コンピュータ理工学特別研究報告書

題目

映像からのルービックキューブ認識とソルバープログラムの開発

学生証番号　1545105

氏名　西岡 直哉

提出日　平成 31 年 1 月 22 日

指導教員　蚊野 浩

京都産業大学
コンピュータ理工学部

要約

ルービックキューブは，1974 年にハンガリーの建築家であったエルノー・ルービックが考案したパズルゲームで，小立方体が 3×3×3 で配置された立方体状の物体である．各面６色に塗り分けられており，各面を回転させることで面の色をバラバラにして，その状態から，再び各面を回転させることで色を揃えるパズルゲームである．このパズルの色の配置は全パターンで 4325 京 2003 兆 2744 億 8985 万 6000 通りもあり，このパズルを揃えることはかなり難しい問題である．

本研究の目的は，人間が手にしたルービックキューブを揃えるための助けとなるようなシステムを開発することである．そのために，色がバラバラになったルービックキューブをカメラで撮影し，その色の配置を，パソコン内に構築したルービックキューブの 3D モデルに反映させる．そして，ルービックキューブを解く手順を人間が学んでいけるシステムを開発する．

本研究で開発したシステムは，以下の通りである．

① ルービックキューブの 3D モデルを開発した．
② 現物のルービックキューブを撮影し，その色配置を認識するプログラムを開発した．
③ 認識した色配置を，ルービックキューブの 3D モデルに反映させる機能を開発した．
④ 人間が手順を理解できる，ルービックキューブの解法プログラムを開発した．

研究の結果，実際のルービックキューブを撮影し，その状態の把握をして，ルービックキューブの 3D モデルに反映できた．それを人間が覚えやすい手順で自動解法することも可能になった．

目次

1章 序論

ルービックキューブはハンガリーの建築家ルビク・エルネー氏が 1974 年に考案した立体パズルで，小立方体が 3×3×3 に配置された立方体状の物体である．６色に塗り分けた各面を回転させることで面の色をバラバラにし，その状態から，面を回転させることで再び揃えて遊ぶものである．簡単そうに見えるが，難度が高いパズルである．これを自力で解けることは頭の良さの一端を示すとも考えられている．

ルービックキューブの面の状態を元とする集合を考え，回転操作をその集合上の演算とみなすと，その集合と演算の組みは群をなす．集合と演算が群であるとは，① 結合法則，② 単位元の存在，③ 逆元の存在，の３条件を満たすことである．したがって，ルービックキューブはどのように回転させても，元に戻すことができる．このような数学的な面白さと解くことの難しさからルービックキューブはよく研究されてきた（[1]など）．インターネット上の記事によると，全パターンを調べ上げることで，どのような状態からでも 20 手以内で全ての面を揃えることができることが証明されている[2]．

ルービックキューブを解く方法はいくつかあり，それらをプログラムとして実装したアプリが多く開発されている（ただし，それらの解は最適解ではなく，一つの手順にすぎない）．また，ルービックキューブを自力で解くことが難しいので，拡張現実感などを使って解決を支援するための研究が行われている[3][4][5]．これらの研究は，ディスプレイや Head Mount Display に表示したルービックキューブ画像に，手がかりとなる操作を重畳表示するようなものである．

AI 技術の急速な進歩により，日常生活でコンピュータが人間の判断や行動を支援することが普通になる社会が現実になりつつある．本研究では，ルービックキューブが解けないで悩んでいる私を，コンピュータが助けるという状況を素材として，色が揃っていないルービックキューブをコンピュータが理解し，それを解いて見せてくれる，という課題にチャレンジした．

2 章 ルービックキューブに関する基本事項

この章では，本研究を理解するために必要な事項を述べる．

2.1 ルービックキューブの用語

ルービックキューブには様々な種類がある．今回使用するものは，図 2.1 のように，3×3 のキューブで構成された立方体で，世界標準配色を用いた．ルービックキューブの全体は 26 個のキューブで構成される．各面の 3×3（9 個）のキューブの中で，中央のキューブは移動しないと考えることができ，そのキューブの色を面の色とみなす．

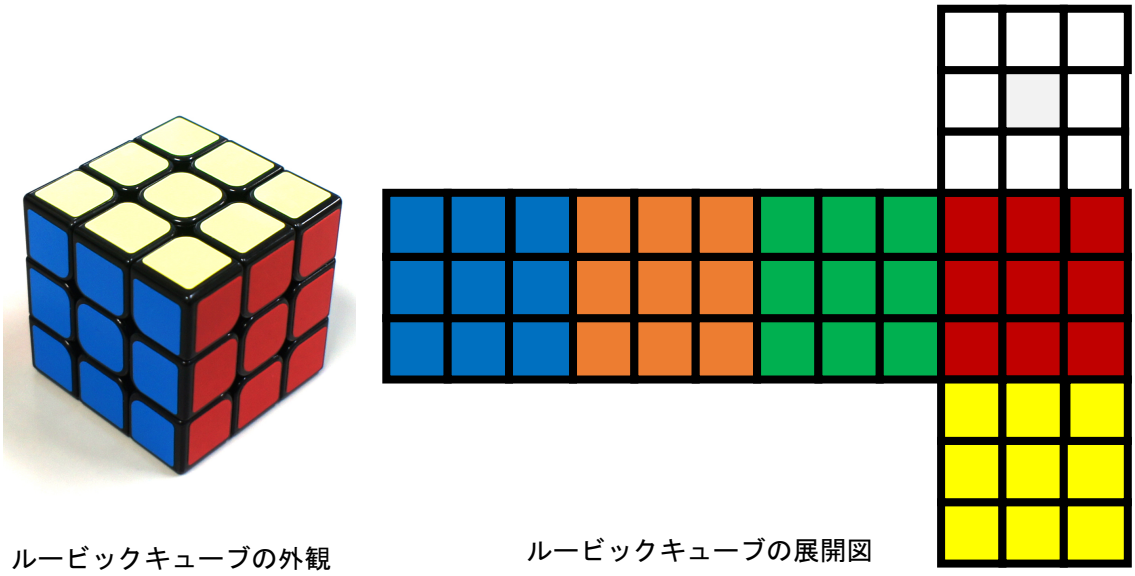

図 2.1 ルービックキューブの外観と展開図

図 2.2 のようにキューブの名称を定義する．各面の中心にあるキューブをセンターキューブとよぶ．センターキューブの上下左右にあるキューブをエッジキューブ，各面の 4 隅にあるキューブをコーナーキューブとよぶ．

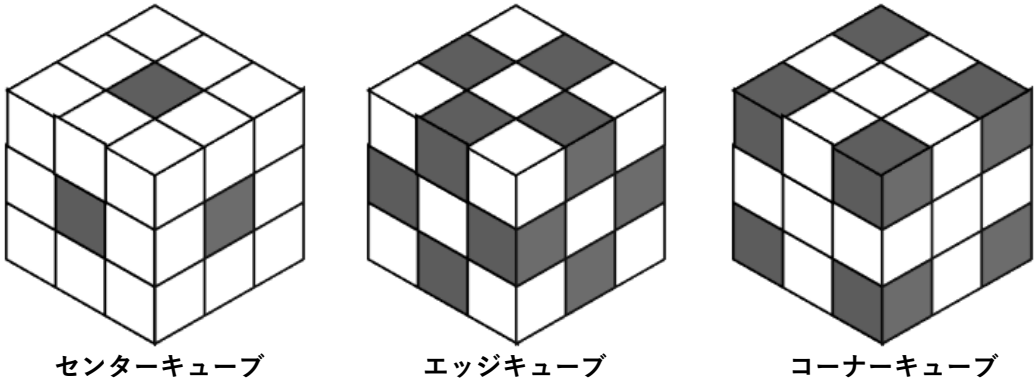

図 2.2 キューブの名称

2.2 ルービックキューブの解法

ルービックキューブの解法には，ツクダ式やCF法，LBL法，CFOP法などがある[6]．これらの解法は人間が覚えて使うことを前提としているので，手順がシステマチックで，記憶すべきパターン数が少ない．今回採用する解法は，最もポピュラーとされているLBL法を基本とする方法である（4章で詳細に説明する）．

LBL法とはLayer By Layer 法の略称である．この解法ではルービックキューブを3層の構造からなると考え，各層ごとに色を揃えて行く．手順が少なく簡単であるため，人が理解しやすい．完成までの流れは以下の4段階になる(図2.3)．

