

人工知能

第4回講義課題 課題番号 12

提出締切: 2013.12.11

提出日: 2013.12.11

工学部電子情報工学科

03-123006 岩成達哉

1 概略

線画の解釈を行うプログラムを Android アプリケーションとして作成した。このアプリケーションでは、2つの方法で入力データを与えることができる。

1つ目は、参考文献 [1] の平賀先生のページで示されるラベル情報をテキスト入力する方法。2つ目は、与えられた画像を処理して線画の解釈する方法である。

当初の目標は、カメラで撮影した画像に対して処理を行い、線画の解釈を行うというものであったが、写真の解像度に左右され、正しい結果は得られなかつた。

一方、図形描画ソフトやパワーポイントなどで作成した画像に対しては、正しい解釈を示せることがわかつたため、それらを読み込んで処理するプログラムを作成した。

2 作成したプログラムの仕様

プログラムは、以下の2つの入力方法によって解釈を行う。

1. 参考文献 [1] の平賀先生のページで示されるラベル情報をテキスト入力する方法
2. 画像を入力する方法

対応する図形は、参考文献 [1] にあるように、

『対象は3面頂点図形とする。つまりすべての面が平面（すべての辺が直線）であり、各頂点には3つの面（したがってその境界として3本の辺）が集まる図形である。』

ということを前提とする。

2.1 テキスト入力による解釈

テキスト入力は、図1のように頂点を分類し、入力する。これは、参考文献 [1] に基づく。

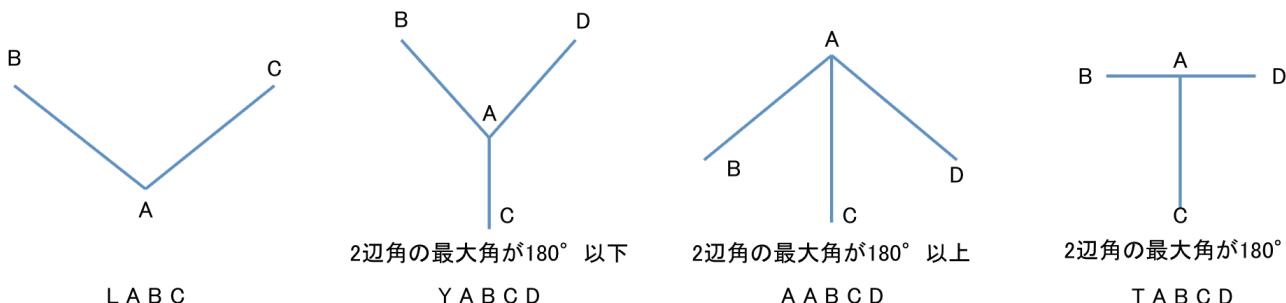


図1: 頂点の分類と入力順

入力は、これらの情報を改行で区切って入力する。また、型の種類はL,A,Y,Tの4つのみであり、頂点の名前は対応さえ正しければ複数の文字でも構わないが、必ず半角スペース1つで区切ることとする。例えば、図2のような図形の時は、リスト1のように入力しなければならない。

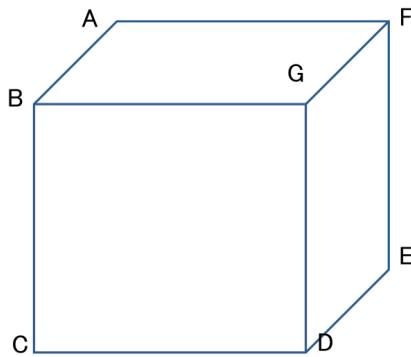


図 2: 図形の入力例

リスト 1: ラベリング情報の入力例

```

1 L A F B
2 L C B D
3 L E D F
4 A B C G A
5 A D E G C
6 A F A G E
7 Y G B D F

```

また，全ての図形は浮いていると判断する．したがって，一番外側の辺は矢印のみ（後述）が候補となる．これによって，境界を入力することで制約条件を追加することができる．境界の辺は端点を半角スペース区切りで入力し，改行によって分ける．これは，リスト 2 のようになる．

リスト 2: 境界の入力例

```

1 A B
2 C D
3 E F

```

2.2 画像入力による解釈

画像を入力することでも解釈を行えるようにした．画像は，jpeg，bitmap，png に対応しているが，画像の解像度が悪いと解釈が正常にできない．例えば入力データは図 3 のような線だけものを入力する．

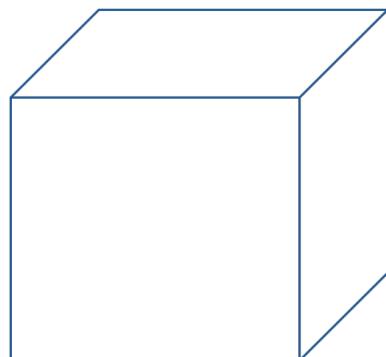


図 3: 画像の入力例

カメラで撮影したものは下処理をする必要があるため，画像はカメラで撮影したものと，描画ソフトで描いたもので区別する．

描画ソフトで描いたものを入力すると，前節で示したラベリング情報と境界情報をテキストデータとして出力する．その出力をテキスト入力の場合の解析器と同じものに入力し，解釈を行う．解釈が一意に定まれば，図にその結果を書き込み，定まらない場合は，テキストと図を両方表示する．結果が一意に決まらない場合も，頂点にアルファベットを A から順に振る．カメラで撮影したものであれば，下処理をして同様の操作を行う．

3 実行環境

- Android 2.3 以上
- OpenCV Manager[3] がインストール済み
このインストール方法については，次章に譲る．
- 本アプリケーションがインストール済み

ただし，実際にテストしたのは Android 4.1, 4.2 のみである．

4 実行の準備

4.1 OpenCV Manager のインストール

今回作成したアプリケーションは，OpenCV[2] をライブラリとして用いている．Android アプリケーションにおいては，OpenCV はライブラリとして完全に独立しており，その主な機能は Android アプリケーションとして，個別に用意されている．これが「OpenCV Manager」である．OpenCV Manager は，Play Store からインストールができる（Play Store において「OpenCV Manager」と検索するか参考文献の URL から発見できる）[3]．Android 端末で，検索を行い，インストールを行う．

インストールを行なっていない場合，本アプリケーションをインストールして起動した際に，その旨が表示され Play Store へ誘導されるため，インストールボタンを押すことでインストールが出来る．

4.2 本アプリケーションのインストール

Play Store で OpenCV Manager をインストールした後，本アプリケーションをインストールし，実行を行う．

提出したもののからの，本アプリケーションのインストールは，「LineArtAnalysis > bin > LineArtAnalysis.apk」という apk ファイルによって行う．まず，この apk ファイルを Android 端末にダウンロードする．ダウンロード方法は，DropBox[4] や Android File Transfer[5] などを用いることで行う．

ダウンロードした apk ファイルを，File Manager(Android ではアストロファイルマネージャー [6] などがある)で検索して，クリックすればインストールが始まる．DropBox などから直接インストールしてもよい．この際に，設定で「提供元不明のアプリケーションをインストール」にチェックを入れておかなければならない．

以上の内容によって，アプリケーションの実行ができる．

4.3 本アプリケーションのコンパイル

本アプリケーションの作成は Eclipse によって行っている．開発環境の構築は参考 URL[7] によって行える．OpenCV を用いた Android アプリケーションの開発は，参考 URL[8] を元に行った．本アプリケーションでは，OpenCV のライブラリをダウンロードし，外部ライブラリとして参照している．

Eclipse では，アプリケーションは自動でビルドされるため，開発環境が整えられていれば，自動的に「LineArtAnalysis > bin」フォルダに apk ファイルができている．

ソースコードは「LineArtAnalysis>src>jp>narit>lineartanalysis」以下のフォルダの内部に，リソースファイルは「LineArtAnalysis>res」以下のフォルダの内部にある．

5 アプリケーションの使い方

図 4 に起動時の画面を示した .

