Advanced Control for Automated Driving

Hongjun Wang 王鴻鈞
Senior Application Engineer, MathWorks

© 2019 The MathWorks, Inc.

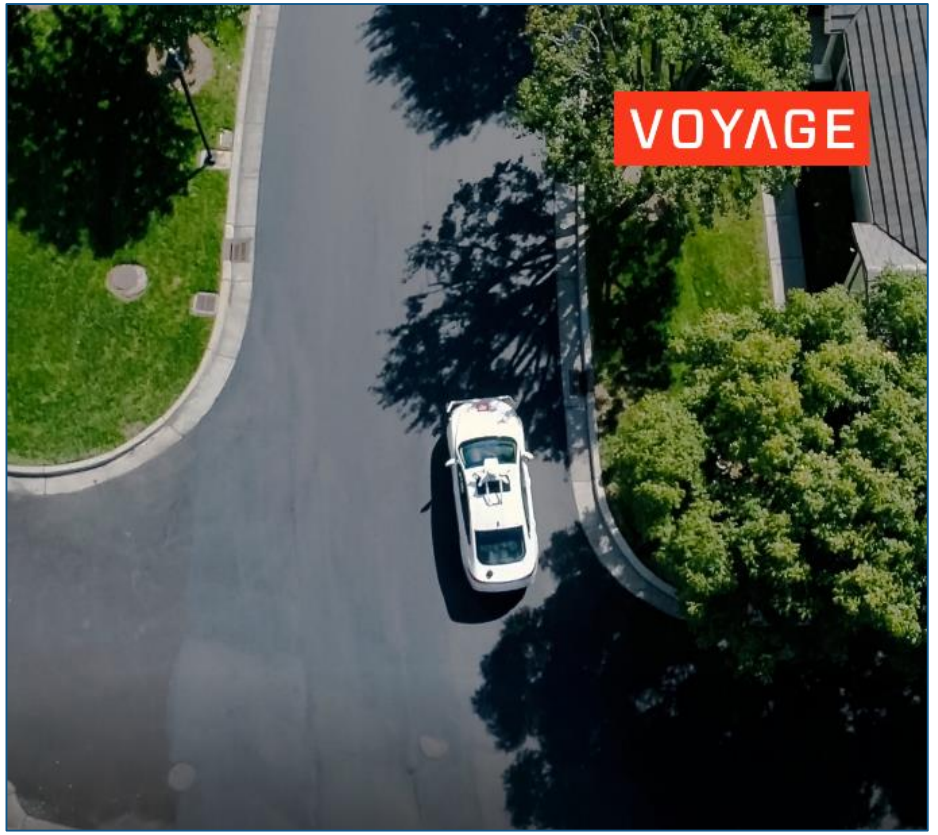
Model Predictive Control

Voyage develops longitudinal controls for self-driving taxis

Designed longitudinal model predictive controller, generated ROS node, and integrated with their open source software framework for perception and planning

"We were searching for a prototyping solution that was fast for development and robust for production. We decided to go with Simulink for controller development and code generation, while using MATLAB to automate development tasks."

- Alan Mond, Voyage

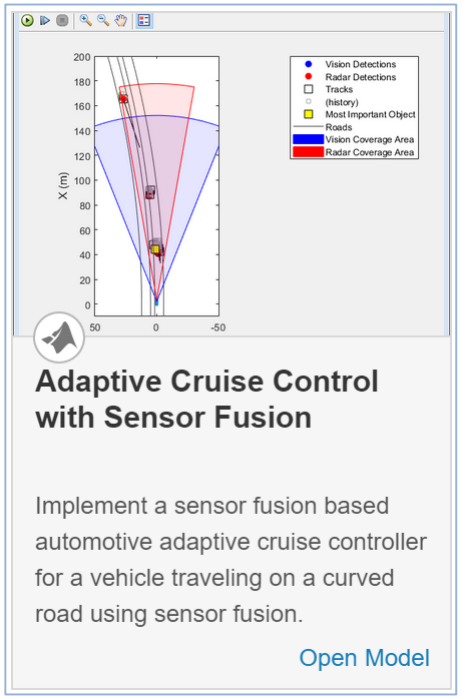


Voyage's self driving car in San Jose, California.

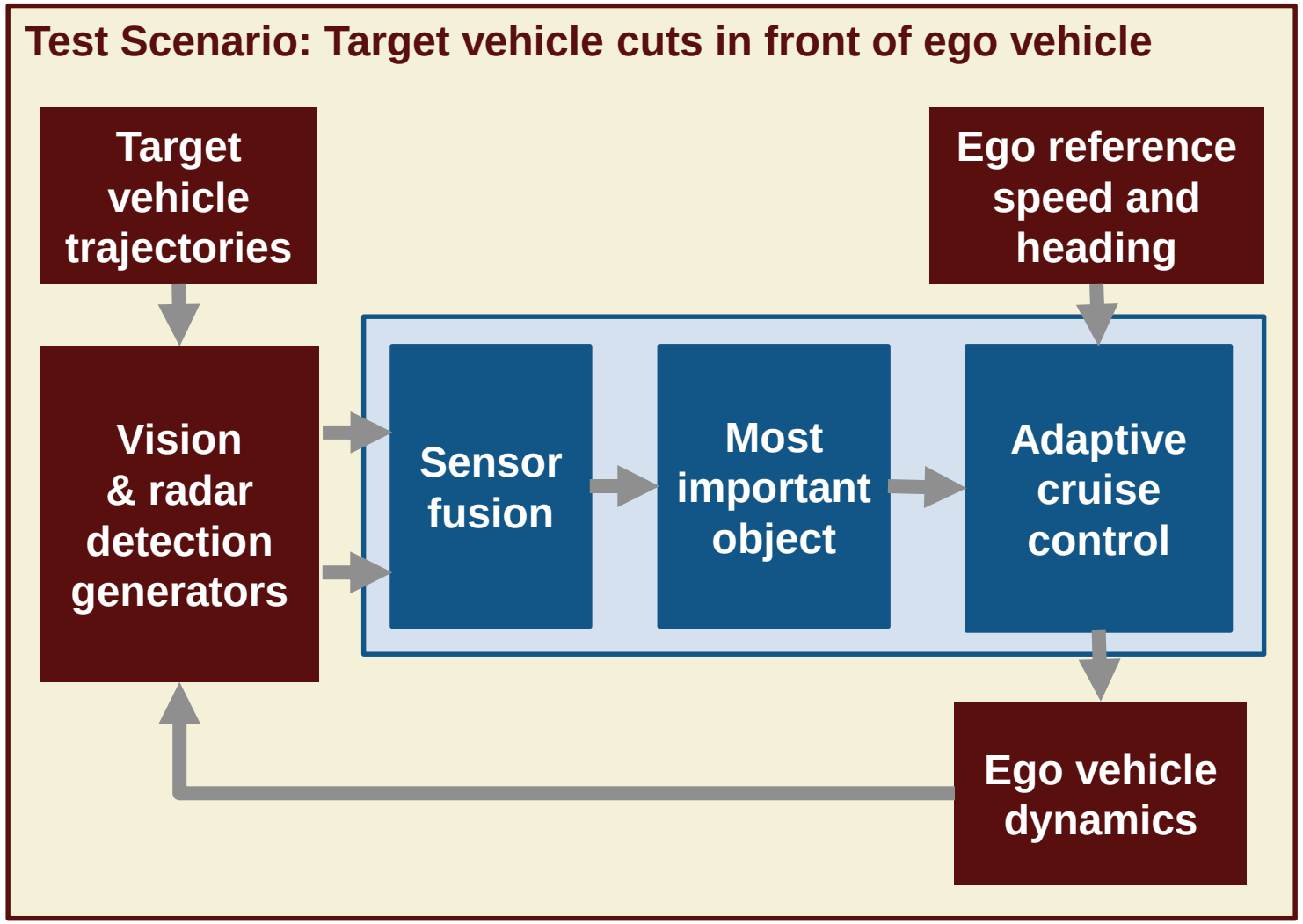
Voyage user story, 2017

Explore modeling patterns in shipping application example:

