EE405A
Vehicle Control

(TA) Hyunki Seong
School of Electrical Engineering
KAIST

April 2, 2021

hynkis@kaist.ac.kr

Experiment Objectives

In this week, you will do the following:

- Understand Vehicle Model (Kinematic)
- Learn how to design Vehicle Control
  - Longitudinal Control (PID Control)
  - Lateral Control (Pure Pursuit, Stanley Method)
- Programming Assignment :
  - Design your path following, speed controller.
  - Reference codes (simulator, controller) will be provided.

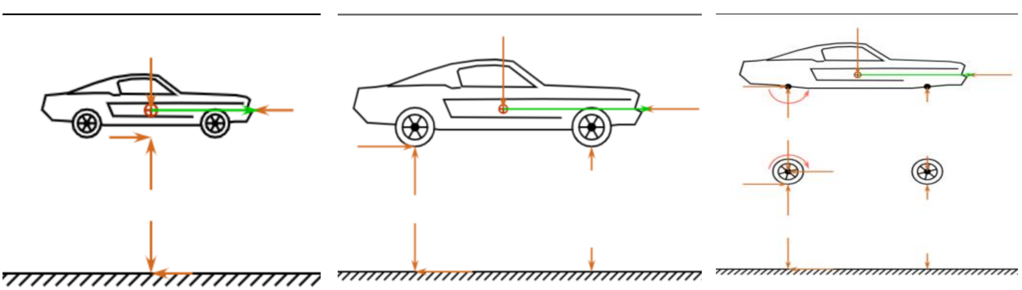
Vehicle Model

Vehicle Model

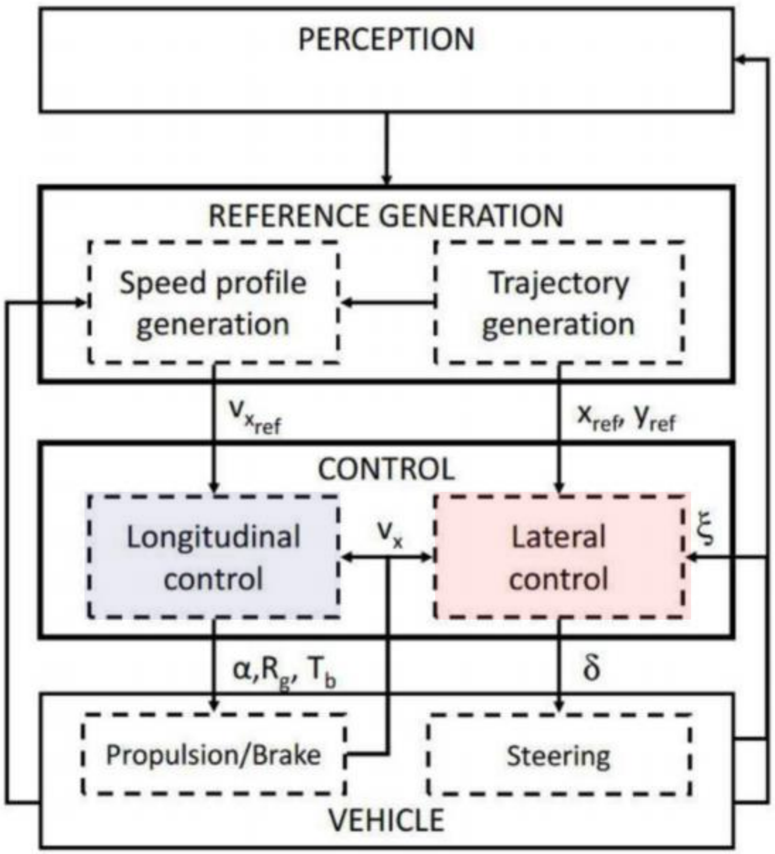
- **Autonomous vehicle controls**
  - ❑ Longitudinal control

    : Speed control with acceleration and braking
  - ❑ Lateral control

    : Steering wheel or angle of tires control
- **Autonomous vehicle controls**
  - ❑ Point mass model
  - ❑ 2D rigid body model
  - ❑ 2D multi-body model



