# 利用 MATLAB/SIMULINK 開發強化式學習應用

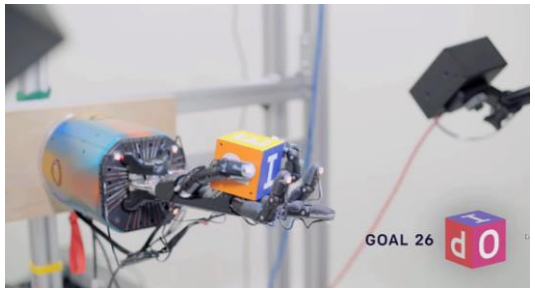
**Sarah Hung**
**Application Engineer**
**2019/09/27**

# What is Reinforcement Learning?

- What is Reinforcement Learning?
  - Type of machine learning that trains an **‘agent’** through repeated interactions with an environment
- How does it work?
  - Through a trial & error process that maximizes success
- Why should you care about Reinforcement Learning?
  - It enables the use of deep learning for controls and decision-making applications

Autonomous driving

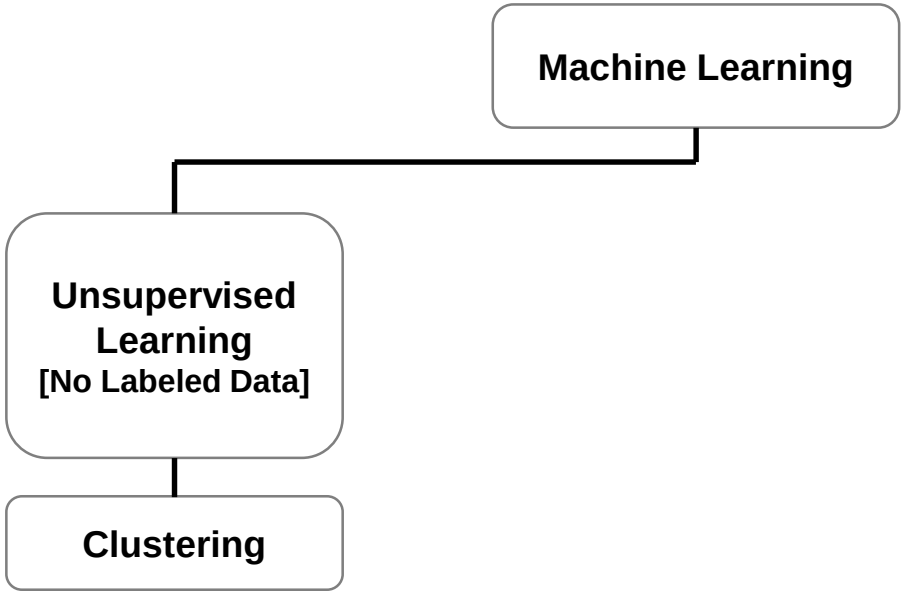
Controls

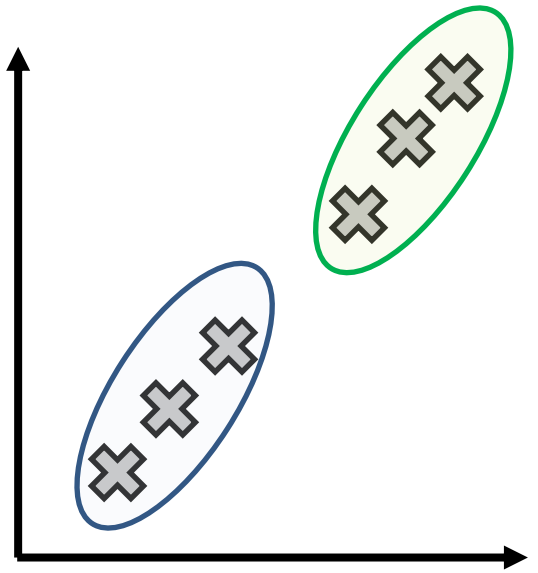



Robotics

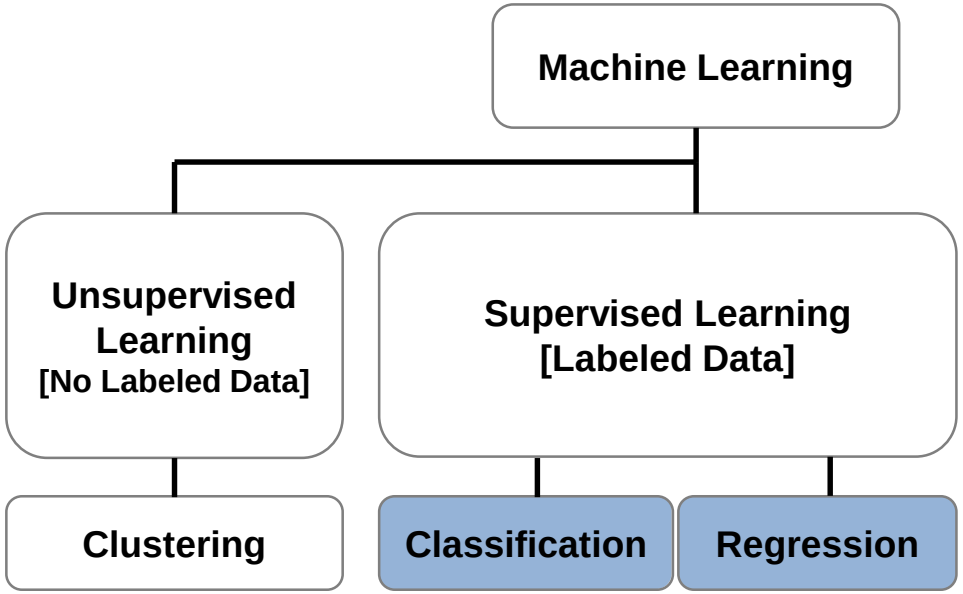

Game Play

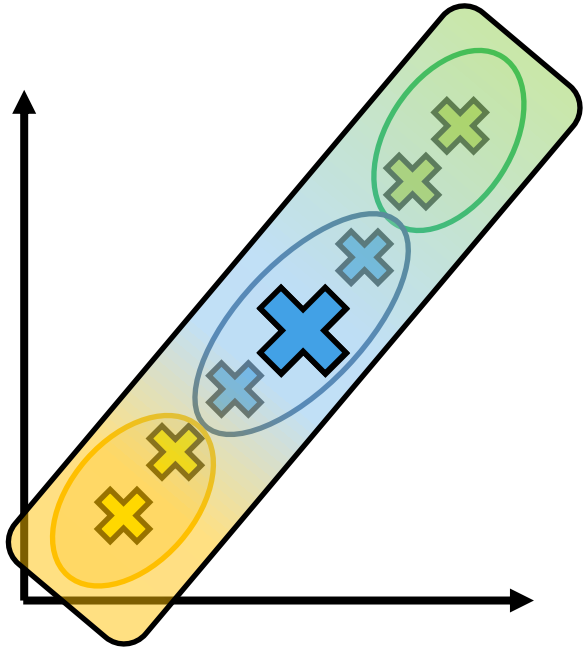
# Machine Learning, Deep Learning, and Reinforcement Learning



