平成 26 年度　特別研究報告書

Kinect による移動ロボットの経路決定法の検討

龍谷大学　理工学部　情報メディア学科

T110497　山田　嵐士

指導教員　三好　力　教授

内容梗概

近年少子高齢化という社会構造的な変化を受け, 介護サービス, 家事代行サービスを利用する高齢者が年々増加傾向である. そのサービスをより効率化するため生活や介護や医療といった幅広い分野でサービスロボットの開発・研究が行われている. その家事代行を行うサービスロボットとしては, 洗濯や掃除や料理といったロボットが存在するが, 家庭用で衣類整理を行うサービスロボットは現在存在していない.

そこで本研究では, 衣類整理中のロボットの移動に着目し, Kinect とラジコンカーを用いて, Kinect のカメラ画像と距離センサにより物体を検出し衣類を収納するスペースまでのラジコンカーの移動経路探索し自立制御を行うシステムの作成を試みた.

目次

第1章　はじめに

1.1　研究背景

近年，少子高齢化という社会構造的な変化を受け，介護サービス，家事代行サービスを利用する高齢者などが年々増加傾向である．そのサービスをより効率化するため生活や介護や医療といった幅広い分野でサービスロボットの開発・研究が行われ世界的に注目を集めている．

サービスロボットは，[1]産業用ロボット（製造業向け），サービスロボット（非製造業・家庭向け），特殊環境下用ロボット（宇宙，原子力，深海，災害現場等）の 3 つに大別されており サービス業など業務用とは別にルンバなどの家庭用ロボットが存在する．家庭用ロボットは，① 家電操作，② コミュニケーション，③ 家事代行の 3 つに大別される．本研究では家事代行ロボットに着目し検証を行う．

家事代行ロボットとは，家庭での生活を助ける目的であり，なんらかの家事を助けるためのロボットである．人が家庭で生活を行う上で必要となる家事とは，料理，買い物，掃除，洗濯，衣類整理などであり，介護サービスや家事代行サービスではそれらの家事のサービスを利用することが可能である．さらに，サービスに代わる家事代行ロボットも普及されている．しかし，衣類整理を補助する家事代行サービスや商品は存在するが，家庭で衣類整理を行うロボットは普及していない(図 1 参照)．

掃除	料理	買い物	洗濯	衣類整理
				

図 1：存在する家庭用ロボット（自動代行サービスを含む）

衣類整理を行うロボットは，衣類整理を行うために①何の衣類かを識別すること，②衣類を掴む，③ 収納スペースまで移動し収納・整理することの①②③を繰り返さなければいけない．そのため，移動中に家具や衣類が障害物となり移動ができない可能性がある．

本研究では，Kinect のカメラ画像と深度画像を用いて物体検出を行い障害物を避け目的地（衣類を収納するスペース）までの移動経路作成を行い，作成された経路をラジコンカーで移動できるかを検証する．

第 2 章　開発環境

2.1　概要

Kinect で RGB 画像と深度画像を取得し, 物体検出を行う. また, 動作ラジコンカーにビュートローバー ARM を使用する. 本章ではそれぞれの機器についての説明を以下に示す.

2.2　Kinect とは

Kinect は Microsoft 社から発売されている家庭用据え置き型ゲーム機 Xbox360 向けのゲームデバイスである.2010 年 11 月 4 日に米国で発売され, 同年 11 月 20 日に日本でも発売された. 特徴として Kinect が内蔵するセンサによって音声指示を認識すること, プレイヤーの動きを捉えることによりコントローラを使用せずにゲームの操作を可能とすることが挙げられる. このような音声やジェスチャーなど「人の自然な動作」で入力を行うユーザインタフェースのことを Natural User Interface(NUI)という. Kinect には, 3 つのセンサおよび専用ソフトウェアを動作させるプロセッサが内蔵されており, 人間を検出しプレイヤーの姿勢や動きを認識することができる. これにより従来のようなボタン操作ではなく, プレイヤー自身の体・声を使って直観的にゲームを操作することが可能となった. 内蔵されているセンサは, 写真撮影や映像録画に用いられる RGB カラー映像認識用カメラセンサ, Kinect から対象物までの距離を測定するための近赤外線距離画像センサ, 音声を認識するための 4 つのマイクセンサの 3 種類である. さらにユーザの動きに対応するため, 上下の角度を自動調整する電動チルト機構も備えている.

Kinect は人物の動きをデジタル的に記録するモーションキャプチャ技術を使用しているが, 上記のセンサを用いることで, キャプチャ時に着る特殊なスーツおよび検出時に使用するトラッカーが必要なくなっている. これによりカメラに被写体を映す事でプレイヤーから Kinect 本体までの距離を測定し, ユーザの骨格のさまざまな動きを検出, ゲーム内のキャラクターの動きにリアルタイムに反映させることが可能となっている.

図 2:Kinect 外観

対象物との距離を計測する近赤外線距離画像センサは，赤外線を照射する近赤外レーザとレーザ照射パターンを捉えるための近赤外カメラで構成されている．また，計測方法としては三角計量を利用している．具体的にまず，Kinect に搭載した近赤外レーザから対象物にスポット光を複数，ランダムに配列したようなパターンを投射し，対象物に映ったパターンをカメラで捉える．そして，捉えたパターンのカメラからの見込み角を求める．これに，レーザから対象物までの距離を算出する．パターン内のスポット光を区別出来れば，1 回の投影によってカメラが捉えた画面内の物体の複数点の距離計測が可能になる．

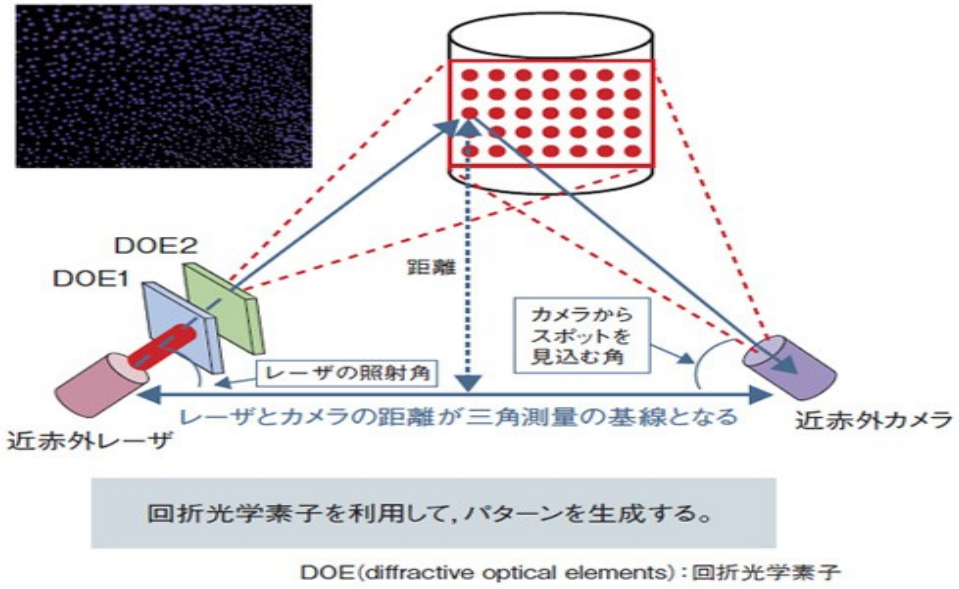

図 3: 三角測量を用いた距離測定

また，Kinect はコンピュータに接続することも可能である．開発元の Microsoft 社は非商用の場合に限り「内部アルゴリズムにアクセスしないこと」と「Xbox 向けのゲームの不正利用に使用しないこと」を条件として OpenNI や NITE 等のオープンソースライブラリを公開しシステム開発を許可している．

表 1: Kinect 開発環境

OS	Windows7
開発ツール	Microsoft Visual Studio 2010
開発言語	C++
ライブラリ	OpenNI 1.5.2

OpenNI とは Kinect のセンサ部を開発した PrimeSence 社が中心となって開発したオープンソースのライブラリである．以下に OpenNI の仕様を示す．

表 2:OpenNI の仕様

対応 OS	WindowsXP, Vista, 7 Ubuntu Mac OS X
言語開発	C, C++, C#
RGB カメラ解像度	640×480
距離カメラ解像度	640×480
認識距離範囲	500-10000mm

以上が Kinect と Kinect の開発環境の説明とする.

