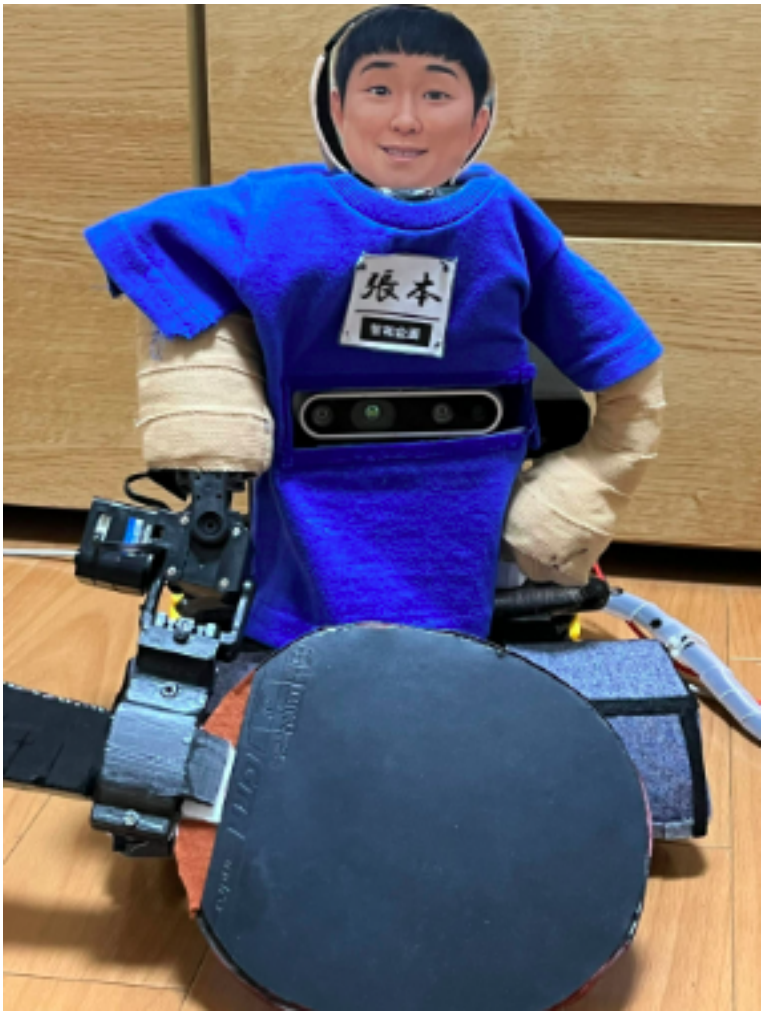
2024A・機械情報冬学期演習・自主プロジェクト

卓球ロボット HARIMOTO



東京大学工学部機械情報工学科３年
学籍番号：　03-240281
氏名：　椿　道智

1　概要

本プロジェクトでは，卓球台上を移動可能な卓球ロボット HARIMOTO（以下，単に「ロボット」という.）を製作した．ロボットは,「プッシュ卓球打ち（＝温泉卓球打ち)」「ドライブ打ち」の二種類の打撃方法により，投げた卓球ボールに対して，タイミングを合わせてラケットを振ることができる．単に，打撃制御や画像処理をするだけではなく，ボールが返球された際の「チョレイ」効果音や外装の工夫により，実際の張本選手感（卓球選手感）を出し，ユーザ側が楽しめるようにも工夫した．

ソフトウェア面では，アームの動作（卓球ボールの打撃)，D435i による画像認識（卓球ボールの認識)，ボールの着地予測，メカナムホイールの制御，音声再生を行った．機械工学少人数ゼミ（岡田慧教授ゼミ）や知能ロボット演習（機械情報冬学期演習）で学んだ euslisp・ROS noetic を用いてシステムを組んだ．

