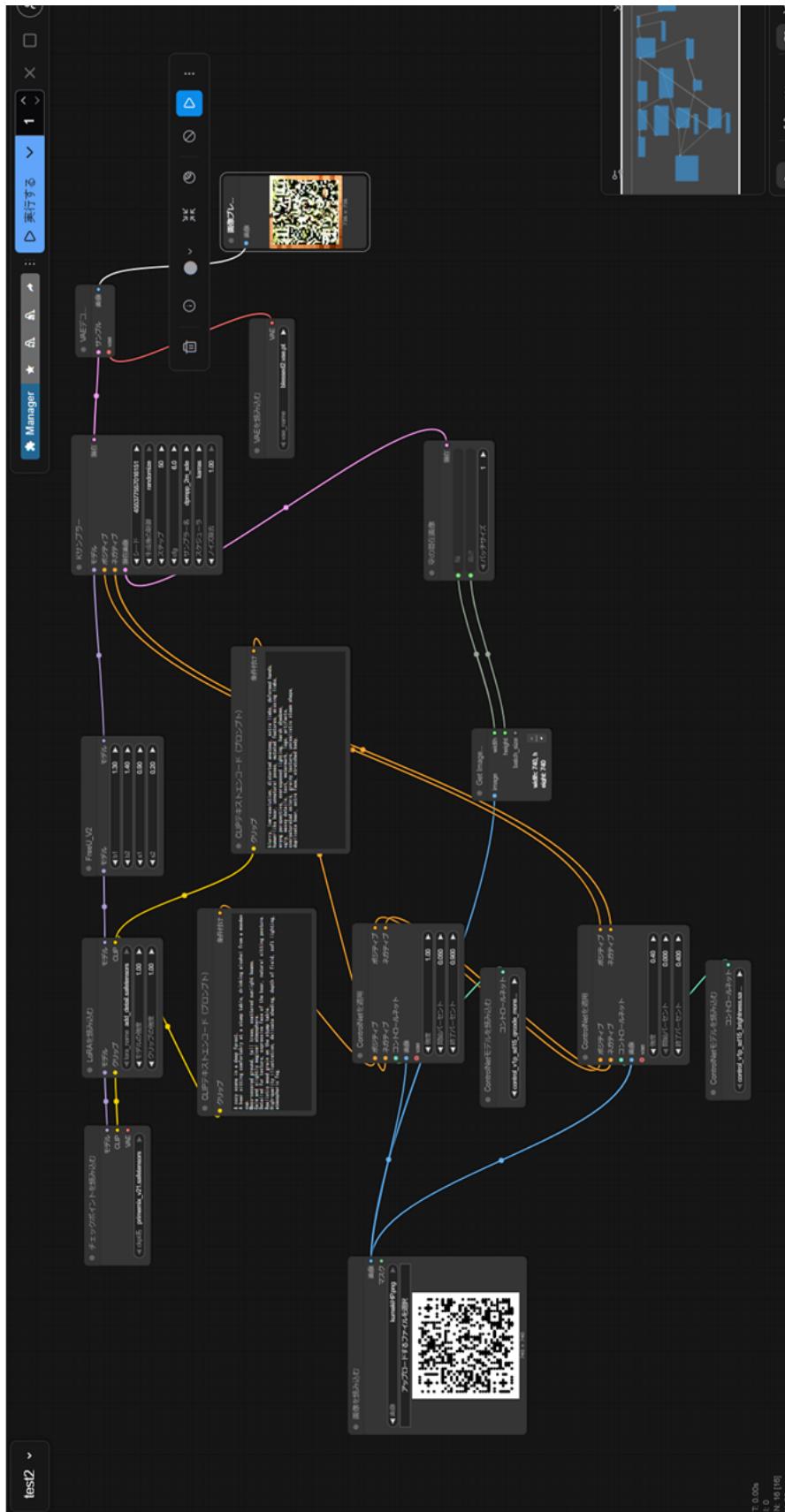


図 5.11: comfyUI の操作画面

ここで comfyUI で使用したノードについて述べる。

5.4.1 VAE (variational auto encoder)

VAE とは画像データの圧縮時の特徴を抽出する方法を決定しているモデルである。従って出力画像の品質に大きく影響を及ぼす。本実験では参考サイトのワークフローに習い、NoCrypt/blessed_vae を使用した。

5.4.2 チェックポイント (ckpt)

