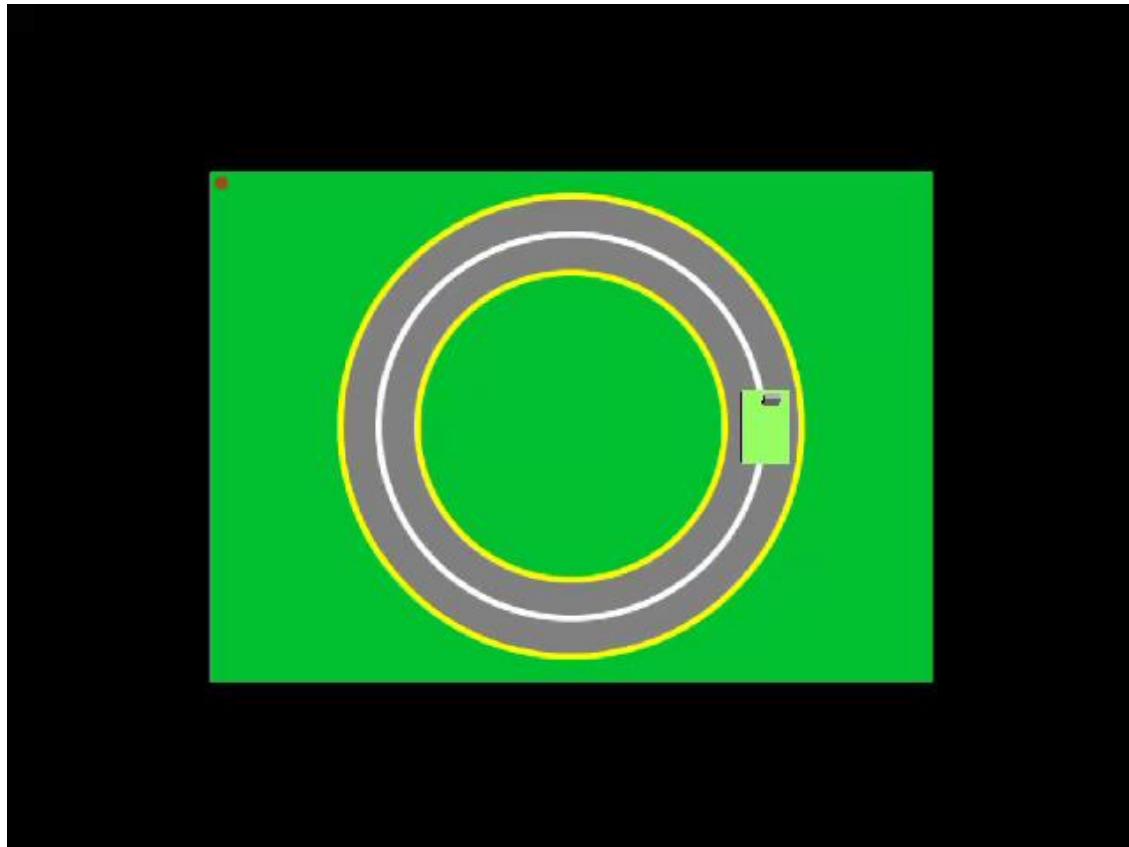


# MATLAB® AI Robotics Workshop

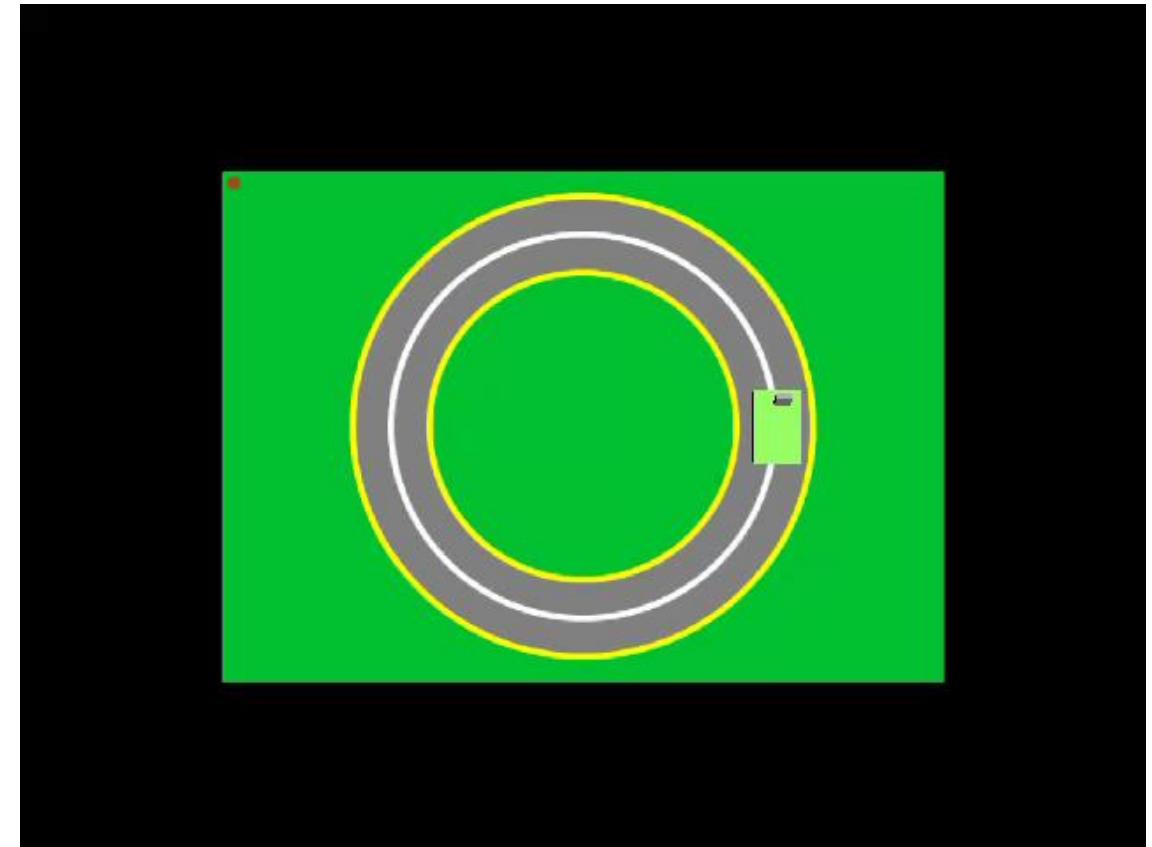
## Line Following Robot with Deep Learning

MathWorks

# Demo: Line Following Robot with Deep Learning



Failure case

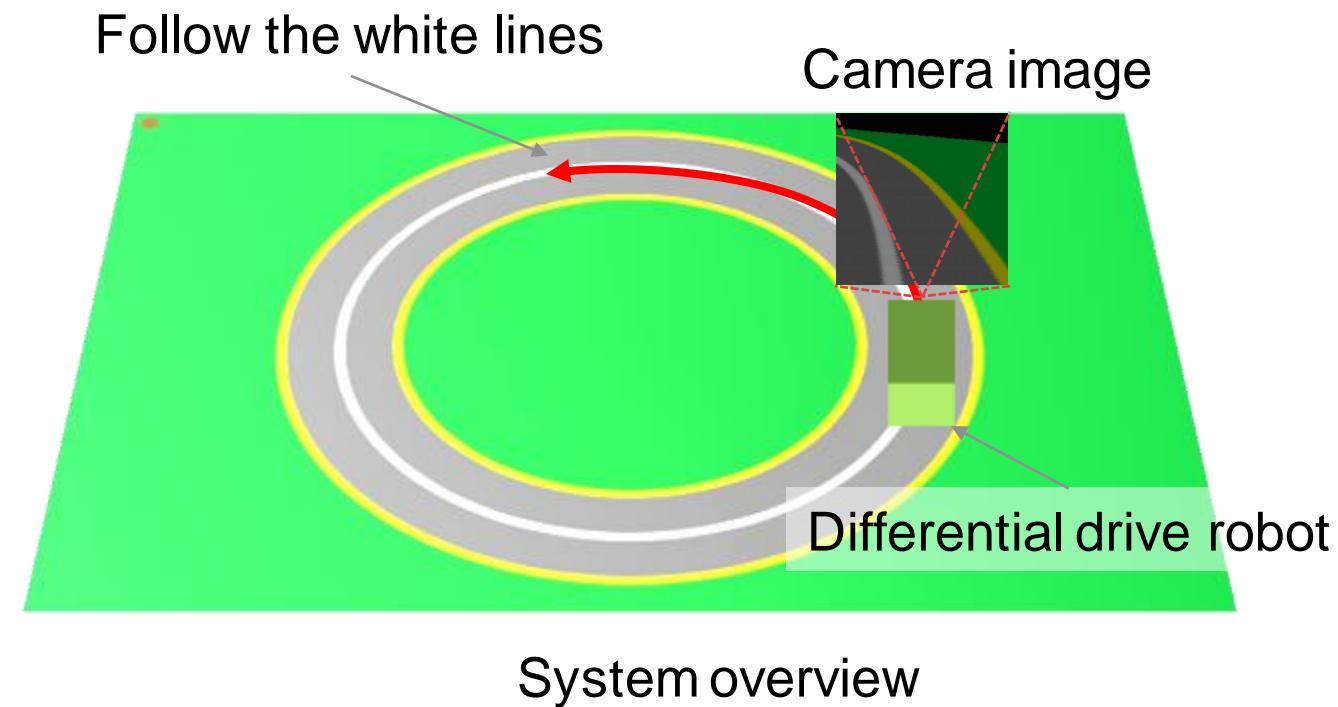


Success case

# Objectives

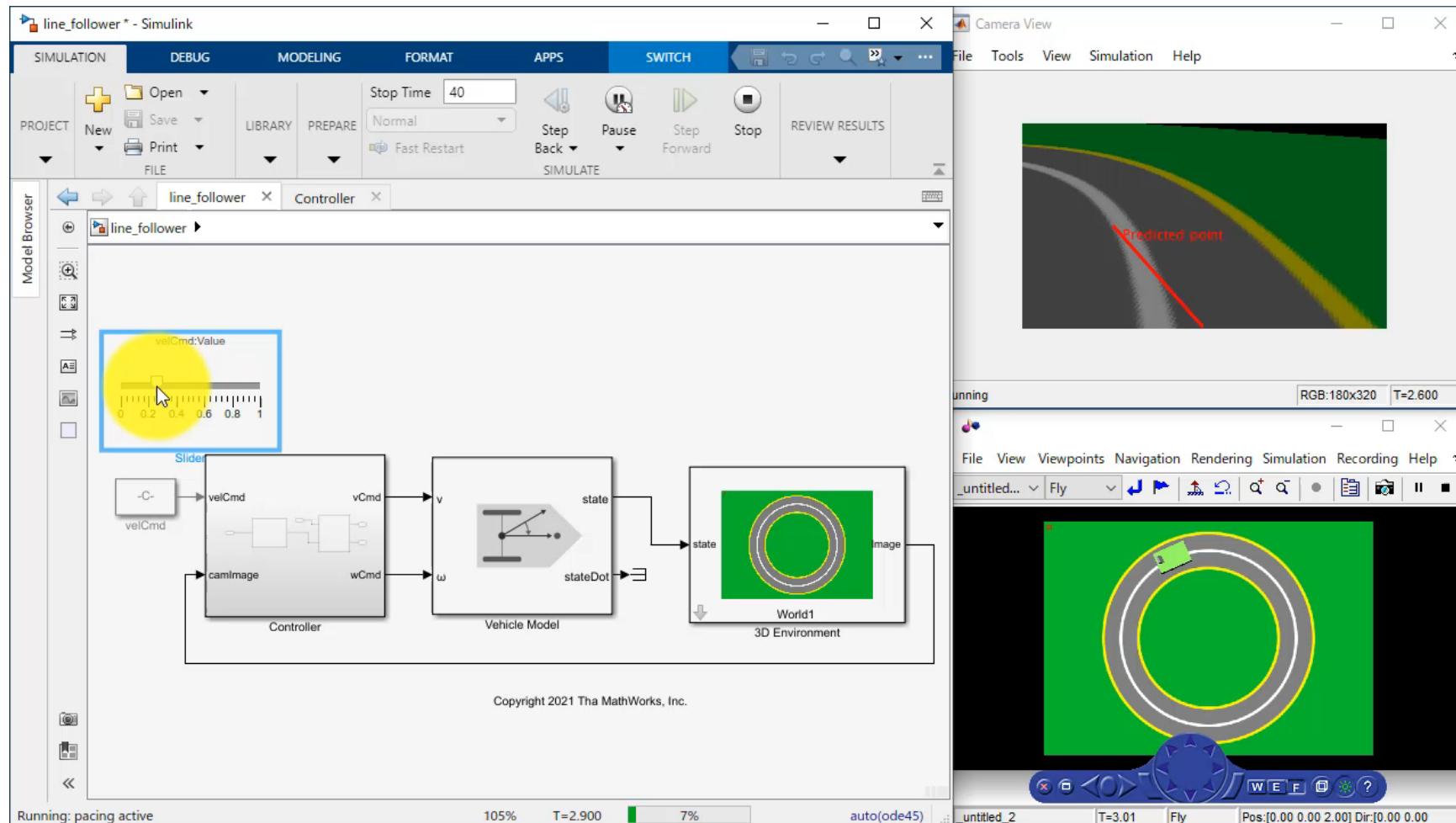
**Goal:** Learn how to implement an AI based autonomous system combining deep learning-based object detection and classic control algorithms

- What's line following?
  - Robot uses a sensor to detect lines on the floor and move along the lines.
- Challenge in line following
  - Rule-based image processing algorithms for line detection is not robust for variation of environmental lighting.
- Solution with deep learning
  - Robust line detection against changing of lighting conditions
  - Need additional tasks
    - Collect bunch of images
    - Annotation
    - Explore training hyper parameters



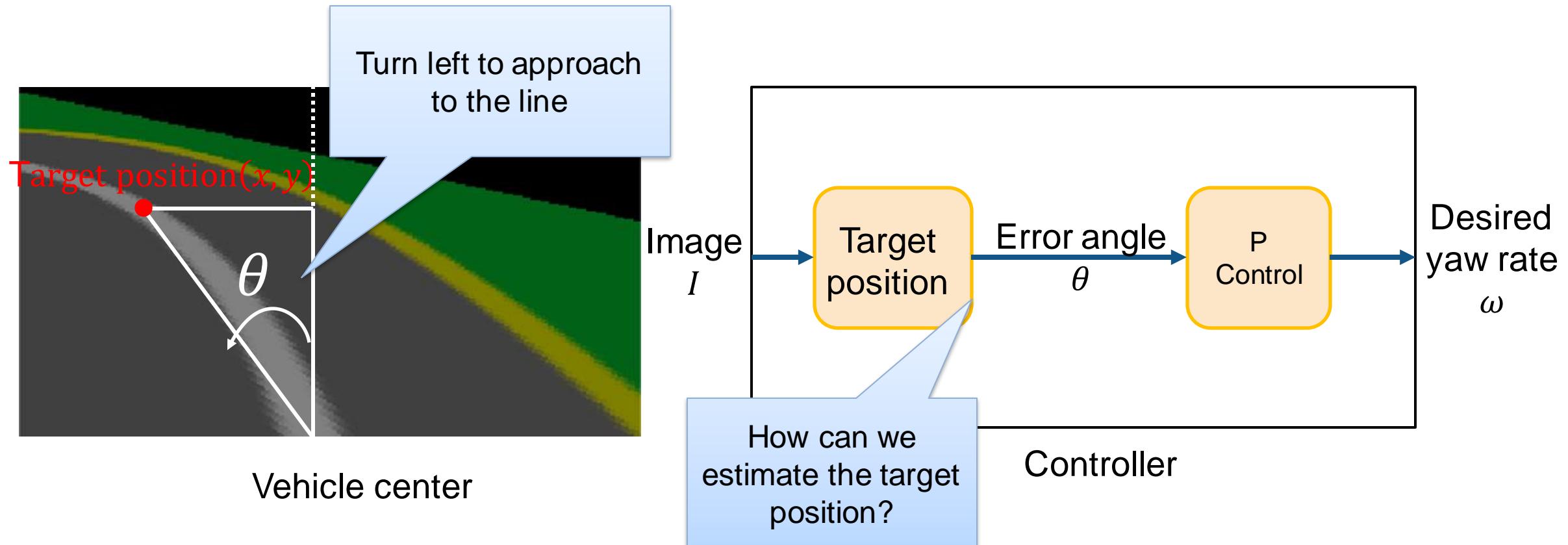
# Simulink Model for Line Following

Goal: Line following with monocular camera image



# Concept of Line Following

- Calculate an error angle  $\theta$  between the ego-vehicle center and target position
- Control the steering to make the error angle to be zero



# Predict Target Position using Deep Learning

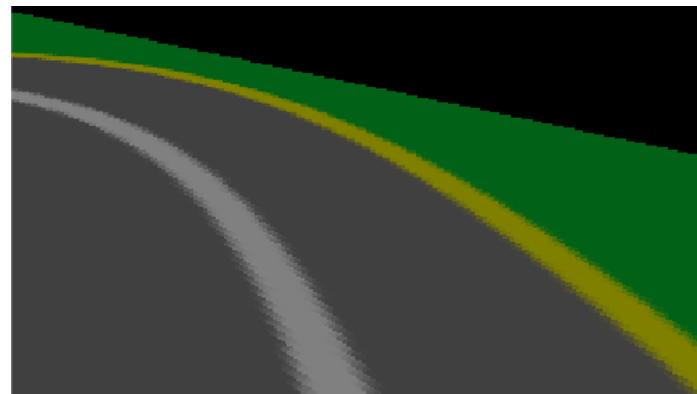
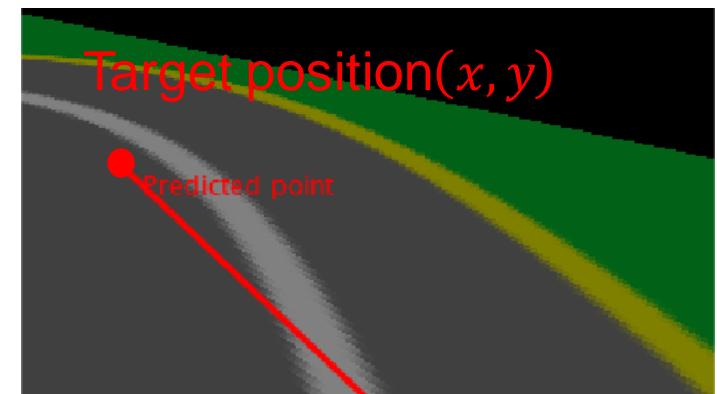


Image from camera  
( $224 \times 224 \times 3$ )



Deep Neural Network



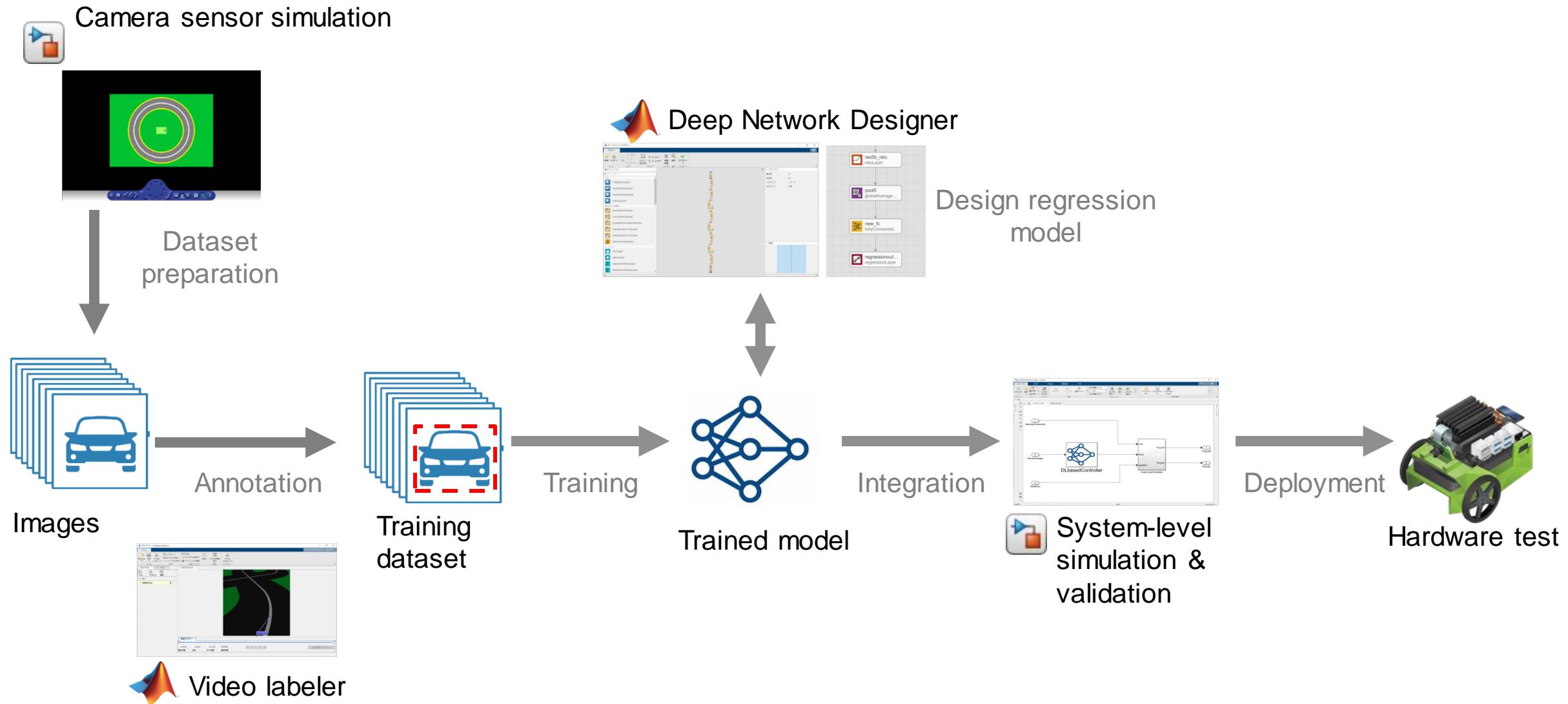
Target position( $x, y$ )  
( $1 \times 2$ )

Predict target position using a deep neural network model

Why deep learning?

- Not easy to extract low-dimensional features from high-dimensional data like images using classical image processing and machine learning
- Deep learning is flexible and can handle those high-dimensional data

# AI-based Autonomous System Development Workflow



# Workshop preparation

1. Launch "MATLAB R2021a"



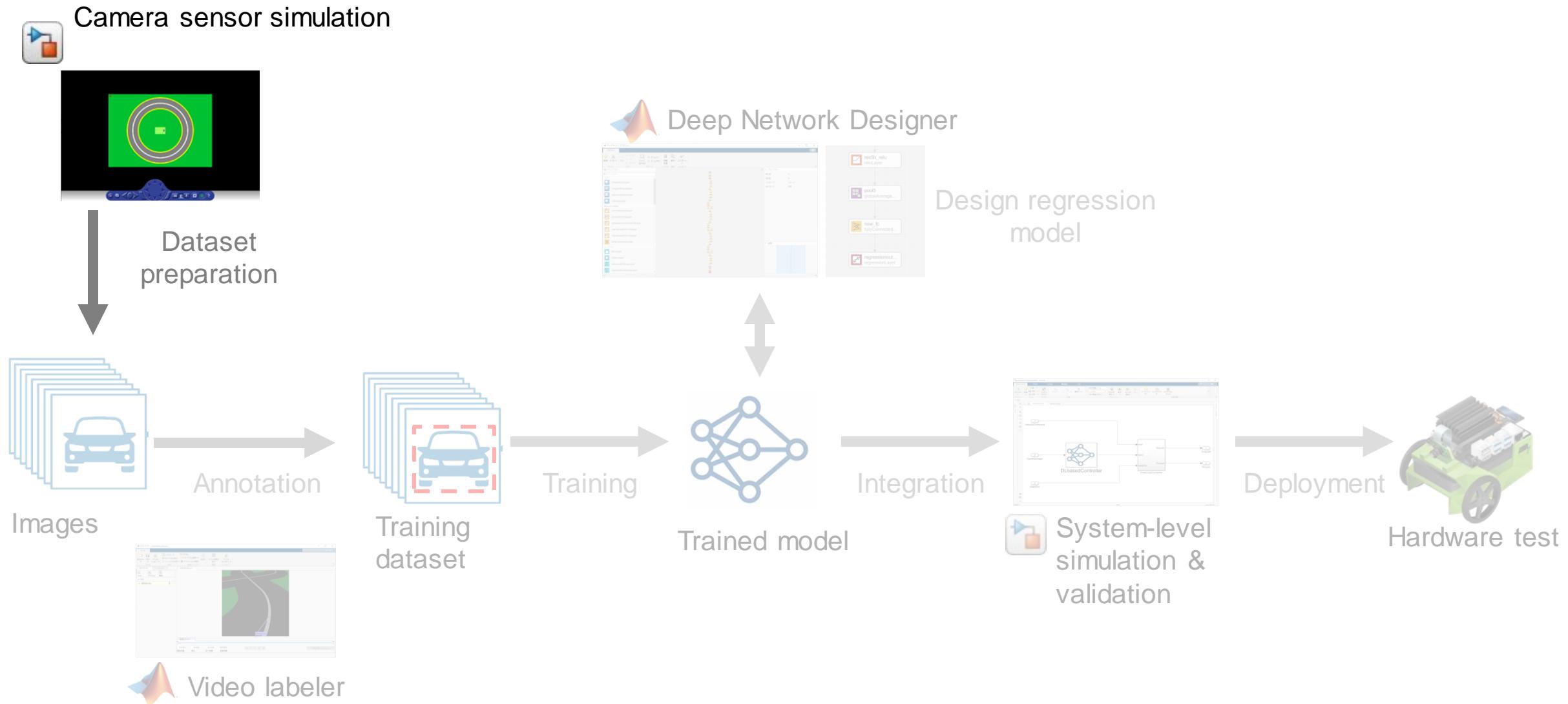
2. Move to the demo folder

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

3. Open the project file

```
>> ai_robotics_workshop.prj
```

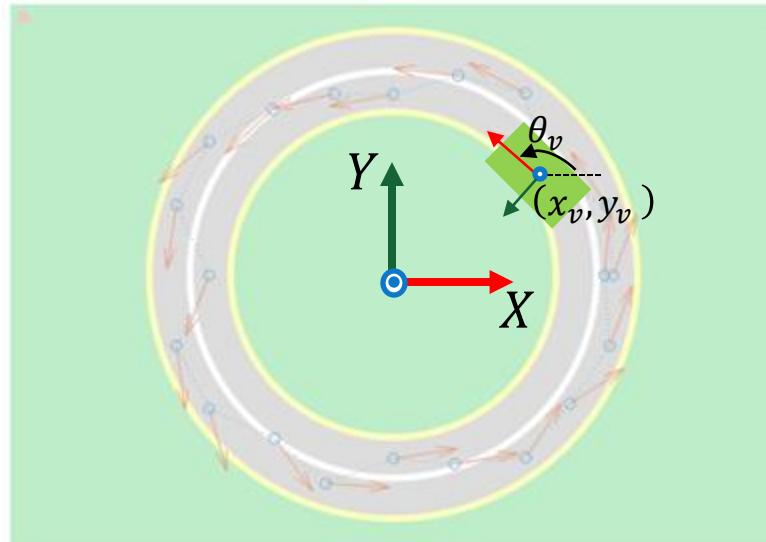
# AI-based Autonomous System Development Workflow



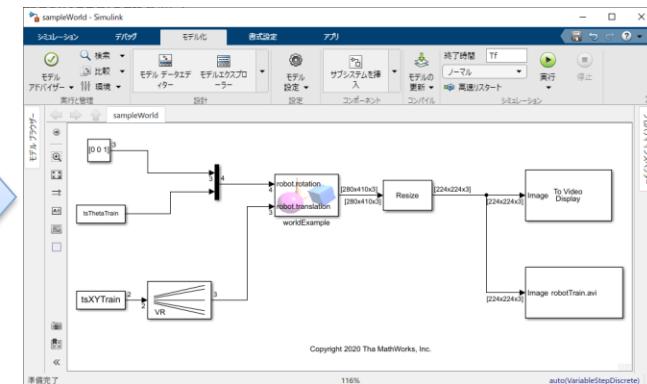
# Generating Video Dataset using Simulation Model

- Deep neural networks require bunch of training dataset including varied camera angles
- Generate vehicle positions randomly and then generate training dataset

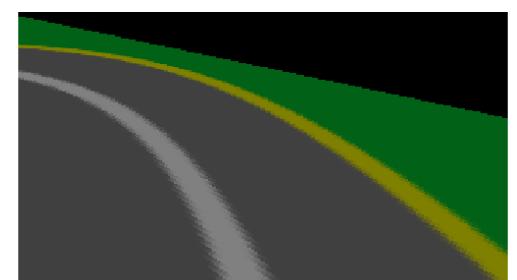
Generate vehicle position and orientation  $(x_v, y_v, \theta_v)$  randomly.