Point mass model | 2D rigid body model | 2D multi-body model

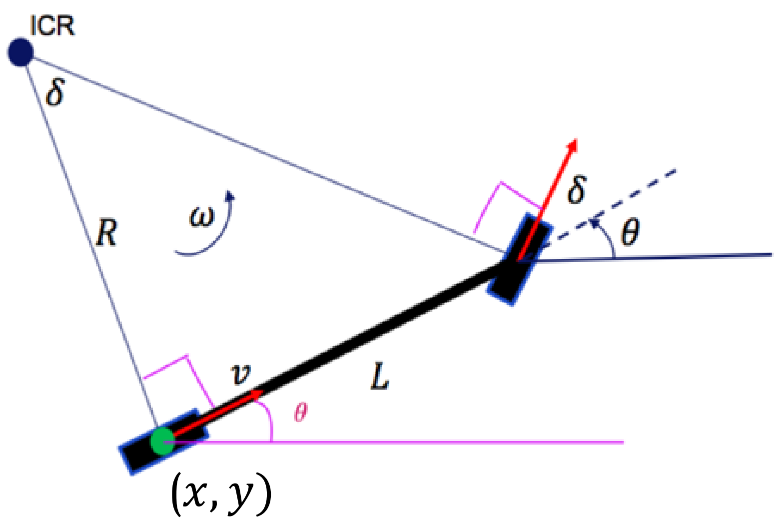


Block diagram for a vehicle control system

Vehicle Model (Kinematic)

Bicycle model (Rear axle centered)

- A car is assumed to drive in a circle with a fixed steering angle. No slip to the sides.
- From observation, we get the equation describing the relationship between steering angle δ and the corresponding turning radius R, given its wheelbase length L.



✓ Velocity

$$\dot{x} = \mathrm{v}\cos\psi$$
$$\dot{y} = \mathrm{v}\sin\psi$$

✓ Acceleration

$$\dot{v} = a$$

✓ Instantaneous Center of Rotation (ICR)

$$\tan\delta = \frac{L}{R} \qquad v = R\omega = R\dot{\psi}$$

$$\dot{\psi} = \frac{v}{L}\tan\delta$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos\psi \\ v\sin\psi \\ \frac{v}{L}\tan\delta \\ a \end{bmatrix}$$

State : $\{x, y, \psi, v\}$
Control input : $\{\delta, a\}$

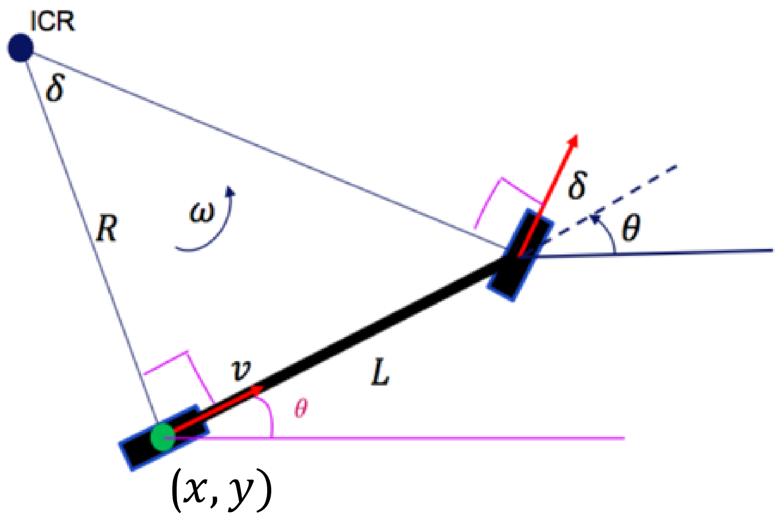
x : position x
y : position y
ψ : yaw angle
v : velocity

δ : steering angle
a : acceleration

Vehicle Model (Kinematic)

Bicycle model (Rear axle centered)

- A car is assumed to drive in a circle with a fixed steering angle. No slip to the sides.
- From observation, we get the equation describing the relationship between steering angle δ and the corresponding turning radius R, given its wheelbase length L.



✓ Velocity

$$\dot{x} = \mathrm{v}\cos\psi$$
$$\dot{y} = \mathrm{v}\sin\psi$$

✓ Acceleration

$$\dot{v} = a$$

✓ Instantaneous Center of Rotation (ICR)

$$\tan\delta = \frac{L}{R} \qquad v = R\omega = R\dot{\psi}$$

$$\Rightarrow \quad \dot{\psi} = \frac{v}{L}\tan\delta$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v} \end{bmatrix}$$

Discrete model in code

x_(t+1) = x_t + x_dot * dt
y_(t+1) = y_t + y_dot * dt
ψ_(t+1) = ψ_t + ψ_dot * dt
v_(t+1) = v_t + v_dot * dt

