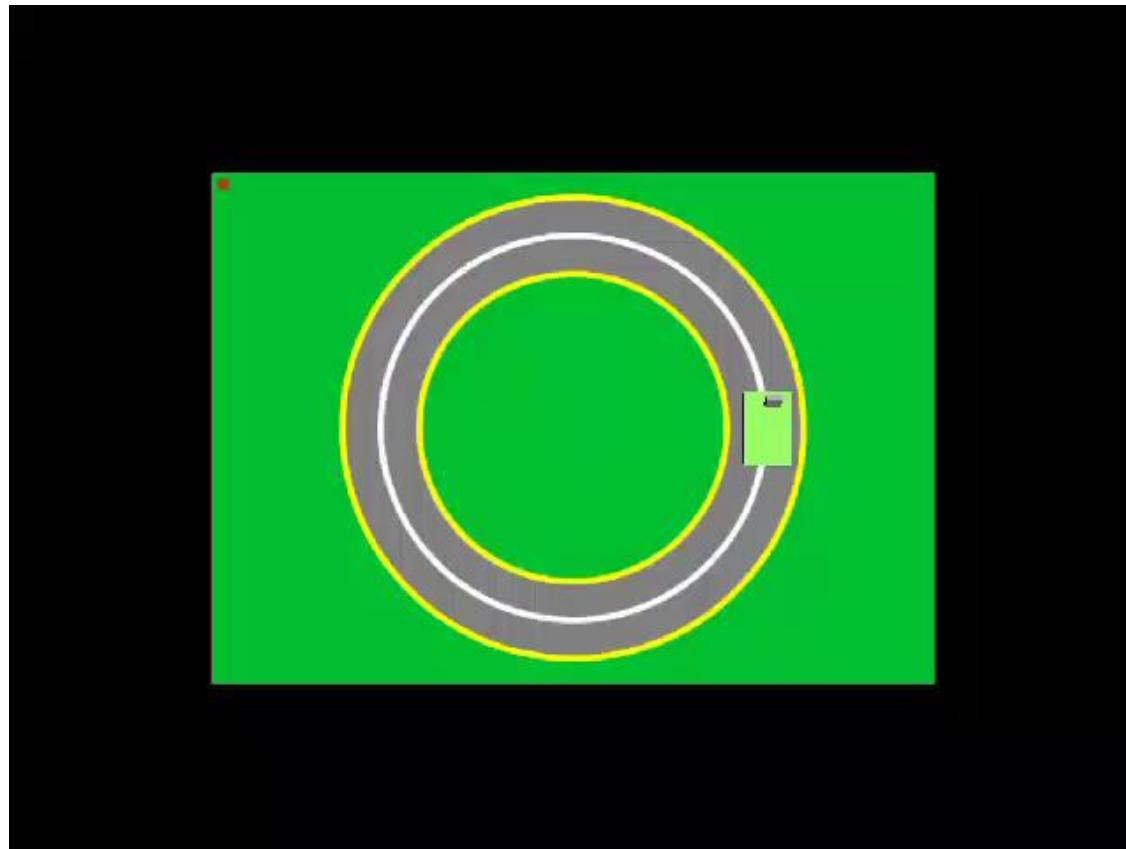


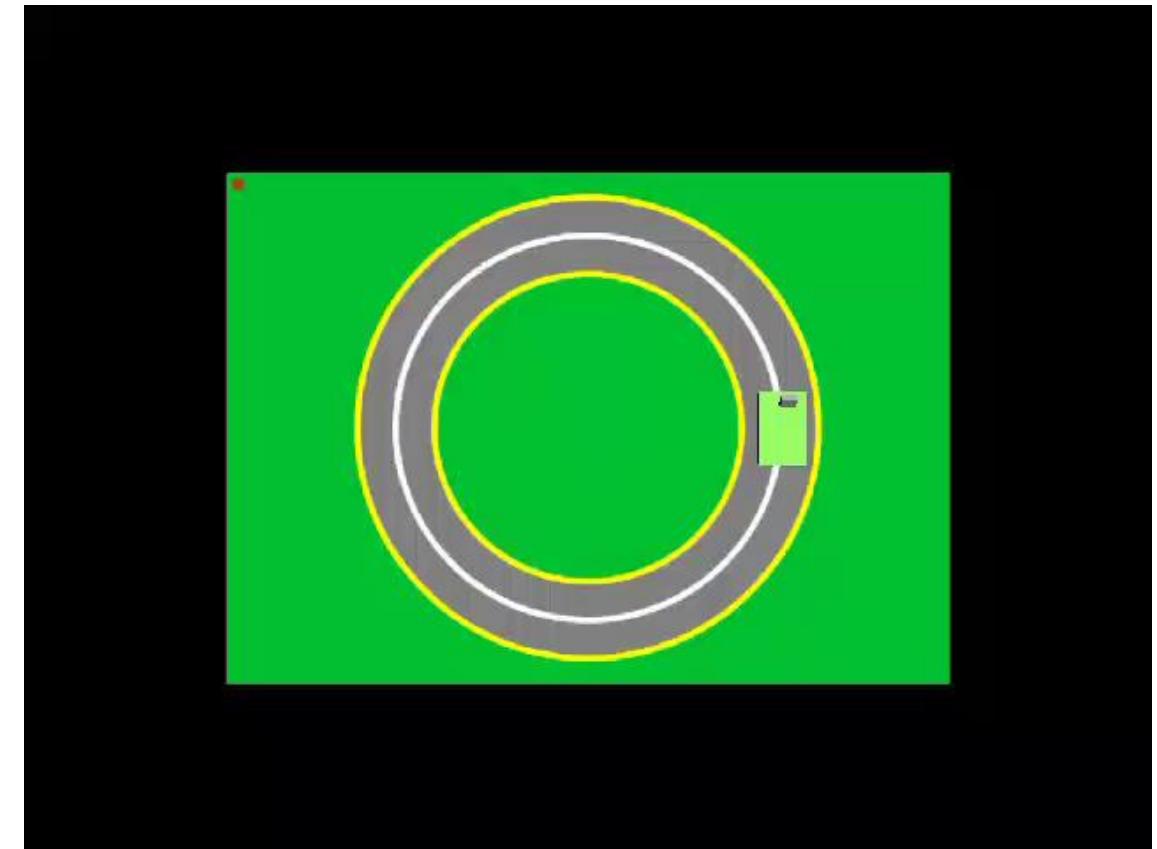
MATLAB®によるAIロボティクスワークショップ ～ディープラーニングによるライントレースロボット～

MathWorks

ライントレースロボットのデモ



失敗例
(脱線してしまう)



成功例
(ラインに沿って移動)

カメラ画像でライントレースロボットを実現しよう

目的：ライントレースロボット開発を通じて知能ロボットシステムの概念や勘所を習得する

- ライトトレースとは？

- ロボットが床面に描かれたラインをセンサで読み取り、ラインに沿って走行すること

ラインに沿って走行させる

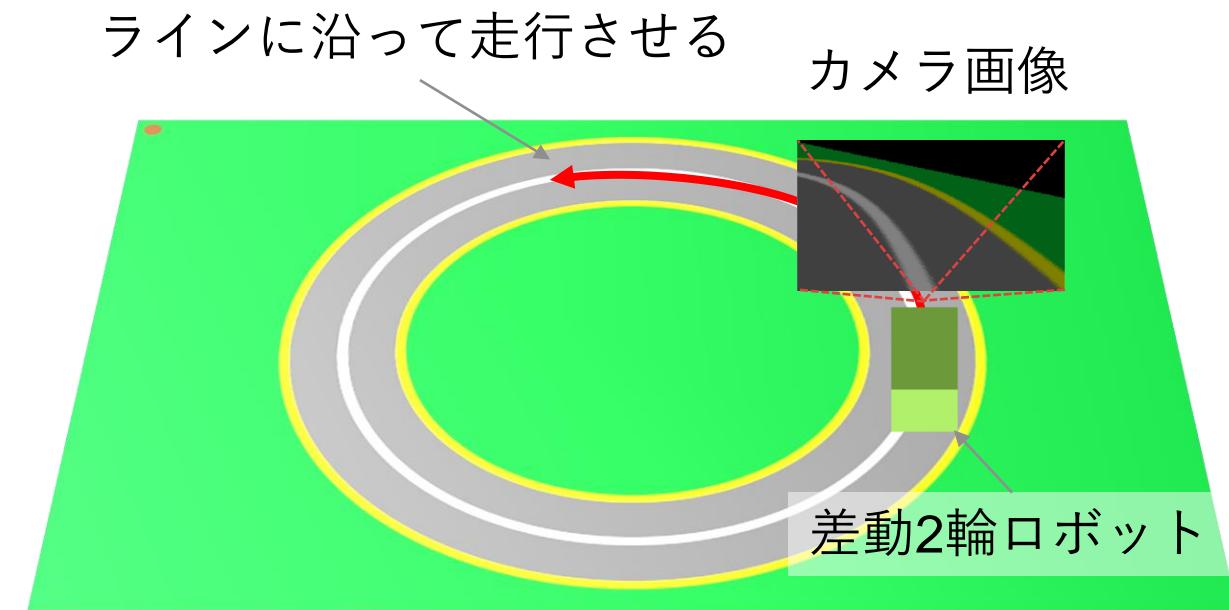
カメラ画像

- ライントレースのチャレンジ

- ルールベースの画像処理アルゴリズムでは明るさなど環境変化に対してロバストなライン検出が難しい

- ディープラーニングによる
ライントレースの特徴

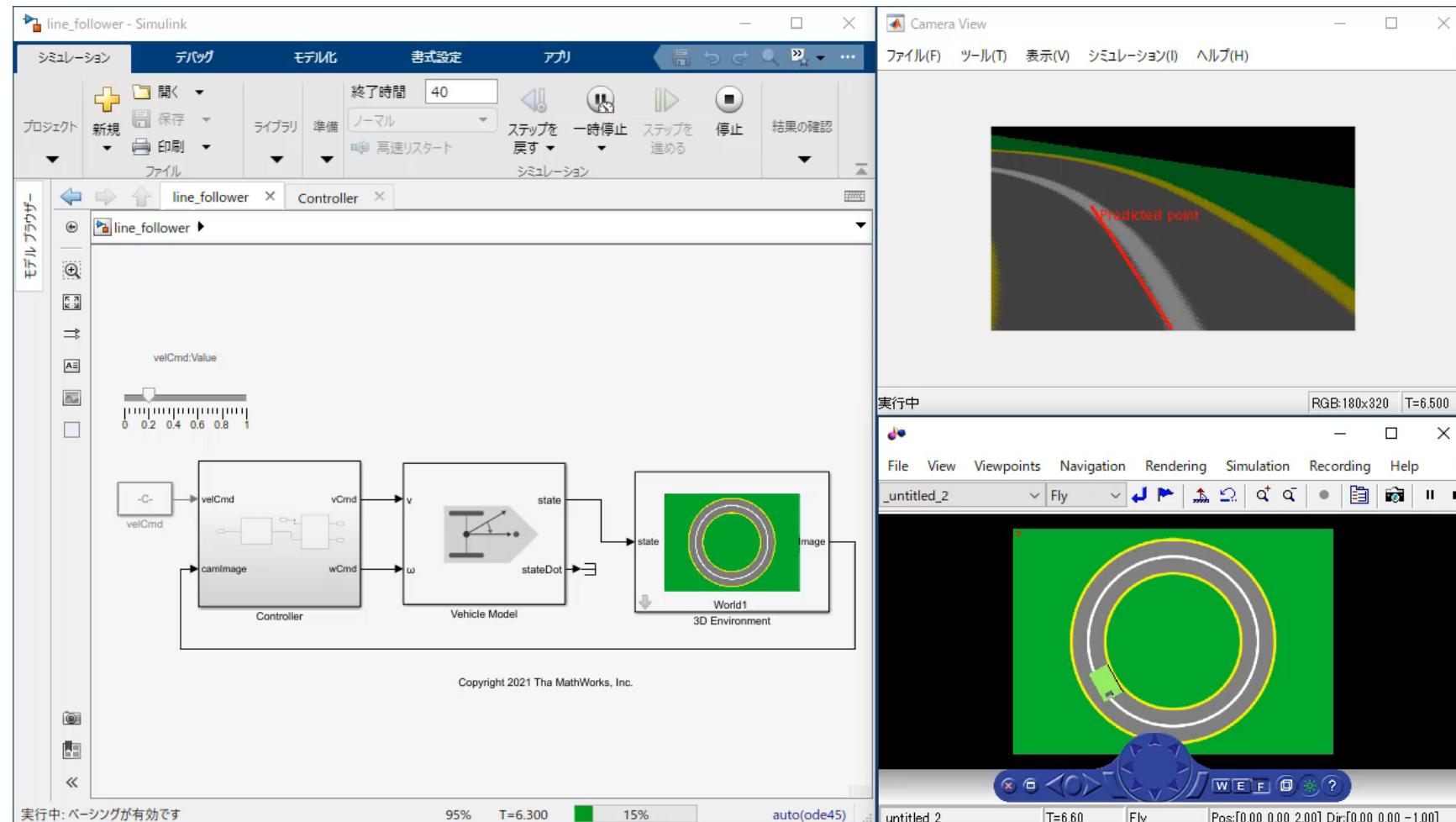
- 環境変化に対して柔軟にライン検出が可能
 - 一方、学習画像の収集やアノテーション、学習パラメータの最適化が必要



タスクの概要

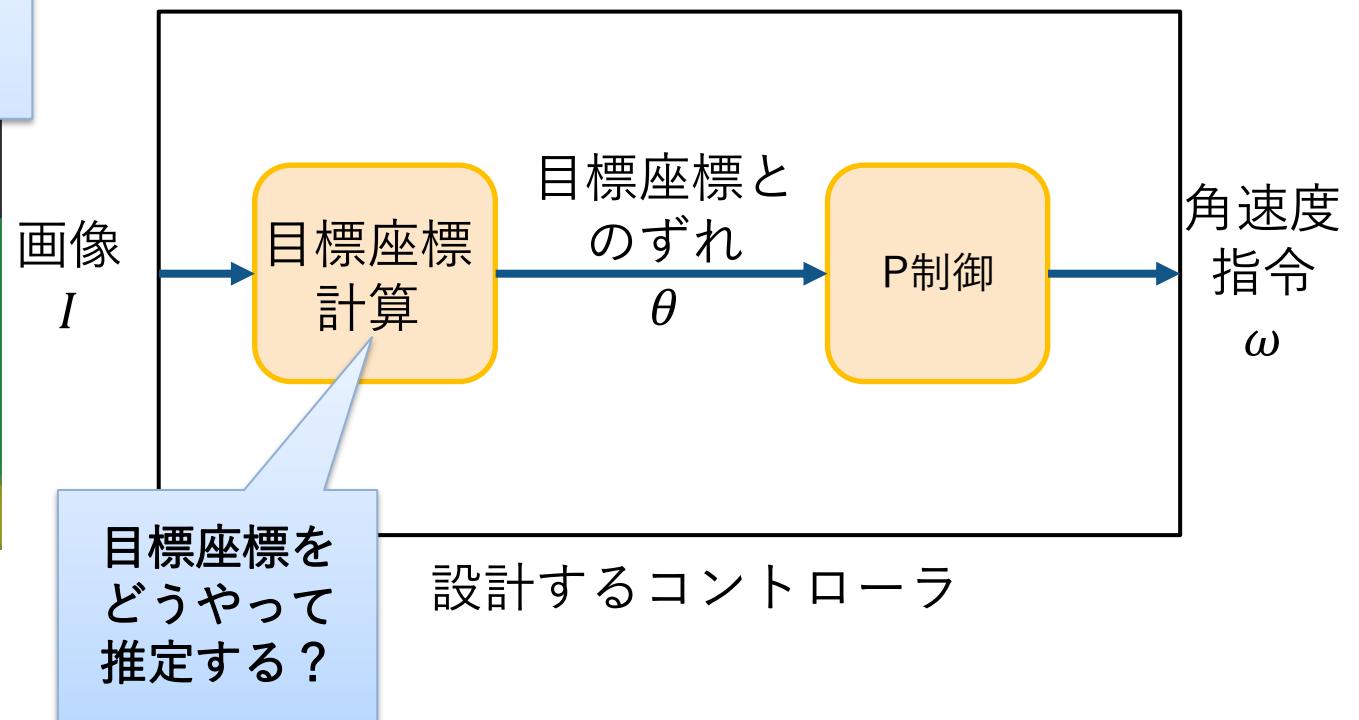
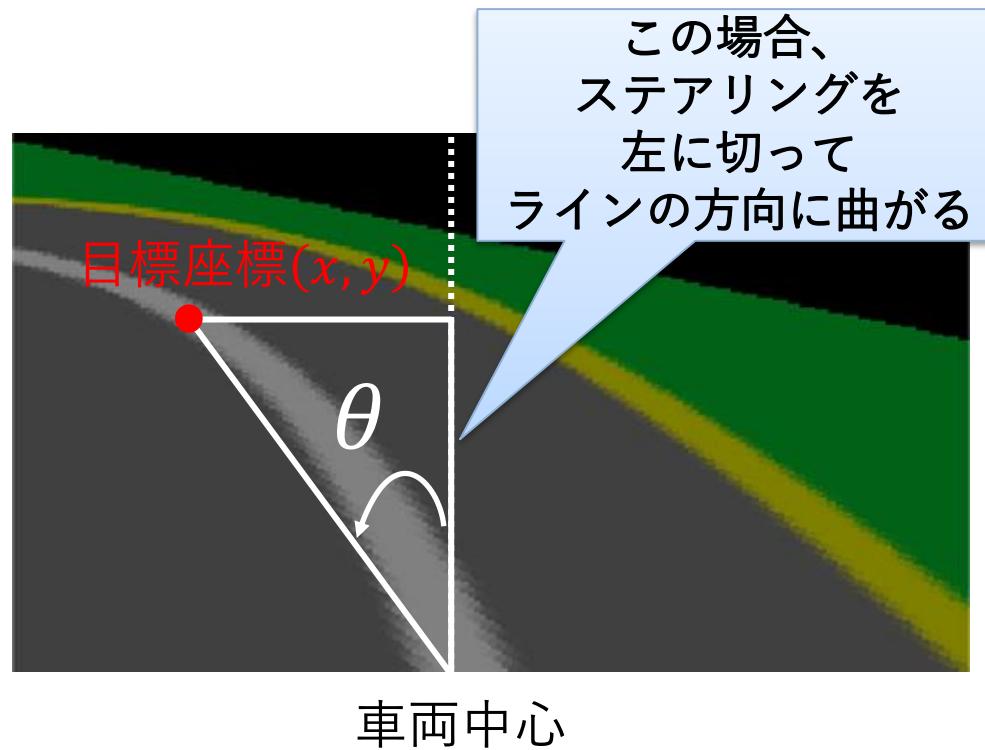
ディープラーニングによるライントレースの実現例

ゴール：カメラ画像のみを用いてライントレースを実現する

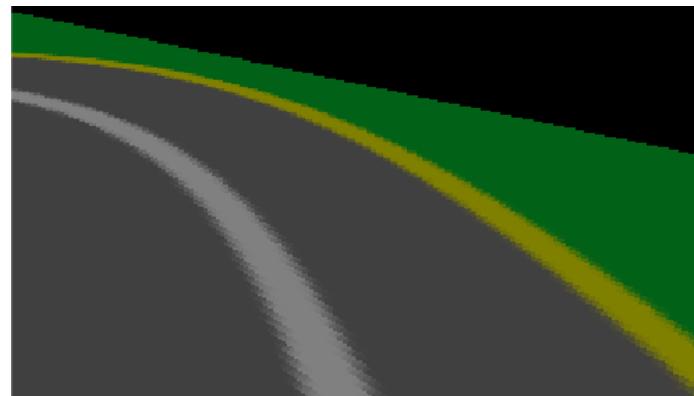


画像によるライントレースの考え方

- 車両中心とライン上の目標座標のずれ量 θ を検出
- ずれ量 θ がゼロになるようにステアリングを切る



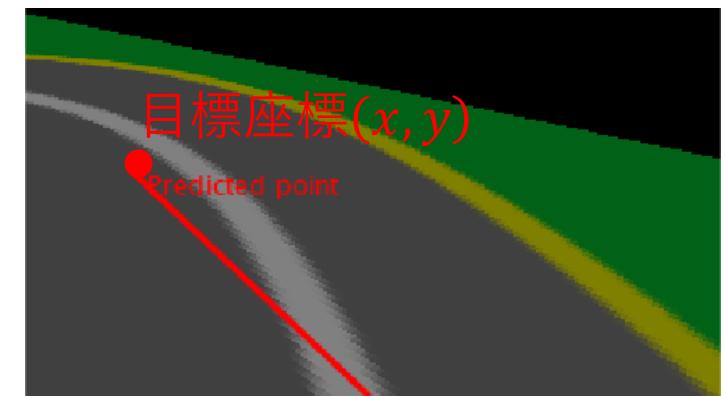
ディープラーニングによって目標座標を予測



カメラからの画像データ
($180 \times 320 \times 3$)



ディープラーニング



目標座標(x, y)
(1×2)

