EE 5550: Autonomy I, Fall 2025

Prof. Ossareh

Final Project: autonomy of self-driving vehicles

Background:
You are a control engineer at an automotive company rapidly advancing toward autonomous vehicles. You are part of a project to upgrade the existing cruise control system. Your project manager has asked you to design and validate an Adaptive Cruise Control + Lane Change (ACC+LC) algorithm for a midsize sedan

Instructions:

- This is your final project and is worth 20% of your final grade.
- You will work in groups of 2 or 3.
- Submit one final report per team (PDF) and your code (`car.py` and `controller.py` in Task 4) via Brightspace by **Dec 9 at 11:59pm**. Late submissions will not be accepted so please do not start late and submit on-time.
- Your audience for the report is your project manager, who knows the problem statement, and has taken Controls and Autonomy I at UVM. You want to convince her that your algorithms work, are validated, and are ready to be implemented.
- Report format: report must be typed (not hand-written/scanned), neat, and clear. Control block diagrams may be hand-drawn.
- What to include in your report:
  - All deliverables requested throughout this document (shown in green text). Show your thought process and reasoning.
  - A brief Introduction that motivates the problem. Do not copy this manual.
  - A Conclusion with summary, lessons learned, challenges, and what you'd do differently next time (please include all these components for full credit).
  - Label all your plots and include units, legends, and caption as appropriate.
  - Be concise and clear; avoid unnecessary verbosity.
- Thoroughly comment your code!
- Do not use generative AI tools such as ChatGPT or Gemini for any part of this assignment.

The project contains four tasks.

<u>Recommended timing:</u> complete Tasks 1-3 by Dec 1. These tasks will be covered on your second exam so be sure to study the material and your solutions carefully. Complete Task 4 before the final deadline (Dec 9).

1 Background

Most production automobiles include a cruise control feature that automatically regulates the speed of the car to a value set by the driver. In the past few years, automobiles have been fitted with sensors such as radars, lidars, cameras, and GPS units. These additions have transformed traditional cruise control into Adaptive Cruise Control (ACC), which adapts to traffic conditions, and may even be utilized to minimize fuel consumption in an intelligent manner. More recently, ACC algorithms have been augmented with lane-keeping and lane-change capabilities, leading to Adaptive Cruise Control with Lane Change (ACC+LC) systems.

Traditional ACC maintains a safe distance between the “ego vehicle” (your vehicle) and the “lead vehicle” (the one ahead). When the lead vehicle slows down, the ACC reduces speed to maintain a safe gap; when it accelerates beyond the set speed, the ACC resumes regular cruise-control operation.

ACC+LC extends this concept by keeping the vehicle centered in its lane (using electric power steering) and deciding if/when to change lanes. If a lane change is both safe and beneficial for maintaining speed, the vehicle initiates and completes the maneuver automatically.

In this project, you will design an ACC+LC system. You will gain experience with control design, controller implementation, design validation, requirements specification, and technical communication.

Your project tasks are as follows:

- Task 1: create a `Car` class (your plant model) that includes simplified longitudinal and lateral dynamics as well as fuel consumption. You will later integrate this class with the provided graphical environment for validation (in Task 4).
- Task 2: linearize the nonlinear longitudinal dynamics of the car and obtain a transfer function. Use it to design a PI controller with anti-windup for standard cruise control.
- Task 3: linearize the nonlinear lateral dynamics and obtain a transfer function. Use it to design a cascaded controller for lane keeping and lane change.
- Task 4: Combine and augment the previous controllers to achieve a complete ACC+LC algorithm that also optimizes fuel consumption.

For controller design, you will treat the longitudinal and lateral dynamics as decoupled. Doing so simplifies the design in Tasks 2 and 3.

Setup Instructions: Download the files attached to this assignment from Brightspace and unzip them. Set up your Python environment by downloading and installing the correct packages, as indicated in requirements.txt. In this project, you will only modify `car.py` and `controller.py`. Do not modify any other files.

