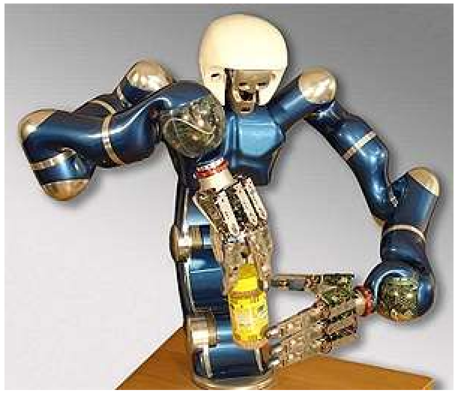
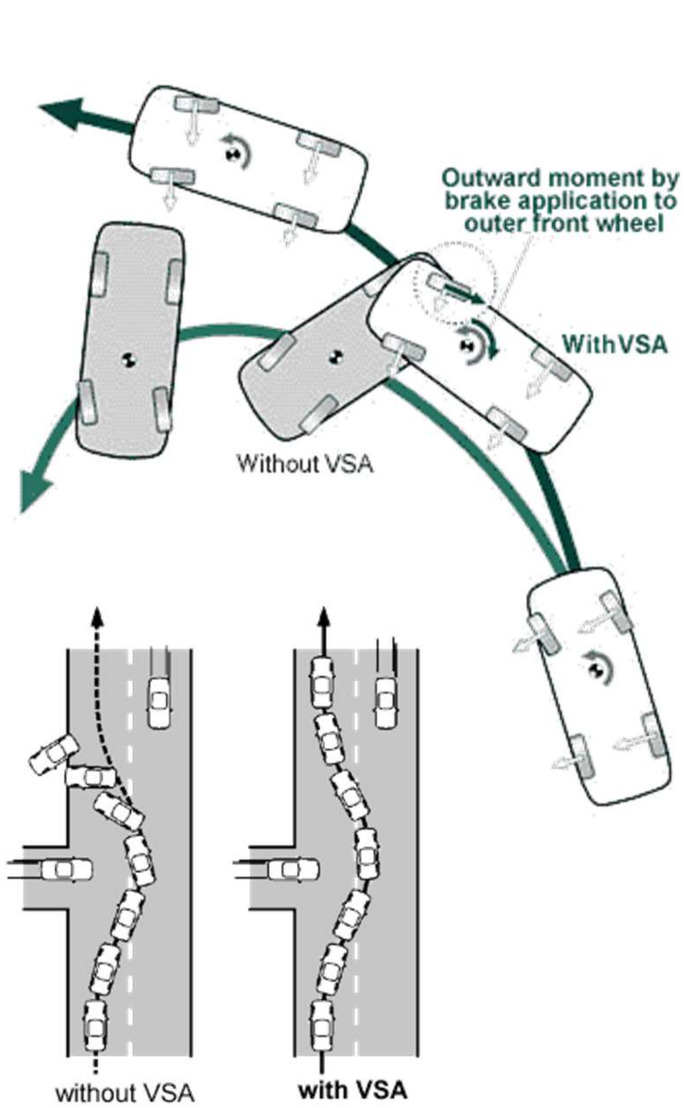
Control System Design for Automated Driving

