平成24年度修士論文

相手の行動を観察・学習し真似をする エアホッケーロボットの開発

大学院情報システム学研究科
情報メディアシステム学専攻

学籍番号 : 1150016
氏名 : 佐藤 雄也
主任指導教員 : 末廣 尚士 教授
指導教員 : 工藤 俊亮 准教授
指導教員 : 布施 哲治 客員准教授
提出年月日 : 平成25年02月22日

目 次

図 目 次

表 目 次

第1章　緒言

1.1　研究背景

近年，ロボットの活躍する場は産業分野のみならず，一般社会にまで深く浸透してきており，人間と密接な関係を持つロボットが増えてきている．このような身近な存在の一つに，エンターテインメントロボットがある [1] [2] [3] [4] [5]．これは，人間を楽しませることを目的として研究・開発が進められているロボットであり，ペット型ロボットや，ロボット玩具はその一例といえる．当研究室ではこれまで，その一分野として，娯楽性を追求したエアホッケーロボットシステムの開発を目指し研究してきた．

エアホッケーロボットにおける対戦アルゴリズムは様々な研究機関より提案されており，近年では娯楽性に加え，戦略も重視した高度な手法が増えてきている．しかし一方で，従来のアルゴリズムのほとんどは，ロボットの動きを事前に定義する必要があるため，対戦中の動作の種類には限度があり，対戦相手である人間にロボットの戦い方を予想されやすい．近年ではそういった状況を避けるため，対戦中の状況に応じて行動を変化させるアルゴリズムがいくつか提案されてきているが，現状はまだ発展途上であるといえる．また，従来のアルゴリズムはロボットの動きを厳密に設計する必要があるため，複雑なアルゴリズムを開発するのが難しい．これらの問題を解決し，ロボットが人間のように柔軟で様々な打ち方をすることができるようになれば，対戦時の娯楽性が飛躍的に向上し，エンターテインメントロボットとして従来よりもさらに対戦相手を楽しませることができることが期待される．

本研究では，上記の視点からエアホッケーロボットの娯楽性を向上させる手法の一つと

して，対戦相手の行動を観察・学習するシステムの実現を目標とした．相手の動きを真似することにより，対戦中に相手の行動を学習することにより戦い方が随時変化するため，対戦相手に戦略を予想されにくくなることが予想される．また，ロボットの戦い方を設計する際においても，人間が直接打ち方を見せることで，ロボットがその動きを学習するようになるため，設計における開発者の負担を軽減することができると考えられる．さらに，ある個人の打ち方を学習させておき再現することで，対戦相手にあたかもその人と対戦しているかのような錯覚を起こすことも実現可能であるといえる．このように，対戦相手の打ち方を模倣することにより，さらに娯楽性が向上することが期待できる．

1.2 関連研究

エアホッケーロボットに関する研究は数多くなされており，ロボットの性能向上やエンターテインメント性の向上において高い成果をあげている．エアホッケーロボットシステム全般における研究では，Bishop らは視覚システムを用いたエアホッケーロボットについて，ホッケー台上における摩擦やパック自体の回転がパックの軌道にどのように影響を与えるかを調査し報告した [6]．覺張らは人間と同等に対戦可能なロボットシステムの開発を目的として，攻撃動作や防御動作を高速に行うことができるエアホッケーロボットを開発・提案した [7] [8] [9]．牧野らはエアホッケー台上を移動するパックの軌道を高精度で把握するための視覚システムを提案し，ロボットアームとの協調動作を実現した [10] [11] [12]

攻撃戦略に関する研究では，那順らはエンターテインメント性を向上させることを目的として，つくば打ちという打ち返し方を提案し，ロボットの攻撃能力の向上を図った [13] [14]．御堂丸らはロボット側に向かってくるパックを一旦止めることで，パックの打ち返しにおけるコントロール性を向上させる手法を提案した [15] [16] [17]．松下らはロボットの運動能力と判断能力を調整することにより，対戦相手のレベルに応じて強さを変更可能なロボットシステムを提案し有効性を検証した [18]．

1.3 研究目的

本研究の目的は，ロボットが対戦相手の打ち方を学習し真似をするアルゴリズムを開発すること，また，アルゴリズムを実機上で対戦できるようなシステムを構築することである．対戦相手の打ち方を真似することにより，従来のアルゴリズムのようにロボットの打ち方を細かく設計し定義することなく，人間の打ち方を実装することが可能となる．また，対戦相手と実際に対戦できる環境を構築することで，実機上でアルゴリズムが有効であるかどうかについて検証可能となる．

1.4 論文構成

本論文の構成は次の通りである．第 2 章では，本研究で提案する模倣アルゴリズムの実現方法や学習データについて述べる．第 3 章では，模倣アルゴリズムを開発・実験する上で構築したエアホッケーロボットシステムについて詳しく述べる．第 4 章では，第 3 章で述べるシステム上に模倣アルゴリズムを実装した場合に，対戦相手の打ち方の特徴を維持しつつ再現可能であるかどうかについて調査する．また，対戦中に学習するデータの探索時間についても計測を行い，本アルゴリズムが有効に動作し打ち返すことができるかどうかについて検証する．最後に第 5 章では，本論文のまとめおよび今後の課題について述べる．

第2章　模倣アルゴリズム

模倣アルゴリズムは，過去に対戦相手がパックを打ち返した際の状況と似たような状況にロボットが遭遇した際，そのときの対戦相手と同じようにマレットを動かし打ち返すことで，当時の打ち方を再現することができるという考えに基づいている．本論文では，これに基づいた模倣の基礎として，簡単なアルゴリズムを提案する．

本アルゴリズムは，対戦相手の打ち返し動作（マレットの動かし方）を，そのときのパックの動き（対戦相手側へ向かうパックの進入速度・角度）や，打ち返した瞬間のパックの位置と結びつけて記憶する．そして，ロボット側に向かってパックが進入してきた際，そのパックの動きが過去に記憶した動きと似ている場合に．当時の打ち返し動作を再現することで打ち返しをする．本章では，本アルゴリズムの概要や，学習・再現に使用するデータについて述べた後，具体的な学習・再現の方法について述べる．

2.1　アルゴリズムの概要

提案する模倣アルゴリズムは，対戦相手の打ち返し方（マレットの動かし方）が，パックの進入速度・角度によって決定されるという考え方に基づいている．本アルゴリズムは，以下の 2 つの処理から構成される．

- 対戦相手の打ち方を学習する処理（学習処理）
- 学習した打ち方を再現しパックを打ち返す処理（再現処理）

学習処理では，対戦相手が打ち返したパックの進入時の速度や角度，打点に対応するマレットの動かし方を，2.2 にて述べる “行動要素” として記憶していく．また，再現処理で

は，ロボット側に向かってくるパックについて，過去に学習した行動要素のうち，データが類似するものを検索し，それに対応するマレットの動かし方を再現し打ち返す．これら 2 つの処理については，2.3 以降で詳しく述べる．なお，本アルゴリズムにおけるホッケー台の座標系（フィールド座標系）は，ホッケー台中央を原点とし，ロボット側から見て右方向を $+X$ ，奥方向を $+Y$ とする（図 2.1）．また，以降では対戦相手側のマレットのことを単にマレットと呼ぶこととする．

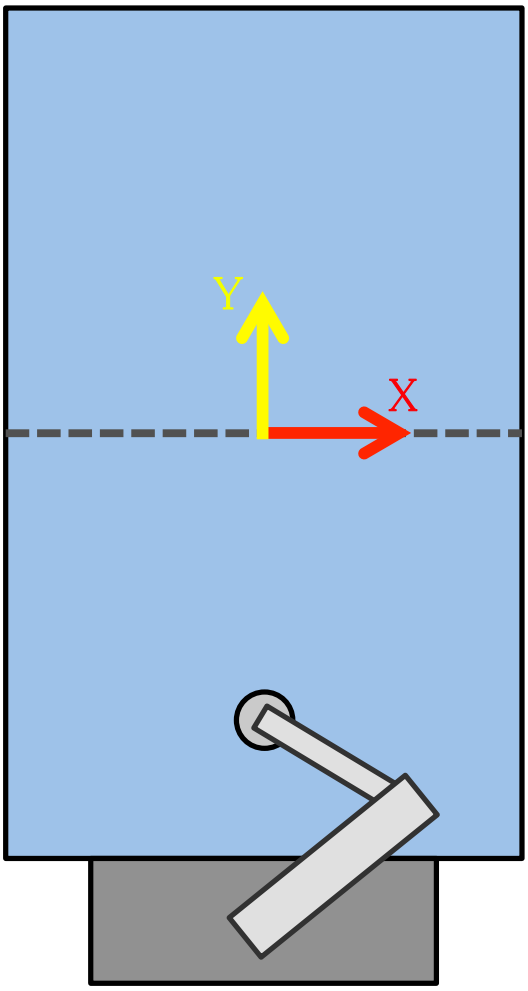

図 2.1: フィールド座標系の定義

2.2 行動要素

パックの打ち方の学習や再現に使用するデータを行動要素と呼ぶ．行動要素は以下の 3 つの要素により構成される（図 2.2）．

（1）パックが対戦相手側の陣地に進入するときのパックの位置・速度ベクトル

（2）パックとマレットが最初に衝突する瞬間のパックの位置

（3）パックが対戦相手側の陣地（ $y < 0$ ）に進入し，再びロボット側の陣地（ $y > 0$ ）に戻るまでの間のマレットの動き

（1）の “パックの位置・速度ベクトル” は，両者の組である $[P_{\mathrm{in}}, \boldsymbol{V}_{\mathrm{in}}]$ で表す．ここで P_{in} は速度ベクトルの始点であり，パックがセンターライン（$y = 0$）上を通る瞬間の座標 $(x_{\mathrm{in}}, y_{\mathrm{in}})$ を表す．また， $\boldsymbol{V}_{\mathrm{in}}$ はパックの速度ベクトルであり， $(v_{x_{\mathrm{in}}}, v_{y_{\mathrm{in}}})$ を表す．(2) の “衝突した瞬間のパックの位置” は座標 $P_{\mathrm{sp}} = (x_{\mathrm{sp}}, y_{\mathrm{sp}})$ で表す．以降，“衝突時パック位置” と表現する．(3) の “マレットの動き” は，パックがセンターライン上を通る瞬間からの経過時間 t_{mallet_i} に対するマレットの座標 $P_{\mathrm{mallet}_i} = (x_{\mathrm{mallet}_i}, y_{\mathrm{mallet}_i})$ の組である，$[t_{\mathrm{mallet}_i}, P_{\mathrm{mallet}_i}]$ の集合で表される．なお，集合は t_{mallet_i} の昇順で並んでおり，時系列データとなっている．

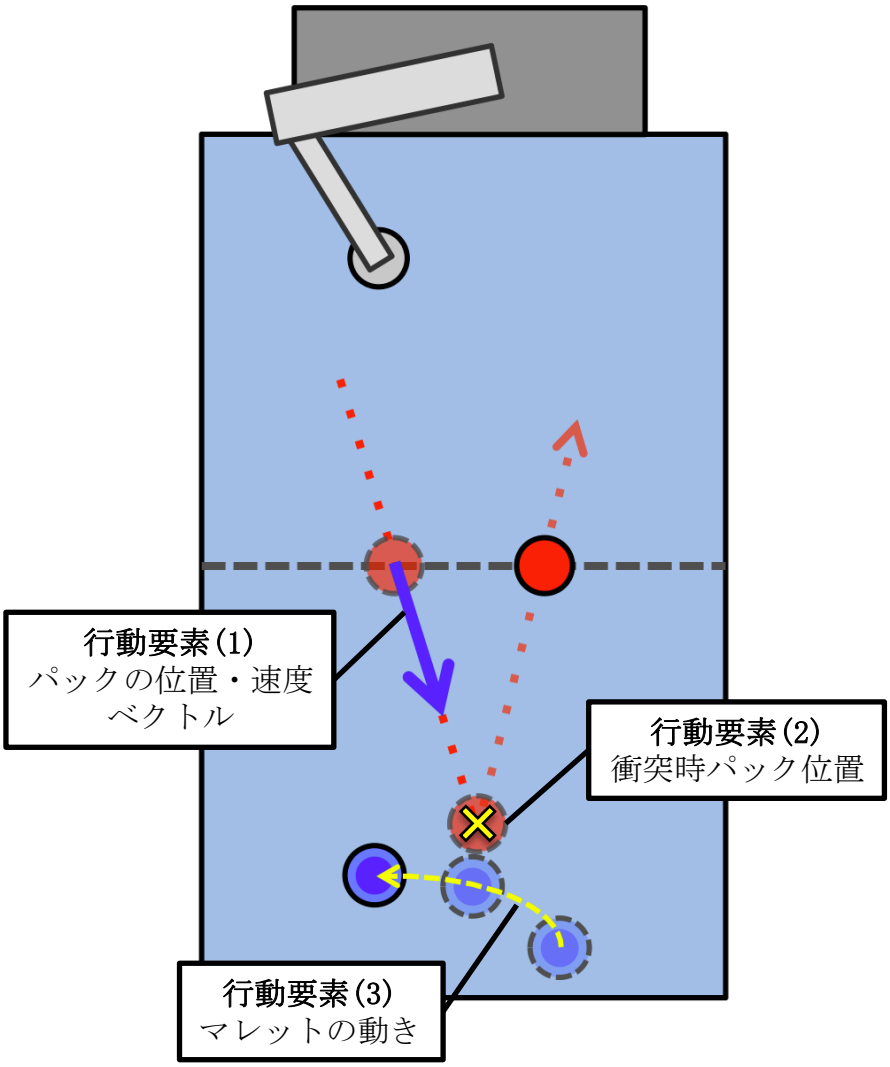

図 2.2: 学習する行動要素

2.3 学習処理

本アルゴリズムにおける学習処理とは，対戦相手が持つマレットの動きやそのときのパックの動きをもとに行動要素を生成し，記録するまでの作業のことをいう．なお，処理の簡単化のため，ここでは以下の条件のもとで学習することとする．

- パックが対戦相手の陣地に進入しロボットの陣地に戻るまで一度も壁に衝突しない
- 対戦相手はパックを打ち損じない（必ずパックとマレットを衝突させて打ち返す）

行動要素を記録する学習処理のフローチャートを図 2.3 に示す．まず，パックがロボット側の陣地から対戦相手側の陣地に進入する際，対戦相手側の陣地に進入するパックの位置・速度ベクトル $[P_{\mathrm{in}}, \boldsymbol{V}_{\mathrm{in}}]$ を求め，行動要素（1）として記録する．また，マレットの動き（行動要素（3））を記録するため，このときからマレットの動きを表す時系列データである $[t_{\mathrm{mallet}_i}, P_{\mathrm{mallet}_i}]$ の一時記憶を開始する．次に，パックがマレットによって打ち返されたとき，両者が衝突した瞬間のパックの座標 P_{sp} を行動要素（2）として記録する．最後に，パックがロボット側の陣地に戻っていった際に，マレットの動きの一時記憶を終了し，行動要素（3）として記録する（行動要素（3）の t_{mallet_i} については，後の再現処理にて再生速度を容易に変更できるよう，パックとマレットが衝突した時刻が 1 となるように時刻をスケーリングする）．なお，行動要素における座標やベクトルについては，後でロボット側の陣地に対して簡単に利用できるよう，対戦相手側の陣地の座標から，ロボット側の陣地の座標へ変換（ π [rad] だけ原点を中心に回転）した状態で記録する．