2. Final Project Instructions

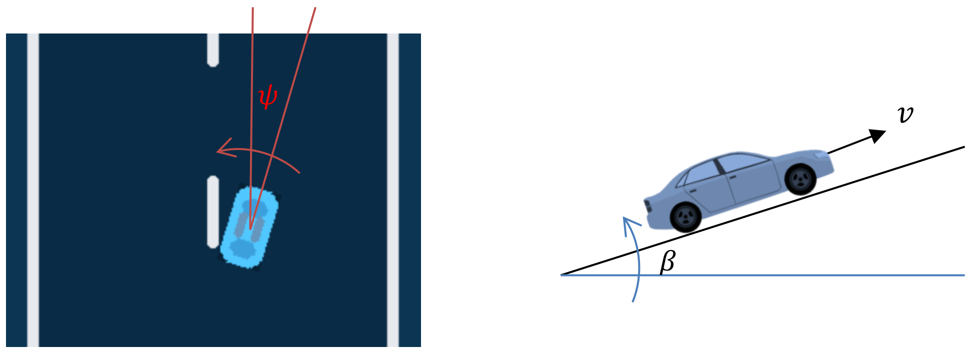
2.1 Assumptions

Every validation platform is built under certain assumptions that should be clearly stated. Here, we list a few of the key assumptions on the ego vehicle: the car is a typical mid-sized sedan with a naturally aspirated gasoline engine; the vehicle is moving at highway speeds and so the transmission is in the highest gear; the rolling resistance of the tires is a constant; the road surface and the suspension are rigid; the tires don't slip; the vehicle is modeled as a rigid body with a point mass at its center of gravity; the dynamics of the powertrain are ignored as they are typically much faster than the vehicle dynamics; we ignore the effects of accessory loads (e.g., A/C, power steering, etc.) that draw power away from the powertrain; the yaw angle changes only slightly, thus, the road grade in the direction of travel is approximated as equal to the longitudinal road grade.

Other vehicles (traffic) are modeled as constant-velocity point masses that do not change lanes and match the ego's speed if they approach from behind. For the purpose of simulation, we will assume the physics are discretized at 1/300 seconds while the controller runs at 1/60 seconds (i.e. 60 Hz).

2.2 Modeling

Consider a car traveling along a highway, as illustrated in the top and side views below.



Top view

Side view

Let v [m/s] denote the tangential (i.e. forward) speed of the vehicle along its path in its local coordinate frame. In the inertial (world) frame, the vehicle's longitudinal and lateral positions along the road are denoted by x [m] and y [m], respectively. The variable β [rad] denotes the road grade in degrees (see figure above on the right), and ψ [rad] denotes yaw/heading (see figure on the left).

With the assumption that road grade and yaw angles are small, the kinematics of the car are given by:

$$\dot{x} = v \cos \psi$$

$$\dot{y} = v \sin \psi$$

To derive the dynamics, we note that the equations of motion of the vehicle in the local coordinate frame are governed by Newton's second law of motion: $m\dot{v} = \sum F$. The dominant propelling force is the driving force (engine/braking), denoted by F_d. The other external forces are the climbing force, F_c, the (1) resistance, F_{roll}, and the aerodynamic drag, F_{air}:

$$m\dot{v} = F_d - F_{air} - F_c - F_{roll}.$$

F_{roll} describes the resistance due to the tires/wheels and can be regarded as a constant at high speeds. The climbing force is the force due to gravity on a road with non-zero grade:

$$F_c = mg\sin(\beta)$$

where g is the acceleration due to gravity.The aerodynamic drag is related nonlinearly to the speed:

$$F_{air} = av^2 + bv.$$

The driving force, F_d, results from the engine torque transmitted through the tires or from the braking force applied by the brakes through the tires. Since both engine and braking forces are bounded, we have:

$$F_{d.min} \leq F_d \leq F_{d,max}$$

The driving force, F_d, is one of the control inputs in this project. If it is positive, the engine will produce this force. If it is negative, the brakes will produce this force.

Finally, the vehicle's yaw motion is governed by

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

where L is the wheelbase, and δ is the steering angle, defined as the angle the front tires make with the road. δ is the second control input in this project (note: there is a dot on top of ψ in the above equation that is hard to see). To ensure safe driving conditions, we impose:

$$-\delta_{max} \leq \delta \leq \delta_{max}$$

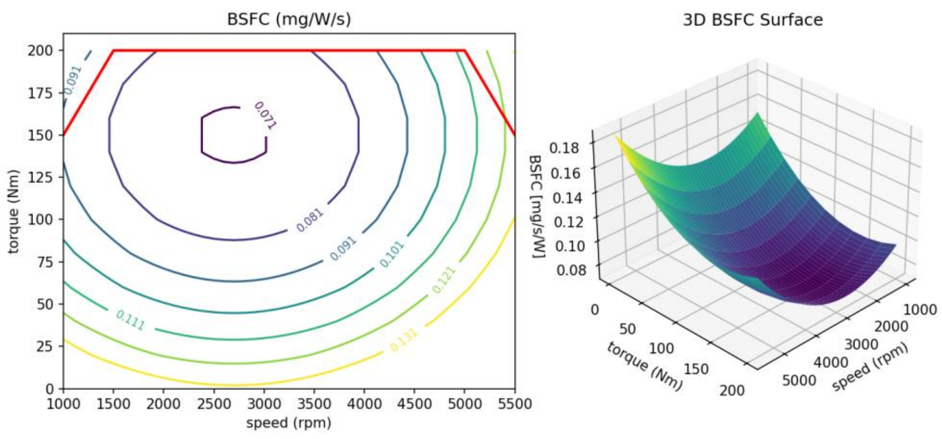
Where δ_{max} is a given steering limit.