Camera sensor simulation



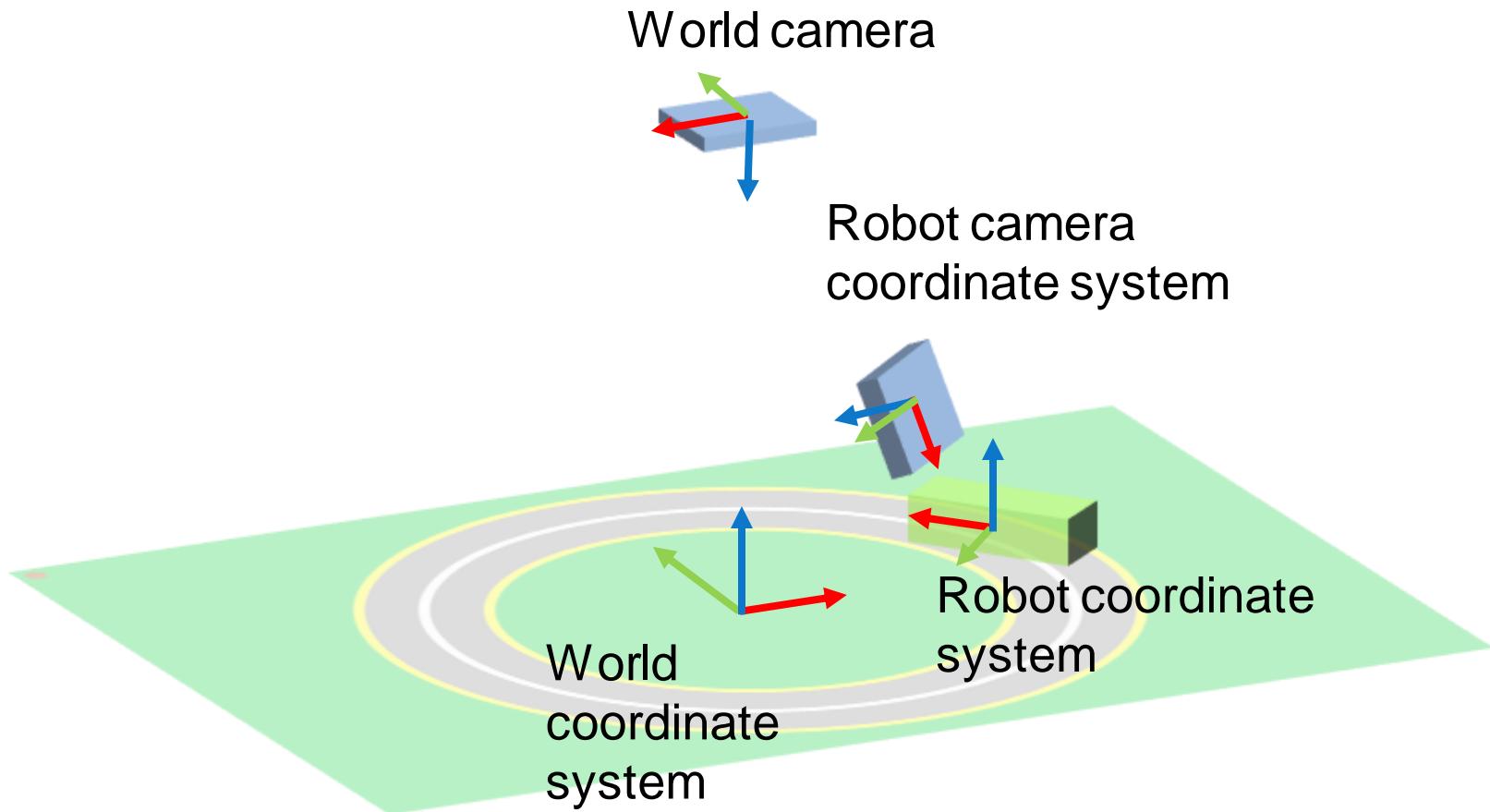
Generate as a video file



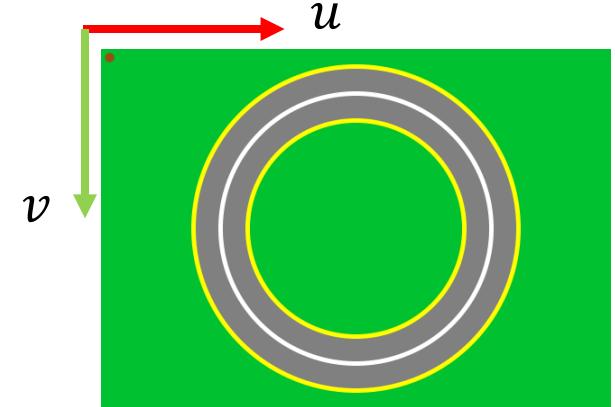
**robotTrain.avi**

# Coordinate System

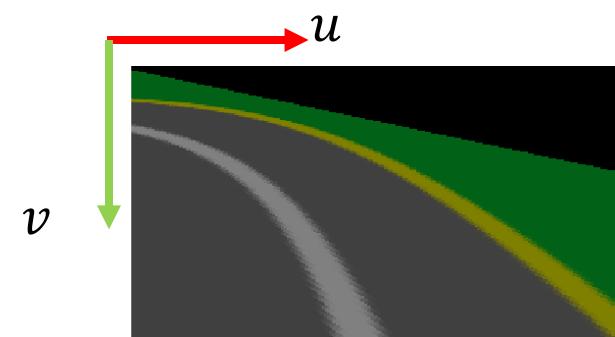
## 3-D coordinate systems



## Image coordinate system



World camera

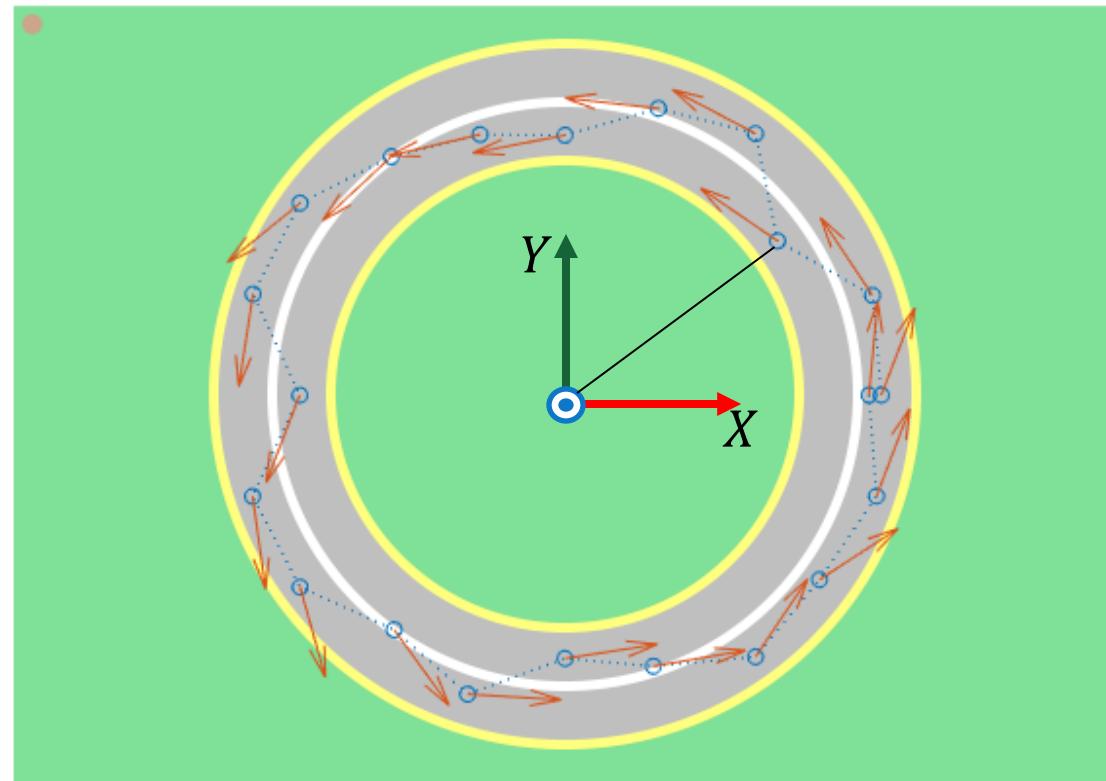


Robot camera

# Exercise 1. Generate Video File

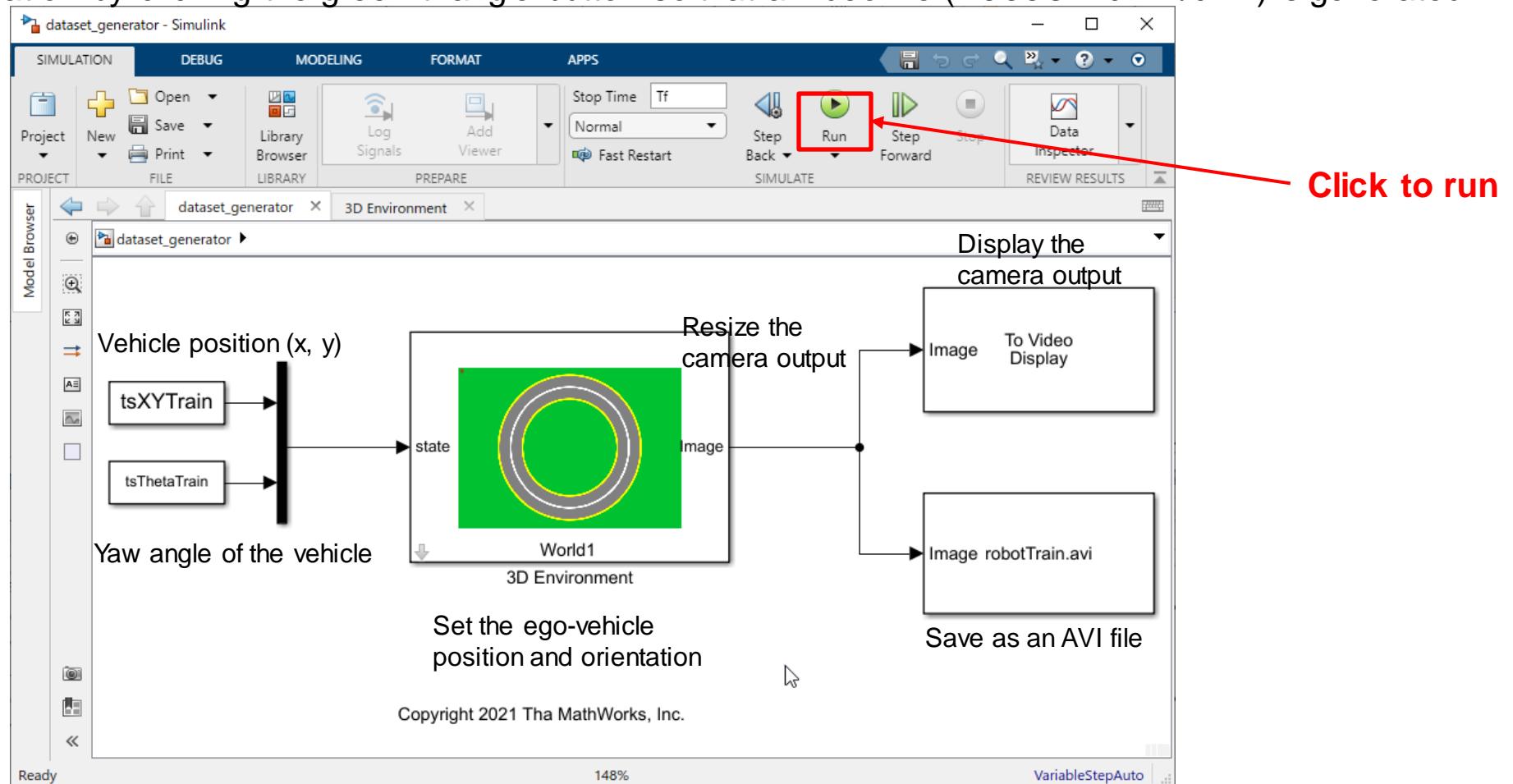
1. Open **genCourseTraj.mlx** and run it to generate trajectories.

```
>> edit genCourseTraj .mlx
```

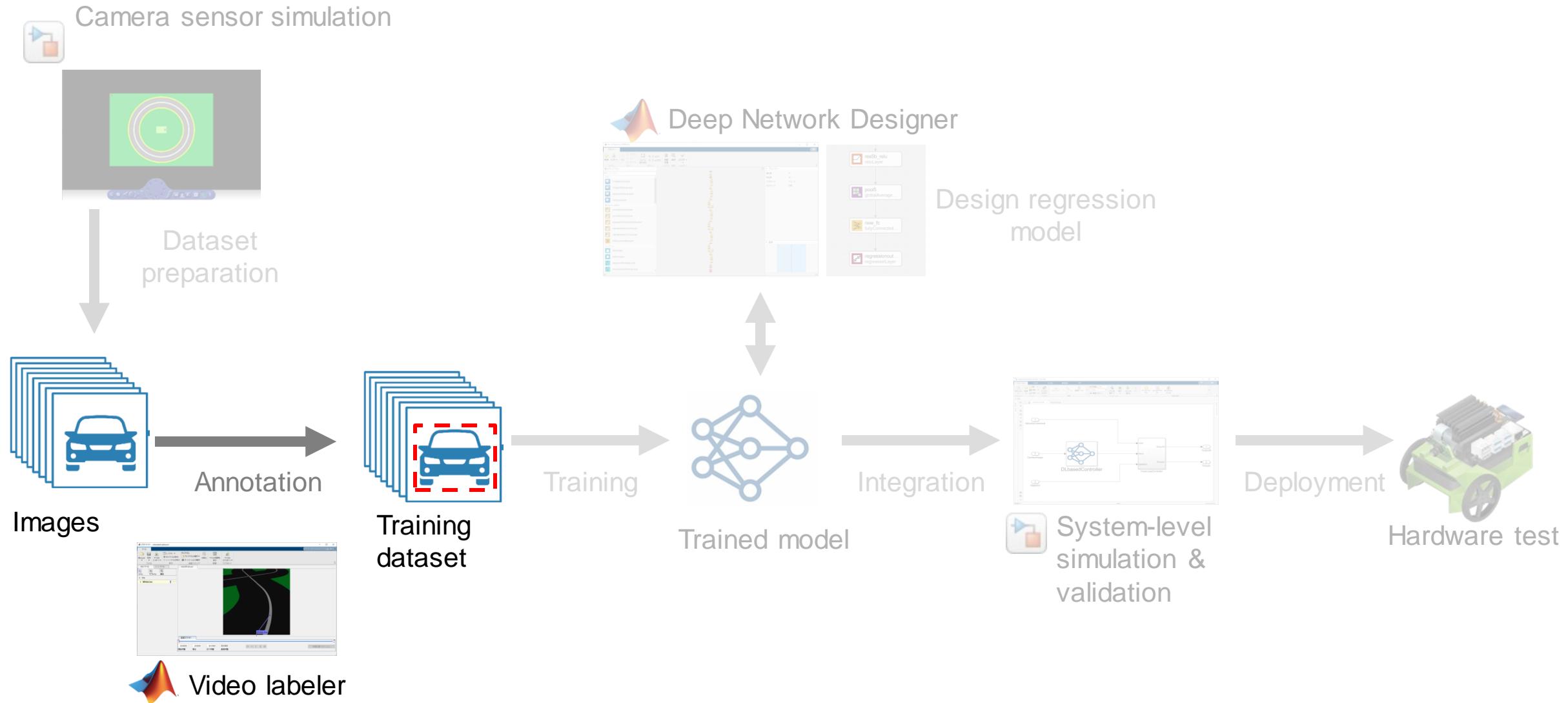


# Exercise 1. Generate Video File

1. Open sampleWorld.slx and simulate the camera sensor outputs  
`>> dataset_generator.slx`
2. Run the simulation by clicking the green triangle button so that a video file (`robotTrain.avi`) is generated.

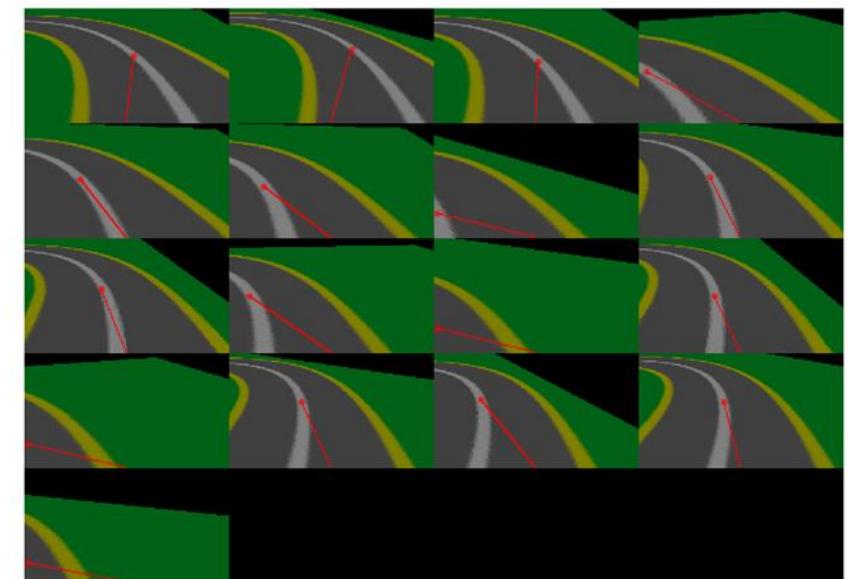
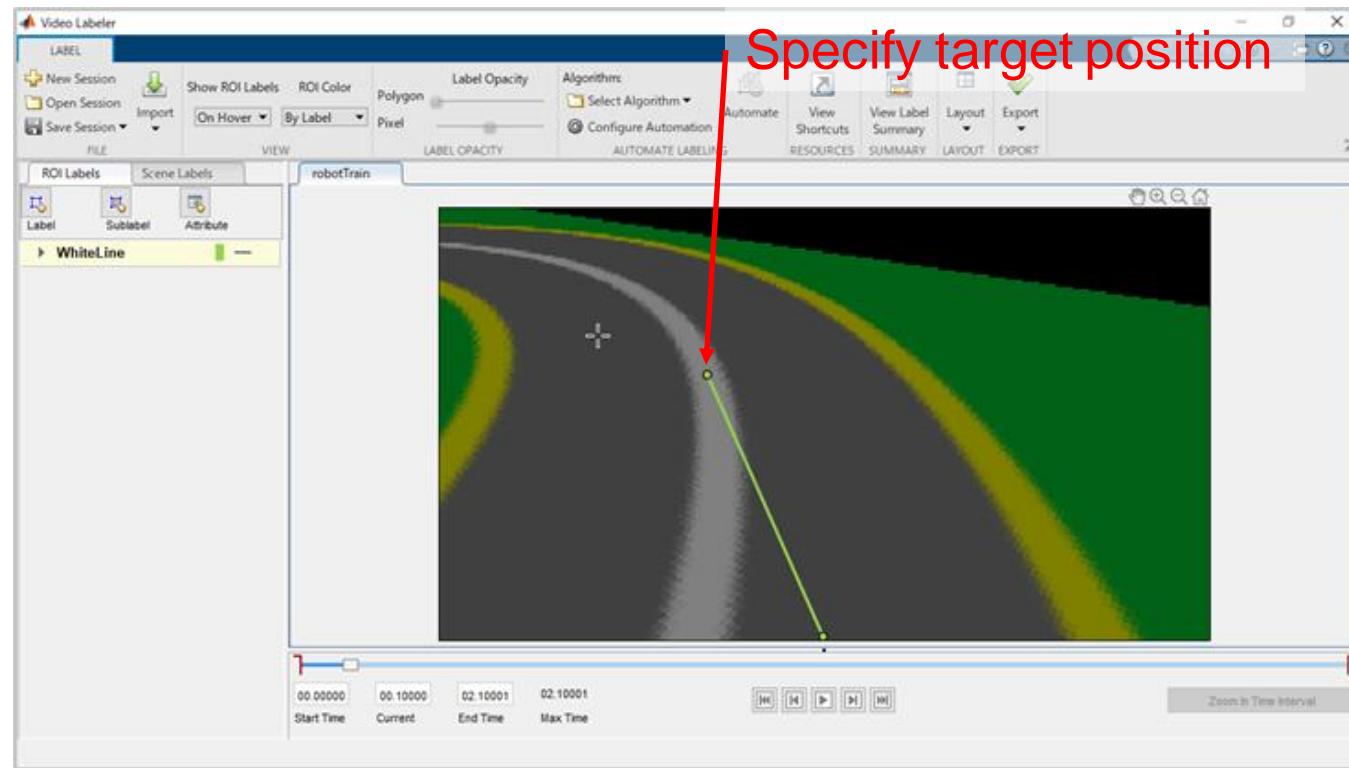


# AI-based Autonomous System Development Workflow



# Annotating Video File

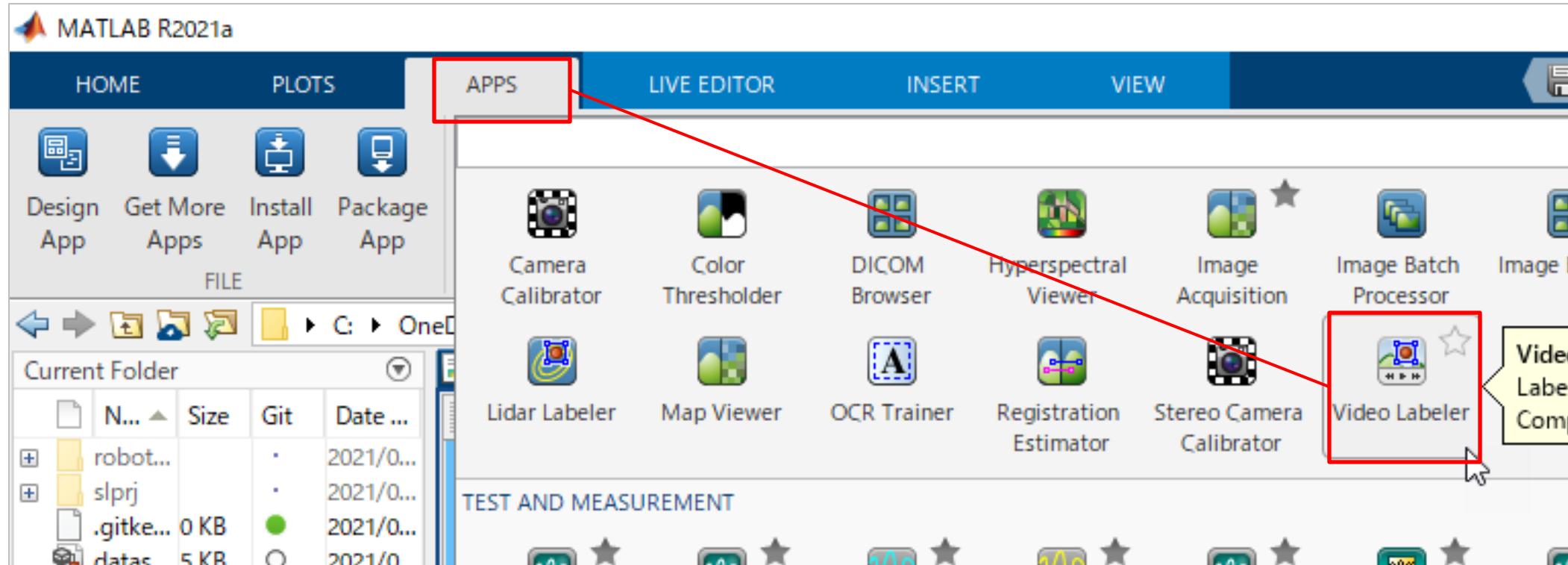
- Annotate target position each frame using Video Labeler in Computer Vision Toolbox to create training dataset



Training dataset  
(Combined dataset with images and  
target positions)

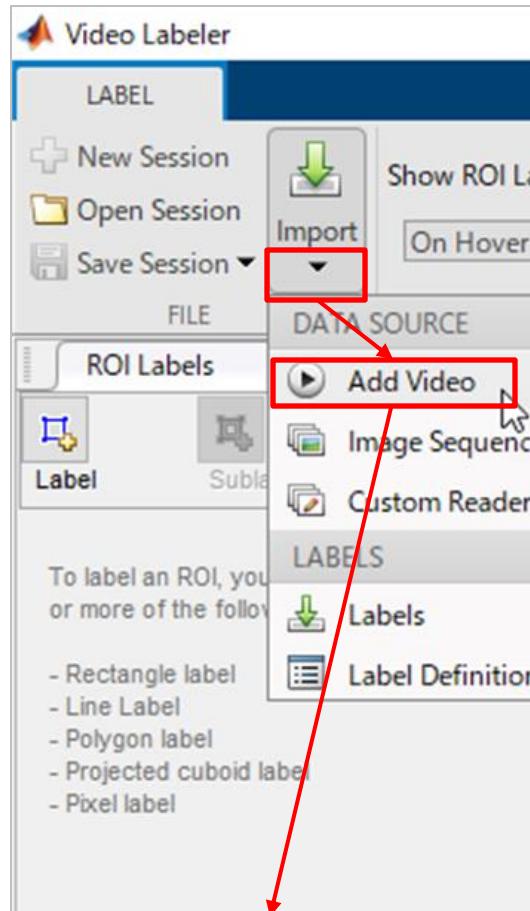
## Exercise 2. Annotate Video File using Video Labeler

1. Launch "Video Labeler" as below.

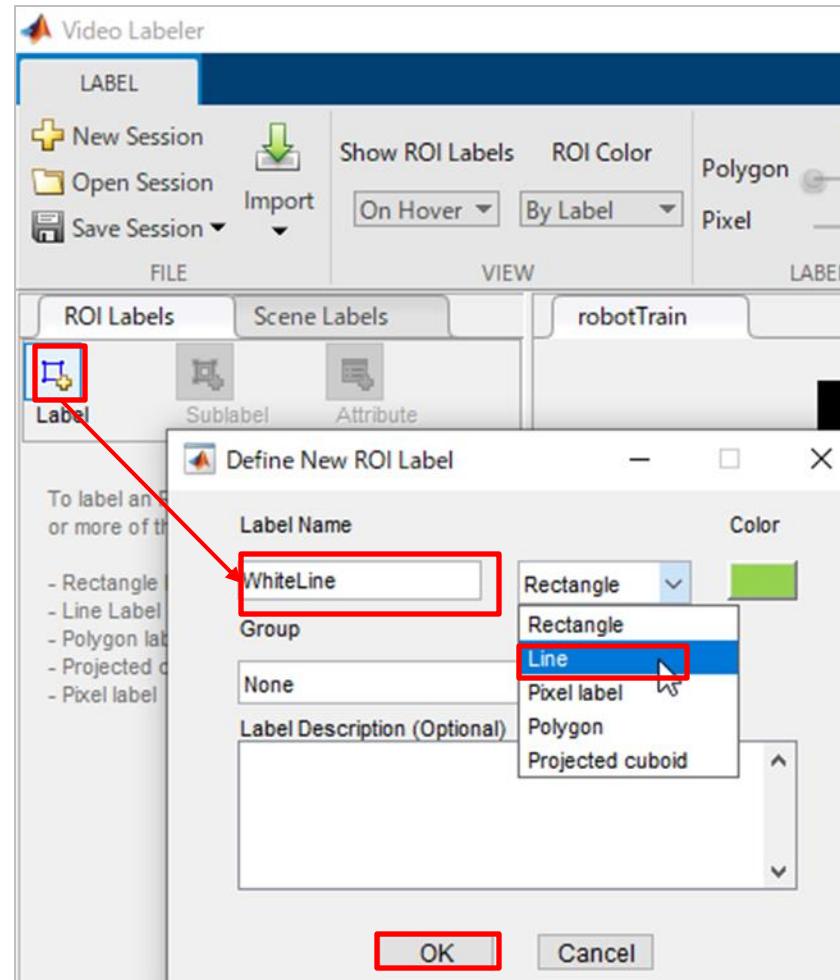


# Exercise 2. Annotate Video File using Video Labeler

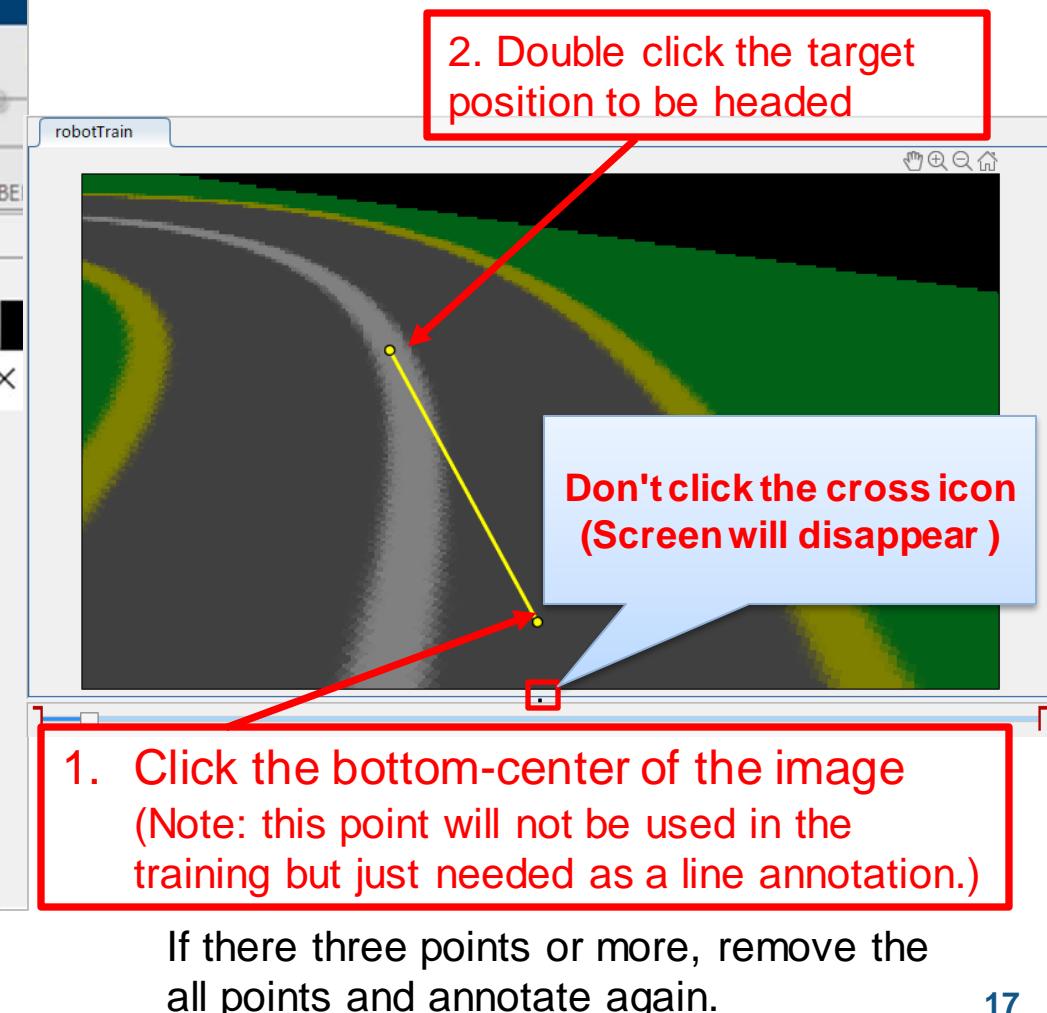
## 2. Load the video file



## 3. Define a label

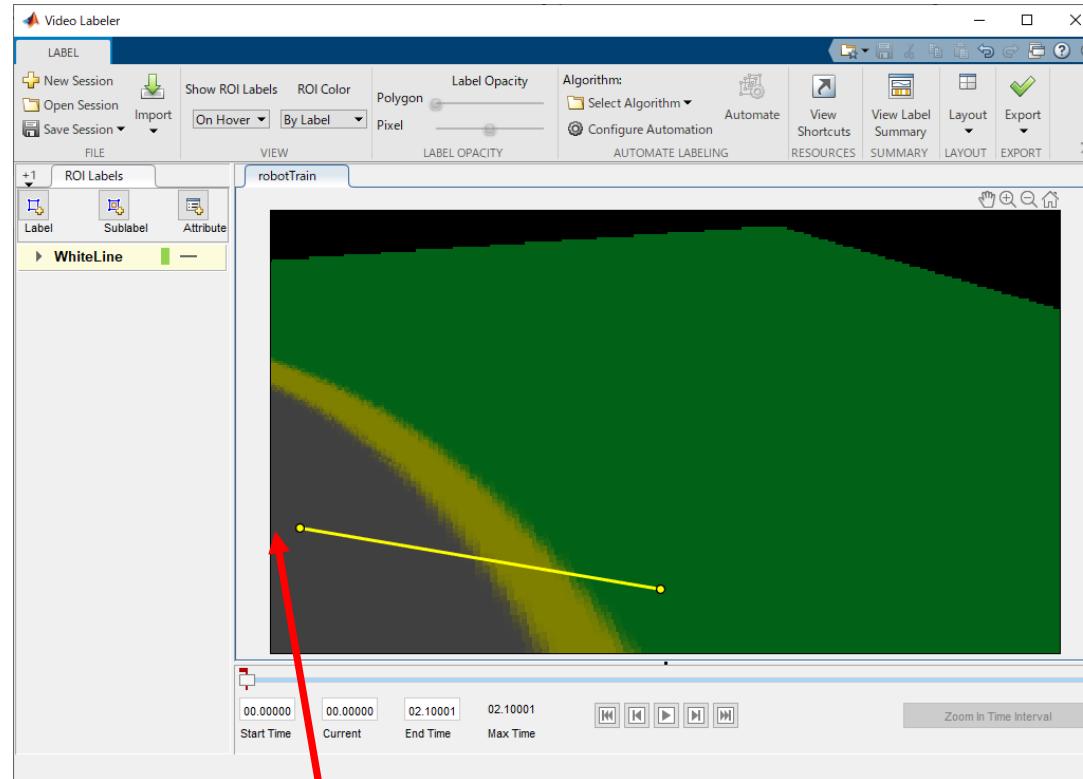


## 4. Annotate about 20 images from the bottom-center of the image to the target position

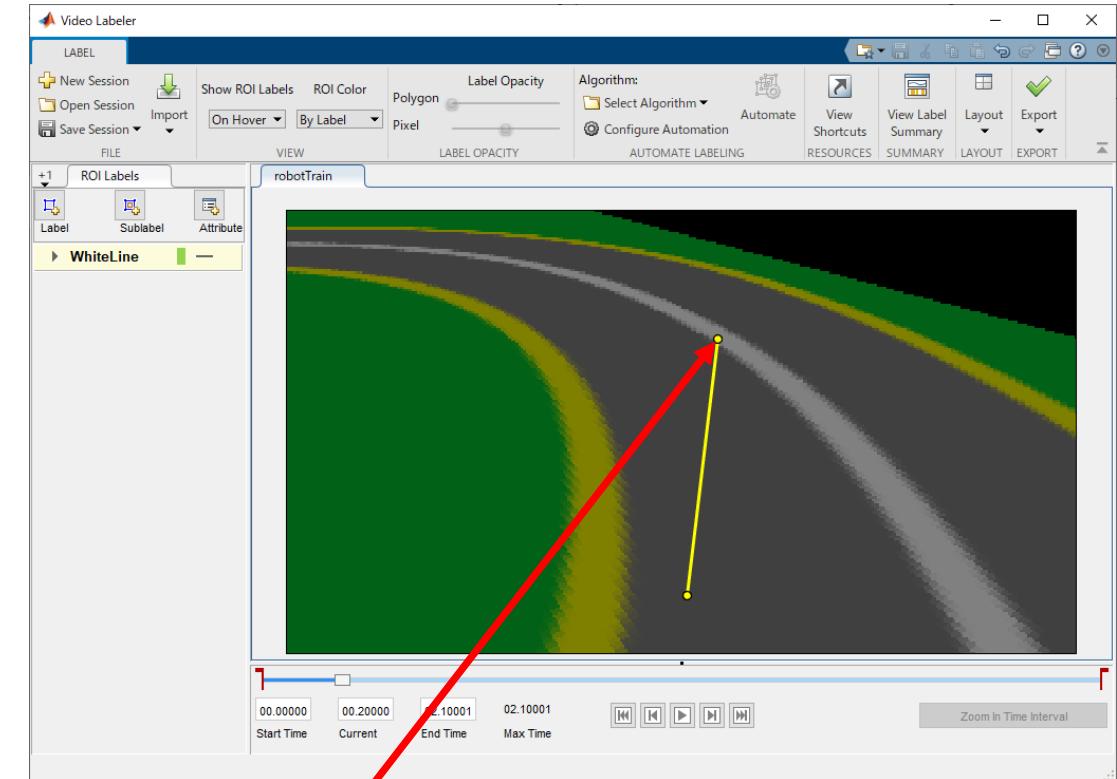


# Exercise 2. Annotate Video File using Video Labeler

Annotation example 1:

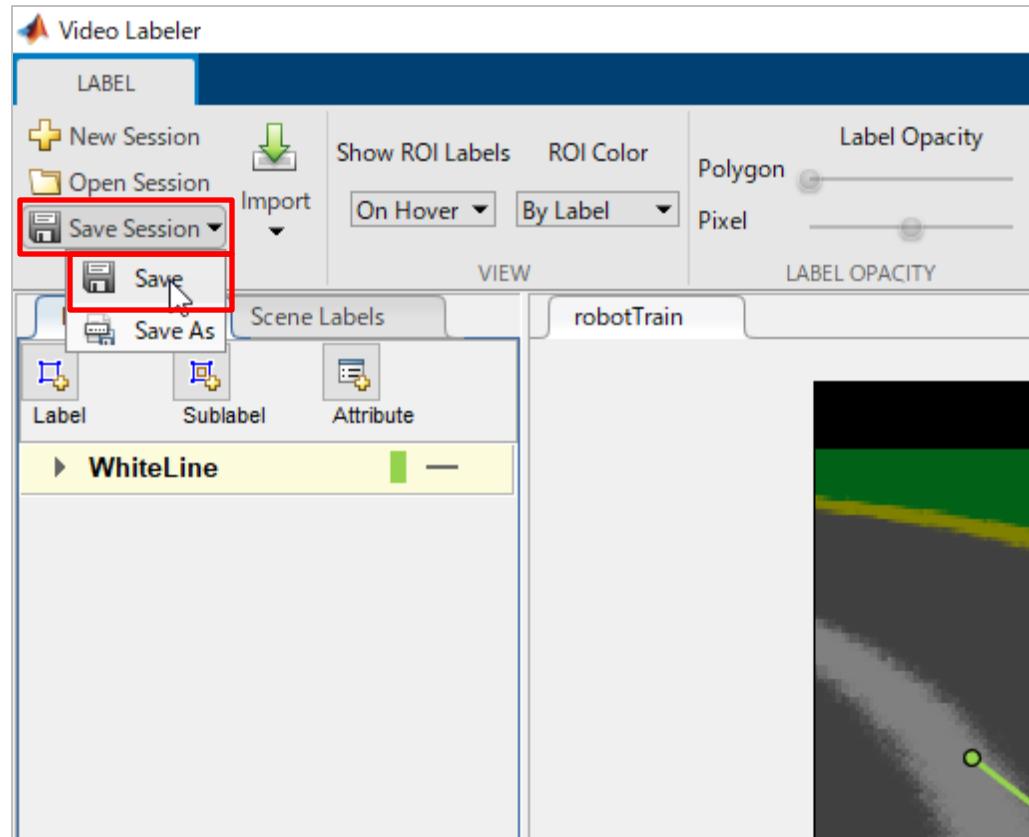


Annotation example 2:

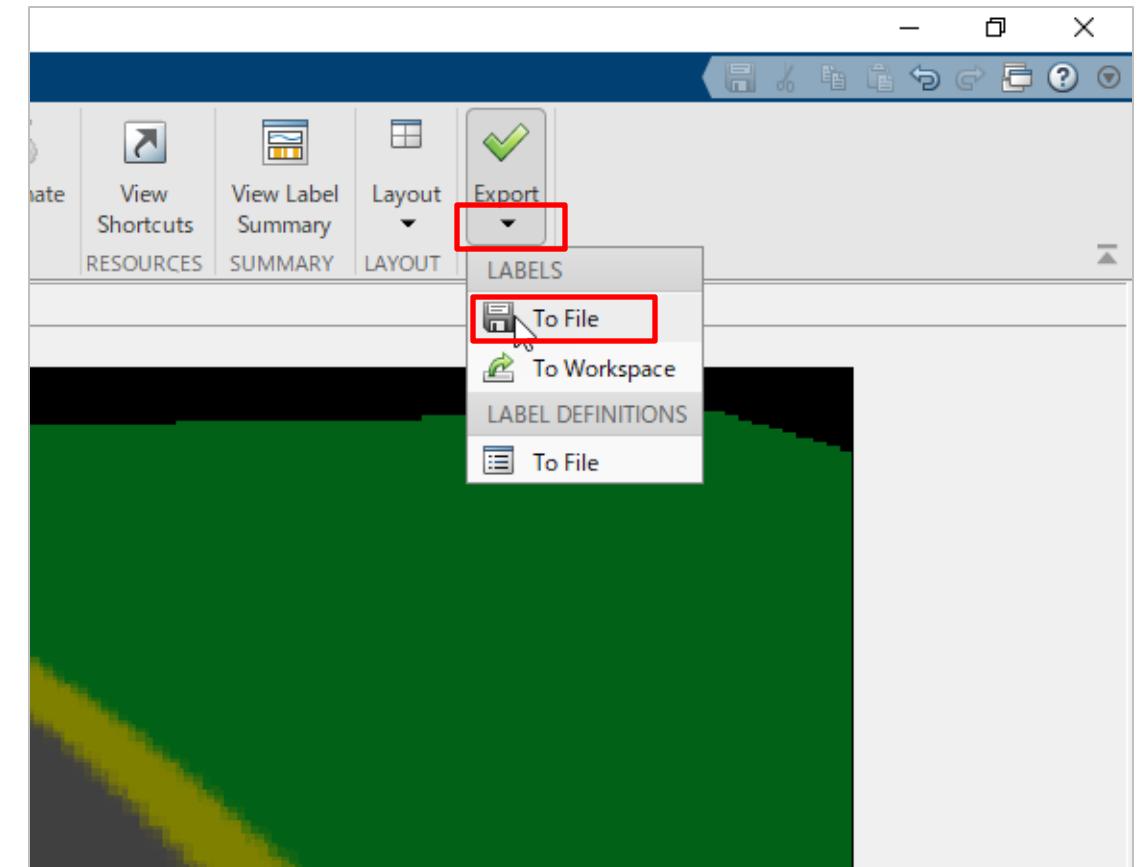


## Exercise 2. Annotate Video File using Video Labeler

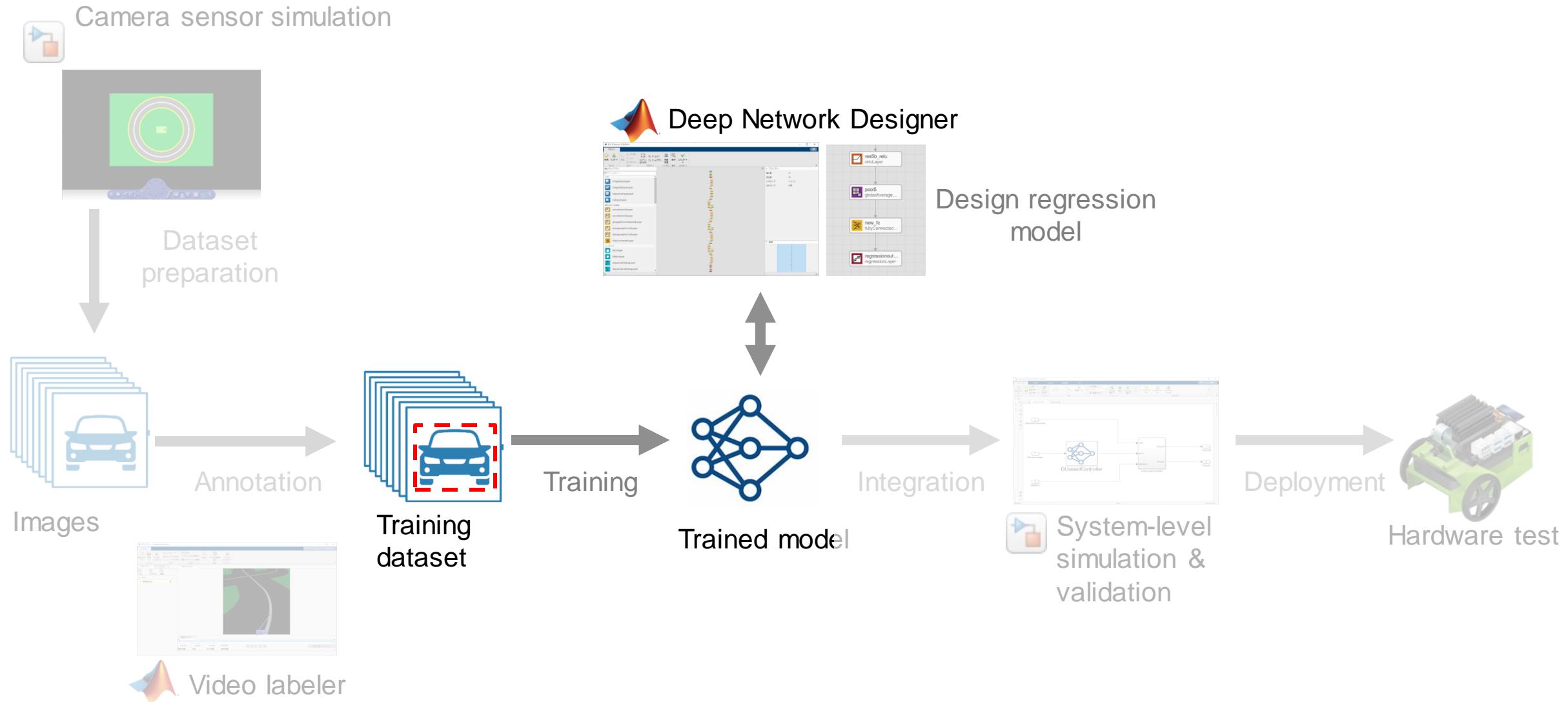
5. Save the session as needed  
(Just leave the file name as default)



6. Export the annotated data as  
[**exportedLabels.mat**] once finish the annotation



# AI-based Autonomous System Development Workflow



# Predict Target Position using Deep Learning

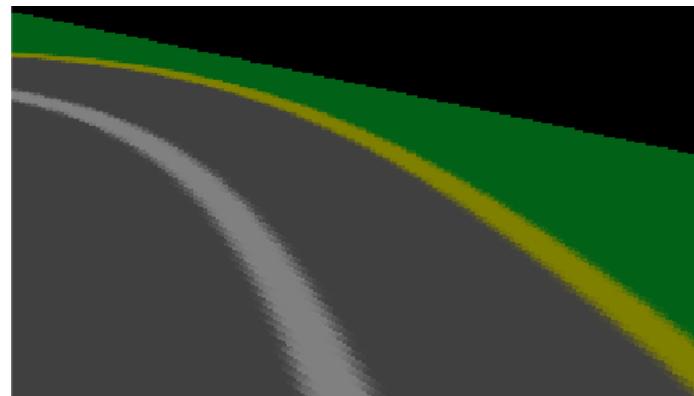
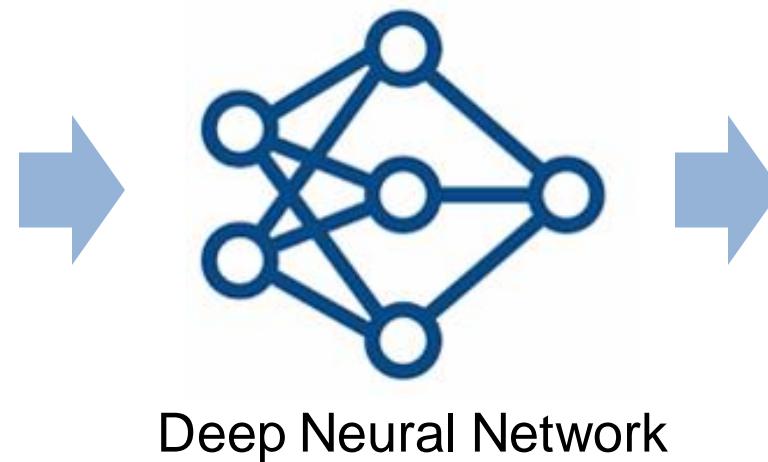


Image from camera  
( $180 \times 320 \times 3$ )



Predict target position using a deep neural regression model

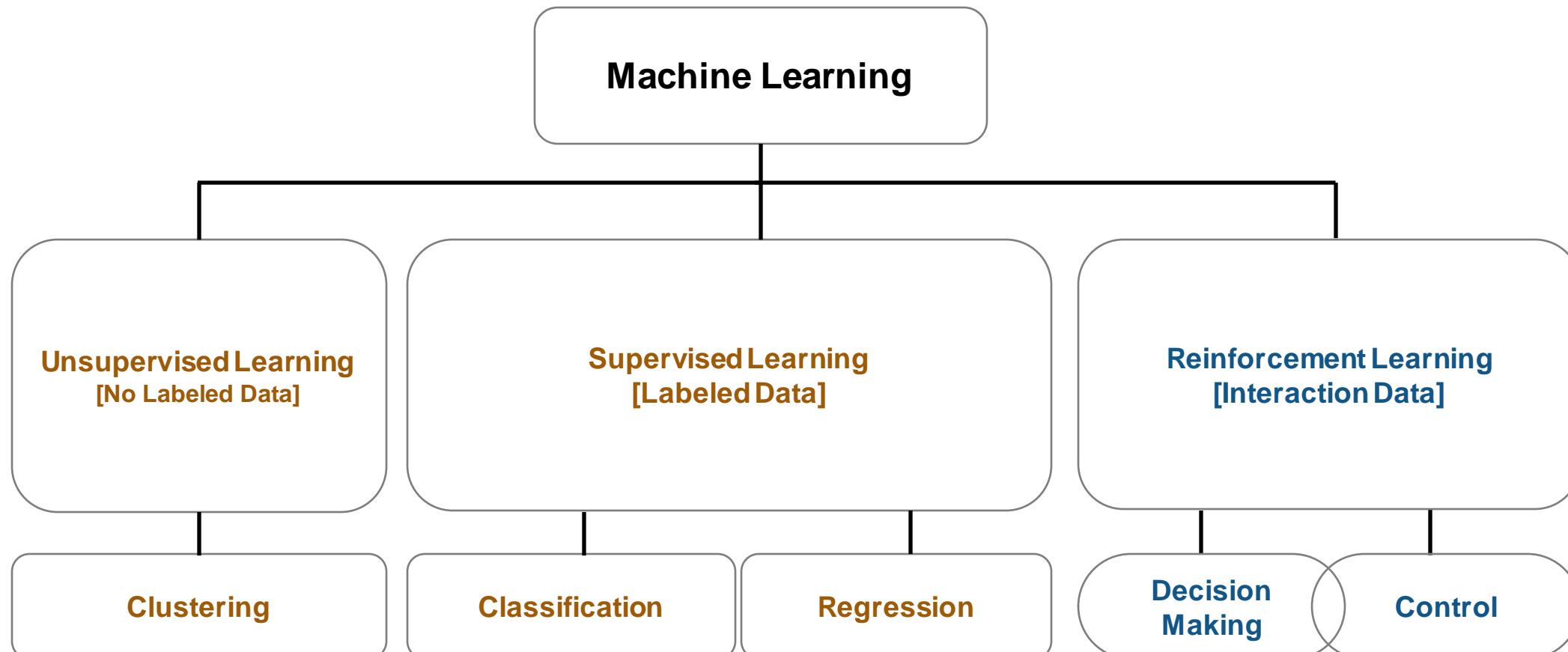


Target position( $x, y$ )  
( $1 \times 2$ )

Why deep learning?

- Not easy to extract low-dimensional features from high-dimensional data like images using classical image processing and machine learning
- Deep learning is flexible and can handle those high-dimensional data

# What is Regression?



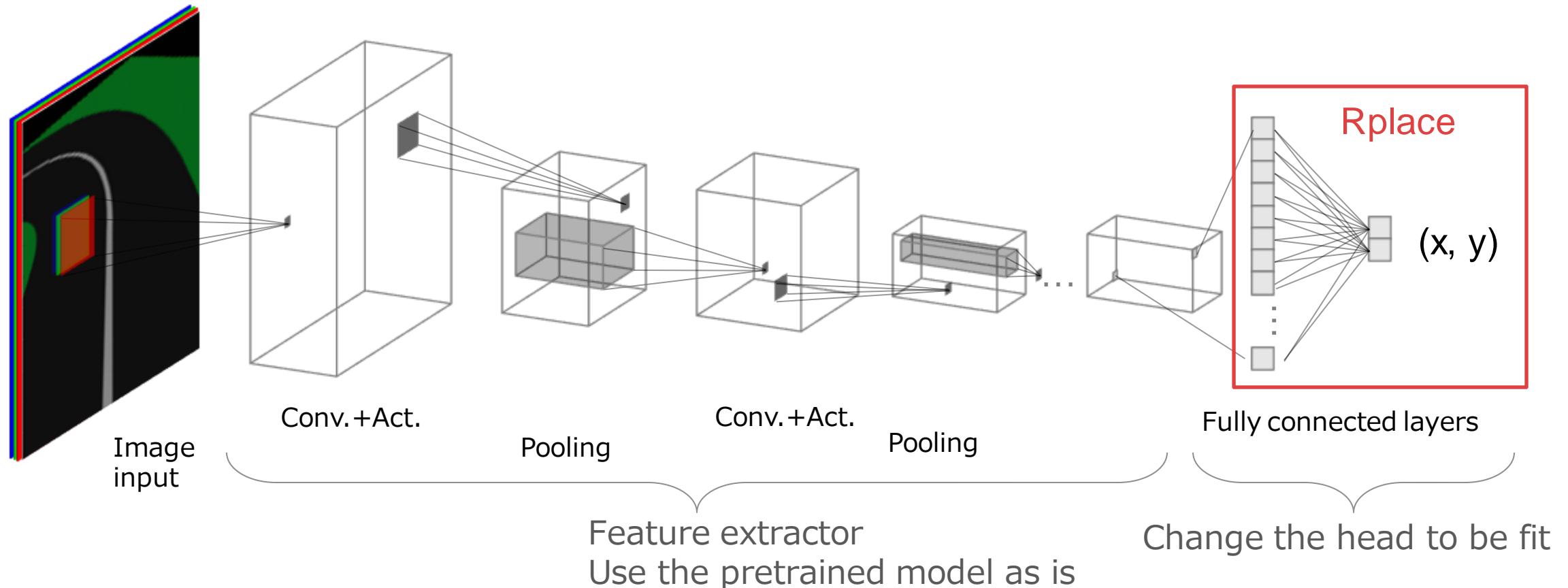
Predict discrete values  
ex. Red apple or green apple

Predict continuous values  
ex. Diameter of an apple

Predicting target position ( $x, y$ ) as continuous values  
from image is a regression problem

# Deep Neural Regression with Transfer Learning

Use a pretrained image classification model as a feature extractor  
Change the head of the model to a regression layer



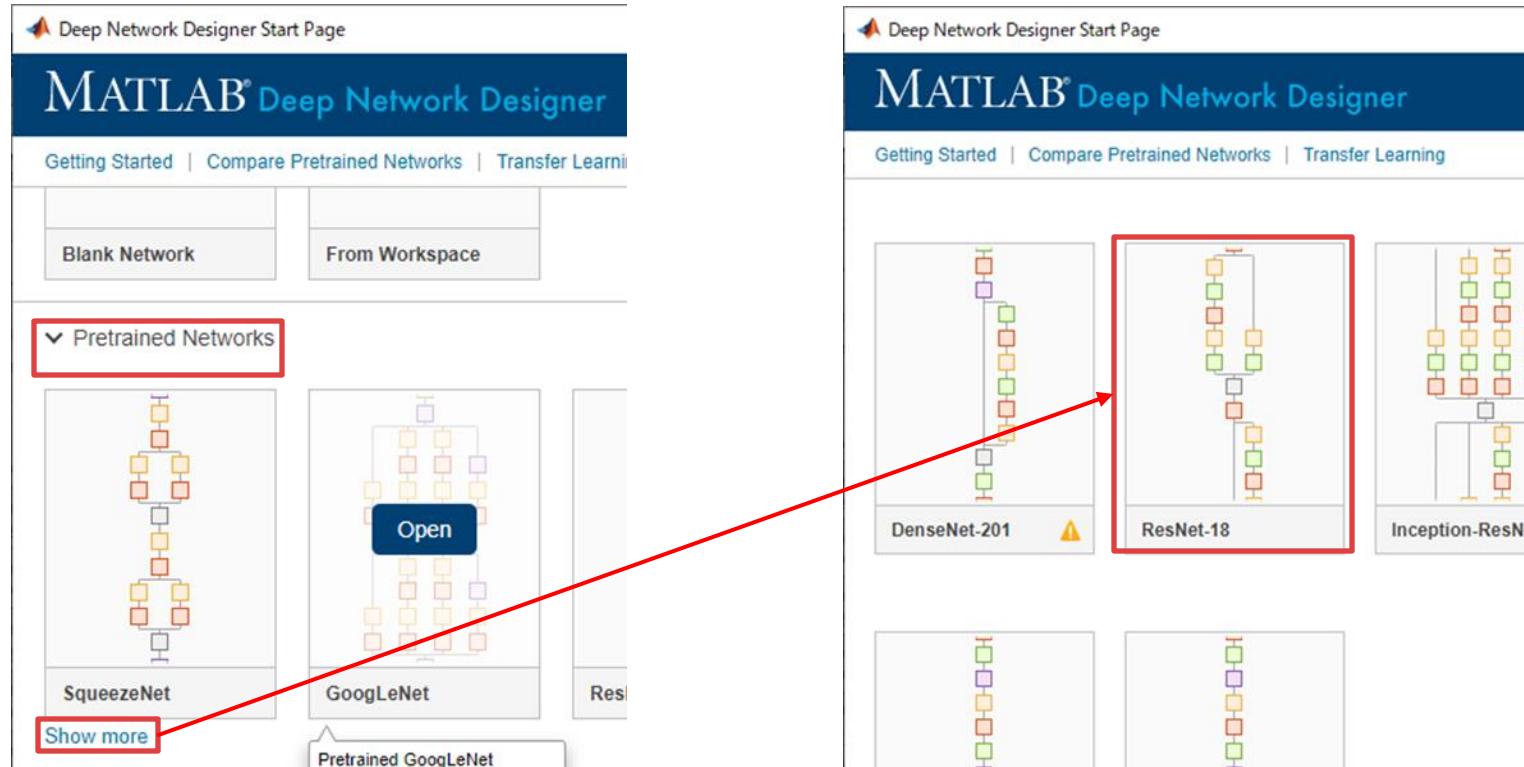
Reference : Convert Classification Network into Regression Network

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

# Exercise 3. Design and Train Regression Model

Train a regression model by modifying the last classification layers in the pretrained image classification model.

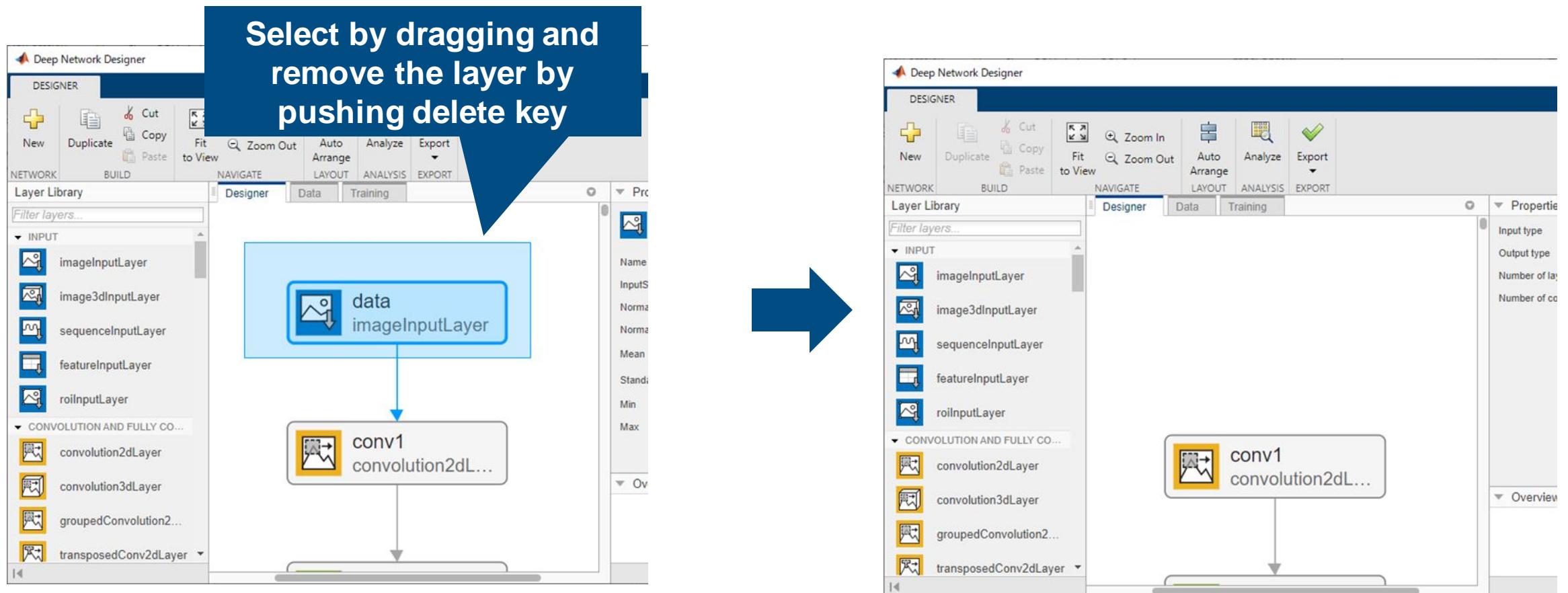
1. Open the live script as below  
`>> edit trainWhiteLine.mlx`
2. Click "Run and Advance" in the "Live Editor" ribbon to line 71
3. Run line 72 and then Deep Network Designer will be launched. Click "ResNet-18".



## Exercise 3. Design and Train Regression Model

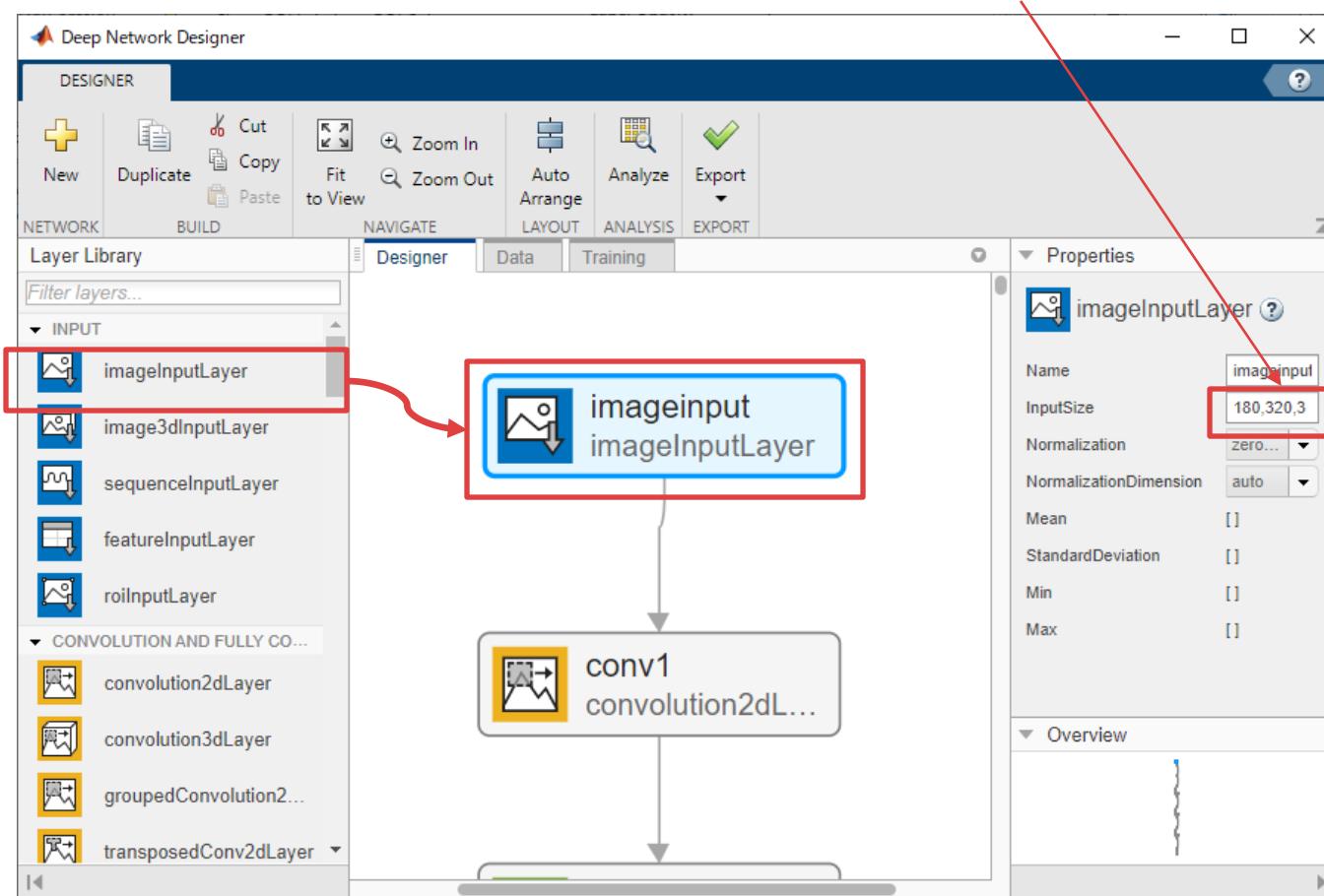
4. Select data layer by dragging and then remove them by pushing delete key

(Need to replace the input layer because the input size of the pretrained ResNet18 is  $224 \times 224 \times 3$  while the size of the training images is  $180 \times 320 \times 3$ )



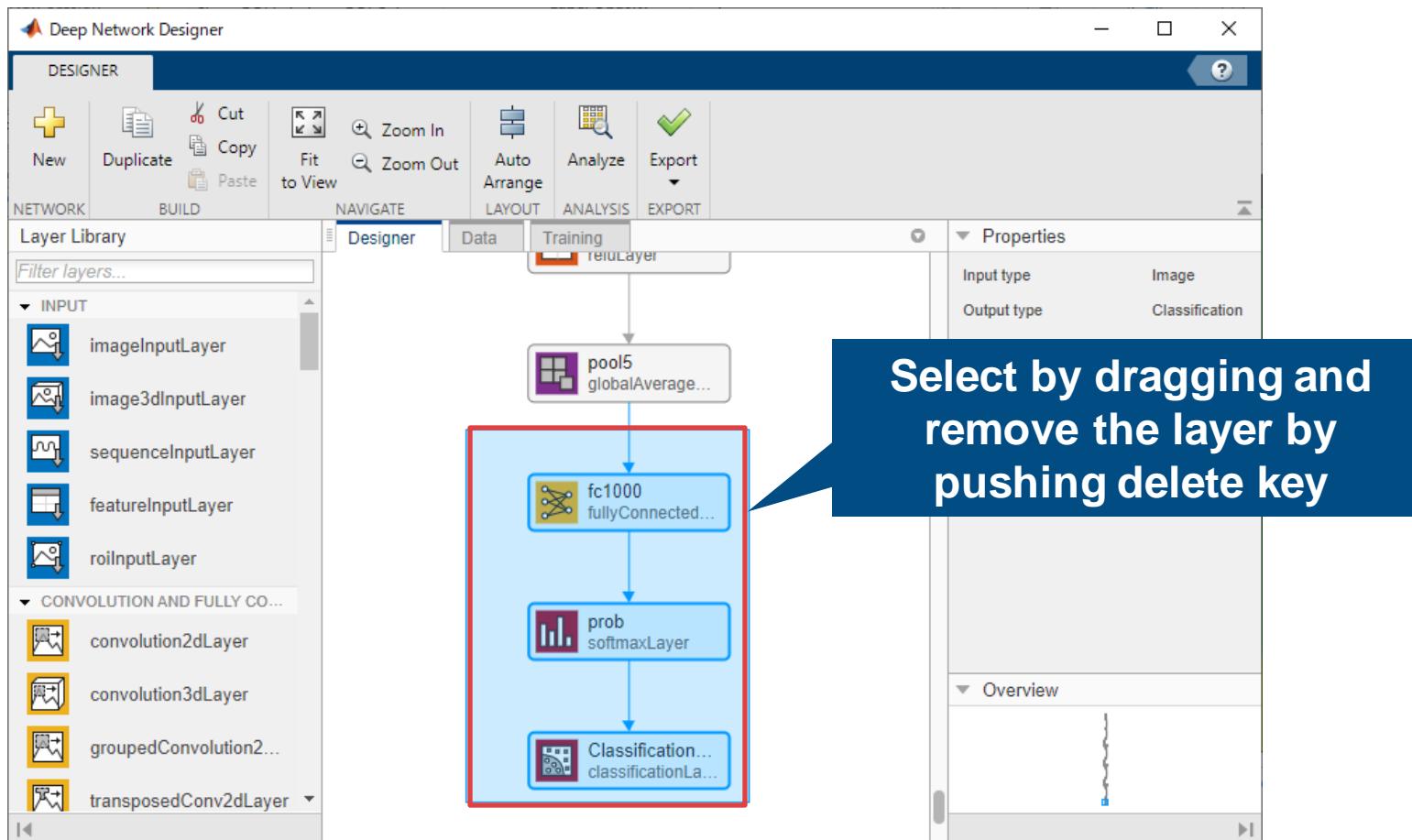
## Exercise 3. Design and Train Regression Model

5. Drag & drop the "imageInputLayer" in the "INPUT" group in the "Layer Library" tab to the designer campus. Drag the edge of the `imageinput` layer to connect to the `conv1`.  
Set the "InputSize" property in the `imageinput` layer to "180,320,3".