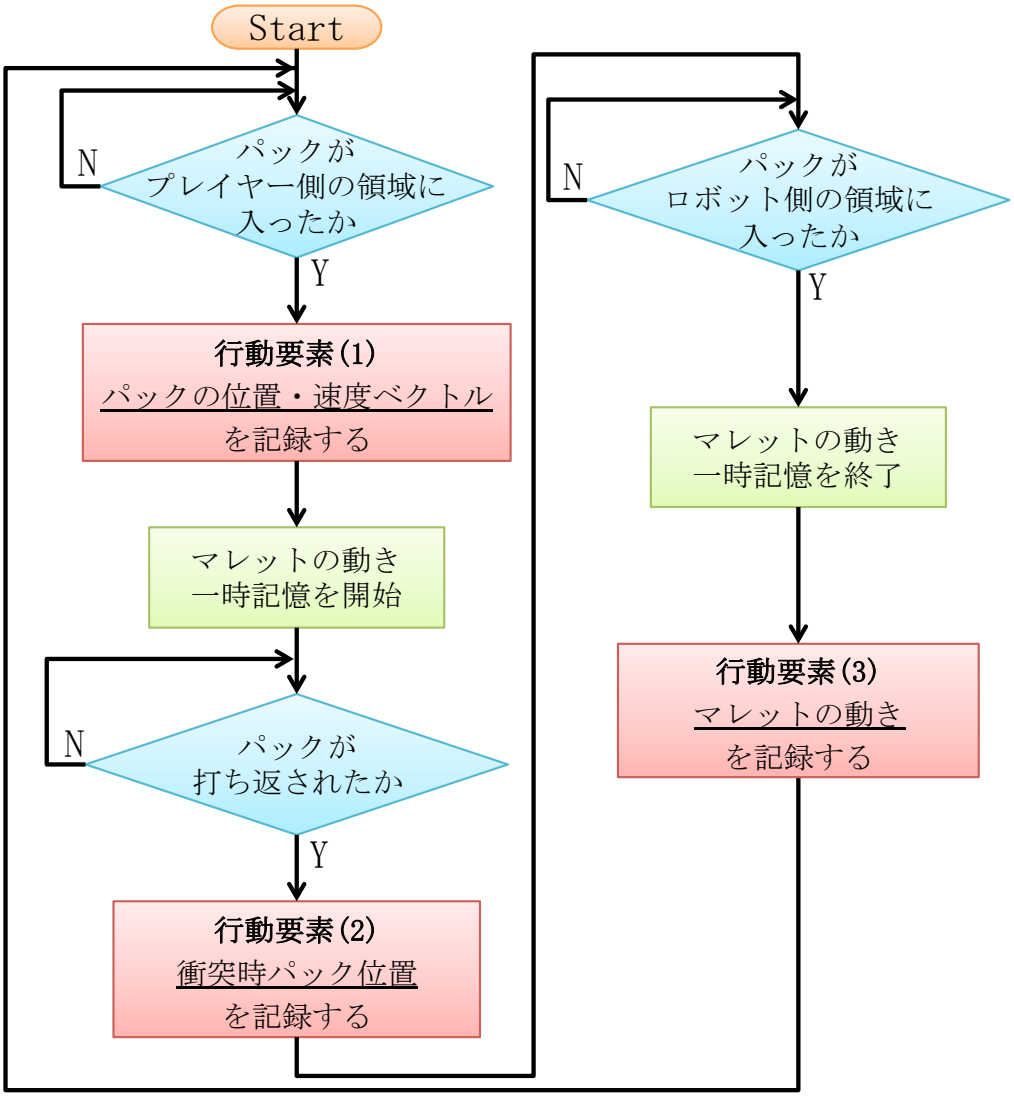

図 2.3: 学習処理のフローチャート

2.4 再現処理

再現処理では，2.3 で生成した行動要素を使い，対戦相手の打ち方の真似をする．再現処理のフローチャートを図 2.4 に示す．対戦相手がロボットに向けてパックを打ち返した際，あらかじめロボット側の陣地に向かうパックの予測軌道（進入位置・速度ベクトル $[P'_{\mathrm{in}}, \boldsymbol{V}'_{\mathrm{in}}]$ ）を求めておく．次に，過去に学習した行動要素の中から，再現すべきデータを検索する．検索処理の基本的な考え方としては，2.3 で生成した行動要素のうち，パックの進入速度や角度（行動要素（1）より求められる），およびパックとマレットが衝突した点（行動要素（2））が現在のパックの予測軌道に近いものを検索する．再現すべきデータが見つかった場合，その行動要素に対応するマレットの動かし方を再現する．以下，具体的な手順について述べる．

2.4.1 パックの軌道が近い行動要素の選択

過去に学習したすべての行動要素の中で，パックの進入速度や角度が大きく異なる行動要素を除外する．具体的には，行動要素を再現すべき候補と見なすための閾値として，速度差および角度差の最大値をそれぞれ v_{diffmax} ， $\theta_{\mathrm{diffmax}}$ とし，以下の条件を満たさない行動要素は除外する．

$$|\|\boldsymbol{V}'_{\mathrm{in}}\| - \|\boldsymbol{V}_{\mathrm{in}}\|| \leq v_{\mathrm{diffmax}}$$

$$|\theta'_{\mathrm{in}} - \theta_{\mathrm{in}}| \leq \theta_{\mathrm{diffmax}}$$

パックの進入速度や角度が大きく異なる場合，対戦相手の打ち方も大きく異なると考えられるため，このような措置を施している．

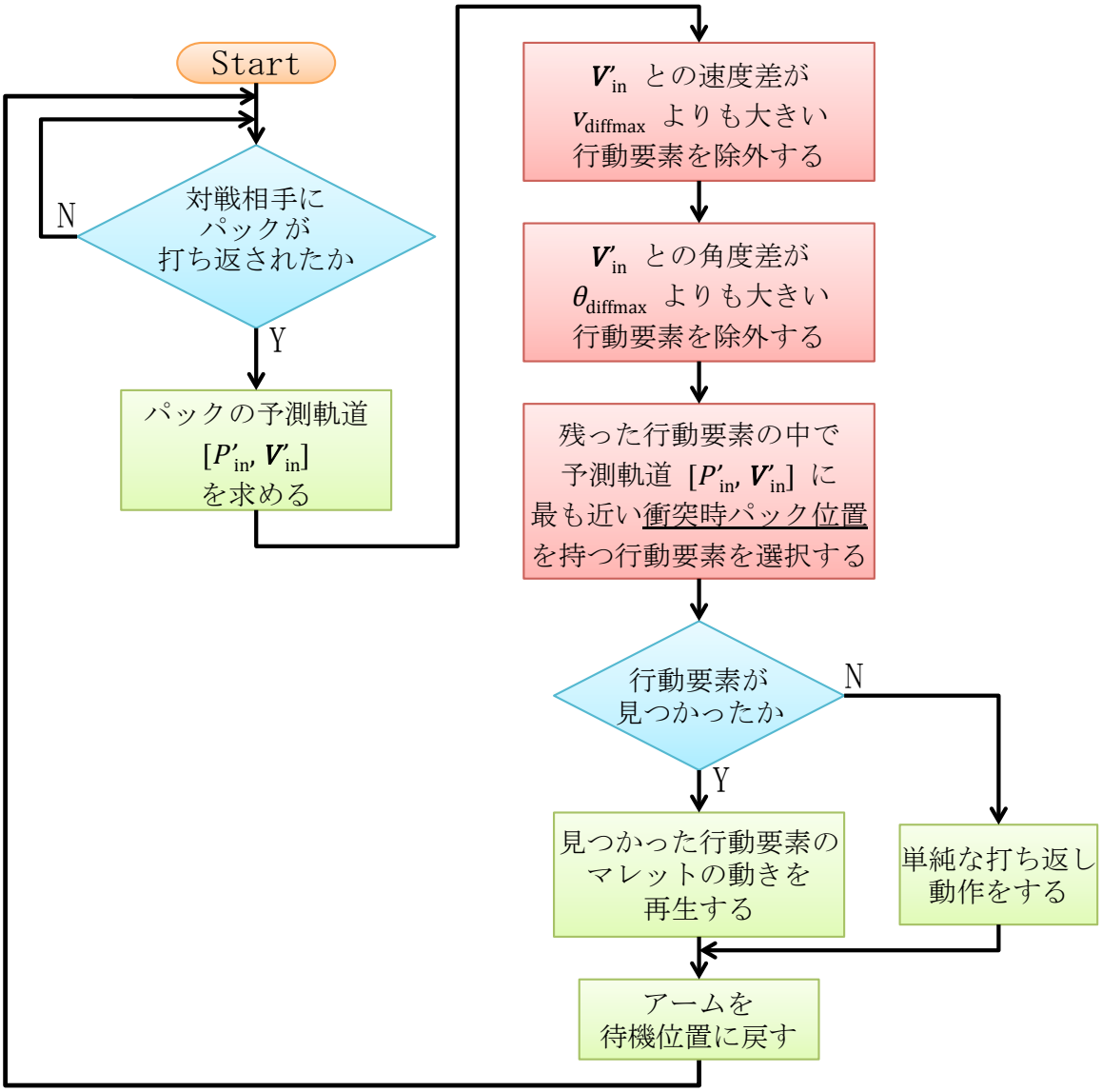

図 2.4: 再現処理のフローチャート

2.4.2　衝突時パック位置が当時の状況に最も近い行動要素の選択

ここまでの処理により，現在向かってきているパックの予測軌道に近い行動要素を抽出することができた．ここでは，残りの行動要素のうち，最終的に再現する行動要素の選択方法について述べる．

再現処理における打ち返し処理では，ロボットの手先を “マレットの動かし方”（行動要素（3））の軌道に基づいて動かすことでパックを打ち返す．しかし，記録した軌道のまま動かすだけでは，学習当時の衝突時パック位置が現在のパックの軌道直線上から離れていることが多く，衝突時パック位置が当時の位置とずれるため，打ち返しは基本的に成功しない．そこで打ち返し処理では，再現する軌道に対して座標変換を施し，行動要素における衝突時パック位置が軌道直線上に乗るようにすることで，打ち返しを可能にしている．

ここで，2.4.1 で述べたように，パックの進入速度や角度が大きく異なる場合，対戦相手の打ち方も変化することが想定されるが，衝突時パック位置に関しても同様であると考えられる．そこで，残りの行動要素のうち衝突時パック位置が最も近いものを選択し，これを最終的に再現する行動要素とする．

最終的に再現する行動要素の選択方法を図 2.5 に示す．再現する行動要素は，各行動要素のうち，$\boldsymbol{V}'_{\mathrm{in}}$ に沿う直線と衝突時パック位置 P_{sp} の距離 L がもっとも近いものとする．ここで，P_{sp} から $\boldsymbol{V}'_{\mathrm{in}}$ の直線上に引いた垂線との交点を $P_{\mathrm{sp}_{\mathrm{new}}}$ とし，これを打ち返しの目標衝突点とする．

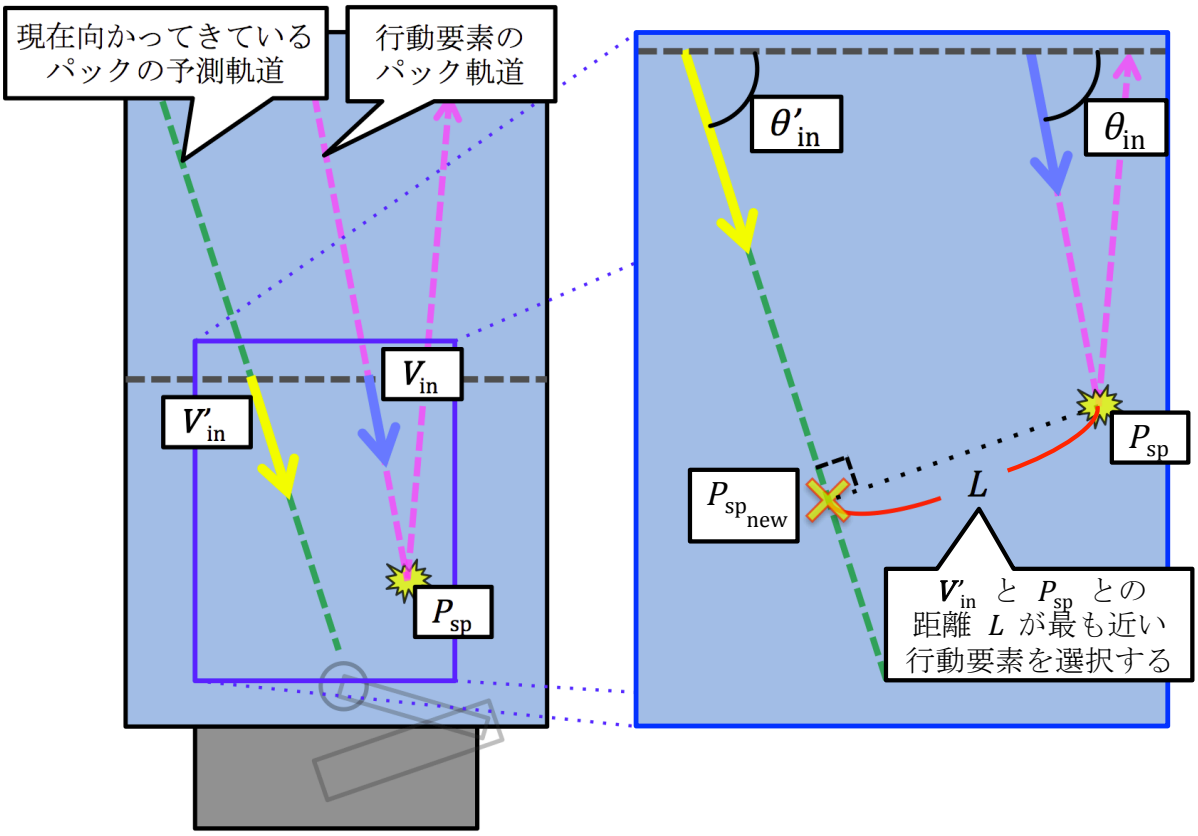

図 2.5: 再現する行動要素の選択

2.4.3 打ち返し処理

ここまでの処理で，各条件を満たす行動要素が 1 つも見つからなかった場合は，過去の学習データによらない単純な打ち返しを行う．これは，ある決まった Y 座標のライン上でパックの予測軌道に合わせてアームの手先を移動しておき，ある程度パックが手先に近づいたら，パックの進行方向と逆方向にむけて手先を移動して打ち返すというものである．

一方で，条件を満たす行動要素を見つけた場合，模倣動作として，その行動要素におけるマレットの動きを再生する（軌道の点をたどるようにアームの手先を動かす）．このとき，先に述べたように，マレットの動きに対して座標変換を施す．また，再生速度についても打ち返しができるように変更する．以下，それぞれの処理について順に述べる．

マレットの動きの座標変換

マレットの動きを座標変換する例を図 2.6 に示す．再生する行動要素の衝突時パック位置である P_{sp} を原点とし，速度ベクトル $\boldsymbol{V}_{\mathrm{in}}$ の方向を $-Y$ とする座標系 O_1 と，打ち返しの目標衝突点である $P_{sp_{\mathrm{new}}}$ を原点とし，パックの予測軌道の速度ベクトル $\boldsymbol{V}'_{\mathrm{in}}$ の方向を $-Y$ とする座標系 O_2 を求める．そして，再生するマレットの動きに対して， O_1 から O_2 への座標変換を施す．以上の手順により， $P_{\mathrm{sp_{new}}}$ を衝突予定の位置としたマレットの動きが生成できる．

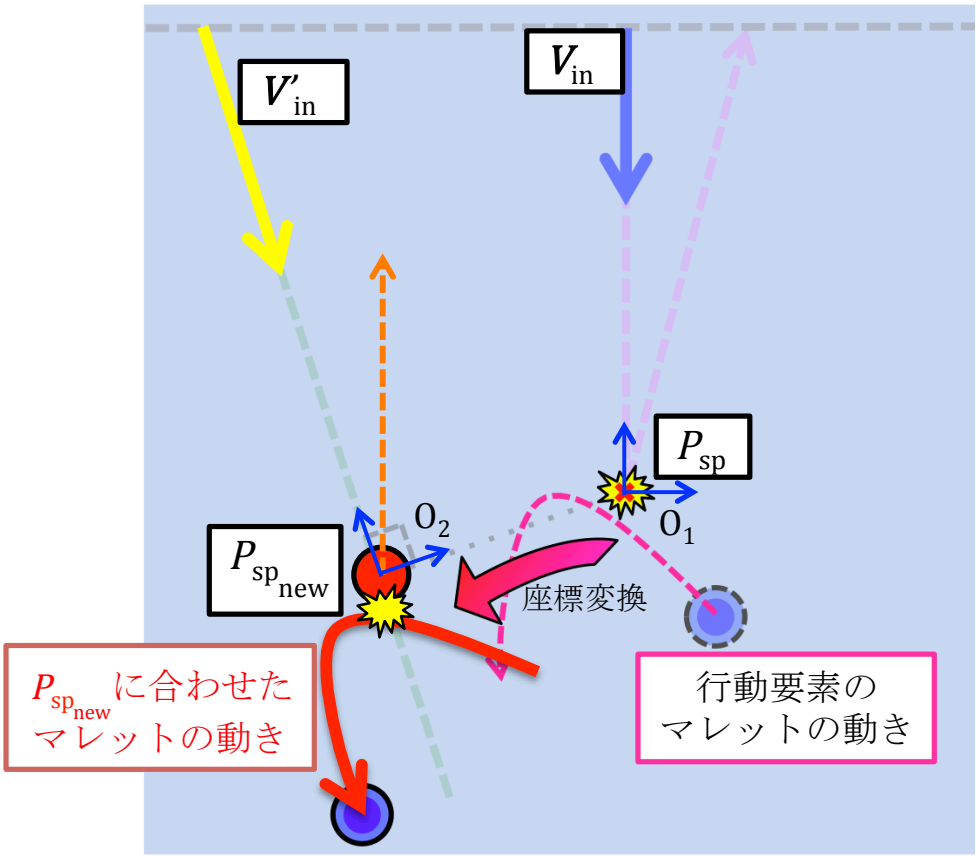

図 2.6: マレットの動きの座標変換

マレットの動きの再生速度調整

座標変換により新しいマレットの軌道を求めたが，このまま学習当時の再生速度で再生した場合，向かってくるパックの速度によっては打つタイミングがずれてしまう．例えば，遅いパックを打ち返した行動要素を，その学習当時よりも速いパックに対して適用する場合，マレットの動かし方が遅すぎるために打ち返しに遅れてしまう．そこで，以下のようにして再生速度を調整する．