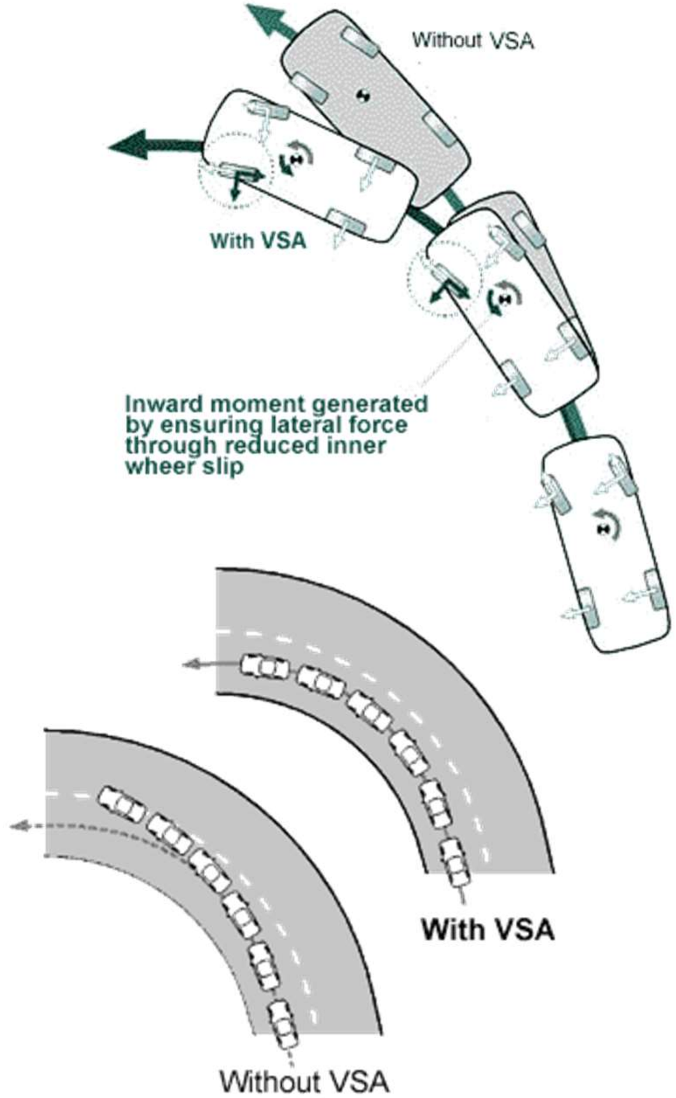
Lecture 08



ESP Control Example



- Oversteer Control



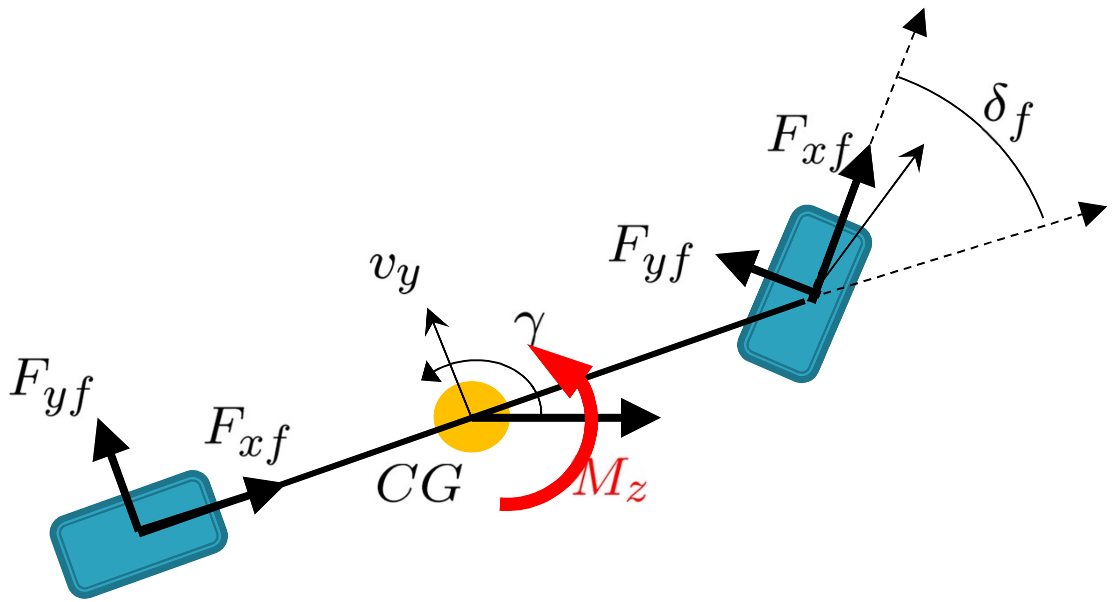
- Understeer Control

https://www.youtube.com/watch?v=LVz9f5WQhCI

Bicycle Model with Compensating Yaw Moment

$$\dot{v}_y = \frac{1}{m}(F_{yf} + F_{yr}) - v_x\gamma$$

$$\dot{\gamma} = \frac{1}{I_{zz}}(l_f F_{yf} - l_r F_{yr} + \boxed{M_z})$$



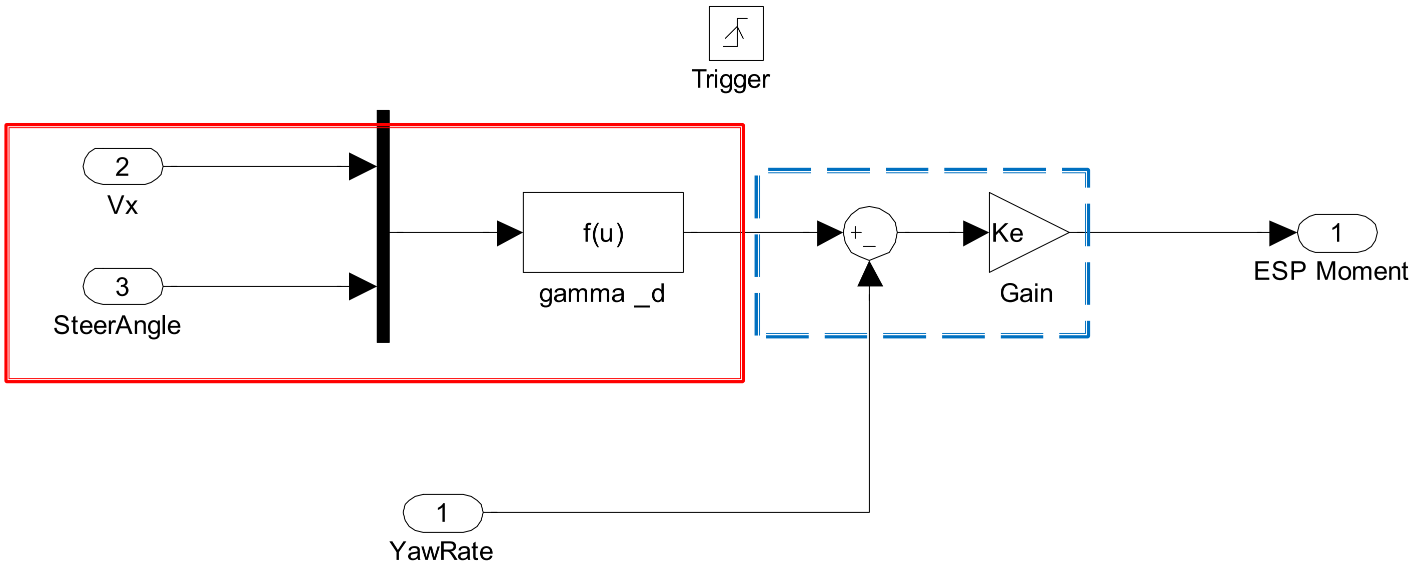
ESP Logic

- ESP logic
  - Desired Yaw Rate

$$\gamma_d = \frac{v_x}{l - \frac{m}{2l}\left(\frac{l_f C_f - l_r C_r}{C_f C_r}\right) v_x^2} \delta_f$$

