matereal: 小型移動ロボットを用いたアプリケーション制作用ツールキットの開発

加藤　淳[†,††] 坂本大介[†,††] 五十嵐健夫[†,††]

近年，小型の移動ロボットが比較的容易に入手できるようになってきており，パーソナルコンピュータとの組み合わせで，現実世界を巻き込んだ面白いインタラクションを実現するアプリケーションを実行できる環境が整いつつある．しかしながら，ロボットを用いたアプリケーション開発は未だ容易とは言えない．そこで我々は，小型移動ロボットを用いたアプリケーション制作を支援するツールキット matereal を開発し，オープンソースで配布した．matereal には，カメラの撮像からマーカ検出を行ってロボットの置かれた床面上に二次元絶対座標系を張り，ロボットや物体の位置座標を取得する機能がプリセットで用意されている．よって，ロボットの動作を撮像内のスクリーン座標や cm 単位の実世界座標を用いて指示できる．指示には，ユーザの指示した振る舞いと衝突回避などの自律動作を共存させられるよう，独自に拡張を施した状態遷移モデルを用いる．本稿では，ロボット用ツールキットの要件についての考察と，それを踏まえて開発された matereal の設計と機能について述べる．

matereal: Development of a Toolkit for Creating Applications with Small Mobile Robots

Jun Kato,[†,††] Daisuke Sakamoto[†,††] and Takeo Igarashi[†,††]

Various inexpensive mobile robots are becoming commercially available, and it is now possible to run interesting robot applications by combining small robots and standard personal computers. However, it is still difficult for software developers to create robot applications because programming robots usually requires prior knowledge of robotics. We therefore designed, programmed and started to distribute a toolkit to create various robot applications with less difficulty. It uses a ceiling mounted camera as a global sensor that detects robots and objects in the real world. It provides a high-level navigation API to move robots on the basis of the sensing result. The users can design behavior of robots using an extended version of state machine model.

1. はじめに

近年，比較的安価な小型の移動ロボットが一般家庭向けに購入できるようになってきた．例えば，掃除ロボット Roomba，教育と趣味用途のネットタンサーや Mindstorms，踊る音楽プレイヤー Rolly などが挙げられる．小型の移動ロボットは，将来的に，掃除や物運び，服畳みや料理など，家庭での日常的なタスクを支援する使途が期待されている．また，プログラムで実世界のロボットを動かせる体験は刺激的であり，知育玩具にも向いている．しかしながら，エンドユーザとインタラクションするロボットアプリケーションの開発は簡単ではない．環境モデルを設計して，センサ入力を適切に処理してロボットや物体の位置を認識できるようにするだけでも，ロボット工学の知識に裏打ちされた相当量のソフトウェアプログラミングが必要となる．また，そのために必要なハードウェアのセットアップが複雑なことも多い．さらに，ロボットは構成パーツによって物理的な特性や機能が異なるため，多くの場合ロボットごとの開発環境でプログラミングすることになる．したがって，GUI アプリケーションのように手軽に，汎用性のある Human-Robot Interaction(HRI) を開発することは困難である．

我々は，とくにロボットの“移動”という機能に注目し，ロボット工学などの前提知識を持たないソフトウェアプログラマでも，この機能を用いた実世界指向アプリケーションを開発できるようにしたいと考えている．Processing[1] は Java の“描画”に関する機能を簡単に使えるようにして，この機能にアクセスでき

† 東京大学 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†† JST ERATO 五十嵐デザインインターフェースプロジェクト
JST ERATO IGARASHI Design UI Project

るプログラマの層を広げることに成功した．我々は，ロボットについても同様のことができると考えている．そして，開発者の層を Human-Computer Interaction(HCI) の研究者や学生，メディアアーティスト，Physical Computing に興味のあるソフトウェアプログラマまで広げることにより，多様なインタラクションデザインが生まれることに期待している．

そこで本稿では，簡単なハードウェアセットアップのもとで，移動ロボットを用いたアプリケーションを容易に開発できるツールキット matereal を提案する．matereal は，視覚マーカを貼りつけたロボットや物体，それらを見下ろすカメラ，そして計算用のパーソナルコンピュータがある**図 1** のような環境で動作する Java 用のオープンソース☆なツールキットである．matereal には，カメラの撮像からマーカ検出を行ってロボットの置かれた床面上に二次元絶対座標系を張り，ロボットや物体の位置座標を取得する機能がプリセットで用意されている．よって，ロボットの動作を撮像内のスクリーン座標や cm 単位の実世界座標を用いて指示できる．指示には，ユーザの指示した振る舞いと衝突回避などの自律動作を共存させられるよう，独自に拡張を施した状態遷移モデルを用いる．ライブラリとして拡張や改良が容易なように配慮して設計しており，今後はユーザコミュニティの協力を得ながら開発を続けていきたいと考えている．

