平成 26 年度　修士論文

紙を用いたレスポンスアナライザにおける 2 次元 AR マーカの湾曲量取得

平成 27 年 2 月 13 日

13350945

程　帥

指導教員 三浦 元喜 准教授

九州工業大学大学院 工学府 先端機能システム工学専攻

概 要

学習者の状況をリアルタイムに集約するレスポンスアナライザシステムとして，リモコン型デバイスや小型情報端末を利用したものが一般的である．しかし，これらは充電や機器管理の手間がかかるため，より安価で低管理負荷な方法として学習者個人 ID を示す 2 次元コードを印刷したマーカシートを，教員側のカメラで読み取る手法（AwareResponse）が提案されてきた．従来の AwareResponse 手法では，既存の 2 次元コード読み取り技術を用いていたため，平面マーカシートの「向き」と「位置」を判別することしかできなかった．本稿ではマーカシート素材としての紙の特性を活かした「曲げ」や「歪み」の程度を読み取り，学習者の反応を拡張する方法について述べる．また，既存の 2 次元コード読み取り技術を改良して「曲げ」や「歪み」を取得する方法について述べ，その認識性能について実験を行いアルゴリズムの有効性を確認した．

目 次

第1章　序論

本論文では AR マーカシートを用いた簡易なレスポンスアナライザシステムについて，その機能を高めるための方法について論ずる．第一章では本テーマを取り巻く背景と世の中に存在する多数の学習者を対象とするインタラクティブな学習教育環境を構築するためのレスポンスアナライザシステムの紹介，そして今回提案する湾曲 AR マーカの認識手法の開発やシステム構築を含めた本研究の目的を説明する．

1.1　背景

双方向性の高い，いわゆるインタラクティブな学習教育環境を実現することを目的として，これまで学習者にデバイスを配布し，教師に情報を集約するレスポンスアナライザシステムが数多く提案・実装されている．Liu らは，2003 年に日々の学習活動を無線技術によって拡張した教室を提案している [1]．Roschelle らは，2002 年に無線でインターネットに接続するデバイスの可能性について言及している [2]．これらのアプローチは無線接続技術や端末の小型化によって促進されてきた．また教室環境においては，クリッカーと呼ばれる学習者端末の導入により，インタラクティブな学習が可能になることが示されてきており [3]，200 名を越える受講者を有する大講義においてもその有用性や効果が確認されている [4]．

これらのレスポンスアナライザシステムの多くは個人の反応を送信・集約するために，リモコン風のキーパッド（図 1.1）や，PDA や携帯電話，スマートフォン等のハンドヘルド機，タブレットや PC 等の情報機器・端末といった電子デバイスを利用している．これらの電子デバイスは近年，低価格化がすすみ一般に普及しているものもある．しかし実際に授業を行う際には公平性の観点から，これらの電子デバイスを指導側が事前準備・提供することが多い．そのような場合，指導側（教師や補助員）がそれらの電子デバイスを授業時間のみ提供し，授業後に再度回収して管理するといった手間が必要になる．こ

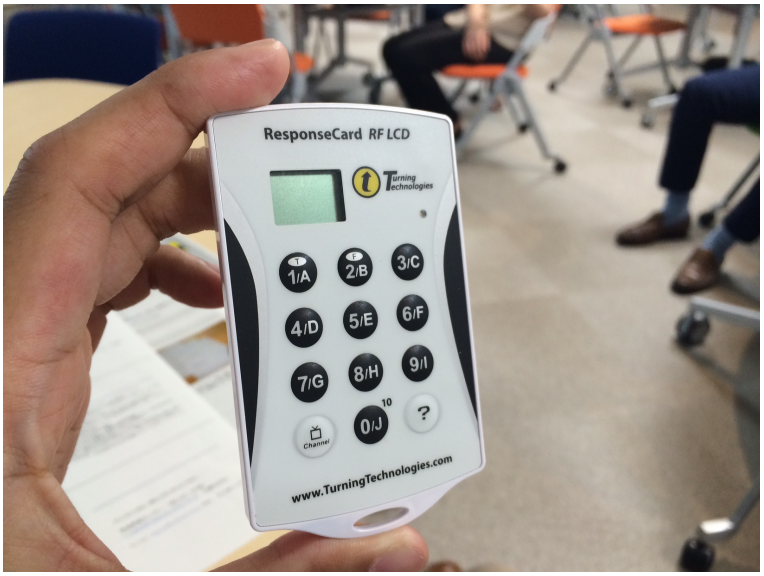

図 1.1: レスポンスアナライザのリモコン

うした授業毎の「配布」と「回収」の手間は，授業を受講する学習者の数に比例するため，大教室・多人数講義になるほど，レスポンスアナライザを使用したインタラクティブな授業実現のためのハードルは高くなるといえる．もしこうした電子デバイスを用いずに，個々の学習者のレスポンスを集約することができれば，デバイスの「配布」と「回収」や「管理」に伴う手間を大幅に軽減できる．

1.2　紙を用いたレスポンスアナライザシステム

こうした背景から，学習者用の装置として電子デバイスを用いずに，安価な「紙」のみを利用したレスポンスアナライザとして，AwareResponse 方式 [5] が提案されている．AwareResponse 方式では，個人を識別するための ID 情報を印刷したマーカシートを事前に学習者に配布しておき，講義中に教師が学習者からの反応を得たいときに，学習者にマーカシートを教員側に設置されたカメラに写るように掲げるよう指示する（図 1.2）．例えば，学習者はマーカシートを掲げるとき，マーカシートのどの辺を上にして掲げるかによって，４つの選択肢から１つを選ぶ問題について回答する．また，マーカシートの向き（方角）によって，共有スクリーンへのポインティング操作を行う方法について

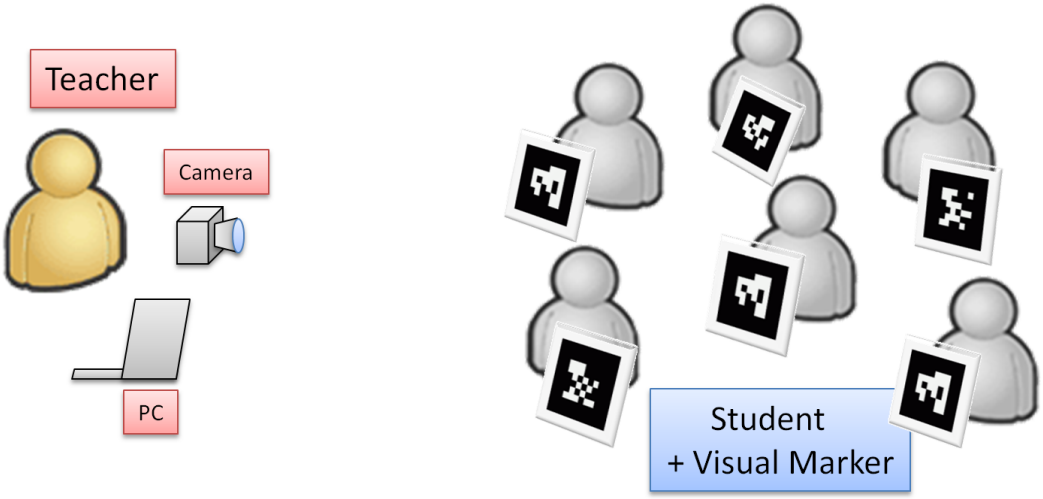

図 1.2: AwareResponse 方式

も示している．また同時期に，Microsoft Research India の Cross らは qCard[6] という類似の方式について，認識精度や認識スピード，そしてクリッカーと比べたコストについて報告している．これらの研究動向から，多数の学習者の反応を低コストで，かつ簡便な方法で取得する手法やシステムの重要性がうかがえる．

1.3　紙を用いたレスポンスアナライザシステムの問題点

紙を用いたレスポンスアナライザシステムは低コストかつ簡便な方法であるが，問題点として，学習者側から送信可能な情報を増やすことが難しいことが挙げられる．

デバイスを用いる方式であれば，ボタンの数を増やしたり，多様な押し方を許容するなどの工夫により，学習者側から送信可能な情報を比較的容易に増やすことが可能である．しかし，紙を用いたレスポンスアナライザシステムでは，ARToolkit[7] や ARToolkitPlus[8] をはじめとする 2 次元マーカ認識技術を用いているため，基本的に，マーカーシートの状態から取得できるものは，マーカーシートの ID と，カメラに対する相対位置，および姿勢情報（Roll / Pitch / Yaw 角，図 1.3）の情報のみとなる．これを解決するためには，学習者に複数のマーカーシートを扱わせることによる方法も考えられるが，紙の管理の手間が増加すると，学習行為への負荷がかかってしまうため，望ましくない．

1.4　紙を用いたレスポンスアナライザシステムの拡張

従来の紙を用いたレスポンスアナライザシステムである，AwareResponse 方式 [5] や qCard 方式 [6] では，学習者 ID が印刷されたマーカシートを一枚の「板」として利用し，印刷されたマーカをカメラで認識したときの回転角度 (Roll) で回答を判別したり，また Pitch 角や Yaw 角（図 1.3）によって，共有スクリーンへのポインティング操作（図 1.4）を行っている．すなわち，2 次元マーカが歪まないことを前提とした操作しか提供していない．

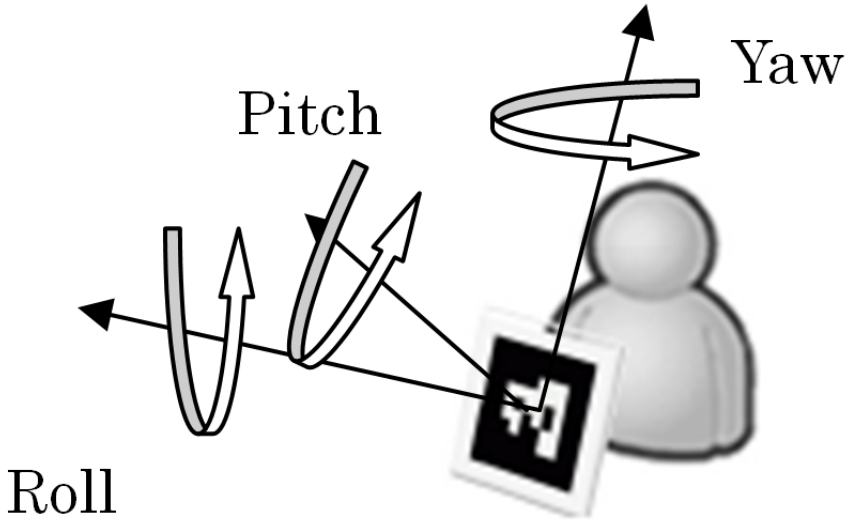

図 1.3: AwareResponse 方式における，マーカシートの姿勢情報

しかし，2 次元マーカは紙に印刷されることが多いことから，紙ならではの特性である「湾曲」や「歪み」を積極的に用いることで，マーカーシートの枚数を増やすことなく，学習者側から送信可能な情報を増やすことが可能になるのではないかと考えた．

1.5　本研究の目的

本研究の目的は，紙を用いたレスポンスアナライザシステムにおいて，学習者側から送信可能な情報を増やすため，マーカーシートの湾曲量を取得する方法を開発し，実際

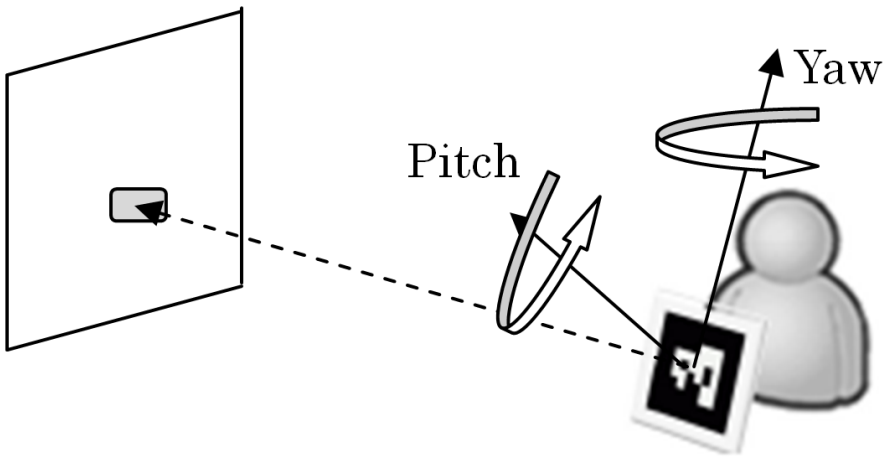

図 1.4: AwareResponse 方式におけるポインティング手法

のシステムに利用可能であるかどうかを検討することである．マーカーシートの湾曲量が取得できると，例えば学習者が自分の回答に自信があるときに，マーカーシートを湾曲させることによって，自信の度合いを教師に明示的に伝えるといったことが可能となる．また，こうした情報を長期的に蓄積することによって，学習者の性格や特性の傾向が明確化され，教師の判断を助けることが可能になると考えられる．

第2章　提案手法

本章では，紙を用いたレスポンスアナライザシステムにおける，学習者側から送信可能な情報を増やすための，マーカーシートの湾曲量を取得する方法を考えるうえでの検討事項と，具体的な方法について述べる．

2.1　設計方針

本節では，マーカーシートの湾曲量を取得するにあたっての検討事項と設計方針について述べる．

我々は，マーカーシートの湾曲量を取得するにあたって，以下の 3 つの設計方針を掲げた．

1. システムを簡素化するため，カメラは一般的に普及している Web カメラを利用する．
2. 湾曲量を取得するためのアルゴリズムは，できるだけ計算量をかけずに済むものを選ぶ．
3. マーカシートの 3 次元形状を正確に取得することが目的ではなく，ユーザ（学習者）の操作意図を反映した，マーカーシートの湾曲量が取得できればよいものとする．

1 つめの設計方針に関しては，近年，2 つ以上のカメラを用いたシステム（いわゆるステレオカメラ）や，Microsoft KINECT に代表される深度情報を取得できるカメラが普及している．これらの方式を用いることによって，マーカーシートの 3 次元形状を直接取得可能である．しかし，これらの方式を用いると，システムの構成が複雑になったり，またシステムを実現するための費用が高くなってしまい，本来，紙を用いたレスポンスアナライザシステムが備える「安価で簡便な方式」という利点が失われてしまう．そのため，単一のカメラのみで動作する方式を検討することにした．単一のカメラのみで動作する

ことの利点として，スマートフォンやタブレットのようなカメラ付きのデバイスを教師側のカメラとして利用することができ，さらなる簡便化につながることが期待できる．

2 つめの設計方針については，紙を用いたレスポンスアナライザシステムの特性により，多数のマーカを同時に読み取る必要があることから，計算量が少ないアルゴリズムを重視することにした．これも，スマートフォンやタブレットのようなデバイスで動作させる場合に，重要な要件であると考えている．

3 つめの設計方針については，1 つめの設計方針とも関連するが，マーカーシートの正確な 3 次元形状を取得することは今回の研究では対象としていない．あくまで，ユーザ（学習者）のシート操作（マーカーシートを曲げる動作）に対応した値が取得できればよいこととする．その理由として，元々の 2 次元マーカの制約から，マーカ部分をカメラから隠してしまうように大幅に湾曲させたマーカーシートは認識したり，ID を検出することが難しいことが挙げられる．そのため，部分的かつ，わずかなマーカシートの湾曲情報を検出することに焦点をあて，従来のマーカ認識アルゴリズムを大幅に書き換えることはしない．