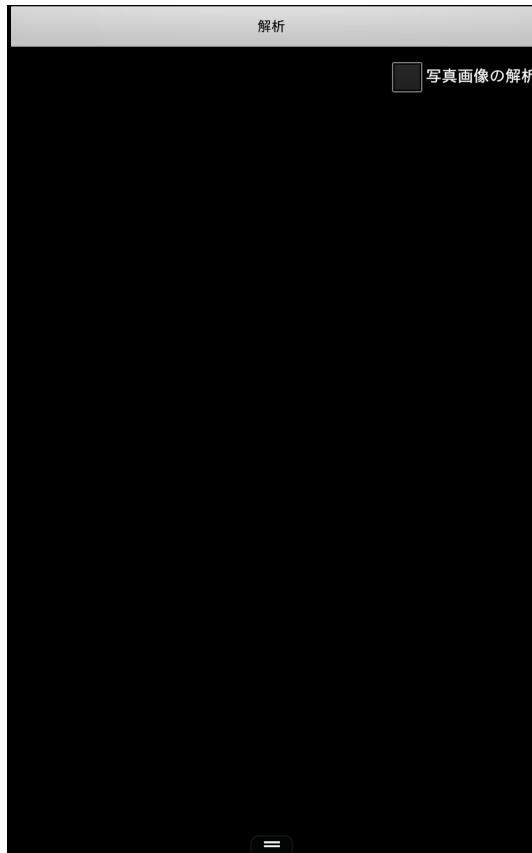


図 4: 起動時の画面

5.1 画像の解析

まずは，画像の解析方法について説明する．図 4 の上にある「解析」ボタンを押すと，図 5 のように端末のファイルエクスプローラが表示される．これは，参考文献 [9] を元に作成した．このエクスプローラを用いて，任意の画像を選択することで，解析させることができる．サンプルとして「Android/data/jp.narit.lineartanalysis/files」にサンプル画像を 7 個用意している．

解析の結果は，解釈が一意に定まる場合は画像のみ，一意に定まらなければテキストと画像で表示される．

画像以外のファイルを選択した場合は，アプリがエラー終了する．撮影した写真を入力する場合は，図 4 の上部にある「写真画像の解析」をチェックすることで行うが，現在のところ正しい結果は得られていない．

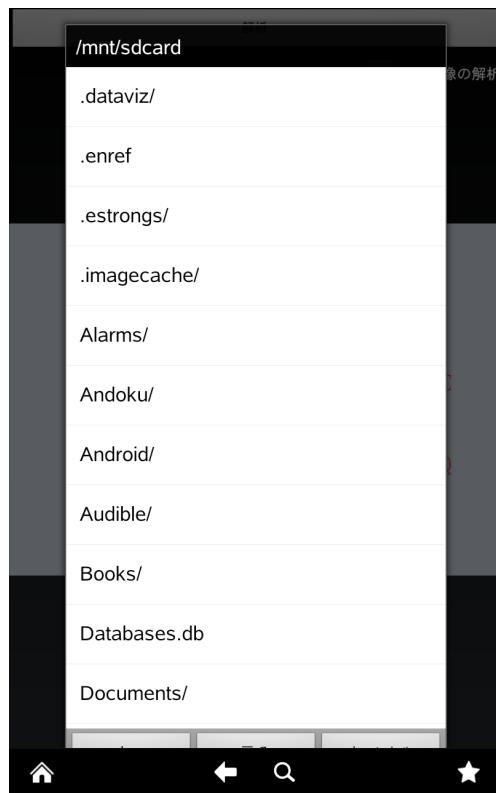


図 5: ファイルエクスプローラ

5.2 テキストの解析

次にテキストデータの解析方法について説明する。図 4 の状態で、Android 端末にある「メニュー」ボタンを選択する。このボタンは端末によって、ハードウェアボタンとして実装されている場合もあれば、ソフトウェアボタンとして実装されている場合もある。メニュー ボタンを押すと、画面の下部に図 6 が表示される。

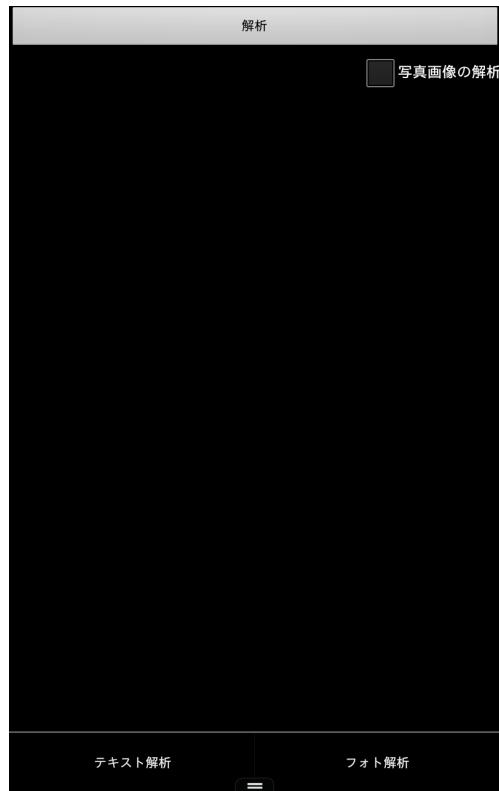


図 6: メニュー

このメニューの「テキスト解析」を選択すると、図 7 の画面へ移動する。

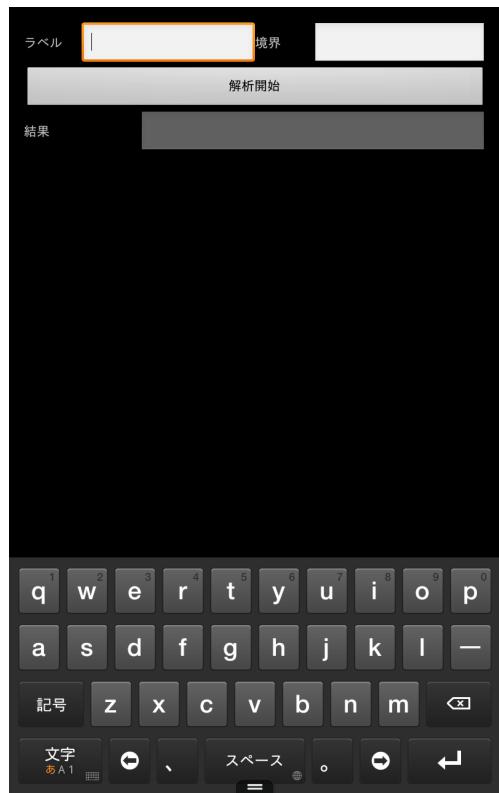


図 7: テキスト解析の画面

ここで、ラベル情報と、境界の情報を仕様に従ってそれぞれのテキストビューに入力した後、「解析開始」ボタンを選択すれば、結果が下部のテキストビューに表示される。

6 アプリケーションの構成とアルゴリズム [10]

