強化学習を用いた実用的な制御設計

MathWorks Japan
アプリケーションエンジニアリング部

アジェンダ

- 背景：制御屋から見た強化学習
- 強化学習の適用方法について検討
- 適用例：倒立振子の強化学習軌道生成制御
- 強化学習機能のマイコン実装、PIL、実機検証

アジェンダ

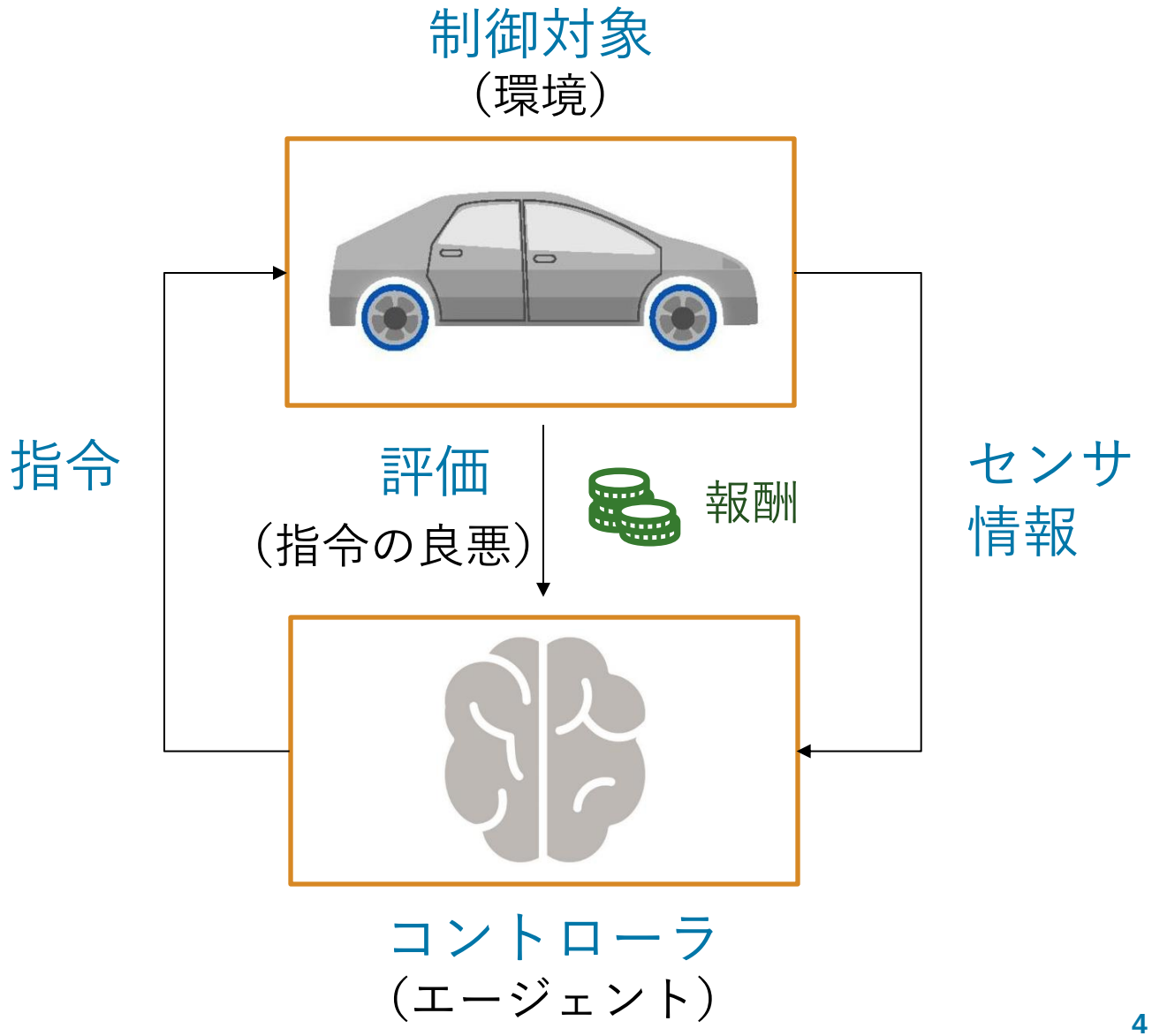
- 背景：制御屋から見た強化学習
- 強化学習の適用方法について検討
- 適用例：倒立振子の強化学習軌道生成制御
- 強化学習機能のマイコン実装、PIL、実機検証

強化学習は上手な制御方法を、試行錯誤しながら習得する仕組み

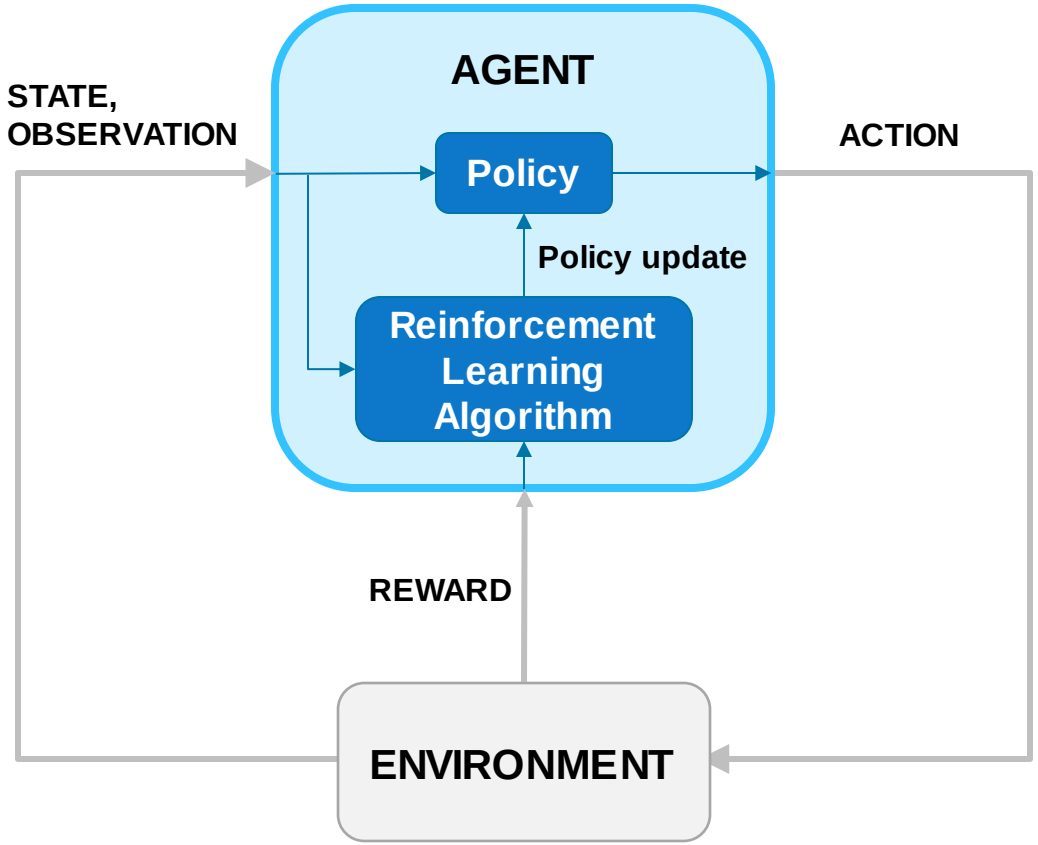
強化学習への期待

- 制御系の高度化・知能化
  - 自律性の獲得
  - 環境変化や不確実性への対応
  - 高い制御性能の達成
- 制御設計の自動化・省力化
  - 複雑な制御則を自動的に獲得
  - 制御則の再利用（再学習）

制御屋からすると、究極の制御の姿



強化学習　コンセプト
例：自動運転制御の場合

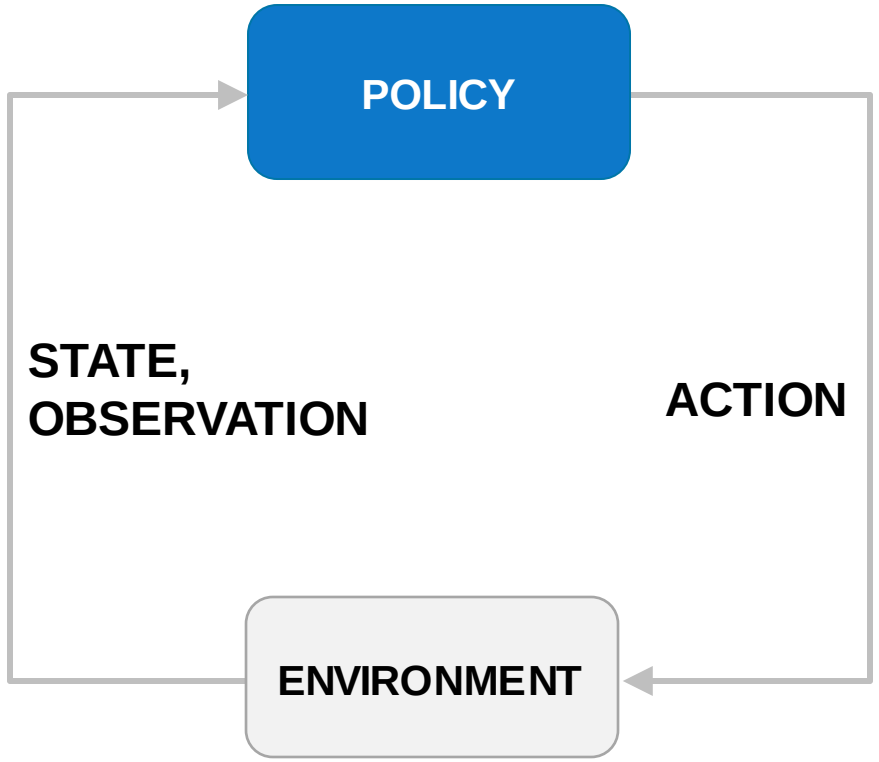


- 車両コントローラはどのように走るかを学習する
  - (**agent**)
- LIDARやカメラからのセンサー値
  - (**state, observation**)
- 路面状態や車両位置を表現する
  - (**environment**)
- ステアリング、ブレーキ、スロットル指令値
  - (**action**)
- (state)から次の(action)を生成する
  - (**policy**)
- ラップタイムや燃費効率などの最適化対象
  - (**reward**)

