第7回ROBO-剣優勝ロボット「逡巡」

山口辰久（チーム山口自動機械）

自己紹介

山口辰久

- Twitter: @qzy13700
- 京都大学学生サークル「機械研究会」OB
- ROBO-剣　第1回大会から参加
- ROBO-ONE 時々参加
- マイクロマウス、知能ロボコン、etc.
- 会社員（ソフトウェア技術職）
- 「山口自動機械」は実在する法人名・屋号ではない
    - ウェブサイトの名前＆ロボコン参加チームの名前

講演の内容

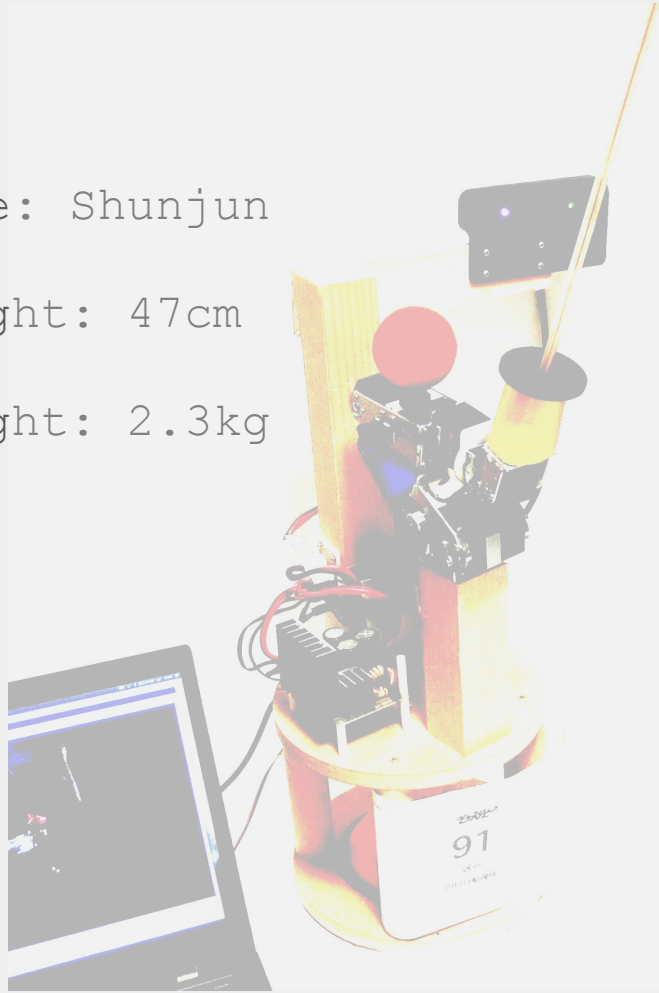
1. ハードウェア
2. 制御アルゴリズム
    a. 標的位置の検出
    b. ハンドアイ校正
    c. 経路計画
    d. 試合の戦略
3. ソフトウェアシステム構成

Chapter 1: Hardware

name: Shunjun

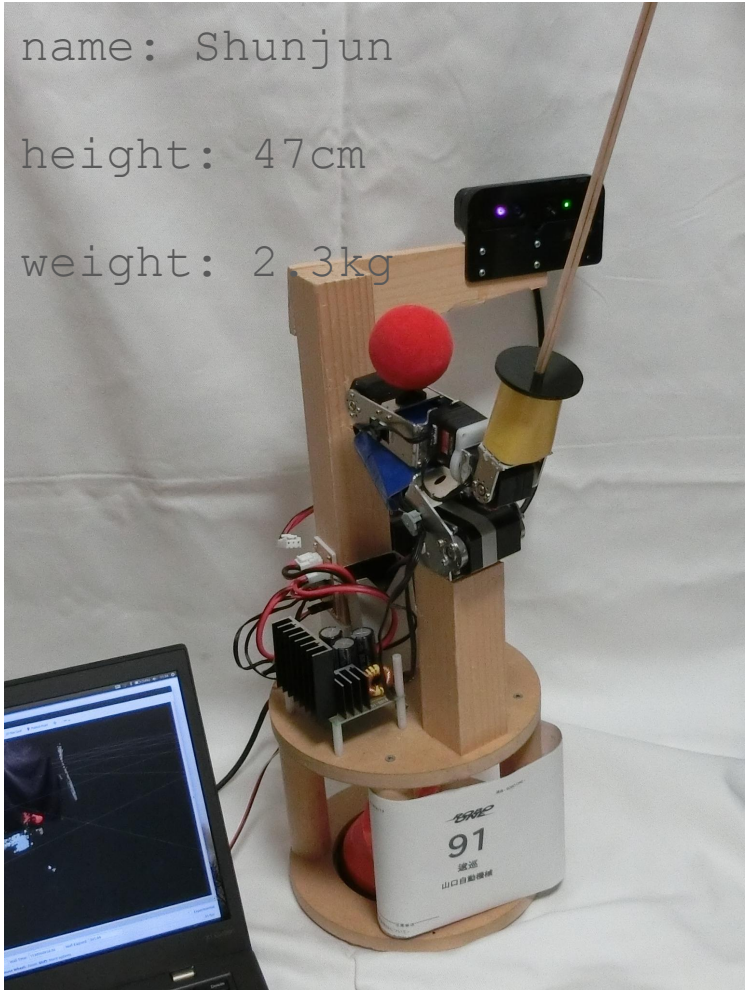
height: 47cm

weight: 2.3kg



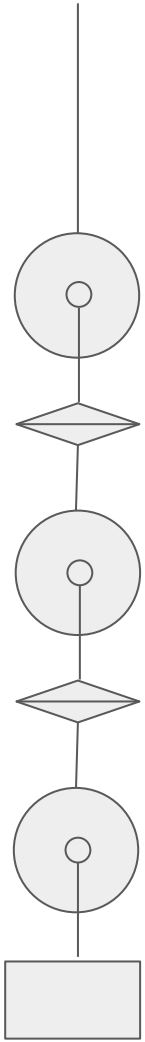
機械

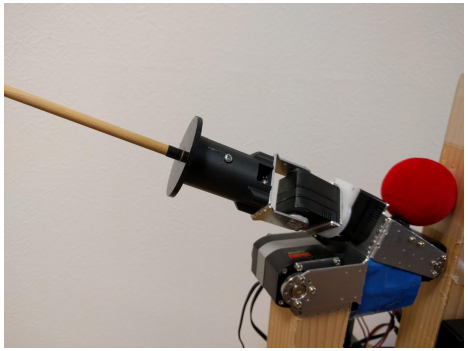
- 台座
  - MDF板12mm
  - 松角材
  - ラミン丸棒
- カメラマウント
  - Softkinetic DS325
  - アームとともに台座に固定
- ５自由度アーム
  - KONDO ロボット用サーボモーター
- 吸盤
  - PROMATEバキュームリフター
  - 耐荷重20kg

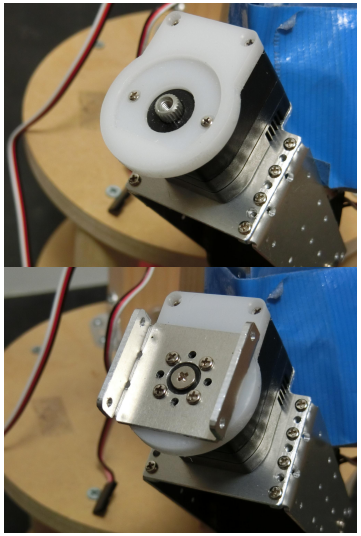


アーム部詳細

- ５自由度
  - 肩ピッチ: KONDO KRS6003RHV
  - 上腕ひねり: KRS4034HV
  - 肘ピッチ: KRS4033HV
  - 下腕(=手首)ひねり: KRS3304 ICS
  - 手首ピッチ: KRS3304 ICS
- 自作ブラケット A5052 T1.5
- 簡易すべりスラスト軸受 (POM)
  - 無限回転を抑制
- 鍔:ABS板
- 柄・面の台:FDM 3Dプリント