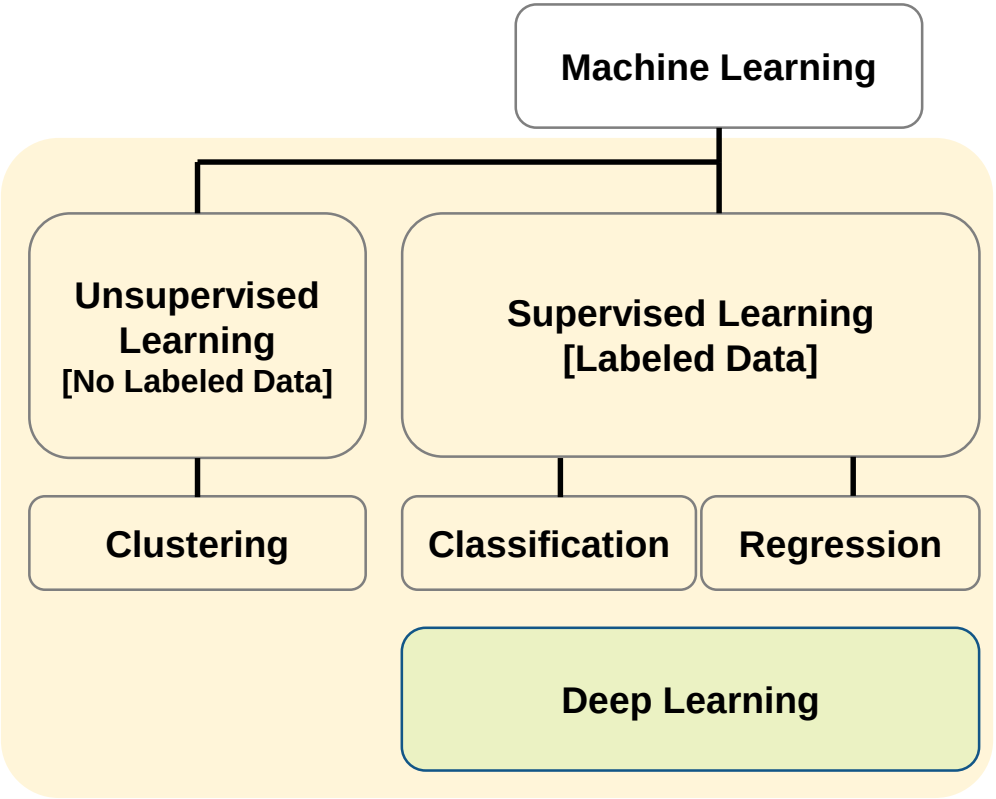



# Machine Learning, Deep Learning, and Reinforcement Learning



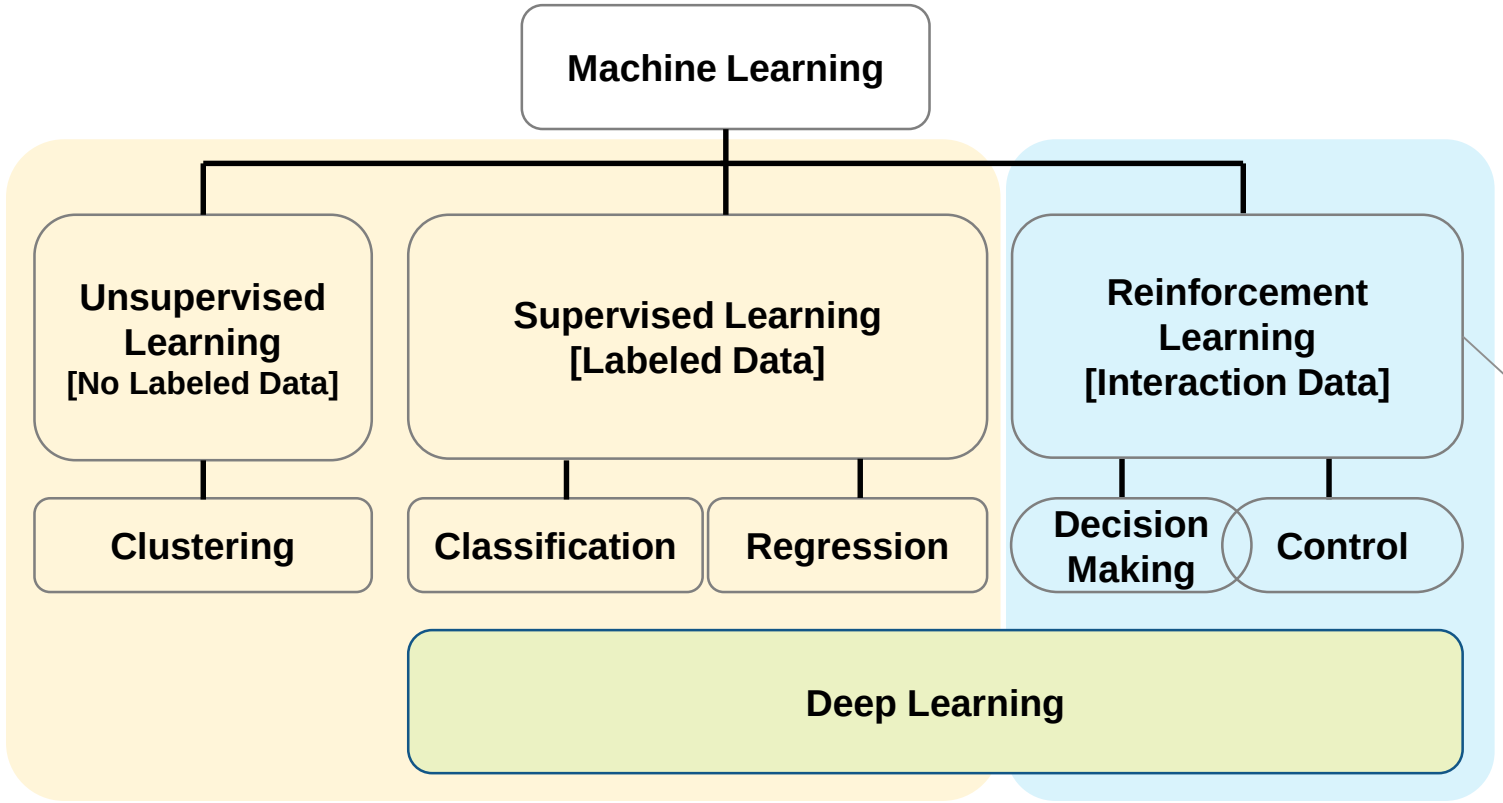
# Machine Learning, Deep Learning, and Reinforcement Learning




Supervised learning typically involves feature extraction

Deep learning typically does not involve feature extraction

# Machine Learning, Deep Learning, and Reinforcement Learning

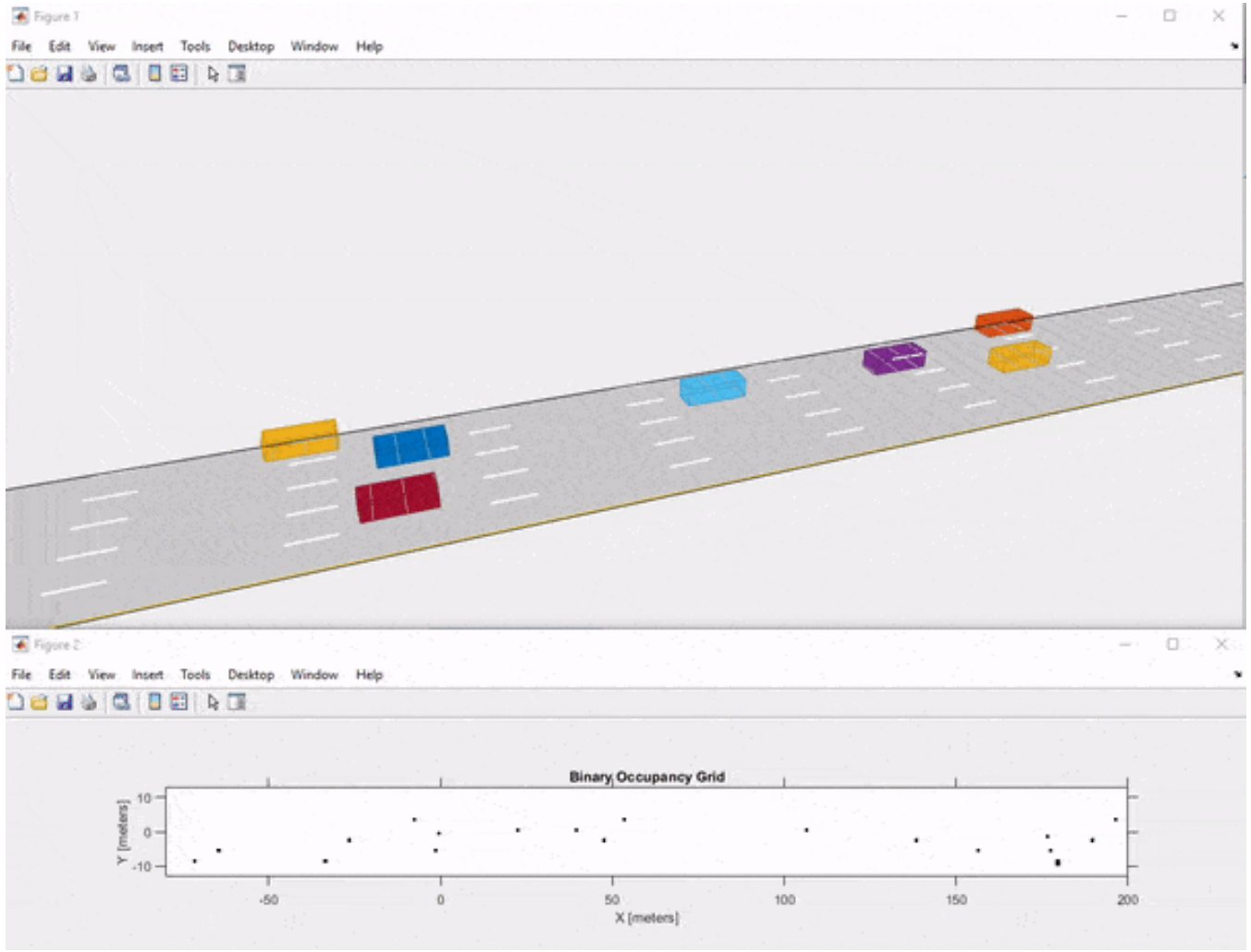



## Reinforcement learning:

- Learning through trial & error [*interaction*]
- Complex problems typically need deep learning [*Deep Reinforcement Learning*]
- It's about learning a **behavior** or accomplishing a **task**