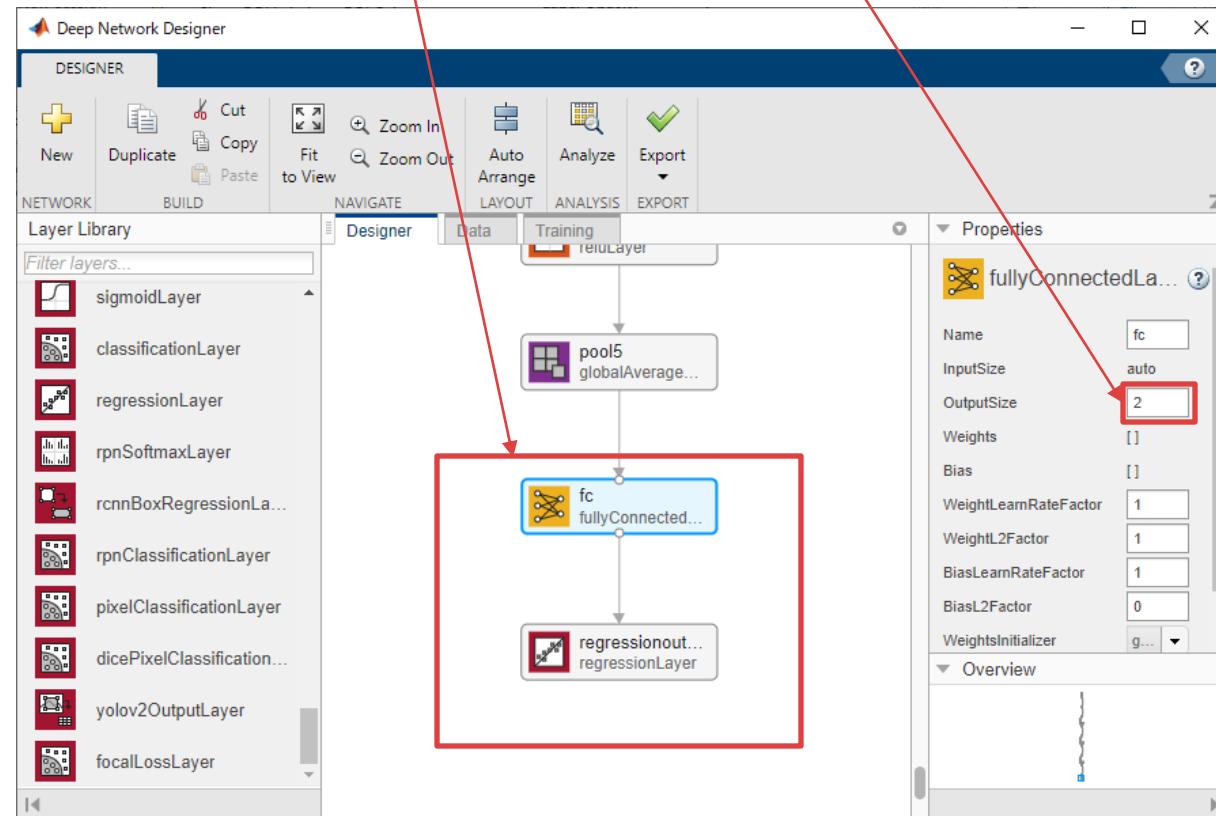
## Exercise 3. Design and Train Regression Model

6. Select "fc1000" layer and the successive layers by dragging and then remove them by pushing delete key  
(Need to change the 1,000 classes classification model to a 2 outputs regression model)



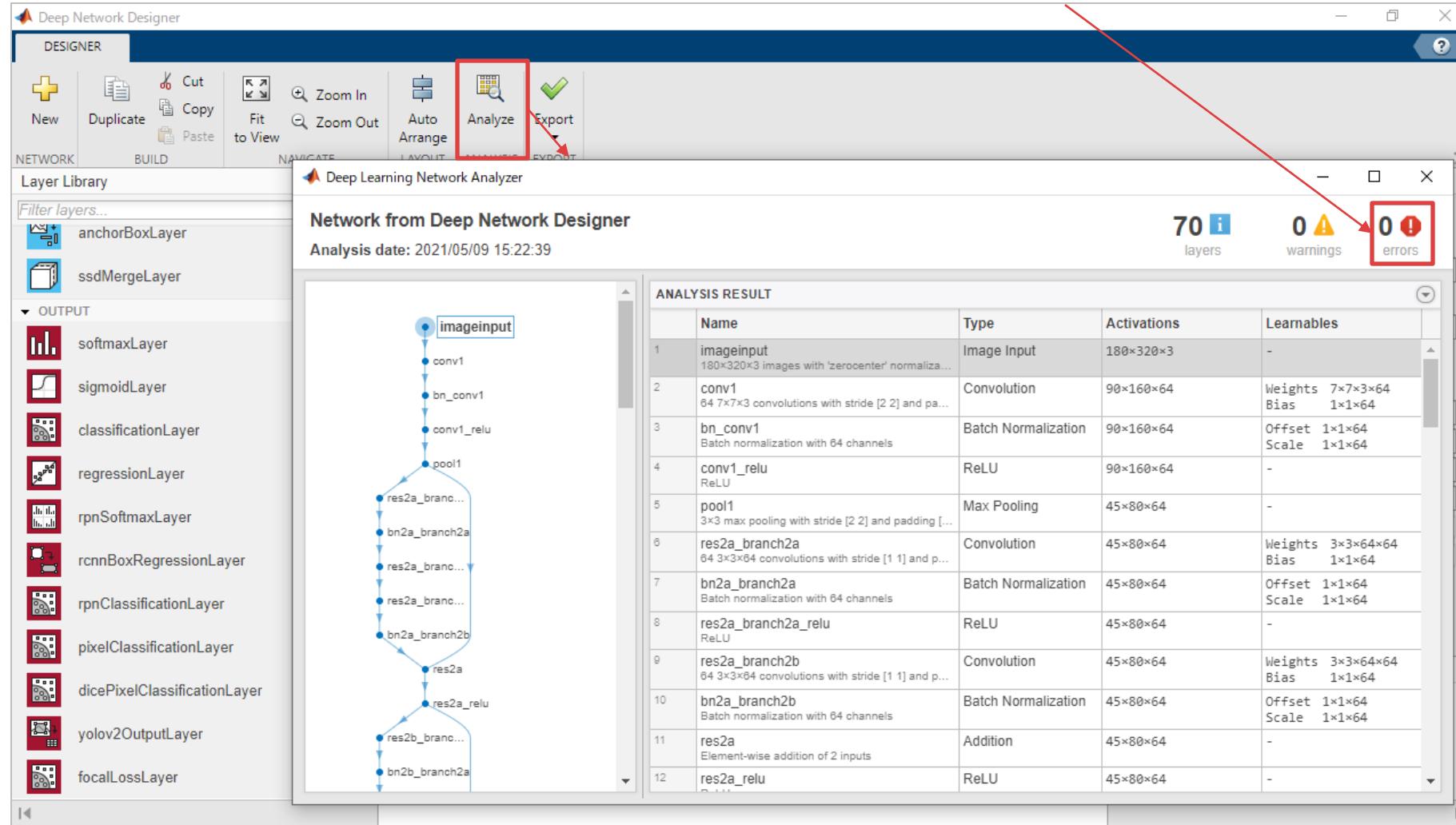
# Exercise 3. Design and Train Regression Model

7. Add fullyConnectedLayer from "Convolution and fully connected layers" section and "regressionOutputLayer" from "Output" section in "Layer Library" located in the left panel.
8. Change the OutputSize to 2 because the number of output values are two (x, y)
9. Connect pool5 to fc and fc to regressionOutput respectively



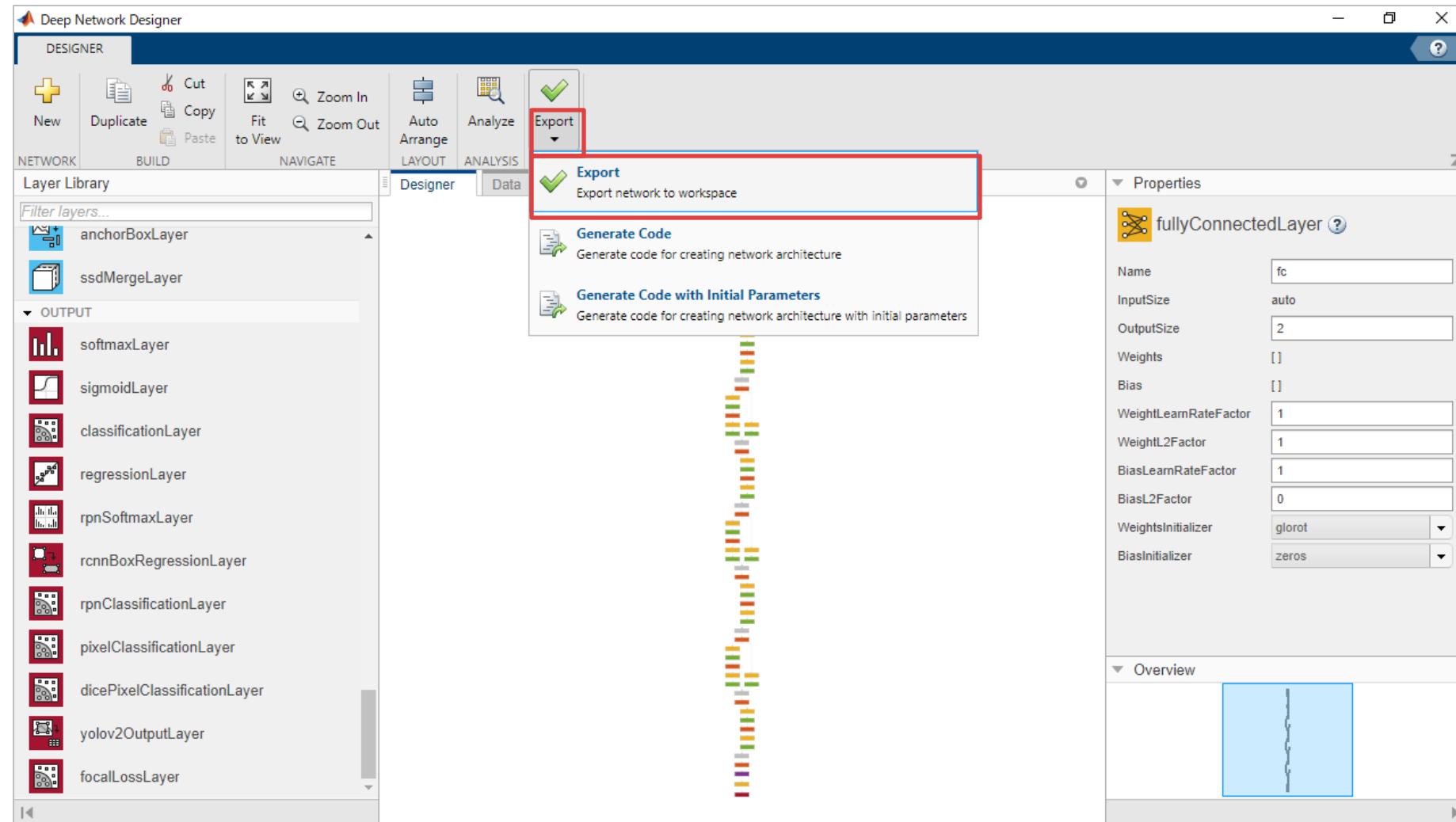
# Exercise 3. Design and Train Regression Model

8. Make sure the network is consistent and the number of errors is zero by clicking "Analyze" button



# Exercise 3. Design and Train Regression Model

9. Export the model to workspace by clicking "Export" in "Export" button.

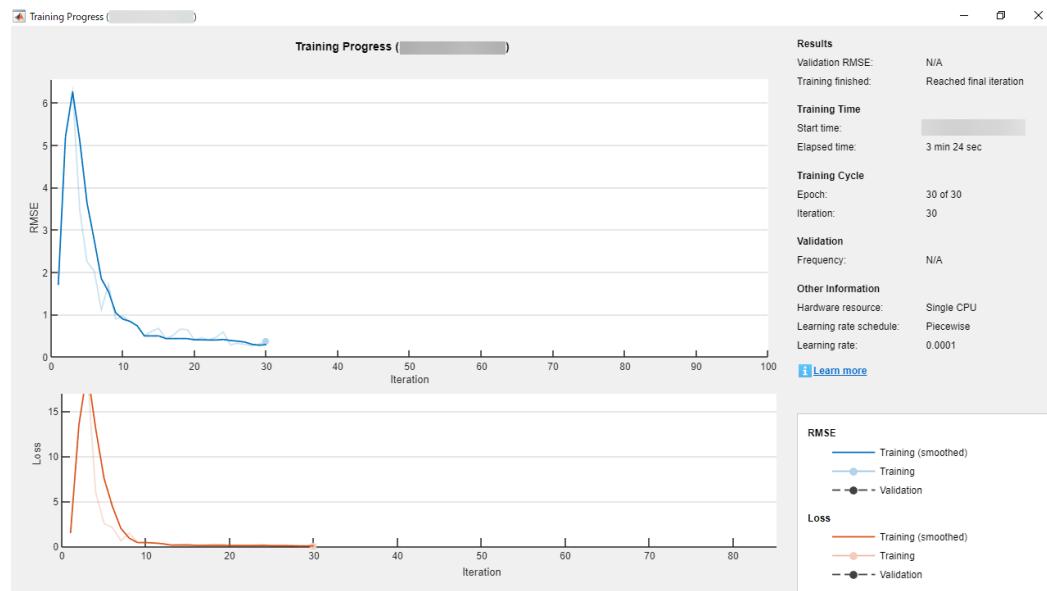


# Exercise 3. Design and Train Regression Model

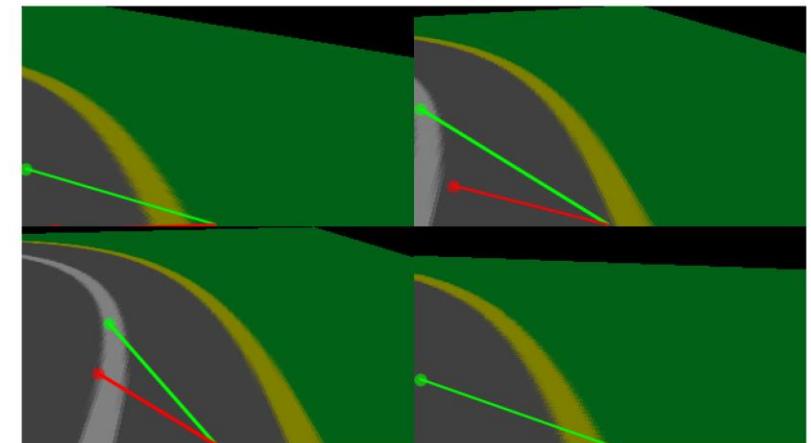
10. Run the rest of the code to end

The training progress will be shown up as below when training begins (It may take 5 minutes with CPU)

11. `mynet_new.mat` is generated once the training is completed

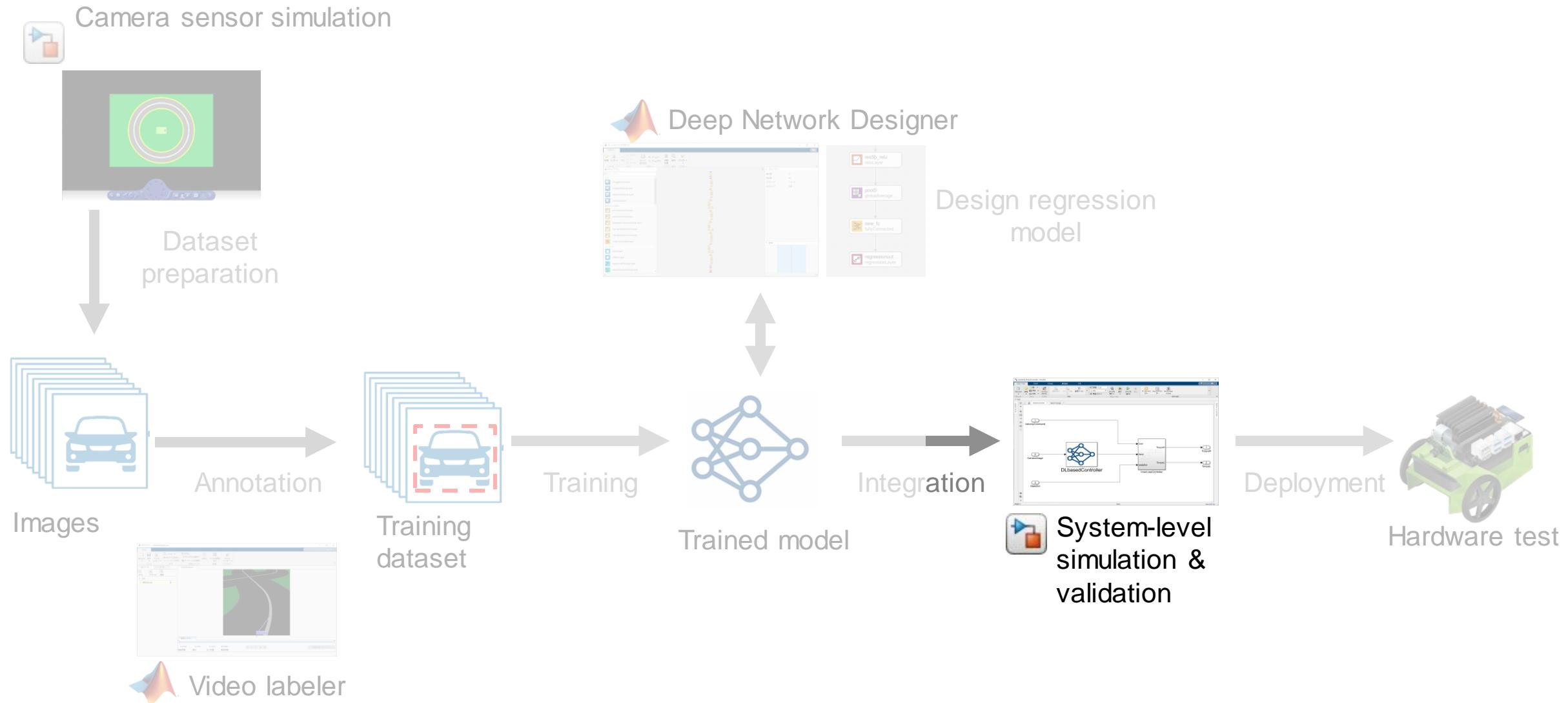


Training progress

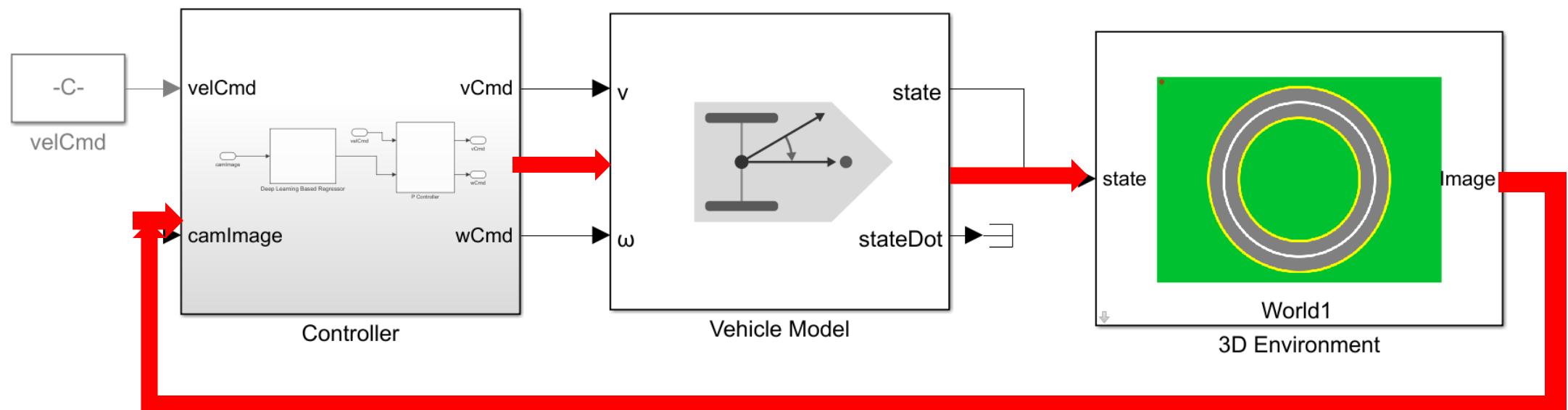
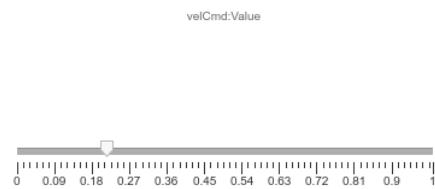


Predicted results  
(Green: ground truth  
Red: predicted)

# AI-based Autonomous System Development Workflow



# Test Line Following Algorithms Integrating Components as System

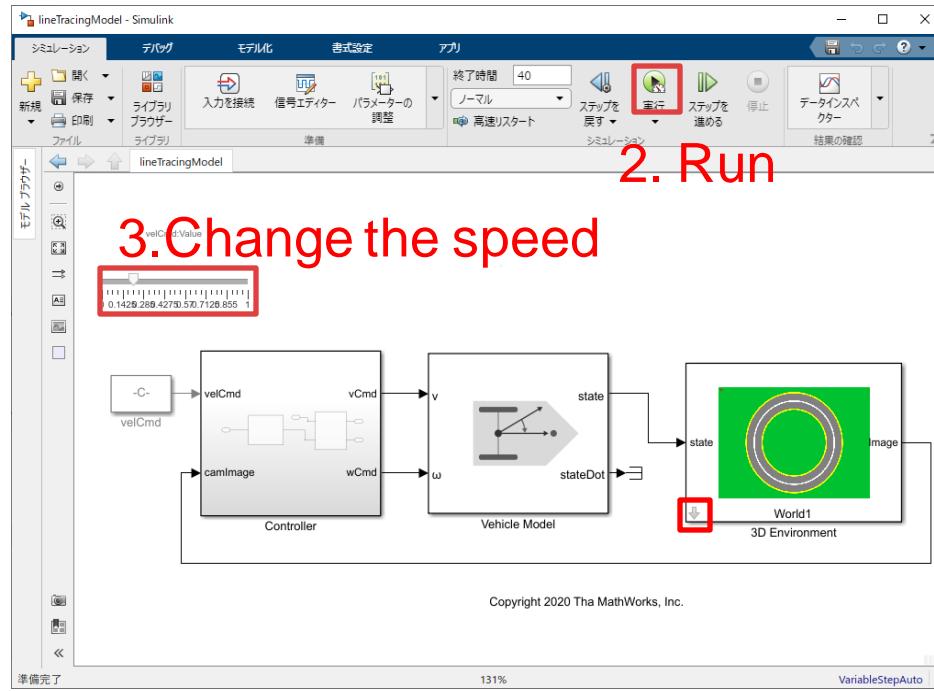


Copyright 2020 Tha MathWorks, Inc.

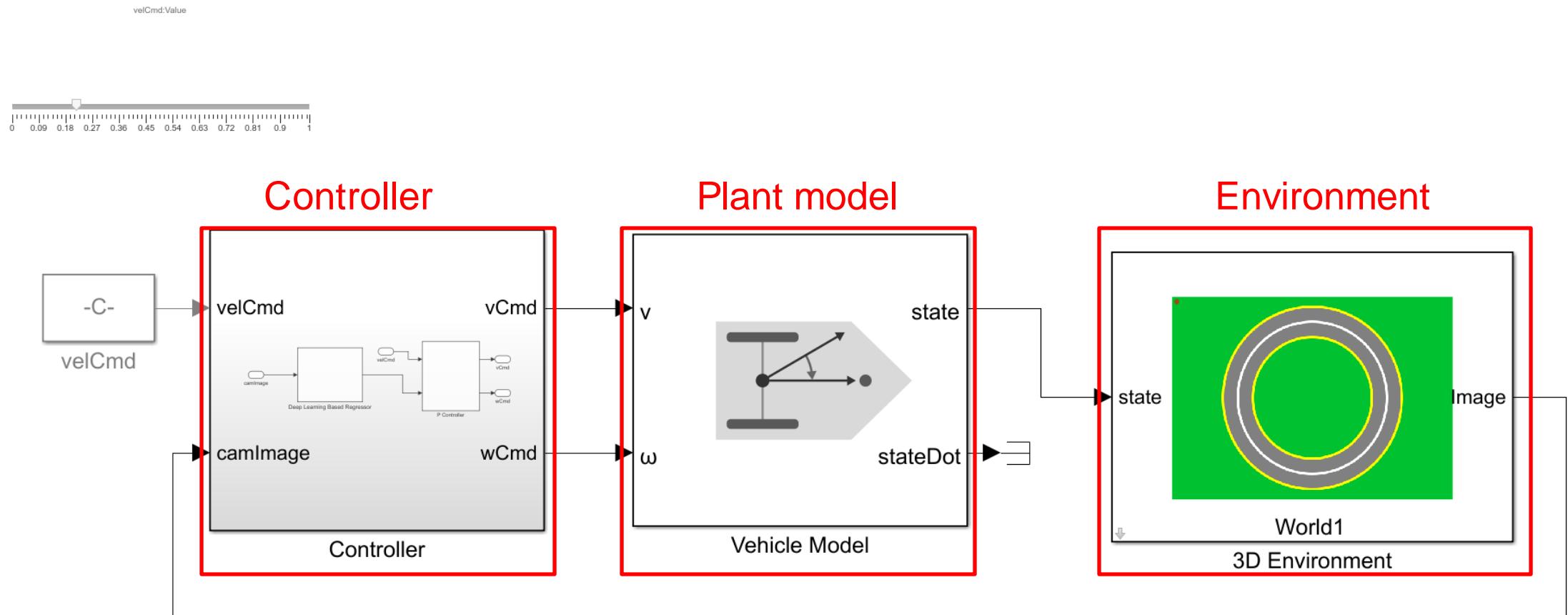
Autonomous system can be a closed loop system  
Performance test and validation as a system are required

# Exercise 4. Test Line Following Algorithm as System

1. Open `line_follower.slx` and click run button in "Simulation" tab  
Click stop button (gray square button) if you want to stop the simulation  
`>> line_follower.slx`
2. Change the bar for speed and then observe what will be happened in the line following
3. Try to change the initial position and observe what will be happened (See [p.35](#))
4. Try to change the P gain and see how the following performance changes (See [p.34](#))
5. Try to change the map data for 3-D world and see how the following behavior changes (See [p.36](#))
6. Save the simulation results as a video (See [p.37](#))

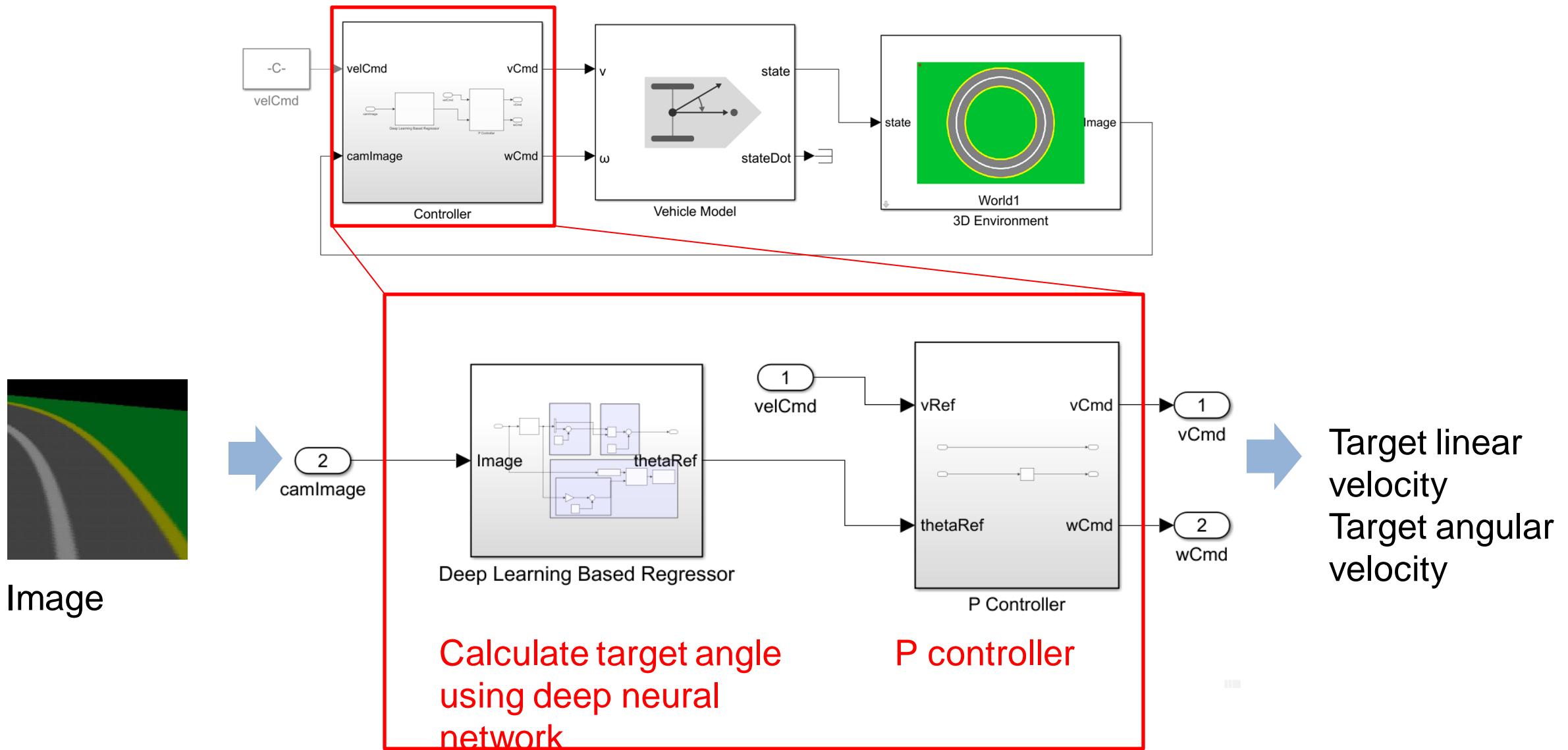


# System Overview

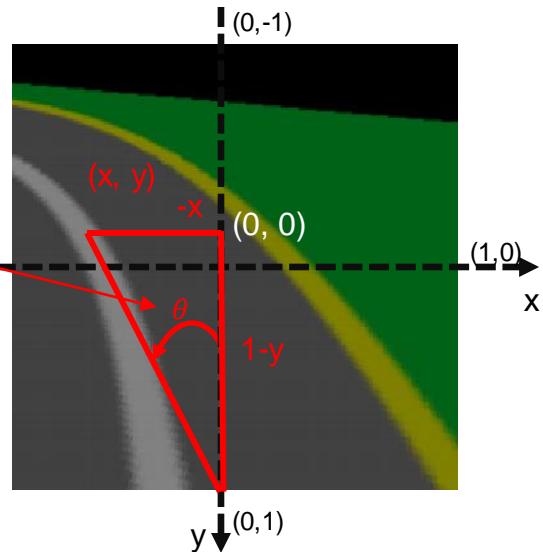
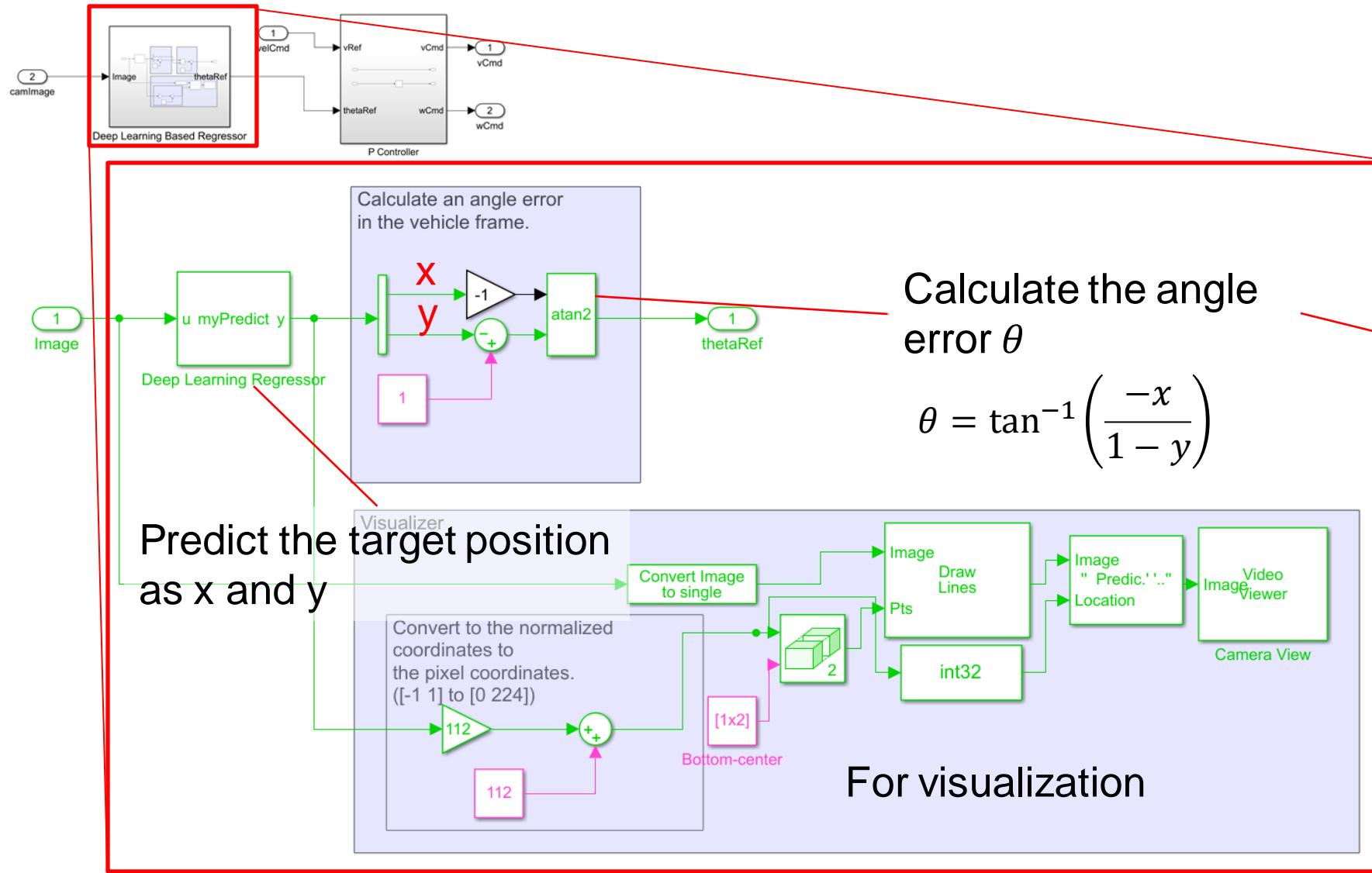


Copyright 2020 Tha MathWorks, Inc.