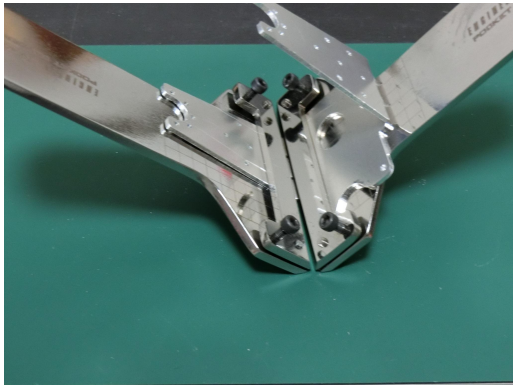




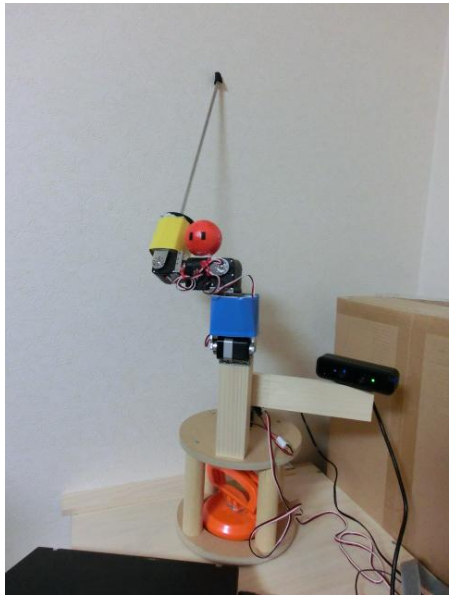


制作風景

台座組み立て中
ホームセンターの
加工サービス利用



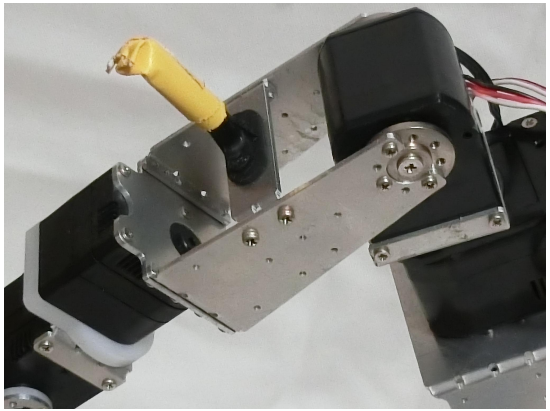
ポケットベンダーによ
る曲げ加工



初期バージョン
市販ブラケット使用
根本側リンクが短い
カメラ視野狭い

各部詳細

固定用吸盤

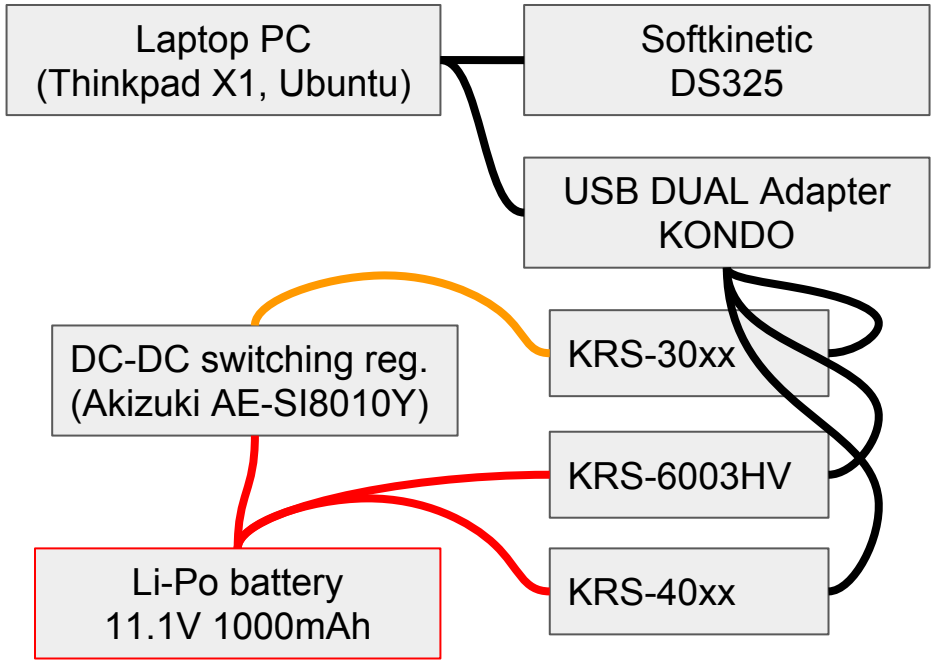


下腕のフレーム構造

カメラマウント
全体ではめ込む構造

Electronics 電装系

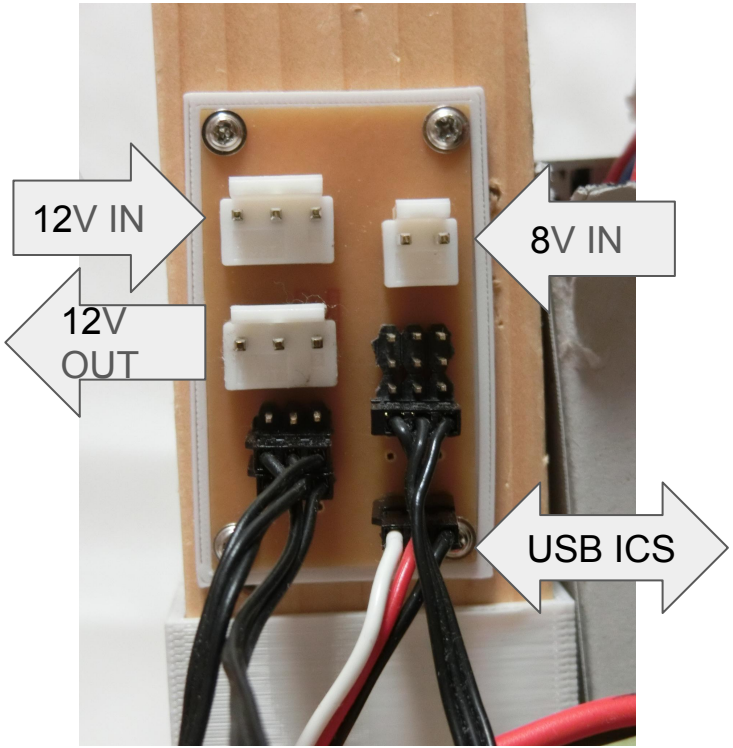
- 制御用PC: Lenovo Thinkpad X1 Carbon
  - CORE i7 vPro
- KONDO DUAL USB アダプター
  - シリアルポートとして動作
  - KONDOのロボット用サーボと通信
  - 1250kbps (KONDOの規格)
- ToF距離画像センサ Softkinetic社 DepthSense DS325
  - color 640x480, depth 320x240
  - USB 2.0
- 配線用ハブ基板
  - 電源と信号線の分配



シンプルにしました

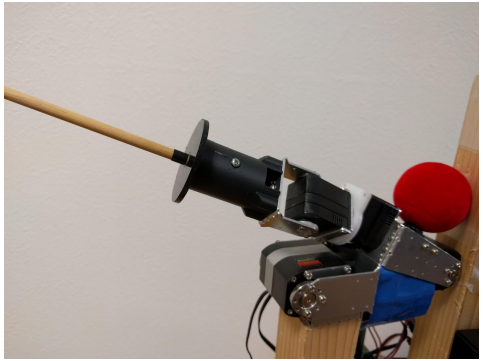
配線ハブ基板

- 信号線と電源を分岐
- デイジーチェーン接続はあまり使わない
  - 根元側の電線を複数サーボが共有
  - QIコネクタの電流容量
  - 瞬間的な大電流で電圧低下
  - 通信失敗が増える

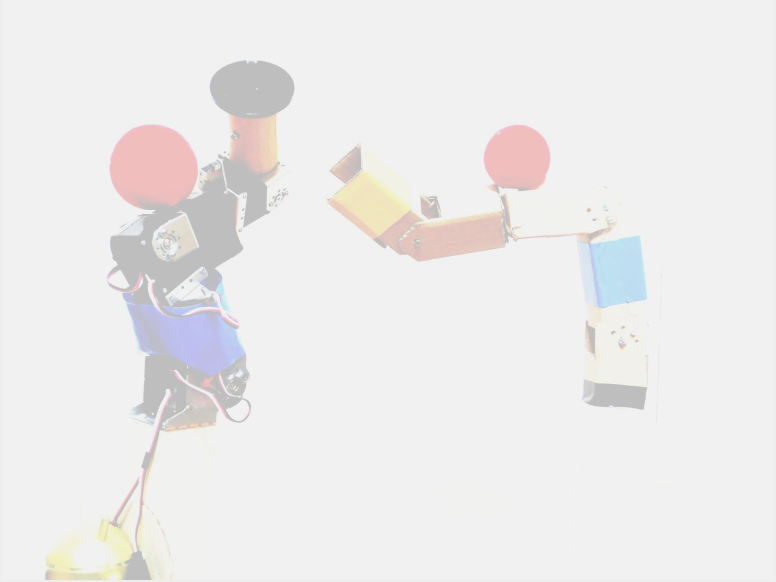


設計方針

- 自動制御前提
  - 軸配置もカメラも、操縦しやすさは考えず
  - 距離画像センサなので、人間が見たとき奥行きがわかりにくいのは気にしない
  - 視界を広くとるため少し後ろに
  - 副作用：相手の小手が鍔に隠れていると見えない＝無駄な攻撃が抑制 (?)
- 剛性高く
  - ピッチ軸は両持ち、ひねり軸は簡易的すべりスラスト軸受
    - 制御しやすい --- 多関節剛体として近似が十分成立する
  - 質量をなるべく中心側に（後に妥協）
- 関節可動域を十分確保
  - 肩: 真後ろまで
  - 肘の折りたたみ:約158度まで。逆向きは面パーツの干渉により制約