本稿では，2 章で関連研究を紹介し，3 章でツールキット設計上の要件について論じる．その後，4 章で要件を満たすようデザインされた matereal の設計を説明し，5 章で現状実装されている機能について紹介する．最後に，6 章で結論を述べる．

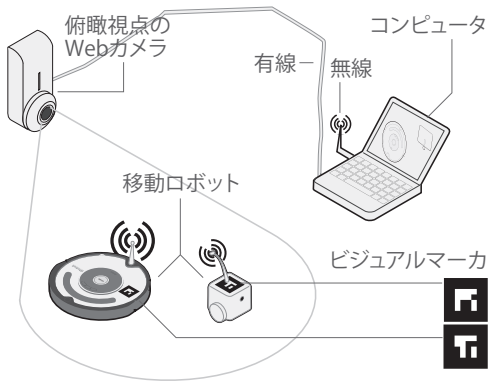

図 1　matereal の動作環境

☆ matereal. http://matereal.sourceforge.jp/

2. 関 連 研 究

2.1 ロボット用のツールキット

ロボットを用いたアプリケーションの開発環境やツールキットは，これまでにも様々なものが開発されてきたが，かつては，ほとんどがロボット工学の研究・教育用途だった．そこで，LEGO Mindstorms[2] や Microsoft Robotics Developer Studio[3] のように，アマチュアのソフトウェアプログラマでもロボットのプログラミングを楽しめる開発環境が登場した．これらの環境では，比較的安価な家庭用ロボットを用いることができ，Visual Programming Language(VPL) によるプログラミングをサポートしている．ただし，あくまでロボットのその場でのローカルな振る舞いを指示する低レベルな機能しか備わっていない．そもそも，ロボットを用いる環境において，グローバルな世界座標系は自明のものではない．ロボット工学では，ロボットの自己位置推定及びロボットが置かれた環境の認識を行う手法の研究が Simultaneous Localization and Mapping(SLAM) と呼ばれる一大研究分野にもなっている．このため，ロボットの位置をグローバルな座標系上で取得し，指定した座標まで移動させられるような高レベルな機能は，センシングから指示出しまで全て自力で実装する必要があった．

2.2 アプリケーションの開発手法

GUI アプリケーションの開発手法は，今やイベント駆動モデルが主流と考えられる．このモデルでは，GUI コンポーネントに対して入力イベントが発生すると，リスナが呼ばれて処理が実行される．一方で，入力によって内部状態が遷移することに注目し，状態を明示的に定義して有限オートマトンを作ることでインタラクティブな GUI を構築する状態遷移モデルを用いる研究[4] もある．状態遷移モデルでは，状態や状態遷移を表すクラスを定義して，メモリ上に有限オートマトンを構築する．状態を定義する手間と引き換えに，プログラムが取りうる状態が限られるため，バグを出しにくいといった利点が得られる．ロボットアプリケーションは，セキュアな動作が求められることもあって，VPL のように状態遷移モデルが基本となる場合がほとんどである．なお，状態遷移モデルの視覚表現である状態遷移図を拡張したステートチャート[5] が提案されている．具体的には，状態を階層的にネストしたり，状態遷移の履歴を記憶しておくといった拡張が提案されている．これらの拡張は，アプリケーションの開発手法に応用できることが示唆されている．

3. matereal の満たすべき要件

我々は，移動ロボットを利用した実世界指向アプリケーションの開発者層を一般のソフトウェアプログラマまで広げたいと考えている．本章では，そのために必要なツールキットの要件について論じる．

3.1 すぐに実機のロボットを動かせる簡便さ

ロボットを用いたアプリケーション開発において，アプリケーションを起動するまでの手間はなるべく少なくあるべきである．ロボットの自律動作に用いるアルゴリズムやシステムを開発する際は，環境をシミュレートできる高性能なシミュレータがあれば事足りるかもしれない．一方で，エンドユーザ，すなわち人間がロボットとインタラクションする方法が問題になるアプリケーション開発では，人間自身をシミュレートできるシミュレータが存在しない限りにおいて，実機を用いる必要がある．したがって，ツールキットは，ハードウェアのセットアップとソフトウェアの API の両面において，ロボットの実機を使うアプリケーションを簡単に起動及び終了できるようにすることが求められる．