現在 $\boldsymbol{V}'_{\mathrm{in}}$ でロボット側に向かってきているパックが $y = 0$ のセンターラインを通過するときの時刻を t_0 とし，$P_{\mathrm{sp_{new}}}$ に到達する（パックとマレットが衝突する）予想時刻を $t_{\mathrm{sp_{new}}}$ とする．2.3 にて述べたように，行動要素（3）のマレットの動きにおける経過時刻 t_{mallet_i} は，学習当時にパックとマレットが衝突した時刻 t_{sp} が 1 となるようにスケーリングされているので，現在時刻を t としたとき，再生時刻を

$$t_{\mathrm{play}} = \left(\frac{t - t_0}{t_{\mathrm{sp_{new}}} - t_0} - 1\right) \cdot \frac{\|\boldsymbol{V}'_{\mathrm{in}}\|}{\|\boldsymbol{V}_{\mathrm{in}}\|} + 1$$

としてマレットの動きを再現する．本式は，パック進入時の時刻を 0 ，パックとマレットの衝突時の時刻を 1 とした現在時刻について，時刻 1 の時点を中心に $\frac{\|\boldsymbol{V}'_{\mathrm{in}}\|}{\|\boldsymbol{V}_{\mathrm{in}}\|}$ 倍だけスケーリングするものである．これにより，パックが $P_{\mathrm{sp_{new}}}$ の位置に来た瞬間にマレットを衝突させ，打ち返すことができる．

2.5 まとめ

本章では，提案する模倣アルゴリズムの具体的な処理内容について述べた．模倣アルゴリズムは学習処理と再現処理に分かれており，学習処理にて対戦相手の打ち方を “行動要素” として記憶し，再現処理にて打ち方を再現している．再現する行動要素の選択は，ロボット側に向かってくるパックの予測軌道に近いものを選択する．ただし，ただ単純に行動要素のマレットの動きを再生するだけでは向かってくるパックを打ち返すことができない．そこで，行動要素の衝突時パック位置からパックの予測軌道上の位置に対して座標変換を行った後，パックの速度に合わせて再生速度を調整することで，学習当時の打ち返し動作の再現を可能とした．

第3章　エアホッケーロボットシステム

第2章で提案した模倣アルゴリズムを実機上に実装し実験するため，過去に当研究室にて開発されたエアホッケーロボットシステムをベースに，任意のアルゴリズムを実機上に実装することのできる汎用的なシステムとして再構築した．本システムは，パックを打ち返すための水平2関節ロボットアーム，パックの位置を検出するための高速カメラ，対戦相手のマレットの位置を検出するための測域センサから構成されており，それぞれを独立したコンポーネントとして制御している．

3.1　概要

当研究室では，過去にエアホッケーロボットシステムを開発し研究していたが，従来システムにおけるロボットアームにはいくつかの問題点があった．

- アームの制御方式がモータードライバ内のオンボードコントローラに依存している
- 対戦アルゴリズムの実装内にデバイス制御処理が組み込まれる

従来システムでアーム制御に使用していたモータードライバ（Fics-Atomsシリーズ，DYNAX）には，モーターを制御するためのコントローラが内蔵されており，外部装置からの指令によって任意の位置へアームを移動することができる．このコントローラは台形速度制御とS字速度制御の二種類をサポートしており，アームを滑らかかつ安全に指令位置まで移動することができる．しかし，アームの耐久性を重視したドライバであるため，アームの加減速度は比較的緩めの設定となってしまい，エアホッケーのように瞬時に反応

すべき状況には向いていなかった．また，対戦アルゴリズムを実装したソフトウェアを生成する際，アーム制御やパックの位置検出などの各種デバイス制御の実装も含め，一つの実行ファイルとして生成していた．この方式は，各デバイスを同一のプロセス単位で制御できるため，デバイス制御を高速に行うことができることに対して，デバイスの状況を常に意識してアルゴリズムを遂行しなければならないことや，複数のアルゴリズムの比較検討を行うような状況において，切り替えを行うごとにデバイスの再初期化をする必要があるなど，システム自体の汎用性が欠けてしまう．これらの問題点を解決するため，モータードライバの換装やソフトウェアの再構成を施した．

また，提案する模倣アルゴリズムは，学習時に対戦相手のマレットの位置を参照する必要があるが，従来システムには対戦相手が持つマレットの位置を検出する機能がなかった．そこで，マレット検出のためのデバイス追加を行った．以下，構築したシステムについて詳しく述べる．

3.2 システム構成

エアホッケーロボットシステムの全体図を図 3.1 に示す．ホッケー台側面には，手先に専用のマレットを装着した水平 2 関節ロボットアームが固定されている．ロボット側から見てセンターライン横の左側には，対戦相手のマレットの位置を検出するための測域センサが固定されている．ホッケー台上部の天井には，パックの位置を検出するための高速カメラが固定されている．なお，高速カメラはシャッタースピードの関係でキャプチャ後の画像が暗いため，ホッケー台の側面に電球型蛍光灯（100[V]/36[W]）を計 10 灯配置し，明るさを補完している．これら計 3 つのデバイスを一つの計算機で制御している．本システムの全体的な接続図を図 3.2 に示す．図は計算機と各デバイスの接続状況を表しており，計算機とロボットアーム，高速カメラ，測域センサの各デバイス間が接続されていることを示している．

3.2.1 各種寸法および座標系の定義

本システムの各種寸法や各デバイスの設置位置，およびフィールド座標系の定義を図 3.3 に示す．ロボットアームの肩軸は $(x, y) = (0, -1355.5)[\mathrm{mm}]$ の位置に設置してある．また，測域センサはセンターライン横の $(x, y) = (-750, 0)[\mathrm{mm}]$ の位置に設置してある．なお，フィールド座標系は第 2 章で述べたように，ホッケー台中央を原点とし，ロボット側から見て右方向を $+X$，奥方向を $+Y$ とする．

図 3.1: エアホッケーロボットシステムの構成

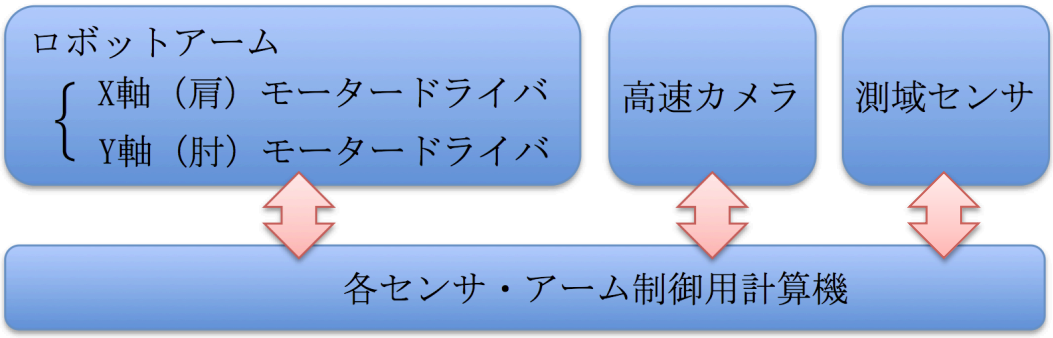

図 3.2: デバイス間の接続関係

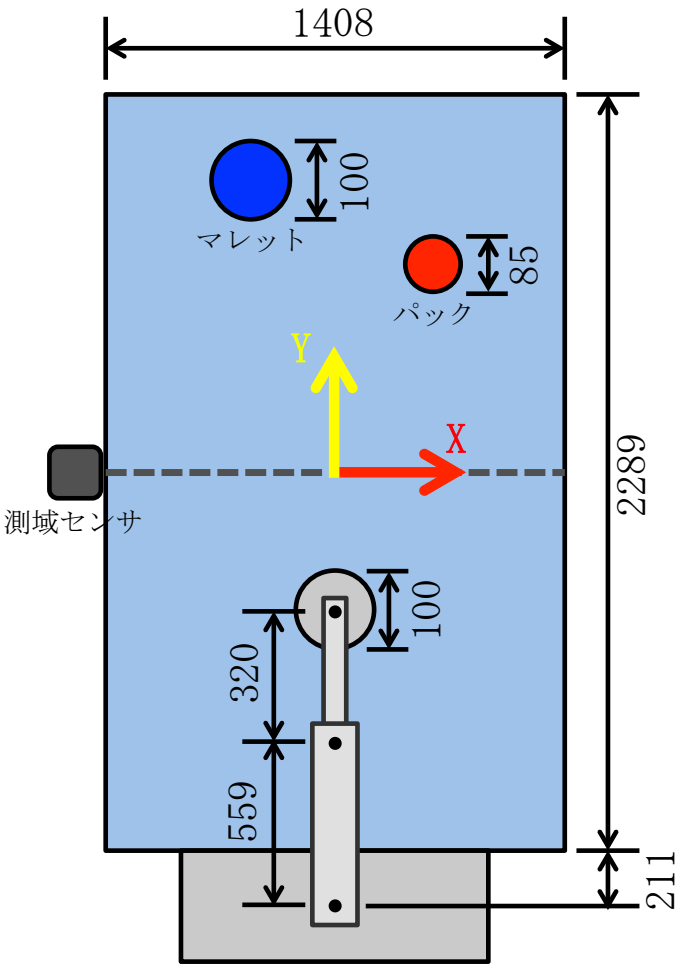

図 3.3: エアホッケー台の寸法 (単位:[mm])

3.2.2 ロボットアーム

ロボットアームは水平 2 関節で構成されている (図 3.4)．肩軸は 400[W] のモーター (MSM041A1，Panasonic) を減速比 1:25 にて，肘軸は 100[W] のモーター (MSM011A1，Panasonic) を減速比 1:15 にて駆動させており，パックへの高速な追従や打ち返しを実現している．また，各モーターを制御するドライバには，それぞれ MSD041A1XX (Panasonic)，MSD011A1XX (Panasonic) を用いており (図 3.5)，計算機からの電圧指令によりモーターの回転数を制御できるようになっている．

ロボットアームの手先には，専用のマレットを装着している．マレットはスポンジとスプリングを使い，台上に押しつけるように装着しており，ホッケー台の歪みを吸収しパックを打ち返せるようになっている．

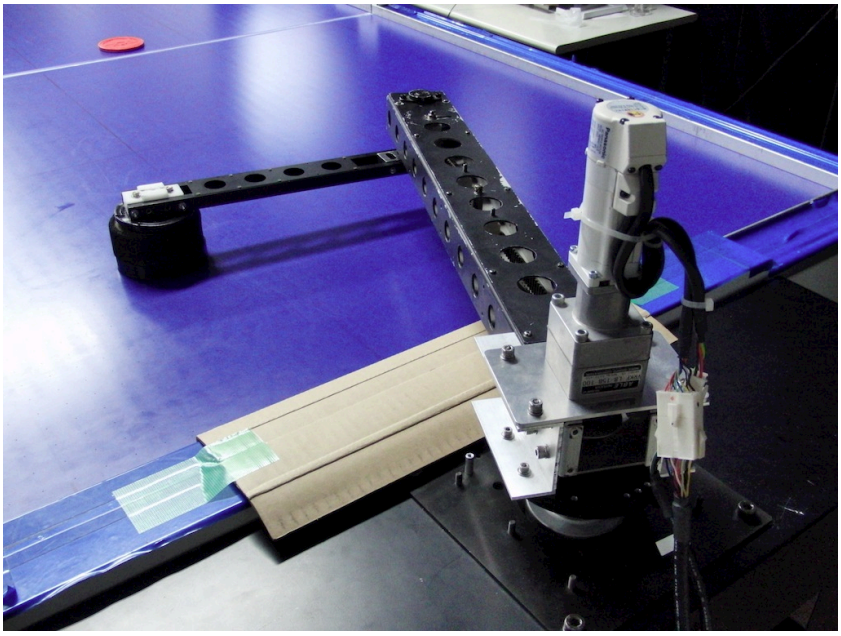

図 3.4: ロボットアーム

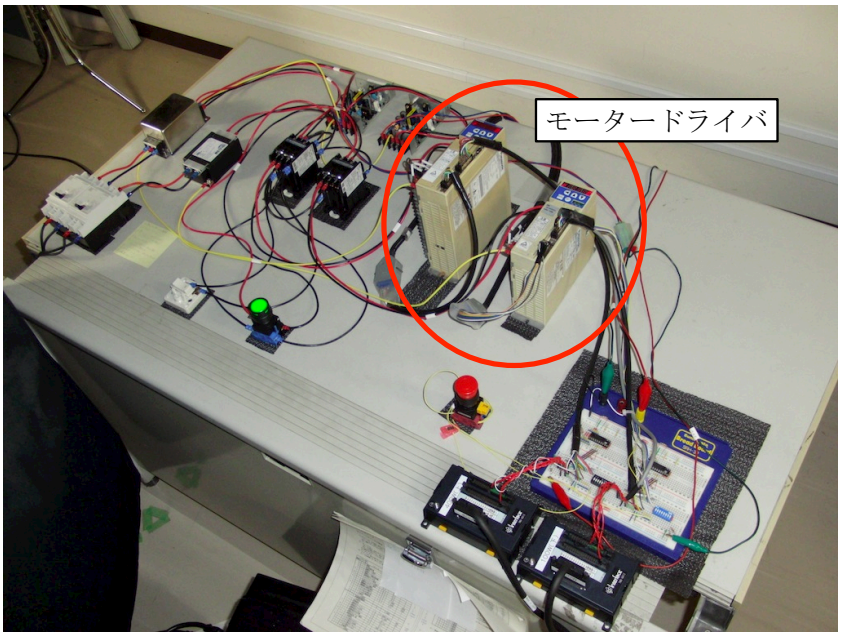

図 3.5: 電源・モータードライバーの配線

3.2.3 高速カメラ

パックの位置を検出するため，ホッケー台上部に高速カメラ (MV-D640C-66-CL-10，Photonfocus) を設置している (図 3.6)．カメラのフレームレートは 120[fps] に設定しており，パックの位置の高速な検出を可能としている．なお，カメラのキャプチャ画像はシャッタースピードの関係で暗く映り，特にホッケー台の四隅においては，パックの検出が困難なほど輝度が低い．そこで，ホッケー台の長辺横に 100[V]/36[W] の電球型蛍光灯を計 10 個配置し，いかなる場所でもパックを検出できるように明るさを調整している．

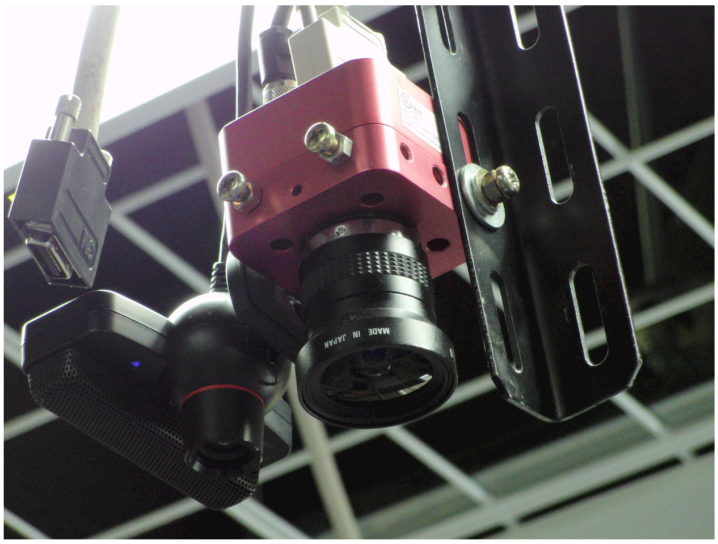

図 3.6: ホッケー台上部の天井に設置した高速カメラ

3.2.4 測域センサ

対戦相手が持つマレットの位置を検出するため，ロボット側から見て台の中央左側に測域センサ (UTM-30LX，北陽電機) を設置している (図 3.7)．測域センサの走査時間は 25[msec] であり，マレットの位置を 1 秒間に 40 回検出している．なお，マレットの位置を確実に検出できるよう，マレットの周囲に厚紙を巻く措置を施している (図 3.8)．