① 1層目の十字部分を揃える．
② 1層目の側面を揃える．
③ 2層目を揃える．
④ 最終層を揃える．

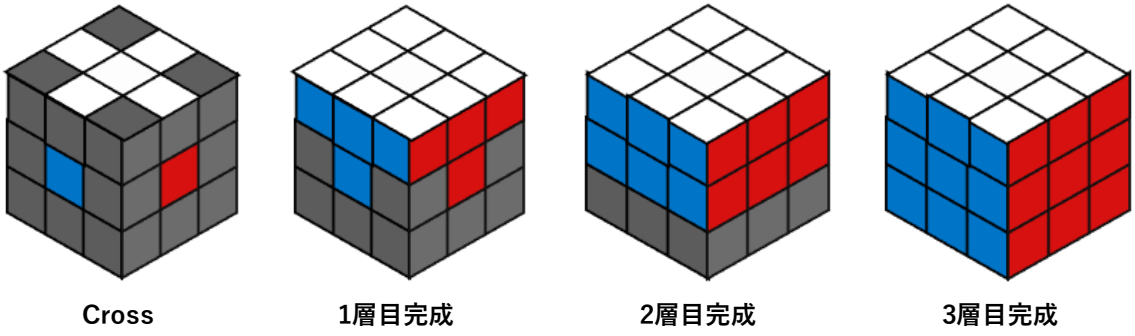

図2.3 LBL法

CFOP法はLBL法を基に考えられた解法である．名前の由来は，手順の中に含まれるCross，F2L(First two Layer)，OLL(Orienting the Last Layer)，PLL(Permuting the Last Layer)の頭文字をとったものである(図2.4)．完成までの流れは次のようになる．

① 1層目の十字部分を揃える（Cross）．
② 1層目と2層目を同時に揃える（F2L）．
③ 3層目の上面を揃える（OLL）．
④ 3層目の側面を揃える（PLL）．

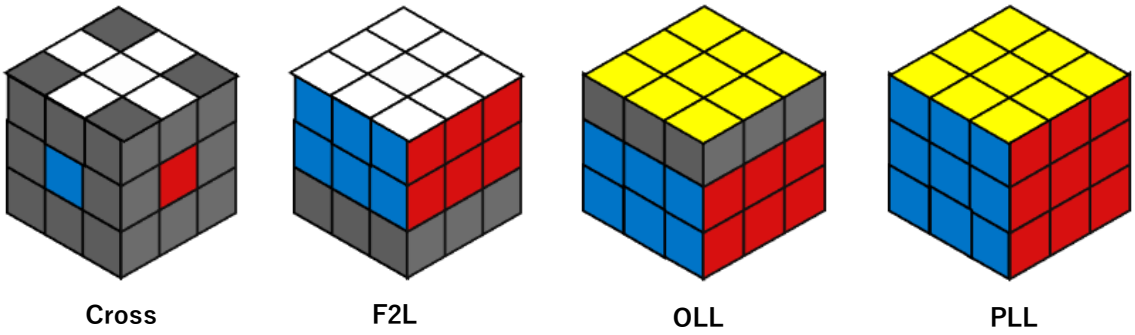

図 2.4 CFOP 法

CF 法は Corner First 法の略称である(図 2.5)．この解法は 1 層目の完成までは LBL 法と同じである．次に 3 層目のコーナーキューブを揃え，次いで，残りを揃えることで 3 層目を完成させる．最後に 2 層目を揃えてルービックキューブの完成となる．この解法は手順が 5 段階になり，LBL 法よりも多いため採用しなかった．

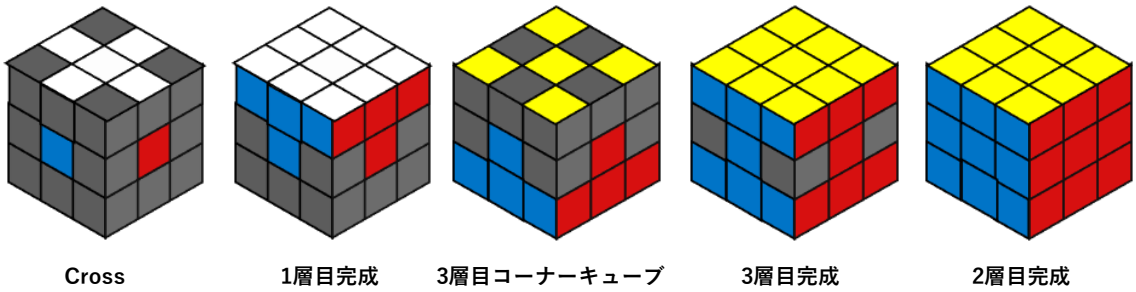

図 2.5 CF 法

ツクダ式は，ツクダ社が販売していたルービックキューブに，付属品としてつけてあった解法である．暗記するパターン数が LBL 法よりも少ないが，揃える時間が少し長くなる．この解法は上記の CF 法とほぼ同じ解法であり，上記と同じ理由で今回採用しなかった．

3 章 ルービックキューブ映像の認識

6 面の色が不揃のルービックキューブの配列をコンピュータに入力するため，面ごとに色の並びを認識するプログラムを開発した．このプログラムは，図 3.1 左のようなルービックキューブをカメラで撮影し，図 3.1 右のように 6 面の色の配置を求めるものである．今回は，6 面の正面像を青→オレンジ→緑→赤→黄→白（各面の中央の色）の順番で撮影することにした．

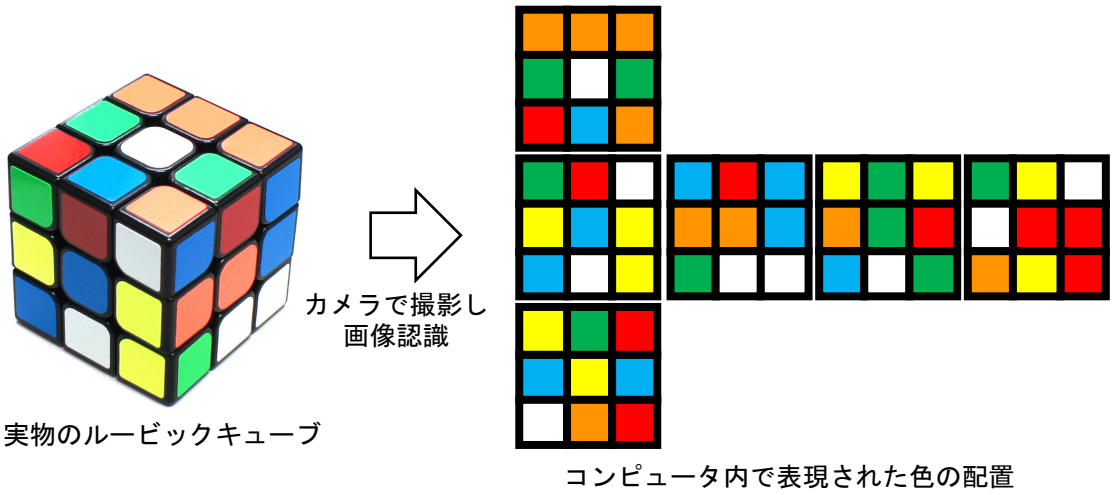

図 3.1 ルービックキューブの画像認識

3.1 画像認識の予備実験

手持でルービックキューブを撮影した場合，画像中のルービックキューブの大きさと姿勢が変化するため，検出が難しくなる．今回は，ルービックキューブを正面に向けて撮影することにした．このように問題を簡単化しても，見かけの大きさと色の並びの変化に対応する必要がある．これを解決する方法として，OpenCV3 の顔認識に使われるカスケード型分類器をルービックキューブ用に学習させる方法と，深層学習による方法を検討した．

OpenCV3 のカスケード型分類器にルービックキューブ画像を学習させるために図 3.2 のようなルービックの正解画像を 3232 枚用意した． 1 枚の元画像から回転・鏡映によって 8 個の画像を生成したので，元画像の枚数は 404 枚である．カスケード型分類器の学習には不正解画像も必要である．不正解画像として VOC2007 のデータセット[7]から 4436 枚を利用した． これらを訓練データとして， OpenCV3 に用意されている createsamples と tranincascade のコマンドツールを使い，分類器を学習させた．学習に利用した PC は HP EliteDesk（インテル i5-4590， 3.3GHz， メモリ 8GB）．学習に要した時間は概ね 10 時間であった．

生成した分類器を cv::CascadeClassfier に入力し，その detectMultiScale()メソッドを使って認識させた．その結果，ルービックキューブの正面が自然な感じで映った映像からルービックキューブの領域を検出することができた．処理速度は実用的なレベルであった．しかし，検出精度は十分とは言えなかった．ルービックキューブの正面が一つだけ映っている映像から，その中心位置を正確かつ確実に検出したいのであるが，検出位置がずれたり，二重・三重に検出してしまう，或いは，明らかに正面が映っているにもかかわらず検出できない，ということが発生した．訓練データの数が不十分である可能もあるが，カスケード型分類器による認識の実験は，一旦，これで終了した．

