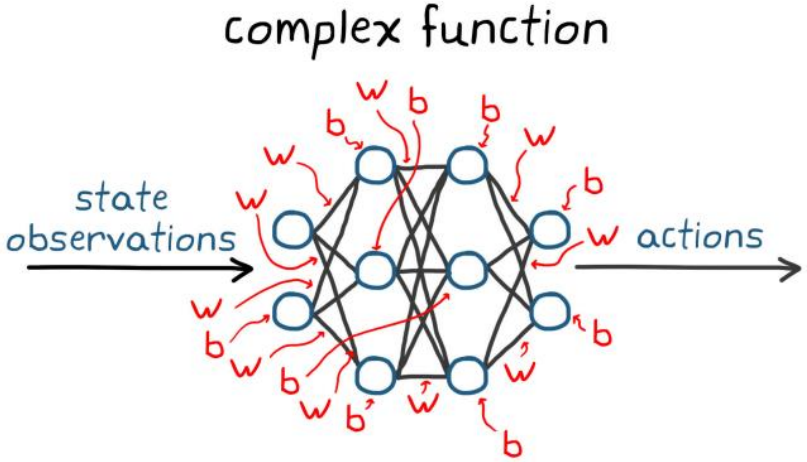
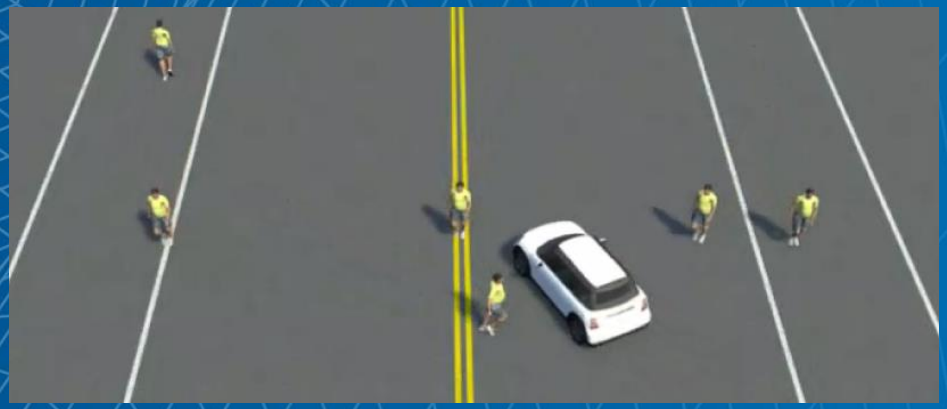
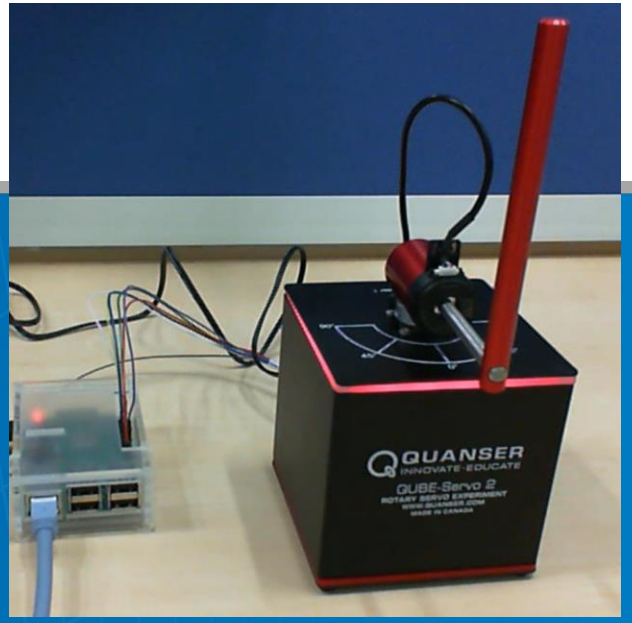
# Controls × AI
# ソリューション紹介




MATLAB EXPO 2023 Japan

MathWorks





# 目次

- 倒立振子を強化学習で倒立させる制御の設計、及び実機実装
- 強化学習とモデル予測制御を用いて、往来する歩行者を避けながら自律運転する制御を設計
- 地に足付けた制御設計機能
- 役に立つコンテンツ集

# 目次

- 倒立振子を強化学習で倒立させる制御の設計、及び実機実装

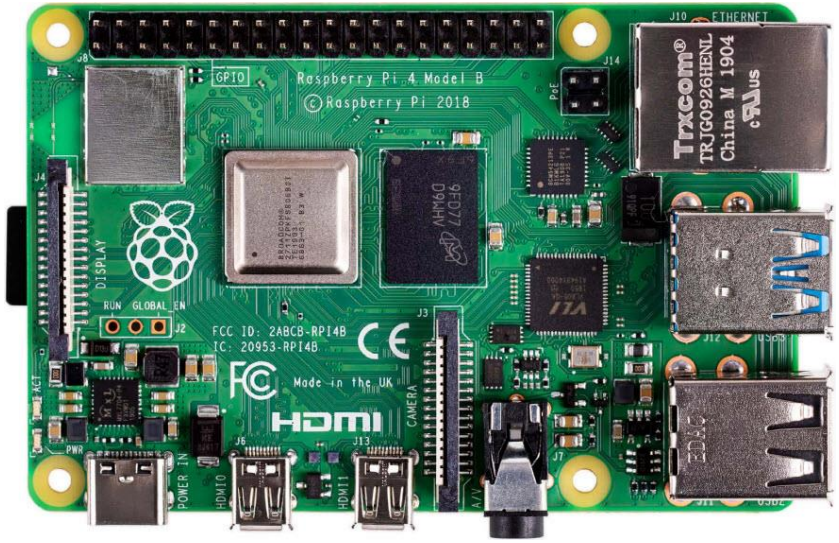
- 強化学習とモデル予測制御を用いて、往来する歩行者を避けながら自律運転する制御を設計

- 地に足付けた制御設計機能

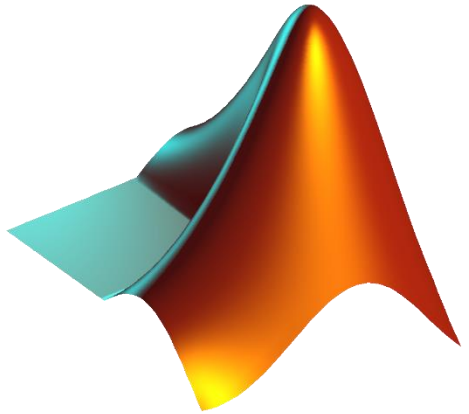
- 役に立つコンテンツ集

# Raspberry Pi と Reinforcement Learning Toolbox を用いた Quanser Qube（倒立振子）の制御


SPI



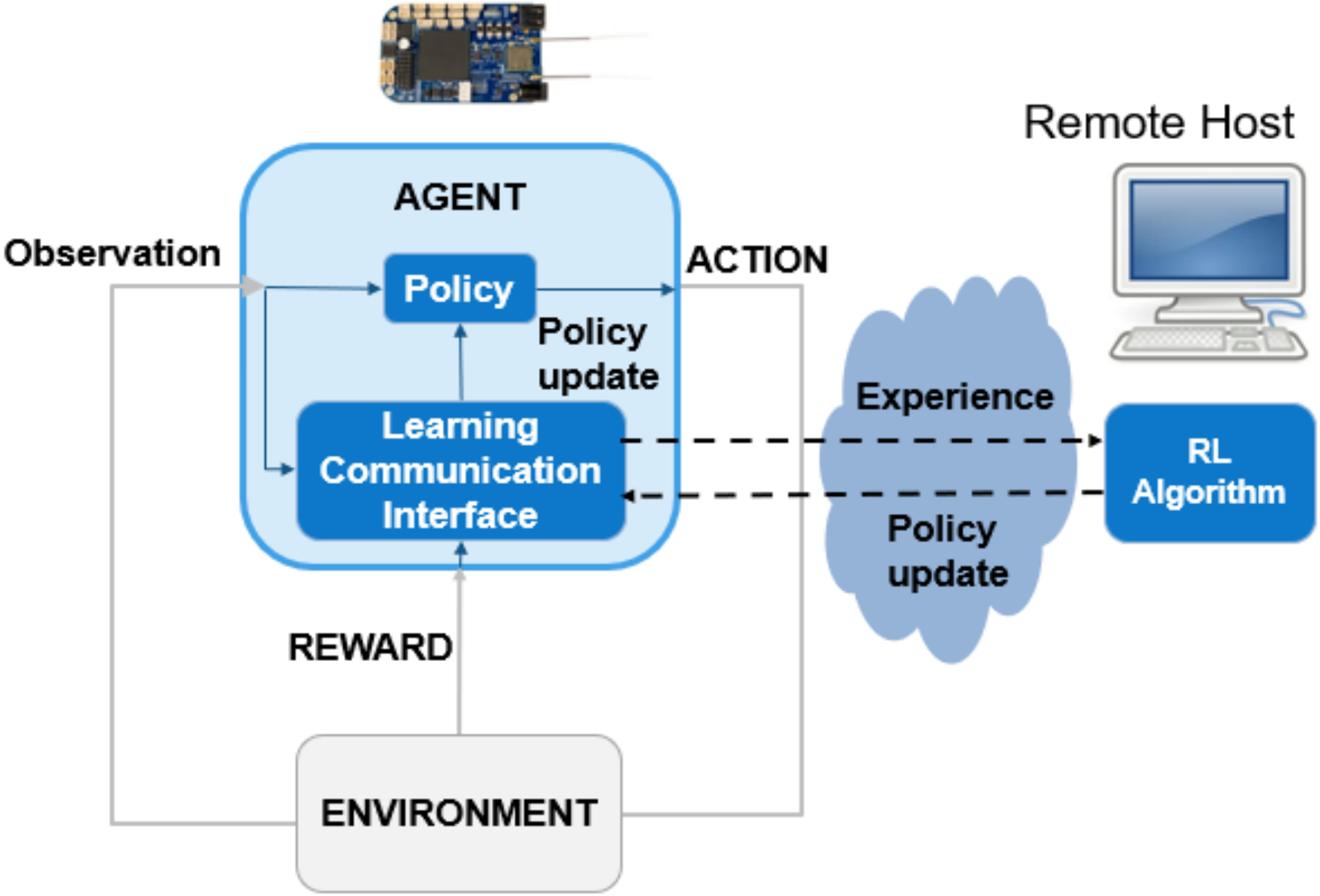

TCP/ROS



# 強化学習のシステム構成

本デモでは、実機を用いて強化学習を行う。
学習アルゴリズムはPCで実行し、学習データを実機から取得する。

システムは右のような構成になる。



# 目的

- 振り子を安定均衡($\phi = \pi$)から不安定均衡($\phi = 0$)へ振り上げるポリシーを作る
  - 劣駆動システム
  - 4状態1入力（モータ回路ダイナミクスは無視）
  - 内蔵ストッパーとの衝突を避けるため、ベース角度を拘束している ($|\theta| \leq$"thresh")
  - ポリシーは5秒経過する前に倒立を達成すること

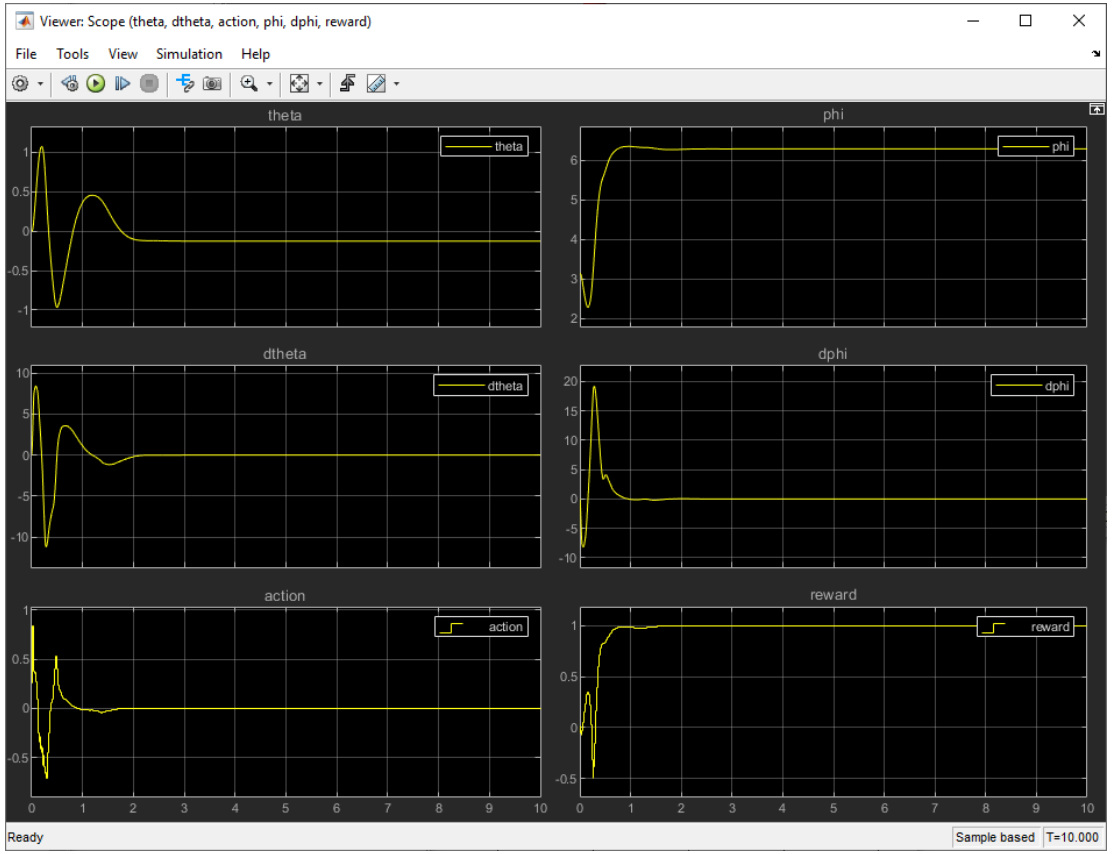
| | |
|---|---|
| $\theta$ | ベースの角度 |
| $\phi$ | 振り子の角度 |
| $v$ | モーター電圧 |

# SACエージェントを設計

- **観測**: 状態 $x = [\theta \quad \dot{\theta} \quad \phi \quad \dot{\phi}]^T$ の式と前回の行動
  - $s_k = [\sin(\theta_k) \quad \cos(\theta_k) \quad \dot{\theta}_k \quad \sin(\phi_k) \quad \cos(\phi_k) \quad \dot{\phi}_k \quad a_{k-1}]^T$
- **行動**: 規格化されたモーター電圧
  - $a_k = v_k / v_{\max}$
- **報酬**: 二次形式のペナルティとシミュレーションが長時間継続した場合の報酬
  - $r_k = (\text{isDone} == 0) * 1 \; - x_k^T Q x_k \; - a_{k-1}^T R a_{k-1}$
- **停止条件**: ベース角度が逸脱した場合
  - $\text{isDone} = \; |\theta| > \text{thresh}$
- **ネットワーク構造**: デフォルトのactor, critic
  - 64 の隠れ層
  - ReLU活性化関数
  - Actionの範囲は[-1 1]

# シミュレーション環境でSACを学習

- 4ワーカーを用いて500エピソード分を学習(所要時間は3時間以内)
- 200エピソード時点で十分な性能が得られた

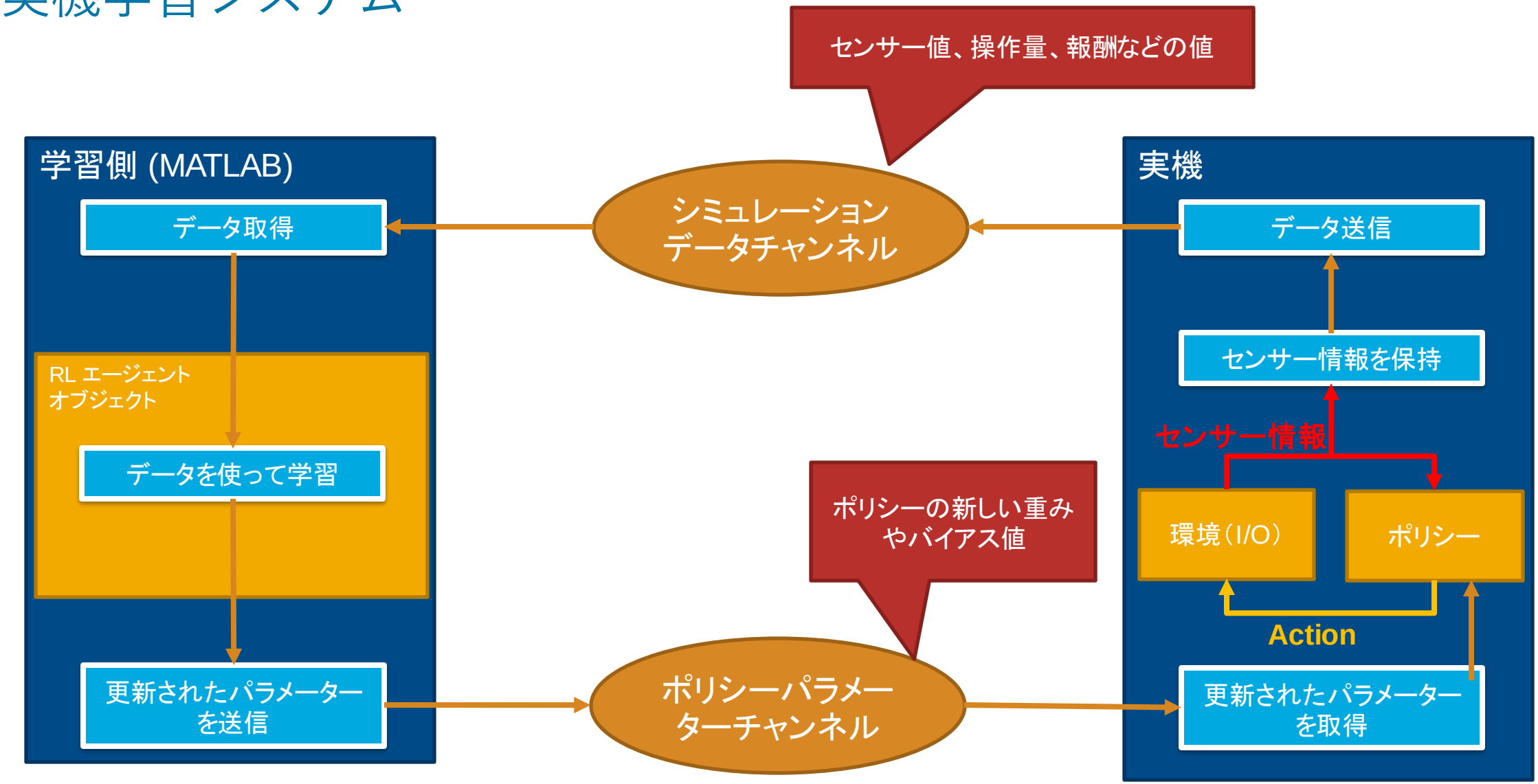



# 通信システムの概要

- SPI通信によりRaspberry Pi 4 Model BとQubeが接続されている
- MATLAB, Simulink は以下を介してRaspberry Piに接続されている
  - TCP
  - ROS

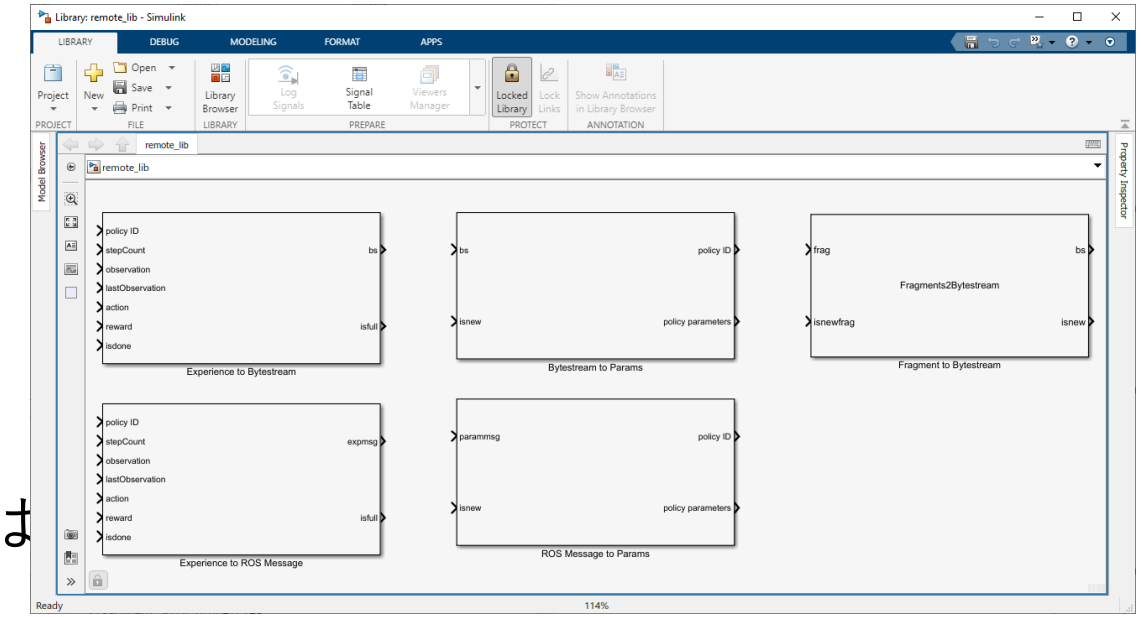
- これにより、実機を対象に学習するシステムを実現

# 実機学習システム


センサー値、操作量、報酬などの値
学習側 (MATLAB)
データ取得
RL エージェント
オブジェクト
データを使って学習
更新されたパラメーター
を送信
シミュレーション
データチャンネル
ポリシーの新しい重み
やバイアス値
ポリシーパラメー
ターチャンネル
実機
データ送信
センサー情報を保持
センサー情報
環境（I/O）
ポリシー
Action
更新されたパラメーター
を取得

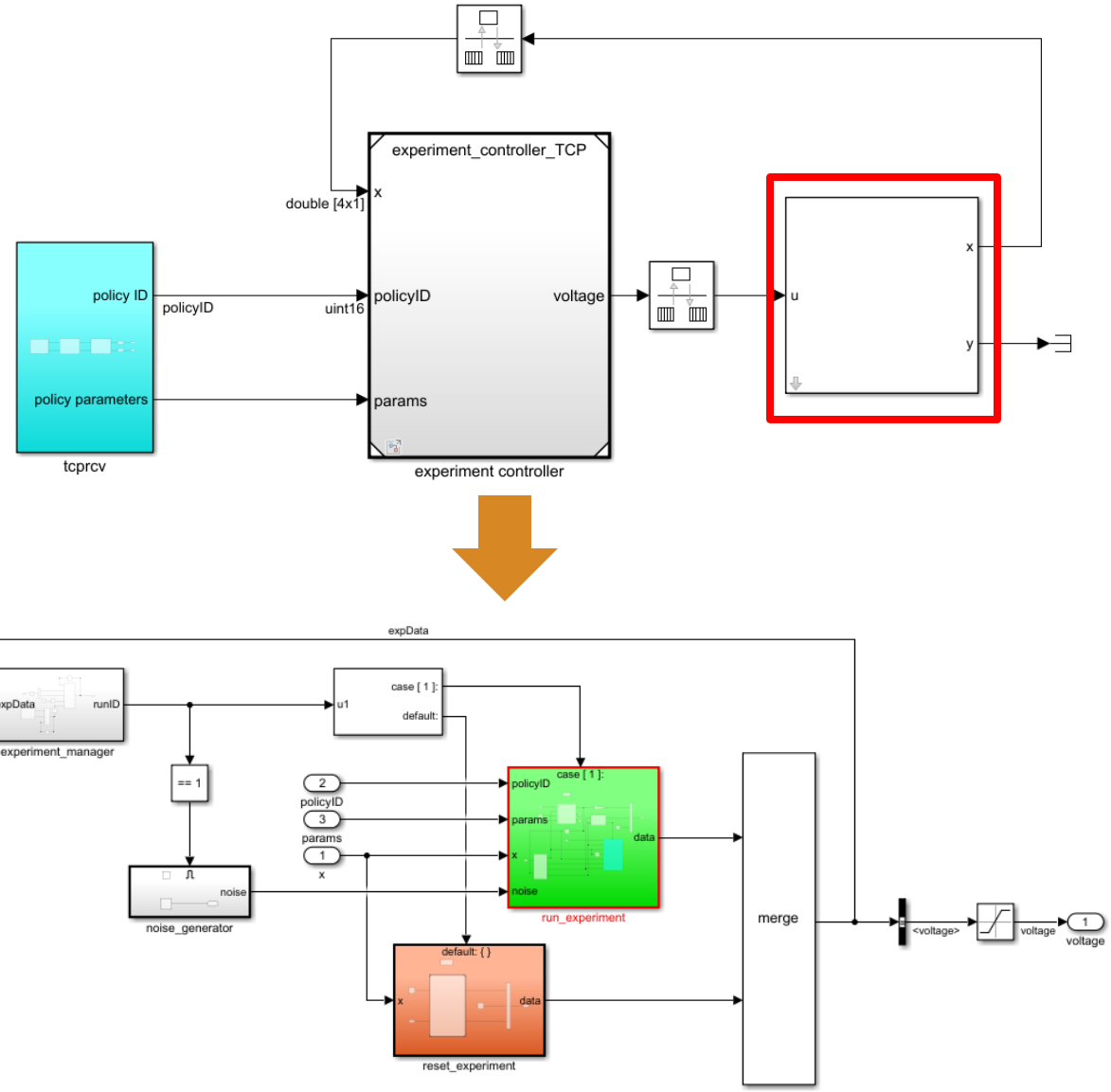
# 実機学習のセットアップ

- TCPとROS通信はメッセージのシリアライズ化が必要である
  - ROS
    - ROS .msg ファイルを定義
    - パッケージをコンパイル
    - ROSによるシリアライズ
  - TCP
    - データはバイト列に変換される
- 実機計測データやポリシーの重みデータは自動的にシリアライズ化される



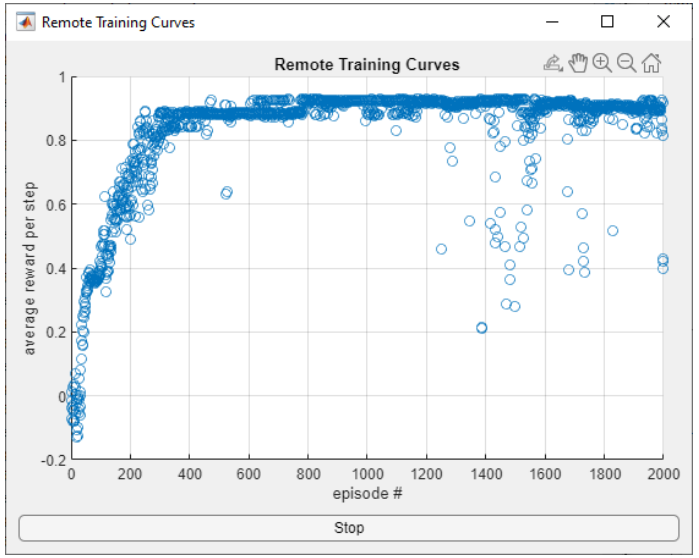
# Raspberry Pi に実装するモデル

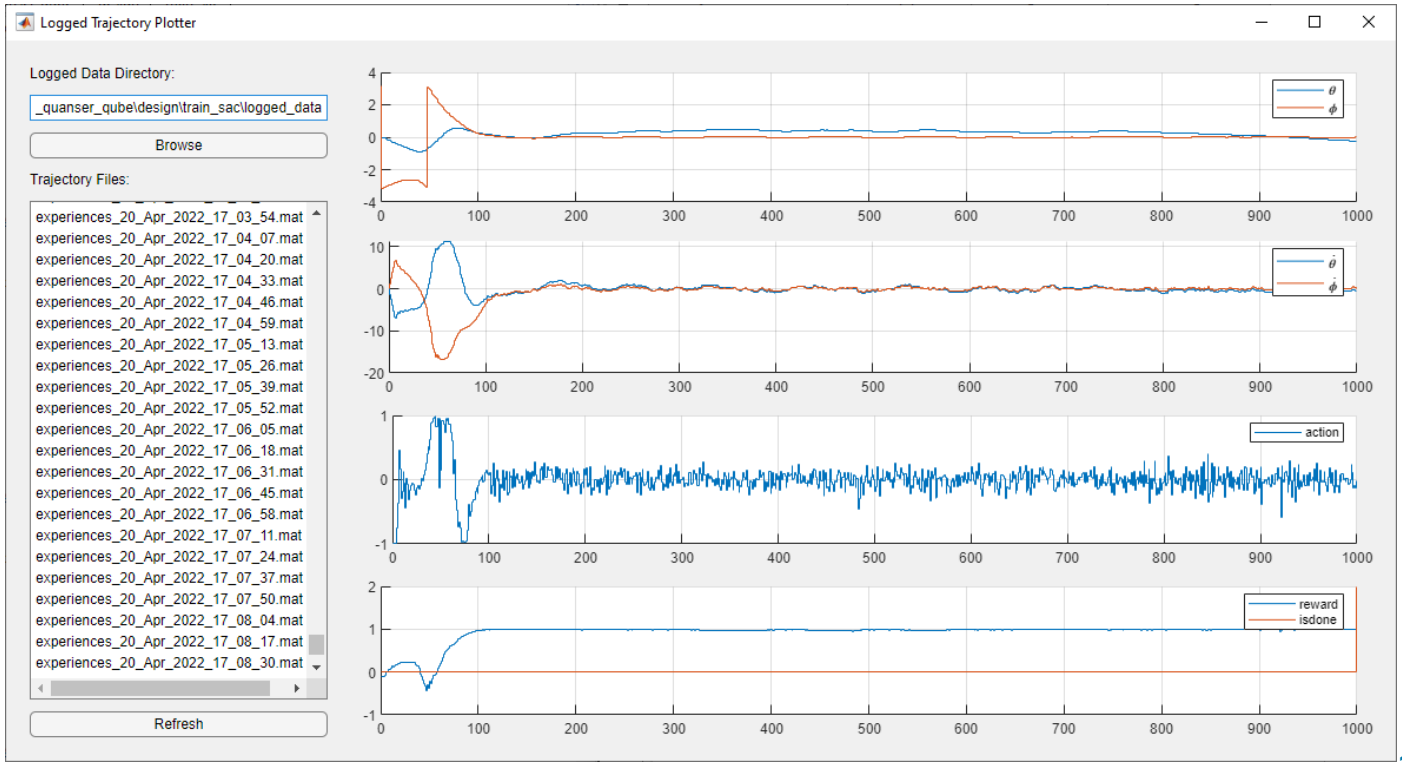
- TCPクライアントからポリシーの重み情報を取得
- 実機の制御器
  - Reset
    - システムを休止状態に持っていく
    - 実験データは送信されない
  - Run
    - 制御を実行し、センサー値などをTCPで送信する
- Simulinkモデルで設計し、そのままRaspberry Piへ実装



# SACを実機学習させた結果

- 学習は非同期で行われた
- 学習側（PC）は手記的に新しいパラメーターを送信
- 2000エピソード分を学習
- 300エピソード時点で十分良い性能が得られた



# 実機学習の様子

# 目次

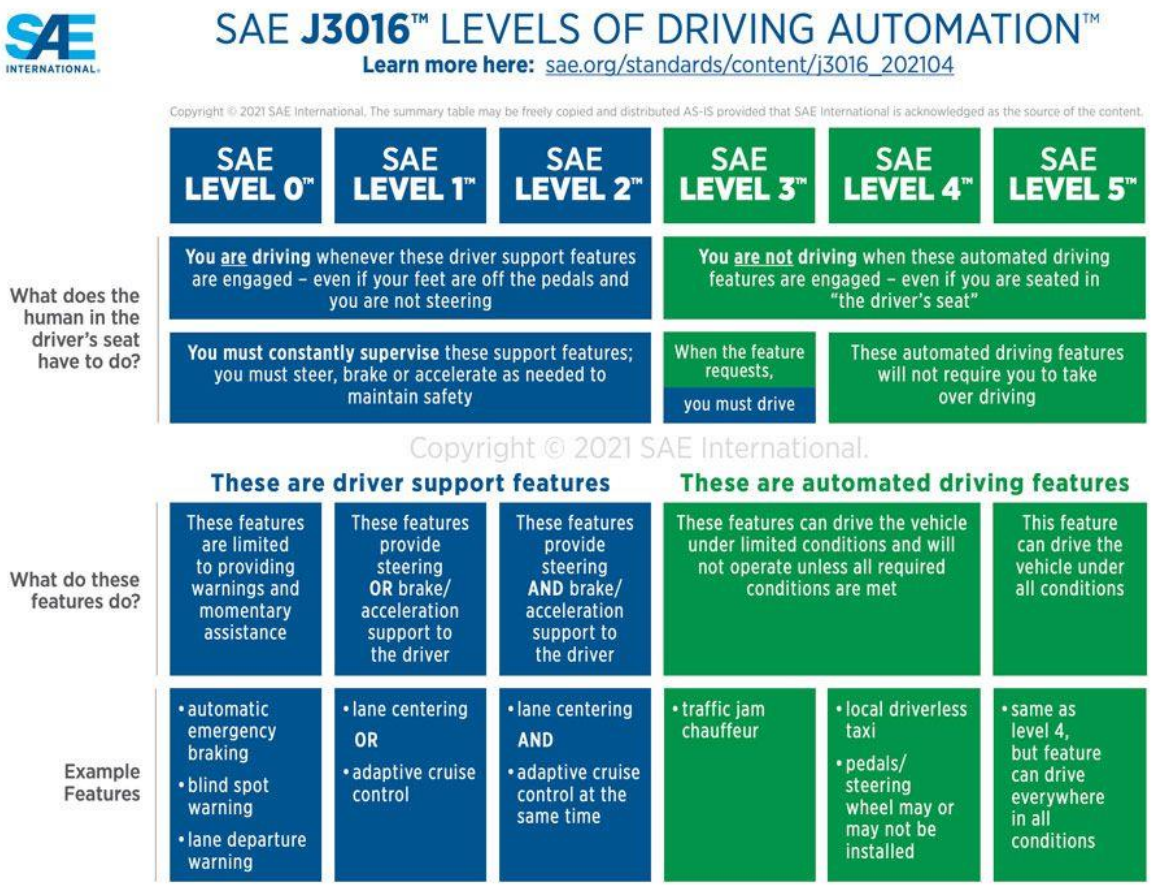
- 倒立振子を強化学習で倒立させる制御の設計、及び実機実装
- 強化学習とモデル予測制御を用いて、往来する歩行者を避けながら自律運転する制御を設計
- 地に足付けた制御設計機能
- 役に立つコンテンツ集