Kinect は介護や医療など様々な分野で応用・研究されている. 以下では Kinect を用いたロボット制御を行った既存技術について以下に示す.

2.2.1 Kinect を用いた自立移動ロボットの制御[4]

移動ロボットに搭載した Kinect のスケルトントラッキング機能を用い, 人間追尾とジェスチャ検出による移動ロボットの制御を行う. ジェスチャで胴体と右手の x 座標および y 座標が重なることで追尾開始または追尾停止の指示を移動ロボットに送り, 追尾対象者の胴体の座標をリアルタイムで取得し, 移動ロボットを常に対象者の真正面に移動するよう「角度」と「距離」データを送信することにより人間追尾を行うことに成功している.

2.2.2 3 台の Kinect を搭載した移動ロボットによる特定物体のハンドリング[5]

ロボット技術を介護や家事などの生活分野に適用した生活支援ロボットの実用化, 少子高齢化社会への大きな支援として「指定された特定の物体を取ってくる機能」を搭載する人間が自然な動作で音声と指差しの指示を出すと, 移動ロボットが指定された特定の物体を取ってくるシステムの構築が成功している. RGB カメラ, 深度センサ, マルチアレイ・マイクロフォンを内蔵する Kinect センサを 3 台移動ロボットに搭載し, それぞれのセンサが音声認識, 指差し認識, 特定物体認識を分担処理し, 認識結果を統合処理することにより実現したものである.

2.2.3 人物の挙動認識に基づく自律移動型ロボット制御の研究[6]

ロボットによる日常生活の支援において自律移動ロボットの人物の挙動認識は重要であり, この人物追跡は特に重要である. この研究では, Kinect による距離計測と人物認識を利用して挙動を認識する. 追跡開始時に対象の衣服の情報を学習し, テンプレートマッチングを用いて追跡対象人物の特定を行う. しかし, カメラ画像全体にマッチングを行うと処理時間が膨大になるので, 距離情報を用いたマッチング範囲の限定を行う. これらを組み合わせることにより, ロバストな人物追跡を実現する手法を

提案する. また, ジェスチャー認識を加え, 人間とロボットがコミュニケーションを行えるようにした. 屋内の廊下での実験により, 提案手法の有効性を確認したものである.

2.3 ビュートローバー ARM とは

ビュートローバー ARM はヴイストン株式会社から発売される組み込みマイコンを搭載したプログラミング学習用教材ロボットである. 特徴としてビュートローバー ARM には赤外線センサ, ブザー, モータの 3 種類が基本機能として搭載され, 拡張部品を利用することにより PC との無線通信を行うことができる. また, NXP 社が公開している LPCXpresso をダウンロードすることによりロボット制御を C 言語プログラミングで容易に動作させることが可能である.

必要とするラジコンカーの条件として, プログラミングを行う際左右のモータの回転方向と回転時間を容易に設定することができるという条件と, 小型ロボットを複数機使用する条件からビュートローバー ARM の使用を決定した.

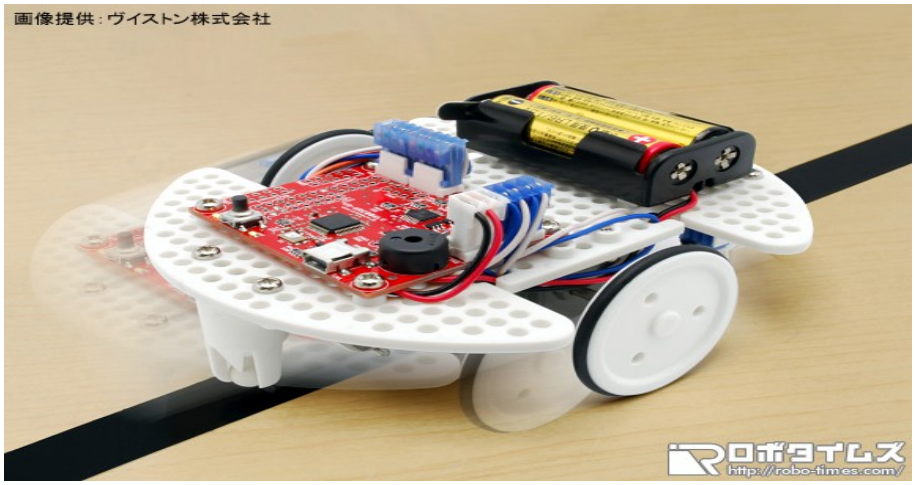

図 5:ビュートローバー ARM 外観

ビュートローバー ARM は Mrt_Run()関数, wait()関数で制御する. 関数についての説明は以下で行う.

モーターの回転方向制御には, Mtr_Run()関数を使用する.

・Mtr_Run_lv(short m1, short m2, short m3, short m4, short m5, short m6)

Mtr_Run_lv()はモーター回転を設定する関数で, 引数 m1～m6 に short 型のデータを与えることで, モーターの回転数, 回転方向を個別に設定できる. 今回は m1 が右モーター, m2 が左モーターの制御を行う.

引数に 0 もしくは-32768 を与えることで停止, 整数を与えると正転, 負数を与えると逆転する. 以下に引数を与えた状態を示す(表 3).

表 3:Mtr_Run_lv の引数

	m1	m2
前進	整数	負数
右タイヤ移動	整数 or 負数	0 or -32768
左タイヤ移動	0 or -32768	整数 or 負数
停止	0 or -32768	0 or -32768

モーターの動作時間制御には, wait()関数を使用する.

・wait(unsigned int msec)

wait()関数は, 引数[msec]で与えた時間だけ待つ関数である. 時間の単位は msec(ミリ秒)で,「wait(1000)」と記述すると, 1 秒間 (=1000 ミリ秒) 待つ処理になる.

第 3 章　洗濯物片付けシステム

3.1 概要

少子高齢化が進むことにより介護サービス, 家事代行サービスを利用する高齢者が増加しているため, 提供者の人数が不足していくため, サービスを効率化することが望ましい. 衣類整理に重点を置きサービスを効率化するため, Kinect を装備した一体型クローゼットに小型ロボットが配備された「洗濯物片付けシステム」(図 6)の検討を行ってきた.

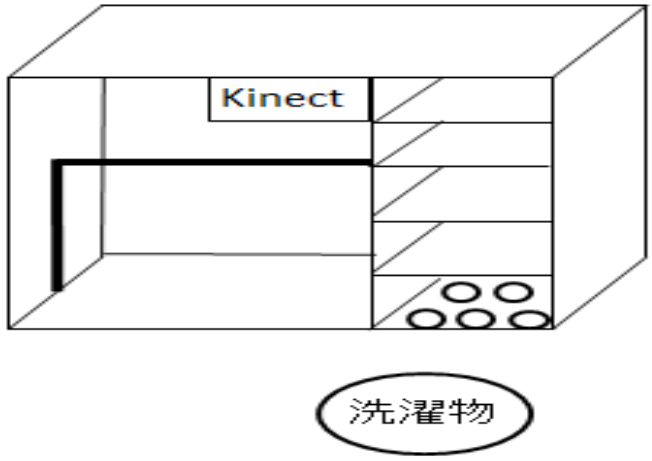

図 6:洗濯物片付けシステム

3.2 衣類整理処理の細分化

衣類整理を人間が行う場合, 衣類を立体的に認識し「衣類の種類が何か」,「どの場所を掴めば畳むことができるか」を瞬時に判断することが可能である. だがロボットが処理を行う場合, 処理が困難である点が 2 点ある. 1 つ目が対象物が衣類であるため判断するための衣類の厚さ, 衣類の大きさが似通ったものが多いため判別が困難である. 2 つ目はロボットで畳む処理を行う際, 定型的な操作の繰り返しでは種類によっては掴む箇所が正しくない可能性があるため困難である. この 2 点よりロボットは洗濯物を畳まずに靴下や下着などの小物は収納スペースに整理, シャツやズボンなど大きなものはクローゼットにかけて収納することによりロボットの収納動作の簡単化を目的とする.

