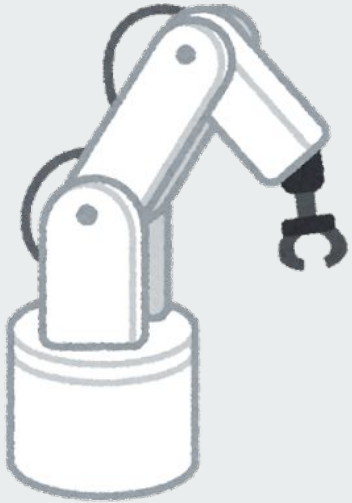
居合斬りロボットの開発



4班　山崎雄介
吉越誠

目次

- 動機
- ロボットの動作
- ソフトウェアの構成
- 開発のスケジュール

動機

イメージ：ロボットは、**硬い・情が湧かない・冷たい**

→ ロボットの動きで、ある程度は克服できるのではないか？

→生物のような、愛嬌のある動きを実現したい

動機

なぜ居合斬りなのか？

→ **栄える**動きだから。基本的にイレギュラーが発生しない動きだからこそ、イレギュラーが　発生した時の動きを目立たせることができると考えた。

どう愛嬌を出していくか？

→ 居合が**失敗**する場合を作り、失敗した時の動きに生物味を出す。　対象物に対して**好き嫌い**を持たせ、感情があるような動きをさせる。

環境・使用するもの

開発環境

・Ubuntu20.04.3 LTS

・ROS-ver:noetic

・Gazebo-ver:11.5.1

・Rviz-ver:1.14.9(noetic)

使用するもの

・Crane_x7　一台

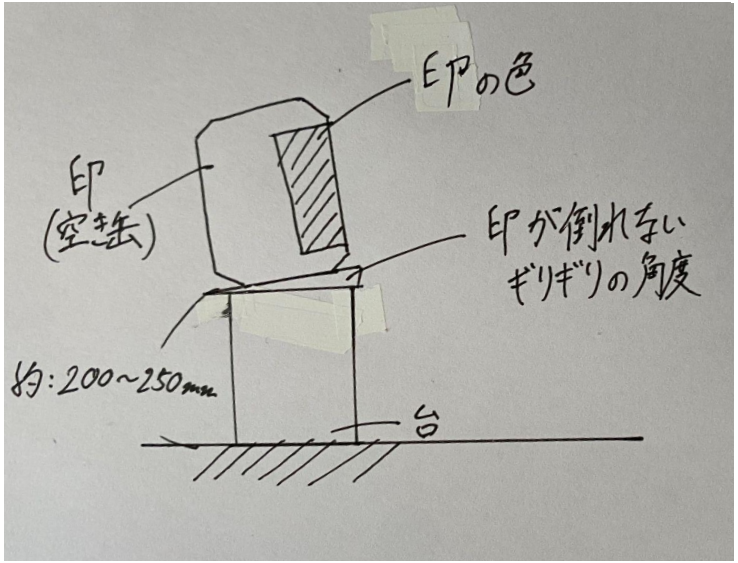
・RealSense　一台

・刀　一本(円柱、プラスチックor樹脂製)
(寸法：長さ300mm、直径45mm±5mm)

