平成 25 年度　特別研究報告書

閉鎖空間における AR を用いたナビゲーションシステム

龍谷大学　理工学部　情報メディア学科

T100392　北出将司

指導教員　三好　力　教授

内容概要

GPS 技術の発達によってナビゲーションの技術が大きく発展した。ナビゲーションは現在地の場所、方角、地図という 3 つの要素が必要である。この 3 つのうち現在地の特定は非常に重要である。現在 GPS によって現在地を取得している。しかし、GPS には地下、トンネル、遮蔽物が多い場所ではナビゲーションを行えないという問題が 存在する。GPS の通信手段は電波であるため遮蔽物が多い場所では現在地の特定ができない。そこで本研究では GPS による現在地特定を用いることなく、AR マーカーをカメラで撮るという行動とマーカーと内蔵地図とのマッチングを用いた現在地特定を用いてナビゲーションを行う AR マーカーを用いたナビゲーションシステムを提案する。

目次

第1章　はじめに

GPSの技術の発達によってナビゲーションの技術は大きく発展した。一昔前は、星を見て自分の現在地を知り、自分達の向かう方角を導き出す専門の職業の人間がいた。コンパスが発明されてからは航海術が大きく発達し、より正確な世界地図が作成されるようになり、コンパスと世界地図を利用した航海術がその時代で一番正確な航海術であった。しかし第二次世界大戦以降、技術が大きく進歩し人類は空を飛び、地球を抜け宇宙に行くことが可能となり、人工衛星を発明し地球の衛星軌道上に飛ばした。そして多くの人工衛星が今、この瞬間にも私たちの頭上を飛んでいる。

人工衛星には多くの種類がある。気象観測専門の人工衛星、地上の写真を撮る人工衛星、衛星軌道上から遠くの宇宙を覗くための望遠鏡を搭載した人工衛星など多くの種類がある。そして、現在のナビゲーションの技術を大きく発達させたのがGPS衛星である。GPSにより正確に自分の現在の位置を知ることができるようになった。GPSと精度の高いコンパス、細部まで鮮明に描かれた地図、これらの技術により、現在のナビゲーションは成り立っている。

ナビゲーションはより快適に、より使いやすくなるように発展している。技術の進化による携帯端末の小型化により、片手でナビゲーションが行えるようになった。道具は携帯端末のみ。しかも目的地を設定するだけで自動でそこに導くルートを示してくれる。遠くの場所からでもルートを示すことができる。俯瞰でルートを示す方法もあれば、現在の場所からカメラで覗くことにより、カメラの画像の中に矢印がでるように表示されるような技術も開発された。携帯端末の普及により、どの場所からでもナビゲーションを行える。

ナビゲーションを行うにあたって必要なものが3つある。現在地、方角、地図である。現在、GPSを用いて現在地を取得しているがGPSには問題が存在する。一つはGPSには約10mの誤差が存在する。研究者達はこの誤差をより小さくしようと現在も研究が続けられている。二つ目は自分の現在地から電波の得ることができるGPSが一定では無いという点である。GPSによる現在地取得を最高の精度で算出するには、現在の位置から最低3つ以上のGPS衛星が見えていないと取得できない。GPS衛星は衛星であるために地球上を絶え間なく動き続けている。動き続けているため安定した個数のGPS衛星から電波を受けとることができない。そして最後に、GPSによる現在地取得をするにあたっての通信手段が電波であるという点。それはGPSの電波が届かない場所、つまり地下、トンネルなどの遮蔽物が多い場所ではナビゲーションが行えないということである。

つまり、今のナビゲーションはGPSによる現在地取得が大前提であるためGPSによる現在地取得ができなければいかに優れた地図や方位磁石があってもナビゲーションを行うことができないという問題点を含んでいる。

そこでGPSを使わないナビゲーション方法を提案する。それはマーカー型ARを用いたナビゲーションである。この方法を用いることにより様々な問題を解決できる。一番大きなものはGPSを用いずにナビゲーションができるという点である。この方法を用いたナビゲーションはGPSを用いた現在地取得を行わない。順序はARマーカーをある空間に設置する。次にその空間を表した内蔵地図を作成する。そして内蔵地図上でマーカーを設置する。マーカー設置場所は現実に配置したマーカーの場所と同じにする。その作業を複数回行い複数の場所にマーカーを設置する。それと同時に内蔵地図上のマーカーも増やしておく。そして、内蔵地図にはマーカーと別に目的地になる建物の名前を登録しておく。そして、目的地を端末で入力すると内蔵地図上でルート検索を行い、登録されているすべてのマーカーに目的地までのルートの矢印が付与される。そしてマーカーを端末のカメラで写すことにより内蔵地図とリンクした矢印のCGが端末の画面上に表示される。そしてその矢印が現実世界の目的地の方向と同じになっているという流れである。先に説明した方法でナビゲーションを行うことができれば、一切GPSを用いることがなく、すべての処理を地球上で行うことができる。よって今回の研究ではARマーカーを用いたナビゲーションを目指す。

第2章　既存手法

2.1　カーナビゲーション

カーナビゲーションは自車の位置を知ることと、自車の位置を基に目的地への道案内をするのが主な機能である。自車位置を知る仕組みとしては、GPS（全地球測位システム）衛星からの位置情報が基本であるが、GPS だけでは誤差があること、トンネルなどでは無効なことから、地磁気センサー（電子コンパス）とカーナビ内の加速度センサーとタイヤの回転に伴う車速信号などの情報による自立航法と併用することが多い。経路案内は詳細な道路情報を含んだ地図データを内蔵することにより、運転者に対して目的地まで進むべき道を示す。

2.1.1　GPS（Global Positioning System）

GPS とはアメリカが開発した GNSS であり、GNSS とは衛星を用いた測位システムの総称である。GNSS は，地上約 2 万 km のところを飛んでいる衛星からの電波に乗せられた時刻情報を受信し計算することで、地球上における位置（緯度，経度，高さ）を知ることのできるシステムである。3 つの衛星から見える場所では緯度と経度、4 つの衛星から見える場所では緯度と経度と高さがわかる。GPS 衛星は，約 2 万 km 上空の 6 つの軌道に 4 基ずつ，計 24 基配置され，約 12 時間で地球を一周する。

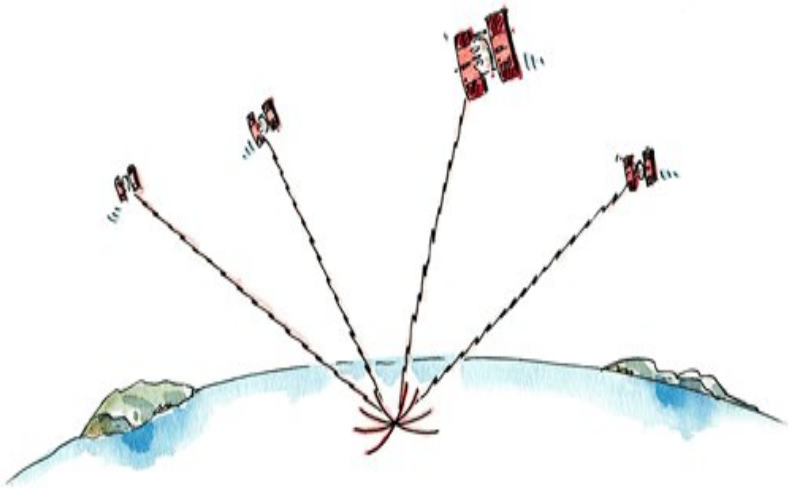


図 1 GPS の位置の割り出している様子

2.1.2　地磁気センサー（電子コンパス）

磁方位センサーは、地磁気を計測することでセンサーが向いている方位を導き出す装置。方位には真方位と磁方位がある。真方位と磁方位は下図のような関係にあり、真方位は緯線（真北）からの角度、磁方位は磁北（地磁気が向いている方向）からの角度を意味している。日本における磁方位は真方位に対して、約6度から7度西にずれており、磁気センサーで検知できるのは磁方位だけである。