チェックポイントは出力画像の画風に影響を与える。Stable diffusion の学習過程で作成された画像の特徴を保存したファイルである。そのため、他のモデルに対して数 GB と大きな容量をもつファイルとなる。本実験では参考サイトのワークフローに習い、primemix_v21 を使用した。

5.4.3 controlnet

画像生成の構図や形状を自在にコントロールできる拡張機能となる。画像生成の過程に割り込むことで、出力画像に大きく影響を与える。QR コードを用いたイラスト改変技術に大きく関わるノードである。本実験で使用した controlnet モデルについては 5.5 節のとおりである。

5.4.4 K サンプラー

出力画像の仕上がりに影響を与える。無数にあるノイズの中から少しづつ画像を作っていく工程を担当するノードである。シード値の変更や、ステップ数の変更が可能である。

5.4.5 LoRA

Low-Rank-Adaptation の頭文字であり、少ないリソースで効率的に学習ができる手法である。画風の調整や、人物やキャラクターの指定など、多岐にわたる調整を可能とするようなモデルが複数存在する。ただし、生成過程に割り込む controlnet に比べて、出力画像に与える影響は比較的少ないとされる。

5.5 実験内容

本実験の目的は、従来、モノクロのQRコードを入力画像として作成するアートQRコードの生成に対して、入力画像にカラーQRコードを使用し、QRコードの可読性がある生成AIイラストを得ることである。従って、従来のアートQRコードのワークフローをベースとして、使用するcontrolnetモデルを追加する改変を加えた。

本実験で使用したcontrolnetモデルを以下に示す。

- monster-labs/control_v1p_sd15_qrcode_monster (以下、qrcode_monster)
- latentcat/control_v1p_sd15_brightness (以下、brightness)
- TencentARC/T2Iadapter_color_sd14v1 (以下、T2Iadapter_color)

従来のワークフローではqrcode_monsterとbrightnessの二つを使用していた。しかし、入力画像をカラーQRコードとする場合には、入力画像の色の位置を出力画像に反映する必要がある。従って、T2Iadapter_colorを新しく採用した。

生成には、各controlnetの強度を変更するパラメータとした。実験の内容が以下の2つである。

- 各controlnetを単体で使用し、強度を変えて生成を行う
- 3つのcontrolnetを併用し、強度を変えて生成を行う

1つ目の実験では、強度を、0.5, 1.0, 1.5と変更して生成を行った。

2つ目の実験では、controlnetの強度に対して、(brightness, T2Iadapter_color, qr_monster) = (0.3, 0.9, 0.3)を基準として、それぞれパラメータを変えて生成を行った。このパラメータを基準として設定した理由は2つある。1つ目は、入力画像が持つ色の情報を出力画像に強く反映させるためである。従って、T2Iadapter_colorの強度を高く設定している。2つ目は、controlnet同士の干渉を防ぐためである。controlnetの強度を極端に高い値に設定すると、モデル同士の干渉が発生してしまい、狙った生成結果が得られないケースが存在する。従って、使用するcontrolnetの強度の合計が1.5となるように基準を設定した。

5.6 実験結果

5.6.1 各controlnetによる出力画像への影響

はじめに、cotrolnetを単体で使用したときの出力結果に与える影響を比較した。ポジティブプロンプト、ネガティブプロンプトへの入力を共に空白（” ”）として生成を行った。入力画像として使用した画像が図5.12である。

latentcat/control_v1p_sd15_brightnessを使用して画像生成を行った結果が図5.13である。次に、monster-labs/control_v1p_sd15_qrcode_monsterを使用した画像生成を行った。本モデルには、新旧2つのバージョンが存在したため、それぞれのバ-

ジョンを用いた生成を行った。生成の結果が図 5.14, 及び 5.15である。さらに, TencentARC/t2iadapter_color_sd14v1 を使用した画像生成を行った。生成の結果が図 5.16である。