3.2 簡単に利用，拡張できる高レベル移動 API

人と協調動作するロボットの行動範囲は人の生活範囲と重なっているため，これらのロボットの活動は，人と同様にほぼ平らな床面上での“移動”が基本となるだろう．しかしながら，既存のロボット用ツールキットでは，ロボットの絶対座標への移動を指示するためにある程度自力で SLAM のアルゴリズムを使ったり実装したりする必要があり，このことがロボットを用いたアプリケーションを開発できる層を狭めることに繋がっていたと考えられる．そこで，より簡単にロボットへ移動を指示できるよう，ロボットや物体の位置を床面の二次元座標系上で取得できたり，ロボットを指定した座標へ移動させられたりする高レベルな API が必要である．ツールキットの主なユーザとして想定しているソフトウェアプログラマにとって馴染み深く直感的な，スクリーン座標系を用いる GUI アプリケーション開発に近い体験を提供できることが望ましいだろう．

一方で，ロボット操作は GUI 操作と異なる特徴も持っている．例えば，ロボットに移動を指示すると，実際に移動が完了するまでに時間がかかるうえ，厳密に指定したパス上で動かすことは難しい．例えば，床の凸凹や差動車輪の左右モーターの出力の違いなど，さまざまな原因によってノイズが混じることになる．つまり，ツールキットとしては，ロボットをある位置へ移動させるメソッドを用意するだけでは不十分で，コースから外れた際のハンドリングなど，ユーザが自分でロボットの移動戦略を簡単に設計できる拡張性も必要である．この拡張性によって，目的地へただ真っすぐ進むのではなくわざと蛇行させるなど，移動という一見単純な振る舞いに多様性を持たせることも可能になるだろう．

3.3 エンドユーザの指示と自律動作を共存させられるアーキテクチャ

ロボットを用いたアプリケーション開発においては，しばしば，ロボットの振る舞いを複数共存させる必要が生じる．例えば，前節とも関係するが，エンドユーザがロボットをある位置へ移動させたいと考えたとき，途中に障害物があったり，他のロボットと衝突しそうになることがあるかもしれない．それでも目的を達するためには，物体との衝突可能性を判定しつつ，衝突しそうなら回避行動を取り，大丈夫なら目的地を目指して進むといった振る舞いが必要になる．ここでは，回避行動を取りながら目的地へ移動するという，二つの振る舞いが共存している．ロボット工学の古典的な考え方である包摂アーキテクチャ[6] では，目的地への移動という高レベルな振る舞いが回避行動という低レベルな振る舞いを包摂していると捉える．そして，低レベルなロジックが動いているときは高レベルなロジックを抑制して，ロボットの全体としての振る舞いに一貫性を持たせることができるとしている．ツールキットは，このように，エンドユーザの指示とロボットの自律動作をうまく調停させられる仕組みをプログラマ向けに用意する必要がある．

3.4 高レベル API の内部処理を機能単位でまとめた低レベルの API

高レベルな移動 API を実装するにあたり，ツールキットは様々な機能を内包することになる．高レベルな API を用意してユーザの手間を省くことは大切だが，一方で，機能単位ごとにコードを分けてモジュラリティを確保する必要もあると考えられる．機能のモジュール化による恩恵は，コードの保守性のみに留まらない．例えば，ロボットに視覚マーカを貼り，カメラでマーカ検出して位置を認識する場合，カメラの撮像を得る機能，撮像からマーカ検出する機能，スクリーン座標と世界座標の変換機能が処理フローに含まれる．別のカメラの映像をユーザインタフェースに表示したい場合，また，マーカの動きからジェスチャ認識させたい場合など，低レベルの機能が API として公開されていると便利なケースは非常に多い．

