修士論文

カラーQRコードの実装と画像生成AIを用いた検出可能イラスト生成手法の検討

(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，従来使用され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し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で，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い，さらには，近年，技術向上が進んでいる画像生成 AI を使用した QR コード検出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．本研究の結果として，カラー QR コードは，従来の QR コードには検出性能で劣るといったデメリットがあるが，従来の QR コードではできなかった用途での利用が可能であることが確認できた．

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 4

第 2 章　カラー QR コードに関する先行研究 **5**
2.1　ステゴパネル 5
2.2　Henryk Blasinski の先行研究 5
2.3　三重大の先行研究 5

第 3 章　カラー QR コード検出システムの開発 **7**
3.1　エンコード方法 7
3.2　色空間について 9
3.3　デコード方法 10
3.4　読み取りアプリの開発 14

第 4 章　検出性能の比較実験 **17**
4.1　実験手法 17
4.2　実験結果 21
4.3　考察 22

第 5 章　カラー QR コードを用いた AI イラストの生成 **25**
5.1　イラストに見える QR コード 25
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 27
5.3　イラスト生成 AI の原理 28
5.4　使用したソフトとモデル 28
5.5　実験内容 29
5.6　実験結果 29
5.7　考察 29

第 6 章　結論 **31**
6.1　まとめ 31
6.2　今後の展望 31

参考文献 **31**

謝辞 **41**

発表論文リスト **43**

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 ファインダパターンの原理 4

2.1 Blasinski によるカラー QR コード 5
2.2 三重大学のカラー QR コードに関する先行研究 5

3.1 加法混色 8
3.2 カラー QR コードのエンコード方法 8
3.3 RGB 色空間での色の表現例 9
3.4 HSV 色空間での色の表現方法 10
3.5 カラー QR コードのデコード方法 11
3.6 デコード時のフローチャート 12
3.7 膨張処理のイメージ 13
3.8 収縮処理のイメージ 13
3.9 読み取りアプリの web 画面表示 15
3.10 QR コード不検出時の web 画面表示 16

4.1 実験に使用したカラー QR コード画像 17
4.2 実験時の画像の撮影環境 18
4.3 20cm から撮影したときの画像 18
4.4 40cm から撮影したときの画像 19
4.5 60cm から撮影したときの画像 19
4.6 80cm から撮影したときの画像 20
4.7 比較対象とするモノクロ QR コードの画像 20
4.8 膨張処理前の画像 22
4.9 膨張処理後の画像 23

5.1 生成 AI の市場規模の予測 25
5.2 アート QR コードの例 26
5.3 アート QR コードの例 27
5.4 comfyUI の操作画面 28

表 目 次

4.1 色抽出処理のみを行った際の検出率の比較 21
4.2 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 21
4.3 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 21
4.4 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 21
4.5 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 22

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と考えられる．それは，技術的背景，権利的背景，社会的背景の3つである．

初めに，技術的背景について，これはQRコードが持つ技術的な優位性から由来する．従来使用され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使用する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，これはQRコードが特許フリーなため誰でも自由に使用可能なことに由来する．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，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，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．日本において，その保有率は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]

[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

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 コードの普及を後押ししただろう．現に SNS アカウントの共有や，市中の広告でも QR コードが使用されている．

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，省スペースに印字をすることができる．QR コードの構造について，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，データセル，切り出しシンボル，タイミングパター

ン，アライメントパターン，フォーマット情報が存在している (図 1.2).[2]

図 1.2: QR コードの構造

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．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。次に、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**??**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およ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。

[2]QR 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

図 1.3: QR コードのバージョン

図 1.4: ファインダパターンの原理

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，面積当たりの情報量を増やすこと．二つ目は．広告媒体等で他のデザインを既存しないような二次元コードを作成することである．

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

2.2　Henryk Blasinskiの先行研究

x

図 2.1: Blasinski によるカラー QR コード

2.3　三重大の先行研究

図 2.2: 三重大学のカラー QR コードに関する先行研究

第3章　カラーQRコード検出システムの開発

本章では，カラー QR コードの読み取りシステムの開発について述べる．はじめに，3.1 節，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概要について述べる．

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 コード生成の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) となる．

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 RB 色区間が挙げられる．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色空間は人間の色近くに近い直感的な表現を目的として提案された．ＳＶ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．これにより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 色空間での色の表現例

図 3.4: HSV 色空間での色の表現方法[1]

3.3 デコード方法

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す．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．

[1]321web—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ような処理のことである．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 3.7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．