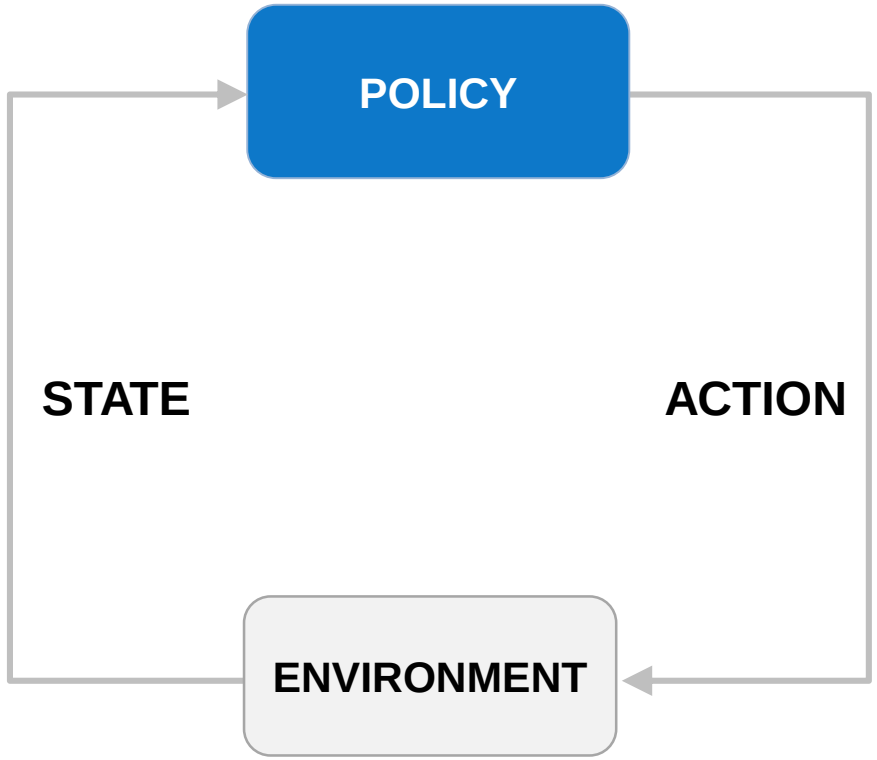
# A Practical Example of Reinforcement Learning

## Training a Self-Driving Car



# A Practical Example of Reinforcement Learning

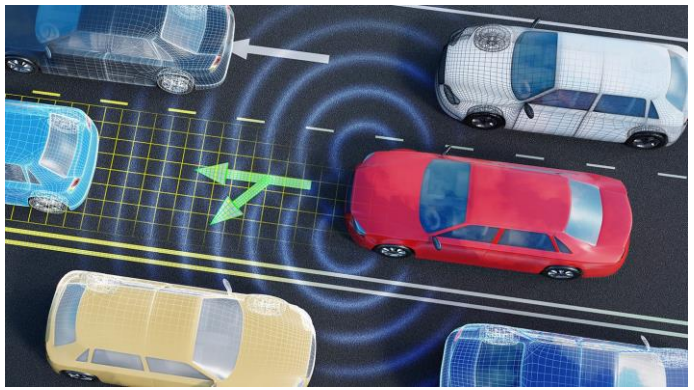
## A Trained Self-Driving Car Only Needs A Policy To Operate

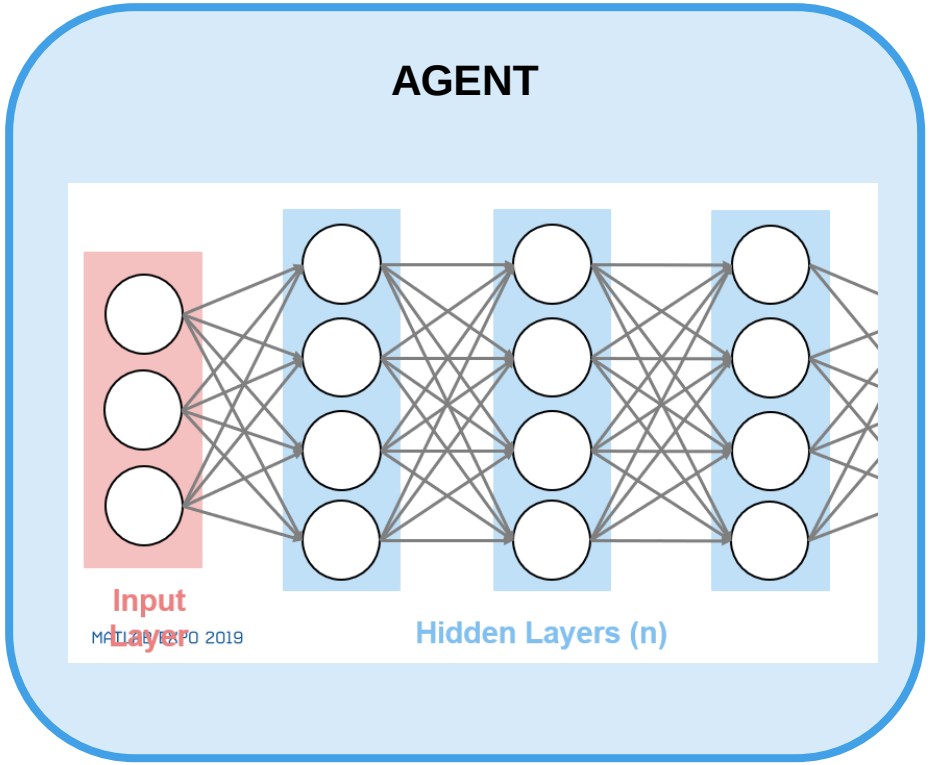



- Vehicle's computer uses the final state-to-action mapping… (**policy**)
- to generate steering, braking, throttle commands,… (**action**)
- based on sensor readings from LIDAR, cameras,… (**state**)
- that represent road conditions, vehicle position,… (**environment**)

