Hanzehogeschool Groningen
University of Applied Sciences
digital society hub

comma.ai

OPENPILOT TAILGATING WARNING

Een bumperkleef detectie- en waarschuwingsservice voor het openpilot systeem

TECHNISCH ONTWERP

Auteur
Jeroen Lammersma

Publicatiedatum
9 juni 2022

Versie
1.0

Auteur
Jeroen Lammersma

E-mailadres
je.lammersma@st.hanze.nl

Studiejaar
Vierde

Studentnummer
362799

Onderwijsinstelling
Hanzehogeschool Groningen

Studie & major
HBO-ICT, Software Engineering

Opdrachtgever
H.M. Groenboom

E-mailadres
h.m.groenboom@pl.hanze.nl

Afstudeerbegeleider
B.L. Heijne

E-mailadres
b.l.heijne@pl.hanze.nl

Bron illustratie voorblad
https://github.com/commaai/openpilot

OPENPILOT TAILGATING WARNING

Een bumperkleef detectie en waarschuwing service voor het openpilot systeem

TECHNISCH ONTWERP

Publicatiedatum
9 juni 2022

Versie
1.0

Hanzehogeschool Groningen University of Applied Sciences | digital society hub

comma.ai

Inhoudsopgave

Figuren- en tabellenlijst

1. Inleiding

In dit document is het technische ontwerp van het project *openpilot Tailgaiting Warning* uitgewerkt. De bumperkleef detectie- en waarschuwingsservice zal geïmplementeerd worden binnen de open-source software *openpilot* (1).

In hoofdstuk 2 zal een objectieve definitie worden opgebouwd van wat er in dit project wordt gezien als 'bumperkleven'. Daarna zal hoofdstuk 3 ingaan op een algemene beschrijving van de gewenste rijcoach. Vervolgens zal er binnen hoofdstuk 4 de services binnen openpilot behandeld worden die belangrijk zijn om de bumperkleef service mee te implementeren. Hoofdstuk 5, communicatie, gaat in op de essentiële communicatie dat nodig is voor de bumperkleef service. Als laatst zal er in hoofdstuk 6 in worden gegaan op het ontwerp van de initiële rijcoach en bumperkleef detectie module.

2. Bumperkleven

Bumperkleven wordt gezien als een langere tijd een te korte volgtijd houden op een voorligger. Deze beschrijving is nog erg ambigu. SWOV (Stichting Wetenschappelijk Onderzoek Verkeersveiligheid) definieert bumperkleven als volgt: *"gedurende langere tijd op een dermate korte volgafstand van je voorganger rijden dat het niet meer mogelijk is om op tijd te stoppen in het geval van een noodstop van die voorganger."* (2).

Toch blijft deze beschrijving ook enigszins onduidelijk en niet voor een eenduidige interpretatie vatbaar. Een objectieve definitie is dus nodig, maar hier blijkt echter geen concreet antwoord voor te zijn. Om deze reden is de hulp van een specialist ingeschakeld om een objectieve definitie vast te leggen.

Alvorens deze zal worden voorgelegd worden eerst twee verkeersveiligheidsindicatoren geïntroduceerd.

2.1. Veiligheidsindicatoren

(Time) headway and time-to-collision zijn nuttige indicatoren om de veiligheid in verkeersituaties te meten (3). Beide indicatoren zullen toegevoegd worden aan openpilot. Echter, voor het detecteren van bumperkleven zal time headway de cruciale indicator zijn.

2.1.1. Time headway (THW)

Time headway (THW) is de volgtijd tot een voorligger, of te wel, hoeveel tijd het kost (in seconden) om met een voertuig hetzelfde punt te passeren als de voorligger op een gegeven moment. THW wordt als volgt gedefinieerd:

> *Time headway, in seconds, between first leadvehicle in same lane as MainTarget and MainTarget. Computed as distance/velocity (in m/s) of MainTarget. Distance is computed as distance along path between rearbumper of leadvehicle and frontbumper of MainTarget. Infinit is velocity of MainTarget = 0.* (4)

Uit deze definitie kan de volgende formule herleid worden:

$$THW = f(distance, velocity) = \begin{cases} \dfrac{distance}{velocity}, & velocity > 0 \\ \infty, & velocity \leq 0 \end{cases}$$

Met deze formule kan dus de volgtijd tot een voorligger berekend worden.