Adaptive Cruise Control with Sensor Fusion

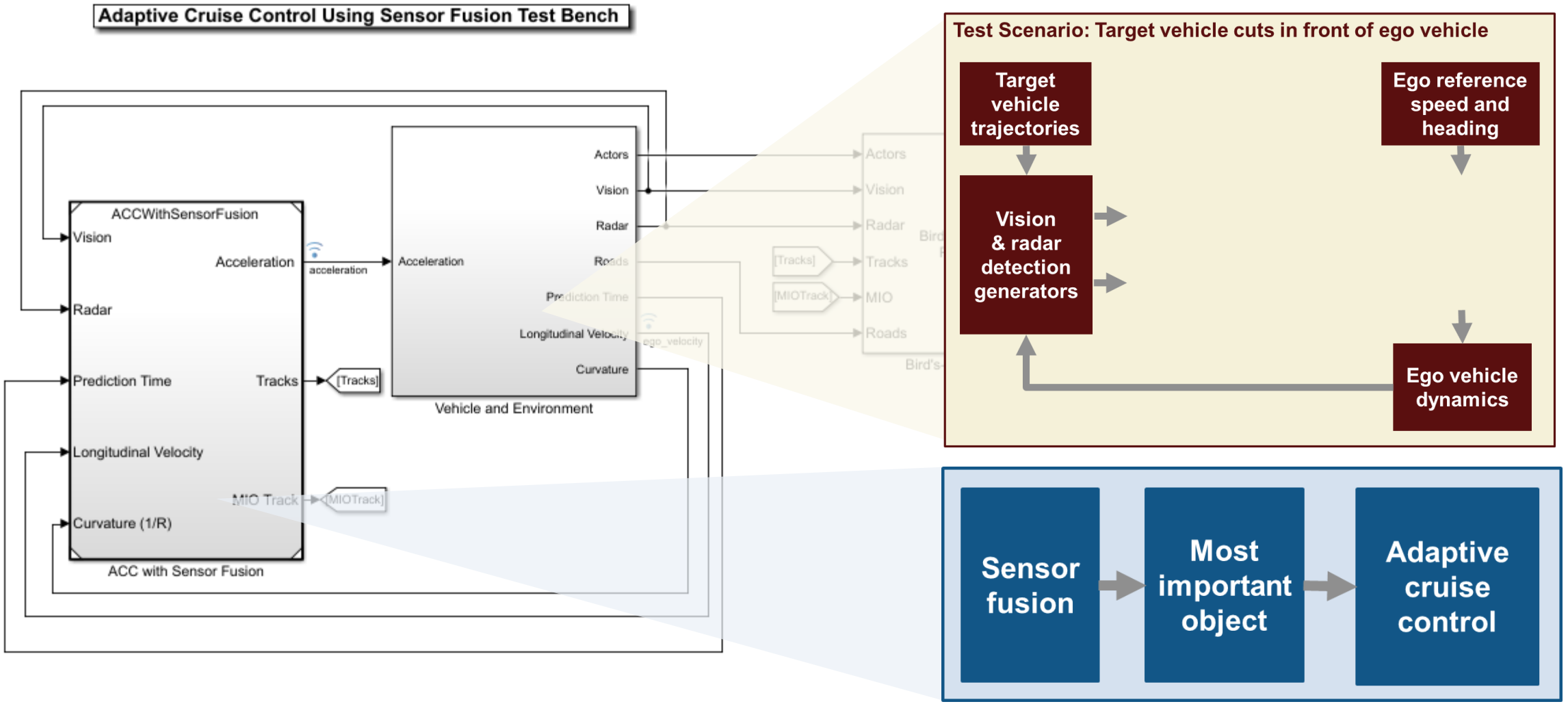


- Automated Driving System Toolbox
- Model Predictive Control Toolbox

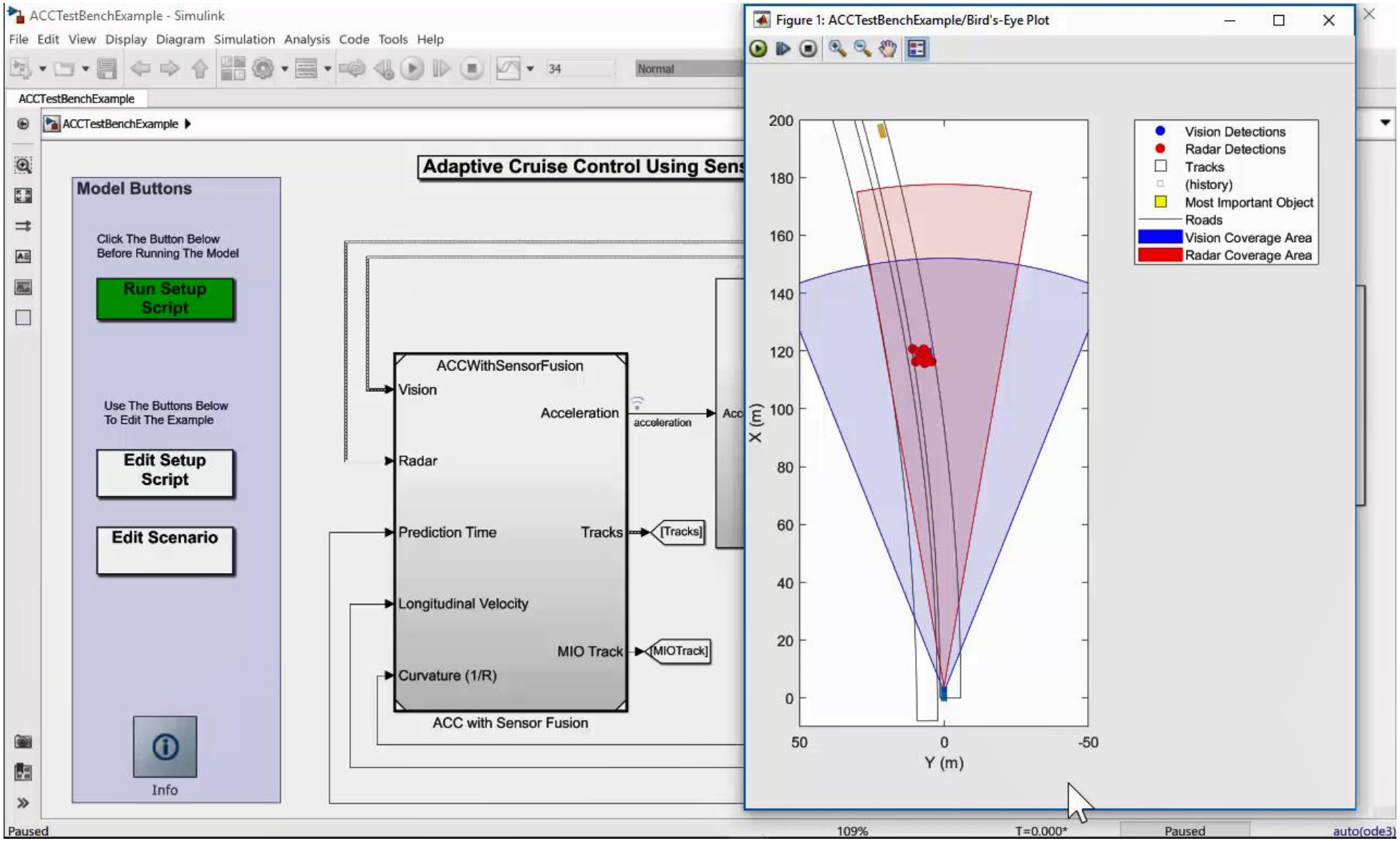


Explore modeling patterns in shipping application example:

Adaptive Cruise Control with Sensor Fusion

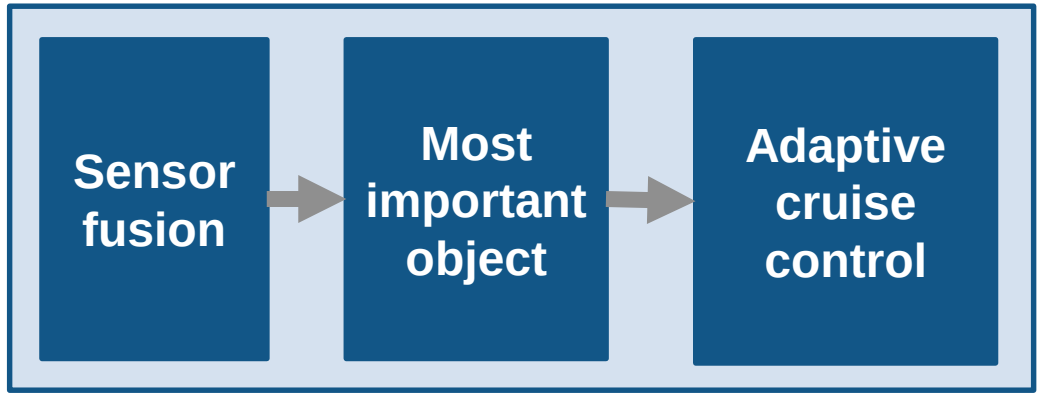


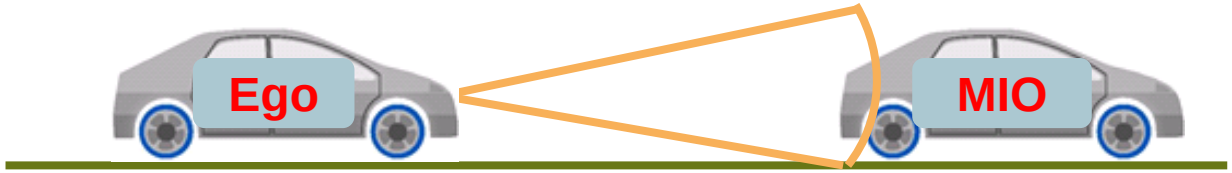
Run simulation and visualize results



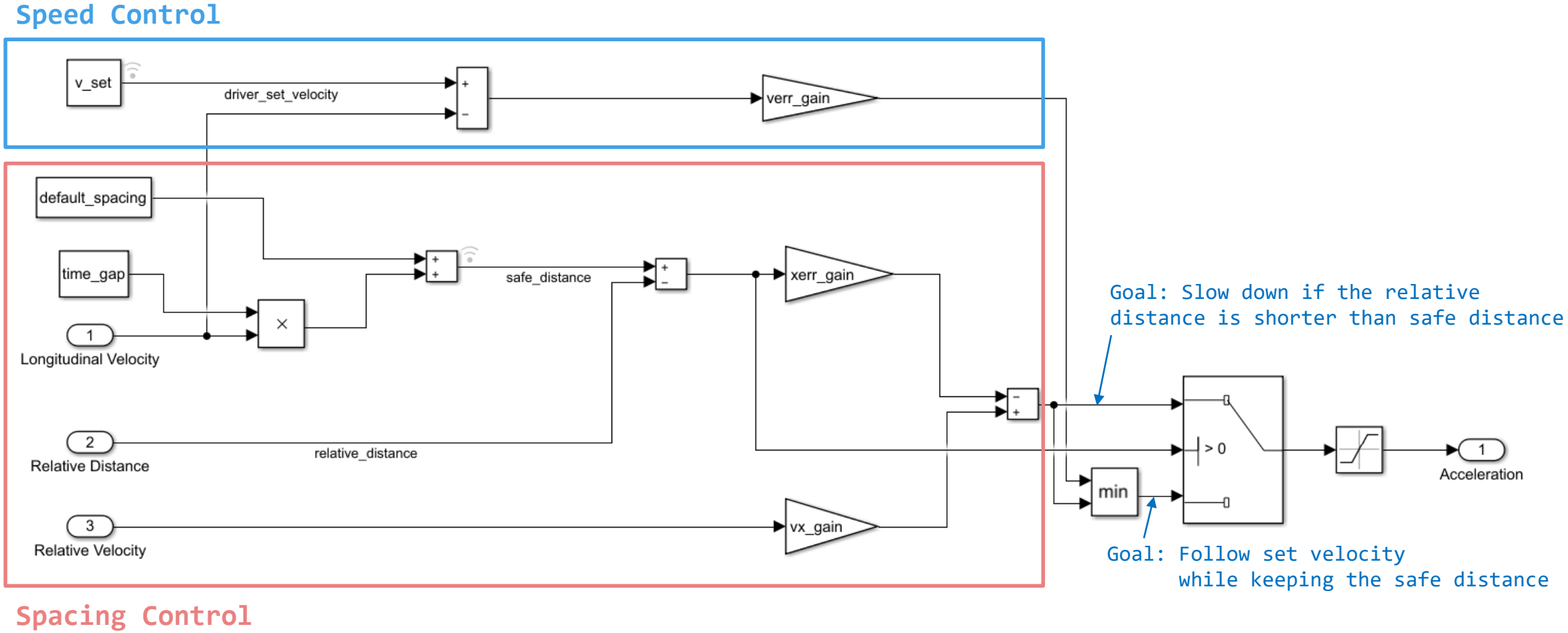
Classic adaptive cruise control modes of operation

1. **Speed control**: Ego vehicle travels at a driver-set velocity
2. **Spacing control**: Ego vehicle maintains a safe distance from Most Important Object (MIO) detected from sensor fusion



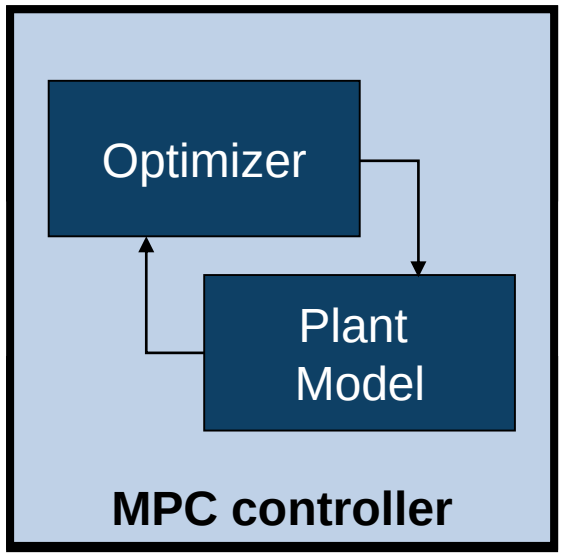


“Classical” adaptive cruise control switches between speed and spacing controllers



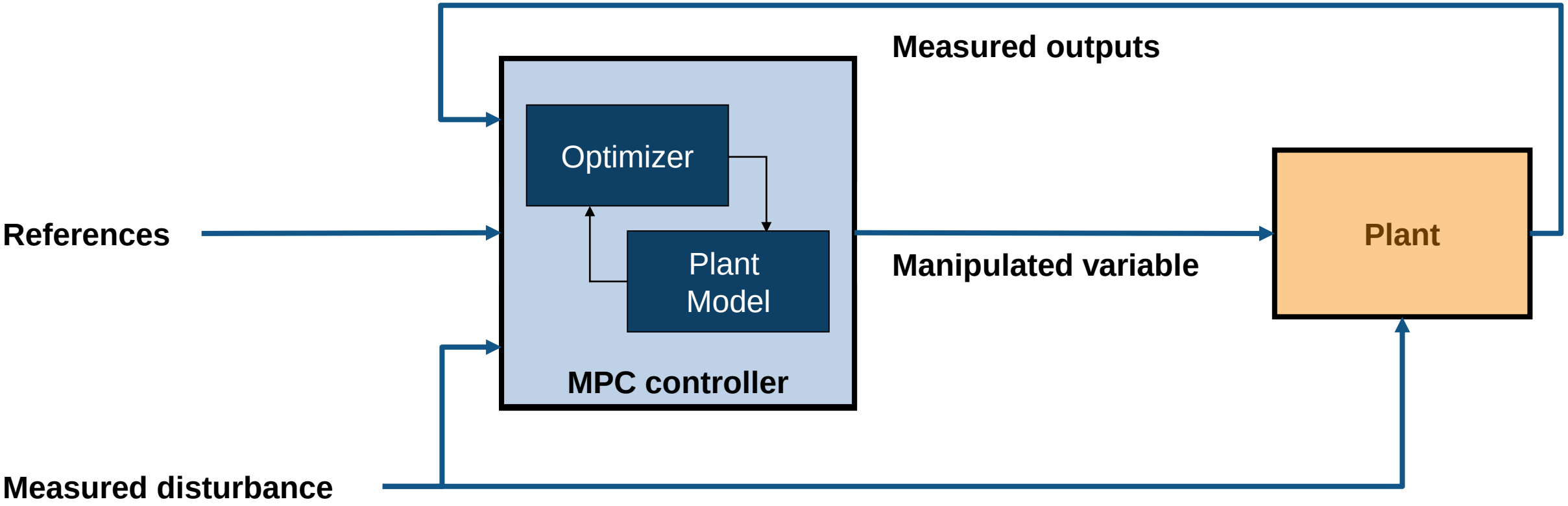
What is model predictive control (MPC)?