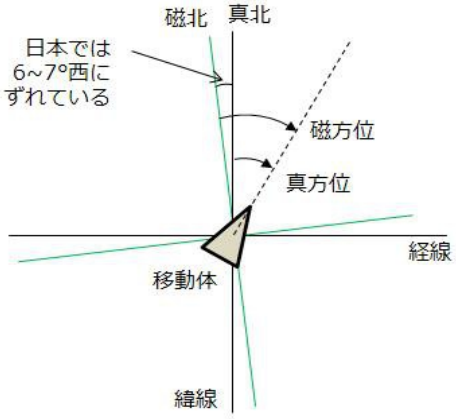


図2 磁方位と方角

2.1.3　加速度センサー（角速度センサー）

加速度センサーはセンサー自体の加速度、つまり速度の変化を検出するセンサーである。重力加速度も検出できるので人や物体の動きや地震などの振動も検出できる。3軸加速度センサーであれば水平状態も検出できる。

角速度センサーは基準軸に対して物体が1s間に何度の回転運動をしているかを検出するセンサーである。別名ジャイロセンサーと呼ばれている。

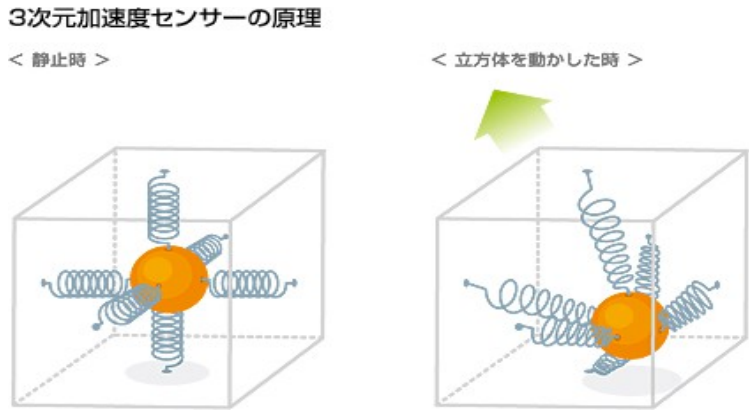


図3　3次元加速度センサーの原理

2.2　ARナビゲーション

ARナビゲーションとは、拡張現実（AR）技術を搭載したナビゲーションシステムのことである。

（例）カロッツェリアサイバーナビで実用化されているこのナビはスカウター、ヘッドアップディスプレイ（HUD）などで構成される。バックミラー手前に取り付けられたスカウターで前方を捉え、運転席のサンバイザー（日除け）部分に取り付けられたヘッドアップディスプレイに情報を重ねて表示する。前方の景色にナビゲーション情報を重ねて表示することで、視線を逸らさずにルート情報や渋滞情報などを参照できる。

図4カロッツェリアサイバーナビ

2.3 AR（Augmented Reality）

拡張現実はバーチャルリアリティ(VR)の変種であり、その時周囲を取り巻く現実環境に情報を付加・削除・強調・減衰させ、文字通り人間から見た現実世界を拡張するものを指す。バーチャルリアリティが人工的に構築された現実感と現実を差し替えるのに対し、拡張現実は現実の一部を改変する技術である。例えばバーチャルリアリティでは、仮想の部屋に居て仮想のテーブルに置かれた仮想のティーポットを見ているかのような五感情報を人に提示するのに対し拡張現実では、例えば人が実際に居る現実の部屋のテーブルの上に、仮想のティーポットが置かれているかのような情報提示を行う。コンピュータにより現実を強化することから強化現実と呼ばれることもある。現実環境を情報ソースとして用いることから必然的にリアルタイム処理を必要とする場合が多い。

図5　カメラに映し出されるCG

2.3.1　ARで使われる位置あわせの技術

・ARにおける整合性

ARにおいて、現実の空間に対してコンピューターが作り出した情報をまるでそこにあるかのように違和感なく提示するためには3つの整合性を実現する必要がある。

- 「幾何学的整合性」
  提示する情報の位置あわせが正しくできているか
- 「時間的整合性」
  環境や見ている人の動きに同期して情報を提示できているか
- 「光学的整合性」
  現実世界と仮想世界の光学現象が正しくリンクしているか

現在のARで1番重要とされているのは幾何学的整合性である。正しい場所にCGを表示されているかどうかがARでは重要だといえる。

2.3.2　位置あわせの技術の分類

- 「センサーを用いる方法」

位置計測のために GPS や無線 LAN 測位、カメラの向きを計測するために方位センサー、ジャイロセンサー、加速度センサーなどを使う。これらはスマートフォンでも用いられている。
室内の専用のスタジオでは、磁気式の位置センサーやモーションキャプチャーを使う場合もある。

例：

図 6 パイオニアのカーナビゲーションシステムのブランド「カロッツェリアサイバーナビ」

- 「マーカーを用いる方法」

マーカーを用いる方法では、専用のパターンを画像認識し画像中の見え方から 3 次元の位置と姿勢を計算する。

例：

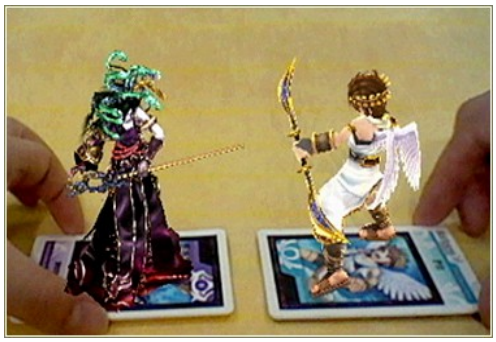


図 7 マーカー型 AR

- 「自然特徴量を用いる方法」

自然特徴量とは、環境に自然に存在する物体の輪郭情報や特徴点のことである。自然特徴量を用いる方法はマーカーのように決まった情報を使わず画像認識を行うため美観を損ねないというメリットがある。

例：

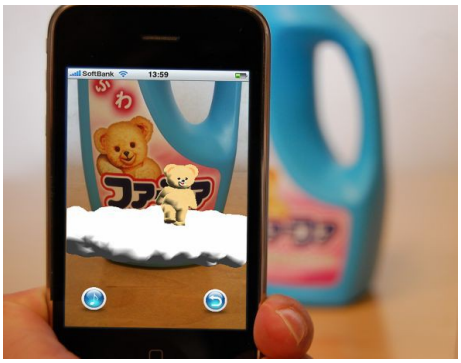


図 8 洗剤メーカーの AR

2.4　地下空間のナビゲーションシステム

現在検討されている方法は GPS アンテナを用いた EZ ナビウォークのシステムである。

GPS を用いて現在位置を把握する EZ ナビウォークなどのナビゲーションサービスは衛星からの電波が届かない地下では利用できない。そのため、実験では屋外に設置したアンテナで GPS からの電波を受信し、地下に再送信する「GPS 再放射システム」を使用している。これにより、地下でも EZ ナビウォークによる歩行案内を可能にする。実験は EZ ナビウォーク対応携帯を持っていれば一般ユーザーでも参加できる。期間中は地下街で専用のメニューから実証実験エリアの店舗が検索できる。

このような方法が現在検討されている。

第3章　提案手法

従来のGPSによる現在地特定を用いたナビゲーションでは地下や障害物の多い場所、要するにGPSの電波が届かない場所ではナビゲーションを行うことができない。そこで私はARマーカーと内蔵地図を用いたARマーカー型ナビゲーションを提案する。マーカーと内蔵地図のマッチングを用いることにより現在地を導き出すことができ、さらにARマーカーのカメラで写すという行動を利用して自分の向いている方向までも利用し使用者を導くことができる。「内蔵地図とマーカーのマッチングを行い、現在地を特定した後はルート検索を行い、目的地までのルートをマーカー上に表示する」というのが全体の流れである。この方法を用いることによりGPSをまったく利用しないナビゲーションシステムが構築できる。以下にこの方法を用いるにおいて重要な要素を示す。

3.1　閉鎖空間内の複数の場所にマーカーを設置

GPSはGPS電波の届く範囲なら現在地が特定できるという利点がある。しかし今回提案する手法はGPSを用いないのでどのような場所でも現在地を特定することができない。それを補う方法として複数の場所にマーカーを設置する方法を提案する。