2.1.2. Time-to-collision (TTC)

Time-to-collision (TTC) is de tijd (in seconden) voordat een botsing ontstaat met een voorligger (mits de richtingskoers en snelheidsverschil hetzelfde blijven). TTC wordt als volgt gedefinieerd:

> *Time-to-collision, in seconds, between first leadvehicle in same lane as MainTarget and MainTarget. Computed as distance/relative velocity (in m/s) of MainTarget. Distance is computed as distance along path between rearbumper of leadvehicle and frontbumper of MainTarget. Infinit is velocity of MainTarget = 0. Relative velocity is velocity of MainTarget – velocity of lead vehicle. If relative velocity <= 0, then TTC is computed as Infinit.* (4)

Uit deze definitie kan de volgende formule herleid worden:

$$TTC = f(distance, velocity, relative\ velocity)$$
$$= \begin{cases} \infty, & velocity = 0 \\ \infty, & relative\ velocity \leq 0 \\ \dfrac{distance}{relative\ velocity}, & relative\ velocity > 0 \end{cases}$$

Met deze formule kan dus de tijd tot botsing met een voorligger berekend worden.

2.2. Definitie bumperkleven

Met de hulp van C. Dijksterhuis, specialist op het gebied van verkeersgedrag en mens-machine interactie, is een definitie van bumperkleven vastgesteld (5). Goed om te benoemen is dat dit gaat om een algemene definitie, zonder rekening te houden met externe factoren; in principe zou namelijk ook rekening gehouden moet worden met onder andere: de remweg, weers- en lichtomstandigheden, reactie snelheid van de bestuurder, massa van het voertuig, et cetera. Al deze extra variabelen zouden voor een dermate complexiteit zorgen dat er is besloten om dit buiten de scope van dit project te houden.

Definitie

De definitie is als volgt vastgelegd:

> *Tailgating between first leadvehicle in same lane as MainTarget and MainTarget. Computed by looking at the value of THW (in s) between leadvehicle and MainTarget. If THW < 1, then tailgating = 1, else tailgating = 0. If velocity of MainTarget < minimal required velocity (5 m/s), then tailgating = 0.*

Er wordt vanuit gegaan dat THW nooit kleiner kan zijn dan nul. Volgens de definitie zal een auto altijd bumperkleven als de THW waarde binnen nul en één valt, ongeacht de duur. Om hier toch rekening mee te houden wordt er in paragraaf 2.3 een waarschuwingsniveau geïntroduceerd.

Formule

Uit de definitie kan de volgende formule herleid worden:

$$tailgating = f(THW, velocity) = \begin{cases} 0, & velocity < 5 \\ 0, & THW \geq 1 \\ 1, & THW < 1 \end{cases}$$

Met deze formule is het dus mogelijk om te achterhalen of een bestuurder aan het bumperkleven is.

2.3. Waarschuwingsniveau

Om rekening te houden met de duur van bumperkleven wordt er een waarschuwingsniveau (warningLevel) toegevoegd. Het doel is om hiermee de ernst van het bumperkleven aan te duiden. Het niveau wordt berekend aan de hand van de duur (duration) in seconden met de volgende formule:

$$warningLevel = f(duration) = \begin{cases} 0, & duration < 5 \\ 1, & 5 \leq duration < 10 \\ 2, & 10 \leq duration < 20 \\ 3, & duration \geq 20 \end{cases}$$

Niveau nul is het laagst en drie is het hoogst. Het berekende waarschuwingsniveau kan worden gebruikt voor het stapsgewijs intensiveren van de auditieve en/of visuele waarschuwingen. Dit wordt toegelicht in paragraaf 6.5.

2.4. Waarschuwingsteken

Om de bestuurder bewust te maken van (gevaarlijk) bumperkleef gedrag is er een waarschuwingsteken ontworpen (figuur 1). Het teken kan zichtbaar worden gemaakt op de user interface van openpilot.

Figuur 1: Ontworpen waarschuwingsteken voor bumperkleven.

Op het teken zijn twee auto's zichtbaar. De linker representeert de bestuurder en de rechter de voorligger. Tussen de auto's is 'frictie' zichtbaar en de voorligger geeft met het uitroepteken aan dat hij of zij de huidige situatie als onwenselijk ervaart.

Hoe het teken op de user interface zal worden gepresenteerd wordt toegelicht in paragraaf 6.6.

3. Rijcoach

Eén van de lange termijn doelen van de opdrachtgever is om een (proof-of-concept) rijcoach te ontwikkelen met openpilot. Er is besloten om tijdens dit project hier rekening mee te houden. Praktisch gezien zal dit betekenen dat het fundament van de rijcoach gelegd zal worden zodat hier gemakkelijk op door ontwikkeld kan worden. De volgende paragrafen zullen concreter ingaan op de manier hoe dit wordt gerealiseerd.

3.1. Nieuw berichttype

Er zal een nieuw berichttype in het leven worden geroepen waarin alle data dat verstuurd wordt vanuit de rijcoach in geplaatst kan worden. Zo zijn alle gegevens op één geconcentreerde en eenduidige wijze beschikbaar. Tegelijkertijd is dit makkelijk uitbreidbaar, zodat het eenvoudig is om hier vervolgens nieuwe data aan toe te voegen. Dit wordt verder toegelicht in paragraaf 0.