Fuel Economy:

The brake specific fuel consumption (BSFC) quantifies the rate at which an engine burns fuel to produce one unit of power. It reflects the engine's efficiency. Manufacturers typically determine the BSFC map experimentally using an engine dynamometer. The result is a 3D surface that characterizes BSFC as a function of engine speed and torque:

$$\text{BSFC} = f(N_e, T_e) \;\; \left[\frac{mg}{s \cdot Watt}\right]$$

where N_e is the engine's rotational speed [RPM] and T_e is the engine torque [N.m]. An example BSFC surface and its contour map is shown below. The level sets on the contour plot denote lines of constant efficiency. The red line shows the maximum torque available at each engine speed. In this example, the engine produces up to 200 Nm between 1500–5000 RPM, and achieves maximum efficiency near 2700 RPM at ~75% of rated torque.



Now, the engine torque can be expressed in terms of the driving force through:

$$T_e = \frac{T_w}{\eta_g \eta_d}, \qquad T_w = r_w F_d$$

where η_g is the transmission gear ratio, η_d is the final drive ratio, and r_w is the tire radius. Since the drivetrain experiences losses (e.g., due to elasticity and friction), the combined relationship becomes:

$$T_e = \frac{1}{\xi} \frac{r_w}{\eta_g \eta_d} F_d$$

where ξ is the drivetrain efficiency. Similarly, engine speed can be expressed in terms of vehicle speed (assuming a fixed gear):

$$N_e = \frac{60}{2\pi} \frac{\eta_g \eta_d}{r_w} v$$

From the BSFC map, the instantaneous fuel burn rate is:

$$\text{fuel rate} = \text{BSFC} \times \text{Power} = \text{BSFC} \times T_e \times N_e \frac{2\pi}{60} = \frac{1}{\xi} \times \text{BSFC} \times F_d \times v \quad \left[\frac{mg}{s}\right]$$

If F_d is negative or small, the engine still burns some fuel to maintain the catalytic converter's internal temperature and minimize emissions. Therefore, the above equation must be clipped to a lower bound:

$$\text{fuel rate} = \max\left(\frac{1}{\xi} \text{BSFC} \times F_d \times v, \bar{F}\right) \quad \left[\frac{mg}{s}\right] \qquad \boxed{(2)}$$

where $\bar{F}$ is the minimum fuel rate of the engine at low torque (for example, during braking). To get the total fuel consumed, one would integrate the above over the time window of interest:

$$\text{total fuel } (t) = \int_0^t \text{fuel rate } (\tau) d\tau$$

2.3 Task 1: Python model of the ego vehicle dynamics

Your first task is to create a `Car` class that models the ego vehicle. This includes the longitudinal and lateral motion and fuel consumption. To begin, open the supplied `car.py` and complete it as instructed in the comments. Use Forward Euler discretization, with discretization step size given in Section 2.1. For the numerical values of the vehicle parameters, see the table below. Compute $F_{d,max}$ yourself using the maximum engine torque and the formulas in the previous section. Include its value in your report.

Assume that the BSFC map is given by the equation

$$\text{BSFC} = \left(\frac{N_e - 2700}{12000}\right)^2 + \left(\frac{T_e - 150}{600}\right)^2 + 0.07$$

and the maximum engine torque is 200 N.m between 1500 and 5000 RPM (red line in the figure on page 4). As you complete the code, do not forget to include saturations on δ and F_d.