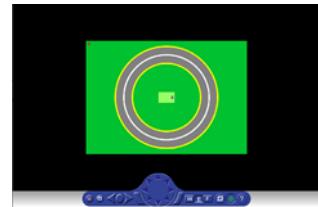
目標座標をディープラーニングモデルを使用して予測

なぜディープラーニングを使うか？

高次元の画像データから低次元の目標座標を出力するため
ディープラーニングによるモデル化が最適

アジェンダ：自律ロボットシステム開発ワークフロー

カメラのシミュレーション
で学習画像生成



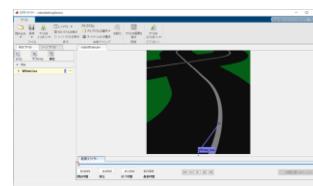
データセットの
準備



ラベリング

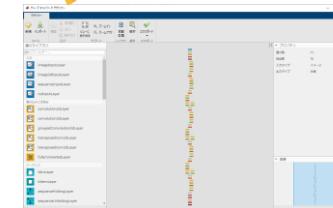


目標位置と
学習用画像



ビデオラベラー

回帰ネットワークモデル



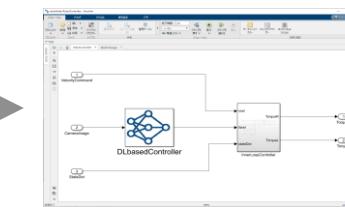
ネットワークの設計



学習

学習済モデル

統合



システム
シミュレーション



実機実行

ワークショップの準備

1. "MATLAB R2021a" を起動



2. デモファイルのあるフォルダに移動

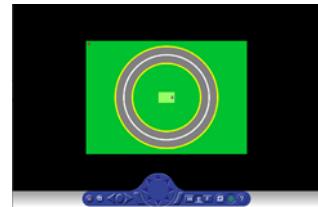
```
>> cd c:\ai-robotics-workshop
```

3. プロジェクトファイルを開く

```
>> ai_robotics_workshop.prj
```

アジェンダ：自律ロボットシステム開発ワークフロー

カメラのシミュレーション
で学習画像生成



データセットの
準備

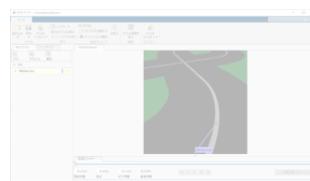


ラベリング



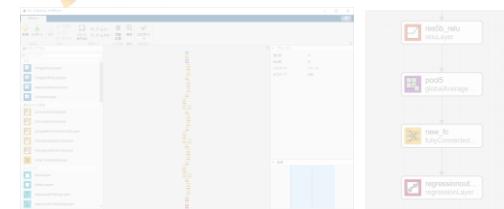
学習

目標位置と
学習用画像



ビデオラベラー

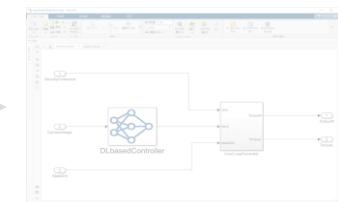
回帰ネットワークモデル



ネットワークの設計



学習済モデル



統合



システム
シミュレーション



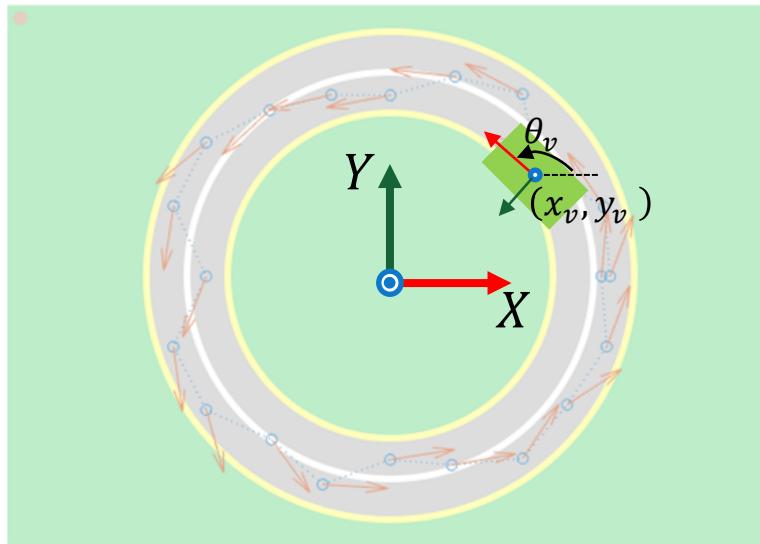
実装

実機実行

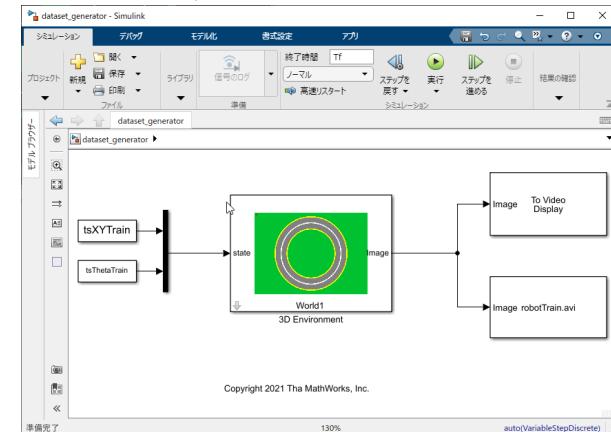
学習用の動画を生成

- ディープラーニングの学習には様々な角度で撮影した画像が大量に必要
- 車両位置や姿勢をランダムに生成し、学習用の動画ファイルを作る

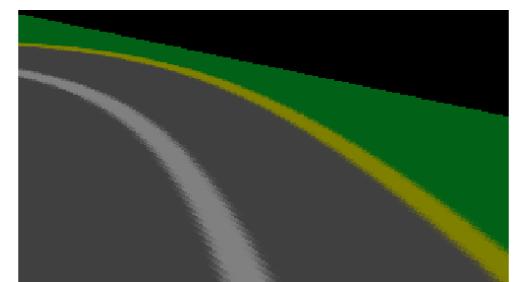
車両位置・姿勢 (x_v, y_v, θ_v) を
ランダムに生成



それぞれの位置で
カメラ画像を
シミュレーション



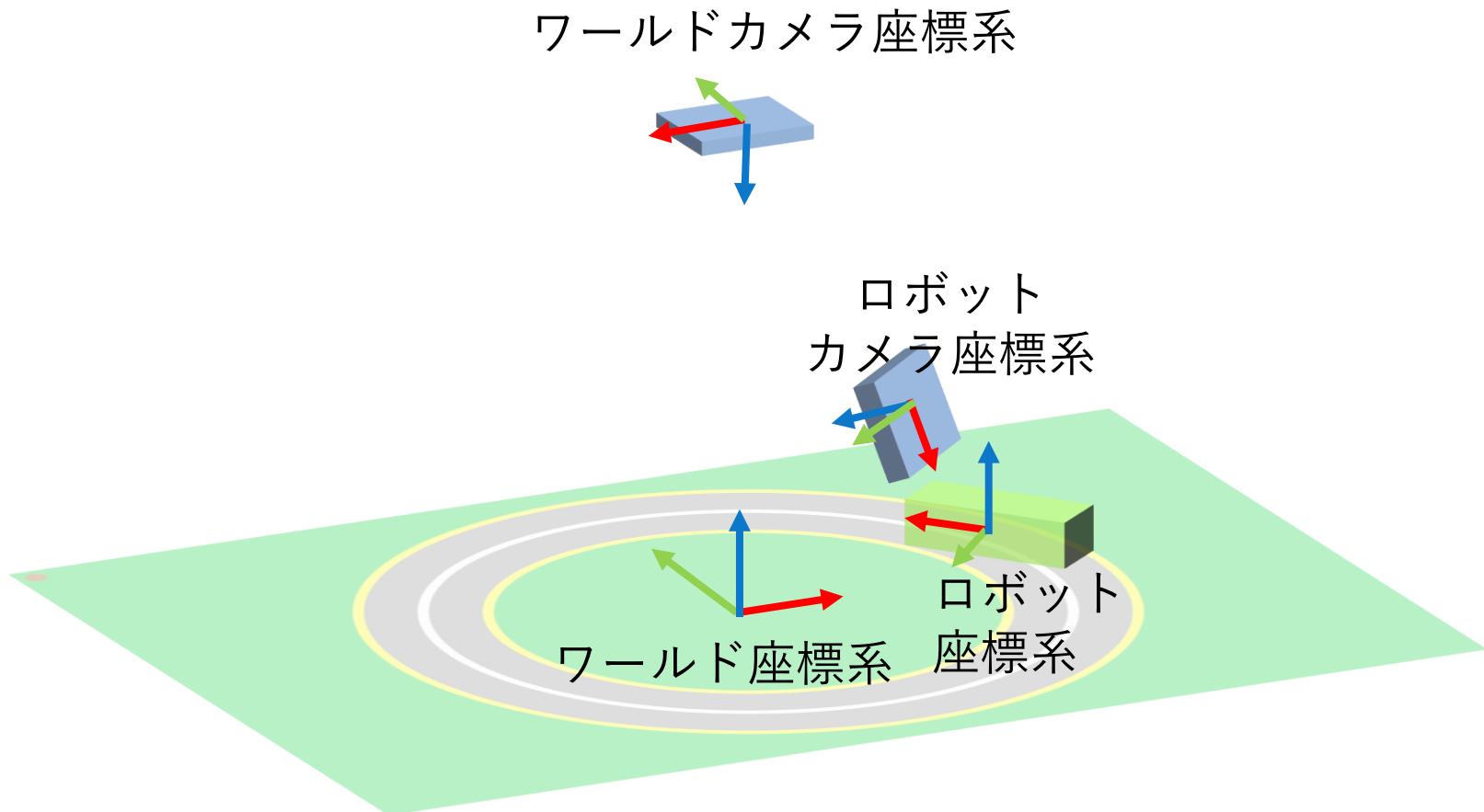
動画ファイルとして生成



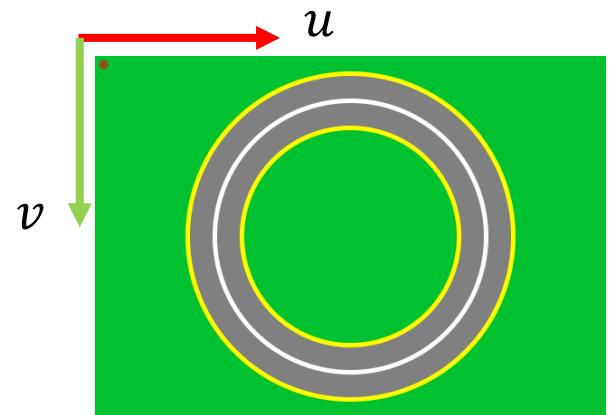
robotTrain.avi

補足：座標系

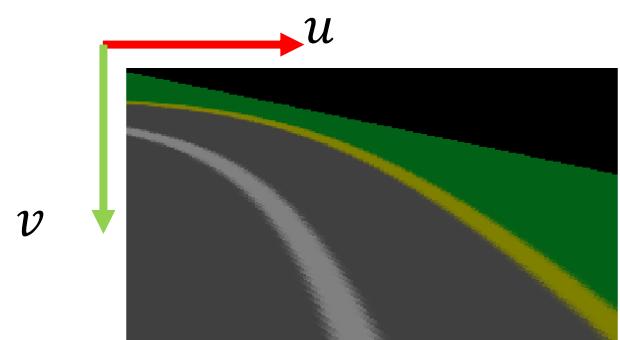
3次元座標系



画像座標系



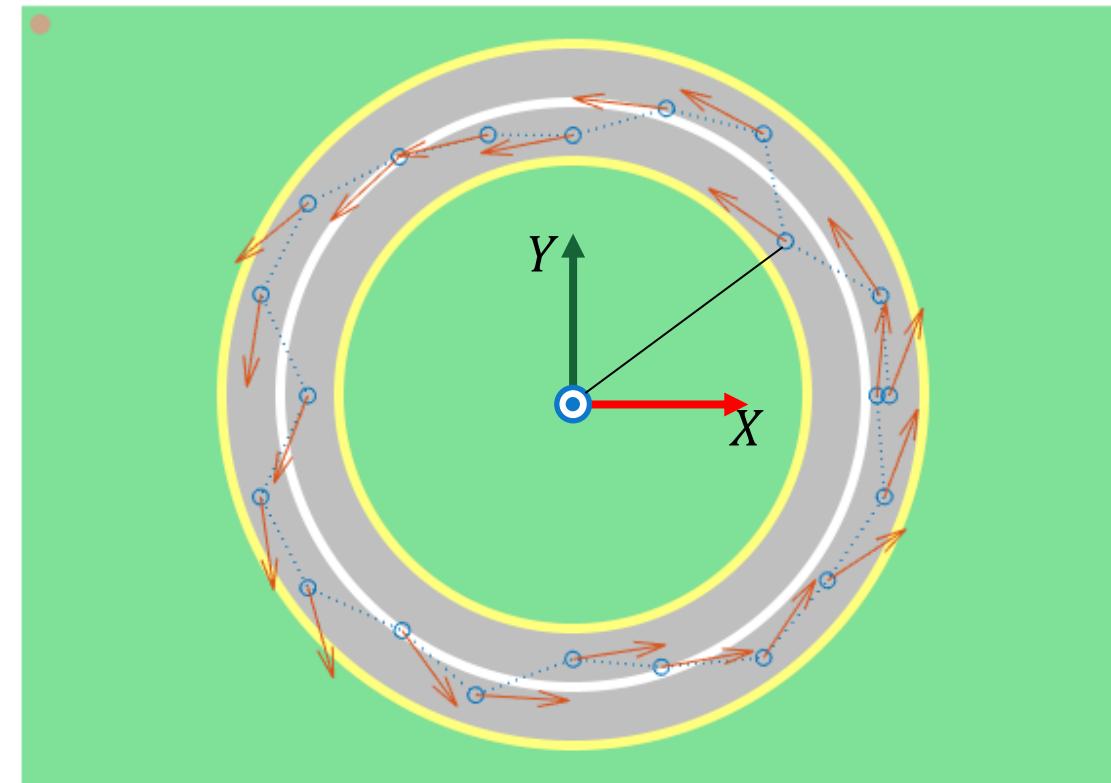
ワールドカメラ
画像座標系



ロボットカメラ
画像座標系

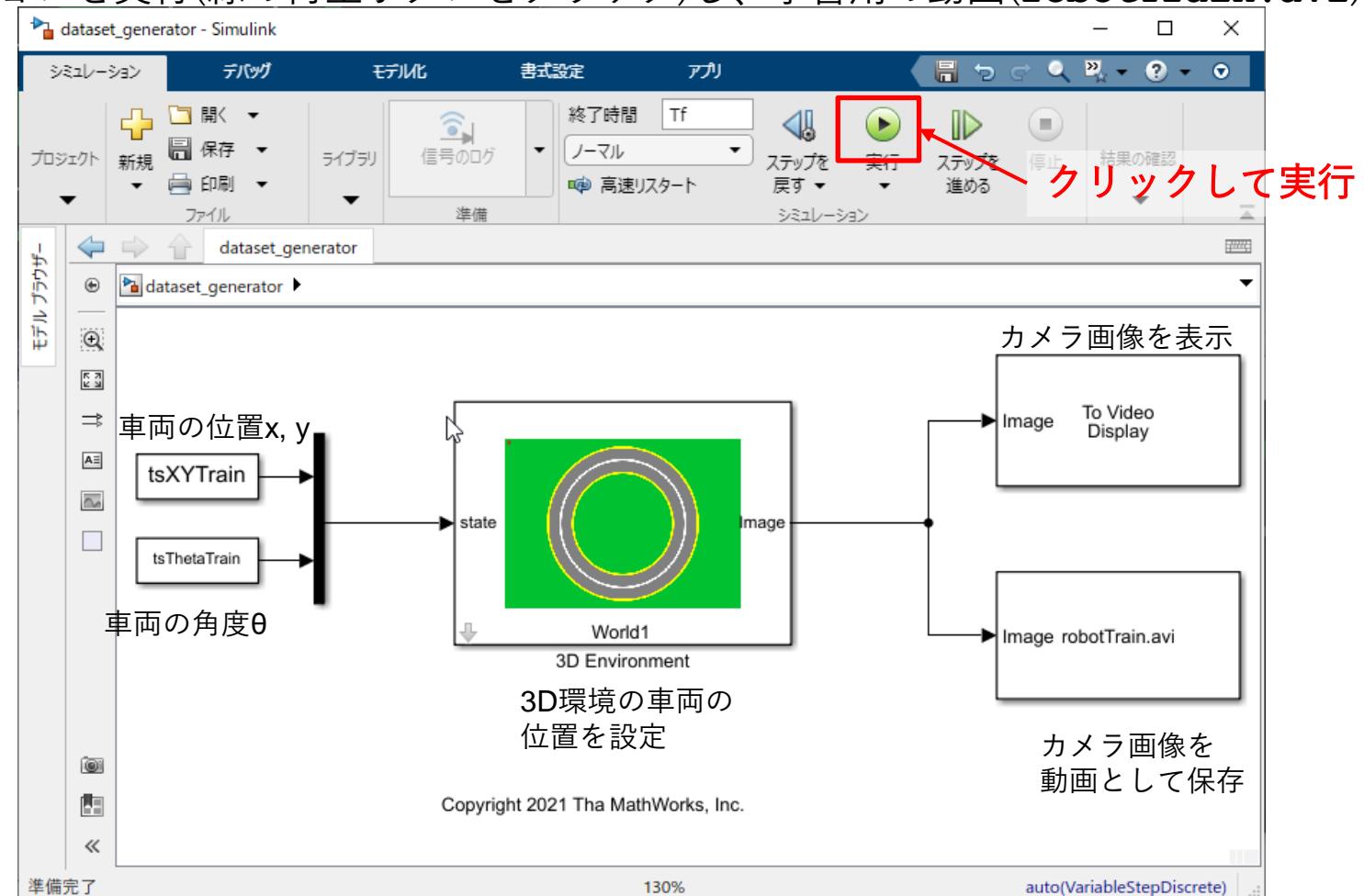
演習1. 学習用の動画を生成

1. MATLABコードを開いて実行し、ランダムに車両の位置や姿勢を生成する
`>> edit genCourseTraj_jp.mlx`



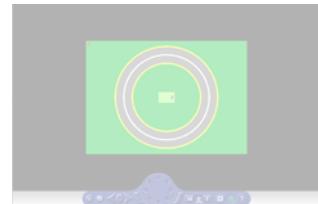
演習1. 学習用の動画を生成

1. 生成した位置や姿勢でカメラ画像をシミュレーションするモデルを開く
`>> dataset_generator.slx`
2. シミュレーションを実行(緑の再生ボタンをクリック)し、学習用の動画(`robotTrain.avi`)を生成



アジェンダ：自律ロボットシステム開発ワークフロー

カメラのシミュレーション
で学習画像生成



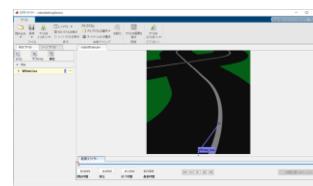
データセットの
準備



ラベリング



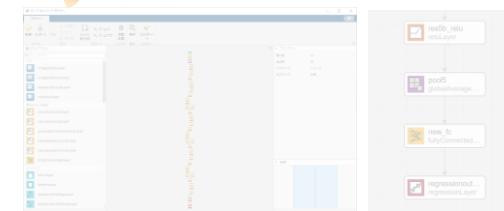
目標位置と
学習用画像



学習用画像

ビデオラベラー

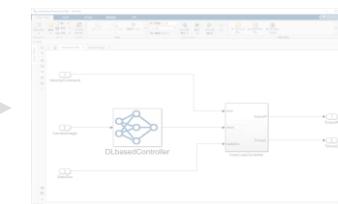
回帰ネットワークモデル



ネットワークの設計



学習済モデル



統合

システム
シミュレーション

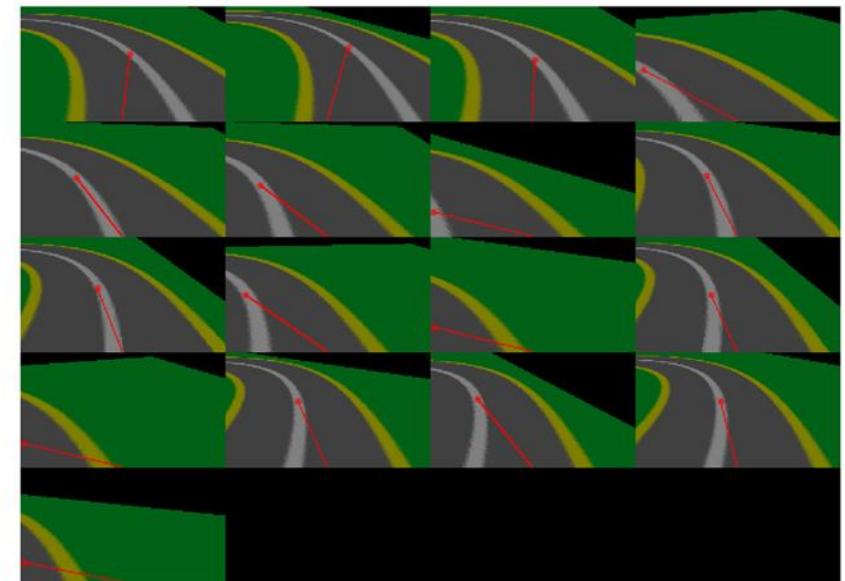
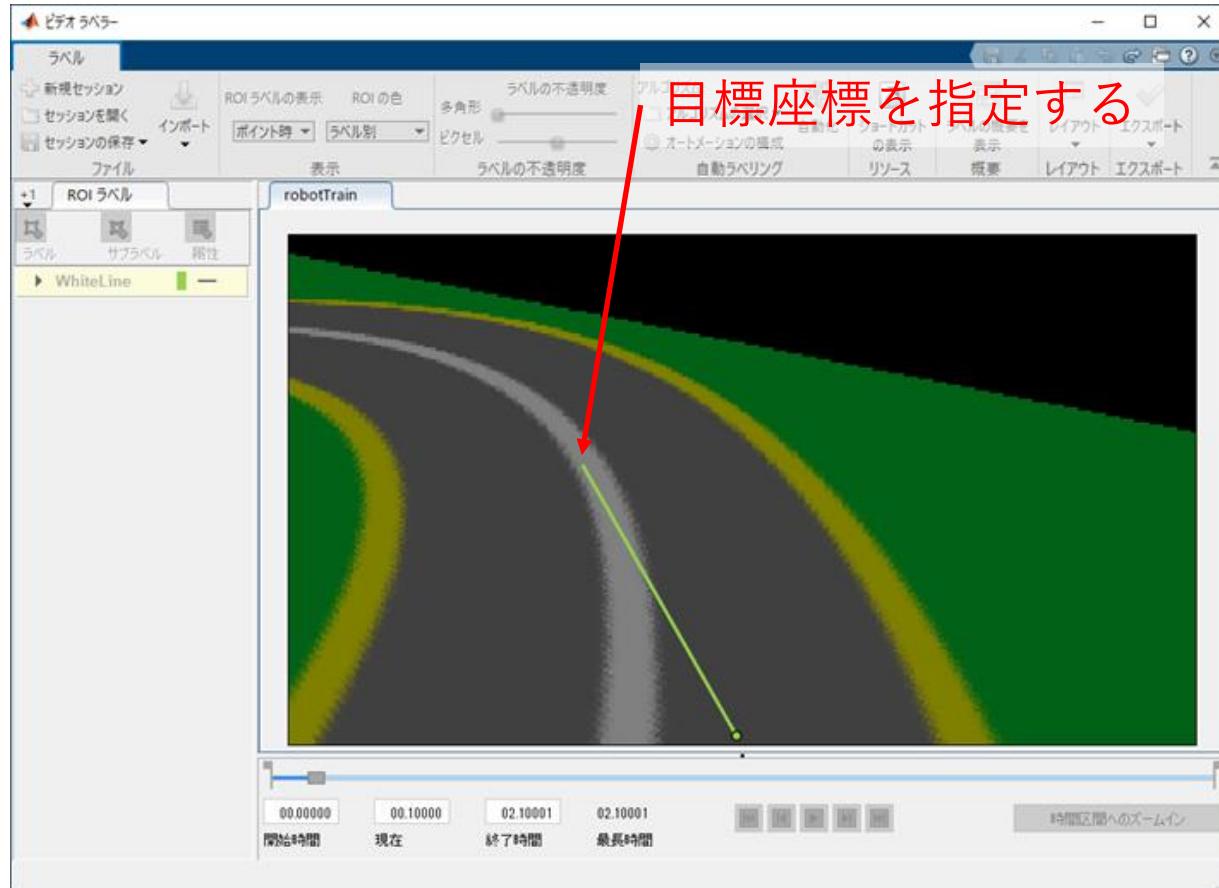


実装

実機実行

動画のラベリング(アノテーション)

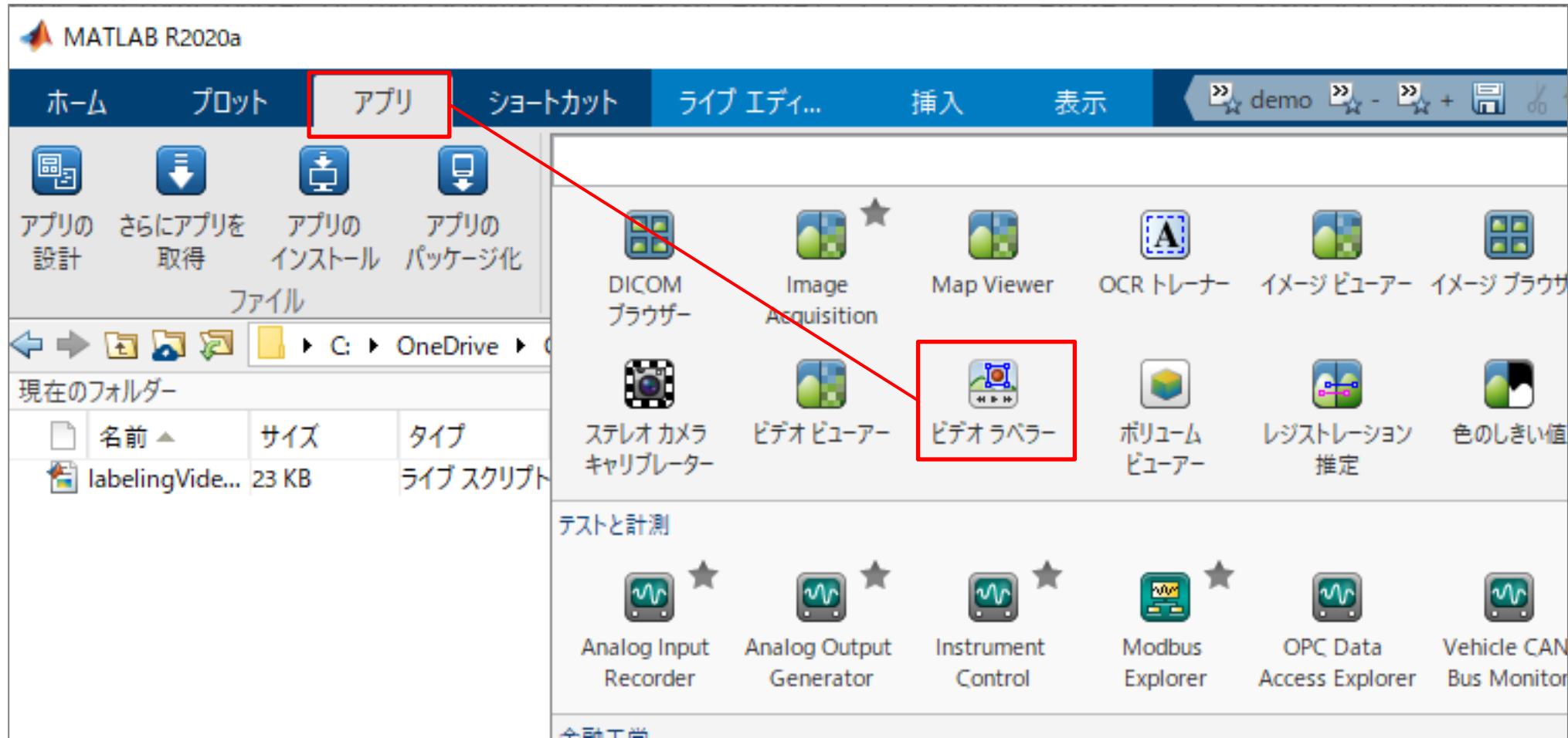
- ビデオラベラーで各フレームごとに「目標座標」のラベル付けし
学習用のデータセット(教師データ)を作成する



学習用のデータセット
(画像と座標がセットになったもの)