・印　任意個

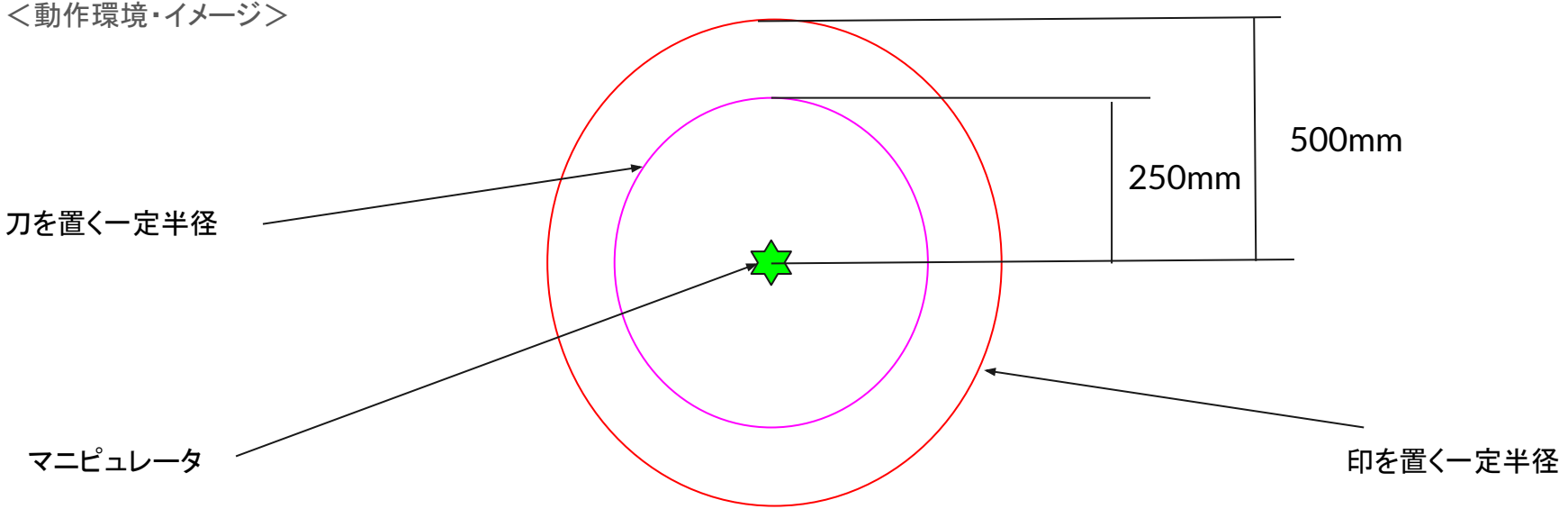
環境・使用するもの

＜印のデザイン・イメージ＞



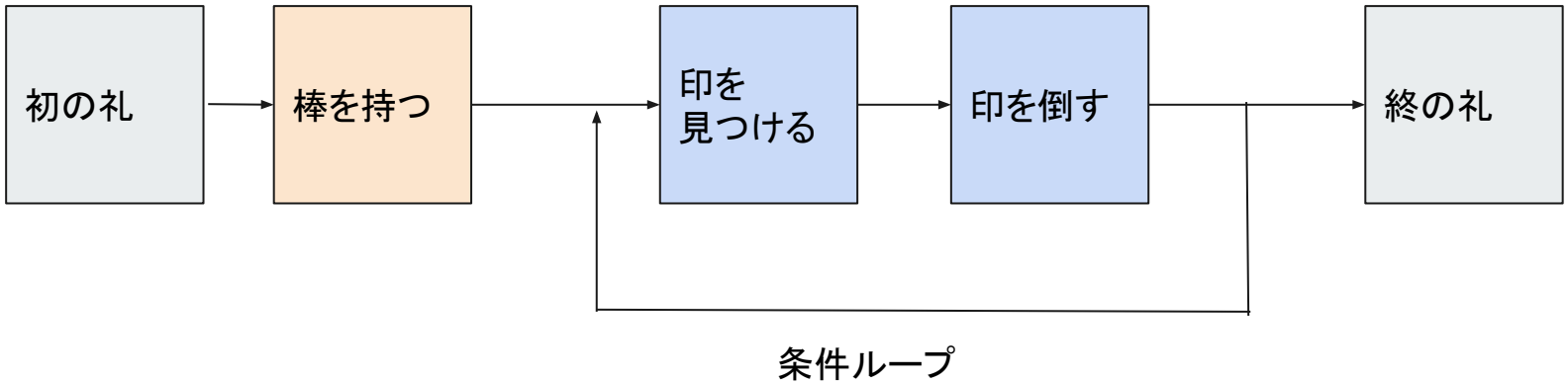
環境・使用するもの

＜動作環境・イメージ＞



ロボットの動作

＜全体の流れ＞



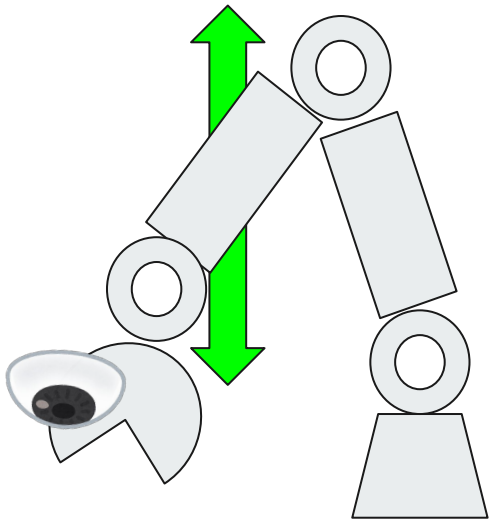
ロボットの動作(詳細)

1. 一礼する

礼儀として、一礼させる。

＊愛嬌ポイント

人間と同じように、礼に始まる。



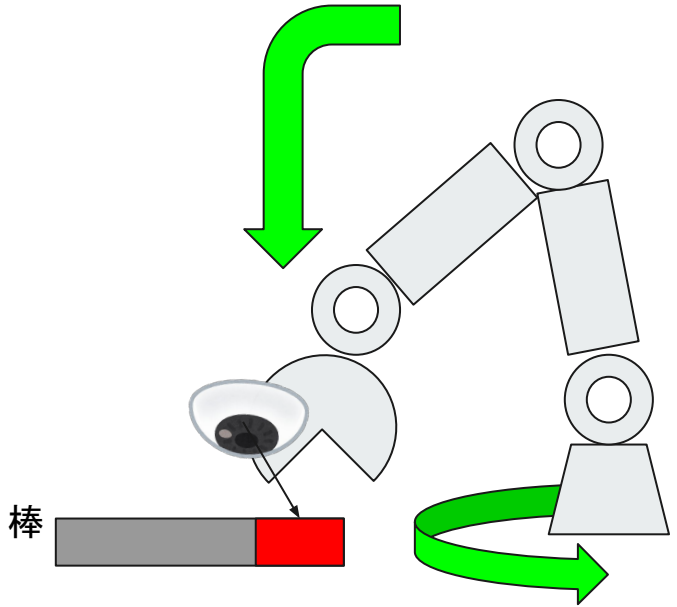
ロボットの動作(詳細)

2. 周辺に置かれている刀(棒)をつかみ、構えの姿勢に入る

一定半径上にカメラが来る姿勢にする。

その体制で周囲を確認し、棒を見つけて取る。

（続く）



ロボットの動作(詳細)