2.2 従来の 2 次元マーカ認識アルゴリズム

本節では，従来の 2 次元マーカ認識アルゴリズムである ARToolkit[7] の認識手法について概説する．

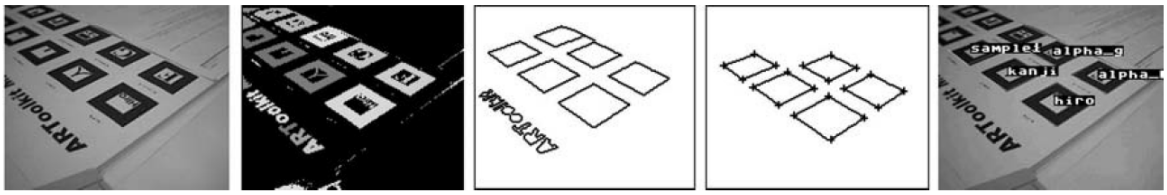

図 2.1: AR マーカを抽出するプロセス ([9, 10] より引用) 左から，(1) オリジナル画像 (2) グレーレベルのしきい値以下の連結コンポーネント (3) 連結コンポーネントにおける外側の枠線 (4) 矩形認識を行い，さらにコーナー点を検出 (5) 検出したマーカにラベルを重畳表示

ARToolkit のマーカは黒枠と，その内部に印刷されたパターンによって構成されている．ARToolkit のマーカ認識は，図 2.1 の手順で行われる．(1) のカメラ画像に対して，グレーレベルのしきい値以下の連結コンポーネントを抽出する (2)．その後，抽出した連結コンポーネントにおける外側の枠線候補を抽出する (3)．抽出した枠線候補に対して，矩形認識を行い，さらにコーナー点を検出する (4)．検出したコーナー点を用いて，射影変換を行い，カメラ画像におけるマーカ ID 部分の画像の歪みを取り除く．そして，ARToolkit では事前に定義しておいたマーカ画像（プロトタイプ）とのパターンマッチを行い，ID と信頼度を計算する．最後に，(1) のオリジナル画像に対して，検出した ID に対応するマーカラベルを重畳表示する．ARToolkitPlus では，マーカ画像のプロトタイプとして，事前に定義された矩形のパターンが用いられる．

2.3 湾曲量の検出

マーカーシートをカメラで撮影したときに，どのような画像になるかの一例を図 2.2 に模式的に示す．このうち，(a) から (d) は，湾曲がない状態の例であり，(e) から (h) は，湾曲がある状態の例を示している．この図で示すように，湾曲したマーカーシートが撮影されると，その特徴はマーカーシートおよび，マーカ（黒枠）の枠線の形状として表れることがわかる．

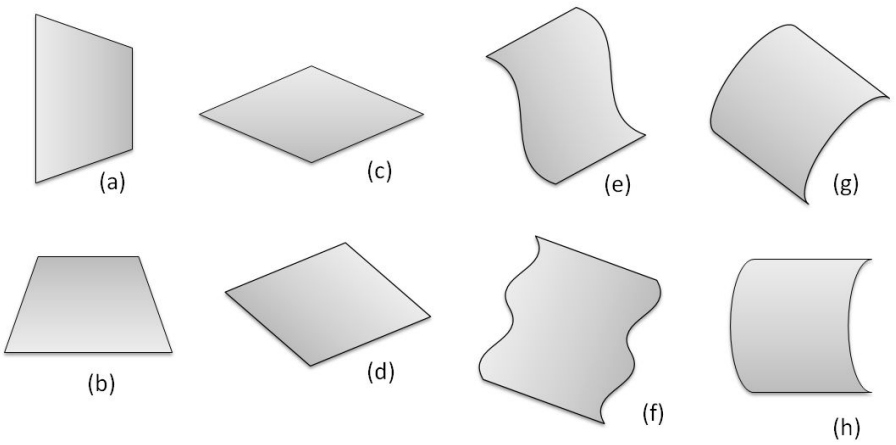

図 2.2: マーカーシートをカメラで撮影したときの画像パターン

当初我々は，マーカのコーナー点の座標から，カメラで撮影・観測されたマーカの 4 辺の長さを求め，その長さの分散を求めることによって，マーカの歪みや湾曲量を簡単に求めることができないかと検討した．しかし，分散を用いた場合には，カメラに対してマーカが正対している場合は，撮影された画像のマーカの 4 辺の長さは等しくなるが，図 2.2(a)(b)(c)(d) のように正対していない場合，マーカーシートは湾曲していないが 4 辺の長さは異なる画像となる．そのため，マーカーシートの湾曲状態を正しく検出することができない．

そこで我々は，マーカのコーナー点の座標に加え，マーカの枠線の形状に対応する黒枠の枠線を構成する座標点を用いて，以下の手順にて計算を行うことにした．

1. 最初に，マーカのコーナー点 (C_0, C_1, C_2, C_3) と，黒枠の 4 辺を構成する座標点を収集する．ただし，黒枠の 4 辺を構成する座標点は，マーカのコーナー点 (C_0, C_1, C_2, C_3) を含まない．また，黒枠の 4 辺を構成する座標点は，必ず 8 方位で隣接するようにする（図 2.3 参照）．

2. 1 つの辺に着目し，マーカのコーナー点を結ぶ線分を考える．

3. 着目している辺を構成する座標点すべてについて，1 点ずつ，この線分との距離を求める．(図 2.3 における点 P を参照)

4. 線分との距離を，すべて足しあわせていく．

5. 同様に，他の 3 辺についても処理を行う．

6. 最後に，距離の合計値を，黒枠の 4 辺を構成する座標点の数で割り，1 点あたりの平均距離を計算し，これを湾曲量とする．平均距離を求めることにより，撮影されたマーカ画像の大きさによる影響を軽減する．

以上の処理により，図 2.2(a)(b)(c)(d) のように，撮影されたマーカ画像における辺が直線である場合は，湾曲量は 0 に近づき，図 2.2(e)(f)(g)(h) のように，辺が曲がっている場合は湾曲量が大きな値となる．

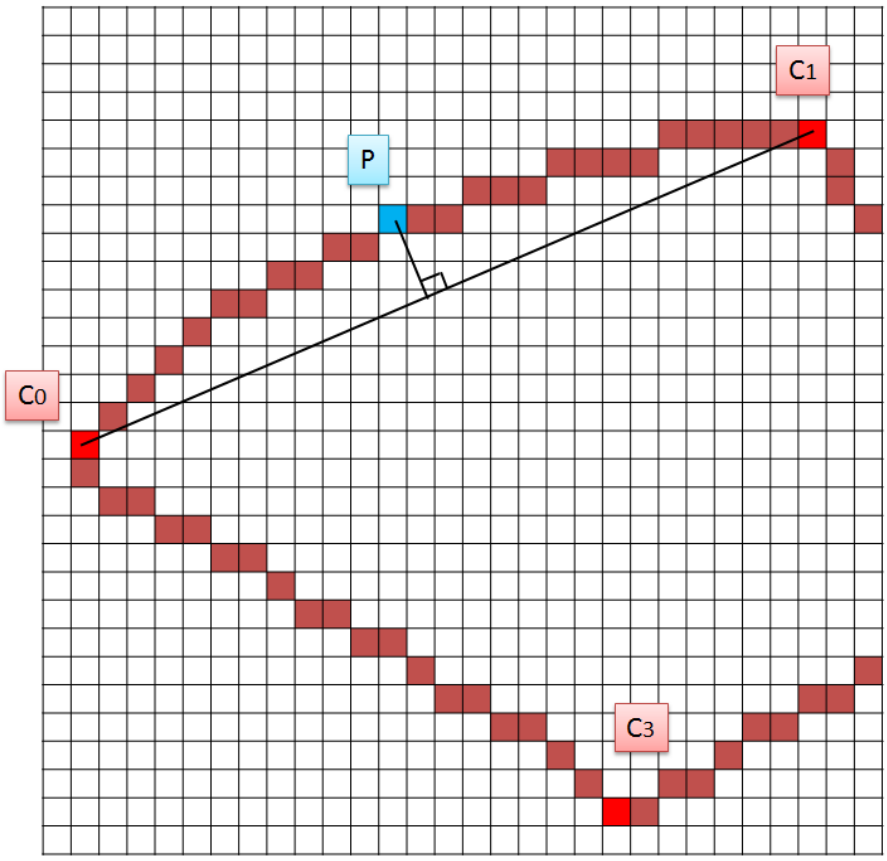

図 2.3: 湾曲量の計算方法

第3章　実装

本章では，提案手法を実現するための実装について述べる．

3.1　NyARToolKit

本研究における提案手法を確認するため，我々はNyARToolkit[1] のJava版を改良して，実験に使用することにした．NyARToolKitとは、ARToolKitを参考に実装し直したARToolKit互換のクラスライブラリであり，ARToolKit/2.72.1の機能を元に、全てのAPIをクラスベースで再実装してあるため，再利用性および拡張性が高い．

また，NyARToolkitには，ARToolKitのオリジナルの機能のほかに，自動閾値検出機能や，独自のシリアルIDマーカシステム（NyARIdMarker[2]）等の拡張機能を実装している。シリアルIDマーカは，大量のマーカシートを作成する必要があるときに，認識用の画像を1つ1つ登録する必要がないことから，本研究が対象とする紙を用いたレスポンスアナライザシステムに適している．また，Javaを用いているが，アルゴリズムが改良されており，マーカの認識速度はオリジナルと同等かそれ以上（JIT有効時）と言われている．このことは，複数のマーカを同時に検出する必要がある今回のような用途についても，処理速度の低下が少なくなることから有効である．

3.1.1　NyARIdMarkerのマーカシステム

NyARToolKitのARマーカのデータ領域仕様には，データ領域を5×5の25セルに分割するModel2から，15×15の225セルに分割するModel7まで，6タイプのマーカシステムが定義されている．

[1] http://nyatla.jp/nyartoolkit/wp/

[2] http://sourceforge.jp/projects/nyartoolkit/docs/standards_document0001/ja/2/standards_document0001.pdf

本研究では，図3.1 に示すように，5×5 の 25 セルに分割する Model2 を採用した．Model2 で同時に利用できるマーカの数は，データドット（図 3.1 の中央，緑の領域）が 9 セルであることから，0 511 の 512 個である．紙を用いたレスポンスアナライザシステムにおいては，同時に受講する学習者の数は高々100 名程度であることや，データドット数が増加すると，マーカを認識するためにマーカーシートを十分カメラに近づける必要があることから，今回は最も ID 数が少ない Model2 を選択した．

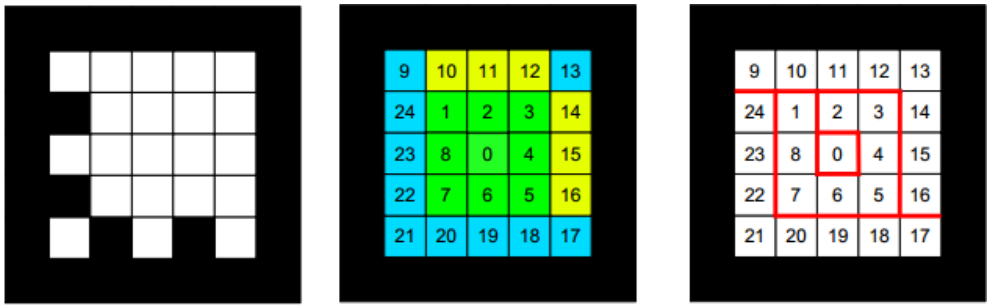

図 3.1: NyARToolKit の AR マーカのデータ領域仕様

3.1.2 NyARToolkit のアルゴリズム

NyARToolkit のアルゴリズムも，基本的には 図 2.1 の手順と同様である．すなわち，(1) ２値化 (2) マーカの黒枠部分の検出 (3) マーカ内部の ID 部分の検出　という順番で，カメラ画像内のすべてのマーカを検出する．２値化に関しては，明点と暗点の PTile 法（P タイル法【PercentileMethod】）を使用している．PTile 法とは，画像の二値化したい領域が全画像の領域に占める割合をパーセント（%）で指定し，二値化する手法である．この方法で求めたしきい値を用いる．すなわち，明点と暗点両側から一定割合の画素を取り除き，その中間値を求める．NyARToolkit では，デフォルトで 15%が選択されている．
2

図 3.2 は AR マーカの角を検出したデータによって，得られた二次元の AR マーカの輪郭線を示す．６回の AR マーカの輪郭線によって，vertex1_index(AR マーカの四頂点の中の任意の一つ頂点の番号) でもらった頂点はランダムである．特に黒枠の初めるゼロ点はランダムである．

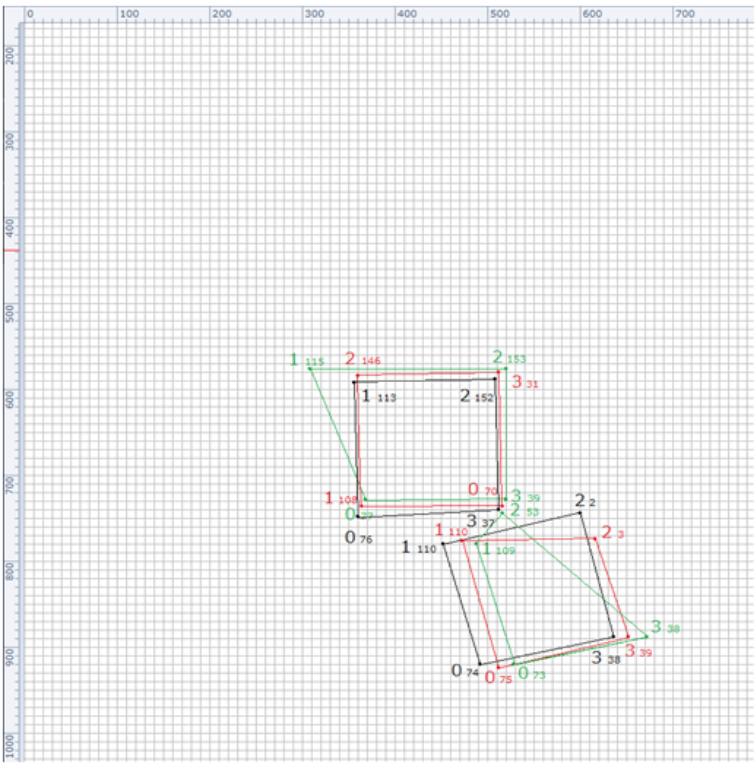

図 3.2: 二次元の AR マーカの輪郭線

3.1.3 NyARToolkit の動作

図 3.3 に，NyARIdMarker の ID=1 のマーカシートを認識したときの様子を示す．NyARToolkit では，JOGL (Java Binding for the OpenGL[3]) を用いて，認識した ID や，マーカの姿勢情報を，カメラ画像上に重畳して表示する機能が備わっている．動作確認を行うプログラムにおいては，認識した ID 情報を，対応するマーカ上に表示するシンプルな構成とした．