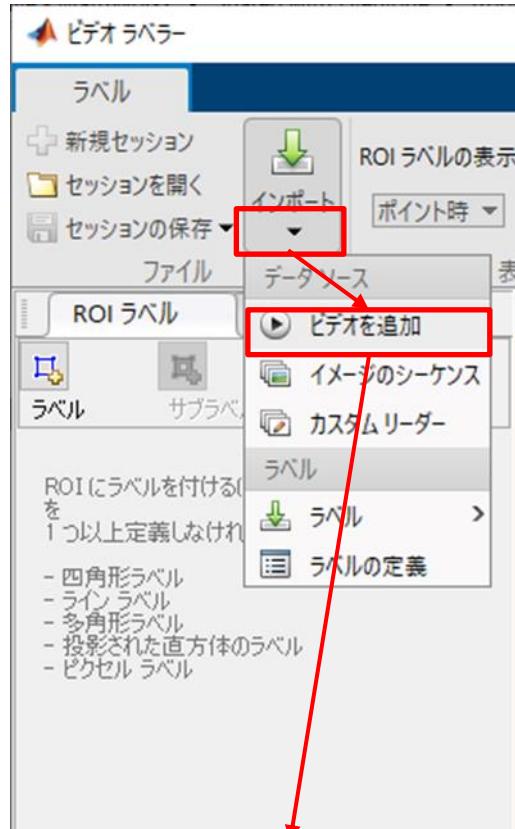
演習2. ビデオラベラーを使ってラベリング

1. ビデオラベラーを起動する



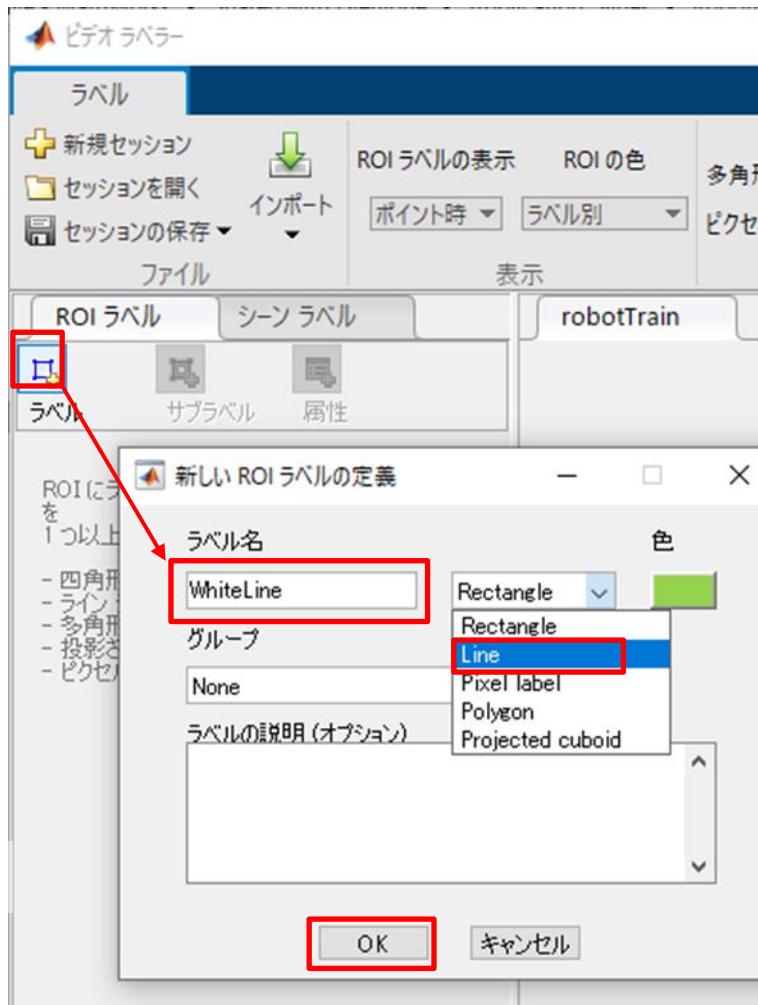
演習2. ビデオラベラーを使ってラベリング

2. 動画を読み込む

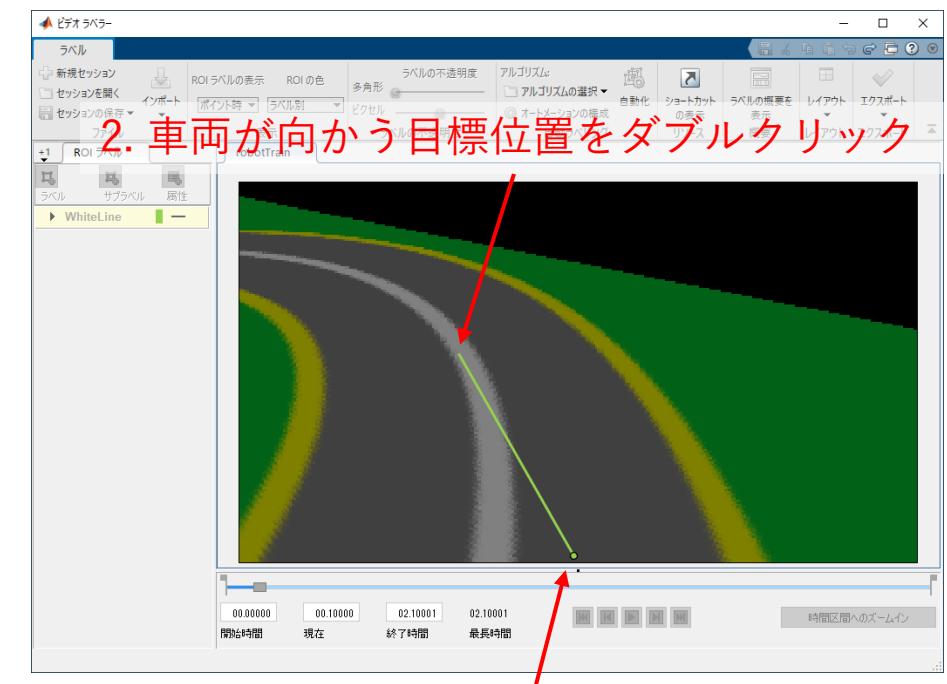


`robotTrain.avi`を開く

3. ラベルを定義する



4. 画像下から上に向かって目標位置の を20枚程度ラベリングする

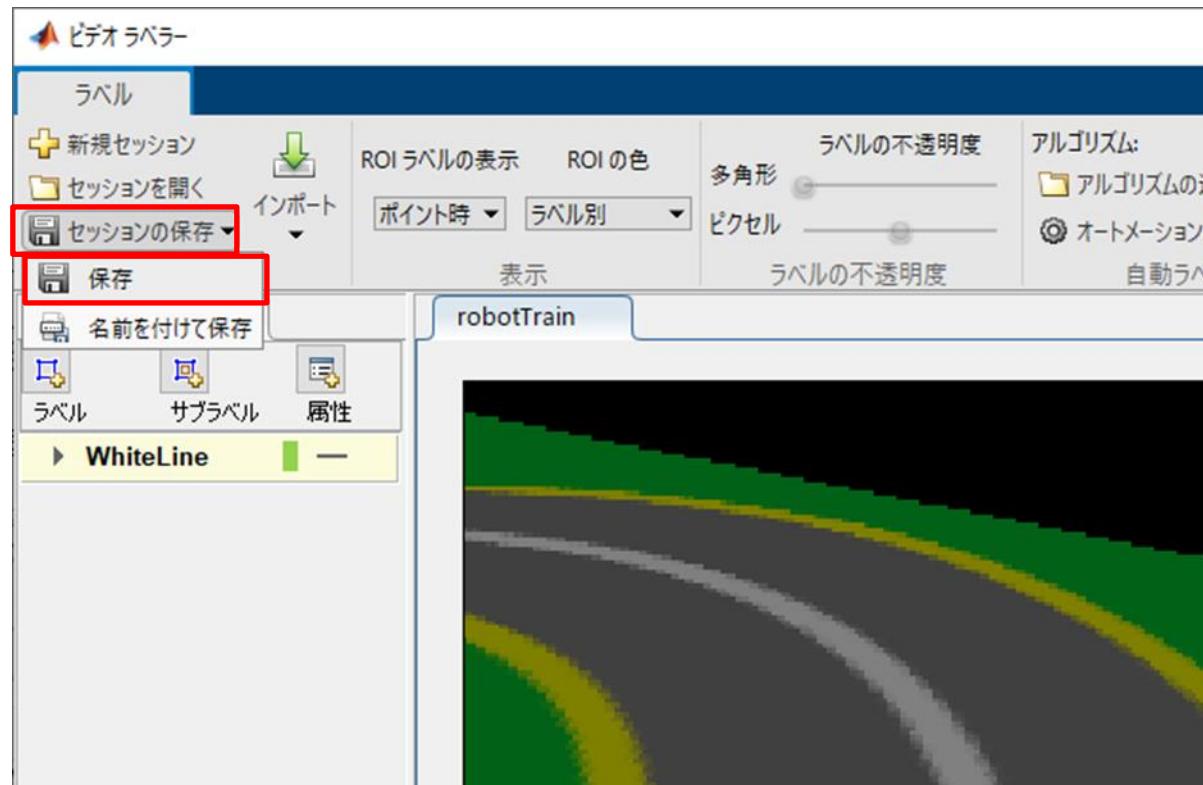


1. 画像の下側の左右中心をクリック
(この座標は学習では使用しないが
ツールでラインとして扱う都合上必要)

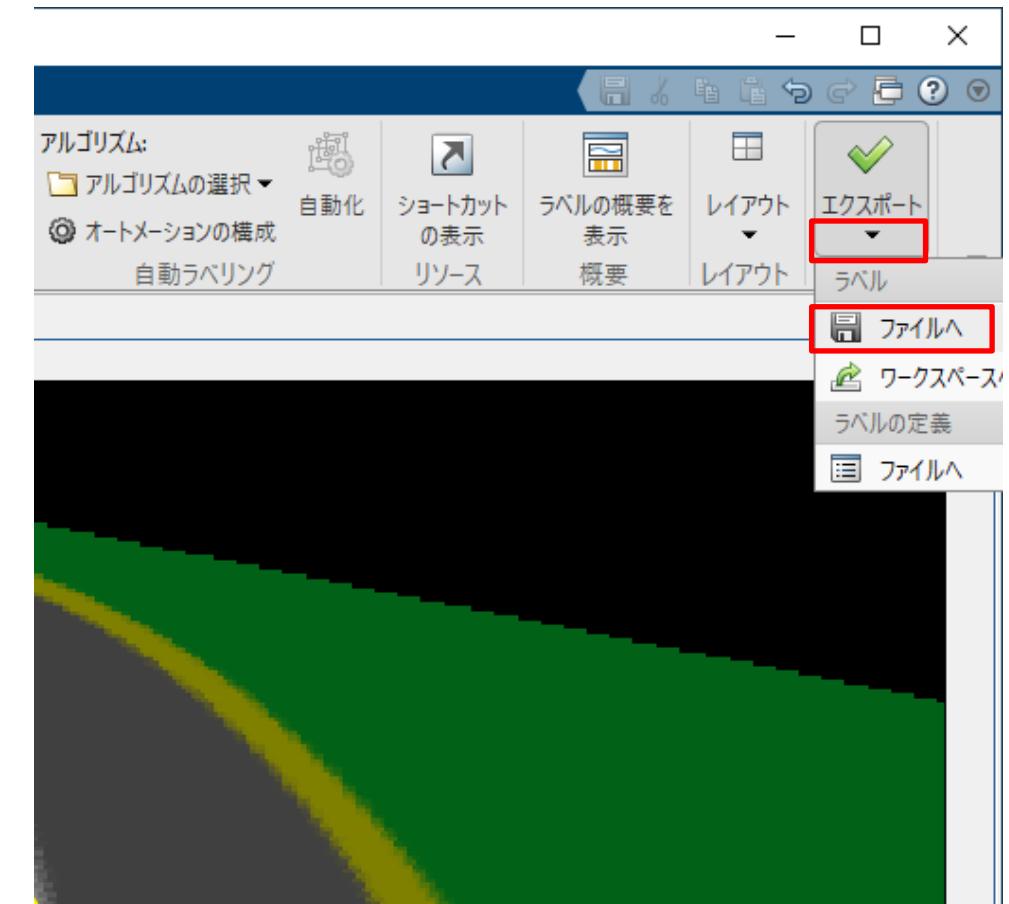
注意：3点以上の折れ線になった場合は
右クリックして削除して作り直す

演習2. ビデオラベラーを使ってラベリング

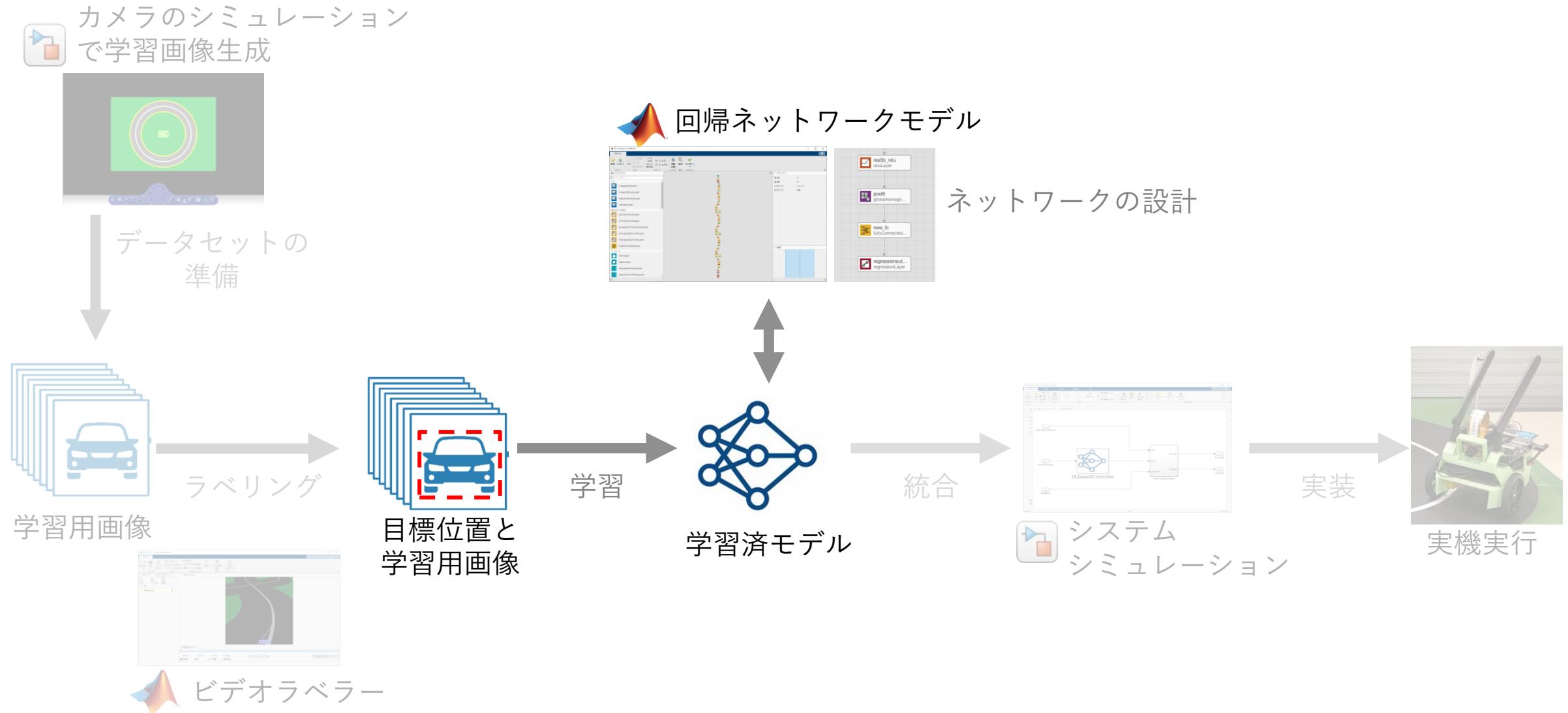
5. 適宜ラベリング結果をセッションとして保存する
(ファイル名はデフォルトで構わない)



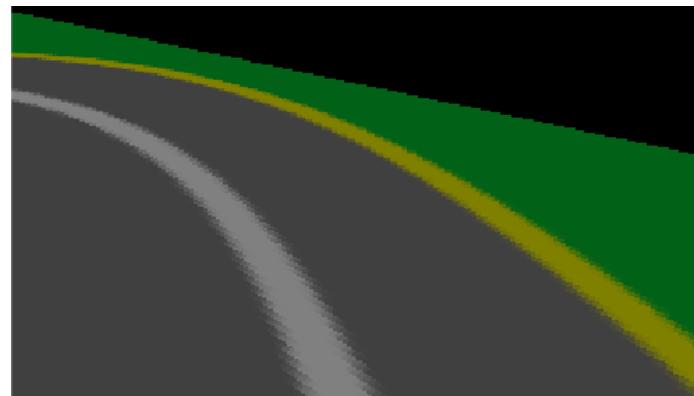
6. ラベルを「`exportedLabels.mat`」
として保存する



アジェンダ：自律ロボットシステム開発ワークフロー



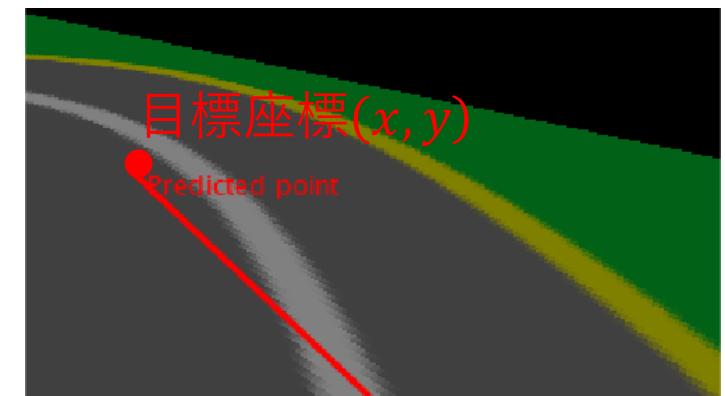
ディープラーニングによって目標座標を予測



カメラからの画像データ
($180 \times 320 \times 3$)



ディープラーニング
(重回帰)



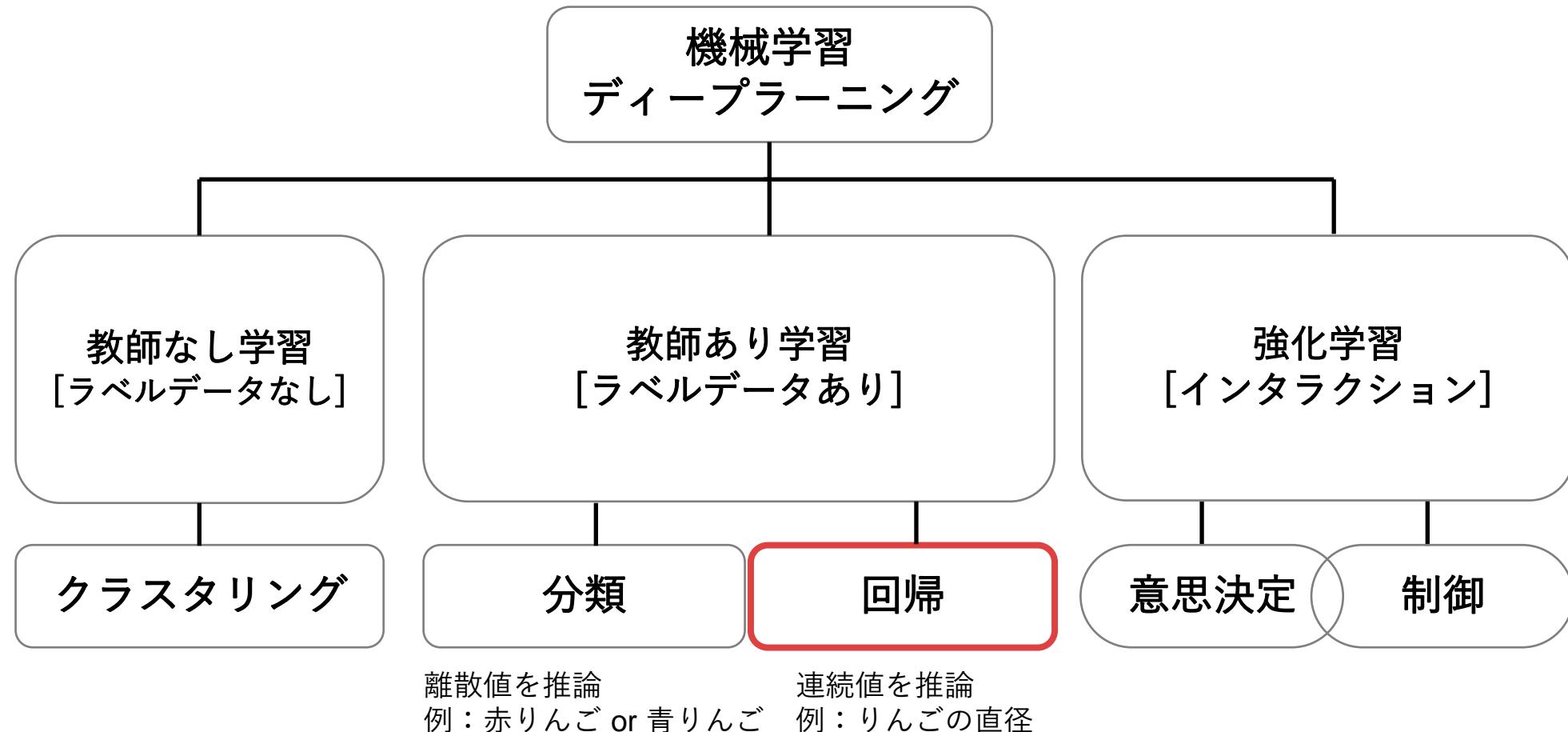
目標座標(x, y)
(1×2)

目標座標をディープラーニングモデルを使用して予測

なぜディープラーニングを使うか？

高次元の画像データから低次元の目標座標を出力するため
ディープラーニングによるモデル化が最適

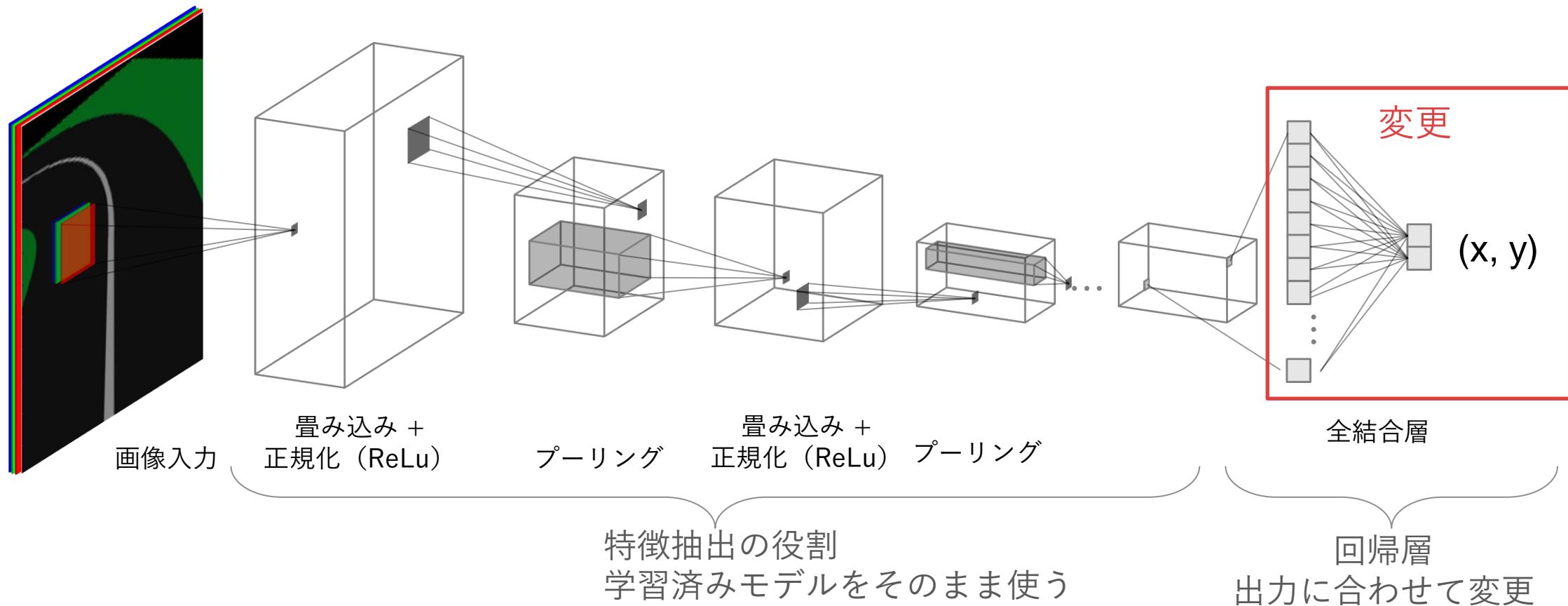
補足：機械学習の分類



今回は画像から目標座標を推論する
説明変数($180 \times 320 \times 3$)で目的変数 $x, y(2)$ の重回帰

転移学習によるディープラーニングの回帰

学習済みの画像分類のネットワークをベースとし、
後段の分類層を回帰層に変更して学習する（転移学習）



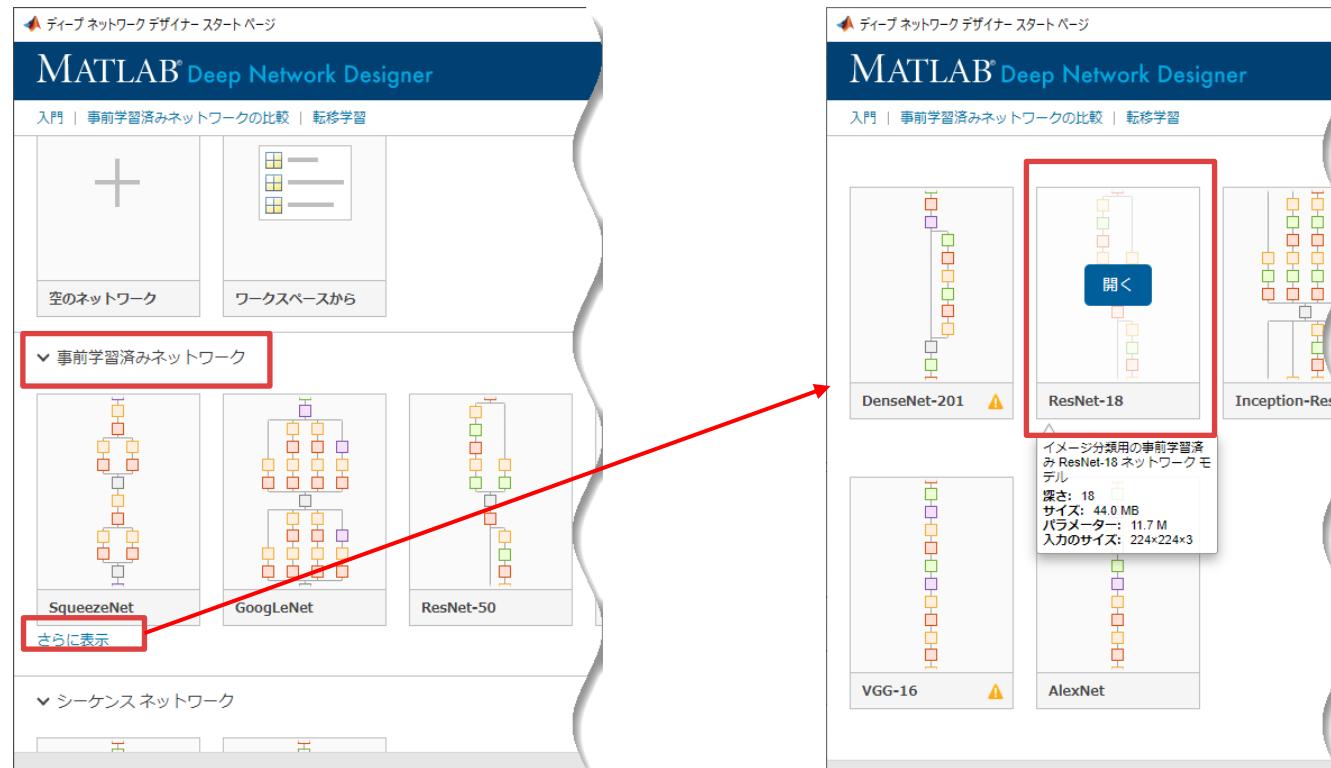
分類ネットワークの回帰ネットワークへの学習

<https://jp.mathworks.com/help/deeplearning/examples/convert-classification-network-into-regression-network.html>

演習3. 回帰ネットワークの構築と学習

学習済みの画像分類のネットワークをベースとし、
後段の分類層を回帰層に変更して学習する（転移学習）

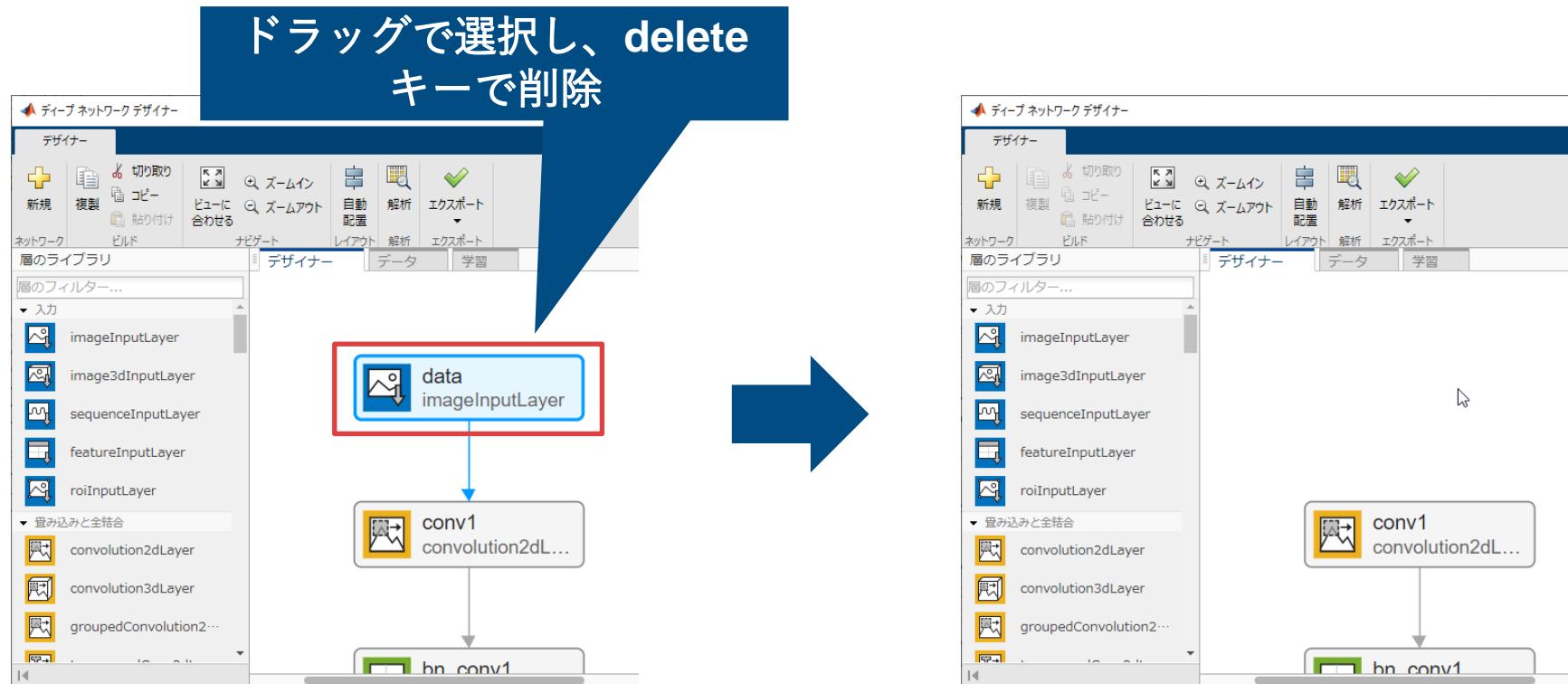
1. ライブスクリプトを開く
`>> edit trainWhiteLine_jp.mlx`
2. 「ライブエディター」タブの「実行して次へ進む」をクリックし、71行目まで実行する
3. 72行目を実行し、ディープネットワークデザイナーを開き、「ResNet-18」を開く