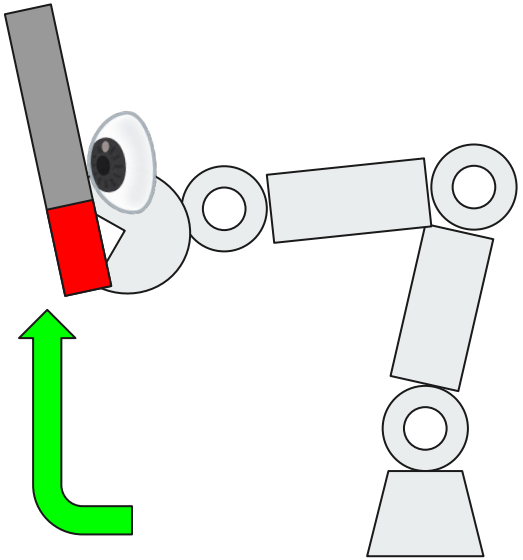
2. 周辺に置かれている刀(棒)をつかみ、構えの姿勢に入る

（続き）

棒を取ったら、構えの姿勢をとる。

＊愛嬌ポイント

見つけた時に**喜ぶ**。

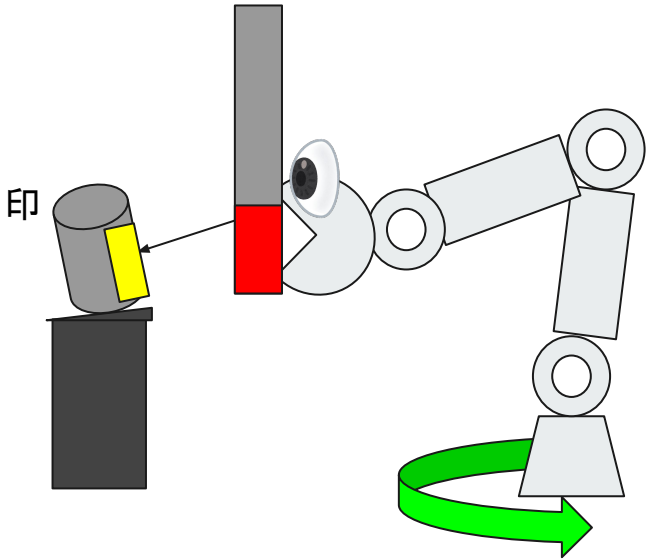


ロボットの動作(詳細)

3. 指定された色の印を探す。

カメラに印の色が映る姿勢を取り、周囲を探す。

見つけ次第、位置調整を行う。



＊愛嬌ポイント

特定の色の印を見つけると、**嫌がる**。

見つけられなかったら**首をかしげて**、次の色を探す。

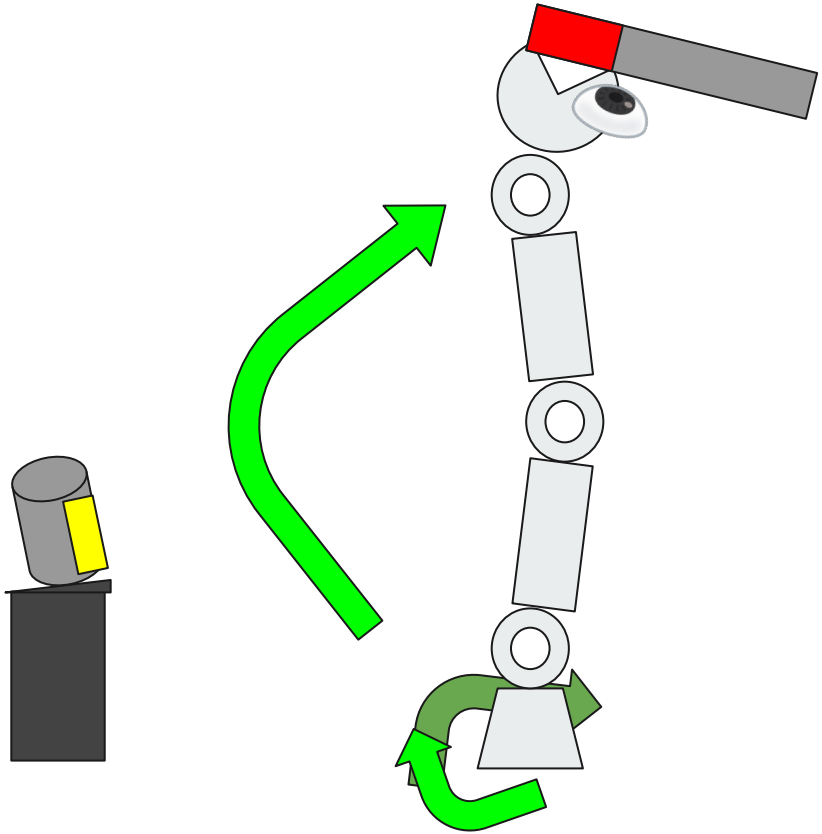
ロボットの動作(詳細)

4. 印を打つ

位置調整終了後、棒を”ゆっくり”持ち上げる。

固定座標系のz軸を基準に少し右回転する。

（続く）



ロボットの動作(詳細)