- **Multi-variable control** strategy leveraging an internal model to predict plant behavior in the near future
- **Optimizes** for the current timeslot while keeping future timeslots in account



- **Mature** control solution used in industrial applications
- **Gaining popularity in automated driving** applications to improve vehicle responsiveness while maintaining passenger comfort

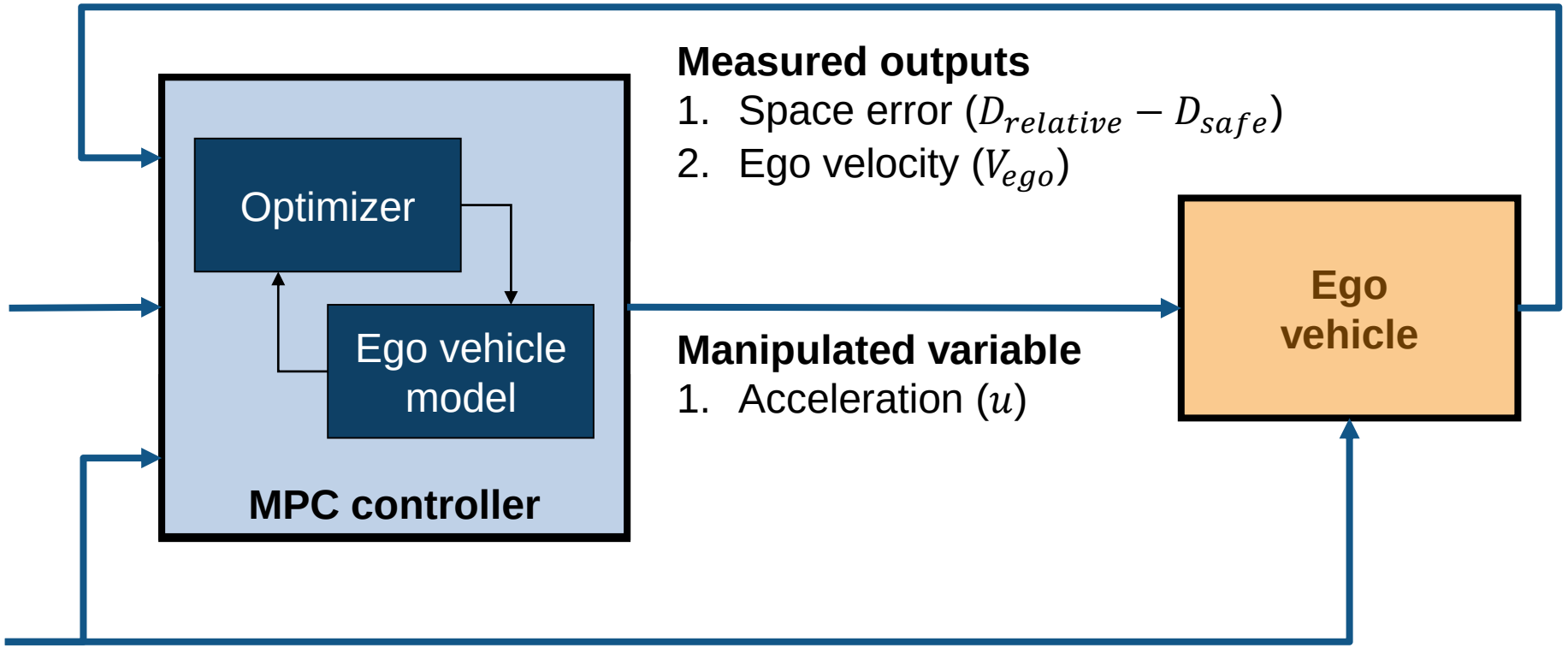
What is model predictive control (MPC)?



How can MPC be applied to adaptive cruise control?

References
1. Zero space error
2. Ego velocity set point (V_{set})

Measured disturbance
1. MIO velocity



Measured outputs
1. Space error ($D_{relative} - D_{safe}$)
2. Ego velocity (V_{ego})

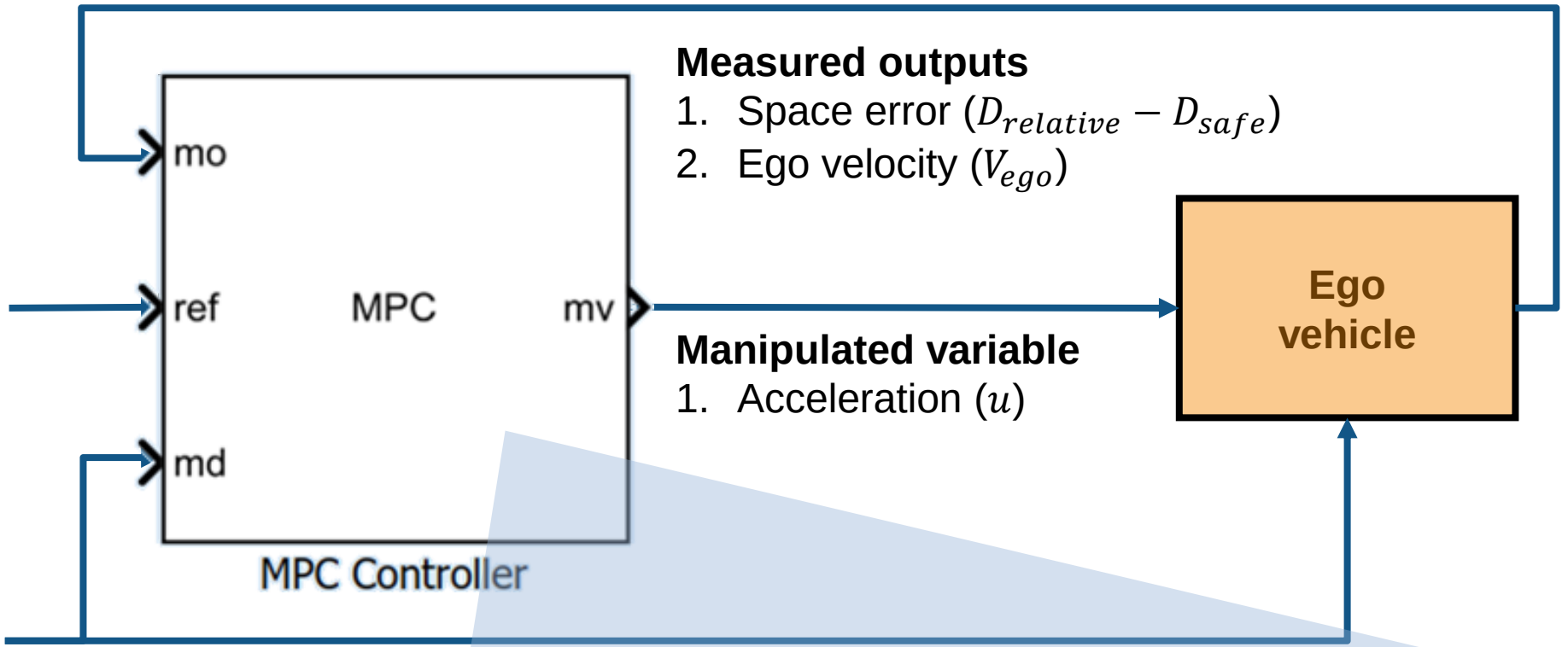
Manipulated variable
1. Acceleration (u)

$$\text{minimize:} \quad \left|V_{ego} - V_{set}\right|^2$$
$$\text{subject to:} \quad D_{relative} - D_{safe} \geq 0$$
$$u_{min} \leq u \leq u_{max}$$

How can MPC be applied to adaptive cruise control?

References
1. Zero space error
2. Ego velocity set point (V_{set})

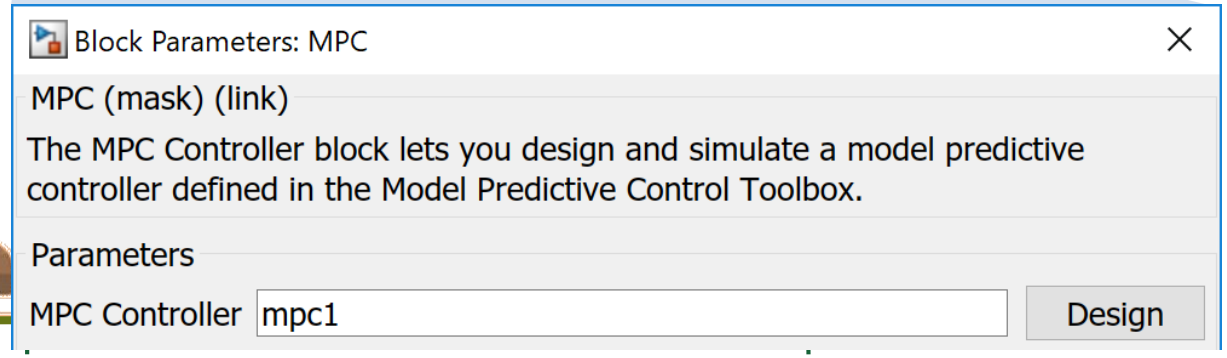
Measured disturbance
1. MIO velocity



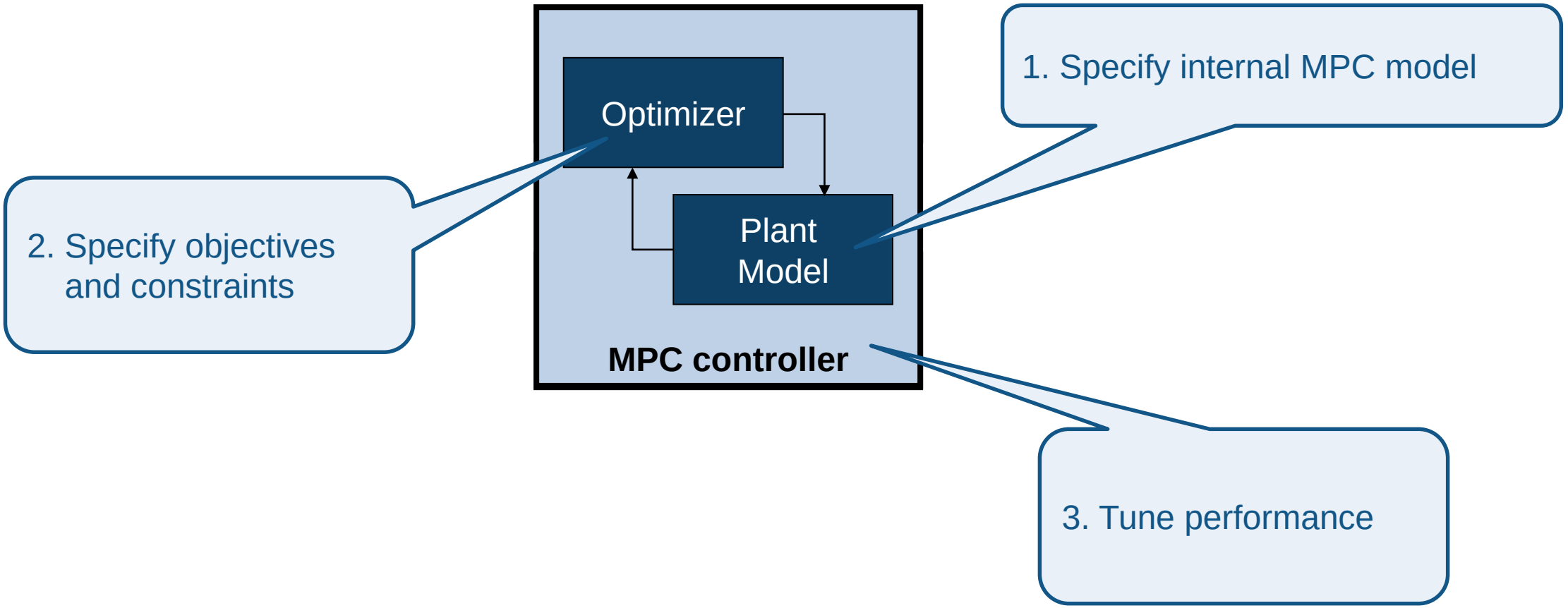
Measured outputs
1. Space error ($D_{relative} - D_{safe}$)
2. Ego velocity (V_{ego})

Manipulated variable
1. Acceleration (u)

$$\text{minimize:} \quad |V_{ego} - V_{set}|^2$$
$$\text{subject to:} \quad D_{relative} - D_{safe} \geq 0$$
$$u_{min} \leq u \leq u_{max}$$



Steps to design MPC controller



1. Specify internal MPC model for ACC

$$\begin{pmatrix} R \\ V_{ego} \end{pmatrix} = \begin{bmatrix} -\left(\frac{1}{s} + Tgap\right)G(s) & \frac{1}{s} \\ G(s) & 0 \end{bmatrix} \begin{pmatrix} u \\ V_{mio} \end{pmatrix}$$

Symbol	Description	MPC internal model
T_{gap}	Desired time gap	Constant parameter
R	Spacing error	Measured output
V_{ego}	Longitudinal velocity for ego	Measured output
u	Longitudinal acceleration or deceleration	Manipulated variable
V_{mio}	Longitudinal velocity for lead vehicle (most important object)	Measured disturbance
$G(s)$	Model from longitudinal acceleration to longitudinal velocity	----------

Estimating output (R) and disturbance (V_{mio}) from sensors

Symbol	Description	Source / Calculation
$D_{relative}$	Distance from ego to MIO	Output of sensor fusion
$V_{relative}$	Relative velocity of MIO with respect to ego	Output of sensor fusion
V_{ego}	Longitudinal velocity for ego vehicle	Output of sensor fusion
T_{gap}	Desired time gap	Constant parameter
D_{offset}	Minimum safe distance between ego and MIO	Constant parameter
D_{safe}	Safe distance from ego to MIO	$= T_{gap}V_{ego} + D_{offset}$
$\boldsymbol{R}$	Spacing error	$= Drelati_{ve} - (TgapV_{ego} + D_{offset})$
$\boldsymbol{V_{mio}}$	Longitudinal velocity for MIO	$= V_{relative} + V_{ego}$

Derivation of internal model for MPC design

- Internal model:

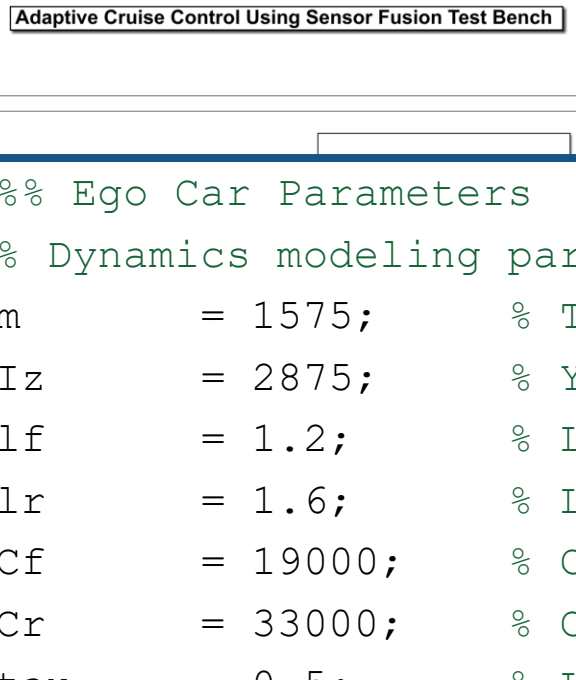
$$\begin{pmatrix} R \\ V_{ego} \end{pmatrix} = \begin{bmatrix} -\left(\frac{1}{s} + Tgap\right)G(s) & \frac{1}{s} \\ G(s) & 0 \end{bmatrix} \begin{pmatrix} u \\ V_{lead} \end{pmatrix}$$

- The dynamics from u to V_{ego} is given by $G(s)$, i.e. $V_{ego} = G(s)u$.
- The dynamics from $\begin{pmatrix} u \\ V_{lead} \end{pmatrix}$ to spacing error R is given by

$$\begin{aligned} R &= D_{relative} - D_{safe} \\ &= \frac{1}{s}\left(V_{lead} - V_{ego}\right) - T_{gap}V_{ego} \\ &= -\left(\frac{1}{s} + T_{gap}\right)V_{ego} + \frac{1}{s}V_{lead} \\ &= -\left(\frac{1}{s} + Tgap\right)G(s)u + \frac{1}{s}V_{lead} \end{aligned}$$

1. Specify internal MPC model for ACC:

1.1 Specify vehicle parameters

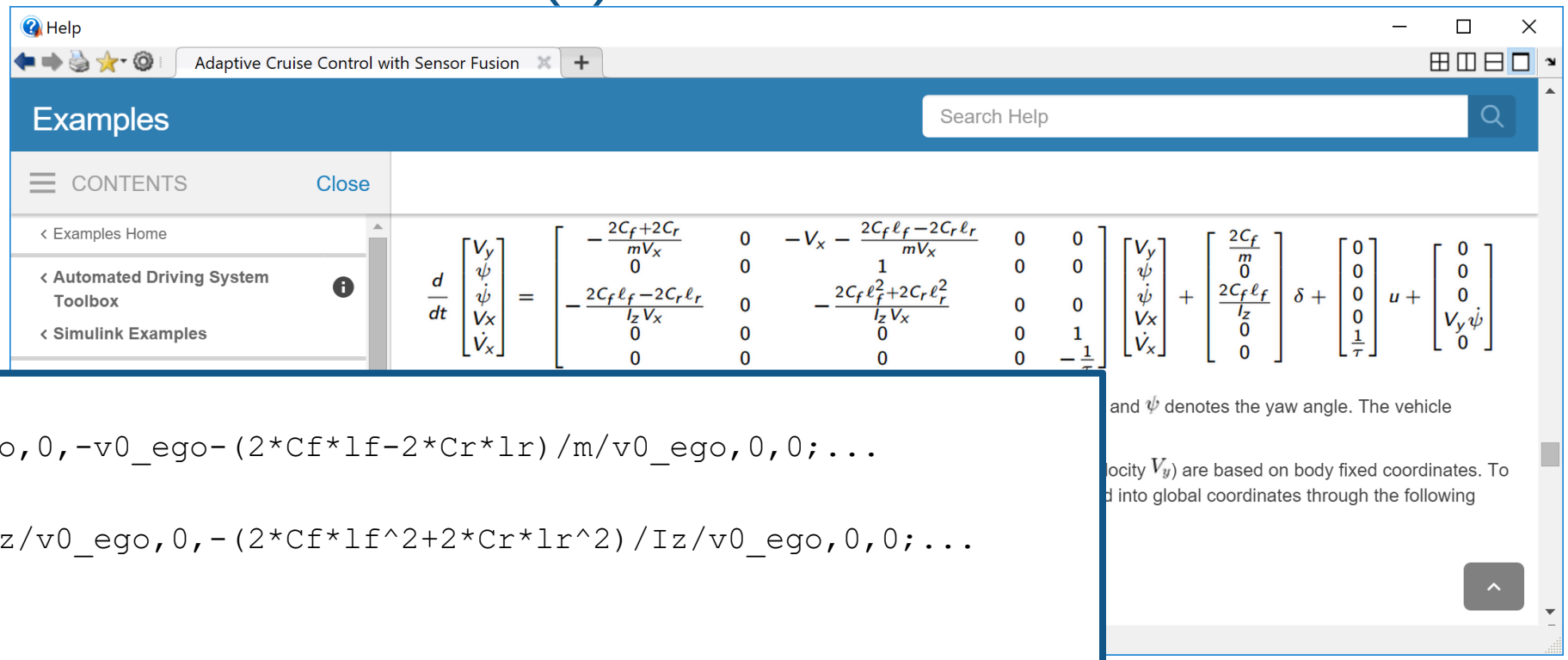



```
%% Ego Car Parameters
% Dynamics modeling parameters
m        = 1575;      % Total mass of vehicle                          (kg)
Iz       = 2875;      % Yaw moment of inertia of vehicle               (m*N*s^2)
lf       = 1.2;       % Longitudinal distance from c.g. to front tires (m)
lr       = 1.6;       % Longitudinal distance from c.g. to rear tires  (m)
Cf       = 19000;     % Cornering stiffness of front tires             (N/rad)
Cr       = 33000;     % Cornering stiffness of rear tires              (N/rad)
tau      = 0.5;       % Longitudinal time constant                     (N/A)

% Initial condition for the ego car
v0_ego = 20.6;          % Initial speed of the ego car          (m/s)
x0_ego = 0;             % Initial x position of ego car         (m)
y0_ego = -757.8;        % Initial y position of ego car         (m)
```

1. Specify internal MPC model for ACC:

1.2 Specify linear vehicle model G(s)




```
% System matrices
A = [-(2*Cf+2*Cr)/m/v0_ego,0,-v0_ego-(2*Cf*lf-2*Cr*lr)/m/v0_ego,0,0;...
     0,0,1,0,0;...
     -(2*Cf*lf-2*Cr*lr)/Iz/v0_ego,0,-(2*Cf*lf^2+2*Cr*lr^2)/Iz/v0_ego,0,0;...
     0,0,0,0,1;...
     0,0,0,0,-1/tau];
% B1 = [2*Cf/m,0,2*Cf*lf/Iz,0,0]';
B2 = [0,0,0,0,1/tau]';
% Linear vehicle model
C = [0,0,0,1,0];
G = ss(A,B2,C,0);
% Convert from state-space to transfer function (frequency domain)
G = tf(G);
```

1. Specify internal MPC model for ACC:

1.3 Create transfer function model

$$\begin{pmatrix} R \\ V_{ego} \end{pmatrix} = \begin{bmatrix} -\left(\frac{1}{s} + h\right)G(s) & \frac{1}{s} \\ G(s) & 0 \end{bmatrix} \begin{pmatrix} u \\ V_{lead} \end{pmatrix}$$

```
% Model: R = 1/s*(Vlead-Vx)-time_gap*Vx; Vx = G*u or
% [R;Vx] = [-(1/s+time_gap)*G, 1/s;G,0] * [u;Vlead]

% ACC parameters
time_gap         = 1.5;  % ACC time gap (s)

% MPC internal model
s = tf('s');
sys = [-(1/s+time_gap)*G, 1/s;G,0];
% Minimal realization
sys = minreal(sys);
```

2. Specify objectives and constraints

2.1 Create MPC object and scale signals

```
%% MPC controller
% MV = Manipulated Variables
% MD = Measured Disturbances
% DV = Disturbance Variables = Measured Disturbances
% OV = Output Variables      = Measured Outputs
sys = setmpcsignals(sys,'MV',1,'MD',2);
mpc1 = mpc(sys,Ts);

%% Specify the controller nominal values based on the simulation initial conditions.
mpc1.Model.Nominal.Y(1) = (x0_lead-x0_ego)-(default_spacing+time_gap*v0_ego);
mpc1.Model.Nominal.Y(2) = v0_ego;
mpc1.Model.Nominal.U(1) = 0;
mpc1.Model.Nominal.U(2) = v0_lead;

%% Specify scale factors based on the operating ranges of the variables.
mpc1.MV.ScaleFactor = 5;       % range of acceleration is 2 - (-3)
mpc1.DV.ScaleFactor = 20;      % typical lead car velocity
mpc1.OV(1).ScaleFactor = 50;   % typical spacing error
mpc1.OV(2).ScaleFactor = 20;   % typical ego car velocity
```

Specify initial conditions since all vehicle are moving at start of simulation

Normalize dynamic ranges to improve optimization

2. Specify objectives and constraints

2.2 Specify optimization weights and constraints

minimize: $|V_{ego} - V_{set}|^2$

subject to: $D_{relative} - D_{safe} \geq 0$

$u_{min} \leq u \leq u_{max}$

```
%% Specify weights for optimization.
mpc1.Weights.OV = [0 1];



%% Specify OV constraints.
mpc1.OV(1).Min = 0;

%% Specify MV constraints.
mpc1.MV.Min = -3; % maximum deceleration
mpc1.MV.Max = 2;  % maximum acceleration
```

Ignore space error during optimization *(weight = 0)*

Minimize error against measured velocity *(weight = 1)*

Constrain space error to ensure safe distance between ego and most important object

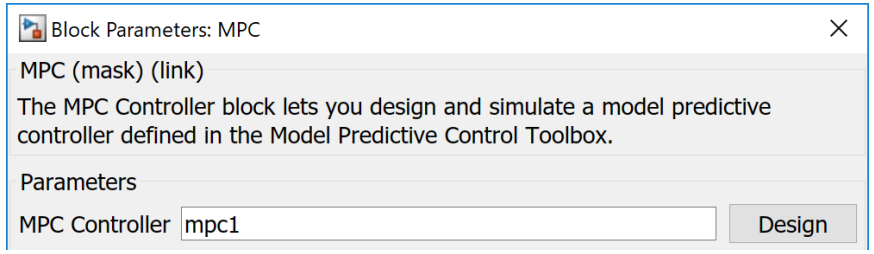
Constrain acceleration to improve ride comfort

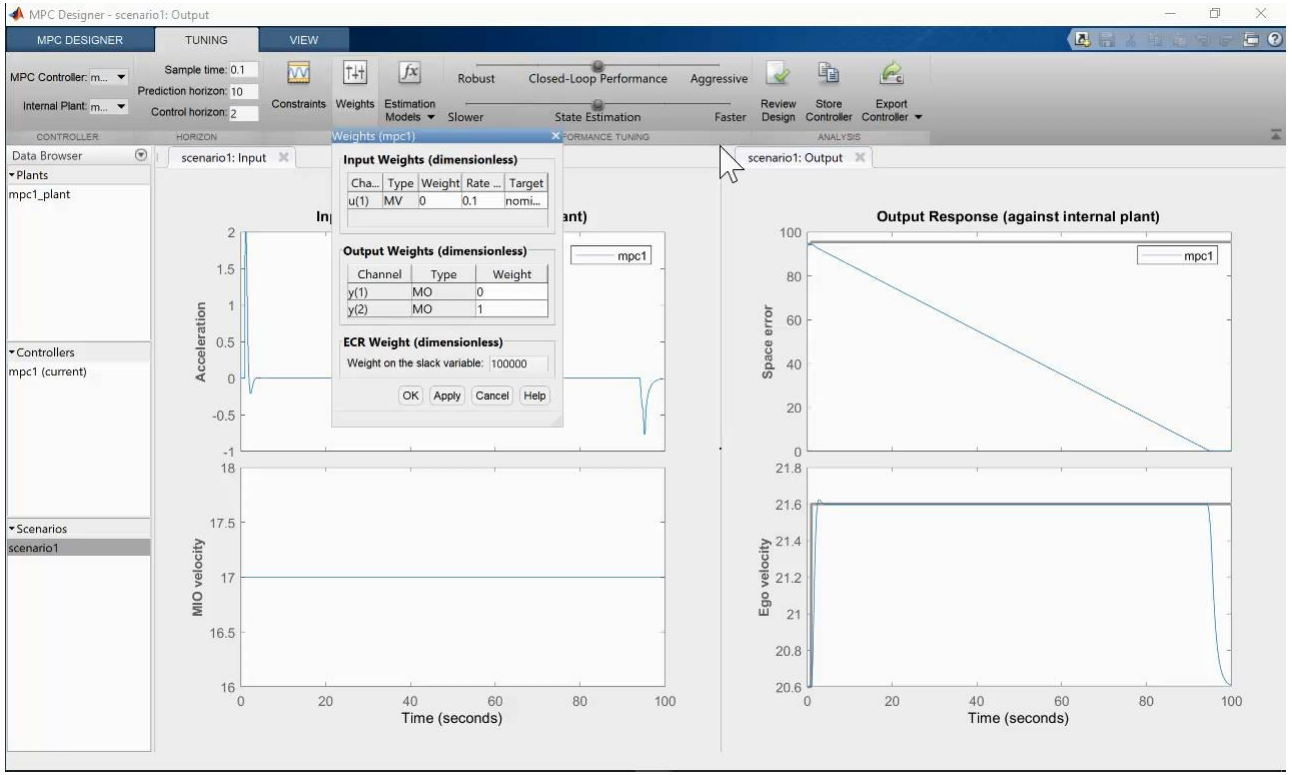
3. Tune Performance with MPC Designer App