- **強化学習のアルゴリズム**により、ポリシーはトライ&エラーで更新される

強化学習　コンセプト
例：自動運転制御の場合

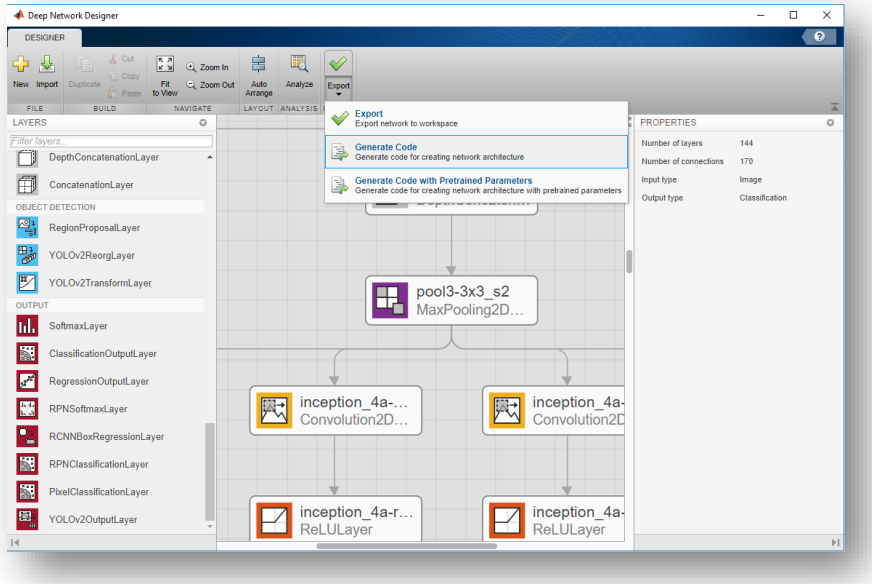
学習後は、学習済みポリシーのみ必要



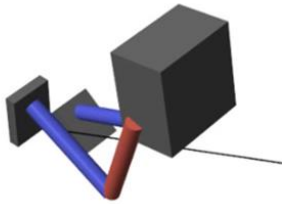
- 車両コントローラは学習後のアルゴリズムで(state)から(action)を得る
  - (**policy**)
- ステアリング、ブレーキ、スロットル指令値
  - (**action**)
- LIDARやカメラからのセンサー値
  - (**state, observation**)
- 路面状態や車両位置を表現する
  - (**environment**)

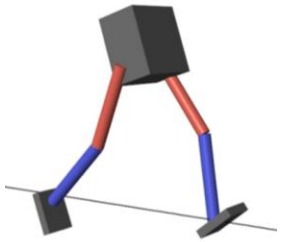
問題を定義することにより、この学習済みポリシーはラップタイムと燃費効率を最大化する

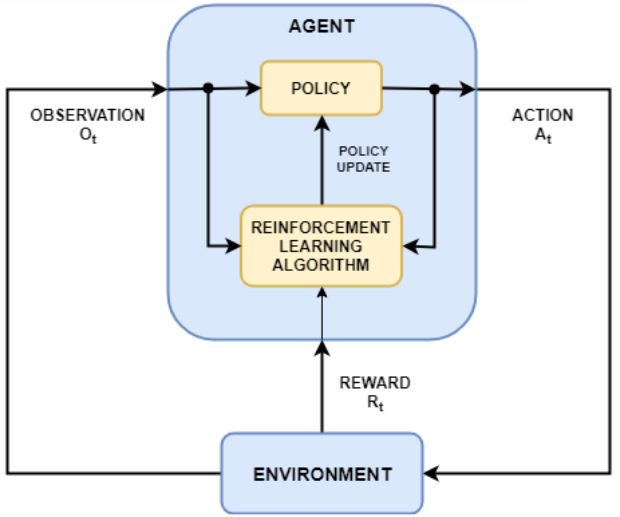
Reinforcement Learning Toolbox™



- 強化学習のフローを網羅的にサポート
  - MATLAB 関数 / Simulink® モデルで表現された環境とのインターフェース
  - エージェント作成のためのネットワーク構築環境
  - 各種アルゴリズムを提供
    - DQN / Double DQN
    - SARSA
    - REINFORCE
    - DDPG / TD3
    - A2C / A3C
    - PPO
    - SAC
  - マルチエージェントに対応
  - 配布のための最適方策の関数化
  - レファレンス・アプリケーションを多数提供
    - https://www.mathworks.com/help/reinforcement-learning/examples.html





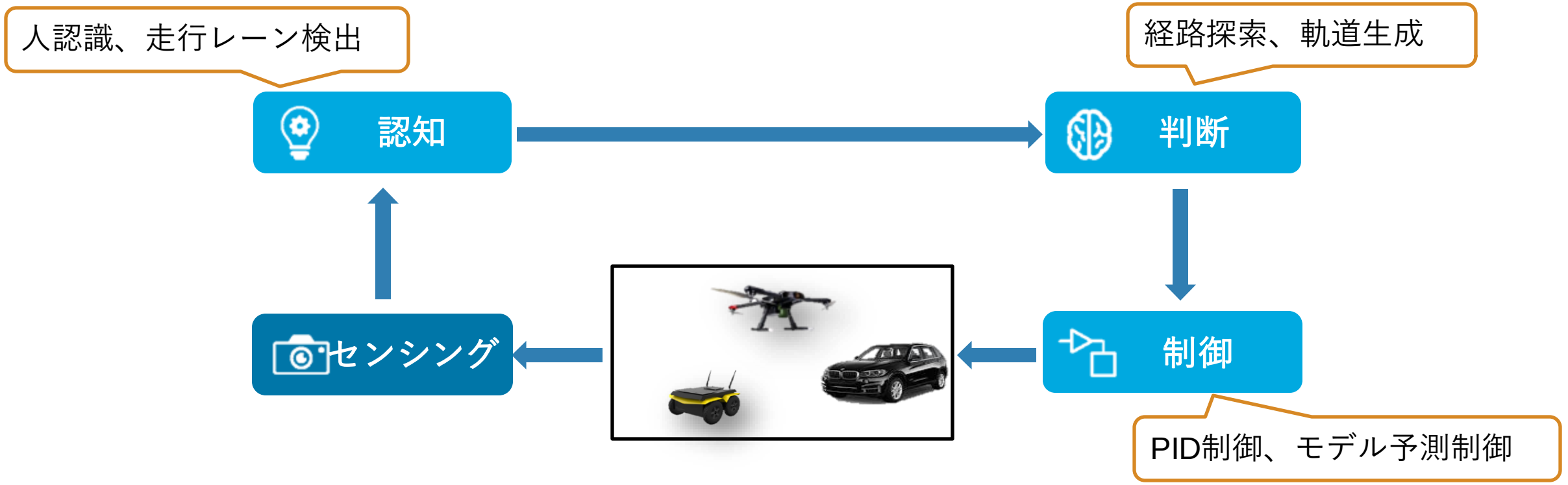


アジェンダ

- 背景：制御屋から見た強化学習
- 強化学習の適用方法について検討
- 適用例：倒立振子の強化学習制御
- 強化学習機能のマイコン実装、PIL、実機検証

自律制御システム

- 自律制御システムは、認知、判断、制御に分けることができる
- 強化学習は、これら全てを同時に実行できるが、その代わりに全てがブラックボックスとなる

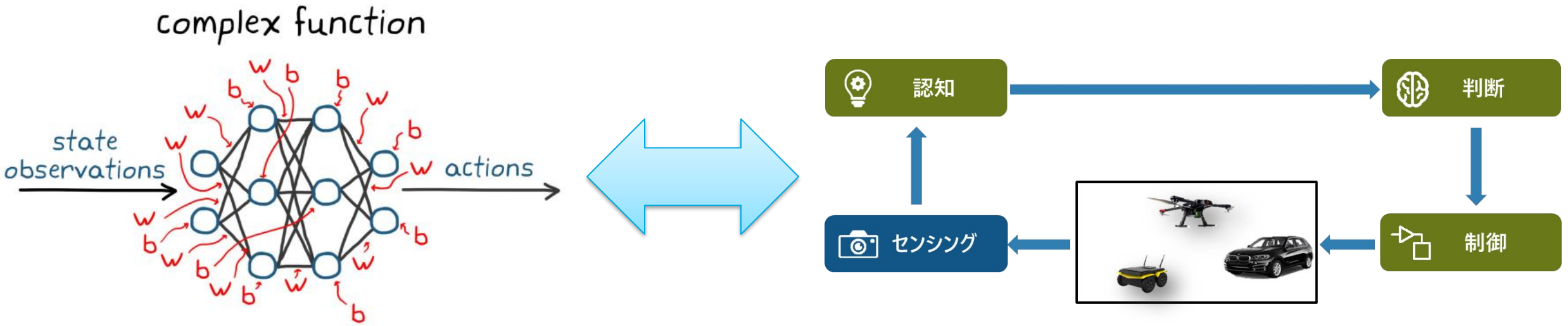


説明できないニューラルネットワーク

「認知」「判断」「制御」など、システムを細かい要素に分割できると、問題の特定をしやすい。しかし、深層ネットワークは、時には数千の重みとバイアス、非線形な活性化関数が組み合わされているため、ネットワークを分割して捉えることは難しい。