# 自動運転

ドライバー不在で車を安全に運転できる自動運転が注目されており、今では自動車メーカーをはじめIT企業なども自動運転技術の開発へ参入している。

自動運転には5段階のレベルがあり、SAE(Society of Automotive Engineers)によって定義されている。

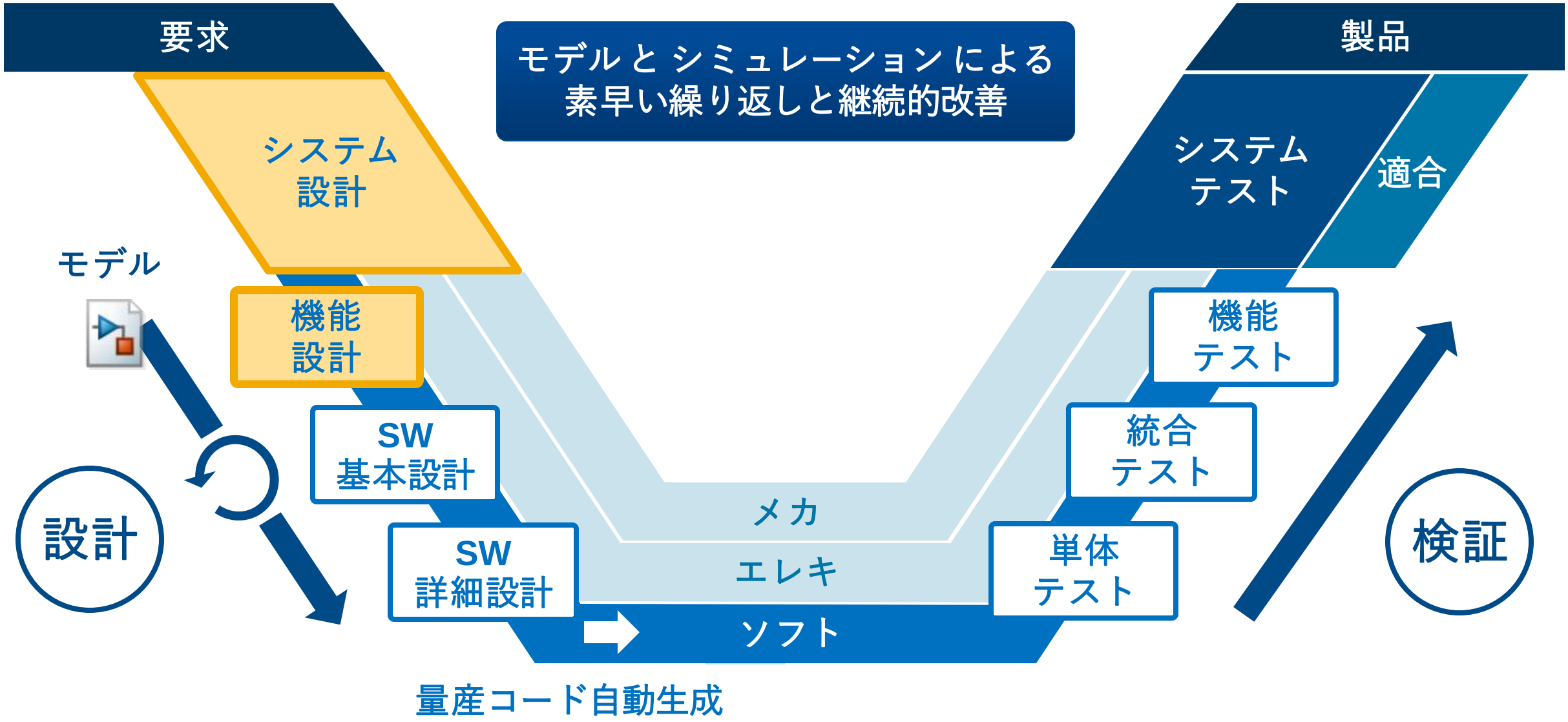
最大のレベル5を実現するには、大規模で複雑なシステムを設計しなければならない。



# 大規模で複雑なシステムを設計するには

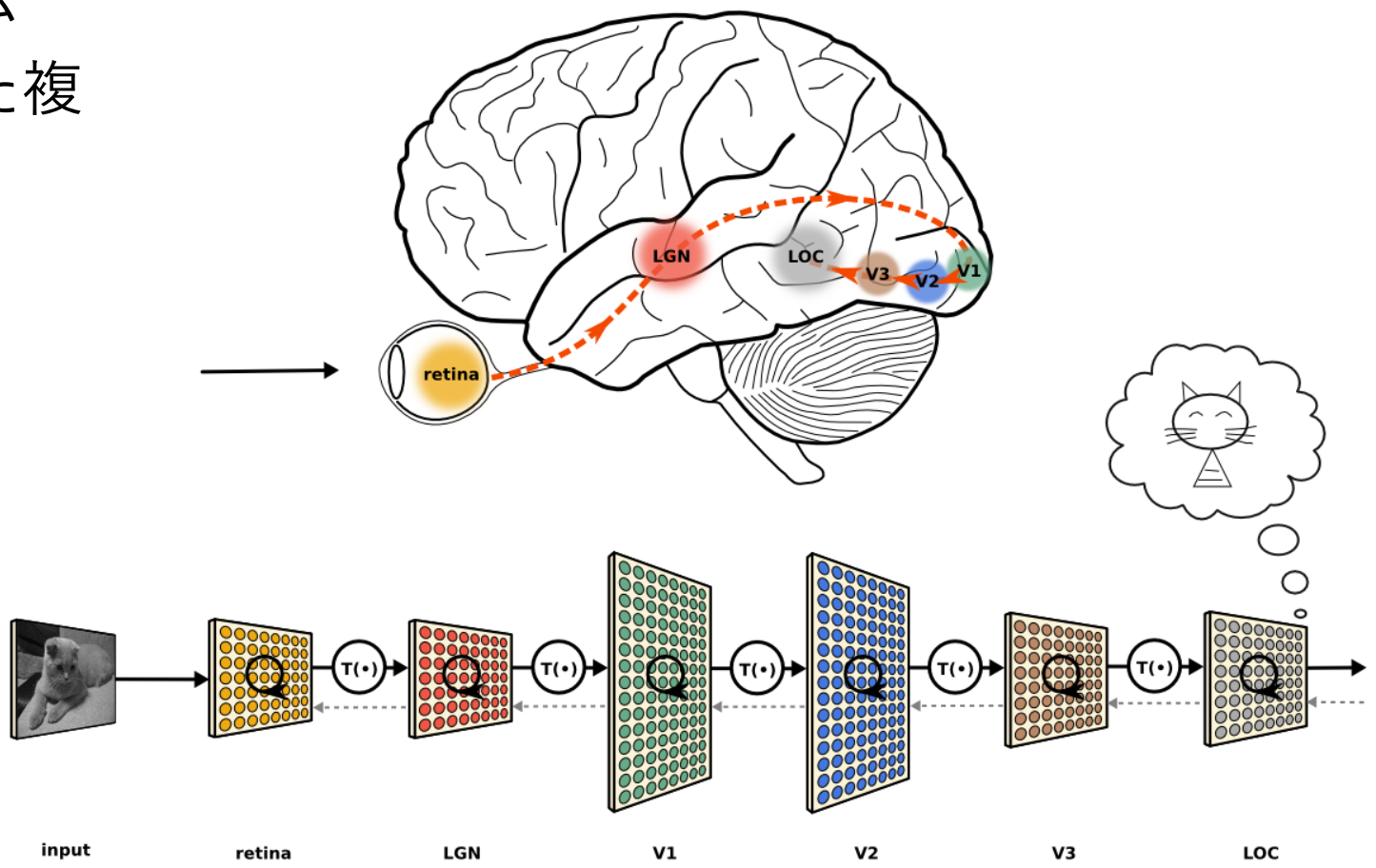
- モデルベースデザイン
- AI（深層学習）
- 疎結合と抽象化ができる機能分割

# モデルベースデザイン


要求
製品
モデルと シミュレーション による
素早い繰り返しと継続的改善
システム
設計
システム
テスト
適合
モデル
機能
設計
機能
テスト
SW
基本設計
統合
テスト
設計
メカ
エレキ
SW
詳細設計
単体
テスト
検証
ソフト
量産コード自動生成

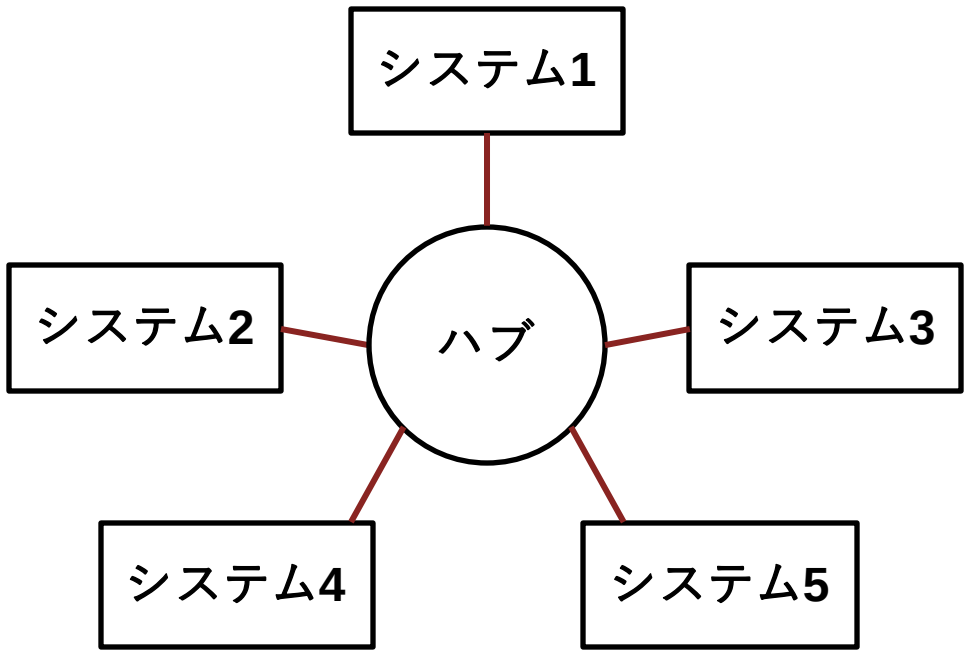
# AI（深層学習）

- 深い階層のニューラルネットワークを用いた分類・回帰アルゴリズム
- 従来の手法では実現できなかった複雑なタスクを実行できる



# 疎結合

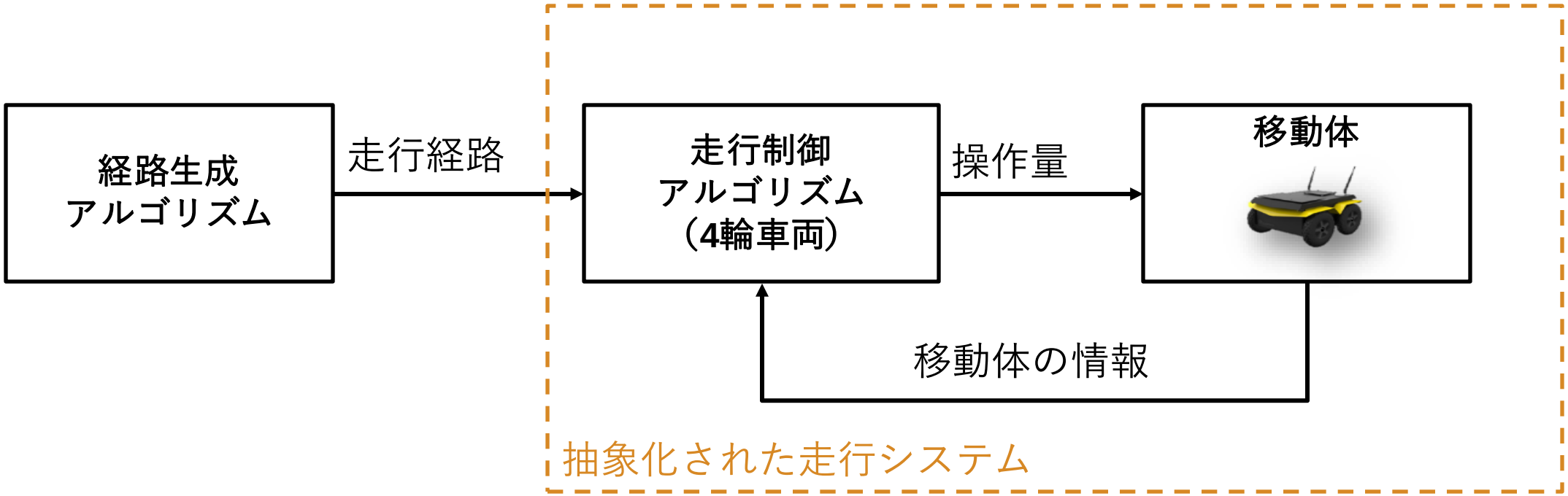
- 大規模システムは、多数のサブシステムを組み合わせて作られる
- サブシステムを変更した時、その他のサブシステムへの影響が少ない状態を「疎結合」と呼ぶ




例えば、左の図のようにシステム同士が繋がっていれば、一つのシステムに変更があったり消滅しても、他のシステムは変わりなく動作し続けることができる。

# 抽象化

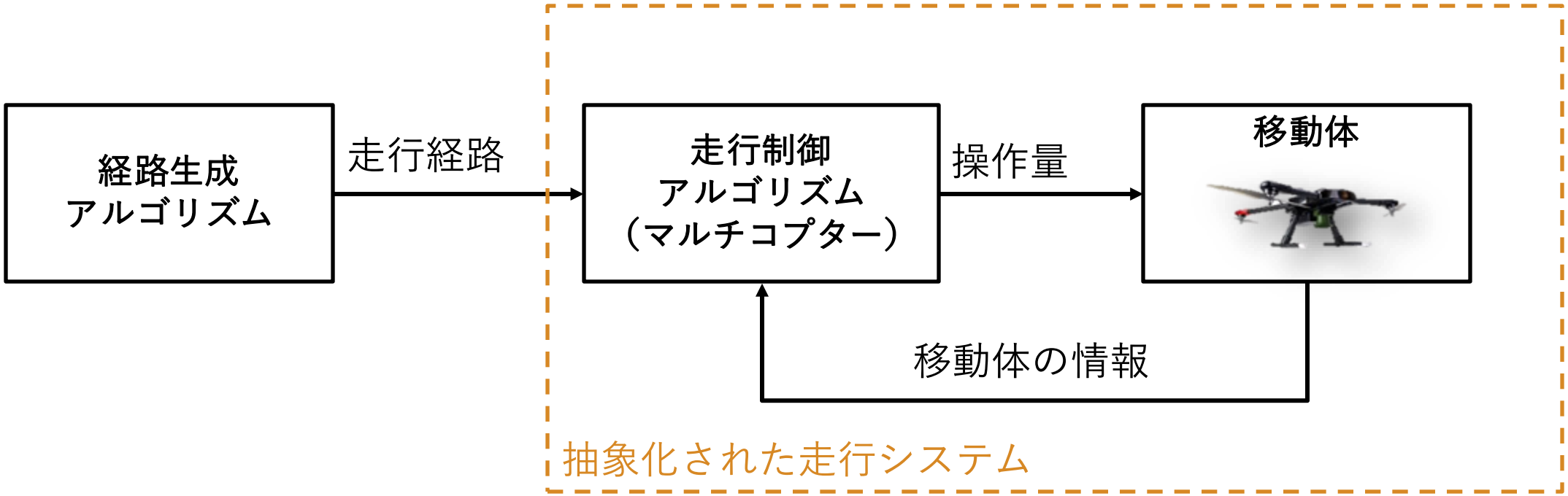
- 上位のシステムから下位のシステムを操作する時、下位のシステム構造が変化しても上位システムを変更しなくてもよい状態を「抽象化」と呼ぶ




例えば、移動体が変わればそれに合わせて走行制御アルゴリズムを変更する必要があるが、「移動する」という概念が同じであれば、経路生成アルゴリズムは変更しなくてよい。

# 抽象化

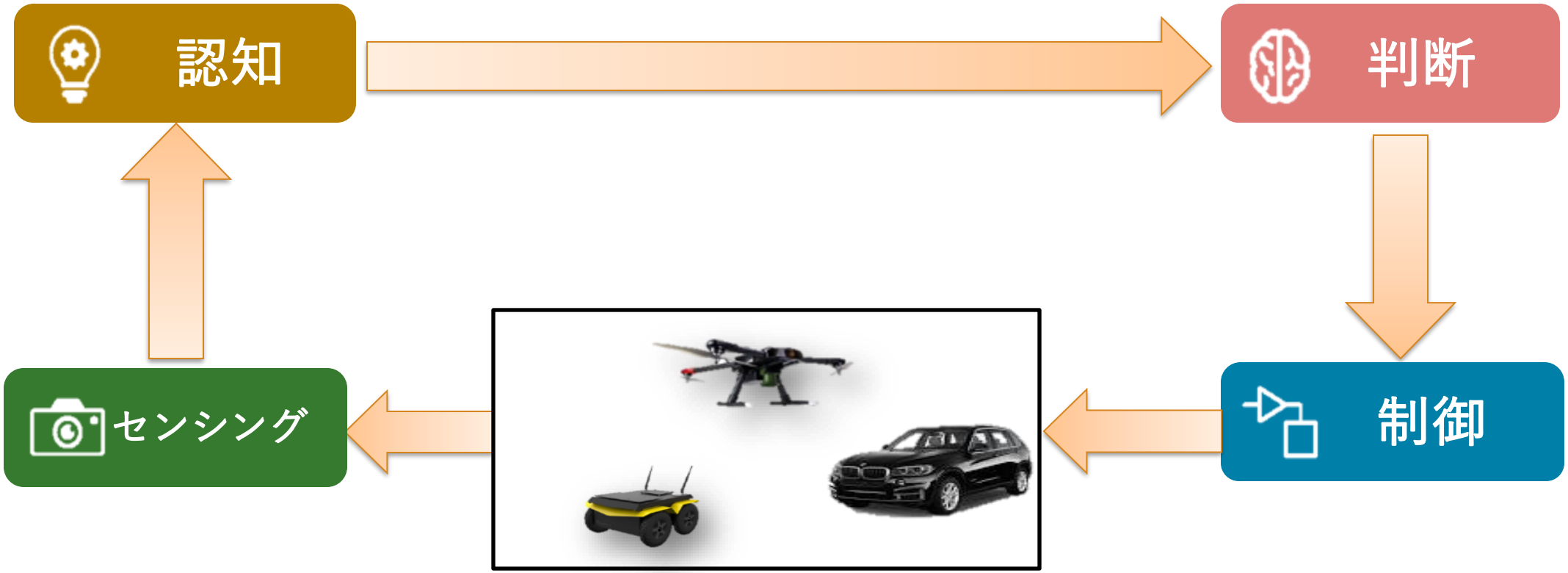
- 上位のシステムから下位のシステムを操作する時、下位のシステム構造が変化しても上位システムを変更しなくてもよい状態を「抽象化」と呼ぶ




例えば、移動体が変わればそれに合わせて走行制御アルゴリズムを変更する必要があるが、「移動する」という概念が同じであれば、経路生成アルゴリズムは変更しなくてよい。

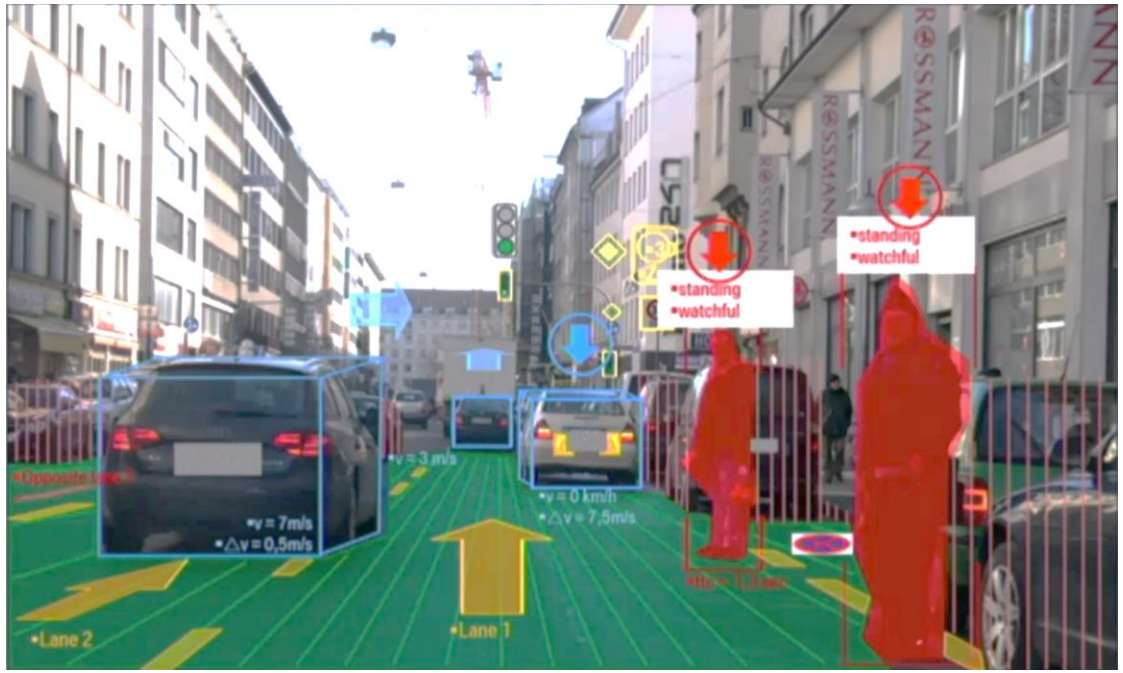
# 自律制御システム

- 自動運転のシステムは複雑なフィードバック制御システムである
- 自動運転に適した機能分割と抽象化の方法として、「認知」「判断」「制御」に分ける方法が推奨される



# ADAS・自動運転システムの開発 – 認知

認知



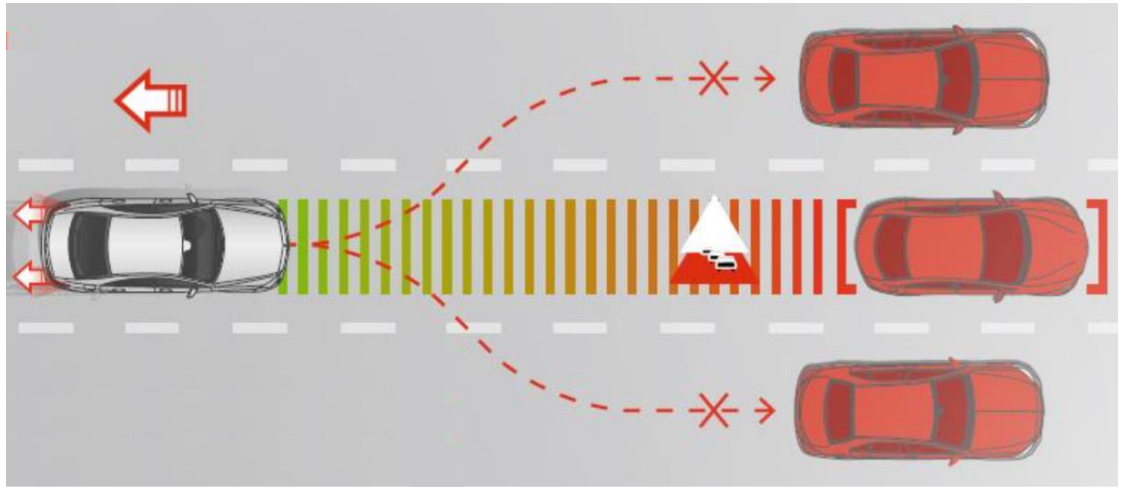

## 関連アルゴリズム・タスク代表例

- 画像処理
- 点群データ処理
- 自己位置推定、SLAM
- センサーフュージョン・トラッキング
- Deep Learning
- レーダー信号処理
- 組み込み機器への実装

# ADAS・自動運転システムの開発 – 判断

認知

判断



## 関連アルゴリズム・タスク代表例

- 占有グリッドマップ、コストマップ生成
- ローカルパスプランニング
- グローバルパスプランニング
- HD Map連携
- 車両軌道生成
- 組み込み機器への実装

# ADAS・自動運転システムの開発 – 制御

認知

判断

制御

## 関連アルゴリズム・タスク代表例

- ログデータ、CANバス接続
- 車両ダイナミクス
- レグレッションテスト
- パス追従制御
- モデル予測制御
- 量産コード生成、実装
- AUTOSAR準拠
- ISO26262

# 「認知」「判断」「制御」に分ける理由

- 「認知」「判断」「制御」それぞれに求められる要件が異なり、必要とする技術分野も異なる
- 分けることにより、役割分担を明確化できる

## 認知

- Lidarやカメラなどを用いて環境を認識する
- 現在位置を推定する
- 障害物の位置を推定する

## 判断

- 障害物や自身の位置情報から、次の行動を決める
- 最適な走行経路を算出する

## 制御

- 目標経路を指令値、自身の位置情報を制御量として操作量を算出する
- リアルタイムのフィードバック制御を行う

# 「認知」「判断」「制御」に分ける理由

- 「認知」「判断」「制御」のそれぞれを独立して設計することができる
- 必要に応じて簡易化することで開発期間を短縮できる

| 認知 | 判断 | 制御 |
| --- | --- | --- |
| 環境情報を完全に把握できる理想モデル | 最適経路を完全に把握できる理想モデル | 指令値軌道に完全に追従できる理想モデル |
| LIDAR、カメラなどを用いて環境認識を行うモデル | 経路推定アルゴリズムによって最適経路を計算するモデル | 車両の物理挙動と制御器の応答を計算するモデル |

# 「認知」「判断」「制御」に分ける理由

- コンセプトを示すための理想システムを作る場合

# 「認知」「判断」「制御」に分ける理由

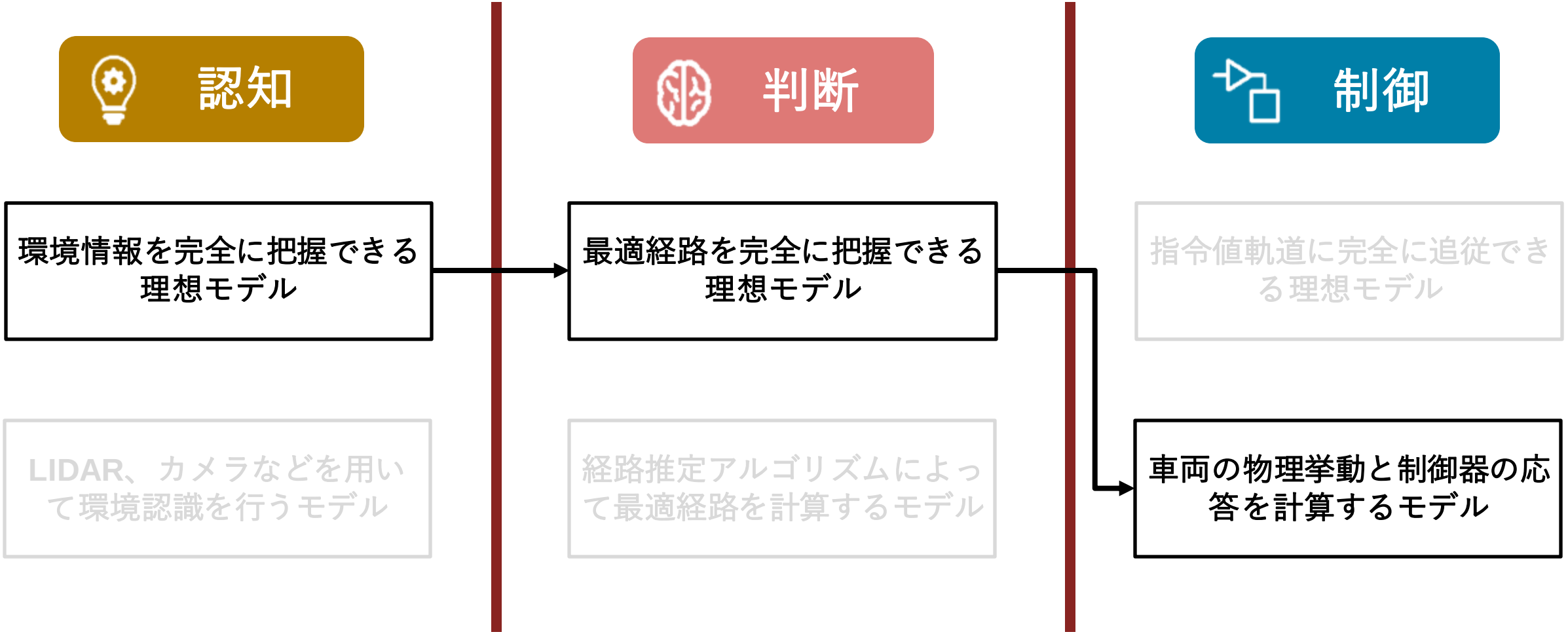
- LIDAR、カメラのセンサーフュージョン機能を設計・検証するシステムを作る場合