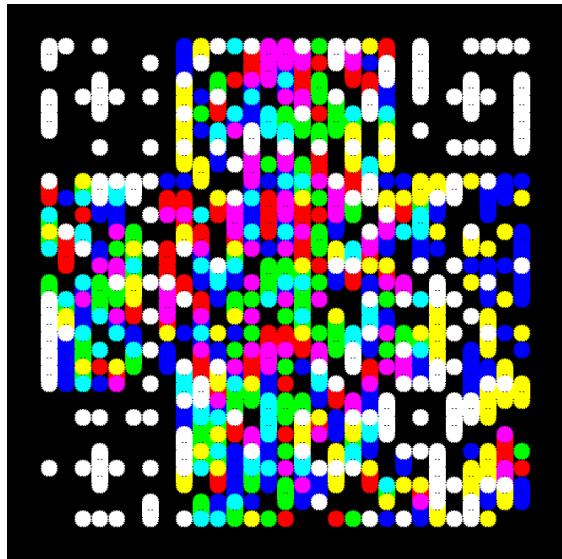


図 5.12: 画像生成に使用した入力画像

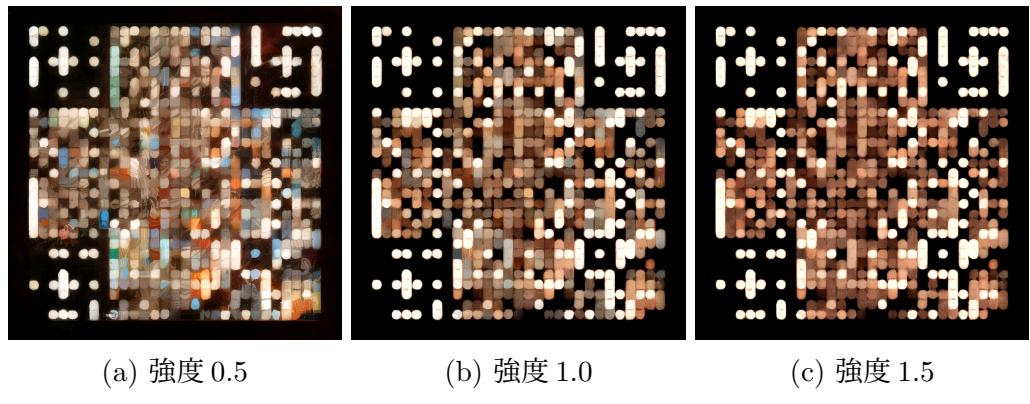


図 5.13: brightness の強度による比較



(a) 強度 0.5

(b) 強度 1.0

(c) 強度 1.5

図 5.14: qrcode_monster_v1 の強度による比較



(a) 強度 0.5

(b) 強度 1.0

(c) 強度 1.5

図 5.15: qrcode_monster_v2 の強度による比較



(a) 強度 0.5

(b) 強度 1.0

(c) 強度 1.5

図 5.16: T2Iadapter_color の強度による比較

5.6.2 3種類の controlnet を使用した画像生成

画像生成の結果が図 5.17、及び 5.17である。出力結果として、入力画像に対する色の再現度が低く、今回実験したパラメータでは可読性のあるカラーアート QR コードを生成することはできなかった。



図 5.17: 3種類の controlnet を使用した生成結果 (1)



図 5.17: 3種類の controlnet を使用した生成結果 (2)

5.7 考察

本実験では、プロンプトを空白として画像生成を行ったが、アニメ風のキャラクターのような出力画像が得られた。これは、生成に使用したチェックポイントの影響によるものと考えられる。生成に使用したチェックポイントである primemix_v21 は、人物表現に優れたモデルであり、学習に使用されたアニメイラストの影響により実験のような結果が得られたと考えられる。また、図 5.13 の結果から、brightness は入力画像の構造を色濃く出力に影響させるモデルであることが分かった。更に、TencentARC/t2iadapter_color_sd14v1 は入力画像に対する出力への影響が比較的少なく、生成画像から QR コードが取得できる色の再現は見られなかった。

3種類の controlnet を使用した実験に関して、図 5.17、及び 5.17 の (f), (j) にみられるように、brightness の値が低いと、入力画像の構造が反映されず、ファインダパターン領域が消失される事が確認できた。また、(a), (b), (c) では、比較的同様の生成結果が得られた。

第6章 結論

本章では、本研究のまとめと今後の展望について述べる

6.1 まとめ

本研究を通じて、RGB を重ね合わせる事によってエンコードされるカラー QR コードの実装と、近距離からの撮影によるデコードが可能となるシステムを構築することができた。また、カラー QR コードを入力画像としたアート QR コードの生成に関して、今回の実験で扱った画像生成手法では、開発した読み取りシステムでデコード可能な画像の生成はできなかった。