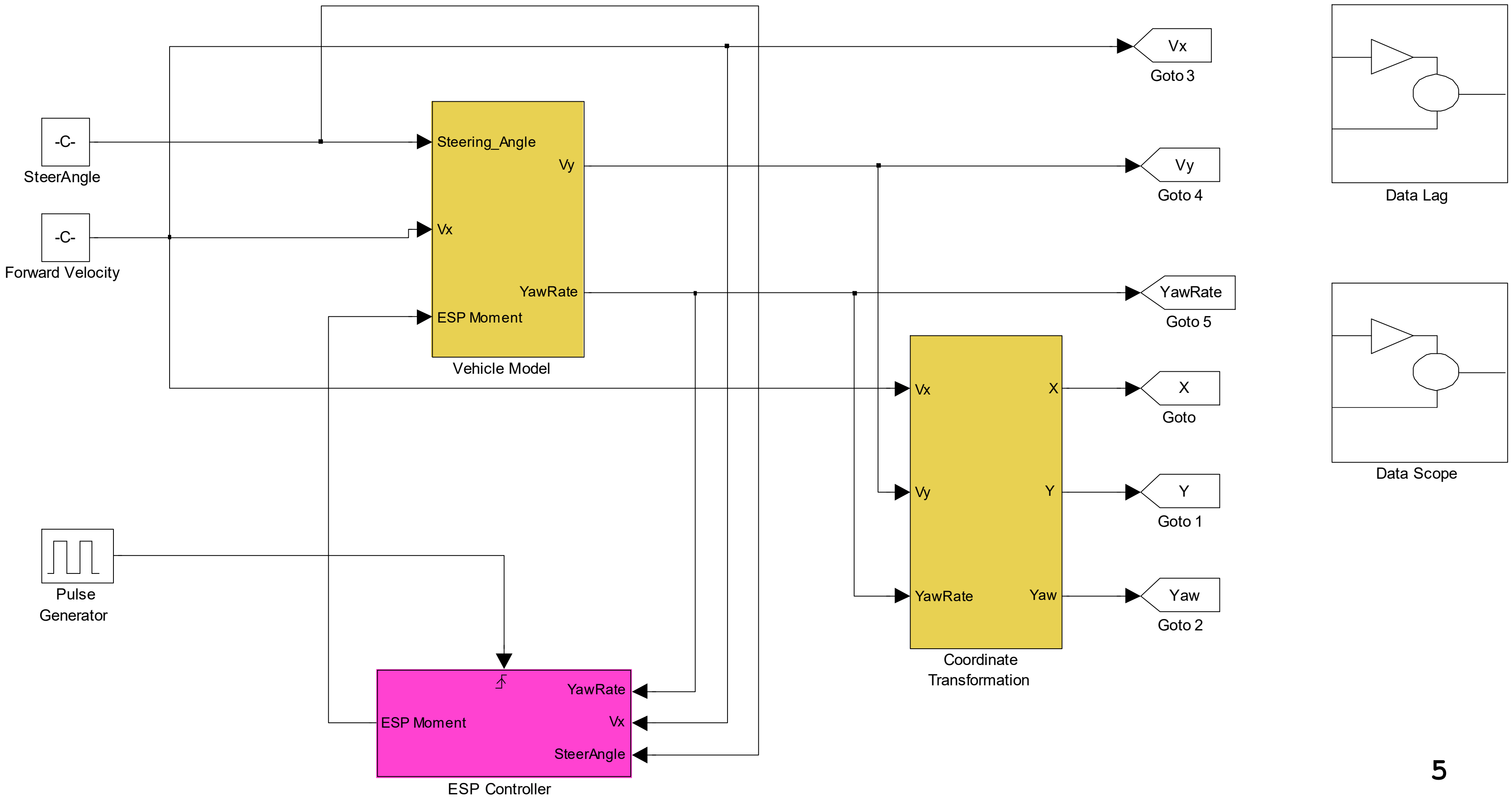
  - proportional control

$$M_z = K_e(\gamma_d - \gamma)$$



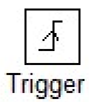
Vehicle Lateral Model with Controller

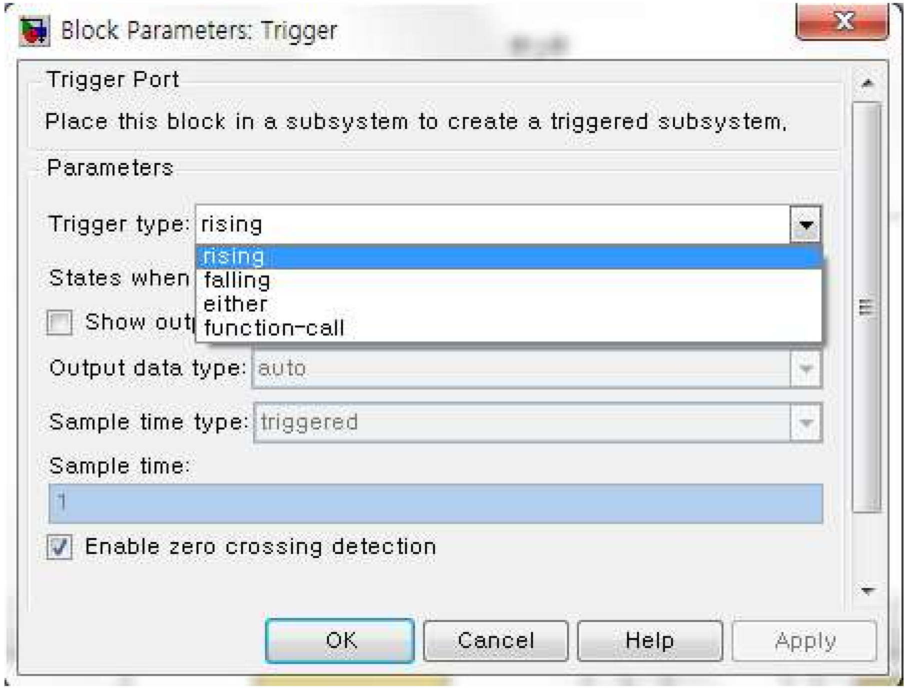
- Open vehicle_lateral_model_controller.mdl file.
- Additional "ESP Controller" block is attached on the vehicle model



Trigger Subsystem

- "Trigger" block is added in the "ESP controller" subsystem for more "realistic" simulation.
- Trigger block enables the subsystem only when the trigger condition is satisfied.
- After the system output is calculated, the same output is held until the next triggering event.





Run two simulations and compare results

- Comparison of the simulation results with and without ESP control.

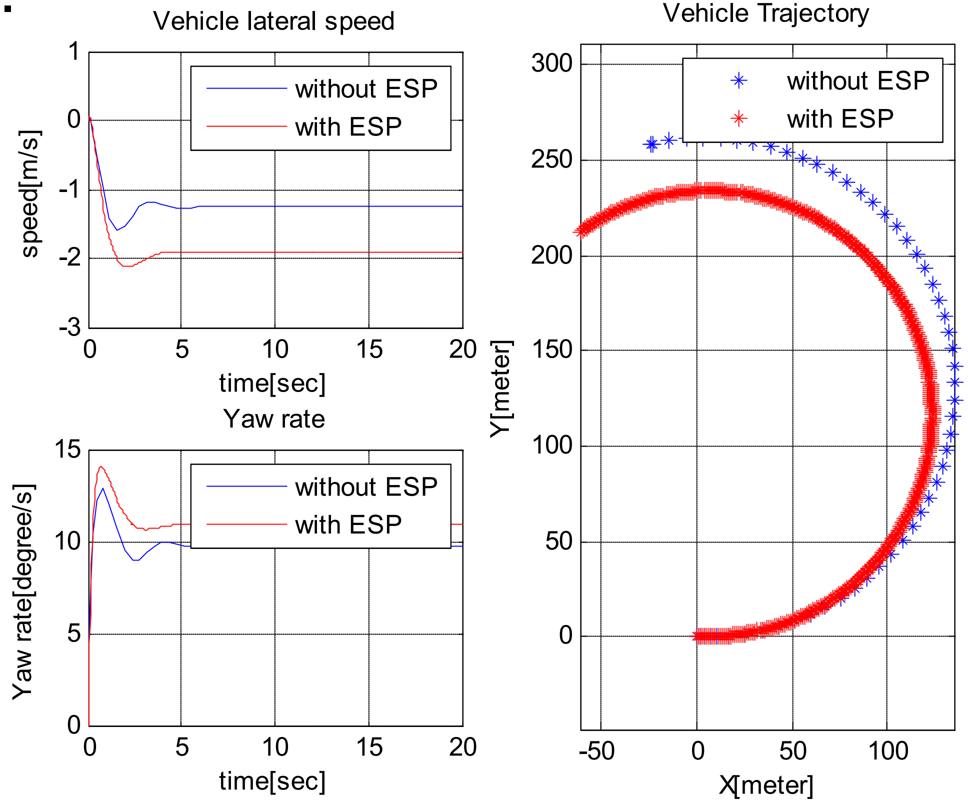
- However, the variable saved in the workspace will be erased after the second simulation starts.

- Use “sim” command to run simulation by Matlab command.’

- In “run_two_simulations.m” file
  - sim('vehicle_lateral_model',Tfinal);

Run two simulations and compare results

- Comparison of the simulation results.
- Data size is different due to addition of "Triggered Subsystem".
  - Continuous time system + Discrete time system
  - Variable-step solver ➔ Fixed step solver increases number of sampling time.



Trajectory Tracking Problem

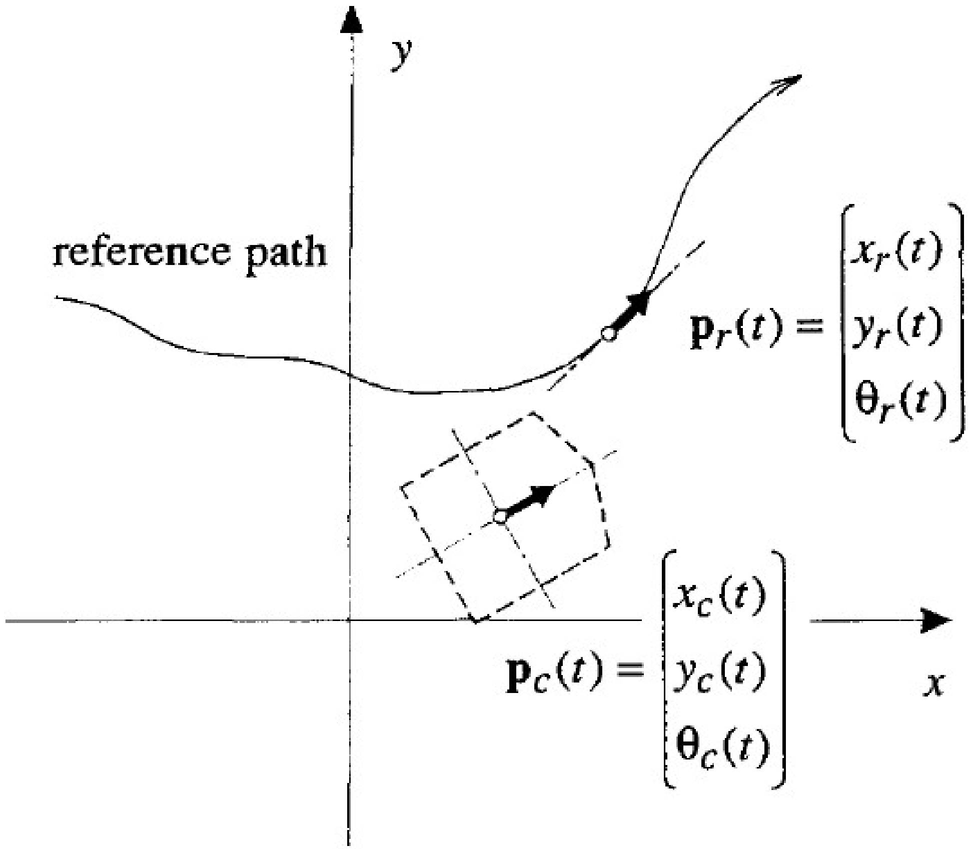


Fig. 1 **Reference and Current Postures**

제어목표

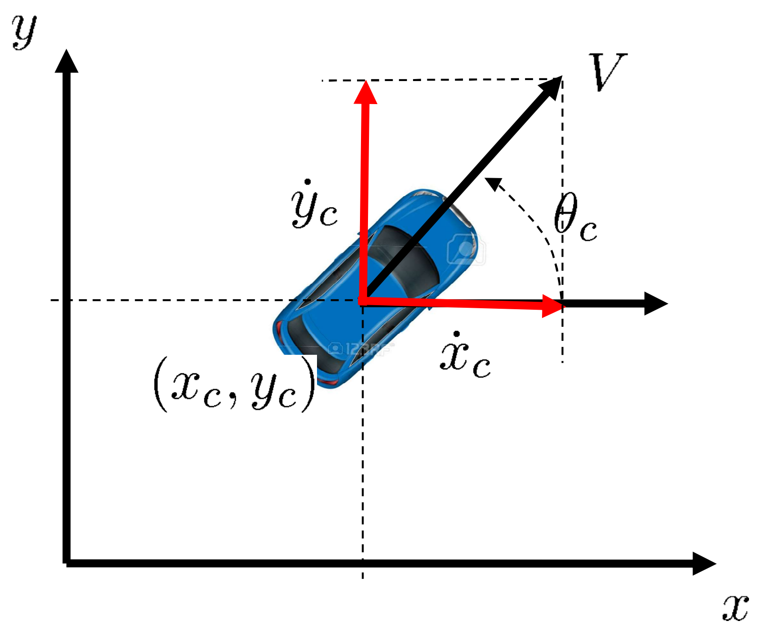
차량이 원하는 궤적(경로) $p_r(t)$를 추종하도록 차량속도와 조향각을 제어