**By definition, this trained policy is optimizing driver comfort & fuel efficiency**

# Deep Networks are commonly found in the agent, because they can model complex problems.






- **Turn left**
- **Turn right**
- **Brake**
- **Accelerate**

# Reinforcement Learning Workflow

## Prepare Data

Data access and preprocessing

Ground truth labeling

**Simulink – generate data for dynamic systems (planes, cars, robots, etc.)**

## Train Model

Reinforcement learning

Training agent to perform task

Developing reward system to optimize performance

## Deployment

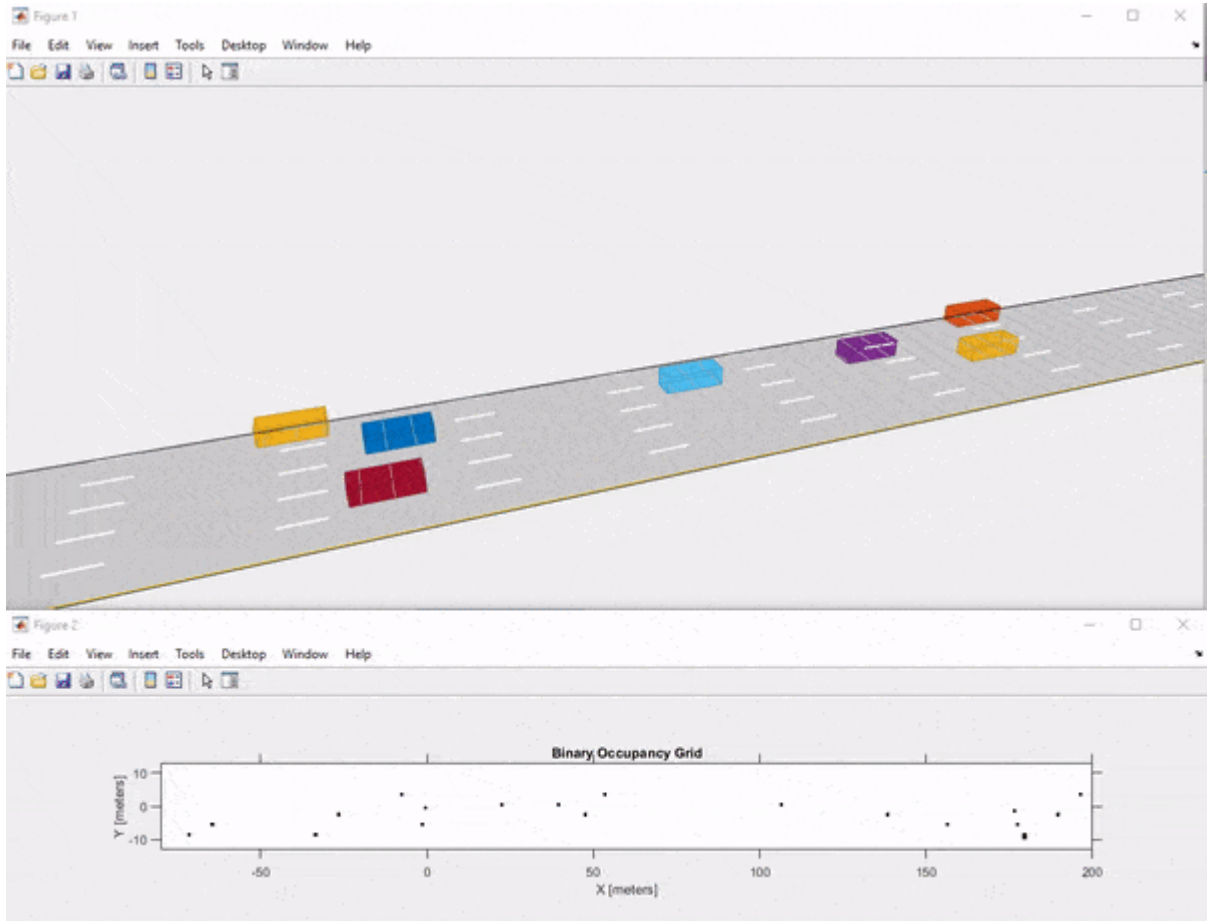
Multiplatform code generation (CPU, GPU)

Edge deployment

Enterprise Deployment

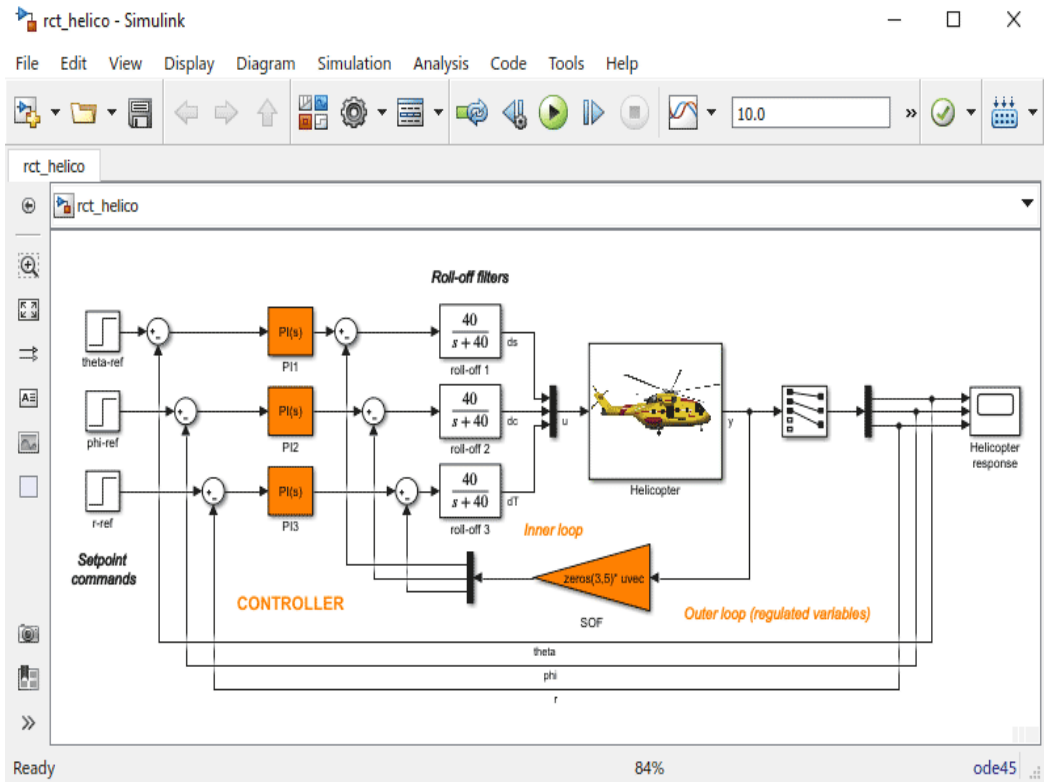
# Why MATLAB and Simulink for Reinforcement Learning?

*Virtual models allow you to simulate conditions hard to emulate in the real world.*

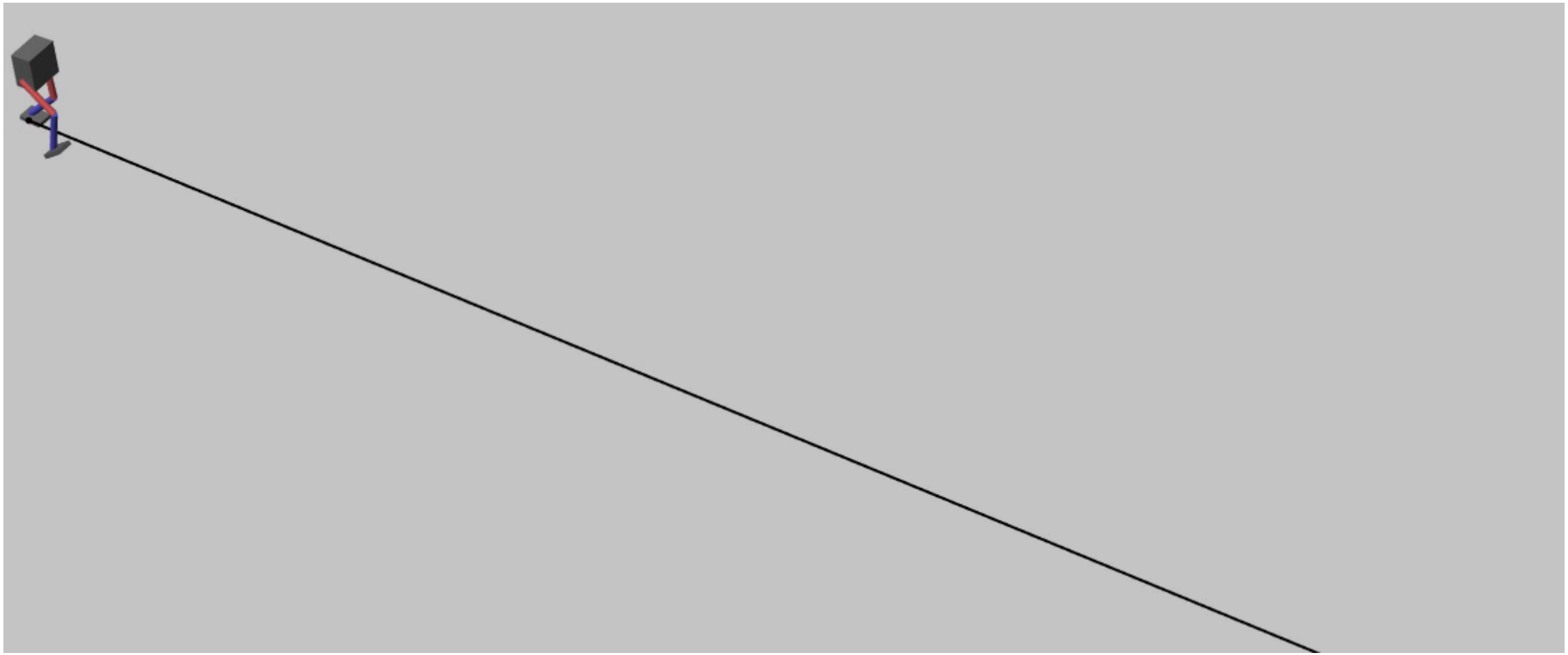


# Using MATLAB and Simulink for Reinforcement Learning

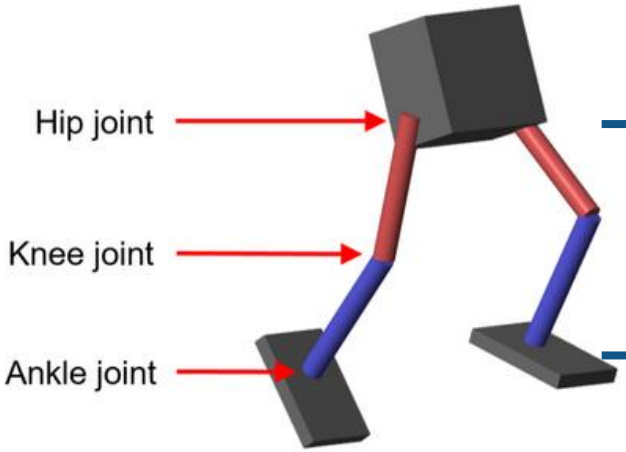
- Reinforcement learning is a dynamic process
- Decision making problems
  - Financial trading, calibration, etc.
- Controls-based problems
  - Lane-keep assist, adaptive cruise control, robotics, etc.