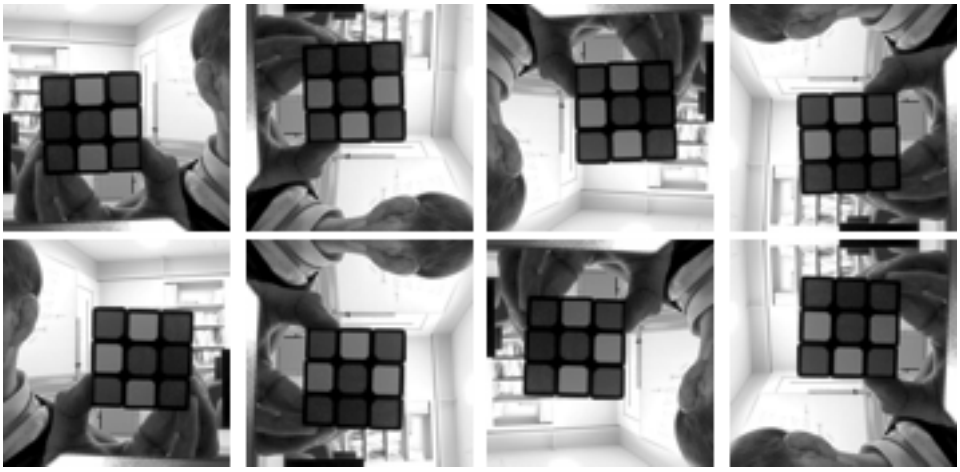

図 3.2 カスケード型分類器の学習に利用した正解画像の例

これらは１枚の元画像から生成した８枚の画像である．カスケード分類器の学習には濃淡画像を用いた．実際に学習させるルービックキューブの領域はマウスで指示した四角形領域である．

第二の予備実験として，深層画像のニューラルネットを利用してルービックキューブを画像認識する実験を行なった．画像認識に使われる深層学習ニューラルネットの中で，認識速度が格段に高速な YOLOv3 を検討した．YOLOv3 の開発環境は，提案者の Joseph Redmod が Linux と Mac OS で利用できるようにしたものがオリジナルである．今回は，それを Windows 環境で利用できるようにした AlexeyAB/darknet[8]を使って YOLOv3 を学習させた．学習済みの YOLOv3 の動作を，AlexeyAB/darknet に準備されているコマンドツールを使って，確認することができる．例えば，Web カメラを入力とした物体検出の能力を容易に見ることができる． YOLOv3 をユーザプログラムに組み込む場合には，OpenCV3 の dnn モジュールを利用することができる．

この環境で YOLOv3 を学習させるために，ルービックキューブの正面画像を 1768 枚準備した．カスケード型分類器の場合と同様に，一枚の元画像から対称型の８枚の画像を生成したので，元画像の枚数は 221 枚である．正面画像の一例を図 3.3 に示す．Darknet

による YOLOv3 の学習では正解画像だけを準備すれば良い．学習は HP Z800（インテル Xeon E5607 2.27GHz，12GB）に NDIVIA Geforce GTX1060 のグラフィックボードを搭載したものを利用した．一晩の学習（darknet の学習回数で 7,000 回）で，誤差が十分に収束したと思われたため，その学習データを利用して認識性能を確認した．

AlexeyAB/darknet のコマンドツールを使って，Web カメラから入力した映像に対してルービックキューブを認識させたところ，速度・検出精度の両方において実用レベルの性能であると思われた．次いで，これとほぼ同様の機能を，OpenCV3 を使った自作のユーザプログラムとして作成し，性能を確認した．この場合，処理速度が不十分（数秒で１枚の処理速度）という結果になった．これは OpenCV3 による実装が GTX1060 を利用していないことが原因であると考えられる．

この予備実験によって，YOLOv3 を GTX1060 程度の GPU を利用できる環境で動作させれば，リアルタイムでのルービックキューブ認識が可能になることがわかった．しかし，GPU で動作する YOLOv3 を自作のプログラムに組み込むには，まだ時間を要すると考えたので，ルービックキューブの位置を自動的に画像認識する機能を実装することは断念した．したがってこの特別研究では，映像中のルービックキューブの中央位置を映像の中央に合わせることにした．

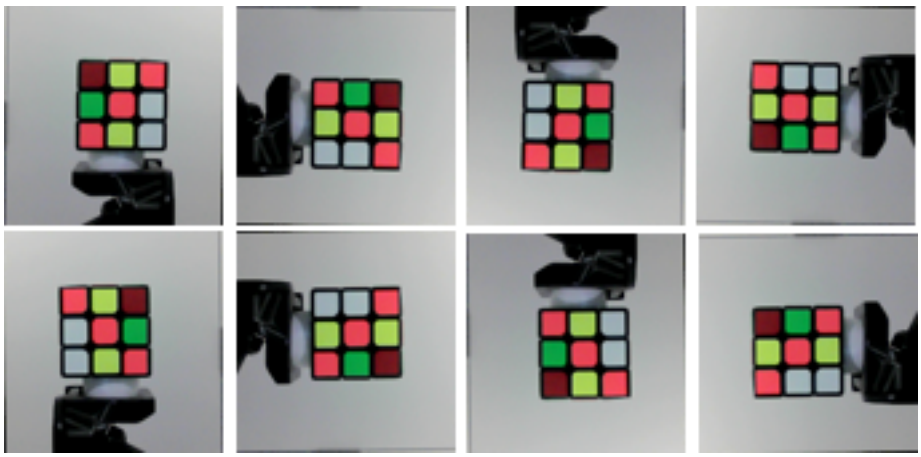

図 3.3 YOLOv3 の学習に利用した正解画像の例．
これらは１枚の元画像から生成した８枚の画像である．YOLOv3 の学習にはカラー画像を用いた．実際に学習させるルービックキューブの領域はマウスで指示した四角形領域である．

3.2 ルービックキューブの色配置の認識手法

最終的に開発したルービックキューブの色配置の認識手法を説明する．図 3.4 のように，カメラ映像の表示画面中央に白丸を描き，ルービックキューブの中央を白丸に合わせる．その後，図 3.5 の手順で９個の領域抽出と各領域の色を判定した．

図 3.4 ルービックキューブの色配置を認識した例.

図 3.5 のフローチャートにおける，各処理ブロックの内容を簡単に説明する．認識は面ごとに進める．この時，面中央の色を指定することで，その他の色の判定を確実に行えるようにした（詳細は割愛）．中央の色領域を抽出するため，ケニーの手法でエッジ抽出した二値画像を利用した．この時，通常の濃淡画像ではなく，RGB の最大値を画素値とする濃淡画像からエッジ抽出した. このようにすることで, 通常の濃淡画像よりも, 正しくエッジを抽出できるようになった. 抽出したエッジ画像で中央の色領域を囲むことができる場合が多いが，それを確実にするために，エッジ画像に膨張処理を加えた. その後，中央の色領域を OpenCV3 の floodfill 関数で抽出した．この関数は，抽出した領域を囲む長方形領域を返すので，これを利用して，中央の色領域の中心・幅・高さを求めた. 次いで, 中央の色領域の周囲に存在する８個の領域の位置を求める. これらを, 中央色領域の中心・幅・高さから推定した．全９領域の位置から RGB 値を取得し，それらの値を「青・オレンジ・緑・赤・黄・白」のいずれかの色に判別した．色の判別には $L^*a^*b^*$色空間を用いた.

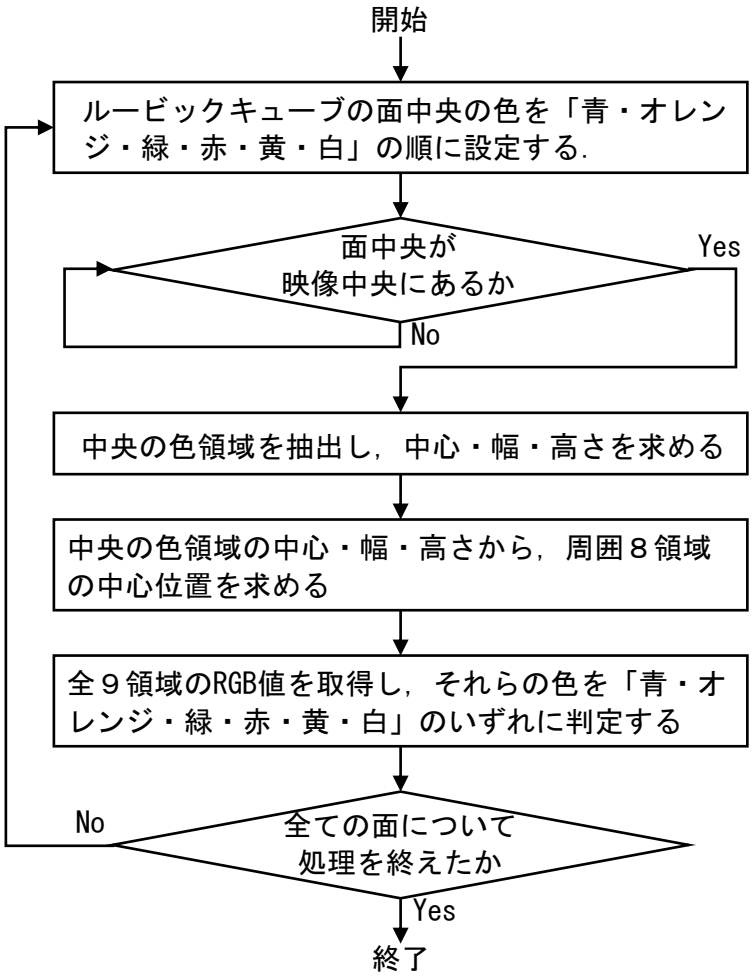

図 3.5 ルービックキューブの色配置を認識するための処理の流れ.