ハードウェア面では，4 自由度アームの設計・製作，ラケットや服，左腕や靴など外装の製作・塗装を行った．

なお，プレゼンでは技術的なことを永遠と喋っていても生産性がないので，わかりやすさ重視で技術的な詳細をあえて省いた．本レポートでは，それを補う意味でも，技術的にこだわった点・細かい工夫などを詳細に記述する（粒度がかなり細かい）ので，興味のないところは適宜読み飛ばしていただきたい．

2　動機・背景

私は手術ロボットや医療ロボットに興味があり，特に縫合など針を使った自律的なタスクに関心がある．そのため，学期の初めには縫合ロボットを作ろうと考えていた．しかし，少人数ゼミで縫合に挑戦する機会が得られたため，自主プロジェクトではあえて全く異なるタスクのロボット製作に取り組むことにした．

自主プロジェクトは「何でも作っていい」という貴重な機会であるため，自分の趣味である卓球ロボットを作り,「自分が楽しむ」ことを最優先に考えた．卓球ロボットといえば，従来研究では卓球台の外に大きなパラレルリンクを設置して返球させる手法が主流なのは周知の通りである．しかし，似たようなものを作ってもオリジナリティがないと思い，本プロジェクトでは，従来の方法とは逆に「卓球台の上で省スペース」「シリアルリンク」の新しい卓球ロボットを制作することを目指した．

3　コンセプト（実現可能性の根拠）

卓球選手の戦型には，前陣型（速攻型)，中陣型、後陣型の 3 つがある．それぞれの戦型は，卓球台からの距離に応じて，前陣型は台から 0〜1m，中陣型は 1〜2m，後陣型は 2m 以上離れた位置でプレーする．

この中で前陣型は，卓球台に非常に近い位置でプレーするため，ボールが台に着地した直後にラケットで返球する．この戦型が存在するということは，理論的にほとんどのボールを台から離れず，着地直後に返球できることが示唆される．

これを前提に，卓球台上でボールを返球するロボットを作成することが可能だと考えた．ロボットは台上で動きながら，ボールが台に落ちた直後に返球できるため，台から離れずに動きながらボールを処理できるというメリットがある．また，ボールが着地した瞬間に返球するため，ラケットの高さを低く設定できる点もメリットである．

一方，前陣型は速攻型とも呼ばれるように，非常に短い時間でボールに反応しなければならないため，ロボットには素早い画像認識と判断能力，移動能力が求められる．この反応速度が，本プロジェクトの課題の一つとなっている．

4　システム構成

ロボットのシステム構成を図 1 に示す．ロボットは，4 自由度ロボットアーム・Interl D435i depth カメラ・メカナムホイール・スピーカーから構成される．これらと，ホストコンピュータの main.l（制御アルゴ