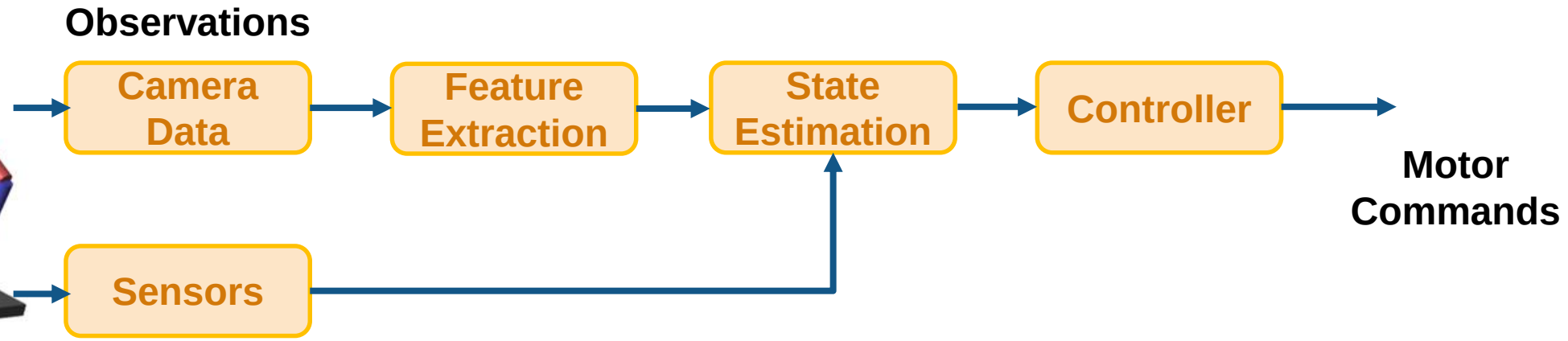


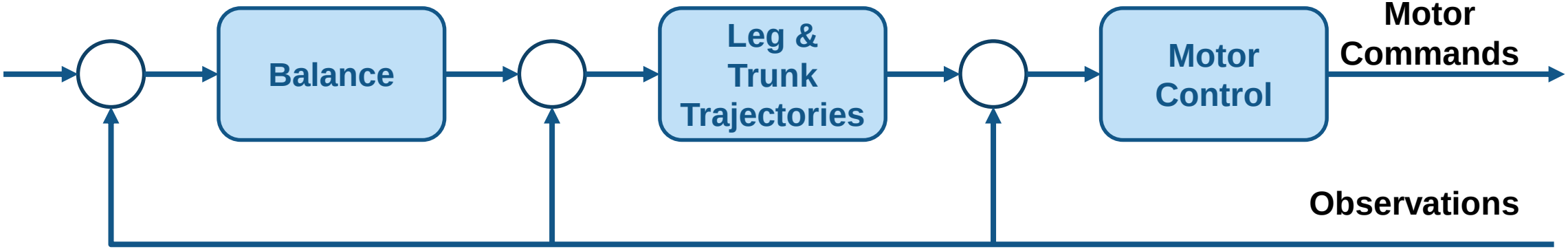
# Teach a robot to follow a straight line



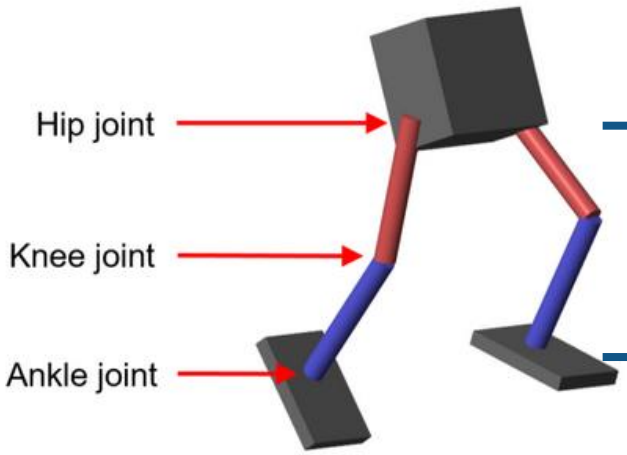
# Let's try to solve this problem the traditional way