4 章 ソルバープログラムの開発

4.1 開発環境

ソルバープログラムの開発環境として Unity を用いた. Unity はゲームエンジンを内蔵した統合開発環境である. プログラミング言語として, C#と JavaScript, Boo の 3 つが利用できる. 本研究では C#を用いた.

ルービックキューブの 3D モデルを Blender で作成した. Blender は 3DCG を作成するためのソフトウェアである. Blender の特徴は, ① オープンソースで無料, ② マルチプラットフォーム対応(Windows/Mac), ③作成したモデルを Unity で簡単に利用できる（実際には fbx フォーマットを使用した), などである.

これら以外に, ルービックキューブとこれを撮影するためにカメラを用いた.

4.2 システムの動作の概要

開発したシステムは図 4.1 のように動作する. まず, ルービックキューブをランダムな配置にする. その状態をカメラで撮影し, 色の配置を画像認識する. 3D モデルのルービックキューブに色の配置を反映させる. 最後にソルバープログラムで反映させたルービックキューブの 3D モデルを解く.

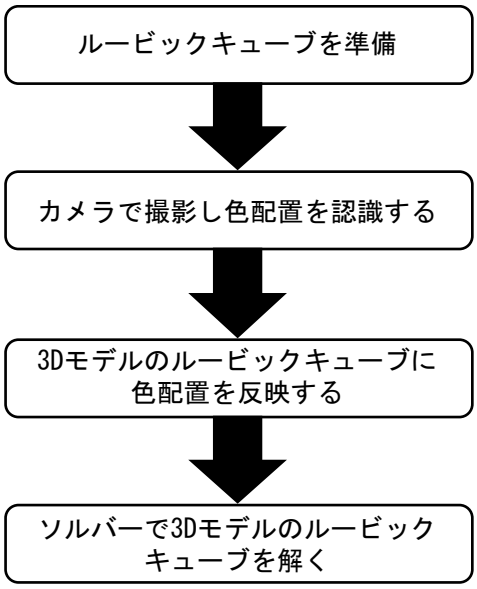

図 4.1 開発したシステムの動作の流れ

4.3 ルービックキューブの 3D モデル

ルービックキューブの 3D モデルを Blender で作成した. 図 4.2 左のように, 一つのキューブをモデリング座標系で定義し, それに適切な幾何学的変換を加えてワールド座標系にコピーする. これを 26 個のキューブについて繰り返すことで図 4.2 中央のルービックキューブの 3D モデルが完成する. この 3D モデルを Unity にインポートする. ソ

ルバープログラムは Unity で作成した．各キューブには固有の名前が付いており，それぞれを区別できる．ソルバープログラムはルービックキューブの色配置を判断し，その都度，必要な回転操作を計算するプログラムである．

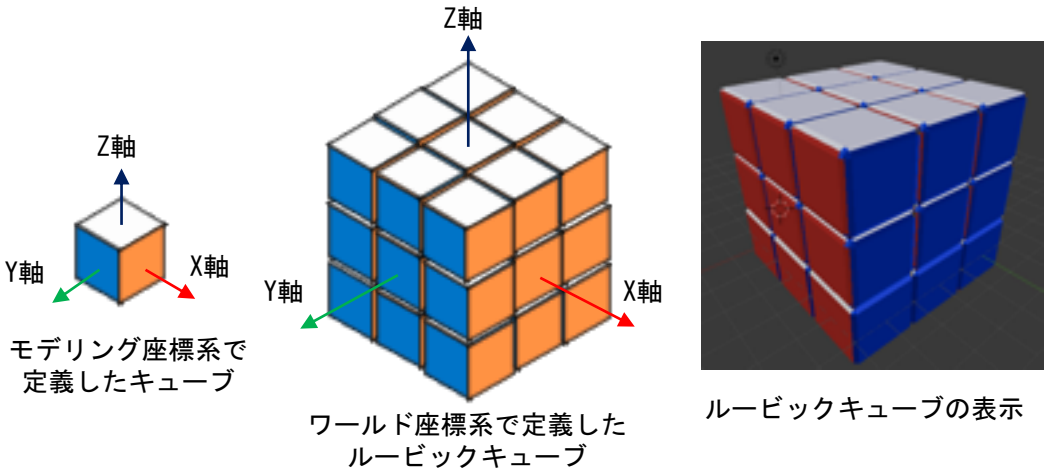

図 4.2 Blender で作成したルービックキューブの 3D モデル

4.4 3D モデルの状態を表現するデータ構造

ルービックキューブを解く操作は，結局，面を回転させることの組み合わせである．ルービックキューブの 3D モデルの面を回転させる動作は図 4.3 のように実行される．

① 回転面（同じ意味であるがセンターキューブの色）を指定する．

② 回転する 9 つのキューブを決定する．

③ 回転角度（±90° の倍数）を指定する．

④ 9 つのキューブを回転させる．

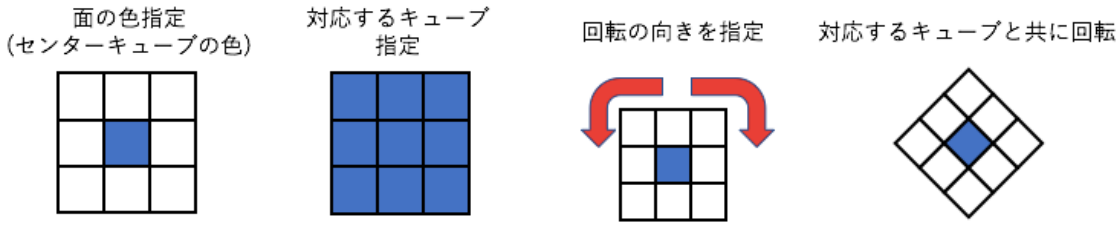

図 4.3 ルービックキューブの 3D モデルの回転動作

この回転においてセンターキューブは移動しない（姿勢は変化するが，見え方は同じである）．他の 20 個のキューブは移動し，その位置に異なるキューブが入る．その結果，ルービックの色配置が変化する．この状態変化をソルバープログラムで処理するために，センターキューブの位置を除く 20 箇所のキューブ位置に図 4.4 の「キューブ位

置番号」をつけた．この番号はキューブの位置を識別する番号であり，各キューブの位置を保存する「キューブ位置配列」を作成した．

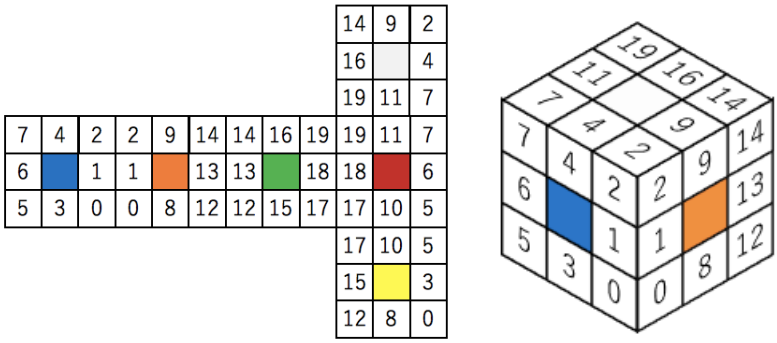

図 4.4 キューブの位置に付与する番号

20 個のキューブ位置には，合計 48 個の可視面が存在する．これらに図 4.5 のように面番号を付与し，各面の色を保存する「面色配列」を作成した．

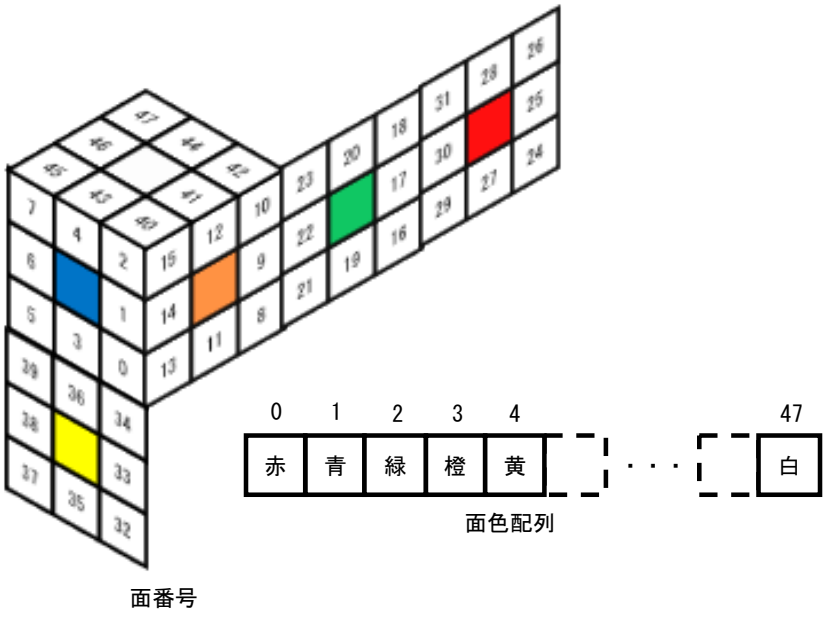

図 4.5 面番号と面色配列

ルービックキューブの基本回転動作は，6 個の面ごとに 3 種類（-90°，90°，180°）あるので 18 種類である．この全てについて，キューブの移動と面の色の変化をあらかじめ計算し，記憶しておく．例えば，図 4.4 で，青色の面を時計方向に 90° 回転する場合，位置の移動は次のようになる．

位置 0 のキューブが位置 5 に移動.