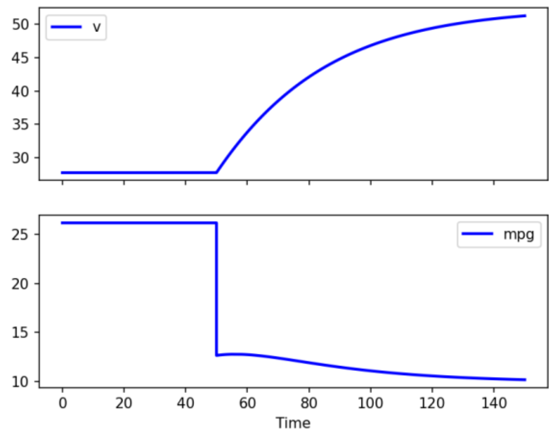
Parameter	Value
m	1300 Kg
F$_{roll}$	100 N
a	0.2 Ns^2/m^2
b	20 Ns/m
g	9.8 m/s^2
$F_{d.min}$	-7000 N
ξ	0.95
η_g	0.8
η_d	3.8
r_w	0.34 m
$\bar{F}$	200 mg/s
L	2.7 m
δ_{max}	0.05 rad

You will now validate (i.e., simulate) your `Car` class to make sure you have implemented it correctly. Create a new python file named `Task1.py` and complete it as follows.

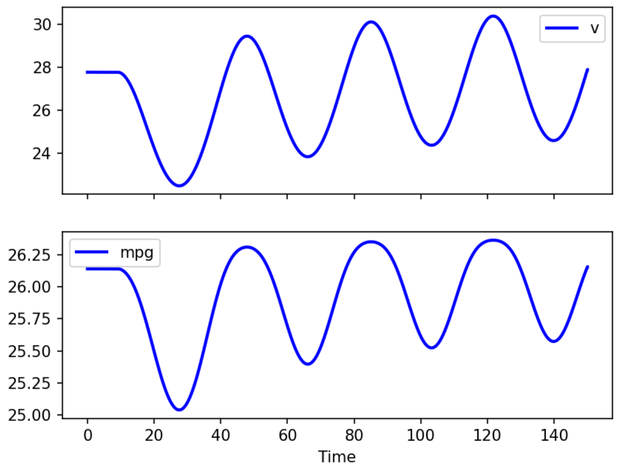
- Create a "road grade profile" (i.e., road grade as a function of longitudinal position along the road):
  ```
  x = np.arange(0, 6101, 1)                         # 6.1 km total
  beta = Amp * np.sin(2*np.pi/1000*x + 300)         # unit is degrees here!
  beta[(x < 500) & (beta < 0)] = 0                   # initial 500m is flat
  ```

  This profile consists of a flat section followed by a sinusoidal segment modeling periodic slopes (e.g., bridge approaches or overpasses). You will later vary amplitude (Amp) to study road-grade effects on speed dynamics.
- Write code to plot the road profile β as a function of *x* (horizontal distance) for an amplitude of 3 degrees (`Amp=3`). Note that profile is defined in terms of x, not time. Include your plot in your report.
- Plot the BSFC surface and contour plots (as shown on page 5) using Matplotlib (look up how to do this on your own). Include your plots in your report.
- Fuel economy in the U.S. is typically expressed in miles per gallon (MPG). Compute the conversion factor to convert fuel consumption in [mg/s] and vehicle speed [m/s] to MPG. Note: 1 gallon of gasoline is 2835 grams. Include your conversion factor in your report.
- Let $F_d^{ss}(v,\beta)$ denote the constant input value required to maintain a given constant speed of v at a constant road grade β. This is called the "equilibrium manifold" of the differential equation in Eq. (1). Analytically (by hand) find the expressions for $F_d^{ss}(v,\beta)$ as a function of v and β. Include the expression in your report. Evaluate $F_d^{ss}(27.78, 0)$ and include this value in your report as well. Note: 27.78 m/s is 100.0 km/h.
- Similarly, let $J^{ss}(v,\beta)$ denote the steady-state value of fuel rate defined in Eq. (2) as a function of a constant forward speed, v, and road grade β. Find the expression for $J^{ss}(v,\beta)$ by hand and include it in your report. You do not need to simplify it. Also, include a surface plot of $J^{ss}(v,\beta)$ in the range of $v \in [23, 28]$ m/s and $\beta \in [-2.5, 2.5]$ degrees (convert to radians).
- **Verification of your model:** set the amplitude (`Amp`) of the road profile to 0, and write the code in `Task1.py` to simulate the car dynamics. To this end, import your `Car` class into `Task1.py`. Instantiate a car object from your `Car` class and create a simulation loop to simulate the car for 150 seconds by calling the `update` function. The input F_d should be a step at $t = 50$ and the steering angle should be $\delta = 0$. The starting value of the step (i.e. from $t = 0$ to $t = 50$ seconds) should be the value you found above ($F_d^{ss}(27.78,0)$) and the final value should be 3000 N.m. Compare your speed

and MPG outputs with the plots below and make sure you are getting the same results. Include your plots in your report (make sure to label your axes, with units). *Hint 1: did you remember to set the initial condition of your car velocity to 27.78* m/s*? Hint 2: at each timestep, you must use the horizontal (x) position of the car to query the corresponding road grade β. You will need to use linear interpolation to interpolate between the β values you defined above.*