- Open from command line

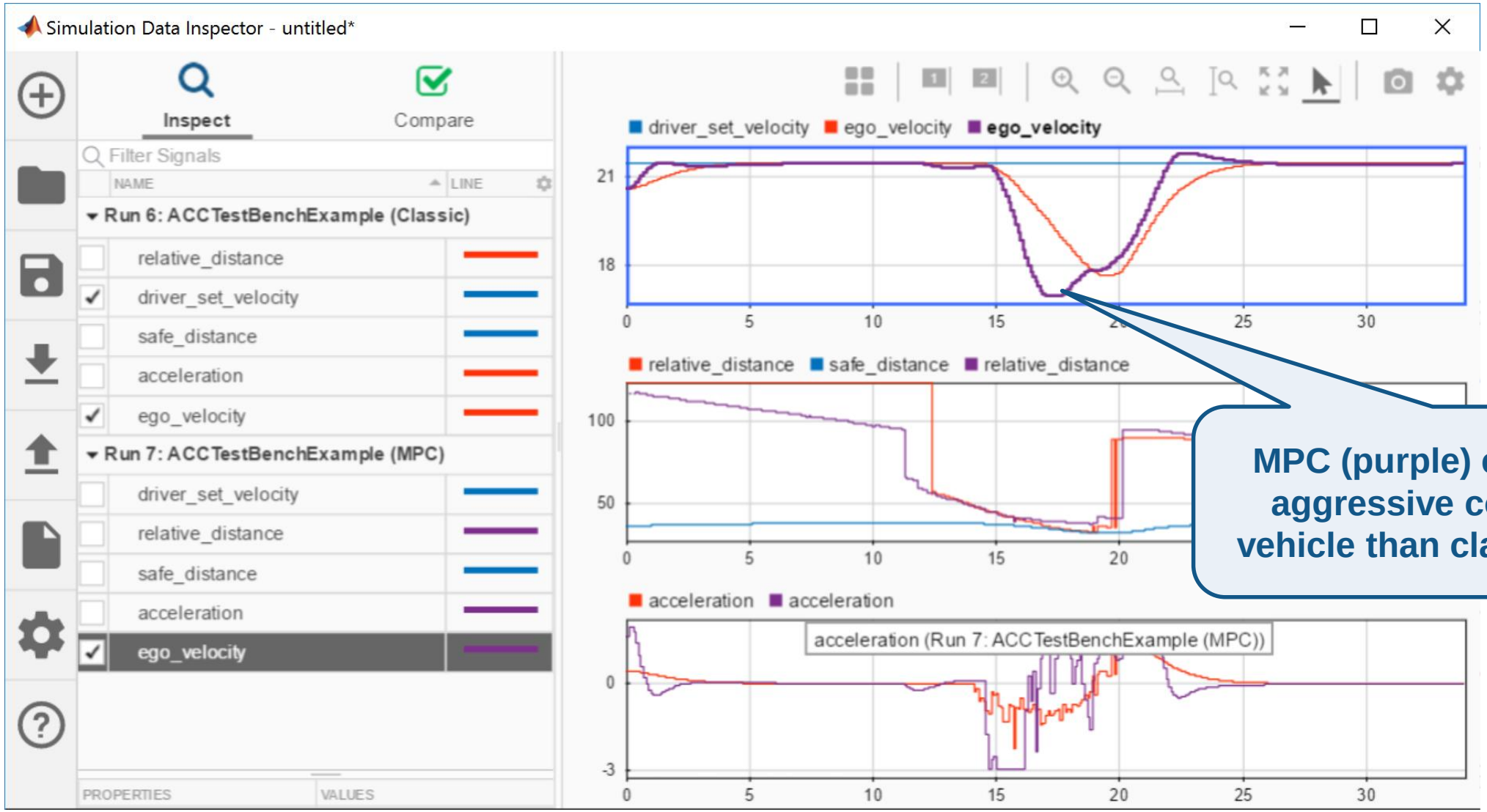
```
>> mpcDesigner(mpc1)
```

- Open with Design button on block dialog





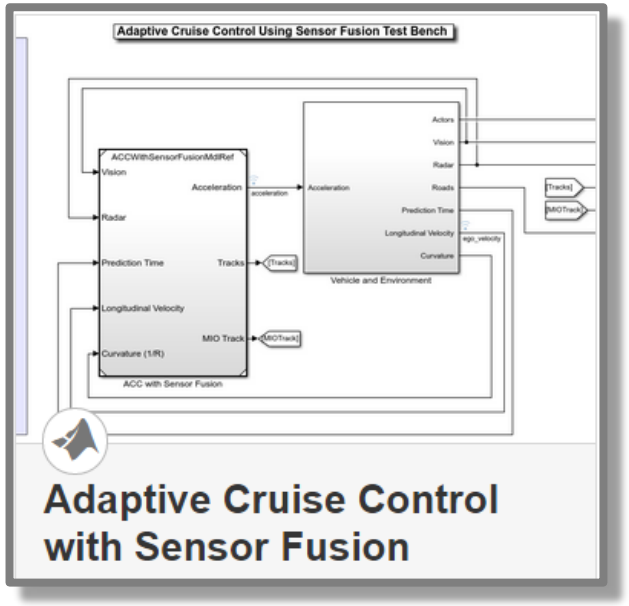
Compare performance of ACC controller variants



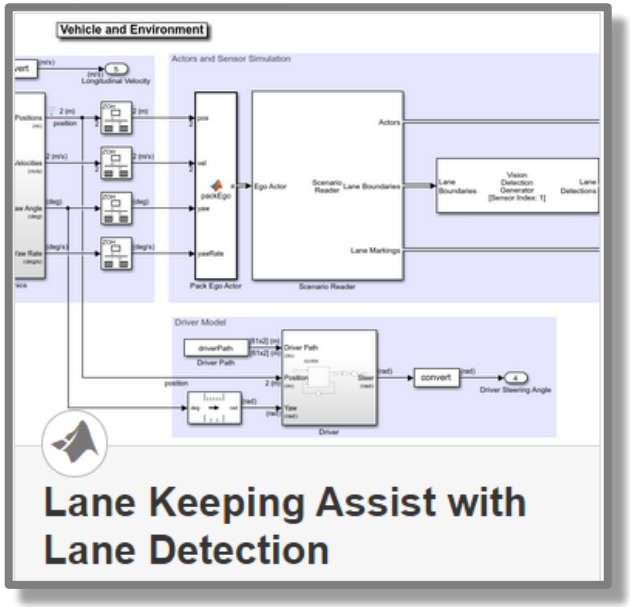
ACC and Lane Following Control for Traffic Jam Assist

Automated Driving Toolbox™

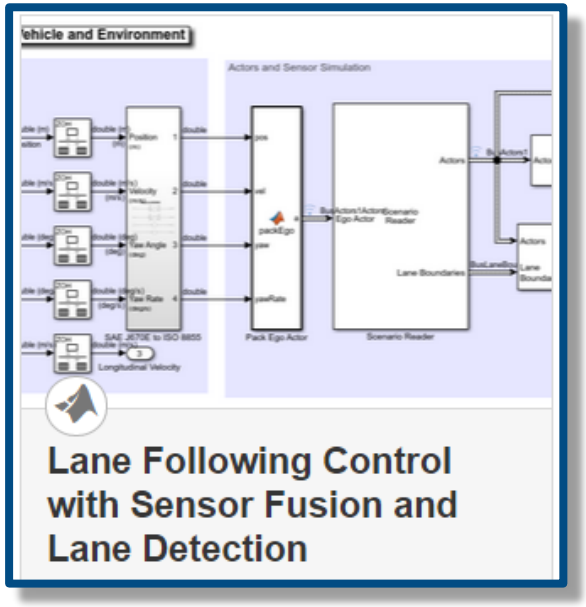
R2017b



R2018a



R2018b



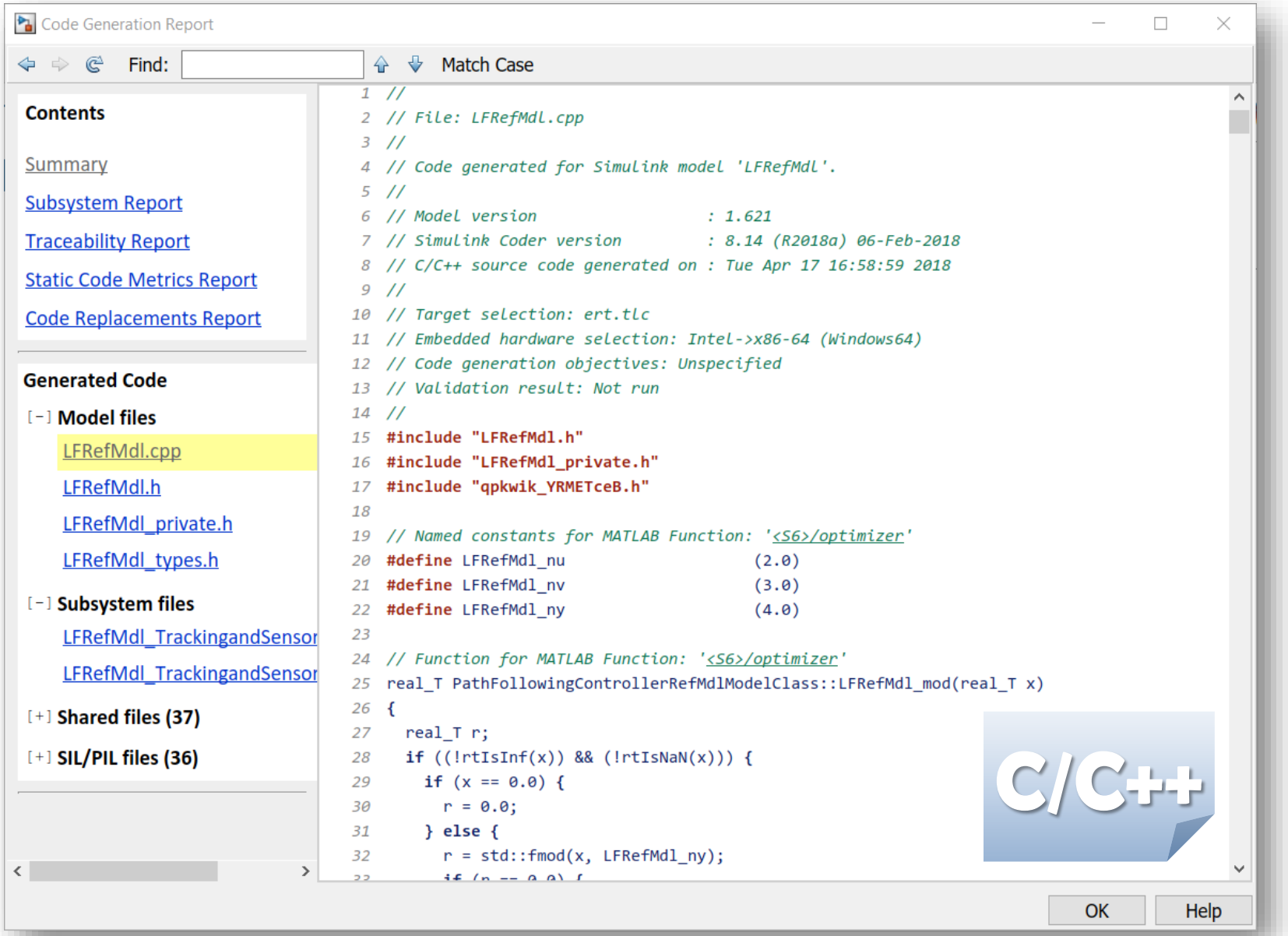
ACC
(Longitudinal Control)

Lane Following
(Lateral Control)

Traffic Jam Assist
(Longitudinal + Lateral Control)