3.3 洗濯物片付け定義

衣類整理をロボットが行動できるように簡単化をするため衣類整理の処理を 4 段階に分割する.分割した 4 段階を表 4 と図 7 として示す.

表 4:洗濯物片付け定義

第一段階	全種類の洗濯物が 1 つの山になっている状態
第二段階	洗濯物山から 1 つ洗濯物を取り出した状態
第三段階	靴下がペアリングさえて並んでいる状態
第四段階	すべての洗濯物の整理が完了している状態

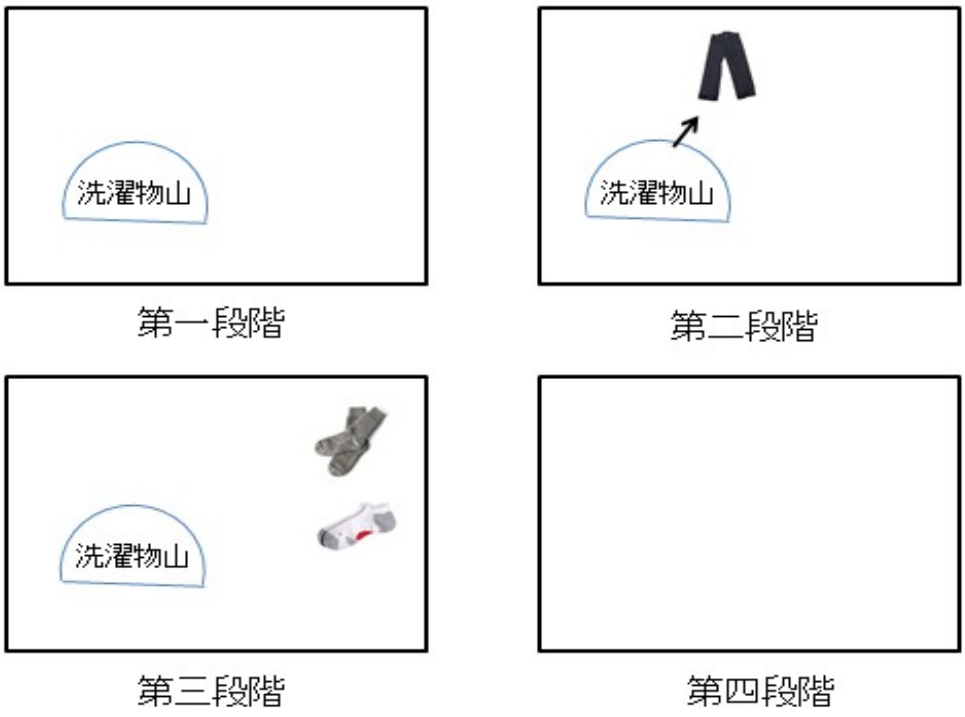

図 7:洗濯物片付け定義

3.4 ロボット動作

洗濯物片付けシステムに求められる機能は「洗濯物の状態の認識」,「洗濯物の判別」,「靴下のペアリング」,「整頓・収納」の 4 つに分類する. 4 つのなかで「整頓・収納」に重点を置き, ロボットが実際に動作できなければシステムが成り立たないため, 収納を行う小型ロボットの移動を研究対象とした. 洗濯物片付け定義の 4 段階をロボットが動作できる命令モジュールの構築を行う.

ロボット動作を 2 段階に定義したものを表 5 に示す.

表 5:ロボット動作定義

第一段階	Kinect で画像情報を取得し, ロボットが移動する予測線を算出する
第二段階	第一段階で算出された予測線をロボットが移動できるようモータ制御モジュールを算出する

このシステムは移動ロボットを用いて洗濯物を取り扱うが, その洗濯物によってロボットが移動できないことが考えられる. そこで洗濯物を避けて目的地まで移動できる経路を決定するシステムの構築が必要である.

3.4.1 予測線の算出

第一段階では, 複数センサを内蔵した Kinect から画像情報を取得し, 取得されたデータを元に深度情報から床を検出する. ロボットと障害物は大きさにより判別を行う. また, ロボットの直線上右側に目的地を設定し, 検出されたロボットから目的地までの移動経路予測線を算出する. 以下にそのアルゴリズムの概要を説明する.

取得された画像データを元にロボットと目的地の座標を設定し, ロボットと目的地を線で結び移動経路予測線とする. 移動経路予測線が算出された際, 予測線上に障害物(洗濯物)が無ければそのまま移動を行い目的地に到達できる(図 8).

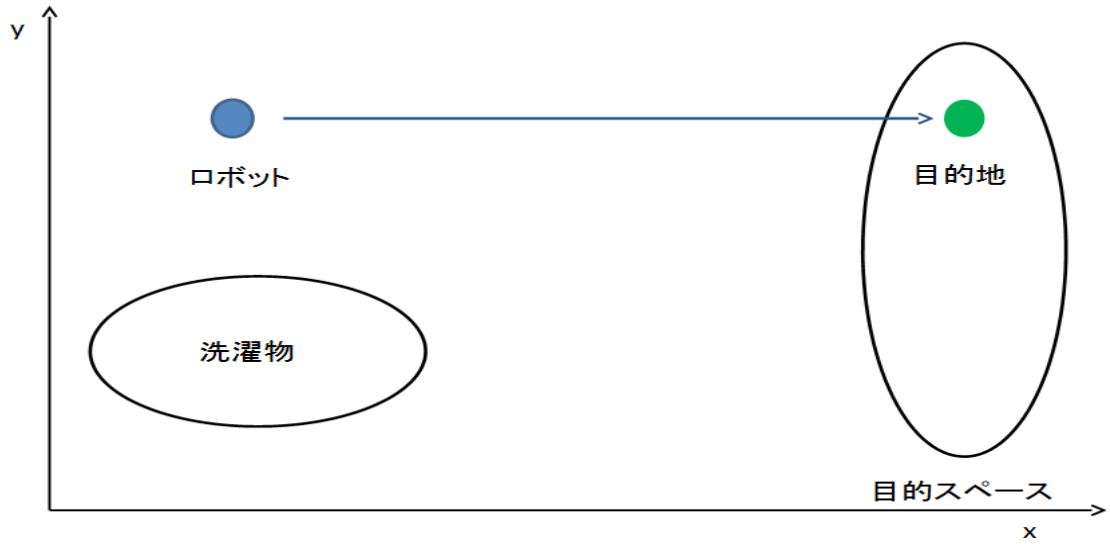

図 8:ロボットから目的地までの移動予測線

予測線上に障害物が存在する場合, その障害物を避けて目的地まで移動しなければならない. 予測線からの接線方向を x, 法線方向を y とし, 検出された障害物の x と y の両頂点座標 a, b, c, d を取得し, a, b, c, d の座標の範囲を囲むことのできる正方形または長方形の障害物として認識する(図 9).