3.2 元レスポンスアナライザシステムの認識方法の比較

元 (平面的な AR マーカ) 認識手法：

1. カメラで AR マーカを撮影して，AR マーカの黒枠の二次元の座標を探して，格納する．長さの単位は 1 画素である．

[3] http://jogamp.org/jogl/www/

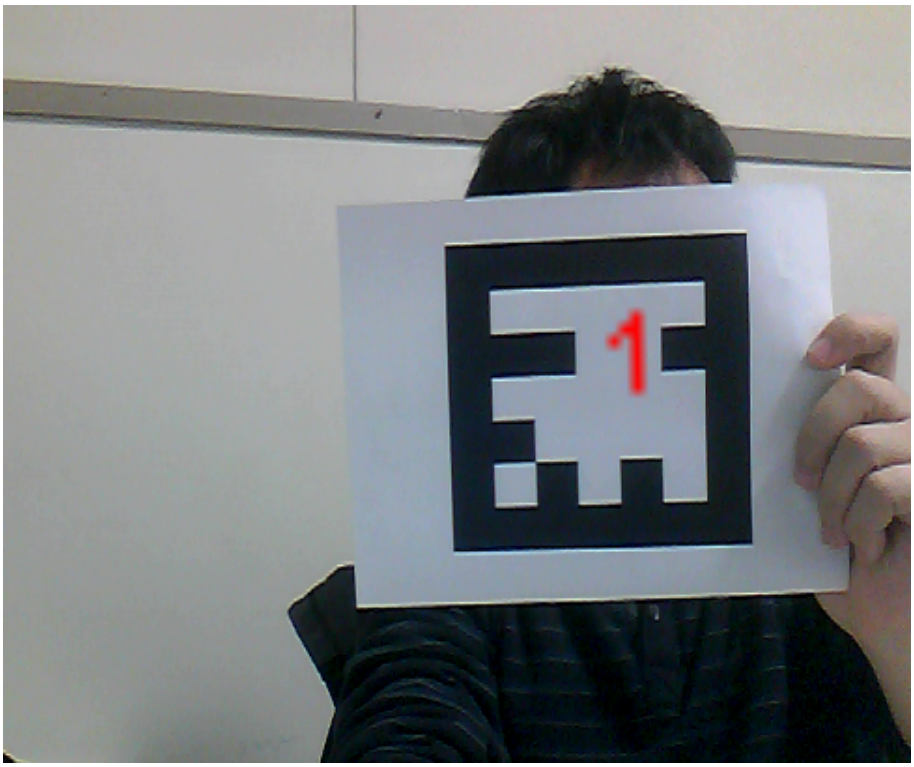

図 3.3: NyARIdMarker の ID=1 のマーカシートを認識したときの様子

2. マーカの黒枠の二次元の座標によって，AR マーカの四頂点を計算する．

3. AR マーカの四頂点と黒枠の位置を通じてマーカを計算すると，その AR マーカの中のデータを示すことができる．

改良した (平面的と湾曲的な AR マーカ) 認識手法：

1. カメラで AR マーカを撮影して，AR マーカの黒枠の二次元の座標を探して，格納する．長さの単位は 1 画素である．

2. マーカの黒枠の二次元の座標によって，AR マーカの四頂点を計算する．

3. 撮影した AR マーカは湾曲 AR マーカので，四辺の直線は曲がった線になった．公式 ax+by=c で四頂点の直線の距離を計算する．

4. 垂線を通じて直線公式 ax+by=c の上の点の座標と曲がった線の上の座標を対応し

て，湾曲 AR マーカの程度を計算すると，カメラで AR マーカを撮影して，その湾曲 AR マーカを認識できる，その AR マーカの中のデータを示すことができる．

3.3 改良したシステムのアルゴリズム

NyARCoord2SquareVertexIndexes() の機能の中，曲げった AR マーカを認識できる機能を増加する．曲げった AR マーカと曲げない AR マーカの区別は輪郭線が違う．曲げない AR マーカの輪郭線はカメラで認識の時，その輪郭線は 4 つ直線である．逆に曲げない AR マーカの輪郭線はカメラで認識の時，その輪郭線は一つ或いは 4 つアッチである．この状況の時，元のアルゴリズムは認識が保証できない．新しいアルゴリズムを作るが必要である．

そのアルゴリズム（図 3.4) は先に AR マーカの 4 つ頂点を確認する．隣接の 2 つ頂点の位置を確認して，直線の公式で仮定した直線を確認する．長さの単位は 1 画素で，隣接の 2 つ頂点の間の全部のアッチの点と仮設した直線の点を確認する．垂線を通じて 2 つ点の距離を計算して，その湾曲の輪郭線を確認する．そして，湾曲マーカ中のデータを画面上に示し，湾曲マーカの湾曲量を確認できるようにした．

図 3.5 は湾曲マーカを認識する時，取得した輪郭線の座標点の集合である．

曲げった AR マーカの認識を実現した，曲げ具合を数値化する時分散の状況が確認できた．原因はマーカを曲げたら，もらった輪郭線の長さが変化した．元のシステムの輪郭線の長さはほとんど同じである．曲げた AR マーカは，カメラで撮影した輪郭線の長さは変化した，曲げた輪郭線は曲げなかった輪郭線より短くなった，その状況の時，元の正方形の AR マーカは長方形になる．その状況は分散の状況が発生した．図 3.6

```
public int getSuisenByCoords(NyARIntCoordinates i_coord, int corneridx1,
        int corneridx2, int midpidx){
    Point2D n0p = i_coord.items[corneridx1].getPoint2D();
    Point2D n1p = i_coord.items[corneridx2].getPoint2D();
    Point2D p = i_coord.items[midpidx].getPoint2D();
    Point2D ret = getSuisen(n0p,n1p,p);
    return (int)ret.distance(p);

}
//垂線との交点をもとめるメソッド(n1p-n2p の線分に，pの点から下ろす)
public Point2D getSuisen(Point2D n0p, Point2D n1p, Point2D p) {
    double a, b, c, py; // 線分を，ax + by = c の形式に変換
    c = n0p.getX() * n1p.getY() - n1p.getX() * n0p.getY();
    b = n1p.getX() - n0p.getX();
    a = n0p.getY() - n1p.getY();
    py = n0p.getY(); // 水平線のときは，線分を構成するどちらかの点のy座標.
    Point2D tempFP = calcfootPoint(a, b, c, py, p);
    double len_1_2 = n0p.distance(n1p);
    double len_1_fp = n0p.distance(tempFP);
    double len_2_fp = n1p.distance(tempFP);
    if (len_1_2 < len_1_fp)
        return n1p;
    else if (len_1_2 < len_2_fp)
        return n0p;
    else
        return tempFP;
}
public Point2D.Double calcfootPoint(double la, double lb, double lc,
        double lpy, Point2D p) {
    double x, y;
    if (la == 0) {// 水平線の時
        x = p.getX();
        y = lpy;
    } else {
        double ld = lb * p.getX() - la * p.getY();
        double bumbo = la * la + lb * lb;
        x = (lb * ld - la * lc) / bumbo;
        y = (lb * x - ld) / la;
    }
    return new Point2D.Double(x, y);
}
```

図 3.4: 改良アルゴリズム

```
0 X:403 Y:116
1 X:404 Y:116
2 X:405 Y:116
3 X:406 Y:116
4 X:407 Y:116
5 X:408 Y:116
6 X:409 Y:116
7 X:410 Y:116
8 X:411 Y:116
9 X:412 Y:116
10 X:413 Y:116
11 X:414 Y:116
12 X:415 Y:116
13 X:416 Y:116
14 X:417 Y:116
15 X:418 Y:116
16 X:419 Y:116
17 X:420 Y:116
18 X:421 Y:116
19 X:422 Y:116
20 X:423 Y:116
21 X:424 Y:116
22 X:425 Y:116
23 X:426 Y:116
24 X:427 Y:116
25 X:428 Y:116
26 X:429 Y:116
27 X:430 Y:116
28 X:431 Y:117
29 X:432 Y:117
30 X:433 Y:117
31 X:434 Y:117
32 X:435 Y:117
33 X:436 Y:117
34 X:437 Y:117
35 X:438 Y:117
36 X:439 Y:117
37 X:440 Y:117
38 X:441 Y:117
39 X:442 Y:117
40 X:443 Y:117
41 X:444 Y:117
42 X:445 Y:117
43 X:446 Y:117
44 X:447 Y:117
45 X:448 Y:117
46 X:449 Y:117
```


図 3.5: 湾曲マーカの輪郭線の座標点の集合

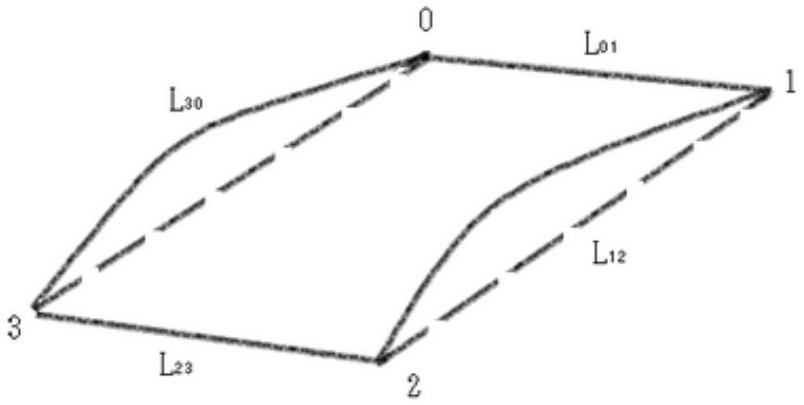

図 3.6: 分散が生じた原因

第4章　評価実験

第 4 章では 2 章で提案された方法を用いて，実際にデータ収集から認識までを行い結果について考察を加える．

4.1　実験の目的

元レスポンスアナライザシステムのアルゴリズムを改良して，AR マーカの認識の範囲を拡大する．同時に AR マーカの認識の湾曲量を取得して，アルゴリズムを改良の方法について考察する．

4.2　実験の準備

OS：windows 7
プログラミング言語：JAVA
開発環境：Eclipse3.5.0
NyAR マーカ（図 4.1）

4.3　実験の方法

元レスポンスアナライザシステムと改良したシステムで，平面的な AR マーカと非平面なマーカを認識して実験する．同じ環境中，2 つシステムで，平面と非平面ゆっくり移動と回転する．改良したシステムで非平面の状況中曲げるのマーカが異なる程度を実験する．毎種類の実験は 10 回で実験する．

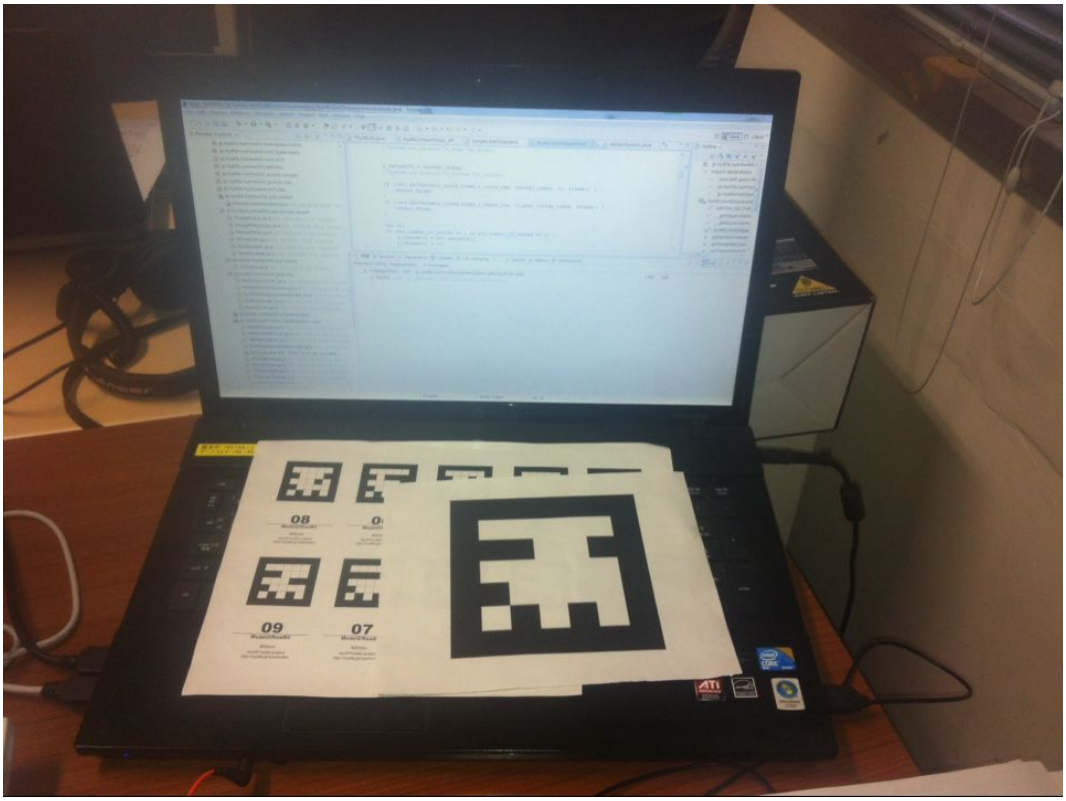

図 4.1: 実験の準備

4.4 実験の結果

元レスポンスアナライザシステムの実験の結果：平面的な AR マーカの認識は成功率と安定性が高いである．移動（図 4.2）と回転（図 4.3）だけでなく，移動と回転のスピードをもっと出したら，AR マーカはまだ認識できる．しかし，非平面の状況中はほとんど認識できない（図 4.4）．

改良したレスポンスアナライザシステムの実験の結果：平面的な AR マーカの認識は成功率と安定性が高いである．平面的な AR マーカの移動（図 4.5）と回転（図 4.6）のスピードをもっと出したら，60 %の状況中は AR マーカを認識できる．非平面的な AR マーカの認識は成功率は 90 %保証できる．特に曲げた AR マーカが回転する時（図 4.7），５０%成功率を達成した．ランダムで曲げた AR マーカを実験するので，人為的影響の可能性がある．曲げた時，スピードをもっと出したら，AR マーカはほとんど認識できな

図 4.2: 横の移動する

かった.

図 4.8 は，今回の実験におけるデータを集計したものである.

図 4.3: 回転する

図 4.4: 曲げた状況

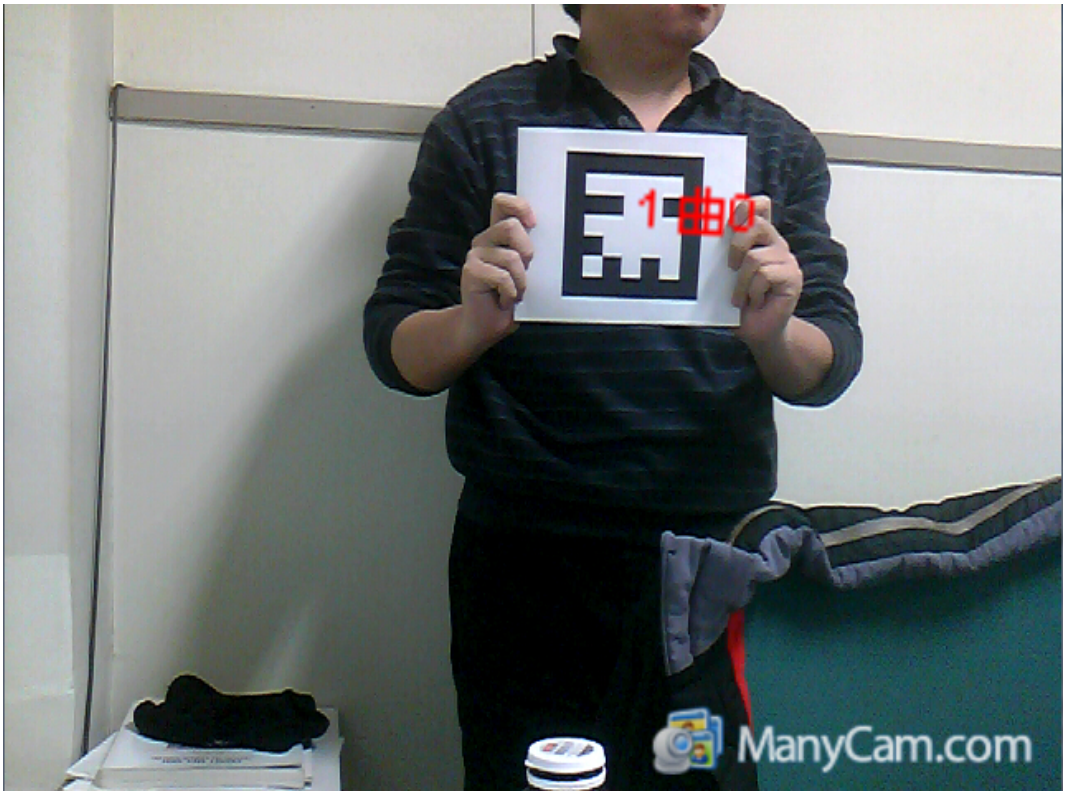

図 4.5: 横の移動する

図 4.6: 曲げた状況—横の移動する

図 4.7: 曲げた状況—-回転する

	元システム	改良したシステム		
平面	100%	100%	横の移動する	ゆっくり
	100%	100%	回転する	
	100%	100%	横の移動する	少し早く
	100%	100%	回転する	
非平面	20%	90%	横の移動する	ゆっくり
	10%	80%	回転する	
	0%	70%	横の移動する	少し早く
	0%	50%	回転する	

