

MATLAB Practical

This is a simulation environment for a cruise controller system. The aim of the controller is to keep the vehicle at the velocity demanded by a virtual driver. The controller can use information from the vehicle velocity sensor and send information to the throttle actuator. For simplicity we can assume that negative throttle signals mean braking. Inside the 'simulation loop' you will see the typical functions needed to implement a cruise controller, which are described below.

- The vehicle raw velocity is obtained via the function `getRawVelocity()`. The term 'raw' indicates that this is a binary number. The number of bits used for encoding is specified by the variable `n bits sensor signal`, which is now set to 16. This is to emulate a signal being read from the communication bus. The sensor can measure velocities that range from 0 to 150 mph, which are then encoded into binary numbers linearly, and sent to the bus. To be able to use the velocity sensor reading for your controller, you will find easier to decode the binary value into a base 10 physical value.
- The if-loop is the control loop. In a real ECU this will be likely to be triggered every sampling time by a timer interrupt. Here the timing is known hence the loop is entered every T_s seconds (the chosen sampling time). Inside this loop the following is performed.
 1. The velocity demand signal is obtained from the driver by the function `getRawVelocityDemand()`. Once again, this is a 16-bit encoded binary signal which will have to be decoded to be used easily within the controller. The demand signal ranges from 30 to 110 mph and can only be increased or decreased at steps of 5 mph.
 2. The control signal is calculated by the function `calcController()`. Here you will have to design a feedback controller such that the vehicle's velocity tracks the driver's demand. It would be easier, for code readability, to write this as a separate function. You can design any controller you like but you need to guarantee zero steady state error. I suspect a PID-type of controller would be the easiest option. The range of the control signal is `[-100,100]` signifying the percentage of throttle input (from 0 to 100) and percentage of brake input (from 0 to -100).
 3. The control signal is to be encoded into a binary number (16-bit for the moment) to be sent into the emulated communication bus. You need to write the function for the encoding. Notice that the control raw binary variable must be a string to be used by other functions. Numbers can be converted to strings using `num2str`.
 4. The control signal is sent to the actuator via the function `sendControlCommand()`. It is assumed that there is a delay between the signal being send and the signal reaching the actuator. This is also emulated by this function and the amount of delay can be decided by changing the variable `actuator signal delay`.
- The rest of the code is only needed to visualise the simulated demand signal, velocity response and throttle/actuator effort. Feel free to plot or display any other variable.

Tasks

1. Convert the demand from raw binary into decimal for your controller.

$$X_{Physical} = \frac{X_{Raw} \times (M - m)}{R} + m$$

2. Convert the control output from decimal to Binary.

$$X_{Raw} = \frac{(X_{Physical} - m) \times R}{M - m}$$

3. Create a controller this will need to know the velocity, previous control signal, the demand, the simulation time step, and the relevant gains.