Goal

- Kinematic Equation

$$\begin{aligned} \dot{x}_c &= V\cos(\theta_c) \\ \dot{y}_c &= V\sin(\theta_c) \\ \dot{\theta}_c &= \omega \end{aligned}$$

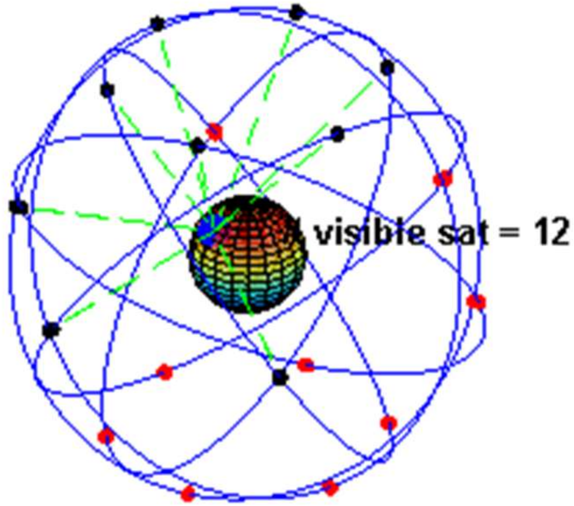
$$\begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta}_c \end{bmatrix} = \begin{bmatrix} \cos\theta_c & 0 \\ \sin\theta_c & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_c \\ w_c \end{bmatrix}$$



- Control Inputs

: Velocity (V), Turn rate (w)

DGPS Sensor



ProPak-V3

Position Accuracy	
Single Point L1	1.8 m CEP
Single Point L1/L2	1.5 m CEP
WAAS L1 only	1.2 m CEP
WAAS L1/L2	0.8 m CEP
DGPS	0.45 m CEP
CDGPS	0.7 m CEP
OmniSTAR	
VBS	1.0 m CEP
XP	0.15 m CEP
HP	0.10 m CEP
RT-20[2]	0.2 m CEP
RT-2	1 cm + 1ppm CEP

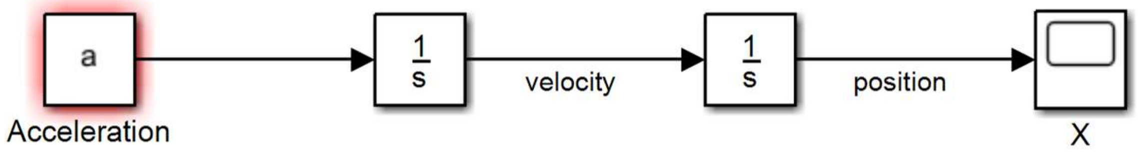
Physical & Electrical

Size	**185 x 160 x 71 mm**
Weight	**1.0 kg**
Power	
Input Voltage[9]	+9 to +18 VDC
Power Consumption	2.2 W (typical)[10]

circular error probable (CEP) (also **circular error probability** or **circle of equal probability**) is a measure of a weapon system's precision. It is defined as the radius of a circle, centered about the mean, whose boundary is expected to include the landing points of 50% of the rounds.

- From Wikipedia.org -

Inertial Navigation Sensor (INS)



Example)

a = 0.1 m/s² ➔ D = 0.5*a*t² ➔ 1분후 오차 180m !!

Output

3D orientation (Quaternions/Matrix/Euler angles)
3D acceleration
3D rate-of-turn
3D earth-magnetic field (normalized)
Temperature

Orientation performance

Dynamic Range:	all angles in 3D
Angular Resolution[1]:	0.05 deg
Static Accuracy (Roll/Pitch):	<0.5 deg
Static Accuracy[2] (Heading):	<1 deg
Dynamic Accuracy[3]:	2 deg RMS

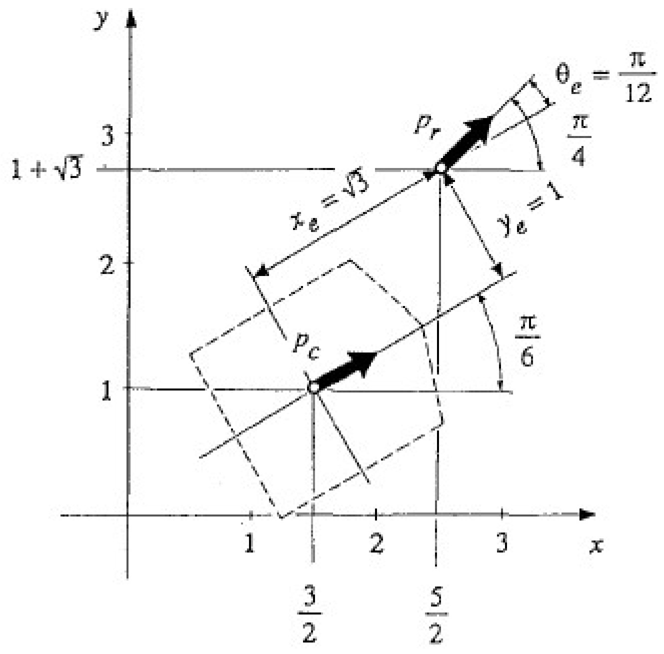
Sensor performance

	rate of turn	acceleration	magnetic field	temperature
Dimensions	3 axes	3 axes	3 axes	-
Full Scale (standard)	± 300 deg/s	± 17 m/s²	± 750 mGauss	-55...+125 °C
Linearity	0.1% of FS	0.2% of FS	0.2% of FS	<1% of FS
Bias stability[4] (1σ)	5 deg/s	0.02 m/s²	0.5 mGauss	0.5 °C accuracy
Scale Factor stability[4] (1σ)	-	0.05%	0.5%	-
Noise density	0.1 deg/s/√Hz	0.001 m/s²/√Hz	0.5 mGauss (1σ)	-
Alignment error	0.1 deg	0.1 deg	0.1 deg	-
Bandwidth (standard)	40 Hz	30 Hz	10 Hz	-

- Error posture Definition

$$\mathbf{p}_e = \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos\theta_c & \sin\theta_c & 0 \\ -\sin\theta_c & \cos\theta_c & 0 \\ 0 & 0 & 1 \end{bmatrix} (\mathbf{p}_r - \mathbf{p}_c) \equiv T_e(\mathbf{p}_r - \mathbf{p}_c)$$

  - Xe : 종방향 오차
  - Ye : 횡방향 오차
  - θ_e : 각도 오차



- Control Input Calculation
  - Control Inputs

$$\mathbf{q} = \begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} v(\mathbf{p}_e, \mathbf{q}_r) \\ \omega(\mathbf{p}_e, \mathbf{q}_r) \end{pmatrix} = \begin{pmatrix} v_r \cos\theta_e + \boxed{K_x x_e} \\ \omega_r + v_r \boxed{(K_y y_e + K_\theta \sin\theta_e)} \end{pmatrix}$$