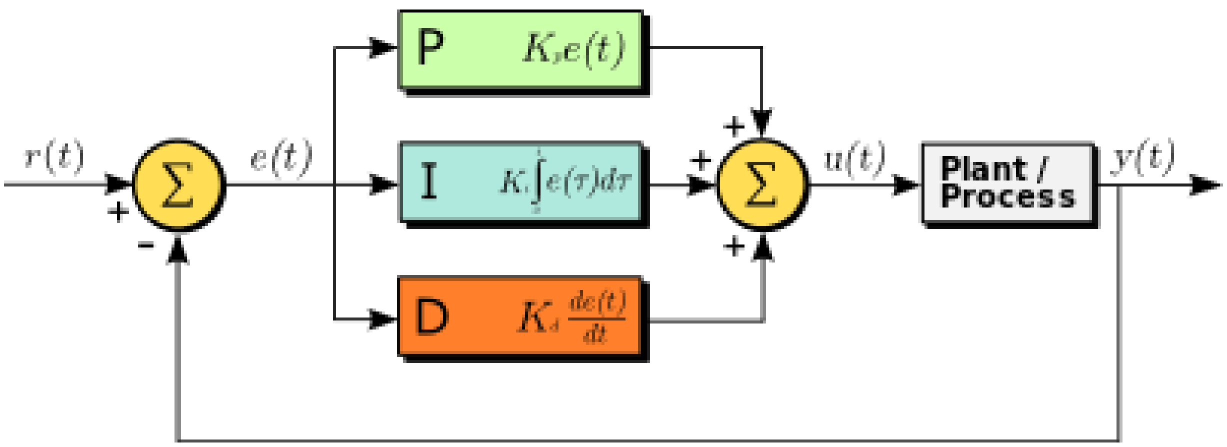
x : position x
y : position y
ψ : yaw angle
v : velocity
δ : steering angle
a : acceleration

Vehicle Control

Vehicle Control (PID Controller)

➢ Proportional-Integral-Derivative Controller (PID Controller)

- ❑ PID Controller consists of **three terms**: proportional(P), integral(I) and derivative(D) term.
- ❑ Each term has a control gain: K_P gain, K_I gain, K_D gain.
- ❑ **P-term** is proportional to the error, $r(t) - y(t)$.
- ❑ **I-term** accounts for past error values and integrates them over time.
- ❑ **D-term** estimates the future trend of the error, based on its current rate of change.



A block diagram of a PID controller in a feedback loop.

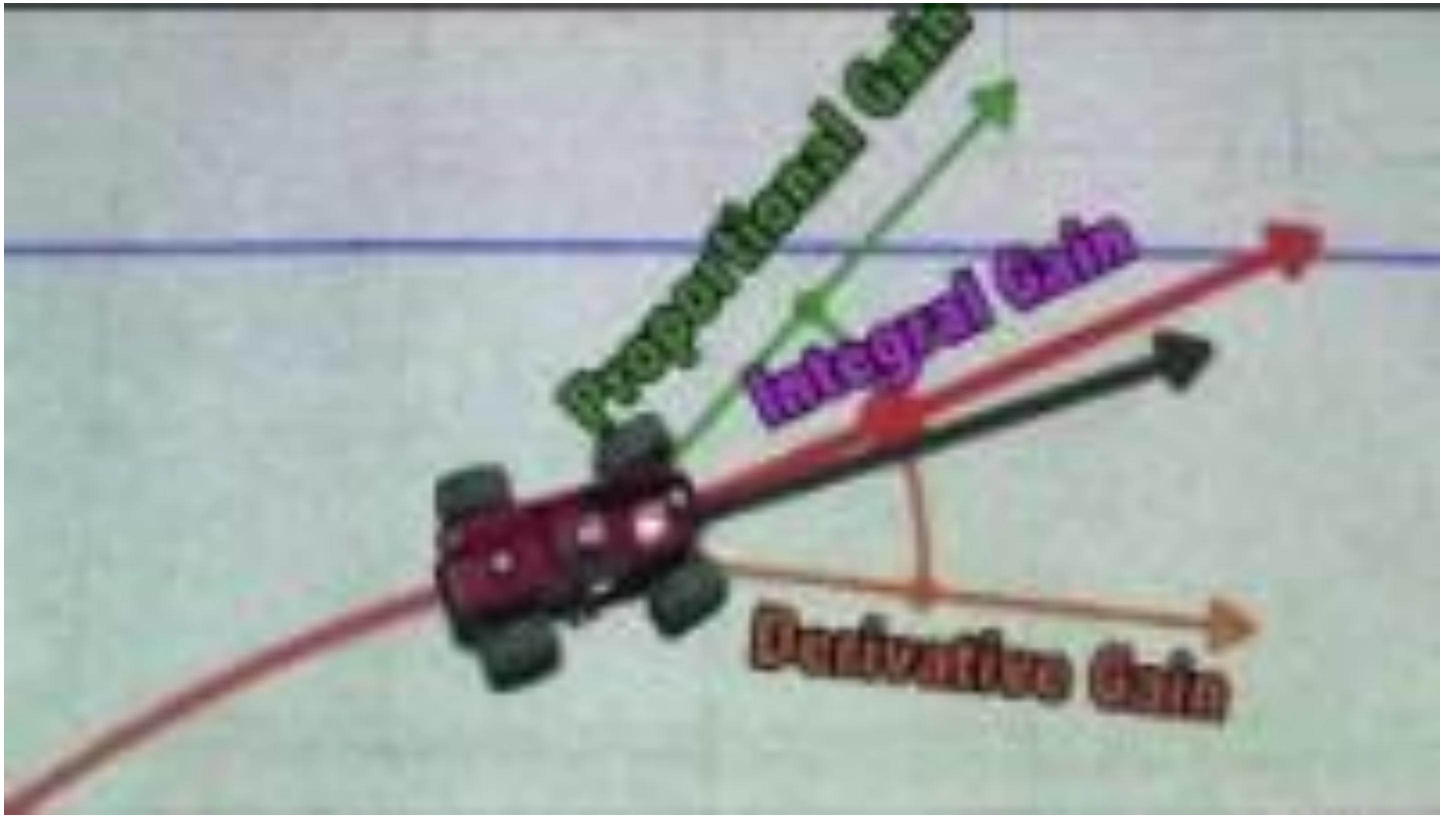
Vehicle Control (PID Controller)