3.2　同一マーカー上で目的地によって表示させる矢印の向きを変える

すべてのマーカーにナビゲーション機能を持たすことがマーカー型ARを用いたナビゲーションの重要な要素であると考えた。そしてそれを実現するにはすべての場所に存在するマーカーが目的地までのルートを指し示す必要がある。よってそれを実現するために同一のマーカー上でも目的地によって異なる矢印を表示する方法を提案する。

3.3　内蔵地図とマーカーのマッチングによる現在地特定

ナビゲーションにおいて一番重要なことは現在地を特定することである。現在地が分からなければ目的地までの正確なルートを導き出せない。そしてもうひとつ重要なことは自分の向いている方向である。以上の2つの点が正確にわかることにより目的地までの正しいルートを示すことができる。

現在では現在地を特定する方法としてGPSが用いられているがGPSの電波が届かない場所では現在地の特定ができない。よって今回実験では現在地を特定する方法としてマーカーと内蔵地図のマッチングを用いる。内蔵地図上に現実のマーカーの場所と同じ場所にマーカーを配置する。そして現実のマーカーと内蔵地図のマーカーの場所をマッチングさせることにより現在地を特定する方法を提案する。

3.4　　3次元方向のナビゲーション

ナビゲーションというのは自分の進行方向に対して正面、後ろ、右、左を組み合わせた方法で行っている。屋外や平地など上下の空間が無い場合は先に述べた4方向を組み合わせて目的地までたどり着くことができる。しかし、ショッピングモールや上下の空間が多くある地下鉄の駅などでは平行方面の指示では目的地にたどり着くことは難しい。よって上下の空間のナビゲーションも可能にするため3次元方向を向く矢印の方法を提案する。

第4章　実験

今回の実験であるマーカー型ARを用いたナビゲーションを行うにあたって場所は龍谷大学7号館2階の通路および演習室、研究室、事務室を使用した。

4.1　実験環境

4.1.1　ソフトウェア

- Processing1.5.1
  電子アートとビジュアルデザインのためのプログラミング言語
- QuickTime　最新版
  メディアプレーヤー(webカメラで撮った動画の再生に使用)
- メタセコイヤ
  モデリングに特化した3DCG作成ソフト（矢印（CG）の製作のために使用）

4.1.2　ライブラリ

- NyAR4psg
  Processingで使えるマーカーベースのARアプリケーションを作るためにのプログラミングライブラリ
- MqoViewer Library for Processing
  Processingにメタセコイヤで作成した画像を表示するライブラリ
- PpopupMenu
  カメラに映し出された画像にクリックするとメニューが表示されるライブラリ

4.1.3　ハードウェア

- USBカメラ(UCAM-DLM130HSV)
- PC(windows7)

4.1.4　ハードウェアの図

動画を撮るためにwebカメラを付け、Processing1.5.1とQuickTime又ARを使うためのライブラリなどをいれた状態の装置である。実験ではこの装置のカメラにマーカーを映すことによりPC上で矢印が表示される。

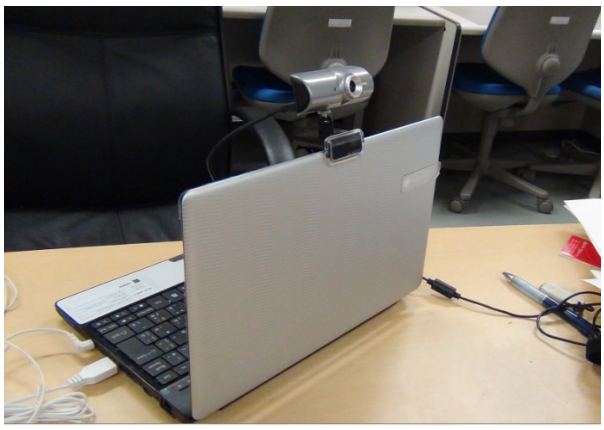


図9　　USBカメラをつけたPC

4.2　マーカー

龍谷大学瀬田キャンパス 7 号館 2 階を実験場所とし、今回は「Hiro」と「人」と「モザイク」の 3 つの模様のマーカーを用意する。

図 10　patthiro　（マーカー）

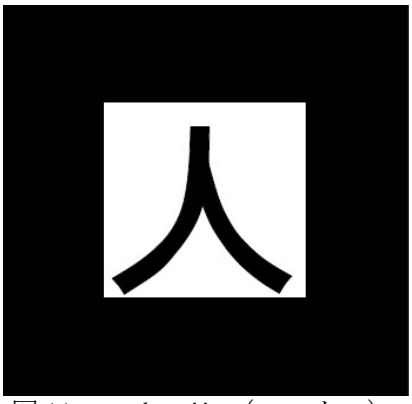


図 11 pattkanji　（マーカー）

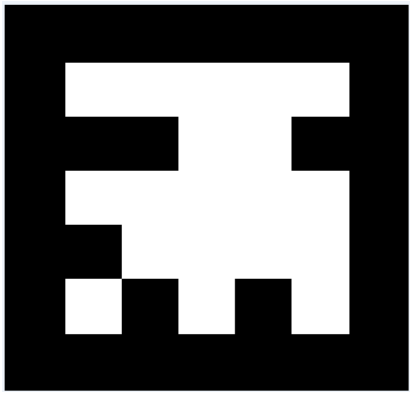


図 12　nyid-m2_id001（マーカー）

4.3　CG

PC 上で表示する矢印をメタセコイヤで作成し、カメラがマーカーを認識したらメタセコイヤで作成した矢印が表示される。

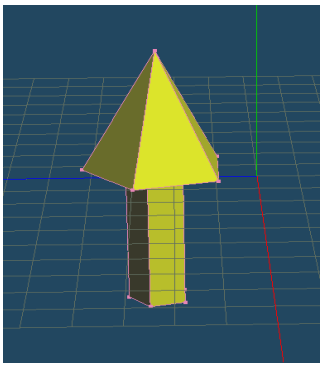


図 13 メタセコイヤで矢印作成

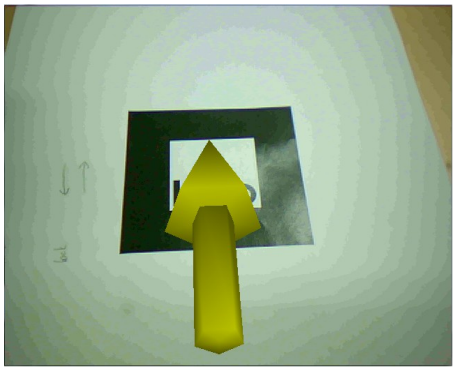


図 14　PC 上で表示される CG

4.3.1　　マーカー型 AR の表示

矢印の向きの変更

Processing における座標系は左手系となっている。つまり、X 軸：親指、Y 軸：人差し指 Z 軸：中指と対応するような座標系である。

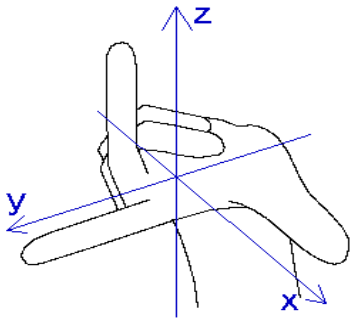


図 15 左手系の座標系(左手)

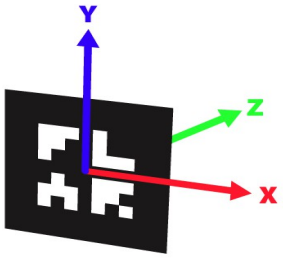


図 16 左手系の座標系(マーカー)

矢印の CG を作るために使用するメタセコイヤの座標軸は図 17 である。上（緑の線）が Y 軸,奥から手前にある線（赤線）は X 軸である。左右に伸びている線（青線）は Z 軸である。