従って、設計者にとってはブラックボックスとなっており、深層ネットワークを微調整をしたい場合に、どの重みをどのように調整すればよいか、ということは分からない。

深層ネットワークを説明可能にする研究も行われているが、現時点では、制御設計をする場合においてはまだブラックボックスである。

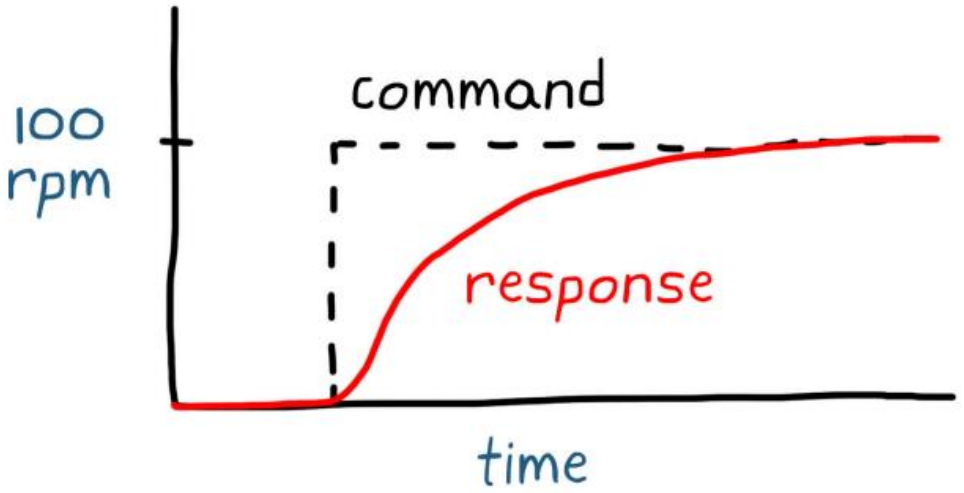


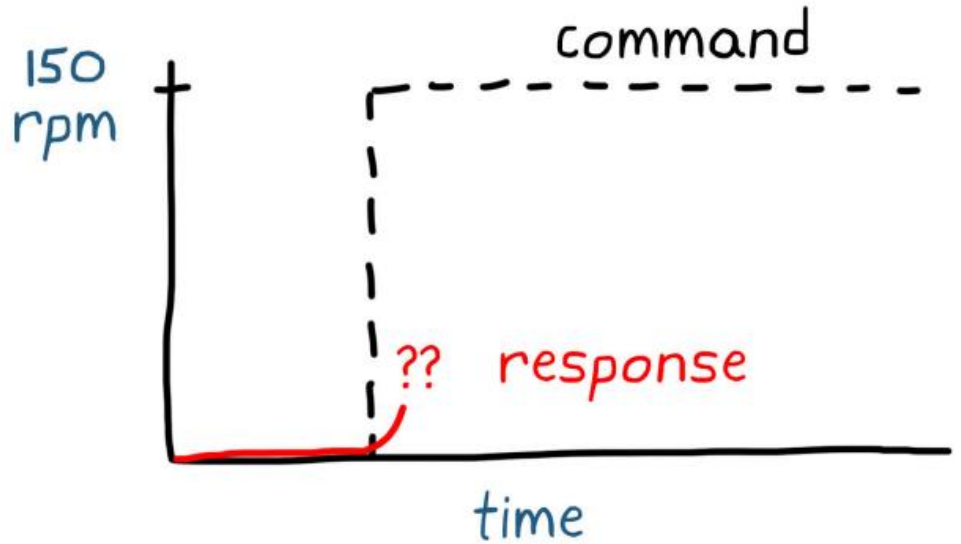
予測できない非線形性

深層ネットワークは非線形である。従って、学習した範囲外の入力や状態になった場合に、何が起こるのかを予想することができない。

例えば、モーターなどのシステムにPID制御を適用した場合、従来の制御理論によって解析できる。指令値に100[rpm]を入れた場合と150[rpm]を入れた場合は、相似の応答を示す。

一方、深層ネットワークで制御器を構成した場合に、0から100[rpm]の指令値までの学習しかさせなかった場合、150[rpm]の入力をした場合に、どのような挙動を示すのか、予想することはできない。





強化学習の使いどころ

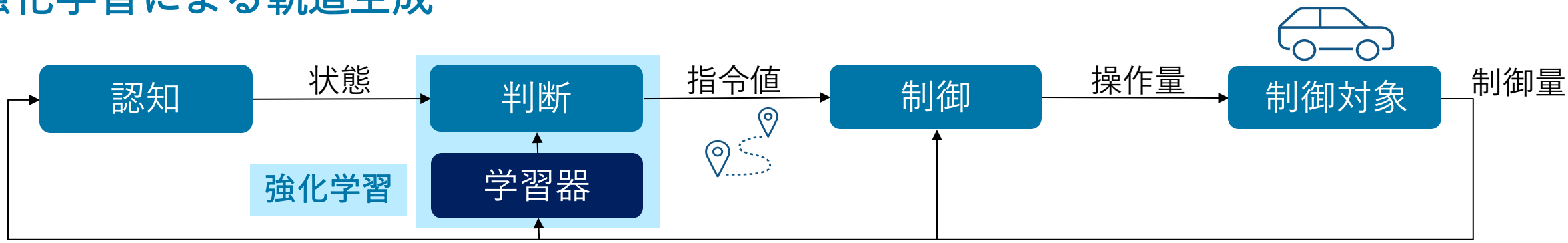
- 強化学習は、「判断」に用いるのがよいと考えられる
- 理由：
  - 強化学習の行動を離散に、または範囲を限定的にすることができる。予想外の行動が選択された場合でも、安定性、安全性の検証、担保は「制御」でできる

- 「認知」ではない理由：
  - 認知機能では、環境との相互作用を考慮する必要がないため、教師あり/なし学習の方が向いている

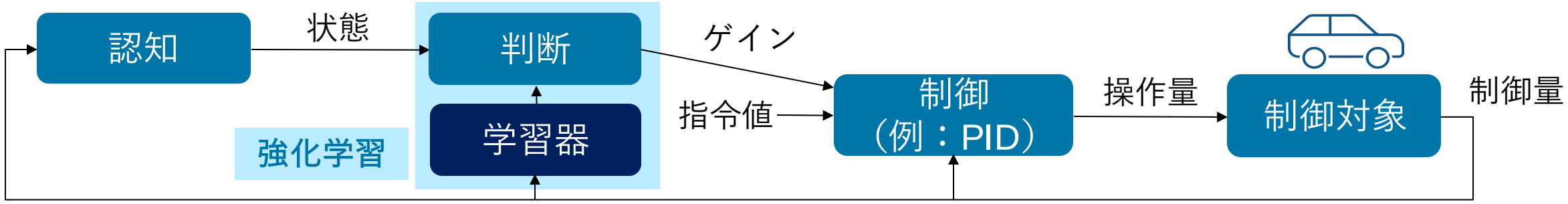
- 「制御」ではない理由：
  - 制御機能では、安全性が特に求められるので、理論的に説明しづらいブラックボックスを用いるのを避けたい

強化学習の適用方法

1. 強化学習による軌道生成



2. 強化学習によるゲインスケジューリング制御



二つの手法の使い分け

1. 強化学習による軌道生成

- 達成すべき動作が複雑である
- 制御器に与える指令値が複雑である

2. 強化学習によるゲインスケジューリング制御

- 高い制御性能が必要である
- 制御器に与える指令値は複雑ではない

強化学習を「判断」にのみ用いることのメリット

- 「制御」において、従来の制御理論に基づく制御器を用いることができる
- 「制御」の実行周期と「判断」の実行周期を別々に設定することができるため、計算時間のかかる深層ネットワークのモデルも使いやすい
- 「達成する制御性能や動特性が複雑」である制御課題に対して、従来の制御器では達成できなかった性能、動作を実現できる

アジェンダ

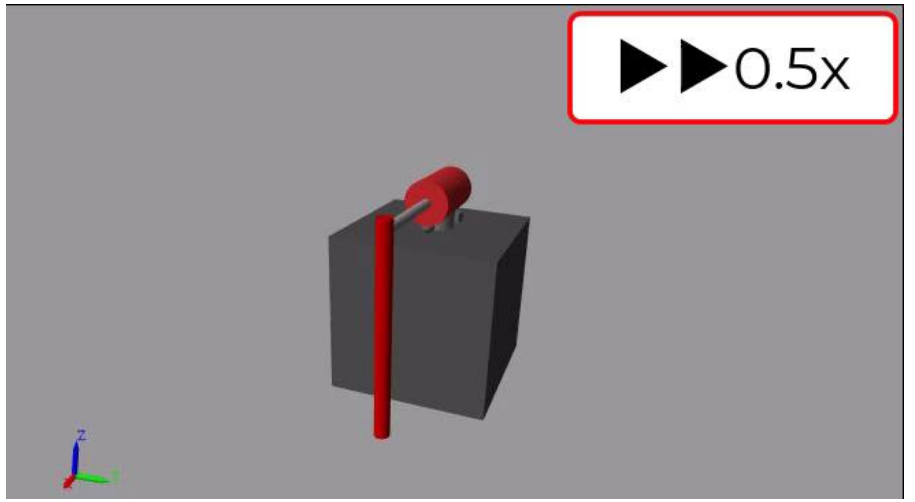
- 背景：制御屋から見た強化学習
- 強化学習の適用方法について検討
- 適用例：倒立振子の強化学習制御
- 強化学習機能のマイコン実装、PIL、実機検証

倒立振子

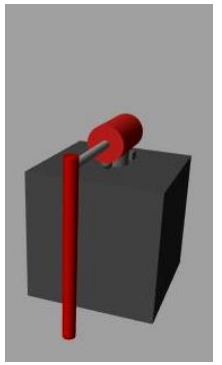
- 制御対象
  - Quanser QUBE – Serve 2
  - コントローラー：Raspberry Pi
- 制御の入出力
  - モーターと振子のそれぞれの角度と角速度を計測し、DCモーターに加える電圧を操作する。
  - 与えられる電圧は-12[V]から+12[V]の間。
- 制御目標
  - 振子を立たせる。

Raspberry Pi

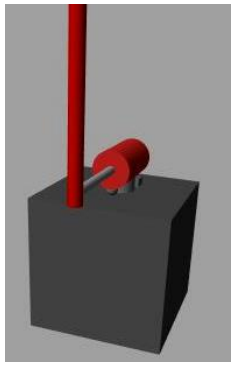
Quanser QUBE – Serve 2



制御目標を分解する（要件定義）



0[deg]



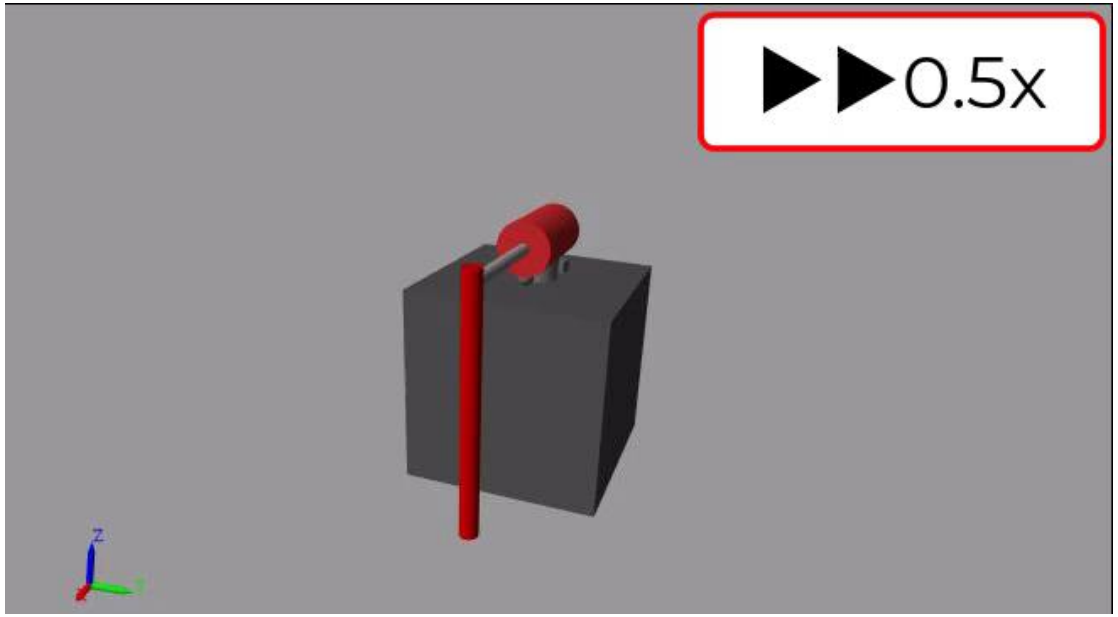
180[deg]