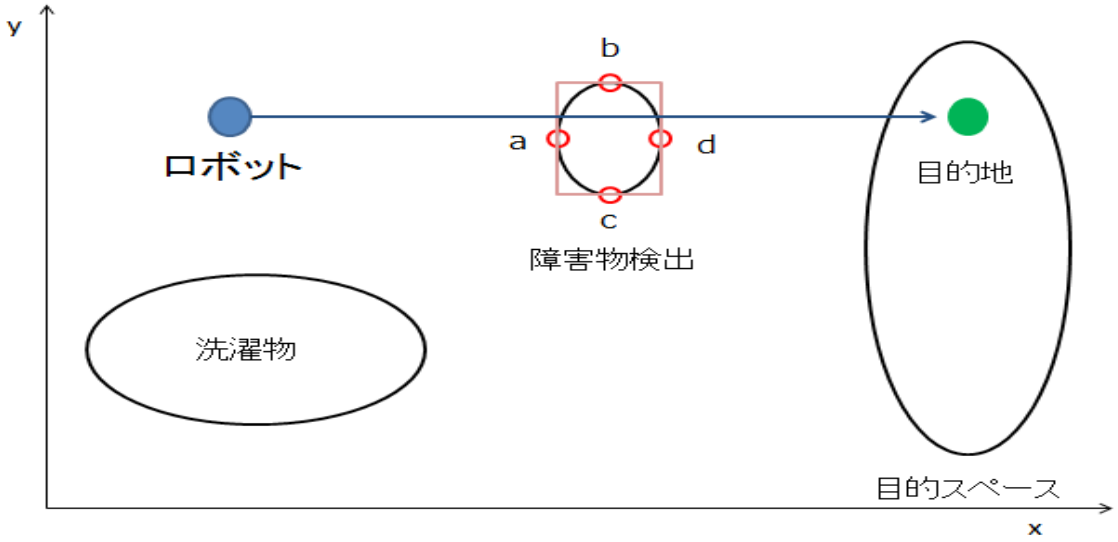

図 9: 予測線上の障害物

ロボットが障害物を避ける経路変更を行うための中間点を予測線上に 4 つ設ける. ロボットの車体分-方向にずれた-x 頂点座標を中間点 1, -x 頂点座標を中間点 2, x 頂点座標を中間点 3, ロボットの車体分+方向にずれた x 頂点座標を中間点 4 として配置する(図 10).

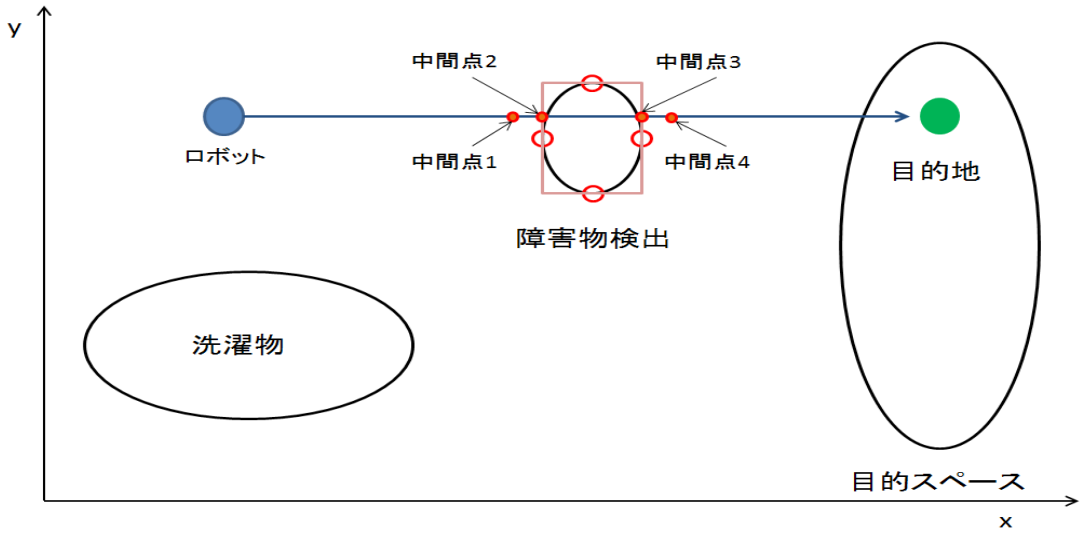

図 10: 予測線上の中間点配置

どの方向に避けるかを決めるため，予測線からの法線距離を y の両頂点座標から計算し距離の短い方向に中間点 2 と中間点 3 を障害物＋ロボットの車体分ずらした位置に移動することにより障害物を避けた予測線を作成することができる(図 11).

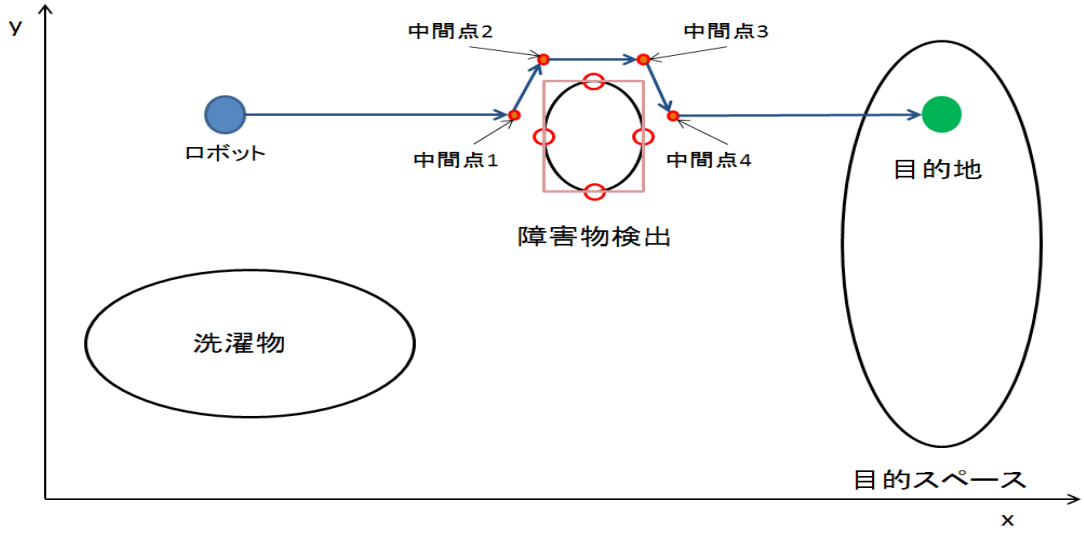

図 11 : 障害物を避けたロボットから目的地までの移動予測線

3.4.2 ロボット命令モジュール

第二段階では，算出された予測線を移動できるようロボットのモータ制御を行うための命令モジュールの算出を行う. ロボットから中間点，中間点から中間点，中間点から目的地への区間距離をピクセル数により計算し，ロボットの移動時間を決定する.

点同士の移動の計算は，ロボットから中間点 1，中間点 1 から中間点 2，中間点 2 から中間点 3，中間点 3 から中間点 4，中間点 4 から目的地の 5 区間である. この区間の移動で y 座標の変化する斜め方向への移動となる区間があるため，点と点の x 座標の差と点と点の y 座標の差を用い三平方の定理より距離を求め，求めた距離を指定したピクセル数で除算した値がその区間を移動する秒数となる.

障害物を避けるため中間点がロボットから目的地までの直線上に無い場合は，中間点で方向転換を行わなければいけない. そのため，中間点 1～4 では三角関数より cosθ の角度を求める.