アプリケーションの構成を図 8 に示した。大きく分けてテキストと画像の 2 つの入力方法がある。テキストで辺のラベル情報が入力された場合は、Analyzer を通して結果をテキストデータとして出力する。

写真が入力された場合は、まず Preprocessor で下処理をする。下処理をされた画像、あるいは画像描画ソフトなどで作成した図の場合は、Decoder で頂点や辺を解析し、テキストデータとして Analyzer へ送る。Analyzer で線画の解釈を行い、解釈が一意に定まれば、Colorer へ結果を入力し、画像に解釈を書き込む。解釈が一意に決まらなければ、テキストデータとして候補を羅列する。

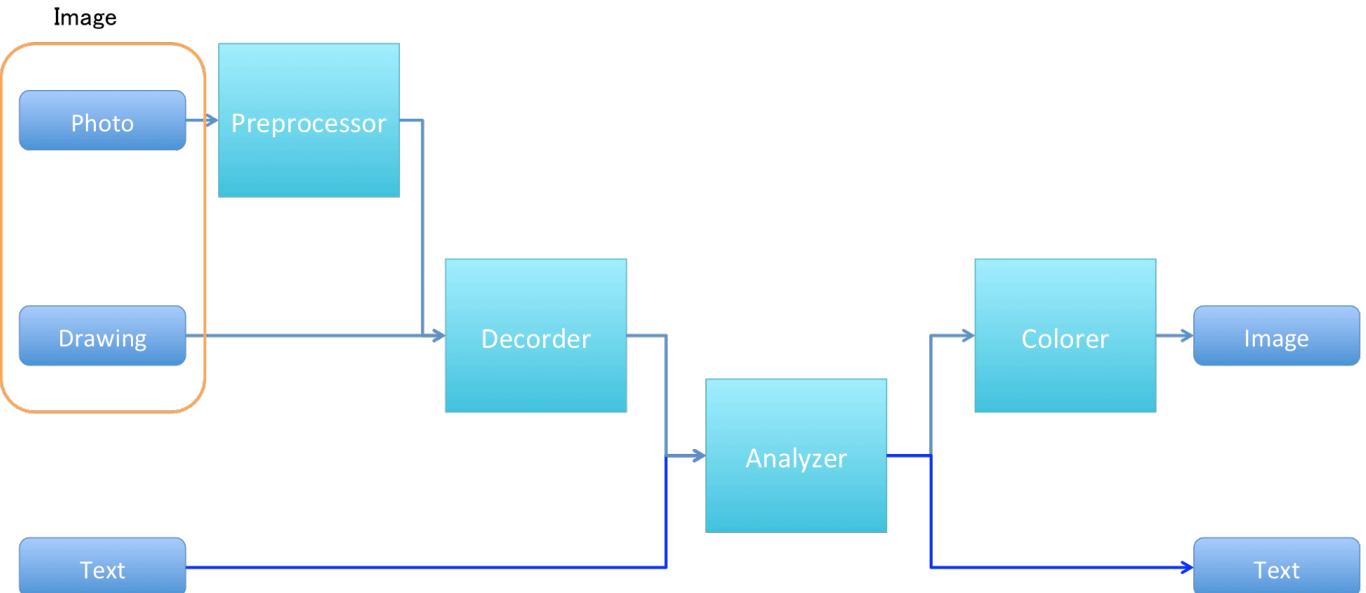


図 8: アプリケーションの構成図

6.1 Preprocessor

写真を入力された時の前処理を行っている。Preprocessor の概要を図 9 に示した。

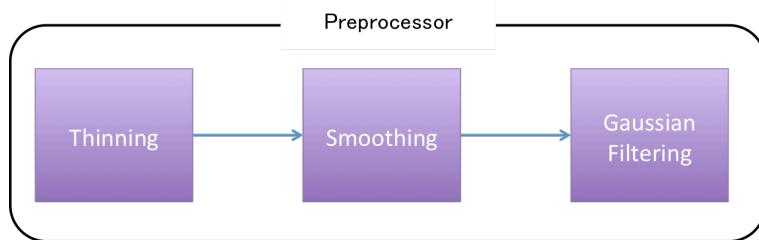


図 9: Preprocessor の概要

前処理では、まず描かれた線画の細線化を行う。これは、描かれた線に幅がある場合を想定したものである。細線化の処理は、参考 URL[11] を元に作成した。

細線化を行った画像に対し、さらに smoothing の処理を施す。つまり、線を滑らかにする処理である。これは、線の歪みやエイリアシングを除くために行った。手法としては、リスト 3 の OpenCV の findContours 関数を用いて線図の辺を点の集合として取得し、隣り合う点を直線近似して描画するという方法を用いた。

リスト 3: findContours 関数

```
1 void findContours(Mat image, List<MatOfPoint> contours, Mat hierarchy, int mode, int method)
```

その後、OpenCV のガウシアンフィルタを掛けて、さらに平坦化を行っている。

以上が Decoder の動作である。

6.2 Decoder

入力された画像を Analyzer の入力に合うように、テキストデータとして出力する役を担う。これによって、テキストデータを解釈するときに使う Analyzer をそのまま用いて線図の解釈が行える。Decoder の構成を図 10 に示した。

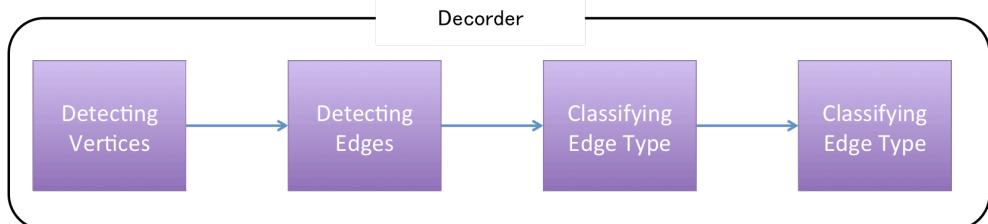


図 10: Decoder の概要

6.2.1 頂点の取得

まずは、線画の頂点を取得する必要がある。これは、入力が画像がノイズが少なく、線が細いものであれば、特徴点を抽出する OpenCV の API に適切な閾値を与えることで取得できる。用いた API は、リスト 4 の goodFeaturesToTrack 関数である。閾値は、実験を通して適切と思われる値に経験から設定した。この際に、写真であれば Harris 検出器を用いたほうが精度が向上したため、そちらを用いている。

リスト 4: goodFeaturesToTrack 関数

```
1 void goodFeaturesToTrack(Mat image, MatOfPoint corners, int maxCorners, double qualityLevel, double minDistance)
2 void goodFeaturesToTrack(Mat image, MatOfPoint corners, int maxCorners, double qualityLevel, double minDistance, Mat mask, int blockSize, boolean useHarrisDetector, double k)
```

6.2.2 辺の取得

次に行うのは、取得したどの頂点とどの頂点が接続しているかを取得することである。まず、リスト 3 の findContours 関数を用いて線画の境界を抽出する。すると、図 11 のように、一番外側や、境界が点の集合として取得できる。つまり、findContours 関数で得られるのは、「点の集合」のリストである。