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)．

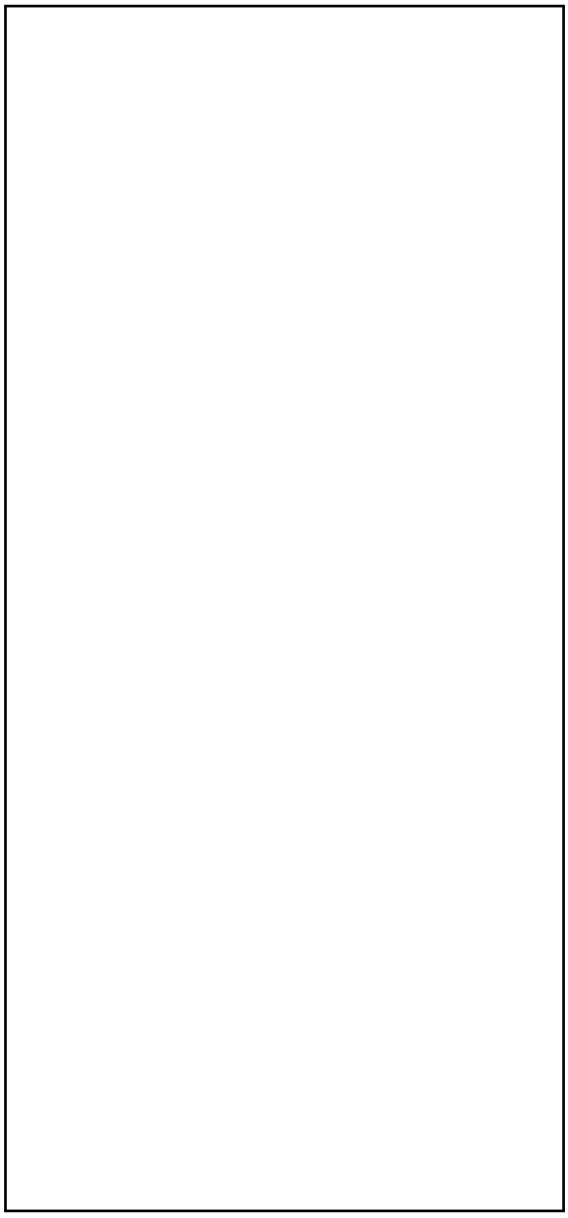

図 3.9: 読み取りアプリの web 画面表示

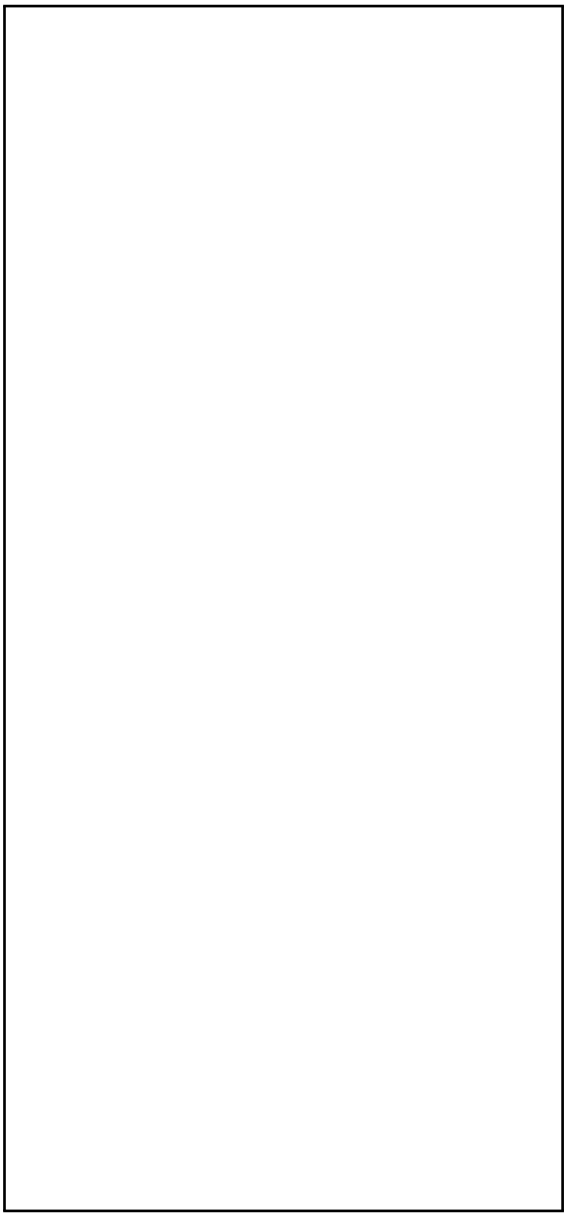

図 3.10: QR コード不検出時の web 画面表示

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 コードの検出率の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．

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

図 4.1: 実験に使用したカラー QR コード画像

図 4.2: 実験時の画像の撮影環境

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．また，対照実験として，一般の QR コードの撮影も行った.