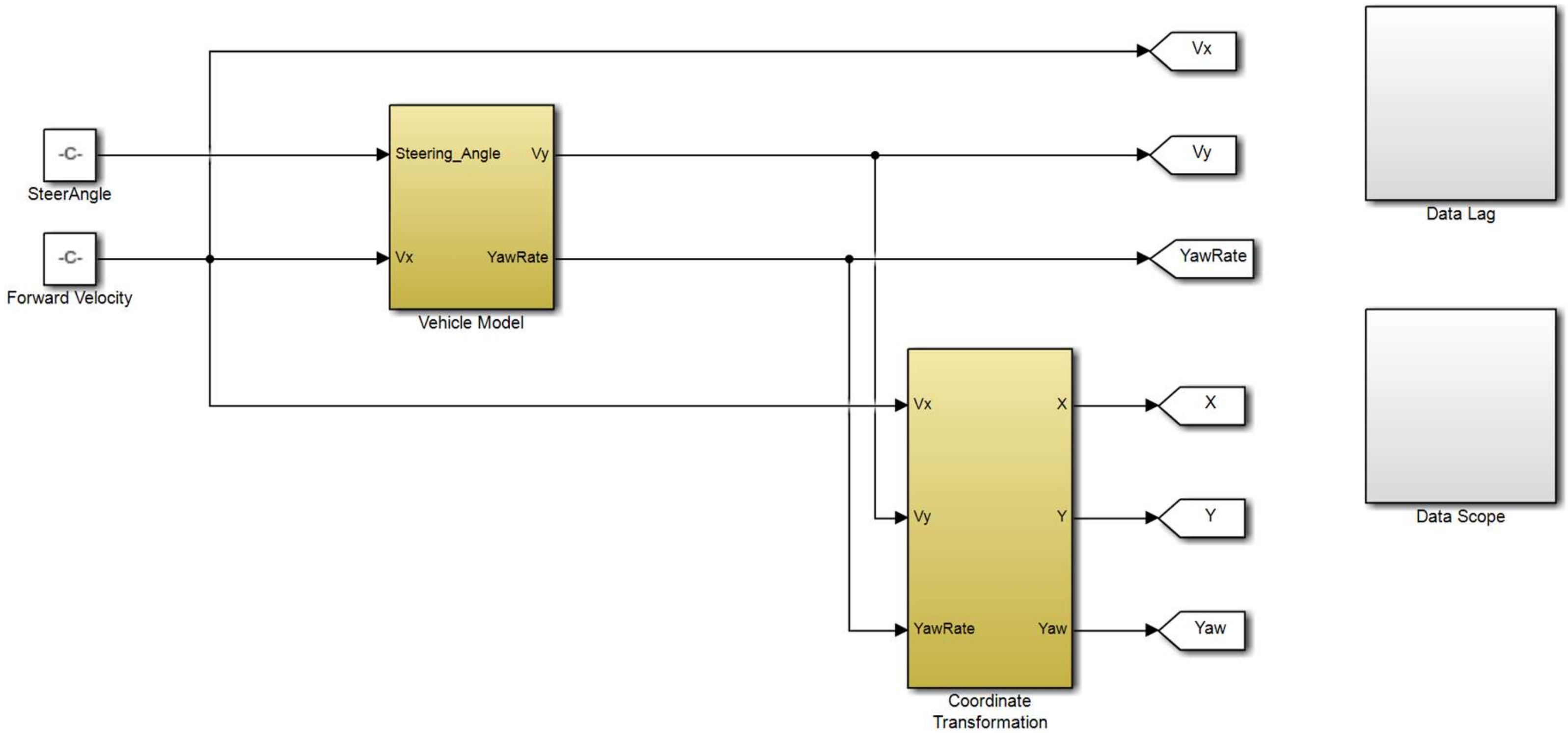
  - Vr : reference velocities
  - Wr : reference angular velocities
  - If you don't have Vr or Wr, let

    Vr = Wr = 0

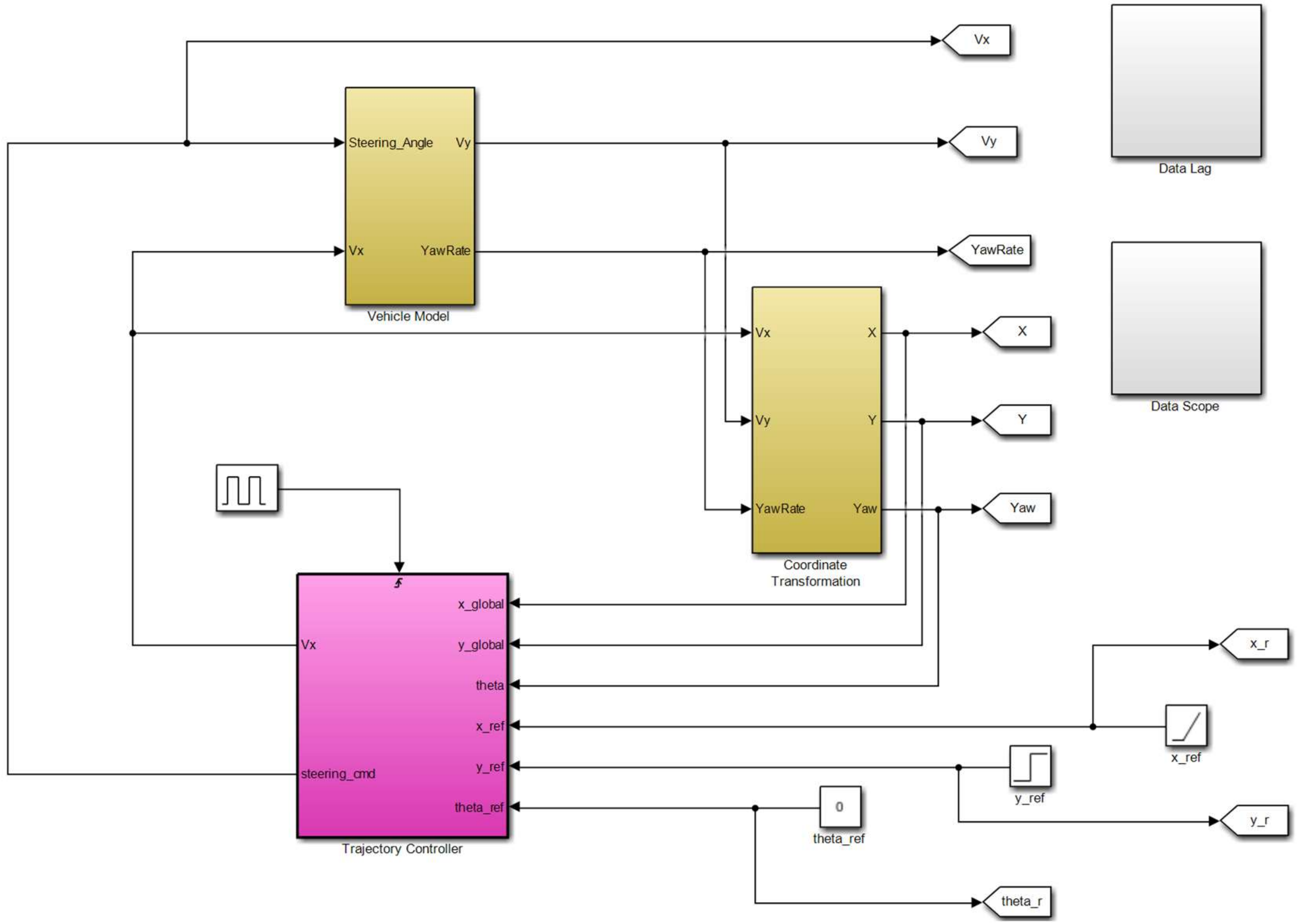
  and determine the gain by trial and error.