リズム），画像処理のノード（detect_ball.launch で起動）を ros topic を介して通信する方法でシステムが構成されている．

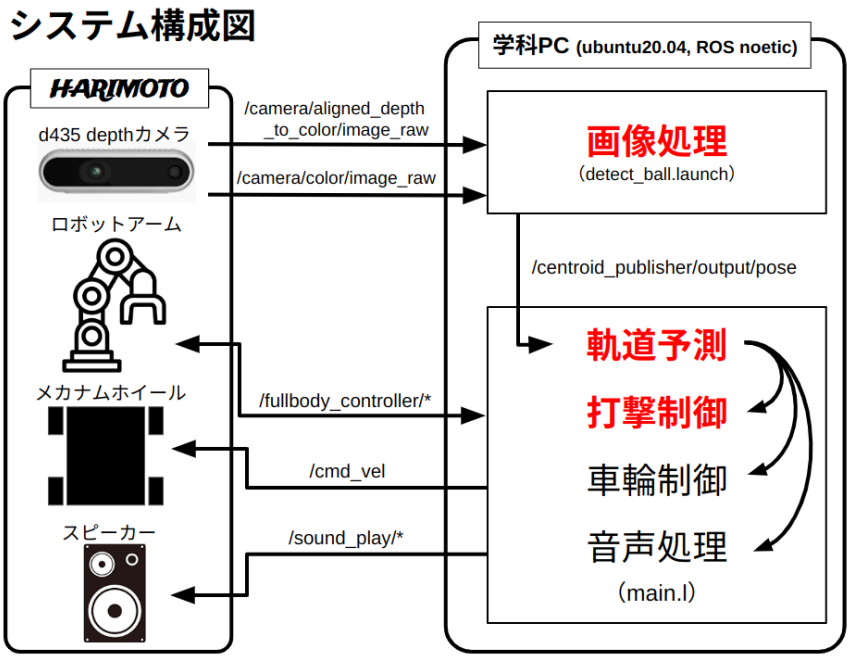


図 1: システム構成図

次の 5 章では図 1 右側の「ソフトウェア」を，6 章では図 1 左側のハードウェアや外装について詳しく述べる．

5　ソフトウェア

この章ではソフトウェアの実装について述べる．まずは全体を概観したのち，詳しい実装について説明する．

5.1　機能の概観

ロボットの主要機能は以下の通りである．

1. **2 種類の速度計測**
   (a) 即時速度
   (b) 移動平均速度
2. **着地予測**
   (a) 着地予想時間
   (b) 着地予想位置
   (c) 距離ごとに異なる方法
      i. 近距離 (NEAR)
      ii. 中距離 (MIDDLE)
      iii. 遠距離 (FAR)

3. **着地地点への移動**

4. **打撃**

   (a) プッシュ打ちモード

   (b) ドライブ打ちモード

5. **音声**

   (a) 打撃した際に「チョレイ」と叫ぶ.

   (b) サーボに長時間負荷をかけるとモータが焼けるので，一定時間経過後アラームを鳴らす.

以下，これらの機能を実行するための各プログラムとその統合について述べる．概観→詳細の順で説明したほうがわかりやすいので，上層のインテグレーションから説明する

5.2 インテグレーション（ROS package HARIMOTO）

ロボットを ROS noetic で動かすために，卓球ロボットの知能処理プログラムとロボットのモデルなどを 1 つの ROS package ”HARIMOTO” としてまとめた．構成は以下の通りである．

```
├── CMakeLists.txt
├── config
│   ├── HARIMOTO_servo_config.yaml    #サーボ ID の設定
│   ├── joint_names_HARIMOTO.yaml    #サーボとモデルのジョイントのインタフェース
│   └── mecanum_drive_controller.yaml    #メカナム制御の設定
├── launch
│   ├── all.launch    #下記の launch+sound_play ノード等をまとめて起動する
│   ├── detect_ball.launch #ボールの画像認識ノードの起動
│   ├── mecanum_drive_controller.launch # ros:ridgeback_control のノードの起動
│   └── minimal.launch    #KXR controller の起動
├── meshes    #ロボットモデルのメッシュファイル (STL)
├── package.xml    #package の依存関係の定義
├── src
│   ├── HARIMOTO-interface.l    #KXR controller と main コードのインタフェース
│   ├── HARIMOTO-speak.l    #音声再生のインタフェース
│   ├── main.l #メインプログラム
│   ├── falling-point-estimation    #着地予測の複雑な処理（今回は使わない）
│   └── scan_ids    #サーボ ID のスキャナ
├── file #「音声再生」に用いるファイル群
│   ├── start.mp3
│   ├── chorei.mp3
│   └── warn.mp3
└── urdf
    └── HARIMOTO.urdf    #ロボットモデル
```

以降では，上記のファイルが実際にどのような働きをするのかに注目しながら，ソフトウェアの実装を述べる.