方向転換を行う角度 θ は，点と点の x 座標の差を a，三平方の定理より斜辺 c とし，以下の式で求めることできる.

$$\cos\theta = acos\left(\frac{a}{c} \times \frac{180}{\pi}\right)$$

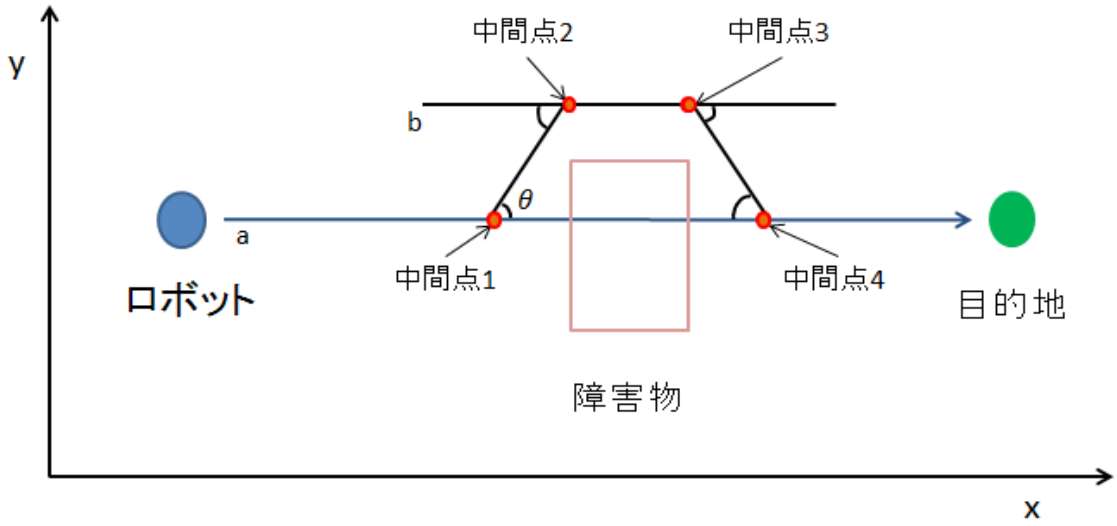

図 12：方向転換の角度

図 10 と図 11 の説明から，中間点 2 と中間点 3 の予測線から y 方向への変位距離が同じであるため，線 a と線 b が平行となる．中間点 2 から中間点 1，中間点 3 から中間点 4 への x 方向への変位距離が同じため，中間点 1 と中間点 2，中間点 3 と中間点 4 の錯角が同じになる．中間点 1 と中間点 4 の内角が同じであるため cosθ のみを求めることで方向転換を行う共通の角度となる．

求めた角度 cosθ と方向転換を行う角度を比較し，方向転換を行う秒数を決定する．方向転換を行う時のモーターの回転数を固定し，1 秒間に移動する角度を 20°に固定した．そのため方向転換を行う角度は 20°刻みとなる．方向転換を行う角度と秒数の関係を表 6 に示す．

図 6：方向転換を行う角度と秒数

移動する角度	秒数	条件
20°	1 秒	求めた角度 cosθ が 0°～20°
40°	2 秒	求めた角度 cosθ が 21°～40°
60°	3 秒	求めた角度 cosθ が 41°～60°
80°	4 秒	求めた角度 cosθ が 61°～80°

第 4 章　実験と評価

4.1 概要

本章では, ロボットから目的地までの移動経路作成とロボットの実動作の検証を行った.

4.2 実験方法

広いスペースを確保できている状態で床から約 1m 離した下向きに Kinect を設置し,画面内にロボットと洗濯物を映るよう配置し検証した. 実験は以下のような手順で行った.

1)Kinect から RGB 画像, 距離画像を取得する
2)距離画像を元に床, ロボット, 洗濯物を判別する
3)目的地をロボットの右側直線上に設定する
4)ロボットから目的地までを線で結び予測線とする
5)予測線上に洗濯物が存在する場合, 洗濯物を正方形または長方形の障害物として
　認識する
6)中間点を 4 点設定し, 障害物を避けた予測線を算出する
7)ロボット命令モジュールのアルゴリズムによりロボットの移動時間と方向転換を行う方
　向・時間を決定する
8)決定された移動時間と方向転換を行う方向・時間からロボットのモーター制御モ
　ジュールを出力する

4.3 実験結果と評価

4.3.1　実験結果

洗濯物(障害物)を配置した状態からロボットが目的地まで到達できるかを検証するため, 移動経路予測線算出実験(図 13)と移動経路実動作実験(図 16)を行った.

決定された座標の一例を図 14 と図 15 に示す. 図 14 と図 15 はロボットと中間点 4 つと目的地の座標を示している. start はロボットの座標, mp1 は中間点 1 の座標, mp2 は中間点 2 の座標, mp3 は中間点 3 の座標, mp4 は中間点 4 の座標, goal は目的地の座標のことである. また, それぞれの座標の位置は図 13 の 1 の画像の描画されている点の左がロボット, 右が目的地である. 図 13 の 2, 3, 4 の画像では描画されている点の左からロボット, 中間点 1、中間点 2, 中間点 3, 中間点 4, 目的地となっている.

図 13 の 1 の画像はロボットの座標と目的地の座標を線で結び，予測線としたものである．図 13 の 2 の画像は障害物が存在したため中間点を 4 つ配置したもので，その時の座標は図 14 の通りである．図 13 の 3 の画像は障害物を避けるため中間点 2 と中間点 3 の y 座標を変化させたもので，その時の座標は図 15 の通りである．図 13 の 4 の画像はロボットから目的地までの座標を順に結び，予測線としたものである．