# 「認知」「判断」「制御」に分ける理由

- 経路生成アルゴリズムを設計・検証するシステムを作る場合

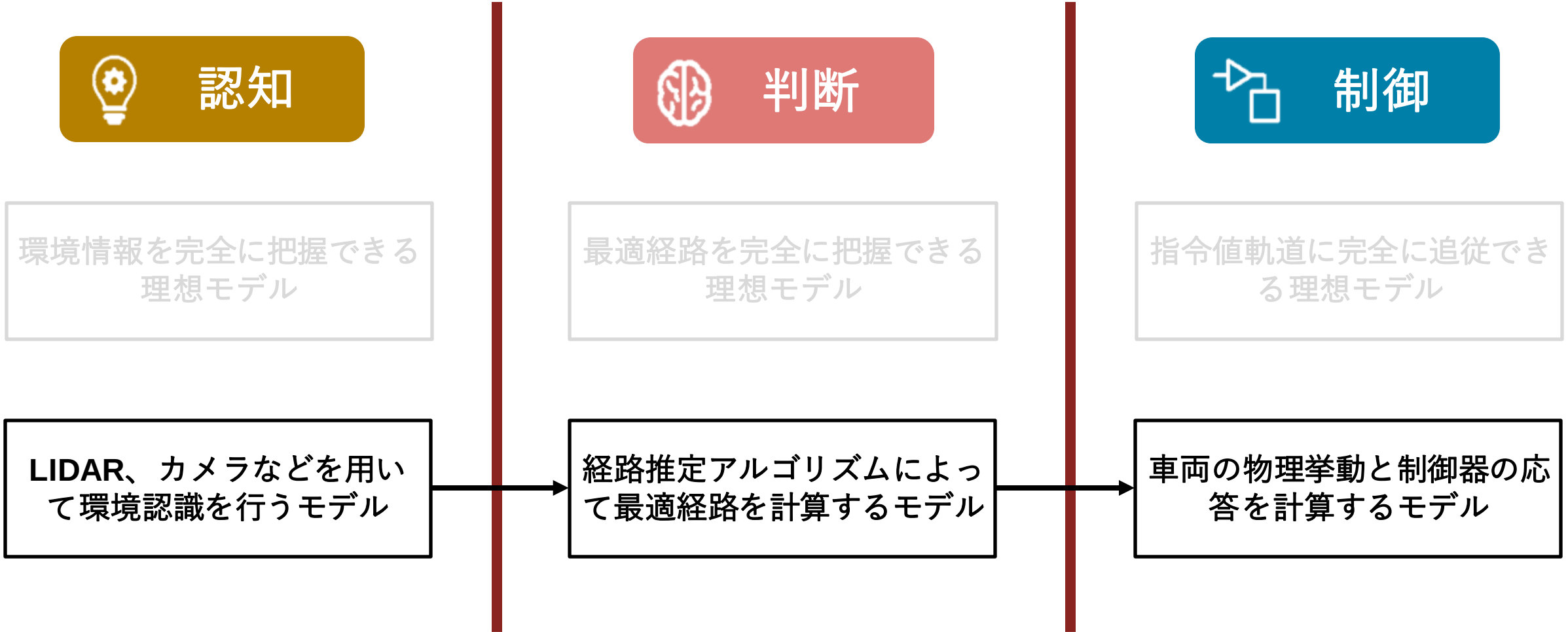
# 「認知」「判断」「制御」に分ける理由

- 走行制御アルゴリズムを設計・検証するシステムを作る場合



# 「認知」「判断」「制御」に分ける理由

- 設計した全てのモデルが機能することを検証するシステムを作る場合（完成版システム）

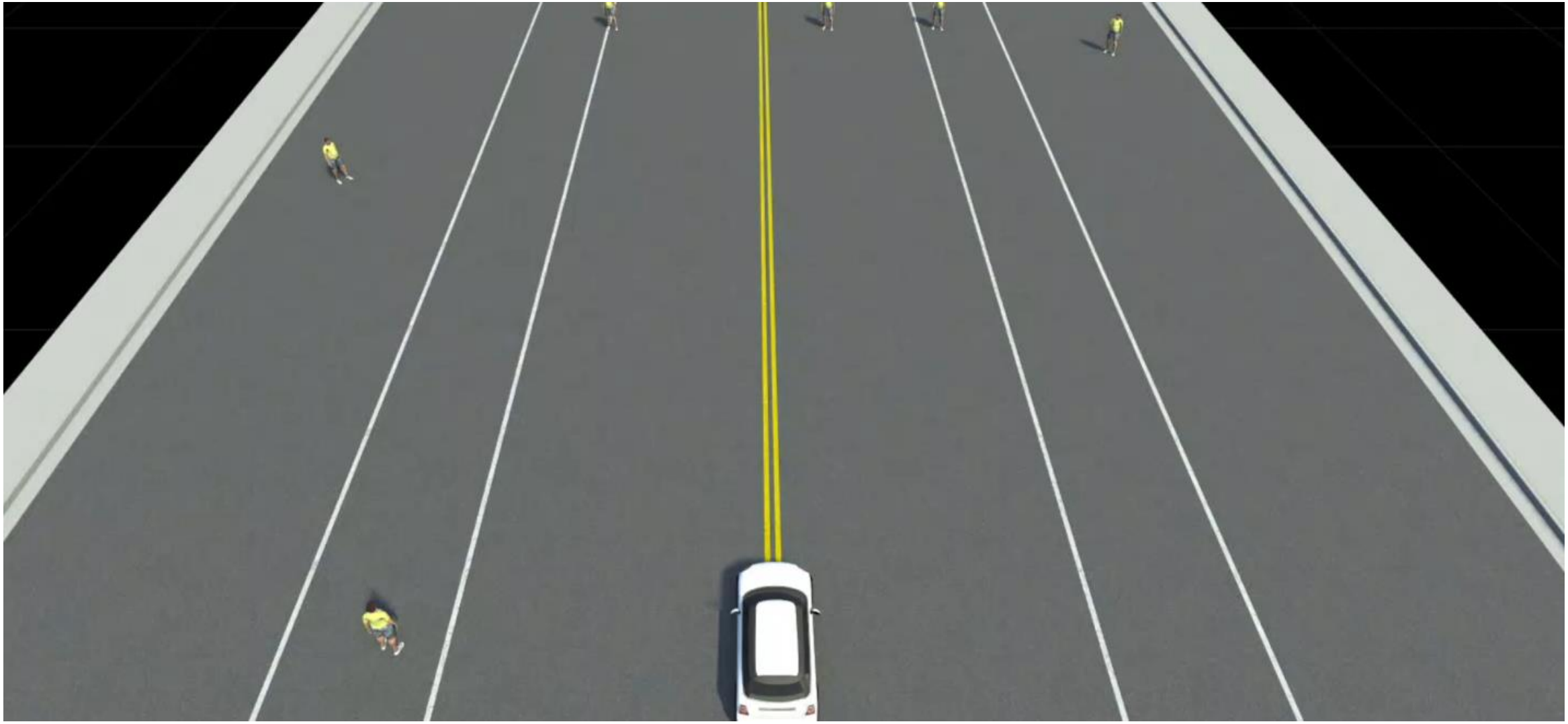


# 歩行者と車両の共存

例えば、歩行者と車両が同じ領域を同時に往来する場合、

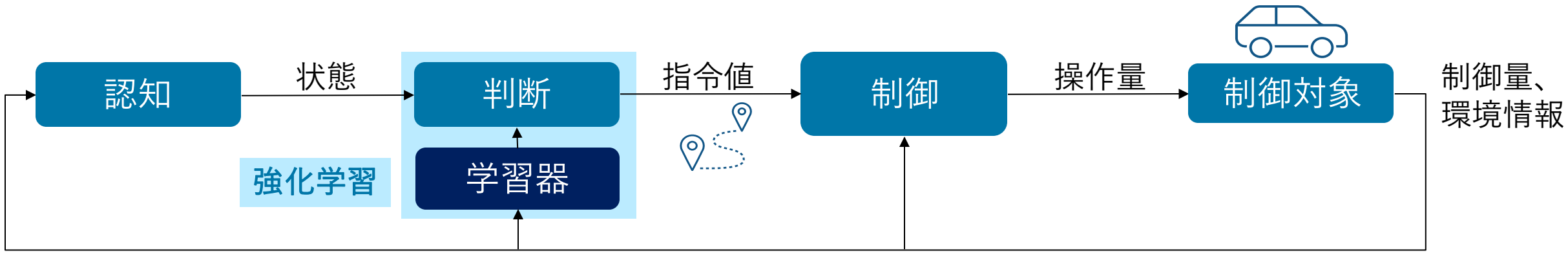
# 歩行者と車両の共存

ゴールへ向かいつつも、人を避ける運転操作が必要となる。

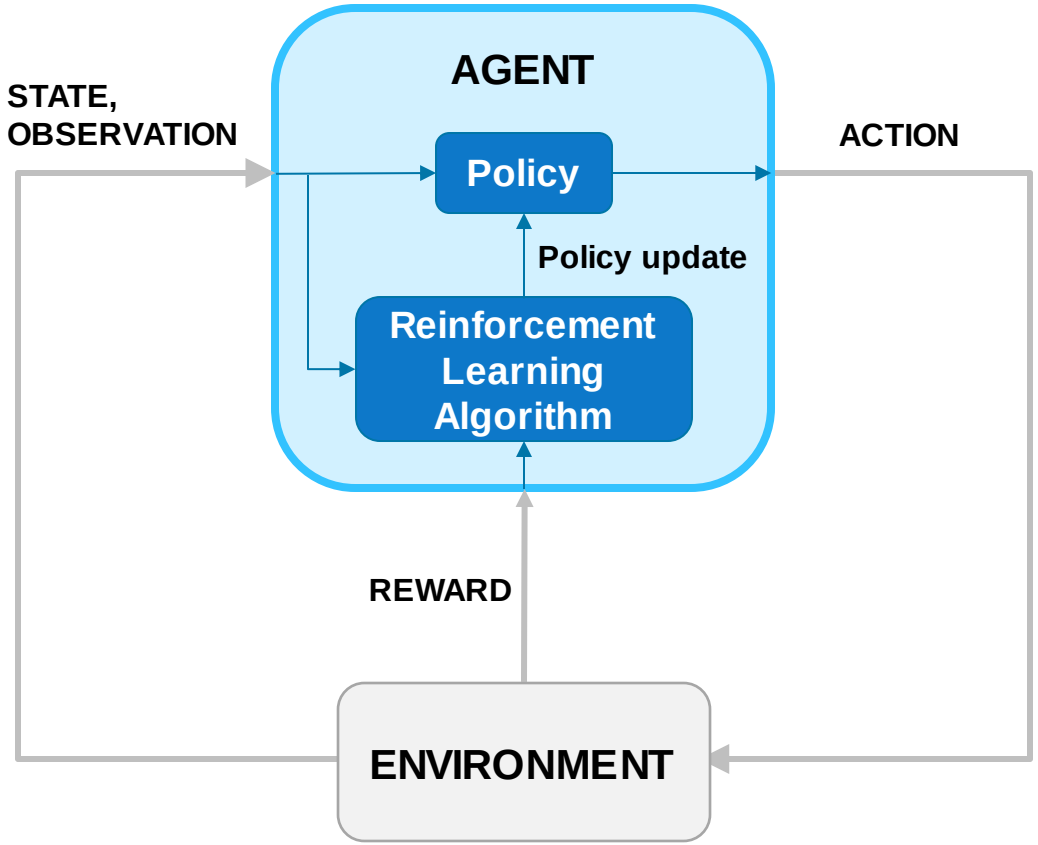


# コンセプト

- 今回は強化学習を「判断」、モデル予測制御（MPC）を「制御」に用い、それぞれのアルゴリズムを設計する
- 「認知」のシステムは理想モデルを用いる



# 強化学習　コンセプト




- 車両コントローラはどのように走るかを学習する
  - (**agent**)
- 車両周囲の障害物位置、元々の走行経路
  - (**state, observation**)
- 車両走行や環境変化を表現する
  - (**environment**)
- 走行経路の指令値軌道
  - (**action**)
- (state)から次の(action)を生成する
  - (**policy**)
- あるべき走行に近ければ高い報酬
  - (**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

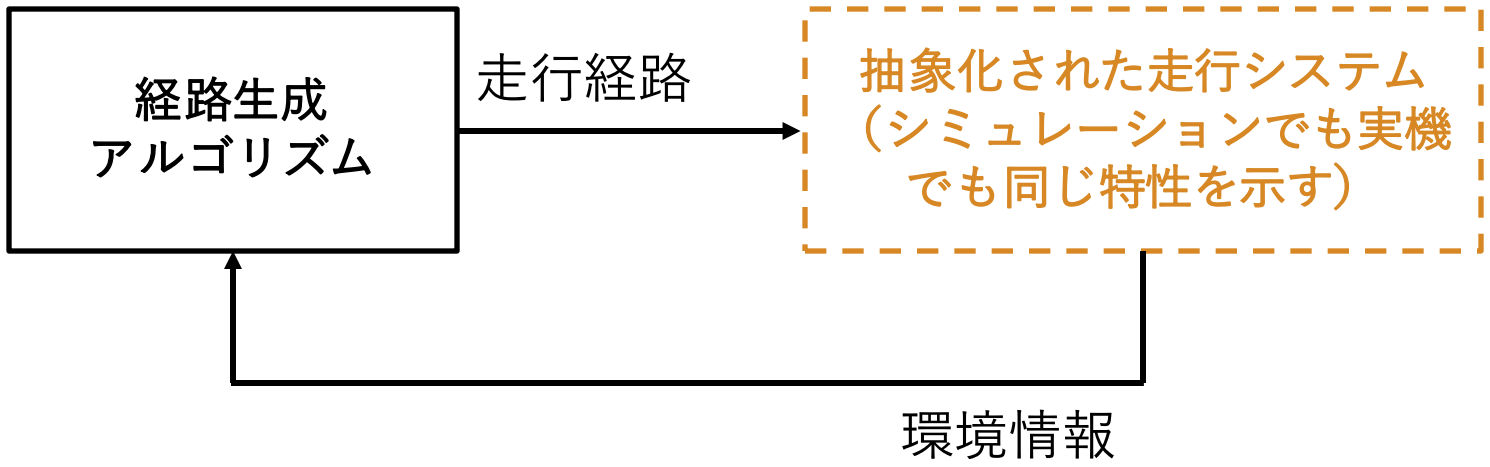
# 強化学習を「判断」に用いることのメリット

- 「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

- 「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
- そのため、応答性能や安定性などの、制御システム特有の課題を「制御」の要件として設定することにより、「判断」では強化学習が扱いやすい要件のみを設定できる

# 強化学習を「判断」に用いることのメリット

- 強化学習を「制御」に用いる場合、シミュレータ（sim）上で学習させた方策を現実世界（real）に持ってくる「sim2real」が必要であるが、プラントモデルの精度が低いと現実世界で想定通り動作しない
- しかし「制御」にモデル予測制御を用い、シミュレーションと現実世界の誤差を十分に小さくすることができれば、「判断」から見た時にシミュレーションと現実世界が同じとなる
- その上で、強化学習を「判断」に用いることで「sim2real」の課題を解消できる

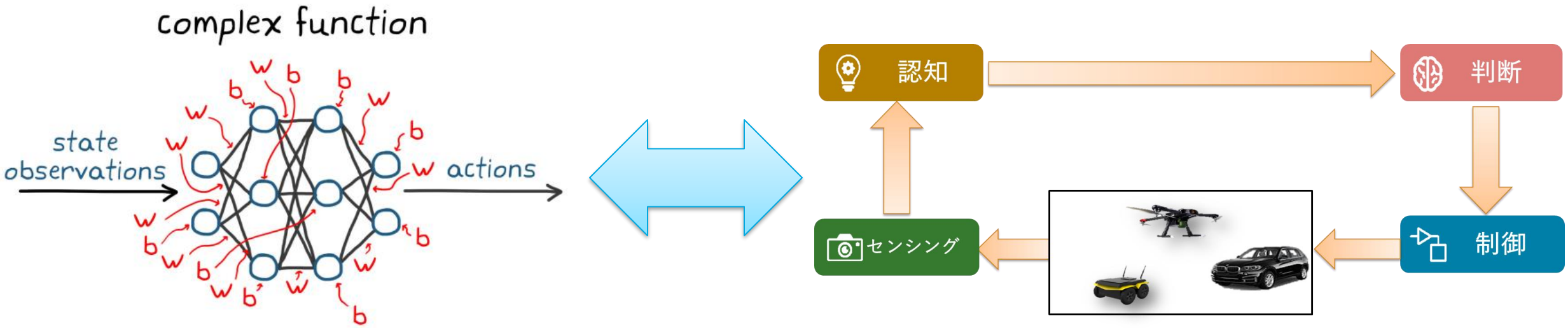


# 深層強化学習の課題1　深層ネットワークの説明可能性の課題

「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。

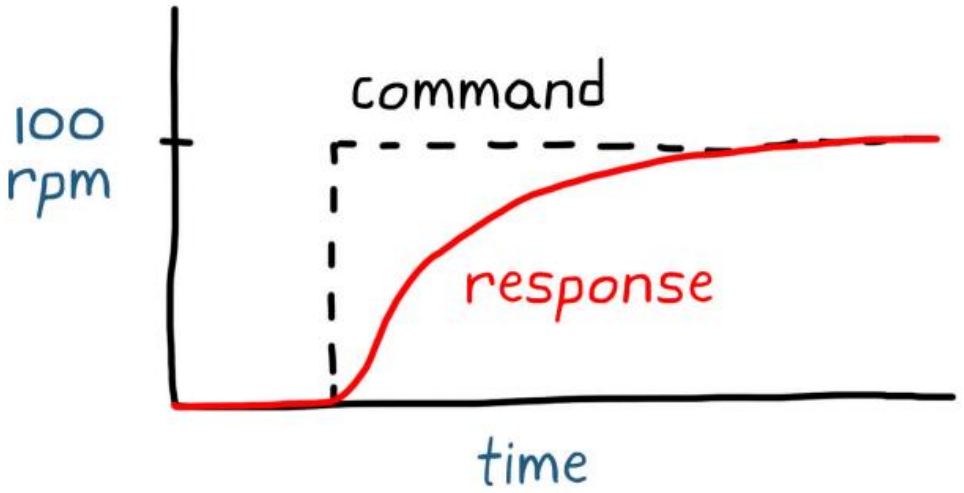


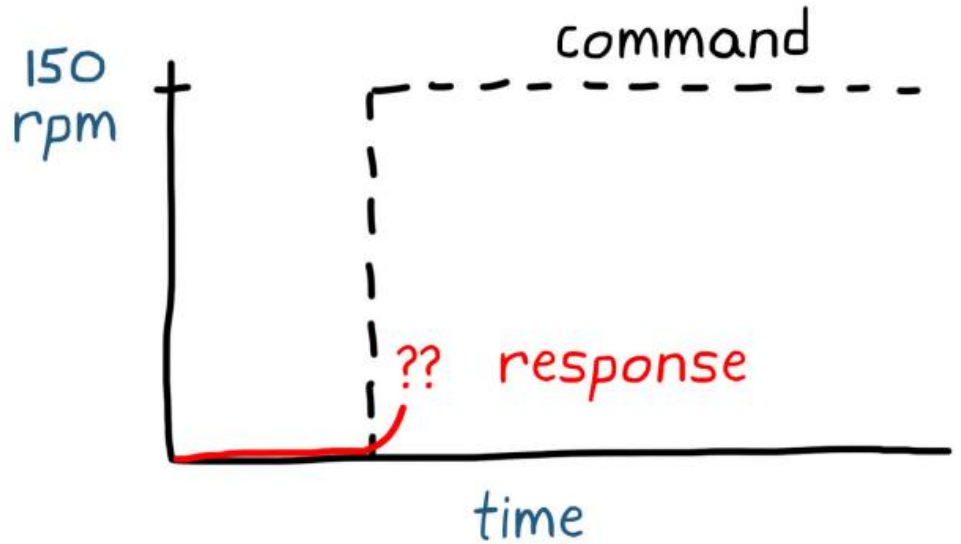
# 深層強化学習の課題2　深層ネットワークの強い非線形性の課題

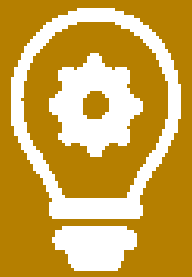
深層ネットワークは非線形である。従って、学習した範囲外の入力や状態になった場合に、何が起こるのかを予想することができない。

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。





# 認知

# 環境の認識方法

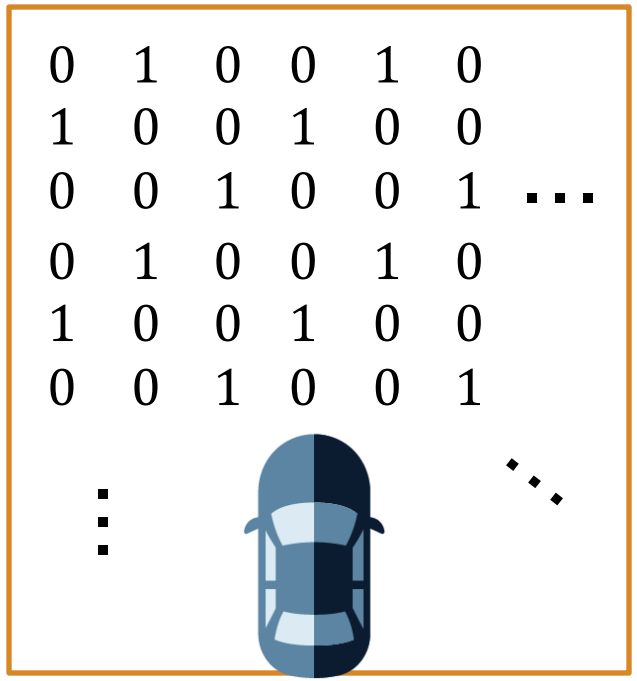
- 環境を2次元のマップで認識し、人が存在する箇所を1、そうでない箇所を0として行列表現する
- 本来はLIDARやカメラなどを用いて自己位置や歩行者位置を推定するが、ここでは理想的に情報を得ることができるものとする

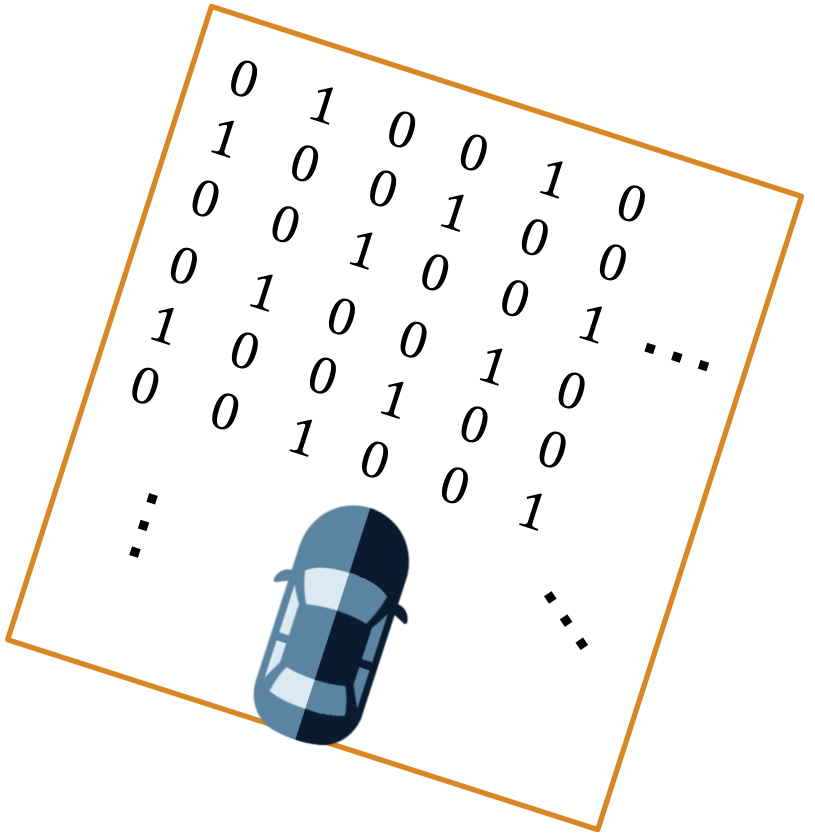
環境認知
システム

$$\begin{matrix} 0 & 1 & 0 & \\ 1 & 0 & 0 & \cdots \\ 0 & 0 & 1 & \\ & \vdots & & \ddots \end{matrix}$$

# 環境の認識方法

- 自車位置から見た周囲の領域を環境として捉える
- 人間の移動だけでなく、自車の位置、速度、方位角、方位角速度の変化によっても環境は変化するものとする



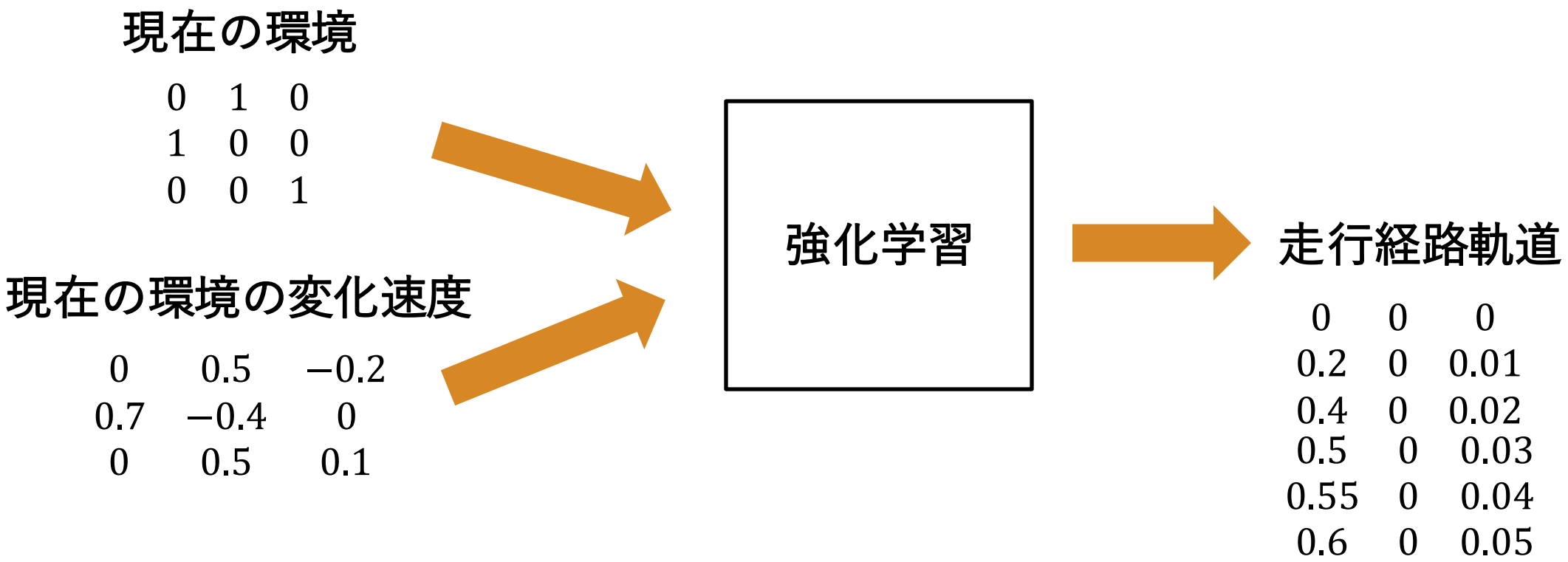


向きが変われば環境の領域も変化する


判断

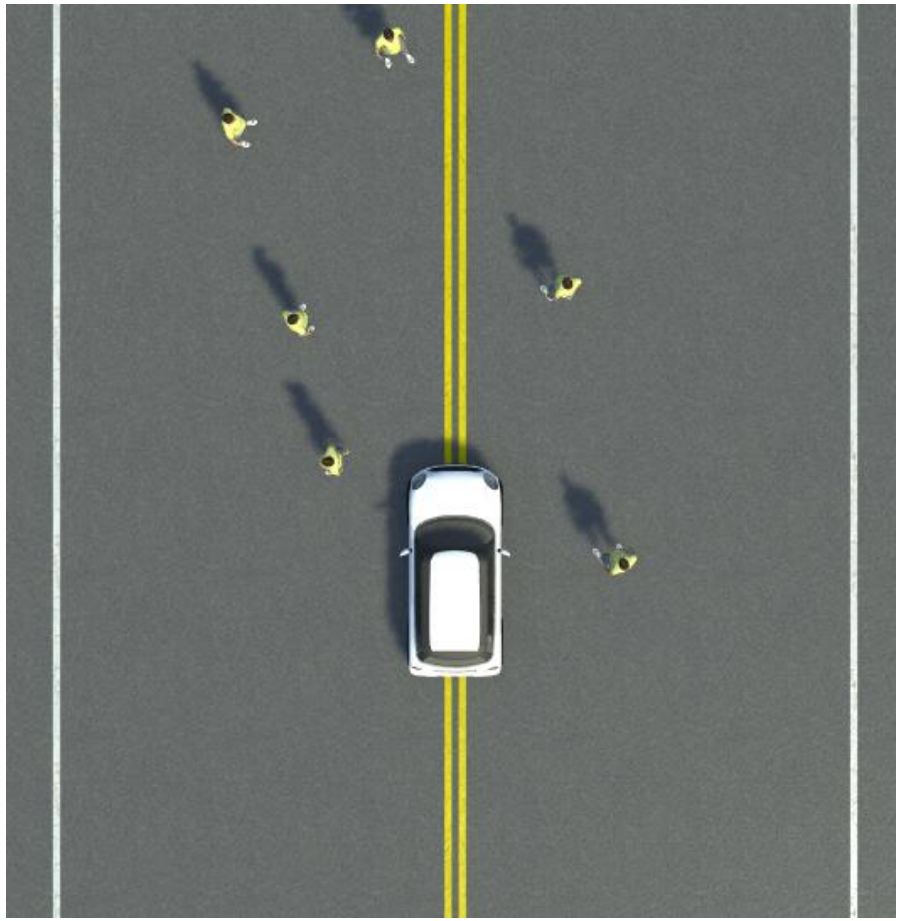
# 経路の生成方法

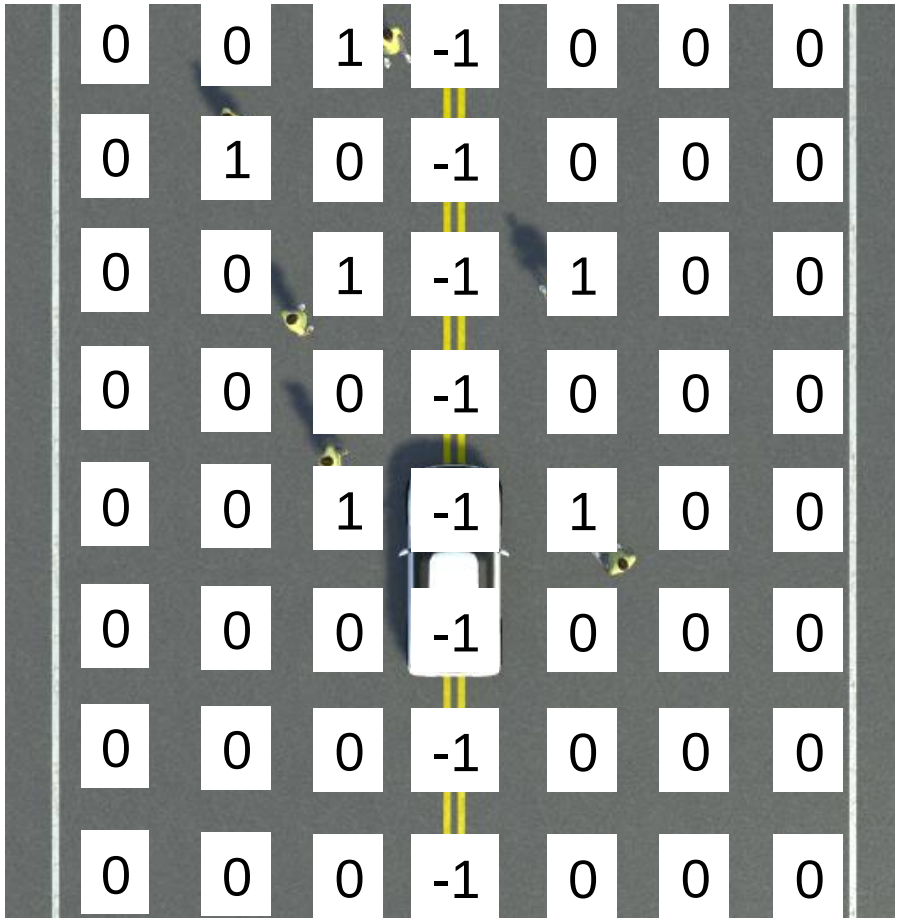
- 環境の状態（前スライドの行列表現）と、環境の変化速度（行列の数値変化の度合い）を計算して、その二つを用いて走行経路軌道を算出する



# 現在の環境

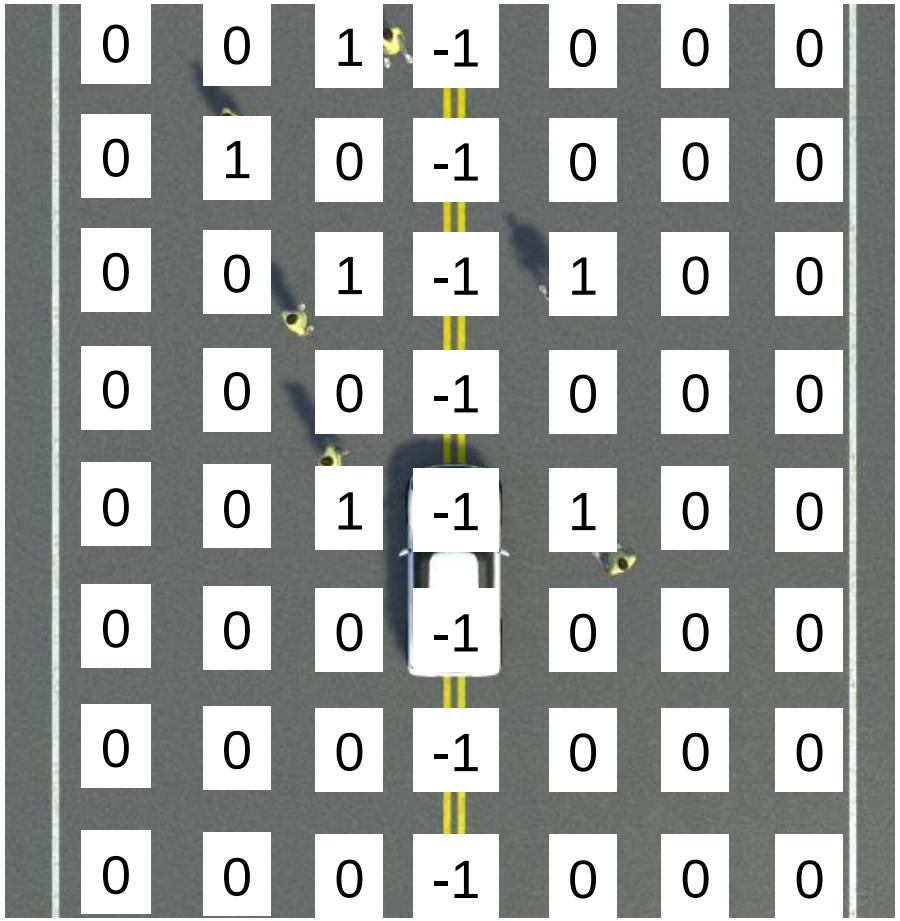
- 可視化範囲は車両周囲15mの正方形領域とする
- 歩行者が存在する箇所は1、元々の走行経路は-1、それ以外は0とする

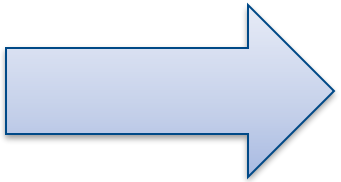


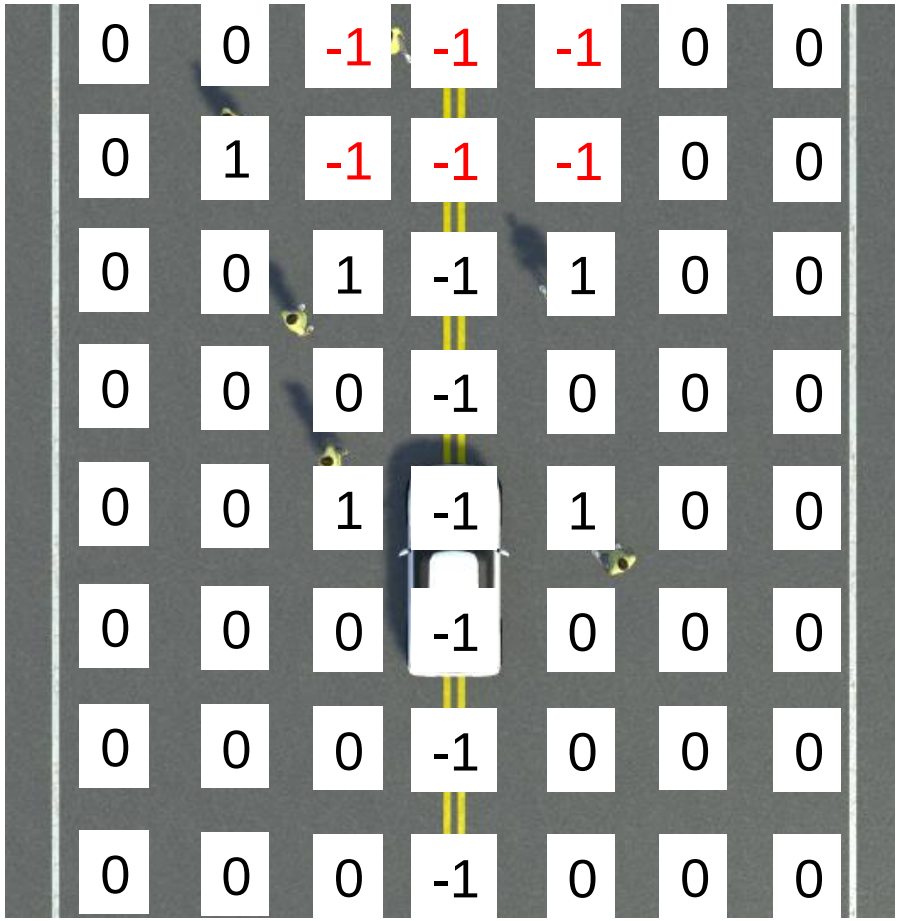


# 現在の環境

- さらに、可視化範囲の中の最もゴール位置に近い点の周囲を-1で埋める
  （行列からゴール方向を認識できるようにするため）





# 現在の環境の変化速度

- 歩行者が存在する箇所を1として環境を作成する（元々の経路情報はここでは考慮しない）
- 前回ステップの環境からの差分をローパスフィルタに入力し、その出力を「環境の変化速度」とする

1. 差分を計算する

今回の環境 − 前回の環境 = 環境の差分

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

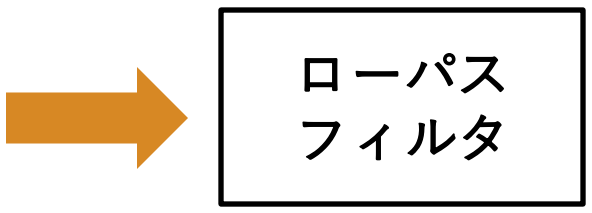
# 現在の環境の変化速度

- 歩行者が存在する箇所を1として環境を作成する（経路は含まない）
- 前回ステップの環境からの差分をローパスフィルタに入力し、その出力を「環境の変化速度」とする