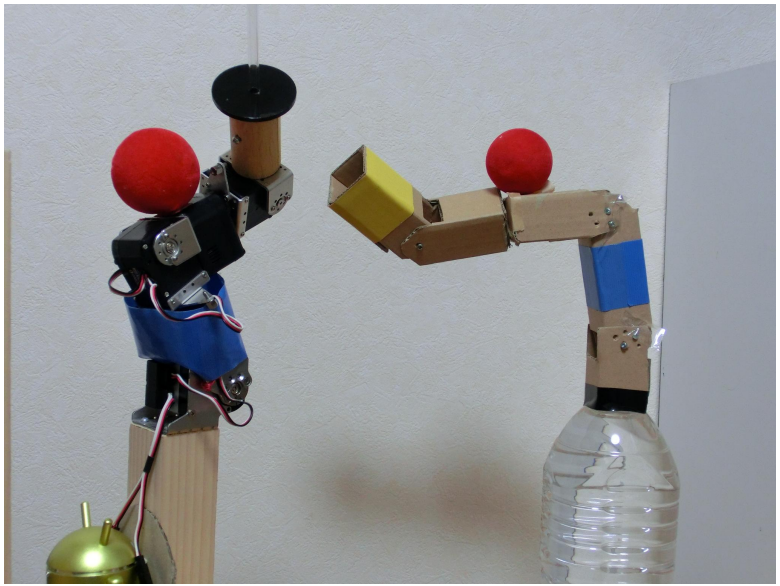


Chapter 2: Algorithm and Strategy



タスクの範囲

- 静止した標的



打突動作とは

- 突き: 竹刀の先端を目標物体に接触させるように動かす
- その他: 竹刀の先から1/3のどこかを目標物体に接触させるように動かす

接触のために要求される精度

- 目安として、正面打ちの場合
  - 左右　±1.5cm
  - 前後　±4cmぐらい (竹刀の長さに依存)
  - 上下　上にずれると当たらない
- 突きの場合
  - 左右　±1.5cm
  - 前後　手前にずれると当たらない
  - 左右　±1.5cm

ROBO-ONE(格闘)よりも数倍、位置の正確さが必要

オープンループ vs クローズドループ

- 打突動作開始後のフィードバックの有無
  - ゴルフ vs ラジコン飛行機
- オープンループ [今回採用]
  - センサー情報を使って動作を計画する
  - 動き出したらセンサーは使わない
- クローズドループ
  - 動作開始後、ずれを計測して修正
  - 処理速度の問題
  - アームの先にセンサーがほしい
  - 画像がブレるかも
  - ロボット自身のアームが見えるので区別する必要

打突タスクの分解

- 1. **標的の検出・位置の認識**
  - センサーで位置を計測できるようになる
- 2. 動作計画
  - **順運動学計算**
    - 関節角度から腕の先端の動きがわかる
  - **アームとセンサーのキャリブレーション**
    - 自分の腕がどこにどう付いているかを知る
  - **逆運動学計算**
    - 接触する瞬間の姿勢を生成する
  - **経路の生成・選択**
    - うまく打突できるような竹刀の振り方を考える
- 3. 動作の実行
  - 角度指令を順次出力する

1. 標的の検出：外界センサー

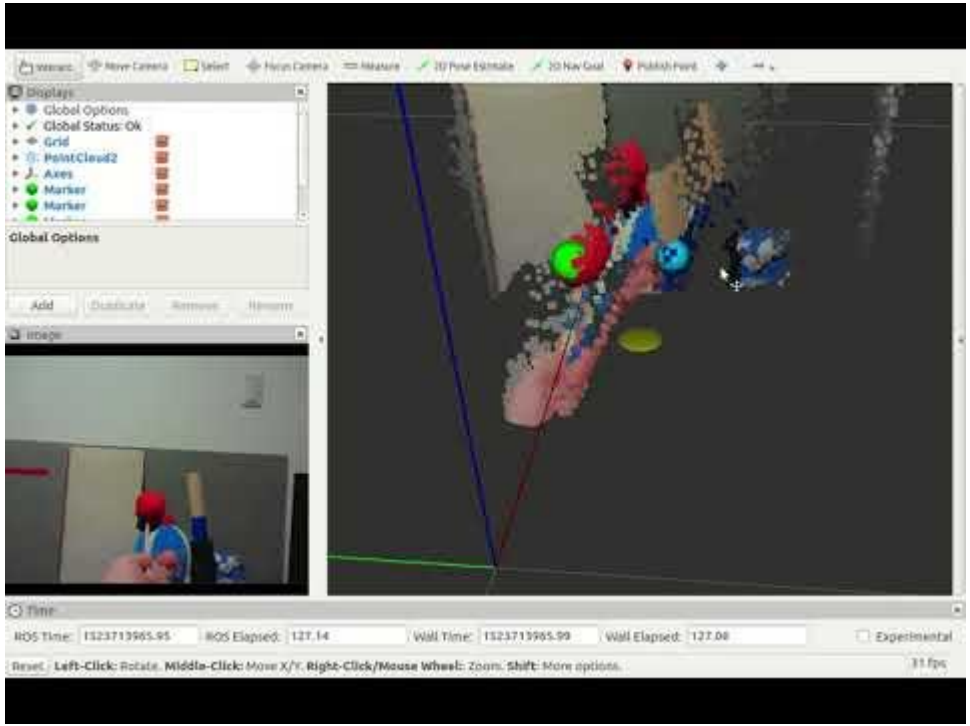
Softkinetic DepthSense DS325　ToF方式距離画像センサ

- 色画像(color): いわゆる「普通の画像」、RGB
- 距離画像(depth): 濃淡、各点でカメラからの遠近を表す
- 確信度画像(confidence): 濃淡、各点で距離がどのぐらい信頼できるかを表す

↓プログラムで処理して得られる情報(ベンダー製ライブラリ等で)

- ポイントクラウド: 3次元空間の点群の位置とその色
  - 1つの点＝(x, y, z, (r, g, b))
  - 距離画像の画素数に対応 (320x240)
  - 確信度の低い点は除外

標的検出のデモ



ポイントクラウドを表示

赤・青・黄色の物体を検出

確認用に球で位置を表示

1. 標的の検出：アルゴリズム

- 0. 前処理
  - ロボットから一定距離以上遠い点を除外（背景など）
- 1. 分節化
  - クラスタリング：点群を塊に分ける
    - 色が似ていて近い点同士を結んで徐々にまとめていく
  - PCLのregion growing segmentation実装を使用
- 2. 標的らしいクラスタを選出
  - HSV色空間(色相・彩度・明度)で色を判別
  - 該当色が60%以上を占めるクラスタを有効打突部位として認識
  - 理想：形状の特徴を見る（未実装） --- 面なら丸い、胴は細長い等
- 3. 標的位置の計算
  - クラスタの重心を求める
  - 理想：球体フィッティング等（未実装）

打突タスクの分解(再掲)

- 1. **標的の検出・位置の認識**
  - センサーで位置を計測できるようになる
- 2. 動作計画
  - **順運動学計算**
    - 関節角度から腕の先端の動きがわかる
  - **アームとセンサーのキャリブレーション**
    - 自分の腕がどこにどう付いているかを知る
  - **逆運動学計算**
    - 接触する瞬間の姿勢を生成する
  - **経路の生成・選択**
    - うまく打突できるような竹刀の振り方を考える
- 3. 動作の実行
  - 角度指令を順次出力する