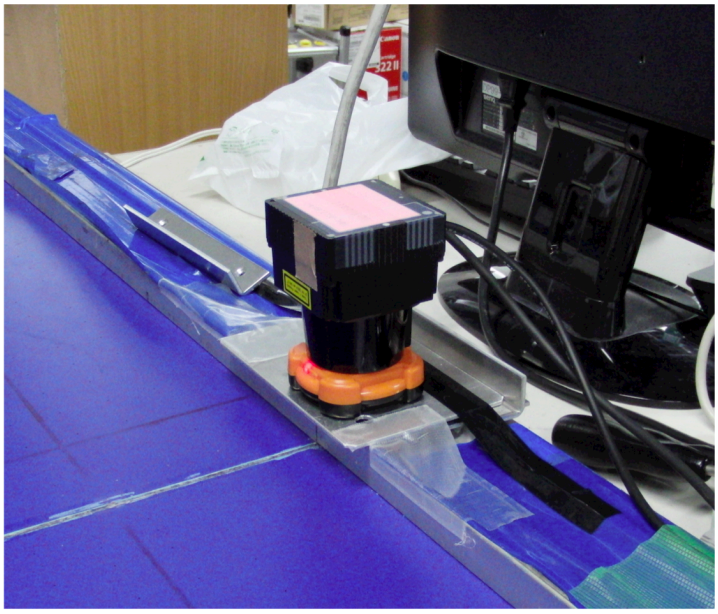

図 3.7: ホッケー台横に設置した測域センサ

図 3.8: マレット

3.2.5　システム全体を管理・制御する計算機

ロボットアームや各種センサの制御，対戦アルゴリズムを動作させる環境として，デスクトップ PC を 1 台配置している．本計算機の構成を表 3.1 に示す．

ロボットアームを制御するため，アナログ入出力ボード（PEX-361316，Interface）を 2 枚搭載しており，それぞれ 100[W]，400[W] のモータードライバ用となっている．本ボードは，モータードライバへモーターの回転数を指令するための電圧を出力することができるほか，モーターの回転量（エンコーダカウント）を計算機側に取り込める機能を有している．また，高速カメラの画像をキャプチャしパック検出を行うため，CameraLink 規格に対応したキャプチャボード（PEX-530421，Interface）を 1 枚搭載している．また，本計算機上に導入した基本ソフトウェアを表 3.2 に，インストールした主要なライブラリやドライバを表 3.3 に示す．

表 3.1: 計算機の構成

種別	型番	備考
CPU	Core i7-2600K @3.40GHz	
主記憶	12GB	
GPU	nVidia Quadro FX 1800	
高速カメラ制御ボード	GPG-5300 (Interface)	
モータードライバ制御ボード	PEX-361316 (Interface) 2枚	A/D, D/A変換ボード

表 3.2: 基本ソフトウェア

基本ソフト	Linux
ディストリビューション	Vine Linux 5.2
Linuxカーネル	2.6.27-74v15

表 3.3: 主なインストール済みライブラリ・ドライバ

種別	バージョン	備考
OpenCV	2.3.1	
Boost	1.50.0	
SCIP2ドライバ (libscip2awd)	0.10.7	
GPG-5300ドライバ	1.40-05	
PEX-361316ドライバ	5.01-37 4.12-35 4.20-18	アナログ入力用ドライバ アナログ出力用ドライバ パルスカウンタ用ドライバ

3.3 OpenRTM-aistによる実装

本システムでは，3.2で述べたハードウェアを動作させるため，ロボットシステムをコンポーネント指向で開発するためのソフトウェアプラットフォームであるOpenRTM-aistを使用している [19]．OpenRTM-aistは，ロボットシステムを作成する際に，機能要素ごと

に RTC(Robotic Technology Component) と呼ばれるプログラムを作成し，それらを自由につなぎ合わせることでシステム構築を行うことができる．また，RTC は C++，Python 等の複数の言語に対応しているため，開発者が使いやすい言語でプログラムを作成することができ，既に開発されたコンポーネントを再利用することで，最低限のプログラミングでシステム構築を行うことが可能である．当研究室では，OpenRTM-aist を用いて様々なコンポーネントを開発し，研究に生かしている [20] [21] [22].

本システムは，表 3.4 に示す RTC 群で構成されている．AHCommonData は，ホッケー台やパック，マレットの寸法のように，RTC 間で共有するパラメータ値を各 RTC に提供する RTC である．HockeyArmController は，ロボットアームを制御する RTC である．PuckTracker，MalletTracker は，それぞれパックの位置検出，マレットの位置検出をする RTC である．これら 4 つの RTC を使い，エアホッケーロボットの対戦アルゴリズムを開発することができる．対戦アルゴリズムを実装した場合の RTC 間の接続図を図 3.9 に示す．図は，前述の RTC 群を使い対戦アルゴリズム RTC の AHAttack を実装した例である．以下，これら RTC の入出力や機能について詳しく述べる．なお，以降で示す全 RTC の入出力（サービスポートを除く）はフィールド座標系における位置 (x, y) を表しており，**TimedDoubleSeq 型**，**要素数：2**，**単位：[mm]** である．

表 3.4: エアホッケーロボットシステムにおけるコンポーネント群

RTC名	機能
AHCommonData	エアホッケーの共通データを管理・提供する
HockeyArmController	ロボットアームを制御する
PuckTracker	パックの位置を検出する
MalletTracker	マレットの位置を検出する

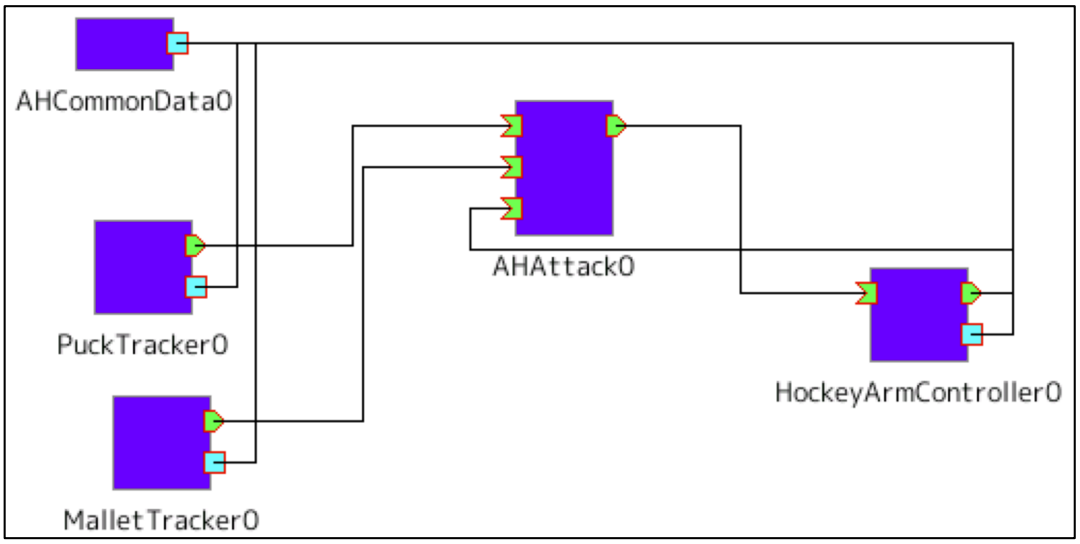

図 3.9: コンポーネント間の接続例（対戦アルゴリズムが AHAttack の場合）

3.3.1 AHCommonData(共有データ管理 RTC)

AHCommonData は，ホッケー台の寸法やパック・マレットの直径など，各 RTC 間で共有すべきデータをサービスポートにより提供する RTC である．本 RTC の構成を図 3.10 に示す．本 RTC はサービスポート出力のみを有しており，このポートより他の RTC から寸法等のデータを要求されると，本 RTC はサービスポートを介し，それに対応するデータを検索し返却する．

本 RTC のコンフィギュレーション変数の内容を表 3.5 に示す．変数 common_data には，本 RTC がサービスポートで提供する共有データのパスを設定する．なお，本 RTC に読み込ませる共有データは，図 3.11 のように，Windows 環境でよく用いられる INI ファイルの形式で記述する．

本 RTC がサービスポートにて提供する機能の定義（IDL ファイル）を図 3.12 に示す．他の RTC から本 RTC に対して共有データを要求し受け取るには，図に示す getData() オペレーションを用いる．たとえば，図 3.11 におけるホッケー台の幅（HockeyParam セクションの TableWidth パラメータ）1408.0 を取得したい場合，本サービスポートに対して，

getData(”**HockeyParam.TableWidth**”, **data**)

と要求する．ここで，data は文字列型の変数である（C++の場合，CORBA::String_var に相当する）．このように要求すると，引数 data に文字列型としてデータ ”1408.0” が格納され，getData() は戻り値として true を返却する．なお，データが見つからなかった場合，引数 data には空文字列 ”” が格納され，getData() は戻り値として false を返却する．

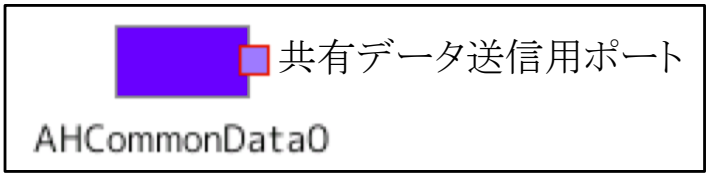

図 3.10: AHCommonData

表 3.5: AHCommonData のコンフィギュレーション変数

名称	型	デフォルト値	説明
common_data	string	../common/AHData.ini	共有データINIファイルへのパス

```
[HockeyParam]
; measurements of air hockey objects
TableWidth = 1408.0
TableDepth = 2290.0
PuckD      =   85.0
; hockey arm
ArmLenX2Y  = 557.0
ArmLenY2M  = 322.0
ArmOffsetY = 203.0
RMalletD   = 100.0
; player
PMalletD = 100.0
; Top-URG
URGPosX  = -724.0
URGPosY  =    0.0
URGRotTh =    0.02094395102
```

図 3.11: 共有データの記述例（デフォルトの AHData.ini）

```
module AHCommon
{
    interface AHCommonDataService {
        boolean getData(in string dataname, out string data);
    };
};
```

図 3.12: サービスポートで提供する機能の定義（IDL ファイル）

3.3.2 HockeyArmController(ロボットアーム制御 RTC)

HockeyArmController は，指定された位置までロボットアームの手先を移動する RTC である．本 RTC の構成を図 3.13 に示す．本 RTC は，ロボットアームの移動先手先位置を受け取るためのポート（図左），現在のロボットアームの手先位置を出力するためのポート（図右上），および AHCommonData に共有データを要求するためのサービスポート（図右下）を有する．なお，手先位置はフィールド座標系ベースである．

本 RTC のコンフィギュレーション変数の内容を表 3.6 に示す．変数 arm_offset は，ロボットアームの肩軸がロボット側の壁からどの程度離れているかを単位 [mm] で表している．変数 arm_len_1 および arm_len_2 は，それぞれ “肩-肘”，“肘-手先” におけるアームの長さ [mm] を表している．変数 initial_enc_x および initial_enc_y は，アームが対戦相手側にまっすぐ伸びた状態におけるエンコーダの値を表しており，それぞれ肩軸のモーター，肘軸のモーターの値である．変数 target_arm_rpm_x および target_arm_rpm_y は，目標とするアームの回転数 [rpm] を表しており，それぞれ肩軸，肘軸の値である．変数 accel_arm_rpm_x および accel_arm_rpm_y は，アーム回転数の加速度 [rpm/sec] を表しており，それぞれ肩軸，肘軸の値である．変数 decel_arm_rpm_x および decel_arm_rpm_y は，アーム回転数の減速度 [rpm/sec] を表しており，それぞれ肩軸，肘軸の値である．

本 RTC に対して移動先のアーム手先位置 (x, y) が入力されると，新たにその位置を目標とし，ロボットアームの手先を移動し始める．なお，本 RTC が以前に入力された手先

位置に向けてアームを移動している途中でも，再度手先位置を入力した場合，すぐにその新しい手先位置に向けて移動を開始する．また，本 RTC は動作中，ロボットアームの手先位置 (x, y) を常に出力し続ける．対戦アルゴリズム RTC は，この出力をフィードバックとして利用することができる．なお，手先位置はモーターのエンコーダの値をもとに順運動学により導出している．

アームの制御方式には台形制御を採用しており，アームに極端な負荷がかからないように配慮している．アームの回転数や加減速のパラメータは，前述したコンフィギュレーション変数により設定可能である．なお，パラメータの初期値はアームに極端な負荷がかからない範囲で最大限の加減速度および回転数を設定した．

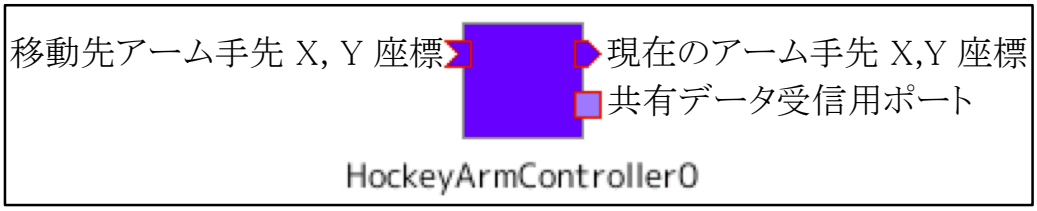

図 3.13: HockeyArmController

表 3.6: HockeyArmController のコンフィギュレーション変数

名称	型	デフォルト値	説明
arm_offset	double	211.0125	ホッケー台のロボット側壁からのアーム肩関節位置オフセット
arm_len_1	double	558.5	アームの長さ（肩-肘）
arm_len_2	double	319.45	アームの長さ（肘-手先）
initial_enc_x	int	225	肩モータのエンコーダ初期値
initial_enc_y	int	1000	肘モータのエンコーダ初期値
accel_arm_rpm_x	double	500	肩モータの加速度[rpm/sec]
accel_arm_rpm_y	double	750	肘モータの加速度[rpm/sec]
decel_arm_rpm_x	double	500	肩モータの減速度[rpm/sec]
decel_arm_rpm_y	double	750	肘モータの減速度[rpm/sec]

3.3.3 PuckTracker(パックの位置検出 RTC)

PuckTracker は，高速カメラによりキャプチャした画像中に存在するパックの位置を検出し，そのフィールド上の座標 (x, y) を計算し出力する RTC である．本 RTC の構成を図 3.14 に示す．本 RTC は，フィールド座標系におけるパックの位置を出力するためのポート（図右上），および AHCommonData に共有データを要求するためのサービスポート（図右下）を有する．

本 RTC のコンフィギュレーション変数の内容を表 3.7 に示す．変数 calibration_data は，カメラの外部・内部パラメータが記述されたカメラキャリブレーションファイルのパスを表している．変数 n_particle は，本 RTC 内でパック検出のために用いているパーティクルフィルタ [23] におけるパーティクルの数を表している．変数 p_noise および v_noise は，それぞれ各パーティクルの位置成分，速度成分のノイズを表しており，パーティクルの分散の程度を決定できる．変数 sigma は，ある画素における色値がパックの色を表しているかどうかを決めるパラメータを表している．変数 min ○/max ○は，パックの色の範囲を決定することができ，R・G・B は RGB 表色系における判定値，H・S・V は HSV 表色系における判定値を表している．変数 HSVorRGB は，パックの色判定に使用する表色系を表している．

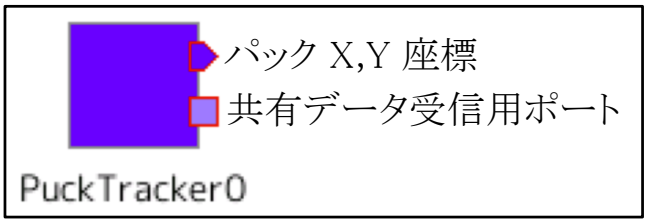

図 3.14: PuckTracker

表 3.7: PuckTracker のコンフィギュレーション変数

名称	型	デフォルト値	説明
calibration_data	string	./camera.xml	カメラのキャリブレーションパラメータ
n_particle	int	400	パーティクルの数
p_noise	int	30	パーティクルの位置成分ノイズ
v_noise	int	5	パーティクルの速度成分ノイズ
sigma	float	50	ユークリッド距離の分散
minH / maxH	float	340 / 360	最小/最大のH値
minS / maxS	float	90 / 255	最小/最大のS値
minV / maxV	float	0 / 255	最小/最大のV値
minR / maxR	float	0 / 255	最小/最大のR値
minG / maxG	float	0 / 255	最小/最大のG値
minB / maxB	float	0 / 255	最小/最大のB値
HSVorRGB	string	HSV	HSVとRGBのどちらの判定方法を使用するか "HSV" : HSVによる判定 "RGB" : RGBによる判定

3.3.4 MalletTracker(マレット位置検出 RTC)

MalletTracker は，測域センサにより取得したフィールド上のスキャン点群からマレットの位置を検出し，そのフィールド上の座標 (x, y) を計算し出力する RTC である．本 RTC の構成を図 3.15 に示す．本 RTC は，フィールド座標系におけるマレットの位置を出力するためのポート（図右上），および AHCommonData に共有データを要求するためのサービスポート（図右下）を有する．

本 RTC のコンフィギュレーション変数の内容を表 3.8 に示す．変数 calibration_data は，マレット位置を補正するためのキャリブレーションパラメータファイルへのパスを表している．変数 n_particle は，本 RTC 内でマレット検出のために用いているパーティクルフィルタにおけるパーティクルの数を表している．変数 urg_pos_x，urg_pos_y，urg_angle は，エアホッケー台上に設置してある測域センサのフィールド座標系における位置および角度を表している．変数 sp_scale_x および sp_scale_y は，測域センサより取得したスキャン点

