RE510

Design a waypoint following controller for the autonomous vehicle.

(TA) Seongwoo Moon
School of Electrical Engineering
KAIST

seongwoo.moon@kaist.ac.kr

Experiment Objectives

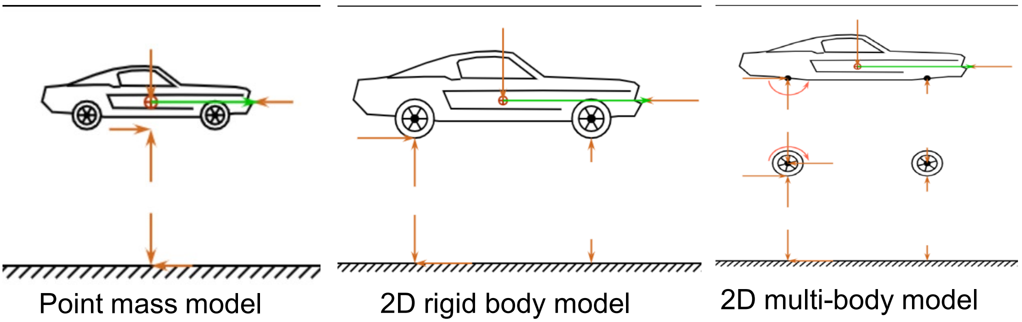
Experiment objectives:

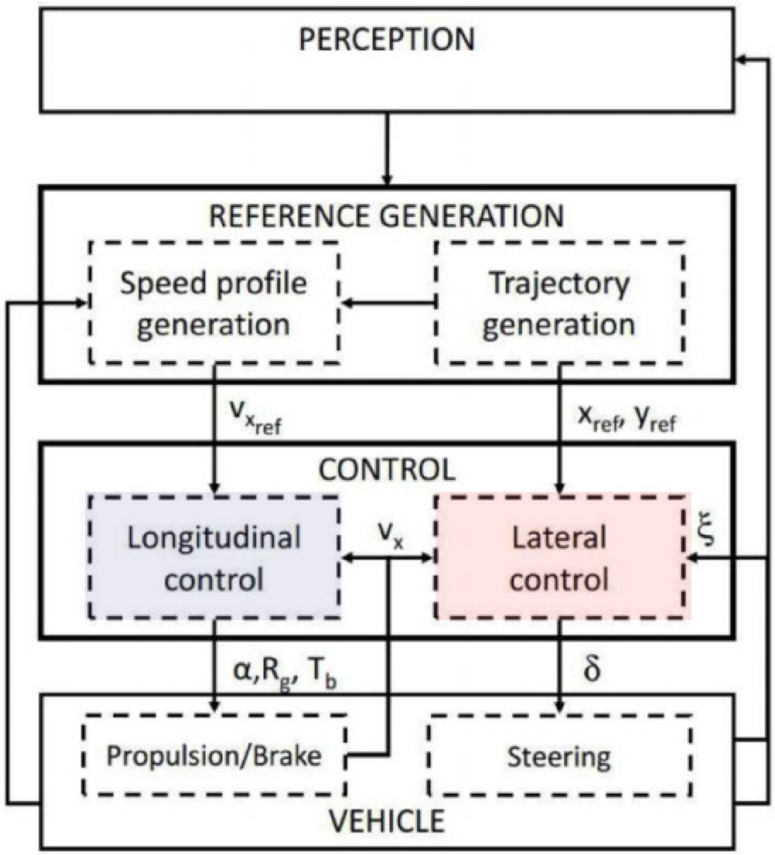
- Understand Vehicle Kinematic Model
- Learn how to design Vehicle Controller
  - Longitudinal Control (PID Control)
  - Lateral Control (Pure Pursuit, Stanley Method)
- Programming Assignment :
  - Design your waypoint following & speed controller.
  - Reference codes (simulator, controller) will be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