実機の関節構造と順運動学

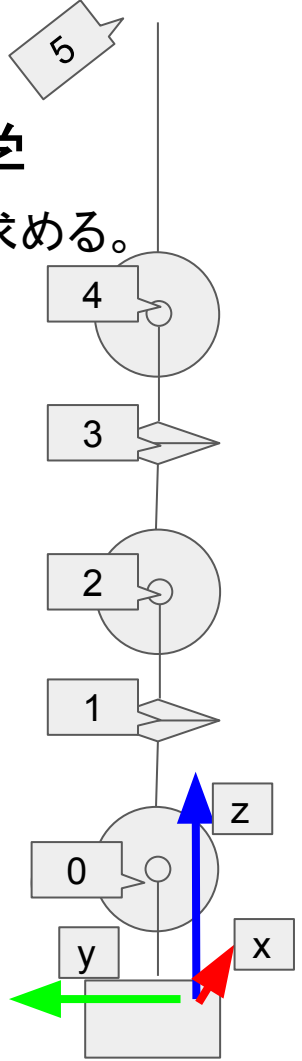
関節角度{θ_0,..,θ_4}から竹刀先端の位置を求める。

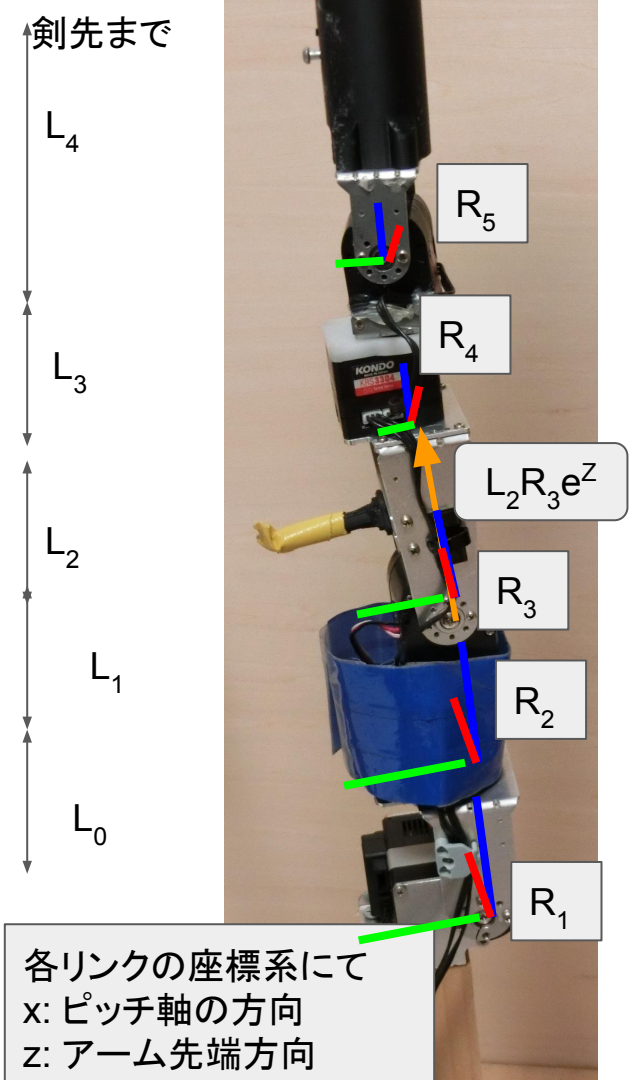
R_n: n番目リンクの姿勢

p_n: n番目のリンク遠心側頂点位置

$$
\begin{aligned}
\boldsymbol{R}_0 &= \boldsymbol{E} \\
p_0 &= \vec{0} \\
\boldsymbol{R}_1 &= \boldsymbol{R}^{\mathrm{X}}(\theta_0)\boldsymbol{R}_0 \\
p_1 &= p_0 + L_0\boldsymbol{R}_1 e^{\mathrm{Z}} \\
\boldsymbol{R}_2 &= \boldsymbol{R}^{\mathrm{Z}}(\theta_1)\boldsymbol{R}_1 \\
p_2 &= p_1 + L_1\boldsymbol{R}_2 e^{\mathrm{Z}} \\
\boldsymbol{R}_3 &= \boldsymbol{R}^{\mathrm{X}}(\theta_2)\boldsymbol{R}_2 \\
p_3 &= p_2 + L_2\boldsymbol{R}_3 e^{\mathrm{Z}} \\
\boldsymbol{R}_4 &= \boldsymbol{R}^{\mathrm{Z}}(\theta_3)\boldsymbol{R}_3 \\
p_4 &= p_3 + L_3\boldsymbol{R}_4 e^{\mathrm{Z}} \\
\boldsymbol{R}_5 &= \boldsymbol{R}^{\mathrm{X}}(\theta_4)\boldsymbol{R}_4 \\
p_5 &= p_4 + L_4\boldsymbol{R}_5 e^{\mathrm{Z}}
\end{aligned}
$$

$$
\begin{aligned}
\vec{0} &= [0,0,0]^{\mathrm{T}} \\
e^{\mathrm{Z}} &= [0,0,1]^{\mathrm{T}} \\
E &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
R^{\mathrm{X}}(\theta) &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix} \\
R^{\mathrm{Z}}(\theta) &= \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}
\end{aligned}
$$





各リンクの座標系にて
x: ピッチ軸の方向
z: アーム先端方向

逆運動学?

- 順運動学計算 (forward kinematics)
  - アーム関節角度から手先の位置を求める
- 逆運動学計算 (inverse kinematics)
  - 手先の位置を希望の場所に持っていくためのアームの関節角度を求める
- 多関節（シリアルリンク）ロボットでは:
  - 順運動学は根本から順番に書いていけば式にできる
  - 逆運動学は必ずしも陽形式に書けない

非線形最小二乗問題、数値計算による反復解法

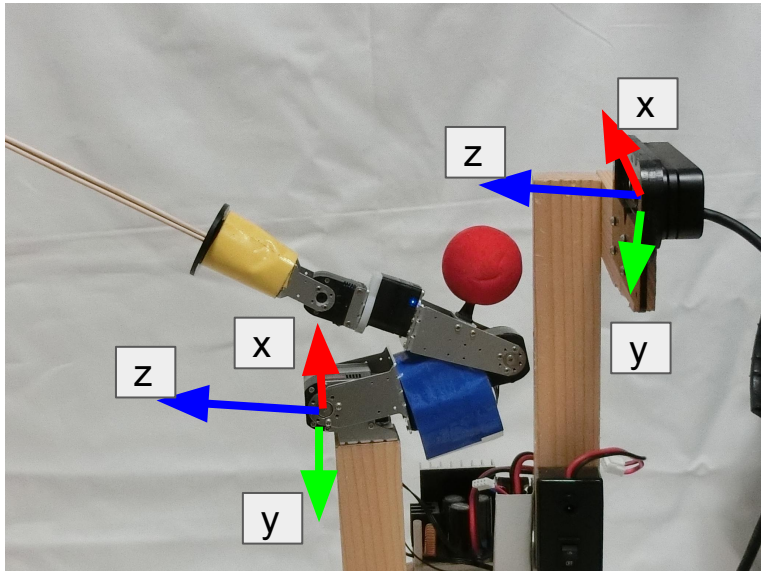
その前に……

打突タスクの分解(再掲)

- 1. **標的の検出・位置の認識**
  - センサーで位置を計測できるようになる
- 2. 動作計画
  - **順運動学計算**
    - 関節角度から腕の先端の動きがわかる
  - **アームとセンサーのキャリブレーション**
    - 自分の腕がどこにどう付いているかを知る
  - **逆運動学計算**
    - 接触する瞬間の姿勢を生成する
  - **経路の生成・選択**
    - うまく打突できるような竹刀の振り方を考える
- 3. 動作の実行
  - 角度指令を順次出力する

座標変換の必要性

- カメラから得られる標的の座標
  - 原点：カメラの光学中心
  - 座標軸の向き：カメラの向き基準
  - スケール：メートル単位、ライブラリから
- 運動学計算による竹刀先端の座標
  - 原点：アームの土台（便宜上）
  - 座標軸の向き：最初の回転軸基準
  - スケール：実測した各リンクの長さに基づく



⇒どちらかに統一する変換が必要
ここでは、**カメラの座標系**で表すよう統一
並進と回転、拡大縮小の形

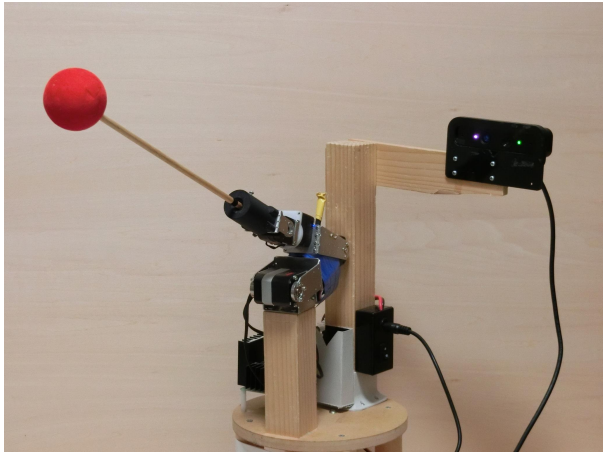
$$f(x) = SRx + t$$

$$S = \mathrm{diag}(1, s_Y, s_Z) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & s_Y & 0 \\ 0 & 0 & s_Z \end{pmatrix}$$