「振り子を立たせる」とは、

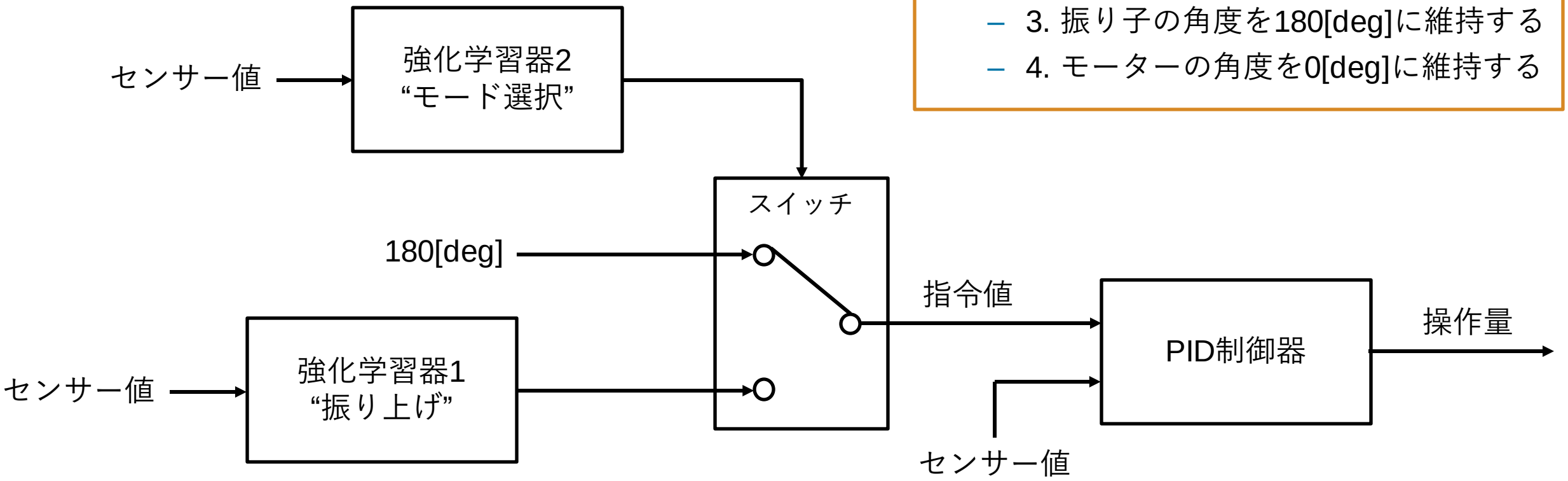
1. 0[deg]で静止している振り子を振動させる
2. 振り子を180[deg]近辺まで持ち上げる
3. 振り子の角度を180[deg]に維持する
4. モーターの角度は±150[deg]を超えてはいけない（ハードウェア制約）

振り子を180[deg]近辺まで持ち上げる動作は複雑であるため、「強化学習による軌道生成」が適用できる。

また、振り子を180[deg]に維持することは、PID制御をベースとした制御器で達成できる。



「判断」と「制御」を具体化



- PID制御器
  - 3. 振り子の角度を180[deg]に維持する
  - 4. モーターの角度を0[deg]に維持する

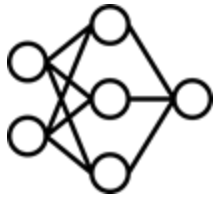
- 強化学習器1 “振り上げ”
  - 1. 0[deg]で静止している振り子を振動させる
  - 2. 振り子を180[deg]近辺まで持ち上げる
  - 4. モーターの角度を0[deg]近辺に維持する

- 強化学習器2 “モード選択”
  - 1. 0[deg]で静止している振り子を振動させる
  - 3. 振り子の角度を180[deg]に維持する

強化学習のワークフロー

Environment

Reward



Policy representation

Agent

Training

Deployment

環境構築（プラントモデリング）

Environment

Reward

Policy representation

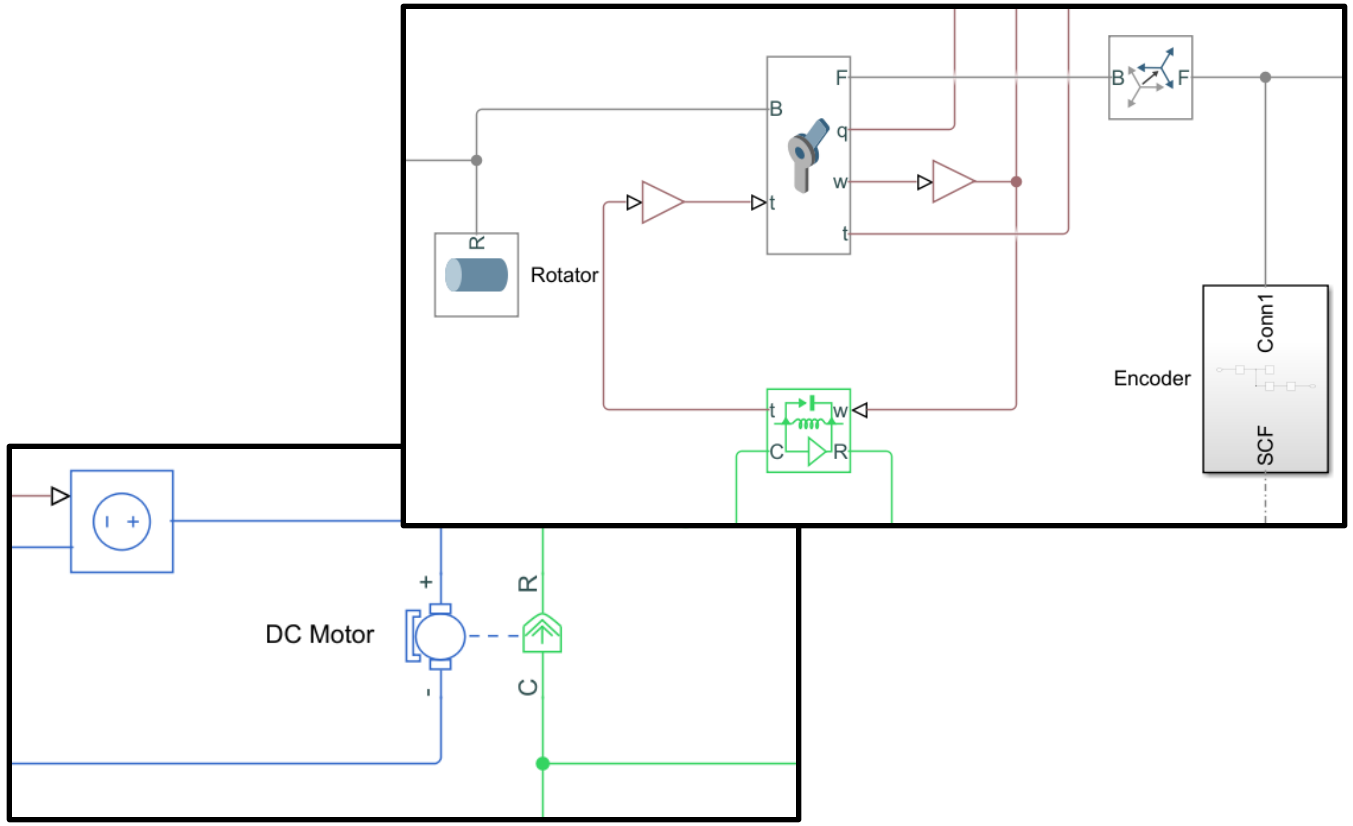
Agent

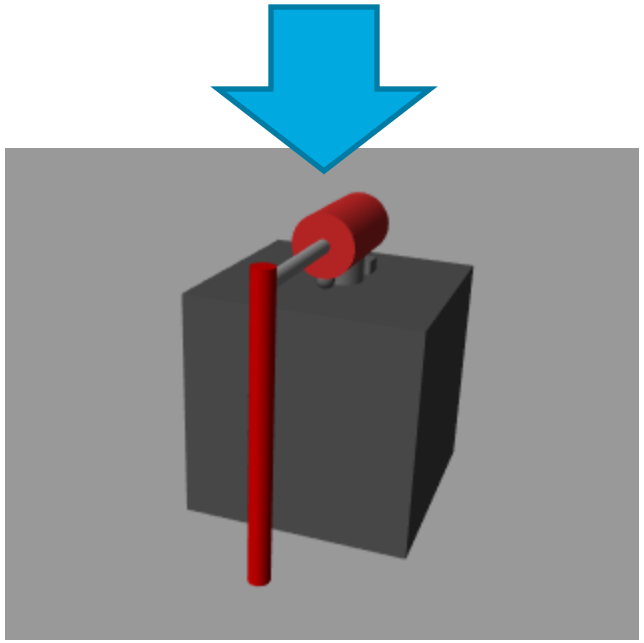
Training

Deployment

Simscapeによるモデル化

- DCモーターによって台座を回転させ、振り子にトルクを与えるシステムである
- Simscape Electrical™, Simscape Multibody™ を用いてモデル化が可能





報酬設計

Environment **Reward** Policy representation Agent Training Deployment

強化学習器1 ”振り上げ” の報酬設計

- 強化学習器1 “振り上げ”
  - 1. 0[deg]で静止している振り子を振動させる
  - 2. 振り子を180[deg]近辺まで持ち上げる
  - 4. モーターの角度を0[deg]近辺に維持する