Code Generation Support

Embedded Coder™



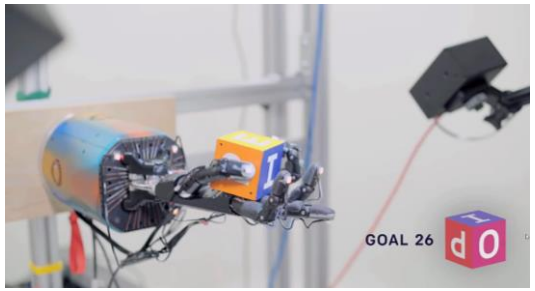
Reinforcement Learning

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

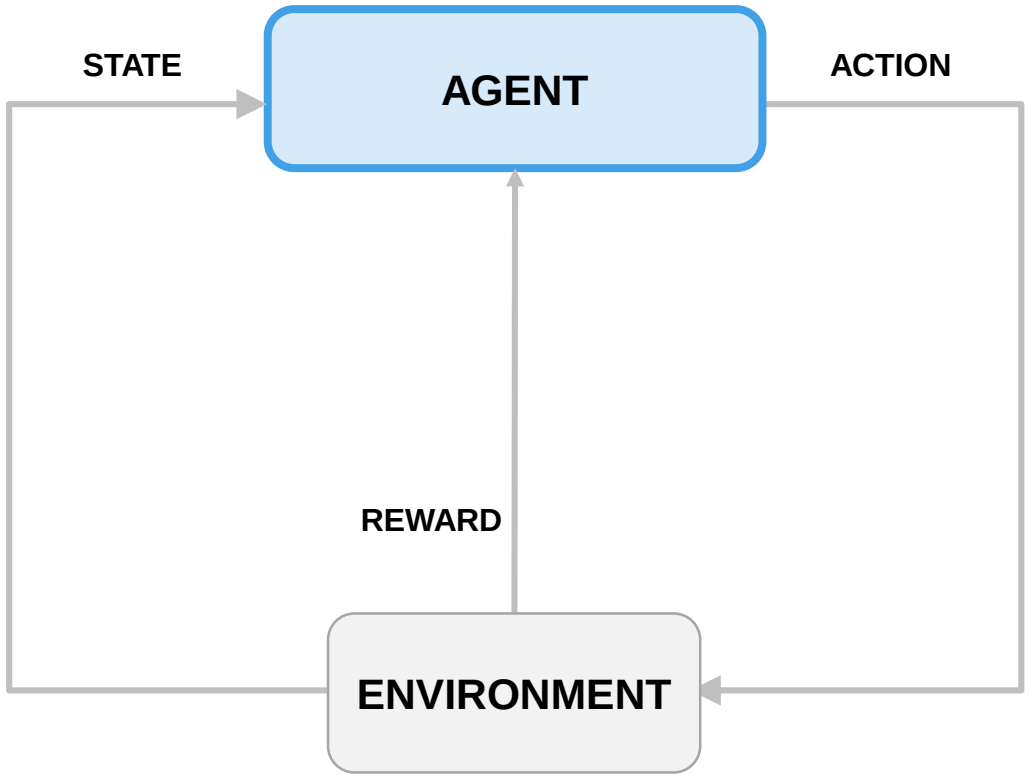
Controls

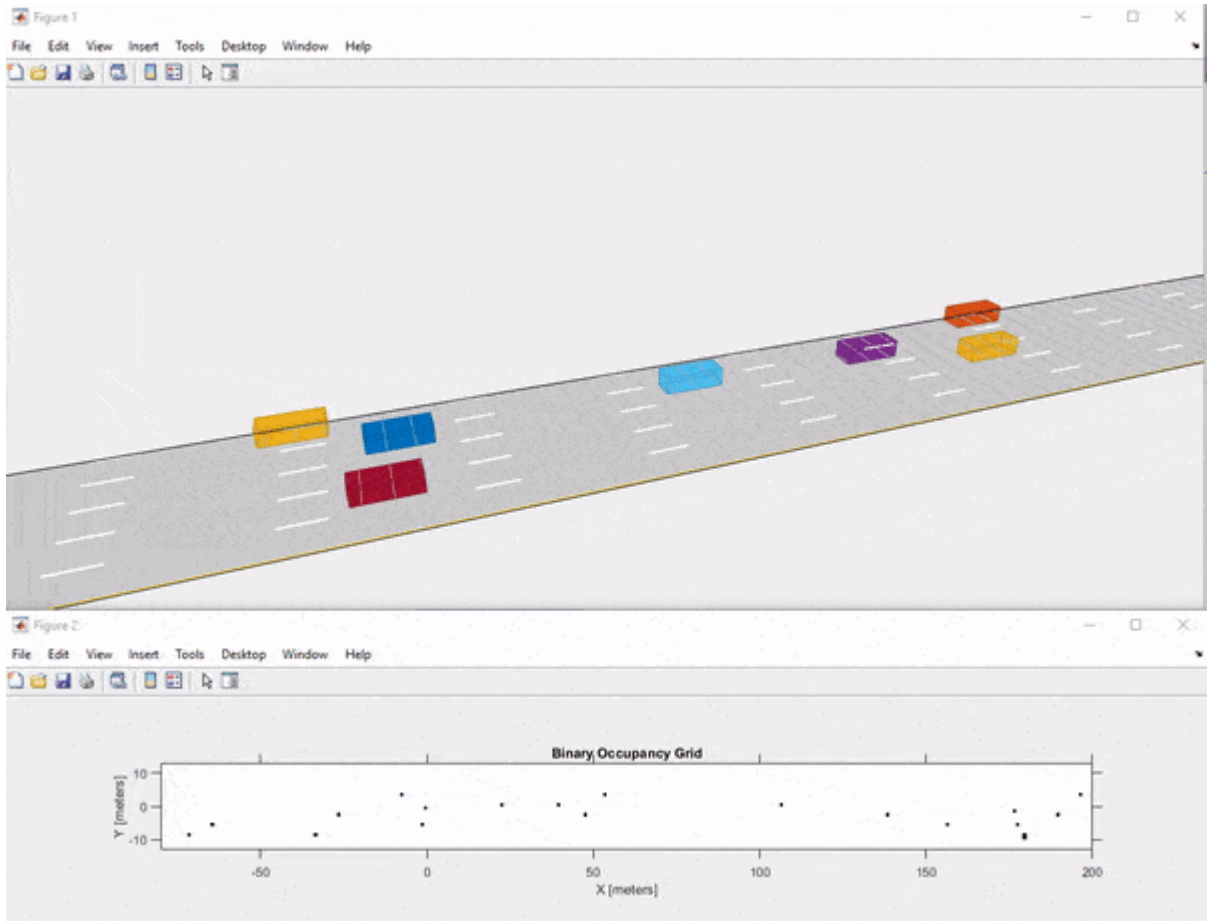


Robotics

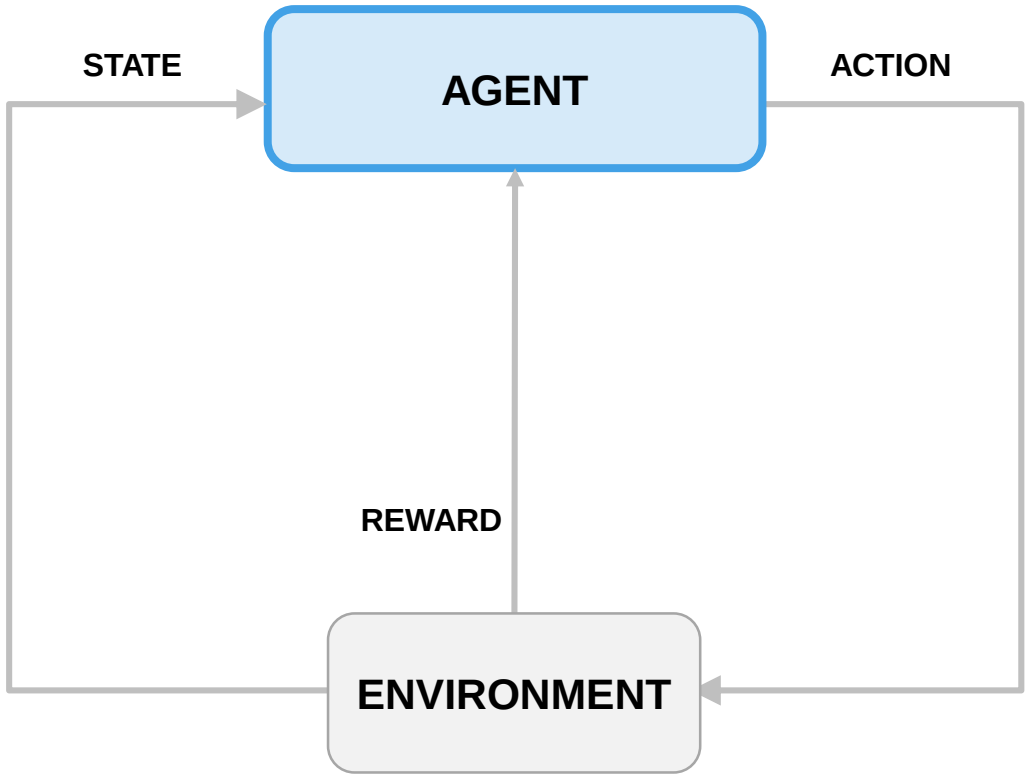
Game Play

How Does Reinforcement Learning Work?





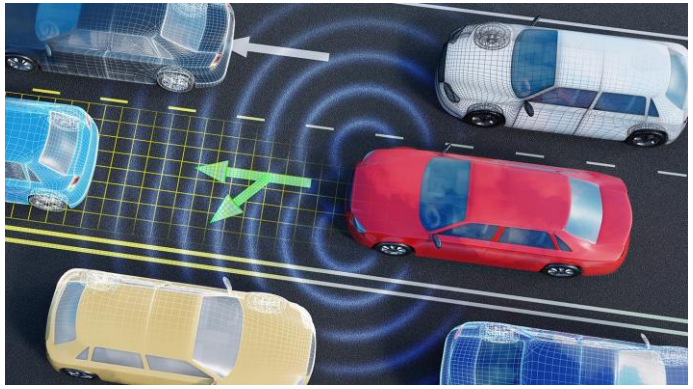
A Practical Example of Reinforcement Learning
Training a Self-Driving Car

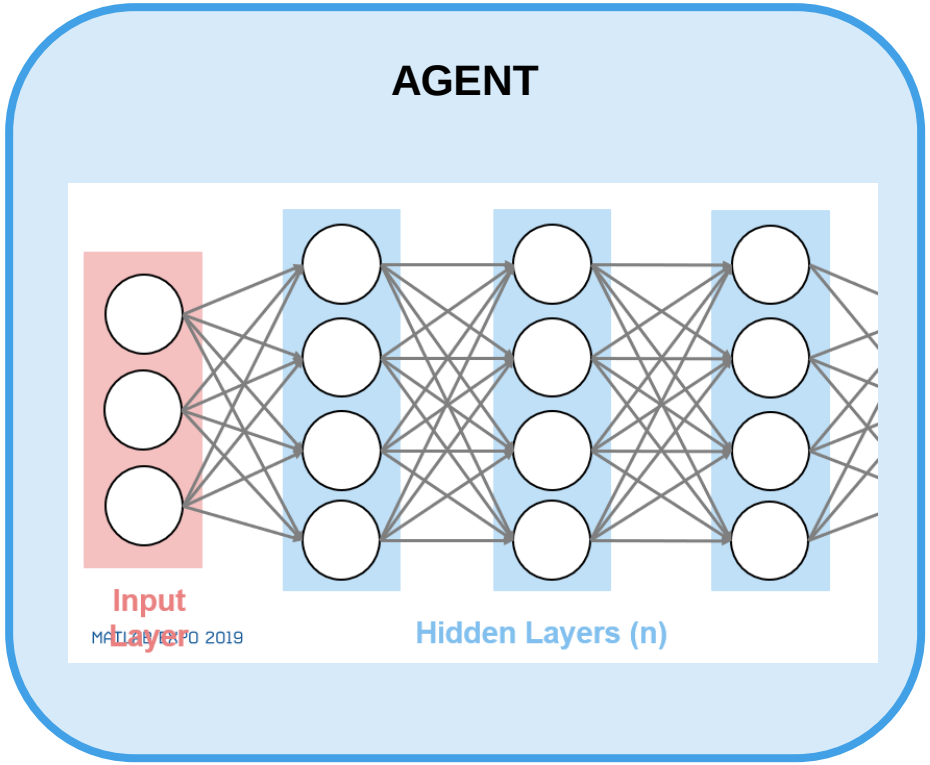


- Vehicle's computer learns how to drive… (**agent**)
- using sensor readings from LIDAR, cameras,… (**state**)
- that represent road conditions, vehicle position,… (**environment**)
- by generating steering, braking, throttle commands,… (**action**)
- to avoid collisions and lane deviation… (**reward**).