6.2 今後の展望

6.2.1 カラー QR コードの読み取りシステムについて

本実験により、中長距離からの撮影で、カラーコードが認識できないという課題が見つかった。この課題に対する展望として、物体検出 AI 等を利用した画像の補正が挙げられる。本研究で行った実験では、撮影対象に対する距離に関わらず、同様のカーネルサイズを用いて、撮影画像に対する前処理を行っていた。これにより、撮影距離が離れるにつれて、撮影画像中の QR コード領域が小さくなり、処理後の画像に劣化が見られた。そこで、物体検出アルゴリズムを用いて、先に QR コード領域の推定を行い、補正画像を作成したうえで、モルフォロジー処理を適用することでカラー QR コードの検出率向上が期待される(図 6.1)。このような処理は、カラー QR コードだけではなく、一般の QR コードの検出率を向上にも役立つものであるだろう。

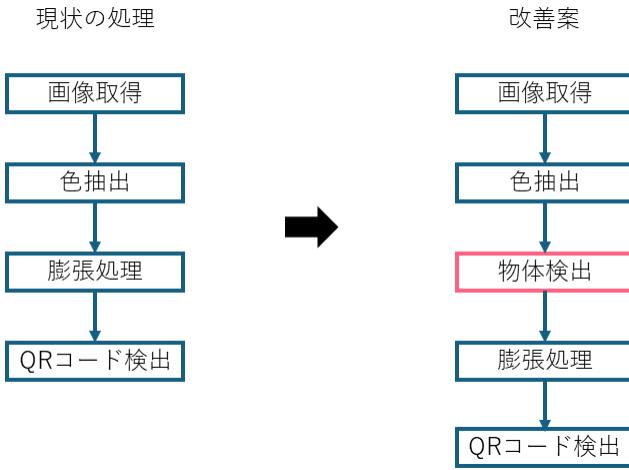


図 6.1: カラー QR コードの検出率を向上させる改善案

6.2.2 カラー QR コード入力によるアート QR コードの生成について

本研究で取り扱った手法では、カラー QR コードの読み取りシステムで検出可能なカラーアート QR コード画像を生成することができなかった。しかし、出力画像に大きく影響するという理由から controlnet に対する工夫は必要不可欠である。本技術に対する展望としては、controlnet モデルの独自開発が挙げられる。インターネット上に公開されている controlnet モデルの多くは、safetensor と呼ばれるファイル形式をとっており、内部はバイナリのファイルとなっている。従って、ダウンロードしたモデルの解析や改変は現実的ではない。そこで開発工数は多くなるが、モデルの独自開発が必要とされると考えられる。

関連図書

- [1] “総務省 令和4年版 情報通信白書.” <https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/>
- [2] “コード ドット コム — qr コード の 情 報 量 と バー ジョ ン.”
<https://www.qrcode.com/about/version.html>.
- [3] “Keyence — qr コード の し く み.” <https://www.keyence.co.jp/ss/products/autoid/codereader/>
- [4] 嶋田拓也, 下村優太郎, 桐原瑠也, 熊木武志, “フリッカパターンを用いた動画向け情報埋め込み手法の実装と評価,” 信学技報, vol. 119, no.2, pp. 117–122, 2020.
- [5] Y.Shimomura, T.Shimada, R.Kirihara, and T.Kumaki, “Live demonstration: Development of dot matrix LED using flicker noise as a base,” IEEE International Symposium on Circuits And Systems (ISCAS), May 2020.
- [6] R.Kirihara, Y.Shimomura, T.Shimada and . T.Kumaki, “Live demonstration: Development of LED-based stego-panel for new smartphone usage,” IEEE International Symposium on Circuits And Systems (ISCAS), 2020.
- [7] T.Shimada, and T.Kumaki, “Development of flicker-noise-based lighting communication method,” RISP International workshop on Nonlinear Circuit, 2020.
- [8] T.Shimada, and T.Kumaki, “A study of led-based spy-photo prevention system using flicker noise for actual environment,” International Technical Conference on Circuits/Systems, 2018.
- [9] 下村優太郎, 嶋田 拓也, 熊木武志, “フリッカノイズによるドットマトリクス LED への実装,” LSI とシステムのワークショップ, no. no.4.
- [10] , “2 次元カラーコード実現を目指した画像中の色認識,” vol. 三重大学修士論文, 2011.
- [11] H. Blasinski, O. Bulan, and G. Sharma, “Per-colorant-channel color barcodes for mobile applications: An interference cancellation framework,” IEEE Transactions on Image Processing, vol. 22, no. 4, pp. 1498–1511, 2013.