5.3 ロボットモデルとハードウェアインタフェース

5.3.1 ロボットモデルの作成

Solidworks(CAD ソフト) でアームの設計をした後，ROS 用のロボットモデルとして”HARIMOTO.urdf” と関連メッシュファイルを urdf-converter で出力した．この自動で出力した URDF（ロボットモデルファイル）には，アームの joint 情報と link 情報しか記述されていないので，さらに，D435i Depth カメラの座標（Camera_link）の情報や，メカナムホイールの JOINT, LINK の情報，ロボットの胴体の LINK の情報を手作業で変更した（この CUI で URDF を書き換える作業がかなり大変だった）．次いで，”servo_config.yaml” や”joint_name.yaml” でサーボ ID 情報や URDF の JOINT 名とサーボ ID の対応付けを行った．

5.3.2 ロボットモデルと euslisp とのインタフェール

”src/HARIMOTO-interface.l” を，知能ロボット行動プログラム演習の”jedy-interface.l” を真似しつつ，不要なメソッドなどを削除することで，記述した．これにより，euslisp で”(send ri :<method>)” によりロボットの行動計画を行うことが可能となった．

5.3.3 ハードウェアインタフェース

コンピュータとロボット（モータ）とのハードウェアインタフェースについては，jedy の演習で用いた”KXR_controller” という package を用いた．その KXR_controller のノードを起動するために jedy 同様”minimal.launch” を記述した．これを立ち上げることで，euslisp で書いたコードが実機で動くようになる．

5.4 打撃〜アームの行動計画〜

本ロボットでは，4 自由度アームの軌道生成によって，打撃を実現している．卓球の場合は，ラケットの向き（＝エンドエフェクタの姿勢）が重要となる．一方，位置と姿勢を一意に指定するためには 6 自由度が必要で inverse kinematics が解けない．[1]そこで今回は，実験的にアームの軌道を求めた．

1. サーボオフにする
2. ロボットのラケットを手で持って打球時の軌跡を描く
3. その時のサーボモータの angle を取得する

の手続きで取得した joint-angle の値ををそのまま再生しても，重力の影響で思っていた軌道よりもたるんでしまう．そこで，その分のオフセットを入れれば一通り打撃することができる．

次に，高いボール・低いボールに対応できるように，高さを引数にできるように改良した．根本の関節（肩の関節）を高さによって変え，それによってラケットの面の向きがわかる分を他の関節で調整するように JOINT 軌道を設計した．完成した関数は，以下の通りである．なめらかな動きを実現するために．*ri* のメンバ関数である:angle-vector-sequence を用いた．

さらに，各アクチュエータの速度は遅いながら，全体として速い軌道を実現するために，落下方向に降るときに「**重力**」を利用したり，前方向に降るときに「**足回り**」を一緒に前に動かすことで，打球時の一瞬にロボットの運動ポテンシャルを最大限発揮できるように工夫して動作を設計している．この部分の軌道設計はかなり苦労した．以下に実際のプッシュ関数の中身を示す．

[1]卓球はラケット平面が決まればその面にそってラケットを滑らせるだけなので，4 自由度で足りるのではないかという仮説に基づいて，コストカットのために 4 自由度にしたが，プログラムの視点で見れば IK が解けないのは絶望的に面倒なことをしたと反省している．

```
(defun push-hit(x) ;;引数xには高さの定数倍が入力される
  (let (p1 *basic-pose-l*) ;; 初期姿勢
       ;; 実験的に求めつつも，高さ引数分の補正を行っている．
       (p2 (float-vector (+ 2 x) 20 10 (+ 30 x)))
       (p3 (float-vector (+ -3 x) 10 10 (+ 20 x)))
       (p4 (float-vector (+ -7 x) 0 10 (+ 10 x)))
       (p5 (float-vector (+ -13 x) -5 15 (+ -0 x)))
       (p6 (float-vector (+ -20 x) -10 20 (+ -15 x)))
       (p7 (float-vector (+ -35 x) -20 25 (+ -35 x)))
       (tm 25))
    ;;
    (send *ri* :send-cmd-vel-raw 0.01 0 0)
    (send *ri* :angle-vector-sequence
          (list p1 p2 p3 p4 p5 p6 p7 p1)
          (list tm tm tm tm tm tm tm 2000)
          :default-controller 0.001
          :min-time 0.00001
          :minjerk-interpolation nil)
    (chorei)
    (ros::duration-sleep 0.5)
    (send *ri* :send-cmd-vel-raw -0.01 0 0)
    (send *irtviewer* :draw-objects))
```