図 4.3: 20cm から撮影したときの画像

図 4.4: 40cm から撮影したときの画像

図 4.5: 60cm から撮影したときの画像

図 4.6: 80cm 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 4.1である．

表 4.1: 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 4.2, 表 4.3, 表 4.4, 表 4.5である．

表 4.2: 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 4.3: 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 4.4: 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 4.5: 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%

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下図のとおりである (図 4.8), (図 4.9).

図 4.8: 膨張処理前の画像

図 4.9: 膨張処理後の画像

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，その技術の根幹を支えるイラスト生成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の市場規模の予測

[0]Fortune 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

もちろん，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，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だろう．このようなイラスト生成 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 AR 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や，図 5.3である．

図 5.2: アート QR コードの例

図 5.3: アート QR コードの例

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 コードの生成は，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で得られる利点について述べる．

カラー QR コードに対応したアート QR コード生成について 2 種類の例が挙げられる．それは以下の 2 種類の方法である．

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.

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.

A の方法による利点として，先述のとおり，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生成

5.3　イラスト生成 AI の原理

5.4　使用したソフトとモデル

カラー QR コードを使用 mfyUI したイラスト生成に伴い，comyUI 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 サイトを参考にした．comfyUI の操作画面が図 5.4である．

図 5.4: 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．

・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．

・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．

・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．

・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ず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できる．

・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和比較的少ないと考えられる．

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 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