ツールキットのユーザは，ツールキットの内部実装

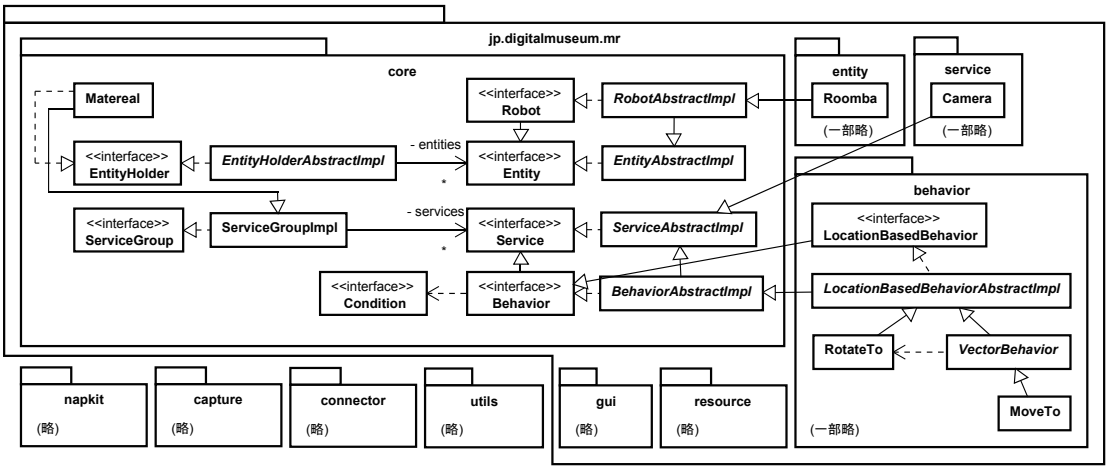

図 2 matereal のクラス図の概要

は知らなくてよいが，何をしているかは理解している．ユーザがそのプロセスの一部を別の目的で使いたいと考えたとき，その機能だけをツールキットから呼び出して使えるよう，階層的な粒度を持った API を整備すべきである．

3.5 GUI API によるエンドユーザ向けサポート

既存のロボットの開発環境では，独自書式で難解な内容の設定ファイルを編集しないとロボットが起動しないことも多かった．しかし，アプリケーションという観点から見ると，何ら事前の設定を必要とせずプログラムが立ち上がり，GUI でロボットの MAC アドレスなど実用に必要な情報をインタラクティブにいくつか自由入力あるいは選択させる，といったユースケースが自然だろう．そこで，アプリケーションが使用する機能ごとの設定 GUI を呼び出せる API がプリセットで用意されているとよい．また，設定の編集だけでなく，カメラの撮像に情報を重畳表示する描画 API など，エンドユーザが使うインタフェースの設計を支援する機能があると便利だろう．

4. matereal の設計

本章では，前章の議論を踏まえて matereal の設計について説明する．まず 4.1 節でハードウェアのセットアップや matereal が動作する環境について述べる．そして，4.2 節以降，matereal の主たる構成要素を紹介しながら，ツールキットのアーキテクチャとその動作を解説する．必要に応じて**図 2** に示したクラス図を参照されたい．本章で紹介する内容は，記載がないものは全て core パッケージに含まれている．

4.1 動作環境

すぐに実機のロボットを動かせるよう，そして，ロボットや物体の位置を検出できるように，matereal は 図 1 のような簡単なセットアップで動作する．また，Bluetooth/WiFi/USB/COM ポート経由でコントロールできるロボットに接続できる．プリセットですぐ使えるロボットについては 5.1 節で紹介する．視覚マーカには ARToolKit[7] マーカを使用している．matereal 自体は，Java 向けのライブラリである．クロスプラットフォームで，外部接続用のライブラリが充実しており，一般に普及している環境として，Java を選んだ．外部接続用のライブラリとの相性もあるが，基本的には Java VM がサポートする Windows/-Mac OSX/Linux の各ディストリビューションで動作する．我々は主に Windows Vista/7 及び Mac OSX 10.5/10.6 で動作確認を行っている．

4.2 API の起点となる Singleton（Matereal）

matereal は，Singleton として実装されている Matereal クラスがインスタンス化されるタイミングで初期化処理が走る．以降，ライブラリはプログラムのメインスレッドとは別のバックグラウンドスレッド上で動作する．ユーザは，Matereal の Singleton インスタンスを起点にしてライブラリの様々な機能にアクセスすることになる．

4.3 ロボット（Robot）と物体（Entity）

ロボットと物体は，ライブラリ上ではそれらを表すクラス（Robot, Entity インタフェースの実装クラス）のインスタンスとして表現される．ユーザは，接続に用いるアドレスを引数にしてロボットのクラスを