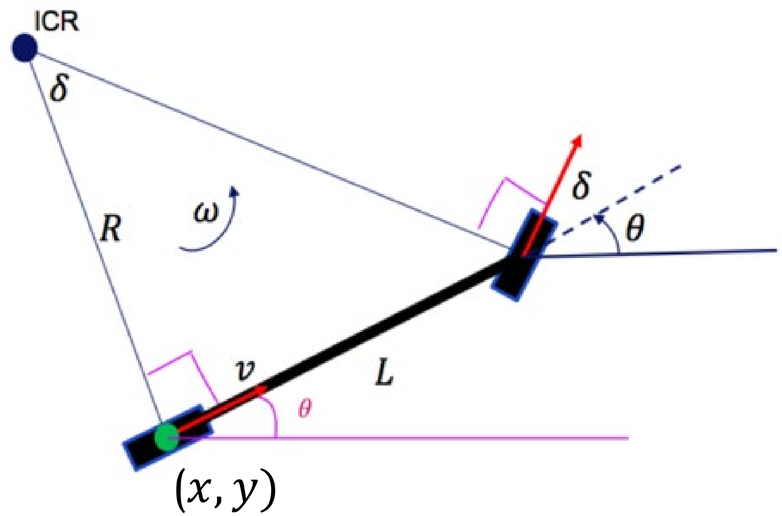




Block diagram for a vehicle control system

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)

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

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

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

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

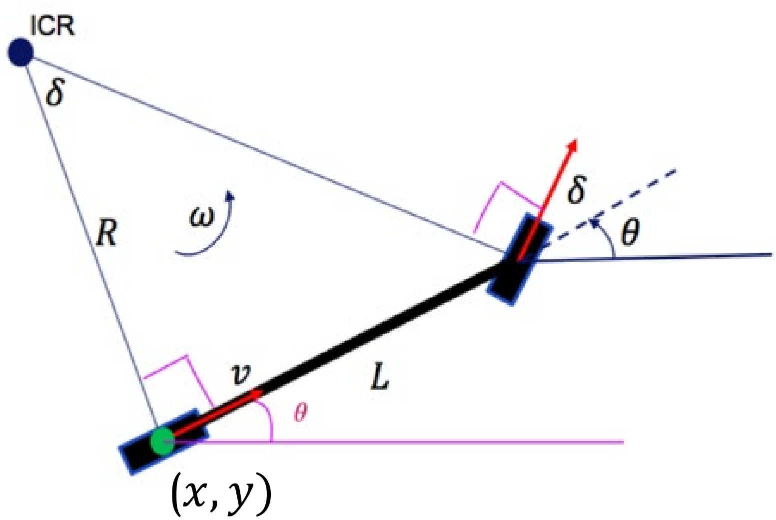
x : position x
y : position y
ψ : yaw angle
v : velocity

δ : steering angle
a : acceleration (throttle)

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)