他にも，ドライブモードを実装したが，オフセットのパラメタ調整が発表までに間に合わなかった．

5.5 ボール位置の検出

Intel RealSense Depth Camera D435i を用いたボールの中心位置の取得するプログラムついて説明する．

はじめは，少人数ゼミや演習で試したように HSI カラーフィルタでフィルタリングした点群のユークリッド距離クラスタリングをして，そのクラスタの中心位置を取得していた [2]．しかし，このシステム構成だと，処理が遅く，ボールを見逃してしまい，卓球ボールの認識システムとして機能しなかった．そこで，矢野倉先生にご相談したところ，クラスタリングの処理が重いということだったので，「HIV フィルター」でマスクした点群の重心 (centroi) を取得することを提案していただき [1]，発表時点では，そのようなシステム構成とした．具体的には，

1. color のフレームに対してカラーフィルタを作る
   opencv_apps/hsv_color_filter
2. color フレームのフィルタを depth フレームに適用できる形にする
   jsk_perception/depth_image_filter
3. depth フレームを 2. のフィルタでマスクする
   jsk_perception/ApplyMaskImage
4. フィルタ後の depth 画像の重心を取得して publish する
   jsk_pcl_utils/CentroidPublisher

のノードを順に実行している．

上記のノードの構成についてはノードを起動するための”detect_ball.launch” に記述してある (https://github.com/Michi-Tsubaki/ping-pong-robot/blob/main/src/HARIMOTO/launch/detect_ball.launch, コメントアウトが旧 ver)．

なお，Depath カメラ D435i の起動には，realsense-ros package を用いて，roslaunch realsense2_camera rs_camera.launch で起動している．rs_camera.launch を書き換えることで，Color と Depth のフレームレートを変更でき，今回は，ハードウェア上の限界である 60fps に設定した．[3] はじめは usb2.0 を使用しており，最大フレームレートが 15fps に留まっていたが，usb3.0 に変えたことにより，最大フレームレートが 60fps に上がった（こういうハードウェア上の制約も勉強になった）．

5.6 着地予測

5.6.1 速度計測

ボールの着地予測をするために，まずはボールの速度を知る必要がある．速度は，その瞬間の速度（即時）速度

$$v(t) = \Big[\frac{x(t)-x(t-1), y(t)-y(t-1), z(t)-z(t-1)}{dt}\Big]^\top$$

と，3 フレーム分の速度が連続で存在したときに，その移動平均を返す．

$$\hat{v}(t) = [\frac{v(t)+v(t-1)+v(t-2)}{3} \quad \text{, if } v(t), v(t-1) \text{ and } v(t-2) \text{ exist.}]$$

移動平均を取ることは，ノイズの影響が小さくなる（ローパスフィルタ）一方で，速度情報が遅れてしまったり，速度が連続で取得できなかった場合は無効となる（出力されない）ので注意が必要である．

5.6.2 着地予想時間

ロボットの着地予想時間は，z 方向 (ロボットの正面方向) のロボットの位置に依存するので，ロボットが打撃できる空間の位置 (z_0) に到達するまでの時間を計測する．

$$\text{Estimated Time} = T = \frac{z(t)-z_0}{v_z(t)}$$

この式の $v_z(t)$ に $v(t)$ の z 成分を用いるか，$\hat{v}(t)$ の z 成分を用いるかは，ボールの距離 $z(t)$ がロボットに近いか遠いかで決める．