3.2. Rijcoach service

Naast een nieuwe berichttype zal er ook een nieuwe service aan openpilot worden toegevoegd. Ook is hier het doel om zoveel mogelijk zaken van de rijcoach op één plek te centreren. In dit geval is dat de logica. Er zal wederom rekening gehouden worden met uitbreidbaarheid: elke nieuwe ‘detector’ of andere functionaliteit moet makkelijk toegevoegd kunnen worden aan deze service. Verdere toelichting is te vinden in paragraaf 4.2.

3.3. Modules

Nieuwe functionaliteiten toevoegen aan de rijcoach zal gaan middels modules. Hiervoor zal een abstracte *base* klasse worden aangemaakt die elke module moet implementeren. Het doel hiervan is om het implementeren van nieuwe functionaliteiten aan de rijcoach te generaliseren en te versimpelen. Details hierover zijn te vinden in paragraaf 6.2.4.

4. Services

openpilot bestaat uit verschillende services die met elkaar communiceren. Alle services worden beheerd via een *manager* (zie paragraaf 4.3). Voorbeelden van services zijn:

- *camerad*, voor het managen en verwerken van de camera beelden;
- *modeld*, voor het voorspellen waar en hoe te rijden door middel van een neuraal netwerk;
- *plannerd*, voor berekenen van laterale- (sturen) en longitudinale planning (gas/rem).

Voor een completer en gedetailleerder overzicht zie de blog post *How Does openpilot Work* op de website van comma.ai (6). Hoe de communicatie tussen deze services verloopt zal in hoofdstuk 5 aan bod komen.

4.1. Belangrijke services

controlsd, *radard* en *ui* zijn met betrekking tot bumperkleef detectie en waarschuwing de belangrijkste services; hiermee zal namelijk informatie mee gewisseld worden.

4.1.1. controlsd

controlsd stuurt de auto daadwerkelijk aan. Het ontvangt het plan van *plannerd* en zet deze om in acceleratie en stuur signalen. Vervolgens worden deze signalen omgezet naar CAN berichten die verstuurd worden naar de auto. Ook verwerkt deze service ruwe CAN berichten gestuurd door de auto en publiceert deze in een standaard formaat.

Eén van deze CAN berichten bevat de snelheid van de auto. *controlsd* verwerkt deze snelheid en publiceert deze in een bericht. Verdere toelichting is te vinden in paragraaf 5.2. Om de snelheid van de auto te achterhalen zal *controlsd* dus geraadpleegd moeten worden.

Verder is *controlsd* verantwoordelijk voor het afhandelen van events. Een event kan zowel auditief als visueel van aard zijn. Events maakt het dus mogelijk om de bestuurder op bumperkleven te waarschuwen op de standaard manier binnen openpilot.

4.1.2. radard

radard verwerkt de ruwe data van de fysieke radars. Het berekent onder andere de afstand tot een voorligger, de relatieve snelheid ten opzichte van een voorligger en Forward Collision Warning (FCW). Naast *radard* voorspelt ook een machine learning model deze waarden en heeft het zelfs de mogelijkheid om dit gelijktijdig voor twee voertuigen uit te voeren. *radard* ontvangt ook deze data en vanaf een bepaalde kans (50%) verwerkt en publiceert de service de data van het model in plaats van de fysieke radars.

Deze service zal verantwoordelijk worden voor het berekenen van de THW en TTC. Deze waarden zullen uiteindelijk toegevoegd worden aan het bericht dat *radard* verstuurd. Details hierover zijn te vinden in paragraaf 5.1.2.

4.1.3. ui

De *ui* (user interface) service is verantwoordelijk voor alles wat er op het scherm getoond wordt. Als de auto is gestart zijn de beelden zichtbaar van één van de camera's

die op de weg gericht is. Over deze beelden worden onder andere de snelheid, rijstrooklijnen, het rij plan en voorliggers geprojecteerd. Een voorbeeld hiervan is te vinden als figuur 2.

Figuur 2: De user interface van openpilot wanneer de auto is gestart.

Om ervoor te zorgen dat het waarschuwingsteken getoond zal worden wanneer nodig moet deze service aangepast worden. Uiteraard zorgt de *ui* er ook voor dat visuele events zichtbaar worden. Hier zijn echter geen aanpassingen binnen de service voor nodig.

4.2. coachd

Voor de rijcoach zal een nieuwe service in het leven worden geroepen: *coachd*. Initieel zal deze service enkel informatie verschaffen over de actuele bumperkleef status. De service wordt verantwoordelijk voor het bepalen of er sprake is van bumperkleven en hier zodoende een gepast waarschuwingsniveau aan te koppelen. Het wisselt informatie met de zojuist genoemde services: *controlsd*, *radard* en *ui*.