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

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

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

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
```

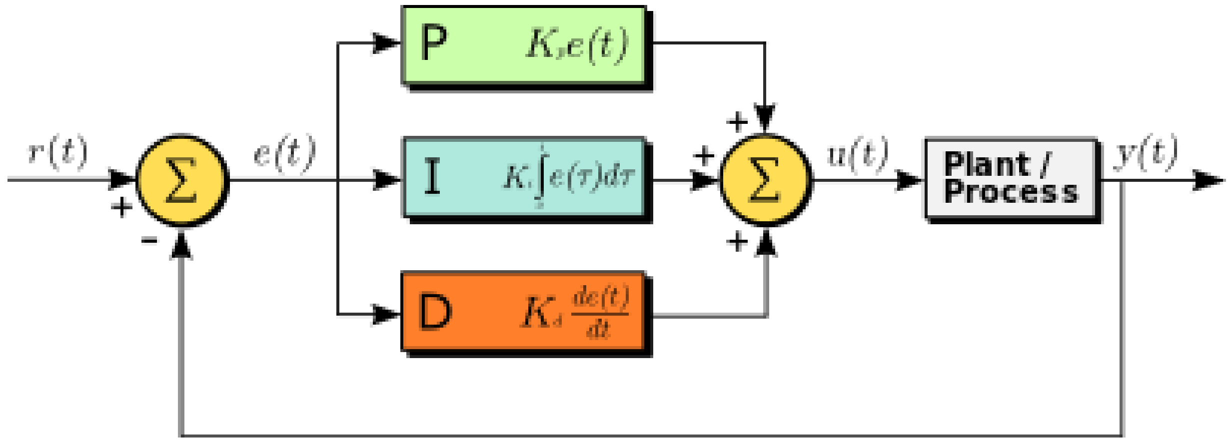
x : position x
y : position y
ψ : yaw angle
v : velocity
δ : steering angle
a : acceleration (throttle)

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.

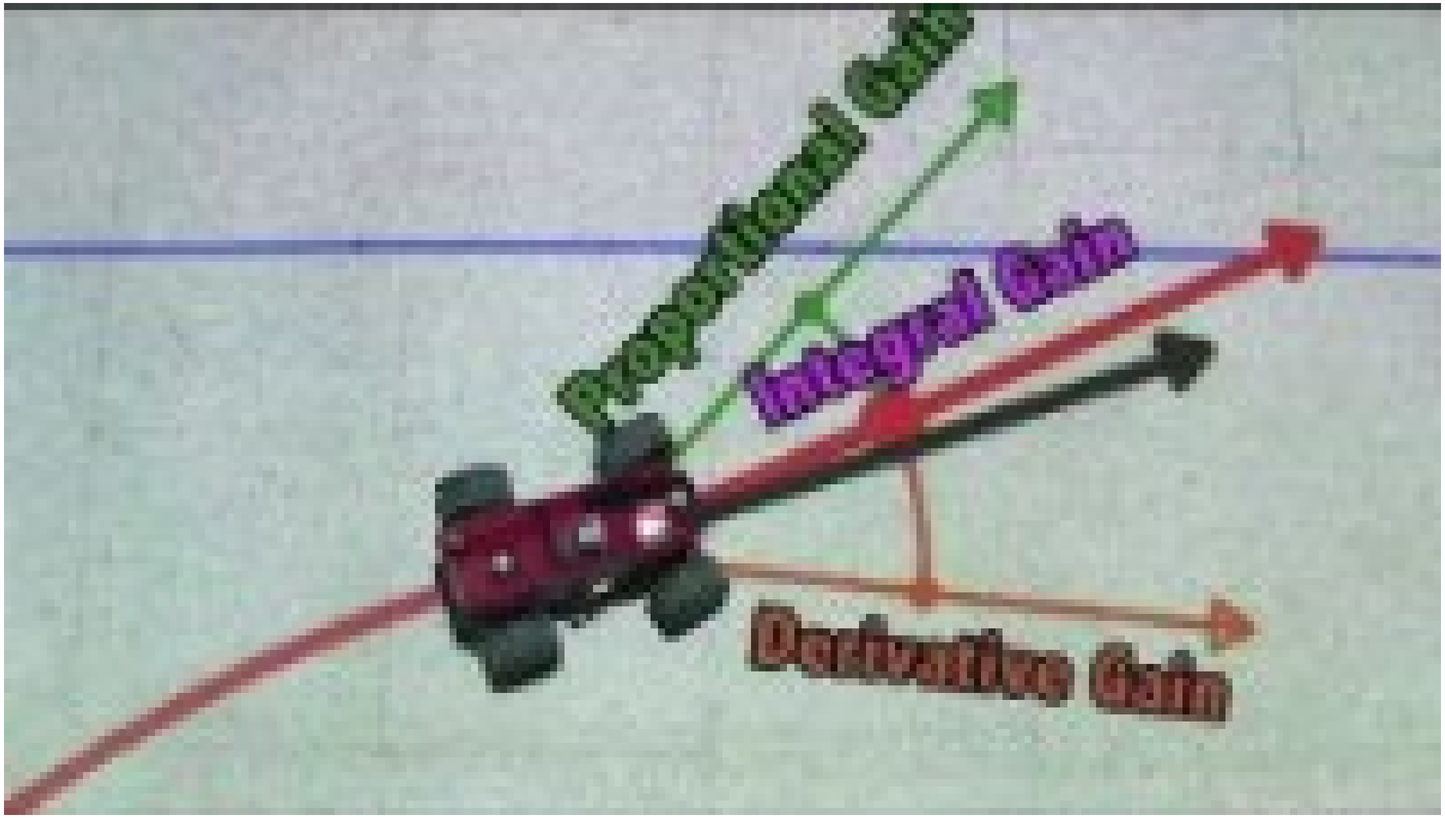
Vehicle Control (PID Controller)

- **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

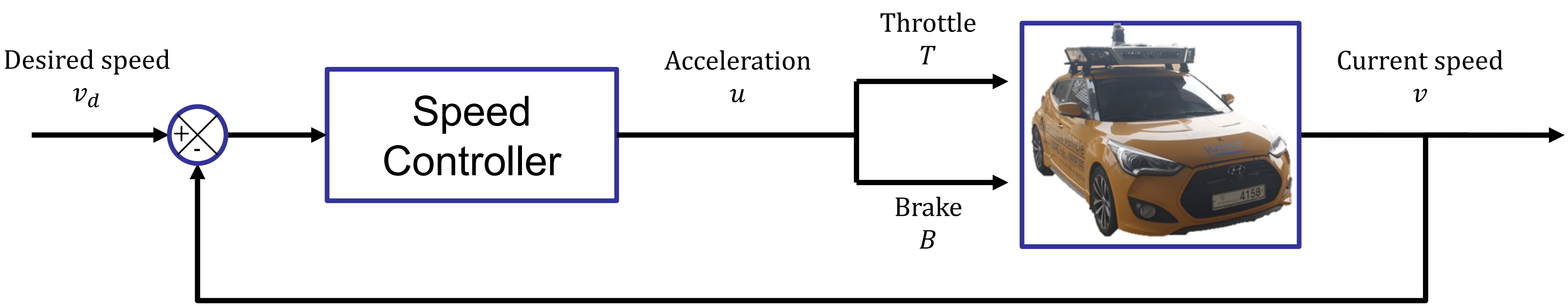


Reference : https://www.youtube.com/watch?v=4Y7zG48uHRo

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 – in global coordinate system

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)

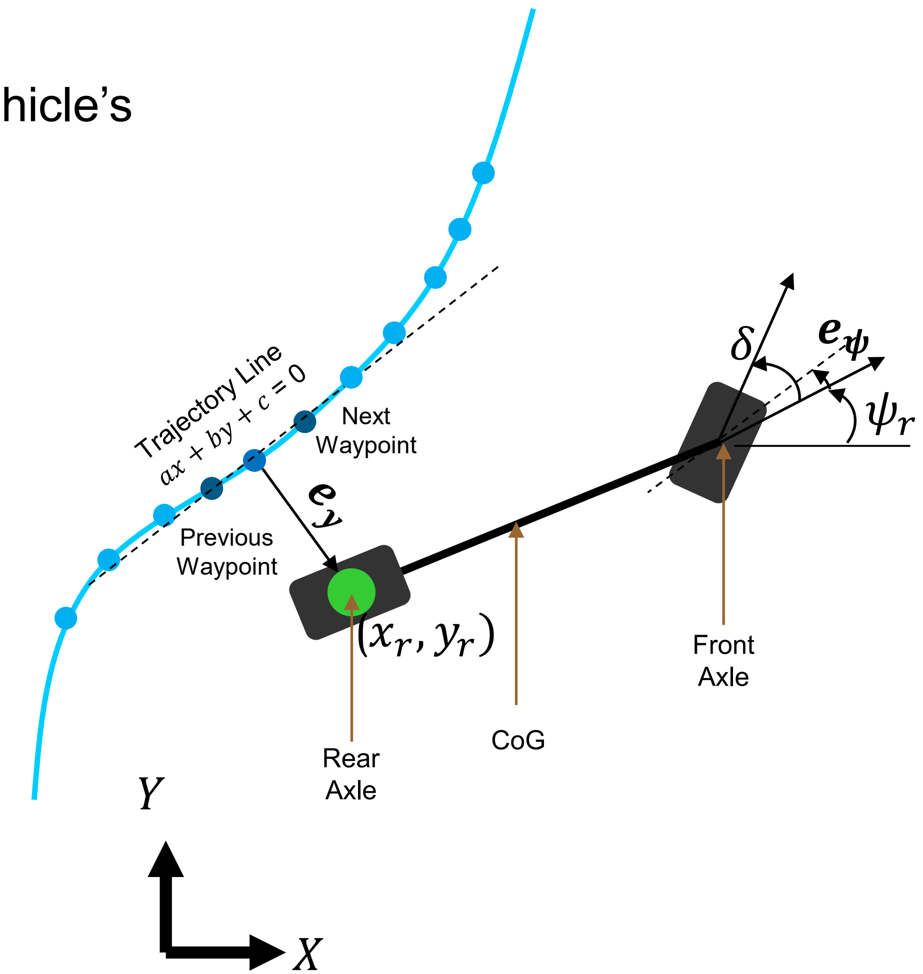
Cross track error

$$e_y = \frac{ax_r + by_r + c}{\sqrt{a^2 + b^2}}$$

- Derived from the distance between point and a line.
- the distance between the ego and **closest waypoint** also can be used.

Heading error

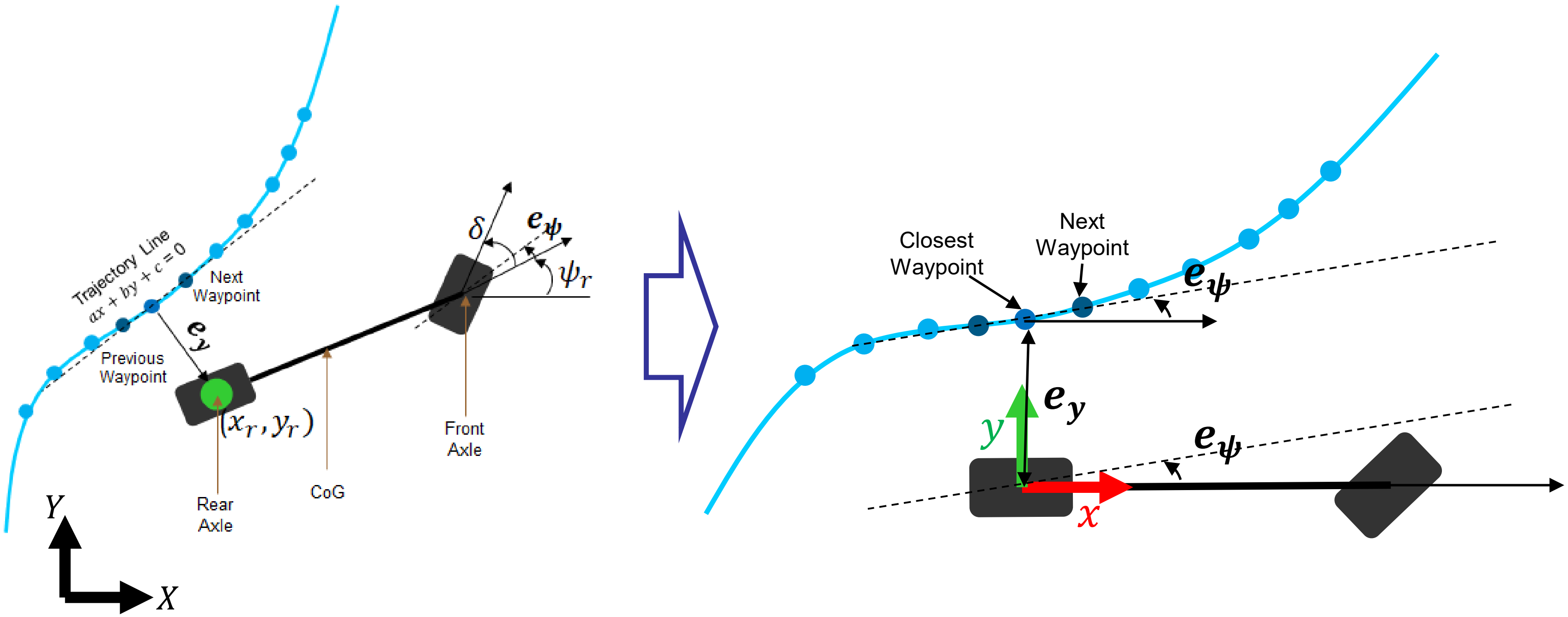
$$e_\psi = \tan^{-1}\left(\frac{-a}{b}\right) - \psi_r$$



Vehicle Control (Lateral)

- **Geometry for Lateral Control – in local coordinate system**

  Transformation from global to local coordinate waypoint trajectory.
  - ✓ Cross track error (e_y) becomes a **y value** of the closest waypoint.