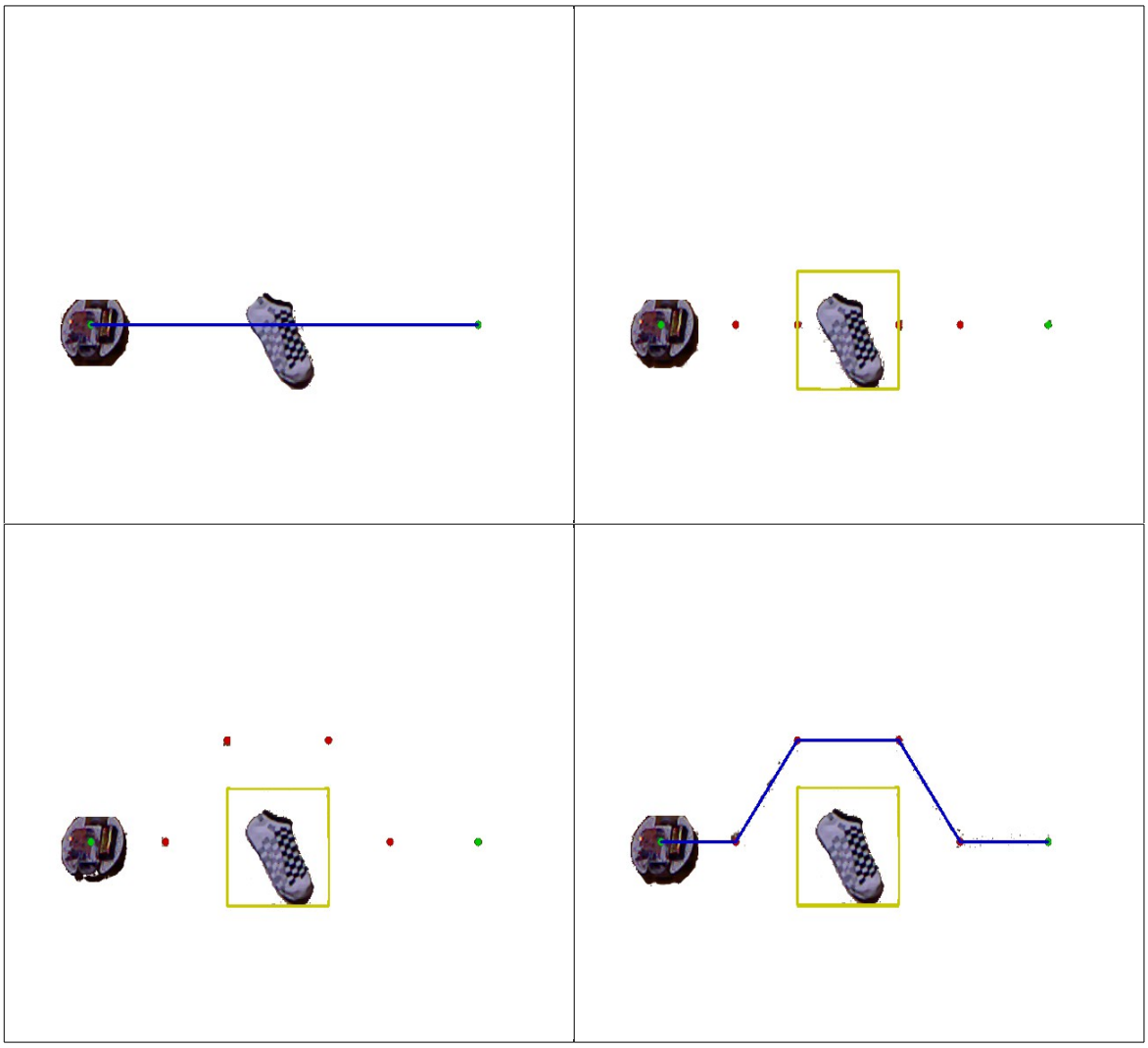

図 13:移動経路予測線算出実験

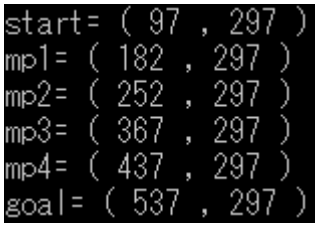

図 14:図 13 の 2 の中間点配置の座標

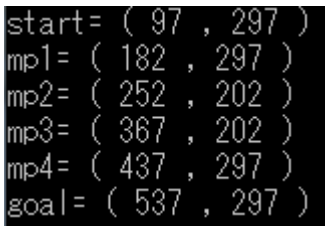

図 15:図 13 の 3 のを避けた座標

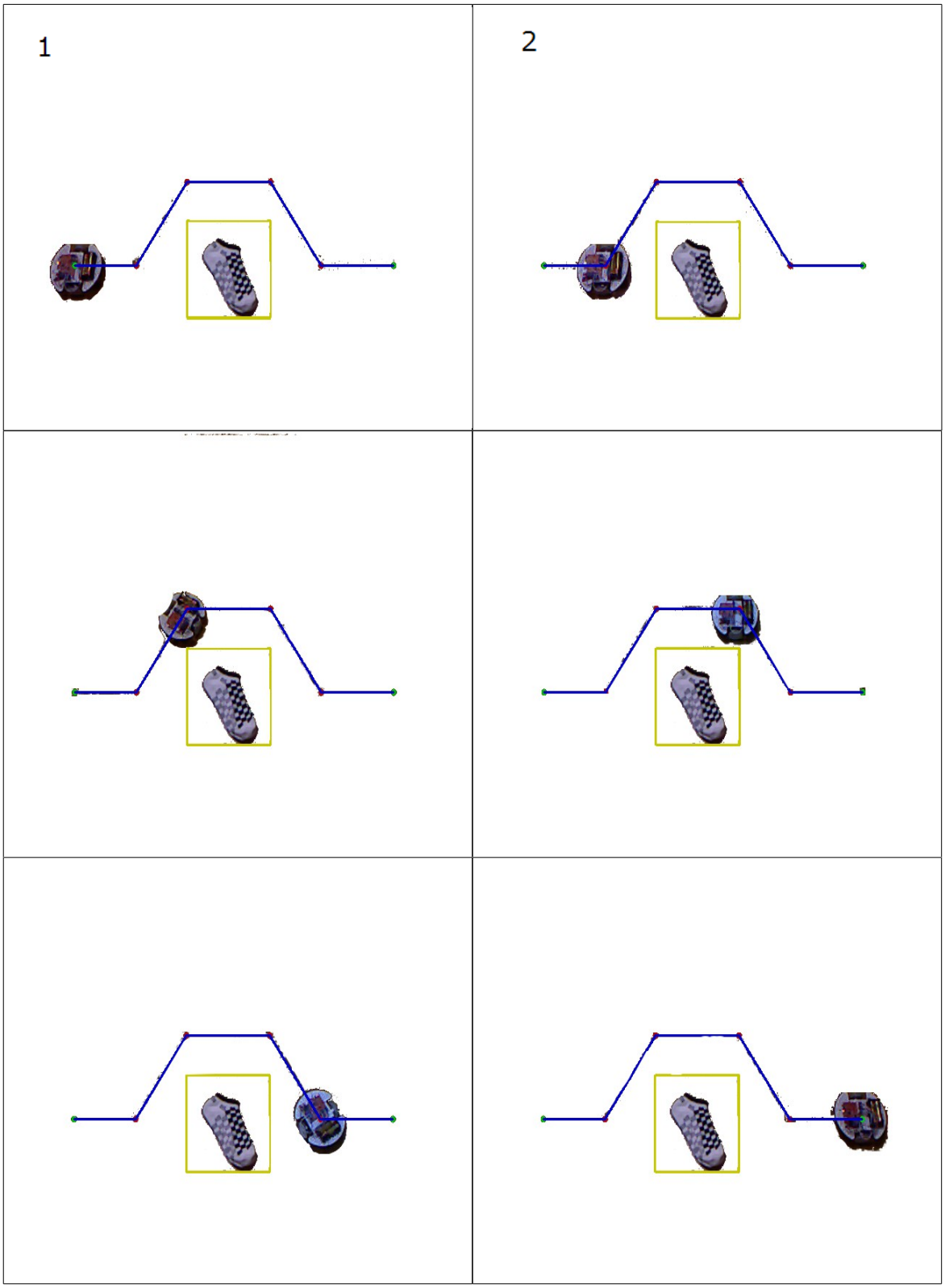

図 16:ロボットの実動作実験

図 16 の 1 の画像は予測線が算出された状態である, 図 16 の 2 の画像は図 16 の 1 の状態から 4 秒前進した状態である, 図 16 の 3 の画像は図 16 の 2 の状態から 60°方向転換し 6 秒前進した状態である, 図 16 の 4 の画像は図 16 の 3 の状態から 60°方向転換し 6 秒前進した状態である, 図 16 の 5 の画像は図 16 の 4 の状態から 60°方向転換し 6 秒前進した状態である, 図 16 の 6 の画像は図 16 の 5 の状態から 60°方向転換し 5 秒前進した状態である.

以下の図 17 は実験で作成されたロボット制御モジュールの命令リストである. Mtr_Run 関数と wait 関数の処理を 1 つの制御とし, Mrt_Run 関数でモーターの回転方向の指定, wait 関数で動作時間を指定する.

```
Mrt_Run_lv(10000,-10000,0,0,0,0); //前進
wait(4000);
Mrt_Run_lv(0,5000,0,0,0,0); //左タイヤのみ後進
wait(3000);
Mrt_Run_lv(10000,-10000,0,0,0,0); //前進
wait(6000);
Mrt_Run_lv(0,-5000,0,0,0,0); //左タイヤのみ前進
wait(3000);
Mrt_Run_lv(10000,-10000,0,0,0,0); //前進
wait(6000);
Mrt_Run_lv(0,-5000,0,0,0,0); //左タイヤのみ前進
wait(3000);
Mrt_Run_lv(10000,-10000,0,0,0,0); //前進
wait(6000);
Mrt_Run_lv(0,5000,0,0,0,0); //左タイヤのみ後進
wait(3000);
Mrt_Run_lv(10000,-10000,0,0,0,0); //前進
wait(5000);
```

図 17:実験での制御モジュール

今回は同じ障害物を使用した実験 5 回と大きさや厚さが異なる障害物 3 種類を使用した実験 3 回の計 8 回の実験を行った.

図 7:使用した障害物と成功回数

使用した障害物	成功した回数(成功回数/実験回数)
靴下	5/5
ズボン	1/1
ペットボトル	1/1
ビー玉	0/1

今回の実験ではビー玉以外の障害物を避けた予測線算出と実動作の確認ができた.