付録

- [12] “RGB と CMYKってなに？意味や変換方法など基本を徹底解説！”
<https://www.profuture.co.jp/mk/column/what-is-rgb>.
- [13] “色の三属性「色相」「明度」「彩度」とは？【HSB/HSV】.”
<https://321web.link/color-attribute/>.
- [14] “Fortune business insights.” <https://www.fortunebusinessinsights.com/jp/>
- [15] “イラストにしかみえない qr コードを作つてみる.”
https://note.com/st_ai/n/ne8908b9f8901.
- [16] “Rakuten mobile — 画像生成 ai の仕組みとは？初心者向けに解説.”
<https://business.mobile.rakuten.co.jp/column/2025/0701-01/>.
- [17] “@ps010(cyumizou) in supership 株式会社 — 図で見てわかる！画像生成 ai 「stable diffusion」の仕組み.” <https://qiita.com/ps010/items/ea4e8ddef4de62d1ab1>.
- [18] “Qr code — comfy ui workflow — openart.”
https://openart.ai/workflows/jackal_thirsty_43/qr-code/uy1ivgb1x0v7Cc2TN3lP.

付録

カラーQRコードにエンコードするプログラム

tricolor_make_mixedQR.py

```
1 import cv2
2 import numpy as np
3 from tkinter import Tk, Label, Button, filedialog
4 from tkinterdnd2 import TkinterDnD, DND_FILES
5 import qrcode
6
7
8 # コードを作成QR
9 qr = qrcode.QRCode(
10     version=1,    # サイズのバージョン
11     error_correction=qrcode.constants.ERROR_CORRECT_H,    # エラー訂正レベル
12     box_size=15,   # 各ボックスのサイズ
13     border=10,    # ボーダーのサイズ
14 )
15 # qr.add_data(data)
16 qr.make(fit=True)
17
18 # 画像を生成
19 img = qr.make_image(fill='black', back_color='white')
20
21 # 画像を保存
22 img.save("chinese2.png")
23
24 class ImageProcessor:
25     def __init__(self):
26         self.images = []
27         self.labels = ["Red", "Blue", "Green"]
28         self.current_label_index = 0
29
30     def load_image(self, file_path):
31         image = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
32         if image is None:
33             print(f"Failed to load image: {file_path}")
34             return None
35         return image
36
37     def process_image(self, image, color):
38         processed_image = np.zeros((image.shape[0], image.shape[1], 3), dtype=np.uint8)
39         processed_image[image == 0] = color # Set black pixels to the specified color
40         processed_image[image == 255] = [0, 0, 0] # Set white pixels to black
41         return processed_image
42
43     def add_image(self, file_path):
44         color_map = {
45             0: ([0, 0, 255], "qr_red_loaded.png"),
46             1: ([255, 0, 0], "qr_blue_loaded.png"),
47             2: ([0, 255, 0], "qr_green_loaded.png")
48         }
49
50         image = self.load_image(file_path)
51         if image is not None:
52             color, filename = color_map[self.current_label_index]
53             processed_image = self.process_image(image, color)
54             cv2.imwrite(filename, processed_image)
55             self.images.append(processed_image)
56             self.current_label_index += 1
57
58         if len(self.images) == 3:
59             self.combine_images()
60
61     def combine_images(self):
62         if len(self.images) == 3:
63             mixed_image = cv2.add(cv2.add(self.images[0], self.images[1]), self.images[2])
64             resized_image = cv2.resize(mixed_image, (288, 288))
65             #image_mixed = "./Desktop/tricolor_panel/image1/mixedQR1.png"
```

付録

```
66         cv2.imwrite("image_mixed.png", resized_image)
67         print("Processed - and - saved - the - mixed - image - as - image_mixed.png")
68     else:
69         print("Error : - Not - enough - images - to - combine .")
70
71 def select_file(image_processor):
72     file_path = filedialog.askopenfilename()
73     #file_path = "./Desktop/tricolor_panel/"
74     if file_path:
75         image_processor.add_image(file_path)
76
77 def main():
78     image_processor = ImageProcessor()
79
80     root = Tk()
81     root.title("Image - Processor")
82
83     label_text = f"Please - load - the - {image_processor.labels[image_processor.current_label_index]} - image"
84     label = Label(root, text=label_text)
85     label.pack(padx=10, pady=10)
86
87     def update_label():
88         nonlocal label_text
89         if image_processor.current_label_index < len(image_processor.labels):
90             label_text = f"Please - load - the - {image_processor.labels[image_processor.current_label_index]} - image"
91             label.config(text=label_text)
92         else:
93             label.config(text="Processing - complete .")
94
95     def load_next_image():
96         select_file(image_processor)
97         update_label()
98
99     button = Button(root, text="Load - Image", command=load_next_image)
100    button.pack(padx=10, pady=10)
101
102    root.mainloop()
103
104 if __name__ == "__main__":
105     main()
```