# What is the alternative approach?




Observations

Camera Data

Sensors

Feature Extraction

State Estimation

Controller

Motor Commands

Camera Data

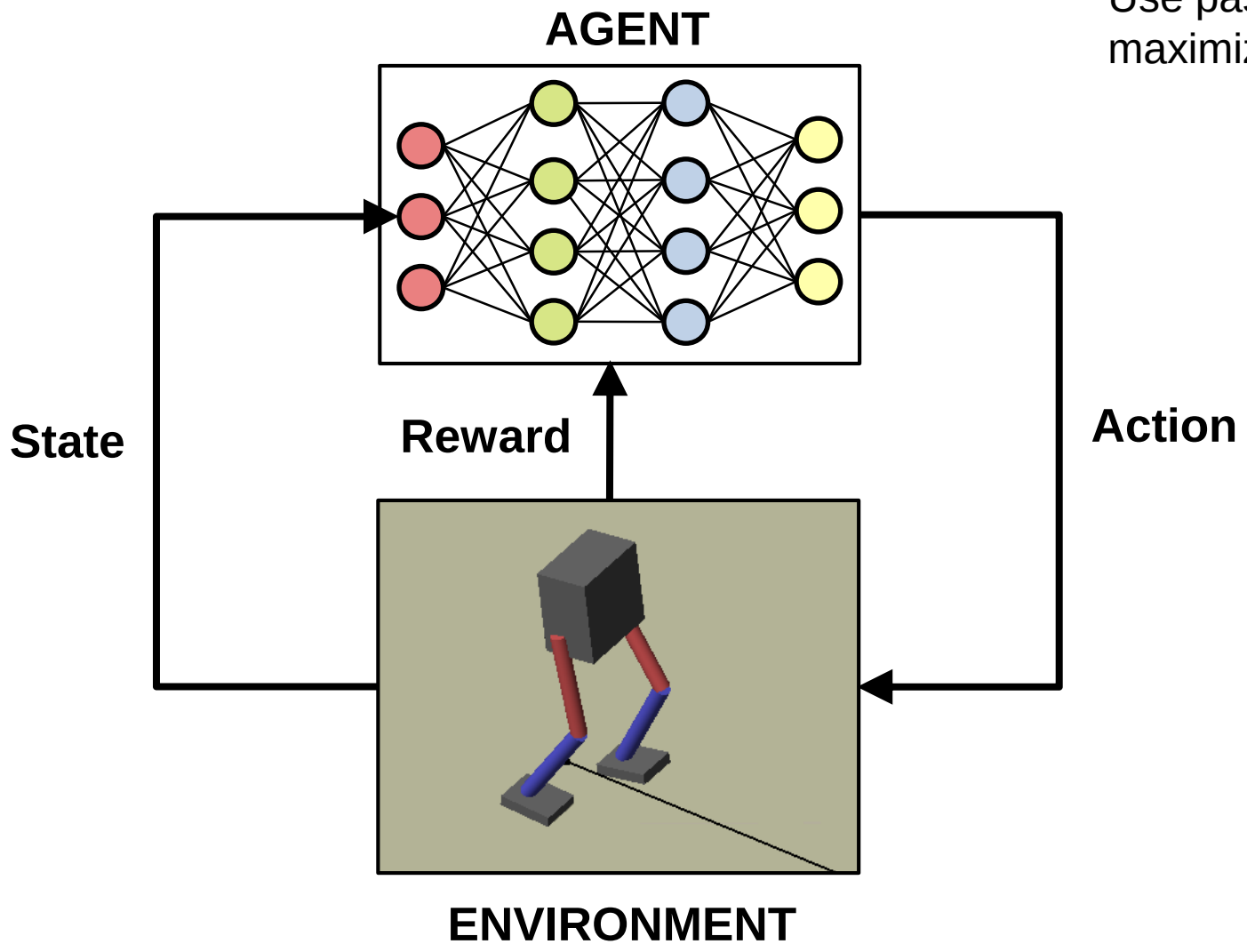
Sensors

Black Box Controller

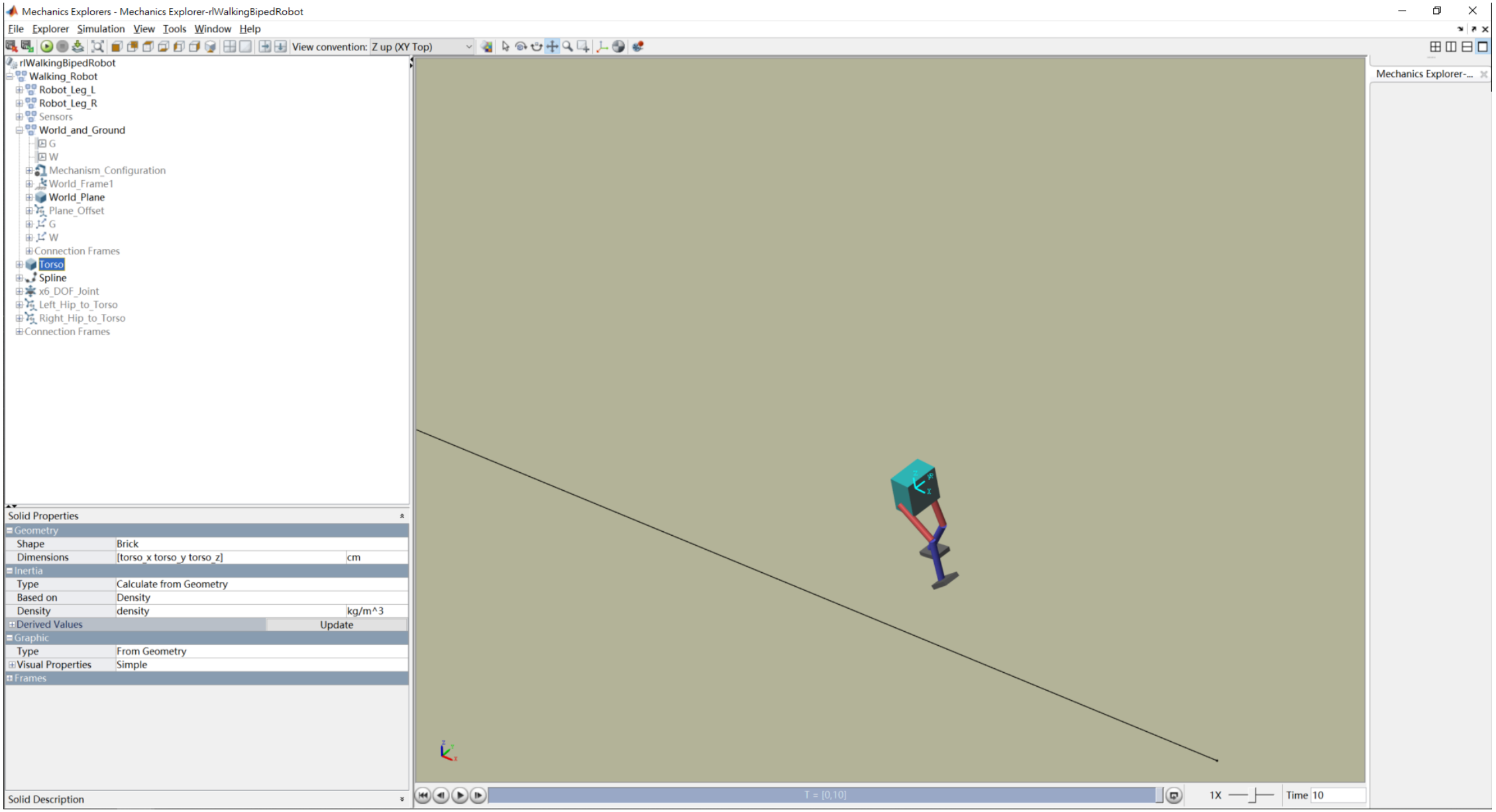
Motor Commands

# Walking Robot Example

**Reinforcement Learning:**
Use past experiences to maximize expected reward

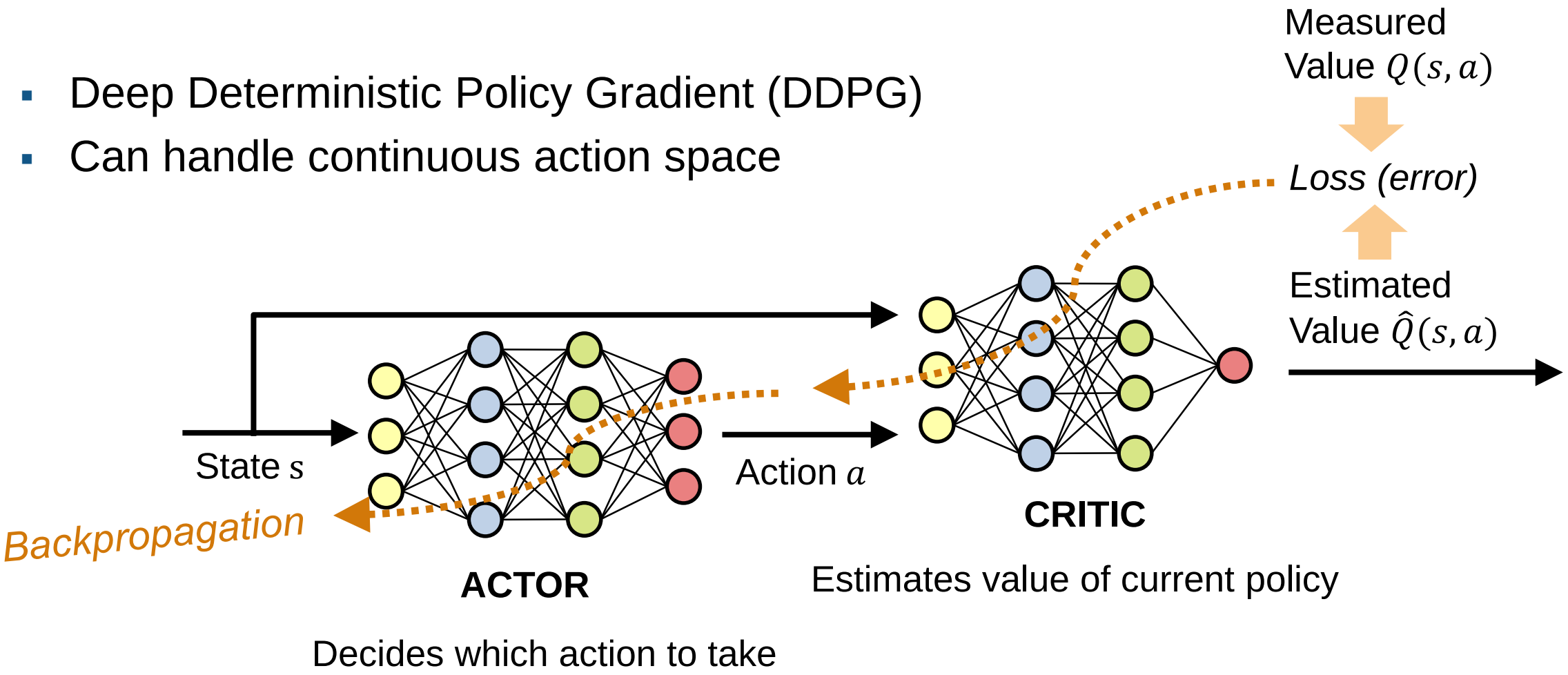


# Environment: model a biped robot in Simscape™ Multibody™

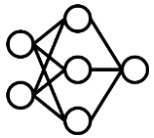


# How Is the Agent Trained?

- Deep Deterministic Policy Gradient (DDPG)
- Can handle continuous action space



# Deep Network Designer



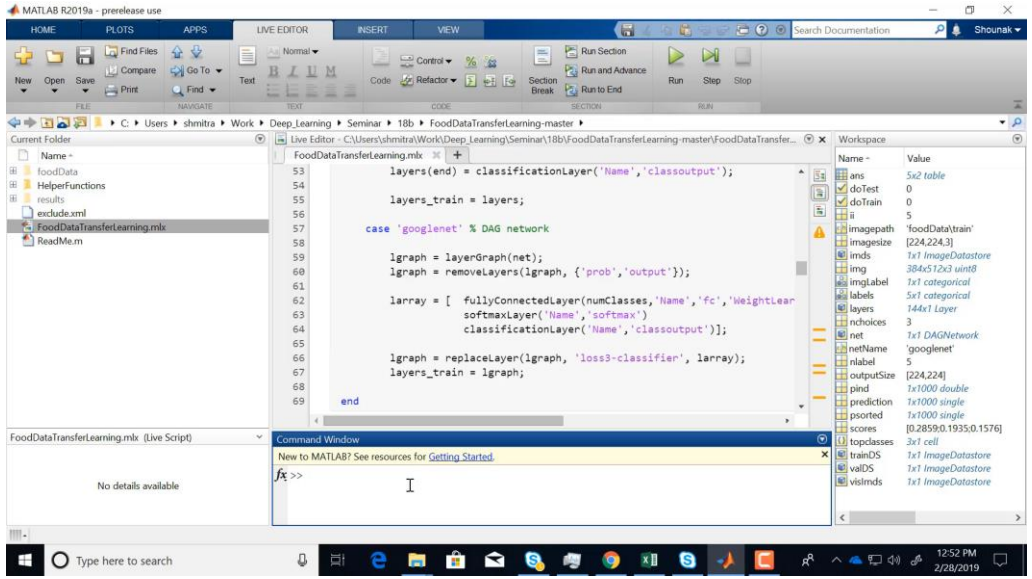
**Policy**

Critic

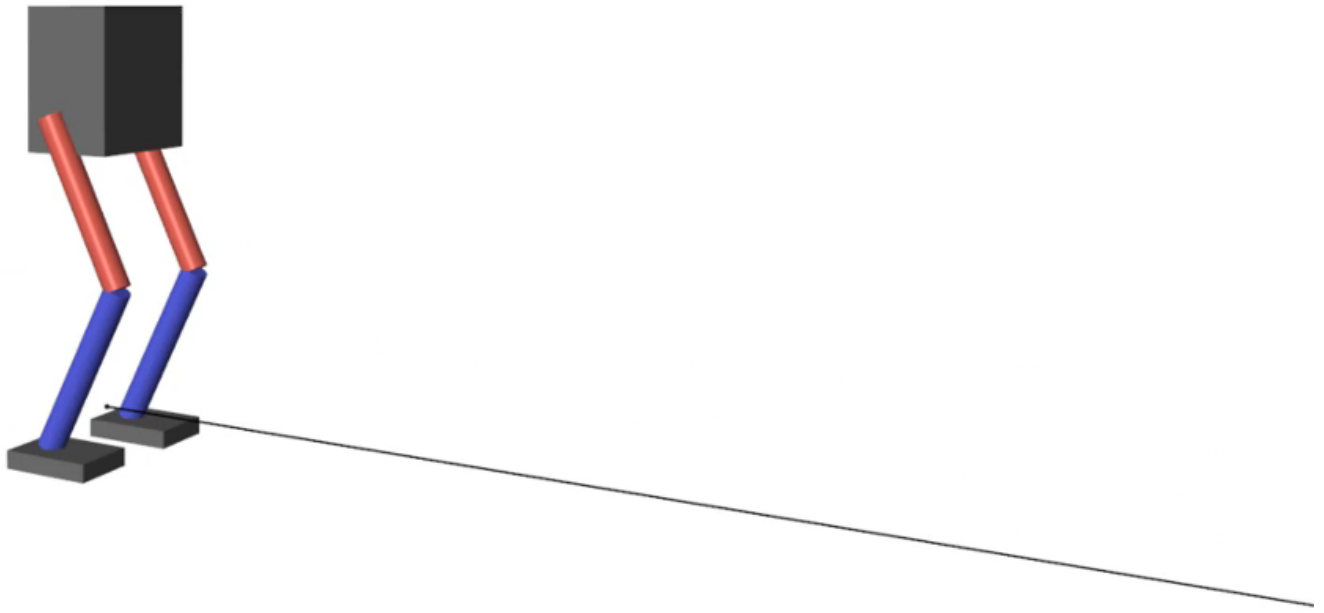
```
criticLayerSizes = [400 300];
statePath = [
    imageInputLayer([numObs 1 1],'Normalization','none','Name', 'observation')
    fullyConnectedLayer(criticLayerSizes(1), 'Name', 'CriticStateFC1', ...
            'Weights',2/sqrt(numObs)*(rand(criticLayerSizes(1),numObs)-0.5), ...
            'Bias',2/sqrt(numObs)*(rand(criticLayerSizes(1),1)-0.5))
    reluLayer('Name','CriticStateRelu1')
    fullyConnectedLayer(criticLayerSizes(2), 'Name', 'CriticStateFC2', ...
            'Weights',2/sqrt(criticLayerSizes(1))*(rand(criticLayerSizes(2),criticLa
            'Bias',2/sqrt(criticLayerSizes(1))*(rand(criticLayerSizes(2),1)-0.5))
    ];
actionPath = [
    imageInputLayer([numAct 1 1],'Normalization','none', 'Name', 'action')
    fullyConnectedLayer(criticLayerSizes(2), 'Name', 'CriticActionFC1', ...
            'Weights',2/sqrt(numAct)*(rand(criticLayerSizes(2),numAct)-0.5), ...
            'Bias',2/sqrt(numAct)*(rand(criticLayerSizes(2),1)-0.5))
    ];
commonPath = [
    additionLayer(2,'Name','add')
    reluLayer('Name','CriticCommonRelu1')
    fullyConnectedLayer(1, 'Name', 'CriticOutput',...
            'Weights',2*5e-4*(rand(1,criticLayerSizes(2))-0.5), ...