位置 1 のキューブが位置 3 に移動.

位置 2 のキューブが位置 0 に移動.

位置３のキューブが位置６に移動.

位置４のキューブが位置１に移動.

位置５のキューブが位置７に移動.

位置６のキューブが位置４に移動.

位置７のキューブが位置２に移動.

また，面の色の変化は次のようになる.

面０の色が面５に移動.

面１の色が面３に移動.

面２の色が面０に移動.

面３の色が面６に移動.

面４の色が面１に移動.

面５の色が面７に移動.

面６の色が面４に移動.

面７の色が面２に移動.

面 13 の色が面 37 に移動.

面 14 の色が面 36 に移動.

面 15 の色が面 34 に移動.

面 40 の色が面 13 に移動.

面 43 の色が面 14 に移動.

面 45 の色が面 15 に移動.

…，以下同様に，全体で 20 個の面の色が移動する.

4.5 ソルバープログラムのアルゴリズム

開発するソルバープログラムは LBL 法を基本とした．LBL 法では，最初に１層目の上面を十字に揃えるが，調査の結果，上面を全て揃える方法[9]があった．この方法は LBL 法よりもプログラムが簡単になるので，これを採用することにした．これにより，処理の流れは以下の 4 段階になる.

① １層目の上面を揃える.

② １層目の側面を揃える.

③ ２層目を揃える.

④ 最終層を揃える.

これを順に説明する.

4.5.1 １層の上面を揃える

一つ目の面をどの色で揃えても良いが，白面を揃える場合について説明する．上面に

移動させる白色を場合分けすると，図 4.6 の 9 パターンに分けることができる．エッジキューブを移動させる場合がパターン１・２・３・４・８、コーナーキューブを移動させる場合がパターン５・６・７・９である．パターン１と４は３層目のエッジキューブを移動させる場合である．パターン２と３は２層目のエッジキューブを移動させる場合である．パターン８は１層目のエッジキューブを移動させる場合である．パターン５・６・７は３層目のコーナーキューブを移動させる場合である．パターン９は１層目のコーナーキューブを移動させる場合である．

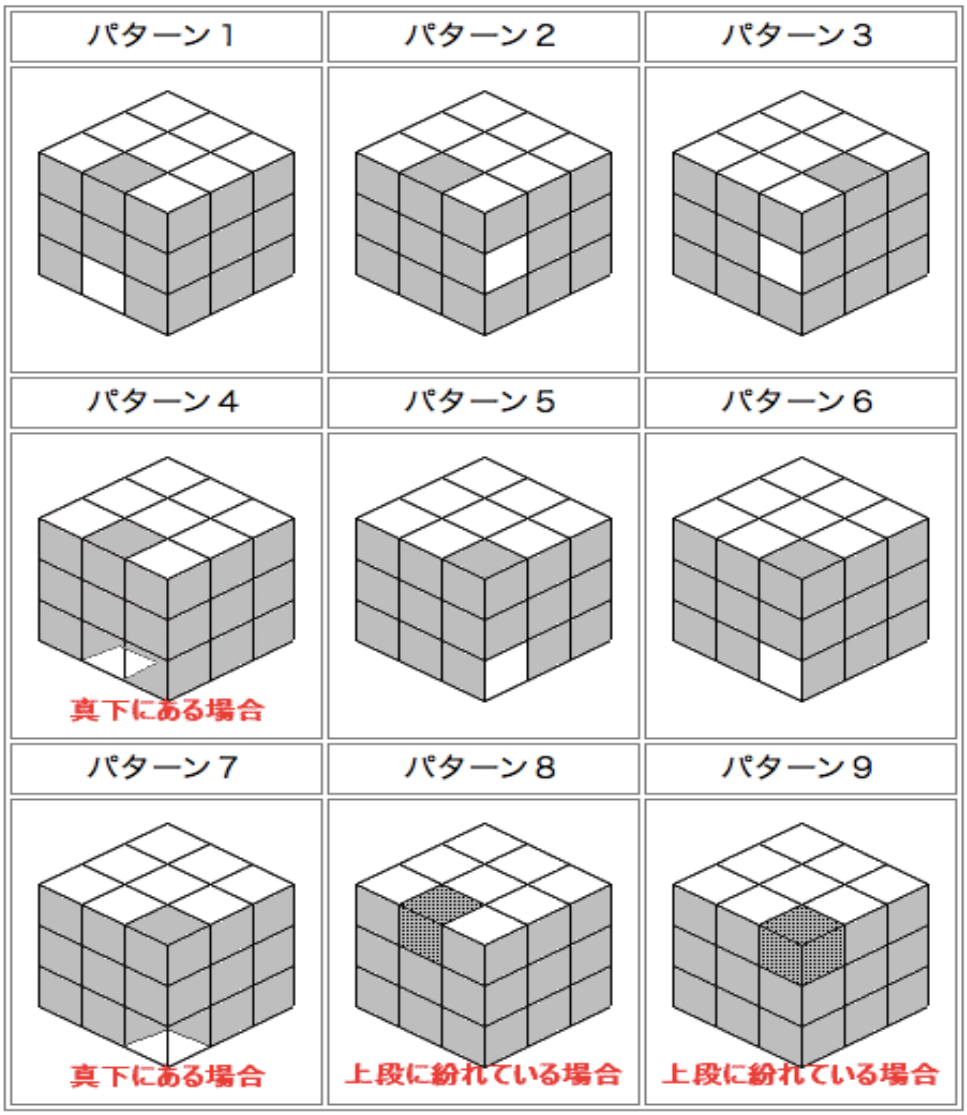

図 4.6 一つの面を揃える方法[9]

処理中のルービックキューブに当てはまるパターンを探す．パターンを見つければ，パターンごとに決まった回転操作を行う．図 4.7 は，図 4.6 のパターンごとに行う回転操作を示している．

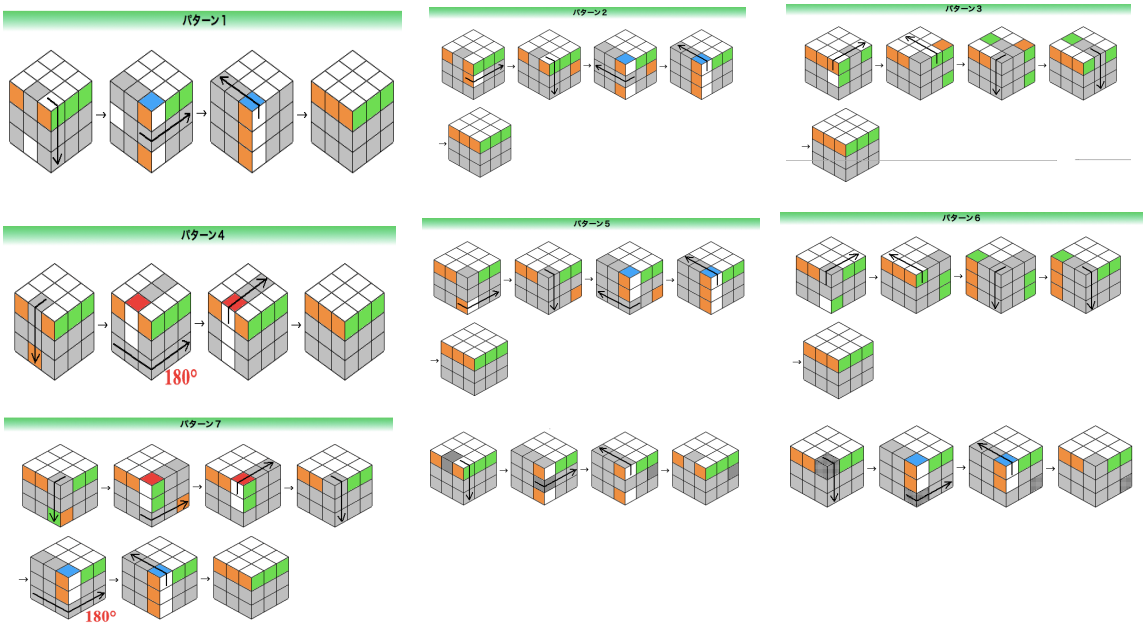

図 4.7 図 4.6 のパターンごとの回転操作[9]