群の X 方向，Y 方向のスケーリング（倍率）を表している．

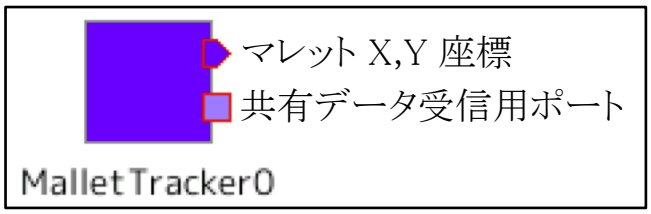

図 3.15: MalletTracker

表 3.8: MalletTracker のコンフィギュレーション変数

名称	型	デフォルト値	説明
calibration_data	string	./MalletCalibData.dat	測域センサのキャリブレーションパラメータ
n_particle	int	200	パーティクルの数
sigma	double	30	尤度計算の分散
urg_pos_x	double	−750	測域センサのX座標
urg_pos_y	double	0	測域センサのY座標
urg_angle	double	0	測域センサの回転角度(rad)
sp_scale_x	double	1.055	スキャン点のX方向スケール
sp_scale_y	double	1.028	スキャン点のY方向スケール

3.3.5　対戦アルゴリズム RTC の基本仕様

これまでに述べた RTC 群を用いて，対戦アルゴリズムを一つの RTC として作成することができる．図 3.9 に示したように対戦アルゴリズム（この場合は AHAttack）を繋げることで，エアホッケーロボットシステムを動作させることができる．対戦アルゴリズムの入出力の作成例を図 3.16 に示す．基本的には，対戦アルゴリズム RTC はパックの位置入力（図左上），マレットの位置入力（図左中），アームのフィードバック位置入力（図左下），およびアルゴリズムが指令するアームの手先位置の出力（図右上）の計 4 つの入出力で構成する．そしてそれぞれのポートに対して，PuckTracker の出力，MalletTracker の出力，HockeyArmController の出力，および HockeyArmController の入力へと接続する．このように対戦アルゴリズム RTC を作成することにより，本システムを用いたアルゴリズムの実機上の実験が行える．

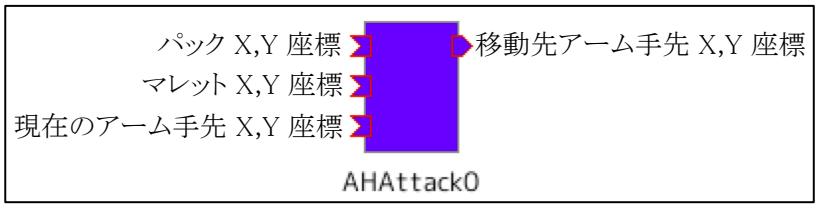

図 3.16: AHAttack

3.4　まとめ

本章では，本研究における模倣アルゴリズムを実機上に実装し実験するために構築したエアホッケーロボットシステムについて述べた．本システムはロボットアーム制御，パックの位置検出，マレットの位置検出を OpenRTM-aist を用いて RTC として提供しており，アルゴリズムの開発・実験を容易に行えるようになっている．なお，本システムは第 4 章で述べる実験で使用する．

第4章　模倣アルゴリズムの実験

第3章では，任意のアルゴリズムを実装可能なエアホッケーロボットシステムについて述べた．本章では，模倣アルゴリズムを本システム上に実装し，本アルゴリズムが対戦相手の打ち方を再現できるのかどうかを実験し検証する．4.1では，3種類の特徴が大きく異なる打ち方を個別にオフライン学習させ、ロボットが打ち方を再現する際に，打ち方の特徴を捉えた打ち返しが可能であるかを検証する．また，4.2では，行動要素を獲得するにつれて，再現する行動要素の検索時間がどのように変化するのかを調査し，人間との打ち合いの中で打ち返しに成功しやすい打ち返し回数を実験で明らかにする．

4.1　オフライン学習における打ち方の再現実験

模倣アルゴリズムの目的の一つとして，対戦相手の打ち方の特徴を再現し打ち返すことが挙げられる．本実験では，3種類の特徴が大きく異なる打ち方を学習させ，打ち返しの際に，それぞれの打ち方の特徴を再現することができるかどうかを検証する．

4.1.1　実験方法

以下に示すような，特徴が大きく異なる3種類の打ち方について検証する．

- カウンター打ち
- カット打ち
- 止め打ち

カウンター打ちとは，向かってくるパックに対して正面衝突させるようにマレットを動かしパックを打ち返す打ち方である（図 4.1）．カット打ちとは，向かってくるパックの進行方向に対し，斜め方向に向けてマレットを動かし，パックを切るようにしてまっすぐ打ち返す打ち方である（図 4.2）．止め打ちとは，向かってくるパックと同じ方向にマレットを動かして当てることでパックを一旦停止させ，その後，相手側に向けて打ち返す打ち方である（図 4.3）．

実験では，これら 3 種類の打ち方を別々にロボットに学習させておき，人間との打ち合いを行う際にそれらの打ち方を再現させる．ロボットがそれぞれの打ち方の特徴を再現し打ち返すことができれば，本アルゴリズムにより対戦相手の打ち方の特徴を捉えて打ち返すことができるといえる．なお，行動要素の獲得数はいずれも 30 個程度とし，行動要素の検索の際に行動要素を選択対象とするパックの進入速度差 v_{diffmax} および角度差 $\theta_{\mathrm{diffmax}}$ は，それぞれ 2000 [mm/sec] および 20 [deg] とした．

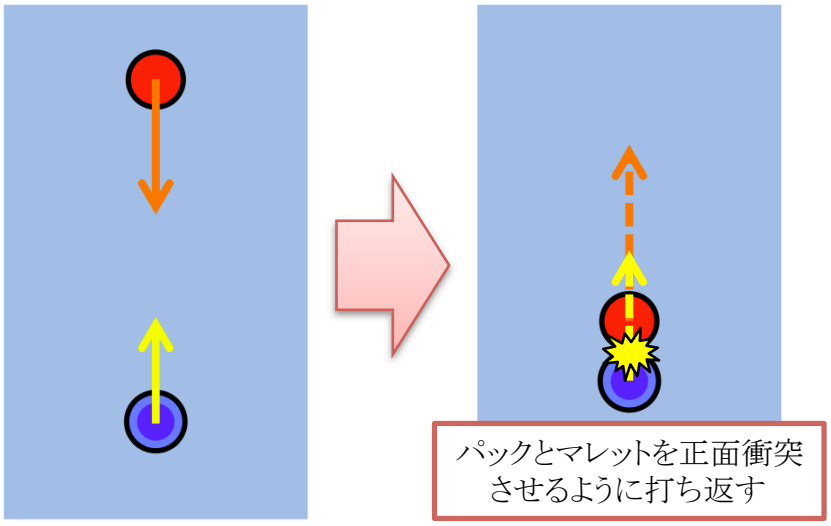

図 4.1: カウンター打ち

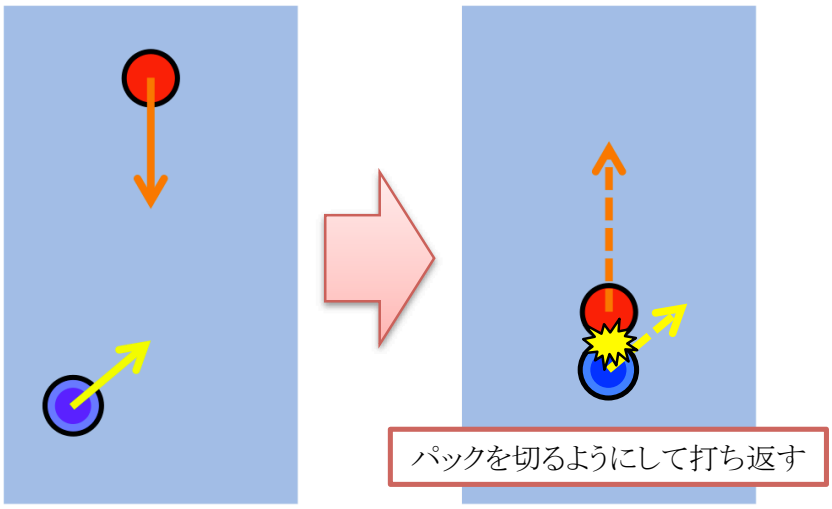

図 4.2: カット打ち

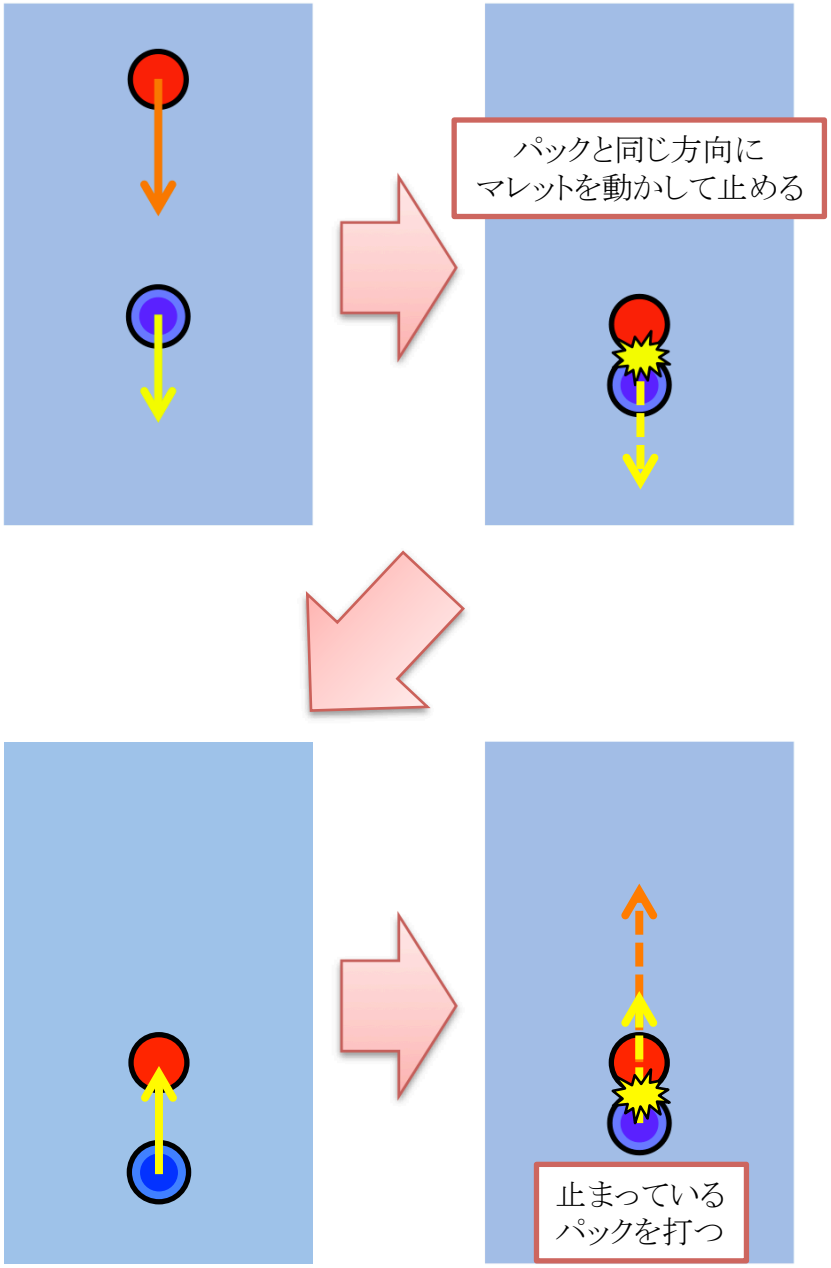

図 4.3: 止め打ち

4.1.2 実験結果

カウンター打ち

カウンター打ちの行動要素を生成する際に用いたパックとマレットの軌跡の一例を図 4.4 に示す．図はロボットに教示する人間側の陣地を表している．紫の枠は左右・下の線が人間側の陣地の壁を表しており，上の横線はセンターラインを表している．赤い点群はパックの軌跡であり，赤丸で囲まれている点は軌跡の開始点を示している．青い点群は人間が持つマレットの軌跡であり，青丸で囲まれている点が軌跡の開始点となっている．黒丸で囲まれている黒点はパックとマレットが衝突した瞬間のパックの位置（行動要素 (2)）を示している．これらのデータをもとに行動要素の各データを生成した．図を見ると，パックの進行方向に対してマレットを正面衝突させるように打ち返していることが分かり，カウンター打ちを行っている（教示している）ことが分かる．

生成した行動要素群をもとに再現処理を行わせた際の打ち返しの様子を図 4.5 に示す．図はロボットの後ろ側より 30[fps] でビデオ撮影した動画を，2 フレームごとに切り出したものである．図より，ロボットに教示したカウンター打ちの特徴を捉えて再現できていることが分かる．なお，ロボットによる再現動作は 50 回程度行ったが，打ち損じは一切発生しなかった．