            'Bias',2*5e-4*(rand(1,1)-0.5))
    ];
```

Actor

```
actorLayerSizes = [400 400];
actorNetwork = [
    imageInputLayer([numObs 1 1],'Normalization','none','Name','observation')
    fullyConnectedLayer(actorLayerSizes(1), 'Name', 'ActorFC1', ...
            'Weights',2/sqrt(numObs)*(rand(actorLayerSizes(1),numObs)-0.5), ...
            'Bias',2/sqrt(numObs)*(rand(actorLayerSizes(1),1)-0.5))
    reluLayer('Name', 'ActorRelu1')
    fullyConnectedLayer(actorLayerSizes(2), 'Name', 'ActorFC2', ...
            'Weights',2/sqrt(actorLayerSizes(1))*(rand(actorLayerSizes(2),actorLaye
            'Bias',2/sqrt(actorLayerSizes(1))*(rand(actorLayerSizes(2),1)-0.5))
    reluLayer('Name', 'ActorRelu2')
    fullyConnectedLayer(numAct, 'Name', 'ActorFC3', ...
                        'Weights',2*5e-3*(rand(numAct,actorLayerSizes(2))-0.5), ...
                        'Bias',2*5e-3*(rand(numAct,1)-0.5))
    tanhLayer('Name','ActorTanh1')
    ];
```

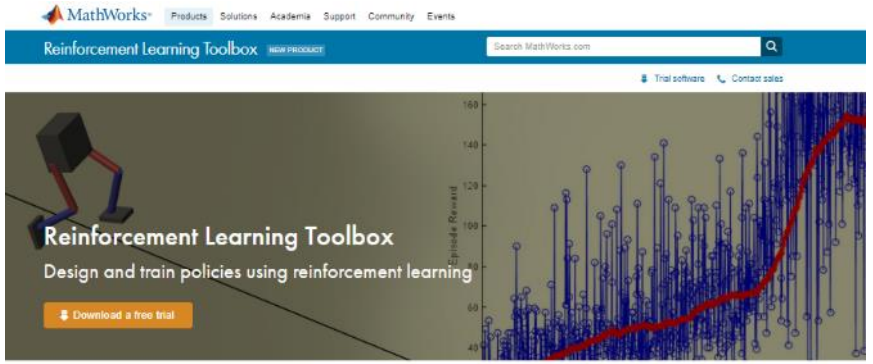


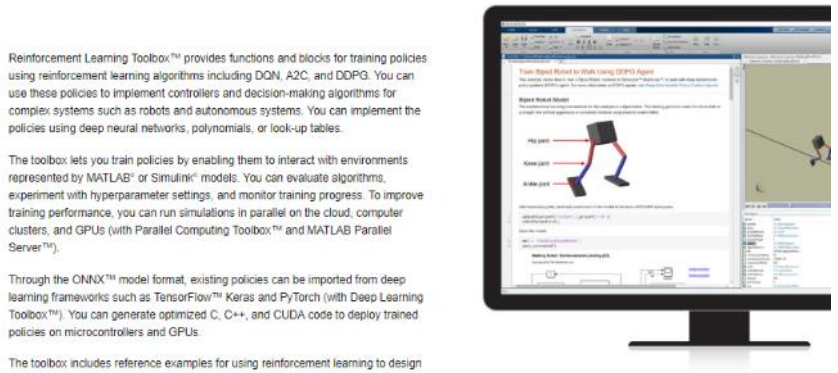
# Steps in the Reinforcement Learning Workflow

# Reinforcement Learning Toolbox
# New Product in R2019a

- Built-in and custom reinforcement learning algorithms
- Environment modeling in MATLAB and Simulink
  - Existing scripts and models can be reused!
- Deep Learning Toolbox support for representing policies
- Training acceleration with PCT and MATLAB Parallel Server
- Deployment of trained policies with GPU Coder and MATLAB Coder
- Reference examples for getting started (control systems, automotive, robotics)



Reinforcement Learning Toolbox™ provides functions and blocks for training policies using reinforcement learning algorithms including DQN, A2C, and DDPG. You can use these policies to implement controllers and decision-making algorithms for complex systems such as robots and autonomous systems. You can implement the policies using deep neural networks, polynomials, or look-up tables.