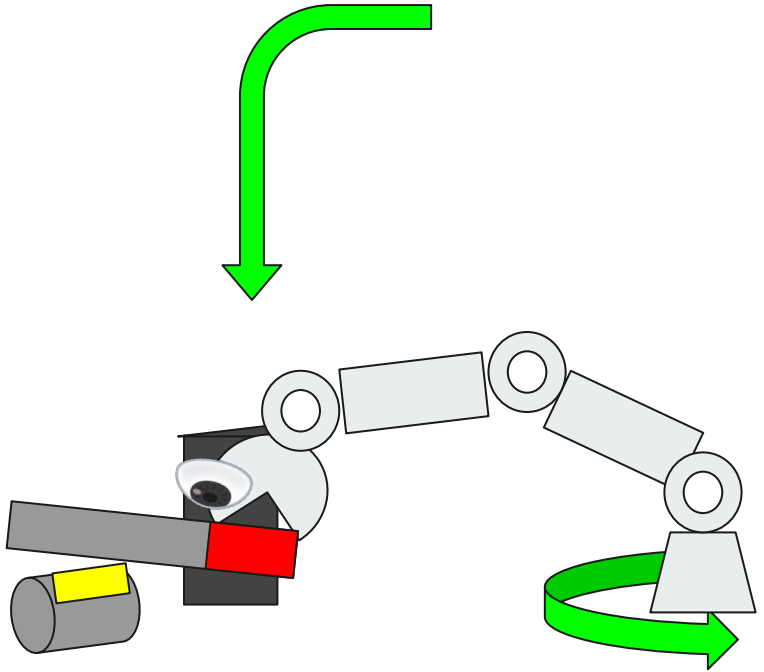
4. 印を打つ

（続き）

棒を上げ終わったら”素早く”、回転しながら印を打つ。

打ち終わった時にハンドが台の左側に来るようにする。

印を倒せない動きをすることもある。



ロボットの動作(詳細)

5. 印が倒れたかどうか確認する。

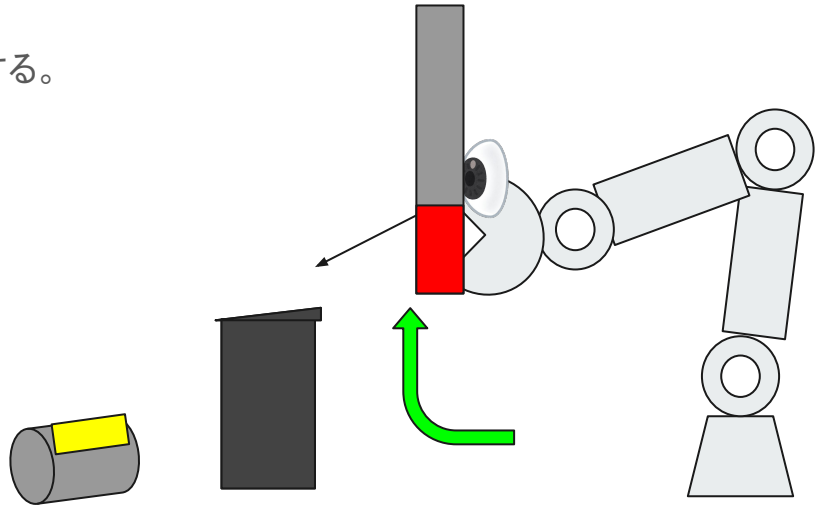
探すときの姿勢に移動し、印があるかどうかを判別する。

倒れている → 次の色を持つ印を探す。

倒れていない → 位置調整からやり直す。

＊愛嬌ポイント

倒れていなかったら**首をかしげる**。



ロボットの動作(詳細)