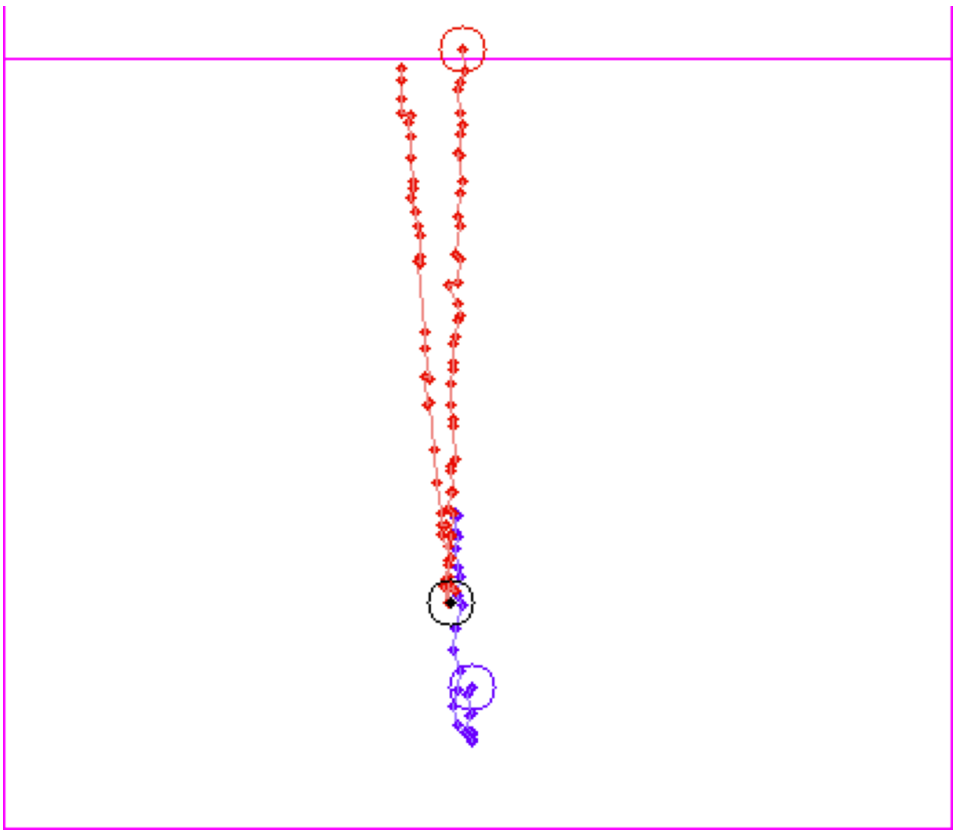

図 4.4: カウンター打ちの学習元データの一例

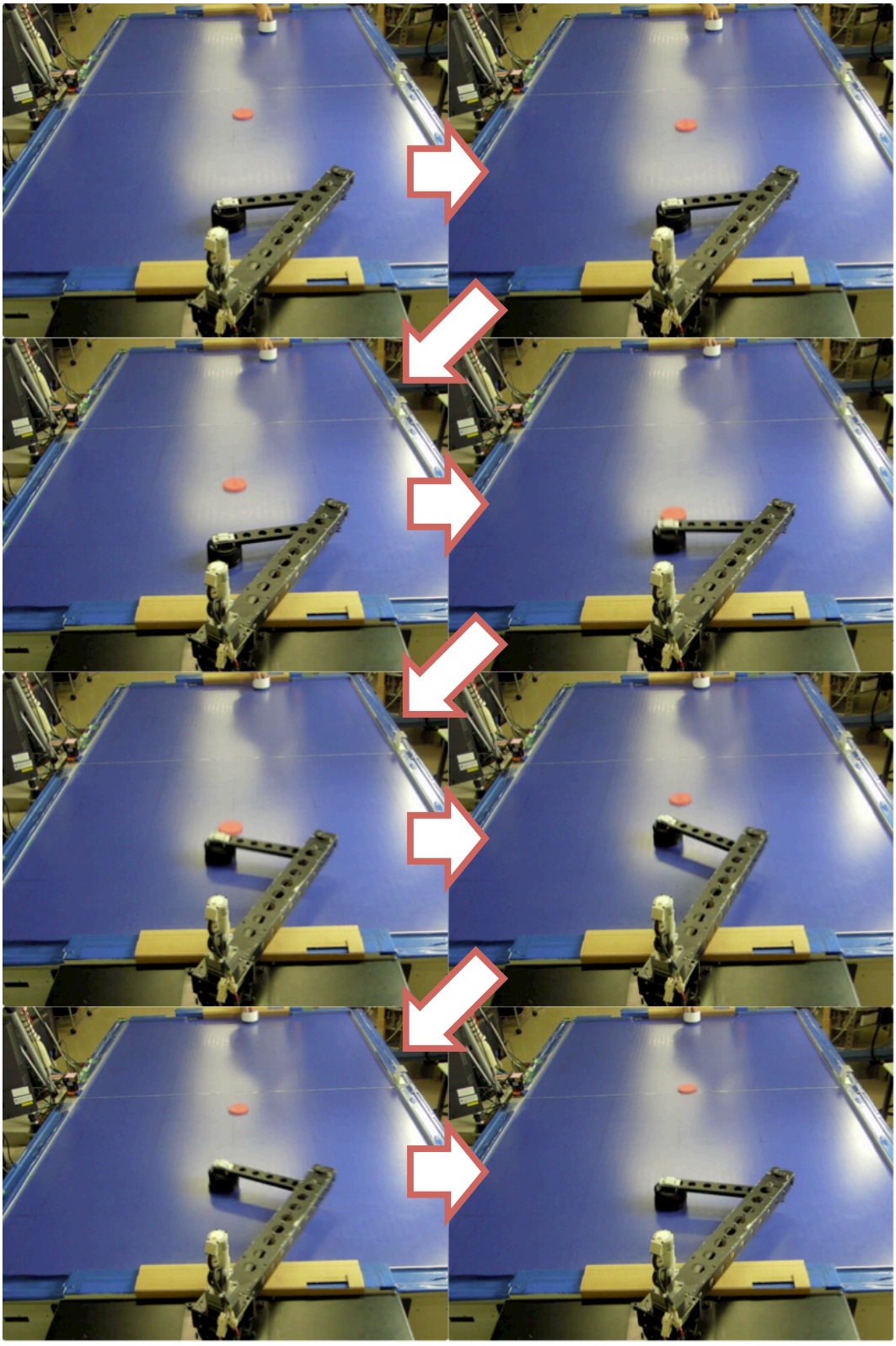

図 4.5: カウンター打ちの様子

カット打ち

カット打ちの行動要素を生成する際に用いたパックとマレットの軌跡の一例を図 4.6 に示す．図の見方は 4.1.2 と同様である．図を見ると，向かってくるパックを斜めに切るようにして打ち返していることが分かり，カット打ちを行っている（教示している）ことが分かる．

生成した行動要素群をもとに再現処理を行わせた際の打ち返しの様子を図 4.7 に示す．図は 4.1.2 と同様に撮影した動画を，3 フレームごとに切り出したものである．図より，ロボットに教示したカット打ちの特徴を捉えて再現できていることが分かる．なお，ロボットによる再現動作は 50 回程度行ったが，打ち損じは一切発生しなかった．

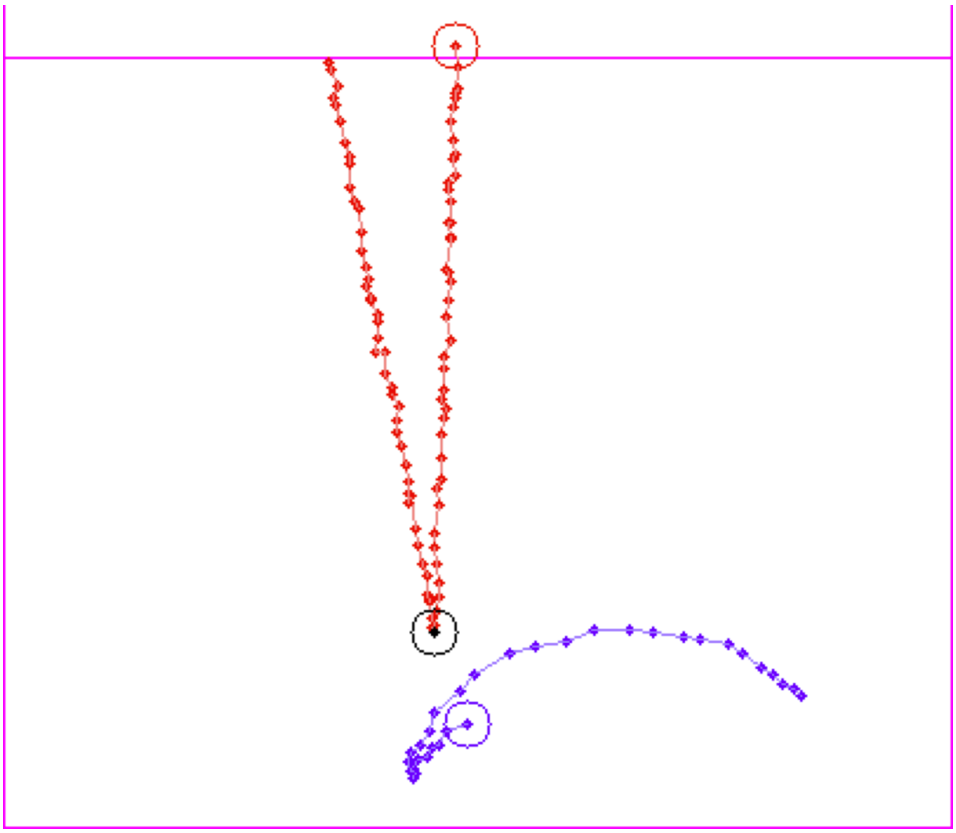

図 4.6: カット打ちの学習元データの一例

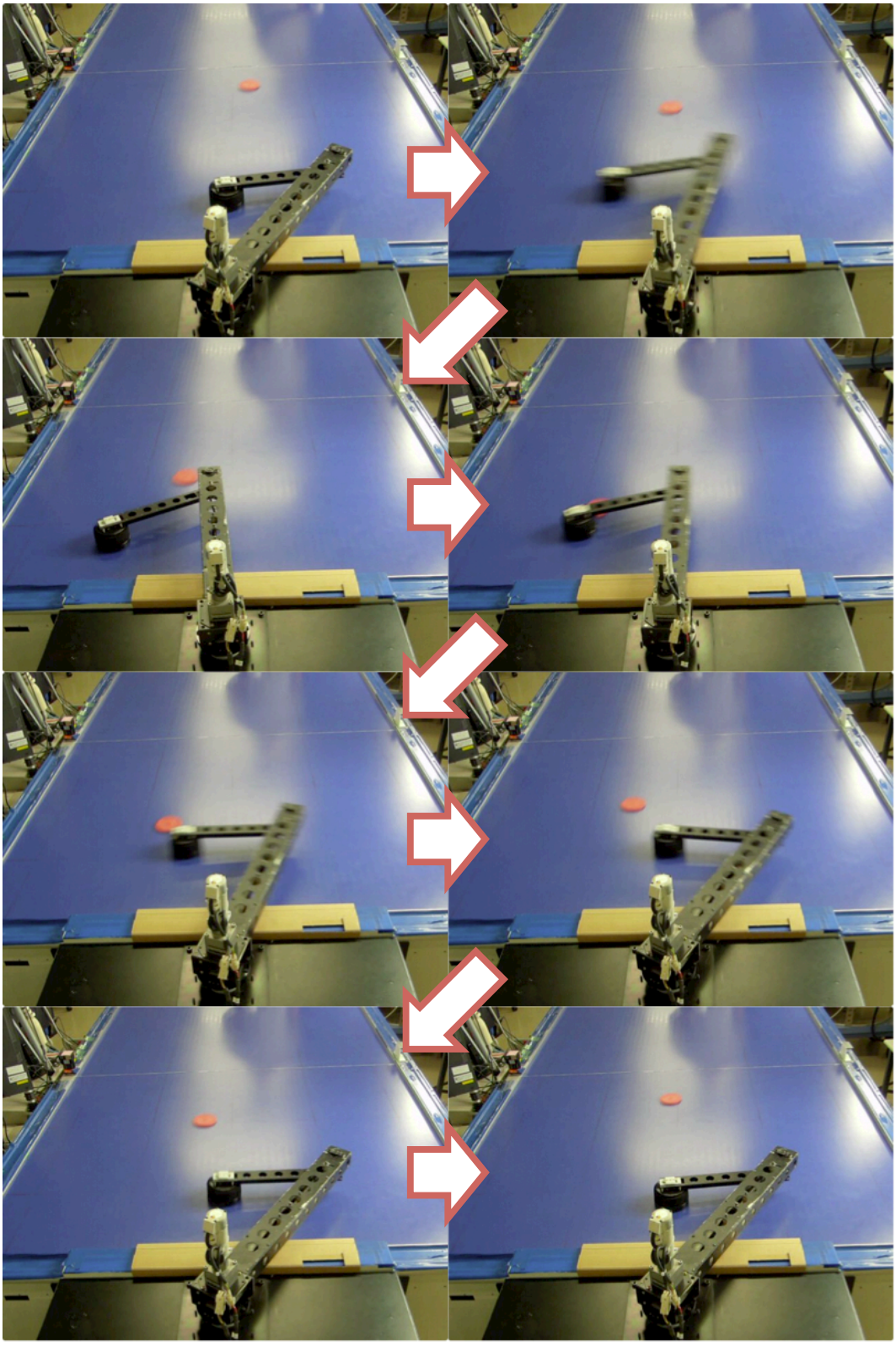

図 4.7: カット打ちの様子

止め打ち

止め打ちの行動要素を生成する際に用いたパックとマレットの軌跡の一例を図 4.8 に示す．図の見方は 4.1.2 と同様である．図を見ると，向かってくるパックを一旦止めた後，パックを打ち返していることが分かり，止め打ちを行っている（教示している）ことが分かる．

生成した行動要素群をもとに再現処理を行わせた際の打ち返しの様子を図 4.9 および図 4.10 に示す．図は 4.1.2 と同様に撮影した動画を，3 フレームごとに切り出したものである．図 4.9 は，進入してきたパックを静止させるまでのロボットの動きを表している．また，図 4.10 は，静止させたパックを相手側へ打ち返すまでの動きを表している．図 4.9 では，向かってきたパックを静止させることができていることが分かる．また，図 4.10 では，静止させたパックを相手側へ打ち返すことができていることが分かる．

一方で，パックを静止させることができなかった例を図 4.11 および図 4.12 に示す．図は 4.1.2 と同様に撮影した動画を，3 フレームごとに切り出したものである．図 4.11 を見ると，パックを静止させようとロボットがパックの進行方向と同方向に手先を引くが，パックを止めることができずに反射させてしまっていることが分かる．そして図 4.12 では，静止しているパックを打ち返す動作をするが，パックの位置は学習時のパックの静止位置と異なる場所に移動しているため，ロボットは打ち返しに失敗してしまう．これは本アルゴリズムにおいて，打ち返しの際にパックとマレットが衝突する回数が 1 回のみであると限定しているためである．そのため，2 回以上当てて打ち返す打ち方については動作が不安定になる．なお，止め打ちに成功する確率は，6 割程度であることが実験により分かった．