変換パラメータはどう決める？

座標変換パラメータのキャリブレーション

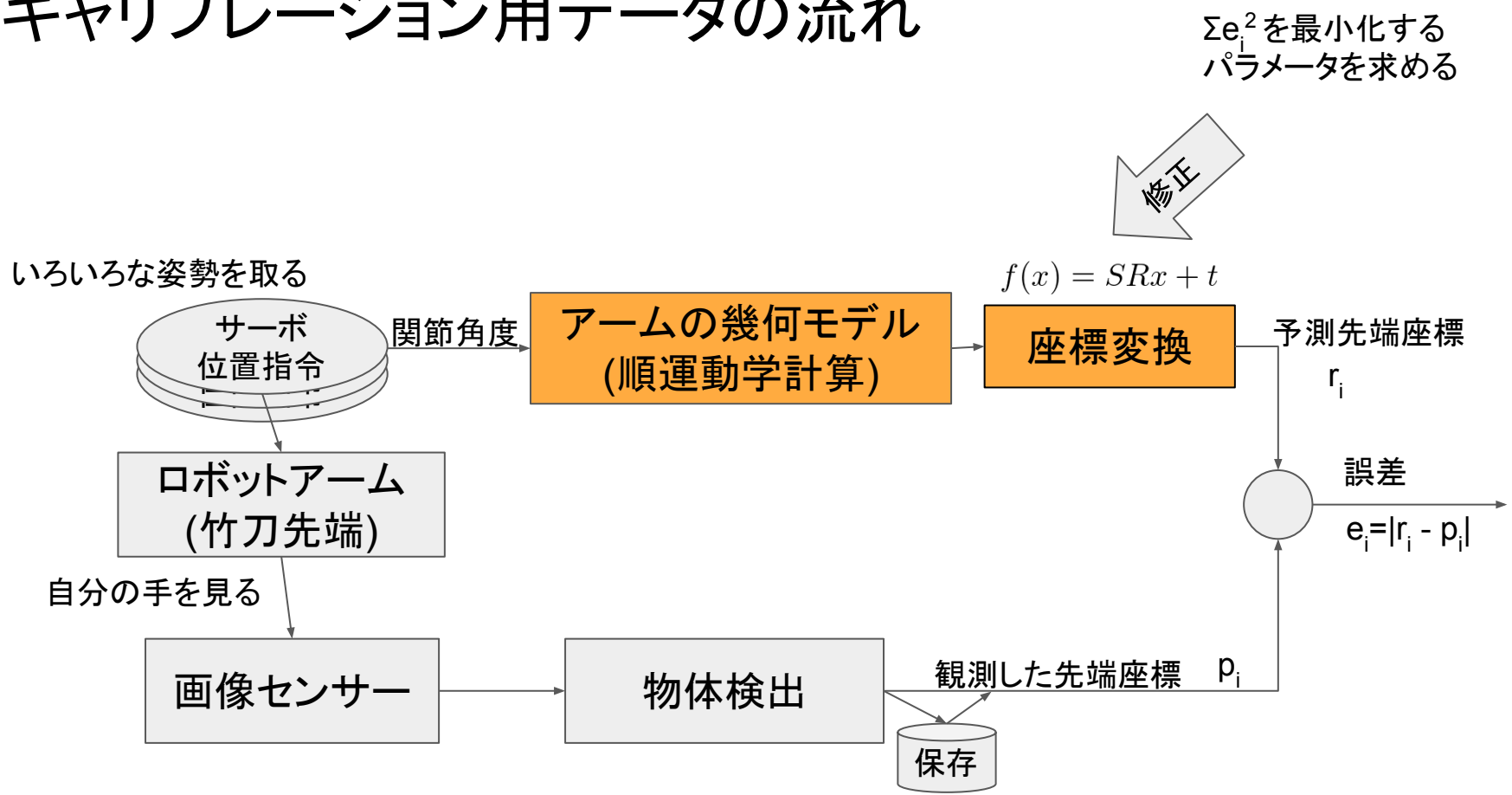
- 定義: 実測と計算によりパラメータを決定
- カメラ取り付け後、運用前に
- 竹刀と同じ長さの棒、先端にマーカーを固定
- 色々なポーズをとりながら観測したマーカー位置と関節角度を記録
- 約50点をサンプリング
- 最も「辻褄の合う」パラメータを計算
- 変換後誤差二乗和を最小化
  - 非線形最小二乗
  - Levenberg-Marquardt法
  - cminpackを利用



実測データの例

竹刀先端座標[m]			関節角度[rad]				
x	y	z	θ0	θ1	θ2	θ3	θ4
0.065294	0.018595	0.47221	-2.6784	-0.14019	2.7167	-0.10838	-0.21913
0.051924	0.14596	0.48924	-2.58	-0.1343	2.7055	-0.10838	0.12134
0.036495	0.083223	0.47985	-2.6289	-0.094837	2.6713	1.2199	-0.054782
-0.055083	0.076712	0.43744	-2.6289	-0.18496	2.666	1.3725	-0.30454
0.090883	0.068661	0.49088	-2.6784	-0.22443	2.6319	1.3725	0.24799
0.10284	0.11132	0.49518	-2.6672	-0.15669	2.6378	0.85059	0.26507
-0.011012	0.0010318	0.44008	-2.6784	-0.10073	2.649	0.85059	-0.29276
0.11954	-0.036163	0.46815	-2.4157	-0.094837	2.6266	-0.35402	-0.63853
				…			
				…			
			以下、観測した姿勢の数だけ続く（約 50）				

キャリブレーション用データの流れ



更に誤差を減らす工夫

- 機械系のモデルパラメータも一緒に最適化
  - 各関節の角度原点オフセット（ラジコンでいうところの「トリム」）　×4自由度
  - 各リンクの長さ　×3自由度；上腕、下腕、竹刀
- まとめると、以下の全ての変数を最適化
  - 回転 3
  - 並進 3
  - y,z軸方向への拡大縮小 ２ (１自由度分はリンク長の等倍と等価)
  - リンク長 3
  - トリム ４ (最初の軸は回転と等価)

キャリブレーション結果の例

初期値 (あまり重要でない):
S,R=単位行列, t=零ベクトル
L_x=定規で計測したアーム長さ

Levenberg-Marquardt法

(回転は、計算中は回転ベクトルで表現)

初期平均誤差:205mm

最終平均誤差:8.24mm

結果

SR =
0.9793281 0.0655680 0.1913563
-0.0664952 1.0027907 -0.0032941
-0.1914953 -0.0094680 0.9832837

t' =
133.55 161.49 169.74

上腕(L_0+L_1): 117 → 124.35 (+7.735)
下腕(L_2+L_3): 121 → 123.45 (+2.448)
手首-剣先(L_4): 337 → 311.703 (-25.297)