・latentcat/control_v1p_sd15_brightness

・TencentARC/t2iadapter_color_sd14v1

使用した各 controlnet モデルについて詳しく述べる．

monster-labs/control_v1p_sd15_qrcode_monster(以下，qrcode_monster) は

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，T2Idapter_color を新しく採用した．

5.6 実験結果

5.7 考察

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

6.2　今後の展望

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/

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

```
import cv2
import numpy as np
from tkinter import Tk, Label, Button, filedialog
from tkinterdnd2 import TkinterDnD, DND_FILES
import qrcode


# コードを作成QR
qr = qrcode.QRCode(
    version=1,  # サイズのバージョン
    error_correction=qrcode.constants.ERROR_CORRECT_H,  # エラー訂正レベル
    box_size=15,  # 各ボックスのサイズ
    border=10,  # ボーダーのサイズ
)
# qr.add_data(data)
qr.make(fit=True)

# 画像を生成
img = qr.make_image(fill='black', back_color='white')

# 画像を保存
img.save("chinese2.png")

class ImageProcessor:
    def __init__(self):
        self.images = []
        self.labels = ["Red", "Blue", "Green"]
        self.current_label_index = 0

    def load_image(self, file_path):
        image = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
        if image is None:
            print(f"Failed to load image: {file_path}")
            return None
        return image

    def process_image(self, image, color):
        processed_image = np.zeros((image.shape[0], image.shape[1], 3), dtype=np.uint8)
        processed_image[image == 0] = color  # Set black pixels to the specified color
        processed_image[image == 255] = [0, 0, 0]  # Set white pixels to black
        return processed_image

    def add_image(self, file_path):
        color_map = {
            0: ([0, 0, 255], "qr_red_loaded.png"),
            1: ([255, 0, 0], "qr_blue_loaded.png"),
            2: ([0, 255, 0], "qr_green_loaded.png")
        }

        image = self.load_image(file_path)
        if image is not None:
            color, filename = color_map[self.current_label_index]
            processed_image = self.process_image(image, color)
            cv2.imwrite(filename, processed_image)
            self.images.append(processed_image)
            self.current_label_index += 1

        if len(self.images) == 3:
            self.combine_images()

    def combine_images(self):
        if len(self.images) == 3:
            mixed_image = cv2.add(cv2.add(self.images[0], self.images[1]), self.images[2])
            resized_image = cv2.resize(mixed_image, (288, 288))
            #image_mixed = "./Desktop/tricolor_panel/image1/mixedQR1.png"
```

```
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  #     # で送られてきた画像データをデコードして保存Base64
```

```
#       image_data = image_data.split(",")[1]
#       image = Image.open(io.BytesIO(base64.b64decode(image_data)))
#       filename = f"{datetime.now().strftime('%Y%m%d%H%M%S')}.png"
#       image_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
#       image.save(image_path)

#       # データベースに保存
#       conn = sqlite3.connect(DB_PATH)
#       c = conn.cursor()
#       c.execute('INSERT INTO images (filename) VALUES (?)', (filename,))
#       conn.commit()
#       conn.close()

#       return image_path

def save_image_formdata(image):
    # 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
    if 'image' not in request.files:
        return jsonify({"status": "error", "message": "画像ファイルがありません"}), 400

    image_file = request.files['image']

    # 撮影日時を取得してファイル名を設定
    timestamp = datetime.now().strftime('%Y%m%d%H%M%S')  # 例: "20241021213045"
    filename = f"{timestamp}.png"
    file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)

    # 画像ファイルを保存
    image_file.save(file_path)

    return file_path


# 色抽出と画像の保存
def extract_color_and_save_HSV(image_path, color_index):
    image = cv2.imread(image_path)
    color = hsv_extract_to_mono(image, color_index)
    color_filename = f"{datetime.now().strftime('%Y%m%d%H%M%S')}_{['red','green','blue'][
        color_index]}.png"
    color_path = os.path.join(app.config['UPLOAD_FOLDER'], color_filename)
    cv2.imwrite(color_path, color)

    return color_path

# 色の抽出を=====================================空間で行う
    HSV=================================================
def hsv_extract_to_mono(image, i):

    # jsonify({'status': 'error', 'message': まで実行'125'})

    # からに変換BGRHSV
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # 赤要素の抽出
    if i == 0:
                # 白色範囲
        lower_white = np.array([0, 0, 100])
        upper_white = np.array([180, 60, 255])

        # 赤色範囲（〜の範囲で設定）0180
        lower_red1 = np.array([0, 80, 100])
        upper_red1 = np.array([15, 255, 255])
        lower_red2 = np.array([165, 80, 100])
        upper_red2 = np.array([180, 255, 255])

        # 黄色範囲
        lower_yellow = np.array([15, 80, 100])
        upper_yellow = np.array([45, 255, 255])

        # マゼンタ範囲
        lower_magenta = np.array([135, 80, 100])
        upper_magenta = np.array([165, 255, 255])

        # 各色のマスクを作成
        mask_white = cv2.inRange(hsv_image, lower_white, upper_white)
        mask_red1 = cv2.inRange(hsv_image, lower_red1, upper_red1)
        mask_red2 = cv2.inRange(hsv_image, lower_red2, upper_red2)
        mask_yellow = cv2.inRange(hsv_image, lower_yellow, upper_yellow)
        mask_magenta = cv2.inRange(hsv_image, lower_magenta, upper_magenta)

        # 赤色のつのマスクを結合2
        mask_red = cv2.bitwise_or(mask_red1, mask_red2)

        # 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
        mask_to_black = cv2.bitwise_or(mask_white, mask_red)
        mask_to_black = cv2.bitwise_or(mask_to_black, mask_yellow)
        mask_to_black = cv2.bitwise_or(mask_to_black, mask_magenta)

        # すべての領域を白に塗りつぶし
        result = np.ones_like(image) * 255

```

```
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）処理の後に（収縮）処理を行う関数DilationErosion
```

```
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%m%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)
```

トリケラパネル制御プログラム

tricolor1127.c

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立命博士氏，琵琶太郎氏, 草津悟志氏に深く感謝致します. そして, 実験を行うにあたってご協力をして頂いた衣笠智樹氏, 茨木慎太郎氏に感謝致します. 最後に, 日頃から様々な事においてお世話になりましたX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.

2023年3月　立命　太郎

研究業績リスト

【国内研究会等発表】

- 立命太郎, 逢坂京太郎, 安藤義男，竹　信孝, 熊木武志, ”2nm プロセスルールの SoC の製造技術可能性, ” ET&IoT West 2021, Jul., 2021.

【国外研究会等発表】

- Taro Ritsumei and Takeshi Kumaki, ”Possibility of 2nm process rule SoC manufacturing technology,” International Computers and communications (ICC), 2021.

【その他研究活動】

- ET&IoT West 2021, Jul., 2021.
- ET&IoT 2021, Nov., 2021.

【受賞】

- Yuta Moritake, Yutaro Shimomura, Ryuya Kirihara, Yuki Hirota, Xiangbo Kong and Takeshi Kumaki, ”Development of invisible information lighting display ”Stegopanel IV”,” IEEE Grobal Conference on Consumer Electronics (GCCE), Excellent Demo! award, Gold prize, Oct., 2021.