カラー QR コードの読み取りプログラム

app.py

```
1 from flask import Flask, request, jsonify, render_template, url_for, redirect
2 import sqlite3
3 import os
4 import cv2
5 from pyzbar.pyzbar import decode
6 from datetime import datetime
7 from PIL import Image
8 import base64
9 import io
10 import numpy as np
11
12 app = Flask(__name__)
13 app.config['UPLOAD_FOLDER'] = 'static/images'
14 DB_PATH = 'db.sqlite3'
15
16 # データベースの初期化SQLite
17 def init_db():
18     conn = sqlite3.connect(DB_PATH)
19     c = conn.cursor()
20     c.execute('''
21         CREATE TABLE IF NOT EXISTS images (
22             id INTEGER PRIMARY KEY,
23             filename TEXT NOT NULL,
24             created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
25         )
26     ''')
27     conn.commit()
28     conn.close()
29
30 # # 画像を保存する関数
31 # def save_image(image_data):
32 #     # で送ってきた画像データをデコードして保存Base64
```

付録

```
33 #     image_data = image_data.split(",") [1]
34 #     image = Image.open(io.BytesIO(base64.b64decode(image_data)))
35 #     filename = f"{datetime.now().strftime('%Y%m%d%H%M%S')}.png"
36 #     image_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
37 #     image.save(image_path)
38
39 #     # データベースに保存
40 #     conn = sqlite3.connect(DB_PATH)
41 #     c = conn.cursor()
42 #     c.execute('INSERT INTO images (filename) VALUES (?)', (filename,))
43 #     conn.commit()
44 #     conn.close()
45
46 #     return image_path
47
48 def save_image_formdata(image):
49     # リクエストの中にPOST 'image' ファイルが含まれているかを確認
50     if 'image' not in request.files:
51         return jsonify({"status": "error", "message": "画像ファイルがありません"}), 400
52
53     image_file = request.files['image']
54
55     # 撮影日時を取得してファイル名を設定
56     timestamp = datetime.now().strftime('%Y%m%d%H%M%S') # 例: "20241021213045"
57     filename = f"{timestamp}.png"
58     file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
59
60     # 画像ファイルを保存
61     image_file.save(file_path)
62
63     return file_path
64
65
66 # 色抽出と画像の保存
67 def extract_color_and_save_HSV(image_path, color_index):
68     image = cv2.imread(image_path)
69     color = hsv_extract_to_mono(image, color_index)
70     color_filename = f"{datetime.now().strftime('%Y%m%d%H%M%S')}-[red,green,blue][{color_index}].png"
71     color_path = os.path.join(app.config['UPLOAD_FOLDER'], color_filename)
72     cv2.imwrite(color_path, color)
73
74     return color_path
75
76 # 色の抽出を=====空間で行う
77 HSV=====
77 def hsv_extract_to_mono(image, i):
78
79     # jsonify({'status': 'error', 'message': 'まで実行'125'})
80
81     # からに変換BGRHSV
82     hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
83
84     # 赤要素の抽出
85     if i == 0:
86         # 白色範囲
87         lower_white = np.array([0, 0, 100])
88         upper_white = np.array([180, 60, 255])
89
90         # 赤色範囲 (～の範囲で設定) 0180
91         lower_red1 = np.array([0, 80, 100])
92         upper_red1 = np.array([15, 255, 255])
93         lower_red2 = np.array([165, 80, 100])
94         upper_red2 = np.array([180, 255, 255])
95
96         # 黄色範囲
97         lower_yellow = np.array([15, 80, 100])
98         upper_yellow = np.array([45, 255, 255])
99
100    # マゼンタ範囲
101    lower_magenta = np.array([135, 80, 100])
102    upper_magenta = np.array([165, 255, 255])
103
104    # 各色のマスクを作成
105    mask_white = cv2.inRange(hsv_image, lower_white, upper_white)
106    mask_red1 = cv2.inRange(hsv_image, lower_red1, upper_red1)
107    mask_red2 = cv2.inRange(hsv_image, lower_red2, upper_red2)
108    mask_yellow = cv2.inRange(hsv_image, lower_yellow, upper_yellow)
109    mask_magenta = cv2.inRange(hsv_image, lower_magenta, upper_magenta)
110
111    # 赤色のマスクを結合
112    mask_red = cv2.bitwise_or(mask_red1, mask_red2)
113
114    # 白、赤、黄、マゼンタのマスクを結合して黒にする領域を指定
115    mask_to_black = cv2.bitwise_or(mask_white, mask_red)
116    mask_to_black = cv2.bitwise_or(mask_to_black, mask_yellow)
117    mask_to_black = cv2.bitwise_or(mask_to_black, mask_magenta)
118
119    # すべての領域を白に塗りつぶし
120    result = np.ones_like(image) * 255
121
```