4.3.2 考察

図 12 と図 16 より移動予測線算出実験では洗濯物を四角形の障害物と認識し，その障害物を避けた経路決定を算出できた．さらに算出された経路から区間を移動する秒数，回転時間の算出をすることができた．実動作実験では誤差が確認されることがあったが目的地まで到達することが確認できた．

今回の実験で二つのことを確認することができた．一つ目はある程度の厚さがある障害物，障害物が一つだけである場合は移動経路を決定し動作することが確認できた．深度情報がしっかりできていなかったためか，薄い・小さい障害物は認識されない，また，一部が極端に薄い洗濯物などは二つの物体として認識されるためより正確な情報を取得できればロボットが目的地まで到達できない回数を減少することができると推測される．

二つ目は移動の際，予測点と移動後の位置に誤差が生じることが確認できた．それは，移動秒数を小数点以下を四捨五入した整数を指定するために生じた誤差であると考えられる．その誤差を軽減するとより正確にロボットの移動を制御できると推測される．

第 5 章　まとめ

本論文では，ロボットの移動経路決定法を提案した．障害物を複雑な経路で回避させると方向転換をこまめに行うのでロボットへの命令回数が増え目的地までの到達時間が遅くなる．命令数を減らすため，ロボット動作を直線軌道に近づけ動作を簡略化することにより，目的地までの到達時間を軽減することが確認できた．しかし，算出された経路を移動するが区間で少しずつズレが生じ，目的地と離れた場所に到達してしまった．これは，移動時間を時間の小数点を四捨五入した整数で指定しているため，その小数点分だけ予測点との誤差になったと考えられる．

今後の課題として，移動時間の指定で小数点部分を考慮したモーター制御を行えるようにすることが考えられる．そのために，移動時間計算に除算する値(ピクセル数)をモーターの回転数を考慮した値に変更することで，同じ秒数でより移動予測線を正確に再現することができると推測する．

謝辞

本論文を作成するに当たり，ご指導頂いた三好力教授には深く感謝いたします．また，様々なアドバイスや相談を下さった同三好研究室の皆様や，友人の皆様に深く感謝いたします．

参考文献

[1]長江 庸康 “我が国におけるサービスロボット技術の戦略的開発：家庭用ロボット掃除機における事例研究”
佐野短期大学(2013)
<http://ci.nii.ac.jp/els/110009614537.pdf?id=ART0010079959&type=pdf&lang=jp&host=cinii&order_no=&ppv_type=0&lang_sw=&no=1414566314&cp>
[2]長江 庸康 ”我が国におけるロボット技術の戦略的開発 -サービスロボット技術の推進”
佐野短期大学(2012)
<http://ci.nii.ac.jp/els/110009424277.pdf?id=ART0009902433&type=pdf&lang=jp&host=cinii&order_no=&ppv_type=0&lang_sw=&no=1414591529&cp>

[3]中村薫(2011)「Kinect センサープログラム」秀和システム

[4]杉山懐吾 Kinect を用いた自立制御ロボットの制御
北海学園大学(2012)
<http://hokuga.hgu.jp/dspace/bitstream/123456789/2088/1/%E8%AB%96%E6%96%8708.pdf>

[5]健名祐希 3 台の Kinect を搭載した移動ロボットによる特定物体のハンドリング
<http://hokuga.hgu.jp/dspace/bitstream/123456789/2382/1/%E2%91%A5%E5%81%A5%E5%90%8D%E8%AB%96%E6%96%87.pdf>

[6]人物の挙動認識に基づく自律移動型ロボット制御の研究
<https://www.ipsj-kyushu.jp/page/ronbun/hinokuni/1002/B-6/B-6-1.pdf#search='Kinect%E3%82%92%E7%94%A8%E3%81%84%E3%81%9F%E3%83%AD%E3%83%9C%E3%83%83%E3%83%88'>