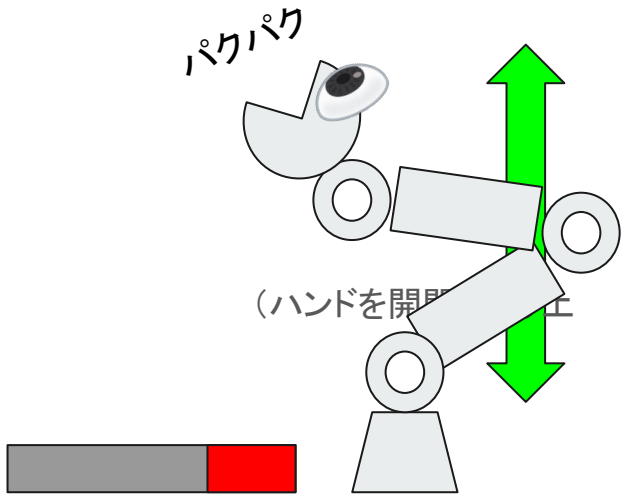
6. すべての印を倒し終えたら棒を置き、一礼をして、喜ぶ

指定される色を全て処理し終えたら、棒を置く。

一礼をして、喜びを表わす。

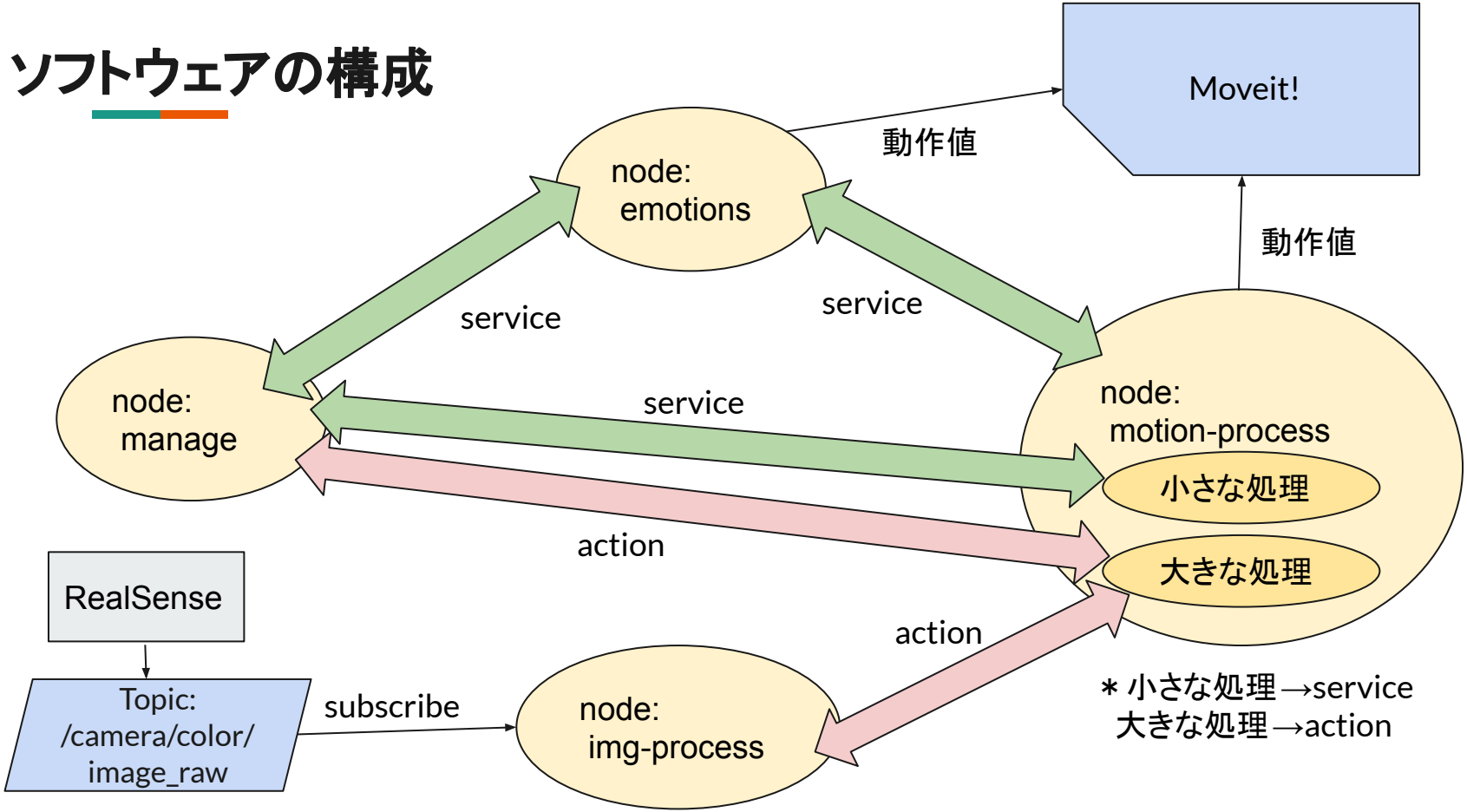
＊愛嬌ポイント

人間が見て**喜んでいる**ことが分かる動きをする。
下に動いたり）



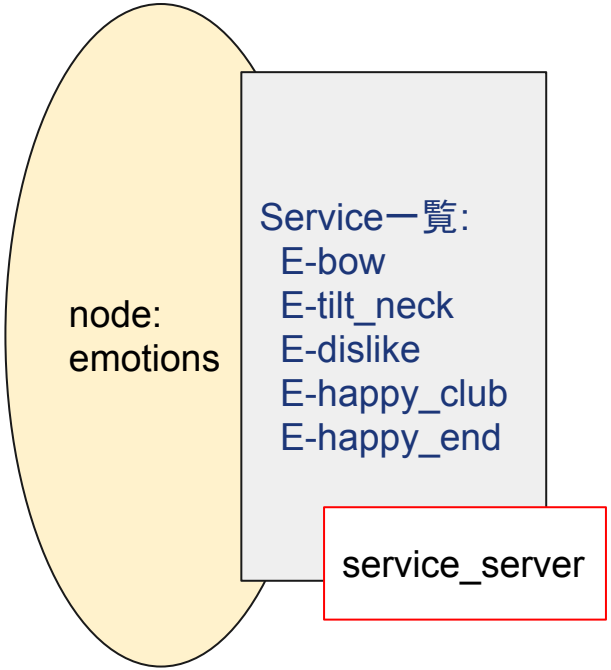
（ハンドを開閉

ソフトウェアの構成


Moveit!
動作値
node:
emotions
動作値
service
service
node:
manage
service
node:
motion-process
小さな処理
大きな処理
action
RealSense
action
Topic:
/camera/color/
image_raw
subscribe
node:
img-process
＊小さな処理→service
大きな処理→action

各nodeの構成

＜emotionsの構成＞



- E-bow：一礼
- E-tilt_neck：首を傾げる
- E-dislike：嫌がる
- E-happy_club：喜ぶ（棒を発見時）
- E-happy_end：喜ぶ（すべて終わった時）