1. モーター角度が±180[deg]以内にある時に、振り子の角度が180±30[deg]以内にあれば100、そうでなければ0の報酬を与える
2. 以下のコスト関数の報酬rを与える

$$r = -\theta^2 - 0.1(\phi - \pi)^2 - 0.1\left(\frac{d\phi}{dt}\right)^2$$

ただし、θはモーター角度、ϕは振り子角度である。

強化学習器2 ”モード選択” の報酬設計

- 強化学習器2 “モード選択”
  - 1. 0[deg]で静止している振り子を振動させる
  - 3. 振り子の角度を180[deg]に維持する

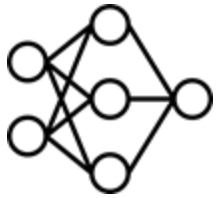
1. 振り子の角度が180±30[deg]以内にあれば1、そうでなければ0の報酬を与える
2. 以下のコスト関数の報酬rを与える

$$r = -\theta^2$$

ポリシー、エージェント構築

Environment

Reward



Policy representation

Agent

Training

Deployment

エージェント選択

Agentによって、離散、連続の対応可否が異なる。
今回は“振り上げ” にSACを採用し、“モード選択” にPPOを採用した。

用いることができるエージェントと離散、連続の対応表

Agent	Observation	Action
Q-Learning	Discrete or Continuous	Discrete
SARSA	Discrete or Continuous	Discrete
DQN	Discrete or Continuous	Discrete
Policy Gradients	Discrete or Continuous	Discrete or Continuous
DDPG	Discrete or Continuous	Continuous
TD3	Discrete or Continuous	Continuous
Actor-Critic	Discrete or Continuous	Discrete or Continuous
PPO	Discrete or Continuous	Discrete or Continuous
SAC	Discrete or Continuous	Discrete or Continuous

それぞれのエージェントを設計

SAC エージェント

```
criticOptions = rlRepresentationOptions('Optimizer','adam','LearnRate',1e-3,...
    'GradientThreshold',1,'L2RegularizationFactor',2e-4);
critic1 = rlQValueRepresentation(criticNet,obsInfo,actInfo,...
    'Observation',{'observation'},'Action',{'action'},criticOptions);
critic2 = rlQValueRepresentation(criticNet,obsInfo,actInfo,...
    'Observation',{'observation'},'Action',{'action'},criticOptions);
```

```
actorOpts = rlRepresentationOptions('LearnRate',1e-04,'GradientThreshold',1);

actor = rlStochasticActorRepresentation(actorNetwork,obsInfo,actInfo,...
    'Observation',{'observation'},actorOpts);
```

```
agentOpts = rlSACAgentOptions(...
    'SampleTime',Ts,...
    'TargetSmoothFactor',1e-3,...
    'ExperienceBufferLength',1e6,...
    'DiscountFactor',0.99,...
    'MiniBatchSize',128, ...
    "UseDeterministicExploitation", true);

agent_swing = rlSACAgent(actor,[critic1, critic2],agentOpts);
```

PPO エージェント

```
criticOpts = rlRepresentationOptions('LearnRate',1e-4);
critic = rlValueRepresentation(criticNetwork,obsInfo,'Observation', ...
    {'observation'},criticOpts);
```

```
actorOpts = rlRepresentationOptions('LearnRate',1e-04,'GradientThreshold',1);

actor = rlStochasticActorRepresentation(actorNetwork,obsInfo,actInfo,...
    'Observation',{'observation'},actorOpts);
```

```
agentOpts = rlPPOAgentOptions(...
    'SampleTime',Ts,...
    'DiscountFactor',0.99,...
    'ExperienceHorizon', floor(Tf/Ts), ...
    'MiniBatchSize', floor(Tf/Ts), ...
    'EntropyLossWeight', 1e-4, ...
    'UseDeterministicExploitation', true);

agent_select = rlPPOAgent(actor, critic, agentOpts);
```

“UseDeterministicExploitation”をtrueにすることで、学習後のPolicyを確定的に実行可能。

学習

Environment

Reward

Policy representation

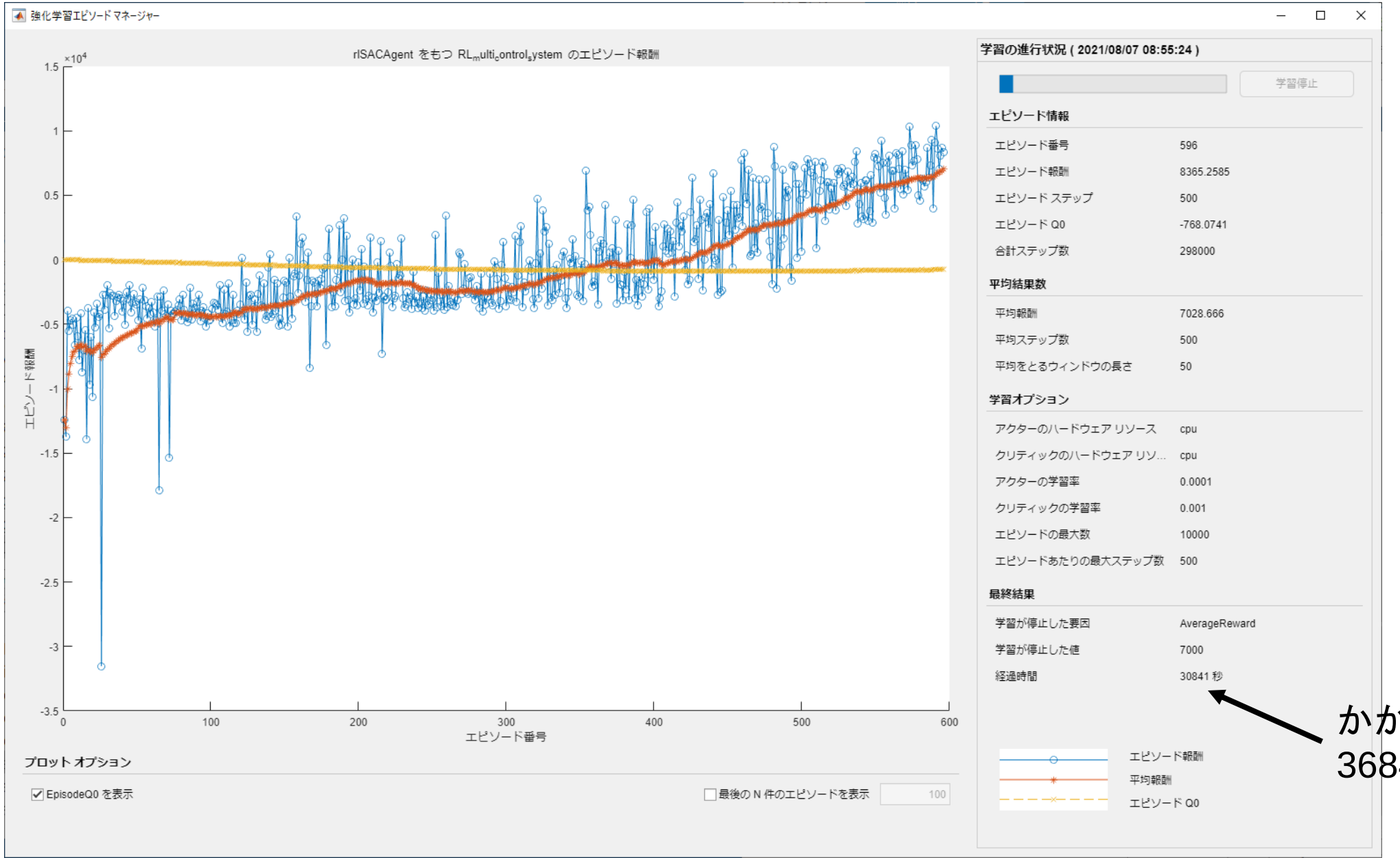
Agent

Training

Deployment

SACによる“振り上げ”を学習

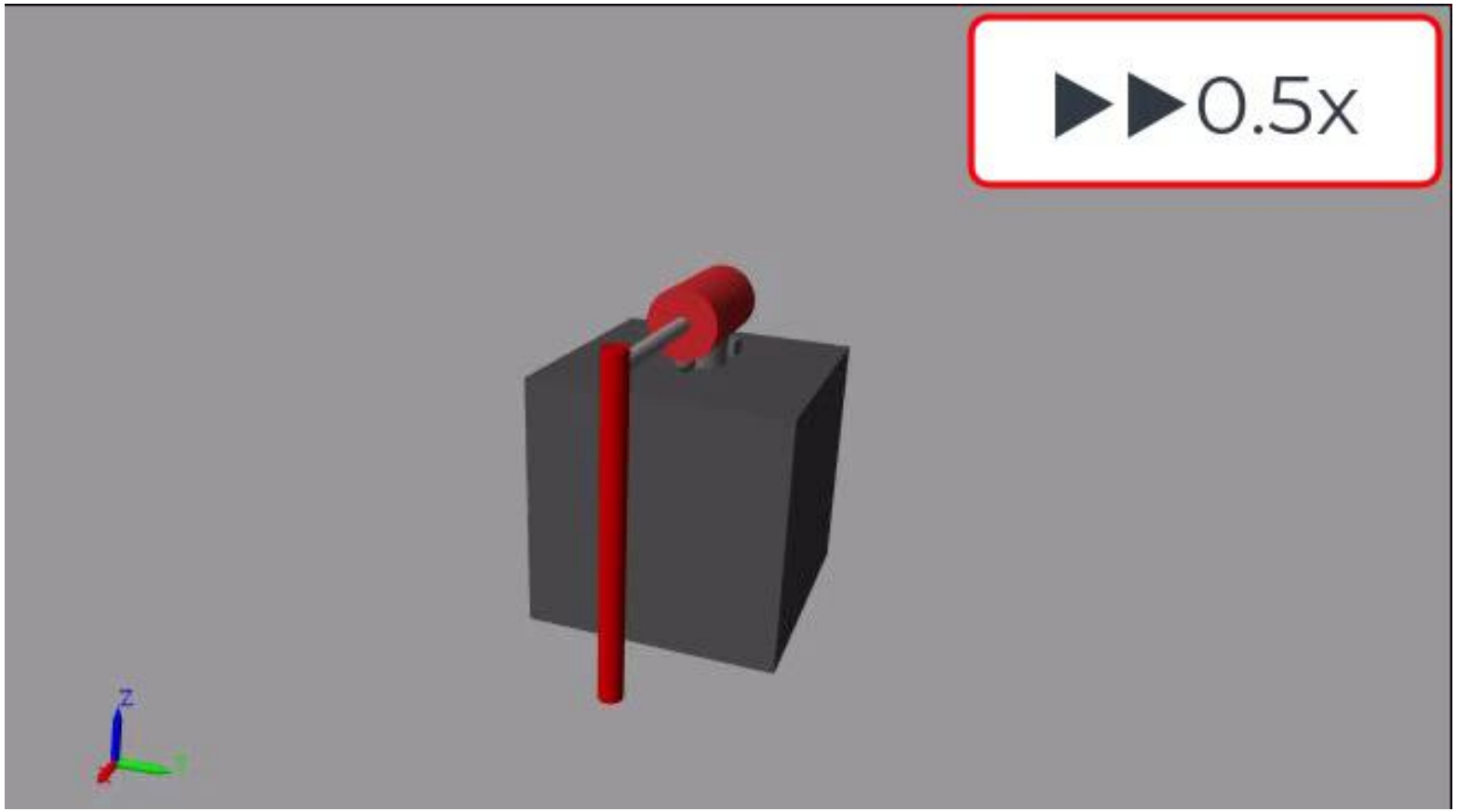
Windows 10, 21H1
Intel® Xeon® CPU E5-1650 v3 @ 3.50GHz x 12
64 GB memory