図 11: 境界の様子

これによって、得られたリストを順に探索していき、前節で取得しておいた頂点のどれかと一致した場合は、次にきた頂点との対応を記録する。これがそのまま辺となるのである。厳密には、境界は線画の辺の外側をなぞるように取得されるため、境界の上に頂点は存在しない。そのため、境界に十分近い頂点があれば、辺の上にあると判定している。

また、この際に、一番外側の境界の情報を元に、外側の境界に属する辺を記録しておくことで、外側の辺を Analyzer での制約条件に加えている。

得られた頂点の対応は、後の Colorer で用いるために、位置情報を含めて保持しておく。

6.2.3 辺の分類

ここまで、頂点と辺の取得が完了している。問題は、辺の分類である。Analyzer は参考文献 [1] の平賀先生のページで示されるラベル情報を入力としてとるため、L, Y, A, T の分類と、頂点の入力順を揃えるという 2 つを行わなければならない。頂点の分類と入力順を図 1 に従う。

まず、頂点の入力順を揃える。これは、構成する 3 つの頂点を取り出して、それらのベクトルを計算し、2 次元の外積を求めることで、どちら向きに並んでいるかを判定することで行う。例えば、図 12 のように 3 点 A,B,C がある場合を考える。

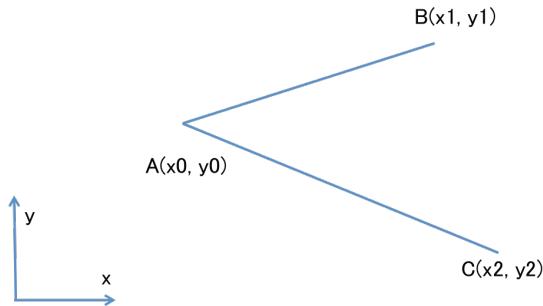


図 12: 2 次元の外積

このときは、 \vec{AB}, \vec{AC} を求め、それらの外積を以下のように定義する。

$$\begin{aligned}\vec{AB} \times \vec{AC} &= (x_1 - x_0, y_1 - y_0) \times (x_2 - x_0, y_2 - y_0) \\ &= (x_1 - x_0) * (y_2 - y_0) - (y_1 - y_0) * (x_2 - x_0)\end{aligned}\quad (1)$$

こうすると、辺 AB と辺 AC が図 12 のようなときに外積が正、辺 AB と辺 AC が逆の位置関係にあるときに負、辺 AB と辺 AC が一直線状になる場合は 0 となる。ただし、これは、図 12 のように、上向きが y 座標となっている場合である。OpenCV では、y 軸が下方向を向くため、符号を反対にして考える。以上を用いて、頂点の順番を図 1 に従うように並び替えた。

次は、L, Y, A, T の分類である。まず、L 型は、構成する辺が 2 つしかない。よって、それをそのまま条件として判定を行えば良い。

一方、辺の数が 3 つとなった時は、Y, A, T の分類が必要となる。これは、すでに頂点の並べ替えが終わっているために比較的容易にできる。求めたいのは、図 13 の辺 AC を含まない方向の $\angle BAD$ の大きさである。

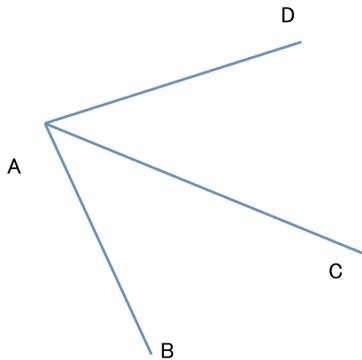


図 13: Y, A, T の分類

ポイントは、ここまで処理で、図のように辺が並んでいることである。これを用いれば、辺 AB と辺 AC, 辺 AC と辺 AD の間の角をそれぞれ求めて和を取り、 $360^\circ (2\pi)$ から引けば結果が求まることがわかる。 $\vec{AB} = (x_0, y_0), \vec{AC} = (x_1, y_1)$ とする、

辺 AB と辺 AC の間の角 θ は、以下の式で求められる。

$$\theta = \arccos \frac{\overrightarrow{AB} \times \overrightarrow{AC}}{|\overrightarrow{AB}| |\overrightarrow{AC}|} \quad (2)$$

ただし、 $0 \leq \theta \leq \pi$

これによって、Y型やT型でも最大角が求まり、分類が行える。

また、辺の数が1つあるいは4つ以上の時は、今回のアルゴリズムでは判定できないか、不可能图形である。よって、その場合はそれを通知する。

最後に、これらを図1で示したようなテキスト情報に変換し、Analyzerに入力する。以上がDecorderの動作である。

6.3 Analyzer

Analyzerは、入力されたテキストデータを元に解釈を行う。入力されるデータは、

1. 図1のようなラベリング情報

2. 最も外側の辺の集合

の2つである。

6.3.1 頂点と辺の登録

まずは、ラベリング情報から、頂点の集合を用意する。頂点は、図1のように入力が与えられたとき、2目につくるもののみ集合に加える。例えば、「L A B C」と与えられた時は、Aを保存する。これを以降では「基準点」ということにする。基準点に対して、入力の1つの引数によって、L, Y, A, Tの型が当てはめられる。初めは、その型でありうる全てのラベルを、対応する頂点に与える。ラベルは図14のようになっている。