付録

```
122     # マスクで指定された領域を黒に変換
123     result [mask_to_black > 0] = [0, 0, 0]
124
125     return result
126
127
128     # 緑要素の抽出
129     elif i == 1:
130         # 白色範囲
131         lower_white = np.array([0, 0, 100])
132         upper_white = np.array([180, 60, 255])
133
134         # 緑色範囲 (~の範囲で設定) 018060
135         lower_green = np.array([45, 80, 100])
136         upper_green = np.array([75, 255, 255])
137
138         # 黄色範囲
139         lower_yellow = np.array([15, 80, 100])
140         upper_yellow = np.array([45, 255, 255])
141
142         # シアン範囲
143         lower_magenta = np.array([85, 80, 100])
144         upper_magenta = np.array([105, 255, 255])
145
146         # 各色のマスクを作成
147         mask_white = cv2.inRange(hsv_image, lower_white, upper_white)
148         mask_green = cv2.inRange(hsv_image, lower_green, upper_green)
149         mask_yellow = cv2.inRange(hsv_image, lower_yellow, upper_yellow)
150         mask_magenta = cv2.inRange(hsv_image, lower_magenta, upper_magenta)
151
152         # 緑色のつのマスクを結合2
153         # mask_red = cv2.bitwise_or(mask_red1, mask_red2)
154
155         # マスクを結合して黒にする領域を指定
156         mask_to_black = cv2.bitwise_or(mask_white, mask_green)
157         mask_to_black = cv2.bitwise_or(mask_to_black, mask_yellow)
158         mask_to_black = cv2.bitwise_or(mask_to_black, mask_magenta)
159
160         # すべての領域を白に塗りつぶし
161         result = np.ones_like(image) * 255
162
163         # マスクで指定された領域を黒に変換
164         result [mask_to_black > 0] = [0, 0, 0]
165
166         return result
167
168
169     # 青要素の抽出
170     elif i == 2:
171         # 白色範囲
172         lower_white = np.array([0, 0, 100])
173         upper_white = np.array([180, 60, 255])
174
175         # 青色範囲 (~の範囲で設定) 0180120
176         lower_blue = np.array([105, 80, 100])
177         upper_blue = np.array([135, 255, 255])
178
179         # シアン色範囲
180         lower_yellow = np.array([75, 80, 100])
181         upper_yellow = np.array([105, 255, 255])
182
183         # マゼンタ範囲300
184         lower_magenta = np.array([135, 80, 100])
185         upper_magenta = np.array([165, 255, 255])
186
187         # 各色のマスクを作成
188         mask_white = cv2.inRange(hsv_image, lower_white, upper_white)
189         mask_blue = cv2.inRange(hsv_image, lower_blue, upper_blue)
190         mask_yellow = cv2.inRange(hsv_image, lower_yellow, upper_yellow)
191         mask_magenta = cv2.inRange(hsv_image, lower_magenta, upper_magenta)
192
193         # 青色のつのマスクを結合2
194         # mask_red = cv2.bitwise_or(mask_red1, mask_red2)
195
196         # マスクを結合して黒にする領域を指定
197         mask_to_black = cv2.bitwise_or(mask_white, mask_blue)
198         mask_to_black = cv2.bitwise_or(mask_to_black, mask_yellow)
199         mask_to_black = cv2.bitwise_or(mask_to_black, mask_magenta)
200
201         # すべての領域を白に塗りつぶし
202         result = np.ones_like(image) * 255
203
204         # マスクで指定された領域を黒に変換
205         result [mask_to_black > 0] = [0, 0, 0]
206
207         return result
208
209
210     # =====
211
212     # (膨張) 处理の後に (収縮) 处理を行う関数 DilationErosion
```