# System-level Simulation : Controller



# System-level Simulation : Calculate Angle Error

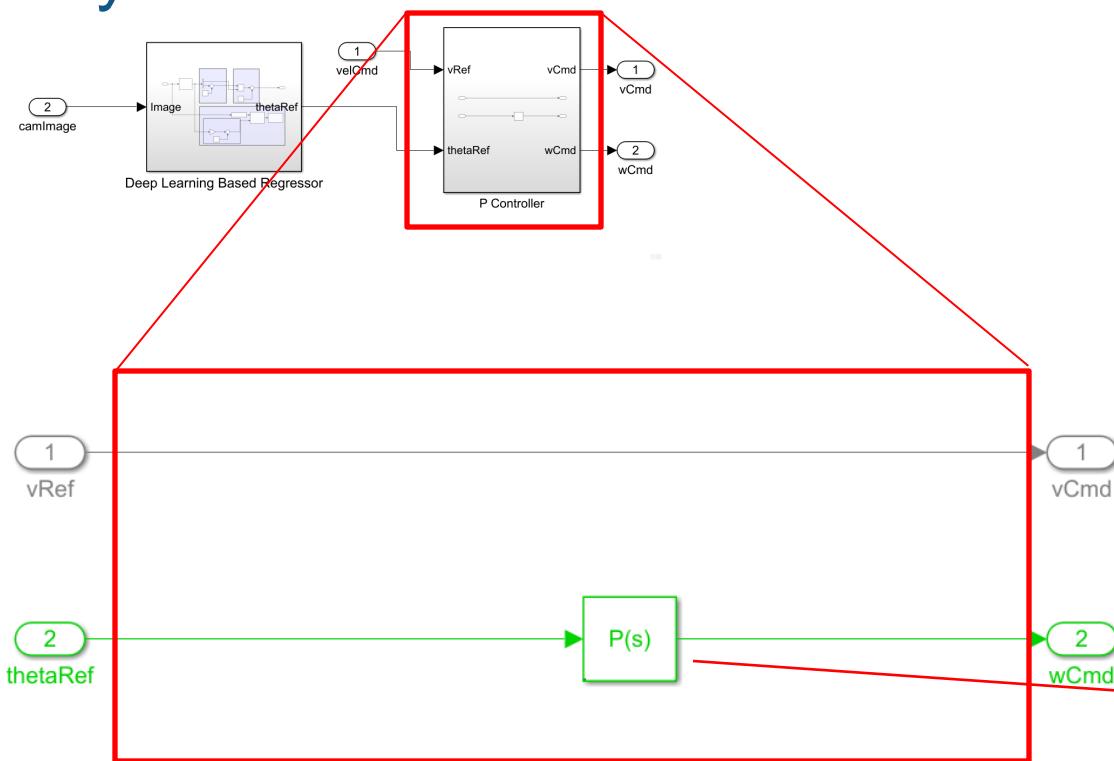


Calculate the angle error  $\theta$

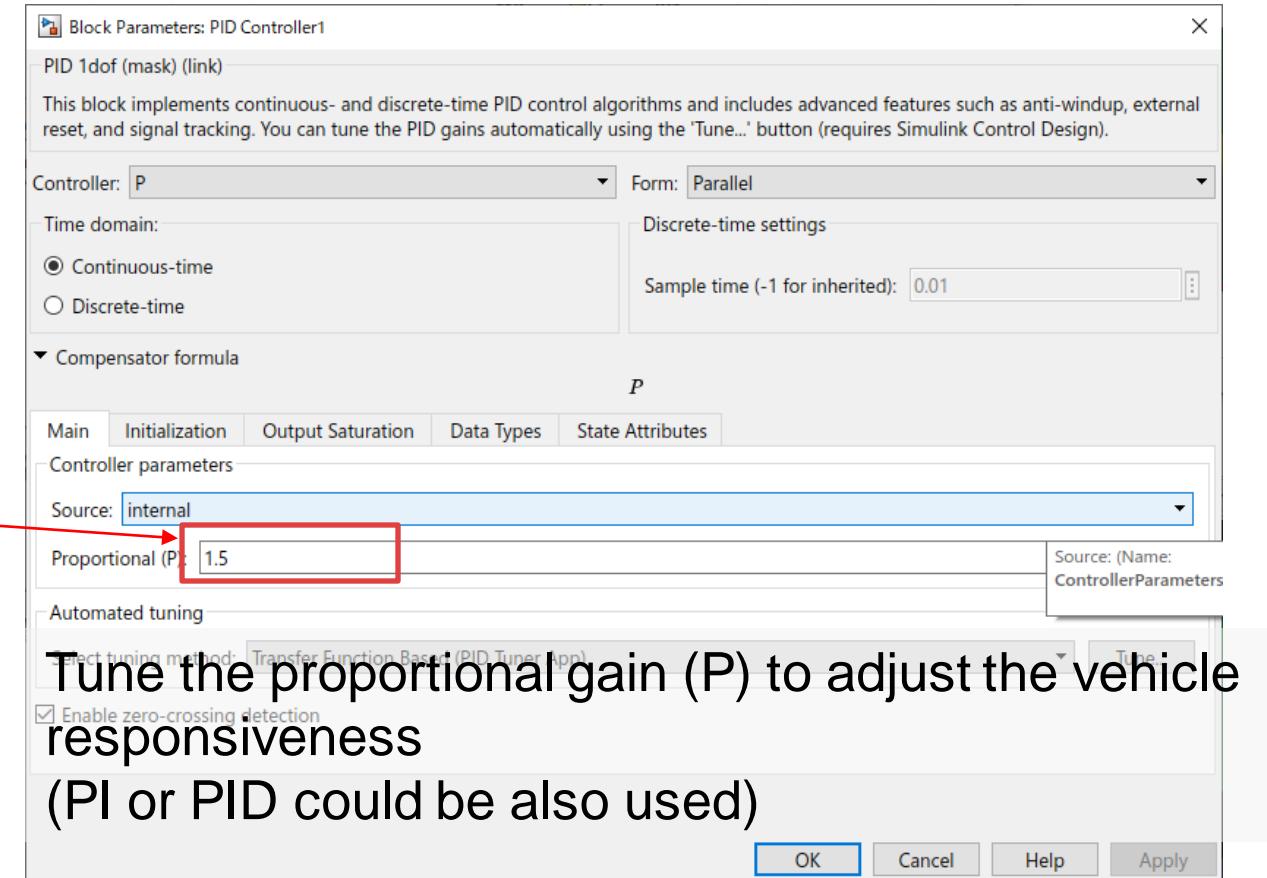
$$\theta = \tan^{-1}\left(\frac{-x}{1-y}\right)$$

For visualization

# System-level Simulation : Feedback Controller

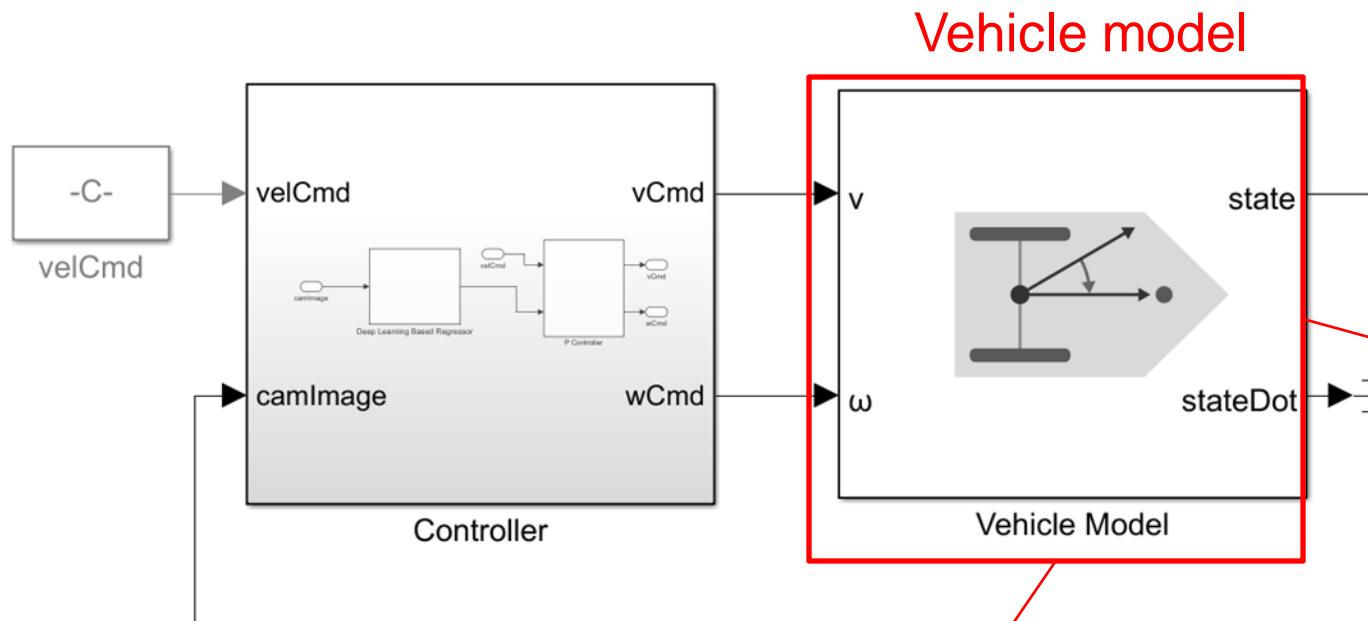


Feedback to make the error angle to be zero  
Control only steering  
Linear velocity is constant

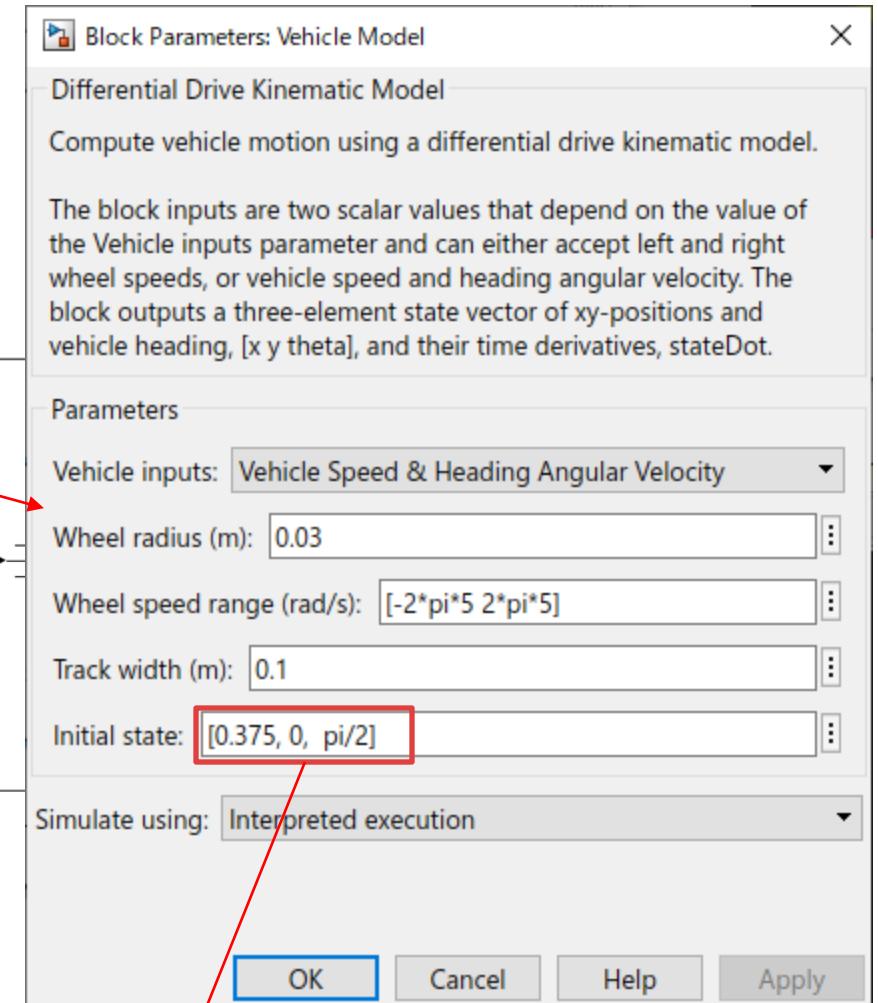


Tune the proportional gain (P) to adjust the vehicle responsiveness  
(PI or PID could be also used)

# System-level Simulation : Plant Model

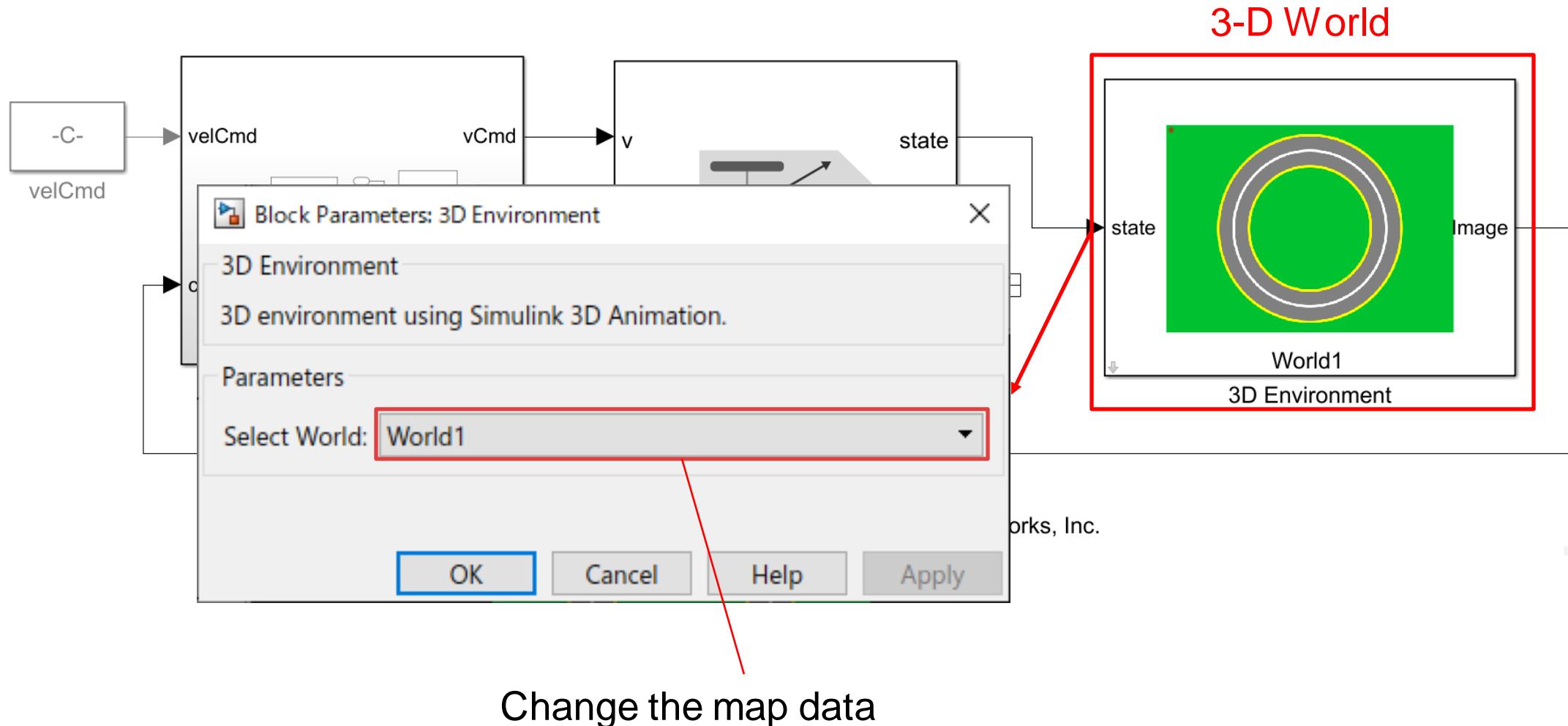


Use a differential drive model which receives linear velocity and angular velocity as inputs then outputs position and orientation ( $x_v, y_v, \theta_v$ )

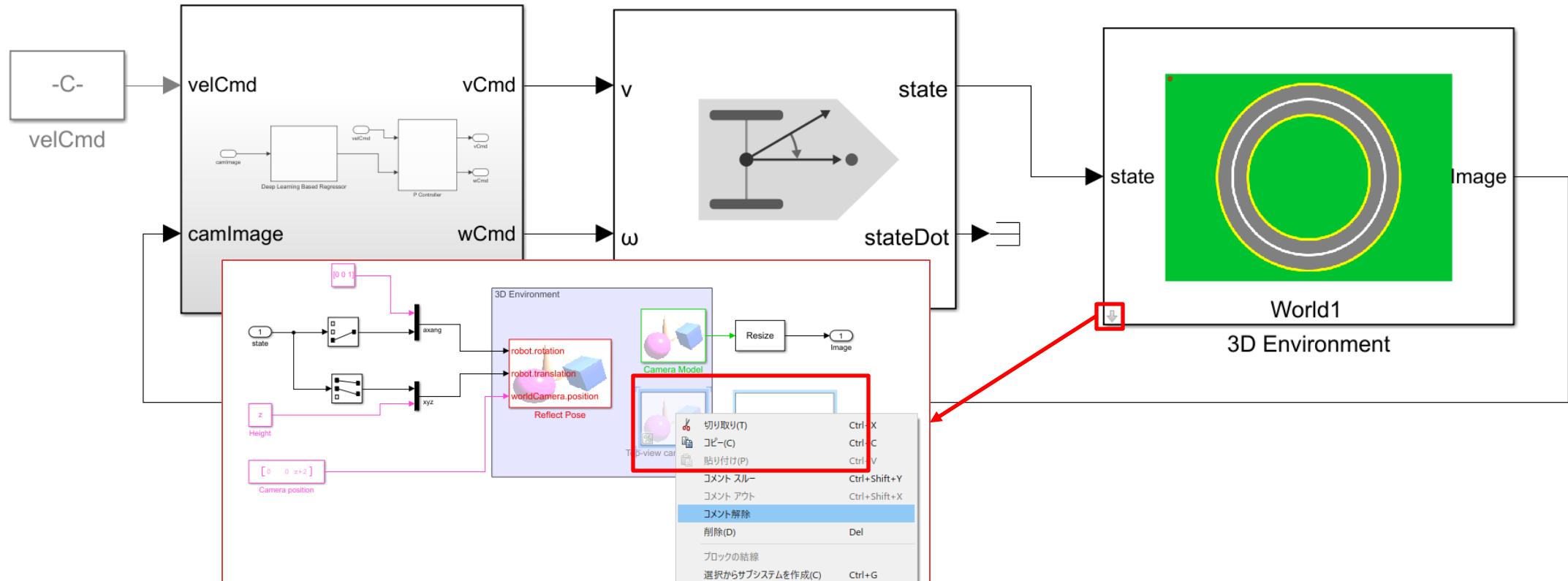


Change the initial states ( $x_v, y_v, \theta_v$ ) and observe how the line following works

# System-level Simulation : Change Map in 3D Environment



# System-level Simulation : Save Simulation Results

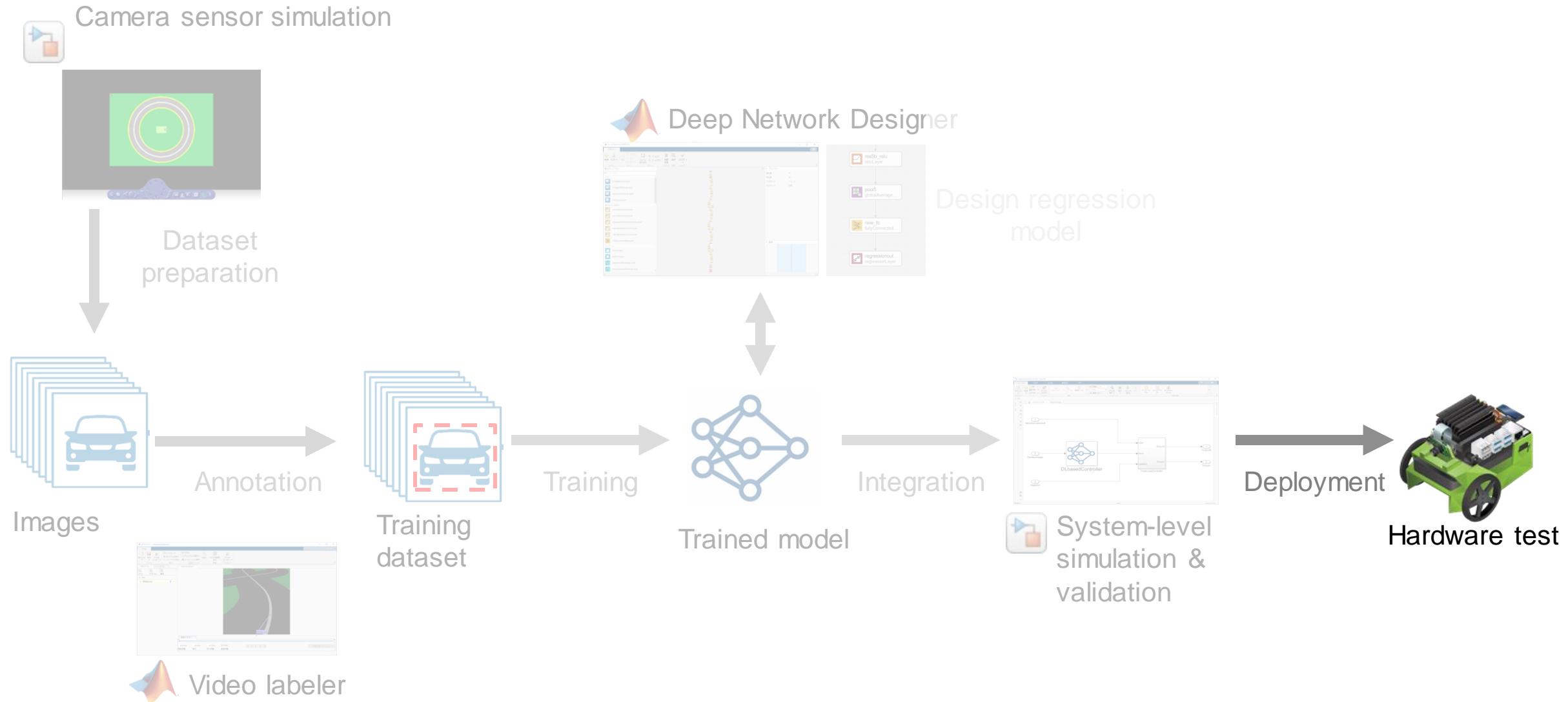


Enabling "Top-view camera" and "To Multimedia File" allows to generate a video file during simulation

# Hints : Improve Line Following Performance

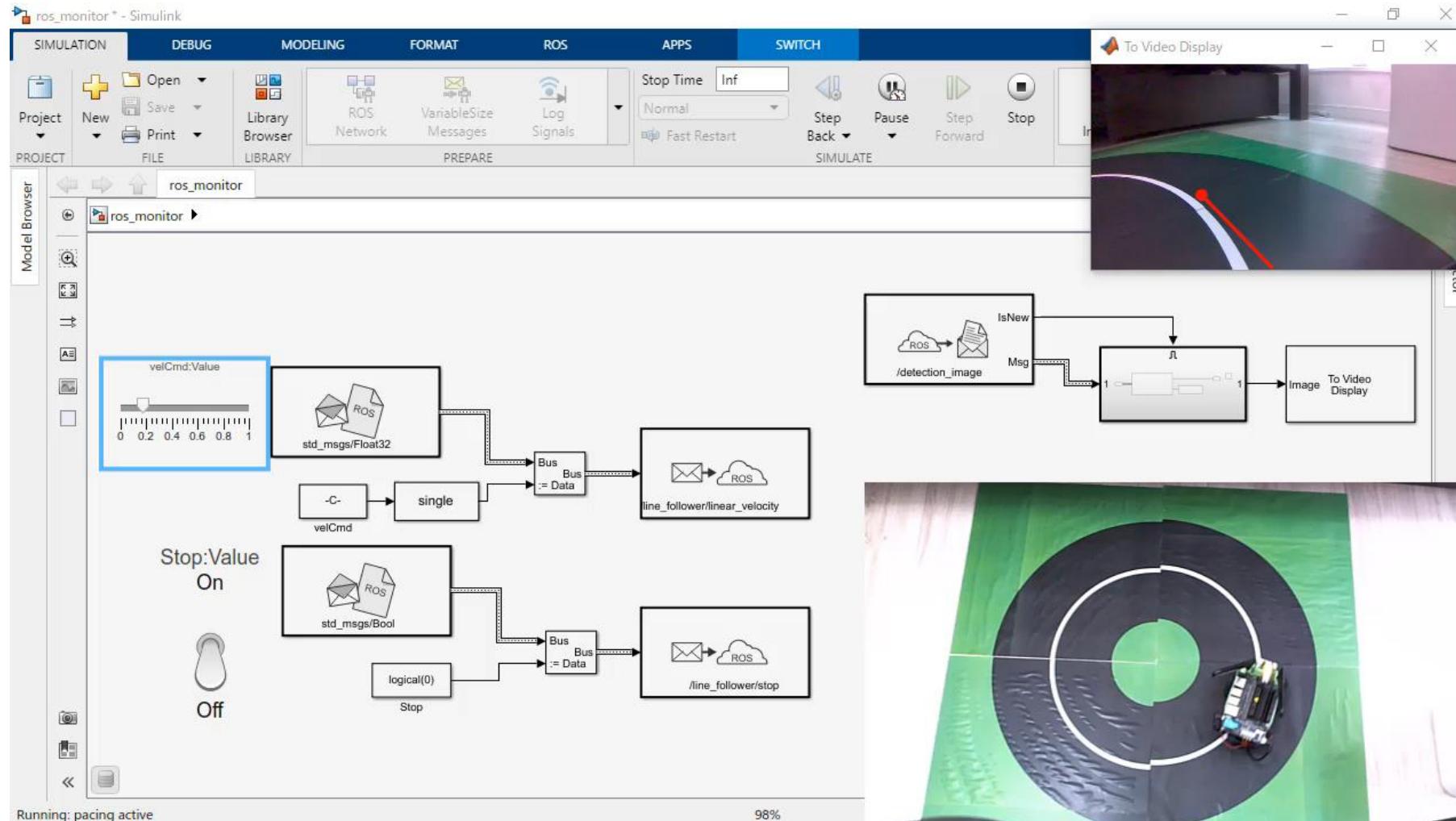
- Increase the training dataset (See ex. 1)
  - Generate more trajectories to generate different perspectives
- Improve the annotation (See ex. 2)
  - Reduce the variation of the target points
- Change the training options (See ex.3)
  - Increase the max number of epochs
  - Change the learning rate and the scheduling
- Change P gain (See ex.4)
  - Tune the P gain in Controller/P Controller block
  - Increase the P gain to improve the following performance
  - Decrease the P gain if oscillating

# AI-based Autonomous System Development Workflow

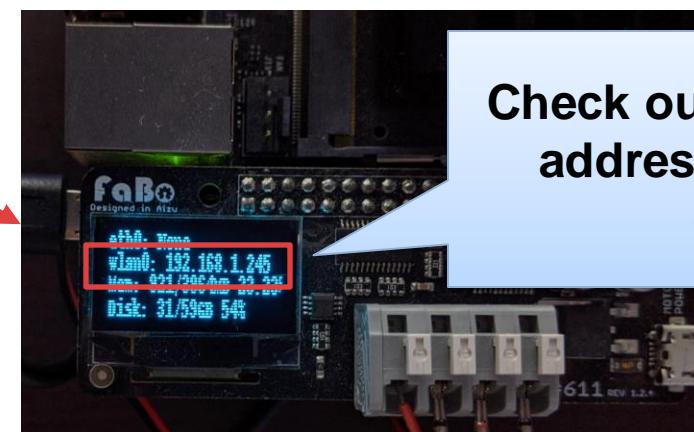
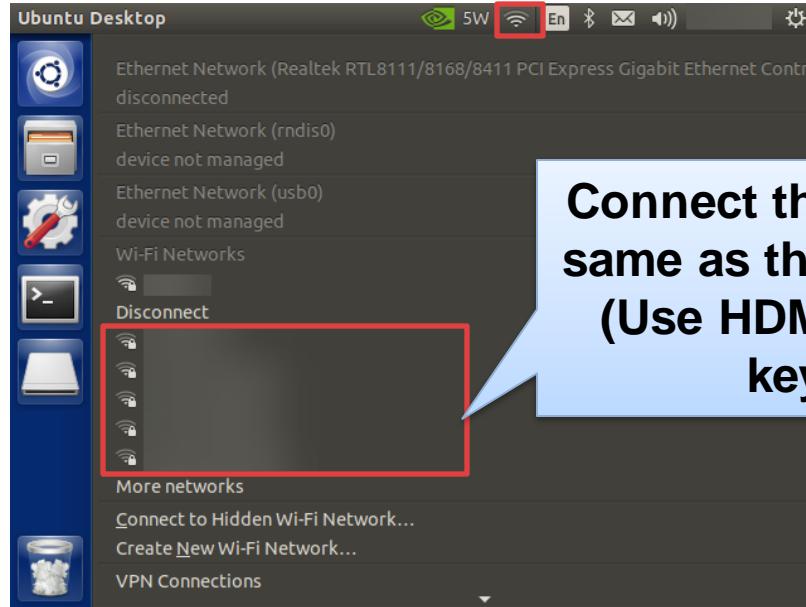
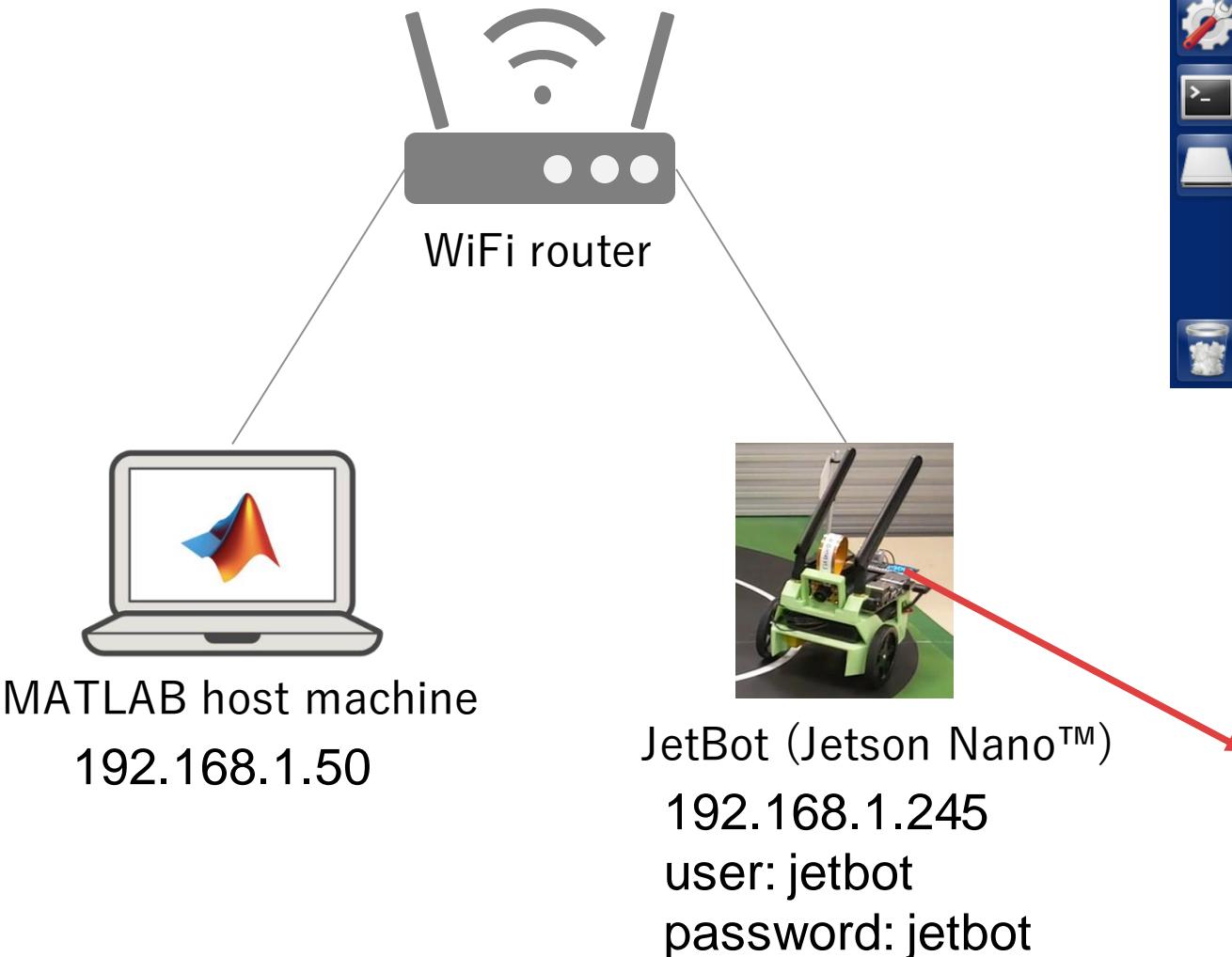