  - ✓ Calculate yaw error (e_{yaw}) becomes a **slope** between closest and next waypoints.

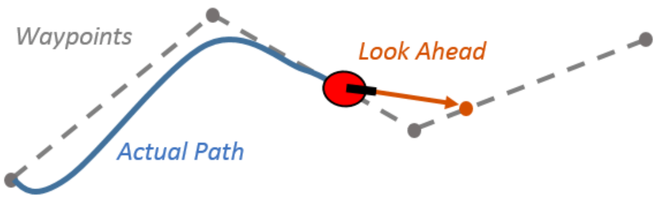


Geometry in the global coordinate | Geometry in the local coordinate

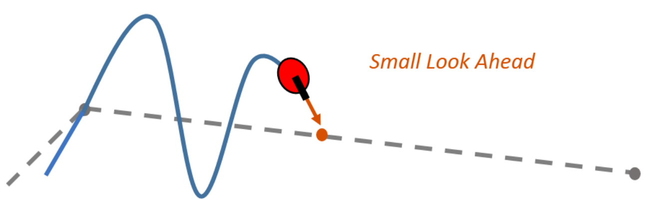
Vehicle Control (Lateral)

Look-Ahead Distance

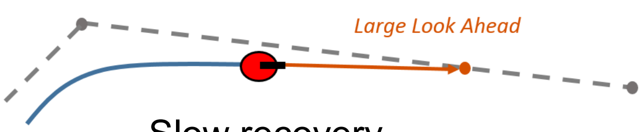
- Look-ahead distance is one of the main tuning parameters for the lateral controller.
- The look-ahead distance is how far along the path the robot should look to compute control commands.



- 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.
- One simple Implementation of look-ahead distance is **adding additional look-ahead distance or index to the closest waypoint index**.



- 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.

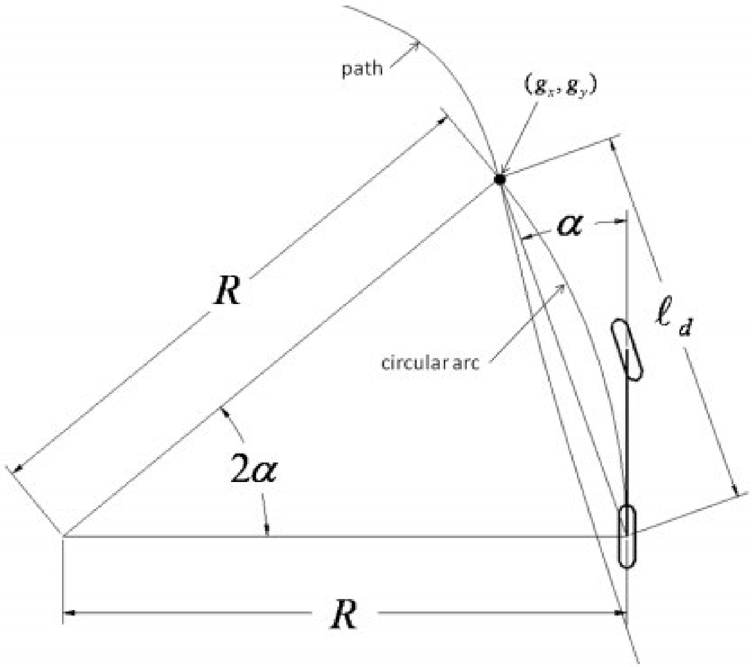
Using the equation of bicycle model,

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

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

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

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



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**

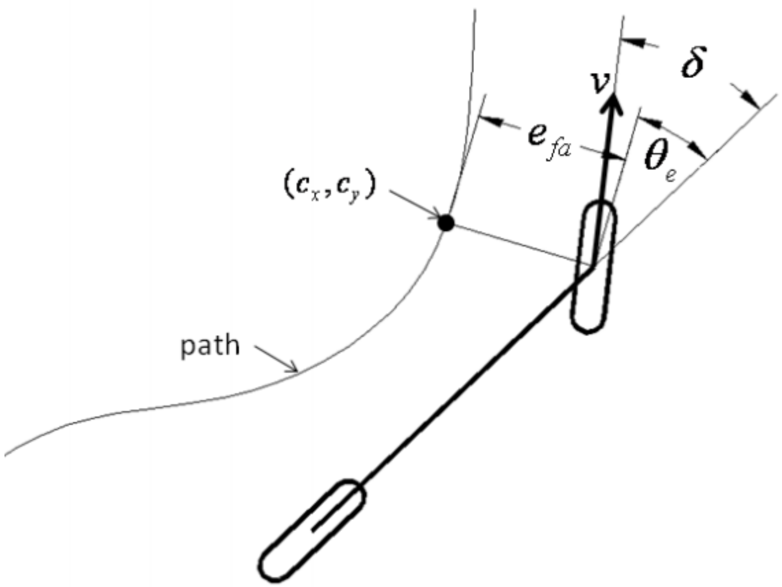
  : The Stanley method is the path tracking approach used by Stanford University's autonomous vehicle entry in the DARPA Grand Challenge, Stanley. The Stanley method is a nonlinear feedback function of the cross track error and heading error.