2. 行列の要素ごとにローパスフィルタを計算する

環境の差分

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$




環境の差分をフィルタしたもの

$$\begin{bmatrix} 0 & 0 & -0.18 & 0 & 0 & -0.01 & 0 \\ 0 & -0.18 & 0 & 0.2 & 0 & 0 & 0 \\ 0.2 & 0 & -0.1 & 0 & -0.2 & 1 & 0 \\ 0 & 1 & 0 & -0.1 & 0 & 0 & 0 \\ 0 & 0 & -0.15 & 0 & -0.15 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0.2 & 0 & -0.01 \\ 0.1 & 0 & 0 & 0 & -0.02 & 0 & 0 \end{bmatrix}$$

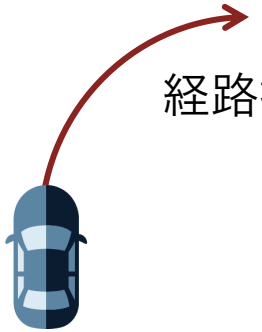
# 走行経路軌道

- 強化学習方策が出力する走行経路は、以下の4パターンとする

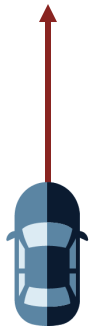
パターン1：停止

その場に留まる
経路指令値

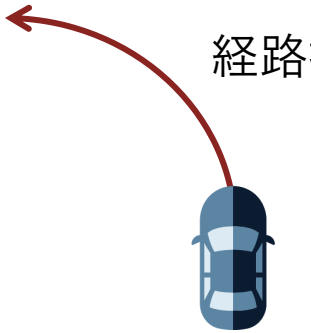
パターン3：右旋回




パターン2：直進

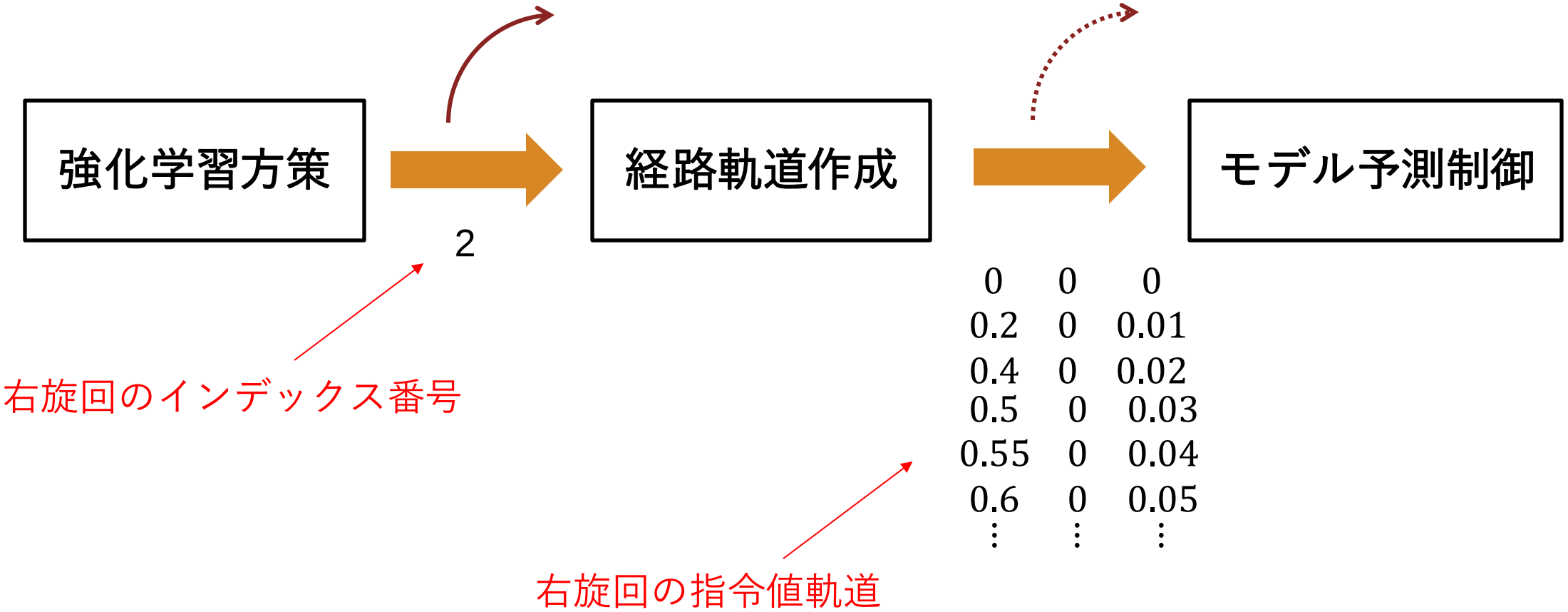



パターン4：左旋回



# 走行経路軌道の具体化

- 強化学習方策は走行経路のインデックス番号を出力する
- そのインデックス番号に応じて具体的な経路軌道を生成してからモデル予測制御に渡す



# 強化学習エージェント

- エージェントはPPO (Proximal Policy Optimization) を用いる
  - Actor-Critic型
  - モデルフリー
  - オンポリシー
  - 比較的簡単な実装でサンプル効率が良い

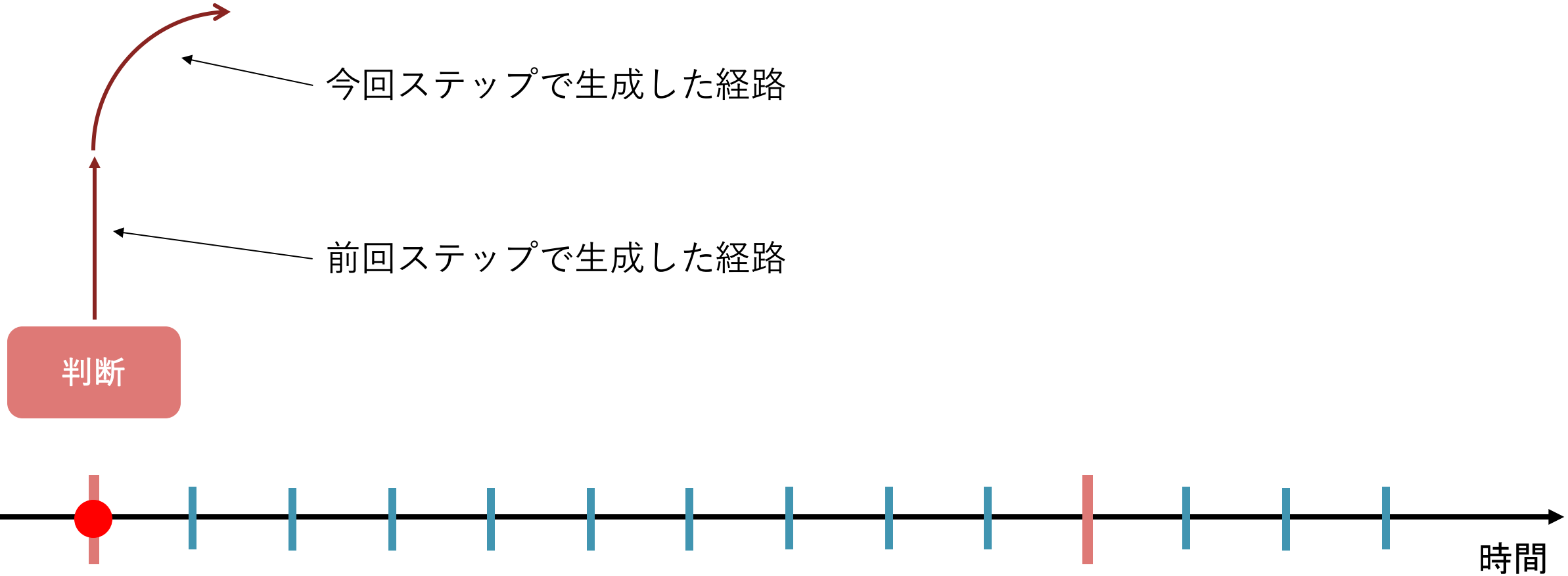
# 制御

# 制御

- 指令値軌道に遅れなく追従できるモデル予測制御を用いる
- 強化学習方策は0.2[s]、モデル予測制御は0.02[s]のタイムステップで実行する
- 方策が実行されていない間は、モデル予測制御は前回までに出力された走行経路に従って制御を実行する

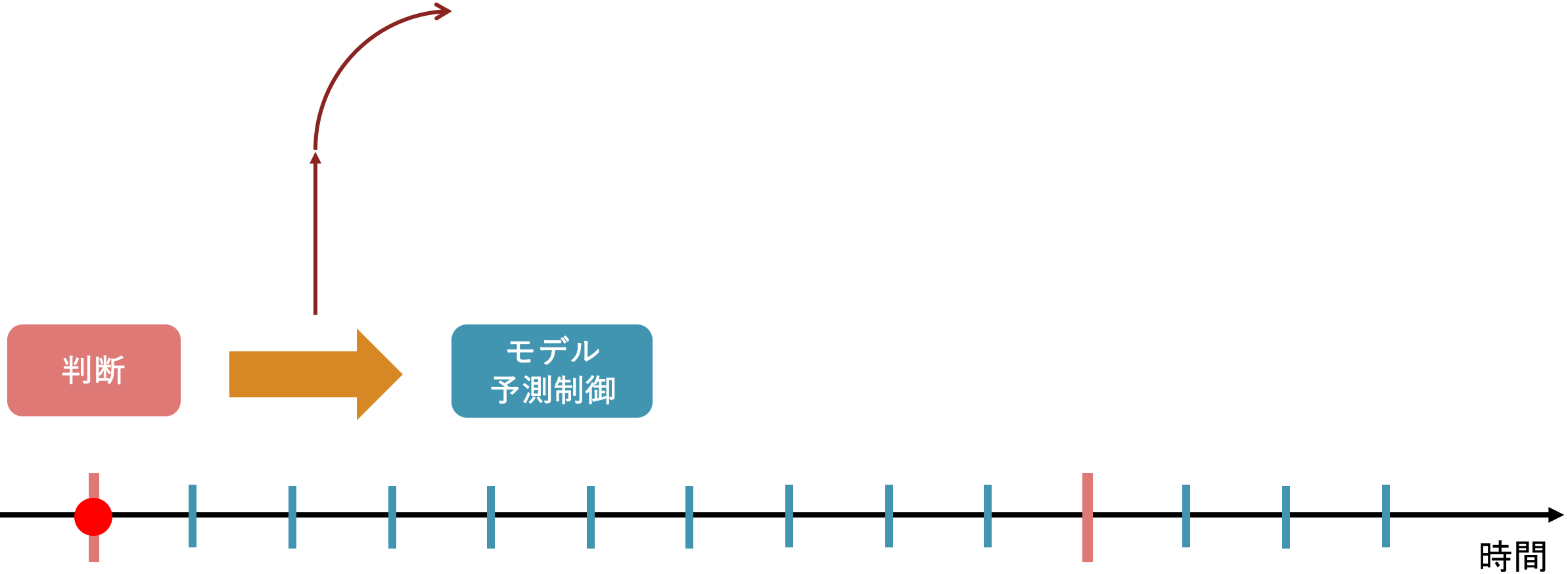
# 走行経路の渡し方

- 判断システム（強化学習方策）は、各タイムステップごとに1ステップ先の走行経路を出力する



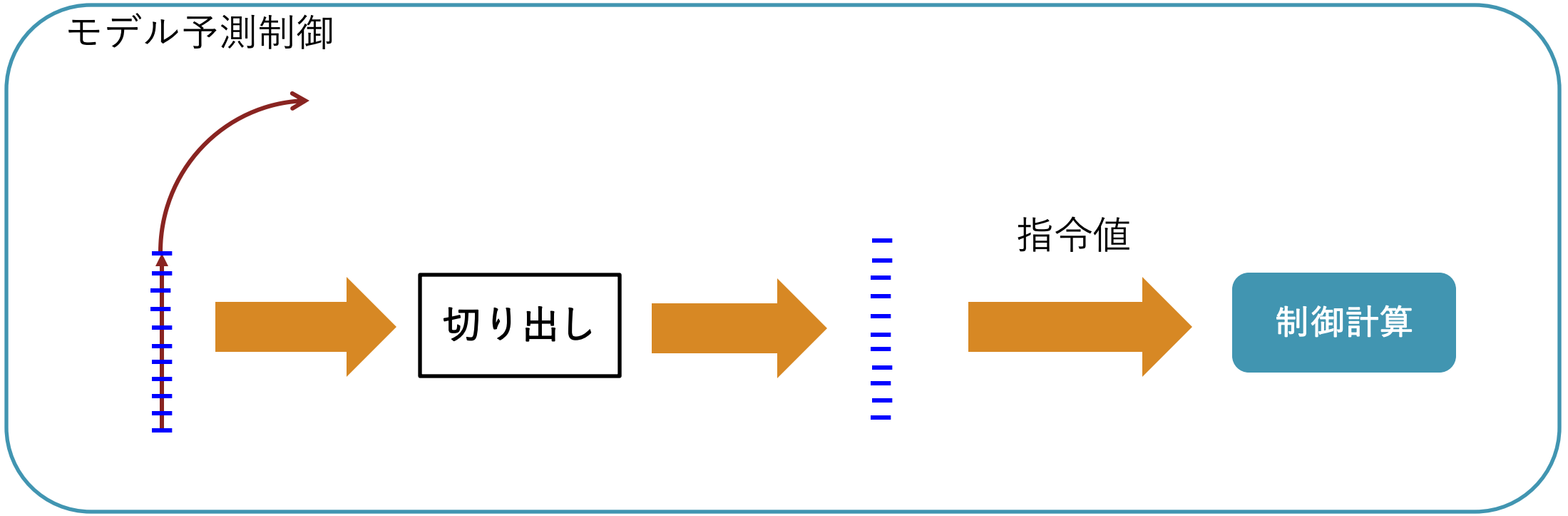
# 走行経路の渡し方

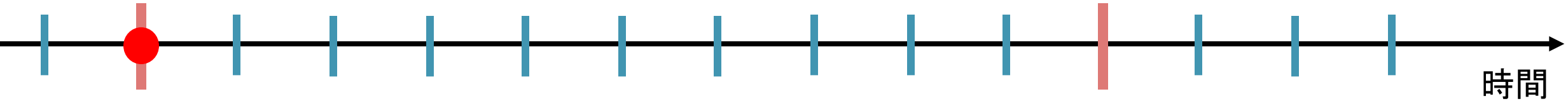
- 生成した経路は、前回生成したものと合わせてモデル予測制御に渡す



# 走行経路の渡し方

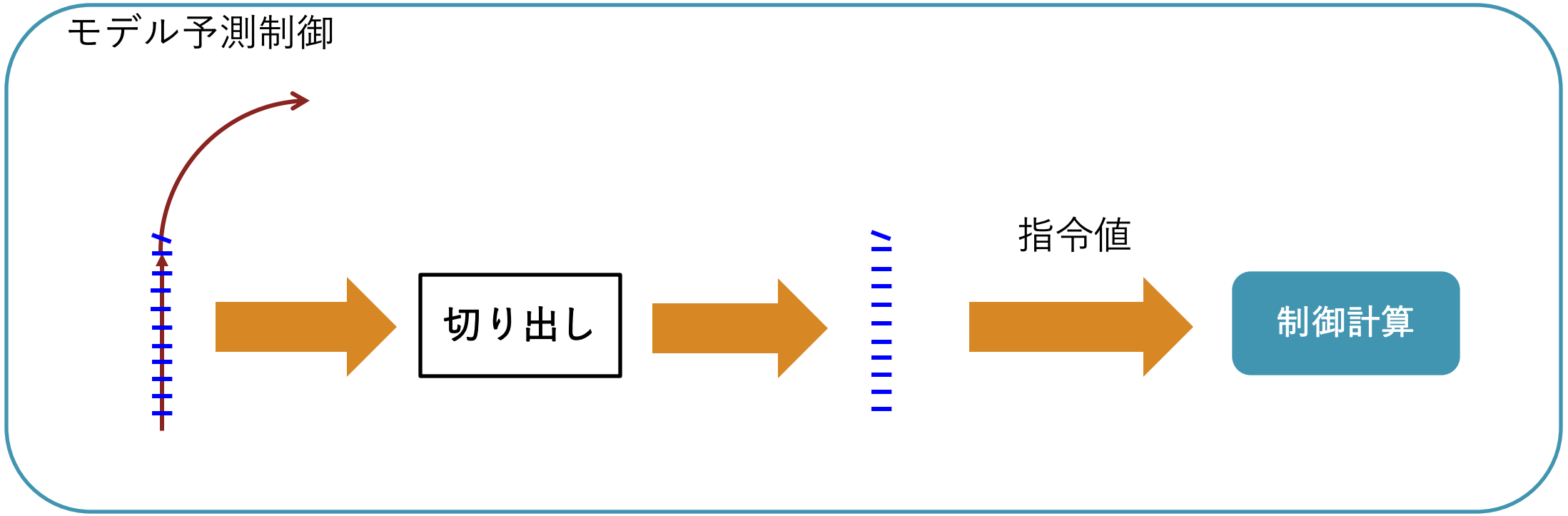
- モデル予測制御は、与えられた経路を予測ホライズン分だけ切り出し、制御の指令値として用いる

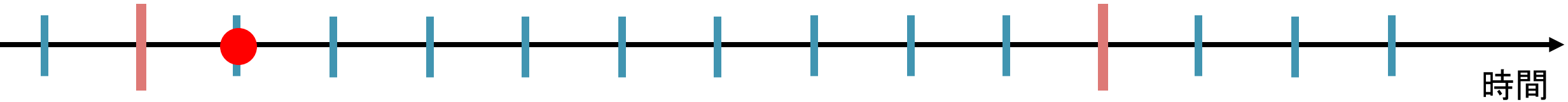


# 走行経路の渡し方

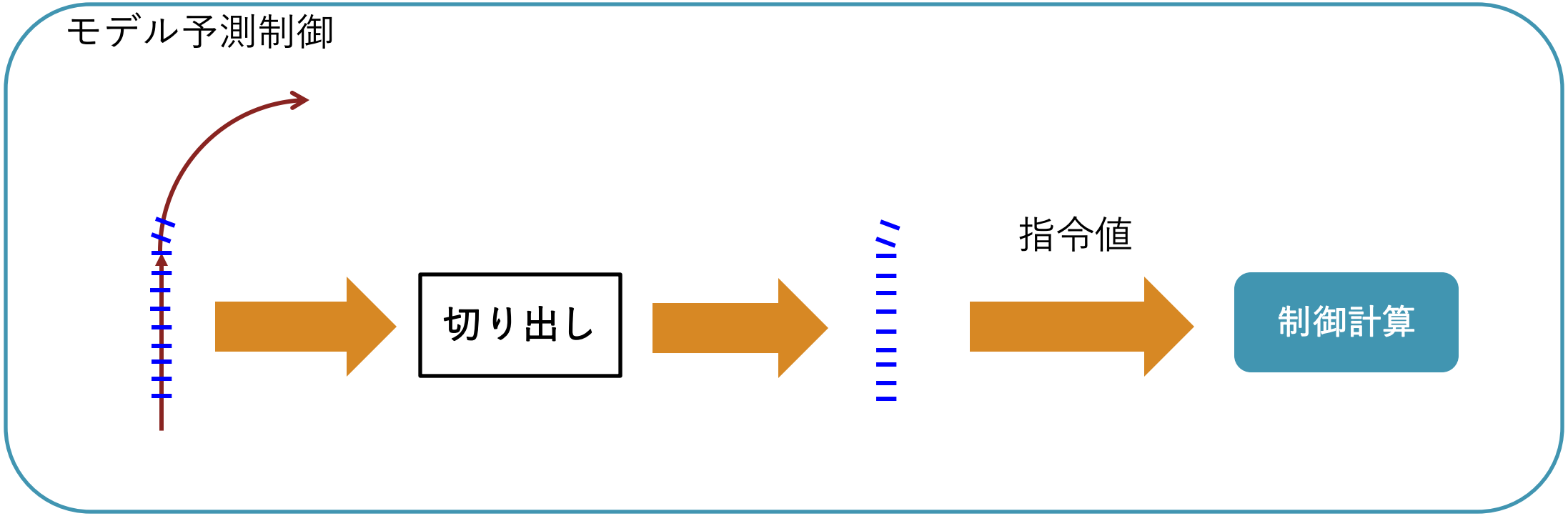
- モデル予測制御は、与えられた経路を予測ホライズン分だけ切り出し、制御の指令値として用いる

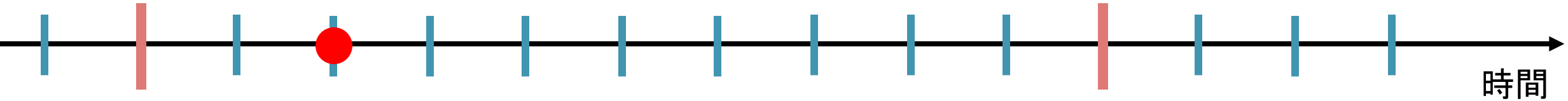


# 走行経路の渡し方

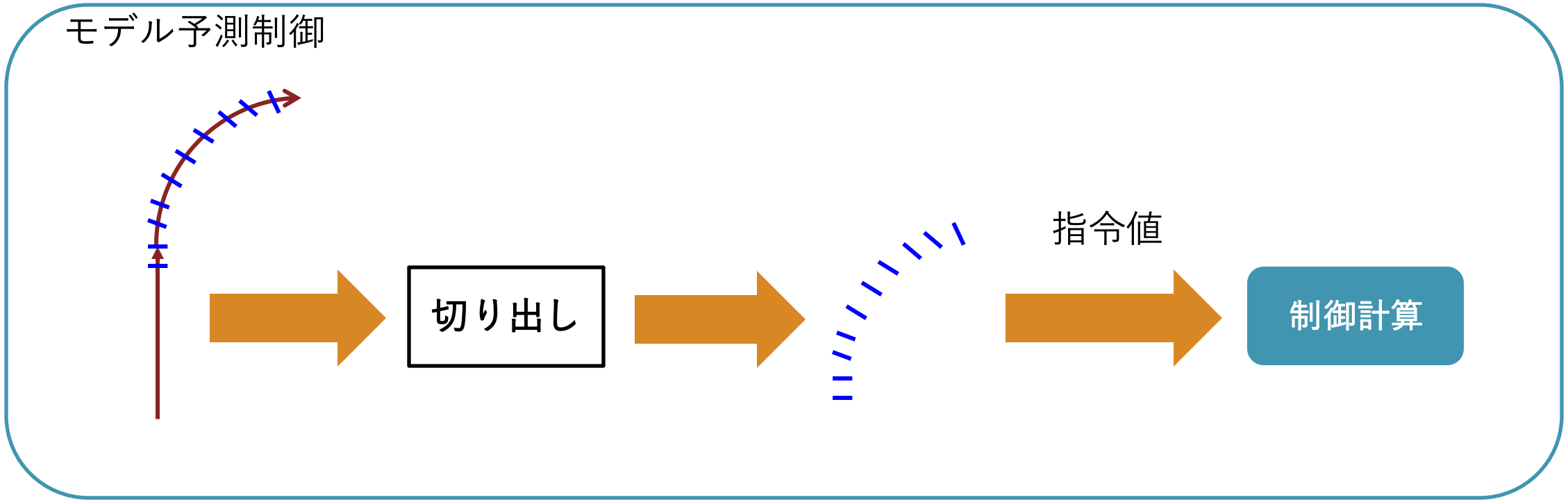
- モデル予測制御は、与えられた経路を予測ホライズン分だけ切り出し、制御の指令値として用いる

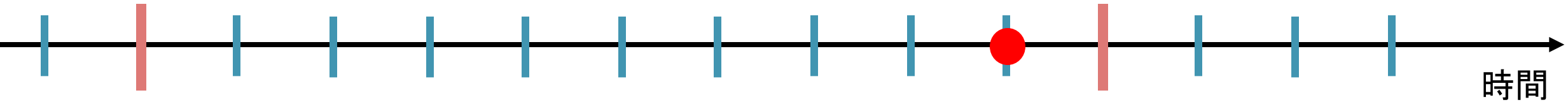


# 走行経路の渡し方

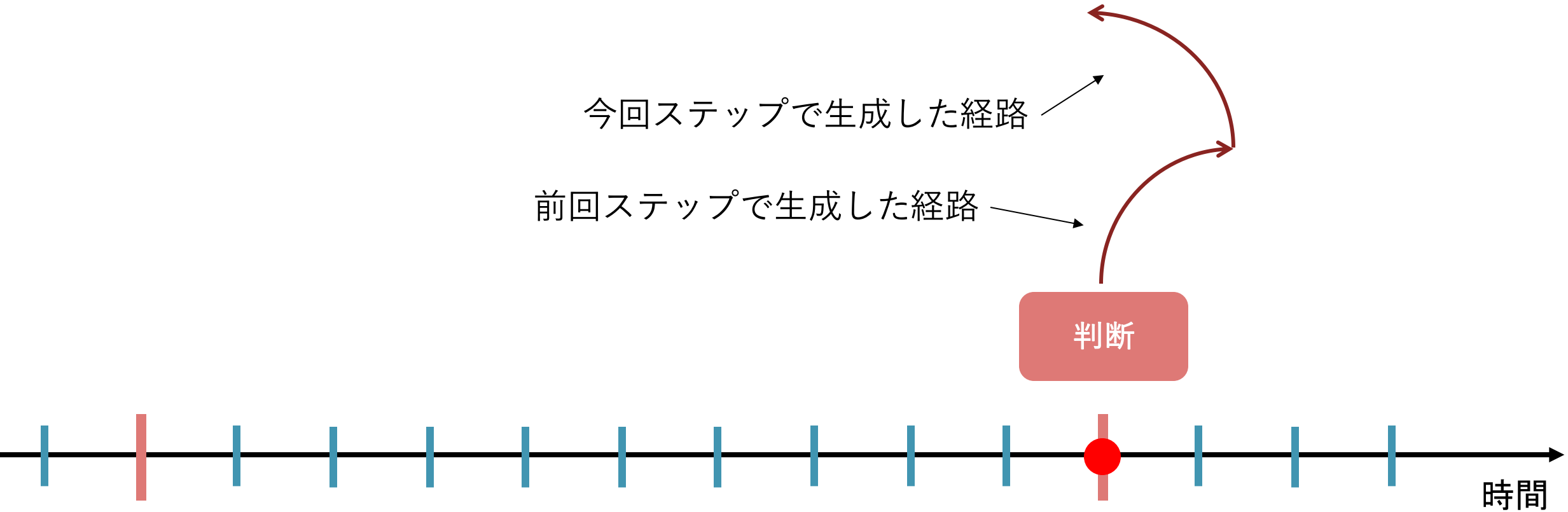
- モデル予測制御は、与えられた経路を予測ホライズン分だけ切り出し、制御の指令値として用いる



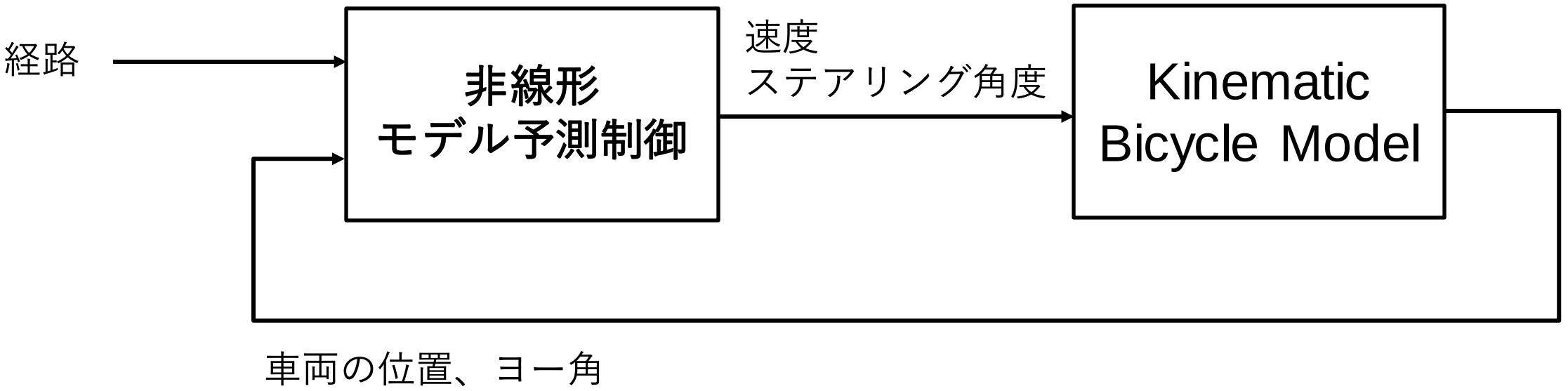
# 走行経路の渡し方

- 1ステップ先の走行経路を出力する
  （以降は繰り返し）

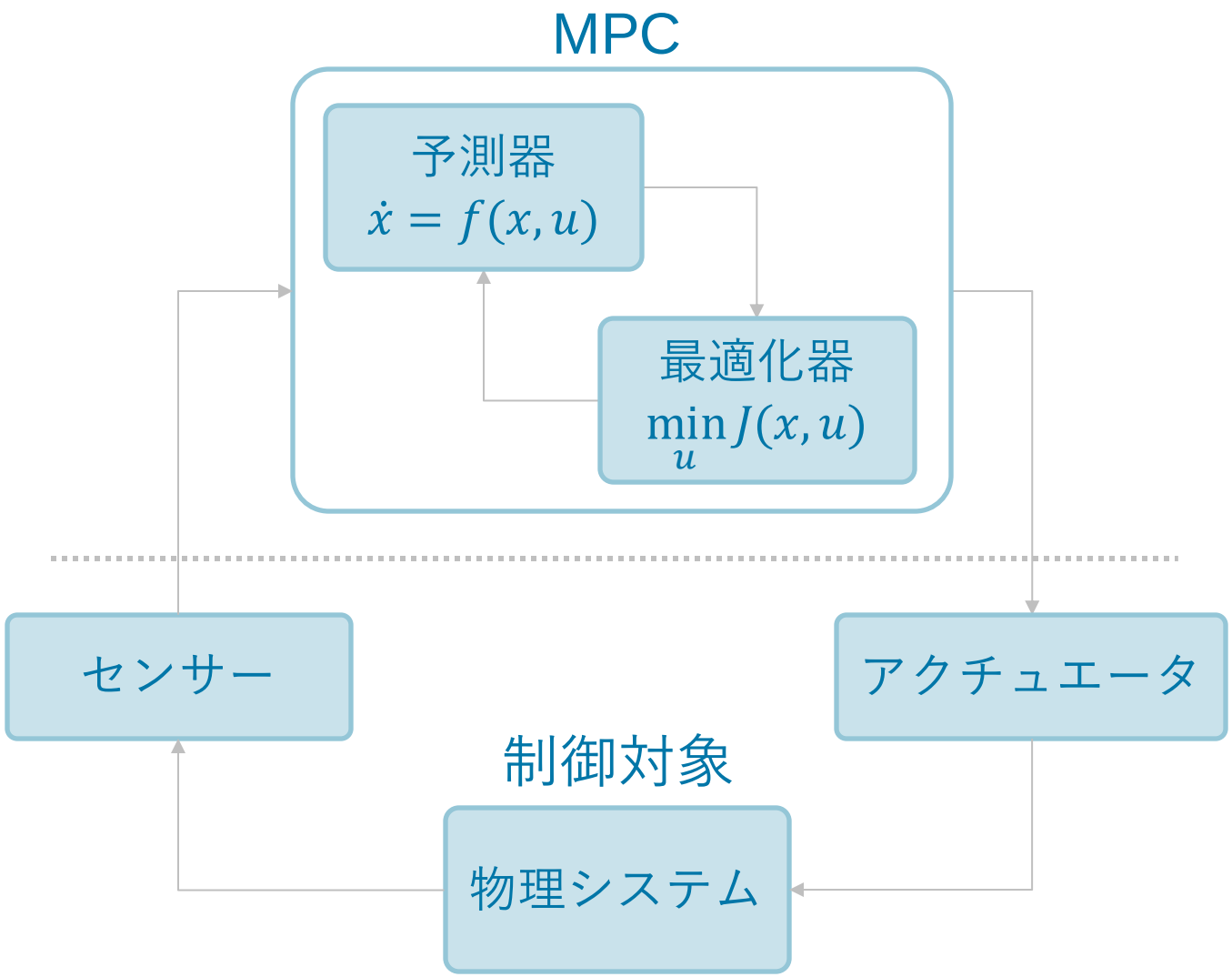


# 非線形モデル予測制御による車両走行制御

- 制御器として非線形モデル予測制御（Nonlinear MPC）を用いる
- 車両モデルはKinematic Bicycle Modelを用いる



# MPC は 「予測」 と 「最適化」 による高性能なフィードバック制御手法




## 考え方： 予測と最適化を高速に繰り返す

各制御サンプリング時間で…

1. 制御対象の未来の振る舞いを予測
2. 制御要求に対して、数値最適化を行い最もよい動かし方を決定

## 利点： 複雑な対象を賢く上手に制御する

| | |
|---|---|
| 高い制御性能 | 最適化 & フィードバック制御 |
| 制約遵守 | 物理/性能/安全の制限を守る（制約条件を陽に考慮） |
| 協調上手 | 大規模/複雑な対象を扱える（多入出力系への拡張性） |