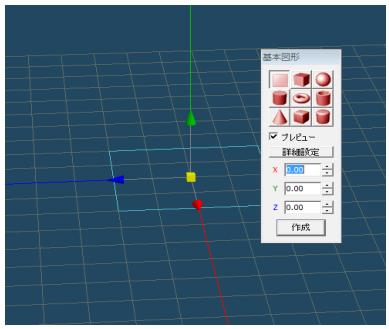


図 17　　メタセコイヤの座標軸

図 13 のように矢印を作成し、角度を変えず（X 軸：0、Y 軸：0、Z 軸：0）に、マーカー上に CG として表示すると図 18 のようになる。

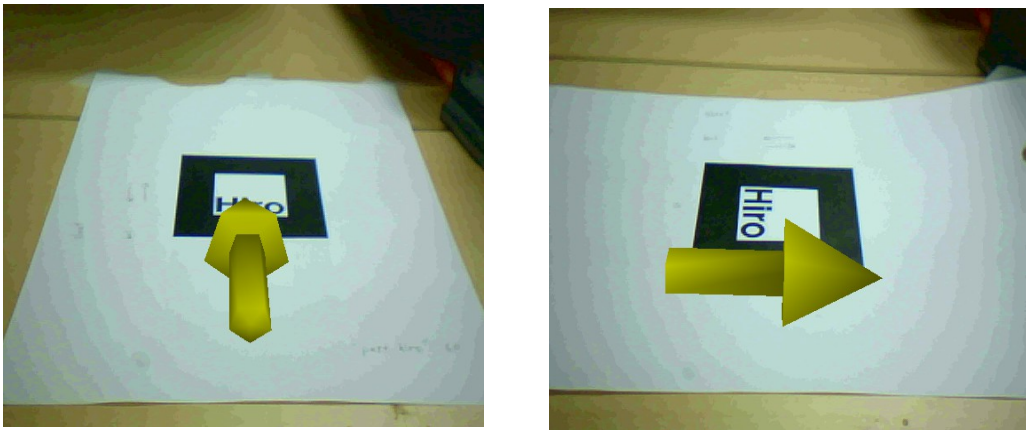


図 18　矢印　（X 軸回転：0、Y 軸回転：0、Z 軸回転：0）

また、X 軸のみ回転させると図 19 のようにマーカーに対して反時計回りに縦回転をする。（マーカーの右側を頭とする）

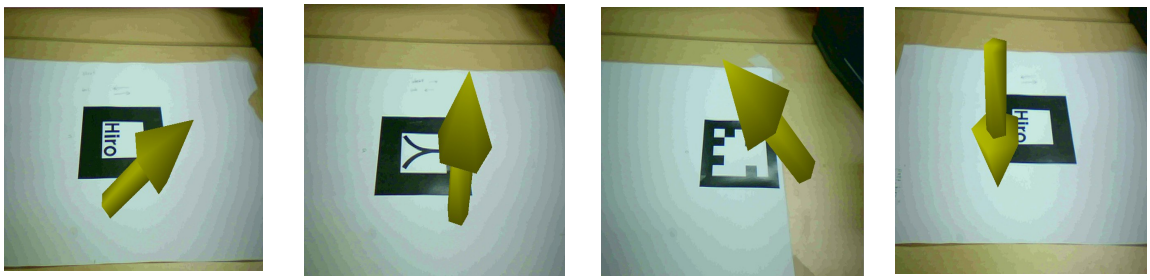


図 19　矢印　（X 軸：（”45 度”、”90 度”、”135 度”、”270 度”）、Y 軸：0、Z 軸：0）

また、Z 軸のみを回転させると図 20 のようにマーカーに対して時計回りに横回転をする。（マーカーの右側を頭とする）

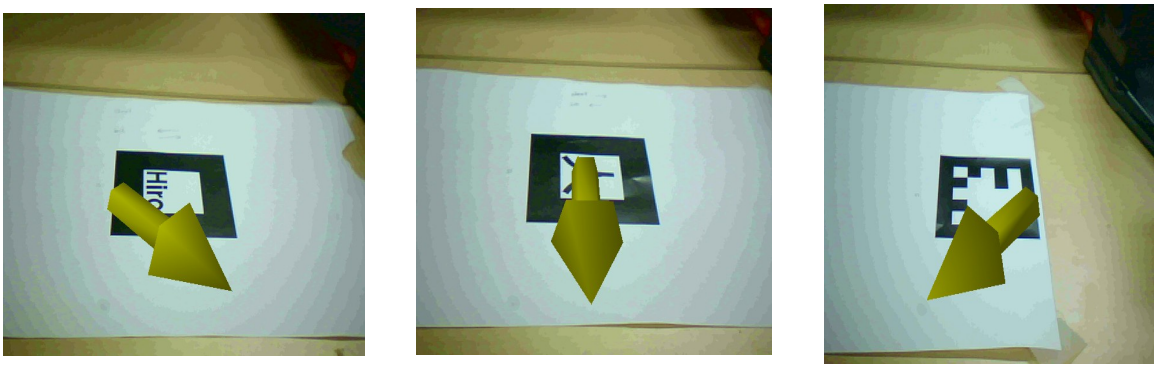


図 20　矢印　（X 軸：0、Y 軸:0、Z 軸:（”45 度”,”90 度”,”135 度”））

Y 軸のみ回転させてみたものを図 21 に示す。

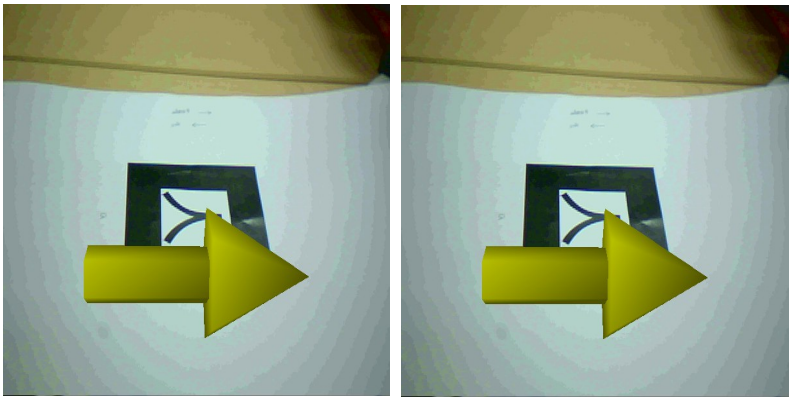


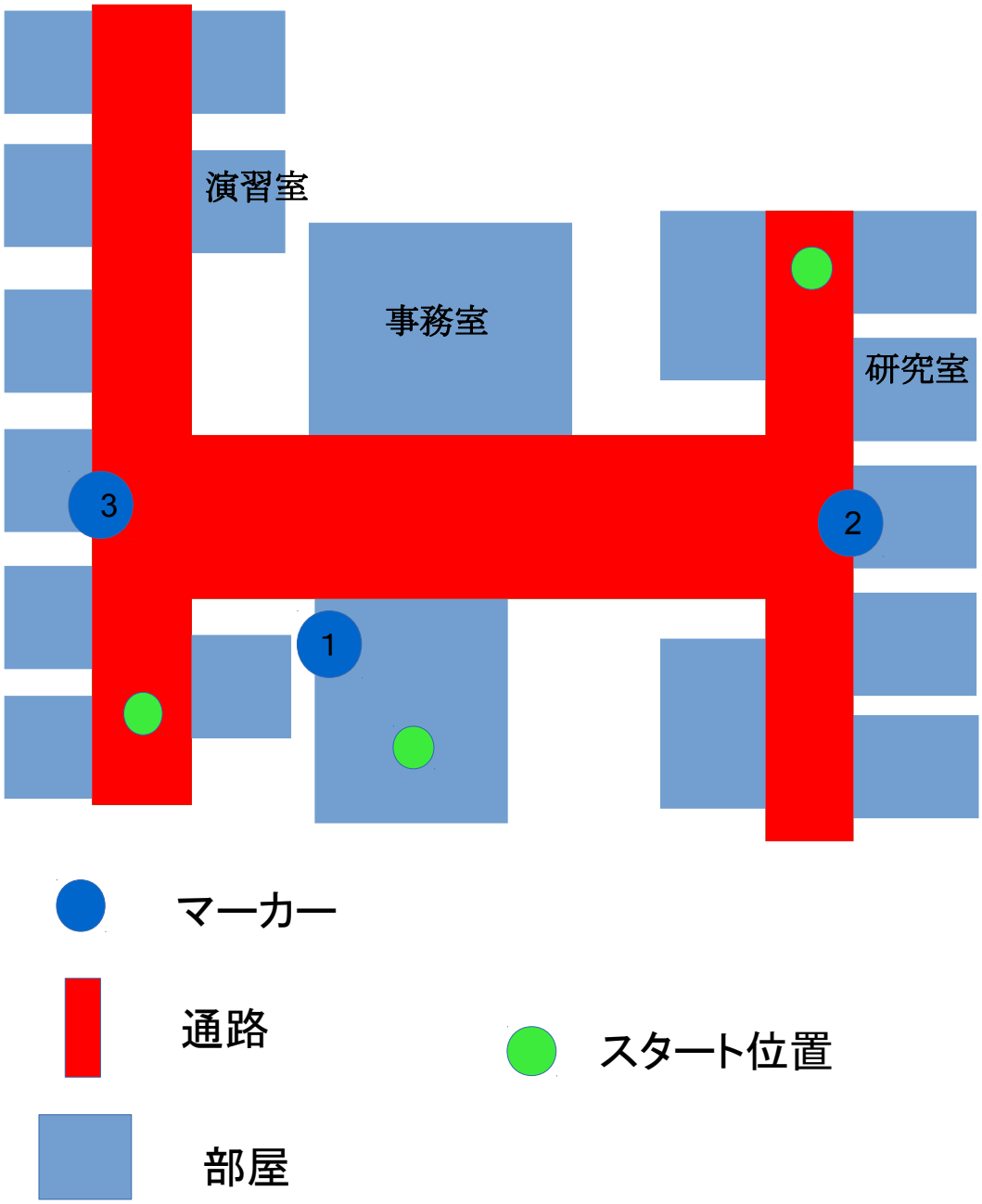
図 21　矢印　（X 軸:0,Y 軸:（”45 度”、”90 度”）,Z 軸:0）

　矢印本体が回転する。目的地の方向を表示したいため今回のナビゲーションの実験において変更する必要がないと考える。

　今回の実験では矢印の向きを変えるにあたって変更するのは X 軸と Z 軸の回転のみとする。この 2 つの軸を変更することで 3 次元の方向で目的地を指す矢印を表示することができる。

4.4　内蔵地図

以下の図のような内蔵地図を作成しマーカーの位置を登録する。
青丸の部分にマーカーを登録する。赤い部分は廊下を水色の部分は部屋をあらわしている。
目的地は今回は分かりやすいように言葉で記しておく。



龍谷大学瀬田キャンパス７号館２階

図 22　　内蔵地図の例

4.5　実験方法

ARマーカーを用いたナビゲーションを実装するために「目的地の方向を指す矢印の表示」、「同一マーカー上で目的地によって異なる向きの矢印の表示」の二つの要素が必要であると考えた。「目的地の方向を指す矢印の表示」を実験１、「同一マーカー上で目的地によって異なる矢印の表示」を実験2で行う。さらに実験1、実験2を行った後、実際のナビゲーションの実験として実験3を行う。

実験3ではナビゲーションを行う場所に[龍谷大学7号館2階]を使用する。実験3.1はスタート場所を[龍谷大学7号館2階]の[右側の通路]とし、目的地を同じ階の[事務室]とする。マーカー①（「hiro」）とマーカー②（「人」）の2つのマーカーを使用する。実験3.2はスタート位置を[龍谷大学7号館2階]の階段とし、目的地を演習室とする。マーカー①（「hiro」）とマーカー③（「モザイク」）の2つのマーカーを使用する。 実験3.3はスタート場所を[龍谷大学7号館2階]の[左側の通路]とし、目的地を同じ階の[研究室]とする。マーカー①（「hiro」）とマーカー②（「人」）とマーカー③（「モザイク」の3つのマーカーを使用する。