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

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/RE510
    - ROS-based simple vehicle simulator **(simple_vehicle_sim/simulate_dynamics.py)**
      - Kinematic bicycle model-based simulator.
    - Reference code for waypoint following controller **(waypoint_follower/controller.py)**
      - The control interface has been already implemented.

        (subscribe to vehicle states and publish control commands)
    - Waypoint visualizer **(waypoint_follower/wpt_loader.py)**
      - Visualize a pre-built waypoint trajectory in Rviz.
    - See the README.md for the details about packages.
      - simple_vehicle_sim/README.md
      - waypoint_follower/README.md

Programming Assignment

- **Control your own vehicle in a simple vehicle simulator**
  - **Install the dependencies.**
    - ✓ Download the ROS packages and put them in your ~/catkin_ws/src.
    - ✓ **Vehicle simulator (eurecarr_vehicle_sim)**
      - Run the following command to install ROS dependencies for the `src/` directory.
        - cd ~/catkin_ws
        - rosdep install --from-paths src --ignore-src -r –y
      - Install Python Dependencies.
        - pip2 install numpy –user
        - pip3 install numpy –user
    - ✓ **Waypoint following controller (waypoint_follower)**
      - Install Python Dependencies.
        - pip2 install numpy pandas –user
        - pip3 install numpy pandas --user

Programming Assignment

- **Control your own vehicle in a simple vehicle simulator**
  - **Run the vehicle simulator and controller.**
    - **Vehicle simulator (simple_vehicle_sim)**
      - Open a terminal and launch the simulation.
        - roslaunch simple_vehicle_sim run_sim.launch
    - **Waypoint following controller (waypoint_follower)**
      - Run the waypoint visualizer (to visualize the waypoint trajectory in Rviz).
        - rosrun waypoint_follower wpt_loader.py
      - Run the controller.
        - rosrun waypoint_follower controller.py
    - **See the README.md for the details about the ROS packages.**
      - simple_vehicle_sim/README.md
      - waypoint_follower/README.md

Programming Assignment

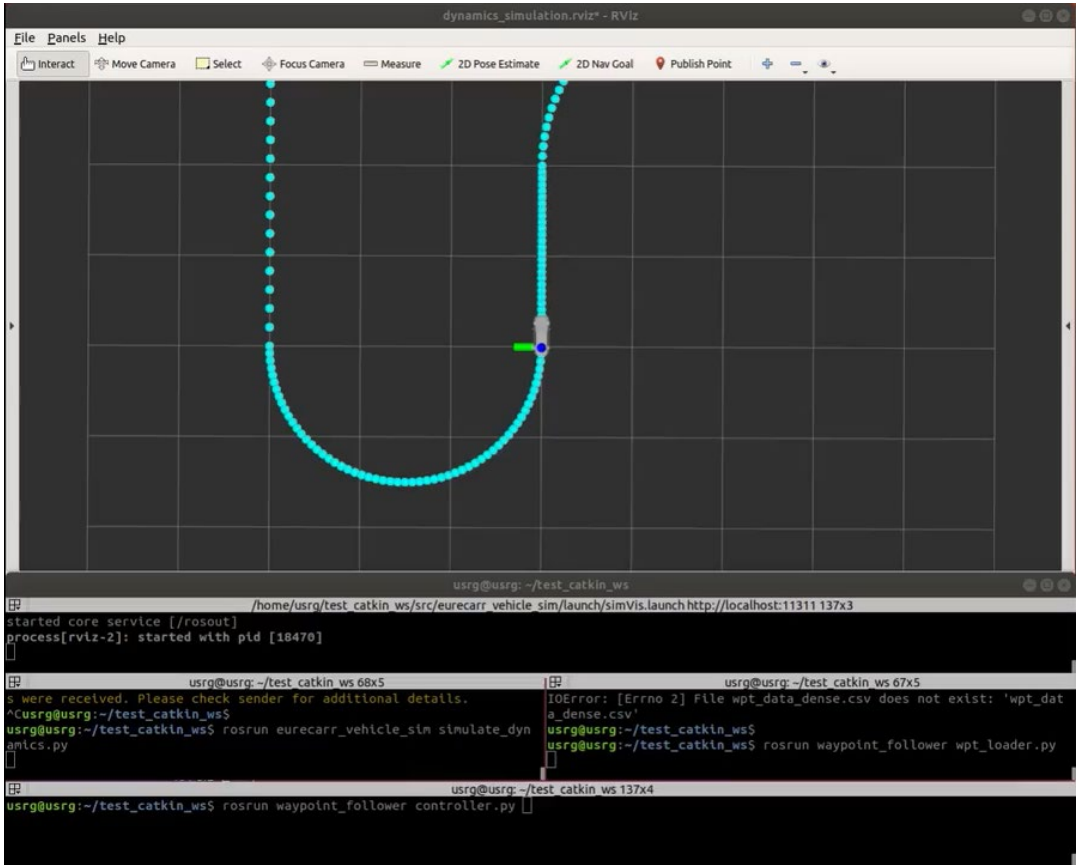
- **Control your own vehicle in a simple vehicle simulator**
  - **Design your waypoint following controller (waypoint_follower/controller.py)**
    - **Implement some helper functions for waypoint following.**
      - *global2local* : transform from global to local coordinate trajectory.
      - *find_nearest_point* : find the nearest point w.r.t. current ego vehicle's pose.
      - *calc_error* : calculate crosstrack error and yaw error w.r.t the look-ahead point.