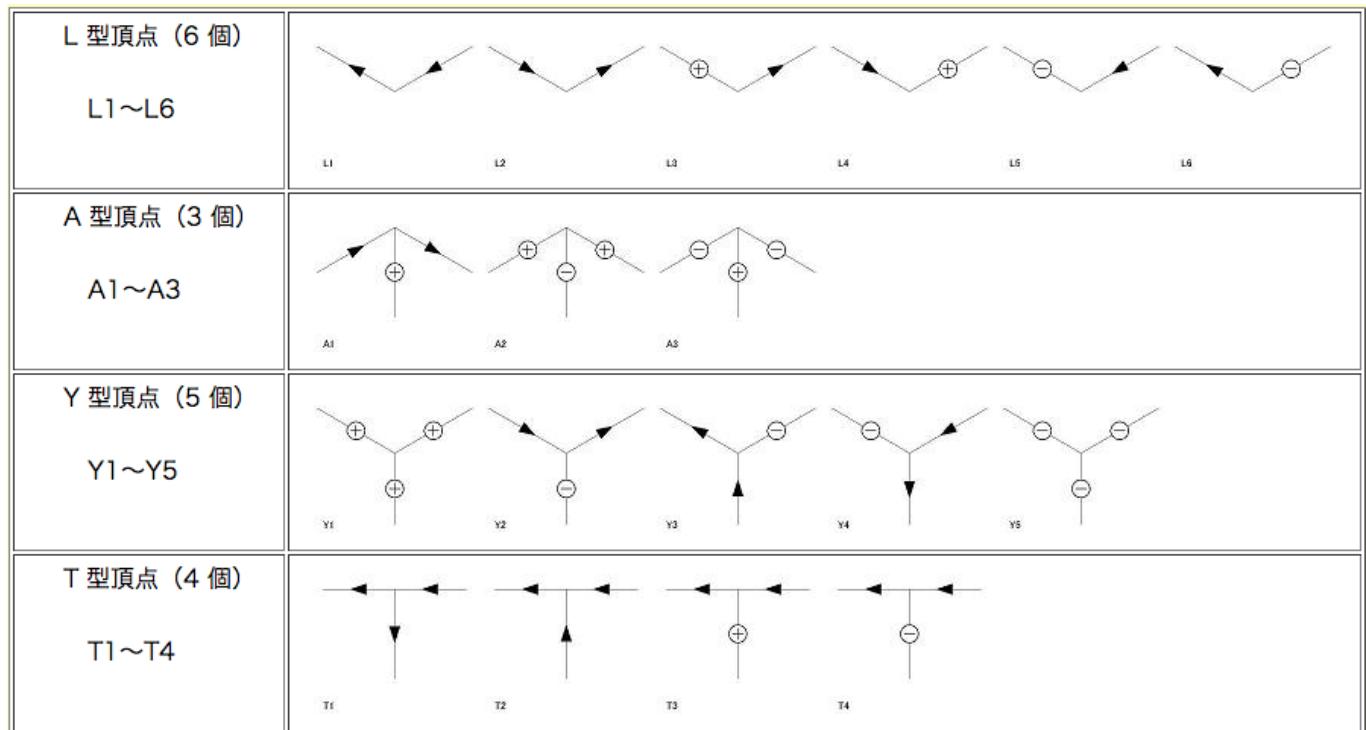


図14: L, Y, A, Tのさらなる分類とラベル ([1]より引用)

以上で、基準点の集合が形成される。

次に、辺の集合を用意する。辺の集合は、入力の引数の2番目の基準点と、3番目以降の頂点を対応付けることで行う。例えば、「L A B C」と与えられたときは、辺 AB, 辺 AC の2つを集合に加える。この際、すでに同じ辺（辺 AB と辺 BA は同じと

見なす)があれば、集合に登録しない。辺も初めは、 $+, -, \rightarrow, \leftarrow$ (\rightarrow は基準点から他の頂点の方向、 \leftarrow は他の頂点から基準点への方向)の4つのラベルすべてを候補として与えておく。

次に、最も外側の辺の集合の情報が与えられた場合は、それらの辺の候補を $<-, ->$ の2つに絞る。例えば、「A B(改行)C D」と与えられた場合は、辺ABと辺CDは外側の辺として、候補を減らすことができる。

以上で、基準点と辺の情報を入力データから全て抽出できることになる。

6.3.2 候補の決定

次は、候補の決定、つまり線画の解釈である。これは、前節で用意した基準点の集合と辺の集合の条件を見比べて、候補を絞っていくことで行う。具体的な流れは以下のとおりである。

1. 基準点の集合に対して、それぞれの基準点に接続している辺のラベルで、基準点の型に当てはまらないものは削除する
例えば、基準点AがT型であり、頂点A,B,C,Dの順で構成されているとする。このとき、辺ABは、図14より、左向きの矢印のみ当てはまるから、他の候補 $(+, -, \rightarrow)$ は削除される。同様に、辺ADも、左向きの矢印のみ当てはまる。
2. 反対に、基準点に接続している辺のラベルを見て、当てはまらない型を基準点の候補から削除する
例えば、基準点AがT型であり、頂点A,B,C,Dの順で構成されているとする。このとき、辺ACの候補が $+, -$ のいずれかであれば、図14より、T1,T2は基準点Aの型の候補から削除される。
3. 以上を更新がなくなるまで繰り返す

これによって、それぞれの辺が接続している2つの基準点の制約によって、少しずつ候補が絞られていき、削除が全く行われなくなった時点で終了する。このように、制約が伝播していくアルゴリズムを制約伝播アルゴリズムという。

全ての辺に対して、候補が一つに絞られていればそれが解となり、どれか一つの辺でも複数の解があれば更なる探索が必要であり、どれか一つでも候補がなくなってしまえば不可能图形である。よって、結果は辺の集合が保持している。テキストとして結果を出力する場合は、全ての辺の候補を辺の集合に列挙させれば良い。

以上がAnalyzerの動作である。

6.4 Colorer

画像が入力され、候補が一つに絞られる場合は、画像に結果を書き込んで見やすくする役割を持つのがColorerである。そのアルゴリズムは単純で、Decorderから出力された端点の位置座標をもつ辺の集合と、Analyzerから出力された候補を保持した辺の集合を見比べることで行える。

まず、Analyzerからの辺の集合を見て、それに相当する辺をDecorderからの辺の集合から取り出す。辺の候補が $+, -$ のいずれかの場合は、2つの端点の中点にその文字を書き出す。一方、矢印の場合は、2つの端点の中点から、適切な方向に適切な長さの辺を2つ描くことで矢印を描く。

以上がColorerの動作である。

7 実験方法

アプリケーションの実験を行った。

7.1 テキスト入力による実験

まず、テキスト入力による実験を行った。これは、図2を想定し、リスト1、リスト2を入力することで行った。

また、テキスト入力による実験は、画像による入力の実験と同じ解析器を用いるため、画像による入力の実験で正しく動作していることは確認できた。

7.2 描画ソフトによる画像の入力に依る実験

描画ソフトで作成した画像を入力して解析を行った。入力した画像は図 15 に示した。いずれもパワーポイントで作成し、ノイズが少なく、線が 1pt の幅を持つ図形である。図 15(c) は少しノイズのある図形を上部に配置している。これらは全て、5.1 節で触れたサンプル画像であり、アプリをインストールした際に組み込まれている。