図 4.8: 統計データ

第5章　結論

第 5 章では提案手法の有効性を検証するための評価実験について述べる．

5.1　まとめ

本研究では，紙を用いた簡易型レスポンスアナライザシステムにおける，学習者が伝達可能な情報量を増やすことを目的として，AR マーカーシートの曲げ具合を認識し，湾曲量として取得するための方法について考察した．実験の結果をよると，改良したレスポンスアナライザシステムは曲げたの状況中認識できる．静止の状態は認識がほとんどできる．しかし，横の移動或いは回転する時，アルゴリズムは足りないところがある．特にスピードが少し早く移動すると，システムの反応が遅くて，認識も困難である．

5.2　今後の課題

実験の結果において，今改良したレスポンスアナライザシステムのアルゴリズムは湾曲 AR マーカを認識すること保証できる．しかし，曲げた AR マーカを認識するの成功率と安定性はまだ足りなかった．これから，アルゴリズムはもっと改良が必要である．

謝辞

修士論文を完成するにあたり，ご指導ご教授くださりました三浦准教授に御礼申し上げます．また，輪講や中間発表においてご指導やご教授を下さりました情報セクションの先生方に御礼申しあげます．加えて，本論文のデータ収集実験や評価実験において，被験者としてご参加頂きました三浦研究室の学生と情報セクションの学生にお礼を述べたいと思います．最後に，私の意思を尊重してくださり大学院進学を応援して頂き，経済面や生活面において，ご支援をして頂いた家族に心から感謝申し上げます．

参考文献

[1] T.C. Liu, H.Y. Wang, J.K. Liang, T.W. Chan, H.W. Ko, and J.C. Yang. Wireless and mobile technologies to enhance teaching and learning. *Journal of Computer Assisted Learning*, Vol. 19, No. 3, pp. 371–382, September 2003.

[2] Jeremy Roschelle and Roy Pea. A walk on the WILD side: How wireless handhelds may change computer-supported collaborative learning. In *Proc. of International Conference on ComputerSupported Collaborative Learning (CSCL-02)*, pp. 51–60, January 2002.

[3] Chi Wei Huang, Jen Kei Liang, and Hsu Yie Wang. EduClick: A Computer-Supported Formative Evaluation System with Wireless Devices in Ordinary Classroom. In *Proc. of Int. Conference on Computers in Education*, pp. 1462–1469, 2001.

[4] Jane E. Caldwell. Clickers in the Large Classroom: Current Research and Best-Practice Tips. *CBE life sciences education*, Vol. 6, No. 1, pp. 9–20, 2007.

[5] Motoki Miura and Toyohisa Nakada. Device-Free Personal Response System based on Fiducial Markers. In *Proceedings of the 7th IEEE International Conference on Wireless, Mobile, and Ubiquitous Technologies in Education (WMUTE2012)*, pp. 87–91, March 2012.

[6] Andrew Cross, Edward Cutrell, and William Thies. Low-cost audience polling using computer vision. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pp. 45–54, New York, NY, USA, 2012. ACM.

[7] Hirokazu Kato and Mark Billinghurst. Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System. In *Proc. of the 2nd IEEE and ACM International Workshop on Augmented Reality '99*, pp. 85–94, October 1999.

[8] Daniel Wagner and Dieter Schmalstieg. ARToolKitPlus for Pose Tracking on Mobile Devices. In *Proc. of 12th Computer Vision Winter Workshop (CVWW'07)*, February 2007.

[9] Mark Fiala. ARTag, a fiducial marker system using digital techniques. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005)*, Vol. 2, pp. 590–596. IEEE, June 2005.

[10] Mark Fiala. Artag revision 1, a fiducial marker system using digital techniques. 2004. `http://nparc.cisti-icist.nrc-cnrc.gc.ca/npsi/ctrl?action=rtdoc&an=5765376`.

```java
 * PROJECT: NyARToolkit JOGL sample program.
package jp.nyatla.nyartoolkit.jogl.sample;

import java.awt.event.*;


/**
 * このプログラムは、JMFからの映像入力からマーカを検出し、そこに立方体を重ねます。
 * 新しいSimpleLiteのサンプルです。
 * スケッチシステム/レンダリングクラスを使わずに、OpenGLAPIをそのまま使用します。
 * 動作は、スケッチサンプル{@link SimpleLiteM}と同じです。
 * ARマーカには、patt.hiro/patt_kanjiを使用して下さい。
 */
public class SimpleLiteMStandard implements GLEventListener
{
    // NyARToolkit関係
    private NyARJmfCamera _camera;
    //  private NyARMarkerSystem _nyar;
    private NyARGlMarkerSystem _nyar;
    private final static String ARCODE_FILE = "./Data/patt.hiro";
    private final static String ARCODE_FILE2 = "./Data/patt.kanji";
    private int[] ids=new int[100];
    public SimpleLiteMStandard(INyARMarkerSystemConfig i_config) throws NyARException
    {
        JmfCaptureDeviceList devlist = new JmfCaptureDeviceList();
        JmfCaptureDevice d = devlist.getDevice(0);
        d.setCaptureFormat(i_config.getScreenSize(),30.0f);
        this._camera=new NyARJmfCamera(d);//create sensor system
        this._nyar=new NyARGlMarkerSystem(i_config);   //create MarkerSystem
        //      this.ids[0]=this._nyar.addARMarker(ARCODE_FILE2,16,25,80);
        //      this.ids[1]=this._nyar.addARMarker(ARCODE_FILE,16,25,80);
        //      this._nyar=new NyARMarkerSystem(i_config);
        for(int i=0;i<100;i++){
            this.ids[i]=this._nyar.addNyIdMarker(i, 40);
        }

        Frame frame= new Frame("NyARTK program");
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
        GLCanvas canvas = new GLCanvas();
        frame.add(canvas);
        canvas.addGLEventListener(this);
        NyARIntSize s=i_config.getNyARParam().getScreenSize();

        frame.setVisible(true);
        Insets ins = frame.getInsets();
        frame.setSize(s.w + ins.left + ins.right,s.h + ins.top + ins.bottom);
        canvas.setBounds(ins.left, ins.top, s.w,s.h);


        this._camera.start();
    }

    public void init(GLAutoDrawable drawable)
    {
        GL gl=drawable.getGL();
        gl.glMatrixMode(GL.GL_PROJECTION);
```

```
102         gl.glLoadMatrixd(this._nyar.getGlProjectionMatrix(),0);
103         gl.glEnable(GL.GL_DEPTH_TEST);
104         gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
105         Animator animator = new Animator(drawable);
106         animator.start();
107         return;
108     }
109     public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height)
110     {
111         GL gl=drawable.getGL();
112         gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
113         gl.glViewport(0, 0, width, height);
114         return;
115     }
116     public void display(GLAutoDrawable drawable)
117     {
118         GL gl=drawable.getGL();
119         synchronized(this._camera)
120         {
121             try {
122                 gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT); // Clear the
    buffers for new frame.
123                 NyARGLDrawUtil.drawBackGround(gl,this._camera.getSourceImage(), 1.0);
124 //              NyARGLDrawUtil.drawBackGround(gl,this._camera.getGsImage(), 1.0);
125                 this._nyar.update(this._camera);
126                 for(int i=0;i<100;i++){
127                     if(this._nyar.isExistMarker(this.ids[i])){
128                         gl.glMatrixMode(GL.GL_MODELVIEW);
129                         gl.glPushMatrix();
130                         gl.glLoadMatrixd(this._nyar.getGlMarkerMatrix(this.ids[i]),0);
131 //                      NyARGLDrawUtil.drawColorCube(gl,20);
132                         int mage = this._nyar.getMarkerMageMatrix(this.ids[i]);
133                         NyARGLDrawUtil.setFontColor(Color.green);
134                         NyARGLDrawUtil.drawText(String.valueOf(i)+" 曲"+mage+"",1.5f);
135                         gl.glPopMatrix();
136 //                      System.out.println(i);
137                     }
138                 }
139                 Thread.sleep(1);
140             } catch (Exception e) {
141                 e.printStackTrace();
142             }
143         }
144     }
145     public void displayChanged(GLAutoDrawable arg0, boolean arg1, boolean arg2)
146     {
147     }
148
149     private final static String PARAM_FILE = "./Data/camera_para.dat";
150     private final static int SCREEN_X = 640;
151     private final static int SCREEN_Y = 480;
152
153     public static void main(String[] args)
154     {
155         try {
156             NyARMarkerSystemConfig config = new NyARMarkerSystemConfig(new
    FileInputStream(PARAM_FILE),SCREEN_X, SCREEN_Y);
157             new SimpleLiteMStandard(config);
158         } catch (Exception e) {
159             e.printStackTrace();
160         }
```

```
161         return;
162     }
163
164 }
165
```

```
 2 * PROJECT: NyARToolkit
31 package jp.nyatla.nyartoolkit.core.squaredetect;
32
33 import jp.nyatla.nyartoolkit.core.NyARException;
38
39
40 /**
41  * このクラスは、{@link NyARLabeling_Rle}クラスを用いた矩形検出器です。
42  * 検出した矩形を、自己コールバック関数{@link #onSquareDetect}へ通知します。
43  * 継承クラスで自己コールバック関数{@link #onSquareDetect}を実装する必要があります。
44  */
45 public class NyARSquareContourDetector_Rle extends NyARSquareContourDetector
46 {
47     /** label_stackにソート後の結果を蓄積するクラス*/
48     protected class Labeling extends NyARLabeling_Rle
49     {
50         public NyARRleLabelFragmentInfoPtrStack label_stack;
51         int _right;
52         int _bottom;
53
54
55         public Labeling(int i_width,int i_height) throws NyARException
56         {
57             super(i_width,i_height);
58             long t=(long)i_width*i_height*2048/(320*240)+32;//full HD support
59             this.label_stack=new NyARRleLabelFragmentInfoPtrStack((int)t);//検出可能な最大ラ
   ベル数
60             this._bottom=i_height-1;
61             this._right=i_width-1;
62             return;
63         }
64         public boolean labeling(INyARGrayscaleRaster i_raster,NyARIntRect i_area,int i_th)
   throws NyARException
65         {
66             //配列初期化
67             this.label_stack.clear();
68             //ラベルの検出
69             boolean ret=super.labeling(i_raster, i_area, i_th);
70             //ソート
71             this.label_stack.sortByArea();
72             return ret;
73         }
74         public boolean labeling(INyARGrayscaleRaster i_raster,int i_th) throws
   NyARException
75         {
76             //配列初期化
77             this.label_stack.clear();
78             //ラベルの検出
79             boolean ret=super.labeling(i_raster,i_th);
80             //ソート
81             this.label_stack.sortByArea();
82             return ret;
83         }
84
85         protected void onLabelFound(NyARRleLabelFragmentInfo i_label)
86         {
87             // クリップ領域が画面の枠に接していれば除外
88             if (i_label.clip_l == 0 || i_label.clip_r == this._right){
89                 return;
90             }
91             if (i_label.clip_t == 0 || i_label.clip_b == this._bottom){
```

```
92                 return;
93             }
94             this.label_stack.push(i_label);
95         }
96 
97     }
98 
99     protected Labeling _labeling;
100    private final NyARLabelOverlapChecker<NyARRleLabelFragmentInfo> _overlap_checker = new
   NyARLabelOverlapChecker<NyARRleLabelFragmentInfo>(32,NyARRleLabelFragmentInfo.class);
101    private NyARContourPickup _cpickup=new NyARContourPickup();
102    private final NyARCoord2SquareVertexIndexes _coord2vertex=new
   NyARCoord2SquareVertexIndexes();
103    private final NyARIntCoordinates _coord;
104    /**
105     * コンストラクタです。
106     * 入力画像のサイズを指定して、インスタンスを生成します。
107     * @param i_size
108     * 入力画像のサイズ
109     */
110    public NyARSquareContourDetector_Rle(NyARIntSize i_size) throws NyARException
111    {
112        this.setupImageDriver(i_size);
113        //ラベリングのサイズを指定したいときはsetAreaRangeを使ってね。
114        this._coord = new NyARIntCoordinates((i_size.w + i_size.h) * 2);
115        return;
116    }
117    /**
118     * 画像処理オブジェクトの切り替え関数。切り替える場合は、この関数を上書きすること。
119     * @param i_size
120     * @throws NyARException
121     */
122    protected void setupImageDriver(NyARIntSize i_size) throws NyARException
123    {
124        //特性確認
125        assert(NyARLabeling_Rle._sf_label_array_safe_reference);
126        this._labeling=new Labeling(i_size.w,i_size.h);
127        this._cpickup=new NyARContourPickup();
128    }
129 
130    private final int[] __detectMarker_mkvertex = new int[4];
131    private final int[] __mage = new int[5]; //曲げ度合い
132    /**
133     * この関数は、ラスタから矩形を検出して、自己コールバック関数{@link #onSquareDetect}で通知します。
134     * @param i_raster
135     * 検出元のラスタ画像
136     * 入力できるラスタの画素形式は、{@link NyARLabeling_Rle#labeling(INyARRaster, int)}と同じです。
137     * @param i_area
138     * 検出する範囲。検出元のラスタの内側である必要があります。
139     * @param i_th
140     * ラベルと判定する敷居値
141     * @throws NyARException
142     *
143     * TODO:
144     */
145    public void detectMarker(INyARGrayscaleRaster i_raster,NyARIntRect i_area,int
   i_th,NyARSquareContourDetector.CbHandler i_cb) throws NyARException
146    {
147        assert(i_area.w*i_area.h>0);
148 
149        final NyARRleLabelFragmentInfoPtrStack flagment=this._labeling.label_stack;
```

```
150         final NyARLabelOverlapChecker<NyARRleLabelFragmentInfo> overlap =
    this._overlap_checker;
151
152         //ラベルの生成エラーならここまで
153         if(!this._labeling.labeling(i_raster, i_area, i_th)){
154             return;
155         }
156         // ラベル数が0ならここまで
157         final int label_num=flagment.getLength();
158         if (label_num < 1) {
159             return;
160         }
161
162         //ラベルリストを取得
163         NyARRleLabelFragmentInfo[] labels=flagment.getArray();
164
165         NyARIntCoordinates coord = this._coord;
166         final int[] mkvertex =this.__detectMarker_mkvertex;
167         final int[] mage = this.__mage;
168
169         //重なりチェッカの最大数を設定
170         overlap.setMaxLabels(label_num);
171
172         for (int i=0; i < label_num; i++) {
173             NyARRleLabelFragmentInfo label_pt=labels[i];
174             // 既に検出された矩形との重なりを確認
175             if (!overlap.check(label_pt)) {
176                 // 重なっているようだ。
177                 continue;
178             }
179
180             //輪郭を取得
181             if(!this._cpickup.getContour(i_raster,i_area,
    i_th,label_pt.entry_x,label_pt.clip_t,coord))
182             {
183                 continue;
184             }
185             int label_area = label_pt.area;
186             //輪郭線をチェックして、矩形かどうかを判定。矩形ならばmkvertexに取得
187             //TODO: ここで、頂点チェックをしている。ついでに、曲げ度数ももらう
188             if (!this._coord2vertex.getVertexIndexes(coord,label_area,mkvertex,mage)){
189                 // 頂点の取得が出来なかった
190                 continue;
191             }
192             //矩形を発見したことをコールバック関数で通知
193             i_cb.detectMarkerCallback(coord,mkvertex,mage);
194
195             // 検出済の矩形の属したラベルを重なりチェックに追加する。
196             overlap.push(label_pt);
197
198         }
199         return;
200     }
201     /**
202      * この関数は、ラスタから矩形を検出して、自己コールバック関数{@link #onSquareDetect}で通知します。
203      * ARToolKitのarDetectMarker2を基にしています。
204      * @param i_raster
205      * 検出元のラスタ画像
206      * 入力できるラスタの画素形式は、{@link NyARLabeling_Rle#labeling(NyARGrayscaleRaster,
    int)}と同じです。
207      * @param i_th
```

```
208      * 画素の二値判定敷居値です。この値は、ラベリングと、輪郭線追跡時に使われます。
209      */
210     public void detectMarker(INyARGrayscaleRaster i_raster,int
    i_th,NyARSquareContourDetector.CbHandler i_cb) throws NyARException
211     {
212         final NyARRleLabelFragmentInfoPtrStack flagment=this._labeling.label_stack;
213         final NyARLabelOverlapChecker<NyARRleLabelFragmentInfo> overlap =
    this._overlap_checker;
214 
215         // ラベル数が0ならここまで
216         flagment.clear();
217         //ラベルの生成エラーならここまで
218         if(!this._labeling.labeling(i_raster, i_th)){
219             return;
220         }
221         final int label_num=flagment.getLength();
222 //      System.out.println("Label Num : "+label_num);
223         if (label_num < 1) {
224             return;
225         }
226         //ラベルをソートしておく
227         flagment.sortByArea();
228         //ラベルリストを取得
229         NyARRleLabelFragmentInfo[] labels=flagment.getArray();
230 
231         NyARIntCoordinates coord = this._coord;
232         final int[] mkvertex =this.__detectMarker_mkvertex;
233 
234         int[] mage = new int[5];
235         //重なりチェッカの最大数を設定
236         overlap.setMaxLabels(label_num);
237 
238         for (int i=0; i < label_num; i++) {
239             final NyARRleLabelFragmentInfo label_pt=labels[i];
240             int label_area = label_pt.area;
241 
242             // 既に検出された矩形との重なりを確認
243             if (!overlap.check(label_pt)) {
244                 // 重なっているようだ。
245                 continue;
246             }
247             //輪郭を取得
248             if(!this._cpickup.getContour(i_raster,i_th,label_pt.entry_x,label_pt.clip_t,coo
    rd)){
249                 continue;
250             }
251             //輪郭線をチェックして、矩形かどうかを判定。矩形ならばmkvertexに取得
252             if (!this._coord2vertex.getVertexIndexes(coord,label_area, mkvertex, mage)) {
253                 // 頂点の取得が出来なかった
254                 continue;
255             }
256             //矩形を発見したことをコールバック関数で通知
257             i_cb.detectMarkerCallback(coord,mkvertex,mage);
258 
259             // 検出済の矩形の属したラベルを重なりチェックに追加する。
260             overlap.push(label_pt);
261 
262         }
263         return;
264     }
265 }
```