Daarbij zal de service zodanig ontworpen worden dat het gemakkelijk uit te breiden valt. Het ontwerp van de *coachd* service is te vinden in hoofdstuk 6.

4.3. Manager

De *manager* service is verantwoordelijk voor het starten en stoppen van services en het controleren of services niet gecrasht zijn als openpilot is gestart. Om services te starten bestaan er binnen openpilot drie soorten processen:

- *DaemonProcess*, voor Python processen die moeten blijven draaien tijdens het opnieuw opstarten van de manager;
- *NativeProcess*, voor services geschreven in C++;
- *PythonProcess*, voor services geschreven in Python.

DeamonProcess wordt alleen gebruikt voor de service *athena* (om ssh connectie te behouden), alle andere services zijn toegevoegd als een *NativeProcess* of een *PythonProcess*.

Voor *coachd* hoeft dus alleen een afweging worden gemaakt in welke programmeertaal de service geschreven zal worden. Verder betekend dit dat *manager* aangepast moet worden zodat het ook deze nieuwe service start en stopt tijdens het opstarten en afsluiten van openpilot.

5. Communicatie

De verschillende services communiceren met elkaar door middel van een pub-sub framework: *cereal*. Services kunnen abonneren op topics (onderwerpen) en tegelijkertijd topics publiceren. Binnen de openpilot repository is cereal een submodule.

Voor de bumperkleef detectie en waarschuwing zal er een nieuwe topic geïntroduceerd worden en worden bestaande topics aangepast.

5.1. cereal

cereal is zowel een berichtenspecificatie als *"generieke high-performance inter process communication"* door middel van het publish subscribe model (7). Het is voor elke service mogelijk om op elke topic te abonneren, maar iedere topic kan slecht één publisher (uitgever) hebben.

Cap'n Proto

Berichten worden opgeslagen en verstuurd middels het *Cap'n Proto* formaat, wat zorgt voor enorm hoge performance. *"Denk aan JSON, maar dan binair"* is wat er wordt genoemd (8). Op het hoogste niveau is ieder bericht van het type *Event*. Binnen het bericht is ruimte voor data van een willekeurige topic (toegelicht in paragraaf 5.1.1). De structuur van elke topic is vooraf gedefinieerd als een Cap'n Proto struct, waarbij elk veld bestaat uit een naam, nummer en type. Alle berichtdefinities zijn te vinden in het bestand *log.capnp* (cereal/log.capnp) (9).

Een uitgebreide uitleg over de werking van cereal binnen openpilot is te vinden op de openpilot wikipagina: *Introduction to openpilot* (10).

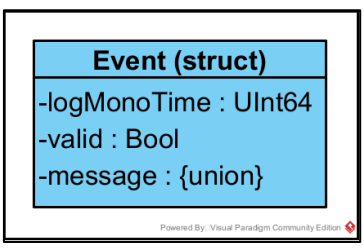
5.1.1. Event struct

Elk bericht wordt verstuurd dat verstuurd wordt is van het datatype *Event* en bevat drie velden:

- *logMonoTime*, de timestamp van het bericht;
- *valid*, om de geldigheid van het bericht mee aan te geven;
- *message*, die ruimte biedt voor data van één van de topics.

Het *message* veld is een union van alle beschikbare topic definities: hierin staan dus alle berichten die verstuurd kunnen worden, gedefinieerd met naam en type (zie het *radarState* bericht in figuur 4 voor een concreet voorbeeld).

Elke instantie van *Event* bevat data van een enkele topic, of te wel, het *message* veld bevat dus precies één bericht van één topic. In figuur 3 is de definitie van *Event* weergegeven.