# 非線形モデル予測制御 = 制約付き非線形計画法

**非線形予測モデル**

$$\begin{cases} \dot{x} = f(x,u) \\ y = g(x,u) \end{cases}$$

**任意の非線形コスト関数**

$$J = cost(x,u)$$

**任意の非線形制約**

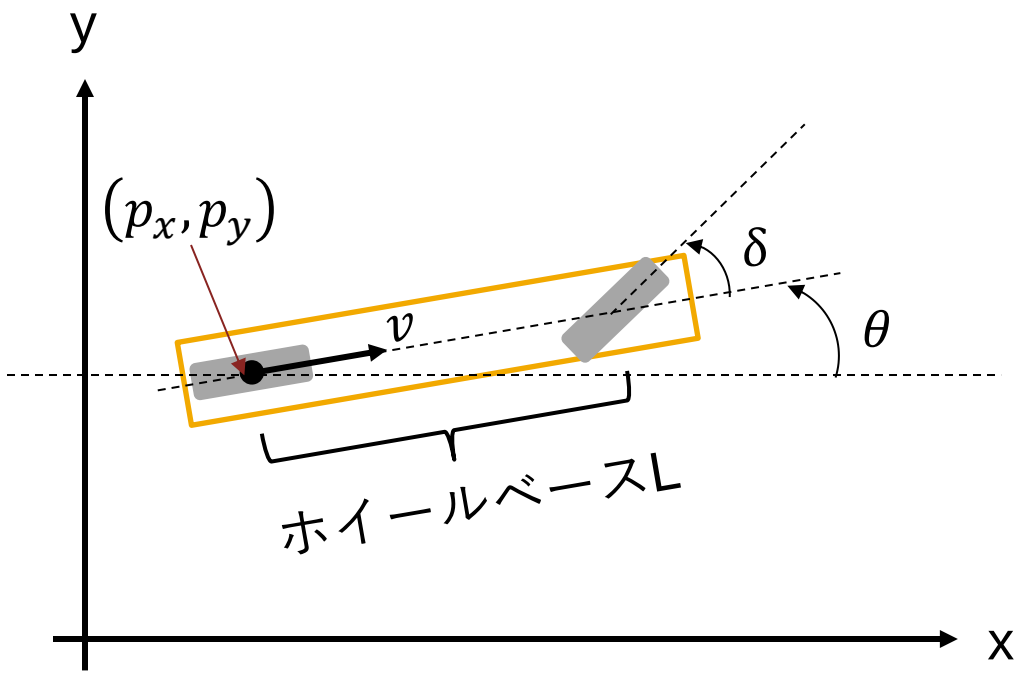
$$c = ineq(x,u),\ ceq = eq(x,u)$$

非線形計画法 (NLP)

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } g_i(x) \le 0 \text{ for each } i \in \{1,\dots,m\} \\ &\qquad\qquad h_j(x) = 0 \text{ for each } j \in \{1,\dots,p\} \\ &\qquad\qquad x \in X. \end{aligned}$$

# Kinematic Bicycle Model

- 車両のダイナミクスをx位置、y位置、ヨー角θの3状態と速度v、ステアリング角δの2入力で表現するモデル




$$\dot{p_x} = v \cdot \cos(\theta)$$

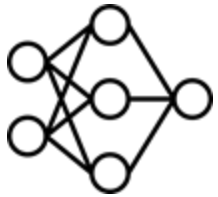
$$\dot{p_y} = v \cdot \sin(\theta)$$

$$\dot{\theta} = v \cdot \frac{\tan(\delta)}{L}$$

# 強化学習のワークフロー

Environment

Reward



Policy representation

Agent


Training

Deployment

# 環境構築

**Environment**



Reward



Policy representation



Agent




Training



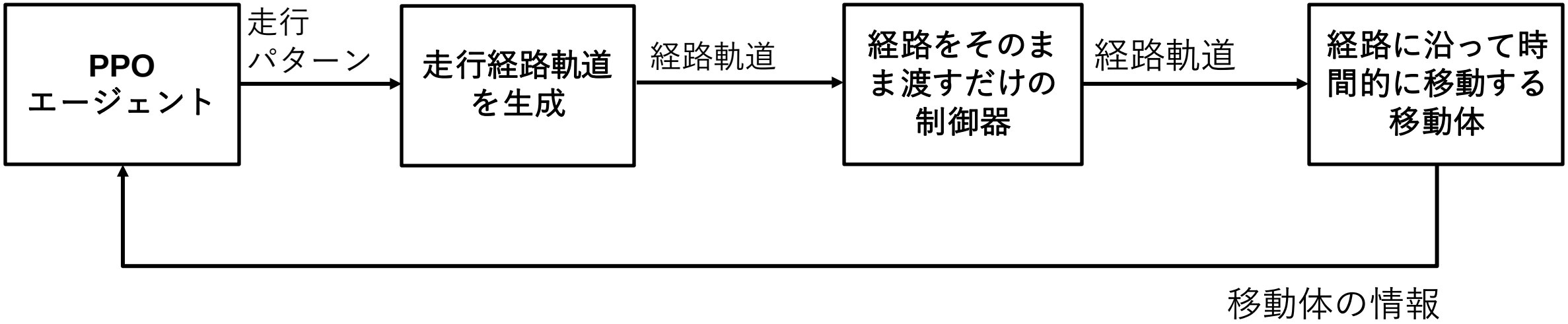
Deployment

# 環境構築

強化学習の学習は数千回の繰り返し実行が必要であるため、1回のシミュレーション実行の時間は可能な限り短くしたい。

そこで、学習を行う際には走行制御とプラントモデルは簡易化し、与えられた走行経路に沿って理想的に動作するモデルとする。

学習後の方策をMPCと結合し、想定通り動作するかどうかを検証する。



# 報酬設計


Environment → **Reward** → Policy representation + Agent → Training → Deployment

# 報酬設計

車両はゴールへ向かって走行しなければならない。加えて、歩行者に衝突してはいけない。
従って、以下のように報酬を与えるものとする。

- ゴールに近づくことができた距離（単位:m）の分だけ報酬を与える
- ゴールに近い実験領域境界を横切ることができれば追加の報酬20を与える
- 歩行者に衝突したらエピソードを終了させる
- ゴールから遠い実験領域境界を横切った場合、エピソードを終了させる

「ゴールに近づくことができた距離（単位:m）の分」は、累積報酬和を意味しているため、各タイムステップでは、前回から今回までに近づいた差分の距離を与える。

Simulinkモデルは時間ベースのシミュレーションを行っているが、報酬は時間に依らないため、Stateflowの状態遷移を用いてイベントベースに報酬を与えるように設計した。

# 方策、エージェント構築



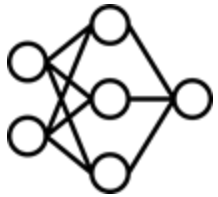
Environment





Reward





**Policy representation**

**Agent**






Training





Deployment

# CriticとActor

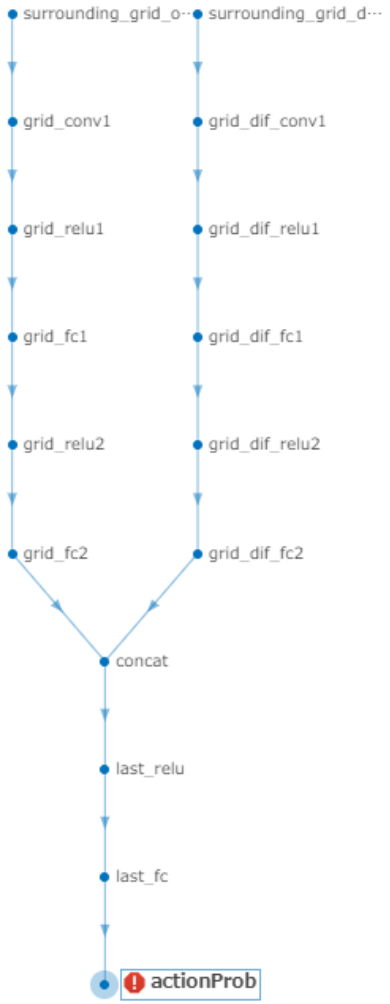
入力は「周囲グリッド」と「周囲グリッドの変化速度」、出力は「走行パターン」である。
「周囲グリッド」と「周囲グリッドの変化速度」に対しては、最初にconvolution2dLayerを用いてスライディング畳み込みフィルターを適用する。これにより、周囲の領域ごとに特徴量を得ることができる。

```
conv_FilterSize = 5;
conv_NumFilter  = 2;
conv_Stride     = 2;
conv_Padding    = 0;

grid_path = [
    imageInputLayer(grid_obs_Bus.Elements(1).Dimensions, ...
        'Normalization', 'none', ...
        'Name', grid_obs_Bus.Elements(1).Name)
    convolution2dLayer(conv_FilterSize, conv_NumFilter, 'Name', 'grid_conv1', ...
        'Stride', conv_Stride, 'Padding', conv_Padding)
```

# Actorの構造

以下のエラーは出力レイヤーに繋いでいないことによるものであるが、強化学習では繋ぐ必要はないため、問題ではない。

## ネットワークアナライザーによる解析結果




問題

解析結果

| | 名前 | タイプ | アクティベーション | Learnables プロパティ | 状態 |
|---|---|---|---|---|---|
| 1 | surrounding_grid_obs<br>75×75×1 イメージ | イメージの入力 | 75(S) × 75(S) × 1(C) × 1(B) | - | - |
| 2 | grid_conv1<br>ストライド [2 2] およびパディング [0 0 … | 2-D 畳み込み | 36(S) × 36(S) × 2(C) × 1(B) | Weights 5 × 5 × 1 × 2<br>Bias 1 × 1 × 2 | - |
| 3 | grid_relu1<br>ReLU | ReLU | 36(S) × 36(S) × 2(C) × 1(B) | - | - |
| 4 | grid_fc1<br>400 全結合層 | 全結合 | 1(S) × 1(S) × 400(C) × 1(B) | Weights 400 × 2592<br>Bias 400 × 1 | - |
| 5 | grid_relu2<br>ReLU | ReLU | 1(S) × 1(S) × 400(C) × 1(B) | - | - |
| 6 | grid_fc2<br>300 全結合層 | 全結合 | 1(S) × 1(S) × 300(C) × 1(B) | Weights 300 × 400<br>Bias 300 × 1 | - |
| 7 | surrounding_grid_dif_obs<br>75×75×1 イメージ | イメージの入力 | 75(S) × 75(S) × 1(C) × 1(B) | - | - |
| 8 | grid_dif_conv1<br>ストライド [2 2] およびパディング [0 0 … | 2-D 畳み込み | 36(S) × 36(S) × 2(C) × 1(B) | Weights 5 × 5 × 1 × 2<br>Bias 1 × 1 × 2 | - |
| 9 | grid_dif_relu1<br>ReLU | ReLU | 36(S) × 36(S) × 2(C) × 1(B) | - | - |
| 10 | grid_dif_fc1<br>400 全結合層 | 全結合 | 1(S) × 1(S) × 400(C) × 1(B) | Weights 400 × 2592<br>Bias 400 × 1 | - |
| 11 | grid_dif_relu2<br>ReLU | ReLU | 1(S) × 1(S) × 400(C) × 1(B) | - | - |
| 12 | grid_dif_fc2<br>300 全結合層 | 全結合 | 1(S) × 1(S) × 300(C) × 1(B) | Weights 300 × 400<br>Bias 300 × 1 | - |
| 13 | concat<br>次元 3 に沿った 2 個の入力の連結 | 連結 | 1(S) × 1(S) × 600(C) × 1(B) | - | - |
| 14 | last_relu<br>ReLU | ReLU | 1(S) × 1(S) × 600(C) × 1(B) | - | - |
| 15 | last_fc<br>4 全結合層 | 全結合 | 1(S) × 1(S) × 4(C) × 1(B) | Weights 4 × 600<br>Bias 4 × 1 | - |
| 16 | actionProb<br>ソフトマックス | ソフトマックス | 1(S) × 1(S) × 4(C) × 1(B) | - | - |

# 学習



Environment





Reward





Policy representation



Agent




**Training**

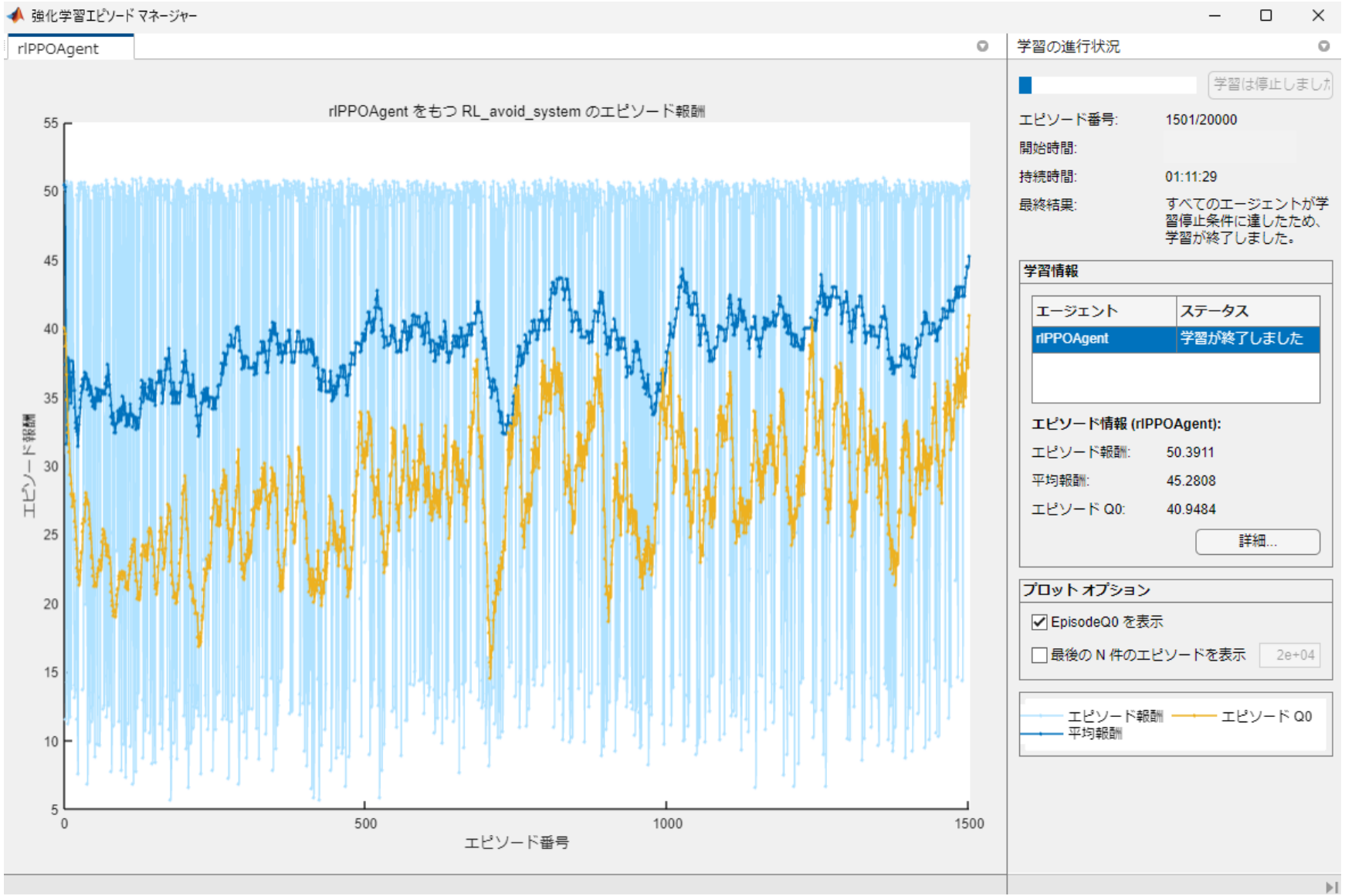




Deployment

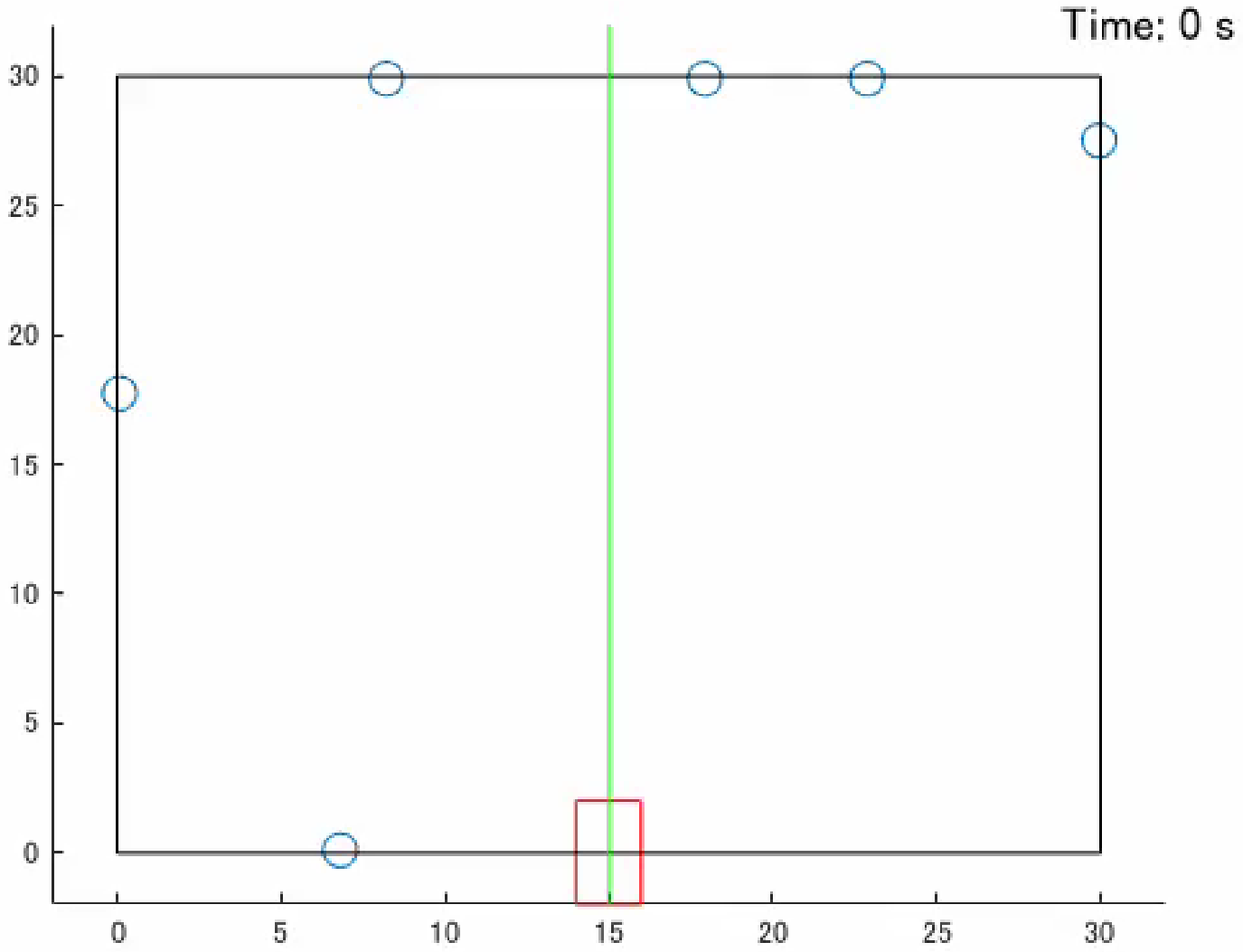
# 学習を実行

```
trainingStats = train(avoid_agent, env, trainOpts);
```



# 学習後のシミュレーション結果

歩行者を回避しながら元々の経路に沿って走行できている。



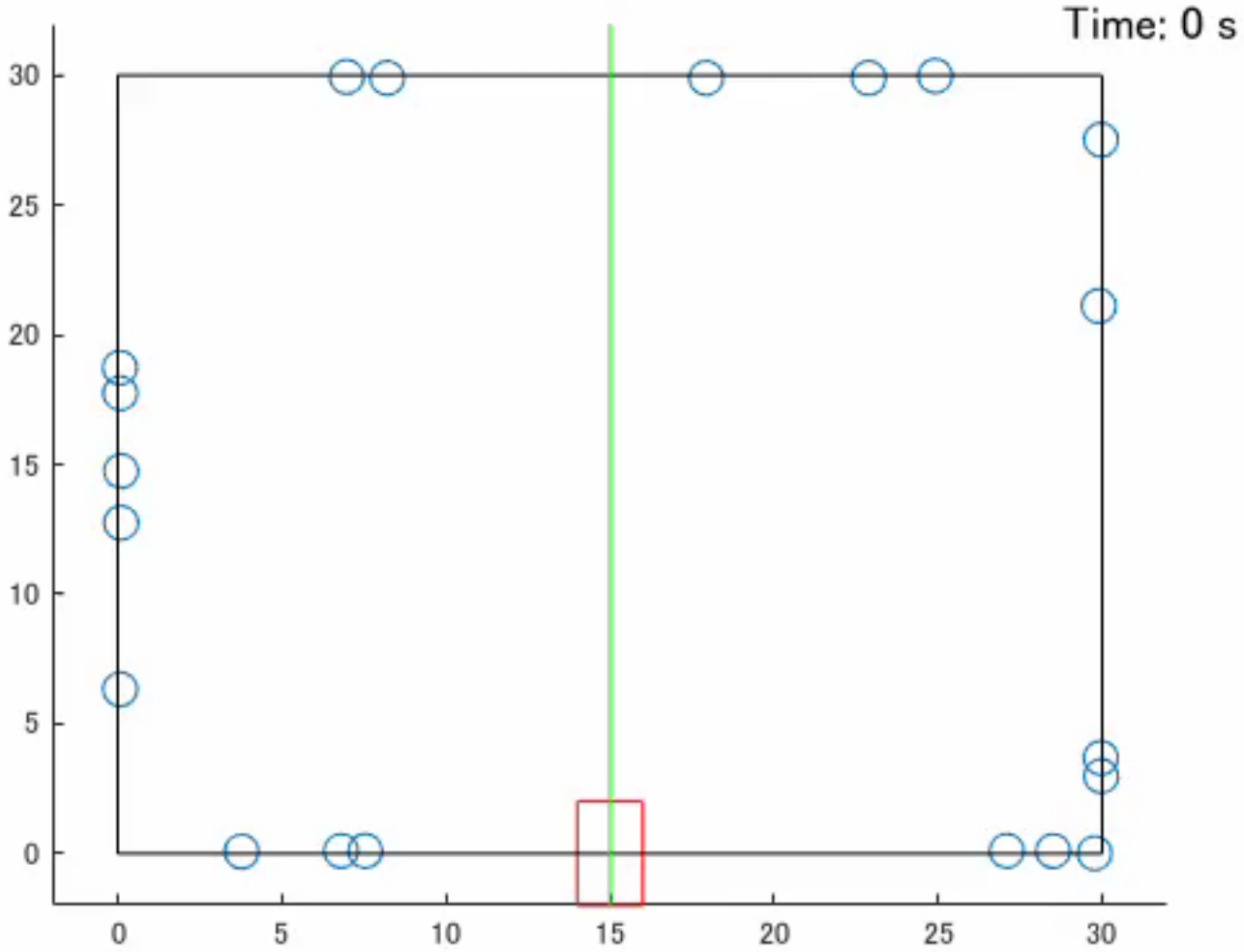

○：歩行者

□：車両

｜：元々の経路

# 複雑な状況に対してどこまで対処できるか

環境の条件を変えることで、様々な状況を再現できる。今回のデモでは、20人の歩行者に対して回避走行を学習できることを確認した。




特にこのケースでは、「歩行者が前方にいなければ前進、そうでなければ停止し、歩行者が離れるのを待つ」という最適解に到達している。
（開始時の位置と姿勢がゴール方向に向いているため、旋回をする必要がない）

# 実装



Environment





Reward





Policy representation



Agent





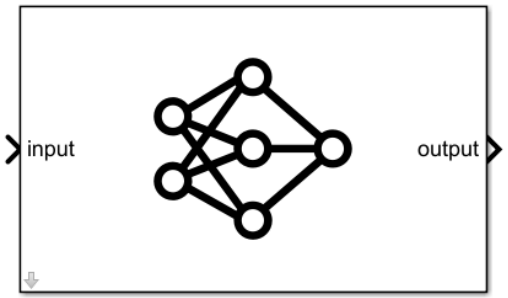

Training



**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

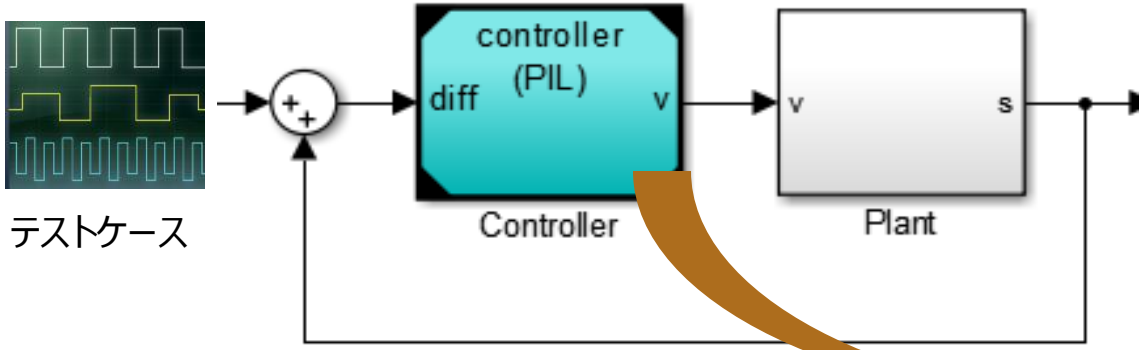
一般的なCPU

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

# 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**

# Model Predictive Control Toolboxを用いたMPC設計の流れ

1. 予測モデリング → 2. MPC 設計 → 3. シミュレーション → 4. 実装・試験

システム同定

伝達関数・状態空間

非線形モデルを線形化

予測モデル

- 線形時不変系
- 線形時変/LPV系

$$\dot{x} = A(t)x + B(t)u$$
$$y = C(t)x + D(t)u$$

設計パラメータ
- 制約条件
- 評価関数重み
- ホライズンなど

設計とシミュレーション

専用 QP ソルバ

設計手段
- GUI
- コマンドライン

OPC 通信

C コード生成

ST コード生成

※ST = Structured Text
PLC用のコード

# 1. 予測モデリング

MPCの内部モデルを数式モデルから構築する。この時 Symbolic Math Toolbox を使って数式計算とモデルの生成を自動化できる。

モデル予測制御は離散時間で動作するため、内部モデルも離散時間モデルを用いることにする。 Kinematic Bicycle Model モデルを以下のように離散化する。

$$\dot{p_x} = v \cdot \cos(\theta)$$

$$\dot{p_y} = v \cdot \sin(\theta)$$

$$\dot{\theta} = v \cdot \frac{\tan(\delta)}{L}$$

離散化

$$p_x[k+1] = p_x[k] + v \cdot \cos(\theta[k]) \cdot dt$$

$$p_y[k+1] = p_y[k] + v \cdot \sin(\theta[k]) \cdot dt$$

$$\theta[k+1] = \theta[k] + v \cdot \frac{\tan(\delta)}{L} \cdot dt$$

# 1. 予測モデリング

ただし 今回、ヨー角の範囲を-180～180[deg]としているため、1回転する場合、不連続に値が変化する。状態空間モデルとして不連続性が存在することは避けたいため、ヨー角をクオータニオンに置き換えることにする。

$$p_x[k+1] = p_x[k] + v \cdot \{2(q_0[k])^2 - 1\} \cdot dt$$

$$p_y[k+1] = p_y[k] + v \cdot (2q_0[k]q_3[k]) \cdot dt$$

$$q_0[k+1] = q_0[k]dq_0 - q_3[k]dq_3$$

$$q_3[k+1] = q_0[k]dq_3 + q_3[k]dq_0$$

ただし、 $dq_0 = \cos\left(\frac{d\theta}{2} \cdot dt\right)$ $dq_3 = \sin\left(\frac{d\theta}{2} \cdot dt\right)$ $d\theta = v \cdot \frac{\tan(\delta)}{L}$

# 【補足】クオータニオン

クオータニオンを使って3次元の姿勢を表現することができる。

$$q_0 = \cos\left(\frac{\theta}{2}\right) \qquad q_1 = \lambda_x \cdot \sin\left(\frac{\theta}{2}\right) \qquad q_2 = \lambda_y \cdot \sin\left(\frac{\theta}{2}\right) \qquad q_3 = \lambda_z \cdot \sin\left(\frac{\theta}{2}\right)$$

この時、クオータニオン積を計算することで、元の姿勢状態$q_{pre}$から回転軸$(\lambda_x, \lambda_y, \lambda_z)$、回転角度$\theta$で回転させた結果のクオータニオン$q_{next}$を得ることができる。

$$\begin{bmatrix} q_{next\_0} \\ q_{next\_1} \\ q_{next\_2} \\ q_{next\_3} \end{bmatrix} = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{bmatrix} \begin{bmatrix} q_{pre\_0} \\ q_{pre\_1} \\ q_{pre\_2} \\ q_{pre\_3} \end{bmatrix}$$

# 【補足】クオータニオン

今回の Kinematic Bicycle Model では、ロール角、ピッチ角は変動しないものとする。従って、$q_1$ 、 $q_2$ は常に0とみなすことができる。
従って、 $q_{next}$の計算式は以下のようになる。

$$q_{next\_0} = q_{pre\_0}q_0 - q_{pre\_3}q_3 \qquad q_{next\_3} = q_{pre\_0}q_3 + q_{pre\_3}q_0$$

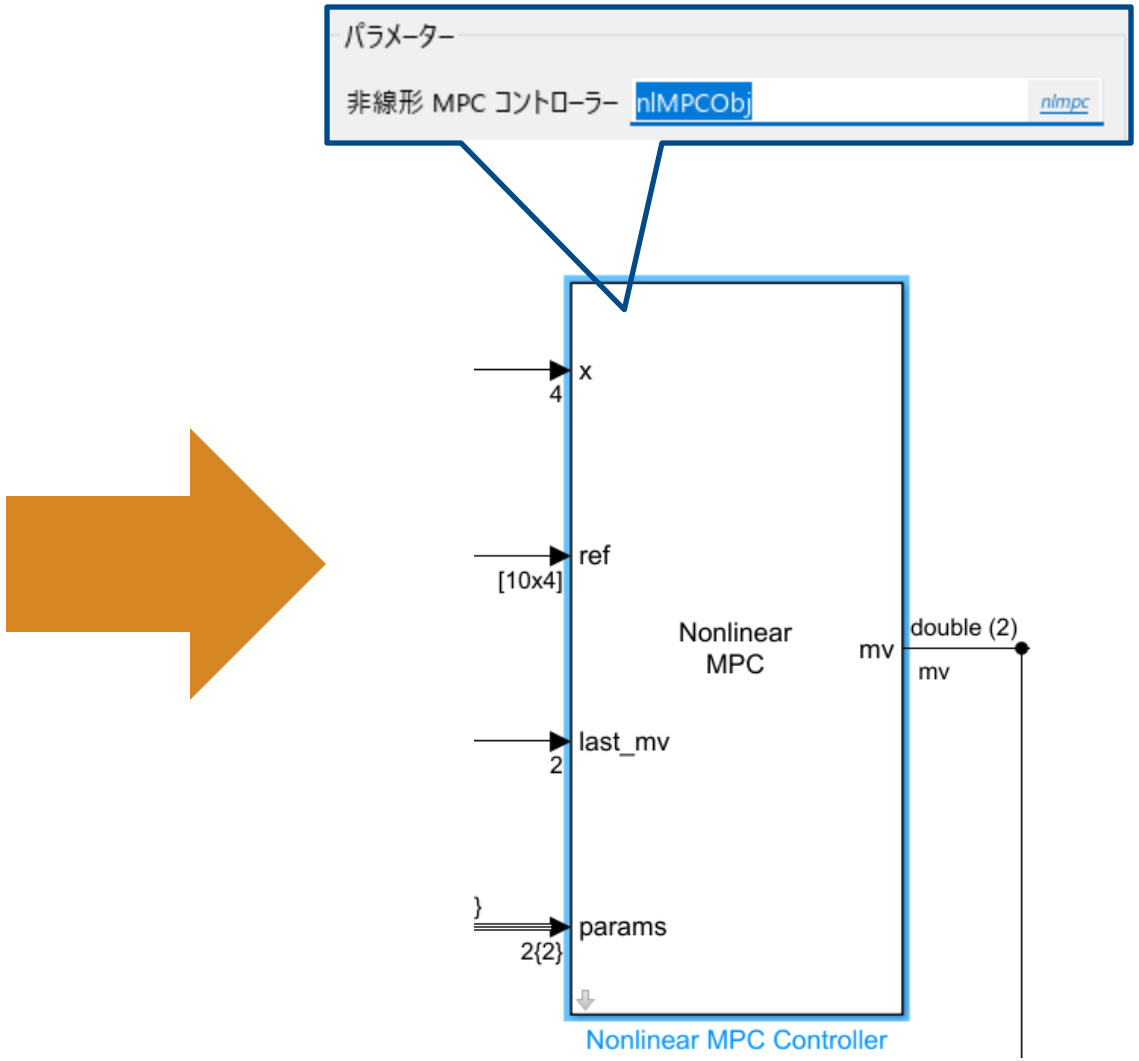
ただし、

$$q_0 = \cos\left(\frac{\theta}{2}\right) \qquad q_3 = \sin\left(\frac{\theta}{2}\right)$$