# Goal: Hardware Testing of Line Following Algorithm

Deploy the line tracing algorithm integrating into a middleware, ROS



# Connect MATLAB and JetBot\*



\*JetBot setup instruction can be found in the appendix section

# What is NVIDIA® Jetson Nano™ and NVIDIA® JetBot?

- NVIDIA® low-cost embedded GPU
  - ARM® Cortex®-A57 + NVIDIA® Maxwell
  - GPIO, CSI and other peripherals and I/Os
  - 5W/10W mode, 5V DCin
  - Linux OS
  - NVIDIA® CUDA/cuDNN/TensorRT
- [MATLAB Coder/GPU Coder hardware support package for NVIDIA Jetson](#) is available
- Applications
  - Image classification
  - Object detection
  - Segmentation
  - Audio processing
- Open robot hardware platform
  - NVIDIA® Jetson Nano™ based
  - Simple monocular camera and differential drive robot
  - The BOM list is opened in the GitHub repository
  - Can build the JetBot from scratch
  - <https://jetbot.org/>
- Application
  - Obstacle avoidance
  - Line following
  - Object recognition



# What is ROS?

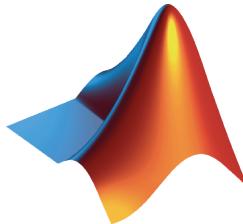
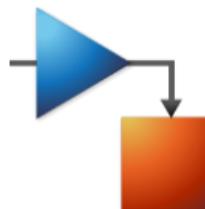


- ROS = Robot Operating System
  - Not an operating system – it's middleware
  - Messaging/distributed computing tools
  - Packaging and build management system
  - Vibrant open-source community for application-related code (drivers, simulation, motion planning, perception, etc.)
  - ROS2 was recently released, and the community is moving to ROS2

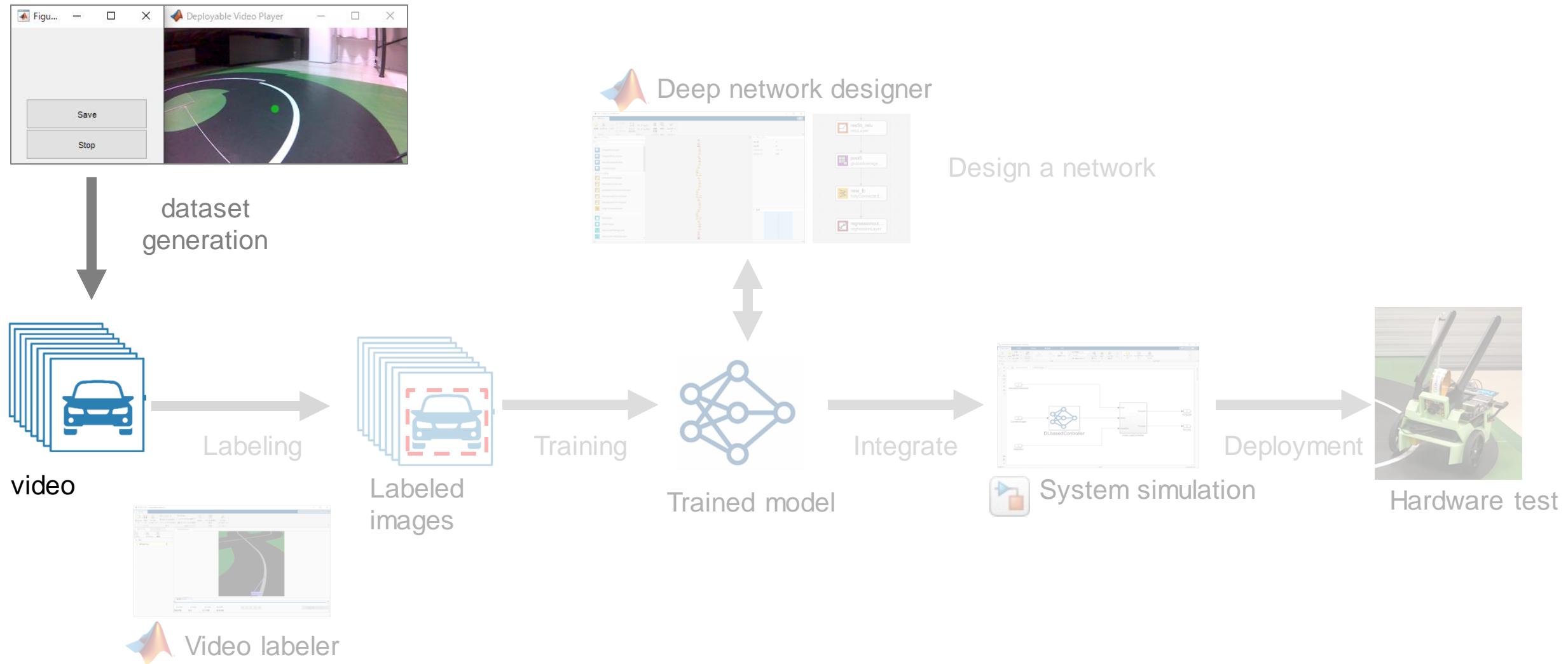
Easy to develop autonomous systems by leveraging ROS

# ROS integration using ROS Toolbox

Used in this workshop

	 ROS	 2
	<ul style="list-style-type: none"><li>• Topic – Publish / Subscribe</li><li>• Service – Server / Client</li><li>• Action – Client</li><li>• Parameter – Get / Set</li><li>• custom message</li><li>• rosbag reading</li><li>• ROS node generation <b>R2021a</b></li></ul>	<ul style="list-style-type: none"><li>• Topic – Publish / Subscribe</li><li>• custom message</li><li>• ros2bag reading <b>R2021a</b></li></ul>
	<ul style="list-style-type: none"><li>• Topic – Publish / Subscribe</li><li>• Service – Call</li><li>• Parameter – Get / Set</li><li>• ROS Time</li><li>• rosbag playback</li><li>• ROS node generation</li></ul>	<ul style="list-style-type: none"><li>• Topic – Publish / Subscribe</li><li>• ROS node generation</li></ul>
ROS Distro	<ul style="list-style-type: none"><li>• ROS Melodic <b>R2020b</b></li></ul>	<ul style="list-style-type: none"><li>• ROS2 Dashing <b>R2020a</b></li></ul>

# Step1: ROS connection and collecting training images

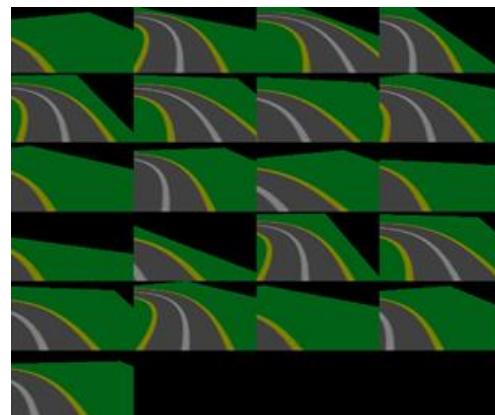


# Difference between simulation and real

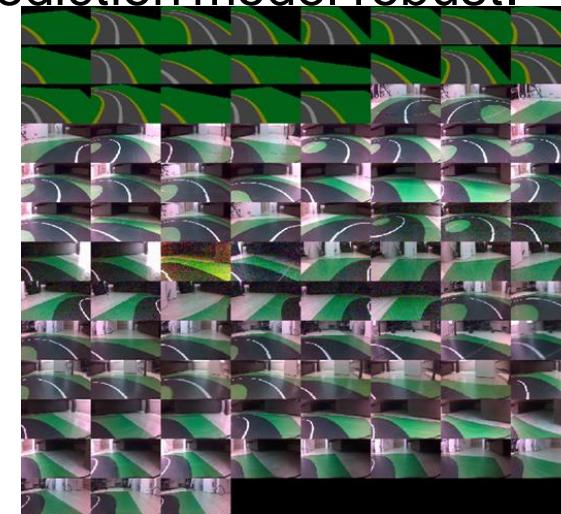
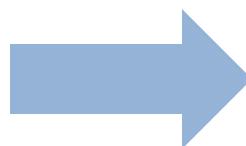
Trained model with the dataset from simulation may not work for the real environment due to the following factors:

- Background objects
- Lighting condition
- Image sensor noise

Collect images from the hardware to make the prediction model robust.



Collected in simulation

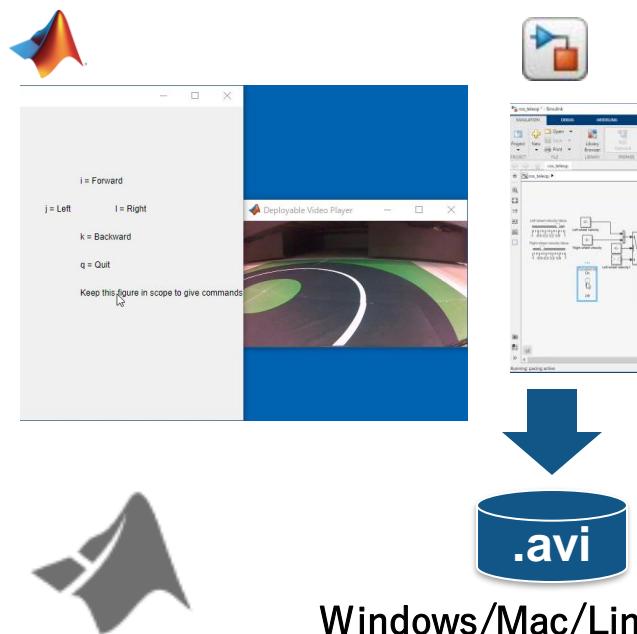


Images taken from the hardware added

# ROS connection and collecting training images

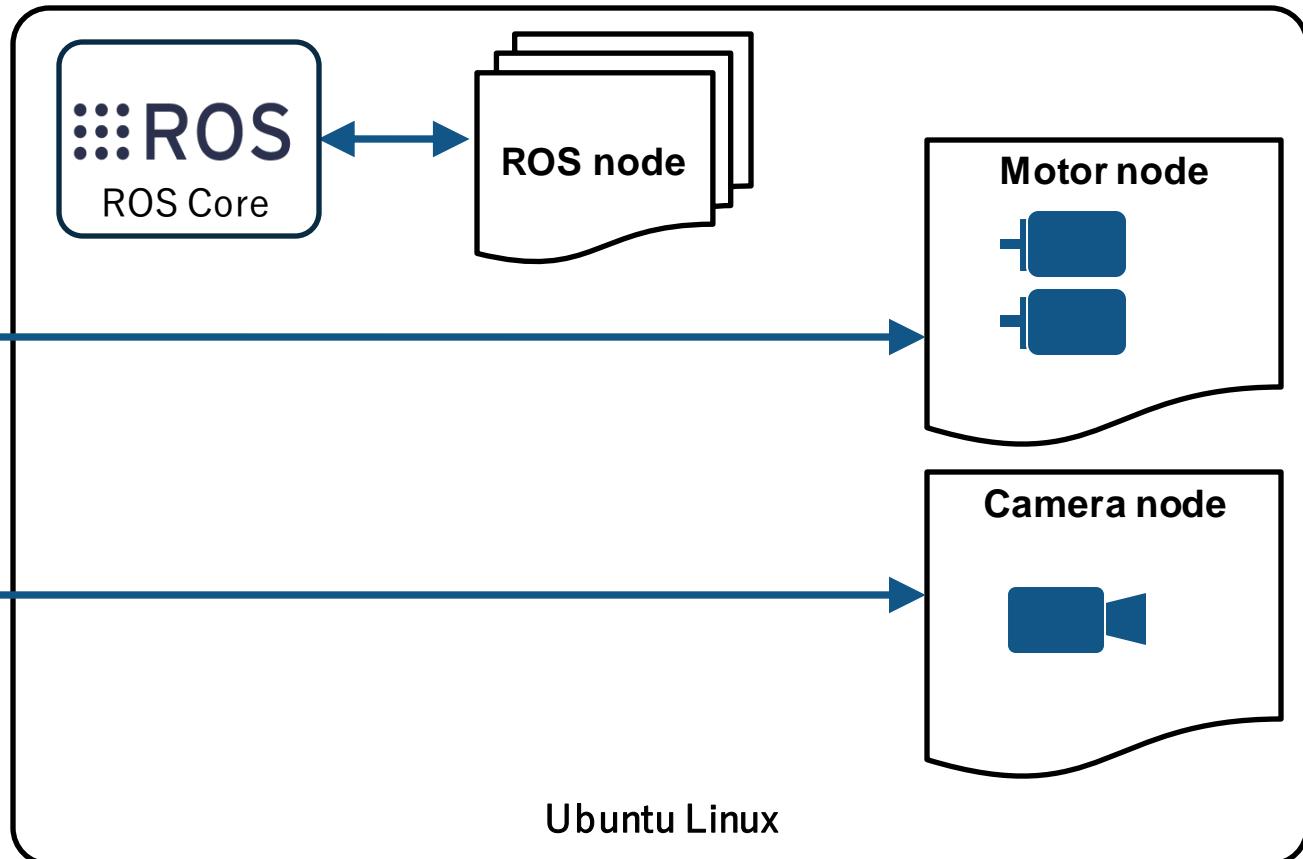
Connect MATLAB and Simulink with JetBot

- Send the motor rotation speeds
- Receive the camera images



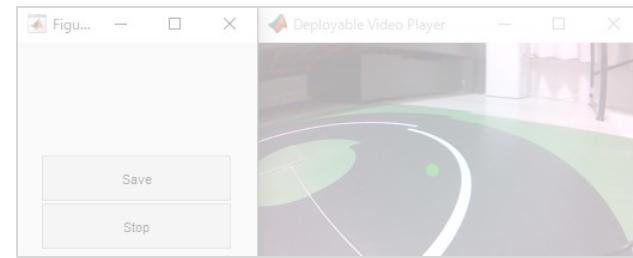
Development Machine

```
>> edit teleopTestJetBot.mlx
```



Locate the JetBot to get the variety of the images or Drive the JetBot

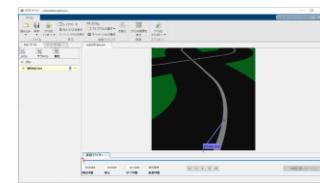
## Step2: Improve line detector



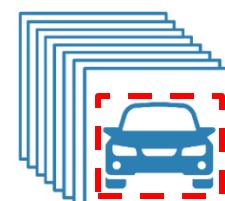
dataset  
generation



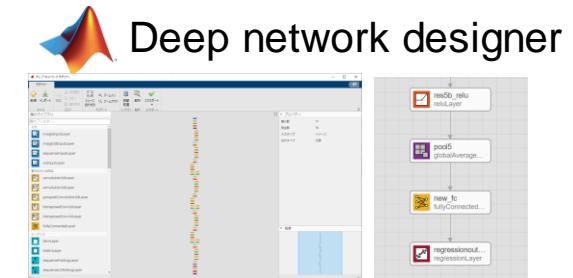
video



Video labeler

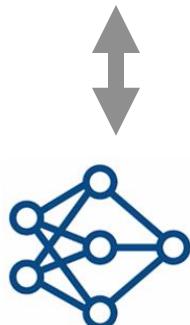


Labeled  
images

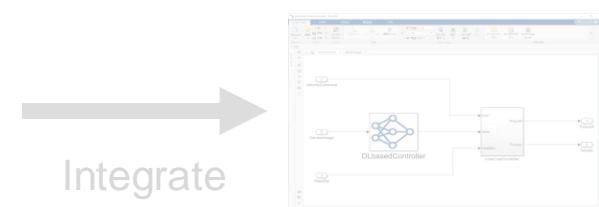


Deep network designer

Design a network



Trained model



System simulation

Integrate



Hardware test

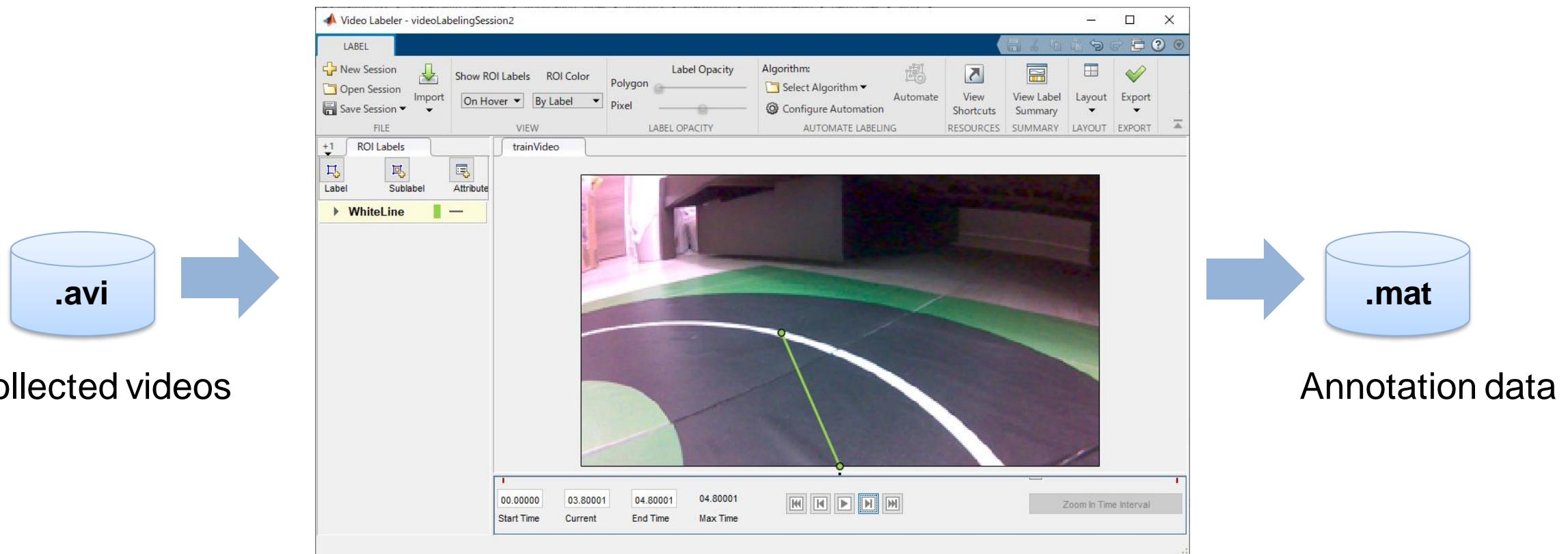
Deployment

Training

Labeling

# Labeling

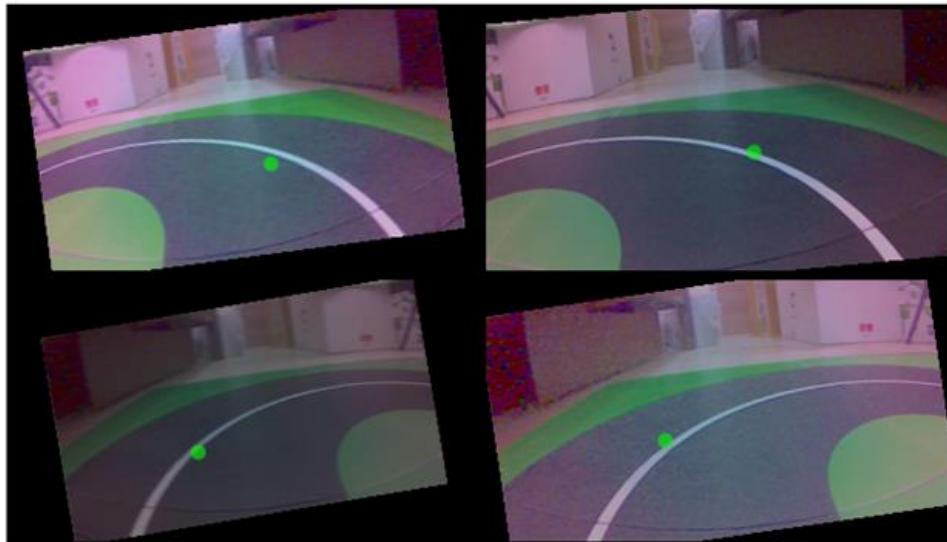
- Annotate the images taken from the hardware as well as simulation's one



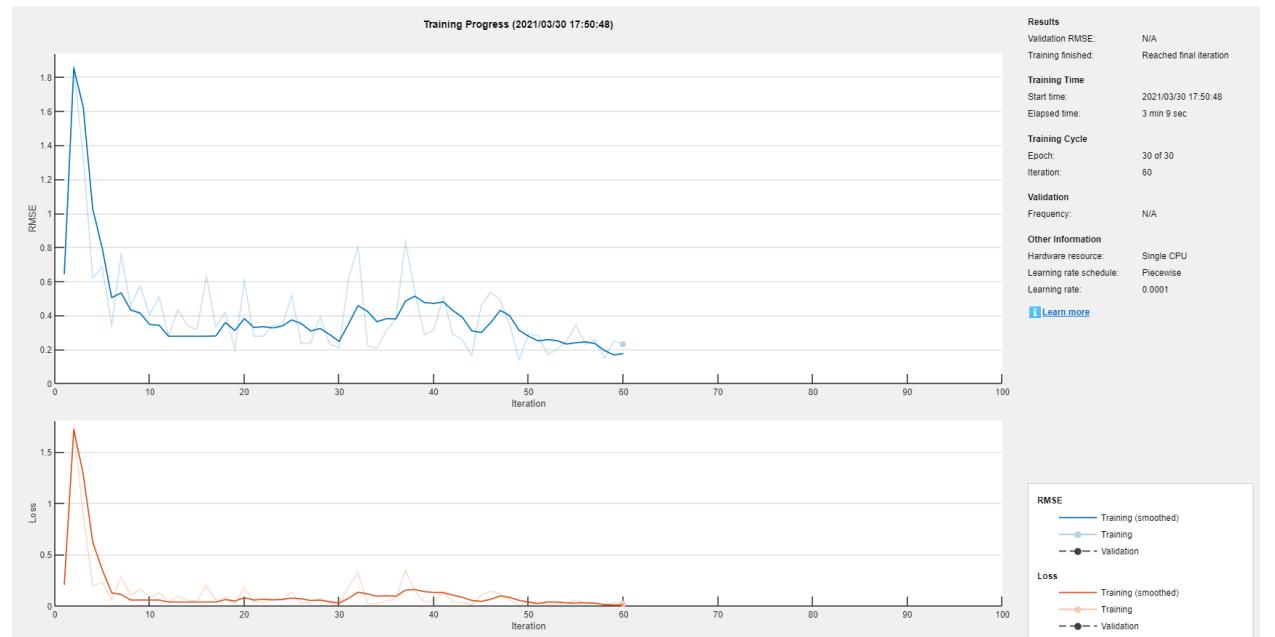
```
>> edit trainWhiteLineWithHardware.mlx
```

# Data augmentation and retraining

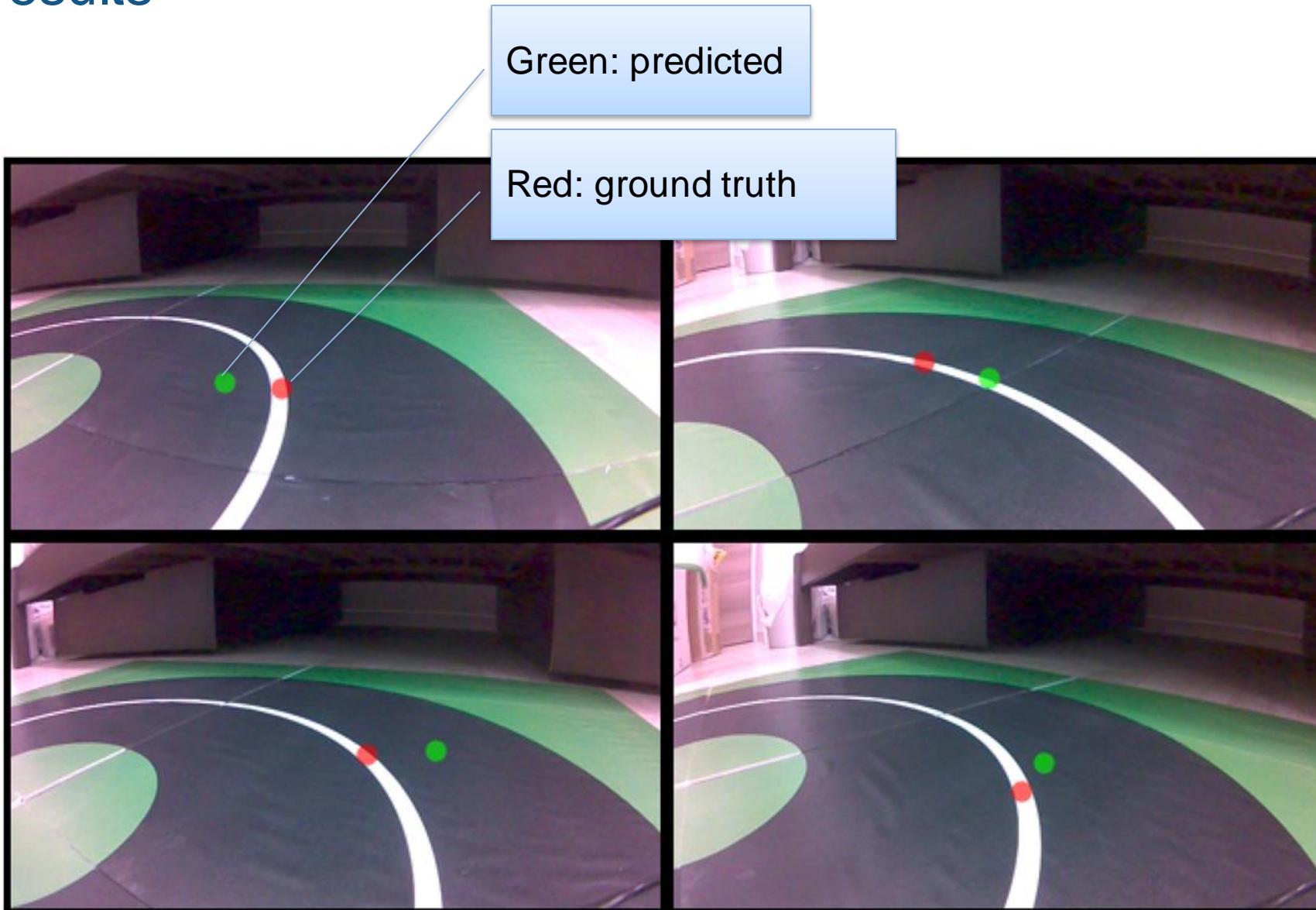
- Retraining the line detector
- Augment the image to mimic the environment change



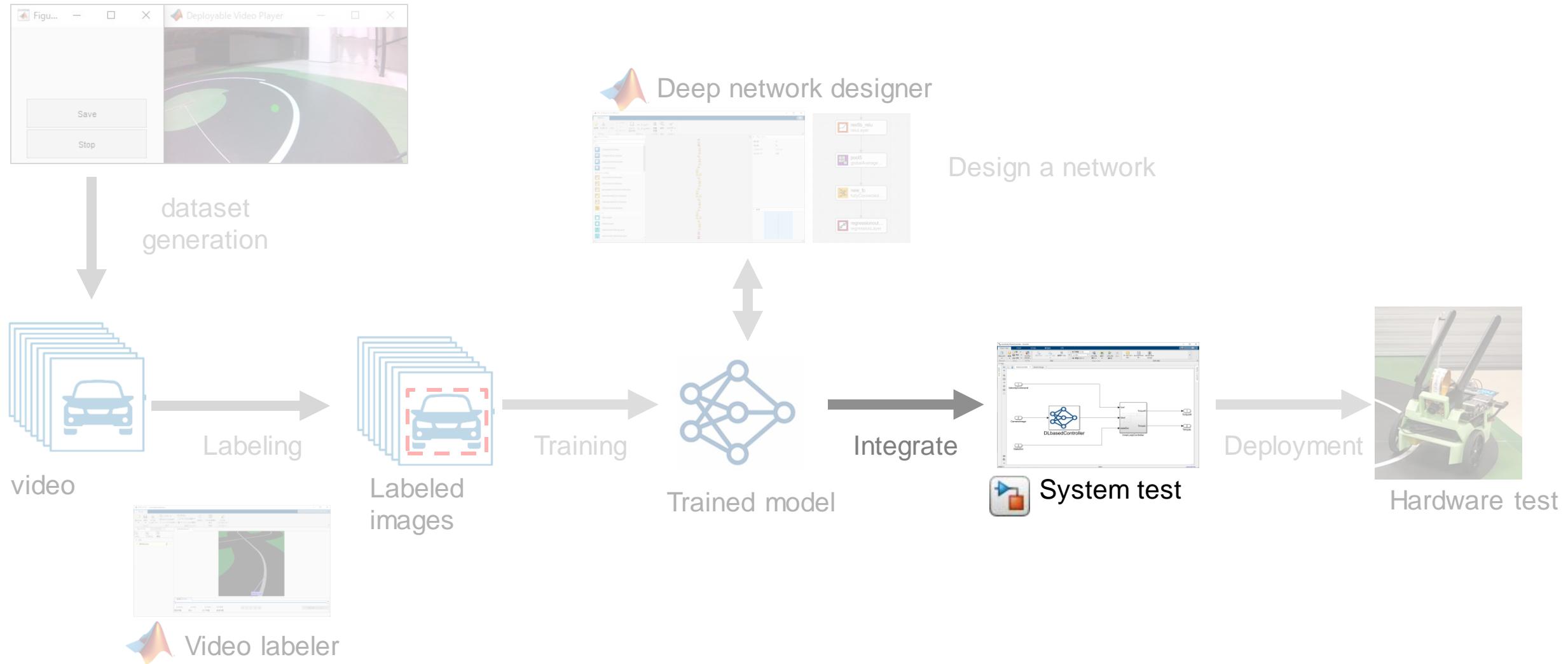
Augment the images



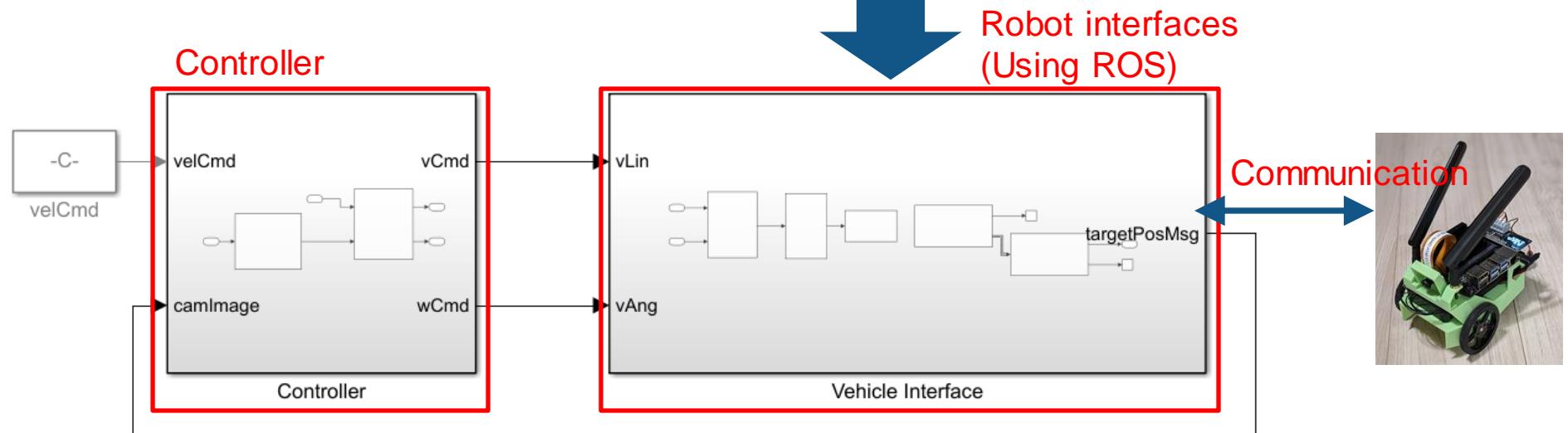
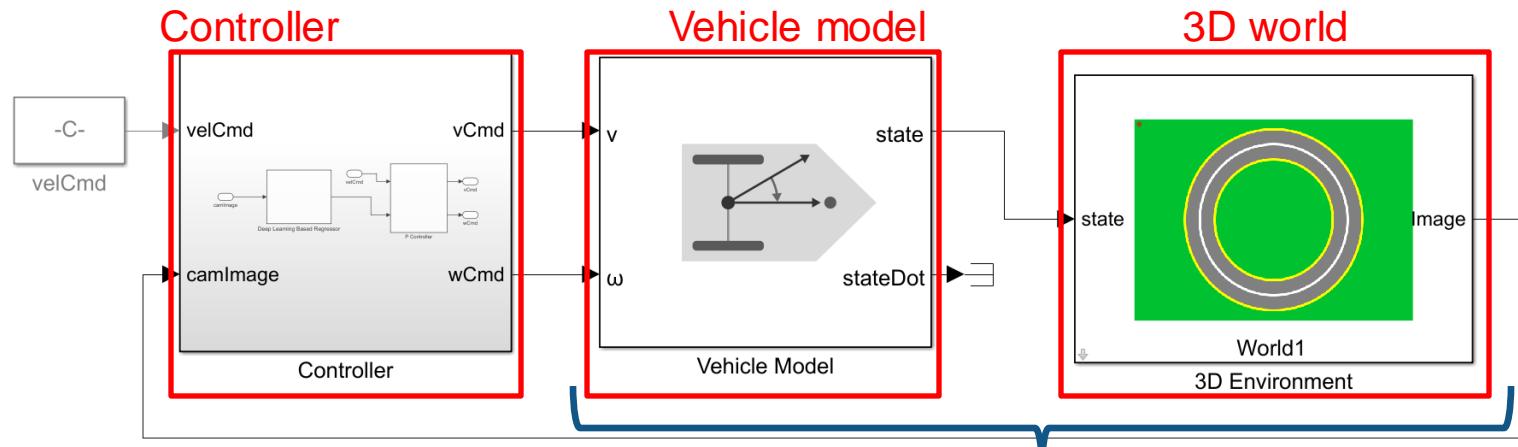
# Training results