インスタンス化したり，Matereal クラスの *lookForRobot(String name)* などのメソッドでインスタンスを取得できる．そして，これらのインスタンスを用いてロボットに指示を出したり物体の位置を取得できる．ライブラリに新しい種類のロボットへのサポートを追加したい場合は，entity パッケージ内に新しい Robot の実装クラスを定義すればよい．

4.4 機能単位（Service）

matereal が管理する機能は，基本的にサービスと呼ばれるクラス（Service インタフェースを実装する ServiceAbstractImpl の子クラス，service パッケージ）として実装されており，ユーザがインスタンス化すると使えるようになる．例えば，*Service camera = new Camera(); camera.start()* と書けばカメラの撮像を取得するサービスが起動する．以降，*camera.getImage()* とすれば撮像を画像オブジェクトとして取得できる．既に起動しているサービスから望みのクラスやインタフェースを実装するものを探すために，Matereal クラスには *lookForServices(Class c)* といったメソッドが用意されている．

サービス間の連携

カメラの撮像を取得するサービスと画像からマーカ検出するサービスなど，サービス同士が連携するケースは非常に多い．そこで，より多くのサービス同士が連携できるよう，一般的な機能に関しては積極的に service パッケージでインタフェースを定義して用いることにしている．例えば，マーカ検出のサービスは画像を配信するサービスのインタフェース ImageProvider から画像を受け取るよう実装されている．したがって，カメラの代わりにネットワーク経由で配信される映像を受信する ImageProvider を新しく実装すれば即座にマーカ検出のソースとして使うことができる．

マルチスレッド対応

サービスはデフォルトでは matereal のスレッド上で定期的にロジック（*run()* メソッド）を呼ばれるが，必要に応じて新しいスレッドを作り，その上で異なる頻度で動作させられる． matereal ではスレッドを ServiceGroup というインタフェースでラップしており，*ServiceGroup group = new ServiceGroupImpl(); camera.start(group)* とすれば，カメラは group が表すスレッド上で動作する，ServiceGroup により，マルチコアの計算資源を有効利用できるだけでなく，*group.setInterval(long ms)* のようにしてロジックの実行頻度を柔軟に変えることができる．例えば，時間のかかる重い処理をこなすサービスだけ，別グループ上で低頻度で実行させられる．

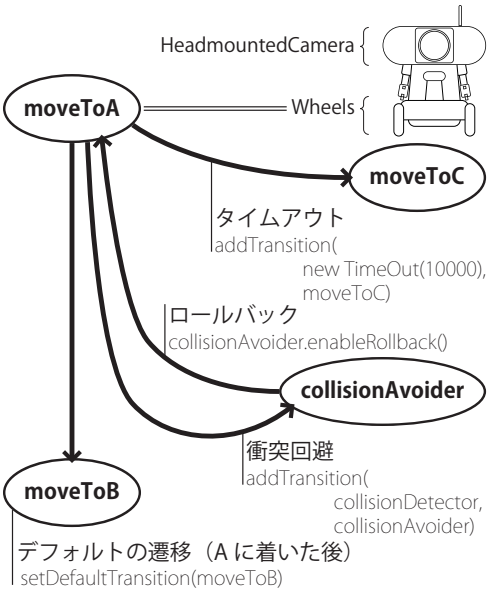

図 3 ロボットに振る舞いを割り当てて状態遷移図を作る様子

4.5 ロボットの振る舞い（Behavior）

ロボットの振る舞いはサービスの一種（Behavior インタフェースを実装する BehaviorAbstractImpl の子クラス，behavior パッケージ）として実装されており，ユーザは *robot.assignBehavior(new MoveTo(x, y))* のようにしてロボットに振る舞いを割り当てることができる．ロボットに割り当てられていない振る舞いのインスタンスは何もしない．**図 3** に，本節で紹介する例の模式図を示す．

状態遷移モデル

振る舞いは，状態遷移モデルにおける状態の役割を果たす．ユーザは，振る舞いから振る舞いへの状態遷移を設定することでロボットの継続的な動作を設計できる．状態遷移は，条件節（Condition）と遷移先の状態からなるペアである．例えば図 3 に示すように，A 地点へ着いたら続けて B 地点へ向かわせたり 10 秒後に A 地点へ着いていなければ行き先を C 地点へ変更したりできる．

リ ソ ー ス

ロボットは，車輪や頭部カメラなどの機能単位ごとに振る舞いを割り当てられるようになっている．matereal ではハードウェアが持つこれらの機能単位を総称してリソースと呼んでおり，インタフェースとして Wheels や HeadmountedCamera などの名前で resource パッケージ内に定義することにしている．インタフェースには，前進後退，左右回転，撮像の取得といったプリ

