Adaptive Frontlight System (AFS)

Objectives

Model and implement an adaptive vehicle frontlight control system. The main system functionality is to control frontlights rotation angle when the driver is turning the vehicle.

To illuminate the curve more effectively the inner frontlight (i.e., the frontlight located on the turning direction side) moves up to 15 degrees while the outer frontlight moves up to 10 degrees. A different angle of rotation provides a significantly wider field of the view. For sake of simplicity, the frontlights rotation angle is calculated as a linear function of the steering wheel rotation angle.

Other then frontlight angle, light intensity is also controlled (length that each frontlight covers).

Depending on the driving side (left or right) light intensities are determined. Intensity of frontlight closer to drivers from other side (inside light) is always lower then farther frontlight (outside frontlight). Default driving side is right, but it could be changed via specific diagnostic DID.

When battery level is below certain level (30%) system should provide only high beam ON/OFF functionality as battery save mechanism.

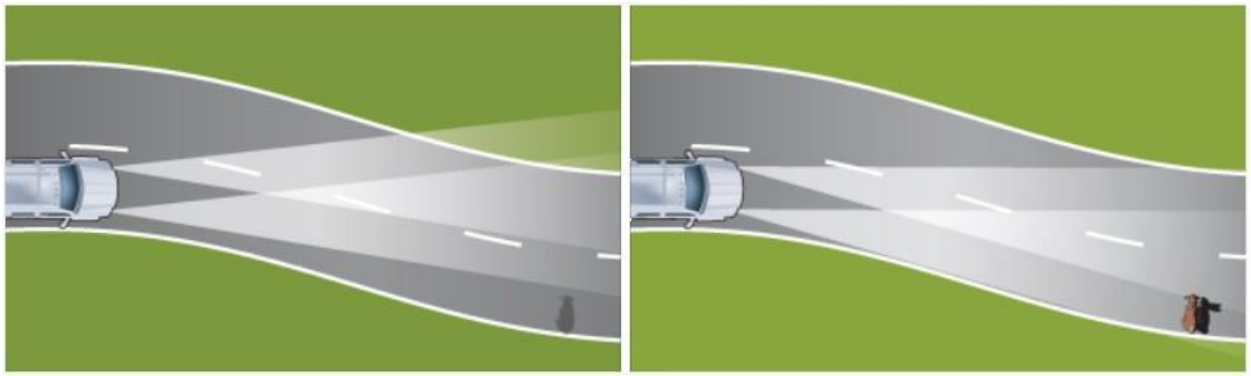

Figure 1- Basic functionality of an adaptive vehicle frontlight control system

Functional requirements

The ECU implements the following SWCs:

〉 CtCddIoHwAb
- Already available, needs to be extended that battery level is read from battery sensor and provided to CtApBatteryManager
- Hardware abstraction SWC
- Handles Dio channels and ports

〉 CtApAFSController
- Already available, control algorithm needs to be extended (consider speed and ambient light, i.e., follow TODOs)
  Based on value in NvM block, about passing line, light intensity is controlled
  Add additional runnable which will handle only high beam ON/OFF functionality. This runnable should be executed when ECU is in BATTERY_LOW mode (hint: use on Mode entry for this runnable)

- Control SWC
- Implements an adaptive vehicle frontlight control algorithm
- Triggered on data reception

〉 CtSaSteeringWheel
- Already available, no changes
- Sensor SWC
- Provides information about steering wheel position to Control SWC
- Time triggered on 100ms

〉 CtSaFrontlight
- Already available, no changes
- Actuator SWC
- Reads and scales frontlight position data from Control SWC
- Scaling factor: `HorizontalAngle * 30 / 200`
- Controls the frontlights (left / right)
- Triggered on data reception

〉 CtSaSpeedometer
- New SWC that needs to be designed and implemented
- Sensor SWC
- Reads data about vehicle speed, value delivered by signal sig_CurrentSpeedSlider, and provide it to CtApAFSController
- Triggered cyclically on 100ms

〉 CtSaAmbientLight
- New SWC that needs to be designed and implemented
- Sensor SWC
- Reads data about ambient light, value delivered by signal sig_AmbientLightBrightness, and provide it to CtApAFSController
- Triggered cyclically on 100ms

〉 CtApBatteryManager
- New SWC that needs to be designed and implemented
  When battery is below 30% set Event_DTC_0x000002 to `DEM_EVENT_STATUS_FAILED` and change Battery mode to BATTERY_LOW
- Sensor SWC
- Reads data about battery status and changes ECU mode accordingly
- Triggered cyclically

Technical details

Frontlights modes:

Vehice speed	**Ambient light**	**Frontlight beam distance**	
		Inside Light	**Outside Light**
0 – 50 km/h	0-5	50m	50m
50 – 120 km/h	0-5	70m	120m
120 – 250 km/h	0-5	140m	200m
120 – 250 km/h	5-10	50m	50m
High beam switch on	0-5	255m	255m
High beam switch on	6-10	255m	255m

Additional Dio ports:

Dio port	Description
env_BatteryLevel	Status of car battery

Battery modes should be implemented with Mode ports:

Mode	Battery level in %	
	From	To
BATTERY_LOW	0	30
BATTERY_MEDIUM*	31	70
BATTERY_HIGH	71	100

* Default mode

Disabled runnables when batter is low (mode BATTERY_LOW):

- RCtApAFSControllerLogic
- RctSaAmbientLightRead
- RctSaSpeedometerRead
- RCtSaSteeringWheelGetPosition

Hint: In trigger tab of runnable there is field Disabled in Modes

Diagnostics, contain Read and Write Did for PassingLane.

Implement DataServices_Diag_RWDI_PassingLane_ReadData server runnable so that passing lane (left side driving or right side driving) can be read via diagnostics.

Implement DataServices_Diag_RWDI_PassingLane_WriteData server runnable to store data about passing lane to NvM (for this purpose corresponding NvM block, PIM buffer and exclusive area should be created).

Additionally, when Event_DTC_0x000002 occurs store current battery value to event memory.

Implementation guidelines

1. Model ECU behavior based on the provided requirements
2. Synchronize Developer and Configurator workspaces
3. Map runnables
4. Create Dio port
5. Take care of workspace errors (if any)
6. Generate source code
7. Implement ECU functionality requirements in C language
8. Test the implementation with CANoe tool