ボールの位置によって，空間を

1. 近距離 〜 25 cm
2. 中距離 25 cm 〜 60 cm
3. 遠距離 60 cm 〜

に分け，中距離では $v(t)$ を使い，遠距離では $\hat{v}(t)$ を用いる．（近距離では予測をせずに即時返球する．）

5.6.3 着地予想位置

着地予想位置については，単純に，

$$\text{Estimated Point} = \mathbf{x} = [x(t)+v_x(t)\times T, y(t)+v_y(t)\times T, z_0]^\top$$

として計算した．二次関数フィッティングも試みたが，卓球ボールがマグヌス効果で二次関数を描かないせいか，単純にボールの軌跡がほぼ直線だからかは考察していないが，着地予想位置の精度がでなかったので，最終的には単純な線形モデルに回帰することになった．

ボールの z 方向の速度が正のとき ($v_x > 0$) はボールが離れていっているので，着地予測をせず，また，ボールの速度がとても大きいときとても小さいときは，ノイズと判断して，空振りを防ぐため，着地予測を

行わないようにした．他にも，予測モデルが線形でシンプルな分は，予測に条件を課すことで，カバーしている．

着地予想位置については，着地予想時間と一緒に ros topic として Publish するようにした．これにより，Rviz で着地予想位置を描画できる．メッセージの型には，geometry_msgs の Pose を用いた．着地予測時間を orientation（クオータニオン）の w に入れた

5.7 メカナムホイール制御

メカナムホイールの制御プログラムについて説明する．メカナムホイールは，ROS Control の Ridgeback Controller を用いることで，左右・前後・斜め移動といったメカナム特有の動きをすることができる [4]．mecanum_drive_controller.launch でこの package のノードを起動した．あわせて，”HARIMOTO-interface.l” で cmd_vel トピックとその動きを接続するインタフェースを作ってあげることで，euslisp で (send *ri* :send-cmd-vel-raw x y rot) と送るだけで，cmd_vel にメッセージが送られロボットの足回りを動かせるようになった．

また，移動用のサブルーチンとして, 基準値よりも左にいるか右にいるか, 前にいるか後ろにいるかによって，逐次 cmd_vel を送信する「move_follow」関数と，指定した相対座標に移動する「move_to」関数を実装した．

1. move_follow
   ロボットが返球に対応できる範囲 (sweet_spot) に対して，逐次更新されるボールの着地予測地点がが右にあるか左にあるか，前にあるか後ろにあるかによって，逐次ロボットが動くことを目的としたサブルーチン．

2. move_to
   camera_link 座標系（カメラに固定された座標）からの相対座標で計算されたボールの着地予想着地地点に直接移動することが可能．

発表会時点では，move_follow 関数を用いて着地予測地点に移動していたので，移動できる範囲が左右前後 10cm くらいに限られていたが，着地予測の精度が高くにあれば，move_to 関数を使ってより広い範囲で移動できるはずだと考えている（スピードがあまり出ないので，当然広々動き回れるわけではない．）

5.8 音声再生

ここでは，音声に関する実装について説明する．少人数ゼミで PR2 ロボットを扱った際に，PR2 に会話をさせるインタフェースとして，”speak.l” が存在した [5]．このソースコードを読むと，ROS の sound_play パッケージと euslips のソースコードのインタフェースになっていることが分かったのでそれを活用した．

(send *ri* :sound-play #P”<mp3 file の path>”)

とすれば，MP3 の音声ファイルを再生することができる．今回は，ユーザが楽しむという観点，サーボモータに長時間負荷をかけ続けてモータが焼けるのを防ぐ観点から

1. 打撃するたびに「チョレイ」と張本のように叫ぶ．
2. 起動するときに「お相手しましょう！」と元気よく叫ぶ．
3. プログラム開始後 2 分後と 5 分後に「肩が疲れたよ」とアラートを発する．