[7]“Kinect – Xbox.com”
<http://www.xbox.com/ja-JP/kinect>

[8]“OpenCV - 逆引きリファレンス”
<http://opencv.jp/cookbook/index.html>

付録

```
#include <iostream>
#include <stdexcept>
#include <sstream>
#include <vector>
//#include <OpenNI.h>
#include <opencv2/opencv.hpp>
#include <opencv/highgui.h>
#include <XnCppWrapper.h>
#include"keiro.h"
#include"buttai.h"

///*
//Debug モード
#pragma
comment(lib,"C:\\OpenCV2.2\\lib\\opencv_core220d.lib")
#pragma
comment(lib,"C:\\OpenCV2.2\\lib\\opencv_imgproc220d.lib")
#pragma
comment(lib,"C:\\OpenCV2.2\\lib\\opencv_highgui220d.lib")
#pragma
comment(lib,"C:\\OpenCV2.2\\lib\\opencv_objdetect220d.lib")
//*/
/*
//Release モード
#pragma comment(lib,"C:\\OpenCV2.2\\lib\\opencv_core220.lib")
#pragma
comment(lib,"C:\\OpenCV2.2\\lib\\opencv_imgproc220.lib")
#pragma
comment(lib,"C:\\OpenCV2.2\\lib\\opencv_highgui220.lib")
#pragma
comment(lib,"C:\\OpenCV2.2\\lib\\opencv_objdetect220.lib")
*/

//設定ファイルのパス
const char* CONFIG_XML_PATH = "SamplesConfig.xml";
//#define SAMPLE_XML_PATH "./SamplesConfig.xml"
using namespace cv;
using namespace xn;

typedef std::vector<float> depth_hist;

int main()
{
int i,j,k;
static const double pi = 3.14159265358979265358979;
//double a,b,c,d,e,f,g,f2,g2,s,bx,by,bx2,by2;
//int sx,sy,t1x,t1y,t2x,t2y,t3x,t3y,t4x,t4y,m2x,m2y,m3x,m3y,gx,gy;
//double t1,t2,t3,t4;
int bright_min,bright_max;
int bright_avg;
//OpenNI
DepthGenerator depthGenerator;
ImageGenerator imageGenerator;
DepthMetaData depthMD;
ImageMetaData imageMD;
SceneMetaData sceneMD;
Context context;
//OpenCV
//cv::Mat img;
Mat img(480,640,CV_8UC3); //RGB 画像
Mat image(480,640,CV_8UC3); //RGB 画像
Mat depth(480,640,CV_16UC1); //深度画像
Mat mask(480,640,CV_16UC1);
Mat initmask(480,640,CV_8UC3);
Mat bgmask(480,640,CV_8UC3);
Mat midmask(480,640,CV_8UC3);
Mat bgMask(480,640,CV_16UC1);
Mat midMask(480,640,CV_16UC1);
Mat tmp_img1(480,640,CV_8UC3);
Mat tmp_img2(480,640,CV_8UC3);
Mat dst_img1(480,640,CV_8UC3);
Mat dst_img2(480,640,CV_8UC3);
//Mat canny_img(480,640,CV_8UC1);
Mat tmp_dst1(480,640,CV_8UC1);
Mat imgtmp1(480,640,CV_16UC1);
Mat tmp_dst2(480,640,CV_16UC1);
Mat imgtmp2(480,640,CV_16UC1);
Mat canny_img;
Mat bin_img;
vector< vector<cv::Point> > contours;
//Mat contourImage(480,640, CV_8U, Scalar(255));
//Mat init_contourImage(480,640, CV_8U, Scalar(255));
const int drawAllContours = -1;
Mat bright_mask(480,640,CV_8UC1);
Mat bright_Mask(480,640,CV_8UC3);
Mat rgbmask(480,640,CV_8UC3);
Mat depth_bg(480,640,CV_16UC1); //深度背景画像
//OpenNI の初期化
context.FindExistingNode(XN_NODE_TYPE_IMAGE,imageGener
ator);
context.FindExistingNode(XN_NODE_TYPE_DEPTH,depthGener
ator);
//context.FindExistingNode(XN_NODE_TYPE_USER,userGenerat
or);
//RGB 画像と振動画像のズレを補正
depthGenerator.GetAlternativeViewPointCap().SetViewPoint(imag
eGenerator);
//ウィンドウの準備
cvNamedWindow("image");
cvNamedWindow("depth");
cvNamedWindow("test",1);
unsigned short* dp = (unsigned short*)depth.data;
unsigned short* dp_bg = (unsigned short*)depth_bg.data;
short tmp_dp;
int key = 0;
while (key!='q'){
context.WaitAndUpdateAll();
imageGenerator.GetMetaData(imageMD);
depthGenerator.GetMetaData(depthMD); //距離取得
memcpy(image.data,imageMD.Data(),image.step *image.rows); //
イメージデータを格納
memcpy(depth.data,depthMD.Data(),depth.step *depth.rows); //深
度データを格納
memcpy(depth_bg.data,depthMD.Data(),depth_bg.step
*depth_bg.rows);
Mat pointCloud_XYZ(480,640,CV_32FC3,cv::Scalar::all(0));
retrievePointCloudMap(depth,pointCloud_XYZ);
cv::circle(img, cv::Point(sx,sy), 3, cv::Scalar(0,200,0), -1,
CV_AA); //start
cv::circle(img, cv::Point(t1x,t1y), 3, cv::Scalar(0,0,200), -1,
CV_AA); //mp1
cv::circle(img, cv::Point(t2x,t2y), 3, cv::Scalar(0,0,200), -1,
CV_AA); //mp2
cv::circle(img, cv::Point(t3x,t3y), 3, cv::Scalar(0,0,200), -1,
CV_AA); //mp3
cv::circle(img, cv::Point(t4x,t4y), 3, cv::Scalar(0,0,200), -1,
CV_AA); //mp4
cv::circle(img, cv::Point(gx,gy), 3, cv::Scalar(0,200,0), -1,
CV_AA); //goal
std::cout << "map1" << std::endl;
std::cout << "start= " << "( " << sx << " , " << sy << " )" <<
std::endl;
std::cout << "mp1= " << "( " << t1x << " , " << t1y << " )" <<
std::endl;
std::cout << "mp2= " << "( " << t2x << " , " << t2y << " )" <<
std::endl;
std::cout << "mp3= " << "( " << t3x << " , " << t3y << " )" <<
std::endl;
std::cout << "mp4= " << "( " << t4x << " , " << t4y << " )" <<
std::endl;
std::cout << "goal= " << "( " << gx << " , " << gy << " )" <<
std::endl;
std::cout << "map2" << std::endl;
std::cout << "start= " << "( " << sx << " , " << sy << " )" <<
std::endl;
std::cout << "mp1= " << "( " << t1x << " , " << t1y << " )" <<
std::endl;
std::cout << "mp2= " << "( " << m2x << " , " << m2y << " )" <<
std::endl;
std::cout << "mp3= " << "( " << m3x << " , " << m3y << " )" <<
std::endl;
std::cout << "mp4= " << "( " << t4x << " , " << t4y << " )" <<
std::endl;
std::cout << "goal= " << "( " << gx << " , " << gy << " )" <<
```

```
std::endl;
//a = m2x - t1x;
//b = m2y - t1y;
//f2 = (a*a+b*b);
//f = sqrt(f2);
//s = std::acos(a/f)*180/pi;
std::cout << "角度 r= " << std::acos(a/f) << std::endl;
std::cout << "角度 θ= " << std::acos(a/f)*180/pi << std::endl;
if(int(s) < 20){
std::cout << "角度: " <<  1 << std::endl;
} else if(int(s) < 40){
std::cout << "角度: " <<  2 << std::endl;
} else if(int(s) < 60){
std::cout << "角度: " <<  3 << std::endl;
} else if(int(s) < 80){
std::cout << "角度: " <<  4 << std::endl;
}
if(serect == true){
std::cout << "左:+ " << std::endl;
std::cout << "左:- " << std::endl;
std::cout << "左:+ " << std::endl;
std::cout << "左:- " << std::endl;
}else{
std::cout << "左:- " << std::endl;
std::cout << "左:+ " << std::endl;
std::cout << "左:- " << std::endl;
std::cout << "左:+ " << std::endl;
}
std::cout << "start-mp1: " <<  t1x - sx << std::endl;
std::cout << "mp1-mp2: " << f << std::endl;
//std::cout << "mp1-mp2: " << m2x - t1x << std::endl;
std::cout << "mp2-mp3: " << m3x - m2x << std::endl;
std::cout << "mp3-mp4: " << g << std::endl;
//std::cout << "mp3-mp4: " << t4x - m3x << std::endl;
std::cout << "mp4-goal: " << gx - t4x << std::endl;
if((t1x - sx) % 20 >= 10){
std::cout << "start-mp1: " <<  ((t1x - sx) / 20) + 1 << std::endl;
} else{
std::cout << "start-mp1: " <<  (t1x - sx) / 20 << std::endl;
}
if(int(f) % 20 >= 10){
std::cout << "mp1-mp2: " << int(f / 20) + 1 << std::endl;
} else{
std::cout << "mp1-mp2: " << int(f / 20) << std::endl;
}
if((m3x - m2x) % 20 >= 10){
std::cout << "mp2-mp3: " << ((m3x - m2x) / 20) +1 << std::endl;
} else{
std::cout << "mp2-mp3: " << ((m3x - m2x) / 20) << std::endl;
}
if(int(g) % 20 >= 10){
std::cout << "mp3-mp4: " << int(g / 20) + 1 << std::endl;
} else{
std::cout << "mp3-mp4: " << int(g / 20) << std::endl;
}
if((gx - t4x) % 20 >= 10){
std::cout << "mp4-goal: " << ((gx - t4x) / 20) + 1 << std::endl;
} else{
std::cout << "mp4-goal: " << (gx - t4x) / 20 << std::endl;
}
cv::rectangle(img,        cv::Point(bx,by),        cv::Point(bx2,by2),
cv::Scalar(0,200,200), 2, 4); //object
cv::line(img,            cv::Point(sx,sy),            cv::Point(t1x,t1y),
cv::Scalar(200,0,0), 2, 4); //s-mp1
cv::line(img,          cv::Point(t1x,t1y),          cv::Point(m2x,m2y),
cv::Scalar(200,0,0), 2, 4); //mp1-mp2.2
cv::line(img,        cv::Point(m2x,m2y),        cv::Point(m3x,m3y),
cv::Scalar(200,0,0), 2, 4); //mp2.2-mp3.2
cv::line(img,          cv::Point(m3x,m3y),          cv::Point(t4x,t4y),
cv::Scalar(200,0,0), 2, 4); //mp3.2-mp4
cv::line(img,            cv::Point(t4x,t4y),            cv::Point(gx,gy),
cv::Scalar(200,0,0), 2, 4); //mp4-g
//BGR を RGB へ
cvtColor(image,image,CV_RGB2BGR);
//RGB→グレースケール
image.convertTo(imgtmp1,CV_8UC3);
cvtColor(imgtmp1, tmp_dst1, CV_BGR2GRAY);
bright_avg = tmp_dst1.data[tmp_dst1.cols *
tmp_dst1.rows /2 + tmp_dst1.cols/2];
bright_min = bright_avg - 50;
bright_max = bright_avg + 50;
if(tmp_dst1.data[k] >= bright_min &&
tmp_dst1.data[k] <= bright_max ){
bright_mask.data[i*bright_mask.cols + j] = 255;
}
else{
bright_mask.data[i*bright_mask.cols + j] = 0;
}
bright_mask.convertTo(bright_Mask,CV_8UC1);
image.copyTo(rgbmask,bright_Mask);
//画面に表示
imshow("image",image);
imshow("depth",depth);
cv::imshow("test",img);
//imshow("player",player);
//imshow("playerDepth",playerDepth);
//imshow("background",background);
//imshow("bgdepth",bgdepth);
//imshow("mask2",midmask);
//imshow("midmask",midMask);
//imshow("laplace1",dst_img1);
imshow("bin",bright_mask);
imshow("canny",rgbmask);

key = waitKey(33);
}
context.Shutdown();
return 0;
}
```