The goal of Reinforcement learning is for the agent to find an optimal algorithm for performing a task

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

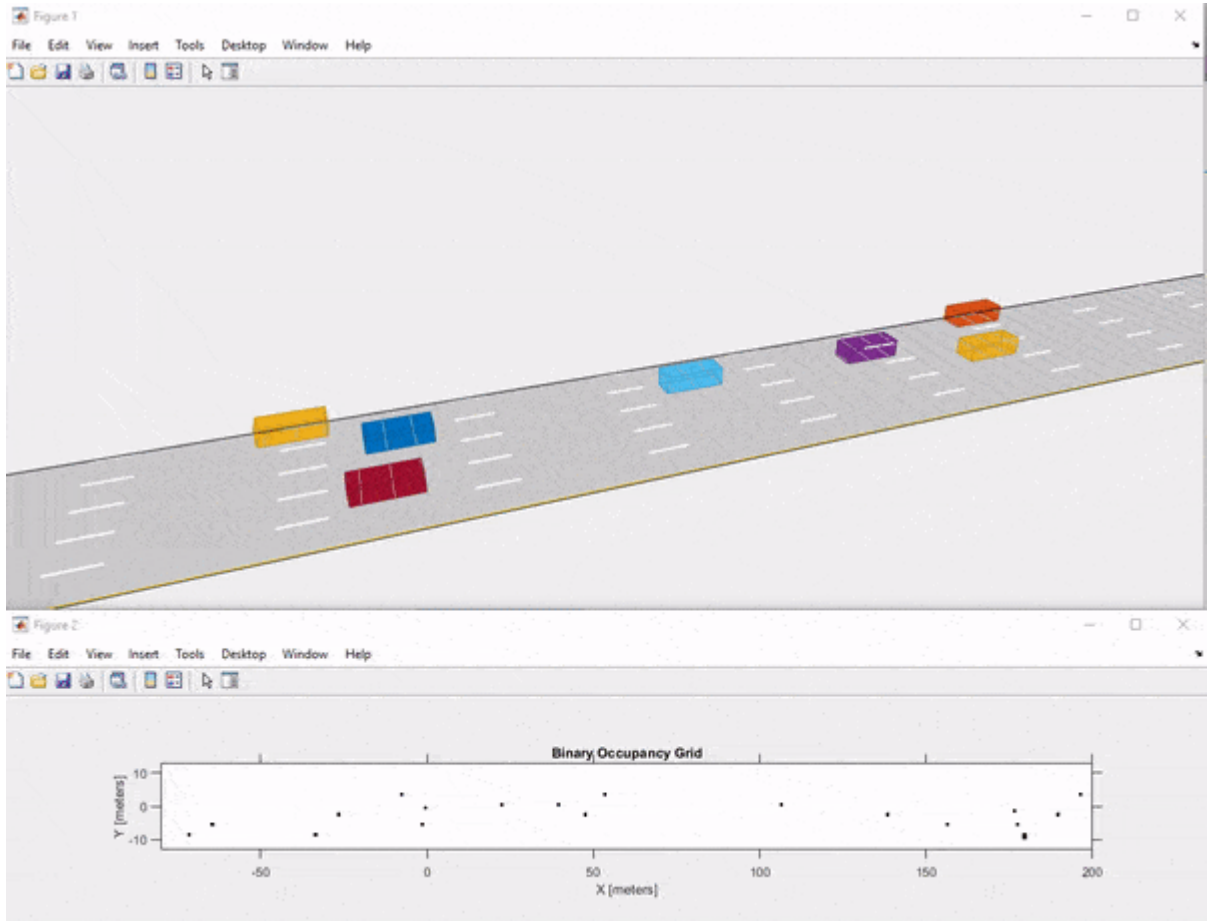
- Reinforcement learning
- Training agent to perform task
- Developing reward system to optimize performance

Deployment

- Multiplatform code generation (CPU, GPU)
- Edge deployment
- Enterprise Deployment

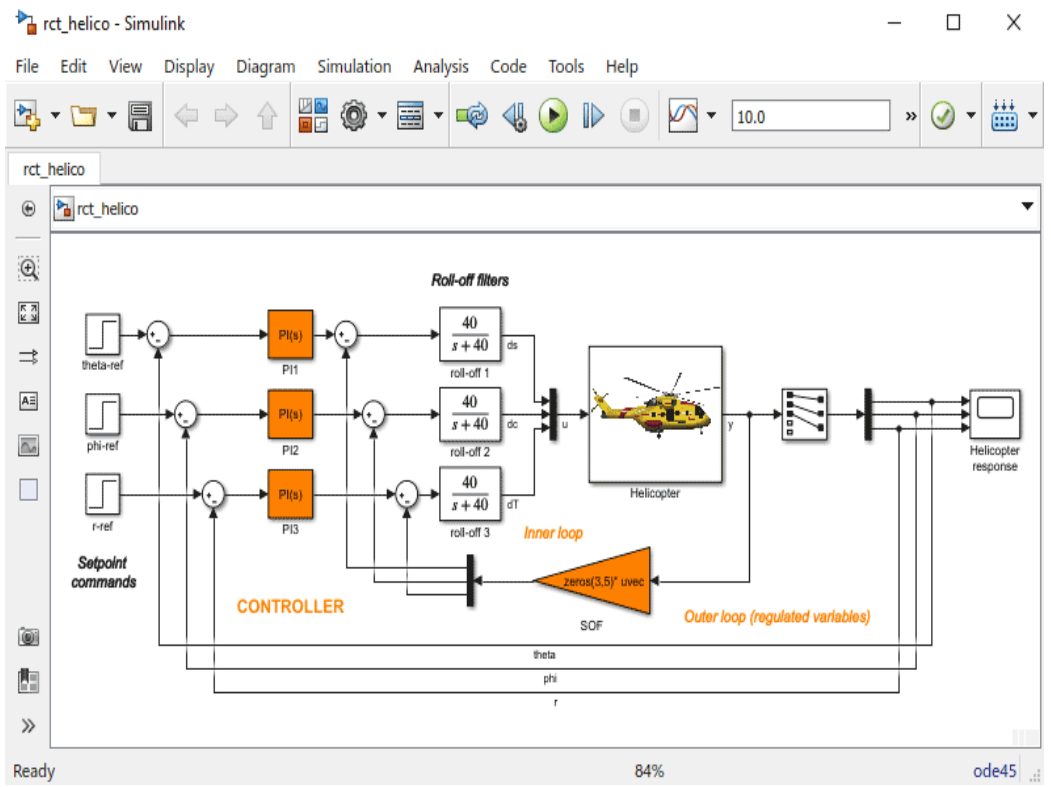
Why MATLAB and Simulink for Reinforcement Learning?

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

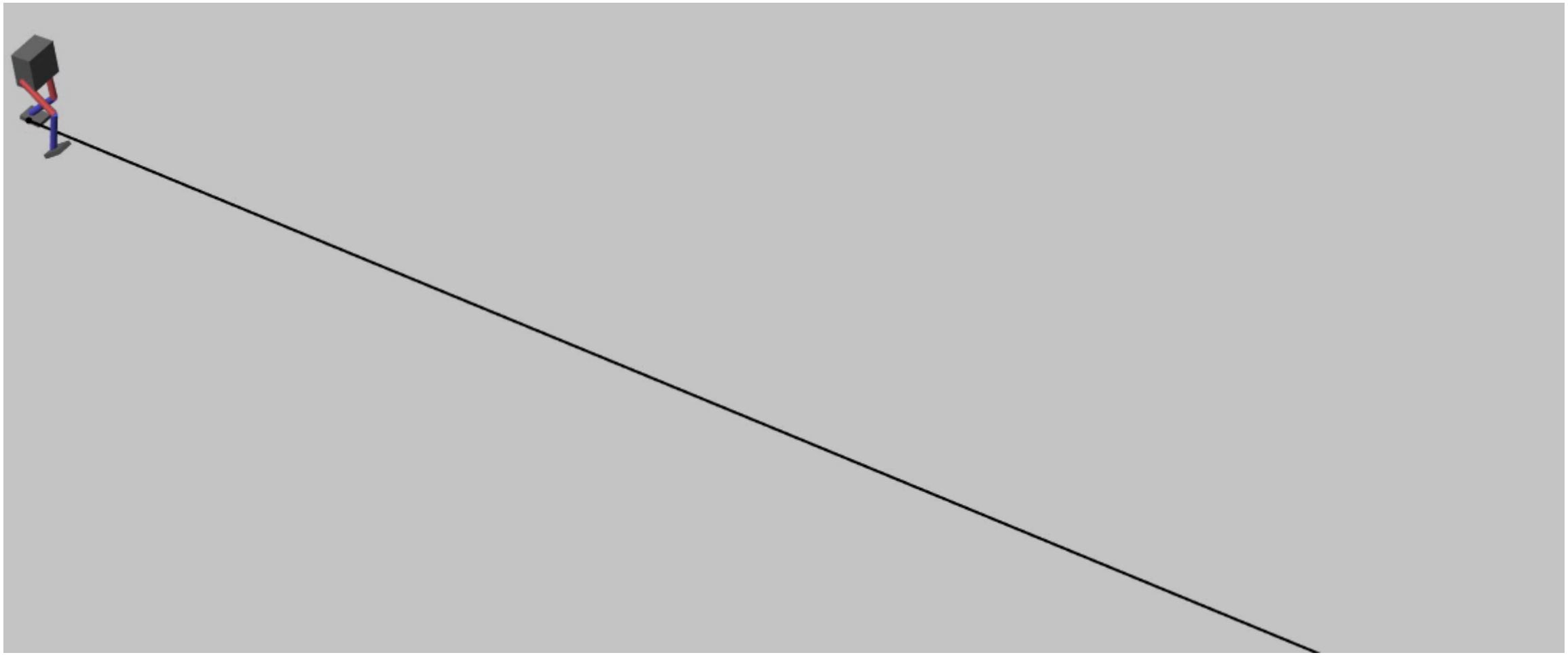


Using MATLAB and Simulink for Reinforcement Learning

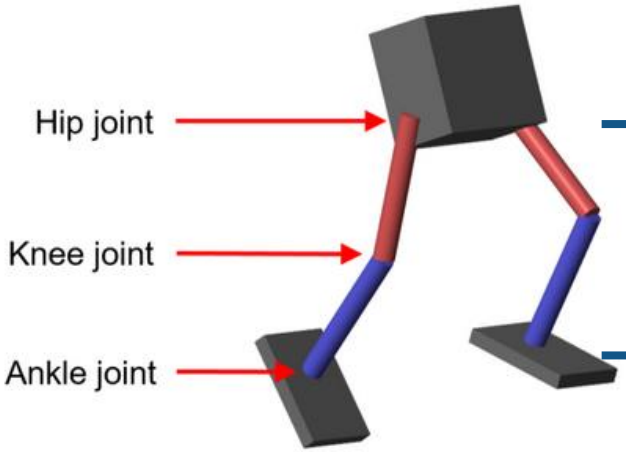
- Reinforcement learning is a dynamic process
- Decision making problems
  - Financial trading, calibration, etc.
- Controls-based problems
  - Lane-keep assist, adaptive cruise control, robotics, etc.