266
267
268
269

```
 2 * PROJECT: NyARToolkit
31 package jp.nyatla.nyartoolkit.core.squaredetect;
32
33 import java.awt.geom.Point2D;
36
37 /**
38  * このクラスは、輪郭線を四角形と仮定して、その頂点位置を計算します。
39  * ARToolKitの四角形検出処理の一部です。
40  */
41 public class NyARCoord2SquareVertexIndexes
42 {
43     private static final double VERTEX_FACTOR = 1.0;// 線検出のファクタ
44     private final NyARVertexCounter __getSquareVertex_wv1 = new NyARVertexCounter();
45     private final NyARVertexCounter __getSquareVertex_wv2 = new NyARVertexCounter();
46     /**
47      * コンストラクタです。
48      * インスタンスを生成します。
49      */
50     public NyARCoord2SquareVertexIndexes()
51     {
52         return;
53     }
54     /**
55      * この関数は、座標集合から頂点候補になりそうな場所を4箇所探して、そのインデクス番号を返します。
56      * @param i_coord
57      * 輪郭点集合を格納したオブジェクト。
58      * @param i_area
59      * 矩形判定のヒント値。矩形の大きさを、そのラベルを構成するピクセルの数で指定します。
60      * (注)このパラメータは、マーカノデザイン、枠の大きさが影響等、ラベルの大きさに影響を受けます。
61      * @param o_vertex
62      * 4頂点のインデクスを受け取る配列です。4要素以上の配列を指定してください。
63      * @param o_mage
64      * [0]に曲げ度合いの合計値、[1]~[4]に、頂点0-1,1-2,2-3,3-0の曲げ度合い
65      * @return
66      * 頂点が見つかるとtrueを返します。
67      * TODO:
68      */
69     public boolean getVertexIndexes(NyARIntCoordinates i_coord, int i_area, int[] o_vertex,
70             int[] o_mage)
71     {
72         final NyARVertexCounter wv1 = this.__getSquareVertex_wv1;
73         final NyARVertexCounter wv2 = this.__getSquareVertex_wv2;
74         //集合の長さ
75         int i_coord_num=i_coord.length;
76
77         //System.out.println("i_coord_num "+i_coord_num);    //
78         int vertex1_index=getFarPoint(i_coord.items,i_coord_num,0);
79         //      System.out.println("vertex1_index "+vertex1_index);
80         int prev_vertex_index=(vertex1_index+i_coord_num)%i_coord_num;
81         //      System.out.println("vertex1_index+i_coord_num "+vertex1_index+i_coord_num);
82         //System.out.println("prev_vertex_index "+prev_vertex_index);
83         int v1=getFarPoint(i_coord.items,i_coord_num,vertex1_index);
84         //System.out.println("v1 "+v1);
85         final double thresh = (i_area / 0.75) * 0.01 * VERTEX_FACTOR;
86         //      System.out.println("i_area "+i_area);
87
88
89         o_vertex[0] = vertex1_index;
90         //      System.out.println("o_vertex "+o_vertex);
91
92         if (!wv1.getVertex(i_coord.items,i_coord_num, vertex1_index, v1, thresh)) {
```

```
 93             return false;
 94         }
 95         if (!wv2.getVertex(i_coord.items,i_coord_num, v1,prev_vertex_index, thresh)) {
 96             return false;
 97         }
 98 
 99         int v2;
100         if (wv1.number_of_vertex == 1 && wv2.number_of_vertex == 1) {
101             o_vertex[1] = wv1.vertex[0];
102             o_vertex[2] = v1;
103             o_vertex[3] = wv2.vertex[0];
104         } else if (wv1.number_of_vertex > 1 && wv2.number_of_vertex == 0) {
105             //頂点位置を、起点から対角点の間の1/2にあると予想して、検索する。
106             if(v1>=vertex1_index){
107                 v2 = (v1-vertex1_index)/2+vertex1_index;
108             }else{
109                 v2 = ((v1+i_coord_num-vertex1_index)/2+vertex1_index)%i_coord_num;
110             }
111             if (!wv1.getVertex(i_coord.items,i_coord_num, vertex1_index, v2, thresh)) {
112                 return false;
113             }
114             if (!wv2.getVertex(i_coord.items,i_coord_num, v2, v1, thresh)) {
115                 return false;
116             }
117             if (wv1.number_of_vertex == 1 && wv2.number_of_vertex == 1) {
118                 o_vertex[1] = wv1.vertex[0];
119                 o_vertex[2] = wv2.vertex[0];
120                 o_vertex[3] = v1;
121             } else {
122                 return false;
123             }
124         } else if (wv1.number_of_vertex == 0 && wv2.number_of_vertex > 1) {
125             //v2 = (v1+ end_of_coord)/2;
126             if(v1<=prev_vertex_index){
127                 v2 = (v1+prev_vertex_index)/2;
128             }else{
129                 v2 = ((v1+i_coord_num+prev_vertex_index)/2)%i_coord_num;
130 
131             }
132             if (!wv1.getVertex(i_coord.items,i_coord_num, v1, v2, thresh)) {
133                 return false;
134             }
135             if (!wv2.getVertex(i_coord.items,i_coord_num, v2, prev_vertex_index, thresh)) {
136                 return false;
137             }
138             if (wv1.number_of_vertex == 1 && wv2.number_of_vertex == 1) {
139                 o_vertex[1] = v1;
140                 o_vertex[2] = wv1.vertex[0];
141                 o_vertex[3] = wv2.vertex[0];
142             } else {
143 
144                 return false;
145             }
146         } else {
147             return false;
148         }
149 //      for(int i=0;i<4;i++) System.out.println("o_vertex["+i+"]"+o_vertex[i]+"    ");
150 //      System.out.println("Found!");
151         o_mage[0] = 0;
152 //      o_mage[1] = 0;
153 //      o_mage[2] = 0;
```

```
154 //      o_mage[3] = 0;
155 //      o_mage[4] = 0;
156
157         for(int vpi = 0; vpi < 4 ; vpi++){
158             int vv1 = vpi;
159             int vv2 = (vpi+1)%4;
160 //          System.out.println("vv1 = "+vv1+"   vv2 = "+vv2);
161             if (o_vertex[vv1] < o_vertex[vv2]){
162 //              System.out.println("    "+(o_vertex[vv2]-o_vertex[vv1]));
163                 for(int i= o_vertex[vv1] +1; i<o_vertex[vv2] ; i++){
164                     int tmp = getSuisenByCoords(i_coord, o_vertex[vv1], o_vertex[vv2], i);
165                     o_mage[0] += tmp;
166 //                  o_mage[vv1+1] += tmp;
167                 }
168             } else {
169 //              System.out.println("   rev "+(i_coord_num-o_vertex[vv1]+o_vertex[vv2]));
170                 for(int i= o_vertex[vv1] +1; i<i_coord_num ; i++){
171                     int tmp = getSuisenByCoords(i_coord, o_vertex[vv2], o_vertex[vv1], i);
172                     o_mage[0] += tmp;
173 //                  o_mage[vv1+1] += tmp;
174                 }
175                 for(int i= 0; i<o_vertex[vv2] ; i++){
176                     int tmp = getSuisenByCoords(i_coord, o_vertex[vv2], o_vertex[vv1], i);
177                     o_mage[0] += tmp;
178 //                  o_mage[vv1+1] += tmp;
179                 }
180             }
181         }
182 //      System.out.println(o_mage[0]+" "+i_coord_num);
183         o_mage[0] = o_mage[0]*18/i_coord_num; //点の数による正規化を行う.
184 //      for(int k=0;k<5;k++){
185 //          System.out.println(" o_mage "+k+" "+ o_mage[k]);
186 //      }
187 //     System.out.println("test");
188         return true;
189     }
190
191
192     public int getSuisenByCoords(NyARIntCoordinates i_coord, int corneridx1,
193             int corneridx2, int midpidx){
194         Point2D n0p = i_coord.items[corneridx1].getPoint2D();
195         Point2D n1p = i_coord.items[corneridx2].getPoint2D();
196         //Point2D n2p = i_coord.items[corneridx3].getPoint2D();
197         //Point2D n3p = i_coord.items[corneridx4].getPoint2D();
198         Point2D p = i_coord.items[midpidx].getPoint2D();
199         Point2D ret = getSuisen(n0p,n1p,p);
200         //System.out.println(" n0p "+ n0p);
201         //System.out.println(" n1p "+ n1p);
202         //System.out.println(" n1p "+ n2p);
203         //System.out.println(" n1p "+ n3p);
204         //System.out.println(" p "+ p);
205         return (int)ret.distance(p);
206
207     }
208     //垂線との交点をもとめるメソッド(n1p-n2p の線分に, pの点から下ろす)
209     public Point2D getSuisen(Point2D n0p, Point2D n1p, Point2D p) {
210         double a, b, c, py; // 線分を, ax + by = c の形式に変換
211         c = n0p.getX() * n1p.getY() - n1p.getX() * n0p.getY();
212         b = n1p.getX() - n0p.getX();
213         a = n0p.getY() - n1p.getY();
214         py = n0p.getY(); // 水平線のときは, 線分を構成するどちらかの点のy座標.
```

```
215         Point2D tempFP = calcfootPoint(a, b, c, py, p);
216         //System.out.println("a: "+a);
217         //System.out.println("b: "+b);
218         //System.out.println("c: "+c);
219         //System.out.println("py: "+py);
220         //System.out.println("p: "+p);
221         double len_1_2 = n0p.distance(n1p);
222         double len_1_fp = n0p.distance(tempFP);
223         double len_2_fp = n1p.distance(tempFP);
224         //System.out.println("len_1_2: "+len_1_2);
225         //System.out.println("len_1_fp: "+len_1_fp);
226         //System.out.println("len_2_fp: "+len_2_fp);
227 
228         if (len_1_2 < len_1_fp)
229             return n1p;
230         else if (len_1_2 < len_2_fp)
231             return n0p;
232         else
233             return tempFP;
234     }
235     public Point2D.Double calcfootPoint(double la, double lb, double lc,
236             double lpy, Point2D p) {
237         double x, y;
238         if (la == 0) {// 水平線の時
239             x = p.getX();
240             y = lpy;
241         } else {
242             double ld = lb * p.getX() - la * p.getY();
243             double bumbo = la * la + lb * lb;
244             x = (lb * ld - la * lc) / bumbo;
245             y = (lb * x - ld) / la;
246             //System.out.println("x: "+x);
247             //System.out.println("y: "+y);
248         }
249         return new Point2D.Double(x, y);
250     }
251 
252     /**
253      * i_pointの輪郭座標から、最も遠方にある輪郭座標のインデクスを探します。
254      * @param i_xcoord
255      * @param i_ycoord
256      * @param i_coord_num
257      * @return
258      */
259     private static int getFarPoint(NyARIntPoint2d[] i_coord,int i_coord_num,int i_point)
260     {
261         //
262         final int sx = i_coord[i_point].x;
263         final int sy = i_coord[i_point].y;
264         int d = 0;
265         int w, x, y;
266         int ret = 0;
267         for (int i = i_point+1; i < i_coord_num; i++) {
268             x = i_coord[i].x - sx;
269             y = i_coord[i].y - sy;
270             w = x * x + y * y;
271             if (w > d) {
272                 d = w;
273                 ret = i;
274             }
275         }
```

```
276         for (int i = 0; i < i_point; i++) {
277             x = i_coord[i].x - sx;
278             y = i_coord[i].y - sy;
279             w = x * x + y * y;
280             if (w > d) {
281                 d = w;
282                 ret = i;
283             }
284         }
285         return ret;
286     }
287 }
288
289
290
291
292 /**
293  * get_vertex関数を切り離すためのクラス
294  *
295  */
296 final class NyARVertexCounter
297 {
298     public final int[] vertex = new int[10];// 6まで削れる
299
300     public int number_of_vertex;
301
302     private double thresh;
303
304     private NyARIntPoint2d[] _coord;
305
306
307     public boolean getVertex(NyARIntPoint2d[] i_coord,int i_coord_len,int st, int ed,
    double i_thresh)
308     {
309         this.number_of_vertex = 0;
310         this.thresh = i_thresh;
311         this._coord = i_coord;
312         return get_vertex(st, ed,i_coord_len);
313     }
314
315     /**
316      * static int get_vertex( int x_coord[], int y_coord[], int st, int ed,double thresh,
    int vertex[], int *vnum) 関数の代替関数
317      *
318      * @param x_coord
319      * @param y_coord
320      * @param st
321      * @param ed
322      * @param thresh
323      * @return
324      */
325     private boolean get_vertex(int st, int ed,int i_coord_len)
326     {
327         //メモ:座標値は65536を超えなければint32で扱って大丈夫なので変更。
328         //dmaxは4乗なのでやるとしてもint64じゃないとマズイ
329         int v1 = 0;
330         final NyARIntPoint2d[] coord = this._coord;
331         final int a = coord[ed].y - coord[st].y;
332         final int b = coord[st].x - coord[ed].x;
333         final int c = coord[ed].x * coord[st].y - coord[ed].y * coord[st].x;
334         double dmax = 0;
```

```
335         if(st<ed){
336             //stとedが1区間
337             for (int i = st + 1; i < ed; i++) {
338                 final double d = a * coord[i].x + b * coord[i].y + c;
339                 if (d * d > dmax) {
340                     dmax = d * d;
341                     v1 = i;
342                 }
343             }
344         }else{
345             //stとedが2区間
346             for (int i = st + 1; i < i_coord_len; i++) {
347                 final double d = a * coord[i].x + b * coord[i].y + c;
348                 if (d * d > dmax) {
349                     dmax = d * d;
350                     v1 = i;
351                 }
352             }
353             for (int i = 0; i < ed; i++) {
354                 final double d = a * coord[i].x + b * coord[i].y + c;
355                 if (d * d > dmax) {
356                     dmax = d * d;
357                     v1 = i;
358                 }
359             }
360         }
361 
362 
363         if (dmax / (double)(a * a + b * b) > thresh) {
364             if (!get_vertex(st, v1,i_coord_len)) {
365                 return false;
366             }
367             if (number_of_vertex > 5) {
368                 return false;
369             }
370             vertex[number_of_vertex] = v1;// vertex[(*vnum)] = v1;
371             number_of_vertex++;// (*vnum)++;
372 
373             if (!get_vertex(v1, ed,i_coord_len)) {