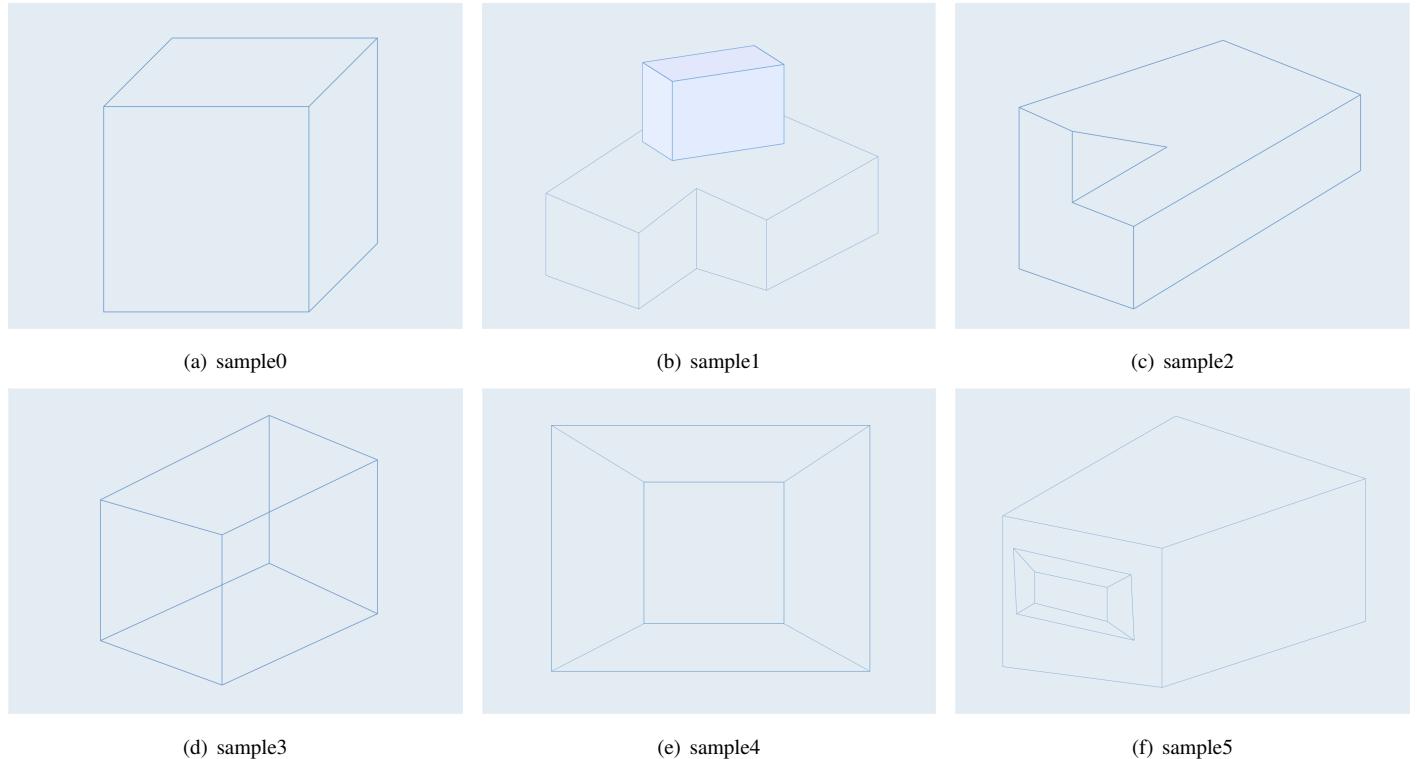


図 15: 入力した画像

7.3 写真画像の入力に依る実験

撮影した画像を入力して解析を行った。入力した写真は、図 16 に示した。

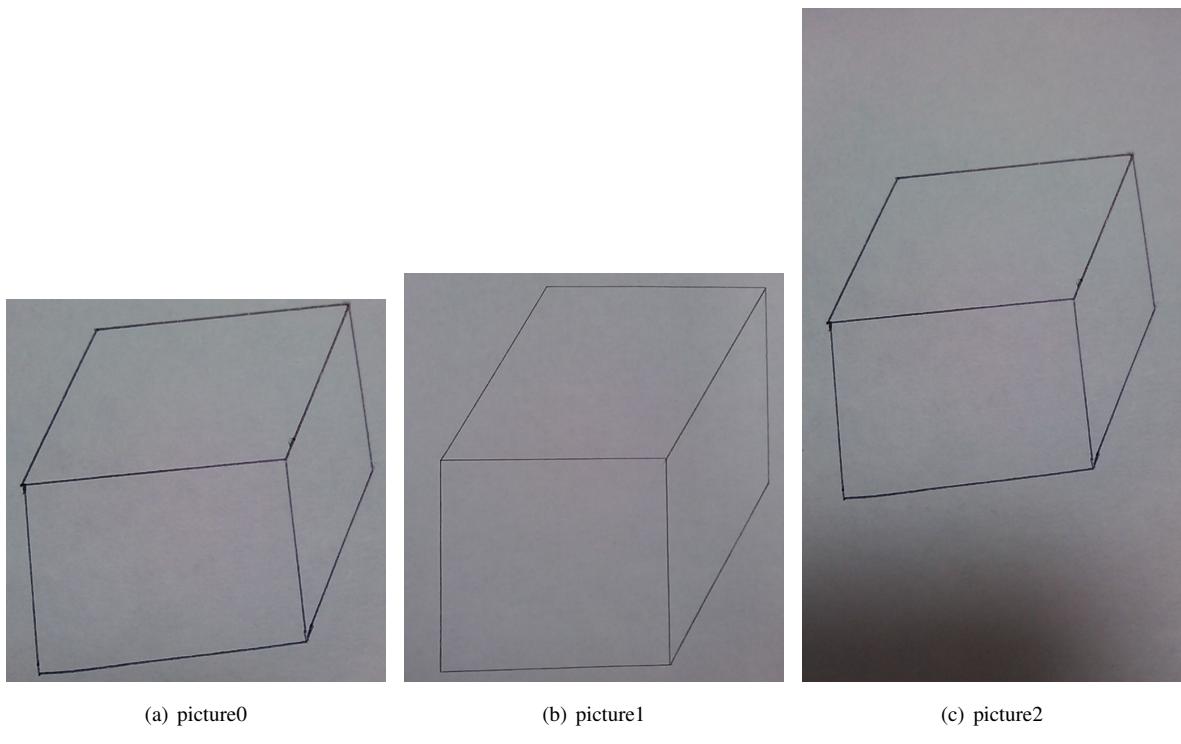


図 16: 入力した写真画像

8 実験結果

8.1 テキスト入力による実験の結果

テキストを入力した結果を図 17 に示した .



図 17: テキスト入力の結果

出力結果を見ると、リスト 5 のようになる。ただし、「A-F: →」などは、A から F に矢印が向いているという意味になる。結果が一意に定まっているとわかる。

リスト 5: テキスト入力の結果

```
1 A-F: ->,
2 A-B: <-,
3 C-B: ->,
4 C-D: <-,
5 E-D: ->,
6 E-F: <-,
7 B-G: +,
8 D-G: +,
9 F-G: +,
```

8.2 描画ソフトによる画像の入力に依る実験の結果

描画ソフトによる画像を入力した結果を図 18 に示した。図 18(a) の図形と、図 2 の図形を見比べると、頂点の対応はいずれに入るものの、前節のテキスト入力に依る実験の結果が正しいことがわかる。また、図 18(b) のように少しノイズがある図形も、特徴点抽出の閾値を調整することで正しく認識できるようになっている。

図 18(c) は不可能図形であるが、制約伝播アルゴリズムによって、「この図形は不可能図形です」と示されている。また、図 18(d) のネッカーキューブは、頂点 D や F が 4 つの辺を持つために、「不可能図形であるか入力が不正です」と示されている。また、当初、4 つの辺のときに、そのうちの 3 つの辺に対してのみ解釈をするようになっていた場合（条件の設定間違をしていた）では、ネッカーキューブに対しても、いくつかの解の候補が表示された。

図 18(e) の図形は、本来は、内側に凹んでいる図形としても認識できる。しかし、この図形が浮いているものとみなし、境界の条件を決めたために、内側が盛り上がった図形として認識されている。一方で、図 18(f) では、図 18(e) の内側が凹んでいるか、盛り上がっているかの 2 通りの解釈ができるため、解の候補をテキストで表示している。