実験3.1と実験3.2で「同一マーカー上で目的地によって異なる向きの矢印の表示」を用いたナビゲーション、実験3.3で奥行きを示す矢印の表示を用いた際のナビゲーションの正確性を実験する。

4.6　実験1:「目的地の方向を指す矢印の表示」

マーカーを実空間上に設置し、目的地を選択するメニューで行きたい目的地を選ぶことでマーカー上に表示される矢印が目的地のルートを向いているかを実験する。尚この実験では3つのマーカーをそれぞれ違う場所に設置して実験を行う。図23に実際に使用したマーカーを示す。

図23　実験に使用する3つのマーカー

図24に実験結果を示す。図24でわかる通り、実際に目的地を指し示す矢印の表示に成功した。

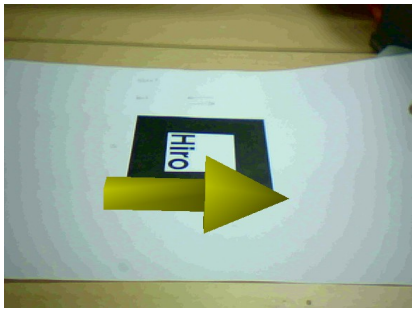


図24　目的地を指す矢印の表示

4.7　実験 2:「同一マーカー上で目的地によって異なる向きの矢印の表示」

実空間にマーカーを設置し、目的地を選択するメニューで行きたい目的地を選びマーカー上に目的地の方向を向いている矢印を表示しておく。その後、目的地を選択するメニューで今選択している目的地とは異なる目的地を選択する。その際表示される矢印は別の目的地の方向を向いているのかを実験する。尚この実験では 3 つのマーカーをそれぞれ違う場所に設置して実験を行う。

図 25、図 26 に実験結果を示す。図 25 は目的地「演習室」、「事務室」の時に垂直方向の矢印を表示、図 26 は目的地「研究室」の時に右側方向の矢印の表示されるように設定した。図 25、図 26 でわかる通り目的地によって異なる矢印の表示に成功した。

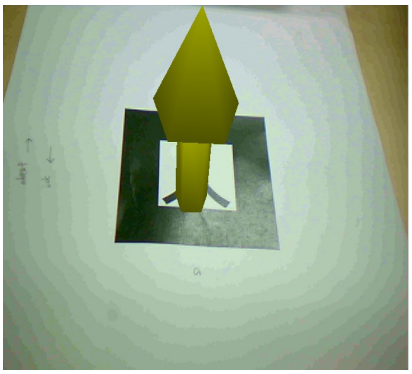


図 25　目的地「演習室」および「事務室」の時

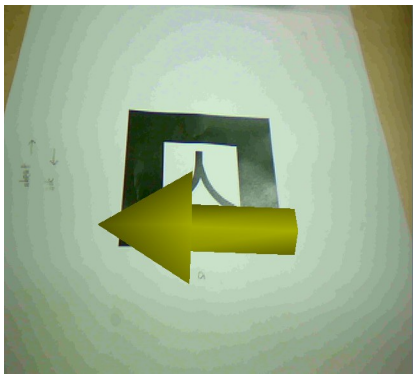


図 26　目的地「研究室」の時

4.8　実験 3：ナビゲーション

実際にナビゲーションの実験を龍谷大学 7 号館 2 階で行う。この実験を行うにあたり実験 1 と実験 2 の実験結果を用いた。

1. 龍谷大学 7 号館 2 階の階段横、右側の通路、左側の通路の 3 点にマーカーを設置する。
2. （階段横：マーカー①、右側の通路：マーカー②、左側の通路：マーカー③とする）
3. 被験者に USB カメラを取り付けた PC を持った状態で階段に立ってもらう。（スタート地点[1]）
4. PC 上で目的地を選んでもらう（今回の目的地は演習室、研究室、事務室の 3 つとする）
5. カメラでマーカーを撮影してもらいマーカー上で表示された矢印に従い、目的地を目指してもらう