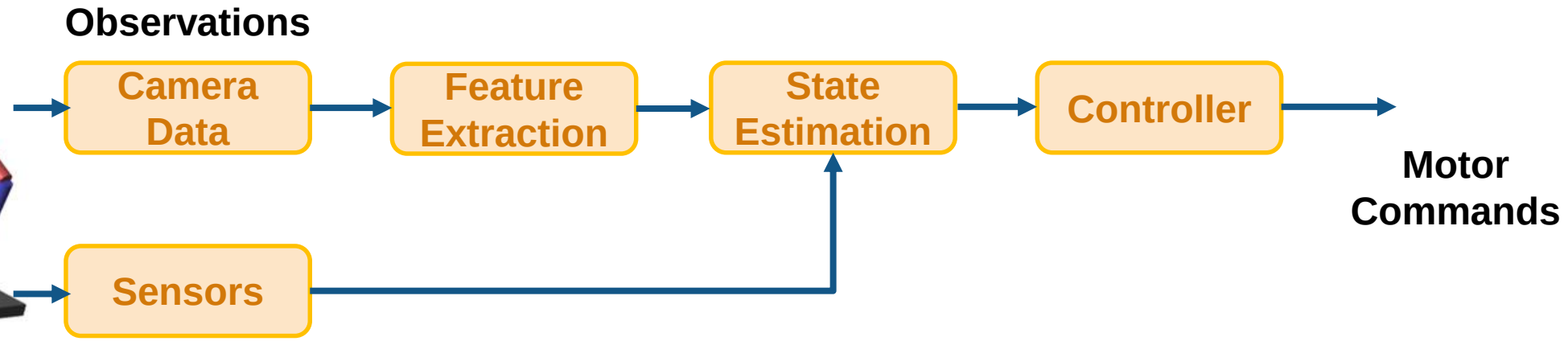


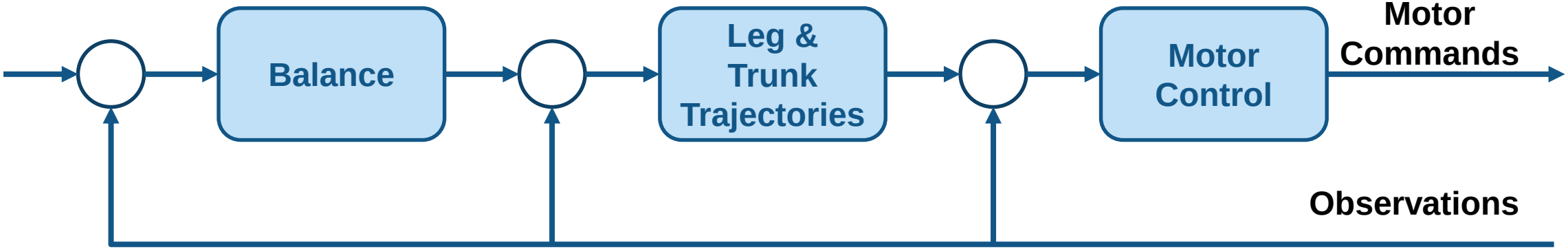
Teach a robot to follow a straight line



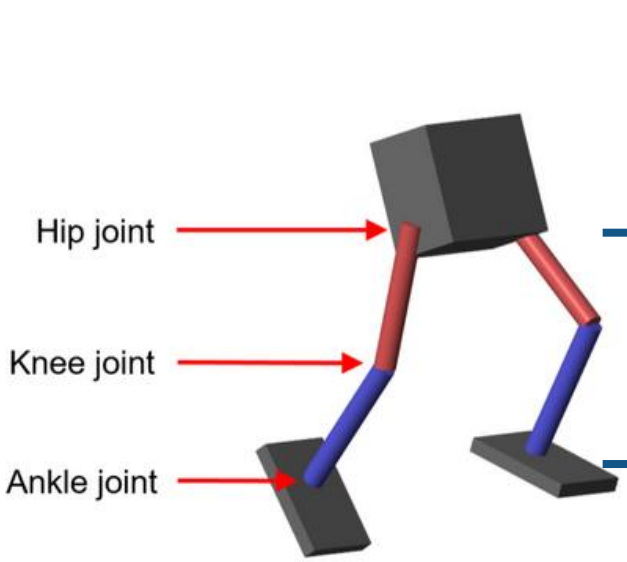
Let's try to solve this problem the traditional way







What is the alternative approach?



Observations

Camera Data

Sensors

Feature Extraction

State Estimation

Controller

Motor Commands

Camera Data

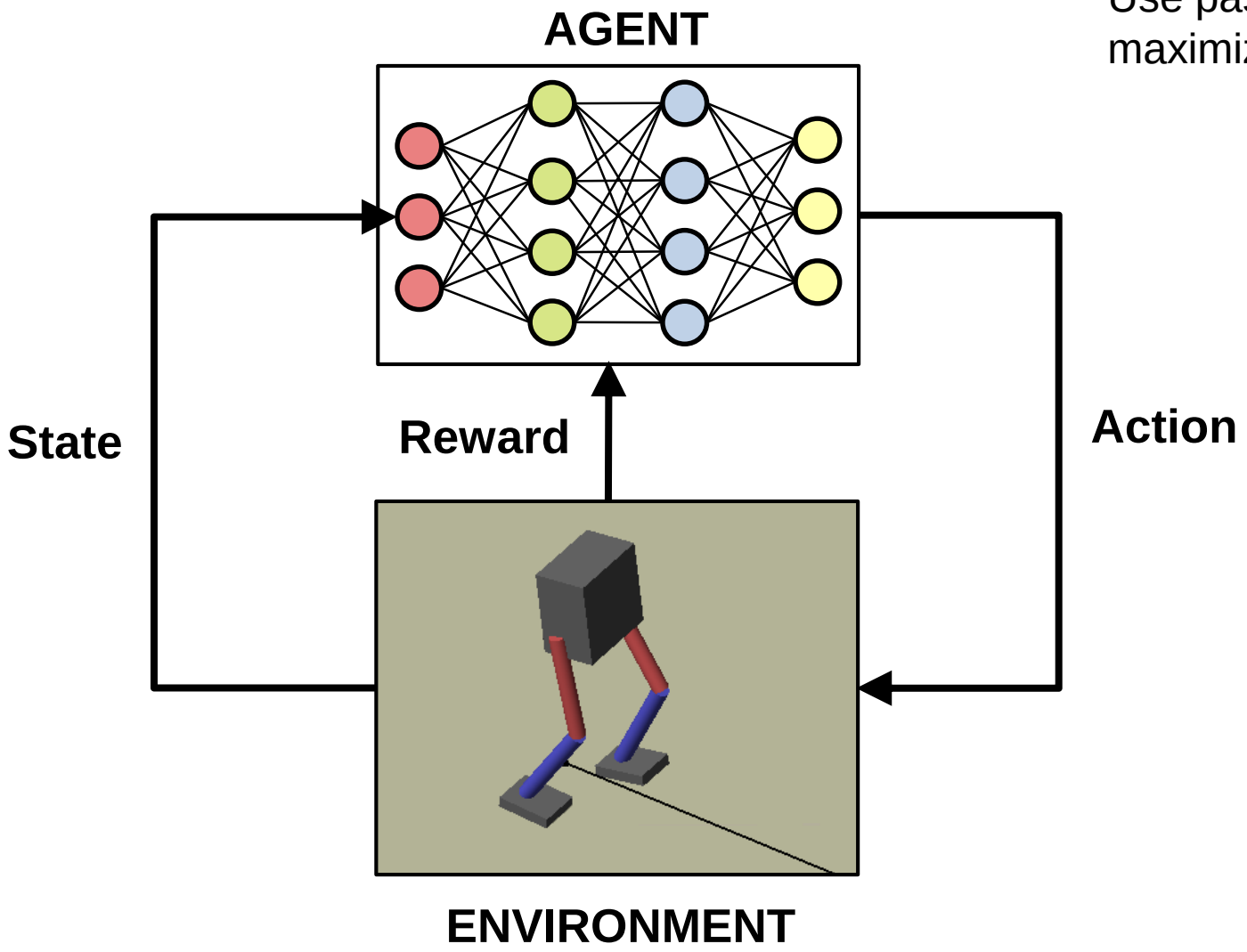
Sensors

Black Box Controller

Motor Commands

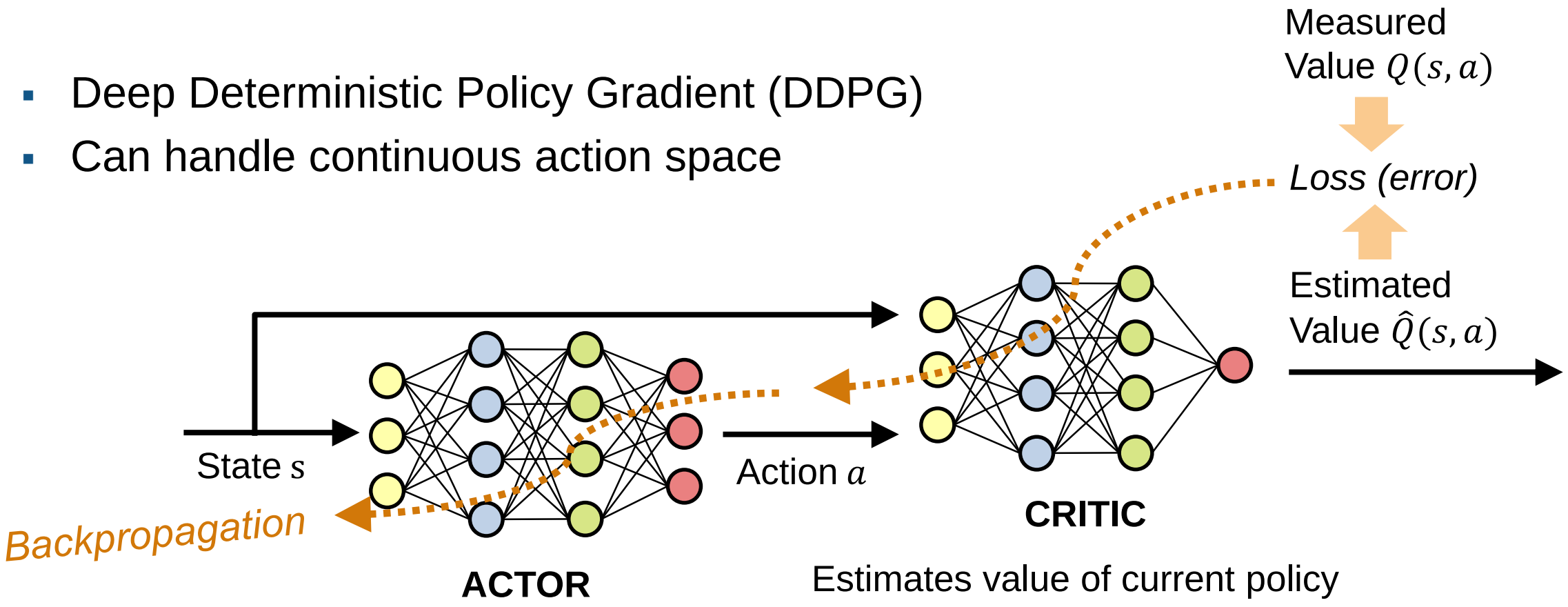
Walking Robot Example

Reinforcement Learning:
Use past experiences to maximize expected reward

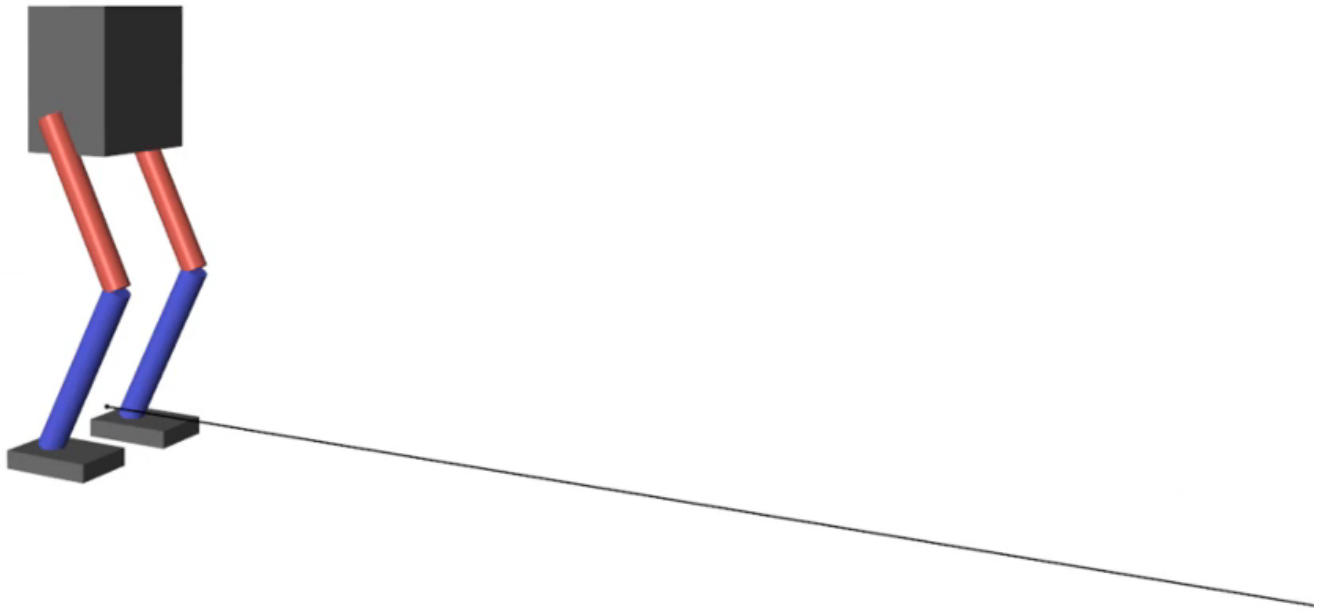


How Is the Agent Trained?

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

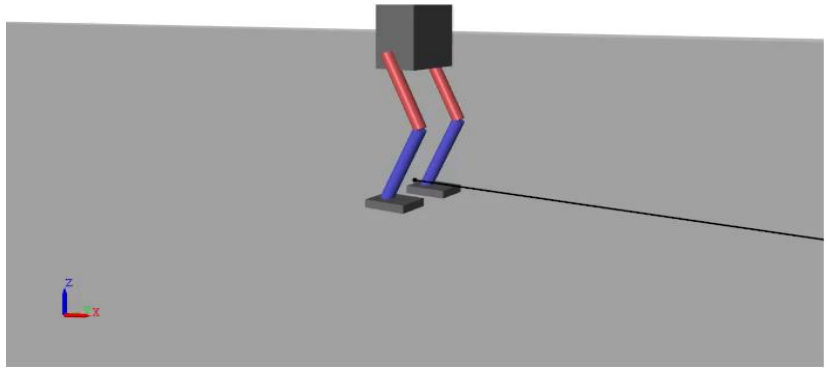


Steps in the Reinforcement Learning Workflow



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!



Train reinforcement learning networks for ADAS controllers

[Train Deep Deterministic Policy Gradient (DDPG) Agent for Adaptive Cruise Control]

- Create environment interface
- Create agent
- Train agent
- Simulate trained agent

Reinforcement Learning Toolbox™

R2019a