# 2. MPC 設計

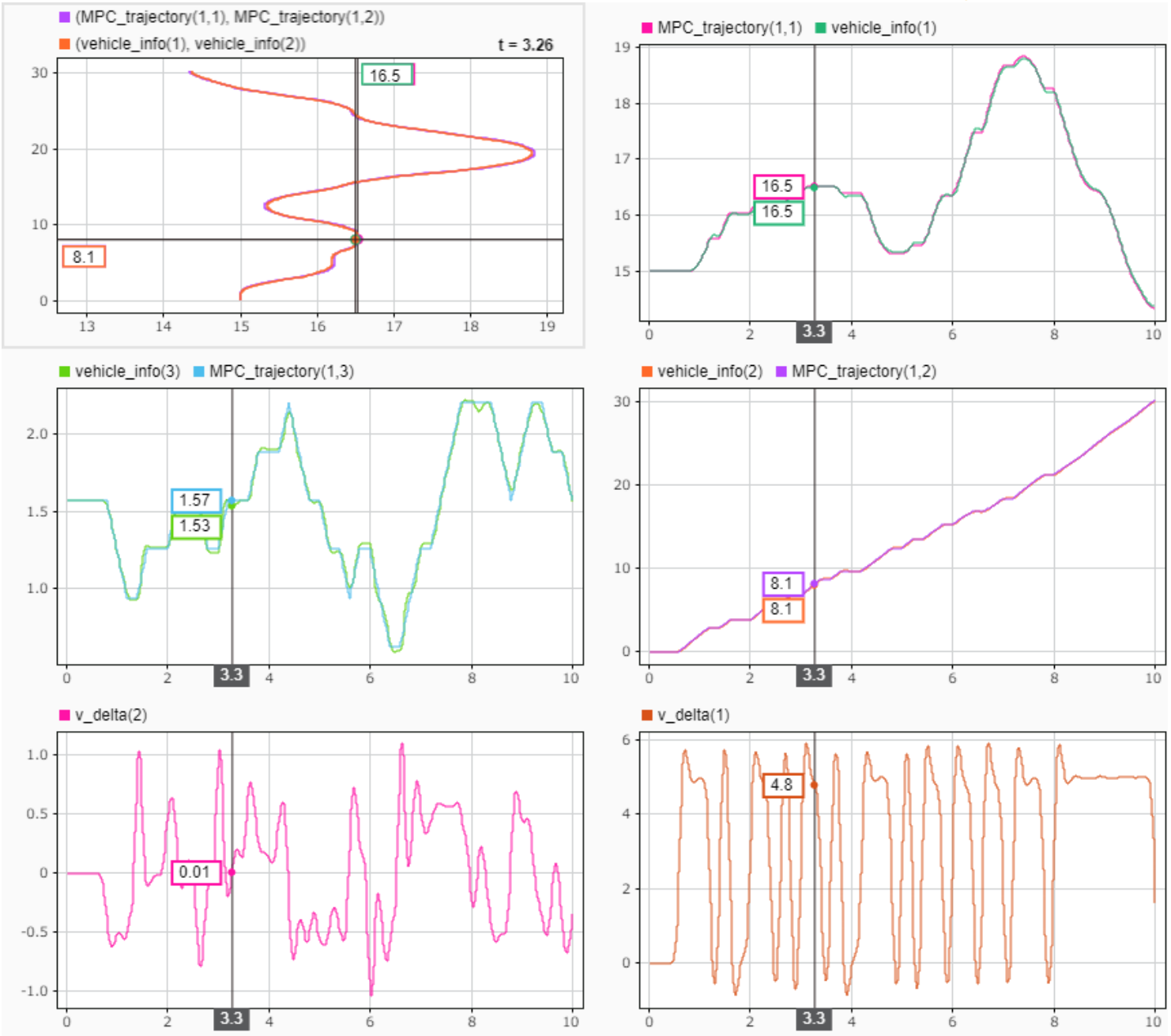
MPCのオブジェクトを設計する。

```
% 「nlmpc」コマンドで状態数、出力数、入力数を引数にしてオブジェクトを作成する。
nlMPCObj = nlmpc(4,4,2);

% サンプリング時間、予測、制御ホライズンを設定する。
nlMPCObj.Ts = MPC_TimeStep;
nlMPCObj.PredictionHorizon = prediction_horizon;
nlMPCObj.ControlHorizon = prediction_horizon;

% 入力制約を設定する。1番目は速度、2番目はステアリング角度である。
nlMPCObj.MV(1).Min = -10;
nlMPCObj.MV(1).Max = 10;
nlMPCObj.MV(2).Min = -1.5;
nlMPCObj.MV(2).Max = 1.5;

% 重みを設定する。
nlMPCObj.Weights.OutputVariables = [1,1,1,1];
nlMPCObj.Weights.ManipulatedVariablesRate = [0.1,0.2];

% パラメータ数を指定する。プラントモデリングのセクションで設計したように、
% サンプリング時間とホイールベースがパラメータとなる。
nlMPCObj.Model.NumberOfParameters = 2;

% 状態方程式、状態方程式のヤコビアン、出力方程式を指定する。
nlMPCObj.Model.StateFcn = "ReedsSheppVehicleStateFcn";
nlMPCObj.Jacobian.StateFcn = "ReedsSheppVehicleStateJacobianFcnRRT";
nlMPCObj.Model.OutputFcn = "ReedsSheppVehicleOutputFcn";

% 状態方程式を離散時間の式として扱う。
nlMPCObj.Model.IsContinuousTime = false;

% 終端条件を指定する。
nlMPCObj.Optimization.CustomEqConFcn = "parkingTerminalConFcn";
```



# 3. シミュレーション

強化学習方策から与えられる経路を元にモデル予測制御を実行し、歩行者を回避しながらの走行が行えることを確認する。

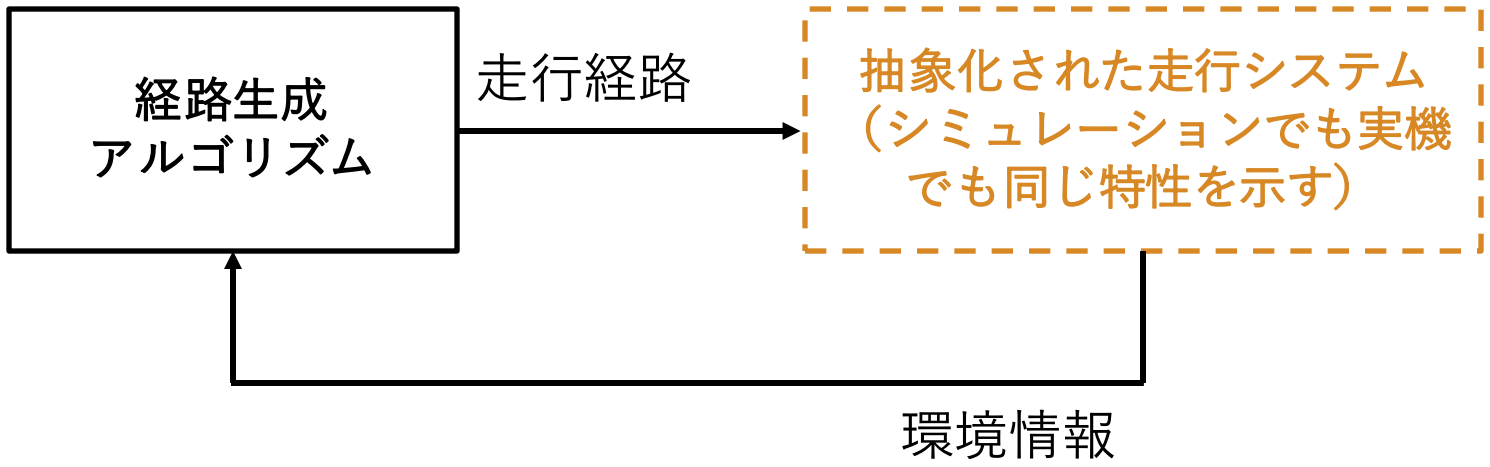


# 「判断」が失敗しないための「制御」要件

強化学習方策が「下位のシステムは学習した時のものと同じ」に見えるように制御応答を作る必要がある。

方策は非線形な深層ネットワークであるため、どのくらい近ければ同じになるのか、数学的に求めることは非常に困難である。

実際の設計では、どのくらい差異が生じたら問題になるのかをシミュレーションで評価することになる。



# まとめ

- 「認知」「判断」「制御」という枠組みの中で、強化学習を「判断」にのみ用いることで、強化学習の強みを活かしつつデメリットを解消した
- 周囲環境をシンプルな行列で表現し、またその変化速度も合わせて行列表現し、その入力を元に経路を生成する強化学習方策を設計した
- 遅れなく指令値軌道に追従できるモデル予測制御を活用し、強化学習による指令値軌道生成と走行制御の協調システムを実現した

# 目次

- 倒立振子を強化学習で倒立させる制御の設計、及び実機実装
- 強化学習とモデル予測制御を用いて、往来する歩行者を避けながら自律運転する制御を設計
- 地に足付けた制御設計機能
- 役に立つコンテンツ集

# MathWorks 制御系設計ツール一覧

| | 製品 | 概要 |
|---|---|---|
| 1 | Control System Toolbox | 線形制御理論（古典/現代）に基づく制御系のモデリング・解析・設計 |
| 2 | System Identification Toolbox | ブラックボックスモデリング（システム同定）、グレーボックスモデリング |
| 3 | Robust Control Toolbox | 不確かさをもつプラントに対するロバスト制御系解析・設計 |
| 4 | Model Predictive Control Toolbox | モデル予測制御の設計・シミュレーション |
| 5 | Fuzzy Logic Toolbox | ファジーロジック系の設計・シミュレーション |
| 6 | Simulink Control Design<br>線形制御理論 | Simulink モデルを起点とした制御系解析・設計<br>・システム応答解析（時間/周波数領域）<br>・システム動作点/平衡点計算<br>・制御器の自動調整（PID はじめ任意構造の線形 SISO/MIMO 系） |
| 7 | Simulink Design Optimization<br>数値最適化 | Simulink モデルのパラメータ自動調整<br>・プラントモデル精度向上： モデルの測定データ（応答波形）への合わせ込み<br>・応答最適化： 制御性能要件（時間/周波数領域）を満たす自動調整<br>・感度解析： 要件に対するパラメータの感度評価 |
| 8 | Predictive Maintenance Toolbox | 状態監視・予知保全向けアルゴリズムの設計・評価 R2018a |
| 9 | Reinforcement Learning Toolbox | 強化学習による方策（Policy）の設計・学習 R2019a |

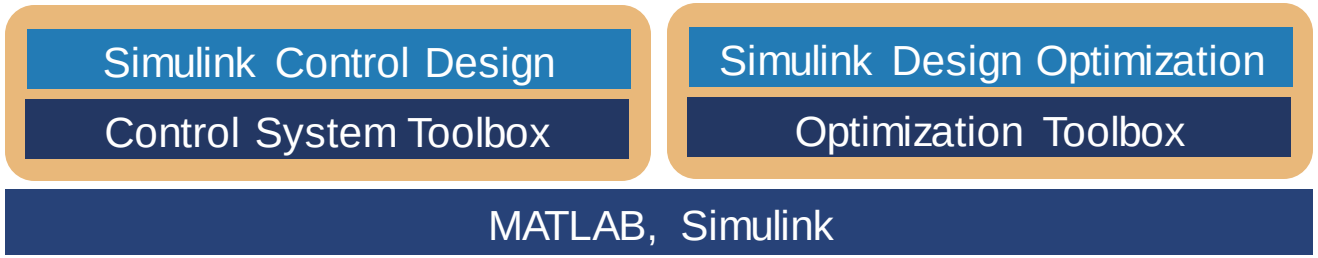
# Simulink Control Design × Simulink Design Optimization

## 制御系設計の初心者から熟練者まで幅広い場面で利用できるツール

1. Simulink モデルを前提としたツール作りがされている
2. 制御系設計の工程全体をほぼ網羅している
3. GUI アプリが充実している（関数も豊富に提供）

## こんな方におススメ

- 制御系設計の初心者の方
- MATLAB 言語による作り込みまでは少し躊躇される方
- 提供されている高水準な機能を利用して自動化・カスタム化を行いたい方



# 2 つのツールをお勧めする理由は、工程全体をほぼ網羅するため

**制御系設計の目的**

- 制御方式/アルゴリズムと定数の導出
- 制御系の振る舞いを予測

制御系定義

制御対象解析

性能要件定義

制御方式設計

制御定数適合

性能評価・検証

制御対象モデル化

アプリ・機能

パラメータ推定ツール
Simulink Design Optimization

システム同定
System Identification Toolbox

モデル・リデューサー
Control System Toolbox

設計用

制御対象モデル獲得ループ

要求獲得ループ

制御対象の特性解析

- 静特性（定常動作点）
- 動特性（時間/周波数応答）

制御系のモデル化

- 制御系構造・I/O
- 制御器・アルゴリズム

制御定数の調整/適合

- 性能要件出し
- ゲインの目途づけ
- 性能要件の達成

制御系のシミュレーション

- 制御性能評価・検証

アプリ・機能

定常状態マネージャー
Simulink Control Design

モデル・リニアライザー
Simulink Control Design

ブロックライブラリ
Simulink & Add-on libraries

PID 調整器
Simulink Control Design

制御システム調整器
Simulink Control Design

応答最適化ツール
Simulink Design Optimization

感度解析ツール
Simulink Design Optimization

# 制御系設計 GUI アプリ例

| アプリ名 | 用途 | 製品 | 導入リリース | 備考 |
|---|---|---|---|---|
| 定常状態マネージャー | 解析（平衡点/動作点） | SLCD | 18b | 18b で GUI が刷新 |
| モデル線形化器 | 解析（応答特性） | SLCD | 11b | 20a で「線形解析ツール」から名称変更 |
| PID調整器 | 制御器の自動調整 | CST / SLCD | 14b/10b | `pidTuner` |
| 制御システムデザイナー | | CST / SLCD | 15a | sisotool の後継 |
| 制御システム調整器 | | CST / SLCD | 16a | RCT に 14a で導入、16a に移譲<br>`controlSystemTuner` |
| 応答オプティマイザー（応答最適化ツール） | | SLDO | 11b | `sdotool` |
| 感度アナライザー（感度解析ツール） | 感度解析 | SLDO | 16a | `ssatool` |
| パラメータ推定器（パラメータ推定ツール） | パラメータ同定 | SLDO | 06a | `spetool` |
| モデル・リデューサー | モデル低次元化 | CST | 16b | `modelReducer` |

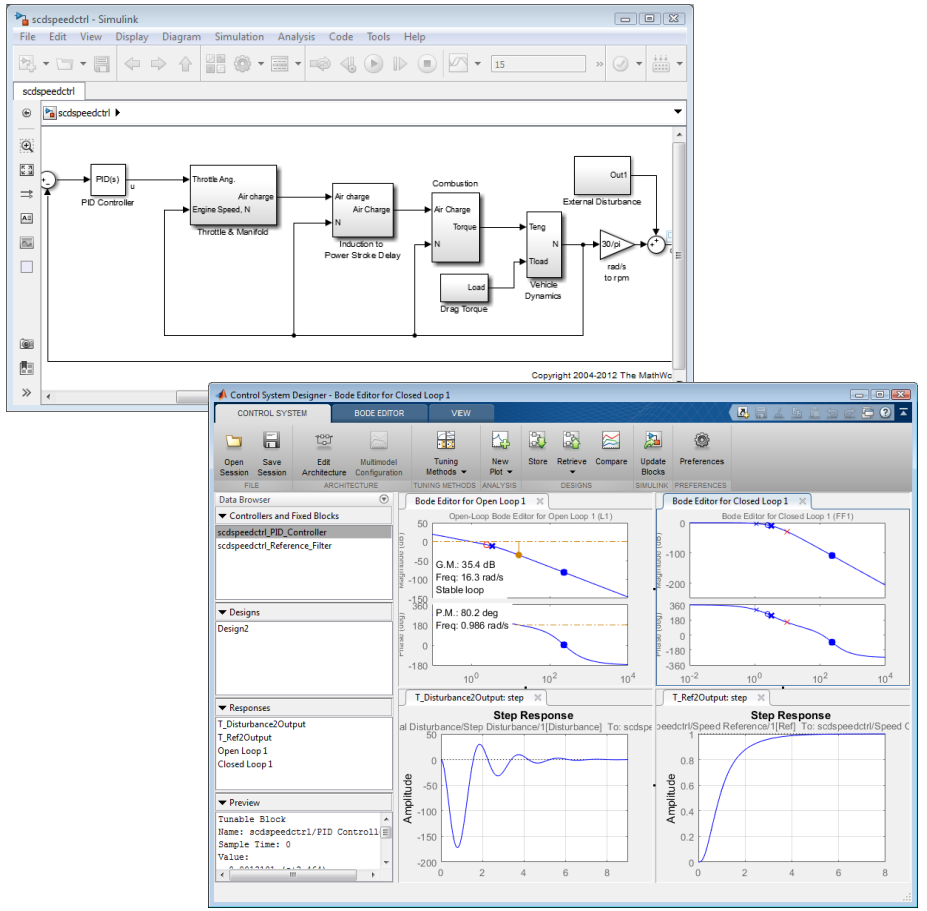
CST: Control System Toolbox　　SLCD: Simulink Control Design
RCT: Robust Control Toolbox　　SLDO: Simulink Design Optimization

# Simulink Control Design

# Simulink Control Design

## モデル線形化と制御系設計

- 主な機能：
  - Simulink での PID 制御、ゲインスケジュール制御、および任意の SISO/MIMO 制御系の自動チューニング
  - 組込ソフトウェアに展開可能な PID 自動調整アルゴリズム
  - 動作点計算（平衡化）とモデル線形化
  - シミュレーションデータからの周波数応答の推定
  - 複数の動作点のバッチ線形化
  - Simulink Design Optimization による時間/周波数領域の要件を満たすように補償器をチューニング



# C コード生成可能な PID のオンライン自動調整が利用可能に

PID Autotuner
Simulink Control Design

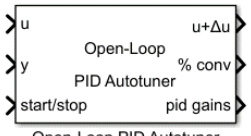
## 目標性能仕様

- 制御帯域幅（応答性）
- 位相余裕（安定度）

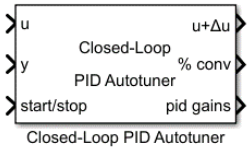
## テスト信号仕様

- 正弦波振幅
- ステップ振幅
  （開ループの場合）

開ループ
PID Autotuner

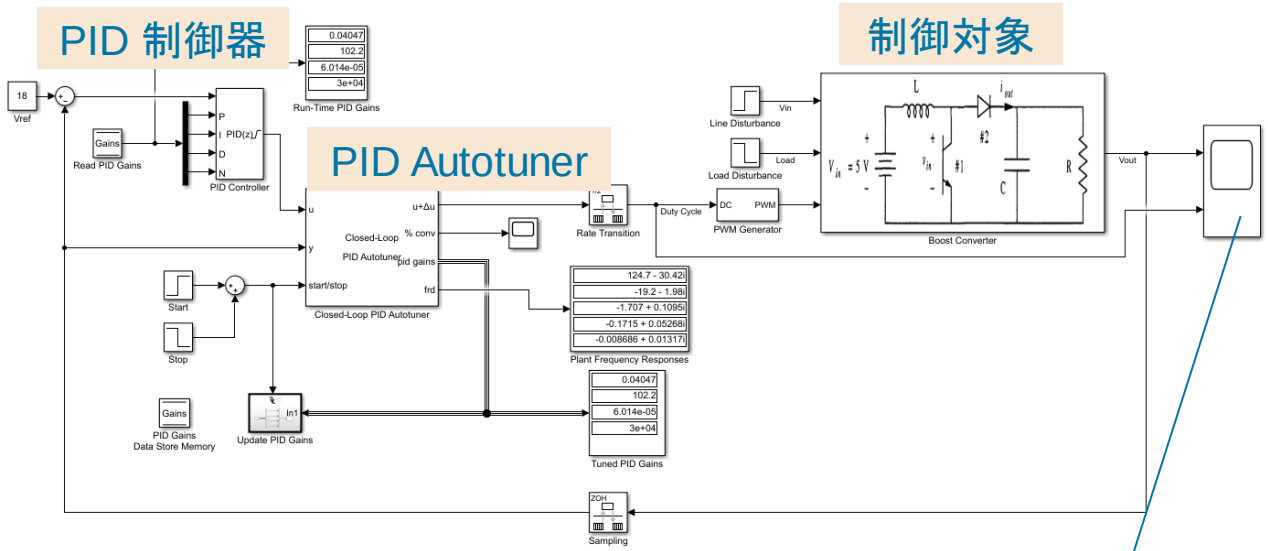


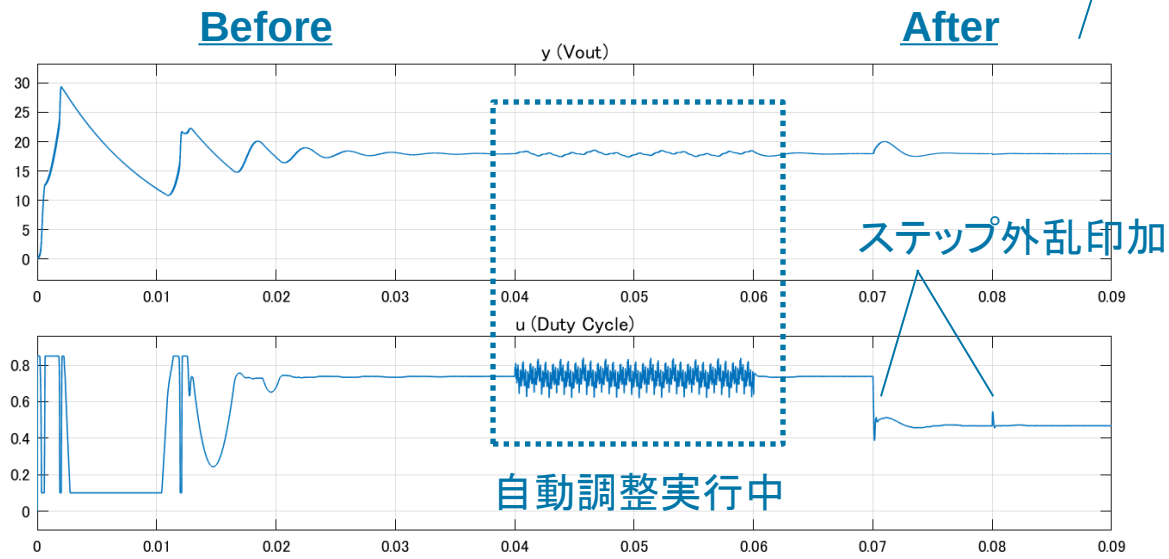

閉ループ
PID Autotuner




自動 C コード生成

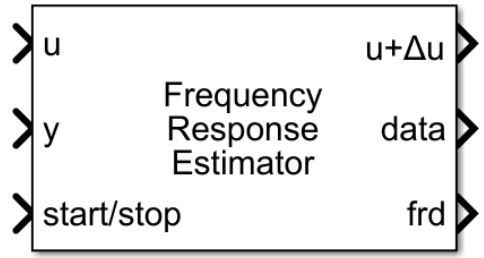
他の計算環境への展開



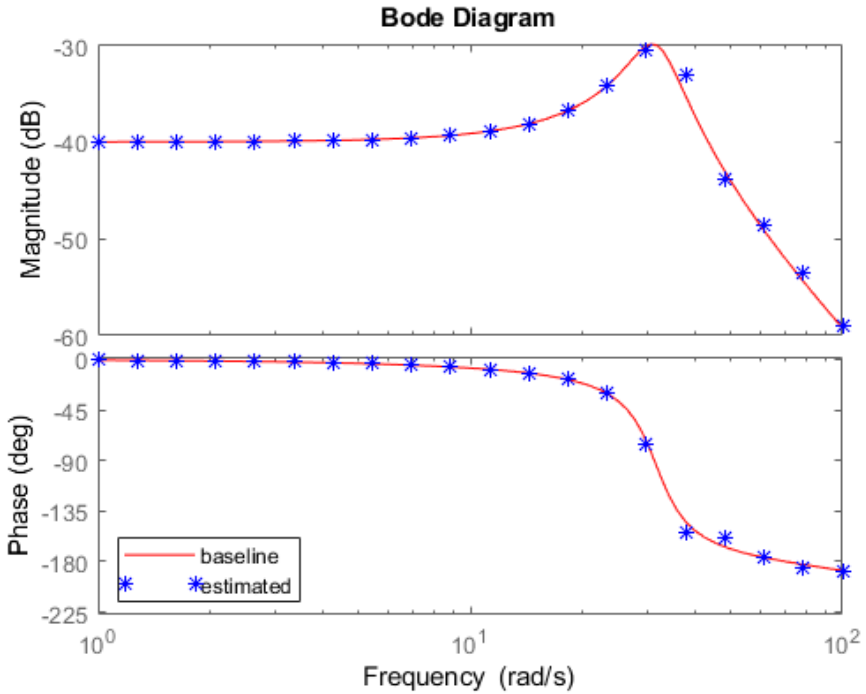



# 周波数応答推定器

- “Closed-Loop PID Autotuner”と同じように、
  - モデルフリーで周波数応答の推定が可能
  - 推定機能をコード生成して実機に実装可能
- 閉ループ、開ループ問わず推定可能

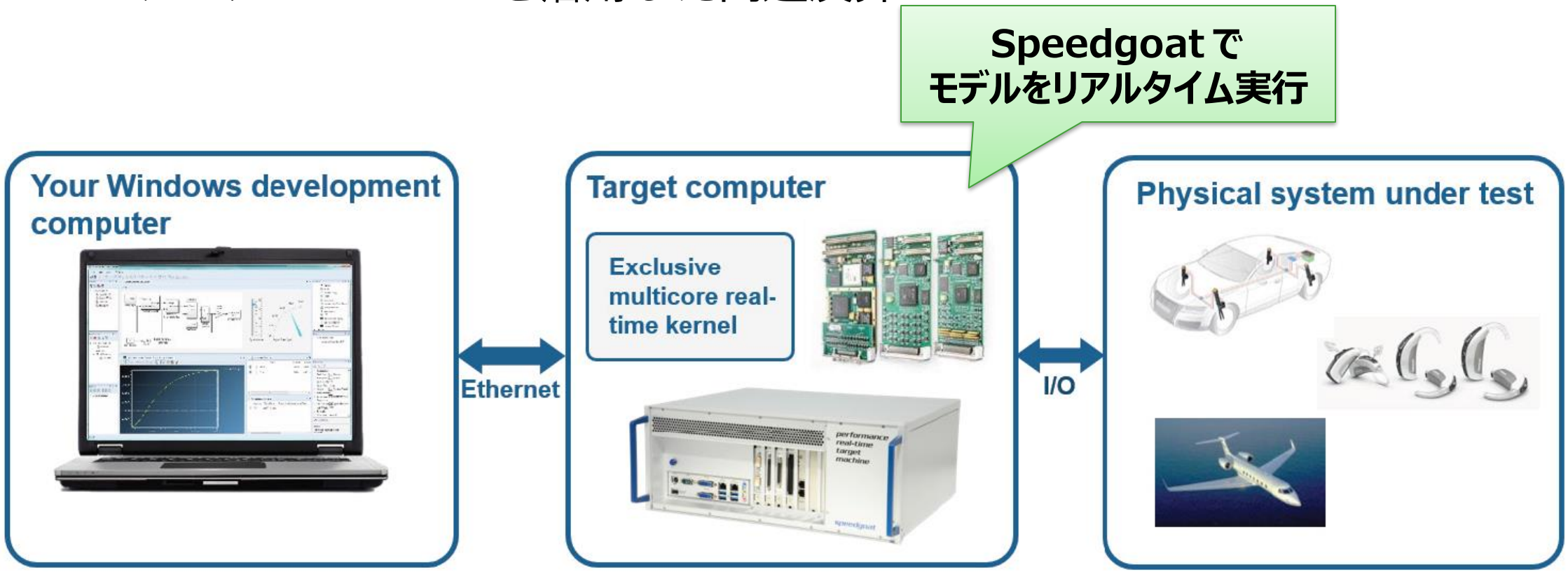



今回は「speedgoat」という
リアルタイムシミュレータに実装し、
実機で周波数応答の推定を行います。

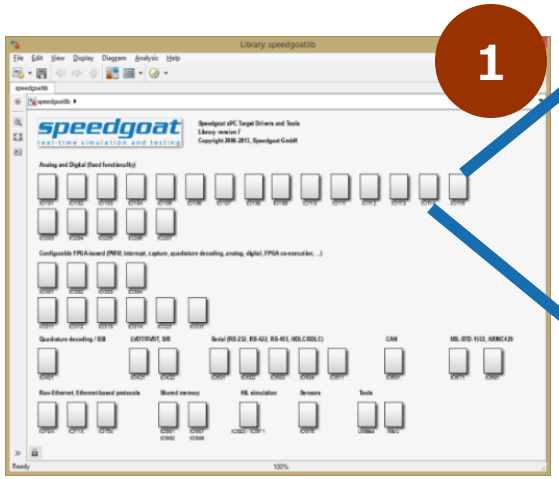


# RCP・HILSテスト環境 Simulink Real-Time/Speedgoat

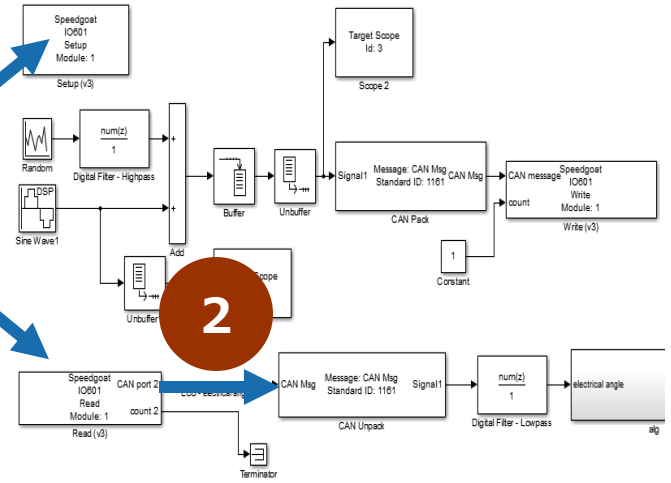
- 専用ハードウェア Speedgoat を活用したRCP/HILSテスト環境
- MATLAB/Simulinkとのネイティブな接続とシームレスな連携
- マルチコアCPU/FPGAを活用した高速演算



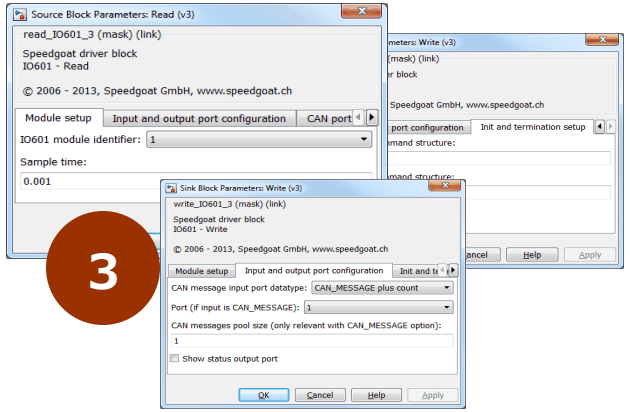
# Speedgoatへの実装ワークフロー



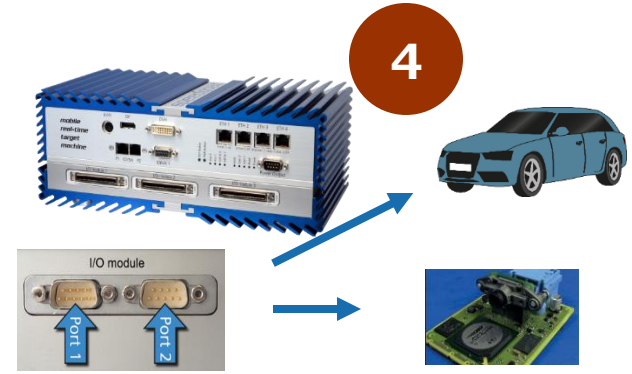

Driver blocks




Simulink Model




Dialog Fields




Connect to real-time application

1 ハードウェアに搭載されたIOボードに対応する設定用ブロック・ドライバブロックをモデルに配置します

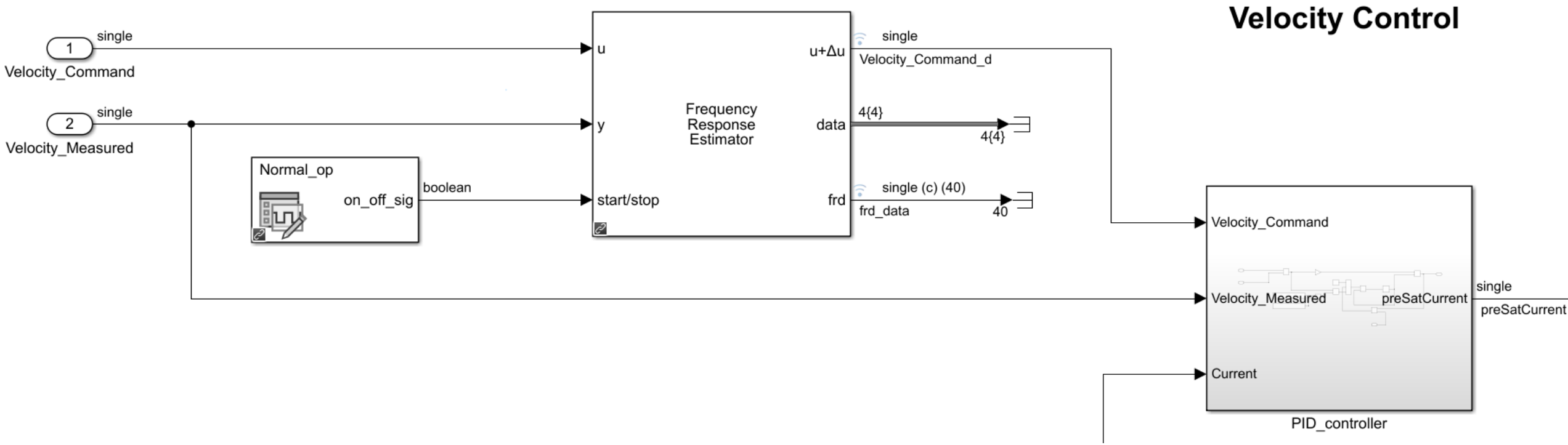
2 アルゴリズムとドライバブロックを信号線で接続します

3 設定用ブロック・ドライバブロックからダイアログを開いてI/Oに関する各種設定を行います

4 ビルドボタンを押してコード生成およびビルドを行いますアプリケーションのダウンロードで実行可能な環境が整います

# 周波数応答推定器を制御モデルに挿入

今回は速度指令値に対する速度実測値の応答が、どのような周波数応答になるのかを調べる。
以下のように速度指令値を”u”、速度実測値を”y”に接続し、”u+Δu”を新しい速度指令値として用いる。