- Now change the amplitude of the road profile back to 3 degrees (note that 3 degrees corresponds to a maximum 5% road grade). Remove the step input and instead set F_d to the constant value of $F_d^{ss}(27.78,0)$ throughout the simulation. Simulate the system for 150 seconds. Make sure your plot matches the plots shown below. Include your plots in your report (make sure to label axes, with units).



The project manager is concerned that, even though the force applied by the engine is held constant during the simulation, the fuel economy seems to vary. What's worse, it seems that as the vehicle speed drops, the MPG degrades instead of improving. Convince them that the results make sense.

Include `Task1.py` in your report (in the PDF. Do not submit separately).

2.4 Task 2: Cruise control for longitudinal dynamics

Linearization:

The longitudinal dynamics of the car are nonlinear. For the design of a cruise controller, we will temporarily ignore the climbing force, F_c, and the rolling resistance, F_{roll}, as they are external to the system (i.e., disturbances). Thus, the only active nonlinearity is the aerodynamic drag: $F_{air} = av^2 + bv$. You will "linearize" the dynamics to obtain a transfer function that you will use to design a cruise controller.

1. Please read the Appendix at the end of this document for an introduction to linearization.
2. Since the only nonlinear term is $F_{air} = av^2 + bv,$ you will only need to linearize this term. Follow the process outlined in the Appendix to find the slope, *c*, of this function at the operating point (i.e., 27.78 m/s, flat road). Then, replace the nonlinear expression with
$$F_{air} = cv$$
and write the linearized ODE.
3. Determine the transfer function of the linearized system from *F_d* to *v*. In your report, include the value of c, the linearized ODE, and the transfer function.
4. You are now going to compare the response of the nonlinear and linearized models. This step is called "validating your linear model". Reset the road to be flat (Amp = 0). Apply a step of size 1 at time t = 50 at the input of the nonlinear system (starting from the equilibrium input found in Task 1). Simulate the response for 150 seconds.
5. Separately, compute the step response of the transfer function you derived above.
6. Plot the responses from step 4 and step 5 on the same plot. Make sure to shift the response of linear system by the equilibrium value as well as in time so the responses can be compared. Do the responses match? *Hint: they should. If not, you might want to include more significant digits in your F_d equilibrium.* Include your plot in your report and interpret your results.
7. Repeat steps 4-6 with a step size of 600. Include your new plot and comment on it.
8. The above analysis was done at the operating point defined by $v = 100$ km/h (i.e. 27.78 m/s). Repeat steps 1-3 for a different operating point, one with $v = 150$ km/h. How did the transfer function change? What does this imply in terms of closed-loop performance and robustness as operating point varies, assuming that you design your controller based on the $v = 100$ km/h operating point?

Note: linearization is done for analysis and design only. Leave the vehicle model nonlinear.

Cruise controller design:

You will now design a PI controller with anti-windup to track a desired speed set by the driver. The only measurement for this task is the vehicle's longitudinal speed v. The design specifications are:

a. Zero steady state tracking error despite constant disturbances.
b. Rise time for step tracking must be between 1 second and 3 seconds (note: rise-time is defined as the time it takes to go from 10% to 90% of the final value).
c. No overshoot for step tracking (you don't want a speeding ticket!).
d. Closed-loop response to a unit step disturbance applied at the plant input should settle in less than 10s (note: define settling time as time it takes for the output to reach and remain within [$-1 \times 10^{-6},\ 1 \times 10^{-6}$]).