374                 return false;
375             }
376         }
377         return true;
378     }
379 }
```

```java
 * PROJECT: NyARToolkit
package jp.nyatla.nyartoolkit.core.squaredetect;

import jp.nyatla.nyartoolkit.core.types.*;

/**
 * このクラスは、矩形情報を格納します。
 * ARToolKitのARMarkerInfoに相当しますが、このクラスは理想座標のみを取り扱います。
 */
public class NyARSquare
{
    /** 矩形の辺の直線式です。*/
    public NyARLinear[] line = NyARLinear.createArray(4);
    /** 矩形の頂点です。line[n]と、line[(n+3)%4]の交点でもあります。*/
    public NyARDoublePoint2d[] sqvertex = NyARDoublePoint2d.createArray(4);
    /** TODO:追加した曲げ具合 */
    public int tmp_mage;
    /**
     * この関数は、矩形の中心点を計算します。
     * @param o_out
     * 結果を格納するバッファ。
     */
    public void getCenter2d(NyARDoublePoint2d o_out)
    {
        o_out.x=(this.sqvertex[0].x+this.sqvertex[1].x+this.sqvertex[2].x+this.sqvertex[3].x)/4;
        o_out.y=(this.sqvertex[0].y+this.sqvertex[1].y+this.sqvertex[2].y+this.sqvertex[3].y)/4;
        return;
    }
    /**
     * この関数は、頂点同士の距離から、頂点のシフト量（回転量）を返します。
     * よく似た2つの矩形の頂点同士の、頂点の対応を取るために使用します。
     * @param i_square
     * 比較対象の矩形
     * @return
     * シフト量を数値で返します。
     * シフト量はthis-i_squareです。1の場合、this.sqvertex[0]とi_square.sqvertex[1]が対応点になる(shift量1)であることを示します。
     */
    public int checkVertexShiftValue(NyARSquare i_square)
    {
        NyARDoublePoint2d[] a=this.sqvertex;
        NyARDoublePoint2d[] b=i_square.sqvertex;

        //3-0番目
        int min_dist=Integer.MAX_VALUE;
        int min_index=0;
        int xd,yd;
        for(int i=3;i>=0;i--){
            int d=0;
            for(int i2=3;i2>=0;i2--){
                xd= (int)(a[i2].x-b[(i2+i)%4].x);
                yd= (int)(a[i2].y-b[(i2+i)%4].y);
                d+=xd*xd+yd*yd;
            }
            if(min_dist>d){
                min_dist=d;
                min_index=i;
            }
        }
```

```
88          return min_index;
89      }
90
91      /** 4とnの最大公約数テーブル*/
92      private final static int[] _gcd_table4={-1,1,2,1};
93      /**
94       * この関数は、頂点を左回転して、矩形を回転させます。
95       * @param i_shift
96       * シフト量。4未満、0以上である事。
97       */
98      public void rotateVertexL(int i_shift)
99      {
100 //      assert(i_shift<4);
101         NyARDoublePoint2d vertext;
102         NyARLinear linet;
103         if(i_shift==0){
104             return;
105         }
106         int t1,t2;
107         int d, i, j, mk;
108         int ll=4-i_shift;
109         d = _gcd_table4[ll];//NyMath.gcn(4,ll);
110         mk = (4-ll) % 4;
111         for (i = 0; i < d; i++) {
112             linet=this.line[i];
113             vertext=this.sqvertex[i];
114             for (j = 1; j < 4/d; j++) {
115                 t1=(i + (j-1)*mk) % 4;
116                 t2=(i + j*mk) % 4;
117                 this.line[t1]=this.line[t2];
118                 this.sqvertex[t1]=this.sqvertex[t2];
119             }
120             t1=(i + ll) % 4;
121             this.line[t1]=linet;
122             this.sqvertex[t1]=vertext;
123         }
124     }
125 }
```

```
 2  * PROJECT: NyARToolkit(Extension)
25 package jp.nyatla.nyartoolkit.markersystem.utils;
26
27 import java.awt.geom.Point2D;
35
36 /**
37  * このクラスは、NyIdの検出結果をマッピングします。
38  */
39 public class NyIdList extends ArrayList<NyIdList.Item>
40 {
41     public static class Item extends TMarkerData
42     {
43         /** MK_NyIdの情報。反応するidの開始レンジ*/
44         public final long nyid_range_s;
45         /** MK_NyIdの情報。反応するidの終了レンジ*/
46         public final long nyid_range_e;
47         /** MK_NyIdの情報。実際のid値*/
48         public long nyid;
49         public int dir;
50         /**
51          * コンストラクタです。初期値から、Idマーカのインスタンスを生成します。
52          * @param i_range_s
53          * @param i_range_e
54          * @param i_patt_size
55          * @throws NyARException
56          */
57         public Item(long i_nyid_range_s,long i_nyid_range_e,double i_patt_size)
58         {
59             super();
60             this.marker_offset.setSquare(i_patt_size);
61             this.nyid_range_s=i_nyid_range_s;
62             this.nyid_range_e=i_nyid_range_e;
63             return;
64         }
65         /**
66          * ここで，曲げ具合を数値化する TODO:
67          */
68         public void mageupdate() {
69             if (false){
70                 double tmp;
71                 double avg = 0;
72                 double avg2 = 0;
73                 int var = 0;
74                 for(int vpi=0;vpi<4;vpi++){
75                     int vv1 = vpi;
76                     int vv2 = (vpi+1)%4;
77                     Point2D n0p = sq.ob_vertex[vv1].getPoint2D();
78                     Point2D n1p = sq.ob_vertex[vv2].getPoint2D();
79                     tmp = n0p.distance(n1p);
80                     avg += tmp;
81                     avg2 += tmp*tmp;
82 //                    System.out.println("["+vv1+"] "+tmp);
83                 }
84                 avg /= 4;
85                 var = (int)((avg2/4)-avg*avg);//分散．．．傾けただけでも値が大きくなる問題がある
86
87                 i_mage = var;
88             } else {
89
90                 i_mage = sq.tmp_mage;
91                 System.out.println("確定MAGE "+i_mage);
```

```
92 //              for(int vpi = 0; vpi < 4 ; vpi++){
93 //                  int vv1 = vpi;
94 //                  int vv2 = (vpi+1)%4;
95 //                  System.out.println("vv1 = "+vv1+"   vv2 = "+vv2);
96 //                  if (sq.ob_vertex[vv1] < sq.ob_vertex[vv2]){
97 //                      System.out.println("    "+(o_vertex[vv2]-o_vertex[vv1]));
98 //                      for(int i= o_vertex[vv1] +1; i<o_vertex[vv2] ; i++){
99 //                          int tmp = getSuisenByCoords(i_coord, o_vertex[vv1],
   o_vertex[vv2], i);
100 //                          sumdist += tmp;
101 //                          o_mage[vv1+1] += tmp;
102 //                      }
103 //                  } else {
104 //                      System.out.println("   rev
   "+(i_coord_num-o_vertex[vv1]+o_vertex[vv2]));
105 //                      for(int i= o_vertex[vv1] +1; i<i_coord_num ; i++){
106 //                          int tmp = getSuisenByCoords(i_coord, o_vertex[vv2],
   o_vertex[vv1], i);
107 //                          sumdist += tmp;
108 //                          o_mage[vv1+1] += tmp;
109 //                      }
110 //                      for(int i= 0; i<o_vertex[vv2] ; i++){
111 //                          int tmp = getSuisenByCoords(i_coord, o_vertex[vv2],
   o_vertex[vv1], i);
112 //                          sumdist += tmp;
113 //                          o_mage[vv1+1] += tmp;
114 //                      }
115 //                  }
116 //              }
117 //              o_mage[0] = sumdist/i_coord_num;
118 //              for(int k=0;k<5;k++){
119 //              System.out.println(" sumdist "+k+" "+ o_mage[k]);
120 //              }*/
121
122             }
123
124         }
125     }
126     private static final long serialVersionUID = -6446466460932931830L;
127     /**輪郭推定器*/
128     private NyIdMarkerPickup _id_pickup;
129     private final NyIdMarkerPattern _id_patt=new NyIdMarkerPattern();
130     private final NyIdMarkerParam _id_param=new NyIdMarkerParam();
131     private final NyIdMarkerDataEncoder_RawBitId _id_encoder=new
   NyIdMarkerDataEncoder_RawBitId();
132     private final NyIdMarkerData_RawBitId _id_data=new NyIdMarkerData_RawBitId();
133     public NyIdList() throws NyARException
134     {
135         this._id_pickup = new NyIdMarkerPickup();
136     }
137     public void prepare()
138     {
139         //nothing to do
140         //sqはtrackingでnull初期化済み
141     }
142     public boolean update(INyARGrayscaleRaster i_raster,SquareStack.Item i_sq) throws
   NyARException
143     {
144         if(!this._id_pickup.pickFromRaster(i_raster.getGsPixelDriver(),i_sq.ob_vertex,
   this._id_patt, this._id_param))
145         {
```

```
146             return false;
147         }
148         if(!this._id_encoder.encode(this._id_patt,this._id_data)){
149             return false;
150         }
151         //IDを検出
152         long s=this._id_data.marker_id;
153         //      System.out.println(s);
154         for(int i=this.size()-1;i>=0;i--){
155             Item target=this.get(i);
156             if(target.nyid_range_s>s || s>target.nyid_range_e)
157             {
158                 continue;
159             }
160             //既に認識済なら無視
161             if(target.lost_count==0){
162                 continue;
163             }
164             //一致したよー。
165             target.nyid=s;
166             //          System.out.println(s); ここは認識初回しかよばれない
167             target.dir=this._id_param.direction;
168             target.sq=i_sq;
169             return true;
170         }
171         return false;
172     }
173     public void finish()
174     {
175         for(int i=this.size()-1;i>=0;i--)
176         {
177             Item target=this.get(i);
178             if(target.sq==null){
179                 continue;
180             }
181 
182             target.mageupdate();
183 
184             if(target.lost_count>0){
185                 //参照はそのままで、dirだけ調整する。
186                 target.lost_count=0;
187                 target.life++;
188                 target.sq.rotateVertexL(4-target.dir);
189                 NyARIntPoint2d.shiftCopy(target.sq.ob_vertex,target.tl_vertex,4-target.dir);
190                 target.tl_center.setValue(target.sq.center2d);
191                 target.tl_rect_area=target.sq.rect_area;
192 
193             }
194         }
195     }
196 }
```

```
 2 * PROJECT: NyARToolkit(Extension)
25 package jp.nyatla.nyartoolkit.markersystem;
26
27 import java.io.FileInputStream;
47
48
49
50
51 /**
52  * このクラスは、マーカベースARの制御クラスです。
53  * 複数のARマーカとNyIDの検出情報の管理機能、撮影画像の取得機能を提供します。
54  * このクラスは、ARToolKit固有の座標系を出力します。他の座標系を出力するときには、継承クラスで変換してください。
55  * レンダリングシステム毎にクラスを派生させて使います。Javaの場合には、OpenGL用の{@link
    NyARGlMarkerSystem}クラスがあります。
56  */
57 public class NyARMarkerSystem extends NyARSingleCameraSystem
58 {
59     /** 定数値。自動敷居値を示す値です。 */
60     public final static int THLESHOLD_AUTO=0x7fffffff;
61     /** マーカ消失時の、消失までのﾃｨﾚｲ(フレーム数)の初期値です。*/
62     public final static int LOST_DELAY_DEFAULT=5;
63
64
65     private static final int MASK_IDTYPE=0x7ffff000;
66     private static final int MASK_IDNUM =0x00000fff;
67     private static final int IDTYPE_ARTK=0x00000000;
68     private static final int IDTYPE_NYID=0x00001000;
69     private static final int IDTYPE_PSID=0x00002000;
70
71     protected INyARMarkerSystemSquareDetect _sqdetect;
72     private int _last_gs_th;
73     private int _bin_threshold=THLESHOLD_AUTO;
74     private TrackingList _tracking_list;
75     private ARMarkerList _armk_list;
76     private NyIdList _idmk_list;
77     private ARPlayCardList _psmk_list;
78     private int lost_th=5;
79     private INyARTransMat _transmat;
80     private final static int INITIAL_MARKER_STACK_SIZE=10;
81
82
83
84
85
86     /**
87      * コンストラクタです。{@link INyARMarkerSystemConfig}を元に、インスタンスを生成します。
88      * @param i_config
89      * 初期化済の{@link MarkerSystem}を指定します。
90      * @throws NyARException
91      */
92     public NyARMarkerSystem(INyARMarkerSystemConfig i_config) throws NyARException
93     {
94         super(i_config.getNyARParam());
95         this.initInstance(i_config);
96
97         this._armk_list=new ARMarkerList();
98         this._idmk_list=new NyIdList();
99         this._psmk_list=new ARPlayCardList();
100        this._tracking_list=new TrackingList();
101        this._transmat=i_config.createTransmatAlgorism();
102        //同時に判定待ちにできる矩形の数
```

```
103         this._on_sq_handler=new
    OnSquareDetect(i_config,this._armk_list,this._idmk_list,this._psmk_list,this._tracking_list,
    INITIAL_MARKER_STACK_SIZE);
104     }
105     protected void initInstance(INyARMarkerSystemConfig i_ref_config) throws NyARException
106     {
107         this._sqdetect=new SquareDetect(i_ref_config);
108         this._hist_th=i_ref_config.createAutoThresholdArgorism();
109     }
110
111
112     /**
113      * この関数は、1個のIdマーカをシステムに登録して、検出可能にします。
114      * 関数はマーカに対応したID値（ハンドル値）を返します。
115      * @param i_id
116      * 登録するNyIdマーカのid値
117      * @param i_marker_size
118      * マーカの四方サイズ[mm]
119      * @return
120      * マーカID（ハンドル）値。この値はIDの値ではなく、マーカのハンドル値です。
121      * @throws NyARException
122      */
123     public int addNyIdMarker(long i_id,double i_marker_size) throws NyARException
124     {
125         return this.addNyIdMarker(i_id,i_id, i_marker_size);
126     }
127     /**
128      * この関数は、1個の範囲を持つidマーカをシステムに登録して、検出可能にします。
129      * インスタンスは、i_id_s<=n<=i_id_eの範囲にあるマーカを検出します。
130      * 例えば、1番から5番までのマーカを検出する場合に使います。
131      * 関数はマーカに対応したID値（ハンドル値）を返します。
132      * @param i_id_s
133      * Id範囲の開始値
134      * @param i_id_e
135      * Id範囲の終了値
136      * @param i_marker_size
137      * マーカの四方サイズ[mm]
138      * @return
139      * マーカID（ハンドル）値。この値はNyIDの値ではなく、マーカのハンドル値です。
140      * @throws NyARException
141      */
142     public int addNyIdMarker(long i_id_s,long i_id_e,double i_marker_size) throws
    NyARException
143     {
144         NyIdList.Item target=new NyIdList.Item(i_id_s,i_id_e,i_marker_size);