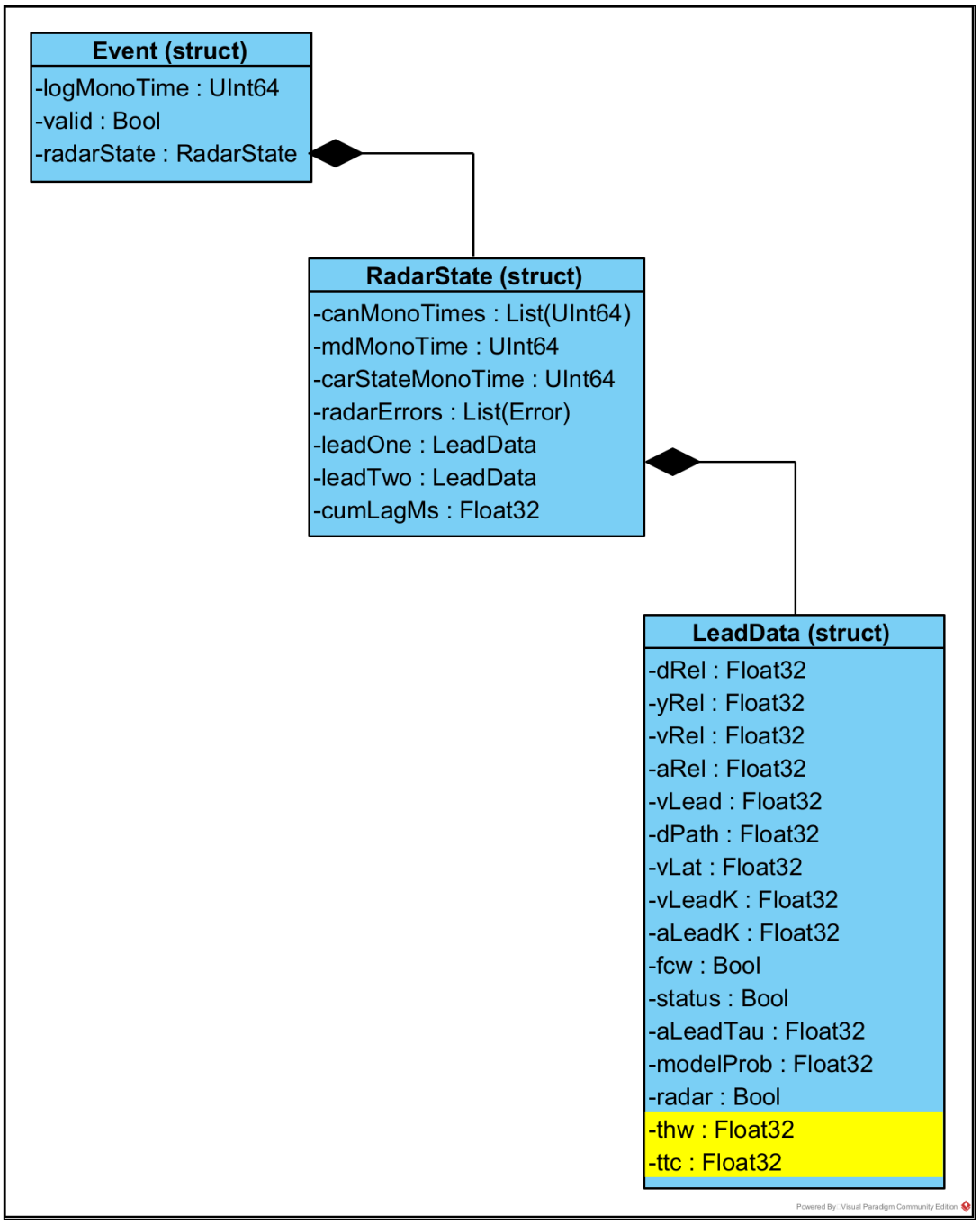


Figuur 3: Definitie van de Event struct.

Omdat er een nieuwe topic wordt toegevoegd aan openpilot (*drivingCoachState*, zie paragraaf 0) zal de definitie van deze struct toegevoegd moeten worden aan de zojuist benoemde union.

5.1.2. RadarState (THW en TTC)

Zoals beschreven in paragraaf 4.1.2 worden THW en TTC toegevoegd aan het reeds bestaande bericht dat *radard* publiceert: *RadarState*. Deze struct bevat twee velden waarin informatie wordt opgeslagen over potentiële voorliggers: *leadOne* en *leadTwo* van het type *LeadData*. Deze struct slaat informatie op zoals afstand tot voorligger (dRel), relatieve snelheid tot voorligger (vRel), et cetera. THW en TTC worden hieraan toegevoegd als additionele velden.



Figuur 4: Definitie van de RadarState struct met de onderliggende LeadData struct binnen de Event struct. Geel gemarkeerd zijn de nieuw toegevoegde velden thw en ttc.

RadarData wordt als *message* geplaatst in *Event*. Dit is mogelijk omdat dit berichttype is toegevoegd is aan de union van *Event* (zoals beschreven in paragraaf 5.1.1).

In figuur 4 is de definitie van *RadarState* weergegeven.