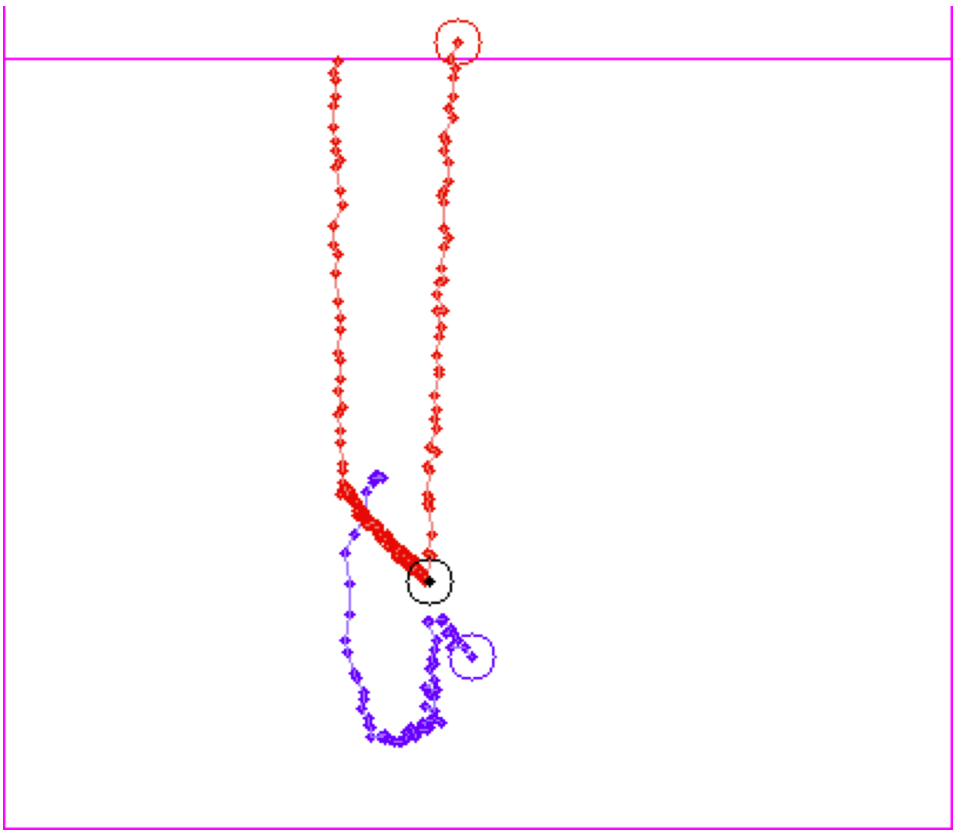

図 4.8: 止め打ちの学習元データの一例

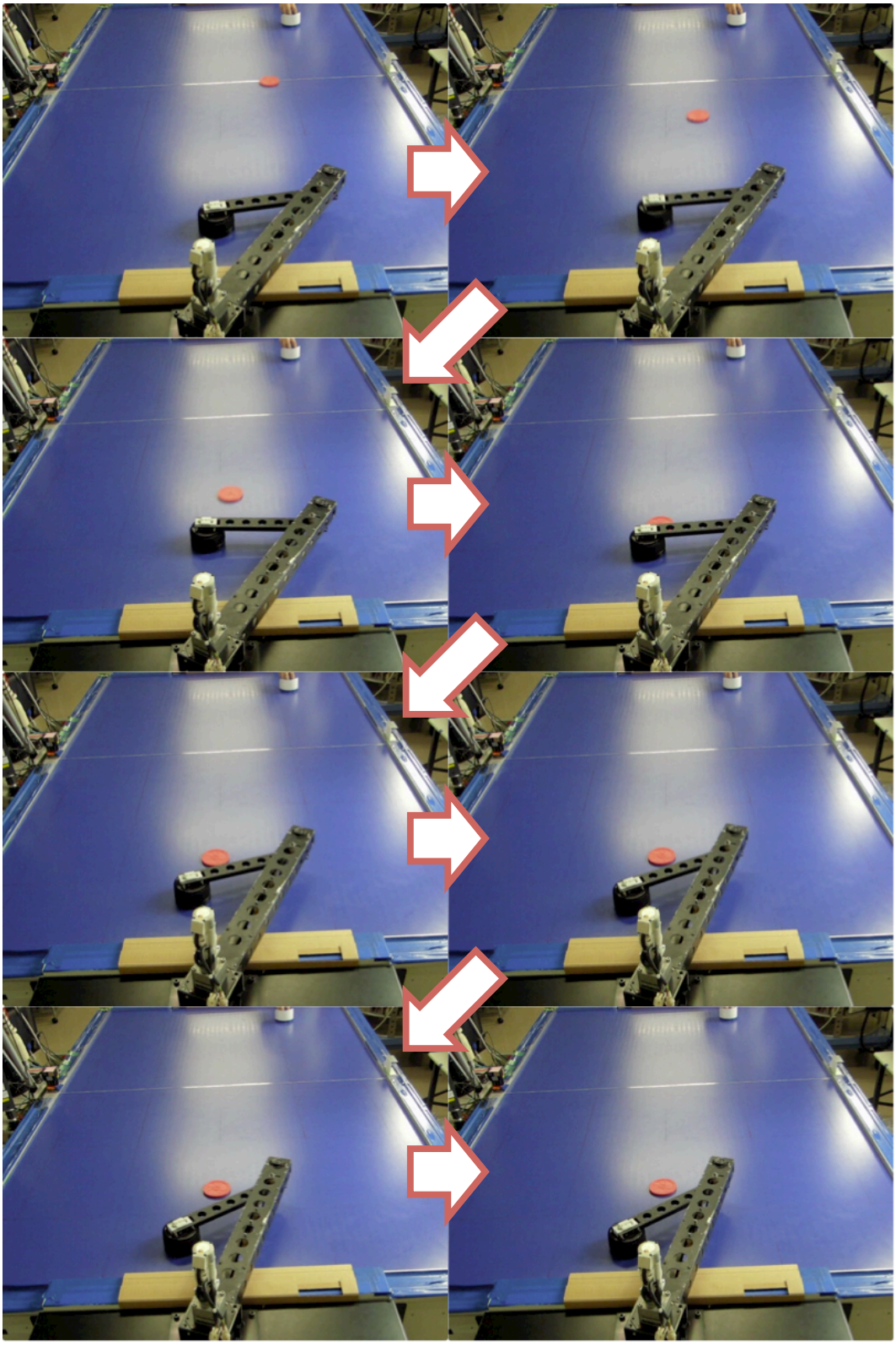

図 4.9: 止め打ちに成功したときの様子（パックを止めるまでの動き）

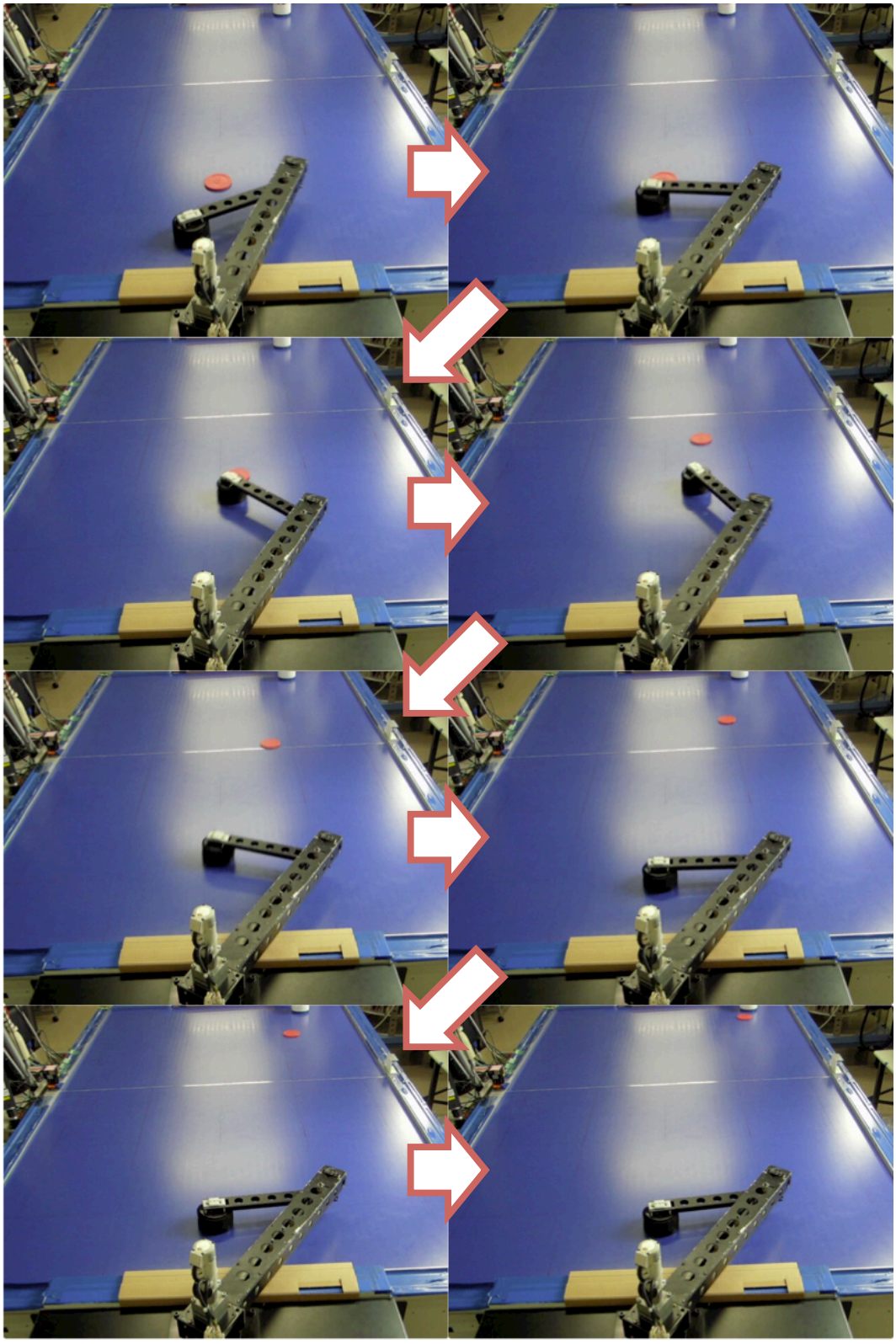

図 4.10: 止め打ちに成功したときの様子（パックを打ち返す動き）

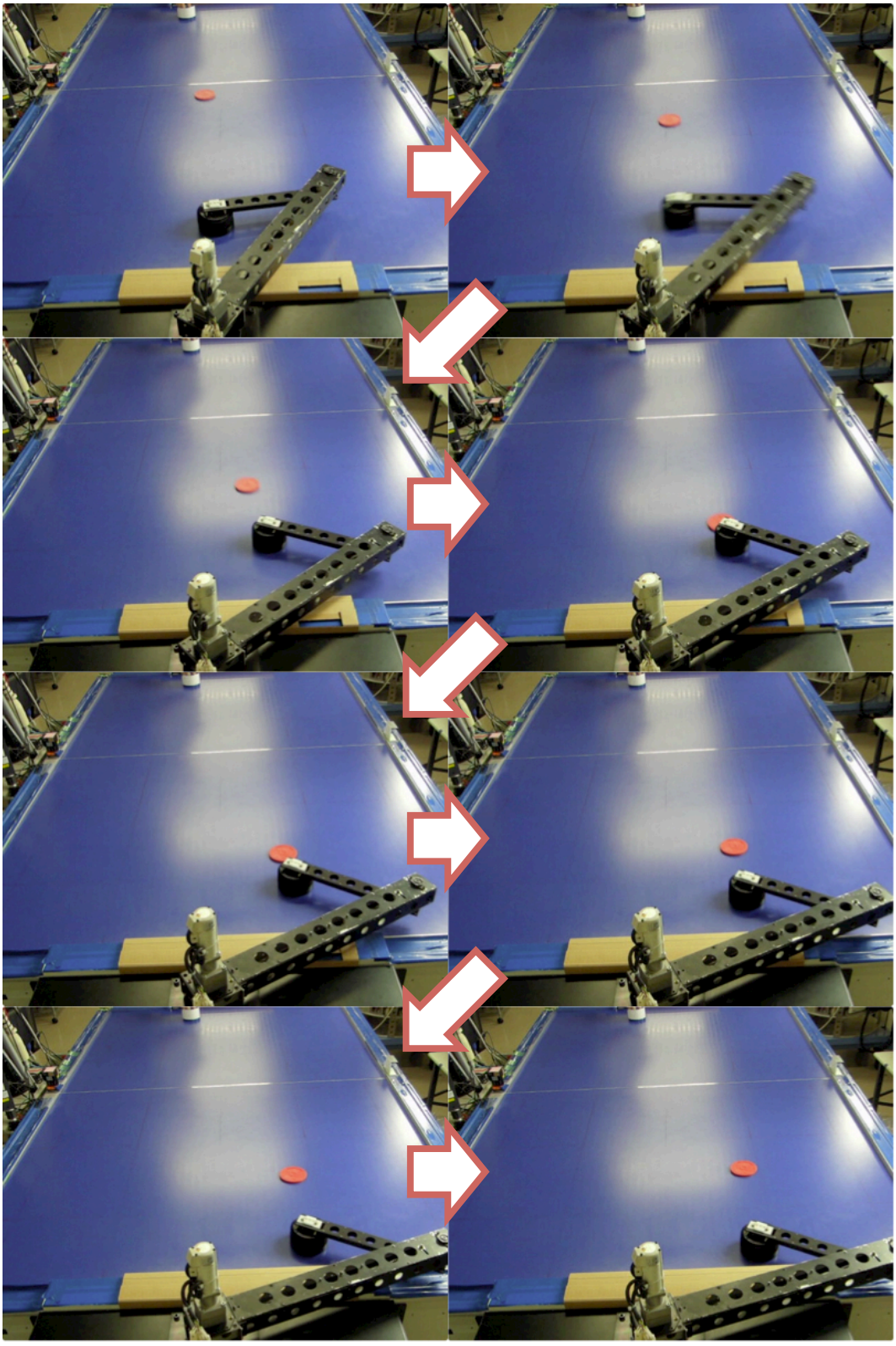

図 4.11: 止め打ちに失敗したときの様子（パックを止めるまでの動き）

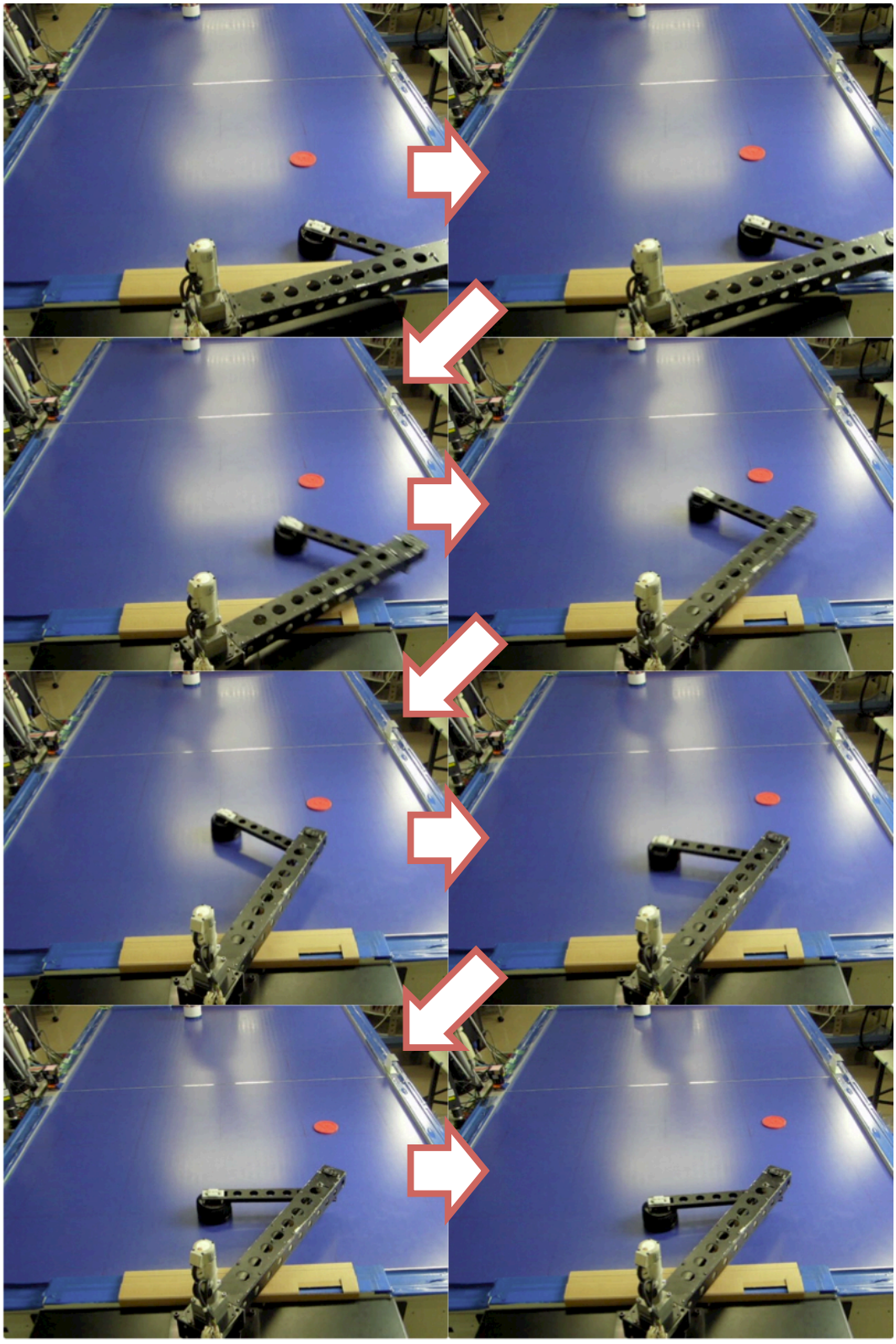

図 4.12: 止め打ちに失敗したときの様子（パックを打ち返す動き）

4.2 オンライン学習における行動要素の検索時間の検証

オンライン学習により対戦中に行動要素を獲得する場合，現状のアルゴリズムは，ロボットの対戦相手がパックを打ち返すたびに行動要素を追加していく．行動要素の増加につれて，再現すべき行動要素の検索時間が線形的に増大すると考えられる．本実験では，オンライン学習による対戦中に，行動要素の検索時間がどのように増加するのかを検証する．また，検索時間の増大により打ち返しが間に合わなくなる場合について検証し，打ち返しに成功しやすい打ち返し回数の範囲を考察する．

4.2.1 検索時間の増加状況の調査

実験方法

模倣アルゴリズムに基づいてオンライン学習をするロボットと人間とで対戦（人間側の打ち返し回数計 100 回）を行い，人間が打ち返した回数に対し，再現すべき行動要素の検索時間について計測する．なお，行動要素の検索の際に行動要素を選択対象とするパックの進入速度差 v_{diffmax} および角度差 $\theta_{\mathrm{diffmax}}$ は，それぞれ 2000 [mm/sec] および 10 [deg] とした．

実験結果

人間の打ち返し回数に対する行動要素の検索時間を図 4.13 に示す．図を見ると，前述の通り検索時間が線形的に増加していることが分かる．また，打ち返し回数が 10 回時点と 100 回時点とでは，検索時間が 10 倍程度に増えていることが分かり，対戦を長く続けた場合，ロボットが反応できずに得点されてしまうことが想定される．

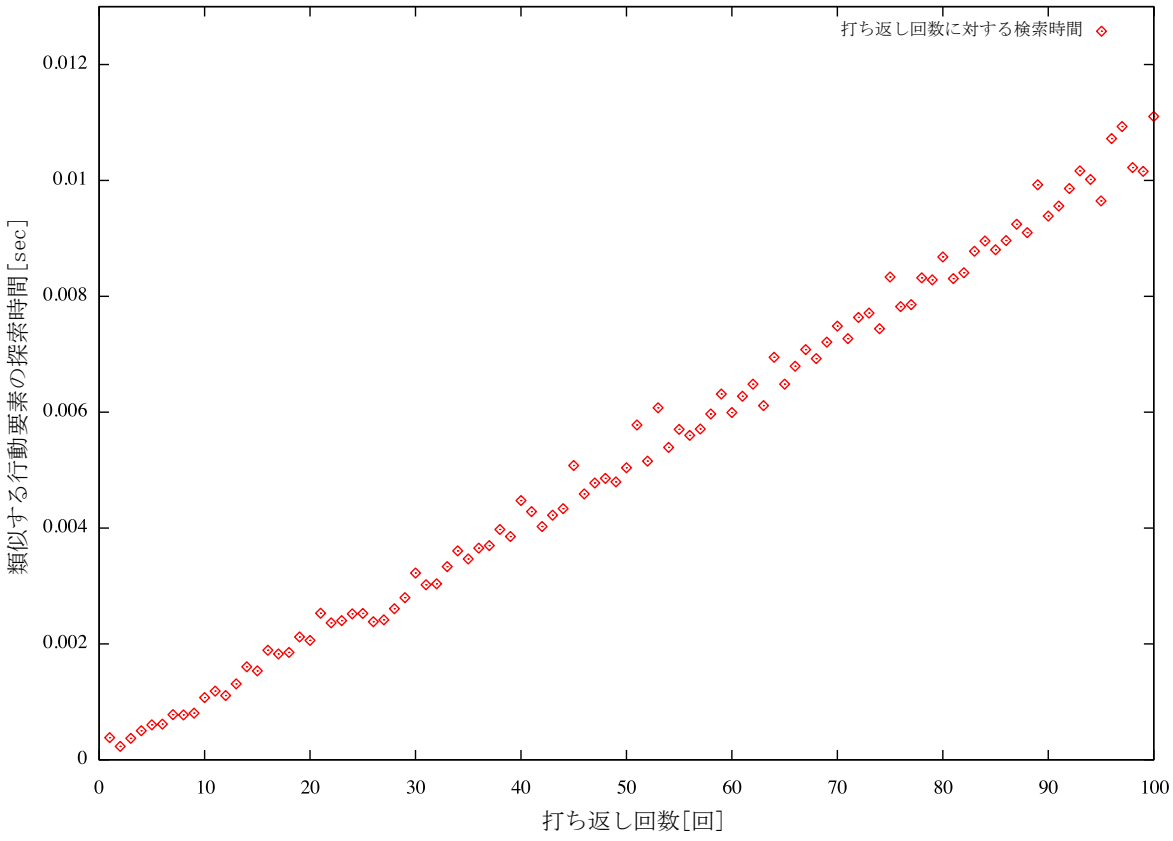

図 4.13: 打ち返し回数に対する行動要素の検索時間の遷移

4.2.2　打ち返しに成功しやすい打ち返し回数の検証

4.2.1 では，打ち合いが続くほど行動要素の検索時間が増加することが分かった．このことから，ロボットがパックを打ち返せるのは，学習がある程度進んだ時点から，ロボットが反応できなくなる学習回数に到達するまでであることが予想される．そこで本実験では，ロボットの打ち返し可否を調査することで，ロボットがパックを打ち返すことのできる最適な打ち返し回数について検証する．