かかった時間：
36841秒 ≒ 10.23時間

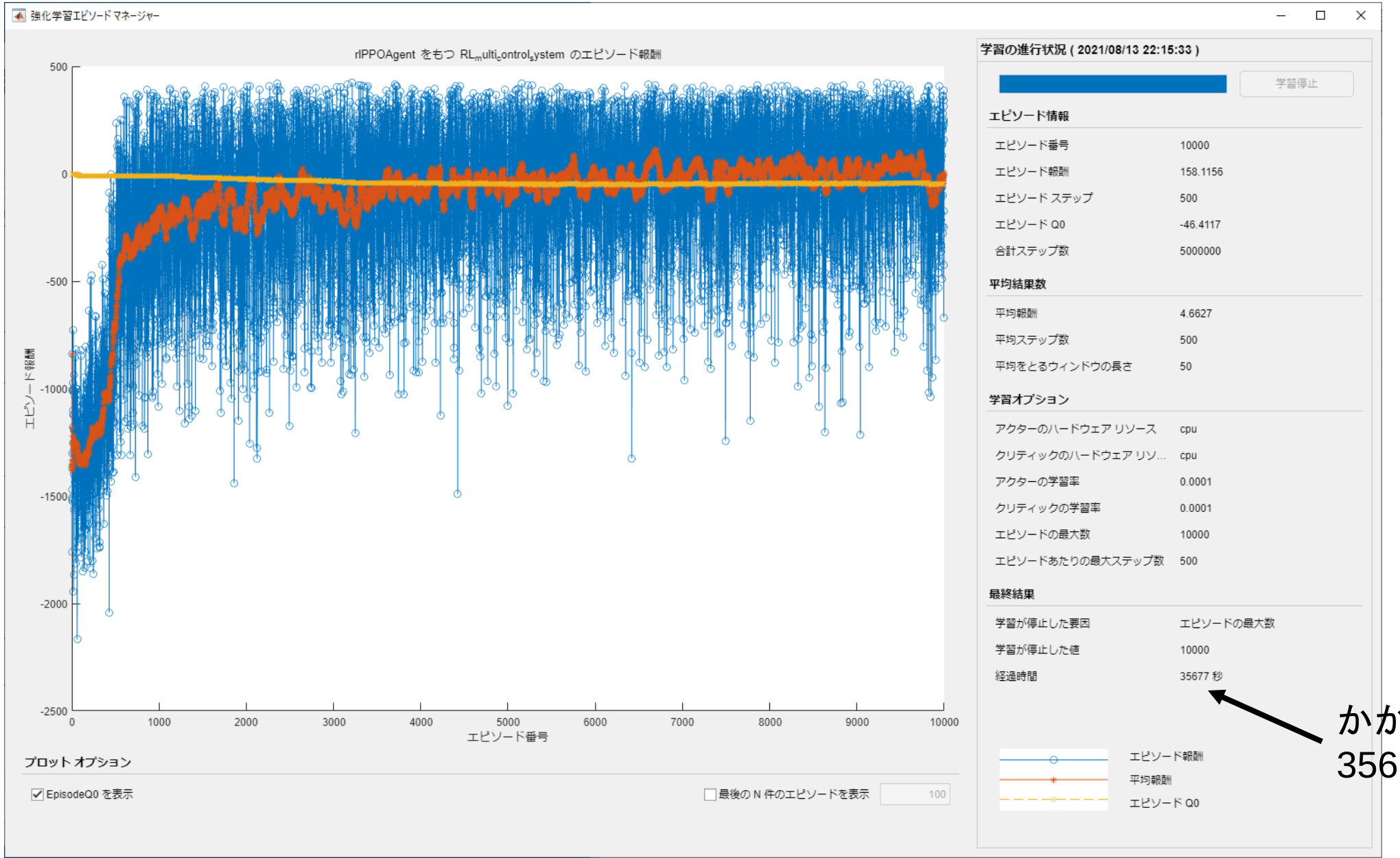
“振り上げ” 学習結果（シミュレーション）

- モーター角度が0付近（台座の正面）を維持しながら、振り子を何度も振り上げる動作を確立した



PPOによる“モード選択”を学習

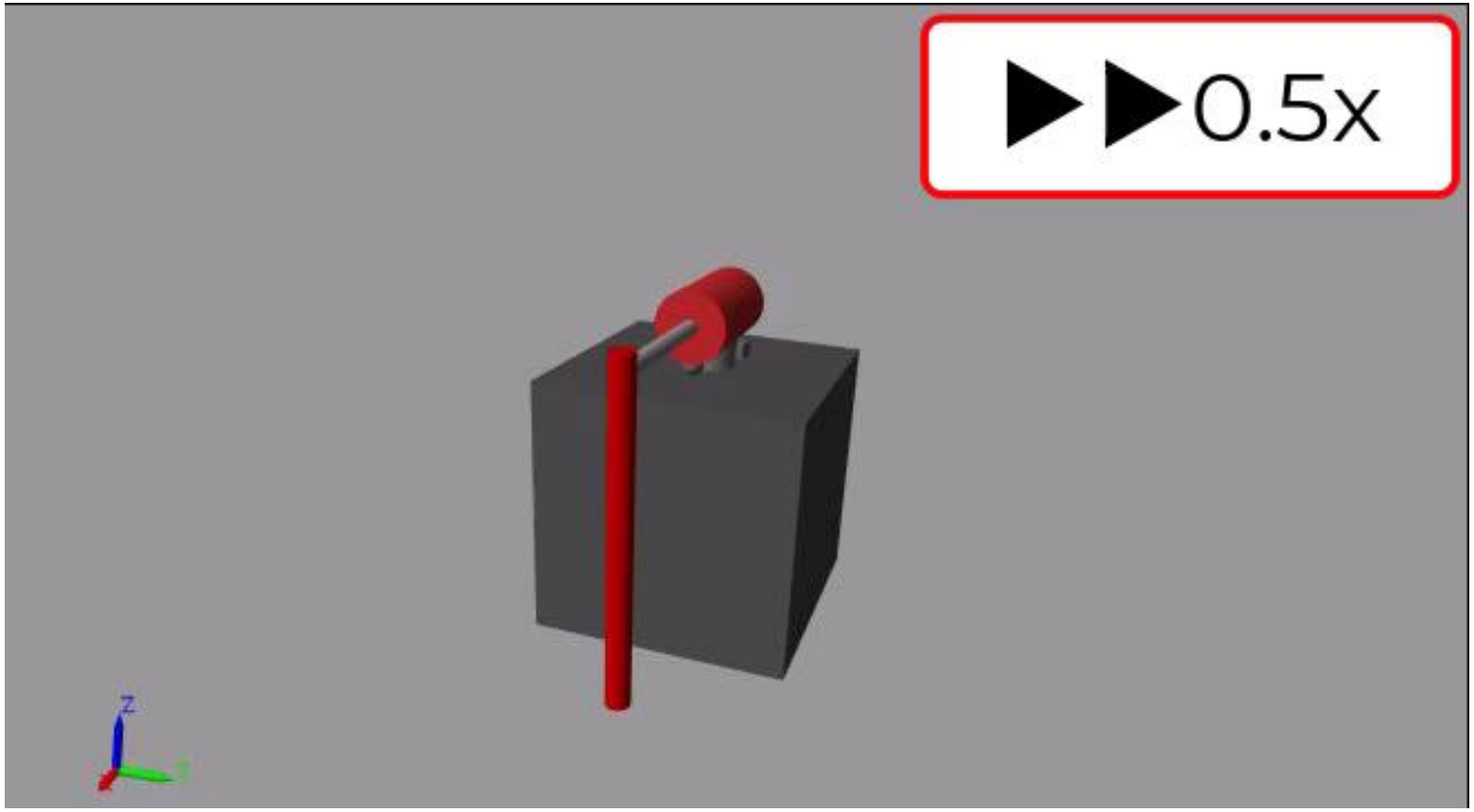
Windows 10, 21H1
Intel® Xeon® CPU E5-1650 v3 @ 3.50GHz x 12
64 GB memory



かかった時間:
35677秒 ≒ 9.91時間

“モード選択” 学習結果（シミュレーション）

- “振り上げ”と”指令値固定”をきちんと切り替え、倒立動作を実現させることができた

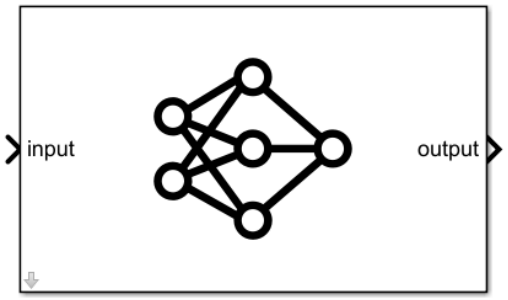


実装

Environment Reward Policy representation Agent Training **Deployment**

学習済みの深層ネットワークを展開する方法

学習済みネットワーク



Coder

GPU Coder

MATLAB Coder

cuDNN

TensorRT

MKL-DNN

ARM Compute

General

NVIDIA社GPU

NVIDIA社GPU

Intel社Xeon,
Xeon Phi

ARM社Mali,
Cortex-A

一般的なCPU

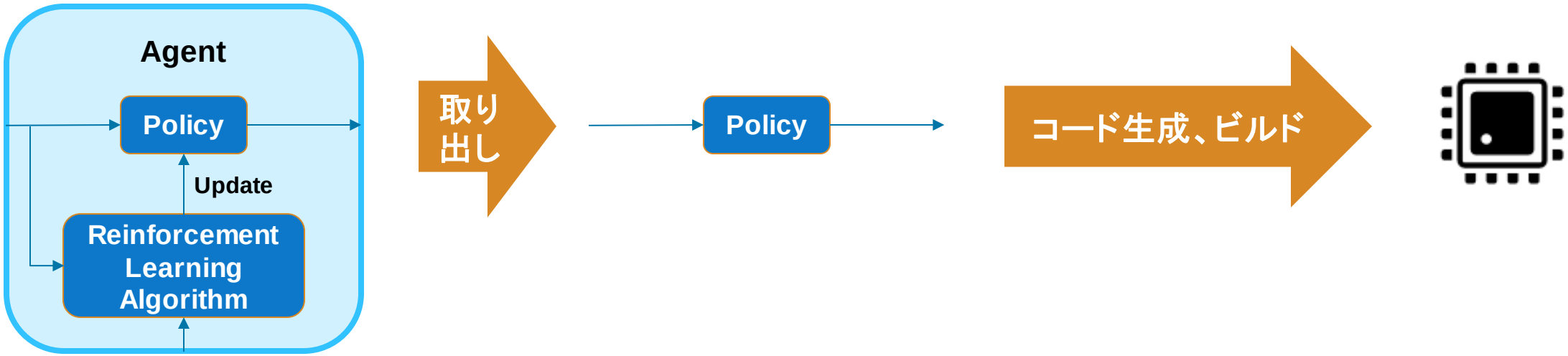
今回の深層ネットワークであれば、
どのタイプのライブラリでもコード
生成することができる