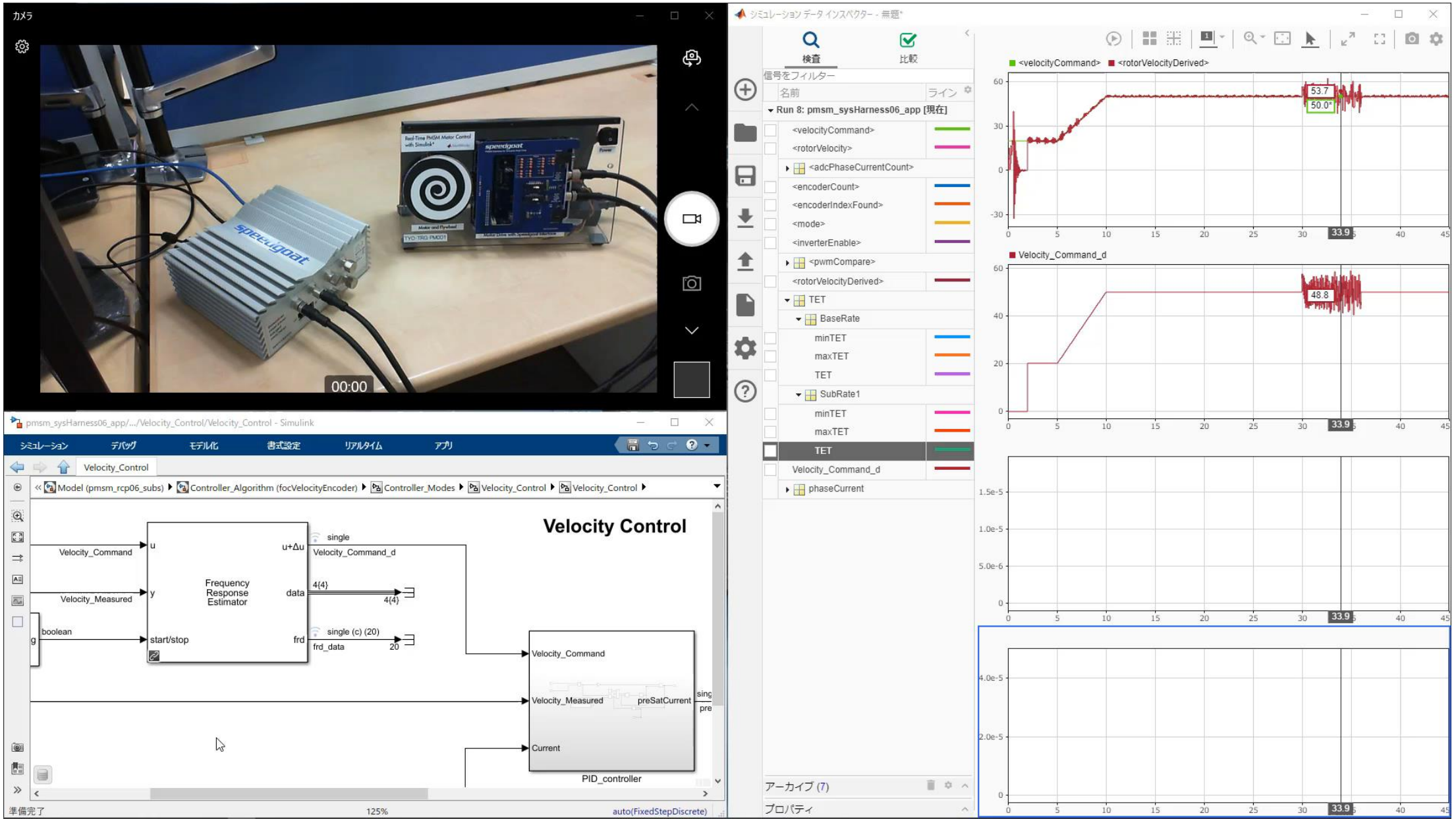
# 実機の構成

MathWorks社のトレーニングで用いる
モーター制御キット


モーター制御基板への
PWM出力、電圧電流、
エンコーダ値などの信号
をやり取りしている

speedgoat Baseline

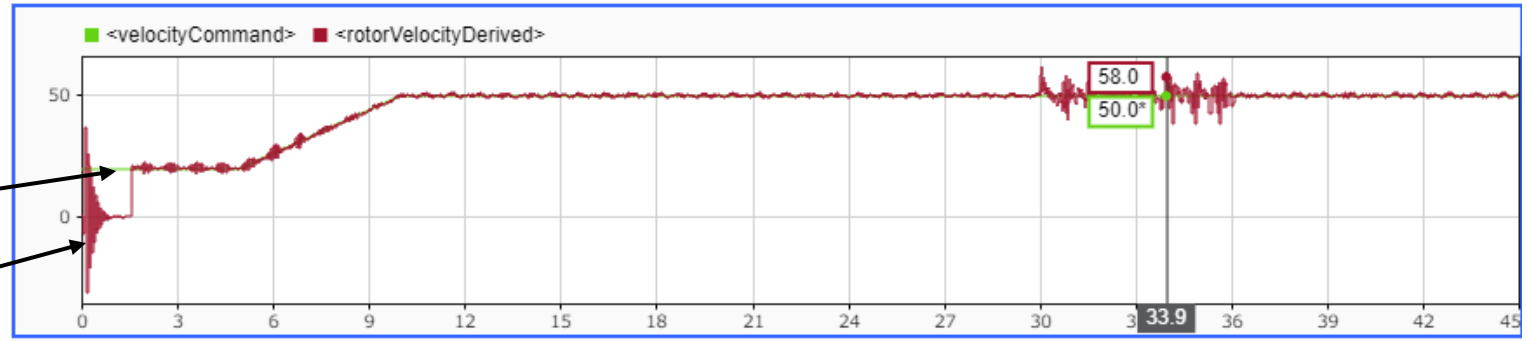
# 実行の様子


カメラ
00:00
pmsm_sysHarness06_app/.../Velocity_Control/Velocity_Control - Simulink
シミュレーション
デバッグ
モデル化
書式設定
リアルタイム
アプリ
Velocity_Control
Model (pmsm_rcp06_subs)
Controller_Algorithm (focVelocityEncoder)
Controller_Modes
Velocity_Control
Velocity_Control
Velocity_Command
Velocity_Measured
boolean
start/stop
u
y
u+Δu
data
frd
Frequency Response Estimator
single
Velocity_Command_d
4{4}
single (c) (20)
frd_data
20
Velocity_Command
Velocity_Measured
Current
preSatCurrent
PID_controller
準備完了
125%
auto(FixedStepDiscrete)
シミュレーション データ インスペクター - 無題*
検査
比較
信号をフィルター
名前
ライン
Run 8: pmsm_sysHarness06_app [現在]
<velocityCommand>
<rotorVelocity>
<adcPhaseCurrentCount>
<encoderCount>
<encoderIndexFound>
<mode>
<inverterEnable>
<pwmCompare>
<rotorVelocityDerived>
TET
BaseRate
minTET
maxTET
TET
SubRate1
minTET
maxTET
TET
Velocity_Command_d
phaseCurrent
アーカイブ (7)
プロパティ
<velocityCommand> <rotorVelocityDerived>
53.7
50.0
33.9
Velocity_Command_d
48.8

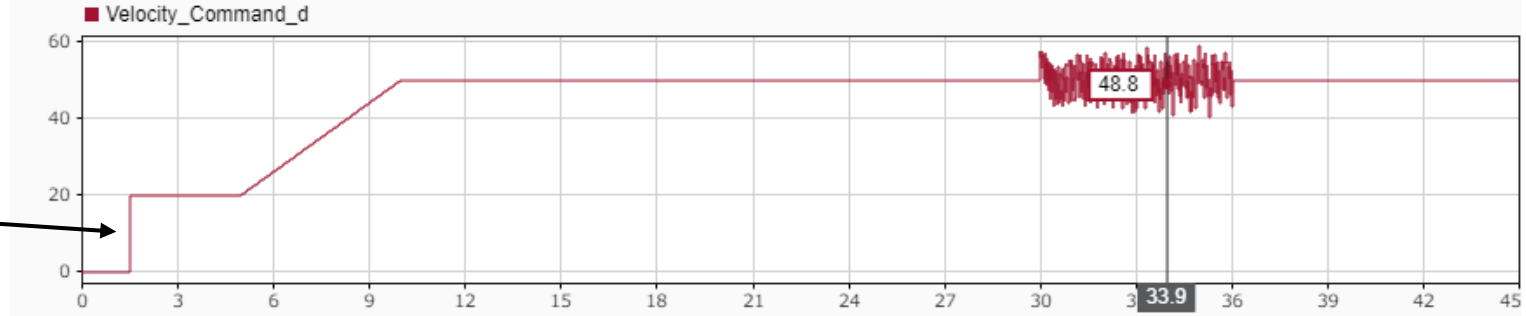
# 実行結果

速度指令値[rpm]

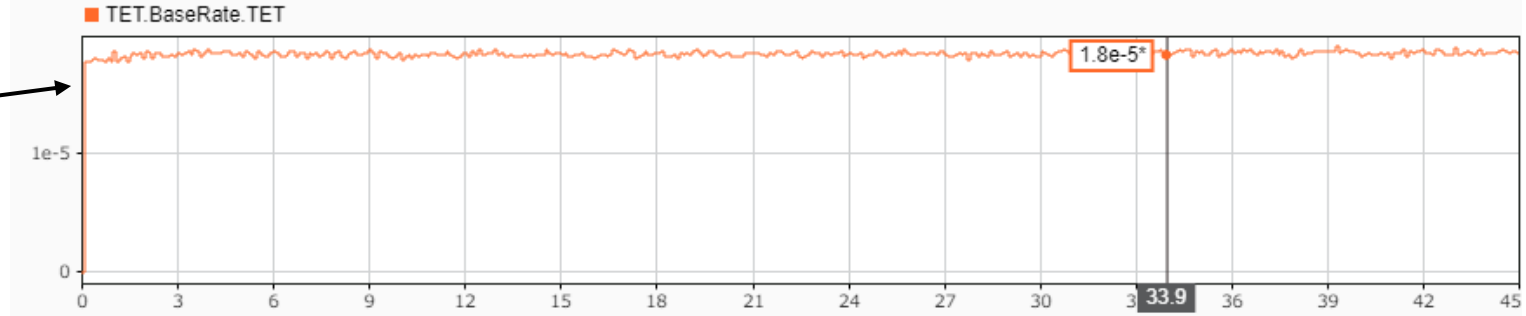
速度実測値[rpm]




擾乱を加えた
速度指令値[rpm]

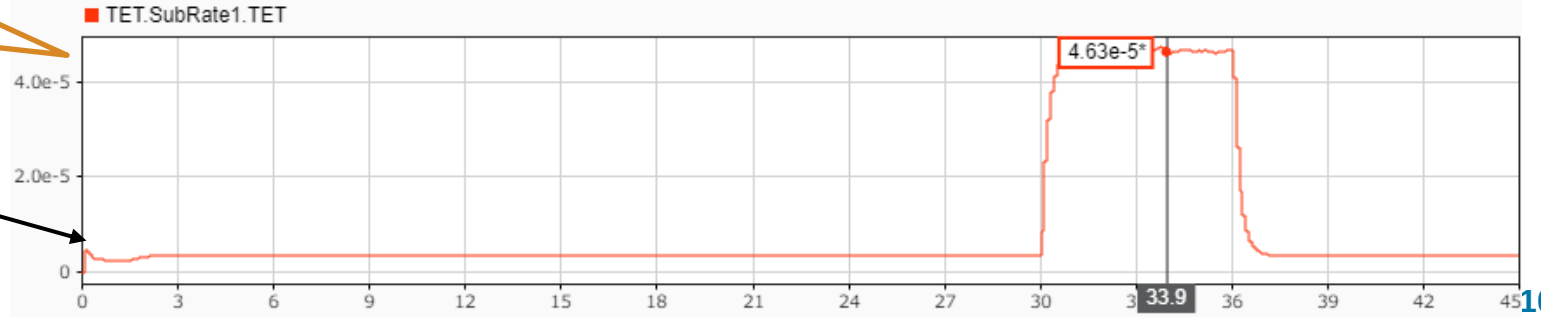



電流制御の
処理時間[s]



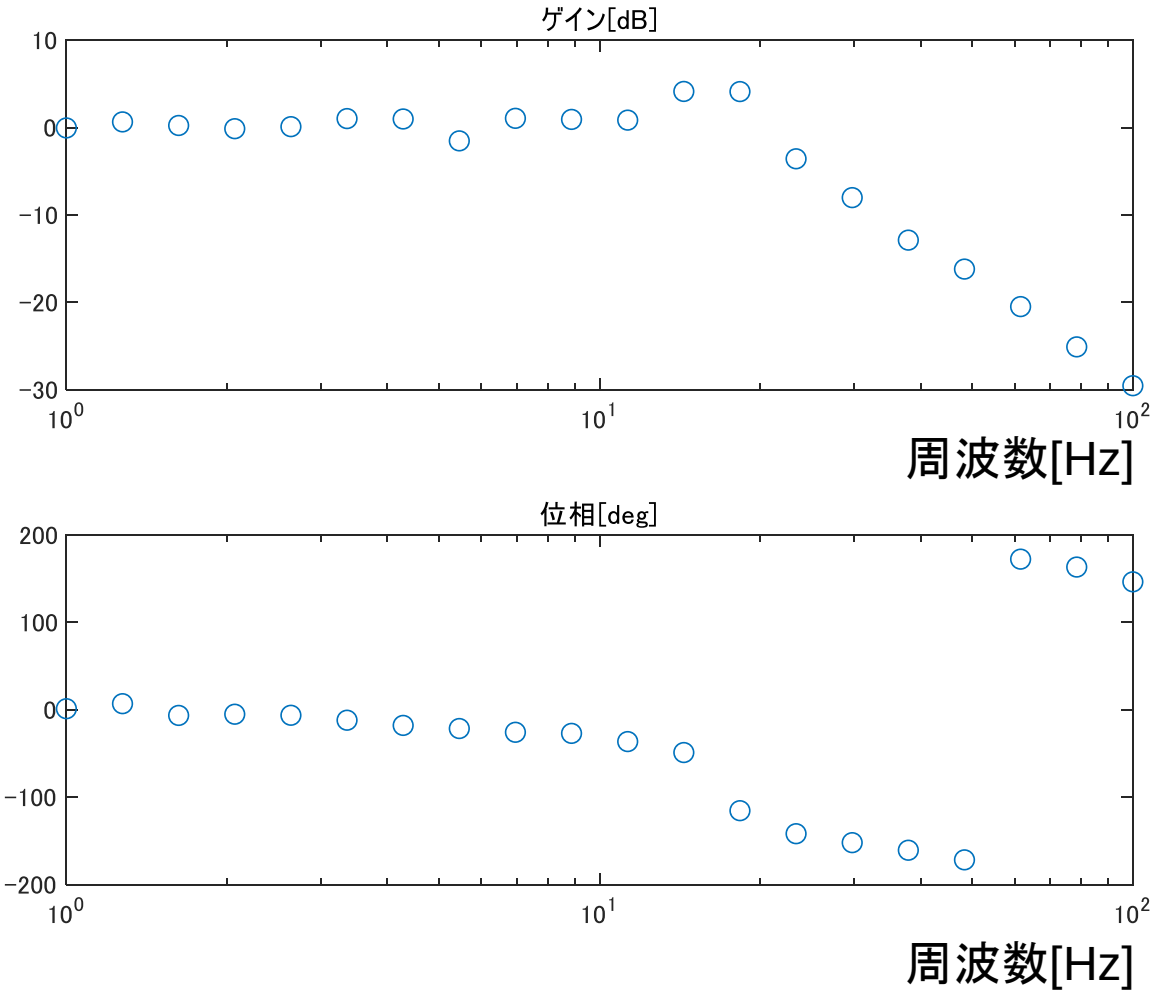

周波数応答推定を行ったことによって、
一時的に計算負荷が増大している

速度制御の
処理時間[s]



# 周波数応答推定結果

得られた複素数のデータ列からボード線図を描画すると、以下のようになる。




**System Identification Toolbox**

周波数応答データから
伝達関数を同定することも可能

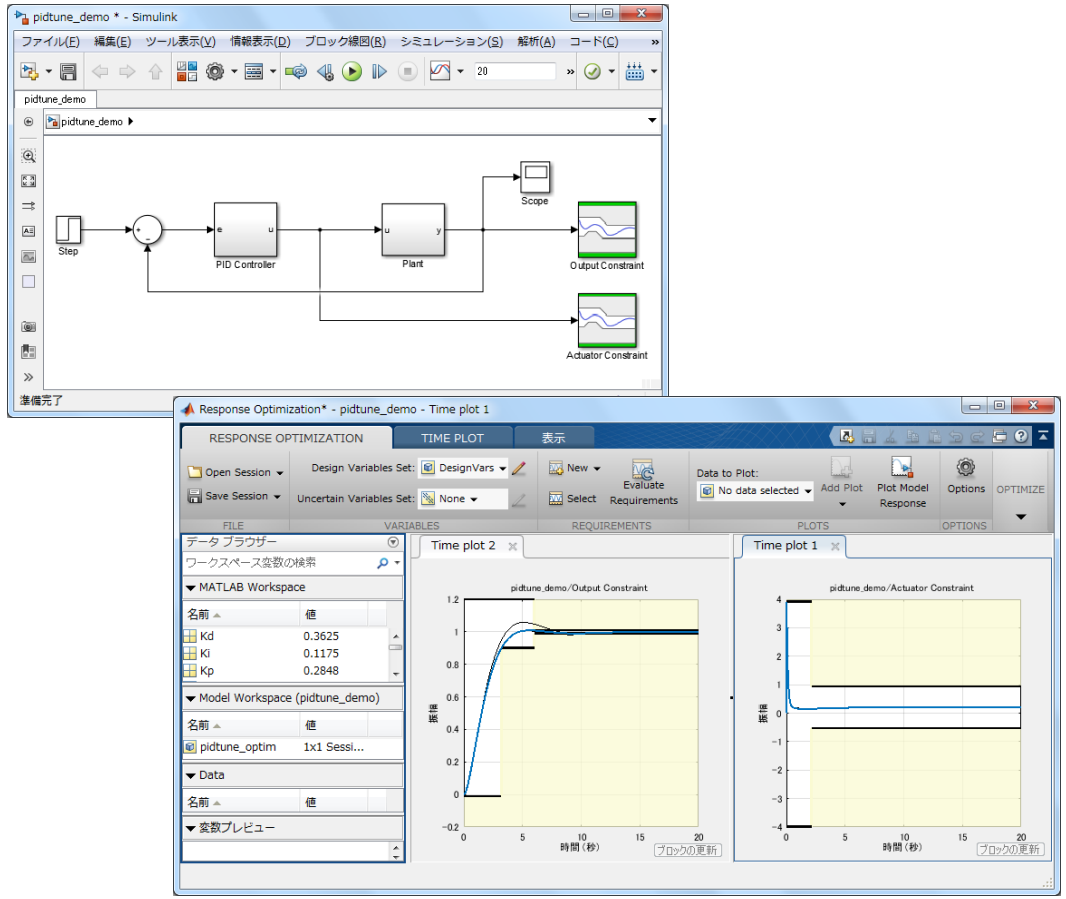
```
estimated_model =

   0.8156 s + 1.103e04
-------------------------
s^2 + 61.1 s + 1.194e04
```

# Simulink Design Optimization

# Simulink Design Optimization

## モデル感度の解析とモデルパラメータのチューニング

- 主な機能：
  - テストデータからパラメータを推定
  - 時間/周波数領域、あるいは、カスタムの性能要件
    を満たすようにパラメータをチューニング
  - 設計空間の探索と感度解析
  - グラフィカルな性能要件の指定と
    最適化の進捗モニタリング
  - パラメータ変動や不確かさを考慮した
    ロバストな設計最適化



# データを活用し、制御対象モデルの精度を向上させる

## 入力

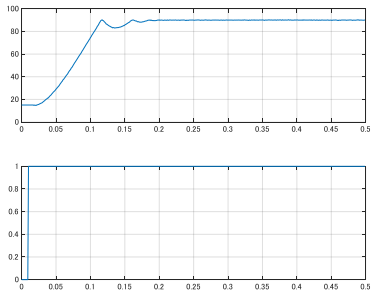
### パラメータ指定

モデルパラメータ
（ワークスペース変数）

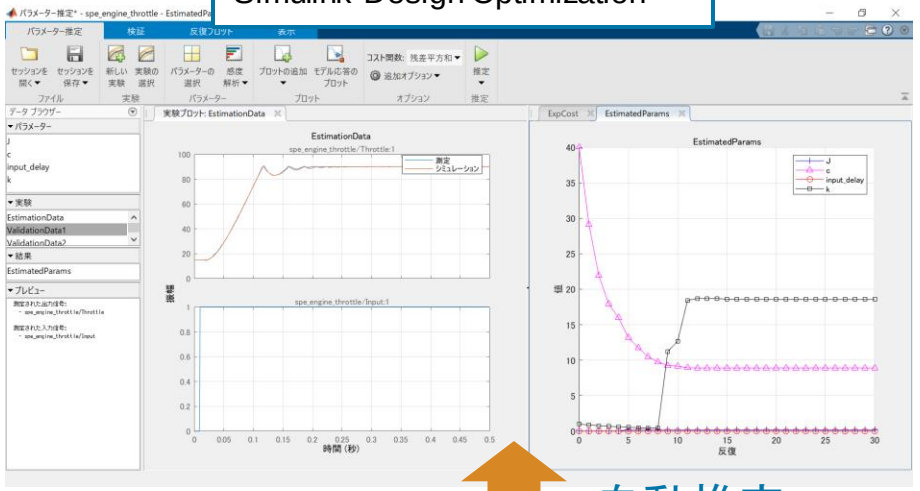
| 名前 | 値 |
|---|---|
| c | 40 |
| input_delay | 0.0200 |
| J | 0.0500 |
| k | 1 |

### 時系列データ

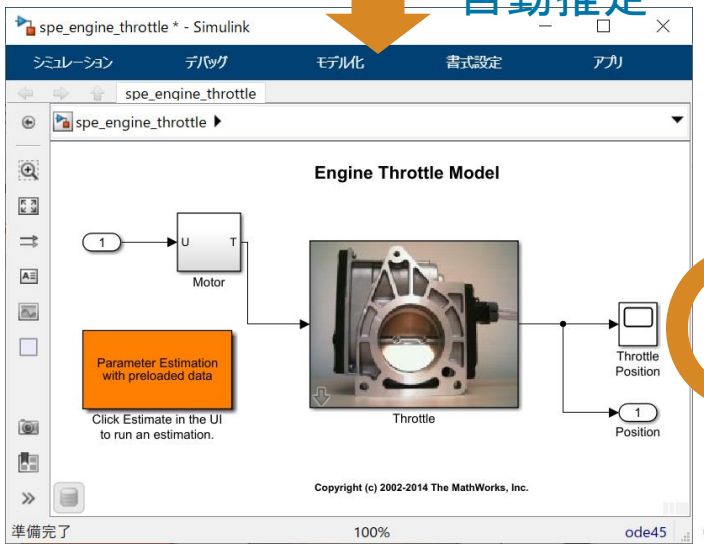
実機/詳細モデルデータ
（合わせ込み用/妥当性確認用）



## パラメータ推定器
Simulink Design Optimization



自動推定




## 出力

### 推定結果

- パラメータ値
- 応答波形可視化
- 推定進行状況レポート

| 名前 | 値 |
|---|---|
| c | 8.9080 |
| input_delay | 0.0099 |
| J | 0.2264 |
| k | 18.5707 |

### 使いどころ

- 所有するモデルの精度を高め、解析・設計の精度を向上

高速リスタート、並列化にも対応

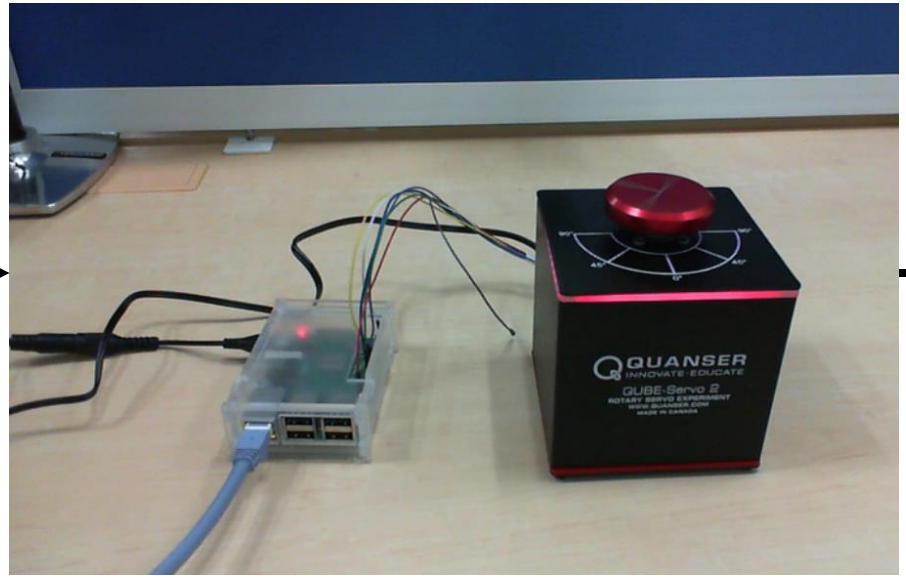
数値最適化技術

Optimization Toolbox

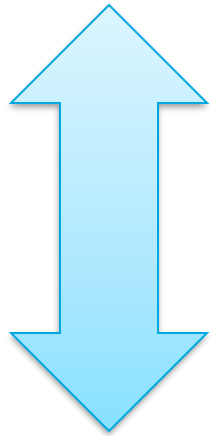
# [例] DCモーターのパラメータを推定する

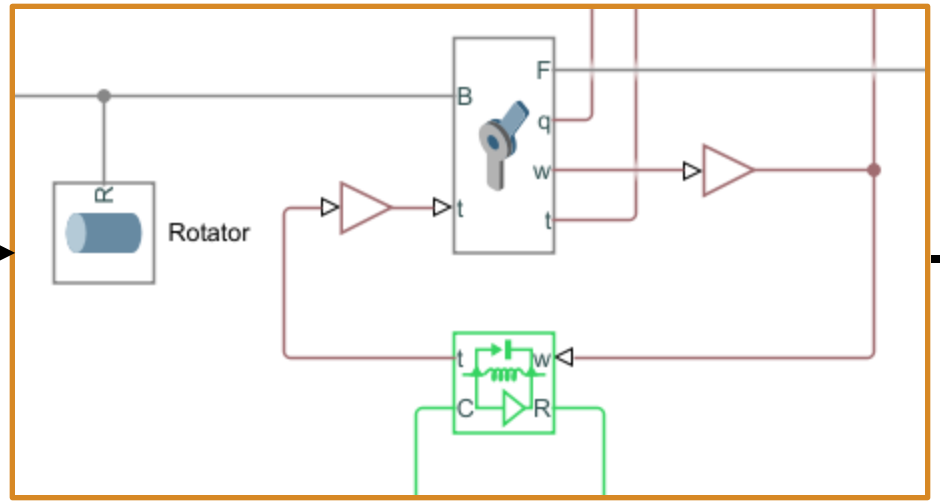
二つが一致するように
シミュレーションモデルの
パラメータを合わせこむ

入力信号




実機出力





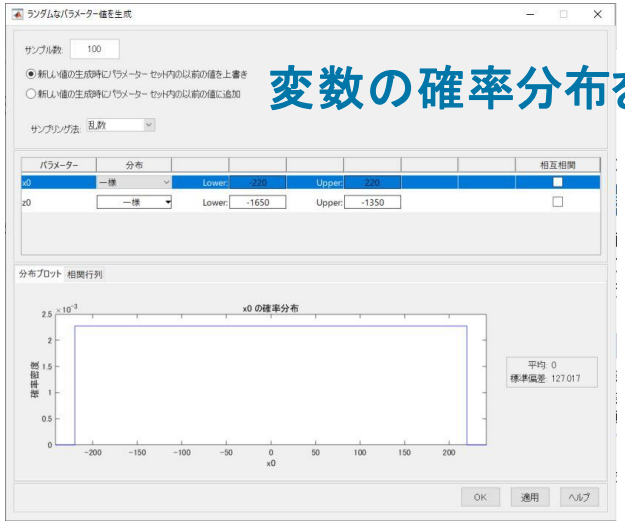

シミュレーション
出力

# モンテカルロシミュレーション（MCS）による感度解析評価

## 入力

### 変数指定

モデルパラメータや
制御定数等



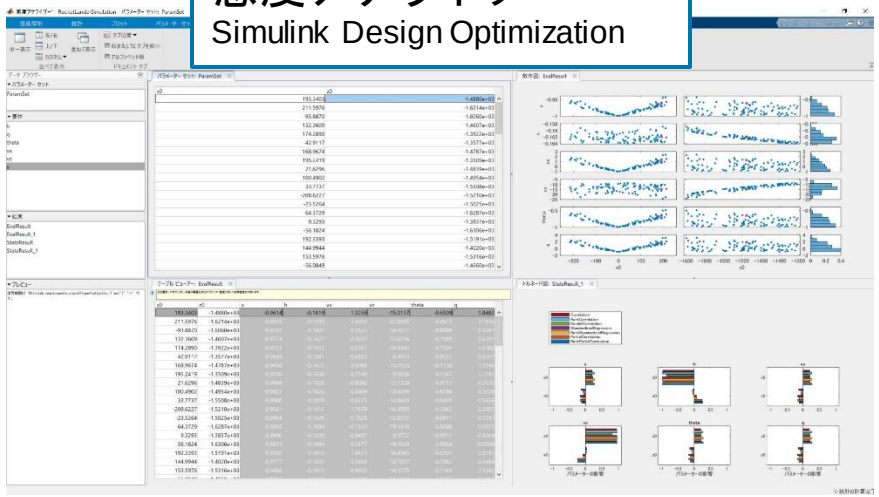

**変数の確率分布を仮定**

### 性能仕様

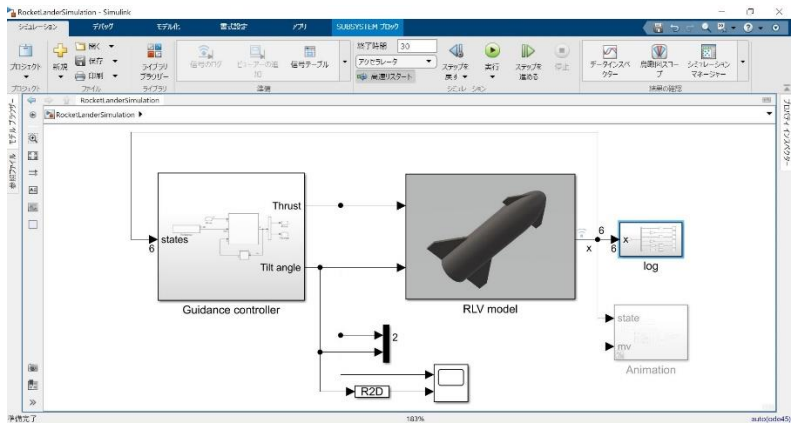
評価するための性能仕様を定義
（時間領域、周波数領域など）

## 感度アナライザー

Simulink Design Optimization



自動評価




## 出力

### 解析結果

- パラメータ値の組合わせ
- 統計的解析評価
  トルネードプ
  ロット図

### 使いどころ

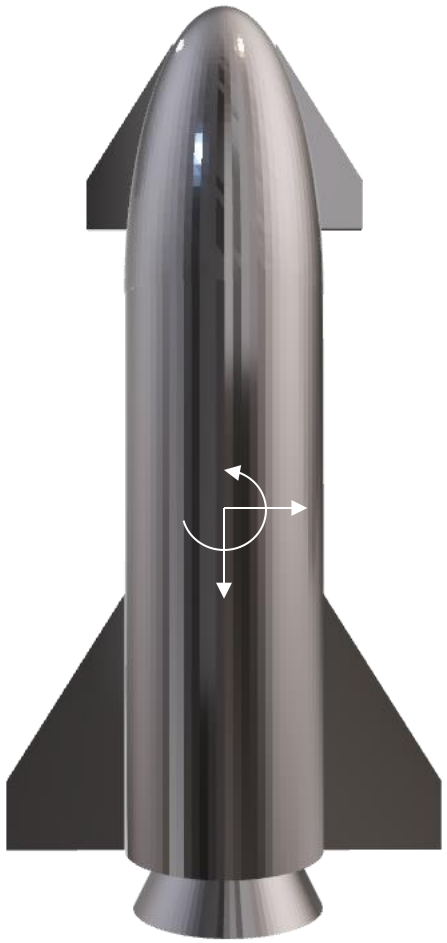
- MCSによる評価
- 影響を与える因子の特定
- 因子に基づく応答最適化
  （応答オプティマイザー）