演習3. 回帰ネットワークの構築と学習

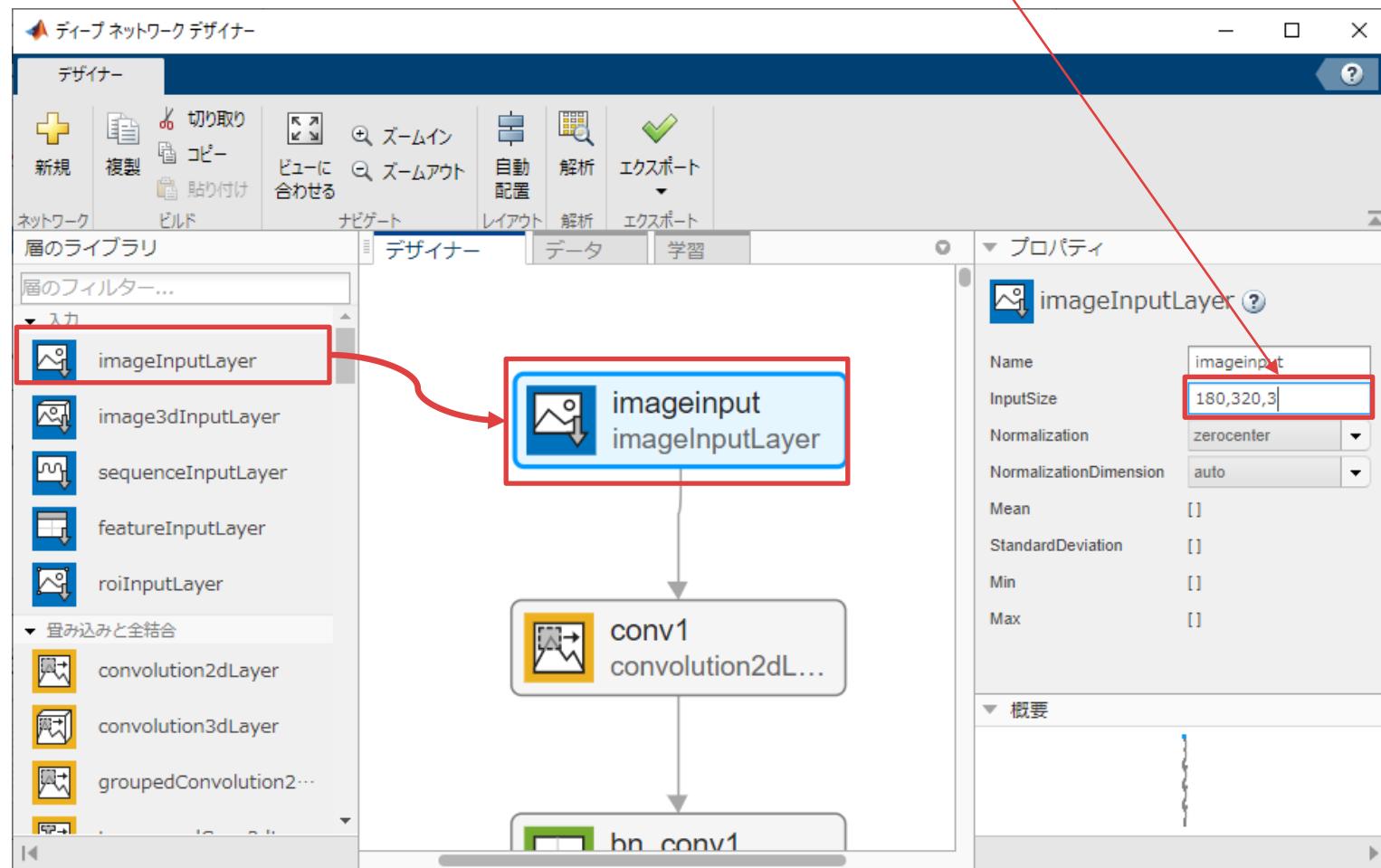
4. 入力の「**data**」レイヤーをドラッグで選択し、**delete**キーで削除

(学習済みの ResNet18 は $224 \times 224 \times 3$ だが学習画像は $180 \times 320 \times 3$ のため
入力層を置き換える必要がある)



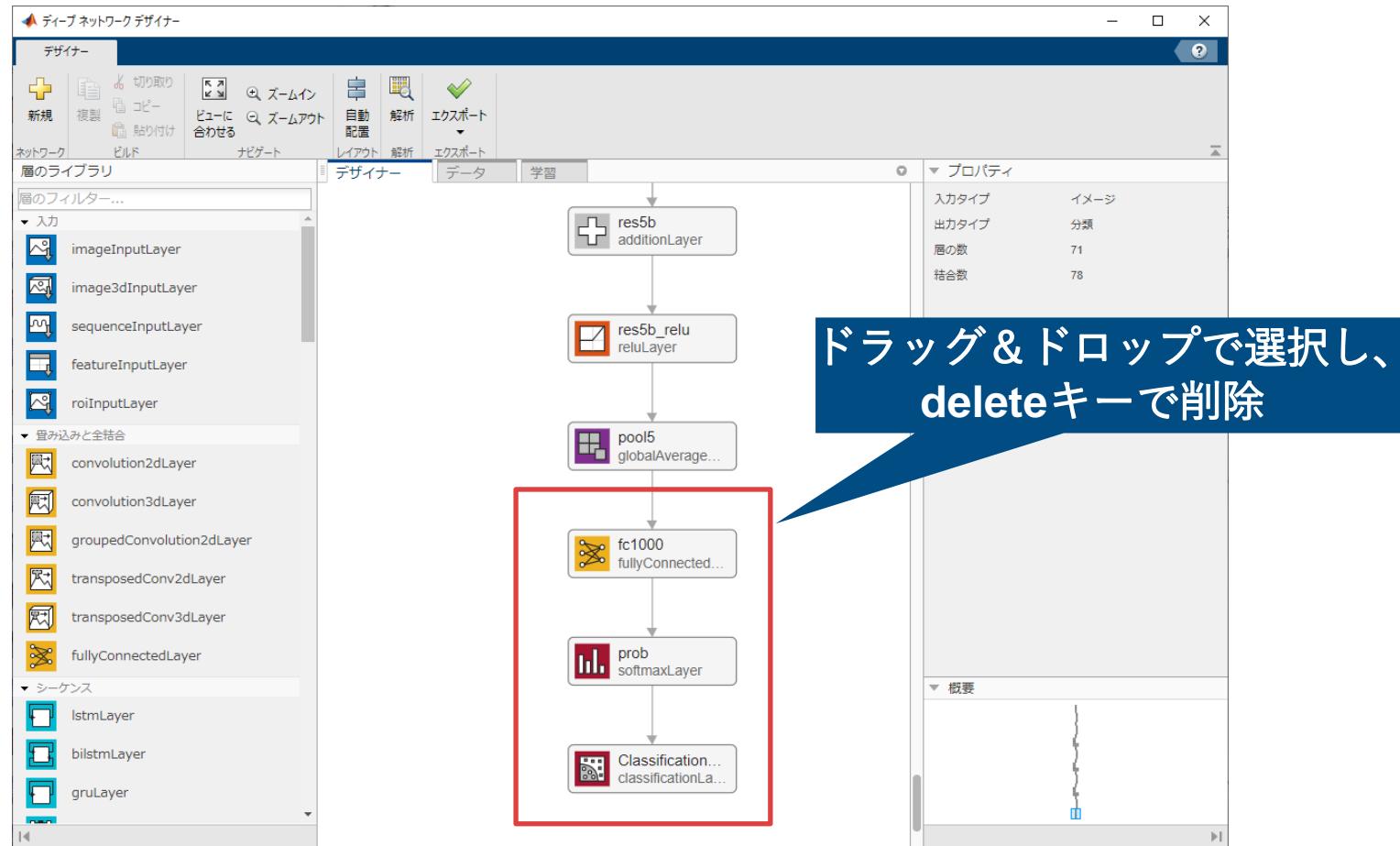
演習3. 回帰ネットワークの構築と学習

5. 「層のライブラリ」の「入力」グループから「**imageInputLayer**」をドラッグ&ドロップで追加
ドラッグ&ドロップで線を引き出し「conv1」と接続
imageinputのプロパティの「**InputSize**」を「**180,320,3**」に設定



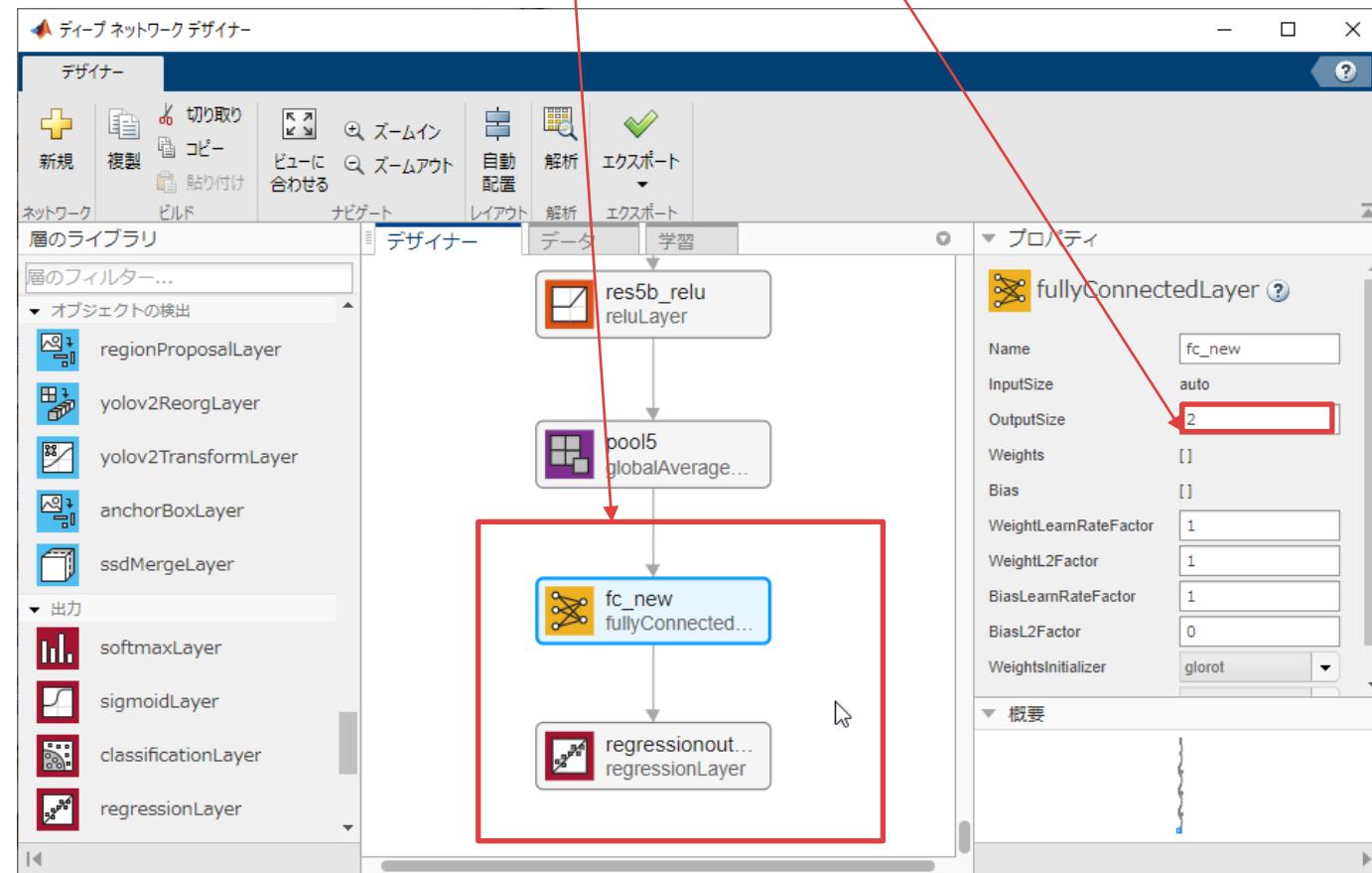
演習3. 回帰ネットワークの構築と学習

6. `fc1000`以降のレイヤーをドラッグ&ドロップで選択し、`delete`キーで削除
(1000種類確率+softmaxによる分類から2種類の回帰に変更するため)



演習3. 回帰ネットワークの構築と学習

7. 「層のライブラリ」の「畳み込みと全結合層」から「**fullyConnectedLayer**」と「出力」から「**regressionOutputLayer**」を追加
8. **fc(fullyConnectedLayer)**の「**OutputSize**」を「2」に変更(目標座標x,yの2変数なので)
9. **pool5**と**fc**、**fc**と**regressionOutput**をそれぞれ接続



演習3. 回帰ネットワークの構築と学習

8. 「解析」をクリックし、深層学習ネットワークアナライザーで整合性を確認
(エラーが0であればOK)



The screenshot shows two overlapping toolbars. The top toolbar is from the 'Deep Network Designer' and the bottom one is from the 'Deep Learning Network Analyzer'. Both toolbars have a 'Analysis' button highlighted with a red box and a red arrow pointing to the 'Deep Learning Network Analyzer' window.

Deep Network Designer Toolbar:

- File: 新規 (New), ピルド (Build), レイアウト (Layout), デザイナー (Designer), エクスポート (Export).
- Tools: 切り取り (Cut), コピー (Copy), 貼り付け (Paste), ピューに合わせる (Fit to View), ズームイン (Zoom In), ズームアウト (Zoom Out), 自動配置 (Auto Layout), 解析 (Analysis), エクスポート (Export).

Deep Learning Network Analyzer Window:

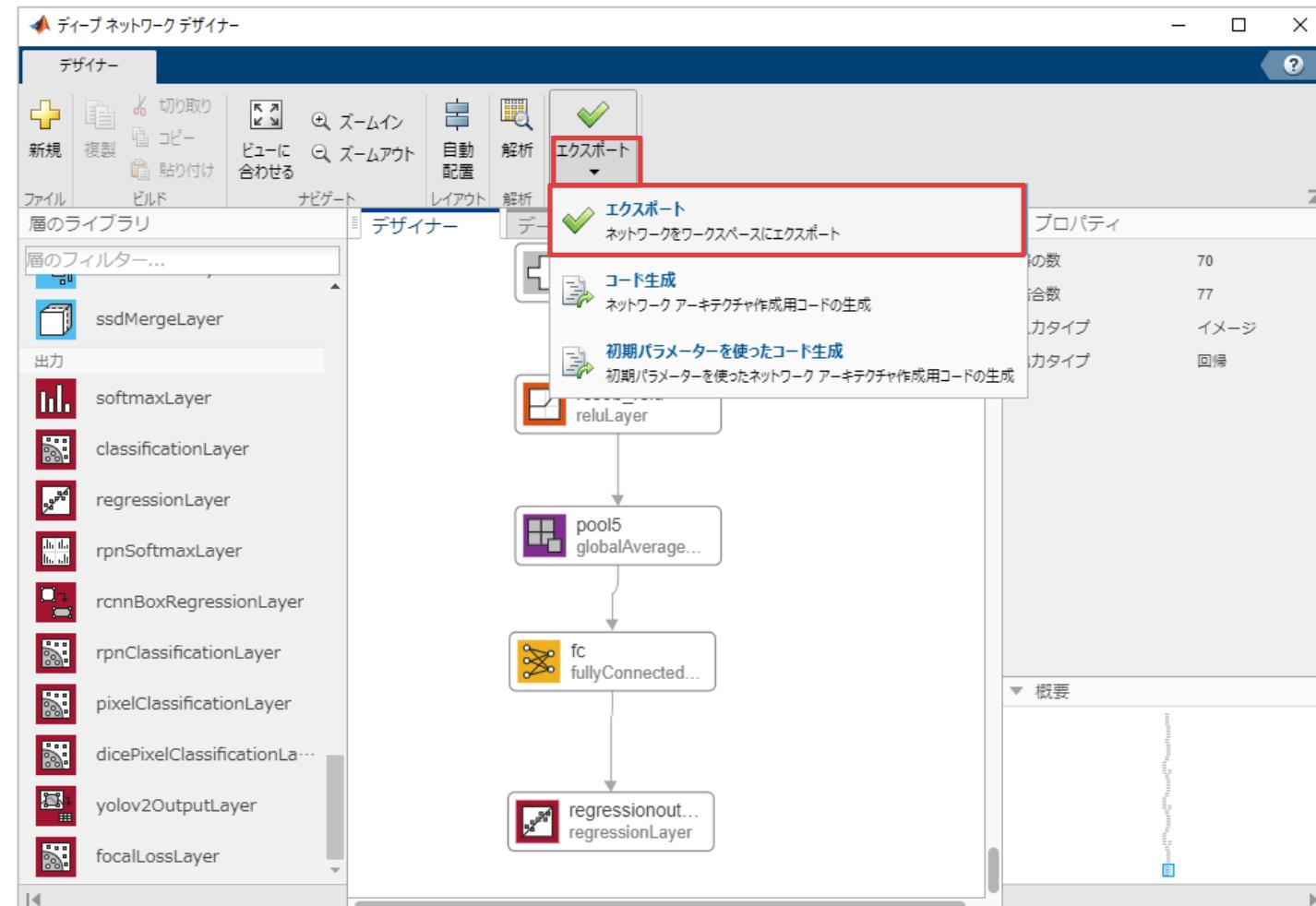
显示了从 Deep Network Designer 导入的网络结构。该窗口包含以下部分：

- Network Summary:** 展示了网络概况，包括层数 (70 层)、警告 (0) 和错误 (0)。
- Diagram:** 显示了网络的层间连接图。
- Analysis Results Table:** 列出了所有层的详细信息，包括名称、类型、激活函数和学习参数。

名前	タイプ	アクティベーション	学習可能
1 data	イメージの入力	-	-
2 conv1	畳み込み	Weights: 7x7x3x64, Bias: 1x1x64	-
3 bn_conv1	バッチ正規化	Offset: 1x1x64, Scale: 1x1x64	-
4 conv1_relu	ReLU	-	-
5 pool1	最大ブーリング	-	-
6 res2a_branch2a	畳み込み	Weights: 3x3x64x64, Bias: 1x1x64	-
7 bn2a_branch2a	バッチ正規化	Offset: 1x1x64, Scale: 1x1x64	-
8 res2a_branch2a_relu	ReLU	-	-
9 res2a_branch2b	畠み込み	Weights: 3x3x64x64, Bias: 1x1x64	-
10 bn2a_branch2b	バッチ正規化	Offset: 1x1x64, Scale: 1x1x64	-
11 res2a	加算	-	-
12 res2a_relu	ReLU	-	-
13 res2b_branch2a	畠み込み	Weights: 3x3x64x64, Bias: 1x1x64	-
14 bn2b_branch2a	バッチ正規化	Offset: 1x1x64, Scale: 1x1x64	-
15 res2b_branch2a_relu	ReLU	-	-

演習3. 回帰ネットワークの構築と学習

9. 「エクスポート」 → 「エクスポート」をクリックしワークスペースに出力
(デフォルトで`lgraph_1`としてエクスポートされる)

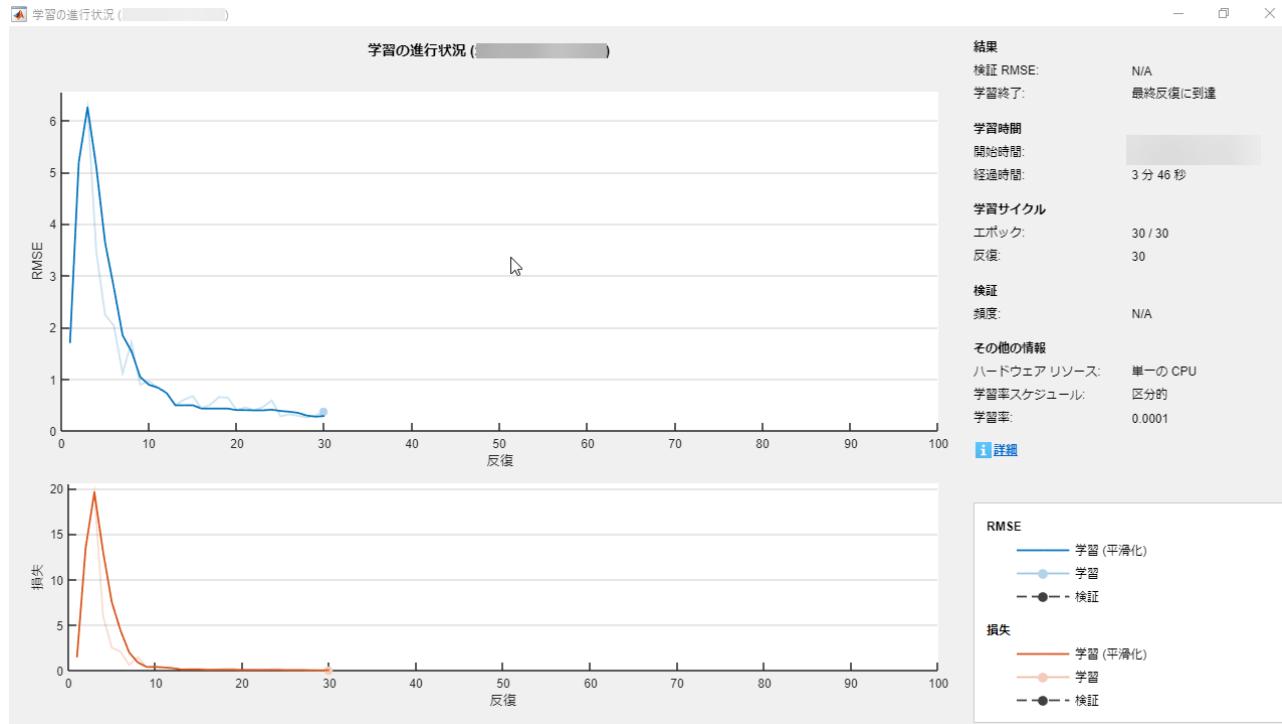


演習3. 回帰ネットワークの構築と学習

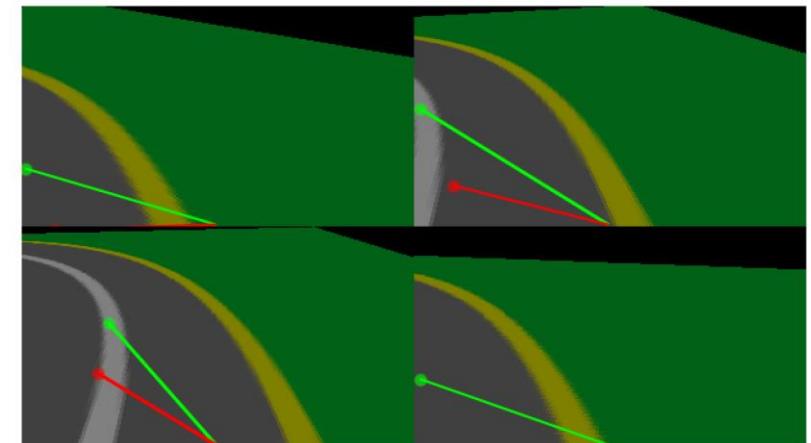
10. 残りのコードを実行する

学習が開始すると下記のような学習の進捗状況が表示される(CPUで5分ほどで完了)

11. 学習が完了すると `mynet_new.mat` がフォルダに作成される (次の演習で使用する)



学習の様子



予測結果の例
(緑: 真値、赤: 予測)

アジェンダ：自律ロボットシステム開発ワークフロー

カメラのシミュレーション
で学習画像生成



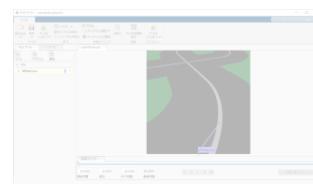
データセットの
準備



ラベリング



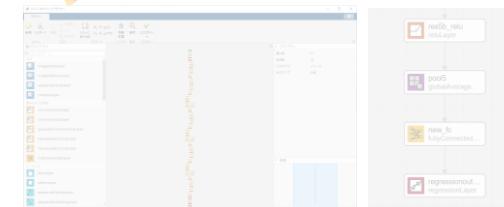
目標位置と
学習用画像



学習用画像

ビデオラベラー

回帰ネットワークモデル



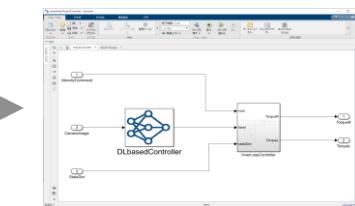
ネットワークの設計



学習

学習済モデル

統合



システム
シミュレーション

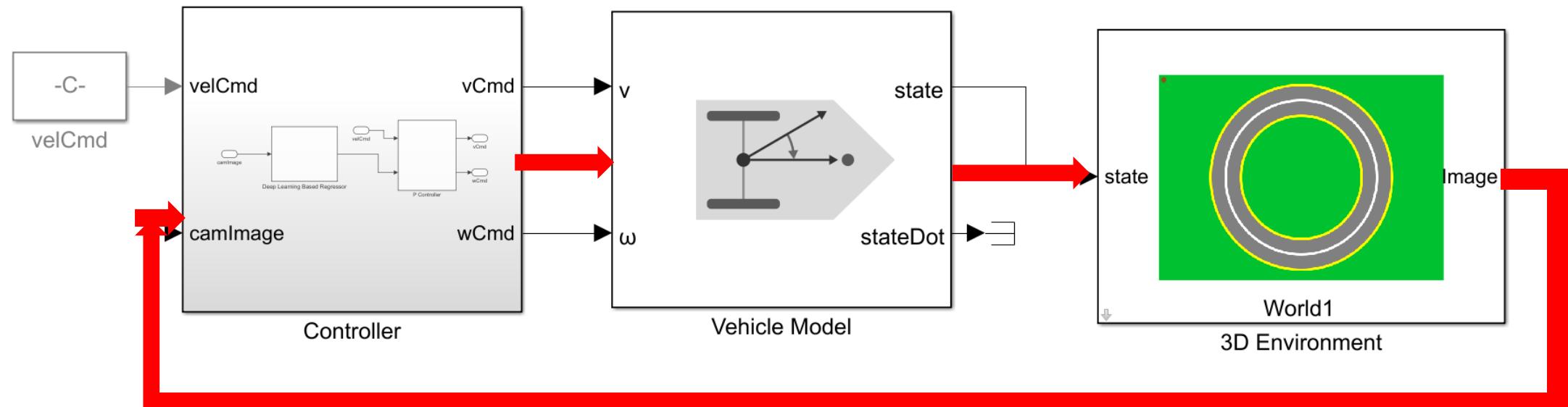


実装

実機実行

システムシミュレーションでライントレースの動作確認

velCmd:Value



Copyright 2020 Tha MathWorks, Inc.