付録

```
213 | def apply_dilation_then_erosion(image_path):
214 |     image = cv2.imread(image_path, 0) # 画像をグレースケールで読み込み
215 |     kernel = np.ones((5, 5), np.uint8) # カーネルサイズは調整可能です
216 |
217 |     # 処理 Dilation
218 |     dilated_image = cv2.dilate(image, kernel, iterations=1)
219 |
220 |     # 処理 Erosion
221 |     processed_image = cv2.erode(dilated_image, kernel, iterations=1)
222 |
223 |     # 処理後の画像を保存
224 |     processed_filename = f'{datetime.now().strftime("%Y%mf%d%H%M%S")}-processed.png'
225 |     processed_path = os.path.join(app.config['UPLOAD_FOLDER'], processed_filename)
226 |     cv2.imwrite(processed_path, processed_image)
227 |
228 |     return processed_path
229 |
230 | # コードを検出する関数QR
231 | def detect_qr_code(image_path):
232 |     image = cv2.imread(image_path)
233 |     decoded_objects = decode(image)
234 |     qr_codes = [obj.data.decode('utf-8') for obj in decoded_objects]
235 |     return qr_codes
236 |
237 | @app.route('/')
238 | def index():
239 |     return render_template('index_phone1106.html')
240 |
241 |
242 | @app.route('/capture', methods=['POST'])
243 | def capture():
244 |     if 'image' not in request.files:
245 |         return jsonify({"status": "error", "message": "画像ファイルが見つかりません"}), 400
246 |
247 |     color = request.form.get('color')
248 |     image = request.files['image']
249 |     # color_choice = data.get('color') # 'red', 'green', or 'blue'
250 |     color_index = {'red': 0, 'green': 1, 'blue': 2}[color]
251 |
252 |     image_path = save_image_formdata(image)
253 |     # mono_image_path = hsv_extract_to_mono(image_path, color_index)
254 |     mono_image_path = extract_color_and_save_HSV(image_path, color_index)
255 |
256 |     # と処理を実行 DilationErosion
257 |     processed_image_path = apply_dilation_then_erosion(mono_image_path)
258 |
259 |     qr_codes = detect_qr_code(processed_image_path)
260 |
261 |
262 |     if qr_codes:
263 |         return jsonify({'status': 'success', 'qr_code': qr_codes[0]})
264 |     else:
265 |         return jsonify({'status': 'error', 'message': 'コードが検出されませんでした。QR'})
266 |
267 |
268 | if __name__ == '__main__':
269 |     # 証明書と秘密鍵のパスを指定
270 |     context = ('server.crt', 'server.key')
271 |     # でサーバーを起動 HTTPS
272 |     app.run(host='0.0.0.0', port=5000, ssl_context=context)
```


謝辞

本論文の作成にあたり、貴重な助言、ご指導をして頂いた立命館大学理工学部電子情報工学科 熊木 武志教授に深く感謝の意を表します。また、本研究に関わりご助言をして頂いた石田勝之介氏に深く感謝致します。最後に、日頃から様々な事においてお世話になりました7期生を始めとするマルチメディア集積回路システム研究室の皆様に最大の感謝をお贈り致します。

2026年3月 池田 新

研究業績リスト

【国内研究会等発表】

- 池田新, 熊木武志, ”RGB カラー QR コードの読み取り実験とその評価について,”
ソサイエティ大会 2024, Sep., 2024.

【国外研究会等発表】

- Arata Ikeda, Takumi Hayashi, and Takeshi Kumaki, ”Development of secure QR code by using invisible information display lighting device,” ITC-CSCC 2024, Jul., 2024
- Arata Ikeda, Tomoki Yamashita, and Takeshi Kumaki, ”Development of RGB micro QR code system using invisible information lighting device 'Stego-panel V',” NOLTA 2025, Oct., 2025

【その他研究活動】

- EdgeTech+ West 2024, Jul., 2024
- EdgeTech+ 2024, Nov., 2024
- EdgeTech+ West 2025, Jul., 2025
- EdgeTech+ 2025, Nov., 2025

【受賞】