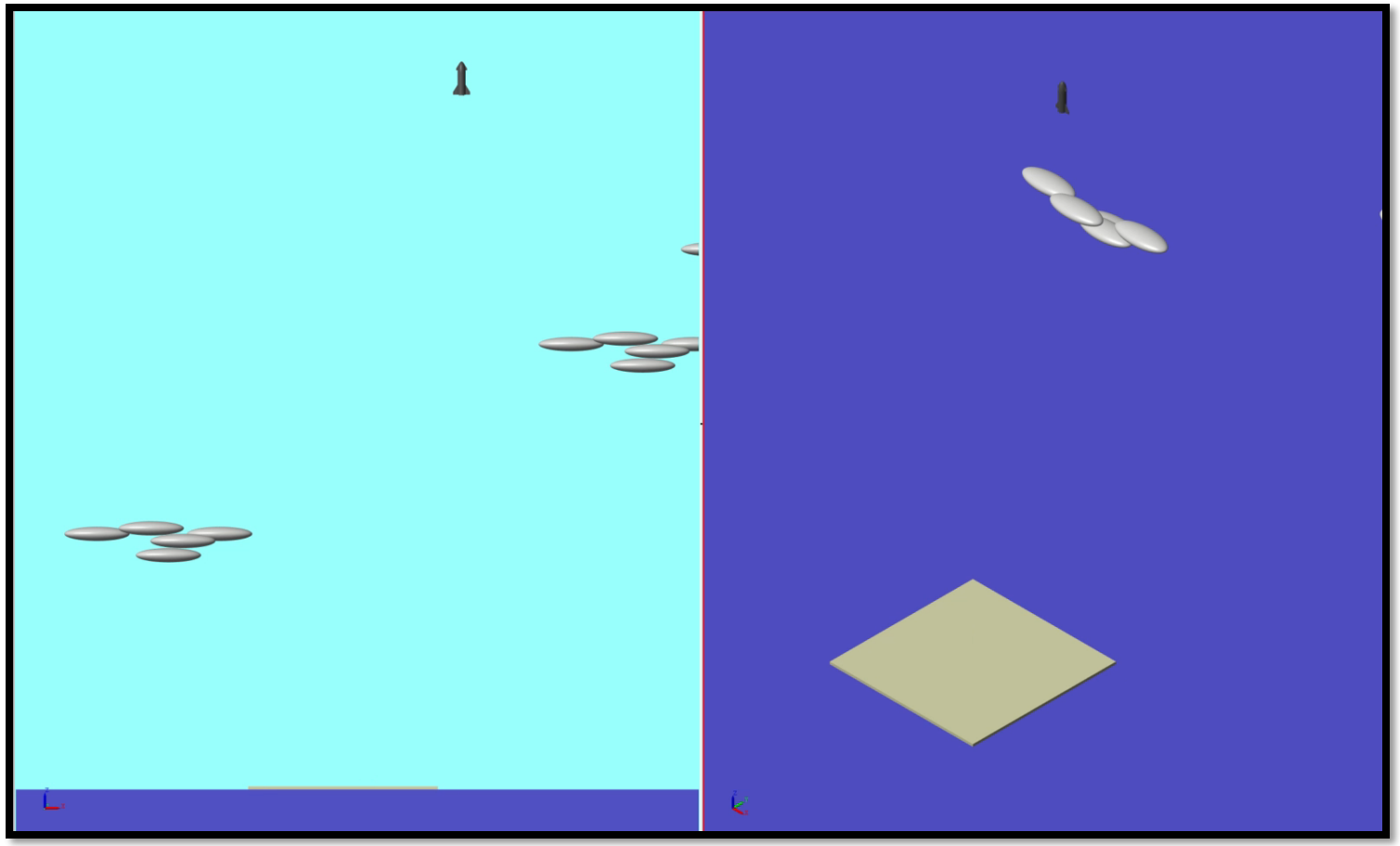
高速リスタート、並列化にも対応

統計学の技術

Statistics Machine Learning Toolbox

# [例] 再利用ロケットの垂直着陸制御に対する感度解析

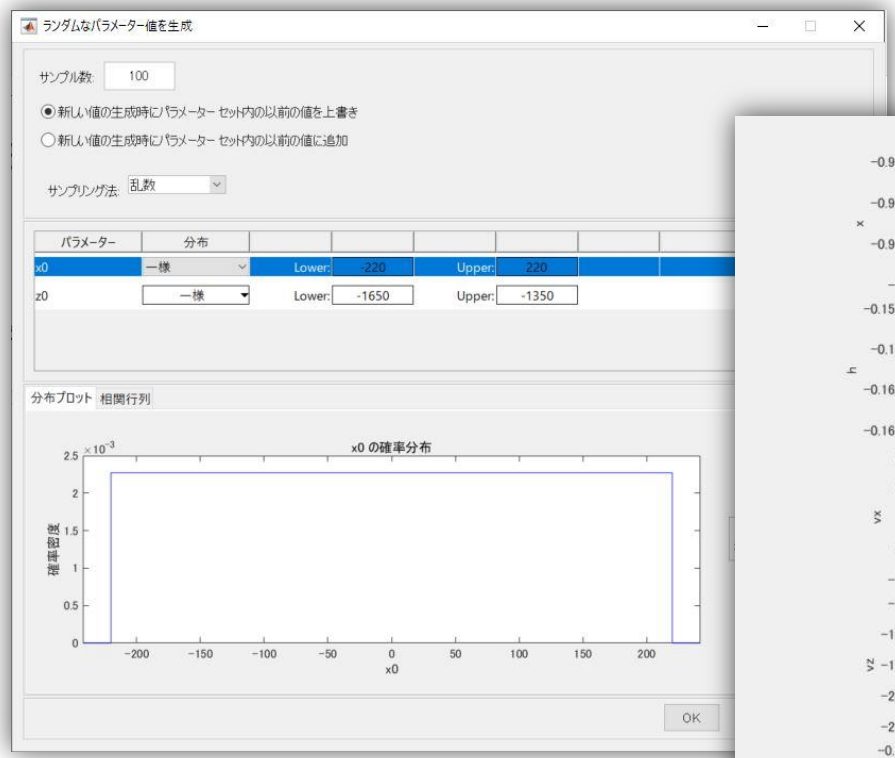


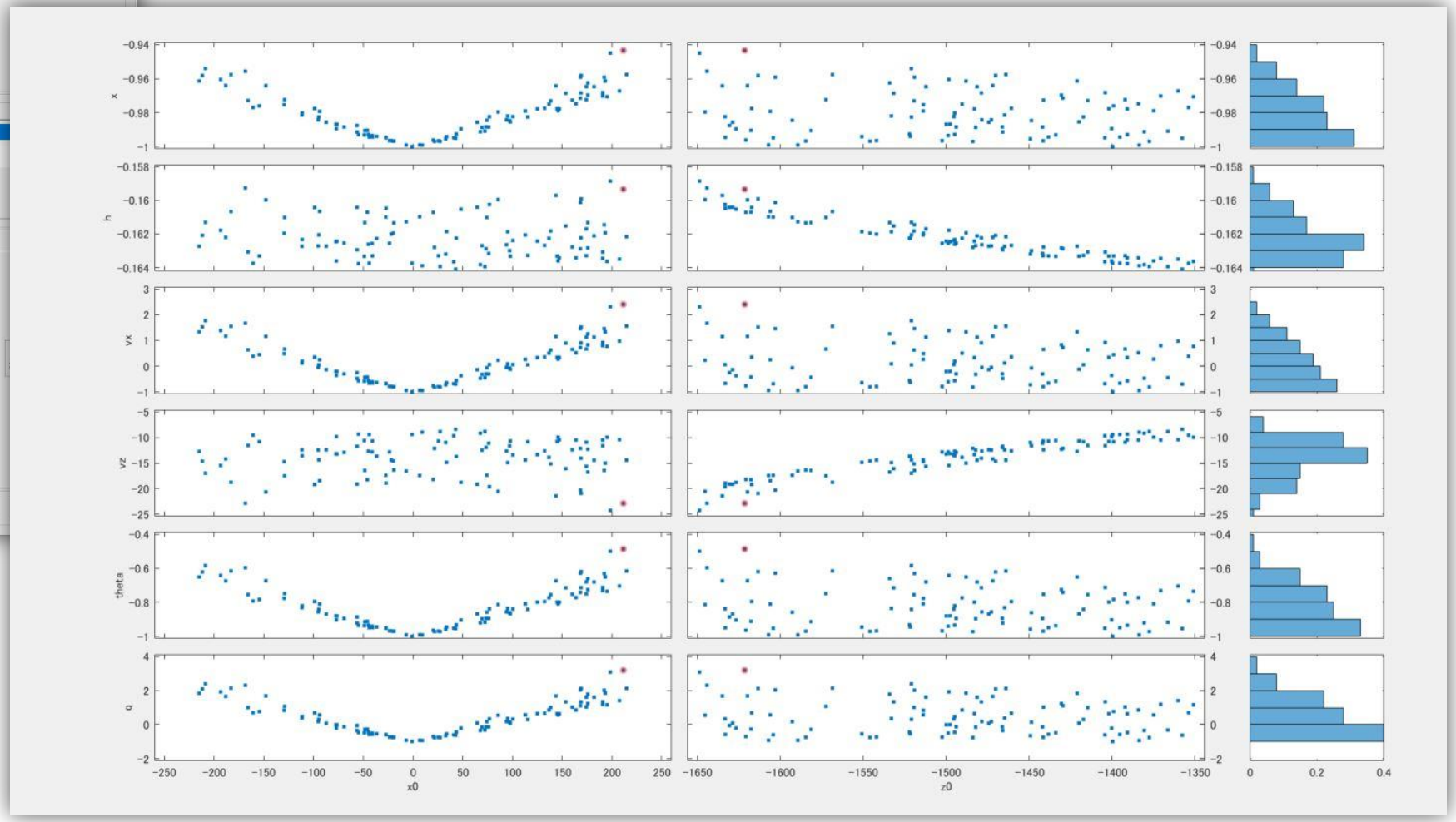



推力偏向制御(Thrust Vectoring Control)による誘導制御

# 不確かなパラメーターの評価

初期位置（ダウンレンジおよび高度）について一様な分布に基づいて100個のサンプルを生成し、着陸時における主要な状態量（高度や姿勢など）を評価



# 統計解析による評価

## トルネードプロット図


解析結果 例:

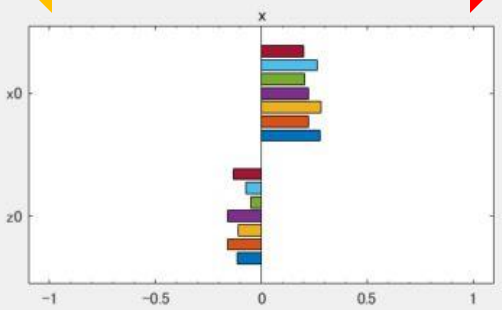
◆ 姿勢角についてx0の影響が最も大きく、正の相関を持つ

→初期ダウンレンジが大きいと着陸時の姿勢角（ブレ）が大きくなる

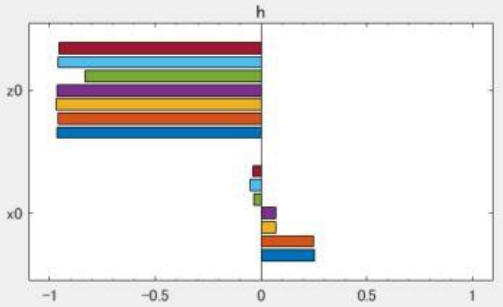
負の相関　　正の相関

影響大

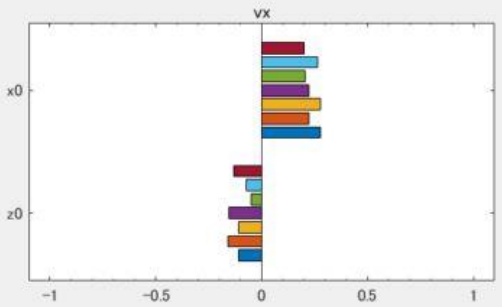
小



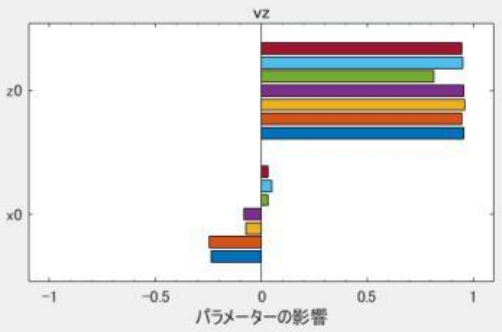

ダウンレンジ



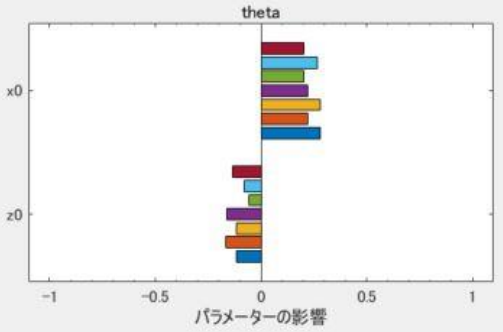

高度



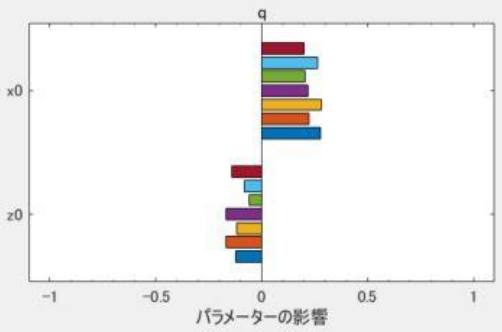

Vx




沈下率Vz




姿勢角




角速度

# 所有するモデルに自動調整を手軽に適用し、工数削減・性能向上を達成する

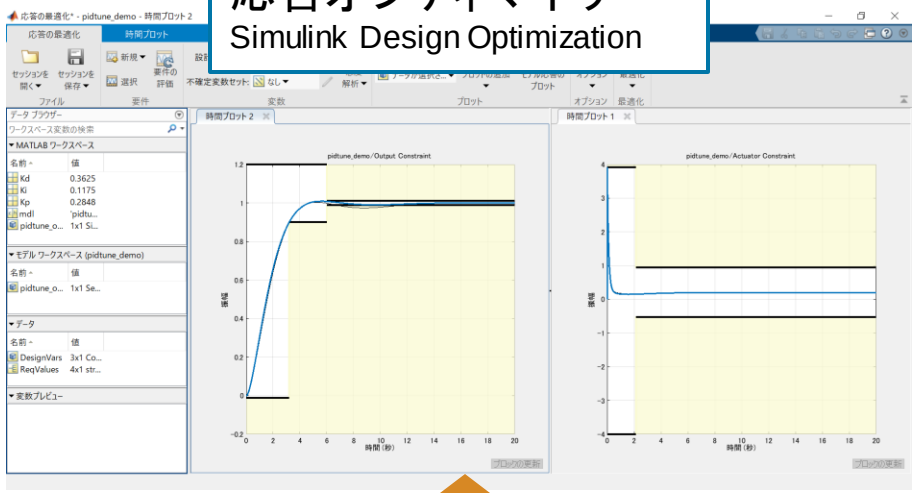
## 入力

**制御定数指定**

モデルパラメータ
（ワークスペース変数）

**目標性能仕様**

## 応答オプティマイザー
Simulink Design Optimization



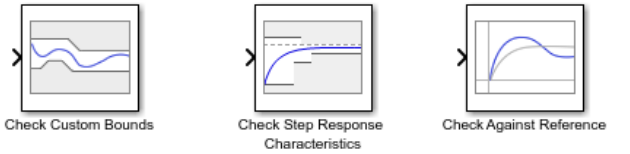
## 出力

**調整結果**

- パラメータ値
- 応答性能可視化
- 最適化進行状況レポート

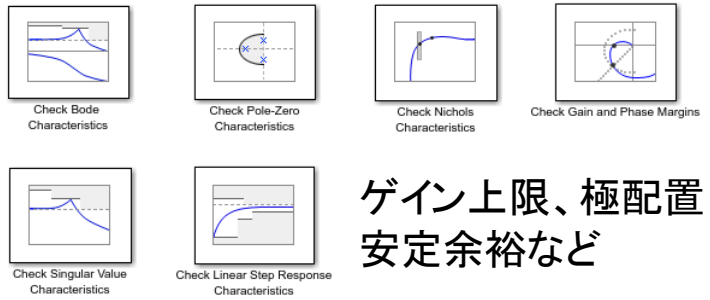
## 性能仕様定義ブロック

時間応答



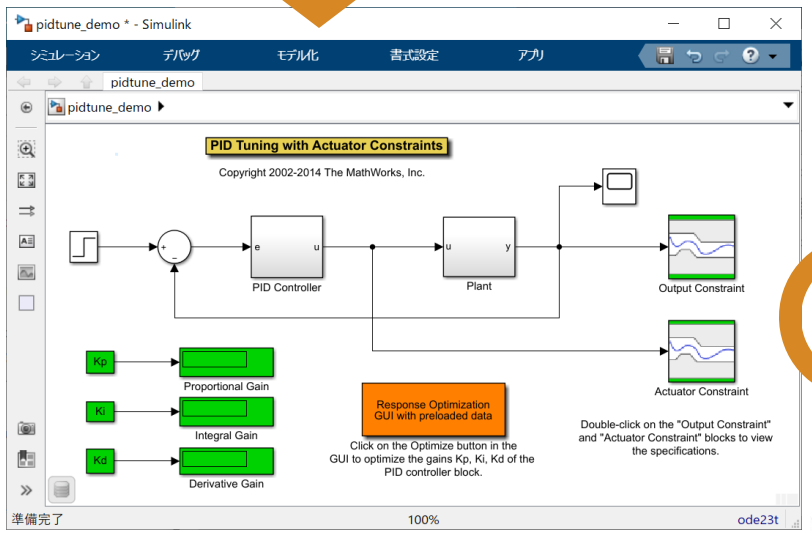

上下限　ステップ応答　目標軌道

周波数応答




ゲイン上限、極配置
安定余裕など

自動調整




## 使いどころ

- 制御性能仕様決め
- 制御性能仕様を満足する制御定数の適合

高速リスタート、並列化にも対応

数値最適化技術

Optimization Toolbox

# PID自動チューニング(Autotuner) vs 応答オプティマイザー

| 特徴 | PID自動チューニング(Autotuner) | 応答オプティマイザー |
|---|---|---|
| 調整対象 | PID制御器のゲイン | 任意のゲイン<br>制御器の構造は自由に構成することができる |
| ゲインスケジューリング設計 | 各操作点におけるPIDゲインを算出することで、ルックアップテーブルを構成する | 各操作点を通るテスト動作を行い、ルックアップテーブルを直接チューニングする |
| 設計パラメータ | 帯域幅、位相余裕、摂動の振幅 | 指定した最適化アルゴリズムのパラメータ |
| 指定可能な要件 | 帯域幅、位相余裕 | 様々な要件を指定可能 |
| 実装 | コード生成して実装可能<br>Simulink Coder、Embedded Coder、Simulink PLC Coderに対応 | コード生成は不可 |

# 目次

- 倒立振子を強化学習で倒立させる制御の設計、及び実機実装
- 強化学習とモデル予測制御を用いて、往来する歩行者を避けながら自律運転する制御を設計
- 地に足付けた制御設計機能
- 役に立つコンテンツ集

# Simulinkによる制御設計入門

- R2020bにて、制御設計ツールの無料チュートリアルがリリースされた


https://jp.mathworks.com/learn/tutorials/control-design-onramp-with-simulink.html

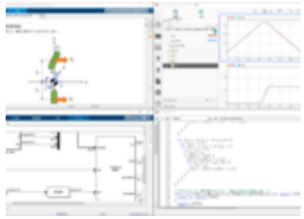
# File Exchange：　モデル予測制御　設計実装ワークフロー紹介

https://jp.mathworks.com/matlabcentral/fileexchange/77879-mpc-implementation-example

MathWorks® Products Solutions Academia Support Community Events Get MATLAB

File Exchange　Search File Exchange　File Exchange

MATLAB Central | Files | Authors | My File Exchange | Contribute | About



## モデル予測制御　設計実装ワークフロー紹介, MPC Implementation Example

version 2.2 (14.8 MB) by Toshinobu Shintai STAFF

当サンプルモデルは、モデル予測制御（MPC）の設計と実装のワークフローを分かりやすく紹介するための資料です。 設計後、コード生成を行い、マイクロコントローラに実装するまでの流れを詳しくまとめています。
https://github.com/mathworks/mpc_implementation_example

Trial software

★★★★★ 1 Rating

17 Downloads

Updated 22 Dec 2020

From GitHub

View Version History

View license on GitHub

+Follow　Download

Overview | Functions | Models | Examples

ダウンロード後は、以下のファイルの使い方を参照してください。
https://github.com/mathworks/mpc_implementation_example/blob/master/MPC_imple_PJ_%E8%AA%AC%E6%98%8E%E8%B3%87%E6%96%99.pdf

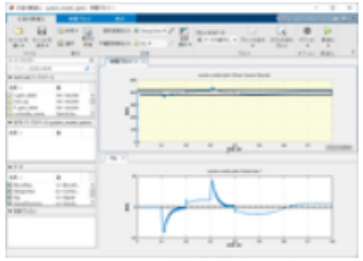
ライブスクリプトを主体として解説をしています。以下リンク先では、ライブスクリプトをMarkdownにしたものをアップロードしていますので、Webブラウザで内容を閲覧することができます。
https://github.com/mathworks/mpc_implementation_example/blob/master/MPC_imple_PJ/common/MPC_Design_index_md.md

- 線形 MPC
- 陽的 MPC
- 適応 MPC
- ゲインスケジュール MPC
- 非線形 MPC

MPC Toolbox による設計ワークフローを一通り体験できる無料サンプルです

# PIDゲインスケジューリング設計のサンプルモデル

非線形なプラントモデルに対してPID Autotunerを適用する応用例を紹介しています。



## PID ゲインスケジューリング設計

version 1.0 (2.25 MB) by Toshinobu Shintai STAFF

非線形なプラントモデルを制御するPID制御器のゲインスケジューリングを、「Closed-Loop PID Autotuner」ブロックまたは「応答の最適化」アプリを用いて設計します。それぞれの手法のメリット、デメリットを把握することができます。
https://github.com/mathworks/pid_autotuing_response_optimization

Overview | Functions | Models | Examples

最初に"PIDゲインスケジューリング設計.pdf"を確認してください。以下のリンク先からも確認できます。
https://github.com/mathworks/pid_autotuing_response_optimization/blob/master/PID%E3%82%B2%E3%82%A4%E3%83%B3%E3%82%B9%E3%82%B1%E3%82%B8%E3%83%A5%E3%83%BC%E3%83%AA%E3%83%B3%E3%82%B0%E8%A8%AD%E8%A8%88.pdf

ライブスクリプトをPDF化したファイルは以下から参照できます。
https://github.com/mathworks/pid_autotuing_response_optimization/tree/master/PDF

[更新履歴]
v1.0 新規作成

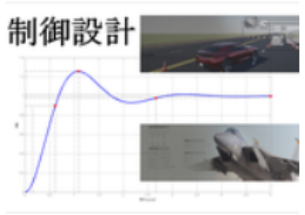
Requires
MATLAB
Simulink
Control Sys
Global Opt
Optimizatic
Parallel Co
Simscape
Simulink C

https://jp.mathworks.com/matlabcentral/fileexchange/79562-pid

# 制御工学ライブ教科書

ライブスクリプトとSimulinkモデルで構成された教科書で、実践的な制御設計を学ぶことができます

MATLAB Central | Files | Authors | My File Exchange | Contribute | About




## 実践的な制御設計とその理論

version 1.0.2 (2.05 MB) by Toshinobu Shintai STAFF

【Learn Control Design and Control Theory with Application Examples】 ライブスクリプトとSimulinkモデルを使って、PID制御を始めとした制御器の実践的な設計方法について学ぶことができます。
https://github.com/mathworks/control_design_and_control_theory

Overview | Functions | Models | Examples

PID制御を始めとした制御器の実践的な設計方法について学ぶことができます。ライブスクリプトとSimulinkモデルを用いて実際にモデルを動かしながら読むことで、理解を深めることができるコンテンツです。

コンテンツの使い方については、以下のReadMeを参照してください。
https://github.com/mathworks/control_design_and_control_theory/blob/master/ReadMe.md

[目次]
はじめに
UAVのホバリング Part 1: PID制御とは
UAVのホバリング Part 2: 積分器、微分器の拡張
UAVのホバリング Part 3: ゲインチューニング

**Requires**
MATLAB
Simulink
Control System Toolbox
Simulink Control Design
Stateflow

**MATLAB Release Comp**
Created with R2021a
Compatible with R2021a

**モデルファイル**：https://www.mathworks.com/matlabcentral/fileexchange/90042
**紹介動画**：https://www.youtube.com/watch?v=1s7zM-q7Wjc

# 例題サンプル集： Simulink Control Design, Simulink Design Optimization

| アプリ/機能名 | 用途 | 例題 |
|---|---|---|
| 定常状態マネージャー | 解析<br>（平衡点/動作点） | エンジンの例：<br>定常状態マネージャーを使用した仕様からの操作点の計算 |
| モデル線形化器 | 解析<br>（応答特性） | タンク水位制御の例：<br>Simulink モデルのモデル操作点での線形化 |
| PID調整器 | 制御器の自動調整 | エンジン速度制御の例：<br>Simulink での PID コントローラーの調整 |
| 制御システム調整器 | | ヘリコプター姿勢制御の例：<br>制御システム調整器を使用した制御システムの調整 |
| PID Autotuner | | 昇圧型コンバータ電圧制御の例：<br>Closed-Loop PID Autotuner ブロックを使用した PID コントローラーのリアルタイム調整 |
| 応答オプティマイザー<br>（応答最適化ツール） | | PID制御の例：<br>PID Tuning with Actuator Constraints |
| パラメータ推定器<br>（パラメータ推定ツール） | パラメータ同定 | エンジンスロットルの例：<br>Estimate Model Parameter Values (GUI) |
| モデル・リデューサー | モデル低次元化 | ビルの振動モデルの例：<br>Model Reducer アプリを使用したモデル次数の低次元化 |

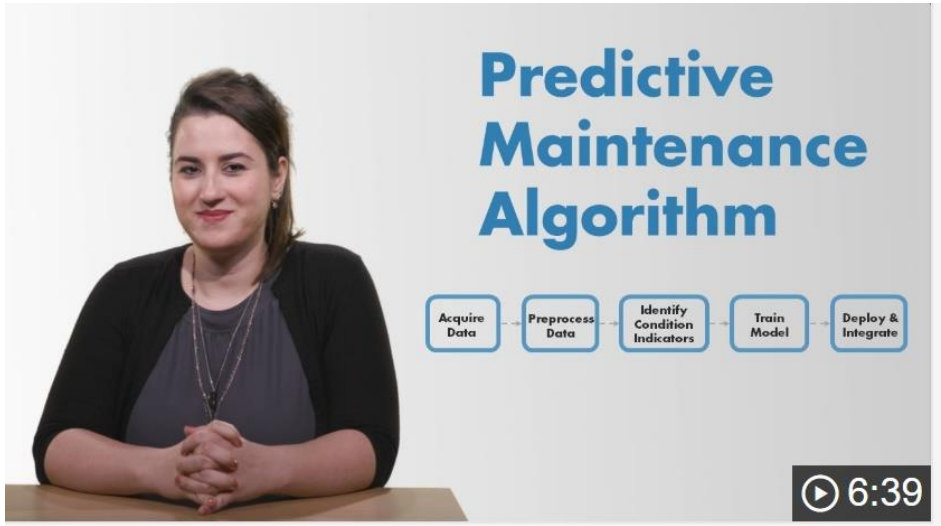
※上記はすべて各製品に付属する例題となります。

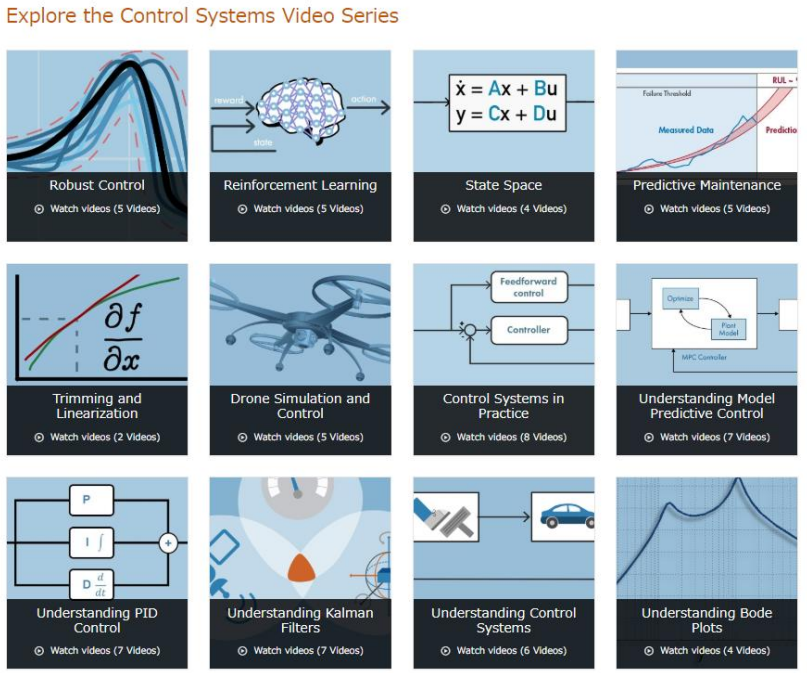
# MATLAB Tech Talks

https://www.mathworks.com/videos/tech-talks/controls.html

フィードバック制御システムの解析・設計に必要な基本概念を
テーマごとに動画で分かりやすく解説（英語）

| テーマ | シリーズ動画 |
|---|---|
| 制御系の基礎 | Understanding Control Systems |
| PID 制御 | Understanding PID Control |
| 制御系の実践 | Control Systems in Practice |
| ボード線図 | Understanding Bode Plots<br>Using Bode Plots |
| 状態空間 | State Space |
| 動作点と線形化 | Trimming and Linearization |
| カルマンフィルタ | Understanding Kalman Filters |
| モデル予測制御 | Understanding Model Predictive Control |
| ロバスト制御 | Robust Control |
| 強化学習 | Reinforcement Learning |
| 予知保全 | Predictive Maintenance |
| ドローン | Drone Simulation and Control |



# MATLAB/Simulink 関連書籍を検索

https://www.mathworks.com/academia/books.html


**MATLAB 入門 - zyBook**

このウェブベースの書籍では、インタラクティブな質問やアニメーション、自動評価を通して、MATLAB の基礎全般を学べます。

結果： 1 - 25 / 299

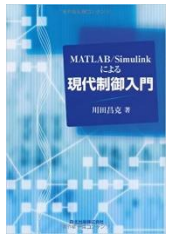
**MATLAB/Simulinkによる制御工学入門**
はじめて制御工学を学ぶ読者に最適な、わかりやすい入門書です。電気系と機械力学系を中心として、例や問題を豊富に収録するとともに、図やグラフを多く用いて、理解しやすいよう解説を工夫されています。
**作成者:** 川田 昌克
**Copyright:** 2020
**言語:** 日本語
**出版社:** 森北出版
**使用可能なコンパニオン ソフトウェア**

**自動車業界MBDエンジニアのためのSimulink入門: 4週間で学ぶSimulink実践トレーニング 第2版**
本書は、自動車業界で新たにモデルベース開発(MBD) を始めるエンジニアにお勧めの Simulink の入門書です。自動車業界の実際の開発業務で使うことを目的に、基礎的な物理法則の復習モデルからトルクコンバーターやクルーズコントロールといった実製品の機能モデルまで Simulink で作成していきます。
**作成者:** 久保 孝行
**Copyright:** 2019
**言語:** 日本語
**出版社:** TechShare
**使用可能なコンパニオン ソフトウェア**

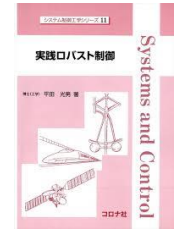
## 分野

- Artificial Intelligence, Machine Learning, and Deep Learning
- 生物科学および生物医学
- 化学および化学工学
- 通信システム
- 制御システム
- Data Science and Statistics
- デジタル信号処理
- 地球科学
- 経済学および金融工学
- エレクトロニクス
- 総合
- 画像および動画処理
- 数学
  - 微分方程式
  - 線形代数
  - 数値的手法
- 機械工学
- ニューラル ネットワークおよびファジー論理
- 神経科学
- 物理学
- プログラミングとコンピューター サイエンス
- ロボット工学
- システム同定
- システム モデリングとシミュレーション
- 実験、計測
- MATLAB および Simulink の使用

# 川田 昌克 教授　舞鶴工業高等専門学校

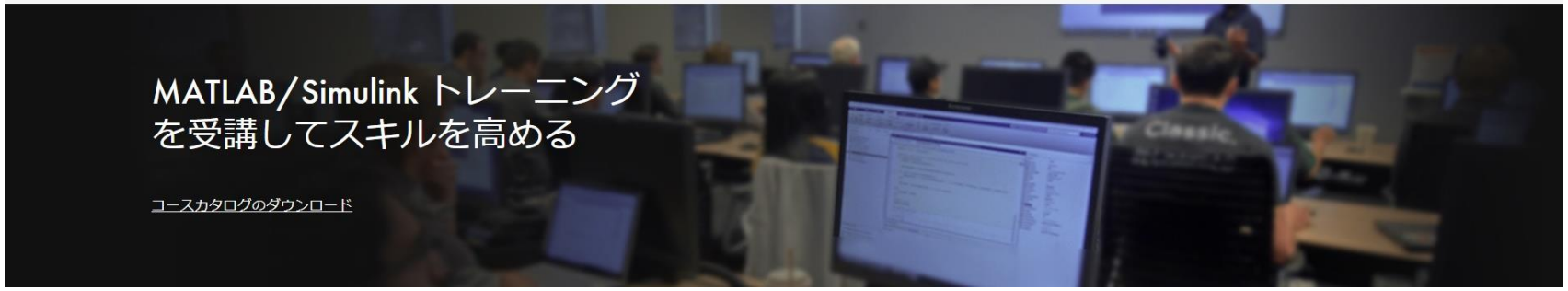
| 分類 | コンテンツ |
|---|---|
| 書籍 | 川田 昌克 ： MATLAB/Simulinkによる制御工学入門，森北出版 (2020)<br>https://www.mathworks.com/academia/books/introduction-to-control-engineering-with-matlab-simulink.html<br><br>内容： はじめて制御工学を学ぶ読者に最適な、わかりやすい入門書です。<br>電気系と機械力学系を中心として、例や問題を豊富に収録するとともに、<br>図やグラフを多く用いて、理解しやすいよう解説を工夫されています。<br>  |
| | 川田 昌克 ： MATLAB/Simulinkによる現代制御入門，森北出版 (2011)<br>https://www.morikita.co.jp/books/book/2396<br><br>内容： 古典制御理論の入門書として評価の高い「わかりやすい制御工学」の続編！<br>前著と同様，例題や演習問題を豊富に盛り込み，理解しやすいように工夫されています。<br>理論を学びながら，MATLAB/Simulinkの使い方もマスターできます。<br>  |
| | 川田 昌克 ： MATLAB/Simulink と実機で学ぶ制御工学 ーPID 制御から現代制御までー，<br>TechShare (2013)<br>https://www.mbd-shop.jp/product/180<br><br>内容： 制御工学を、制御対象のモデリング、制御器の設計、シミュレーション、実機による<br>検証という制御系設計の流れを、シミュレーションと実機実験を通して学習できる書籍です。<br>  |
| 研究室 HP | https://www.maizuru-ct.ac.jp/control/kawata/ |

# 平田 光男 教授　宇都宮大学

| 分類 | コンテンツ |
| --- | --- |
| 書籍 | 平田 光男： 実践ロバスト制御，コロナ社 (2017)<br>https://www.coronasha.co.jp/np/isbn/9784339033113/<br><br>内容： 本書では，与えられた制御問題にいかにして答えを求めるかという実践的な側面から，H∞制御およびμ設計法について，これらの設計法をロバスト制御系設計のツールとして使いこなせるようにすることに主眼を置いて平易に解説します。<br>  |
| | 平田 光男： ArduinoとMATLABで制御系設計をはじめよう!，TechShare (2012)<br>https://www.mathworks.com/academia/books/book102451.html<br><br>内容： 本書は、制御系設計をこれから学ぶ方のための入門書です。Arduino と MATLAB /Simulink を使い制御実験教材を作り、実際に動かしながら制御理論を学んでいきます。さらに古典制御から現代制御まで一通りの制御理論の内容が盛り込まれており、制御理論の基礎の習得とモデルベース開発を実体験できる書籍です。<br>  |
| MATLAB EXPO 公開スライド | 実例から学ぶ！ モデルを活用したモータ制御系開発<br>https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/images/events/matlabexpo/jp/2016/g1-motor-control-system-development.pdf |
| 研究室 HP | http://hinf.ee.utsunomiya-u.ac.jp/ |

# トレーニング：　無料オンラインコースもご用意

https://matlabacademy.mathworks.com/jp




### 自己学習形式

解説ビデオの視聴、演習および確認テストをご自身のペースで進めることができます

### クラスルーム形式

弊社施設（東京・名古屋・大阪）のクラスルームでご受講いただけます

### ライブ形式

お客様のPCからオンラインでご受講いただけます

### オンサイト形式

カスタマイズされたカリキュラムをお客様の施設でご受講いただけます

**人気コースの例**

- Simulink 入門
- MATLAB と Simulink による制御設計
- Simulink モデルの検証と妥当性確認
- Simulink Real-Time と Speedgoat ハードウェア による リアルタイムテスト
- Simulink への外部コードの取り込み


Accelerating the pace of engineering and science