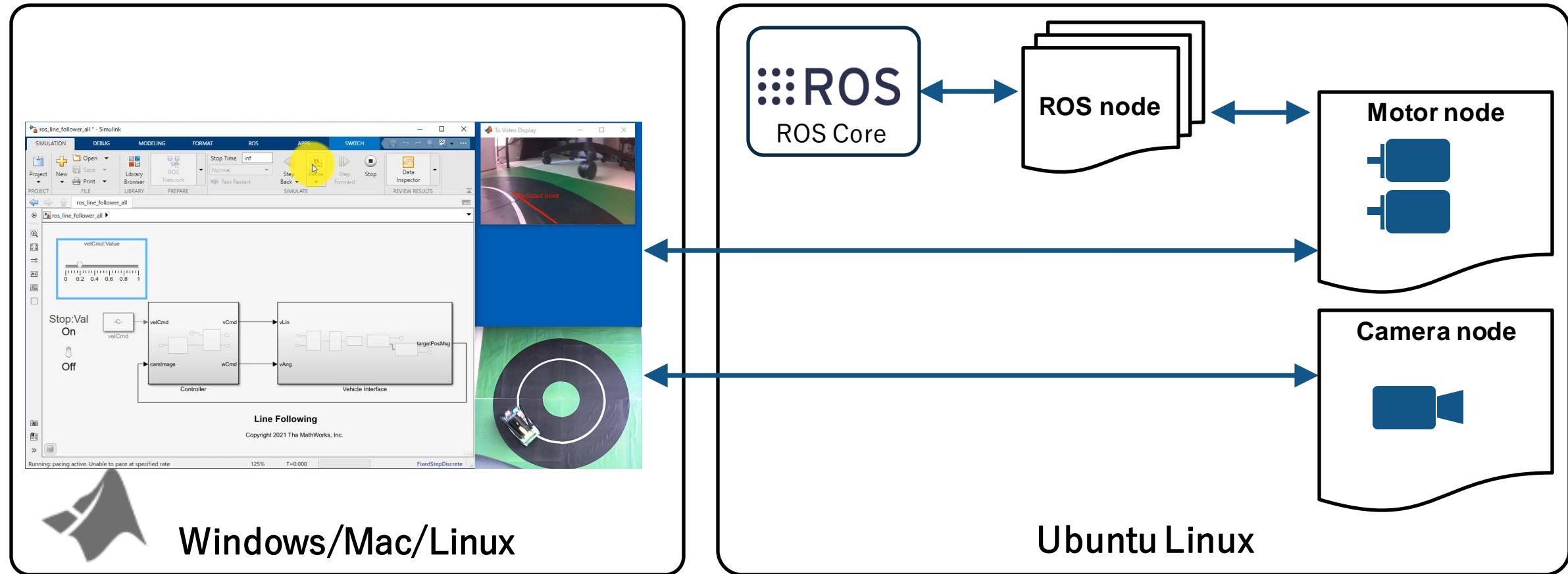
# Step3: Test algorithms through hardware connection



# Replace the simulation models with ROS interfaces



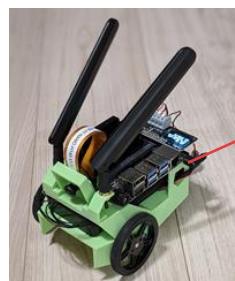
# Algorithm test through hardware communication



Development Machine

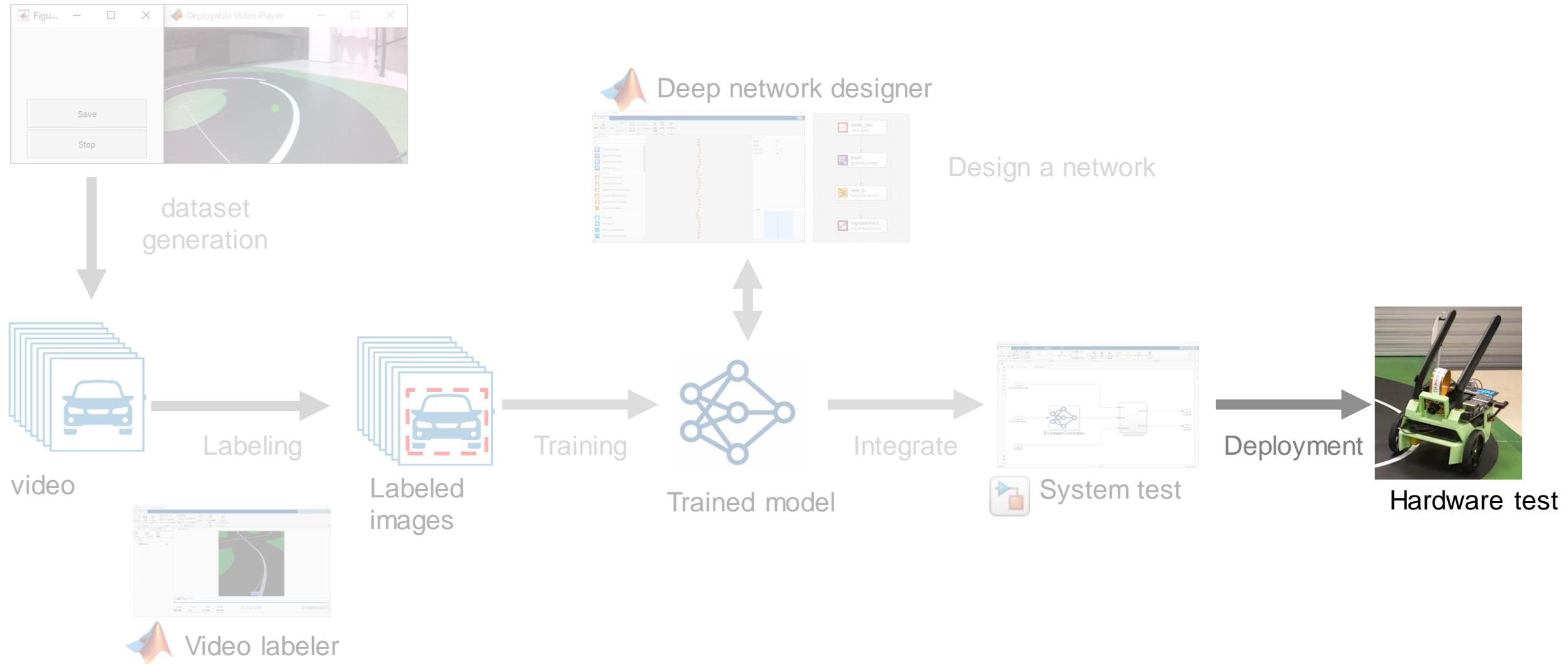
```
>> edit testLineFollowerWithROS.mlx
```

Communication delay may affect, but the functional algorithm test can be done.

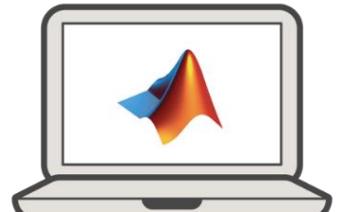
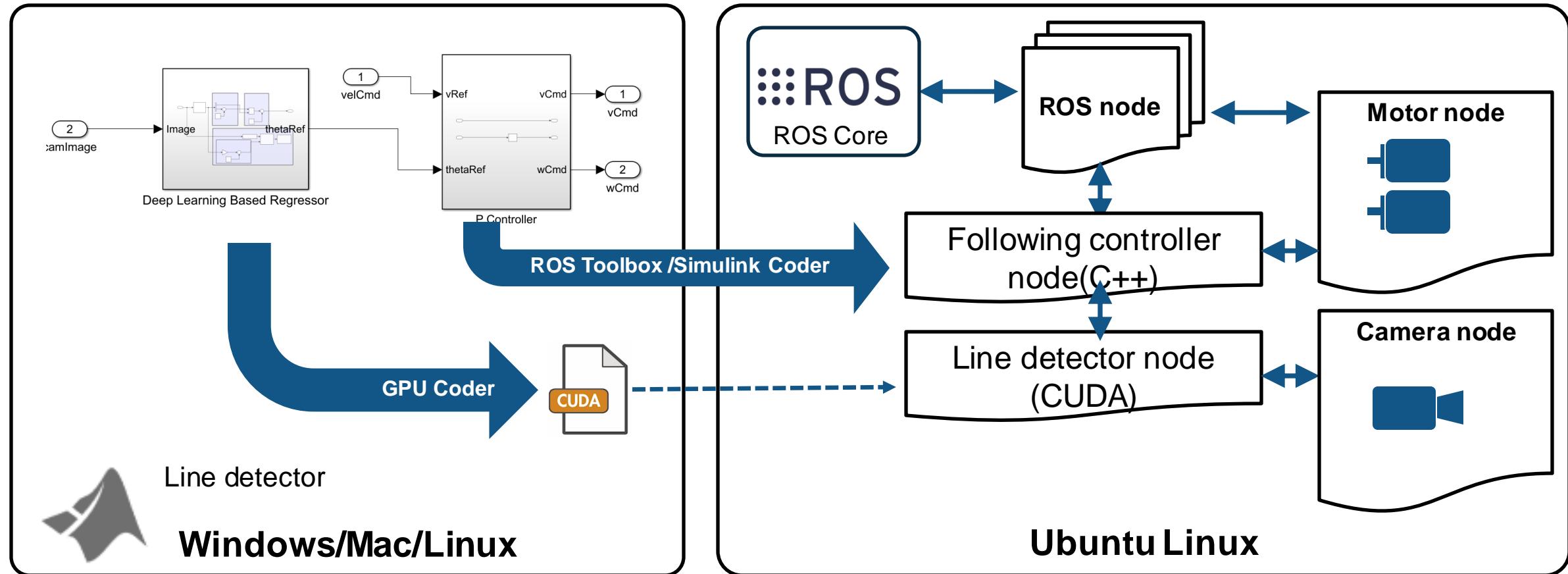


Jetson Nano™

# Step4: Standalone hardware deployment using code generation



# Generate ROS nodes and run them on the hardware



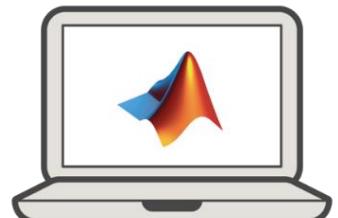
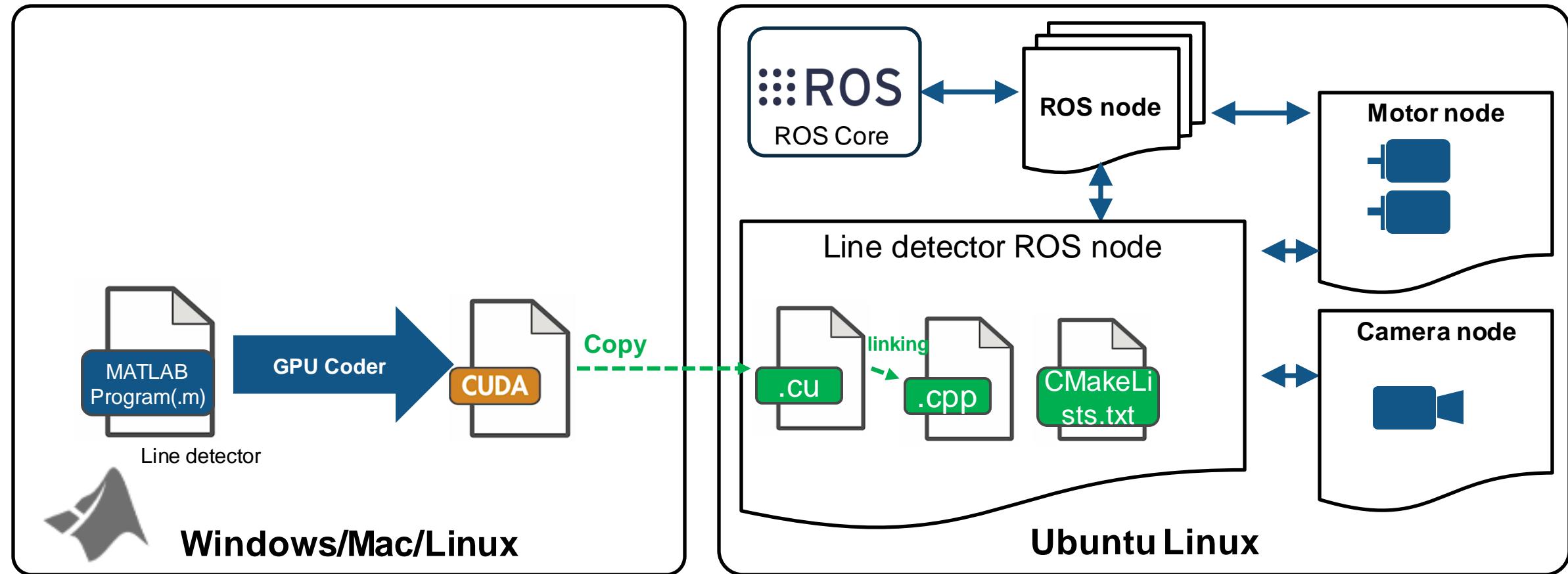
Development Machine

Split the original Simulink model into the detector part and the following controller part and then generate ROS nodes, respectively.



Jetson Nano™

# CUDA code generation and wrap it as ROS node



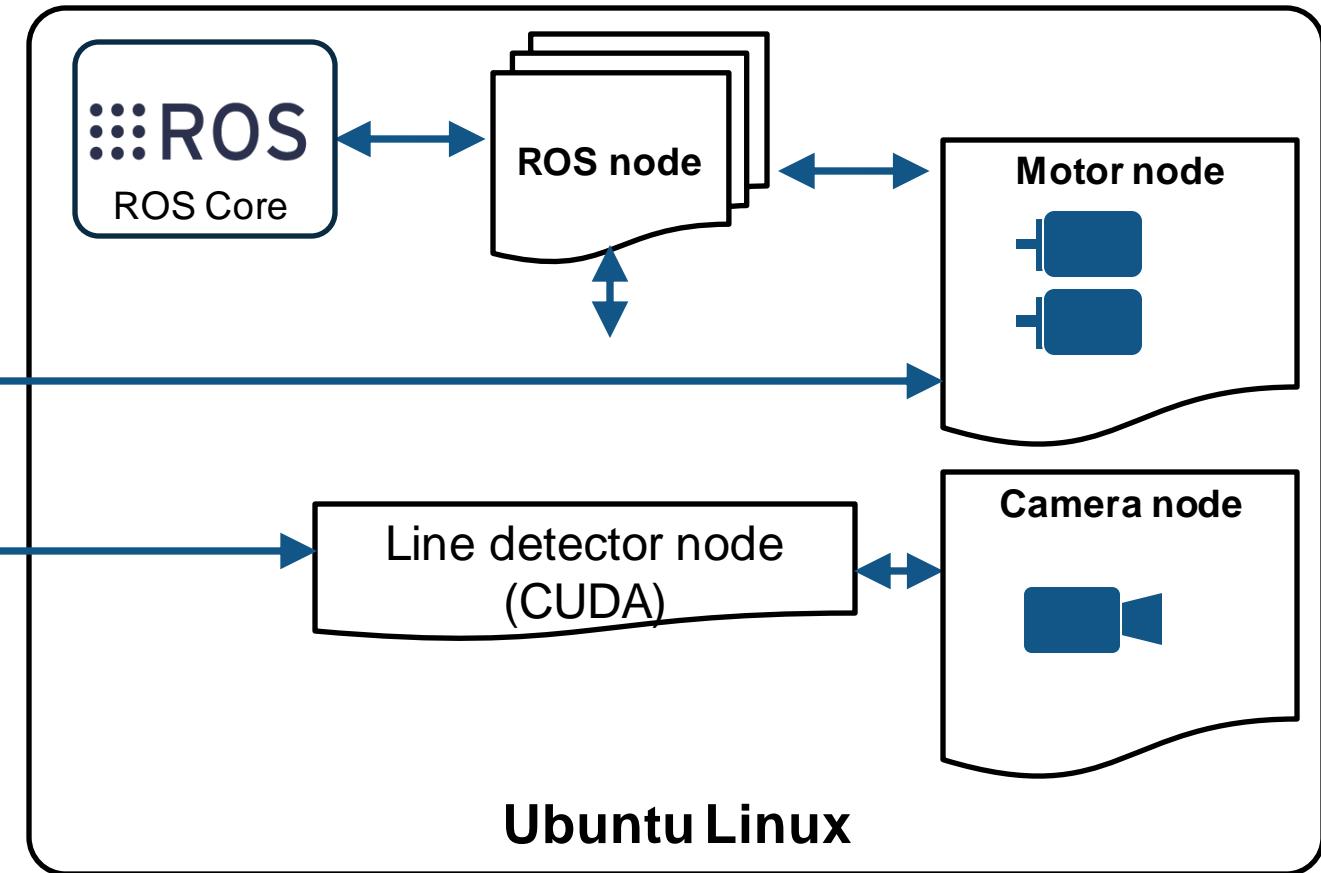
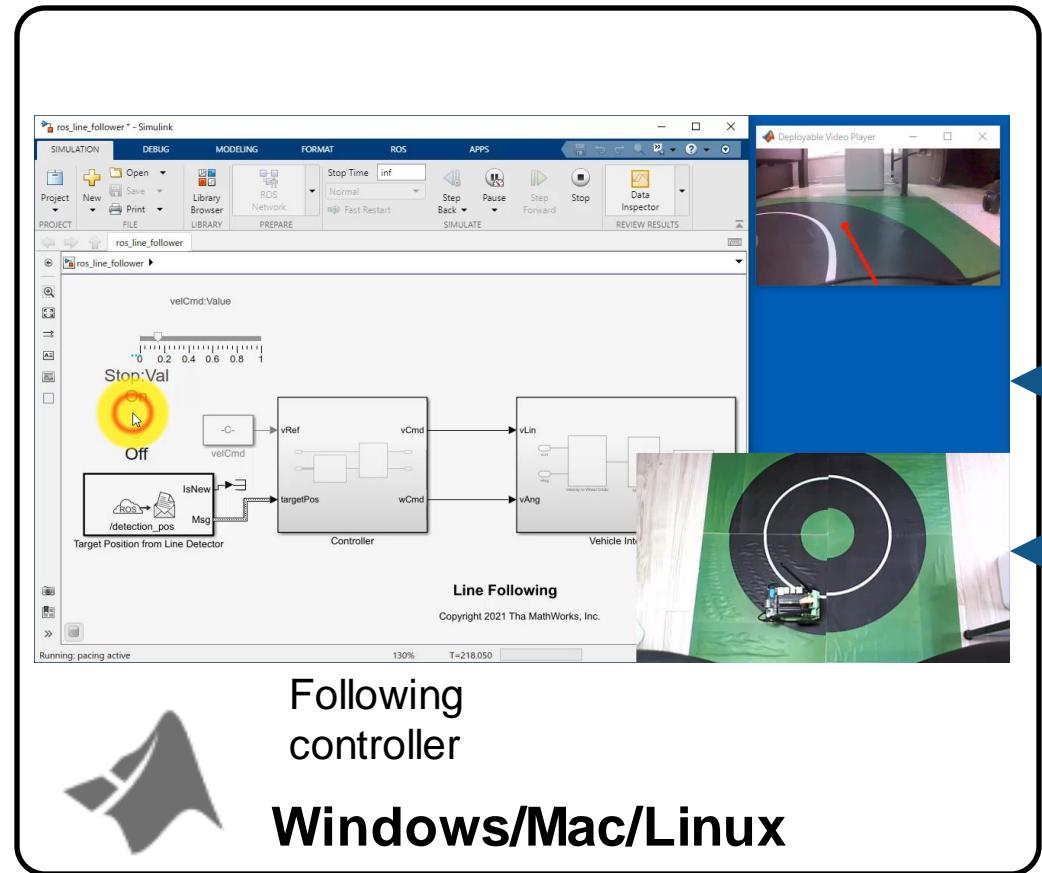
Development Machine

CUDA code generation using GPU  
Coder and wrap the CUDA code as a  
ROS node

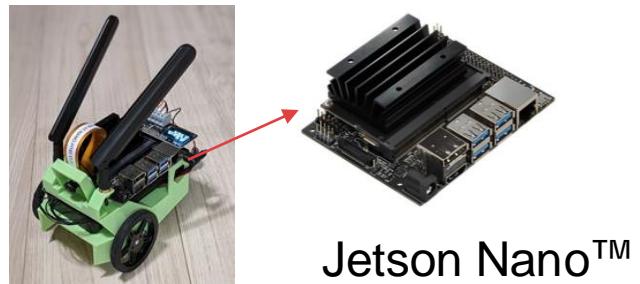


Jetson Nano™

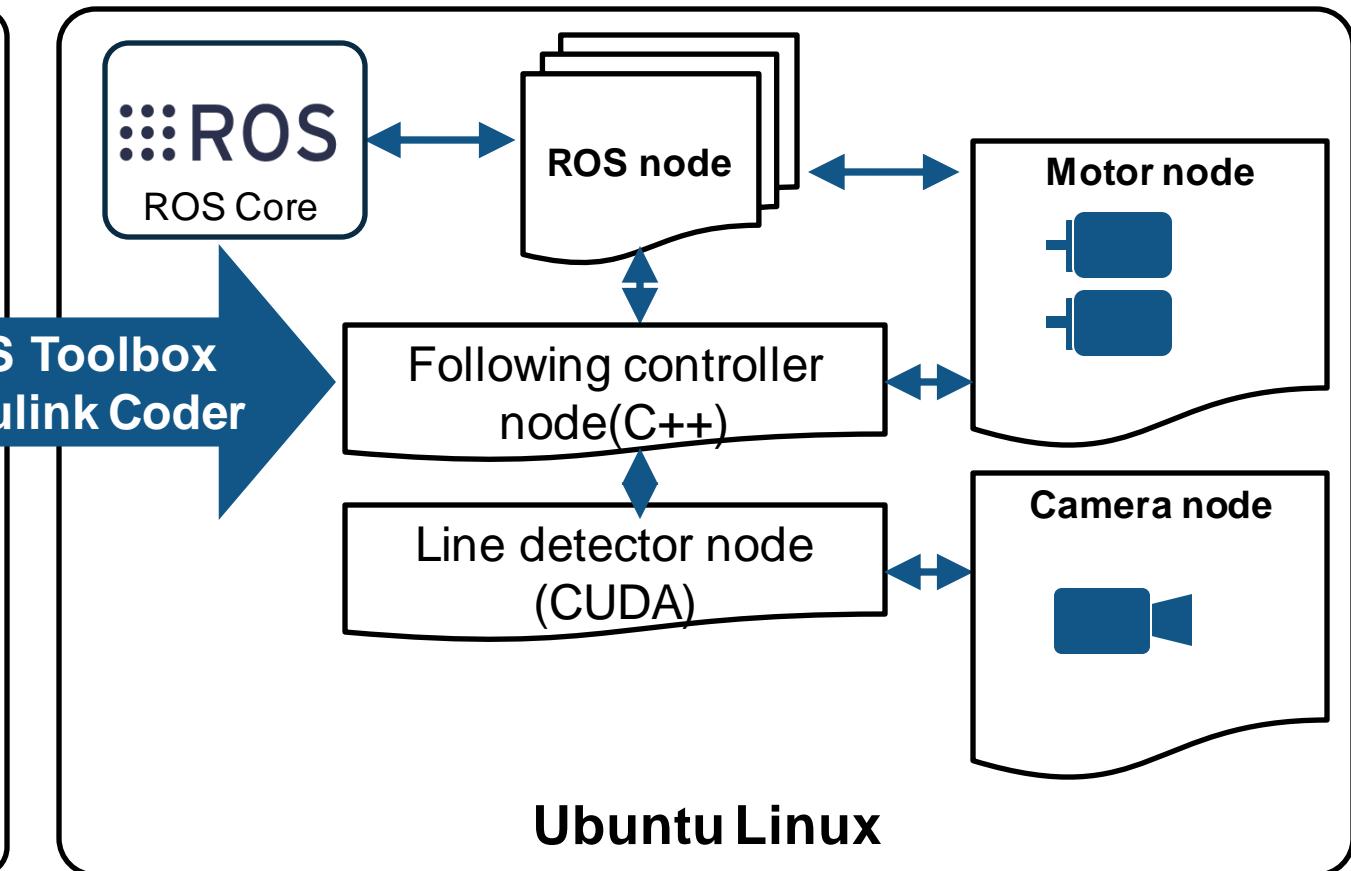
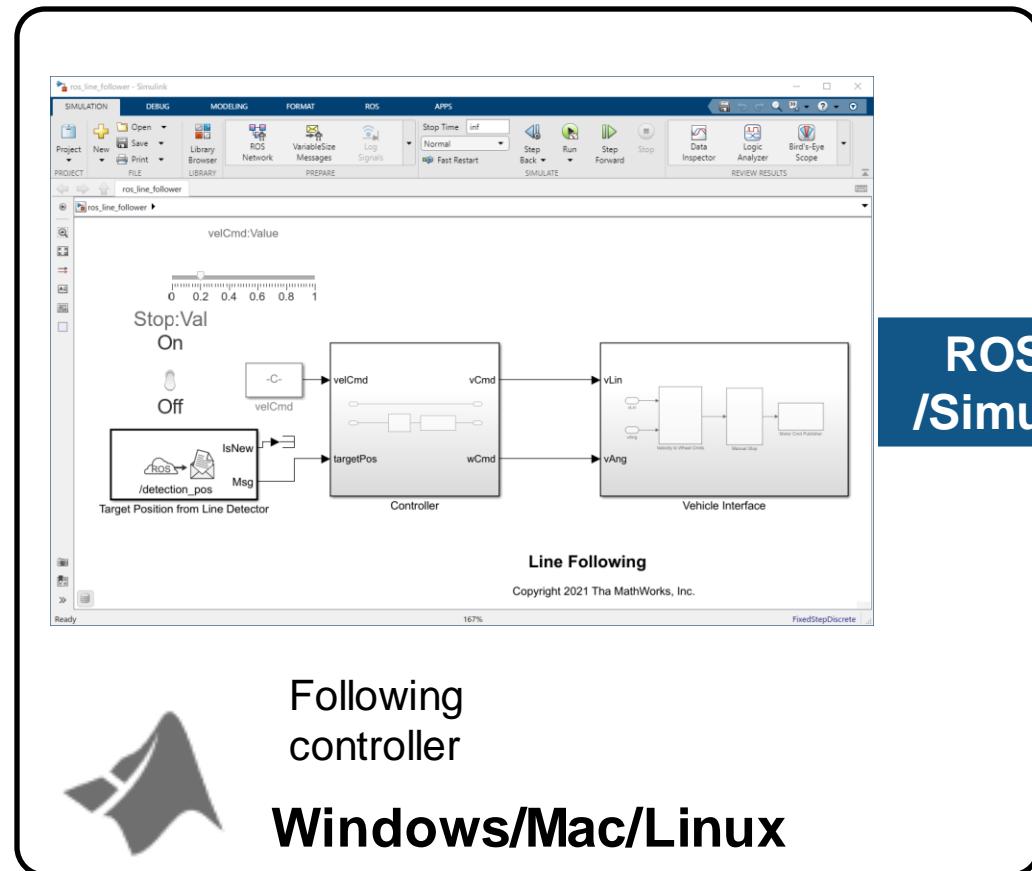
# Co-execution with deployed line detector



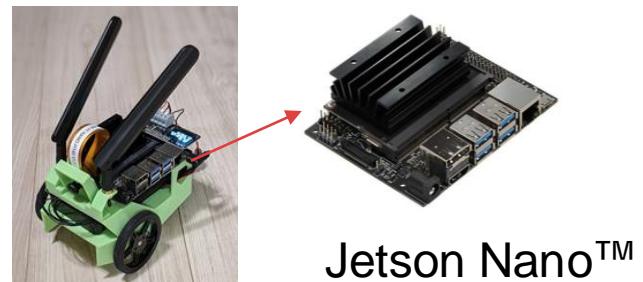
Co-execution with the deployed line detector to check the behavior of the following controller model.