- **Proportional-Integral-Derivative Controller (PID Controller)**
  - A brief introduction of PID Control

Reference : https://www.youtube.com/watch?v=UR0hOmjaHp0

Vehicle Control (PID Controller)

- **Proportional-Integral-Derivative Controller (PID Controller)**
  - How P, I, D gains affect the performance of the vehicle

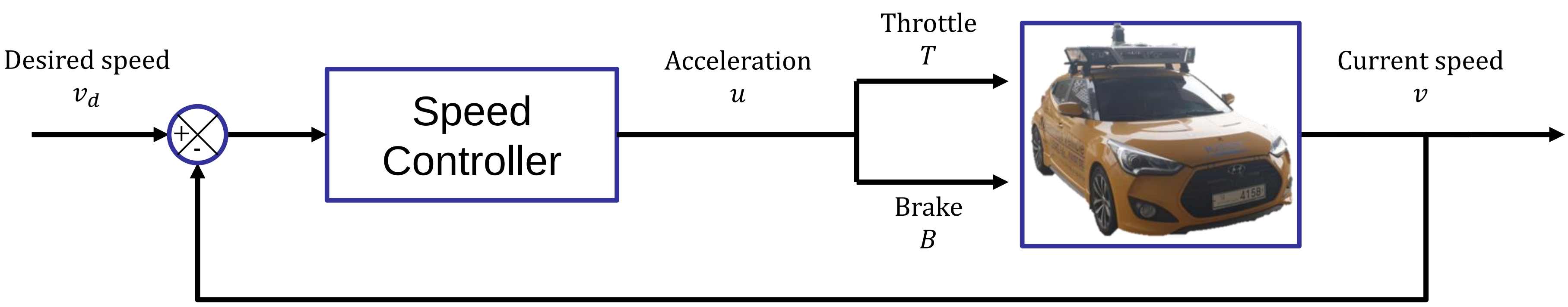


Reference : https://www.youtube.com/watch?v=4Y7zG48uHRo

Vehicle Control (Longitudinal)

Longitudinal Control

- For maintaining the desired speed of a vehicle, **a longitudinal(speed) controller** should be designed.
- A feedback control system is used to minimize an error between **current** and **desired** speed.
- The control value u is mapped to throttle T or brake B pedal position.



- **PID controller** can be used for the speed control.

$$u = \underbrace{K_P(v_d - v)}_{\text{Proportional Term}} + \underbrace{K_I \int_0^t (v_d - v)dt}_{\text{Integral Term}} + \underbrace{K_D \frac{d(v_d - v)}{dt}}_{\text{Derivative Term}}$$

Vehicle Control (Lateral)

➢ Geometry for Lateral Control

Vehicle states

(x_r, y_r, ψ_r) : x, y, yaw of the ego vehicle's reference point

δ : Steering angle

The reference point can be whether:
- Rear/Front axle
- Center point (Center of Gravity, CoG)