の 3 種類の音声再生を実装した．

以上が，ロボットのソフトウェアの実装である．

6　ロボットモデル・ハードウェア・メカニクス

6.1　ロボット設計・製作

本節では，ロボット全体の設計思想・製作，自作パーツの設計について説明する．

6.2　ラケットの自作

6.3　外装

当初の計画では，本プロジェクトはソフトウェアに注力し，ハードウェアはあまり凝らない予定だったが，「楽しめる／発表のときに楽しんでみてもらえる」ロボットを作るためには，ソフトだけではなく，やはり外装も重要だという結論

7　本プロジェクトを通した気づき（結論）

ここまで述べてきたように，このロボットには，様々な苦労と工夫が詰まっているので，敢闘賞を頂けたのは嬉しいことだが，次章で述べるとおり，未だ課題も多い．

特に，本プロジェクトを行う前は，ボールの認識について「カメラの仕様的に大丈夫だろう」という甘い考えで望んでいたが，実際には，高速で移動する物体の認識がかなり難しいということを実感した．

8　今後の展望

本プロジェクトを始めた当初の目標は，大きな目標として，

1. 来たボールを返球する
2. ボールの着地予測地点に移動して返球する
3. ボールの回転に応じて返球方法を動的に変えられるようにする

の 3 点を掲げた．自主プロ発表会では，1. と 2. を部分的ついて部分的に成功し発表することができた．一方で，

1. 返球できる確率は $\frac{1}{2}$ 程度に留まった
2. 移動して対応できる距離はせいぜい $\pm 10cm$ であり，それ以上は返球が間に合わないため狭い範囲に留まった
3. ドライブ打ちモードをコード上は実装したものの，動作パラメタのチューニングが間に合わなかった．また，プッシュ打ちとドライブ打ちを動的に変更するのは実現できなかった．

は今後課題として残った．

ロボットを五月祭に展示するにあたり，以下の課題について春休みに調査・開発を行うことにした．

8.1　画像認識：Depth 情報のみからボールの位置を取得するアルゴリズムの開発

D435i を用いたボールの位置判定を行う際，Depth 情報を使用する前にカラーフィルターを用いたマスク処理をしている．そのため，Color と Depth の両方の情報を同時に取得する必要があり，この場合の取得周期は「60fps」が限界である（両者が同期する必要があるため）．この制約が位置取得の遅れのボトルネックとなり，返球できる確率を制限していると考えられる．

D435i は，Depth 情報のみを使用する場合には 300fps を実現できるという公式の発表がある [6]．Depth 情報だけを用いてボールの位置，速度，着地地点を予測するアルゴリズムを開発できれば，Color 情報は不要となり，300fps のフレーム情報を活用できるため，ラケットの振り遅れを大幅に減らし，返球確率を向上させることができると考えている．

Depth 情報を処理する際にクラスタリングを行うと処理速度が遅くなることを，このプロジェクトを通じて痛感したため，クラスタリングを経由しないアルゴリズムを考案する必要がある．

以下は、本課題に対するアプローチで現時点で有用だと考えられるアプローチ（仮説）である．

- 深尾教授のロボットコントロールの講義で学んだ自動運転における「時間的に連続するフレームの差分を取ることで移動する障害物（人など）の情報を抽出する」手法を応用することで，移動物体であるボールを効率的に検出できるのではないかと考えている．時間計算量は処理で扱う点数（特徴点の数）に依存するため，この特徴点抽出により処理全体が高速化できると予想している．
- 國吉教授のロボットインテリジェンスの講義で学んだ「Time to Collision（衝突までの時間）」の概念を用いることで，ボールの着地予測時間を簡単に算出でき，計算処理の高速化が期待できる．
- 矢野倉先生から助言をいただいた「半円殻（卓球ボールの像）」のマッチングや，岡田ゼミで使用した「Hough 変換による円抽出」を 3 次元空間に応用する方法（射影処理など）を通じて，ボールを抽出する手法を取り入れたいと考えている．