今回の実験で

- 実験 1：スタート地点[1](右側の通路)から目的地[事務室]
- 実験 2：スタート地点[2]（階段）から目的地[演習室]
- 実験 3：スタート地点[3](左側の通路)から目的地[研究室]

のナビゲーションを 3 人の被験者にしてもらう

3 人の被験者はそれぞれ別々のスタート地点から別々の目的地を目指してもらう。道中に存在するマーカーを見つけ次第持っている PC のカメラでマーカーを撮影してもらい表示された矢印の向きに従ってもらう。

4.8.1　実験 3.1:スタート地点を[1]（右側の通路）から目的地「事務室」までのルートのナビゲーション

スタート地点を右側の通路にし、マーカーを以下の図の青丸の位置に設置する。それぞれのマーカーの番号は階段の左側の壁をマーカー①、右側の通路をマーカー②、左側の通路をマーカー③とする。目的地は事務室なので、目的地にたどり着くルートはスタートして直進し、マーカー②のところで右折し直進、マーカー①のところで右折をするルートとなる。

スタート地点　　[1]（右側の通路）
目的地　　事務室
使用したマーカー　マーカー①、マーカー②

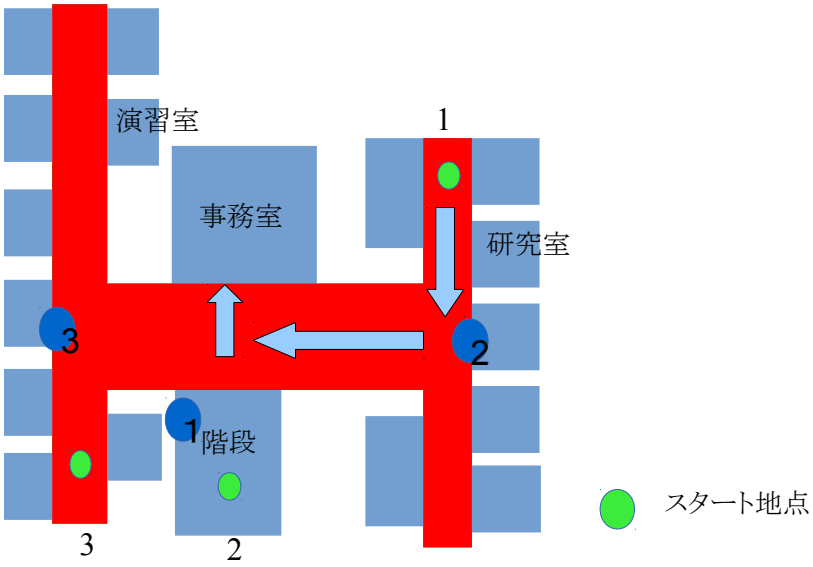


図 27　理想とするルート

図 28　マーカー①を撮影している様子

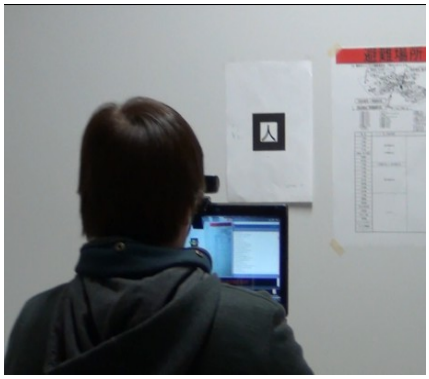


図 29　マーカー②を撮影している様子

4.8.2　実験 3.2：スタート地点を[2]（階段）から目的地「演習室」までのルートのナビゲーション

スタート地点を階段にし、マーカーは実験 3.1 と同じ位置に同じマークを設置し実験を行う。目的地は演習室なので、目的地にたどり着くルートはスタートしてマーカー①のところを左折し直進、マーカー③のところで右折し直進するルートとなる。このルートで目的地にたどり着くにはマーカー①で表示される矢印が実験 3.1 と異なる方向を向いていなければならない。

スタート地点　　[2]（階段）
目的地　　演習室
使用したマーカー　マーカー①、マーカー③

理想とする歩行ルート

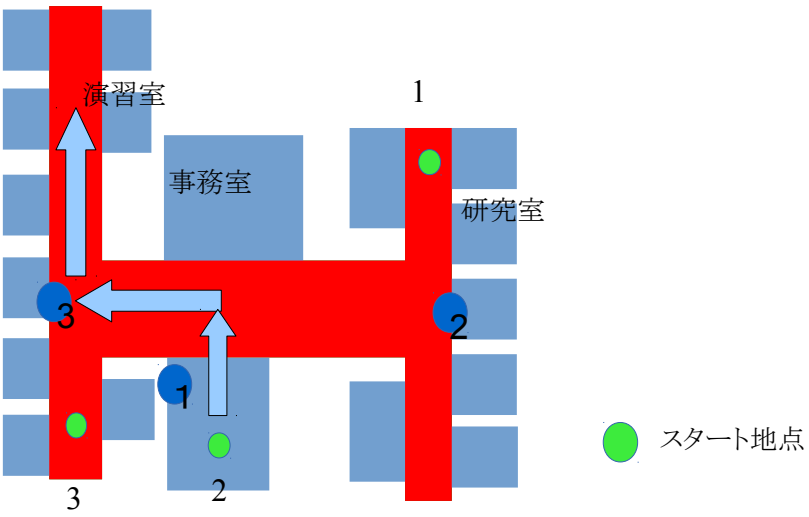


図 30 理想とするルート

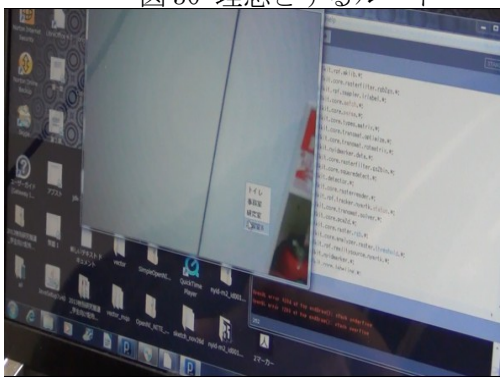


図 31　目的地「演習室」を選択している様子

図 32　目的地「演習室」

4.8.3　実験 3.3:スタート地点を[3]（左側の通路）から目的地「研究室」までのルートをナビゲーション

スタート地点を左側の通路にし、マーカーは実験 3.1 と同じ位置に同じマークを設置し実験を行う。目的地は研究室なので、目的地にたどり着くルートはスタートしてマーカー③のところを右折し直進、マーカー①のところを直進し、マーカー②のところで左折し直進するルートとなる。このルートで目的地まで辿りつくにはマーカー①、②、③が実験 3.1、実験 3.2 と異なる向きの矢印を表示しなければしけない。なおかつ、マーカー 2 は直進を示す向きの矢印を表示しなければいけない。