145         if(!this._idmk_list.add(target)){
146             throw new NyARException();
147         }
148         this._tracking_list.add(target);
149         this._on_sq_handler.setMaxDetectMarkerCapacity(this._tracking_list.size());
150         return (this._idmk_list.size()-1)|IDTYPE_NYID;
151     }
152     /**
153      * この関数は、1個の範囲を持つARプレイマーカをシステムに登録して、検出可能にします。
154      * インスタンスは、i_id_s<=n<=i_id_eの範囲にあるマーカを検出します。
155      * 例えば、1番から5番までのマーカを検出する場合に使います。
156      * 関数はマーカに対応したID値（ハンドル値）を返します。
157      * @param i_id_s
158      * Id範囲の開始値 (1<=n<=6)
159      * @param i_id_e
```

```
160      * Id範囲の終了値 (1<=n<=6)
161      * @param i_marker_size
162      * マーカの四方サイズ[mm]
163      * @return
164      * マーカID(ハンドル)値。この値はIDの値ではなく、マーカのハンドル値です。
165      * @throws NyARException
166      */
167     public int addPsARPlayCard(int i_id_s,int i_id_e,double i_marker_size) throws
    NyARException
168     {
169         assert(i_id_s>0 && i_id_s<=6);
170         assert(i_id_e>0 && i_id_e<=6);
171         ARPlayCardList.Item target=new ARPlayCardList.Item(i_id_s,i_id_e,i_marker_size);
172         if(!this._psmk_list.add(target)){
173             throw new NyARException();
174         }
175         this._tracking_list.add(target);
176         this._on_sq_handler.setMaxDetectMarkerCapacity(this._tracking_list.size());
177         return (this._psmk_list.size()-1)|IDTYPE_PSID;
178     }
179     /**
180      * この関数は、1個のARプレイマーカをシステムに登録して、検出可能にします。
181      * 関数はマーカに対応したID値(ハンドル値)を返します。
182      * @param i_id
183      * PSARプレイマーカのID。1-6までの数値です。
184      * @param i_marker_size
185      * マーカの四方サイズ[mm]
186      * @return
187      * マーカID(ハンドル)値。この値はIDの値ではなく、マーカのハンドル値です。
188      * @throws NyARException
189      */
190     public int addPsARPlayCard(int i_id,double i_marker_size) throws NyARException
191     {
192         return this.addPsARPlayCard(i_id,i_id,i_marker_size);
193     }
194     /**
195      * この関数は、ARToolKitスタイルのマーカーを登録します。
196      * @param i_code
197      * 登録するマーカパターンオブジェクト
198      * @param i_patt_edge_percentage
199      * エッジ割合。ARToolkitと同じ場合は25を指定します。
200      * @param i_marker_size
201      * マーカの平方サイズ[mm]
202      * @return
203      * マーカID(ハンドル)値。
204      * @throws NyARException
205      */
206     public int addARMarker(NyARCode i_code,int i_patt_edge_percentage,double i_marker_size)
    throws NyARException
207     {
208         ARMarkerList.Item target=new
    ARMarkerList.Item(i_code,i_patt_edge_percentage,i_marker_size);
209         if(!this._armk_list.add(target)){
210             throw new NyARException();
211         }
212         this._tracking_list.add(target);
213         this._on_sq_handler.setMaxDetectMarkerCapacity(this._tracking_list.size());
214         return (this._armk_list.size()-1)| IDTYPE_ARTK;
215     }
216     /**
217      * この関数は、ARToolKitスタイルのマーカーをストリームから読みだして、登録します。
```

```
218      * @param i_stream
219      * マーカデータを読み出すストリーム
220      * @param i_patt_edge_percentage
221      * エッジ割合。ARToolkitと同じ場合は25を指定します。
222      * @param i_marker_size
223      * マーカの平方サイズ[mm]
224      * @return
225      * マーカID(ハンドル)値。
226      * @throws NyARException
227      */
228     public int addARMarker(InputStream i_stream,int i_patt_resolution,int
    i_patt_edge_percentage,double i_marker_size) throws NyARException
229     {
230         NyARCode
    c=NyARCode.createFromARPattFile(i_stream,i_patt_resolution,i_patt_resolution);
231         return this.addARMarker(c, i_patt_edge_percentage, i_marker_size);
232     }
233     /**
234      * この関数は、ARToolKitスタイルのマーカーをファイルから読みだして、登録します。
235      * @param i_stream
236      * マーカデータを読み出すストリーム
237      * @param i_patt_edge_percentage
238      * エッジ割合。ARToolkitと同じ場合は25を指定します。
239      * @param i_marker_size
240      * マーカの平方サイズ[mm]
241      * @return
242      * マーカID(ハンドル)値。
243      * @throws NyARException
244      */
245     public int addARMarker(String i_file_name,int i_patt_resolution,int
    i_patt_edge_percentage,double i_marker_size) throws NyARException
246     {
247         try{
248             NyARCode c=NyARCode.createFromARPattFile(new
    FileInputStream(i_file_name),i_patt_resolution,i_patt_resolution);
249             return this.addARMarker(c,i_patt_edge_percentage, i_marker_size);
250         }catch(Exception e){
251             throw new NyARException(e);
252         }
253     }
254     /**
255      * この関数は、画像からARマーカパターンを生成して、登録します。
256      * ビットマップ等の画像から生成したパターンは、撮影画像から生成したパターンファイルと比較して、撮影画像の色調
    変化に弱くなります。
257      * 注意してください。
258      * @param i_raster
259      * マーカ画像を格納したラスタオブジェクト
260      * @param i_patt_resolution
261      * マーカの解像度
262      * @param i_patt_edge_percentage
263      * マーカのエッジ領域のサイズ。マーカパターンは、i_rasterからエッジ領域を除いたパターンから生成します。
264      * ARToolKitスタイルの画像を用いる場合は、25を指定します。
265      * @param i_marker_size
266      * マーカの平方サイズ[mm]
267      * @return
268      * マーカID(ハンドル)値。
269      * @throws NyARException
270      */
271     public int addARMarker(INyARRgbRaster i_raster,int i_patt_resolution,int
    i_patt_edge_percentage,double i_marker_size) throws NyARException
```

```
272     {
273         NyARCode c=new NyARCode(i_patt_resolution,i_patt_resolution);
274         NyARIntSize s=i_raster.getSize();
275         //ラスタからマーカパターンを切り出す。
276         INyARPerspectiveCopy pc=
    (INyARPerspectiveCopy)i_raster.createInterface(INyARPerspectiveCopy.class);
277         NyARRgbRaster tr=new NyARRgbRaster(i_patt_resolution,i_patt_resolution);
278         pc.copyPatt(0,0,s.w,0,s.w,s.h,0,s.h,i_patt_edge_percentage,
    i_patt_edge_percentage,4, tr);
279         //切り出したパターンをセット
280         c.setRaster(tr);
281         return this.addARMarker(c,i_patt_edge_percentage,i_marker_size);
282     }
283 
284 
285     /**
286      * この関数は、マーカIDに対応するマーカが検出されているかを返します。
287      * @param i_id
288      * マーカID(ハンドル)値。
289      * @return
290      * マーカを検出していればtrueを返します。
291      * @throws NyARException
292      */
293     public boolean isExistMarker(int i_id) throws NyARException
294     {
295         return this.getLife(i_id)>0;
296     }
297     /**
298      * この関数は、ARマーカの最近の一致度を返します。
299      * {@link #isExistMarker(int)}がtrueの時にだけ使用できます。
300      * 値は初期の一致度であり、トラッキング中は変動しません。
301      * @param i_id
302      * マーカID(ハンドル)値。
303      * @return
304      * 0&lt;n&lt;1の一致度。
305      */
306     public double getConfidence(int i_id) throws NyARException
307     {
308         if((i_id & MASK_IDTYPE)==IDTYPE_ARTK){
309             //ARマーカ
310             return this._armk_list.get(i_id &MASK_IDNUM).cf;
311         }
312         //Idマーカ？
313         throw new NyARException();
314     }
315     /**
316      * この関数で、マーカの曲げ度合いを取得
317      *
318      * @param i
319      * @return
320      */
321     public int getMarkerMageMatrix(int i_id) throws NyARException {
322         if((i_id & MASK_IDTYPE)==IDTYPE_NYID){
323             //Idマーカ
324             if (this._idmk_list==null) return 0;
325             else if (this._idmk_list.get(i_id &MASK_IDNUM)==null) return 0;
326             else return this._idmk_list.get(i_id &MASK_IDNUM).i_mage;
327         }
328         //ARマーカ？
329         throw new NyARException();
330     }
```

```
331
332     /**
333      * この関数は、NyIdマーカのID値を返します。
334      * 範囲指定で登録したNyIdマーカから、実際のIDを得るために使います。
335      * @param i_id
336      * マーカID(ハンドル)値。
337      * @return
338      * 現在のNyIdの値
339      * @throws NyARException
340      */
341     public long getNyId(int i_id) throws NyARException
342     {
343         if((i_id & MASK_IDTYPE)==IDTYPE_NYID){
344             //Idマーカ
345             return this._idmk_list.get(i_id &MASK_IDNUM).nyid;
346         }
347         //ARマーカ？
348         throw new NyARException();
349     }
350     /**
351      * この関数は、現在の2値化敷居値を返します。
352      * 自動敷居値を選択している場合は、直近に検出した敷居値を返します。
353      * @return
354      * 敷居値(0-255)
355      */
356     public int getCurrentThreshold()
357     {
358         return this._last_gs_th;
359     }
360     /**
361      * この関数は、マーカのライフ値を返します。
362      * ライフ値は、フレーム毎に加算される寿命値です。
363      * @param i_id
364      * マーカID(ハンドル)値。
365      * @return
366      * ライフ値
367      */
368     public long getLife(int i_id) throws NyARException
369     {
370         switch(i_id & MASK_IDTYPE)
371         {
372         case IDTYPE_ARTK:
373             return this._armk_list.get(i_id & MASK_IDNUM).life;
374         case IDTYPE_NYID:
375             return this._idmk_list.get(i_id & MASK_IDNUM).life;
376         case IDTYPE_PSID:
377             return this._psmk_list.get(i_id & MASK_IDNUM).life;
378         default:
379             throw new NyARException();
380         }
381     }
382     /**
383      * この関数は、マーカの消失カウンタの値を返します。
384      * 消失カウンタの値は、マーカを一時的にロストした時に加算される値です。再度検出した時に0にリセットされます。
385      * @param i_id
386      * マーカID(ハンドル)値。
387      * @return
388      * 消失カウンタの値
389      * @throws NyARException
390      */
391     public long getLostCount(int i_id) throws NyARException
```

```
392     {
393         switch(i_id & MASK_IDTYPE)
394         {
395         case IDTYPE_ARTK:
396             return this._armk_list.get(i_id & MASK_IDNUM).lost_count;
397         case IDTYPE_NYID:
398             return this._idmk_list.get(i_id & MASK_IDNUM).lost_count;
399         case IDTYPE_PSID:
400             return this._psmk_list.get(i_id & MASK_IDNUM).lost_count;
401         default:
402             throw new NyARException();
403 
404         }
405     }
406     /**
407      * この関数は、スクリーン座標点をマーカ平面の点に変換します。
408      * {@link #isExistMarker(int)}がtrueの時にだけ使用できます。
409      * @param i_id
410      * マーカID(ハンドル)値。
411      * @param i_x
412      * 変換元のスクリーン座標
413      * @param i_y
414      * 変換元のスクリーン座標
415      * @param i_out
416      * 結果を格納するオブジェクト
417      * @return
418      * 結果を格納したi_outに設定したオブジェクト
419      */
420     public NyARDoublePoint3d getMarkerPlanePos(int i_id,int i_x,int i_y,NyARDoublePoint3d
    i_out) throws NyARException
421     {
422         this._frustum.unProjectOnMatrix(i_x, i_y,this.getMarkerMatrix(i_id),i_out);
423         return i_out;
424     }
425     private NyARDoublePoint3d _wk_3dpos=new NyARDoublePoint3d();
426     /**
427      * この関数は、マーカ座標系の点をスクリーン座標へ変換します。
428      * {@link #isExistMarker(int)}がtrueの時にだけ使用できます。
429      * @param i_id
430      * マーカID(ハンドル)値。
431      * @param i_x
432      * マーカ座標系のX座標
433      * @param i_y
434      * マーカ座標系のY座標
435      * @param i_z
436      * マーカ座標系のZ座標
437      * @param i_out
438      * 結果を格納するオブジェクト
439      * @return
440      * 結果を格納したi_outに設定したオブジェクト
441      */
442     public NyARDoublePoint2d getScreenPos(int i_id,double i_x,double i_y,double
    i_z,NyARDoublePoint2d i_out) throws NyARException
443     {
444         NyARDoublePoint3d _wk_3dpos=this._wk_3dpos;
445         this.getMarkerMatrix(i_id).transform3d(i_x, i_y, i_z,_wk_3dpos);
446         this._frustum.project(_wk_3dpos,i_out);
447         return i_out;
448     }
449     private NyARDoublePoint3d[] __pos3d=NyARDoublePoint3d.createArray(4);
450     private NyARDoublePoint2d[] __pos2d=NyARDoublePoint2d.createArray(4);
```

```
451
452
453     /**
454      * この関数は、マーカ平面上の任意の4点で囲まれる領域から、画像を射影変換して返します。
455      * {@link #isExistMarker(int)}がtrueの時にだけ使用できます。
456      * @param i_id
457      * マーカID(ハンドル)値。
458      * @param i_sensor
459      * 画像を取得するセンサオブジェクト。通常は{@link #update(NyARSensor)}関数に入力したものと同じものを指
    定します。
460      * @param i_x1
461      * 頂点1[mm]
462      * @param i_y1
463      * 頂点1[mm]
464      * @param i_x2
465      * 頂点2[mm]
466      * @param i_y2
467      * 頂点2[mm]
468      * @param i_x3
469      * 頂点3[mm]
470      * @param i_y3
471      * 頂点3[mm]
472      * @param i_x4
473      * 頂点4[mm]
474      * @param i_y4
475      * 頂点4[mm]
476      * @param i_raster
477      * 取得した画像を格納するオブジェクト
478      * @return
479      * 結果を格納したi_rasterオブジェクト
480      * @throws NyARException
481      */
482     public INyARRgbRaster getMarkerPlaneImage(
483         int i_id,
484         NyARSensor i_sensor,
485         double i_x1,double i_y1,
486         double i_x2,double i_y2,
487         double i_x3,double i_y3,
488         double i_x4,double i_y4,
489         INyARRgbRaster i_raster) throws NyARException
490     {
491         NyARDoublePoint3d[] pos  = this.__pos3d;
492         NyARDoublePoint2d[] pos2 = this.__pos2d;
493         NyARDoubleMatrix44 tmat=this.getMarkerMatrix(i_id);
494         tmat.transform3d(i_x1, i_y1,0,  pos[1]);
495         tmat.transform3d(i_x2, i_y2,0,  pos[0]);
496         tmat.transform3d(i_x3, i_y3,0,  pos[3]);
497         tmat.transform3d(i_x4, i_y4,0,  pos[2]);
498         for(int i=3;i>=0;i--){
499             this._frustum.project(pos[i],pos2[i]);
500         }
501         return i_sensor.getPerspectiveImage(pos2[0].x, pos2[0].y,pos2[1].x,
    pos2[1].y,pos2[2].x, pos2[2].y,pos2[3].x, pos2[3].y,i_raster);
502     }
503     /**
504      * この関数は、マーカ平面上の任意の矩形で囲まれる領域から、画像を射影変換して返します。
505      * {@link #isExistMarker(int)}がtrueの時にだけ使用できます。
506      * @param i_id
507      * マーカID(ハンドル)値。
508      * @param i_sensor
```

```
509      * 画像を取得するセンサオブジェクト。通常は{@link #update(NyARSensor)}関数に入力したものと同じものを指定します。
510      * @param i_l
511      * 矩形の左上点です。