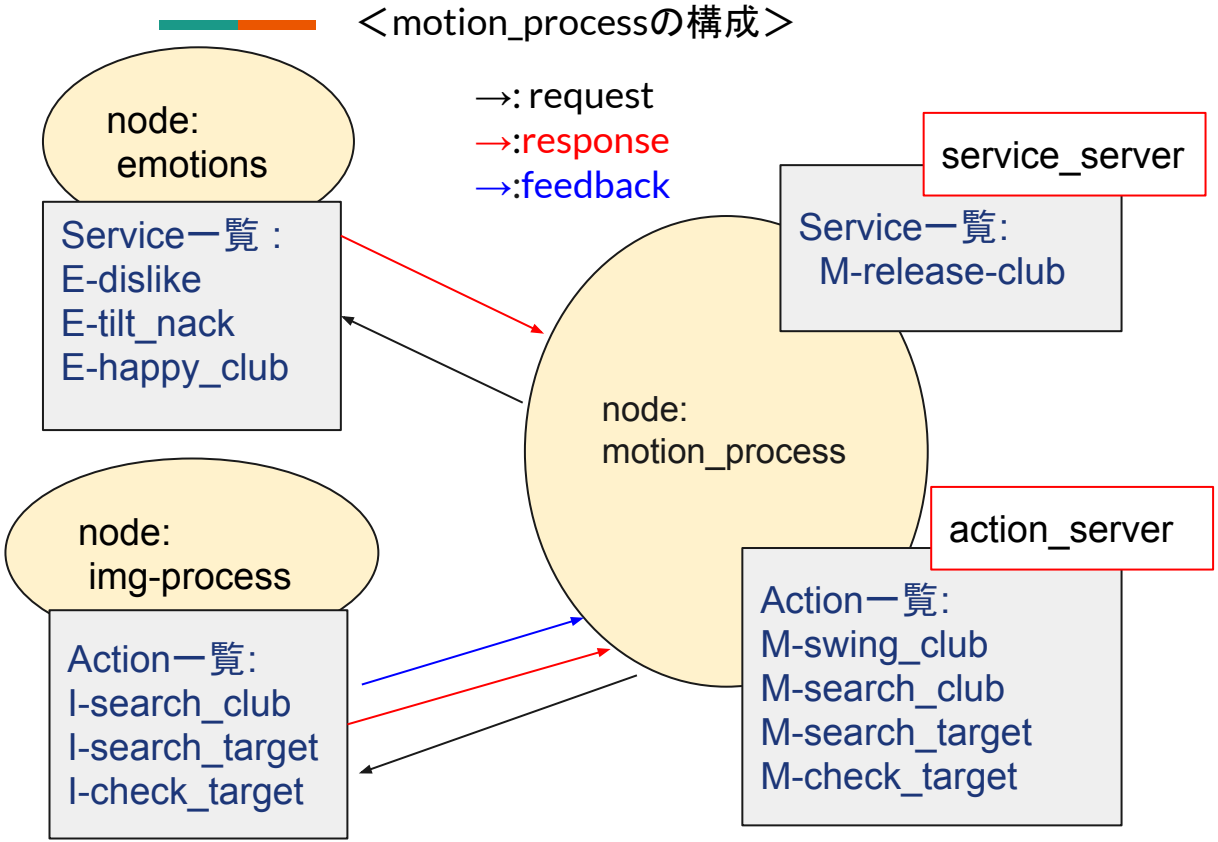
感情表現の動作を行う。

すべての動作はSRDFに記述

serviceの入出力内容は同一

（入力：動作命令、出力：完了報告）

各nodeの構成



必要な動作を行う。

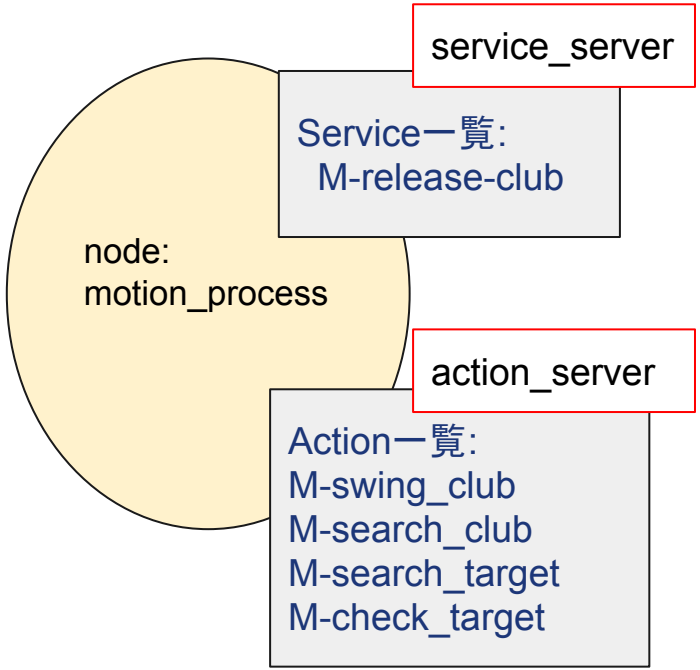
比較的大きな処理はactionで通信する。

画像処理を伴う動作はimg-processのaction_clientになる。

感情表現が必要な場合は、emotionsからのserviceを受ける。

各nodeの構成

＜motion_processの構成＞



service:

- M-release-club: 棒を離す
  入力:動作命令, 出力:完了報告

action:

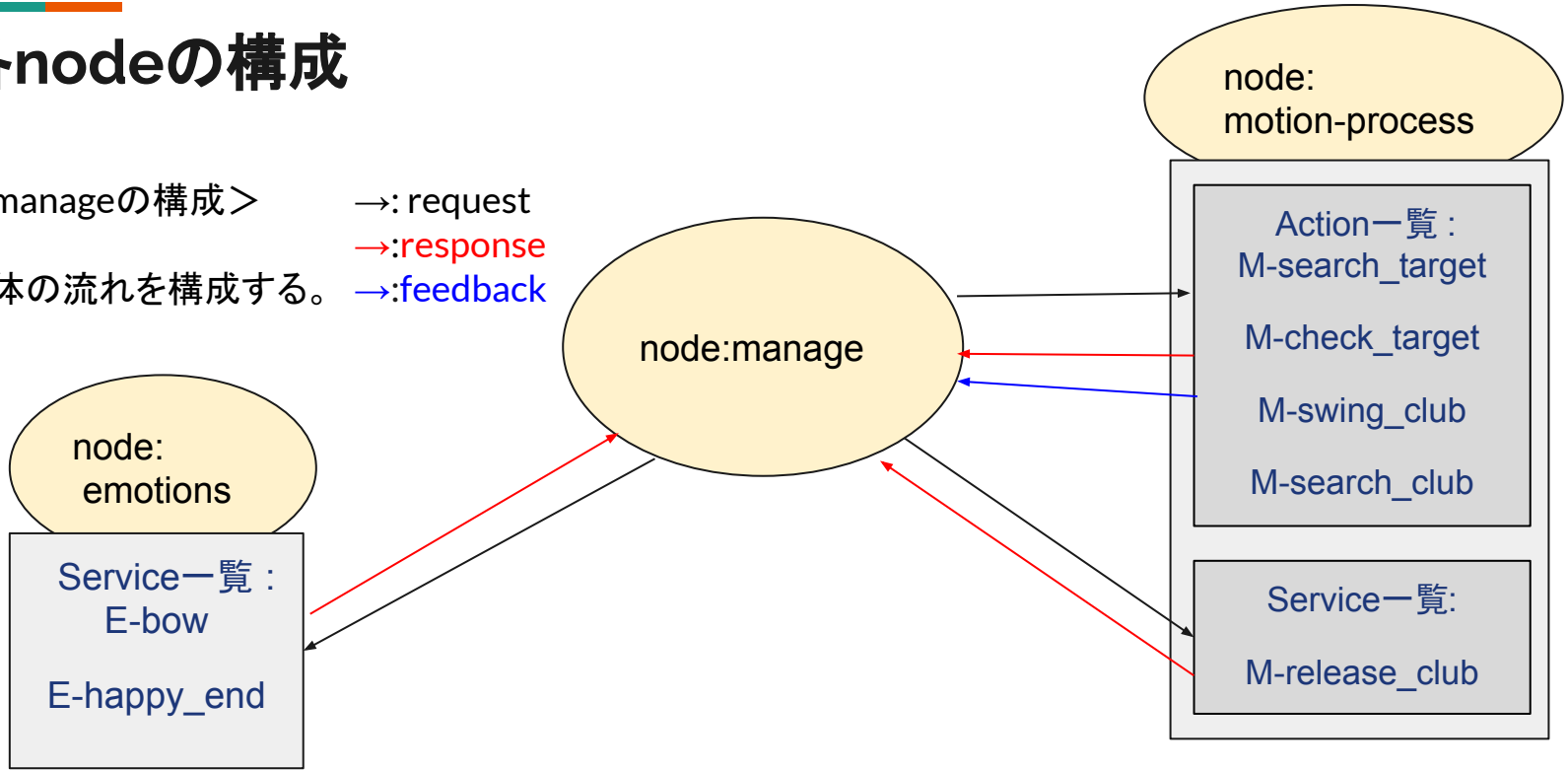
- M-search_club: 棒を探し、掴む
  入力:棒の色, feedback:状態, 出力:完了報告
- M-swing_club: 棒を振り上げ、打つ
  入力:動作命令, feedback:状態, 出力:完了報告
- M-search_target: 印を探す
  入力:印の色, feedback:状態, 出力:完了報告
- M-check_target: 倒れているか確認
  入力:印の色, feedback:状態, 出力:完了報告