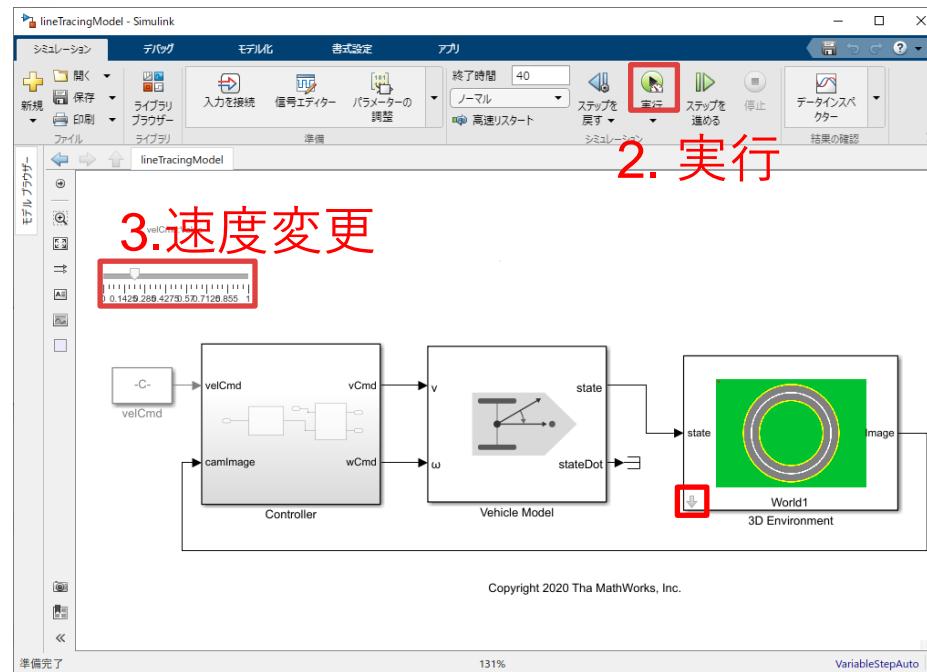
全体で大きなフィードバックループを形成している
システムレベルでの性能の確認やチューニングが不可欠

演習4. システムシミュレーションでライントレースの動作確認

- line_follower.slx** を開き、「シミュレーション」タブの実行ボタン(下図の緑色の再生ボタン)をクリックする。シミュレーションを停止する場合は四角の灰色ボタンをクリックする。

>> line_follower.slx

- 速度(下図のバー)を変更しライントレースの様子がどう変わるか観察する。
- 初期位置を変えたときに挙動がどうなるか観察する ([スライドp.35](#)参照)
- Pゲインの値を変更し、追従性能がどう変わるか観察する ([スライドp.34](#)参照)
- 3Dワールドの地図データ変更し、挙動がどうなるか観察する ([スライドp.36](#)参照)
- 実行結果を動画に保存する ([スライド p.37](#)参照)

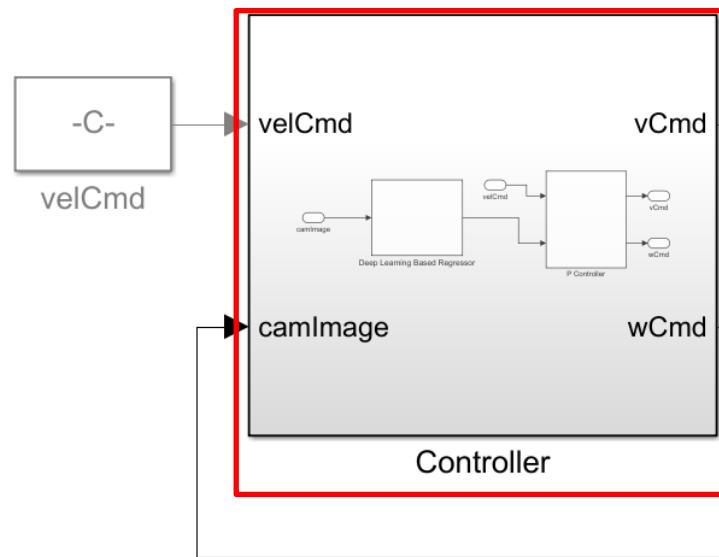


シミュレーションによる検証: モデルの解説

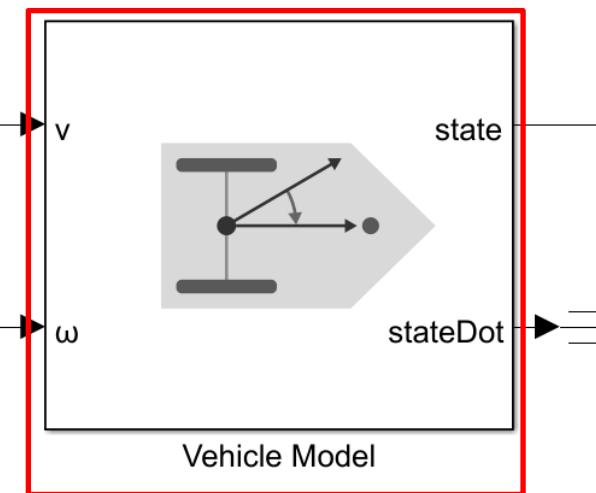
velCmd:Value



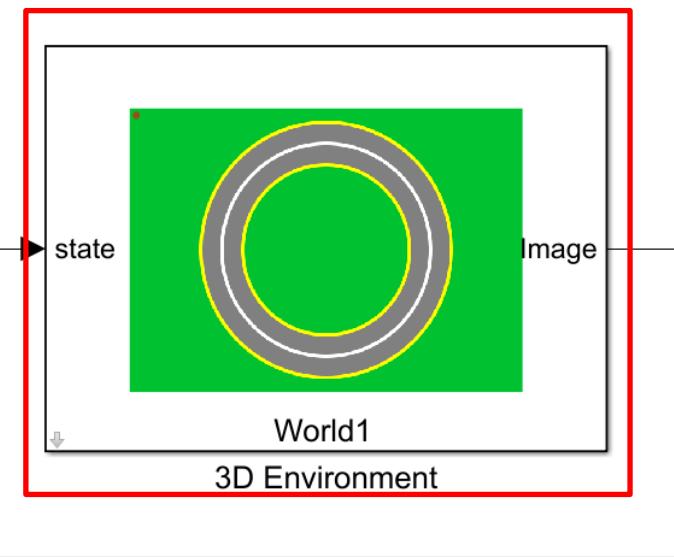
コントローラ



車両モデル

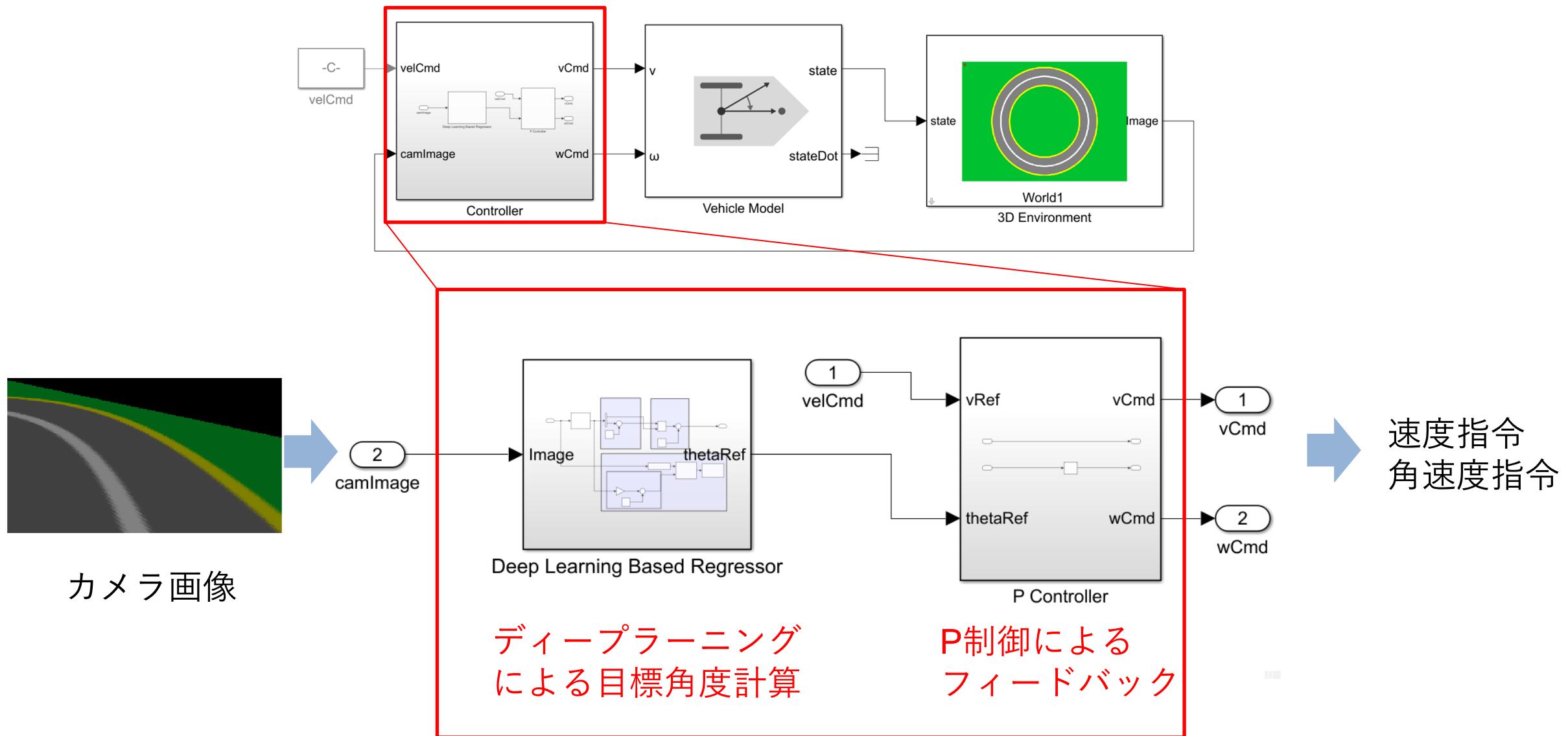


3Dワールド

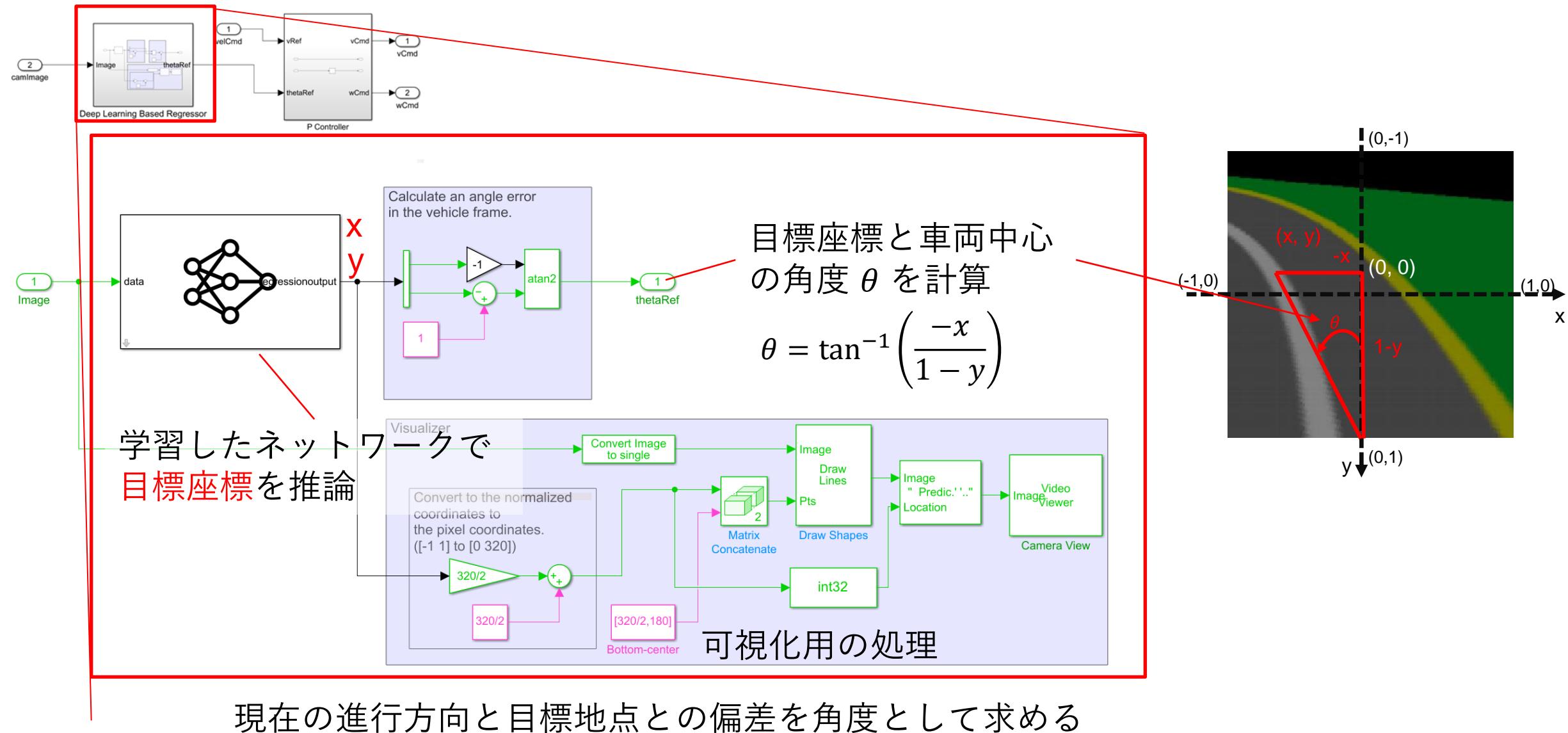


Copyright 2020 Tha MathWorks, Inc.

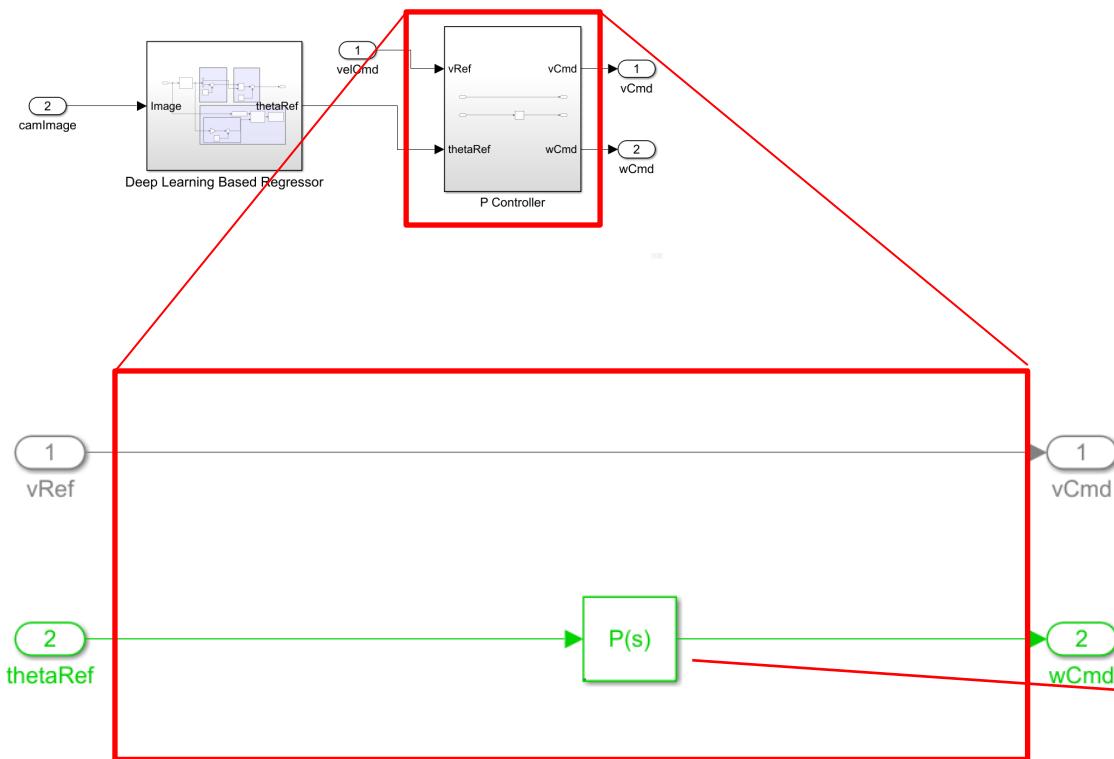
シミュレーションによる検証: コントローラ



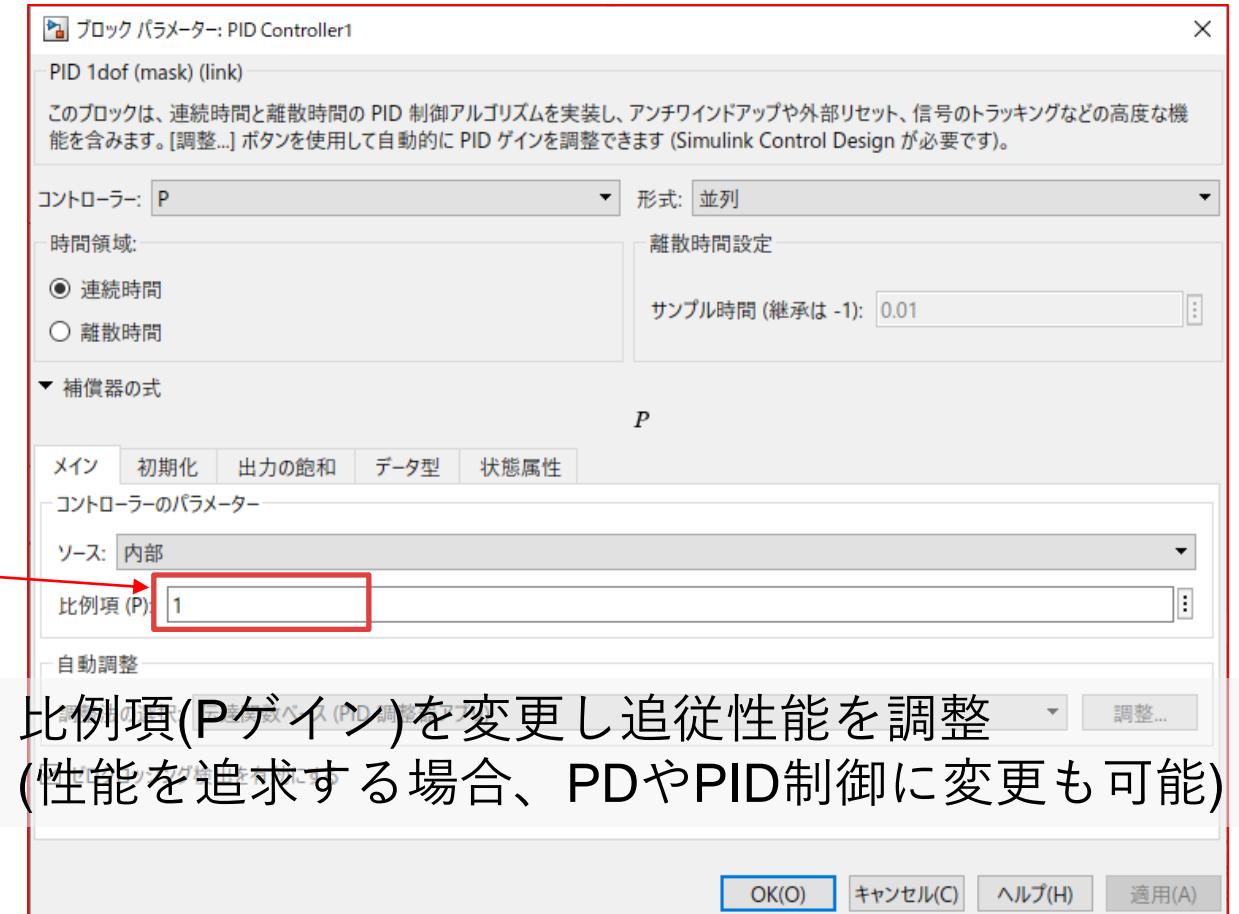
シミュレーションによる検証: ディープラーニングで角度偏差推定



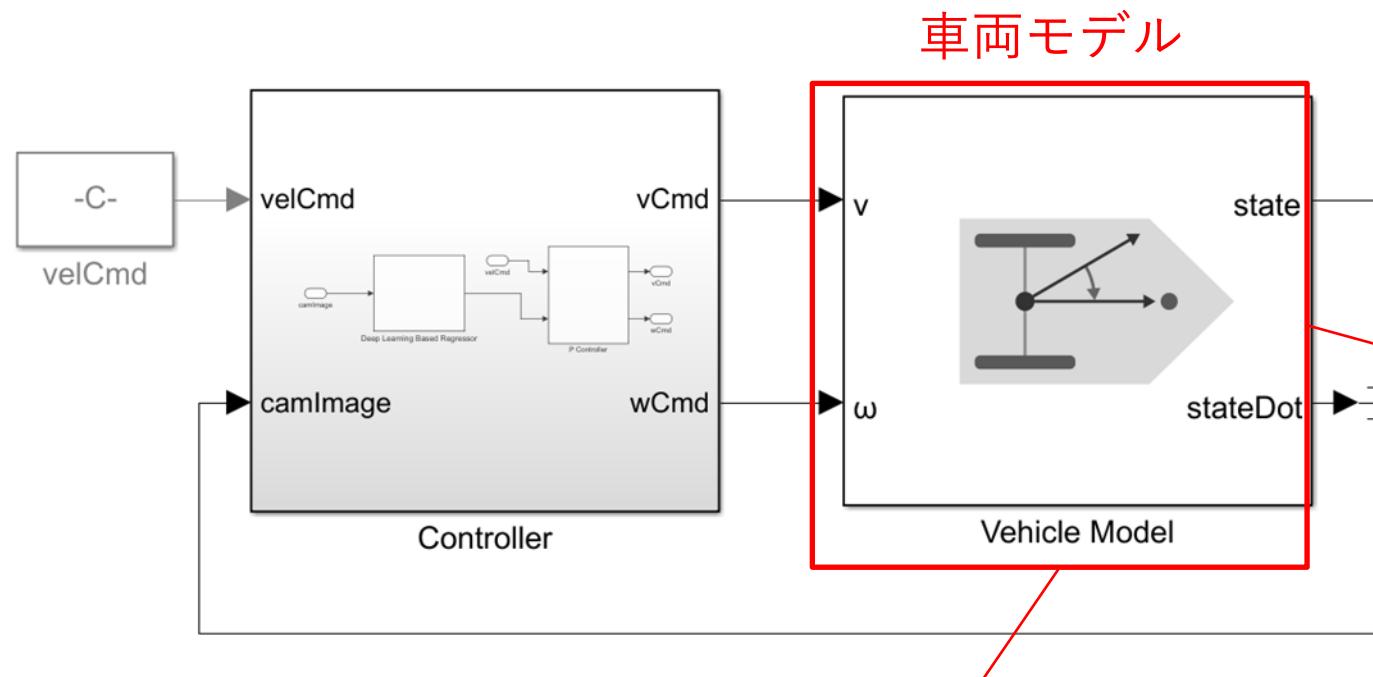
シミュレーションによる検証: フィードバック制御



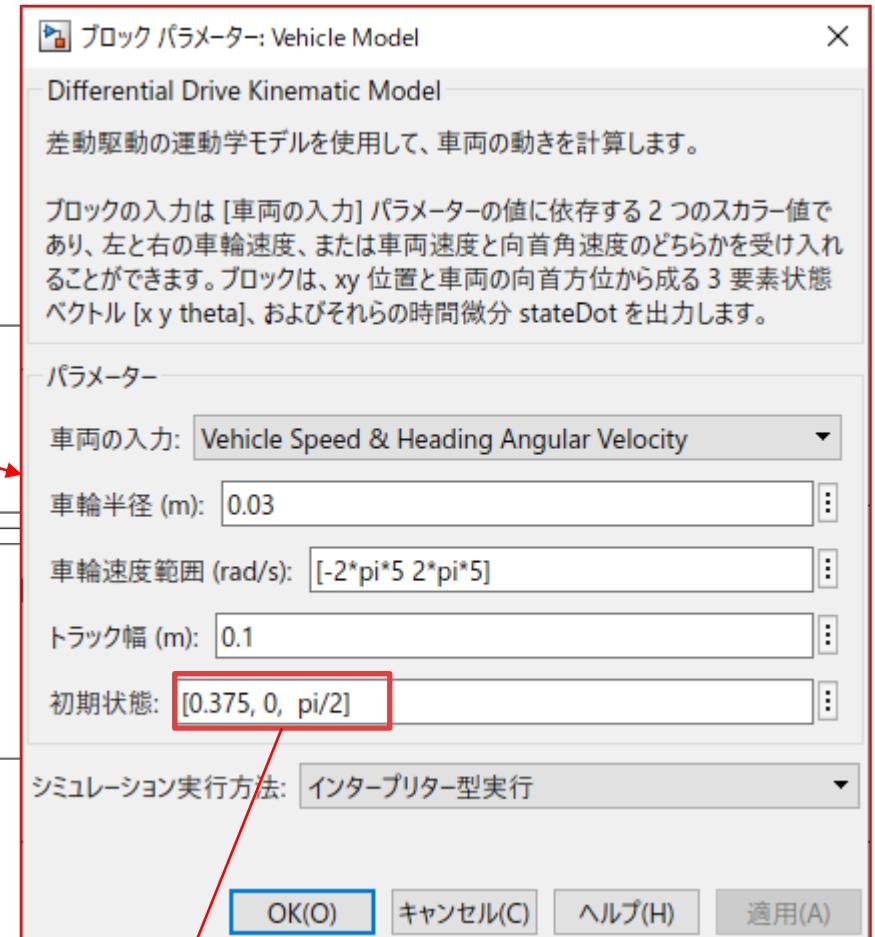
角度の偏差が0になるように
フィードバック制御(P制御)
ステアリングのみを制御
速度は一定