アジェンダ

- 背景：制御屋から見た強化学習
- 強化学習の適用方法について検討
- 適用例：倒立振子の強化学習制御
- 強化学習機能のマイコン実装、PIL、実機検証

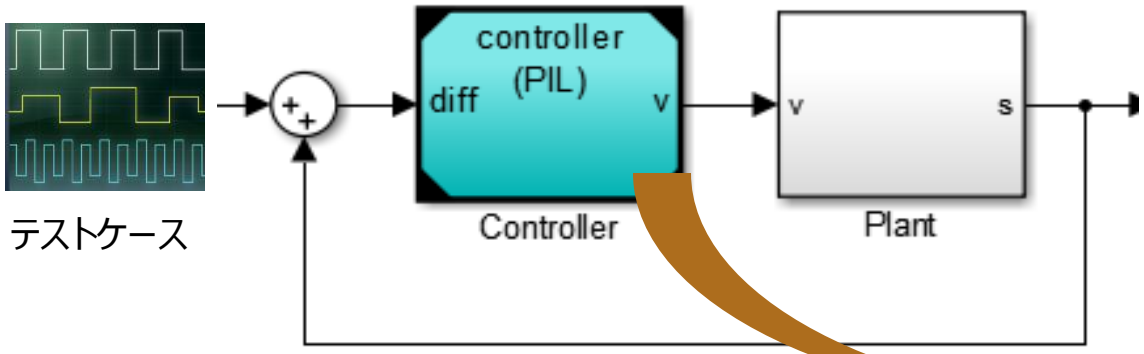
強化学習制御器の実装

- エージェントからPolicyを取り出し、コード生成、ビルドして組み込み実装する
- 現時点では、学習機能をコード生成して実装することはできない



Embedded Coder® によるPIL検証

- モデルと生成コードの計算結果の等価性を確認
- 各制御ステップごとの計算時間を評価
- 結果のレポート出力



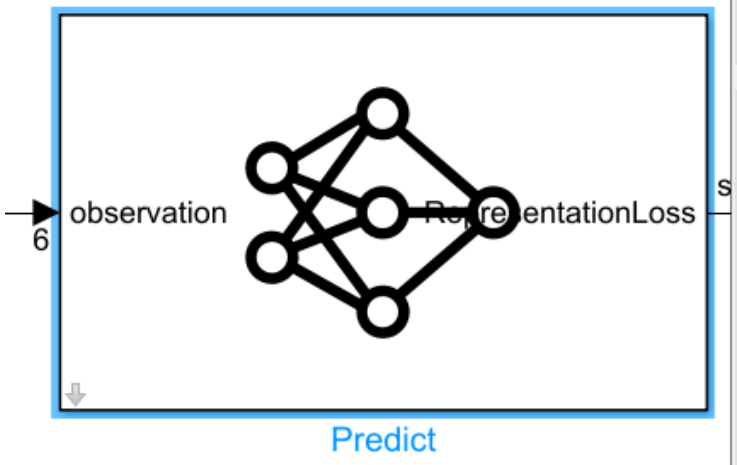
Raspberry Pi 3 Model B+
CPU: 1.4GHz ARM Cortex-A53
RAM: 1GB

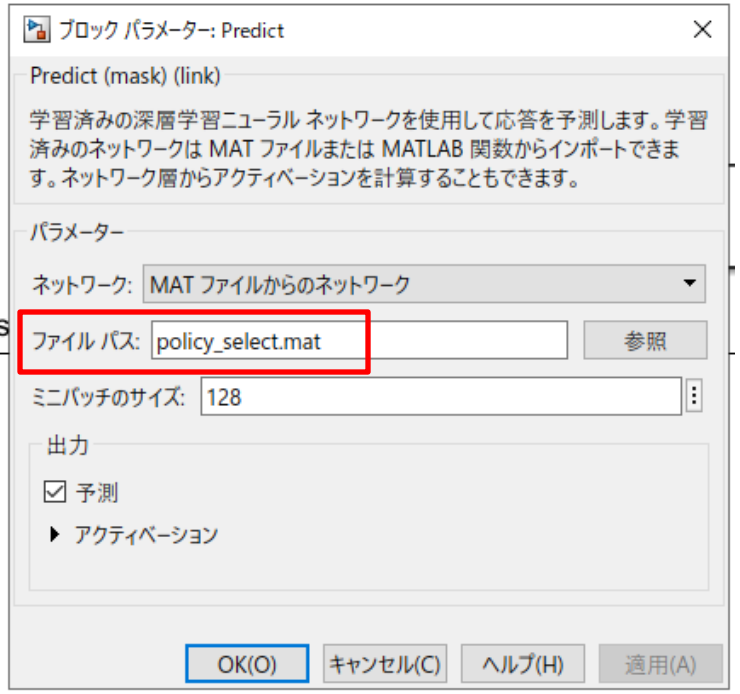
Simulinkモデルから生成したコードをRaspberry Pi で実行し、Simulink上のプラントモデルの実行と同期

強化学習エージェントからPolicyの取り出し

- “generatePolicyFunction”コマンドにより、Policyのみを抽出してmatファイルに変換する
- 抽出したmatファイルを”Predict”ブロックから参照する

```
generatePolicyFunction(agent_swing, 'MATFileName', "policy_swing.mat");
generatePolicyFunction(agent_select, 'MATFileName', "policy_select.mat");
```





PIL検証結果（実行時間の計測）

2. Profiled Sections of Code

Section	Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls
RL_multi_controller_deploy_initiali...	1250	1250	1250	1250	1
RL_multi_controller_deploy_Init	209	209	209	209	1
RL_multi_controller_deployTID0 [0.005 0]	2917	656	2917	656	2001
RL_multi_controller_deployTID1 [0.02 0]	3111596	2676483	3111596	2676483	501

3. CPU Utilization [hide]

Task	Average CPU Utilization	Maximum CPU Utilization
RL_multi_controller_deployTID0 [0.005 0]	0.01312%	0.05834%
RL_multi_controller_deployTID1 [0.02 0]	13.38%	15.56%
Overall CPU Utilization	13.4%	15.62%