512      * @param i_t
513      * 矩形の左上点です。
514      * @param i_w
515      * 矩形の幅です。
516      * @param i_h
517      * 矩形の幅です。
518      * @param i_raster
519      * 出力先のオブジェクト
520      * @return
521      * 結果を格納したi_rasterオブジェクト
522      * @throws NyARException
523      */
524     public INyARRgbRaster getMarkerPlaneImage(
525         int i_id,
526         NyARSensor i_sensor,
527         double i_l,double i_t,
528         double i_w,double i_h,
529         INyARRgbRaster i_raster) throws NyARException
530     {
531         return this.getMarkerPlaneImage(i_id,i_sensor,i_l+i_w-1,i_t+i_h-1,i_l,i_t+i_h-1,i_l,i_t,i_l+i_w-1,i_t,i_raster);
532     }
533     /**
534      * この関数は、マーカの姿勢変換行列を返します。
535      * マーカID(ハンドル)値。
536      * @return
537      * [readonly]
538      * 姿勢行列を格納したオブジェクト。座標系は、ARToolKit座標系です。
539      */
540     public NyARDoubleMatrix44 getMarkerMatrix(int i_id) throws NyARException
541     {
542         switch(i_id & MASK_IDTYPE)
543         {
544         case IDTYPE_ARTK:
545             return this._armk_list.get(i_id &MASK_IDNUM).tmat;
546         case IDTYPE_NYID:
547             return this._idmk_list.get(i_id &MASK_IDNUM).tmat;
548         case IDTYPE_PSID:
549             return this._psmk_list.get(i_id &MASK_IDNUM).tmat;
550         default:
551             throw new NyARException();
552         }
553     }
554     /**
555      * この関数は、マーカの4頂点の、スクリーン上の二次元座標を返します。
556      * @param i_id
557      * マーカID(ハンドル)値。
558      * @return
559      * [readonly]
560      */
561     public NyARIntPoint2d[] getMarkerVertex2D(int i_id) throws NyARException
562     {
563         switch(i_id & MASK_IDTYPE)
564         {
565         case IDTYPE_ARTK:
566             return this._armk_list.get(i_id &MASK_IDNUM).tl_vertex;
```

```
567         case IDTYPE_NYID:
568             return this._idmk_list.get(i_id &MASK_IDNUM).tl_vertex;
569         case IDTYPE_PSID:
570             return this._psmk_list.get(i_id &MASK_IDNUM).tl_vertex;
571         default:
572             throw new NyARException();
573         }
574     }
575     /**
576      * この関数は、2値化敷居値を設定します。
577      * @param i_th
578      * 2値化敷居値。{@link NyARMarkerSystem#THLESHOLD_AUTO}を指定すると、自動調整になります。
579      */
580     public void setBinThreshold(int i_th)
581     {
582         this._bin_threshold=i_th;
583     }
584     /**
585      * この関数は、ARマーカ検出の、敷居値を設定します。
586      * ここで設定した値以上の一致度のマーカを検出します。
587      * @param i_val
588      * 敷居値。0.0&lt;n&lt;1.0の値を指定すること。
589      */
590     public void setConfidenceThreshold(double i_val)
591     {
592         this._armk_list.setConficenceTh(i_val);
593     }
594     /**
595      * この関数は、消失時のディレイ値を指定します。
596      * デフォルト値は、{@link NyARMarkerSystem#LOST_DELAY_DEFAULT}です。
597      * MarkerSystemは、ここで指定した回数を超えて連続でマーカを検出できないと、マーカが消失したと判定します。
598      * @param i_delay
599      * 回数を指定します。
600      */
601     public void setLostDelay(int i_delay)
602     {
603         this.lost_th=i_delay;
604     }
605     private long _time_stamp=-1;
606     protected INyARHistogramAnalyzer_Threshold _hist_th;
607     private OnSquareDetect _on_sq_handler;
608     /**
609      * この関数は、入力したセンサ入力値から、インスタンスの状態を更新します。
610      * 関数は、センサオブジェクトから画像を取得して、マーカ検出、一致判定、トラッキング処理を実行します。
611      * @param i_sensor
612      * {@link MarkerSystem}に入力する画像を含むセンサオブジェクト。
613      * @throws NyARException
614      */
615 //  TODO:
616     public void update(NyARSensor i_sensor) throws NyARException
617     {
618         long time_stamp=i_sensor.getTimeStamp();
619         //センサのタイムスタンプが変化していなければ何もしない。
620         if(this._time_stamp==time_stamp){
621             return;
622         }
623         int
    th=this._bin_threshold==THLESHOLD_AUTO?this._hist_th.getThreshold(i_sensor.getGsHistogram()
    ):this._bin_threshold;
624         //解析
625         this._tracking_list.prepare();
```

```
626         this._idmk_list.prepare();
627         this._armk_list.prepare();
628         this._psmk_list.prepare();
629         //検出
630         this._on_sq_handler.prepare(i_sensor.getPerspectiveCopy(),i_sensor.getGsImage(),th);
631         this._sqdetect.detectMarkerCb(i_sensor,th,this._on_sq_handler);
632 
633         //検出結果の反映処理
634         this._tracking_list.finish();
635         this._armk_list.finish();
636         this._idmk_list.finish();
637         this._psmk_list.finish();
638         //期限切れチェック
639         for(int i=this._tracking_list.size()-1;i>=0;i--){
640             TMarkerData item=this._tracking_list.get(i);
641             if(item.lost_count>this.lost_th){
642                 //連続で検出できなかった場合
643                 item.life=0;//活性off
644             }else if(item.sq!=null){
645                 //直前のsqを検出できた場合
646 //                System.out.println("[Tracking]");
647                 if(!this._transmat.transMatContinue(item.sq,item.marker_offset,item.tmat,it
    em.last_param.last_error,item.tmat,item.last_param))
648                 {
649                     if(!this._transmat.transMat(item.sq,item.marker_offset,item.tmat,item.l
    ast_param)){
650                         item.life=0;//活性off
651                     }
652                 }
653             }
654         }
655         //各ターゲットの更新
656         for(int i=this._armk_list.size()-1;i>=0;i--){
657             TMarkerData target=this._armk_list.get(i);
658             if(target.lost_count==0){
659                 target.time_stamp=time_stamp;
660                 //lifeが1(開始時検出のときのみ)
661                 if(target.life!=1){
662                     continue;
663                 }
664 //                System.out.println("[ARMarker]");
665                 this._transmat.transMat(target.sq,target.marker_offset,target.tmat,target.l
    ast_param);
666             }
667         }
668         for(int i=this._idmk_list.size()-1;i>=0;i--){
669             TMarkerData target=this._idmk_list.get(i);
670             if(target.lost_count==0){
671                 target.time_stamp=time_stamp;
672                 //lifeが1(開始時検出のときのみ)
673                 if(target.life!=1){
674                     continue;
675                 }
676 //                System.out.println("[NyIdARMarker]");
677                 this._transmat.transMat(target.sq,target.marker_offset,target.tmat,target.l
    ast_param);
678             }
679         }
680         for(int i=this._psmk_list.size()-1;i>=0;i--){
681             TMarkerData target=this._psmk_list.get(i);
682             if(target.lost_count==0){
```

```
683                 target.time_stamp=time_stamp;
684                 //lifeが1(開始時検出のときのみ)
685                 if(target.life!=1){
686                     continue;
687                 }
688 //                System.out.println("[PSARMarker]");
689                 this._transmat.transMat(target.sq,target.marker_offset,target.tmat,target.l
    ast_param);
690             }
691         }
692         //解析/
693         //タイムスタンプを更新
694         this._time_stamp=time_stamp;
695         this._last_gs_th=th;
696     }
697 }
698
699 /**
700  * コールバック関数の隠蔽用クラス。
701  * このクラスは、{@link NyARMarkerSystem}からプライベートに使います。
702  */
703 class OnSquareDetect implements NyARSquareContourDetector.CbHandler
704 {
705     private TrackingList _ref_tracking_list;
706     private ARMarkerList _ref_armk_list;
707     private NyIdList _ref_idmk_list;
708     private ARPlayCardList _ref_psmk_list;
709     public SquareStack _sq_stack;
710     public INyARPerspectiveCopy _ref_input_rfb;
711     public INyARGrayscaleRaster _ref_input_gs;
712     public int _ref_th;
713
714     private NyARCoord2Linear _coordline;
715     public OnSquareDetect(
716         INyARMarkerSystemConfig i_config,
717         ARMarkerList i_armk_list,NyIdList i_idmk_list,ARPlayCardList i_psmk_list,
718         TrackingList i_tracking_list,int i_initial_stack_size) throws NyARException
719     {
720         this._coordline=new
    NyARCoord2Linear(i_config.getNyARParam().getScreenSize(),i_config.getNyARParam().getDistort
    ionFactor());
721         this._ref_armk_list=i_armk_list;
722         this._ref_idmk_list=i_idmk_list;
723         this._ref_psmk_list=i_psmk_list;
724         this._ref_tracking_list=i_tracking_list;
725         //同時に判定待ちにできる矩形の数
726         this._sq_stack=new SquareStack(i_initial_stack_size);
727     }
728     /**
729      * 同時に検出するマーカの最大数を設定します。
730      * 関数は、少なくともi_max_number_of_marker以上のマーカを同時に検出できるようにインスタンスを設定します。
731      * @throws NyARException
732      */
733     public void setMaxDetectMarkerCapacity(int i_max_number_of_marker) throws NyARException
734     {
735         //prepare enough stack size.
736         if(this._sq_stack.getArraySize()<i_max_number_of_marker){
737             this._sq_stack=new SquareStack(i_max_number_of_marker+5);
738         }
739         return;
740     }
```

```
741     /**
742      * {@link #detectMarkerCallback}コール前に1度だけ呼び出してください。
743      * @param i_max_detect_marker
744      * @param i_pcopy
745      * @param i_gs
746      * @param th
747      * @throws NyARException
748      *
749      *  * TODO:
750      *
751      */
752     public void prepare(INyARPerspectiveCopy i_pcopy, INyARGrayscaleRaster i_gs, int th)
753     {
754         this._ref_input_rfb=i_pcopy;
755         this._ref_input_gs=i_gs;
756         this._ref_th=th;
757         // initialize square stack
758         this._sq_stack.clear();
759     }
760     //TODO:
761     public void detectMarkerCallback(NyARIntCoordinates i_coord,int[] i_vertex_index, int[]
    mage) throws NyARException
762     {
763 //      System.out.println("本当にきてる？");
764         //とりあえずSquareスタックを予約
765         SquareStack.Item sq_tmp=this._sq_stack.prePush();
766         //確保できない(1つのdetectorが複数の候補を得る場合(同じARマーカが多くある場合など)に発生することが
    ある。)
767         if(sq_tmp==null){
768             return;
769         }
770         //観測座標点の記録
771         for(int i2=0;i2<4;i2++){
772             sq_tmp.ob_vertex[i2].setValue(i_coord.items[i_vertex_index[i2]]);
773 //          System.out.println(i2+"    "+sq_tmp.ob_vertex[i2].x+"
    "+sq_tmp.ob_vertex[i2].y);
774         }
775         //頂点分布を計算
776         sq_tmp.vertex_area.setAreaRect(sq_tmp.ob_vertex,4);
777         //頂点座標の中心を計算
778         sq_tmp.center2d.setCenterPos(sq_tmp.ob_vertex,4);
779         //矩形面積
780         sq_tmp.rect_area=sq_tmp.vertex_area.w*sq_tmp.vertex_area.h;
781 
782         sq_tmp.tmp_mage = mage[0];//*12/((sq_tmp.vertex_area.w+sq_tmp.vertex_area.h));
783 //      System.out.println(" mage "+sq_tmp.tmp_mage);
784 
785         boolean is_target_marker=false;
786         for(;;){
787             //トラッキング対象か確認する。
788             if(this._ref_tracking_list.update(sq_tmp)){
789                 //トラッキング対象ならブレーク
790                 is_target_marker=true;
791                 break;
792             }
793             //@todo 複数マーカ時に、トラッキング済のarmarkerを探索対象外に出来ない？
794 
795             //nyIdマーカの特定(IDマーカの特定はここで完結する。)
796             if(this._ref_idmk_list.size()>0){
797                 if(this._ref_idmk_list.update(this._ref_input_gs,sq_tmp)){
798                     is_target_marker=true;
```

```
799                     break;//idマーカを特定
800                 }
801             }
802             //PSARマーカの特定(IDマーカの特定はここで完結する。)
803             if(this._ref_psmk_list.size()>0){
804                 if(this._ref_psmk_list.update(this._ref_input_gs,sq_tmp)){
805                     is_target_marker=true;
806                     break;//idマーカを特定
807                 }
808             }
809             //ARマーカの特定
810             if(this._ref_armk_list.size()>0){
811                 //敷居値により1個のマーカに対して複数の候補が見つかることもある。
812                 if(this._ref_armk_list.update(this._ref_input_rfb,sq_tmp)){
813                     is_target_marker=true;
814                     break;
815                 }
816             }
817             break;
818         }
819         //この矩形が検出対象なら、矩形情報を精密に再計算
820         if(is_target_marker){
821             //矩形は検出対象にマークされている。
822             for(int i2=0;i2<4;i2++){
823                 this._coordline.coord2Line(i_vertex_index[i2],i_vertex_index[(i2+1)%4],i_co
    ord,sq_tmp.line[i2]);
824             }
825 //          sq_tmp.mage = mage;
826 
827             for (int i2 = 0; i2 < 4; i2++) {
828                 //直線同士の交点計算
829                 if(!sq_tmp.line[i2].crossPos(sq_tmp.line[(i2 + 3) %
    4],sq_tmp.sqvertex[i2])){
830                     throw new NyARException();//まずない。ありえない。
831                 }
832             }
833         }else{
834             //この矩形は検出対象にマークされなかったので、解除
835             this._sq_stack.pop();
836         }
837     }
838 }
839 
840 
841 
842 
843 class SquareDetect implements INyARMarkerSystemSquareDetect
844 {
845     private NyARSquareContourDetector_Rle _sd;
846     public SquareDetect(INyARMarkerSystemConfig i_config) throws NyARException
847     {
848         this._sd=new NyARSquareContourDetector_Rle(i_config.getScreenSize());
849     }
850     public void detectMarkerCb(NyARSensor i_sensor,int
    i_th,NyARSquareContourDetector.CbHandler i_handler) throws NyARException
851     {
852         this._sd.detectMarker(i_sensor.getGsImage(), i_th,i_handler);
853     }
854 }
855 
856 
```