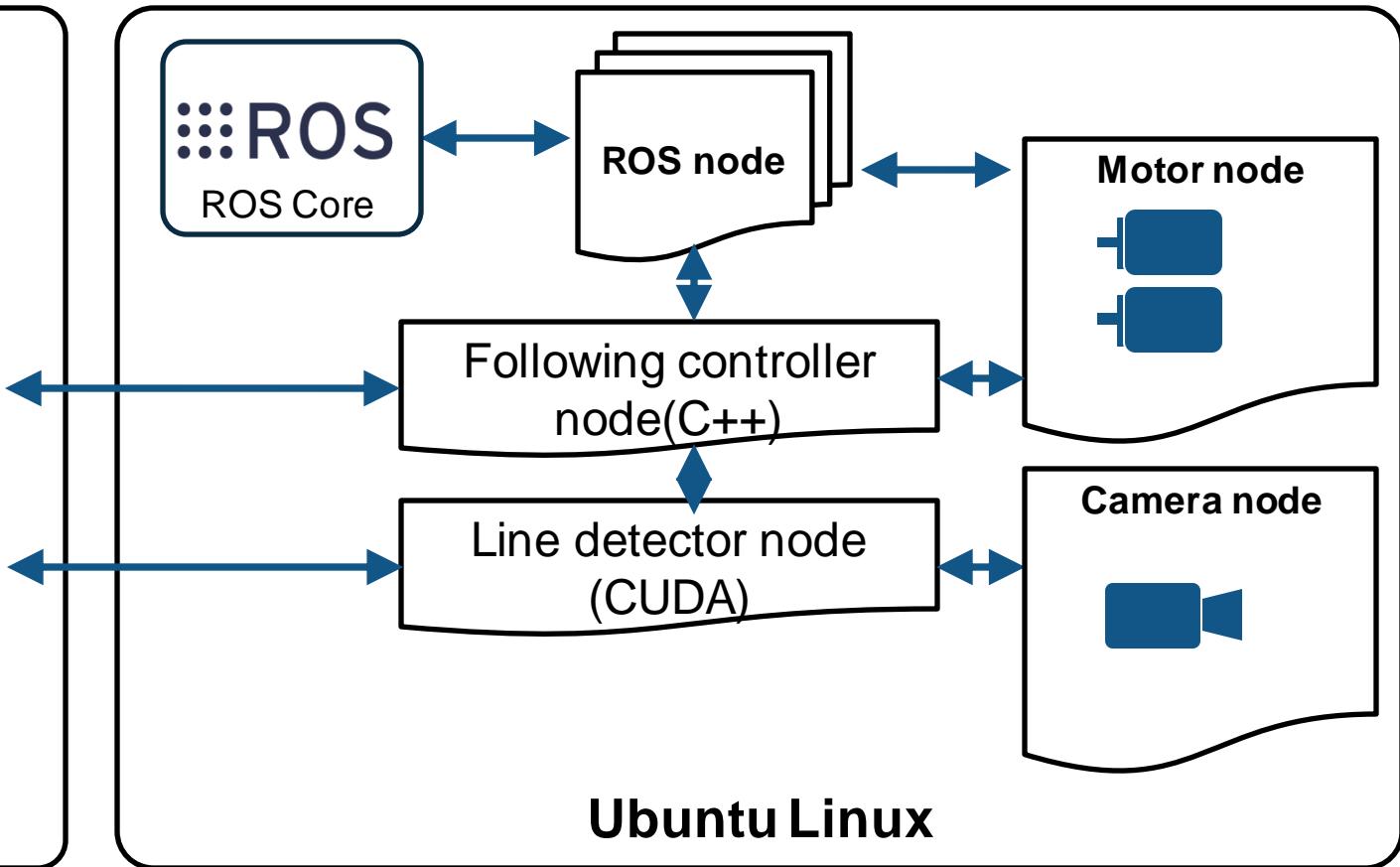
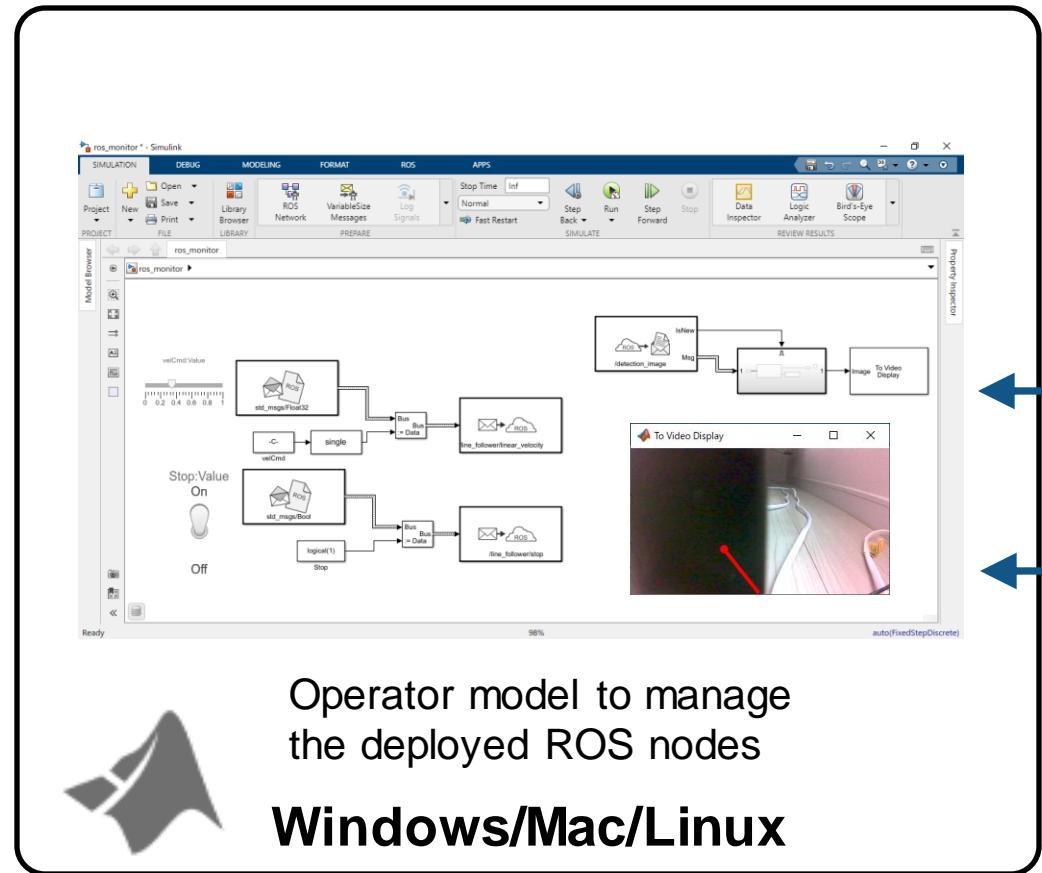
# Deploy the following controller model as ROS node



Deploy the following controller model as ROS node using ROS Toolbox and Simulink Coder



# Standalone execution with all deployed ROS nodes



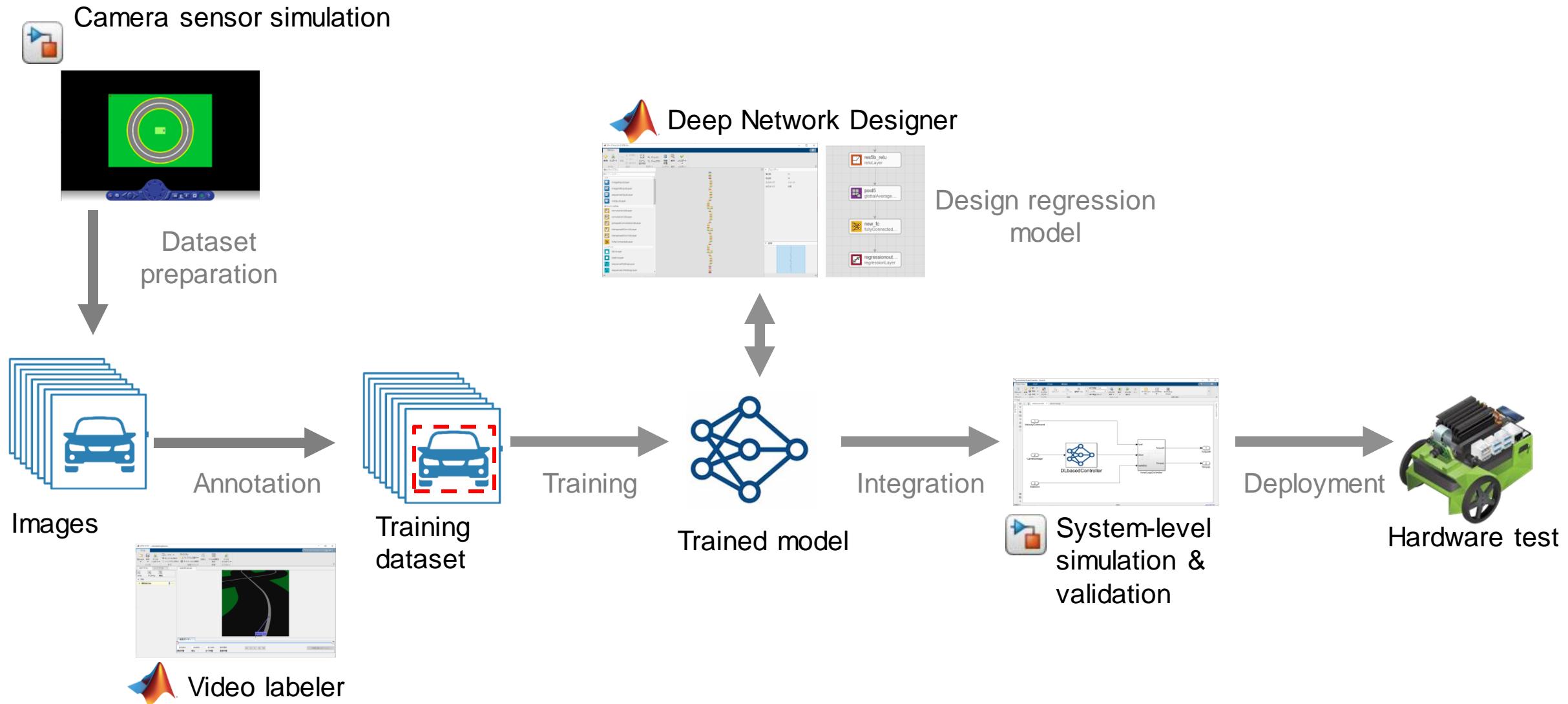
Development Machine

Standalone execution on the target  
Another Simulink to specify the target velocity  
and the start/stop operation.



Jetson Nano™

# Recap: AI-based Autonomous System Development Workflow



# Appendix A. Preparation for simulation

# Workshop host machine

- System requirements
  - <https://mathworks.com/support/requirements/matlab-system-requirements.html>
- Required software
  - MATLAB R2021a or later
  - Simulink
  - Image Processing Toolbox
  - Computer Vision Toolbox
  - Deep Learning Toolbox
  - Robotics System Toolbox
  - Simulink 3D Animation
  - Deep Learning Toolbox Model for ResNet-18 Network
    - Type the below command and follow the appeared instruction
    - `>> resnet18`
  - Parallel Computing Toolbox (if NVIDIA® GPU is available)

# Appendix B. hardware deployment (Setting up a JetBot)

# Development environment : MathWorks Products

- The following toolboxes are additionally required to the simulation part
- Toolbox
  - MATLAB Coder
  - GPU Coder
  - Simulink Coder
  - ROS Toolbox
- Add-ons
  - MATLAB Coder Support Package for NVIDIA® Jetson™ and NVIDIA® DRIVE™ Platforms
    - The installation procedure can be found below
    - <https://www.mathworks.com/help/supportpkg/nvidia/ug/install-support-for-nvidia-devices.html>
  - GPU Coder Interface for Deep Learning Libraries

# Development environment for hardware deployment

- Hardware
  - NVIDIA® JetBot (Tested with [FaBo JB-4GB-S-G](#))
    - Jetson Nano™ 4GB
- Software
  - 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 package
    - [https://github.com/dusty-nv/jetbot\\_ros](https://github.com/dusty-nv/jetbot_ros)
    - Tested with the revision c7d0e611b037947a0df855293c9848a7c8bc6e90 by patching
    - ROS Melodic + JetBot's camera and motor ROS driver nodes
  - Jetson CSI camera ROS nodelet package
    - [https://github.com/sfalexrog/jetson\\_camera](https://github.com/sfalexrog/jetson_camera)
    - Tested with rev. 7ec8e364880fce5bbe31800bd6224af2370ca1b

# Setting up JetBot

- Setup JetBot by following vendor's instructions
  - Tested with [FaBo's JB-4GB-S-G](#)
  - Follow vendor's instruction manual to setup
- Setup JetBot's software
  - [https://jetbot.org/v0.4.3/software\\_setup/sd\\_card.html](https://jetbot.org/v0.4.3/software_setup/sd_card.html)
  - Use jetbot\_image\_v0p4p0.zip (7.3GB) as an SD card image

# Setting up JetBot

- Download "jetbot\_image\_v0p4p0.zip"

The screenshot shows the GitHub repository for JetBot. At the top, there is a search bar and a GitHub icon. Below the search bar, there is a table titled "Using SD Card Image" and a "Table of contents" sidebar.

**Using SD Card Image Table:**

Platform	JetPack Version	JetBot Version	Download	MD5 Checksum
Jetson Nano 2GB	4.5	0.4.3	<a href="#">jetbot-043_nano-2gb-jp45.zip</a>	e6dda4d13b1b1b31f648402b9b742152
Jetson Nano (4GB)	4.5	0.4.3	<a href="#">jetbot-043_nano-4gb-jp45.zip</a>	760b1885646bfad8590633acca014289

**Table of contents:**

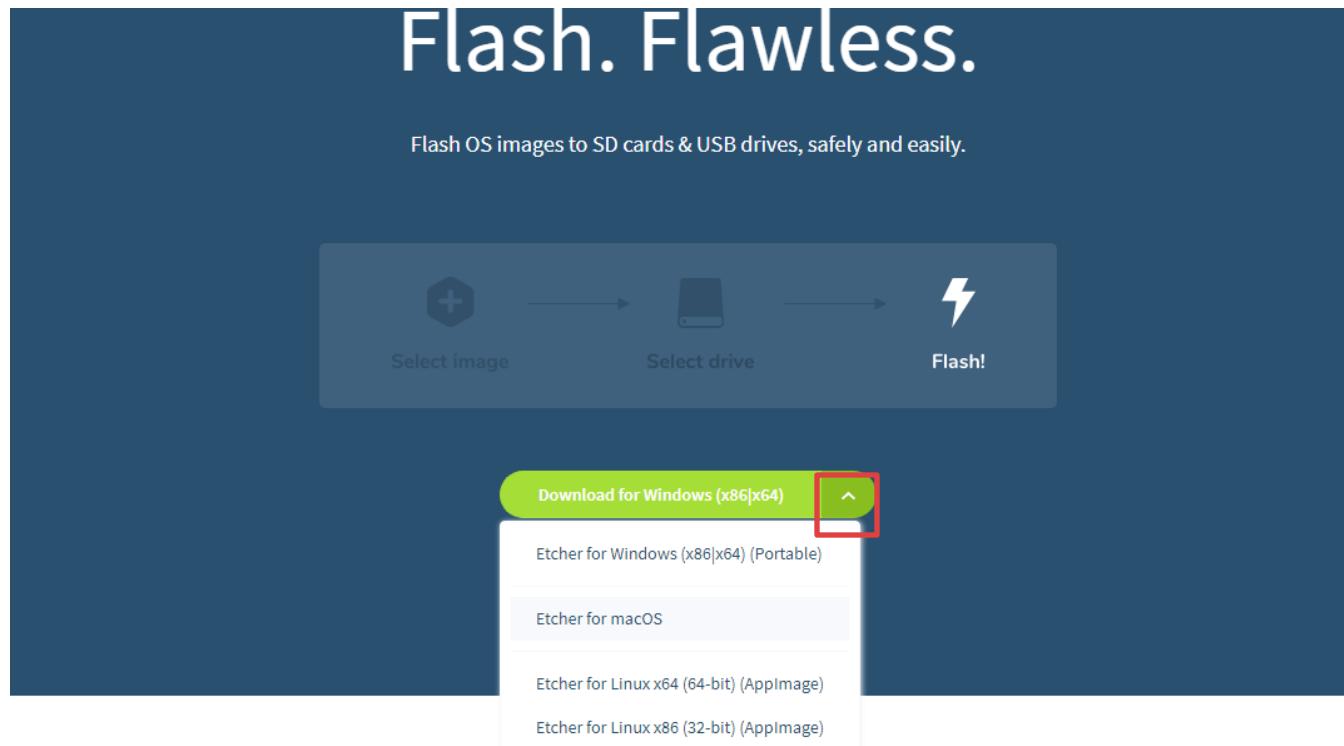
- Step 1 - Download the pre-built JetBot SD card image
- Latest Release
- Old releases
- Step 2 - Flash JetBot image onto SD card
- Step 3 - Boot Jetson Nano
- Step 4 - Connect JetBot to WiFi
- Step 5 - Connect to JetBot from web browser

**Old releases Table:**

Platform	JetPack Version	JetBot Version	Download
Jetson Nano 2GB	4.4.1	0.4.2	<a href="#">jetbot-042_nano-2gb-jp441.zip</a>
Jetson Nano (4GB)	4.4.1	0.4.2	<a href="#">jetbot-042_nano-4gb-jp441.zip</a>
Jetson Nano 2GB	4.4.1	0.4.1	<a href="#">jetbot-041_nano-2gb-jp441.zip</a>
Jetson Nano (4GB)	4.4.1	0.4.1	<a href="#">jetbot-041_nano-4gb-jp441.zip</a>
Jetson Nano (4GB)	4.3	0.4.0	<a href="#">jetbot_image_v0p4p0.zip</a>
Jetson Nano (4GB)	4.2	0.3.2	<a href="#">jetbot_image_v0p3p2.zip</a>

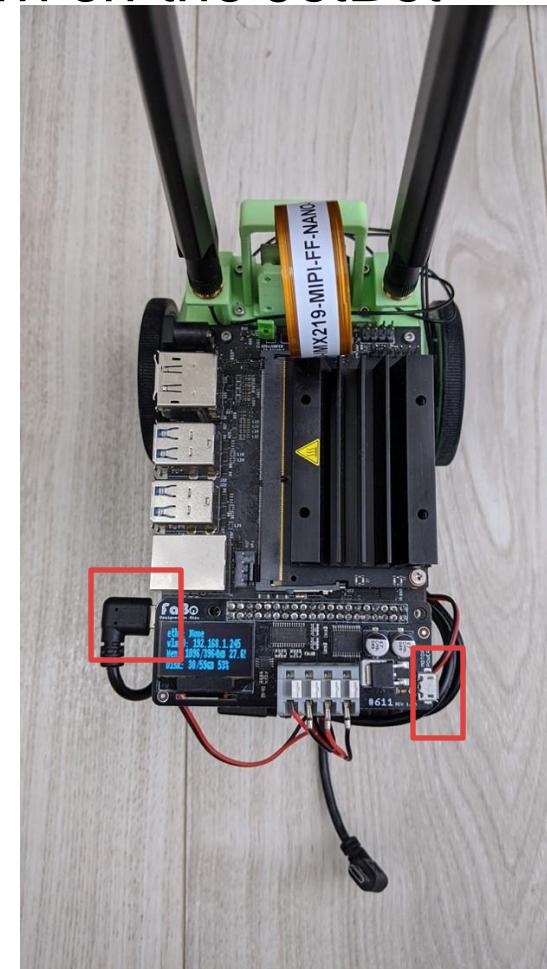
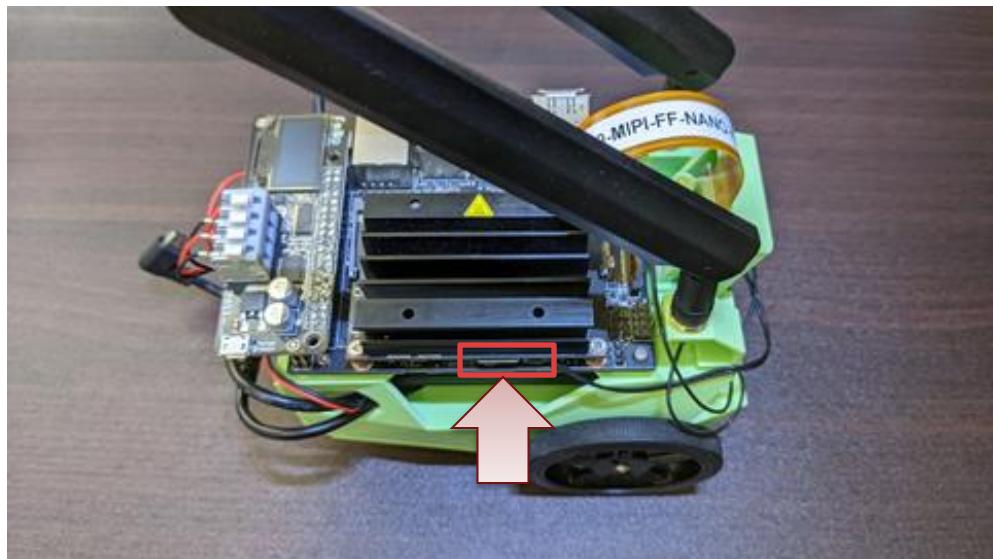
# Setting up JetBot

- Insert a micro SD card to your machine (The volume of the micro SD card should be 32GB or more )
- [Write the downloaded image to the card through Etcher](#)



# Setting up JetBot

- Insert the micro SD card to your Jetson Nano on JetBot
- Turn on the JetBot



# Setting up JetBot

- Setup ROS package for JetBot
  - Complete "[Build ros\\_deep\\_learning](#)" section from the beginning in the following repository.
  - [https://github.com/dusty-nv/jetbot\\_ros](https://github.com/dusty-nv/jetbot_ros)

# Setting up JetBot

- In the "Build jetbot\_ros" section, you'll need to follow the below instructions due to patching

```
▪ $ cd ~/workspace/catkin_ws/src  
▪ $ git clone https://github.com/dusty-nv/jetbot_ros  
▪ $ cd jetbot_ros/  
▪ $ git checkout c7d0e611b037947a0df855293c9848a7c8bc6e90  
▪ $ cd ~  
▪ $ git clone https://github.com/mathworks-robotics/ai-robotics-workshop.git  
▪ $ cd ~/workspace/catkin_ws/src/jetbot_ros  
▪ $ patch -p1 < ~/ai-robotics-workshop/setup/jetbot_ros_for_c7d0e61.patch  
patching file CMakeLists.txt  
patching file launch/jetbot_ros.launch  
patching file package.xml  
patching file scripts/jetbot_motors.py  
patching file src/jetbot_camera.cpp  
▪ $ cd ~/workspace/catkin_ws  
▪ $ catkin_make
```

Make sure no errors

Make sure no build error

Reasons for the patch:

Motor velocity control is not implemented in the original code

Need to downscale the image to reduce the communication delay between MATLAB and JetBot

# Setting up JetBot

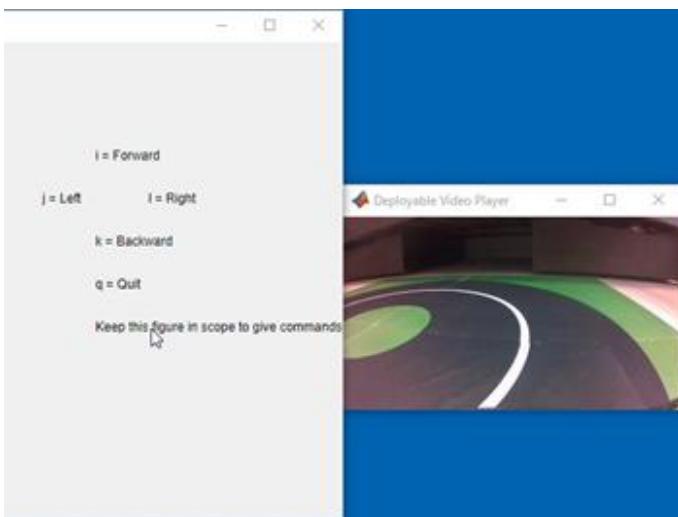
- If you come across 'Adafruit\_MotorHAT is not found' error, you'll need to specify 'pip2' instead of 'pip'
  - `$ pip2 install Adafruit-MotorHAT`
  - `$ pip2 install Adafruit-SSD1306`
- Add environment variables to the end line of `~/.bashrc`
  - `$ gedit ~/.bashrc`
  - `export ROS_IP=$(hostname -I | awk '{print $1;}' | tr -d [:blank:])`
  - `export ROS_MASTER_URI=http://$ROS_IP:11311`
  - You can fix the IP address as needed

# Setting up JetBot

- Setup Jetson CSI camera ROS nodelet for standalone deployment
  - \$ cd /home/jetbot/workspace/catkin\_ws/src/
  - \$ git clone [https://github.com/sfalexrog/jetson\\_camera.git](https://github.com/sfalexrog/jetson_camera.git)
  - \$ cd /home/jetbot/workspace/catkin\_ws
  - \$ catkin\_make
- If you see 'OpenCV3 is not found' error, you'll need to remove "3" for OpenCV4
- (because JetPack4.3's default is OpenCV4)
  - \$ gedit ~/workspace/catkin\_ws/src/jetson\_camera/CMakeLists.txt
  - find\_package(OpenCV **3** REQUIRED)
- **/usr/include/opencv is not found error**
- Create a symbolic link to opencv4 as below
  - \$ cd /usr/include/
  - \$ sudo ln -s opencv4 opencv

# Setting up JetBot

- Connection test from MATLAB
  - Launch MATLAB R2021a
  - `>> cd c:\ai-robotics-workshop`
  - `>> matlab_ai_robotics_workshop.prj`
  - `>> edit teleopTestJetBot_jp.mlx`
- Success if JetBot is moving and the camera image is shown up



# Setting up JetBot

If the rotation direction of wheels are opposite or wheels are side to side:

1. Open `jetBotKeyboardControl.m`

Uncomment in the line 144

```
>> edit jetBotKeyboardControl.m
    % Change rotation direction forward/reverse
    % wlCmd = -wlCmd;
    % wrCmd = -wrCmd;

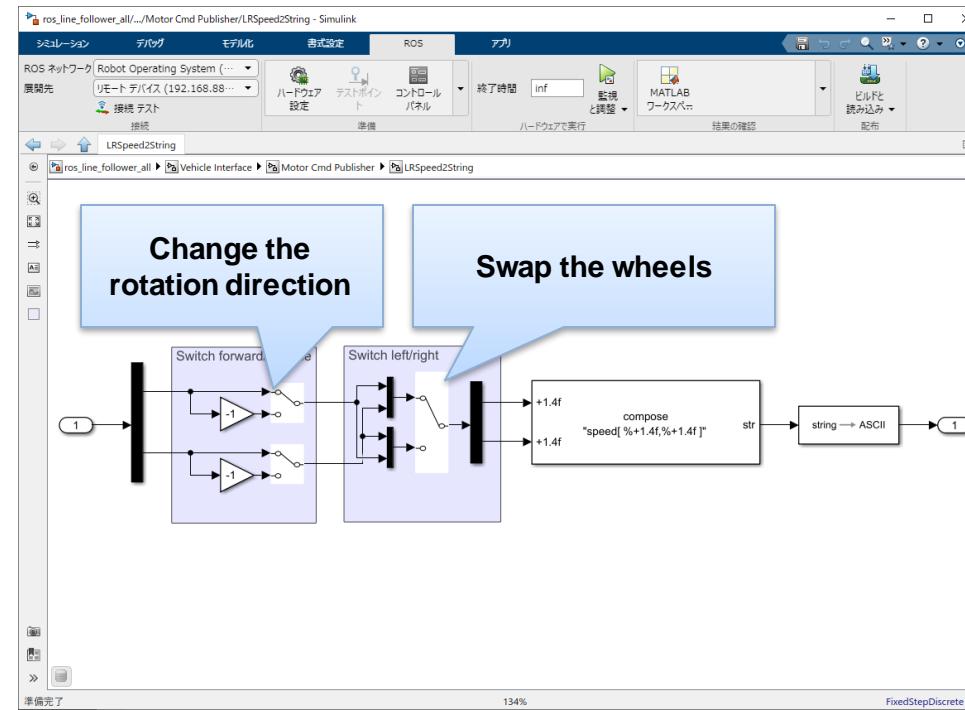
    % Swap wheels side to side
    % tmp = wlCmd;
    % wlCmd = wrCmd;
    % wrCmd = tmp;
```

2. Open `teleopTestJetBot_jp mlx` and run `jetBotKeyboardControl` to test

# Setting up JetBot

If the rotation direction of wheels are opposite or wheels are side to side:

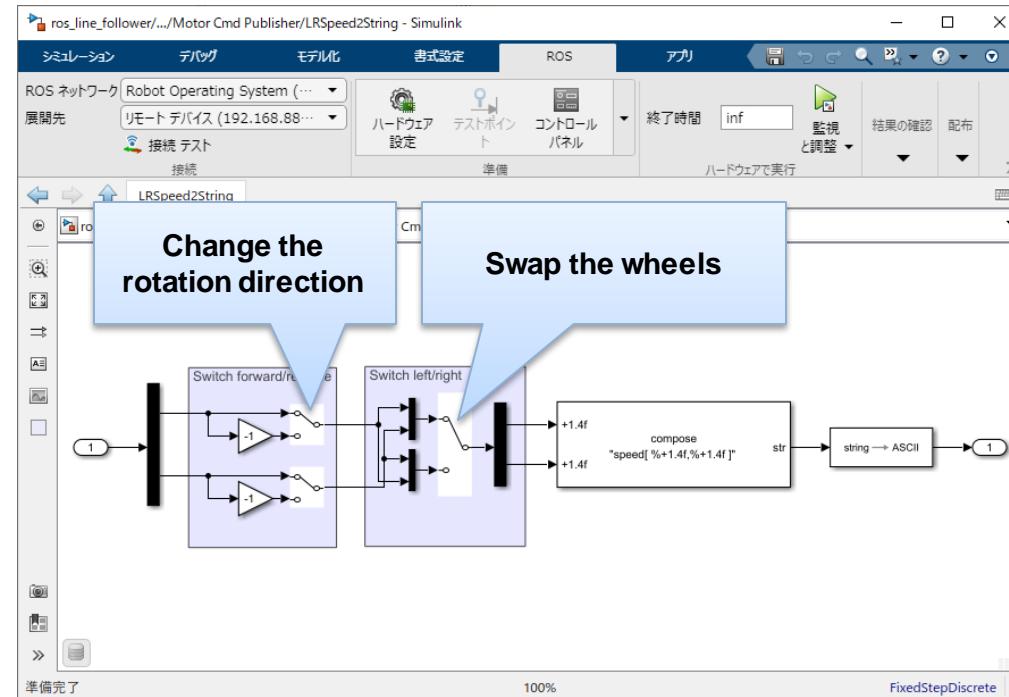
1. Open `ros_line_follower_all.slx`  
`>> open_system('ros_line_follower_all');`
2. Open "Vehicle Interface"->"Motor Cmd Publisher"->"LRSpeed2String"
3. Change the rotation direction and/or swap the wheels as needed
4. Run `testLineFollowerWithROS_jp mlx` and make sure the line following works



# Setting up JetBot

If the rotation direction of wheels are opposite or wheels are side to side:

1. Open `ros_line_follower.slx`  
`>> open_system('ros_line_follower');`
2. Open "Vehicle Interface" → "Motor Cmd Publisher" → "LRSpeed2String"
3. Change the rotation direction and/or swap the wheels as needed
4. Run `deployLineFollower_jp.mlx` and make sure the line following works



# Setting up JetBot

If the camera image is upside down:

1. Open `jetBotKeyboardControl.m` and uncomment the line 79

```
>> edit jetBotKeyboardControl.m
```

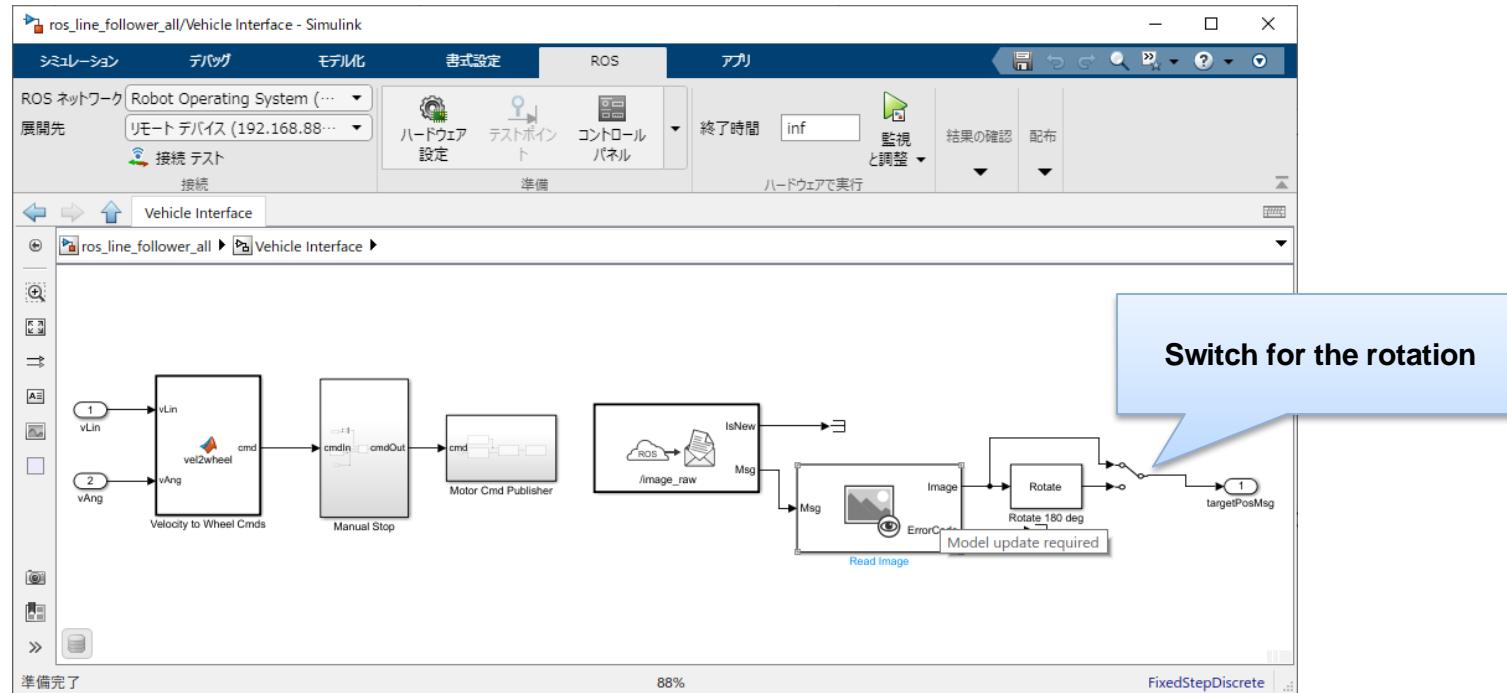
```
75  
76      % Subscribe image  
77 -      imMsg = receive(camSub);  
78 -      I = rosReadImage(imMsg);  
79      %I = imrotate(I,180);  
80
```

2. Open `teleopTestJetBot_jp mlx` and run `jetBotKeyboardControl` to test

# Setting up JetBot

If the camera image is upside down:

1. Open `ros_line_follower_all.slx`  
`>> open_system('ros_line_follower_all');`
2. Open "Vehicle Interface"
3. Change the switch to enable 180 deg rotation
4. Run `testLineFollowerWithROS_jp.mlx` to check if the line following works



# Setting up JetBot

If the camera image is upside down:

1. Open `line_detection.m` and uncomment the line 15

`>> edit line_detection.m`

```
12 % Convert image data format OpenCV BGR
13 - img = ocv2mat(I);
14
15 %img = imrotate(img,180)
16
17 - sz = size(img);
18 - sizeWH = sz([2 1]);
19
```

# Setting up standalone deployment

- Setup "MATLAB Coder Support Package for NVIDIA Jetson and NVIDIA DRIVE Platforms"
  - <https://www.mathworks.com/help/releases/R2021a/supportpkg/nvidia/ug/install-and-setup-prerequisites.html>
- Install SDL and V4L-utils
  - `$ sudo apt-get install libsdl1.2-dev v4l-utils sox libsox-fmt-all libsox-dev`

# Setting up standalone deployment

- Configure environment variables on Jetson
  - To enable 'nvcc' command through SSH, need to add relevant environment variables to .bashrc

```
$ gedit ~/.bashrc
```

```
# If not running interactively, don't do anything
case $- in
  *i*) ;;
  *)
    export PATH=$PATH:/usr/local/cuda/bin
    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64
    export LIBRARY_PATH=/usr/local/cuda/lib64
    return;;
esac
```

- Note: environment variables should be passed even for non-interactive shell

<https://www.mathworks.com/help/releases/R2021a/supportpkg/nvidia/ug/install-and-setup-prerequisites.html>