軸原点オフセット [rad]
0.00000, 0.03636, -0.02826, 0.02369, -0.05309

カメラから見た
アームの正面向き
（Z軸）

回転とスケール
約10度鉛直軸回り

カメラから見た
アームの原点位置

ロボットの形状も
補正

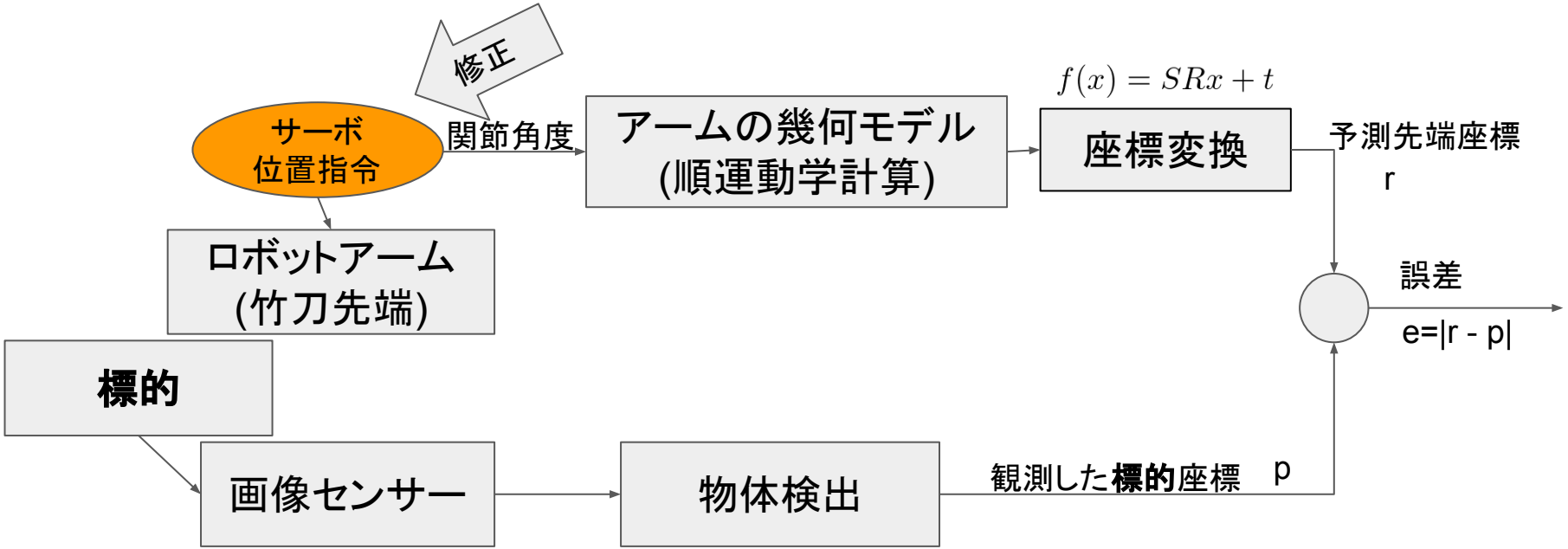
最大で2.8度
補正

打突タスクの分解(再掲)

- 1. **標的の検出・位置の認識**
  - センサーで位置を計測できるようになる
- 2. 動作計画
  - **順運動学計算**
    - 関節角度から腕の先端の動きがわかる
  - **アームとセンサーのキャリブレーション**
    - 自分の腕がどこにどう付いているかを知る
  - **逆運動学計算**
    - 接触する瞬間の姿勢を生成する
  - **経路の生成・選択**
    - うまく打突できるような竹刀の振り方を考える
- 3. 動作の実行
  - 角度指令を順次出力する

逆運動学計算(キャリブレーションと比較した図)

順運動学計算ができれば、反復計算により逆運動学計算ができる



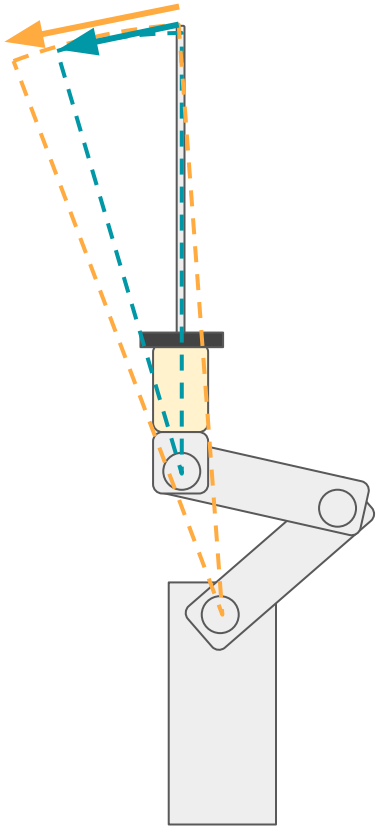
逆運動学計算

- 打撃完了姿勢を生成するには？
- 接触している：竹刀と標的の距離が0
- 竹刀と標的の距離は関節角度(5次元)の関数
- これも非線形最小二乗問題
- 竹刀先端と標的が近付く方向に姿勢を少しずつ変える(最急降下法等)
- 問題：根本の軸の角度の微係数が大きいので先に動く
- 手首関節と標的の距離が竹刀の長さに一致するとき最小となる関数をコストに加算

関節による微係数の違い

関数
$f(\theta_0, \ldots, \theta_4)$=(竹刀先端と標的のユークリッド距離)
を最小化

同じ回転角なら中心側関節の方が竹刀
先端を大きく動かす
↓
微係数の絶対値が大きい
↓
そのため根本関節が先に動いて
つんのめる形になりがち

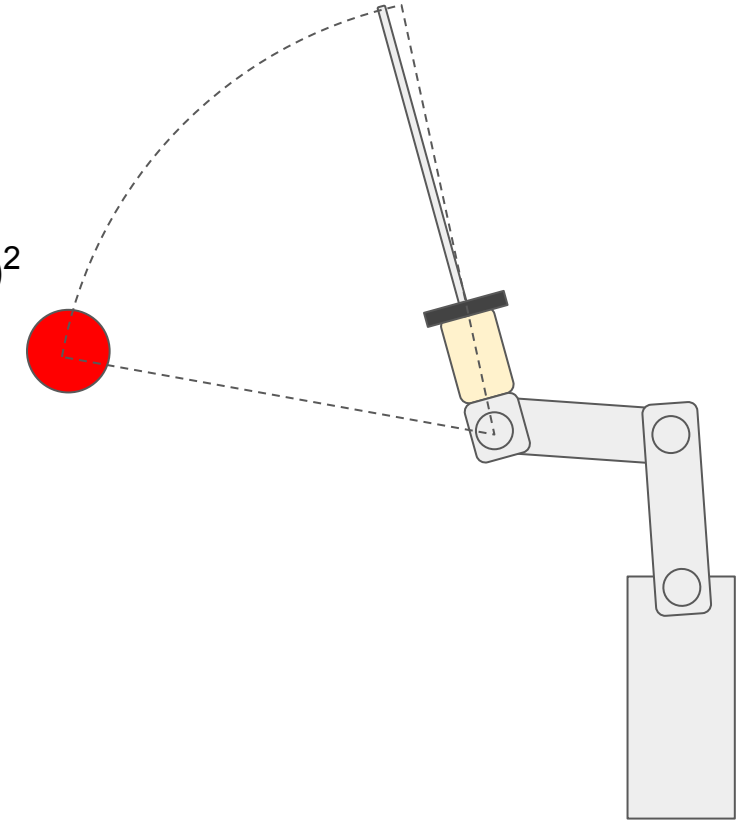


らしい動きにするための工夫

関数
$f(\theta_0, \ldots, \theta_4)=$
(竹刀先端と標的のユークリッド距離)
$+ \alpha \cdot$(手首と標的のユークリッド距離 $- L_4)^2$
を最小化

手首関節と標的との距離を適切に保つ
項を導入

α: 適当な係数



打突タスクの分解(再掲)

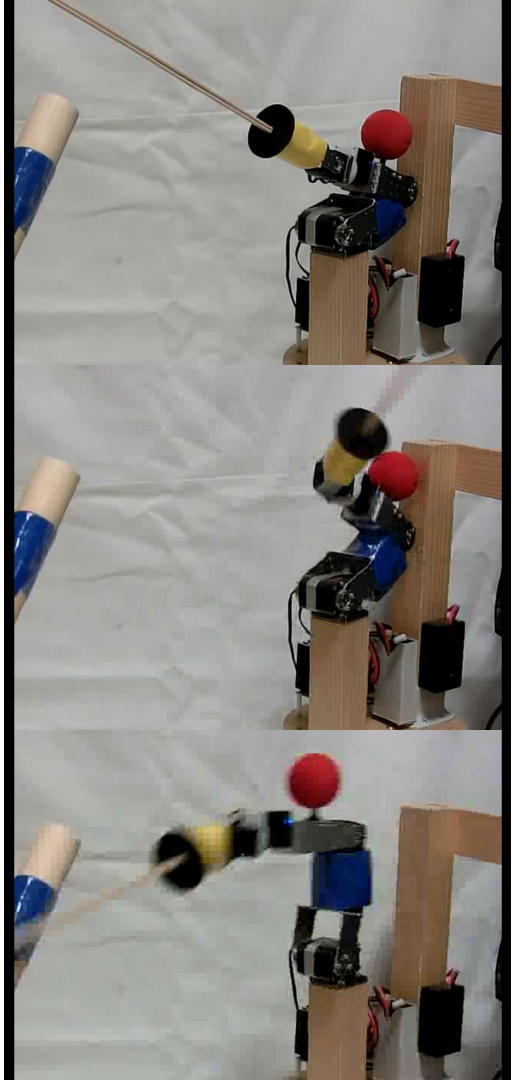
- 1. **標的の検出・位置の認識**
  - センサーで位置を計測できるようになる
- 2. 動作計画
  - **順運動学計算**
    - 関節角度から腕の先端の動きがわかる
  - **アームとセンサーのキャリブレーション**
    - 自分の腕がどこにどう付いているかを知る
  - **逆運動学計算**
    - 接触する瞬間の姿勢を生成する
  - **経路の生成・選択**
    - うまく打突できるような竹刀の振り方を考える
- 3. 動作の実行
  - 角度指令を順次出力する