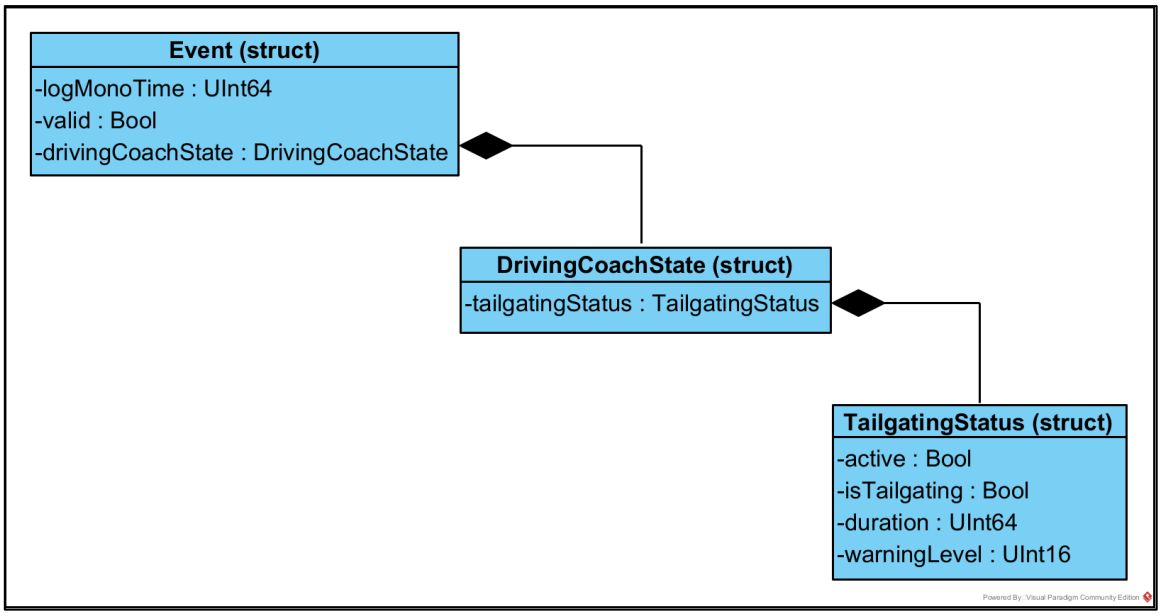
5.1.3. DrivingCoachState

Voor de rijcoach en bumperkleef detectie en waarschuwing wordt er een nieuwe topic geïntroduceerd: *drivingCoachState*. Tegelijkertijd zal hiervoor een nieuwe struct worden gedefinieerd met dezelfde naam: *DrivingCoachState*. *drivingCoachState* wordt als veld toegevoegd aan de union van *Event* en krijgt *DrivingCoachState* als type.

DrivingCoachState biedt de basis waarin de data van de rijcoach in kan worden geplaatst. Echter, op dit moment zal het enkel data bevatten over de actuele status van bumperkleven in het veld *tailgatingStatus*. Hiervoor wordt *TailgatingStatus* als tweede, nieuwe, struct gedefinieerd. Deze bevat vier velden:

- *active*, om aan te geven of de module actief is;
- *isTailgating*, om aan te geven of er sprake is van bumperkleven;
- *duration*, waarin de duur van het bumperkleven in wordt gezet;
- *warningLevel*, om het actuele waarschuwingsniveau mee aan te duiden.

In figuur 5 is de definitie van *DrivingCoachState* weergegeven.



Figuur 5: Definitie van de DrivingCoachState struct met de onderliggende TailgatingStatus struct binnen de Event struct.

drivingCoachState zal uiteidelijk door de *coachd* service gepubliceerd worden. Verdere toelichting is te vinden in hoofdstuk 6.