Perform these steps to complete your design and implementation:

1. Using the transfer function identified above at the operating point of 100 km/h, design a PI controller to track speed setpoints. You will note that the zero of the PI controller. Design a pre-compensation to cancel it. You must perform analysis for this on your own. For your report:
(1) Describe your design process, and include your final controller,
(2) Draw a block diagram of the control system,
(3) Include the step response of the closed-loop system from desired speed to speed (with the

linearized plant and pre-compensator),

(4) Discuss the stability robustness of your design (gain and phase margins),

2. Now open your blank `controller.py` (not the one supplied with the assignment) and write a controller class with `__init__` and `update` functions. The update function should have the form: `update(self, speed, desired_speed)` and should implement the update law of your PI controller with pre-compensation and anti-windup (instantiate a PID object from the PID class developed earlier in the course). Use the controller sampling time provided in Section 2.1. Include your controller code in your PDF (do not submit it as a separate file).
3. Now, set the anti-windup gain to 0 to disable anti-windup. In a separate python file (`Task2.py`), import your controller and simulate the step response (step at time t=50, starting from 100km/h to 101 km/h) of the closed-loop system with your controller and the nonlinear plant. Note that the controller and physics should be simulated with the correct sampling times. *Hint 1: the output from t=0 to t=50 must remain constant. If not, maybe you forgot to set the initial condition of the integrator output in the PI controller or the pre-compensator. Hint 2: if you are getting overshoot due to a closed-loop zero, you may have forgotten the pre-compensator or perhaps you did not initialize the state of the pre-compensator correctly. Hint 3: speeds in the model are all in m/s.* Include your code (controller.py and task2.py) in your report (do not submit them as separate files). Also, include the step response in your report and make sure the specifications are met.
4. Repeat step 3 with a step size of 50 km/h (i.e., step from 100 to 150 converted to m/s). Include the step response and comment on your observations. If the overshoot degraded, explain why.
5. Repeat steps 4 with anti-windup compensation (choose an appropriate value for the anti-windup gain). Include your step responses and comment on your observations.
6. Set the road grade amplitude to 3 degrees (i.e., `Amp=3`). Turn off the step input (i.e., set it to a constant 27.78 m/s). Run the simulation with two sets of gains:
   a. The PI gains you designed above
   b. Half of the PI gains you designed above

   In each case, find the total fuel consumed. Comment on the effect of controller tuning on performance **and** the total fuel usage. Discuss your observations and the implications?

2.5. Task 3: Lane keeping controller for lateral dynamics

Linearization:

The lateral dynamics of the car are also nonlinear because of $\sin\psi$ and $\tan\delta$ terms. For the design of a lane keeping/lane change controller, you will linearize the lateral dynamics from Task 1 to obtain a transfer function that you will use to design a lane change controller.

1. Linearize the lateral dynamics and find the transfer function from δ to ψ and from ψ to y about a straight-line path. You must select an equilibrium point that makes physical sense for this problem. In your report, include the linearized ODE (show your work) and the final transfer function.
2. Validate the linear model against the nonlinear simulation by applying a small pulse to δ (i.e., step up and then step back down to 0). Make sure the amplitude of the pulse is small so that the linear approximation remains valid. Include any relevant plots and discussions in your report. Make sure your arguments are compelling.

Lane keeping/lane change design:

You will now design a controller that computes steering angle δ such that the lateral position, y, tracks a desired lateral position y_{ref} (eventually, y_{ref} will be the midpoint of one of the lanes). You can assume the vehicle's lateral position, y, and the vehicle heading, ψ, are both measured. The design specifications are:

a. Zero steady state tracking error despite constant disturbances.
b. Rise time for step tracking must be between 1 second and 4 seconds
c. No overshoot for step tracking.

Perform these steps:

1. Using the transfer function identified, design a controller to achieve the specifications. You must select a controller structure and justify your answer. Hint: cascaded control with pre-compensation works well. For your report:
   (1) Describe your design process, and include your final controller gains,
   (2) Include the step response of the closed loop system from y_{ref} to y (with the linearized plant),
   (3) Discuss the stability robustness of your design (gain and phase margins),
   (4) Draw the block diagram of the entire system, including both lateral and longitudinal dynamics. All the loops must be visible, and the plant should be clearly marked. Don't forget saturation blocks, as necessary.

2. Now open your `controller.py` from Task 2 and add your lateral controller to the longitudinal controller code you previously designed. The format of the update function is now modified to: `update(self, speed, y, phi, desired_speed, desired_y).` Do not forget pre-compensation and anti-windup as appropriate.
3. In a new file, `Task3.py`, write code to simulate the lateral dynamics (i.e. a lane change maneuver). To this end, set the road grade to 0, and let y_{ref} be a step signal. Include two plots, one with step response to a small step (0.1 meters) such that anti-windup is not active, and another with a large step such that anti-windup becomes active (e.g., 20 meters). Include the plots in your report and make sure the specifications are met. Your report should also include your code in Task3.py and your code in controller.py (do not submit them as a separate files).