シミュレーションによる検証: 車両モデル

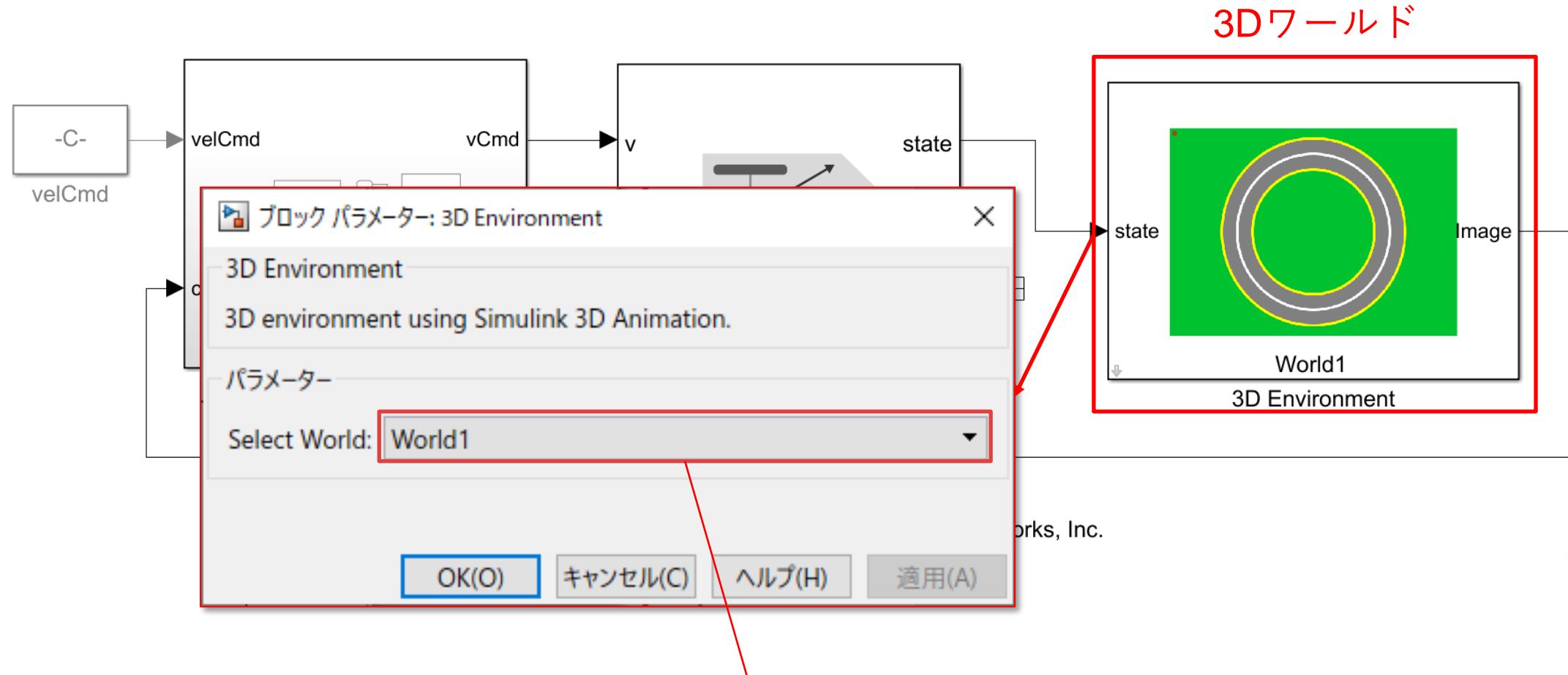


差動2輪の運動学モデルに
速度と角速度を入力し、
位置(x_v, y_v, θ_v)を得る

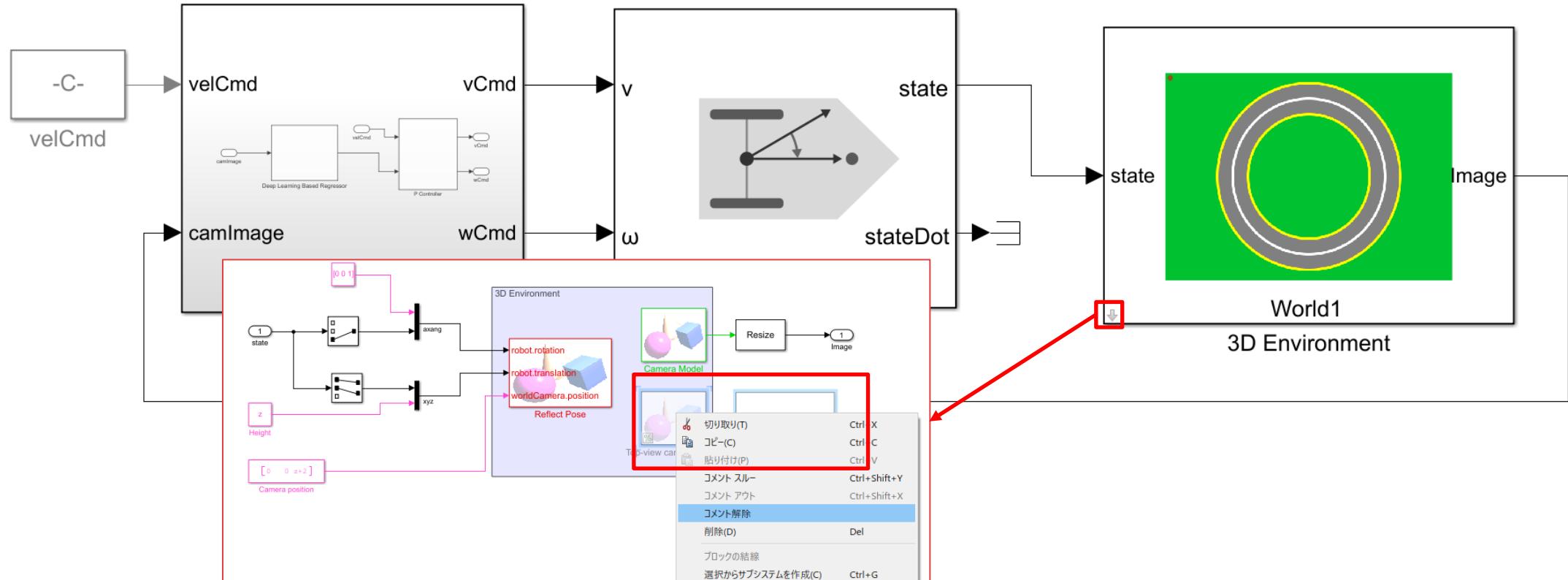


初期状態(x_v, y_v, θ_v)を変えたときに
ライントレースの挙動はどうなるか

シミュレーションによる検証: 3Dワールドの地図データ変更



シミュレーションによる検証: 3Dワールドの実行結果動画保存



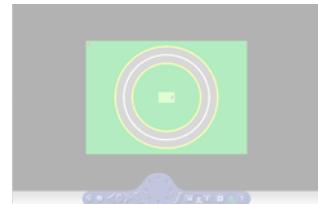
Top-view cameraとTo Multimedia Fileのコメントを
解除するとシミュレーション実行後に動画が生成される
(ただし、シミュレーションが遅くなるため最終結果を
保存するときのみコメント解除すること)

改善するためのヒント

- 学習画像を増やす(演習1)
 - 様々な視点から見た学習画像を準備する
- 学習画像のアノテーション(ラベル付け)を工夫する(演習2)
 - 予測する点を遠くに設定する、もしくは、近くに設定する
- 学習のパラメーターを変更してみる(演習3)
 - エポック数を大きな値に変えてみる
 - 学習率を変えてみる
- P制御のゲインを調整してみる(演習4)
 - Controller/P Controllerブロックの中のPゲインを調整
 - 追従できない場合はゲインを上げてみる
 - 発振する場合にはゲインを下げてみる

アジェンダ：自律ロボットシステム開発ワークフロー

カメラのシミュレーション
で学習画像生成



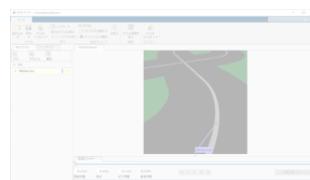
データセットの
準備



ラベリング

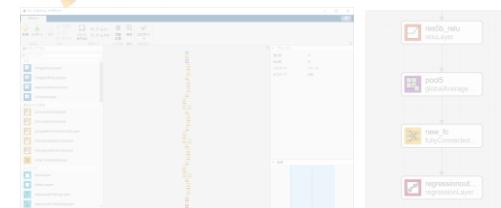


目標位置と
学習用画像



ビデオラベラー

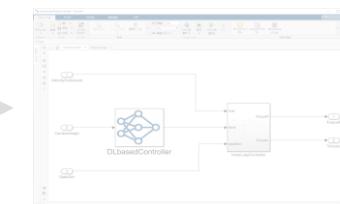
回帰ネットワークモデル



ネットワークの設計



学習済モデル



システム
シミュレーション

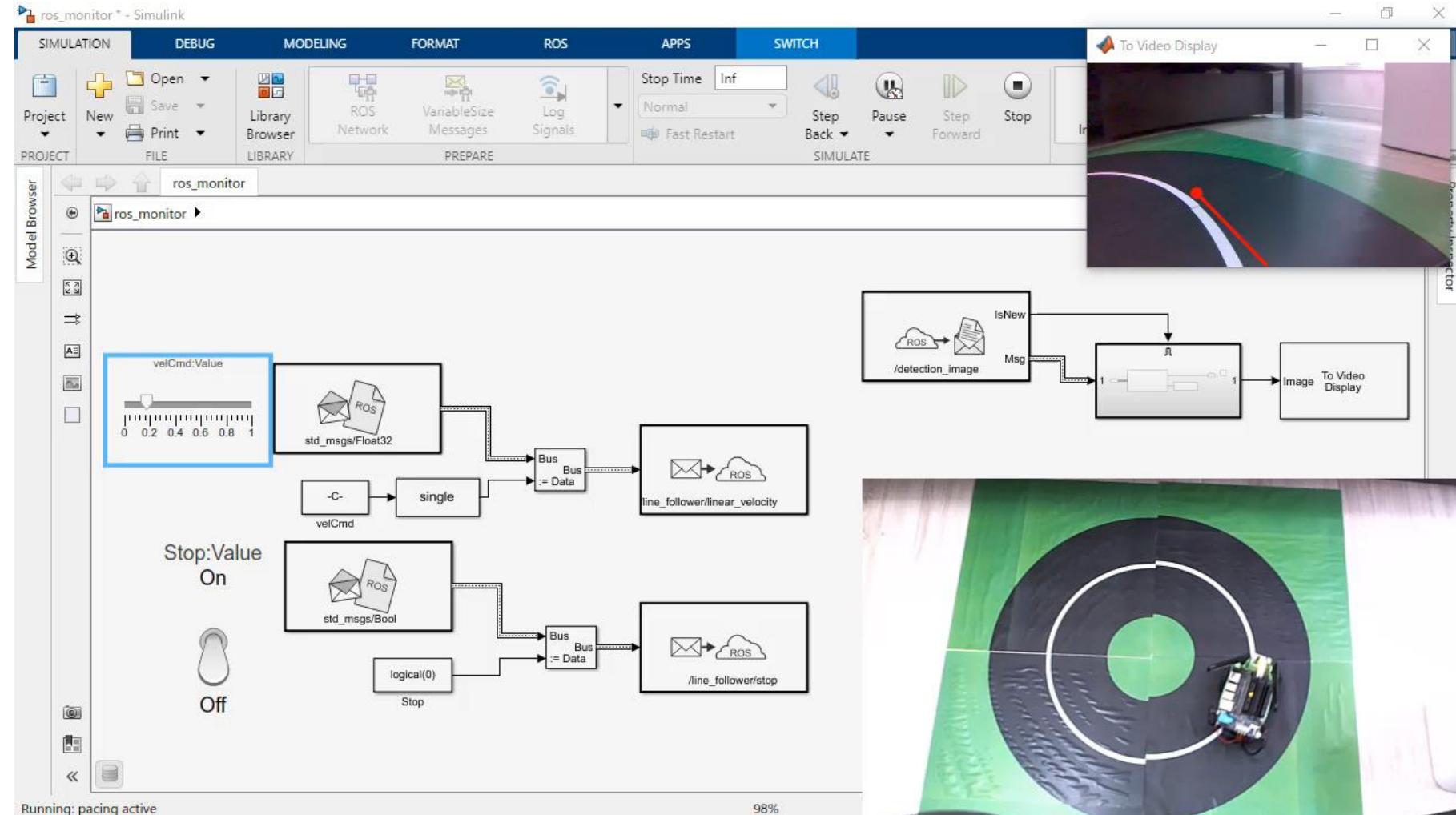


実装

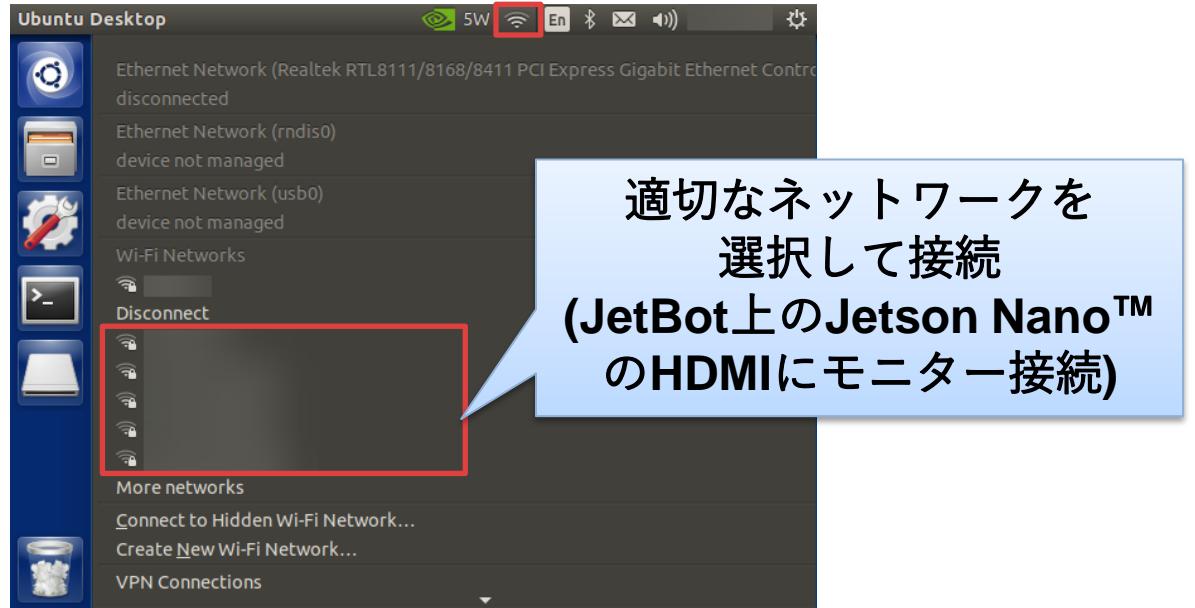
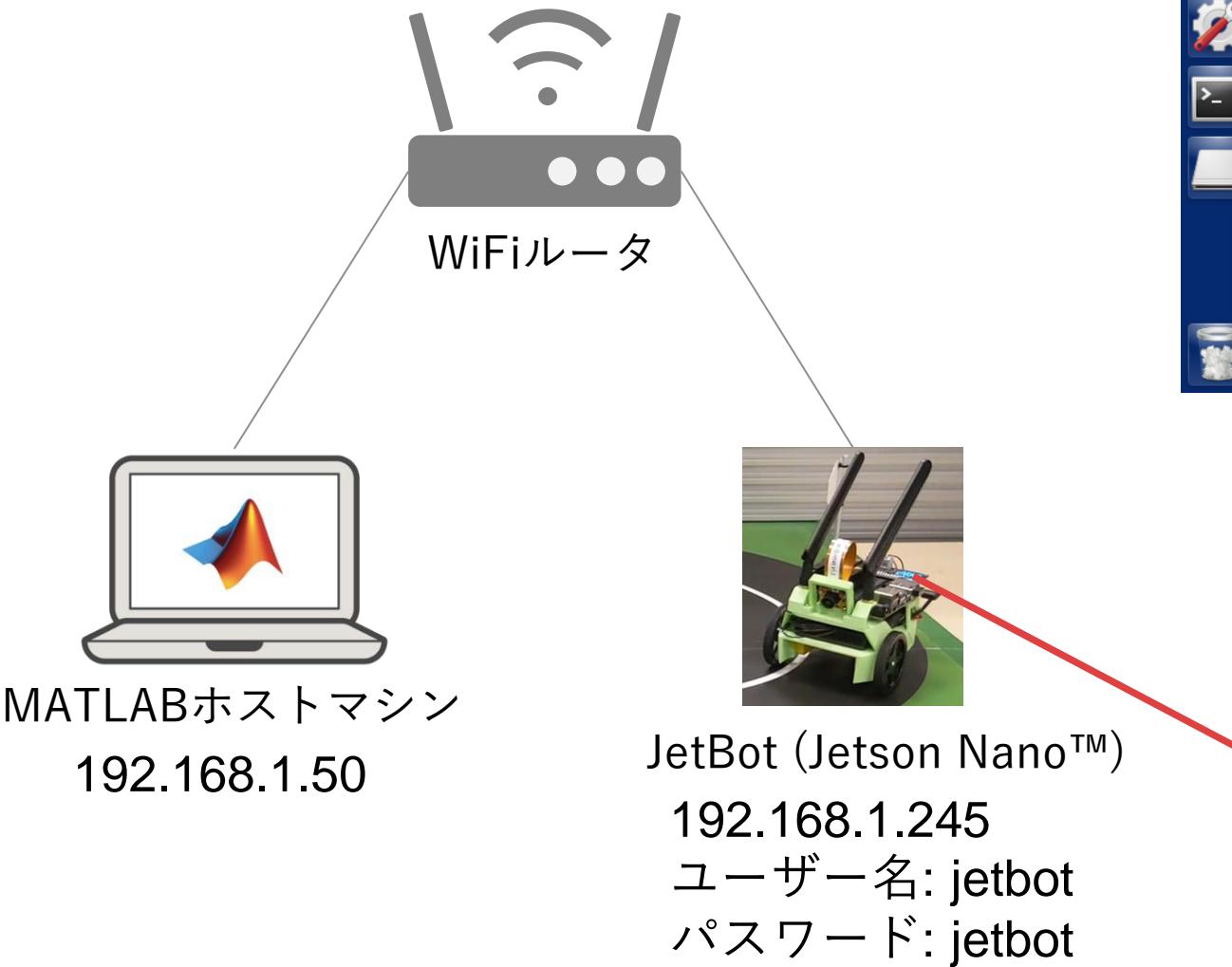
実機実行

目標：ライントレースロボットの実機実装

ミドルウェアROSと連携し、ライントレースアルゴリズムを実機実装



MATLABとJetBot*を接続



*JetBotのセットアップは末尾の付録参照

NVIDIA® Jetson Nano™ および NVIDIA® JetBotとは？

- NVIDIA®の廉価な組み込みGPU
 - ARM® Cortex®-A57 + NVIDIA® Maxwell
 - GPIO、CSI等を含む様々なI/O
 - 5W/10Wモード, 5V DCin
 - Linux OS
 - NVIDIA® CUDA/cuDNN/TensorRT
- MATLAB Coder/GPU Coderのハードウェアサポートパッケージでサポート
- アプリケーション
 - 画像分類
 - 物体検出
 - セグメンテーション
 - 音声処理
- オープンロボットハードウェア
 - NVIDIA®のJetson Nano™をベースとしたAIロボットプラットフォーム
 - シンプルな単眼カメラセンサーとモーター駆動の差動二輪
 - 部品のリストがGitHubで公開されている
 - Jetson Nano™を購入し、3Dプリンターで部品を作れば作成可能
 - <https://jetbot.org/>
- アプリケーション
 - 衝突回避
 - 白線追尾
 - 物体認識



ROS とは?



- ROS = Robot Operating System
- 自律分散システムのためミドルウェア
 - 一般的なOSではない – ミドルウェアの一種
- メッセージング通信/分散コンピューティングのためのツール群
- アプリケーションのパッケージ化とビルドのための管理システム
- 活発なコミュニティ
 - ドライバー、シミュレーション、モーションプランニング、認知、etc.
- ROS 1を刷新したROS 2が近年リリース
 - コミュニティでもROS 2への移行が始まっている

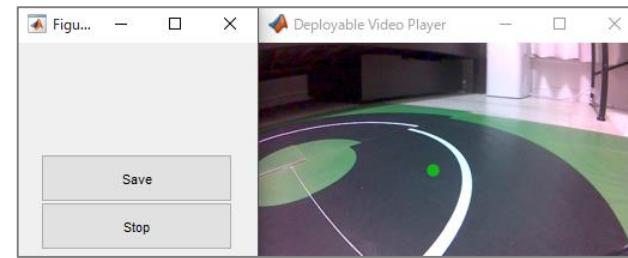
ROSを活用することで自律分散システムを効率的に開発可能

ROS Toolbox によるROS連携

ワークショップで使用する機能

	ROS	ROS 2
MATLAB	<ul style="list-style-type: none">• Topic – Publish / Subscribe• Service – Server / Client• Action – Client• Parameter – Get / Set• カスタムメッセージ• rosbag 読み込み• ROSノード生成 R2021a	<ul style="list-style-type: none">• Topic – Publish / Subscribe• カスタムメッセージ• ros2bag 読み込み R2021a
Simulink	<ul style="list-style-type: none">• Topic – Publish / Subscribe• Service – Call• Parameter – Get / Set• ROS Time• rosbag playback• ROSノード生成	<ul style="list-style-type: none">• Topic – Publish / Subscribe• ROSノード生成
ROS Distro	<ul style="list-style-type: none">• ROS Melodic R2020b	<ul style="list-style-type: none">• ROS2 Dashing R2020a

ステップ1: ROSの接続テストと学習画像収集

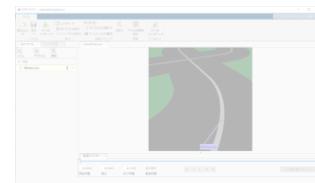


データセットの
準備



動画

ラベリン
グ



ビデオラベラー

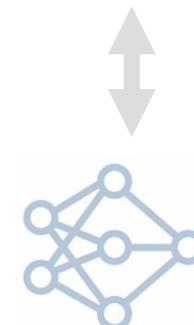


ラベル済み
画像



回帰ネットワークモデル

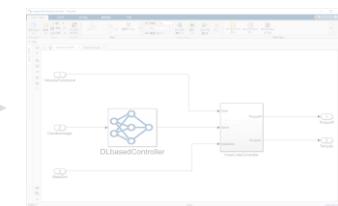
ネットワークの設計



学習

学習済モデル

統合



システム
シミュレーション

実装



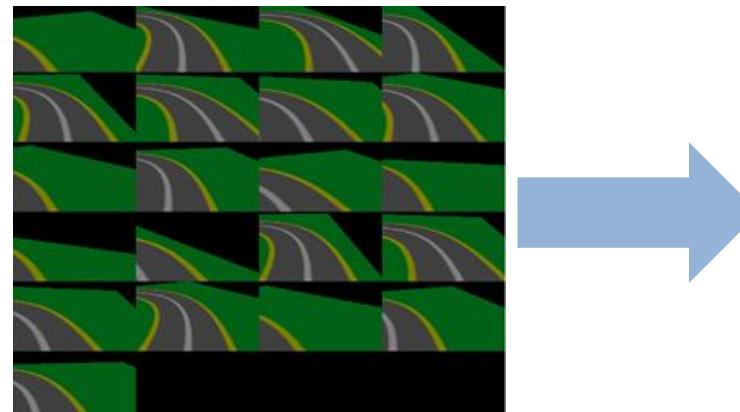
実機実行

シミュレーション環境と実機環境の差分

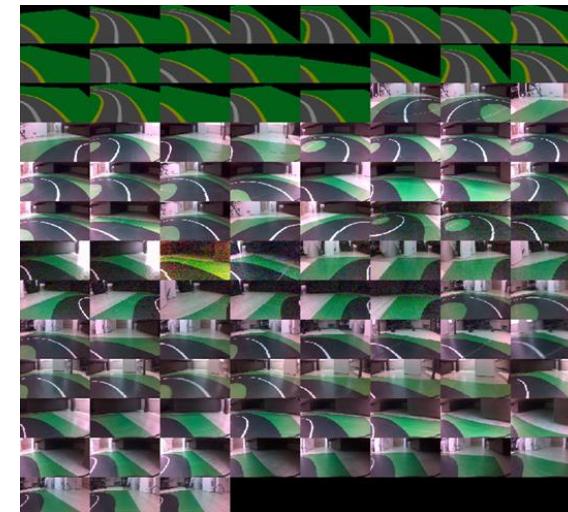
シミュレーション環境で学習した検出モデルでは実機環境に対応できない場合がある
例えば下記の差分が存在する*