この春休みを通じて，上記の仮説を検証し，計算量を削減するための特徴点抽出と Depth フレームからの卓球ボール高速検出の課題に取り組んでいく．

8.2　ドライブ打ちモードのパラメタチューニング

前述したとおり，ドライブモードをできるシステム構成は完成しているので，ロボットアームの初期姿勢やラケットを振るアームの速度・経路を調整する．また，ボールがこの範囲に着地するときにラケットを振るという基準値もプッシュ打ちモードとは異なるはずなので，その範囲（スイートスポットの値など）も調整する．

9　謝辞

本プロジェクトに取り組むに際し，ラフスケッチ・詳細スケッチに対して実現可能性や計画性，機構面で改善すべき点についてアドバイスをくださった小島邦生先生，Depth Camera による画像処理の方法や KXR の Configuration についての逐次の質問やロボットパーツなどの資材の貸出対応で大変お世話になった矢野倉伊織先生，3D プリンタ環境をご提供くださったメカノデザイン工房の技術職員の方々，アドバイスをくれた機械情報工学科の同級生に御礼申し上げます．

参考文献

[1] iory, “color_filter.launch,” GitHub Gist. [Online]. Available: https://gist.github.com/iory/c0b13faa5bceb062dcc37376a7af9f1b

[2] Michi-Tsubaki, “recognize_wound.launch,” jsk_demosm GitHub. [Online]. Available: https://github.com/Michi-Tsubaki/jsk_demos/blob/jsk_2024_10_semi/jsk_2024_10_semi/pr2_surgery/launch/recognize_wound.launch

[3] porizou1, ”ubuntu20.04 と ROS Noetic で RealSense を使う手順” Qiita. [Online]. Available: https://qiita.com/porizou1/items/be1eb78015828d43f9fb

[4] iory, “[jedy_bringup] support mecanum drive using ridgeback_control #2,” robot-programming, GitHub. [Online]. Available: https://github.com/iory/robot-programming/pull/2

[5] jsk-ros-pkg, “speak.l,” jsk_pr2eus, GitHub. [Online]. Available: https://github.com/jsk-ros-pkg/jsk_pr2eus/blob/master/pr2eus/speak.l

[6] intel realsense, ”High Speed Capture for Intel® RealSense™ Depth Cameras”, April 8, 2020, https://www.intelrealsense.com/high-speed-mode-at-300-fps-for-depth-camera/

10 （参考）環境構築・実行方法（Github）

本プロジェクトに関する全てのソースコードと自作パーツのモデルは, https://github.com/Michi-Tsubaki/ping-pong-robot で公開している. ソースコードについてはそこを参照してほしい. また, ロボットの URDF を作成しているため，実機がなくても手元の環境で誰でもシミュレーション可能である.

10.1 環境構築

知能行動ロボットプログラミングの演習を受講後の学科 PC の環境（ubuntu20.04, ros noetic）で開発を行った．以下を実行し，リポジトリをホームディレクトリにクローンする.

```
cd
git clone https://github.com/Michi-Tsubaki/ping-pong-robot.git
cd  /ping-pong-robot
```

10.2 実行方法

ターミナルを開いて，以下を実行し，

```
cd  /ping-pong-robot
source devel/setup.bash
roslaunch HARIMOTO all.launch #画像処理やアームコントローラのノードを起動
```

また，別のターミナルで

```
cd  /ping-pong-robot
source devel/setup.bash
roscd HARIMOTO/src
emacs -nw main.l
M+x shell RET
roseus
load ”main.l”
```

を実行することでロボットのメインプログラムを動かすことができる．

build 済みの環境を clone することになる. エラーが出る場合は, パスがおかしかったり, 必要な ros package が存在しない可能性がある．この場合，sudo apt update や rosdep install を実行する必要がある．その後, catkin build をやり直す（clean build がよい）.