2.6. Task 4: self-driving mode

In this task, you will design a self-driving controller that combines cruise control (Task 2) and lane changing (Task 3) behaviors, and further introduces safe following logic and fuel optimization. Open the supplied `controller.py` file and modify only this file (do not change any others). See the in-file comments for a description of each function's arguments.

Overview of the control problem: The driver specifies a desired speed v_{driver} and a desired lane L_{driver}. In the simulator, you will specify v_{driver} using the up/down arrow keys and L_{driver} using the left/right keys. We assume the maximum speed limit on the road is 27.78 (100km/h) and the minimum speed limit is 20.83 (75 km/h). The road centerline is the reference position (0 meters) for the lateral direction, the midpoint of the right lane is at location $+11.25$ meters and the midpoint of the left lane is at -11.25 meters (very large lanes!).

The self-driving controller will keep vehicle speed close to v_{driver} but treats speed as a slack for fuel optimization. It chooses a desired speed v_{des} within the range $[\mathrm{sat}_{20.83}^{27.78}(v_{driver} - 3), [\mathrm{sat}_{20.83}^{27.78}(v_{driver} + 3)]$, such that the total fuel consumed is minimized over a prediction horizon. The vehicle should initiate a lane change when a slower vehicle is ahead and a safe lane change is possible.

More specifically:

- If there are no vehicles ahead, the controller computes v_{des} as described above to minimize fuel consumption over a chosen prediction horizon. This must be formulated as an optimization problem. The cruise controller from Task 2 is then employed to track v_{des}. You may query road grade at any X_POSITION along the road by running:
```
        beta = math.atan(grade.grade_at(X_POSITION) / 100.0)   # radians
```
- If there is a vehicle ahead, the vehicle must maintain a minimum safe distance of at least 10 meters with the vehicle in front. If the vehicle ahead is going slower than the ego vehicle, the vehicle should perform a lane change (using the controller you designed in Task 3) when it is safe to do so. This can be done using an optimization-based approach or a path planning heuristic, your choice. Note that the relative position and speed of all vehicles are available to the controller in the `other_cars` variable. Each row of `other_cars` contains information about one of the vehicles in view: [relative distance (between ego and vehicle) in the longitudinal direction, relative distance in the lateral direction, relative speed between]. You can use this information in your controller.
- Driver safety (i.e., no collisions) is the most important attribute here. At no point shall the distance with vehicle in front be less than 7 meters. You will receive a red warning message at the terminal if your car gets closer than 7 meters to any other vehicle.
- Driver comfort is important as well - no unnecessary accelerations, decelerations, lane changes, or oscillations.

Your final controller must leverage the longitudinal and lateral controllers from Tasks 1 and 2. You must use optimization for fuel, ideally a fast LP/QP (e.g., linearize BSFC/fuel). Be mindful that if you use a nonlinear optimization solver blindly, your frame rate may drop (and the controller will miss its sampling deadlines).

After you've designed your controller, run `interactiveSim.py` to run the simulator. Verify that the specifications are met.

In your report, thoroughly explain your methodology (trial-and-error alone will not be accepted). Include a robustness analysis to convince your manager that your solution will work on a realistic car in a realistic driving environment (e.g., robustness to parameter mismatch, control delay, etc.).

Please submit a single PDF file containing your report. Do not forget introduction and conclusion, see page 1. Please also submit your Task 1's `car.py` and Task 4's `controller.py` file. Do not submit anything else! The report must be self-contained, so include sufficient detail for me to know what you've done (including Python scripts such as `Task2.py`). I will grade your report and will copy your `car.py` and `controller.py` into my own folder and run my own `interactiveSim.py`.

APPENDIX: linearization

Equilibrium Points

By definition, an equilibrium of a differential equation $\dot{x} = f(x, u)$ is defined as a "point" (defined by initial conditions and constant inputs) that satisfies the following property: if the system starts at an equilibrium point at time $t = 0$, it will remain there for all time. For example, in the inverted pendulum experiment from undergrad controls, the upward position and downward positions are both equilibria (though one is a stable equilibrium and the other is an unstable equilibrium).

In general, stable linear systems have a unique equilibrium point: $y_{ss} = P(0)u_{ss}$, where ss denotes steady state value and $P(0)$ is the DC-gain of the system. If the system output starts at y_{ss}, and u_{ss} is constantly applied at the input, the output will remain at y_{ss} forever, which implies that y_{ss} is the equilibrium associated with u_{ss}. In contrast to linear systems, nonlinear system can have multiple equilibria. These equilibria can be stable or unstable. To learn more about equilibria and how to control nonlinear systems, consider taking "Nonlinear System Theory" next year.