- ・周囲の障害物の有無
- ・環境高の明るさや反射
- ・画像センサーのノイズ

実環境からもデータを収集し、よりロバストな学習モデルを構築する



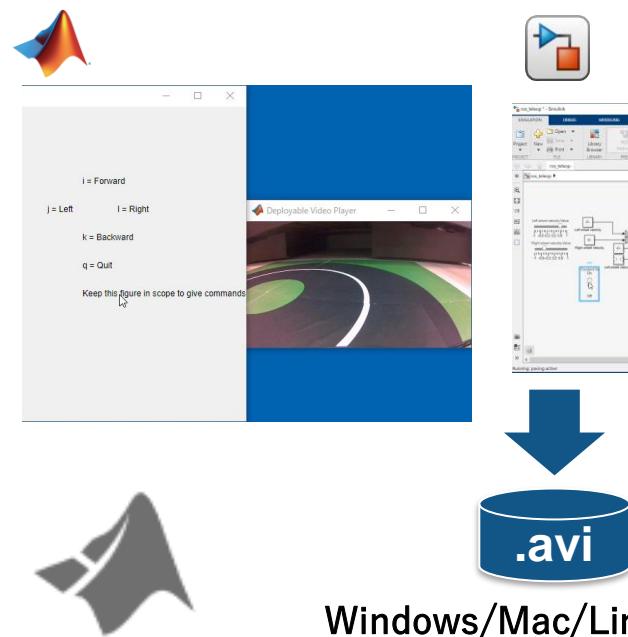
シミュレーション環境で
収集した画像



実機環境で収集した
画像を追加

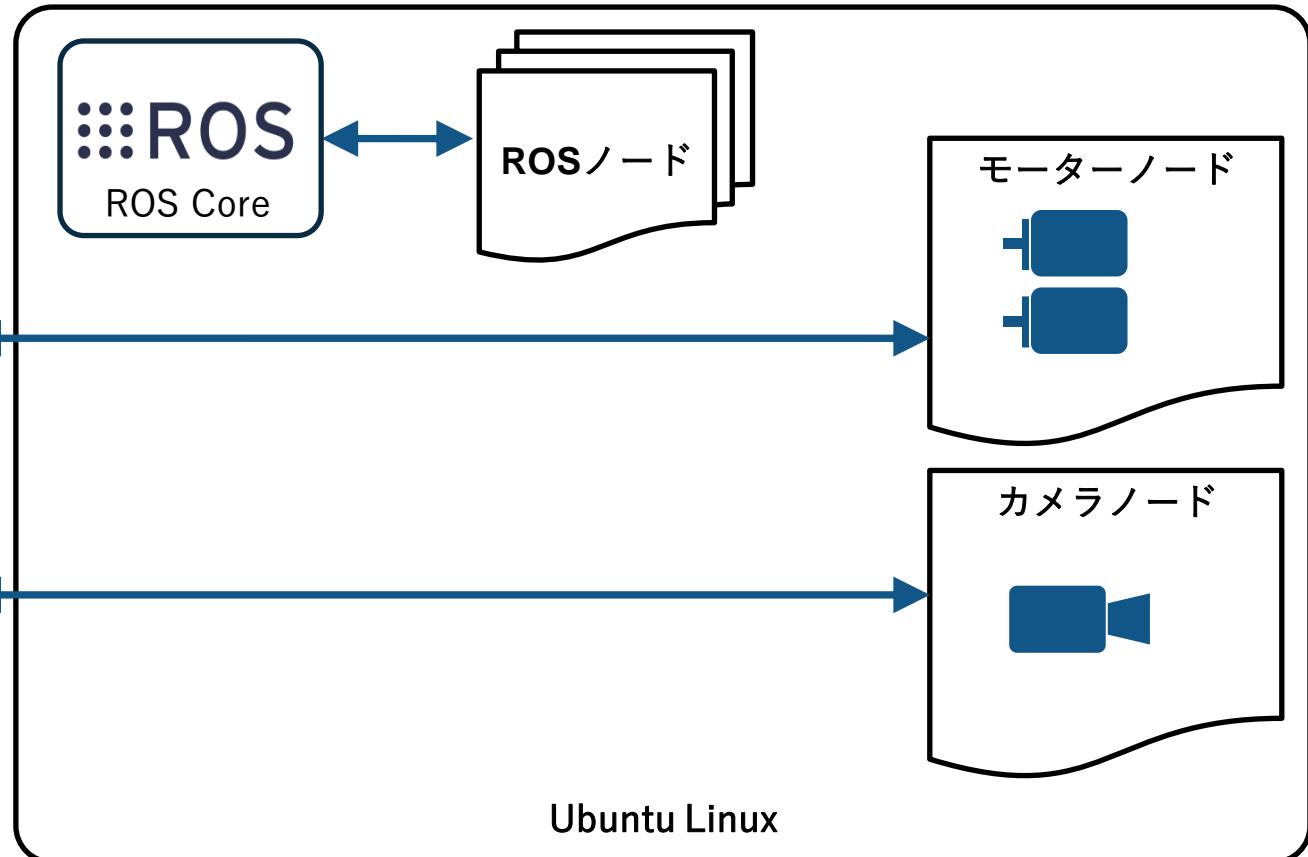
ROSの接続テストと学習画像収集

- ROS経由でMATLAB/SimulinkとJetBotを接続
- 両輪のモーターの指令値を送信
 - カメラ画像を受信し保存



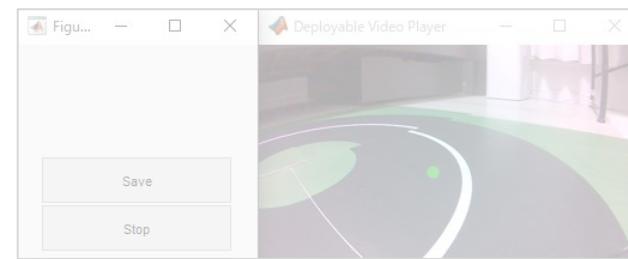
Development Machine

`>> edit teleopTestJetBot_jp.mlx`



JetBotをいろいろな
場所に設置する
or
走行させる

ステップ2: ライン検出器の改善



データセットの
準備

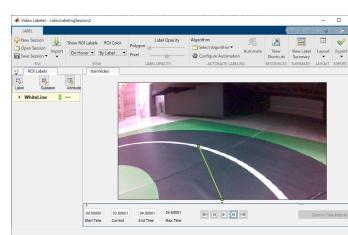


ラベリング



ラベル済み
画像

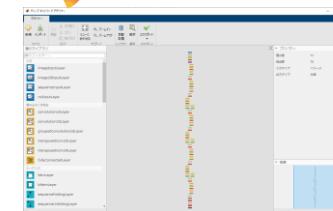
動画



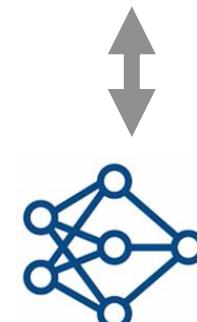
ビデオラベラー



回帰ネットワークモデル



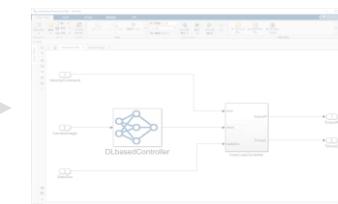
ネットワークの設計



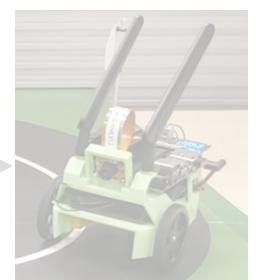
学習

学習済モデル

統合



システム
シミュレーション

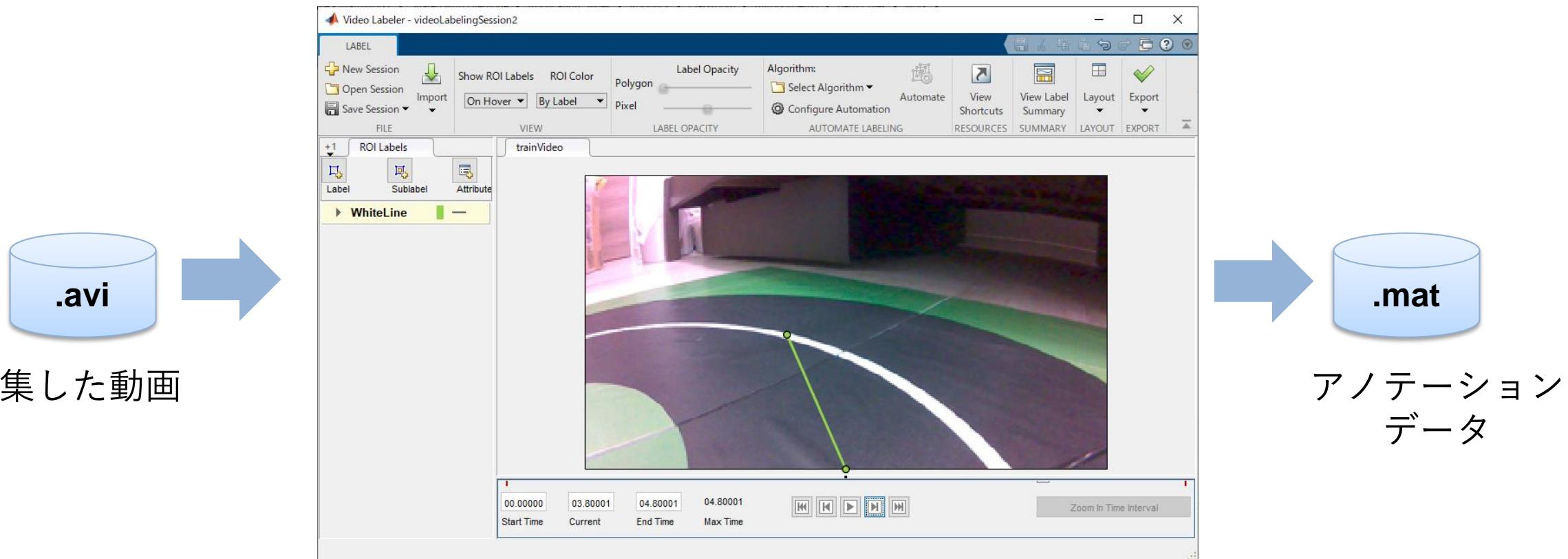


実装

実機実行

ラベリング

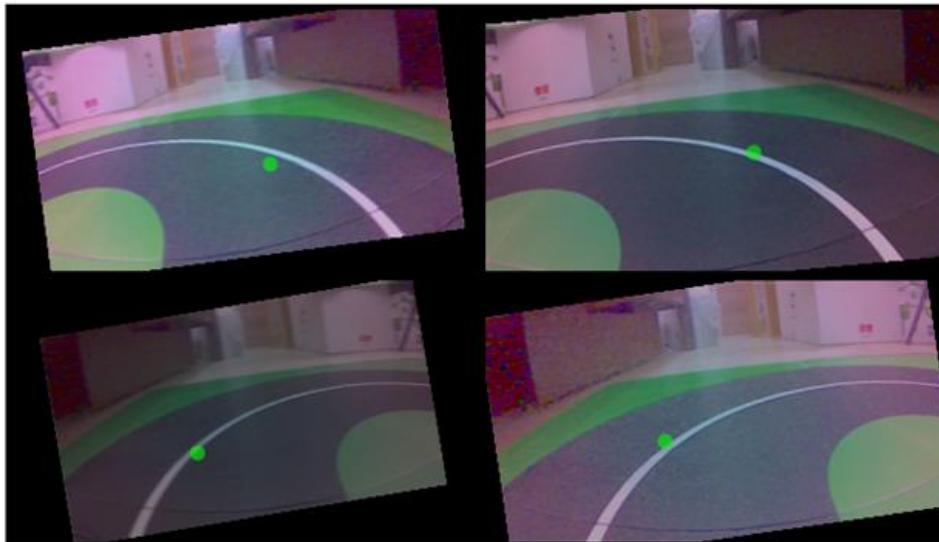
- 収集した画像データをシミュレーションデータ同様にラベリング



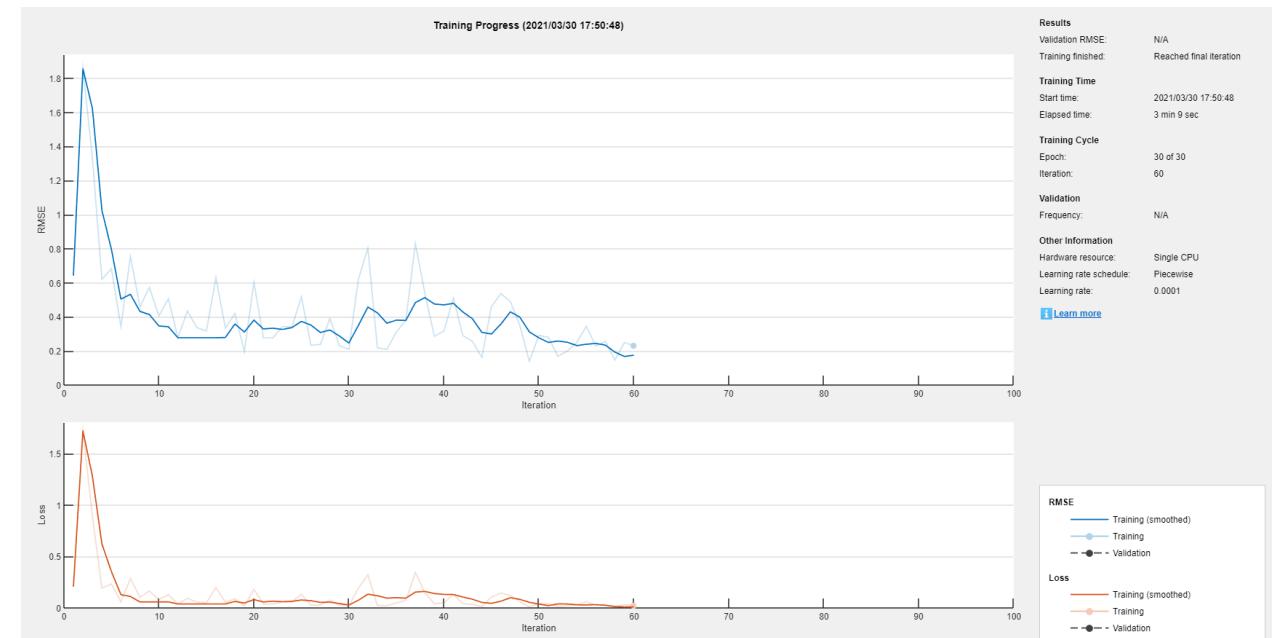
```
>> edit trainWhiteLineWithHardware_jp mlx
```

データの水増しと再学習

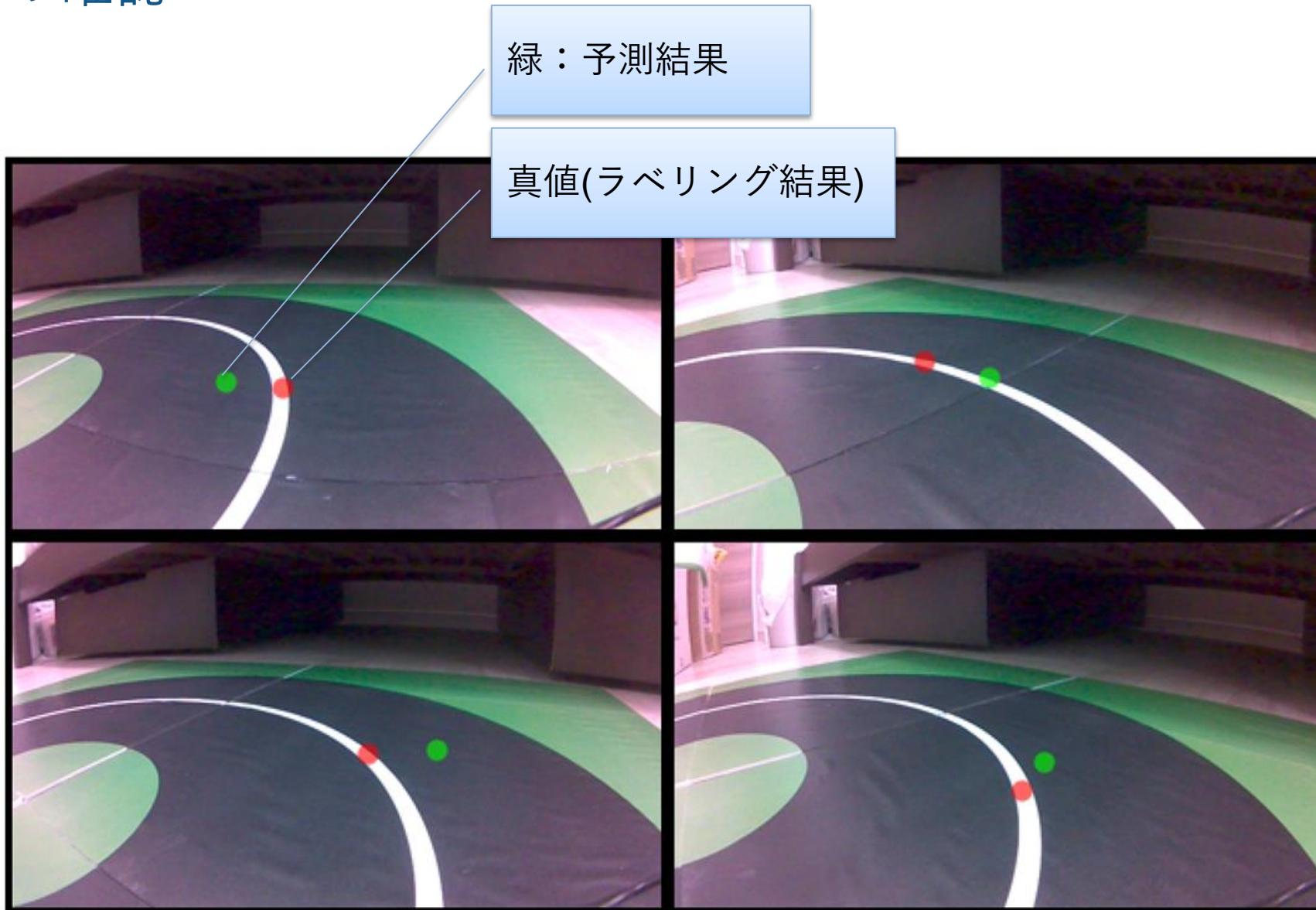
- シミュレーション時と同様に再学習
- 実機の複雑な環境変化を模擬するためオーグメンテーションとして画像を加工



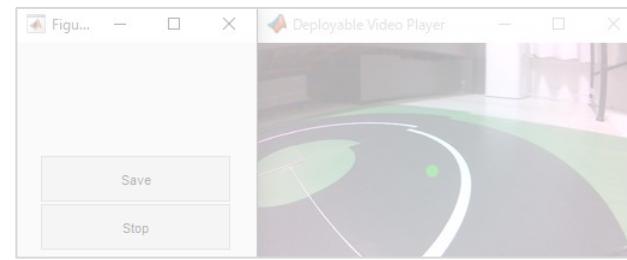
画像を加工し、環境変化を模擬



学習結果の確認



ステップ3: ライントレースアルゴリズムを実機テスト



データセットの
準備

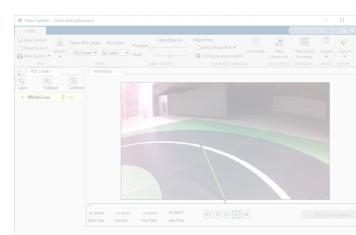


動画

ラベリング

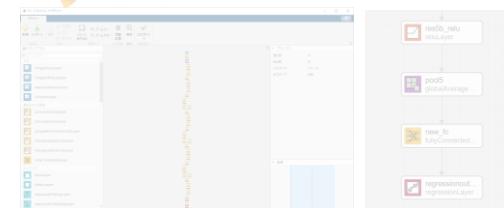


ラベル済み
画像



ビデオラベラー

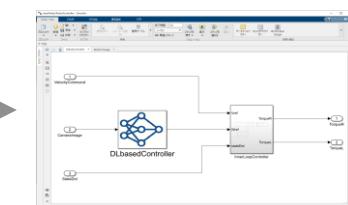
回帰ネットワークモデル



ネットワークの設計



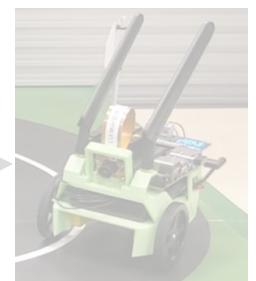
学習済モデル



統合



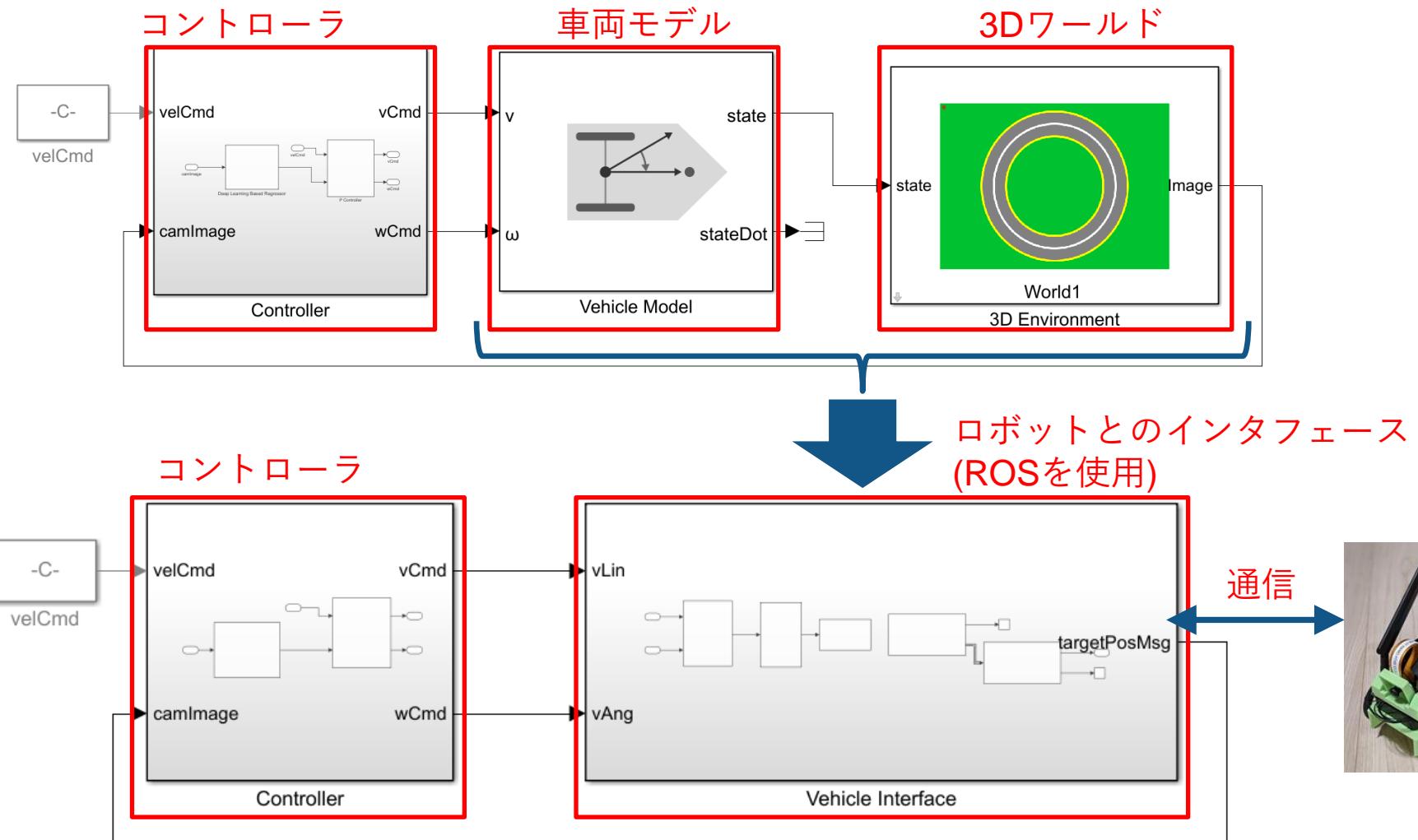
統合検証



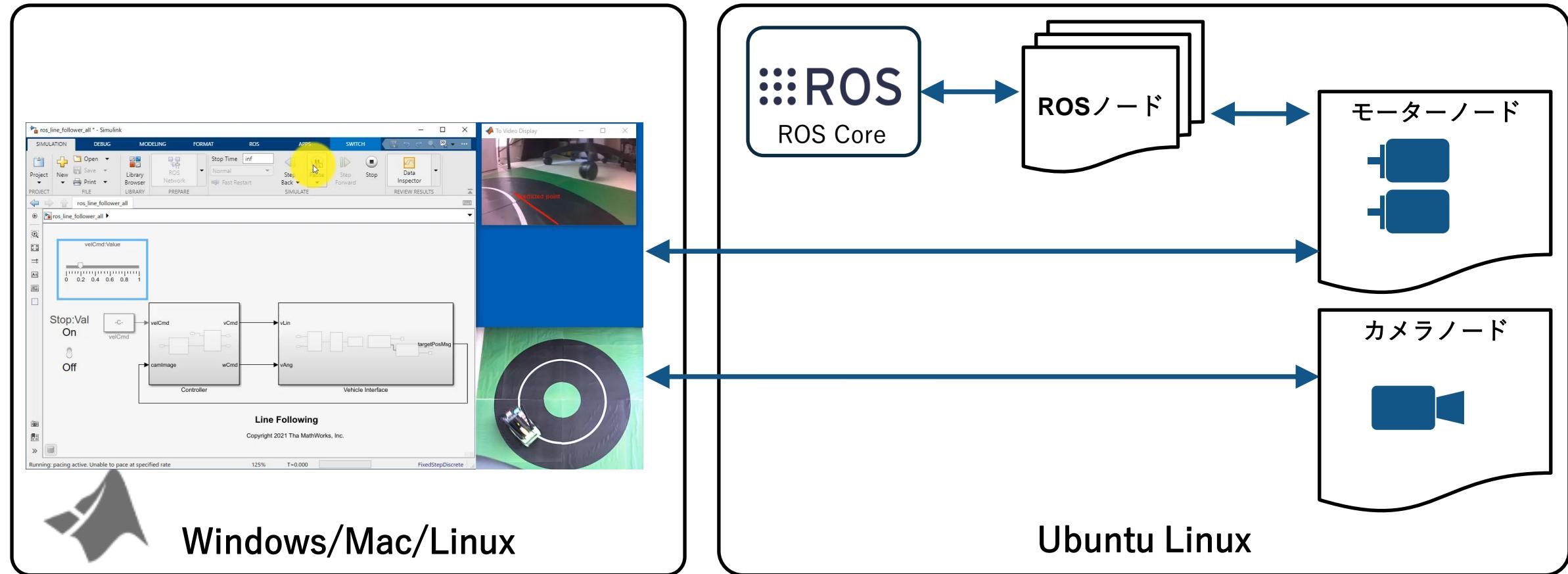
実装

実機実行

インタフェースをROSに置き換え

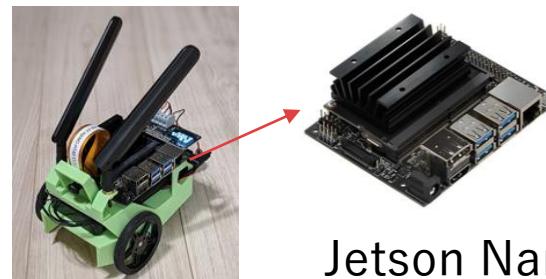


実機通信でアルゴリズムをテスト

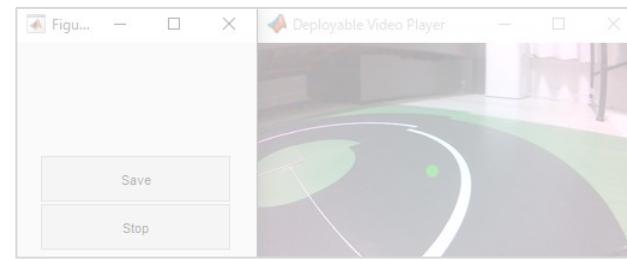


ライン検出および追従制御は
ホストマシン上で実行
(通信遅延あり)

```
>> edit testLineFollowerWithROS_jp.mlx
```



ステップ4: コード生成による実機へのスタンドアローン実装



データセットの準備

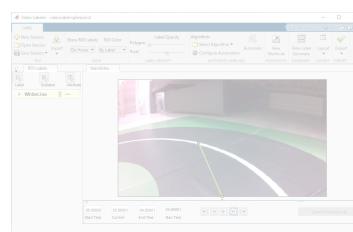


動画

ラベリング

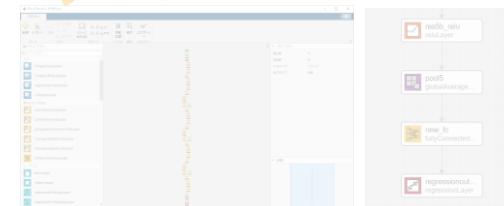


ラベル済み
画像



ビデオラベラー

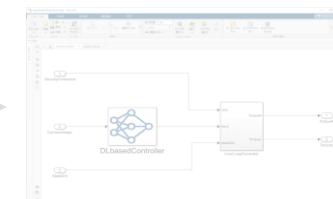
回帰ネットワークモデル



ネットワークの設
計



学習済モデル



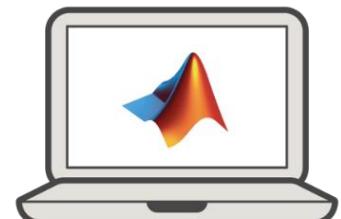
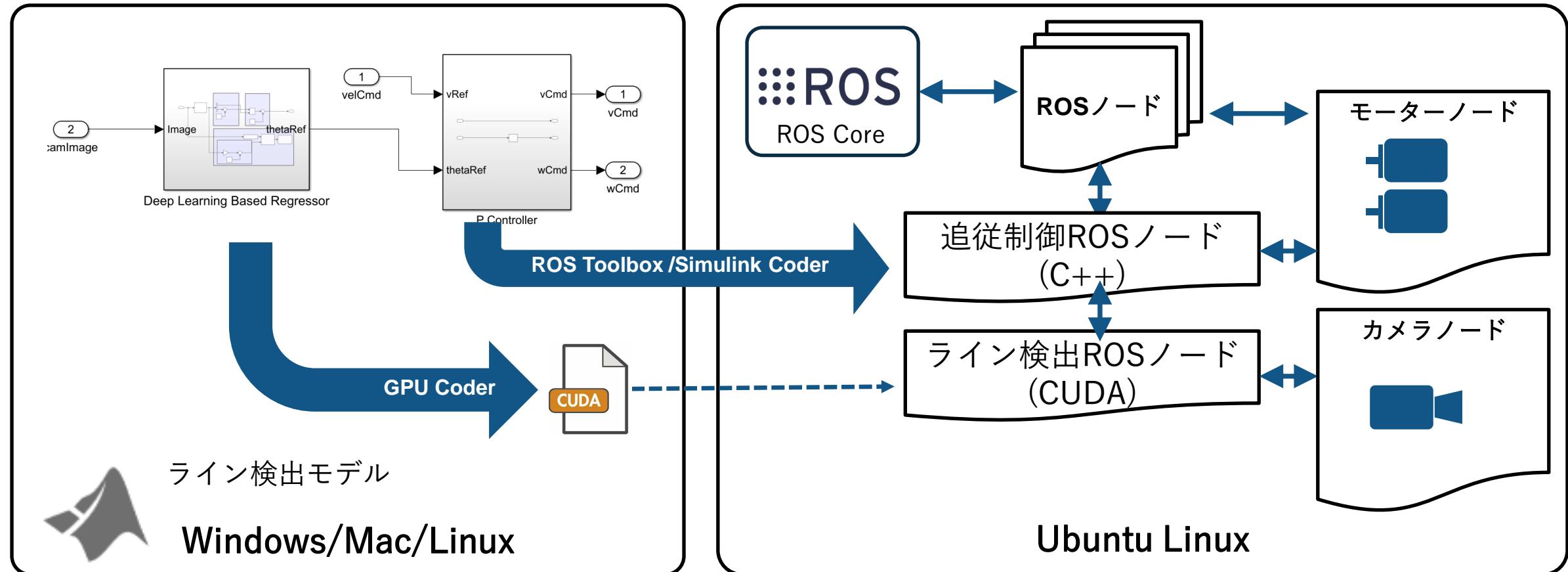
統合検証