これらを実装するため面色配列を利用する．白面のセンターキューブ以外の 8 個キューブの色は面色配列[40]～[47]に格納されている．面色配列[40]から順に白か否かを判定する．白の場合は何もせず次に進む．白でない場合，図 4.6 のパターン 1 から 7 で当てはまるものを探す．当てはまるものがあれば，図 4.7 の対応する回転操作を加える．この処理が面色配列[47]まで終了すれば，残ったパターン 8, 9 について，面色配列[40]～[47]を調べ，該当する回転操作を加える．例として，面色配列[43]が白ではなく，面色配列[3]が白であれば，図 4.6 のパターン 1 に該当する．その場合，図 4.7 のパターン 1 を参照して，青面を 90° 回転，白面を 90° 回転，黄面を-90° 回転，赤面を-90° 回転させる（2 層目の回転操作は白面+黄面の回転で行っている）．

4.5.2 1 層目の側面を揃える

1 段階目で揃えた面の側面を揃えていく[10]．まず，どのコーナーキューブでも良いので一つを基準のキューブとして選択し，そのキューブが正しい位置になるように白面を回転させる．今回は図 4.8 左のように，白赤青のコーナーキューブを基準のキューブとして説明する．

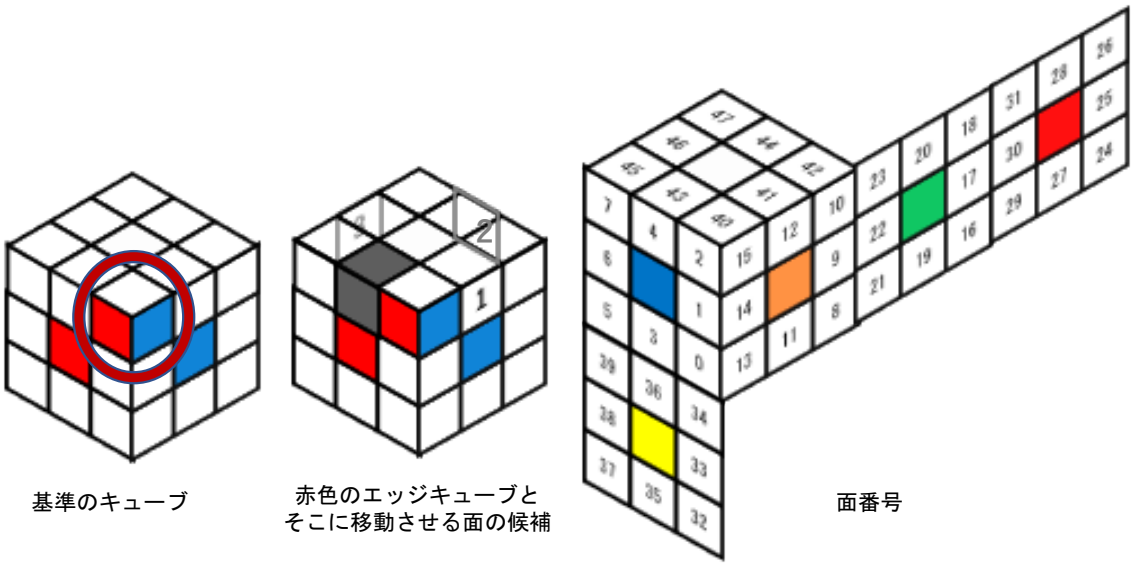

図 4.8　1 層目の側面を揃える方法を説明する図

赤色センターキューブの上のエッジキューブが揃ってない場合, 図 4.8 中央で示す三ヶ所のいずれかに必要な面がある．この三ヶ所は，面番号をつけた右側の図では，[4][12][20]に該当する．図 4.6 を使い当てはまるパターンを探す．続いて，図 4.7 を使いパターンに応じた回転操作を行うと，白赤のエッジキューブを[28]の位置に移動させることができる. 面番号[28]の左隣にある面番号[31]に該当するコーナーキューブを同じように揃える．これで赤色の 1 層目の側面が揃う．同様の方法で緑・オレンジ・青を揃えると図 4.9 の 1 層目が揃った状態になる.

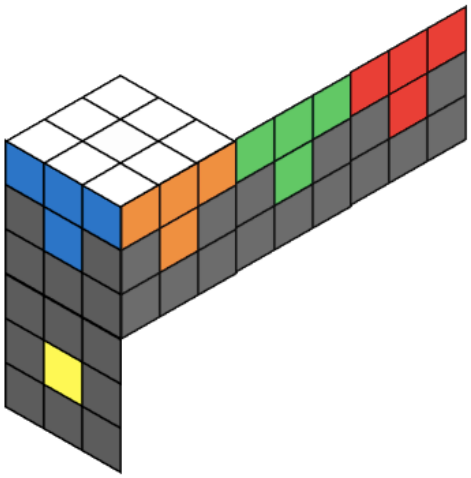

図 4.9　1 層目の完成

4.5.3 2 層目を揃える

2 層目で揃えるキューブは図 4.10 のグレーで示しているエッジキューブである.

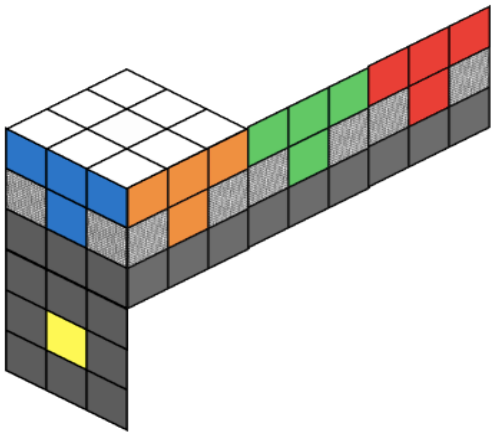

図 4.10 2 層目の揃えるキューブ位置

このエッジキューブを揃える前に 1 層目を下にする. 2 層目のエッジキューブの揃え方は 3 パターンある. 図 4.11 に, 青オレンジのエッジキューブを例にした 3 パターンを示す. この 3 パターンに応じた回転操作を, 2 層目の青オレンジ, オレンジ緑, 緑赤, 赤青の 4 つのエッジキューブに対して行うと, 図 4.12 の 2 層目が揃った状態になる.

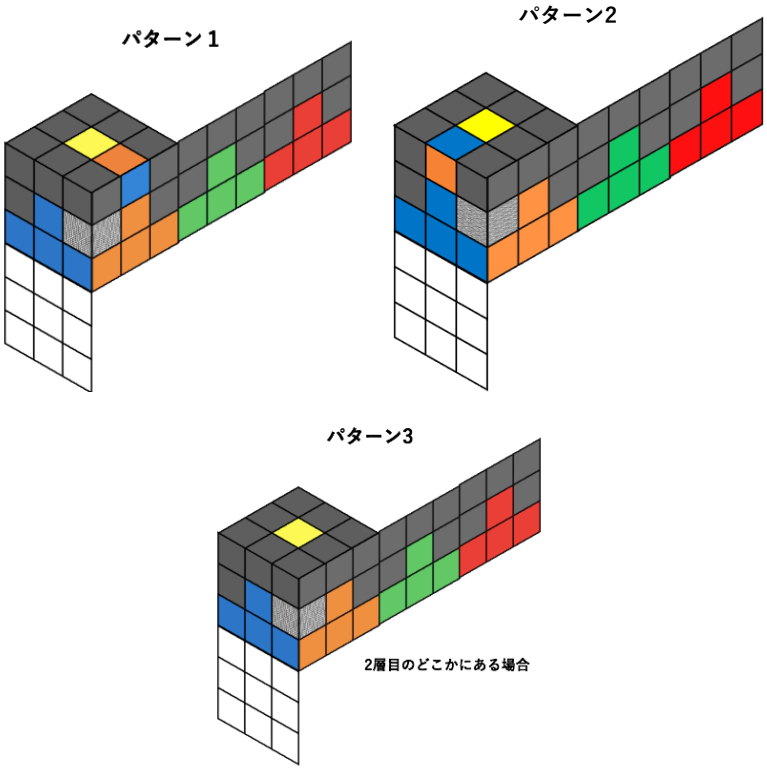

図 4.11 2 層目揃える場合の 3 つのパターン
(青オレンジのエッジキューブ)