Given a nonlinear differential equation, the equilibria can be found by setting **all** the time-derivatives to zero (i.e., force all outputs/variables to be constant as a function of time) and solving for the states/outputs. Mathematically, given a differential equation $\dot{x} = f(x, u)$, with x being the [vector of] states/outputs and u the [vector of] constant inputs, the equilibrium points are those x's and u's that solve $f(x, u) = 0$, i.e., the roots of the function *f*. As an example, for the differential equation $\ddot{x} + \dot{x} = \sin(x)$, where for simplicity we have assumed no input. The equilibrium is found by solving $\sin(x) = 0$, which yields $x = \pm k\pi$.

Linearization

Nonlinear differential equations do not have transfer functions because Laplace transform cannot be applied to nonlinear terms. What is commonly done is to *linearize* the nonlinear system about an equilibrium point. This results in a linear system, which ***approximates*** the dynamics of the nonlinear system in a neighborhood of the equilibrium point. If the inputs and outputs stay close to the equilibrium, the linearized system behaves very similar to the nonlinear system in the sense that stability properties are the same and the simulated responses will be close.

Given a differential equation $\dot{x} = f(x, u)$, the linearization process involves expanding f in Taylor series and keeping the first order (i.e. linear) terms. If f is composed of many terms added together, linearization can be done one term at a time. A brief review of Taylor series expansion is provided next.

Let's first assume you have a single-variable function $f(x)$. Recall, the first-order approximation of $f(x)$ using Taylor series is given by:

$$f(x) \approx f(x_0) + \left(\frac{df}{dx}\Big|_{x=x_0}\right)(x - x_0)$$

where x_0 is the point around which the series is expanded. Similarly, the expansion of a two-variable function $f(x_1, x_2)$ is:

$$f(x_1, x_2) \approx f(x_{10}, x_{20}) + \left(\frac{\partial f}{\partial x_1}\Big|_{x_{10}, x_{20}}\right)(x_1 - x_{10}) + \left(\frac{\partial f}{\partial x_2}\Big|_{x_{10}, x_{20}}\right)(x_2 - x_{20})$$

where ∂ denotes partial derivative (for example, to compute $\frac{\partial f(x_1, x_2)}{\partial x_1}$ you treat x_2 as a constant and differentiate with respect to x_1). Writing this in vector form, with $x = (x_1, x_2)$, we have the more compact expression:

$$f(x) \approx f(x_0) + \nabla f(x_0)(x - x_0)$$

where ∇ denotes the gradient.

We now apply Taylor series to differential equations. Consider a scalar differential equation

$$\dot{x} = f(x,u)$$

where both x and u are scalars. To linearize this ODE, we expand $f(x,u)$ in Taylor series **around an equilibrium** x_0, u_0 as follows:

$$f(x,u) \approx f(x_0,u_0) + \frac{\partial f}{\partial x}|_{x_0,u_0}(x-x_0) + \frac{\partial f}{\partial u}|_{x_0,u_0}(u-u_0)$$

We note that $f(x_0,u_0)=0$ given the definition of equilibrium. We thus obtain the linearized ODE:

$$\dot{x} = \left(\frac{\partial f}{\partial x}|_{x_0,u_0}\right)(x-x_0) + \left(\frac{\partial f}{\partial u}|_{x_0,u_0}\right)(u-u_0)$$

Note that the terms inside the parentheses are constant coefficients. Note also that all partial derivatives are with respect to the states, not time. The above process generalizes to differential equations with additional inputs and outputs. In general, given a high dimensional ODE $\dot{x} = f(x,u)$, where x and u are both vectors, the linearized equation is given by:

$$\dot{x} = \nabla f_x(x_0,u_0)(x-x_0) + \nabla f_u(x_0,u_0)(u-u_0)$$

where ∇f_x is the gradient with respect to x and ∇f_u is the gradient with respect to u. As an example, consider again the equation $\ddot{x} + \dot{x} = \sin(x)$. Linearizing this equation at the equilibrium $x_0 = 0$ yields:

$$\ddot{x} + \dot{x} = x$$

because $\frac{d}{dx}\sin(x)\,|_{x=0} = 1$. This [linearized] ODE is accurate (i.e. its solution is close to that of the original nonlinear ODE) if x remains close to 0.

Similarly, linearizing the equation around the equilibrium $x_0 = \pi$ yields:

$$\ddot{x} + \dot{x} = -(x-\pi)$$

This ODE remains accurate when x remains close to π. As you can see, the linearized system changes depending on which equilibrium point is chosen for linearization.