各nodeの構成

＜manageの構成＞

全体の流れを構成する。

→: request
→:response
→:feedback

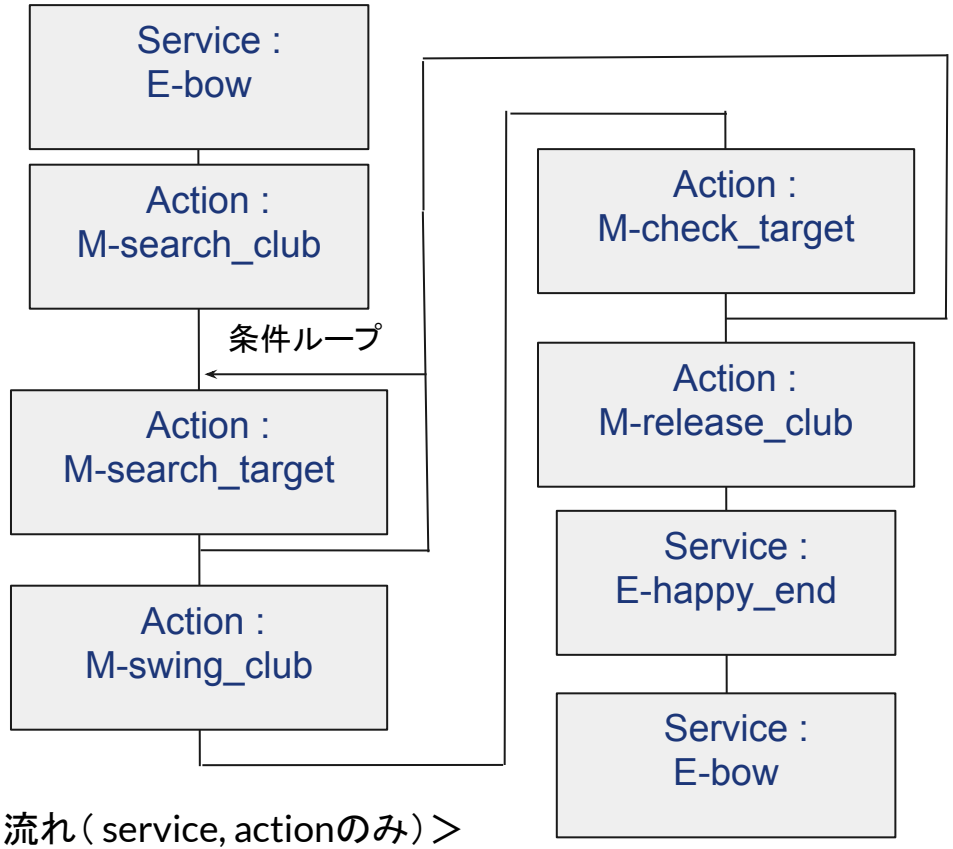


各nodeの構成

＜manageの構成＞

各service, actionのclientとなり、
順次通信を行い、実行していく。

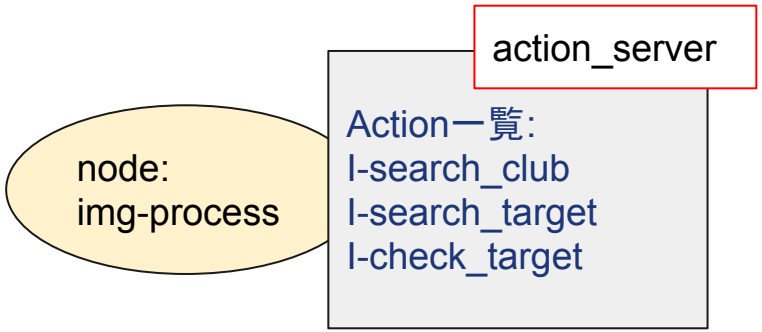
通信する順番を決めることで、
動作の一連の流れを実現する。



＜manageの大体の動作の流れ（service, actionのみ）＞

各nodeの構成

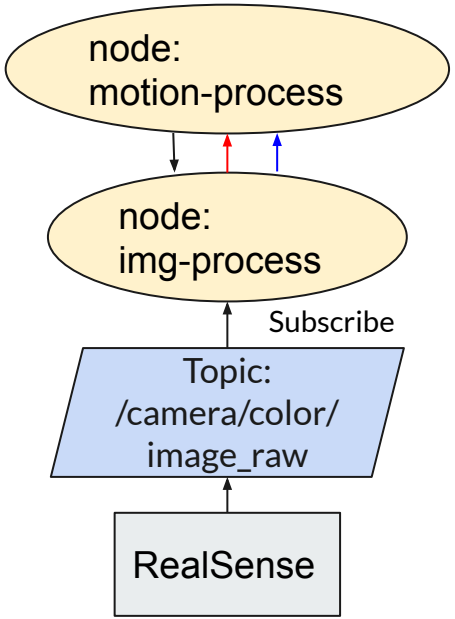
＜img_processの構成＞



- l-search_club:　棒の持つ場所を検出、
  現在地からの動作量を計算
- l-search_target:　印を検出、画面内の座標から
  動作量を計算
- l-search_target:　印が倒れているか確認

入力:検出する色, feedback:状態, 出力:動作量
画像処理には、OpenCVを使用

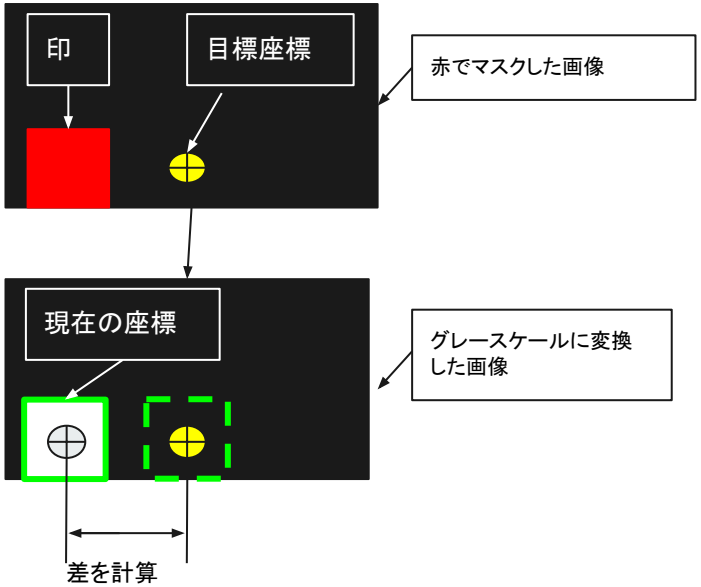
画像処理の流れ



→: request
→:response
→:feedback

1. motion_processからリクエストを受信
2. 指定のトピック(/camera/color/image_raw)から画像をsubscribe
3. 取得した画像をもとに、画像処理を実行(指定色でマスク)
4. 指定の色を検出するまで、まだ見つけていないことをfeedback
5. 指定の色を検知したら、見つけたことをfeedback
6. 各処理を行い(→次スライド)、結果を送る

画像処理の具体的な方法



<例>　赤色の印を検出する

1. RGB画像からHSV画像に変換
2. 画面を指定の色でマスク(inRange等)
3. マスクされた画像をグレースケールに変換
4. 印の輪郭を取得(threshold、findContours)
5. 輪郭を四角で囲む(boundingRect、rectangle)
6. 四角の中心の座標を計算
7. 目標の座標との差を計算
8. 実際の動作量に変換

開発のスケジュール

＜スケジュールの主な流れ＞

動きを個別に作る → 組み合わせて流れを作る

＜スケジュール目標＞

～中間審査：モデルの完成。モーションの完成。各感情表現の動きができる。棒を探して掴める

～最終発表：すべての動きの完成。manageに統合し、流れの完成。目標の実現。

＜懸念点＞

試行錯誤が必要なものに、予測以上の時間がかかってしまう可能性　（面白さ、モーションの作成(SRDF)、画像処理）

開発のスケジュール

Backlogのガントチャートを参照 https://uedalab.backlog.com/gantt/RS3