統合



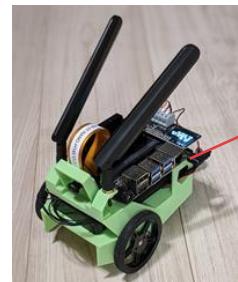
実機実行

ROSノード化し、ターゲット上で動作



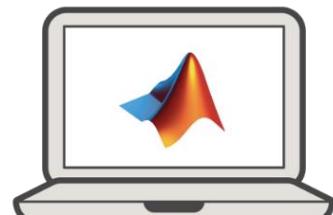
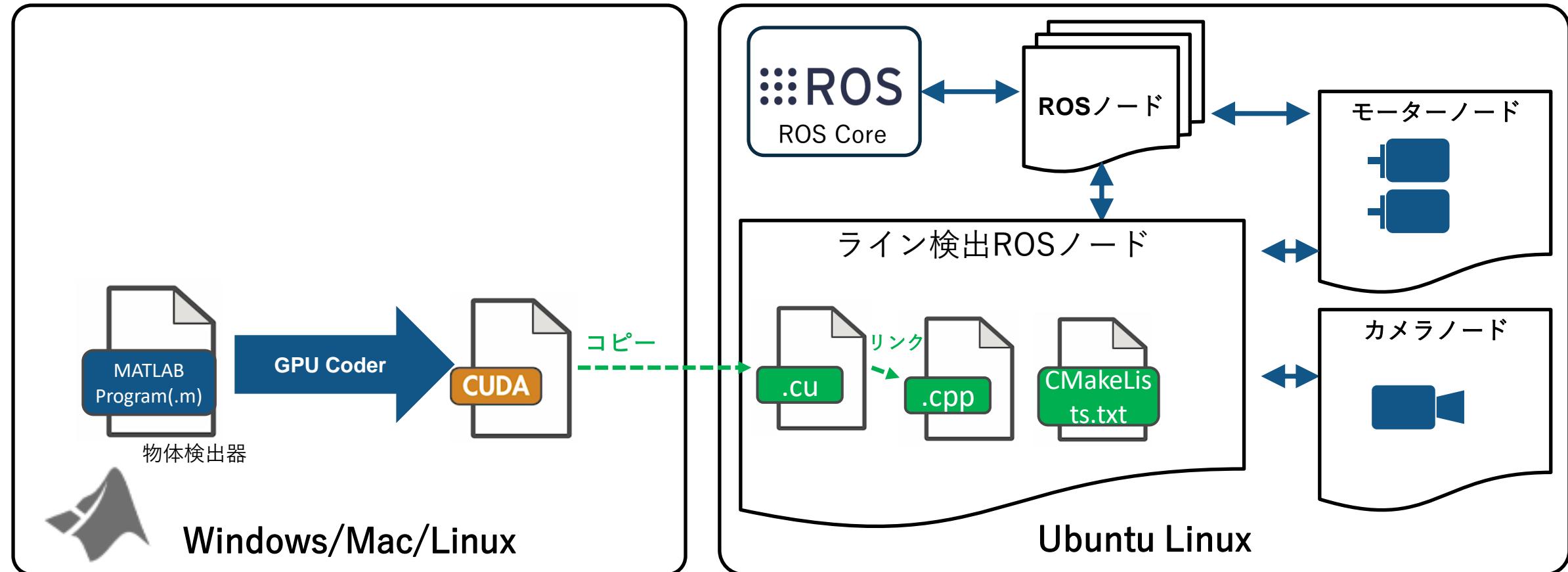
Development Machine

Simulinkで実現したモデルを
ライン検出と追従制御に分割し、そ
れぞれROSノード化
スタンドアローン実行可能にする



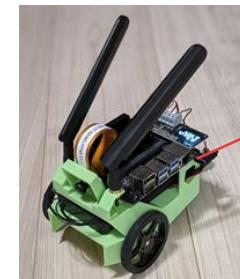
Jetson Nano™

CUDAコード生成とROSノード化



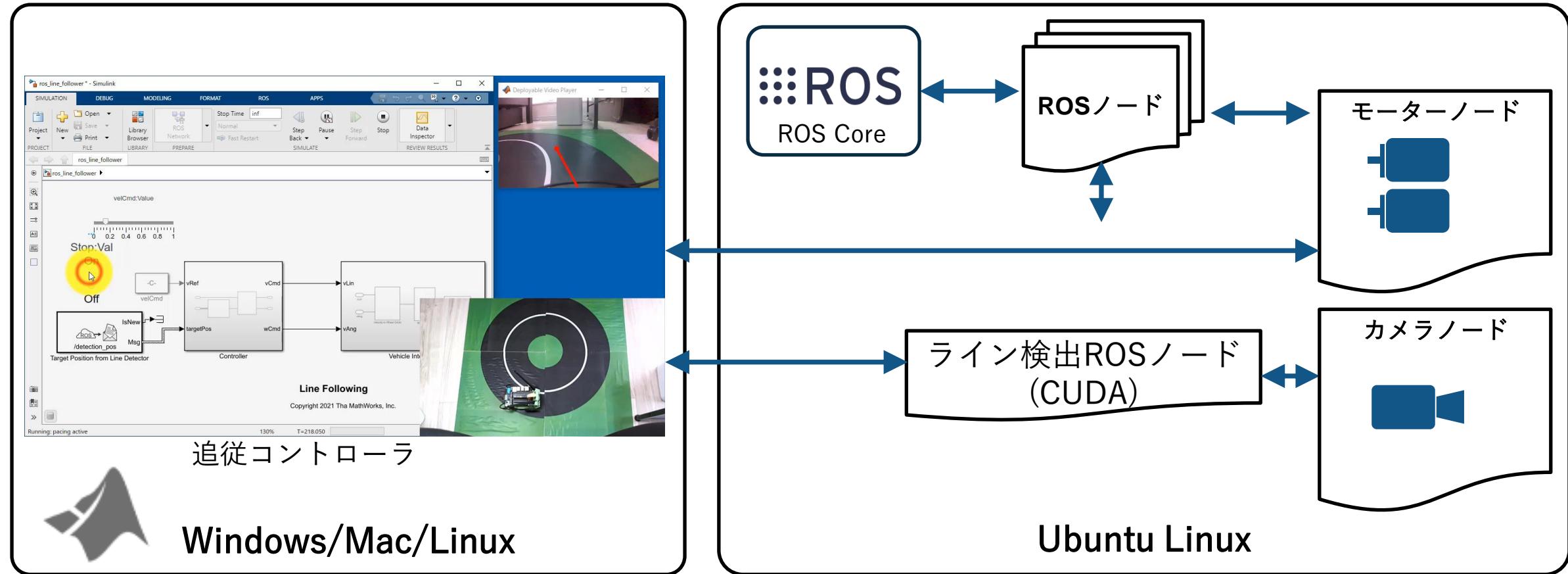
Development Machine

ライン検出モデルについては
GPU Coderを使いCUDAコード生成
その後、ROSノード化

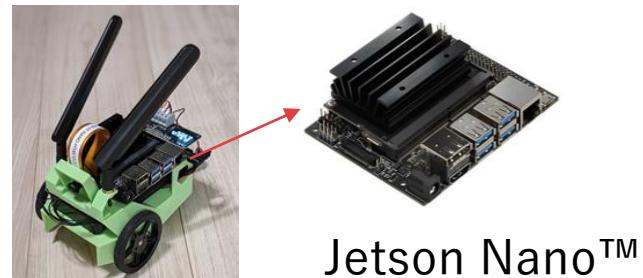


Jetson Nano™

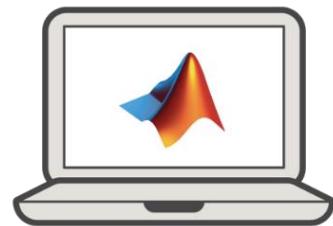
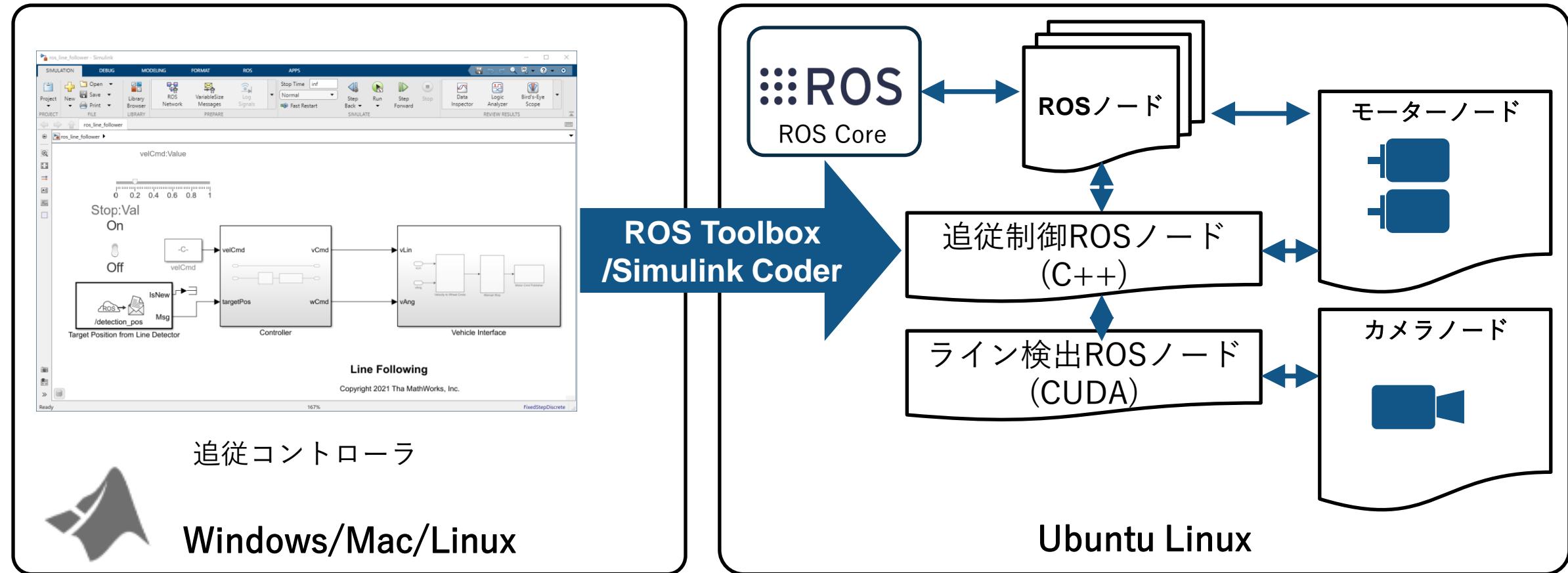
実装したライン検出器と追従コントローラの協調動作確認



分割した追従コントローラと
実装したライン検出ROSノードの動作を確認

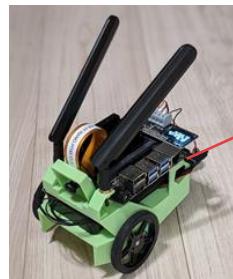


追従コントローラのROSノード化



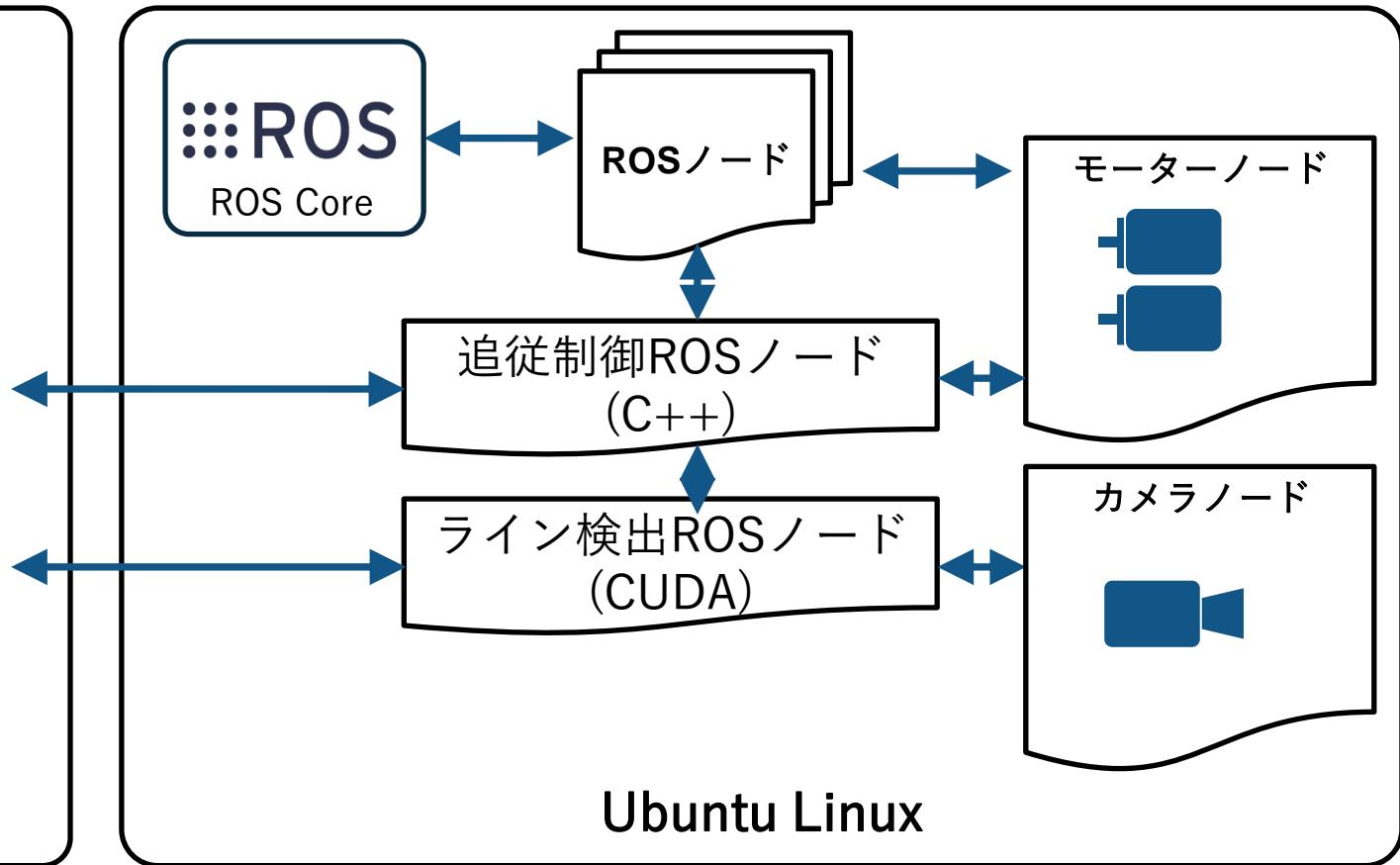
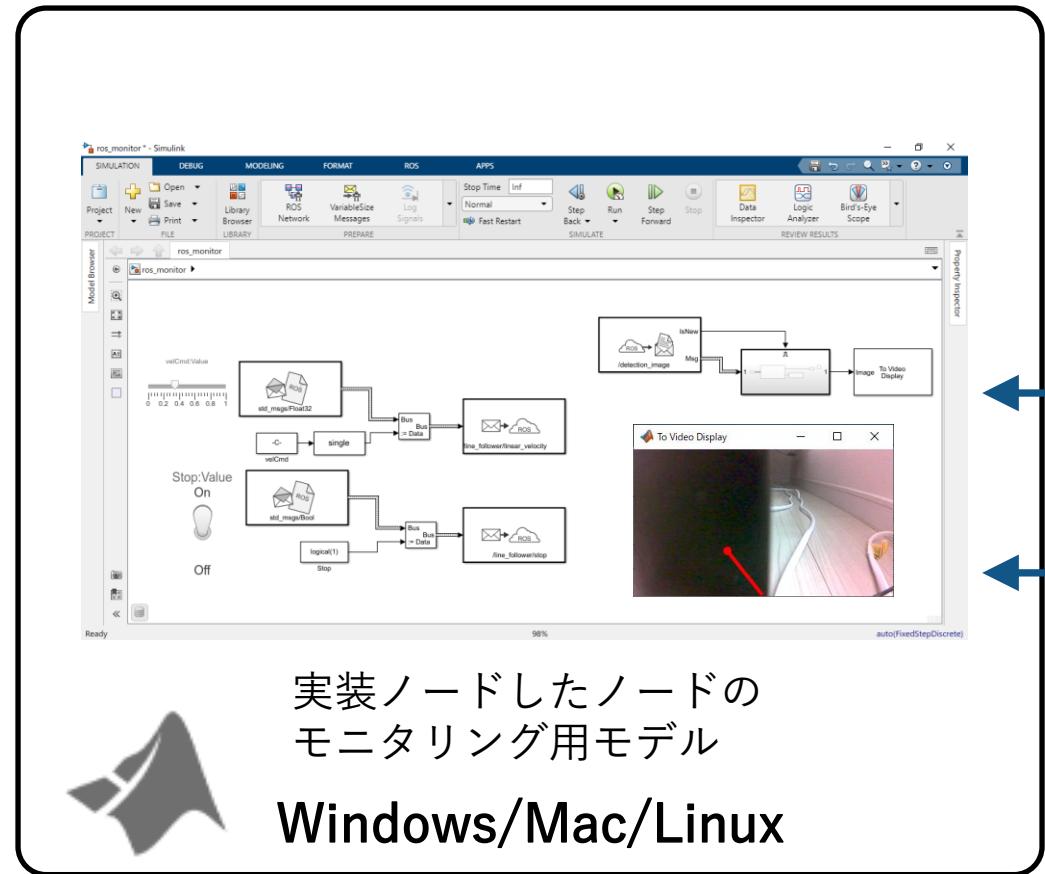
Development Machine

追従コントローラをROS Toolboxと
Simulink Coderを使ってROSノード化

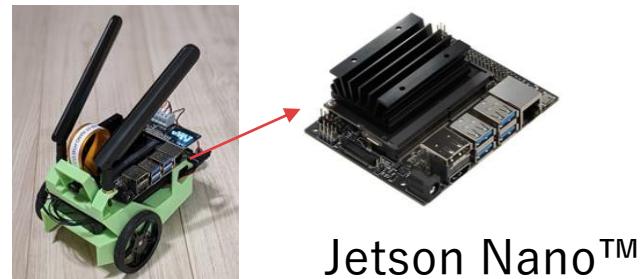


Jetson Nano™

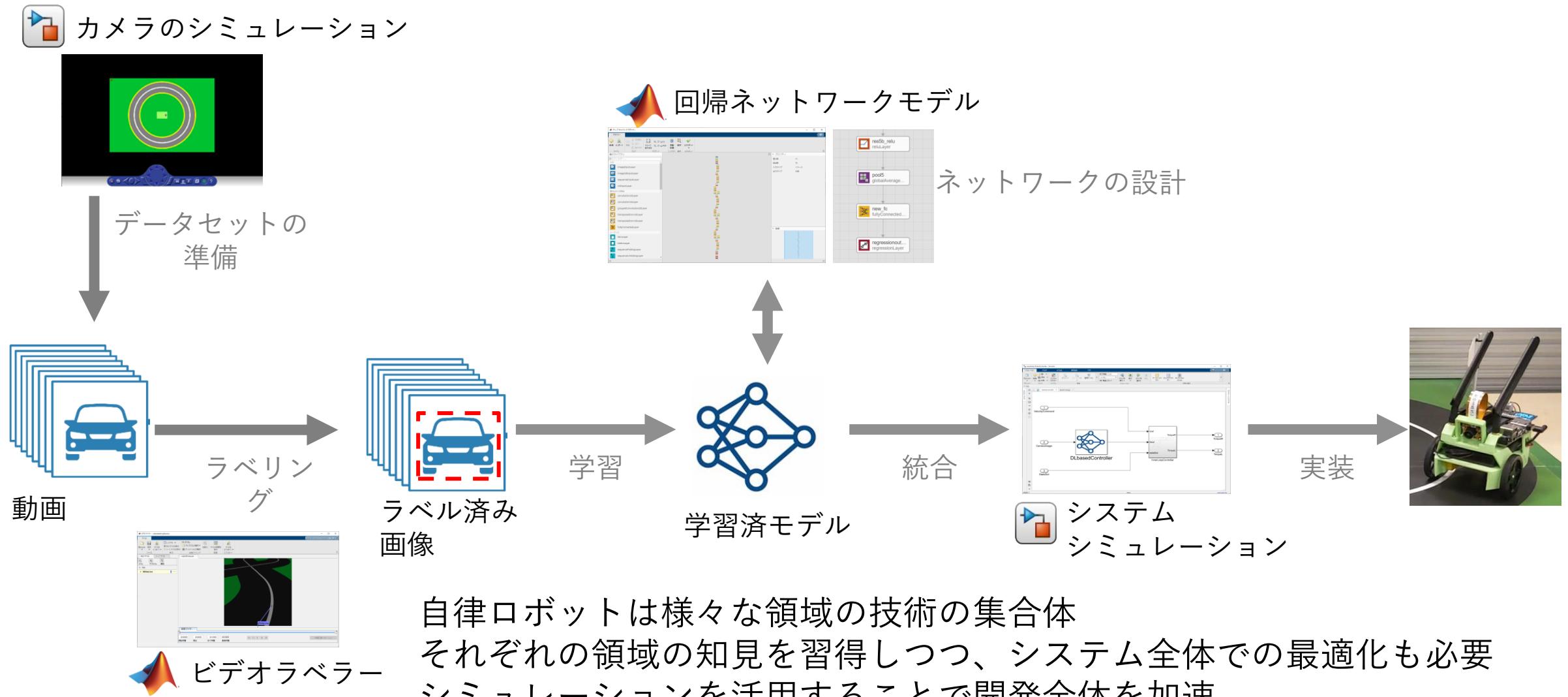
全ROSノードをスタンドアローン実行



実機上でスタンドアローン実行
速度制御や進行停止などの指令は
別のSimulinkモデルを利用



まとめ：自律ロボットシステム開発ワークフロー



付録A. ワークショップ環境準備

ワークショップ用PC

- システム要件
 - <https://jp.mathworks.com/support/requirements/matlab-system-requirements.html>
- 必要なソフトウェア
 - MATLAB R2021a 以降
 - Simulink
 - Image Processing Toolbox
 - Computer Vision Toolbox
 - Deep Learning Toolbox
 - Robotics System Toolbox
 - Simulink 3D Animation
 - Deep Learning Toolbox Model for ResNet-18 Network
 - 下記のようにMATLAB上でタイプいただきますとインストールのリンクが表示されます
 - `>> resnet18`
 - Parallel Computing Toolbox (NVIDIA® GPUが使える場合)

付録B. 実機実装 (JetBotのセットアップ)

開発環境：MathWorks製品

- シミュレーションに加えて以下のソフトウェアが必要
- Toolbox
 - MATLAB Coder
 - GPU Coder
 - Simulink Coder
 - ROS Toolbox
- アドオン
 - MATLAB Coder Support Package for NVIDIA® Jetson™ and NVIDIA® DRIVE™ Platforms
 - インストール手順は下記参照
 - <https://www.mathworks.com/help/supportpkg/nvidia/ug/install-support-for-nvidia-devices.html>

開発環境：ハードウェア関連

- ハードウェア
 - NVIDIA® JetBot ([FaBo製 JB-4GB-S-G](#)でテスト済)
 - Jetson Nano™ 4GB
- ソフトウェア
 - JetPack 4.3, JetBot 0.4.0 ([jetbot_image_v0p4p0.zip](#))
 - L4T R32.3.1 (K4.9)
 - Ubuntu 18.04 LTS aarch64
 - CUDA 10.0
 - cuDNN 7.6.3
 - TensorRT 6.0.1
 - OpenCV 4.1
 - JetBot ROSパッケージ
 - https://github.com/dusty-nv/jetbot_ros
 - リビジョン c7d0e611b037947a0df855293c9848a7c8bc6e90 にパッチをしてテスト
 - ROS Melodic + JetBot用のカメラ/モーターのROSドライバノード
 - Jetson CSIカメラのROS nodeletパッケージ
 - https://github.com/sfalexrog/jetson_camera
 - リビジョン 7ec8e364880fce5bbe31800bd6224af2370ca1b でテスト

JetBotのセットアップ (1/3)

- 下記に沿ってJetBotのソフトウェアをセットアップする
 - https://jetbot.org/v0.4.3/software_setup/sd_card.html
 - このときJetBotのイメージファイルとして jetbot_image_v0p4p0.zip を使用する
- ROS環境をセットアップする
 - 下記の手順に沿ってセットアップ
 - https://github.com/dusty-nv/jetbot_ros
 - jetbot_rosを使用する際は動作確認済みのリビジョンを使う
 - \$ git clone https://github.com/dusty-nv/jetbot_ros
 - \$ git checkout c7d0e611b037947a0df855293c9848a7c8bc6e90
- ROSの環境変数を設定
 - \$ 'export ROS_IP=\$(hostname -I | awk '{print \$1;}' | tr -d [:blank:])' >> ~/.bashrc
 - \$ 'export ROS_MASTER_URI=http://\$ROS_IP:11311' >> ~/.bashrc

JetBotのセットアップ (2/3)

- jetbot_rosにパッチを当てる
 - 上記のjetbot_rosではモーターの速度制御が実装されていない
 - モーターの速度制御ができるようにパッチを当てる
 - setupフォルダのjetbot_ros_for_c7d0e61.patchをJetBotにコピーする
 - ```
$ cd /home/jetbot/workspace/catkin_ws/src/jetbot_ros
```
  - ```
$ patch -p1 < ~/jetbot_ros_for_c7d0e61.patch
```
- Jetson CSIカメラのROS nodeletパッケージをセットアップ
 - ```
$ cd /home/jetbot/workspace/catkin_ws/src/
```
  - ```
$ git clone https://github.com/sfalexrog/jetson\_camera.git
```
 - ```
$ cd /home/jetbot/workspace/catkin_ws
```
  - ```
$ catkin_make
```
 - OpenCV3が見つからないというエラーが出る場合は下記の"3"を削除
(JetPack4.3のデフォルトがOpenCV4のため)
 - ```
$ gedit ~/workspace/catkin_ws/src/jetson_camera/CMakeLists.txt
```

```
find_package(OpenCV 3 REQUIRED)
```

## JetBotのセットアップ (3/3)

- MATLABからの接続確認
  - MATLAB R2021aを起動
  - `>> cd c:\Yai-robotics-workshop`
  - `>> matlab_ai_robotics_workshop.prj`
  - `>> edit teleopTestJetBot_jp.mlx`
- JetBotが動き、画像が取得できれば動作確認完了