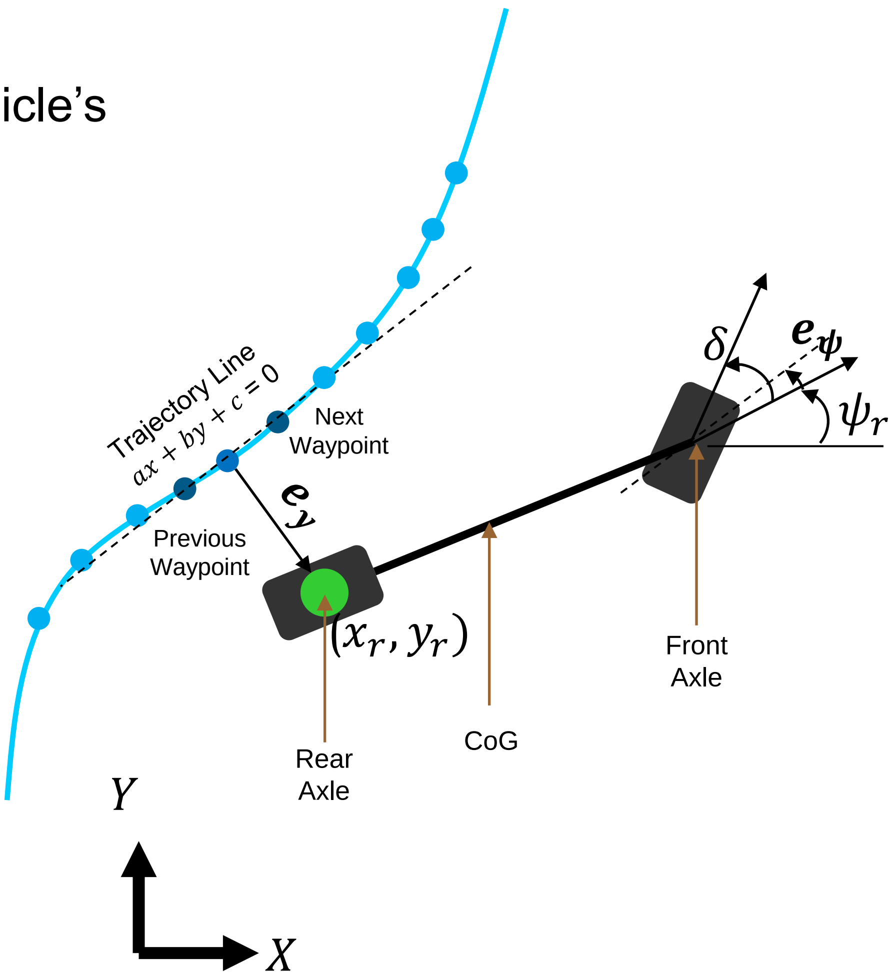
Cross track error

$$e_y = \frac{ax_c + by_c + c}{\sqrt{a^2 + b^2}}$$

Or the distance between the ego and **closest waypoint**.

Heading error

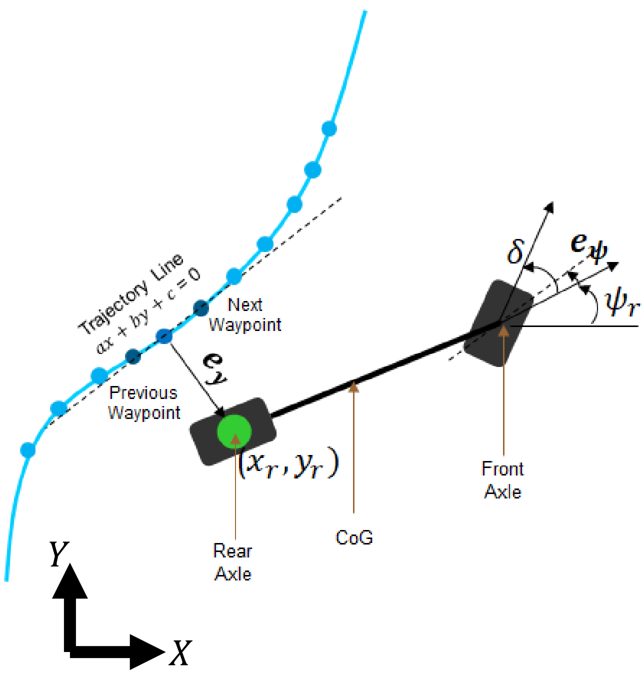
$$e_\psi = \tan^{-1}\left(\frac{-a}{b}\right) - \psi_r$$