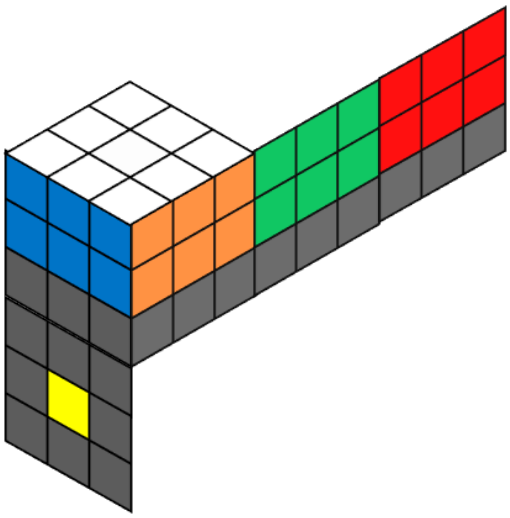

図 4.12 2 層目の完成

4.5.4 3 層目を揃える

ソルバープログラムの最終段階である 3 層目を揃えるには図 4.13 で示す段階がある．下面（今回の説明では黄色面）を十字型に作成し，そこから下面を完成．最後にその側面を揃えてルービックキューブの完成となる．

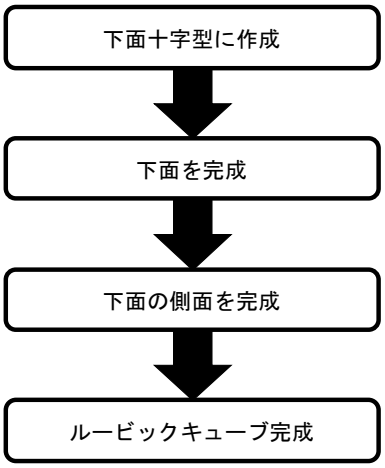

図 4.13 最終段階の手順

以上の手順を順に行う．今までと同様に，それぞれのパターンに応じた回転操作を行う．詳しいパターンの分け方は省略するが，3 層目完成には合計 12 パターンの作成が必要となる．その内訳は，下面十字型の作成にはループする 1 パターン．下面の完成には，初めに 4 パターン，次にループする 1 パターン，最後に 2 パターンのどちらかを行う．下面の側面を揃えるには 4 パターンある．すべてを行うと図 4.14 のようにルービックキューブが完成する．

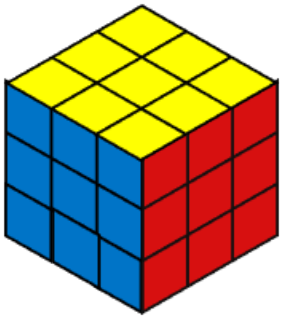

図 4.14 ルービックキューブ完成

4.6 映像認識プログラムとの統合

ルービックキューブの状態を 3D モデルに反映させるために，ルービックキューブの画像認識プログラムとソルバープログラムの 2 つを組み合わせた．

組み合わせに必要となる機能は，① 画像認識プログラムから得るルービックキューブの色配置に関する情報をソルバープログラムに読み込む，② 各キューブの位置を格納している「キューブ位置配列」と面の色情報を格納している「面色配列」にそれぞれ情報を格納する，③ 各キューブの位置や姿勢を適切に変更する，の 3 点である．これらに付いて説明を補足する．

① 図 4.15 の色配置のルービックキューブを例に説明する．ルービックキューブの画像を撮影する時，白面が上に来るようにルービックキューブを持ち，青面→オレンジ面→緑面→赤面と回転させながら撮影する．残る黄面と白面の撮影は，青面に戻してから上下方向に回転させ，黄面→白面と撮影する．このように撮影することで，「キューブ位置配列」と「面色配列」の配列の番号と合わせやすくなる．

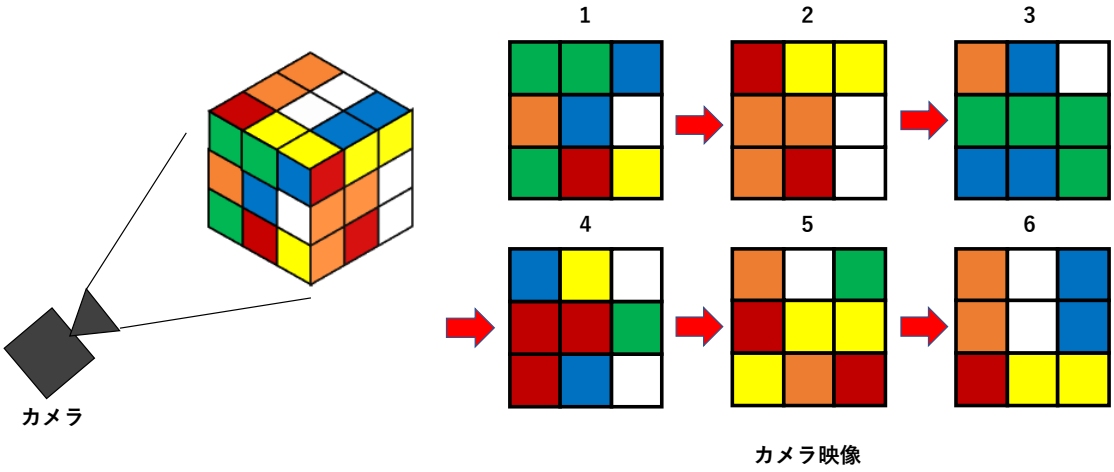

図 4.15 面の撮影順と面の向き

撮影した色配置をテキストファイルに書き出す. センターキューブの色を除く８色を, 各面ごとに１行のテキストファイルとした. 色と文字の関係は, 青面:B, オレンジ:O, 緑面：G, 赤面：R, 黄面：Y, 白面：W とした. テキストファイルに書き出す順番と面位置の関係を図 4.16 に示す. これは面色配列の順番と同じである.

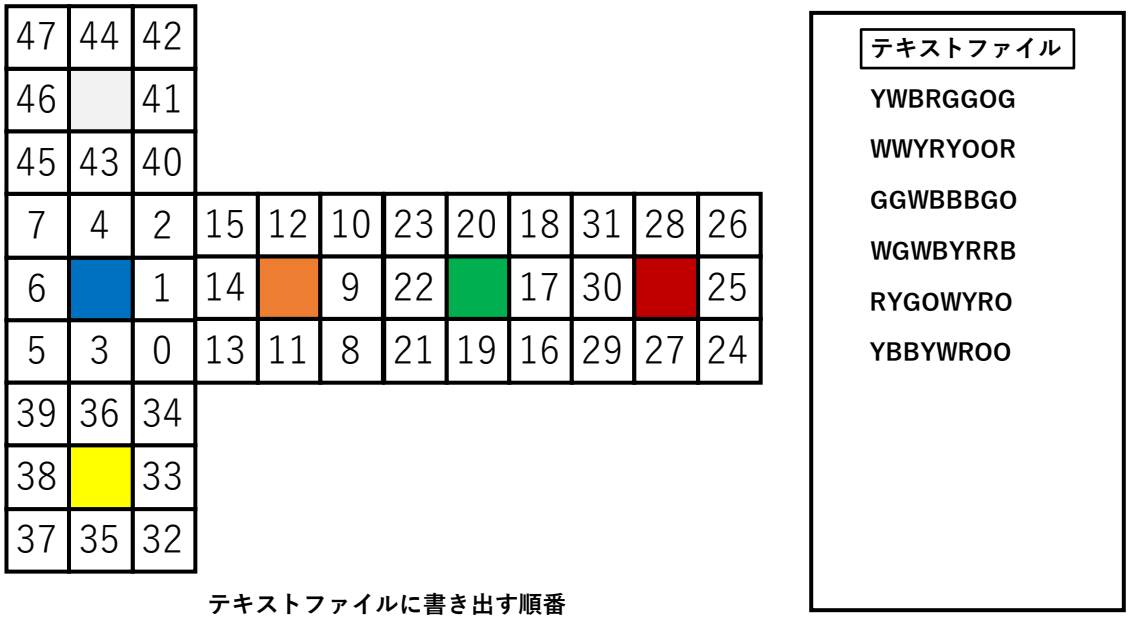

図 4.16 カメラで撮影した面をテキストファイルに書き出す

② 色配置を記述したテキストファイルを Unity で作成したソルバープログラムに読み込む. ソルバープログラムは面ごとの色情報を「面色配列」に格納する. また, キューブ位置情報を面色の配置から分類して格納する.

③ 面の色の配置からそれぞれのキューブを正しい位置・姿勢に変更する. 正しい位置に変更するには, 面色配列と各キューブの位置を示す配列を図 4.17 のように比較することでわかる. 例えば面色配列[0][13][34]はキューブ位置[0]に対応する.