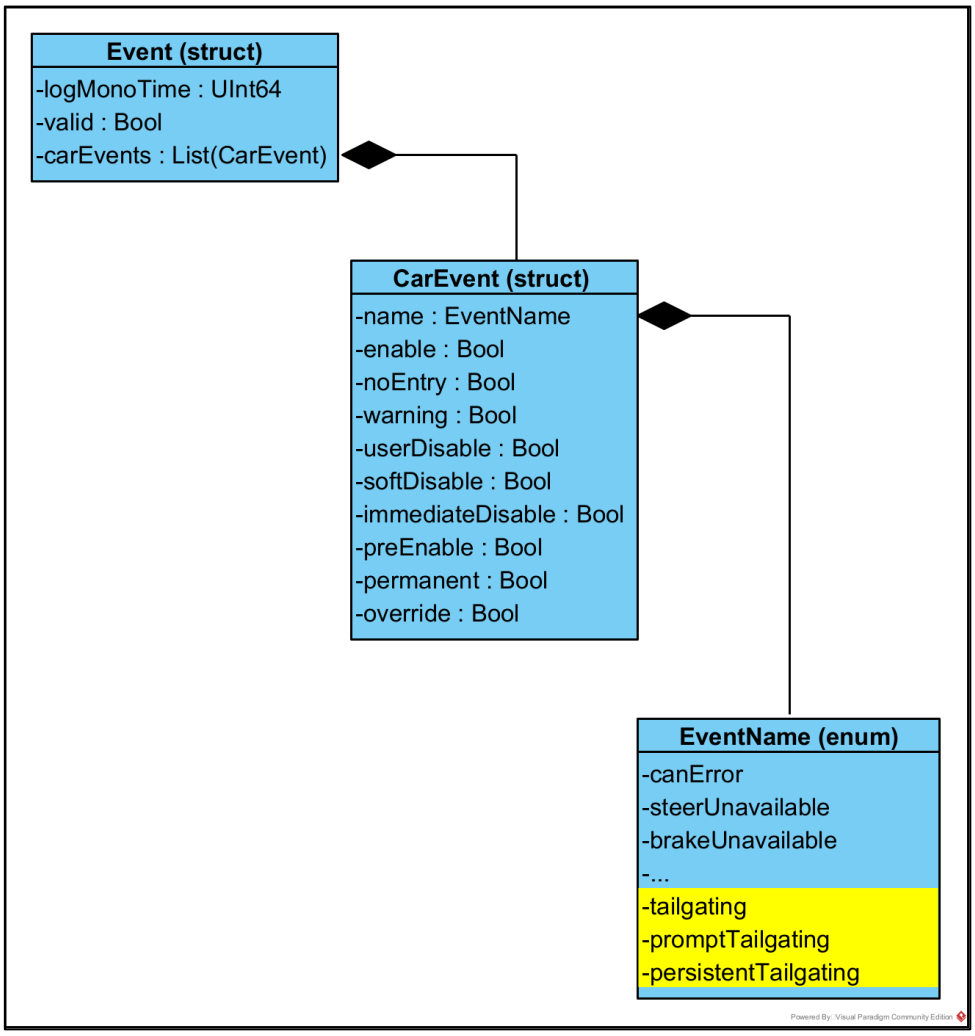
5.1.4. CarEvent struct

Om auditieve en/of visuele waarschuwingen (*carEvents*) af te vuren binnen openpilot is het nodig om deze te vermelden in de *EventName* enum binnen de *CarEvent* struct. Deze is te vinden in het bestand *car.capnp* (cereal/car.capnp) (11).

Voor het waarschuwen op bumperkleven worden drie events aangemaakt: *tailgating*, *promptTailgating* en *persistentTailgating*. Deze moeten dus ook toegevoegd worden

aan *EventName*. Details over de implementatie van deze events zijn te vinden in paragraaf 6.5.

In figuur 6 is de definitie van *CarEvent* weergegeven.



Figuur 6: Definitie van de CarEvent struct met de onderliggende EventName enum binnen de Event struct. De EventName enum bestaat uit 100+ namen waarvan het grootste deel is weggelaten. Geel gemarkeerd zijn de nieuw toegevoegde events tailgating, promptTailgating en persistentTailgating.

CarEvent hoeft niet verder behandeld te worden wat betreft communicatie; *controlsd* is namelijk verantwoordelijk over de logica van het afvuren van *carEvents* binnen openpilot en andere services hoeven hier niet direct mee te communiceren.

5.2. Essentiële communicatie

Binnen openpilot vindt er ontzettend veel communicatie plaats tussen de verschillende services. Echter, de meeste hiervan is niet van belang voor de bumperkleef detectie en waarschuwing. Deze paragraaf zal daarom enkel de essentiële communicatie beschrijven die daarvoor nodig is.

5.2.1. Overzicht berichten

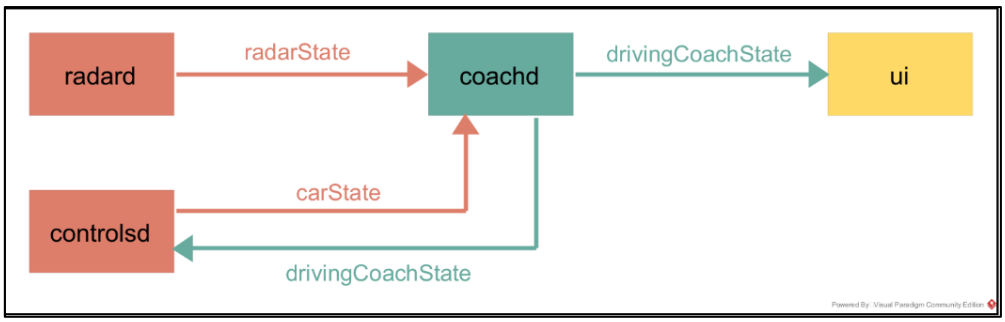
Allereerst volgt een overzicht van de communicatie tussen de *coachd*, *controlsd*, *radard* en *ui* services. Deze services spelen allemaal een rol in de bumperkleef detectie en

waarschuwing. In tabel 1 is van elke service zichtbaar op welke berichten het abonneert en welke het publiceert.

Tabel 1: Overzicht van de berichten waarop de coachd, controlsd, radard en ui services op abonneren en welk bericht ieder publiceert.