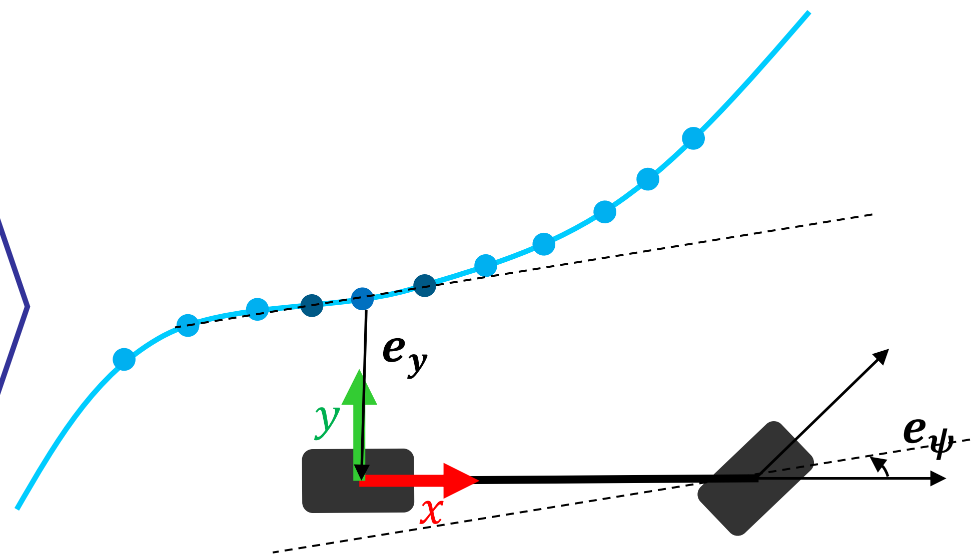
Vehicle Control (Lateral)

➢ **Geometry for Lateral Control**

Coordinate transformation from global to local frame



Global coordinate

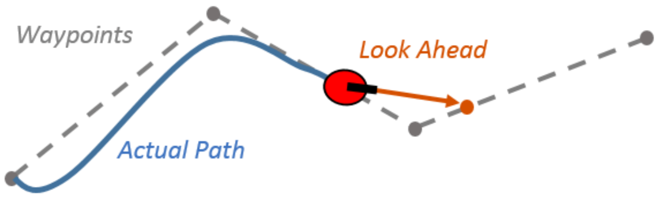


Local coordinate

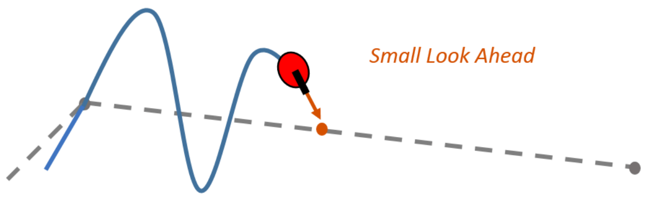
Vehicle Control (Lateral)

➢ Look Ahead Distance

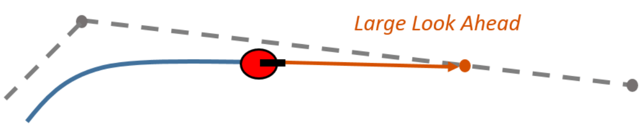
- ❑ Look ahead distance is one of the main tuning parameters for the controller.
- ❑ The look ahead distance is how far along the path the robot should look to compute control commands.



- ❑ The effect of changing the distance can change how the robot tracks the path.
- ❑ Usually, closer distance during slow speed; farther distance during fast speed, for stability.



- Fast recovery
- Large oscillation



- Slow recovery
- Small oscillation

Reference : https://kr.mathworks.com/help/nav/ug/pure-pursuit-controller.html

Vehicle Control (Lateral)

➢ Pure pursuit method

: The pure pursuit method consists of geometrically calculating the curvature of a circular arc that connects the rear axle location to a goal point on the path ahead of the vehicle. The goal point is determined from **a look-ahead distance** from the current rear axle position of the vehicle to the desired path.