Plant Model

- Simulink Model of Vehicle Kinematics
  - Vehicle Lateral Dynamics



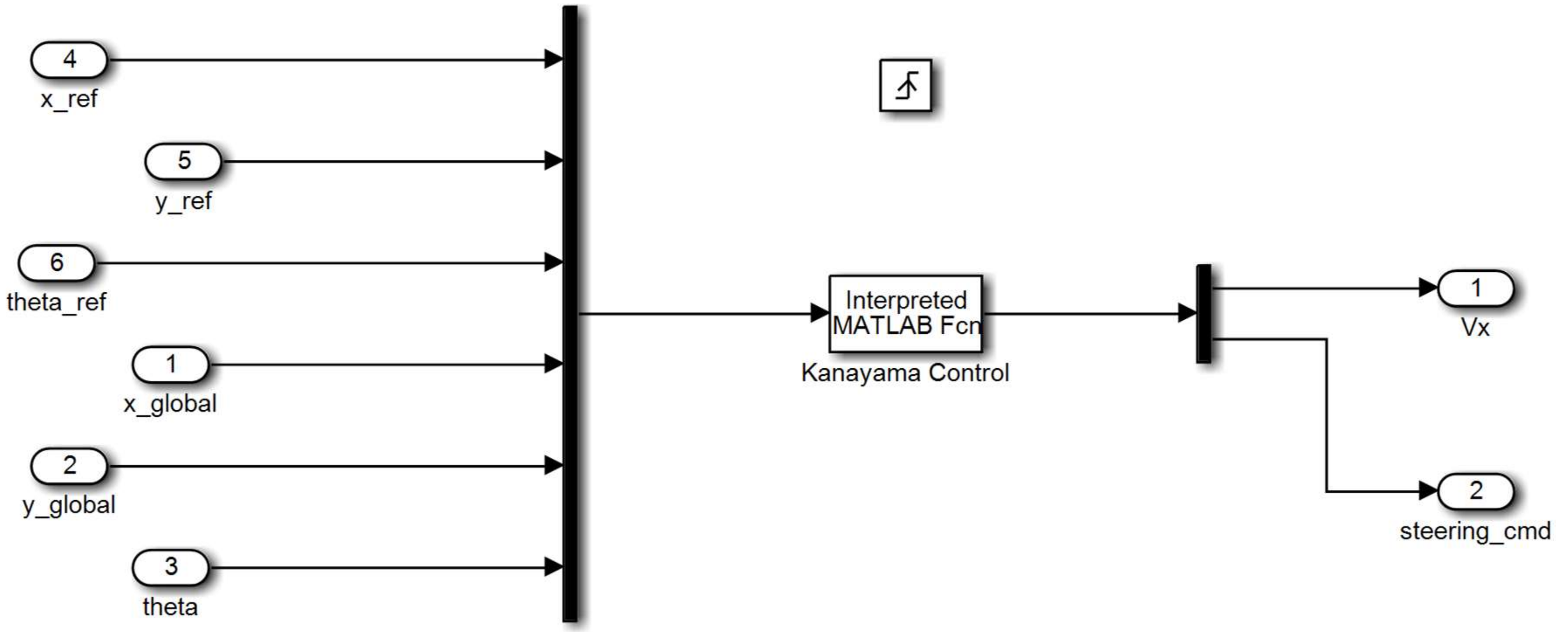
- Vehicle Model with Trajectory Controller



Plant Model with Controller

- Controller (Vr and Wr are ignored for simplicity)

$$\mathbf{q} = \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} v(\mathbf{p}_e, \mathbf{q}_r) \\ \omega(\mathbf{p}_e, \mathbf{q}_r) \end{bmatrix} = \begin{bmatrix} \cancel{v_r \cos\theta_e} + K_x x_e \\ \cancel{\omega_r} + \cancel{v_r}(K_y y_e + K_\theta \sin\theta_e) \end{bmatrix}$$



Interpreted Matlab Function Block

- User-defined Functions Library

- Applies the specified Matlab function or expression to the input.

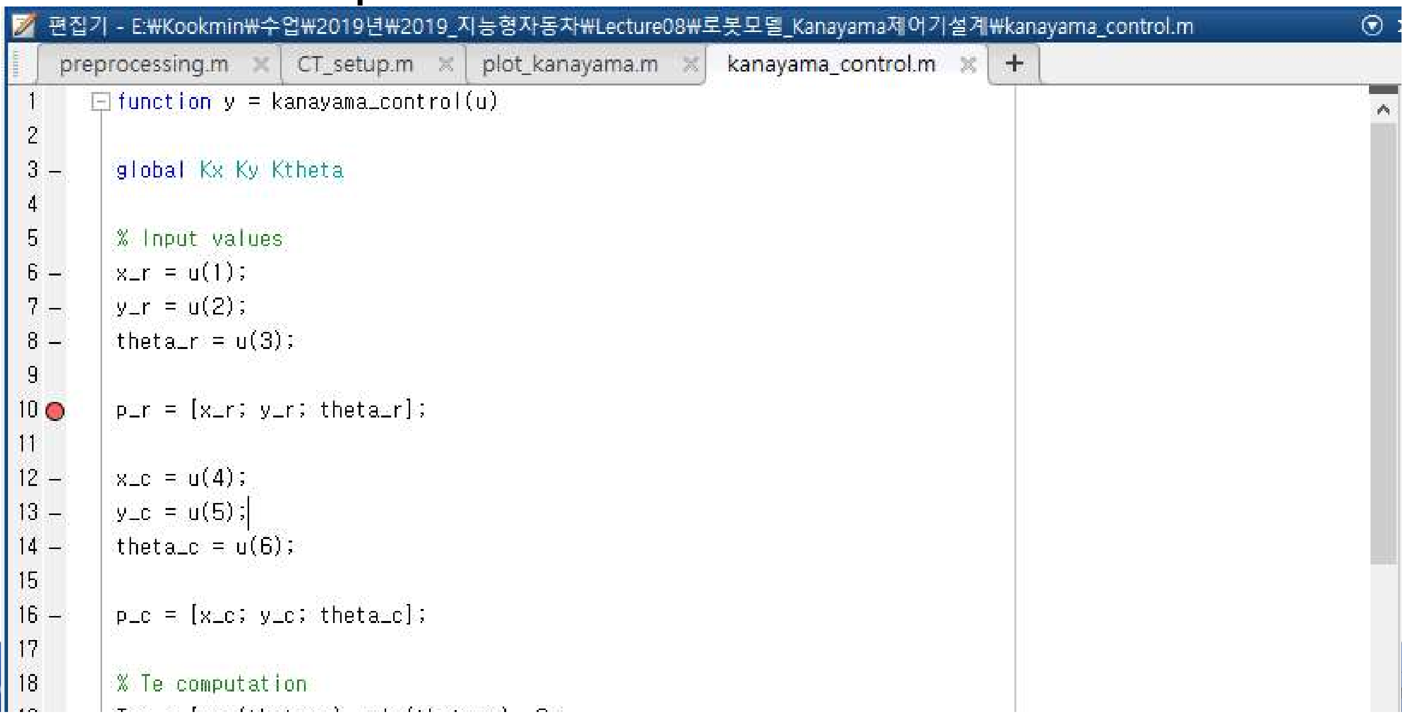
- The output of the function must match the output dimensions of the block

- In particular, when you want to implement a controller that is too complicated to calculate using atomic Simulink blocks.

- For Automatic C code generation, use Matlab Function block instead. Matlab Function block requires more strict programming rules for code generation in general.