ミティブなメソッドが用意されており，同じリソースを持っていればロボットの種類によらず同じ振る舞いを割り当てられる．リソースごとに状態を管理することで，移動と同時に頭部カメラの撮像を得たり，腕を上げたり下ろしたりするなど，ロボットが持つ複数の機能を同時に活用できる．

ロールバック

matereal は状態遷移の履歴をスタックに積んで記憶しており，いったん遷移した状態を元に戻せる．matereal ではこれをロールバックと呼んでおり，自律アルゴリズムとユーザの指示を共存させる目的で用いる．例えば，目的地へ移動させる振る舞いに，物体などと衝突しそうになったら衝突回避の行動を取る状態遷移を指定する．衝突回避の振る舞いでロールバックを有効にすれば，衝突回避後に自動的に元の振る舞いを再開させられる．すなわち，必要に応じて自律移動のアルゴリズムを働かせながら，ロボットをエンドユーザの指示に従って動かせる．

サブルーチン

振る舞いを実装する際には，内部で別の振る舞いをサブルーチンとして呼び出すことができる．具体的には，BehaviorAbstractImpl の実装クラス内で *delegate(Behavior behavior)* メソッドを呼ぶことにより，内部的に behavior へ状態遷移を起こせる．このように状態遷移に階層を持たせることで，クラスの再利用性を高められ，同じ内容の振る舞いを何度もコーディングする手間を省けるだろう．

5. matereal のプリセット機能

本章では，前章で説明した基本設計の上に実装されている matereal のプリセット機能，すなわち図 2 における core パッケージ以外の部分を説明する．

5.1 市販ロボットのサポート

matereal は，**図 4** に示す 3 種類の市販ロボットをサポートしている．それぞれを表すクラスが entity パッケージに定義されている．

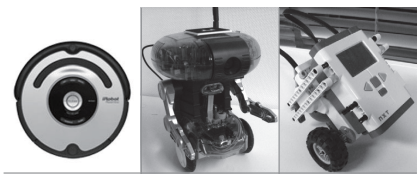

図 4 サポートしている市販ロボット 3 種

5.2 床面の二次元座標系

matereal には，ロボットや物体に視覚マーカを貼り，カメラの撮像からマーカ検出を行うことによって床面上での二次元座標系を扱えるようにする機能が備わっている．本節では，座標系を扱う機能の紹介に続いて，移動を指示する方法について説明する．

service パッケージでは，サービスのインタフェースとして画像を配信する ImageProvider と物体の座標情報を提供する LocationProvider などが定義されている．それらを実装するサービスとしては，カメラの撮像を取得するサービス（Camera）と撮像からマーカ検出するサービス（MarkerDetector）が用意されている．**図 5** は次のコードを実行したときの画面である．

```
1 Camera camera = new Camera();
2 camera.start();
3 MarkerDetector detector = new
      MarkerDetector(camera);
4 detector.start();
5 new DisposeOnCloseFrame(new
      CoordProviderPanel(camera)).setVisible(
      true);
6 Robot robot = new MyRobot("Noopy␣1号", "
      btspp://0000CAFEBABE");
7 detector.put(new NapMarker("marker.patt"),
      robot);
```

1,2 行目でカメラのサービス，続いて 3,4 行目で撮像からマーカ検出するサービスを起動している．そして，5 行目で画面に座標系の設定用 GUI を表示している．さらに，6 行目でロボットとの接続を確立し，7 行目でロボットとマーカを紐づけしている．以降，ロボットの位置を *detector.getLocation(robot)* により取得できるようになる．

behavior パッケージには，絶対座標を利用した振る舞いを表す LocationBasedBehavior インタフェース

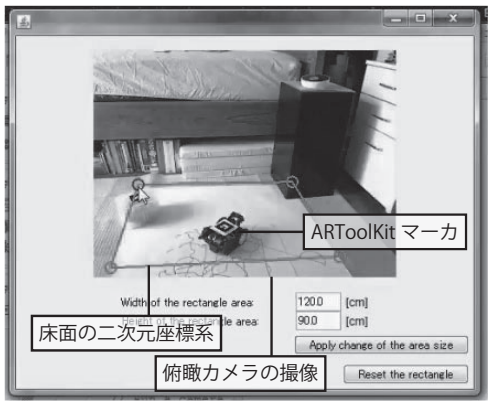

図 5 俯瞰視点のカメラと視覚マーカで二次元座標を取得する様子

と，実際に絶対座標を取得する機能を実装した抽象クラスが用意されている．これを継承する振る舞いクラスでは，抽象クラスで定義された *getLocation()* などのメソッドを呼び出すだけで，適切な LocationProvider が提供する座標の情報を得ることができる．プリセットで用意されている，ロボットに指定した方向を向かせる RotateTo クラスは，この抽象クラスを継承して書かれている．