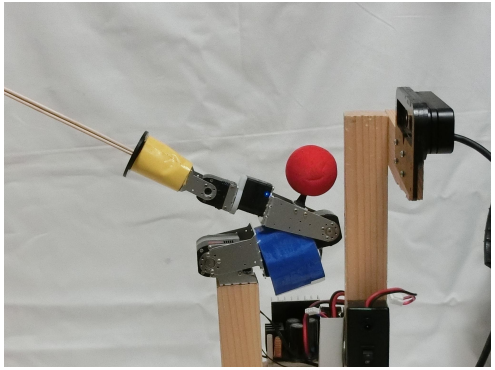
経由姿勢(waypoint)の設定

- 正面、左右45度・90度に振りかぶる姿勢を用意
- 構え→経由姿勢→打撃完了姿勢→経由姿勢→構え

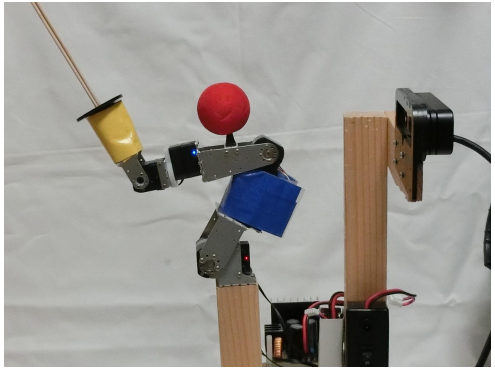
何のため？

- 1. 打撃らしく振りかぶるため
  - 中段の構えから直接打撃完了姿勢に行くと「突き」に近い
- 2. 反復逆運動学計算の都合
  - 関節可動域には限界がある
  - 局所最適解に陥ってしまうケース
  - 初期値として使用、誤差が小さく収束したものを選択
- 3. 障害物回避のため
  - ヒューリスティックス
  - 胴、小手の場合必ず横や斜めに振りかぶる姿勢を採用

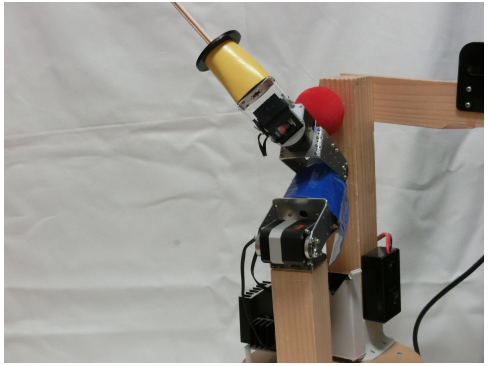




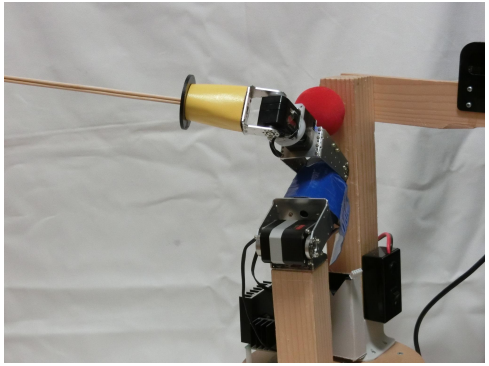
ホームポジション(中段)



正面打ち前の経由姿勢



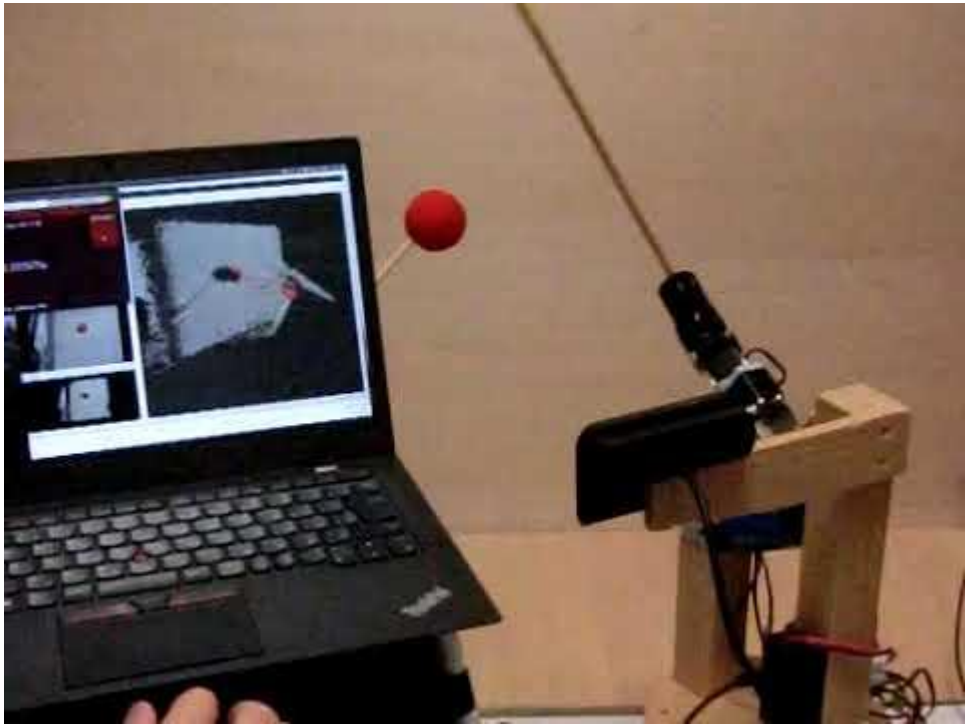
45°(左右面あるいは小手)



90°(胴)

自動打撃のデモ

- 「面を打つ」指示



試合の戦略 / アルゴリズム

- ROBO-剣のルール
  - 同じ技の連続攻撃禁止 (3秒以内)
  - 異なる技で連続攻撃最大 3回まで、その後 3秒攻撃禁止

1. 打っても良い箇所の集合を求める
   $a \subseteq \{x \in \{小手、面、胴\}\}$
2. センサーで発見した標的の集合bを求める
3. a∩bからランダムに選んで打つ → 経路計画へ
4. 打った箇所と時間を記録、連続技回数を加算
5. 「中段」の構えに戻る

その他、気づいたこと

- 対象位置検出精度は重要
  - 昨年より精度・信頼性向上
  - キャリブレーションが自動で精度よくできるようになった
- 止まっている相手にしか当たらない
  - 相手のモーション開始後射程に入る →1秒ぐらいかかって到達 →もうその場所にはいない
- 隙のないホームポジションによる防御は有効
- 相手が構えていると遠くてどこも打てないことがある

今後の拡張

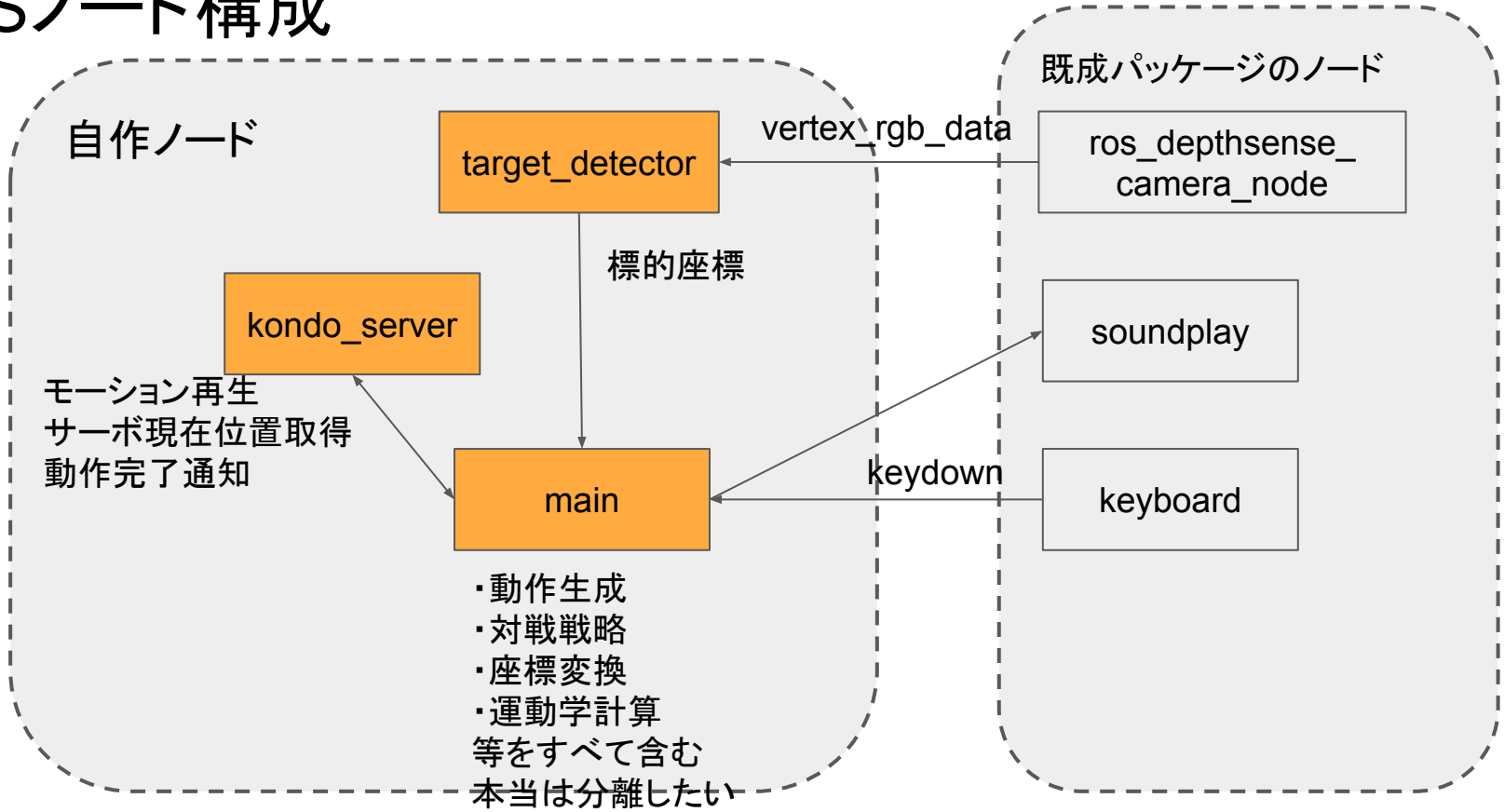
- 動く標的への対応
  - 標的の移動を多項式で近似
  - 自分が振り下ろすのにかかる時間を考慮、未来の位置を狙う
- より賢い障害物回避
  - 標的以外の障害物（相手の竹刀、フレーム、鍔など）も見えているので …
  - 1. 振り下ろす軌道上にないことを確認してから打つ
  - 2. 迷路探索のように障害物を回避する軌道を生成する