ただし、Android では画像の読み込みを個々の端末の解像度やメモリにあわせて調整するため、同じ結果が得られない場合があることがわかった。

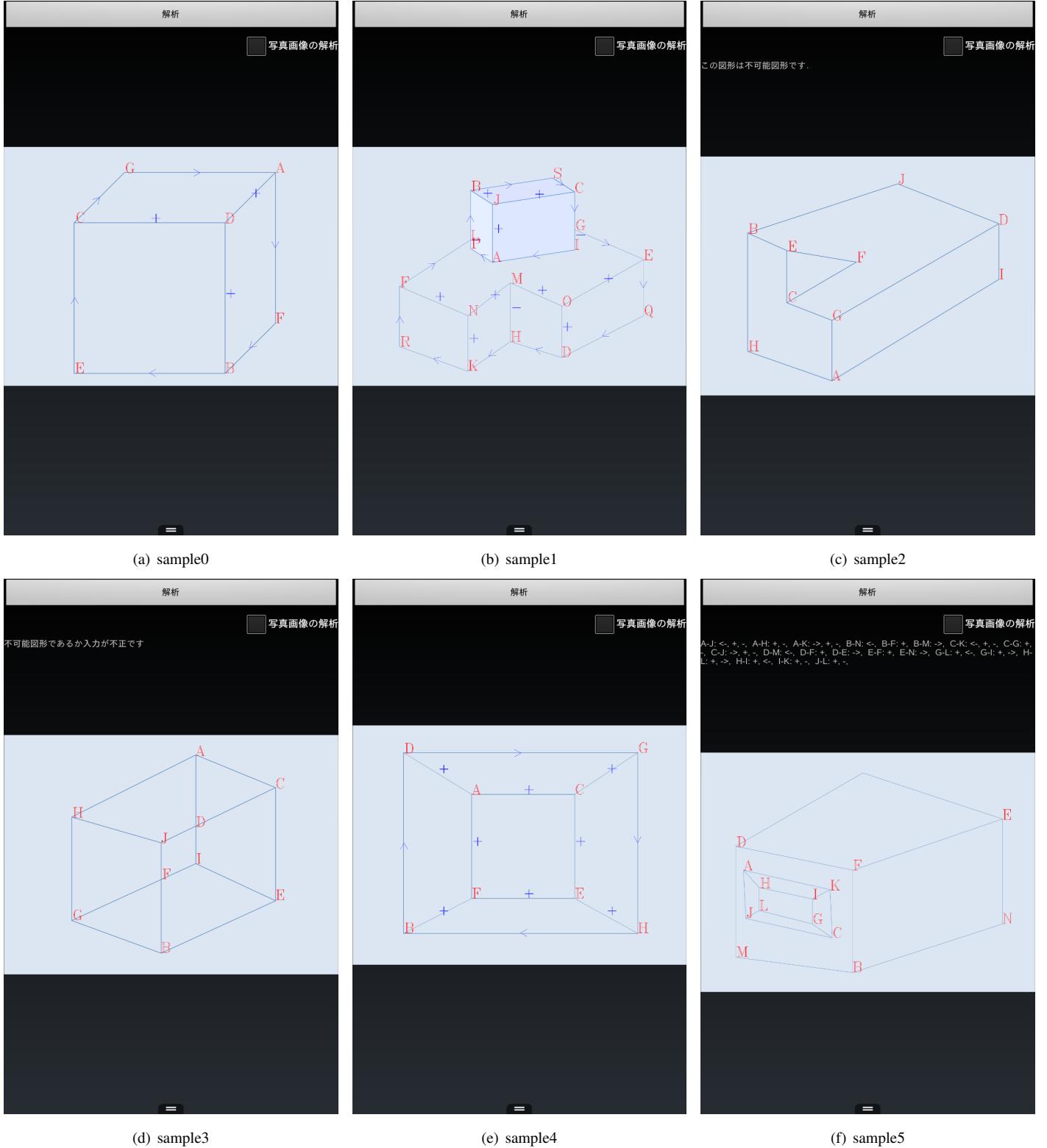


図 18: 入力した画像の解釈結果

8.3 写真画像の入力に依る実験の結果

写真画像を入力した結果を図 19 に示した。picture0 と picture1 は下処理を行った場合と、行ってない場合の両方を示した。図 19(a) と図 19(b) を見比べると、特徴点抽出がわずかに向上している事がわかる。しかし、元の画像で厚みがあった部分や曲がった部分などで特徴点が検出されてしまっている。

図 19(c) と図 19(d) を見比べると、下処理によって、斜めに検出されていた特徴点が全く検出されなくなり向上しているが、下処理の前にはなかった横向きの線に対して、特徴点が検出されてしまっているため正しく動作していない。

さらに、図 19(e)に関しては、影に特徴点が集中し、結果の出力に 20 秒程度の時間がかかった。

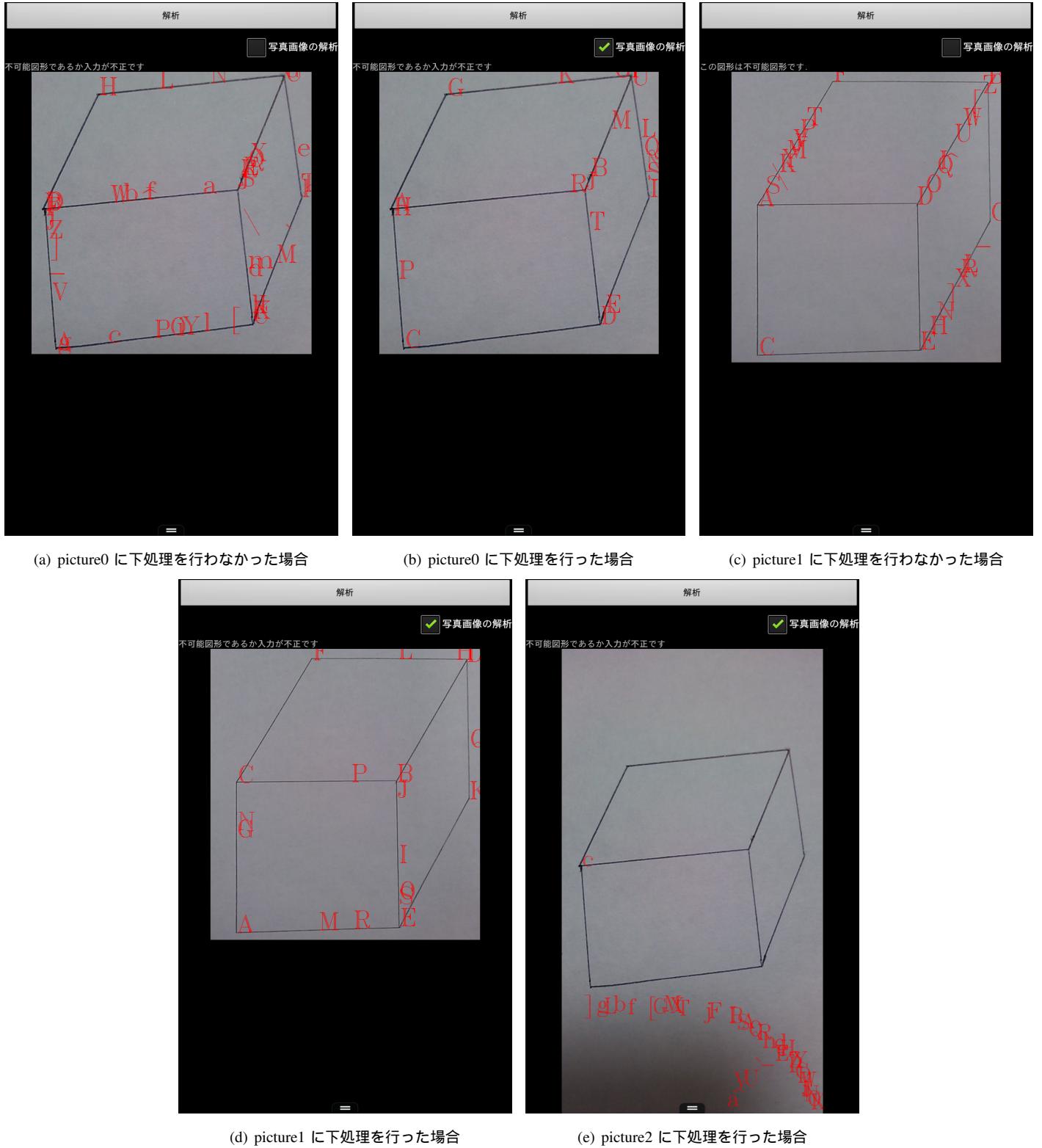


図 19: 入力した写真画像の解釈結果

9 考察

9.1 テキスト入力による実験の考察

テキスト入力に対する結果と、画像による入力結果、参考文献 [1] をあわせて考えると、テキスト入力による解釈は正しく動作していると考えられる。しかし、今回のアルゴリズムでは、解が一意に定まらなかった場合に、頂点と辺によって候補を絞つ