実験方法

実験条件は 4.2.1 で述べたものと同様である．ただし今回は 800 回の打ち返しを行うこととする．実験では，“ロボットが打ち返しに成功した場合”，“行動要素が見つからなかった場合”，“検索時間が掛かりすぎて打ち遅れた場合” の 3 つについて，ロボットが打ち返し動作をするたびにカウントする．なお，本来の模倣アルゴリズムは行動要素が見つからなかった場合，単純な打ち返し動作をするが，本実験ではこの機能を無効にし，上記 3 つの状態を目視で調査することとする．

実験結果

実験結果を図 4.14 に示す．横軸は打ち返し回数に対する打ち返し可否の割合を示しており，各点は打ち返し 20 回ごとの割合を示している．緑色の棒は，ロボットが打ち返しに成功した割合を示している．赤色の棒は，行動要素が見つからず打ち返しすることができなかった割合を示している．紫色の棒は，行動要素の検索に時間が掛かり打ち返しに間に合わなかった割合を示している．

赤色のデータに注目すると，対戦開始直後は学習データが存在しないため，必然的に打ち返しができなくなっているが，学習を進めるにつれて，その数が減っていくことが分かる．また，紫色のデータに注目すると，打ち返しを続けるにつれて打ち遅れる回数が増え

ていくことが分かる．以上のデータを総合すると，緑色のデータから，打ち返しを 100 回程度行った時点で，ロボットの打ち返しが成功しやすくなっていくことが分かる．

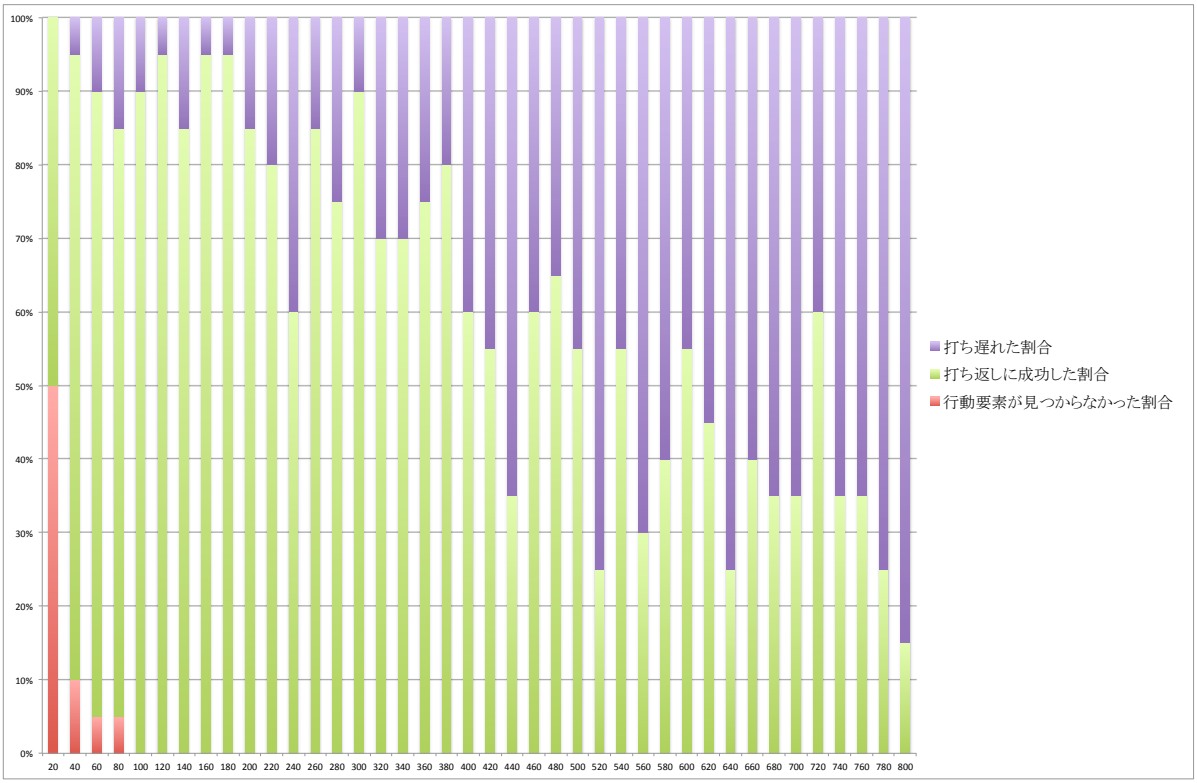

図 4.14: 打ち返しに成功した割合

4.3 まとめ

本章では，提案する模倣アルゴリズムが実際のロボットシステム上において有効であるかどうかについて検証した．4.1 では，模倣アルゴリズムが学習当時の相手の打ち方の特徴を再現できるかどうかについて，3 種類の特徴が大きく異なる打ち方をオフラインで学習させ再現させる実験を行い検証した．実験では，3 種類すべての打ち方について，特徴を再現することが成功し，本アルゴリズムが対戦相手の打ち方の特徴を真似することができることが確認できた．しかし止め打ちについては，まれにパックを静止させることができずに打ち返しに失敗する場合があることが判明した．このことから本アルゴリズムは，カウンター打ち・カット打ちのように一回だけ当てて打ち返す打ち方については有効であるが，止め打ちのように二回以上手元で当てて打ち返す打ち方については，学習当時の打ち方を再現できない場合があることが分かった．

4.2 では，ロボットにオンライン学習をさせた場合に，再現する行動要素の検索時間がどの程度掛かるのかについて，人間との対戦中に検索時間の計測を行い検証した．実験では，行動要素の学習数に対して検索時間が線形的に増加することが分かり，学習を進めるにつれてロボットが打ち返しに遅れることが分かった．また，人間との対戦において，ロボットが打ち返しに成功するパターン，行動要素が見つからずに打ち返しに失敗するパターン，および打ち方の再現開始が遅れ打ち返しに失敗するパターンについて調べ，打ち返しに最も成功する打ち合いの数について検証した．実験より，打ち合いの数が 100 回程度の場合において，ロボットが打ち返しに成功しやすくなることが分かった．

第5章　結言

5.1　結論

本研究では，対戦相手の打ち方を学習し真似をするロボットシステムを実現することを目的とし，次のことを行った．

- 対戦相手の打ち方を真似する簡単な模倣アルゴリズムの提案
- 模倣アルゴリズムを実装し稼働させるためのエアホッケーロボットシステムの構築
- 模倣アルゴリズムの有効性の検証

第 2 章では，対戦相手の打ち方を真似する模倣アルゴリズムを提案した．提案した模倣アルゴリズムは，対戦相手の打ち方が，パックの移動速度や角度，打点によって決定されるという単純なモデルに基づくものである．本アルゴリズムは学習処理と再現処理の二つに分かれており，学習処理では，対戦相手の方向へパックが向かっていくたびに，行動要素という単位でそのときのパックの動きや打ち返し方を記憶していく．また，再現処理では，ロボット側にパックが向かってくる際は，過去に学習した行動要素の中で現在の状況に最も近いものを検索し，見つけた行動要素における打ち方を再現し打ち返す．再現の際，そのままマレットの軌道を再生するだけでは打ち返しができない場合があるため，向かってくるパックの移動速度や方向に合わせて軌道の座標変換，再生速度の変更を行い，パックとマレットを衝突させ打ち返すことができるようにする．

第 3 章では，提案した模倣アルゴリズムの有効性を実機のロボット上に実装し検証するためのエアホッケーロボットシステムを構築したことについて述べた．本システムは，

パックを打ち返すための水平2軸ロボットアーム，パックの位置検出をするための高速カメラ，対戦相手のマレットの位置検出をするための測域センサで構成されており，それぞれを一つのPCで集中管理するものである．各デバイスの制御処理は，OpenRTM-aistを用いてそれぞれを単一のコンポーネントとして構成したため，対戦アルゴリズムを容易に変更可能であり汎用的なエアホッケーロボットシステムとなった．

第4章では，模倣アルゴリズムが実環境において有効であるかどうかを検証するため，今回構築したエアホッケーロボットシステム上に本アルゴリズムを実装し，主に二つの検証実験を行った．一つ目の実験は，本アルゴリズムが対戦相手の打ち方の特徴を再現してパックを打ち返すことができるかどうかについて，3種類の特徴が大きく異なる打ち方をオフラインで学習させ，ロボットに打ち返しをさせる実験を行った．実験では，カウンター打ち，カット打ち，止め打ちの特徴を捉えつつ打ち返しをすることができた．一方で，止め打ちのように2回以上パックとマレットを衝突させて打ち返す打ち方については成功率が低く，現状のアルゴリズムが対応しきれていないことが判明した．二つ目の実験は，本アルゴリズムの再現処理において，打ち返しに使用する行動要素の検索時間がどの程度掛かるのか計測し検証した．また，対戦を進めていく中で，打ち返しに成功しやすい打ち返し回数についても検証し，100回程度の打ち返しが最適であることが判明した．

以上より，今回提案した模倣アルゴリズムを用いて，カウンター打ちやカット打ちのように，マレットとパックを1回当てて打ち返すような単純な打ち方について，対戦相手の打ち方を再現することが十分可能であることが実験により明らかになった．簡単ではあるが，今回提案したアルゴリズムにより，対戦相手の打ち方を真似するエアホッケーロボットが実現された．本研究で示したように，人間のプレイスタイルを学習し真似することで，より娯楽性の高いエンターテインメントロボットの実現が期待される．

5.2 今後の課題

第 4 章における実験により，今回提案したアルゴリズムにはいくつかの問題点が存在することが判明した．本アルゴリズムは，カウンター打ちやカット打ちのように，1 回だけ手元で当てて打ち返すような打ち方に対しては有効であることが確認できた．しかし，止め打ちのように，手元で 2 回以上当てて打ち返すような打ち方については，打ち返しがうまくいかない場合が多く，アルゴリズムが不十分であることが判明した．これは，” 最初にマレットとパックが衝突する点” のみを打ち返し位置の基準としており，2 回以上マレットとパックを当てる動作に関しては考慮していないためである．また，打ち方の再現処理において，再現すべき行動要素の検索にかかる時間が学習を進めるたびに増加するという問題点がある．これは，対戦相手がパックを打ち返す度に新しく行動要素を記録してしまうためである．今後の課題としては，2 回以上パックとマレットを衝突させて打ち返す打ち方を再現可能にすること，また，検索時間が増大しないデータのまとめ方や検索手法を採用することが挙げられる．

謝辞

本研究を進めるにあたり，様々な視点から多くのご助言やご指導をいただいた末廣 尚士教授，工藤 俊亮准教授，冨沢 哲雄助教に感謝の意を表します．また，エアホッケーロボットシステムの再構築や修理にあたり，ご指導ご協力をいただきました平井 雅尊氏，ソフトウェアの開発面や修士論文執筆等においてご指導ご協力をいただきました村松 聡氏に感謝の意を表します．さらにエアホッケーロボットの故障の際にお立ち会いいただき，ロボットの修理や改良をしていただきました松田 啓明氏をはじめとし，日常の議論を通じて多くの知識や示唆をいただいた知能システム学講座の皆様に感謝の意を表します．
皆様のおかげで，楽しい2年間の研究生活を送ることができました．本当にありがとうございました．

参考文献

[1] 藤田 雅博 “エンタテインメントロボット：AIBO,” 映像情報メディア学会誌，Vol. 54, No. 5, pp. 657–661, 2000.

[2] 藤田 雅博 “エンターテインメントロボット AIBO の開発,” 精密工学会誌，Vol. 66, No. 2, pp. 181–164, 2000.

[3] 黒木 義博 “高度な運動能力を有する小型二足歩行エンタテインメントロボット SDR-4X,” 映像情報メディア学会誌，Vol. 57, No. 1, pp. 71–74, 2003.

[4] 柴田 崇徳 “メンタルコミットロボット・パロとロボット・セラピーの展開,” 日本ロボット学会誌，Vol. 24, No. 3, pp. 319–322, 2006.

[5] 今井 岳, 金岡 利和, 立田 隼人, 渡辺 一郎, 安川 祐介, “親和的なインタラクションに基づくサービスを提供する子ぐま型ソーシャルロボットの開発,” 第 12 回計測自動制御学会システムインテグレーション部門講演会, 1C2–2, 2011.

[6] Bradley E. Bishop and Mark W. Spong, “Vision-Based Control of an Air Hockey Playing Robot,” IEEE Control Systems Magazine, pp.23–32, 1999.

[7] 覺張陽則, “エアホッケーロボットの高速動作の制御と戦術の研究,” 電気通信大学 IS 研究科 U 専攻 2000 年度修士論文, 2000.

[8] 学張陽則, 羽田芳朗, 高瀬国克, “エアホッケーロボットの高速動作の制御と戦術の研究,” 第 18 回日本ロボット学会学術講演会, pp. 279–280, 2000.

[9] 覺張陽則, 羽田芳朗, 高瀬國克, “人間との対戦能力を有するエアホッケーロボットの開発,” 電気学会システム制御研究会, pp. 71–76, 2001.

[10] 牧野道徳, “エアーホッケーロボットのためのハンドアイ協調動作の研究,” 電気通信大学 IS 研究科 U 専攻 2007 年度修士論文, 2007.

[11] 牧野道徳, 根来寿, 賈松敏, 中後大輔, 高瀬國克, “PC ベースの高速画像処理を用いたエアホッケーロボットの開発,” 日本機械学会ロボティクス・メカトロニクス講演会’07, 1A2–C07, 2007.

[12] 牧野道徳, 根来寿, 賈松敏, 中後大輔, 高瀬國克, “PC ベースの高速画像処理を用いたエアホッケーロボットの開発　第 2 報：ハンドアイ協調による動作システムの実装と評価,” 計測自動制御学会第 8 回 SICE システムインテグレーション部門講演会, 2C2–3, 2007.

[13] 那順巴音, “エアホッケーロボットの攻撃戦略,” 電気通信大学 IS 研究科 MS 専攻 2010 年度修士論文, 2010.

[14] 那順巴音, 冨沢哲雄, 末廣尚士, “エアホッケーロボットの攻撃戦略,” 第 10 回計測自動制御学会システムインテグレーション部門講演会, 1N1–1, 2009.

[15] 御堂丸圭介, “エアホッケーロボットの止め打ち動作,” 電気通信大学 IS 研究科 MS 専攻 2011 年度修士論文, 2011.

[16] 御堂丸圭介, 冨沢哲雄, 工藤俊亮，末廣尚士, “エアホッケーロボットの止め打ち動作,” 第 12 回計測自動制御学会システムインテグレーション部門講演会, 3B1–3, 2011.

[17] 御堂丸圭介, 冨沢哲雄, 工藤俊亮，末廣尚士, “エアホッケーロボットの止め打ち動作の実現,” 第 13 回計測自動制御学会システムインテグレーション部門講演会, 1A1–7, 2012.

[18] 松下左京, 並木明夫, “ゲーム状況に応じた意思決定を行うエアホッケーロボットの開発,” 日本ロボット学会誌, Vol. 29, No. 10, pp. 954–962, 2011.

[19] OpenRTM-aist official website: *http://www.openrtm.org/openrtm/ja/content/openrtm-aist-official-website*, (参照 2013-01-24).

[20] TRINH VAN VINH, 冨沢 哲雄, 工藤 俊亮, 末廣 尚士, “マニピュレータによる柔軟紐の片手結び,” 第 11 回計測自動制御学会システムインテグレーション部門講演会, 2E3–2, 2010.

[21] DOGO WAKOUBO JEAN, 冨沢 哲雄, 工藤 俊亮, 末廣 尚士, “OpenRAVE と OpenRTM を連携させるためのコントローラに関する研究,” 第 12 回計測自動制御学会システムインテグレーション部門講演会, 3P2–2, 2011.

[22] 前田 雄哉, 冨沢 哲雄, 工藤 俊亮, 末廣 尚士, “折り紙公理に基づいた作業記述-ロボットアームによる折り紙作業実現に向けて-,” 第 12 回計測自動制御学会システムインテグレーション部門講演会, 3B2–3, 2011.

[23] Michael Isard and Andrew Blake, “CONDENSATION – Conditional density propagation for visual tracking,” International Journal of Computer Vision 29(1), pp. 5–27, 1998.