Chapter 3: Software System Integration

ソフトウェアシステム構成

- Linux (Ubuntu 16.04 LTS)
- ROS Lunar
- C++

ROSノード構成


既成パッケージのノード
自作ノード
target_detector
vertex_rgb_data
ros_depthsense_
camera_node
標的座標
kondo_server
soundplay
モーション再生
サーボ現在位置取得
動作完了通知
main
keydown
keyboard
・動作生成
・対戦戦略
・座標変換
・運動学計算
等をすべて含む
本当は分離したい

利用したライブラリ類

- ROSツール
  - RViz
- ROSノード
  - ros_depthsense_camera: DS325のドライバ・データ取得
  - sound_play: 音声ファイル再生
  - keyboard: 手動操作用 （開始・終了・脱力・デバッグ用動作）
- ライブラリ
  - pcl: ポイントクラウド操作
  - cminpack: Levenberg-Marquardt法による非線形最適化
  - gflags: コマンドラインオプションの処理
  - protobuf: Protocol Buffer、構造化データクラス生成とシリアライズ・デシリアライズ
    - （ディスクへの保存に）

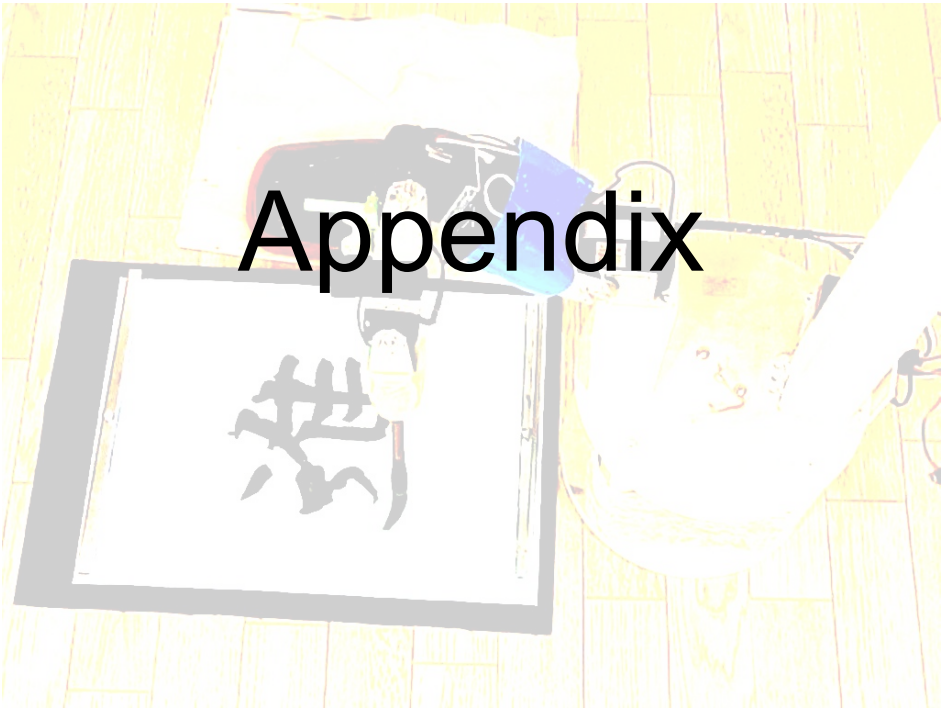
補足

- 作業時間：およそ4日
  - 使わなかったコードや実験を除く
  - 昨年までのシステムをROSへ移植
  - ポイントクラウドからの物体検出を新規実装
  - 大会当日早朝に初めて全機能が揃った
- MoveIt! は未使用
  - 多関節アームの動作プランニングを行える ROSのライブラリ
  - 既に昨年までに作っていたプログラムを移植したため使用せず
  - アームの寸法等をキャリブレーションするのは入門書の範囲を超えると予想
  - 一応、control_msgs/FollowJointTrajectgoryGoalをメッセージ型に採用

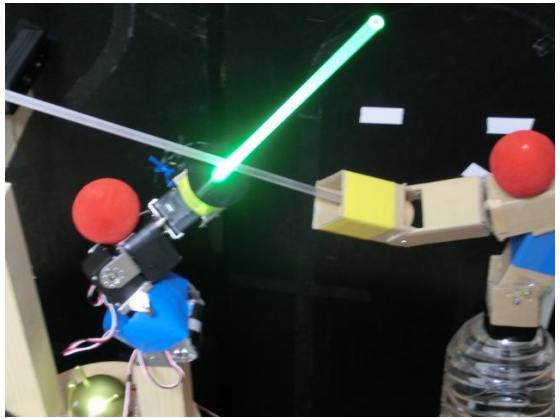
ソフトウェアの一部公開

- GitHubにてLinux用のKONDO ICS通信ライブラリを公開
- https://github.com/yamaguchi-am/kondo-ics
- PCから直接KONDOのロボット用サーボを制御する
- (これをもとにROSノード作成した)
- 三種類のボーレートに対応(115200bps, 625000bps, 1.25Mbps)
- ICS 3.5に対応
- ID以外のEEPROM変更操作には未対応(開発中)
- ICS 3.6で増えたコマンドには未対応

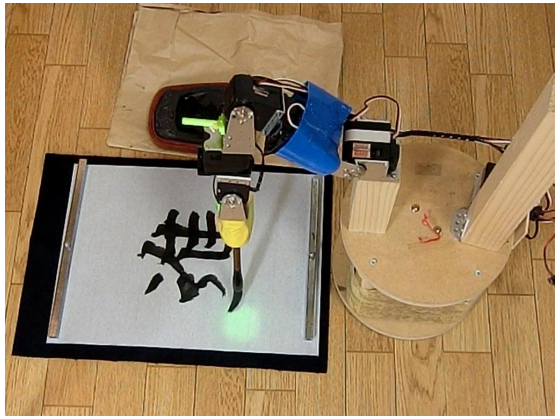
Appendix



大会以外でのロボットの活用



柄内部にパワーLED
透明樹脂の竹刀を光らせる



一発芸の書道
時系列の直接教示を再生

ハロウィンの仮装
時系列の直接教示を再生

名前の由来

逡巡(しゅんじゅん)
ためらい。
また、10^{-14}等の小さい数を表す語としても使われる。(塵劫記)

類似の小数を表す語で「瞬息」「弾指」「刹那」は極めて短い時間の意味がある。

c.f. 阿伽羅（あから）：10の224乗
「あから2010」 情報処理学会のコンピュータ将棋プログラムの名に採用された

別のアプローチで

手先のカメラ
フィードバック制御のみ