スタート地点　　[3]（左側の通路）
目的地　　研究室
使用したマーカー　マーカー①、マーカー②、マーカー③

理想とする歩行ルート

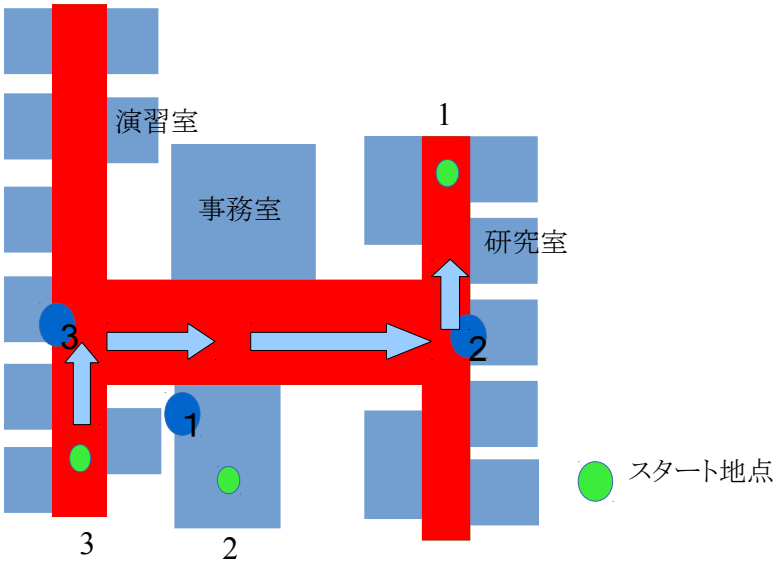


図 33　理想とするルート

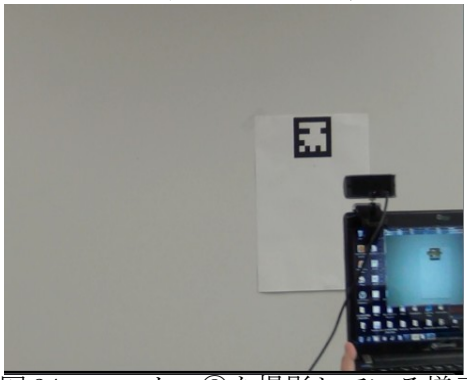


図 34　マーカー③を撮影している様子

図 35　マーカーで映し出される CG

第5章　まとめ

GPS電波の届かないところの現在地の取得を行う方法としてマーカーと内蔵地図のマッチングの方法を考え、マーカーと内蔵地図のマッチングにより算出した現在地の情報を得ることにより、GPSを一切用いることの無いマーカー型ARを用いたナビゲーション方法を提案した。

今回の実験3.1、実験3.2、実験3.3の実験結果によりARマーカーを用いたナビゲーションは可能であると考えることができる。

また、現在のARの技術である自然特徴量を利用したマーカー型ARの技術を用いることによりポスターや絵画などをマーカーとし、それらの媒体からでもナビゲーションが行うことができるなどの多くの応用ができると思われる。今回は平行方向の矢印でナビゲーションを行ったが3次元方向のナビゲーションも実装したいと考えているため今後はポスターや絵画などの自然特徴量を利用したマーカー型ARを用いたナビゲーションを研究して行くと同時にポスターや絵画に表示される矢印を用いて3次元空間のナビゲーションを研究していきたいと考える。

謝辞

本研究を行うにあたり多くの御助言、ご指導をしていただきました。心より深く御礼申しあげます。又、論文の作成のご支援をしていただき心より深く御礼申しあげます。

参考文献

[1] [processingの公式サイト]
http://wwww.processing.org/
[2] [NyAR4psg]
http://nyatla.jp/nyartoolkit/wiki/index.php?NyAR4psg
[3] [ARToolKit Marker Generator Online]
http://flash.taro.org/ar/MarkerGeneratorOnline.html
[4] [NyID Marker maker]
http://sixwish.jp/AR/Marker/idMarker/
[5] [メタセコイヤ]
http://www.metaseq.net/metaseq/
[6] [processing上でメタセコイヤ形式データを表示するライブラリ]
http://www.hyde-ysd.com/reco-memo/mqoview.html
[7] [processing-picking-library]
http://code.google.com/p/processing-picking-library/
[8] [PpopupMenu]
https://githug.com/hixi-hyi/PPopuoMenu
[9] [Quick time 最新版]
http://www.apple.com/jp/quicktime/download/
[10] [WinVDIG1.01]
http://www.eden.net.nz/7/20071008/
[11] [ARプログラミング　processingで作る拡張現実のレシピ　読者サポートページ]
http://kougaku-navi.net/AR-ProgrammingBook/
[12] [ARのカメラ出力]
http://www.d-improvement.jp/learning/processing/2011-b/06.html

[13]橋本　直著,ARプログラミング　processingで作る拡張現実のレシピ
[14]マット・ピアソン著,ジェネラティブ・アート　processingによる実践ガイド
[15]谷尻　かおり著,Processingによる画像処理とグラフィックス