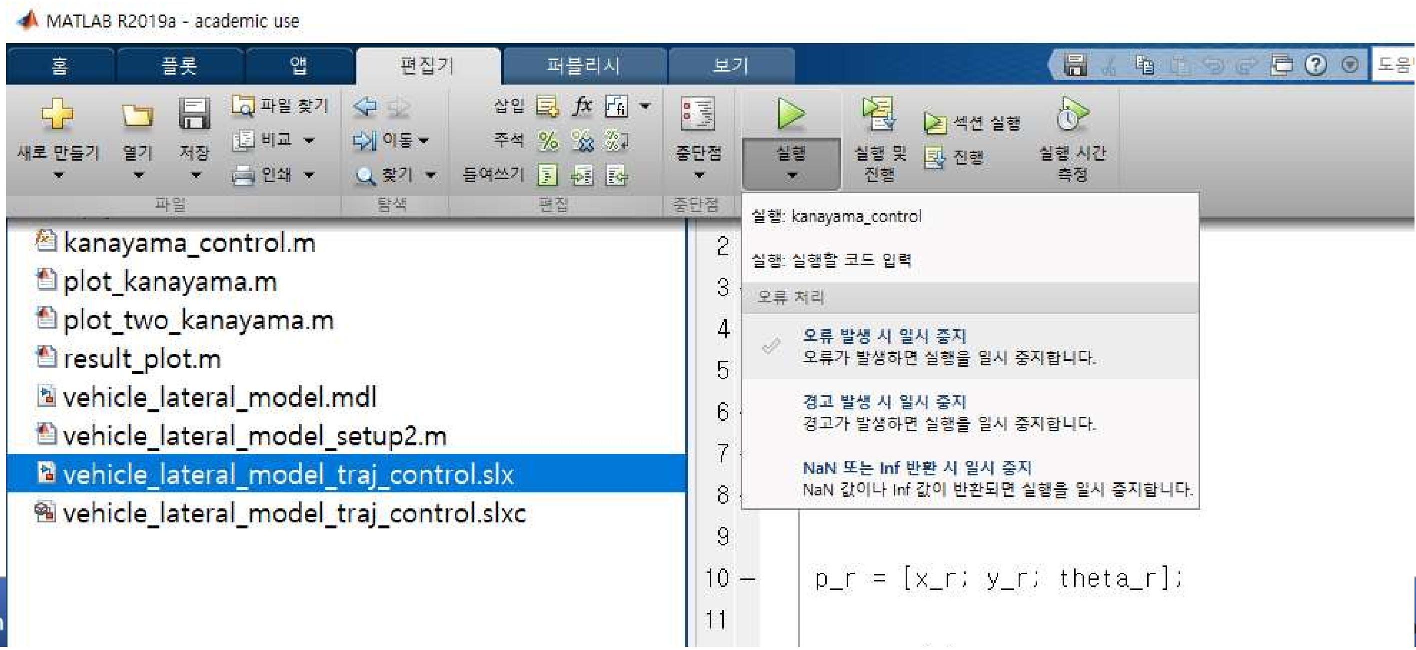
- However, in this example, interpreted Matlab function block is used to employ global variable Kx, Ky, Ktheta.

- Debugging Tool
  - break point
    - Click next to the line number, where you would like to stop the program during the simulation.
    - Run the simulation, then the simulation will stop at the break point line



- Debugging Tool
  - break point
    - Press F10 to progress one line
    - Press F11 to step into the function
    - Press F5 to continue
    - Press "디버그중지" to stop the code
    - More break points can be added after the stop.

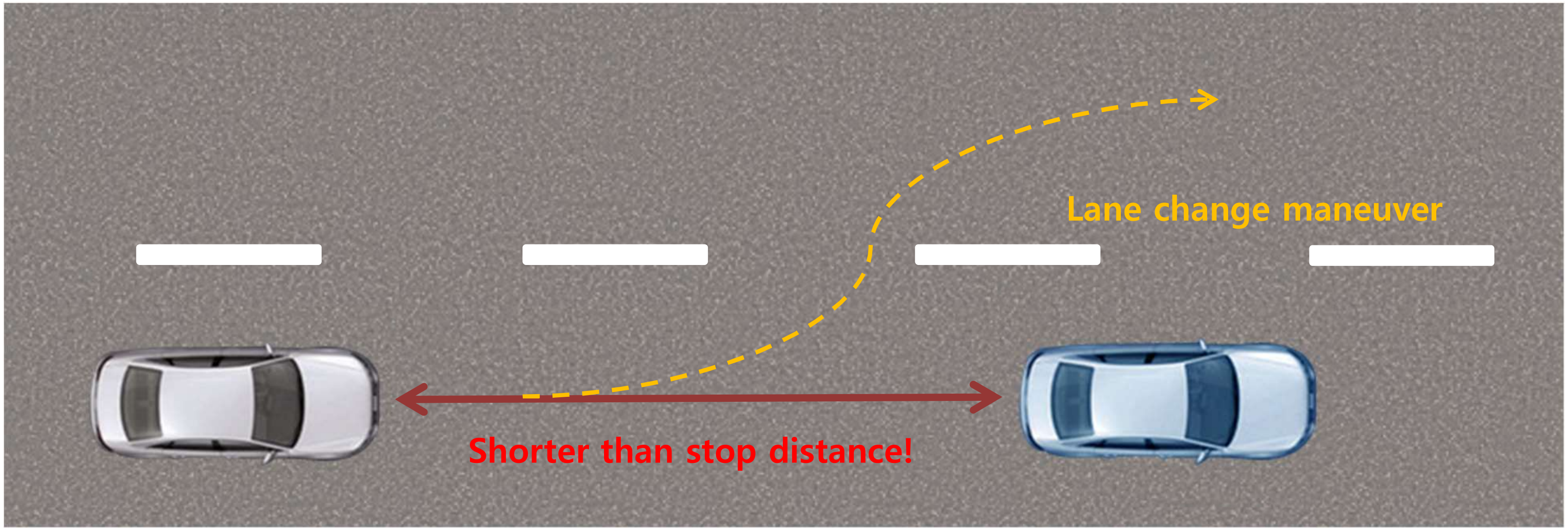
- Debugging Tool
  - "오류발생시 일시중지" Option
    - Stops running the code at the line where the error occurs.
  - "경고 발생시 일시중지" Option
  - "Nan 또는 Inf 반환시 일시중지" Option



Simulation Scenario

<Autonomous Emergency Steering (AES) System>

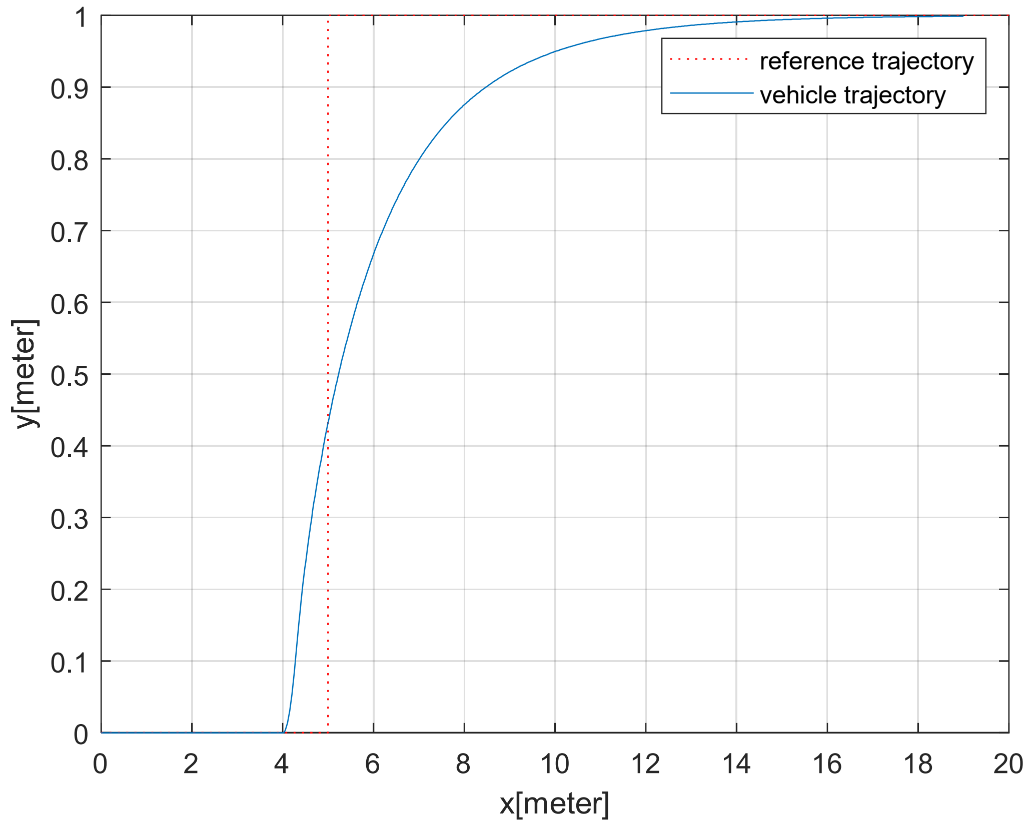
- **Avoiding Rear-End collision that AEB cannot avoid.**
- **During the scenario, the desired lane switches to the next lane when the AES is activated.**
- **Control wheel steering angle & forward velocity.**
- **Assumption : The vehicle can identify the location of the nearby obstacles and make decision that switching to the next lane is the safest way to avoid collision.**