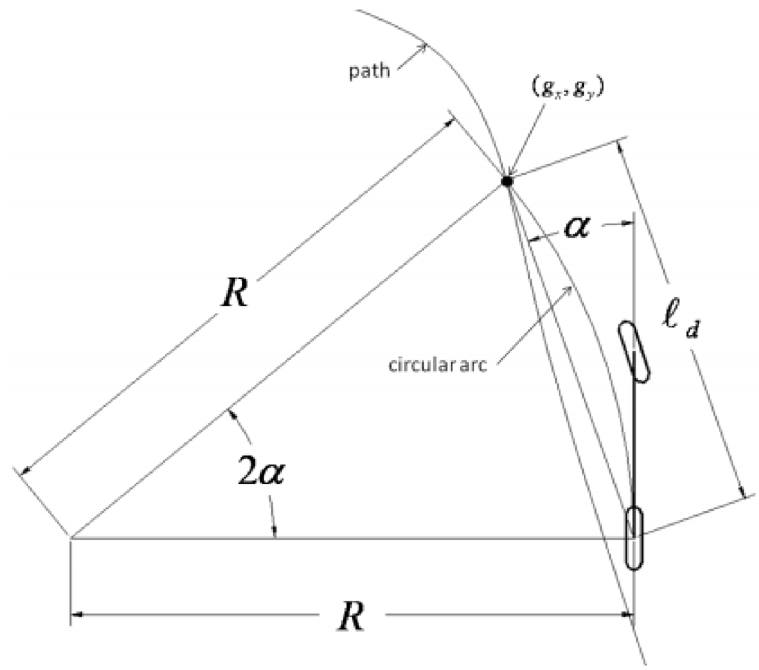
Using the equation of bicycle model,

$$\frac{\ell_d}{\sin(2\alpha)} = \frac{R}{\sin\left(\frac{\pi}{2} - \alpha\right)}$$

$$\frac{\ell_d}{2\sin(\alpha)\cos(\alpha)} = \frac{R}{\cos(\alpha)}$$

$$\frac{\ell_d}{\sin(\alpha)} = 2R$$

$$\boxed{\delta(t) = \tan^{-1}\left(\frac{2L\sin(\alpha(t))}{\ell_d}\right)}$$



Pure pursuit geometry

Reference : https://www.ri.cmu.edu/pub_files/2009/2/Automatic_Steering_Methods_for_Autonomous_Automobile_Path_Tracking.pdf

Vehicle Control (Lateral)

- **Stanley method**

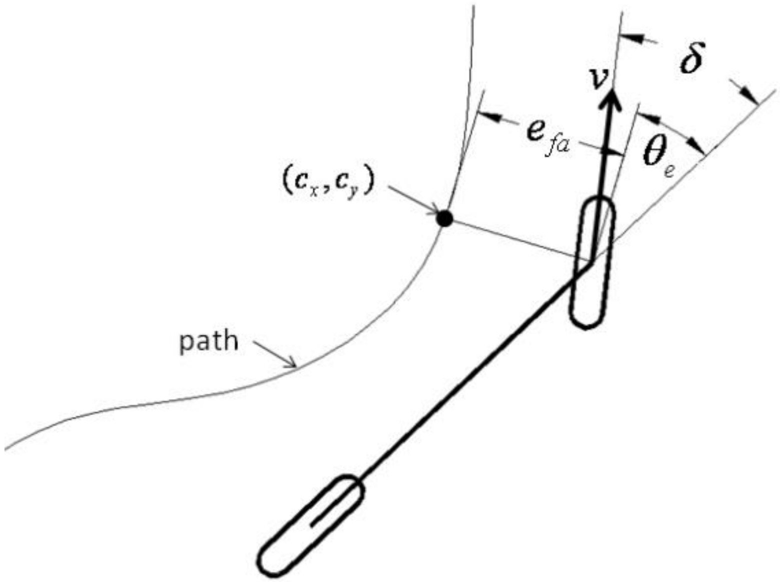
  : The Stanley method is the path tracking approach used by Stanford University's autonomous vehicle entry in the DARPA Grand Challenge, Stanley. The Stanley method is a nonlinear feedback function of the cross track error and heading error.

$$\theta_e = \theta - \theta_p,$$

where θ is the heading of the vehicle and θ_p is the heading of the path at (c_x, c_y)

$$\boxed{\delta(t) = \theta_e(t) + \tan^{-1}\left(\frac{k e_{fa}(t)}{v_x(t)}\right)}$$

Heading error term   Position error term



Stanley method geometry

Reference : https://www.ri.cmu.edu/pub_files/2009/2/Automatic_Steering_Methods_for_Autonomous_Automobile_Path_Tracking.pdf