付録

```
import processing.video.*;
import ppopupmenu.*;
import picking.*;

import
jp.nyatla.nyartoolkit.rpf.reality.nyartk.*;
import jp.nyatla.nyartoolkit.core.*;
import jp.nyatla.nyartoolkit.utils.*;
import
jp.nyatla.nyartoolkit.core.labeling.artoolkit.*;
import jp.nyatla.nyartoolkit.core.raster.*;
import
jp.nyatla.nyartoolkit.core.types.stack.*;
import jp.nyatla.nyartoolkit.core.utils.*;
import jp.nyatla.nyartoolkit.core.pickup.*;
import
jp.nyatla.nyartoolkit.rpf.tracker.nyartk.*;
import
jp.nyatla.nyartoolkit.core.rasterfilter.rgb2bin.
*;
import
jp.nyatla.nyartoolkit.core.labeling.rlelabeling
.*;
import
jp.nyatla.nyartoolkit.core.analyzer.raster.*;
import jp.nyatla.nyartoolkit.rpf.mklib.*;
import
jp.nyatla.nyartoolkit.core.rasterfilter.rgb2gs.*
;
import
jp.nyatla.nyartoolkit.rpf.sampler.lrlabel.*;
import jp.nyatla.nyartoolkit.core.match.*;
import jp.nyatla.nyartoolkit.core.param.*;
import
jp.nyatla.nyartoolkit.core.types.matrix.*;
import
jp.nyatla.nyartoolkit.core.transmat.optimize.*
;
import
jp.nyatla.nyartoolkit.core.transmat.rotmatrix.
*;
import
jp.nyatla.nyartoolkit.nyidmarker.data.*;
import
jp.nyatla.nyartoolkit.core.rasterfilter.gs2bin.*
;
import
jp.nyatla.nyartoolkit.core.squaredetect.*;
import jp.nyatla.nyartoolkit.detector.*;
import
jp.nyatla.nyartoolkit.core.rasterreader.*;
import
jp.nyatla.nyartoolkit.rpf.tracker.nyartk.status.
*;
import
jp.nyatla.nyartoolkit.core.transmat.solver.*;
import jp.nyatla.nyartoolkit.core.pca2d.*;
import jp.nyatla.nyartoolkit.core.raster.rgb.*;
import
jp.nyatla.nyartoolkit.core.analyzer.raster.thres
hold.*;
import
jp.nyatla.nyartoolkit.rpf.realitysource.nyartk.
*;
import jp.nyatla.nyartoolkit.nyidmarker.*;
import jp.nyatla.nyartoolkit.core.labeling.*;
import jp.nyatla.nyartoolkit.test.*;
import
jp.nyatla.nyartoolkit.core.transmat.optimize.a
rtoolkit.*;
import
jp.nyatla.nyartoolkit.core.analyzer.histogram.
*;
import
jp.nyatla.nyartoolkit.core.rasterfilter.*;
import jp.nyatla.nyartoolkit.core.transmat.*;
import jp.nyatla.nyartoolkit.*;
import jp.nyatla.nyartoolkit.core.types.*;
import jp.nyatla.nyartoolkit.rpf.utils.*;
import jp.nyatla.nyartoolkit.processor.*;

//import codeanticode.gsvideo.*;

/* リスト 3.4 AR マーカの上に MQO 形式の 3D
モデルを表示するサンプル */

import javax.media.opengl.*;  // Java の
OpenGL（JOGL)の機能を使うために必要
import processing.opengl.*;   // Processing の
OpenGL の機能を使うために必要
import jp.nyatla.nyar4psg.*;  // NyAR4psg を
使うために必要
```

```
  // ビデオ入力を扱うので必要
import jp.nyatla.kGLModel.*;              //
MQO ローダを使うために必要
import
jp.nyatla.kGLModel.contentprovider.*; //
MQO ローダを使うために必要

Capture cam = null;   // キャプチャ
MultiMarker ar = null; // AR マーカに関する処
理をするオブジェクト
int id1,id2,id3;
int a1,a3,b1,b3,c1,c3;
// マーカに割り当てられる ID 番号
KGLModelData    model_data;       // モデル
データ
ContentProvider  content_provider;  // ファイ
ル読み込み
Picker picker = null;
PPopupMenu menu = null;

void setup() {
  size(640, 480, OPENGL);
  cam = new Capture(this, width,
height,"USB 2.0 Camera-WDM"); // キャプ
チャの準備
  ar = new MultiMarker(this, width, height,
"camera_para.dat",
NyAR4PsgConfig.CONFIG_PSG );
  id1 = ar.addARMarker("patt.hiro", 80); // パ
ターンファイルとマーカサイズの登録
  id2 = ar.addARMarker("patt.kanji", 80);
  id3 = ar.addNyIdMarker(1, 70);
  //cam.start();

// picker = new picker(this);
 menu = new PPopupMenu(this);

 menu.addMenuItem("トイ
レ","changevectorahead");
 menu.addMenuItem("事務
室","changevectorback");
 menu.addMenuItem("研究室","vectorA");
 menu.addMenuItem("演習室 8","vectorB");

  // MQO モデルの準備
  content_provider = new
LocalContentProvider(this,dataPath("vector3
.mqo"));
  PGraphicsOpenGL pgl =
(PGraphicsOpenGL) g;
  GL gl = pgl.beginGL();
  model_data =
KGLModelData.createGLModelPs(
            this, gl, null,
            this.content_provider,0.3,

KGLExtensionCheck.IsExtensionSupported(
gl,

"GL_ARB_vertex_buffer_object"),
            true );
  pgl.endGL();
}

//クリック判定 メニュー表示
void mouseClicked() {
  if(mouseButton==RIGHT){
    menu.show();
    }
}

void draw() {
  if (cam.available()==false) return;
  cam.read();            // カメラ画像の読み込み
  background(0);         // 画面の初期化
  ar.drawBackground(cam); // 背景画像の描画
  ar.detect(cam);        // マーカ認識

  // マーカが検出されたら
  if ( ar.isExistMarker(id1) ) {
    ar.beginTransform(id1);
      PGraphicsOpenGL pgl =
(PGraphicsOpenGL) g;
      GL gl = pgl.beginGL();
       // Processing の行列を OpenGL で使える
ようにする

gl.glMatrixMode( gl.GL_PROJECTION );

gl.glLoadMatrixf( convMatrixPsg2GL(((PGr
aphics3D)g).projection), 0 );

gl.glMatrixMode( gl.GL_MODELVIEW );
        gl.glLoadIdentity();
        gl.glScalef( 1.0, -1.0, 1.0 );

gl.glMultMatrixf( convMatrixPsg2GL(((PGr
aphics3D)g).modelview), 0 );

        // テクスチャとポリゴンの描画に関する設定

gl.glTexEnvf(GL.GL_TEXTURE_ENV,
GL.GL_TEXTURE_ENV_MODE,
```

```
GL.GL_MODULATE);
        gl.glEnable(GL.GL_CULL_FACE);
        gl.glCullFace(GL.GL_FRONT);

        // 光源の設定
        mySetLight(gl, 0, -100, 100);
        gl.glRotatef(a1, 1, 0, 0); // 立たせるため
に回転
        gl.glRotatef(0,0,1,0);   // n が向き　正面
を向かせるために回転
        gl.glRotatef(a3,0,0,1);//ベクトルの角度調
整
        model_data.draw() ;
      pgl.endGL();
    ar.endTransform();
  }

  if ( ar.isExistMarker(id2) ) {
    ar.beginTransform(id2);
      PGraphicsOpenGL pgl =
(PGraphicsOpenGL) g;
      GL gl = pgl.beginGL();
        // Processing の行列を OpenGL で使える
ようにする

gl.glMatrixMode( gl.GL_PROJECTION );

gl.glLoadMatrixf( convMatrixPsg2GL(((PGr
aphics3D)g).projection), 0 );

gl.glMatrixMode( gl.GL_MODELVIEW );
        gl.glLoadIdentity();
        gl.glScalef( 1.0, -1.0, 1.0 );

gl.glMultMatrixf( convMatrixPsg2GL(((PGr
aphics3D)g).modelview), 0 );

        // テクスチャとポリゴンの描画に関する設定

gl.glTexEnvf(GL.GL_TEXTURE_ENV,
GL.GL_TEXTURE_ENV_MODE,
GL.GL_MODULATE);
        gl.glEnable(GL.GL_CULL_FACE);
        gl.glCullFace(GL.GL_FRONT);

        // 光源の設定
        mySetLight(gl, 0, -100, 100);
        gl.glRotatef(b1, 1, 0, 0); // 立たせるため
に回転
        gl.glRotatef(0,0,1,0);   // n が向き　正面
を向かせるために回転
        gl.glRotatef(b3,0,0,1);
        model_data.draw() ;
      pgl.endGL();
    ar.endTransform();
  }
if ( ar.isExistMarker(id3) ) {
    ar.beginTransform(id3);
      PGraphicsOpenGL pgl =
(PGraphicsOpenGL) g;
      GL gl = pgl.beginGL();
        // Processing の行列を OpenGL で使える
ようにする

gl.glMatrixMode( gl.GL_PROJECTION );

gl.glLoadMatrixf( convMatrixPsg2GL(((PGr
aphics3D)g).projection), 0 );

gl.glMatrixMode( gl.GL_MODELVIEW );
        gl.glLoadIdentity();
        gl.glScalef( 1.0, -1.0, 1.0 );

gl.glMultMatrixf( convMatrixPsg2GL(((PGr
aphics3D)g).modelview), 0 );

        // テクスチャとポリゴンの描画に関する設定

gl.glTexEnvf(GL.GL_TEXTURE_ENV,
GL.GL_TEXTURE_ENV_MODE,
GL.GL_MODULATE);
        gl.glEnable(GL.GL_CULL_FACE);
        gl.glCullFace(GL.GL_FRONT);

        // 光源の設定
        mySetLight(gl, 0, -100, 100);
        gl.glRotatef(c1, 1, 0, 0); // 立たせるため
に回転
        gl.glRotatef(0,0,1,0);   // n が向き　正面
を向かせるために回転
        gl.glRotatef(c3,0,0,1);
        model_data.draw() ;
      pgl.endGL();
    ar.endTransform();
  }
}

// 光源の設定を行う関数
void mySetLight(GL gl, float x, float y, float
z) {
  float[] light_diffuse  = { 0.2, 0.2, 0.2, 1.0 };
// 拡散反射光
  float[] light_specular = { 0.3, 0.3, 0.3, 1.0 };
// 鏡面反射光
  float[] light_ambient  = { 0.3, 0.3, 0.3,
```

```
0.1 };  // 環境光
  float[] light_position = { x, y, z,
0.0 };       // 位置と種類

  gl.glLightfv( gl.GL_LIGHT0,
gl.GL_DIFFUSE,  light_diffuse, 0); // 拡散反
射光
  gl.glLightfv( gl.GL_LIGHT0,
gl.GL_SPECULAR, light_specular,0); // 鏡面
反射光
  gl.glLightfv( gl.GL_LIGHT0,
gl.GL_AMBIENT,  light_ambient, 0); // 環境
光
  gl.glLightfv( gl.GL_LIGHT0,
gl.GL_POSITION, light_position,0); // 位置と
種類
  gl.glShadeModel( gl.GL_SMOOTH ); //
シェーディングの種類の設定
  gl.glEnable( gl.GL_LIGHT0 );     // 光源 0
の有効化
  gl.glEnable( gl.GL_LIGHTING );   // ライ
ティングの有効化
}

// Processing の行列を OpenGL の行列に変換
する関数
float[] convMatrixPsg2GL( PMatrix3D
mat_psg ) {
  float[] mat_gl = new float [16];
  PMatrix3D mat = mat_psg.get();
  mat.transpose();
  mat.get(mat_gl);
  return mat_gl;
}

void changevectorahead(){
  a1=90;
  a3=0;
  b1=0;
  b3=90;
  c1=90;
  c3=0;
}
void changevectorback(){
  a1=0;
  a3=90;
  b1=90;
  b3=0;
  c1=90;
  c3=0;
}
void vectorA(){
  a1=90;
  a3=0;
  b1=0;
  b3=270;
  c1=90;
  c3=0;
}
void vectorB(){
  a1=90;
  a3=180;
  b1=90;
  b3=0;
  c1=0;
  c3=90;
}
```