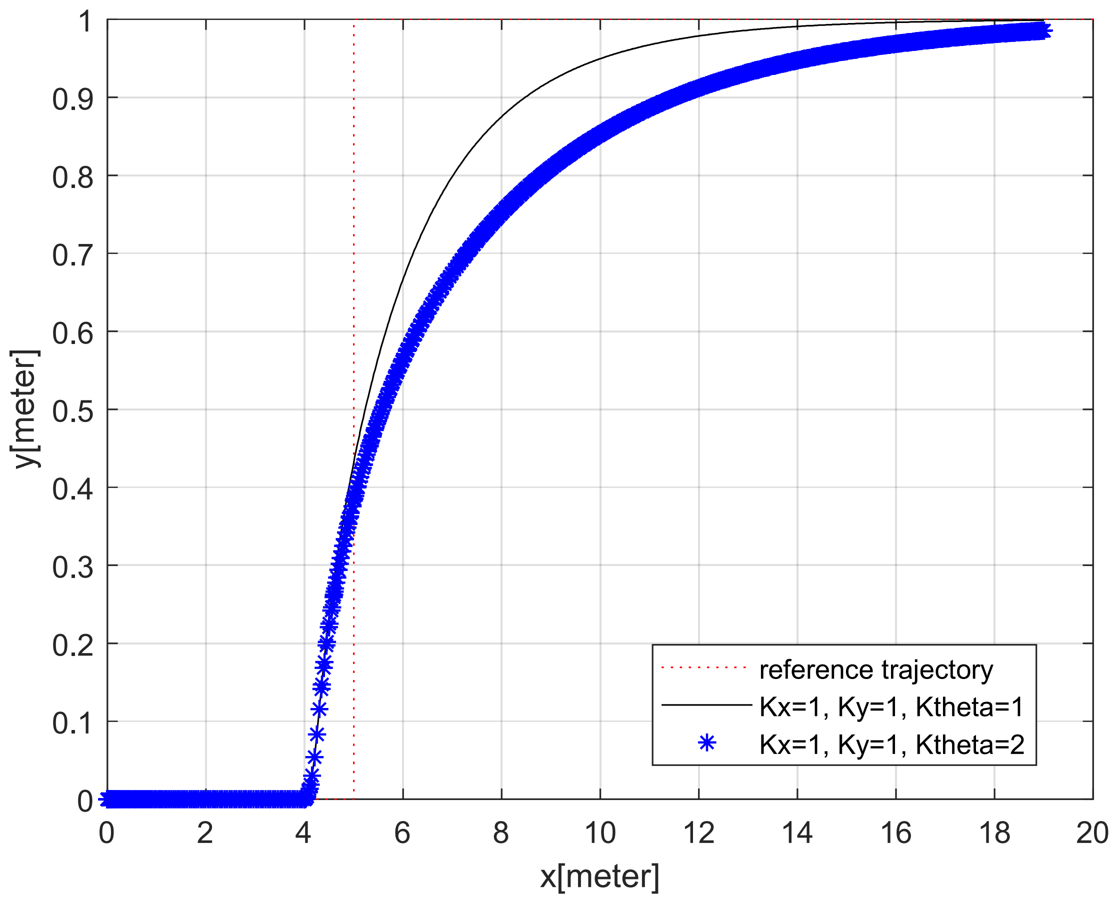
<Emergency lane change maneuver>

Simulation Result

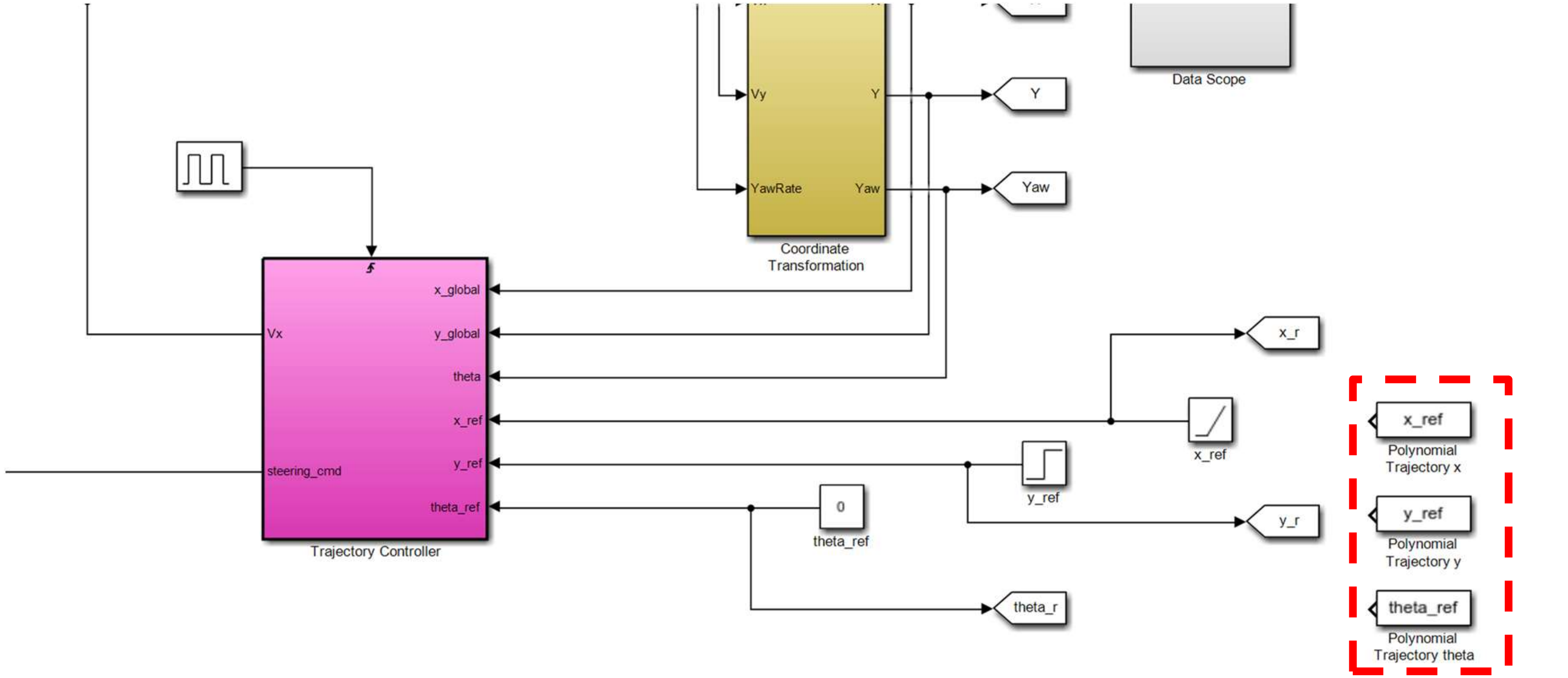
- Simple Implementation with Kx = Ky = Ktheta = 1.



- Run with different K gains to compare results



- Use From Workspace blocks to generate a reference x, y, theta trajectory for Kanayama Controller .



- Structure Format Data Generation Example
  - Reference Trajectory Definition

$$Y = 0.01 \times X(X-5)(X-10)$$

```
% Reference Trajectory Generation

time = 0:0.1:t_final;

x_ref.time = time';
y_ref.time = time';
theta_ref.time = time';

x_ref.signals.values = time';
y_ref.signals.values = [0.01*time.*(time-5).*(time-10)]';
dx = diff(x_ref.signals.values);
dy = diff(y_ref.signals.values);
theta_ref.signals.values = atan2(dy, dx);
theta_ref.signals.values = [theta_ref.signals.values; theta_ref.signals.values(length(time)-1)];
```

- Simulation Result