ベクトル場に沿った移動（VectorBehavior）

matereal では，**図 6** のような二次元のベクトル場を用いてロボットの移動を指示できる．ロボット工学の分野では，障害物を回避しながら目的地へ移動するパスプランニングの研究で，ポテンシャル場を用いる手法が提案されている[8]．この手法では，フィールドに仮想的な高低差を定義し，ロボットの移動する向きをその場の勾配によって決める．すなわち，ロボットの位置に関してスカラー場を微分して二次元ベクトルを計算している．各位置ごとに二次元ベクトルを持つベクトル場でロボットの移動方向を直接指示できれば，より一般性があり，ユーザが直感的に使えるだろう．そこで，VectorBehavior という LocationBasedBehavior の抽象実装クラスが用意されている．VectorBehavior を継承するクラスでは，位置座標を引数に取り，二次元ベクトルを返すメソッドのみ実装すれば，ロボットの移動を指示する振る舞いとして使える．動作としては，ロボットの現在の向きとベクトルを比べ，差が大きい場合には RotateTo を呼び出して方向を合わせ，それ以外の場合には前進する．ただし，ベクトルのノルムが指定した閾値より小さい場合は止まる．次項で具体例を示す．

絶対座標への移動（MoveTo）

behavior パッケージには特定の絶対座標へロボットを移動させる振る舞いを表す MoveTo クラスが用意されており，*robot.assignBehavior(new MoveTo(x, y));* と書けばロボットが指定した座標へ移動を開始する．このクラスは VectorBehavior の実装クラスであり，ロジックの部分は，現在の位置ベクトルから目的地の位置ベクトルへの相対ベクトルを返している次の 3 行（実質 1 行）である．

```
1 public Vector2D getVector(Position position
      ) {
2         return new Vector2D(
3                         destination.getX()-
                              position.getX(),
4                         destination.getY()-
                              position.getY());
5 }
```

なお，behavior パッケージには，ロボットを含む他の物体の位置と方向を取得して，その後ろへ移動する Follow という振る舞いなども含まれている．

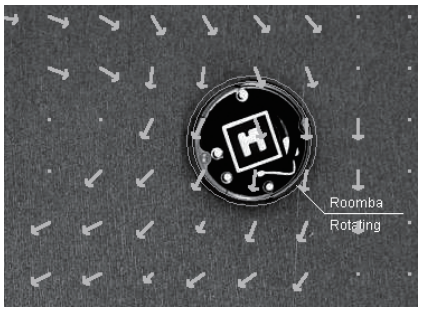

図 6　ベクトル場を可視化した例

5.3 衝突回避

matereal では，衝突の可能性を検知する CollisionDetector と衝突を回避する振る舞いを表す CollisionAvoider の二つのモジュールで衝突回避の仕組みが実装されている．図 3 にあるように，条件節に衝突可能性を判定する CollisionDetector を，遷移先の状態に CollisionAvoider を与えれば，衝突を回避させられる．ロールバックを有効にすれば，衝突を回避したあと遷移元の振る舞いを再開させられる．

5.4 代理ロボット（ProxyRobot）

matereal では，ロボットとその構成部品が基本的には全てインタフェースとして定義されている．したがって，必ずしも物理的に存在する一台のロボットでなくとも実装クラスを作って利用できる．例えば，複数のロボットを代理する仮想的なロボットを表す ProxyRobot インタフェース及びその抽象実装クラスを core パッケージで定義している．これを用いて，一台の代理ロボットへの指示を複数台へ複製する CloneWheelsRobot クラスを entity パッケージで定義している．この代理ロボットを動かすと関連付けられたすべてのロボットが同じ動きをする．なお，これらは実験的なインタフェース及びクラスなので図 2 には載せていない．