The toolbox lets you train policies by enabling them to interact with environments represented by MATLAB® or Simulink® models. You can evaluate algorithms, experiment with hyperparameter settings, and monitor training progress. To improve training performance, you can run simulations in parallel on the cloud, computer clusters, and GPUs (with Parallel Computing Toolbox™ and MATLAB Parallel Server™).

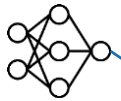
Through the ONNX™ model format, existing policies can be imported from deep learning frameworks such as TensorFlow™ Keras and PyTorch (with Deep Learning Toolbox™). You can generate optimized C, C++, and CUDA code to deploy trained policies on microcontrollers and GPUs.

The toolbox includes reference examples for using reinforcement learning to design controllers for robotics and automated driving applications.

# Reinforcement Learning Toolbox Workflow

Environment

Reward



Policy

Agent


Training

Deployment

## Reinforcement Learning Toolbox

R2019a

Design and train policies using reinforcement learning

Reinforcement Learning Toolbox™ provides functions and blocks for training policies using reinforcement learning algorithms including DQN, A2C, and DDPG. You can use these policies to implement controllers and decision-making algorithms for complex systems such as robots and autonomous systems. You can implement the policies using deep neural networks, polynomials, or look-up tables.

The toolbox lets you train policies by enabling them to interact with environments represented by MATLAB® or Simulink® models. You can evaluate algorithms, experiment with hyperparameter settings, and monitor training progress. To improve training performance, you can run simulations in parallel on the cloud, computer clusters, and GPUs (with Parallel Computing Toolbox™ and MATLAB Parallel Server™).

Through the ONNX™ model format, existing policies can be imported from deep learning frameworks such as TensorFlow™ Keras and PyTorch (with Deep Learning Toolbox™). You can generate optimized C, C++, and CUDA code to deploy trained policies on microcontrollers and GPUs.

The toolbox includes reference examples for using reinforcement learning to design controllers for robotics and automated driving applications.

Release Notes
PDF Documentation

**Getting Started**
Learn the basics of Reinforcement Learning Toolbox

**MATLAB Environments**
Model reinforcement learning environment dynamics using MATLAB

**Simulink Environments**
Model reinforcement learning environment dynamics using Simulink models

**Policies and Value Functions**
Define policy and value function representations, such as deep neural networks and Q tables

**Agents**
Create and configure reinforcement learning agents using common algorithms, such as SARSA, DQN, DDPG, and A2C

**Training and Validation**
Train and simulate reinforcement learning agents

**Policy Deployment**
Code generation and deployment of trained policies

# Examples

Help

Reinforcement Learning Toolbox — Examples

Documentation | All | Examples | Functions | Blocks | Search Help

CONTENTS | Close

« Documentation Home

« Examples

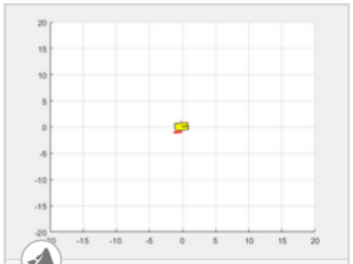
**Category**

| | |
|---|---|
| Predictive Maintenance Toolbox | 18 |
| **Reinforcement Learning Toolbox** | **23** |
| Getting Started with Reinforcement Learning Toolbox | 3 |
| MATLAB Environments | 1 |
| Training and Validation | 22 |
| Robotics System Toolbox | 50 |
| Robust Control Toolbox | 38 |
| Sensor Fusion and Tracking Toolbox | 33 |
| Signal Processing Toolbox | 146 |
| SimEvents | 26 |
| Simscape | 65 |
| Simscape Driveline | 43 |

**Type**

| | |
|---|---|
| All | 23 |
| MATLAB | 13 |
| Simulink | 10 |

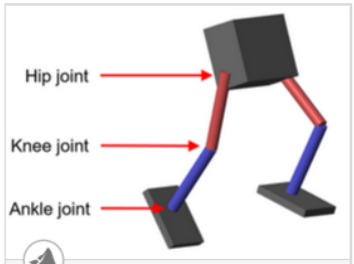
## Training and Validation



### Train DDPG Agent to Control Flying Robot

Train a reinforcement learning agent to control a flying robot model.

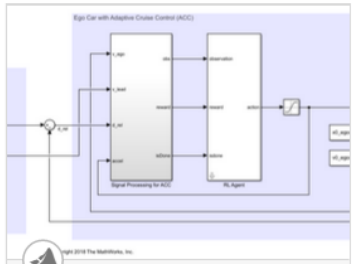
Open Live Script




### Train Biped Robot to Walk Using DDPG Agent

Train a reinforcement learning agent to control a biped walking robot modeled in Simscape Multibody.

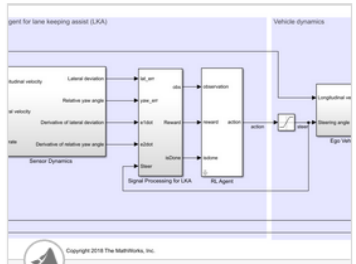
Open Live Script



### Train DDPG Agent for Adaptive Cruise Control

Train a reinforcement learning agent for an adaptive cruise control application.

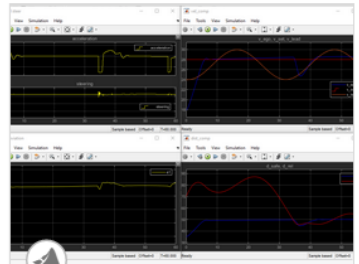
Open Live Script



### Train DQN Agent for Lane Keeping Assist

Train a reinforcement learning agent for a lane keeping assist application.

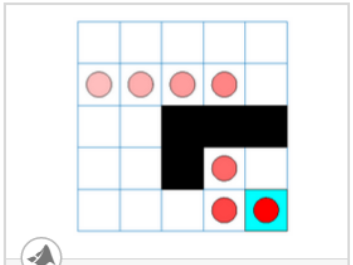
Open Live Script



### Train DDPG Agent for Path Following Control

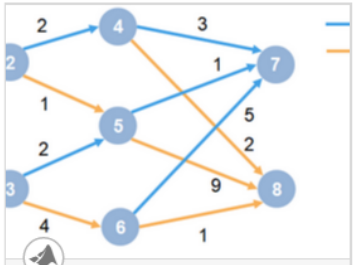
Train a reinforcement learning agent for a lane following application.

Open Live Script



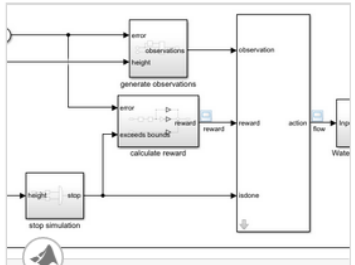
### Train Reinforcement Learning Agent in Basic Grid World

Train Q-learning and SARSA agents



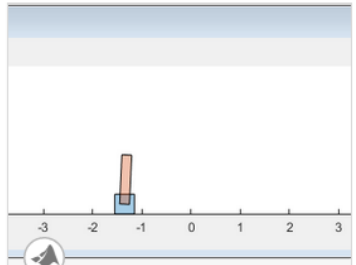
### Train Reinforcement Learning Agent in MDP Environment

Train a reinforcement learning agent



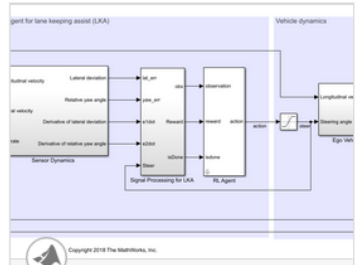
### Create Simulink Environment and Train Agent

Train a controller using



### Train AC Agent to Balance Cart-pole System Using Parallel Computing

Train actor-critic agent using



### Train DQN Agent for Lane Keeping Assist Using Parallel Computing

Train a reinforcement learning agent

# Key Takeaways

- Reinforcement learning can solve complicated (control) problems.
- Reinforcement learning can use *neural networks* to handle continuous or high-dimensional state and action spaces.
- Choosing good states, actions, and reward functions is extremely important!
- Reinforcement learning requires *a lot* of simulation data.
- DDPG is a *high variance* algorithm.
  Run several times and validate results!