Programming Assignment

Programming Assignment

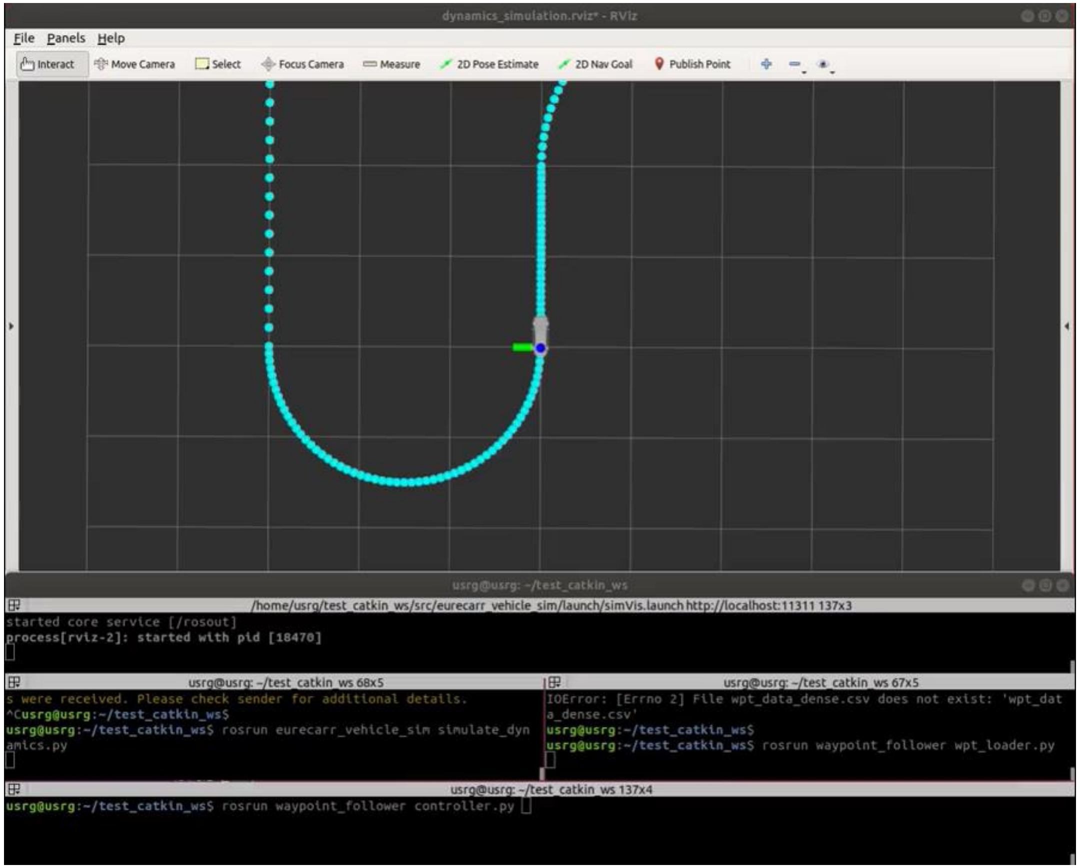
- **Control your own vehicle in a simple vehicle simulator**
  - **Feel free to use and check the following ROS packages.**
    - ✓ Link : https://github.com/hynkis/EE405A/tree/main/Week5/Materials
    - ✓ ROS-based simple vehicle simulator **(eurecarr_vehicle_sim/simulate_dynamics.py)**
      - Kinematic bicycle model-based simulator.
    - ✓ Reference code for vehicle control **(waypoint_follower/controller.py)**
      - The control interface has been already implemented.

        (subscribe to vehicle states and publish control commands)
      - See how to implement
        - Subscribing to an Odometry message.
        - Coordinate transform (from global to local)
        - Calculating cross track error (‘error_y’), heading error (‘error_yaw’).
    - ✓ Waypoint visualizer **(waypoint_follower/ wpt_loader.py)**
      - Visualize a pre-built waypoint trajectory in Rviz.

Programming Assignment

- **Control your own vehicle in a simple vehicle simulator**
  - **Design your waypoint following controller (waypoint_follower/controller.py)**
    - ✓ Tune the P-controller in the reference code.
      - Currently, all the gains (‘kp_y’, kp_yaw’) are 0.
      - You need to find proper controller gains.
      - See ‘TODO’ in the reference code (controller.py).
    - ✓ Implement a PI-controller
      - Try to implement PI controller.
      - You need to compute additional integral of the errors.
    - ✓ (Optional) Implement a Pure Pursuit or Stanley Method.
  - **Design your speed controller (waypoint_follower/controller.py)**
    - ✓ Tune the P-controller in the reference code.
      - Currently, the gain (‘kp_v’) is 0.
      - You need to find proper controller gains.

- **Control your own vehicle in a simple vehicle simulator**
  - **Design your waypoint following controller (waypoint_follower/controller.py)**
    - ✓ Demonstration



Programming Assignment

- **Send followings to hynkis@kaist.ac.kr until 21.04.14**
  - Your **ROS package (waypoint_follower)**
  - Your **Report**
    - ✓ Write **what you have learned** this week.
    - ✓ You can use both **KOR/ENG** in your report.
    - ✓ **Discuss** the following topics:
      - Difference between P and PI control you implemented.
      - The reason for the large cross track error when driving corners.
      - Techniques to minimize the cross track error.

        (hint: look ahead distance, velocity).

  - Please **zip your ROS package and Report** with the following filename.

    EE405A_[lecture_date(YYMMDD)]_[Student ID]_[Full name]

    (e.g., EE405A_210402_20215169_Hyunki_Seong.zip)

Q & A

Email : hynkis@kaist.ac.kr