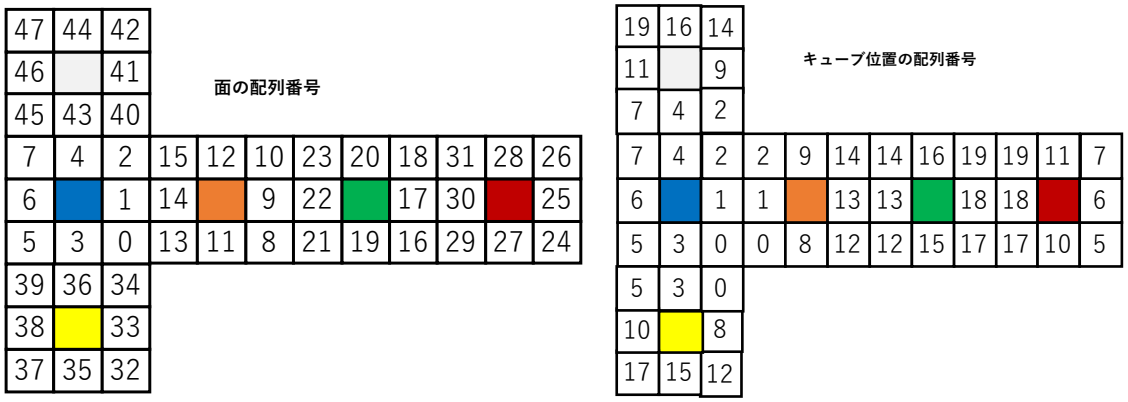

図 4.17 面色配列とキューブ位置配列

これらを利用すると，各キューブの位置と姿勢がわかるので各キューブをルービックキューブの正しい位置に移動し，キューブの位置と面色の配置から適した姿勢に変更する．センターキューブを除くキューブの数は 20 個あり，そのうちエッジキューブは 12 個，コーナーキューブは 8 個である．エッジキューブには 2 面，コーナーキューブには 3 面ある．そのため，エッジキューブの姿勢を変更するには(12×2)×12 = 288 パターン必要であり，コーナーキューブの姿勢を変更するには (8×3)×8 = 192 パターン必要となるので各キューブの姿勢を変更するには合計 480 パターン必要である．

5 章 実験結果と考察

開発したシステムを統合し，動作を検証した.

5.1 ソルバープログラム単体での動作確認

ソルバープログラムを単体で動作させて，正しい結果になるかどうかを確認した. この時，ルービックキューブをランダムに回転させ，初期状態とした. 図 5.1 はソルバーの動作を LBL 法の段階ごとに示した結果である. これらの結果から正しく動作をしていることが確認できた.

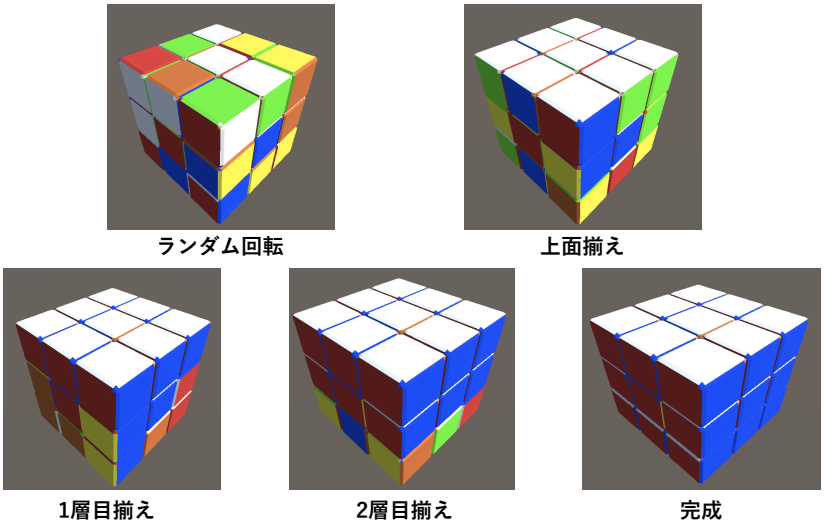

図 5.1 ソルバープログラムの動作結果

5.2 映像認識プログラムとの統合動作確認

色の揃ってないルービックキューブをカメラで撮影し，その色配置をソルバープログラムの初期状態に反映させた. 図 5.2 は実際にルービックキューブを撮影して確認した例である. これらの状態からも，ルービックキューブを正しく解くことができた.

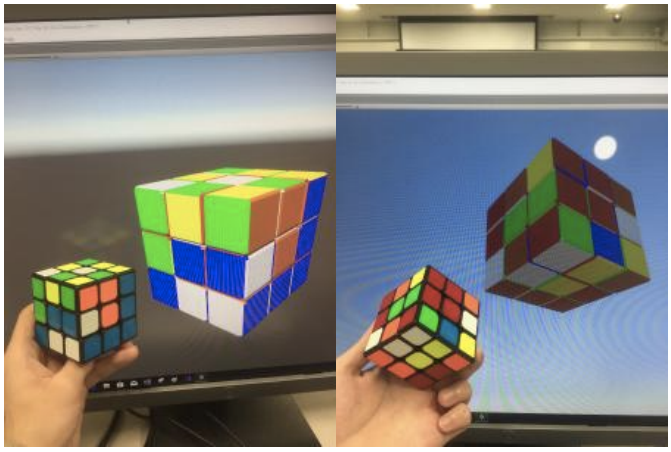

図 5.2 カメラで撮影したルービックキューブを反映させた例

5.3 考察

本研究の結果，実際のルービックキューブの状態を反映し，自動的に解くことができた．以下の点が課題としてあげられる．

① カメラでの色の認識において，赤色・オレンジ色のような区別しにくい色や光の加減によって誤認識が発生する．

② ルービックキューブの撮影手順を間違えた際，その部分だけを撮影し直すことが不可能であり，始めから撮影する必要がある．

③ 間違いに気づかずに撮影すると，ソルバー処理の途中でルービックキューブの形が崩れてしまう．

④ ルービックキューブの解法を示す時，画面で行われている操作を人間がなぞりにくい．回転の向きを矢印で示すなど，認識しやすくするべきである．

6 章 結論

本研究では，実際のルービックキューブを撮影し，それを解いてみせるシステムを開発した．そのために，Blender を使ったルービックキューブモデルの作成，Unity を使ったルービックキューブのソルバープログラムの開発，カメラを利用してルービックキューブの色配置を撮影し認識するプログラムの開発を行った．

作成した後の課題としては，扱いやすいデザイン性や人のミスを防ぐための機能の追加などが挙げられる．

結論として，複雑な 3 次元パズルであるルービックキューブの解き方は Web サイトに様々な方法が記されており，ルービックキューブ初心者でも手順を把握しながら理解することで，ソルバープログラムを作成できることがわかった．これより，ニコリのパズルといった複雑なパズル問題でも自動解法システムを作成してしまえば，新聞のおまけに挑戦している子供などにすらすら作成している姿を見せるなどして親の威厳などを保てるといったことができるだろう．

参考文献

[1] 大島知樹，「ルービックキューブの探索プログラム」，2013 年．
(http://www-mmc.es.hokudai.ac.jp/~masakazu/Student/Ooshima/poster27.pdf)

[2] God's Number is 20 (http://www.cube20.org)

[3] 橋塚和典，神原誠之，萩田紀博，「拡張現実感によるルービックキューブの解法教示システム」，情処研究報告 Vol.2014-CVIM-190 No.25.

[4] Augmented reality Rubik's cube solving assistant (Youtube ビデオ)
(https://www.youtube.com/watch?v=Pz5xvVwr9Ds，2016 年 5 月 30 日公開)

[5] 高井基己，「3 次元物体操作における VR と AR の効果」，早稲田大学大学院情報生産システム研究科，2017 年度修士論文.

[6] Cube Solver
(https://cube-solver.dikky.net/rbk_solver/various_methods)

[7] VOC2007 のデータセット
(http://host.robots.ox.ac.uk/pascal/VOC/index.html)

[8] YOLOv3 を Windows 環境で使えるようにしたサイト
(https://github.com/AlexeyAB)

[9] ルービックキューブ攻略法
（https://www.pazru.net/rubic/index.htm）

[10] ルービックキューブ簡単 6 面完成攻略法
（http://www.macozy.com/rubik/3rc/index.html）

謝辞

本論文を作成にあたり,丁寧な御指導を賜りました蚊野浩教授に感謝いたします.

付録　本研究で開発したプログラムの説明など

[プログラム名]
FaceColorCap.cs
[内容]
　ルービックキューブの 3D モデルの選択した面を回転させる機能. 回転方向を選択する機能．LBL 法を基本としたソルバープログラムの機能.

[プログラム名]
Camera_Move.cs
[内容]
　Unity でのゲームカメラの制御機能. ルービックキューブの 3D モデルを周囲 360° 見渡せる機能.

[プログラム名]
ExeStart.cs
[内容]
　映像からのルービックキューブ認識プログラムを Unity から起動させる機能.

[プログラム名]
Text_Read.cs
[内容]
　テキストファイルに書き出されたルービックキューブの色の配置を読み取り, ルービックキューブの 3D モデルの各キューブの姿勢・位置を変更する機能.

[プログラム名]
Source.cpp
[内容]
　映像からのルービックキューブ認識機能．ルービックキューブの色の配置をテキストファイルに書き出す機能.