ていくだけであったため、全体としてみればそれぞれの制約条件から結果が一意に定まる場合を考慮できていない。

複雑な図形でも、頂点情報を正しく入力できれば正しい結果が得られると推察される。

9.2 描画ソフトによる画像の入力に依る実験の考察

描画ソフトに依る画像の入力に対しては、実験結果から正しく動作することがわかる。ただし、Androidでは画像の読み込みを個々の端末の解像度やメモリにあわせて調整するため、同じ結果が得られない場合があることがわかった。これに対しては、写真画像と同様にした処理を行ってから画像を入力することで解決できると考えられる。

ただし、テストした環境では正しく解釈されたことが結果からわかる。特に図18(b)のような複雑な図形に対しても解釈が行っている。この図は、当初、上部の図形が滑らかでなかったために、特徴点が角でないところで検出されるなどの問題があつたが、経験から設定した閾値によって、正しく解釈されるようになっている。

また、不可能図形に対しても、その旨が表示できており、アルゴリズムが正しく動作されていることがわかる。一方、ネッカーキューブのような図では、結果でも触れたように、接続している辺が3つよりも多くても、そのうちの3つに対して処理を行うと、解釈の候補が幾つか表示された。本来、このような処理にならないようにするべきで、結果に示したように「不可能図形であるか入力が不正です」と表示されるべきであった。この失敗から得られた結果について考えてみる。この結果を、図20に示した。

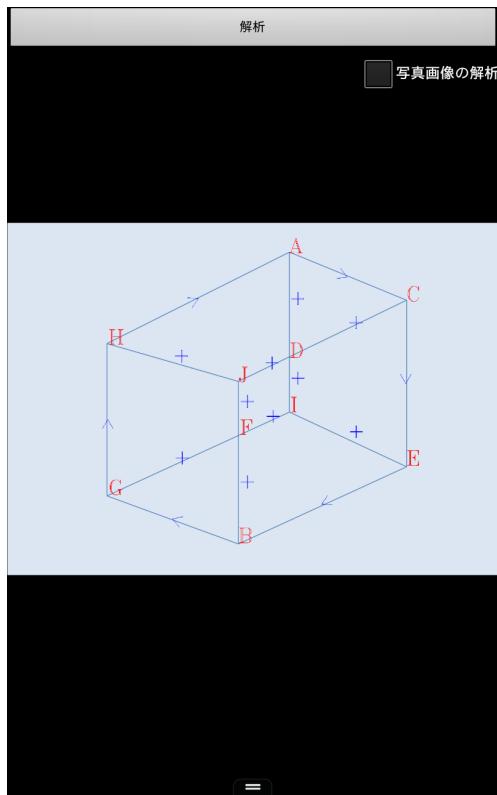


図20: ネッカーキューブの解釈

この結果から、辺AI,EI,GIによる角と、辺HJ,CJ,BJによる角のどちらもが飛び出して見えていることがわかる。これは、図18(e)のように境界による条件で、内側が飛び出した図形と判断されるためと推察される。つまり、このアルゴリズムでは、人間と同様にどちらの角も飛び出しているように解釈できるというわけである。また、境界の条件をなくせば、内側が凹んでいる図形と解釈されることが図18(f)の結果から推察される。このことを反対にアルゴリズムの視点から捉えると、人間はこのようなどちらの解釈もできる図形に対して、同様のアルゴリズムにしたがって解釈を行っている可能性が考えられる。

また、画像からの解釈でもテキスト入力の際に用いた解析器を用いているため、正しく動作していることからテキスト入力による解釈の正しさも示された。

9.3 写真画像の入力による実験の考察

写真画像の入力による実験では、正しい結果が得られなかった。その原因は、実験に使用した図から、

- 細線化の際にエイリアシングを作ってしまっている
- 影が図形として解釈される
- 線に膨らみがある場合、細線化を行うとよりそれが際立ってしまう

などが挙げられる。一方で、細線化と smoothing によって、無駄な特徴点が検出されないように改善されている図もあり、方針としては正しいと考える。現在は直線近似によって、線を滑らかにし、そこから角を検出しているが、あらかじめ角の検出に特化したアルゴリズムを用いて角を正確に抽出できれば、それらをつなぐことでよりなめらかな線図を出力できると推察される。

10 結論

本課題では、テキストや画像からの線画解析を試みた。その結果、テキスト入力に関しては正しい結果を返し、画像でもノイズの少ないもの解釈であれば、正常に処理を行えることが確認できた。ただし、写真からの認識は、線画へと下処理をする部分で、画像処理による知識が乏しく、正しく認識することが出来ないという結果になってしまったことが反省としてあげられる。提出締め切りまでさらに検討してみようと考えている。

それらの手法では、OpenCV という画像処理ライブラリに依る部分が多く、初めて使用したもの、その便利さがよくわかった。また、アルゴリズムでは、参考文献 [1] の通りに実装することで実現ができた。他の参考文献からも、様々な情報を得ることが出来た。感謝を示したい。

参考文献

- [1] 『レポート課題：「制約伝播：線画のラベリング」資料』, <http://www.slis.tsukuba.ac.jp/~hiraga.yuzuru.gf/AI/label.shtml>
- [2] 『OpenCV』, <http://opencv.org/>
- [3] 『Google Play - OpenCV Manager』, <https://play.google.com/store/apps/details?id=org.opencv.engine&hl=ja>
- [4] 『DropBox』, https://www.dropbox.com/login?lhs_type=anywhere
- [5] 『Android File Transfer』, <http://android-file-transfer.softonic.jp/mac>
- [6] 『アストロファイルマネージャー』, <https://play.google.com/store/apps/details?id=com.metago.astro&hl=ja>
- [7] 『Mac (OS X Mountain Lion) に Android の開発環境を構築』, <http://blog.wadous.com/it/android/424.html>
- [8] 『OpenCV for android 2.4.3』, <http://d.hatena.ne.jp/Kazzz/20121128/p1>
- [9] 『【Android】ファイル選択ダイアログを作成』, http://alldaysyu-ya.blogspot.jp/2013/09/android_12.html
- [10] 『OpenCV.jp』, <http://opencv.jp/>
- [11] 『細線化』, <http://www.eml.ele.cst.nihon-u.ac.jp/~momma/wiki/wiki.cgi/OpenCV/%E7%B4%B0%E7%B7%9A%E5%8C%96.html#h7>
- [12] 『cagylogic』, <http://www.cagylogic.com/archives/2009/03/03000100.php>
- [13] 『基礎の基礎編 その1 内積と外積の使い方』, http://marupeke296.com/COL_Basic_No1_InnerAndOuterProduct.html
- [14] 伊庭 齊志:『人工知能と人工生命の基礎』, オーム社 (2013.5.24)