Fig.1 コード実行プロファイル

深層ネットワークを実行するタスクは、平均2.68[ms]で計算されており、CPU負荷率は13.38%である。

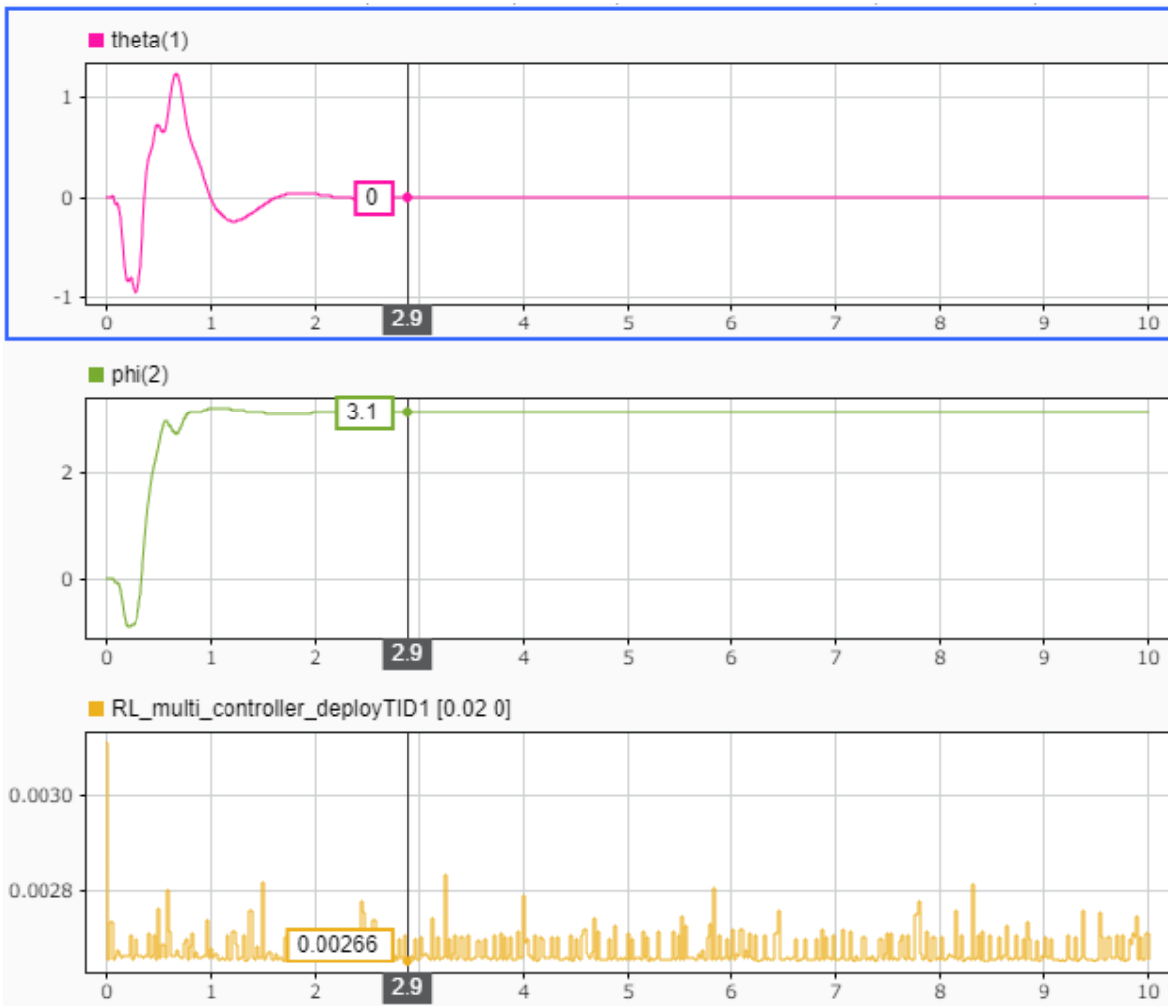
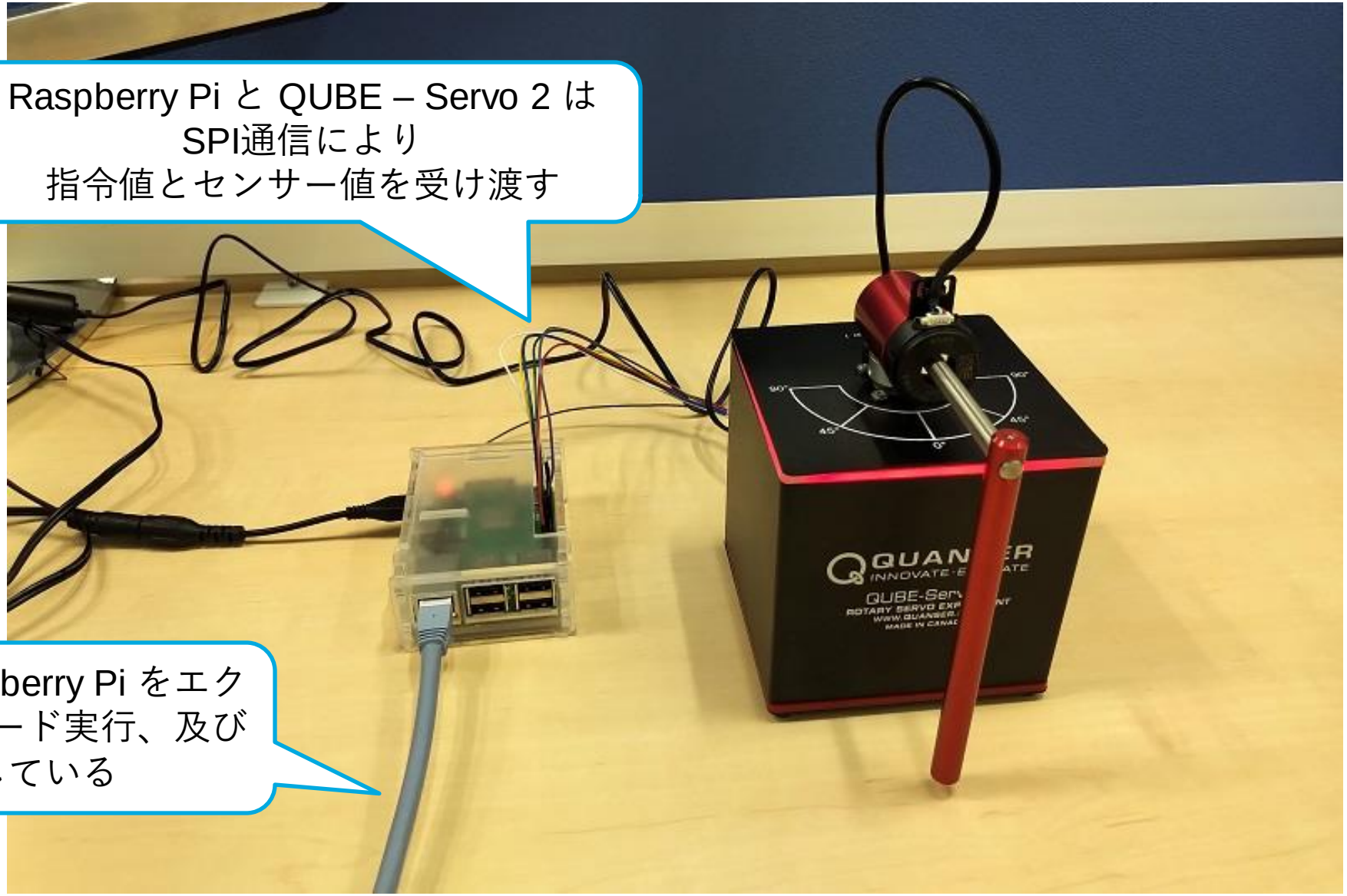
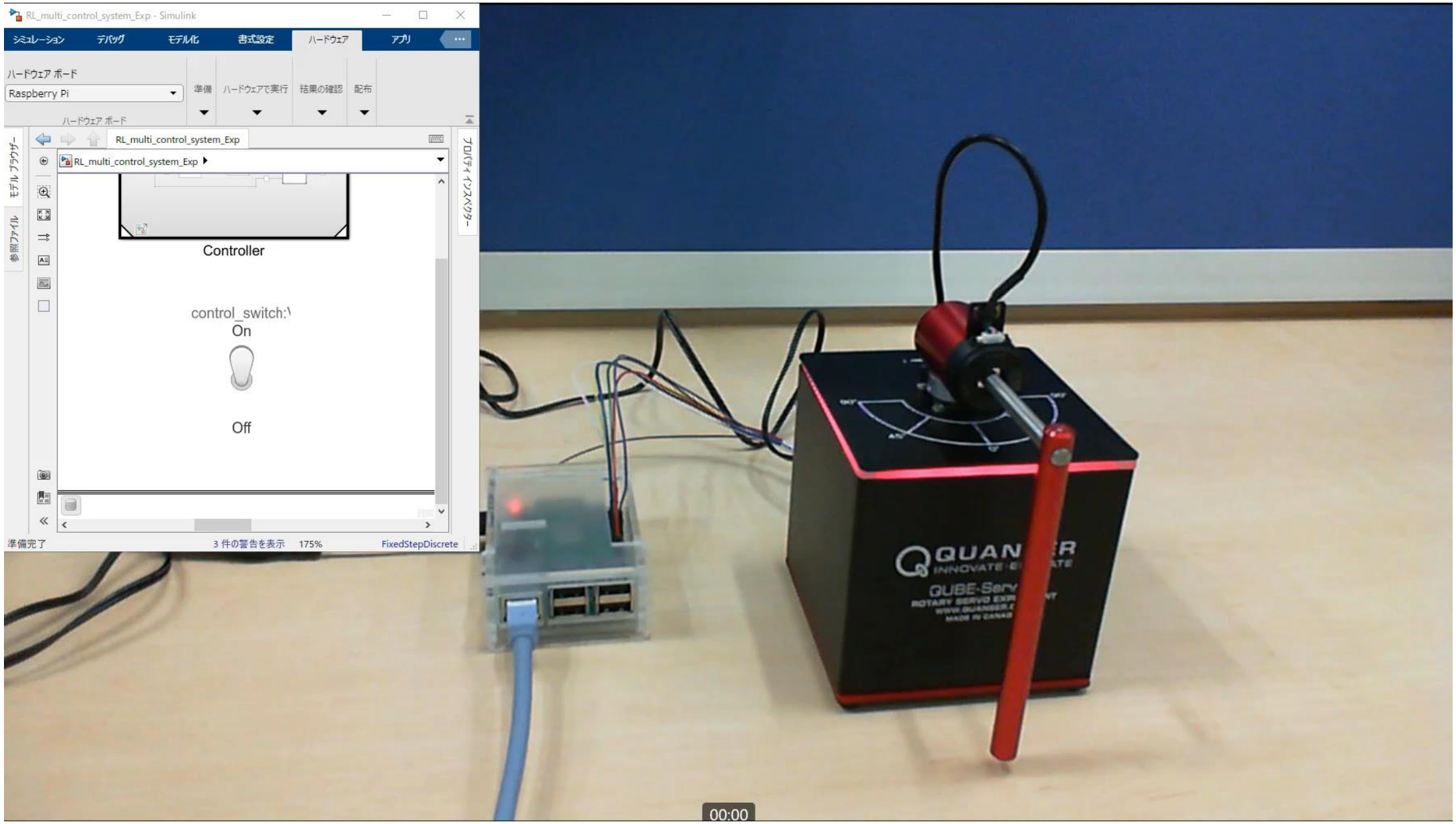


Fig.2 各ステップごとの実行時間

実機検証（Rapid Code Prototyping, RCP）



実機の動作


RL_multi_control_system_Exp - Simulink
シミュレーション
デバッグ
モデル化
書式設定
ハードウェア
アプリ
ハードウェア ボード
Raspberry Pi
準備
ハードウェアで実行
結果の確認
配布
RL_multi_control_system_Exp
Controller
control_switch:1
On
Off
準備完了
3 件の警告を表示
175%
FixedStepDiscrete
00:00

考察

- 倒立が成功しない場合がある
- モーターの角度が±150[deg]を超えてしまう場合がある

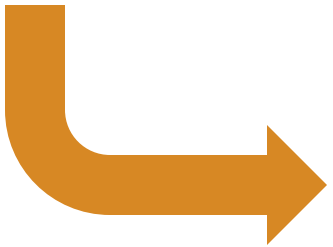
- 物理モデルの精度が低い、またはモデル化していない要素の影響が出ている可能性がある

- より安定して倒立を成功させるには、ロバスト性のある強化学習モデルを構築することが重要
- これまでにないアイデアや、発想の転換が必要

まとめ

まとめ

- 強化学習による制御について、重要な考え方を説明
- 強化学習を制御に適用する方法として、軌道生成制御を紹介
- 強化学習による軌道生成制御の設計手順を紹介
- マイコンに実装し、実機検証するワークフローを紹介



MATLABとSimulinkを使うと、アルゴリズム設計、モデリング、シミュレーション、コード生成、SIL/PIL、RCPを包括的に行うことができる！

参考資料

- Reinforcement Learning Toolbox 製品ページ
  - https://jp.mathworks.com/products/reinforcement-learning.html
- レファレンス・アプリケーション
  - https://www.mathworks.com/help/reinforcement-learning/examples.html
- MATLABおよびSimulinkによる強化学習 eBook
  - https://jp.mathworks.com/campaigns/offers/reinforcement-learning-with-matlab-ebook.html
- Reinforcement Learning ビデオシリーズ
  - https://jp.mathworks.com/videos/series/reinforcement-learning.html