5.5 設定用 GUI とカタログ

本ツールキットを用いる上で必要になる環境設定は，できるだけ GUI で済ませられるようになっている．まず，gui パッケージではプリセットのサービスごとにオプションを指定するための GUI パーツが定義されている．パーツは，図 5 で示したように Java の標準 GUI ライブラリ Swing 向けのコンテナとして実装されている．サービスを引数にしてこれらのコンテナをインスタンス化し，ウィンドウ（JFrame）な

どに貼り付けると，インタラクティブにサービスを設定できる．また，サービス以外にも，ツールキットが管理しているロボットの接続用アドレスやロボットに貼りつけたマーカとロボットの対応付けなどの情報は環境依存である．これらを GUI で編集でき，結果をカタログと呼ばれる XML ファイルとして保持する支援クラス Catalog が用意されている．カタログを作ると，Matereal クラスの Singleton インスタンスで *loadCatalog(Catalog catalog)* メソッドを呼ぶことでロボットを使うための準備が全て完了するようになる．

5.6 外部パッケージ

本節では，matereal に含まれているが，matereal 外でも利用できるサブプロジェクトを紹介する．基本的に，他の著者によるライブラリをまとめたラッパーになっている．

connector パッケージ

connector パッケージは，ロボットとの接続を実現するためのパッケージである．接続手法を抽象化したインタフェース Connector と実装クラス群が定義されている．また，接続先アドレスのプレフィックス（btspp://，http://など）をもとに適切な Connector のインスタンスを取得するファクトリークラス ConnectorFactory がある．現状，Bluetooth，TCP/IP，COM ポート向けの実装クラスが存在する．Bluetooth 接続には JSR-82 で定められた仕様の一部を実装する BlueCove☆ライブラリを，TCP/IP 接続には JRE の標準ライブラリを，COM ポート接続には RXTX☆☆ライブラリを使用する．

capture パッケージ

capture パッケージは，Web カメラによるキャプチャを実現するためのパッケージであり，基本的な撮像機能を表すインタフェース VideoCapture と実装クラス群が定義されている．カメラとの接続は OS ごとに利用するライブラリが異なり，Windows では DirectShow，Mac OSX では QuickTime，Linux では Java Media Framework 経由で Video4Linux を用いる．前項の connector と同様に，ファクトリークラス VideoCaptureFactory が用意されている．

6. おわりに

本稿では，移動ロボットを用いたアプリケーション開発に必要なツールキットの要件について考察し，実際に作成したツールキット matereal を紹介した．matereal は，カメラの撮像から視覚マーカを検出して，ロボットや物体の位置を床面の二次元座標系上で取得できる．二次元座標系は，ベクトル場などを用いたロボットへの高レベルな移動指示にも利用できる．また，独自に拡張を施した状態遷移モデルにより，ユーザによる振る舞いの指示と衝突回避などの自律アルゴリズムの共存を可能にした．ツールキットは現在もオープンソースで開発を続けており，今後はユーザによるフィードバック，拡張や改良を受け容れていく．現在の予定では，Processing 版の開発や Augmented Reality 関連の演出を支援する機能などにより，ツールキットをさらに実用的で扱いやすいものにしていきたいと考えている．

謝辞 matereal は，2008 年 12 月から 2009 年 8 月まで，2008 年度下期未踏 IT 人材発掘・育成事業の支援を受けて開発を進めた．ご指導を頂いたプロジェクトマネージャの石川裕先生とプロジェクト管理組織のメルコホールディングスの人々に感謝したい．

☆ BlueCove. `http://bluecove.org/`

☆☆ RXTX. `http://users.frii.com/jarvi/rxtx/`

参 考 文 献

1) Processing. `http://processing.org/`.
2) LEGO Mindstorms NXT Software. `http://mindstorms.lego.com/`.
3) Microsoft Robotics Developer Studio. `http://www.microsoft.com/Robotics/`.
4) Appert, C. and Beaudouin-Lafon, M.: SwingStates: adding state machines to the swing toolkit, *Proceedings of the 19th annual ACM symposium on User interface software and technology*, ACM, p.322 (2006).
5) Harel, D.: Statecharts: A visual formalism for complex systems, *Science of computer programming*, Vol.8, No.3, pp.231–274 (1987).
6) Brooks, R.: A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation*, Vol.2, No.1, pp.14–23 (1986).
7) Kato, H. and Billinghurst, M.: HMD calibration for a video-based augmented reality conferencing system, *Proc. IWAR'99*, pp. 85–94 (1999).
8) Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots, *The International Journal of Robotics Research*, Vol. 5, No.1, p.90 (1986).