      - See 'TODO' in the reference code (controller.py).
    - **Design a steer and speed controller functions.**
      - Use above-mentioned helper functions.
      - *steer_control* : compute proper steering angle command.
      - *speed_control* : compute proper throttle command (acceleration in the kinematic model).
      - Feel free to change the code if you want.
      - See 'TODO' in the reference code (controller.py).
    - **Record a demo video of your vehicle driving more than two laps.**
    - **Evaluate the average cross-track & speed errors for the two laps.**
      - The example code computes and prints the average of the cross-track and speed errors.

        ※ The cross-track error for evaluation is calculated in terms of the closest waypoint.

        ※ Do not change the initial position of the vehicle.

Programming Assignment

- **Control your own vehicle in a simple vehicle simulator**
  - **Design your waypoint following controller (waypoint_follower/controller.py)**
    - ✓ Demonstration (https://youtu.be/__jTwuCmPd0)



Programming Assignment

- **Control your own vehicle in a simple vehicle simulator**
  - Please zip your 1) **ROS package (waypoint_follower), 2) Report and 3) Demo video** with the following filename.

    RE510_[Student ID]_[Full name]

    (e.g., RE510_20240000_Seongwoo_Moon.zip)

  - In your report, you need to
    - ✓ Write **what you have learned** this week.
    - ✓ Mention **the average cross-track & speed errors** of your controller.
    - ✓ **Discuss** the following topics:
      - Effects of the **lookahead distance.**
      - Strategies to **minimize the cross-track error** at **straight lines** and **corners**.

Programming Assignment

- **Control your own vehicle in a simple vehicle simulator**
  - Score criteria
    - ✓ Make runnable code & record video (50%).
      - Fill the codes in TODOs of *waypoint_follower/controller.py*. (30%)

        ※ You are allowed to implement another controller in the 'controller.py' if your controller can make the vehicle follow the waypoints.
      - Record a video of your vehicle driving more than two laps. (20%)

        ※ Your video should be playable on Windows.
    - ✓ Write report (30%)
      - About what you have learned this week. (10%)
      - About the discussion. (20%)

        ※ Page limit: 4 pages
    - ✓ Implement a robust waypoint controller (20%)
      - Average of the cross-track error for two laps. (10%)
      - Average of the velocity error for two laps. (10%)

$$\frac{error_{max} - error_{yours}}{error_{max} - error_{min}} \times 10\%$$

※ The cross-track error for evaluation is calculated in terms of the closest waypoint.

Thank you

Email : seongwoo.moon@kaist.ac.kr