Service	Geabonneerd op	Publiceert
coachd	carState radarState	drivingCoachState
controlsd	drivingCoachState	carState
radard	-	radarState
ui	drivingCoachState	-

Een visuele weergave van tabel 1 is te vinden als figuur 7.



Figuur 7: Communicatie tussen de coachd, controlsd, radard en ui services.

Tabel 1 en figuur 7 maken zichtbaar dat *coachd* berichten ontvangt van *controlsd* en *radard*. Verder wordt duidelijk dat *controlsd* en *ui* juist berichten ontvangen van *coachd*.

5.2.2. Gebruikte velden

De meeste van de zojuist benoemde berichten bevatten veel velden, maar ze zijn niet allemaal nodig voor bumperkleef detectie en waarschuwing. In tabel 2 is van elk bericht te zien welke velden gebruikt zullen worden, wat het type is van ieder en voor welk doel het veld nodig is.

Tabel 2: Velden van de berichten gebruikt bij bumperkleef detectie en waarschuwing.

Bericht	Veld(en)	Type	Beschrijving en doel
carState	vEgo	Float32	Snelheid ego. Bepalen of snelheid vereiste minimum overschrijdt.
drivingCoachState	warningLevel	UInt16	Waarschuwingsniveau bumperkleven. 1. controlsd: bepalen of een CarEvent afgevuurd moet worden. 2. ui: bepalen of het bumperkleef icoon zichtbaar moet zijn.
radarState	leadOne / leadTwo	LeadData	Voorligger. Bepalen of ego aan het bumperkleven is op de dichtstbijzijnde voorligger.
	LeadData.dRel	Float32	Afstand tot voorligger. Bepalen van de dichtstbijzijnde voorligger.
	LeadData.thw	Float32	Volgtijd. Bepalen of volgtijd tussen 0 en de ingestelde drempelwaarde ligt.

6. Rijcoach en bumperkleef detectie en waarschuwing

De rijcoach wordt als nieuwe service toegevoegd aan openpilot. Detectors, zoals de bumperkleef detectie, worden als modules toegevoegd. Dit maakt de rijcoach makkelijk uitbreidbaar. De THW en TTC berekeningen toegevoegd aan de *radard* service om de bumperkleef detectie mogelijk te maken. Verder moet de service vermeld worden in cereal en moeten er waarschuwingen worden afgevuurd wanneer de bestuurder aan het bumperkleven is. Als laatst moeten er worden gekeken hoe het waarschuwingsteken op de user interface zichtbaar kan worden gemaakt.

6.1. Aanpassingen aan radard

Om ervoor te zorgen dat de nieuwe velden in het *RadarState* bericht ook daadwerkelijk gevuld worden met waarden moeten er wat aanpassingen plaatsvinden in de *radard* service. De THW en TTC formules, beschreven in paragraaf 2.1, moeten als functies toegevoegd worden. Daarnaast moeten deze functies gebruikt worden om de *thw* en *ttc* velden van waarden te voorzien.

Alle aanpassingen zullen plaatsvinden in het *radar_helpers.py* bestand (openpilot/selfdrive/controls/lib/radar_helpers.py) (12). Het gaat om kleine, eenvoudige, veranderingen en daarom zijn er geen diagrammen gemaakt: de aanpassingen zullen tekstueel beschreven worden.

6.1.1. THW en TTC functies

Om THW en TTC te berekenen worden er twee nieuwe functies toegevoegd: *calculate_thw* en *calculate_ttc*. In figuren 8 en 9 zijn de declaraties van deze functies gegeven.

```
calculate_thw(d_rel: float, v_ego: float) : float
```

Figuur 8: Declaratie van de functie calculate_thw.

```
calculate_ttc(d_rel: float, v_ego: float, v_rel: float) : float
```

Figuur 9: Declaratie van de functie calculate_ttc.

6.1.2. Track klasse

Binnen de klasse *Track* moet de functie *update* aangepast worden. De functie moet twee nieuwe attributen toevoegen: *thw* en *ttc*. De functies *calculate_thw* en *calculate_ttc* worden hierbij aangeroepen om ze van waarden te voorzien.

6.1.3. Cluster klasse

De *Cluster* klasse bevat een lijst met meerdere *Track* instanties: *tracks*. Ten eerste moeten er twee nieuwe properties toegevoegd worden aan de *Cluster* klasse: *thw* en *ttc*. Elke property berekend de gemiddelde *thw* of *ttc* waarde van de *tracks* (afhankelijk van de property) en geeft deze terug. Ten tweede moeten de *thw* en *ttc* velden toegevoegd worden aan de dictionary die de functie *get_RadarState* teruggeeft. Deze velden worden met de *thw* en *ttc* properties van waarden voorzien.

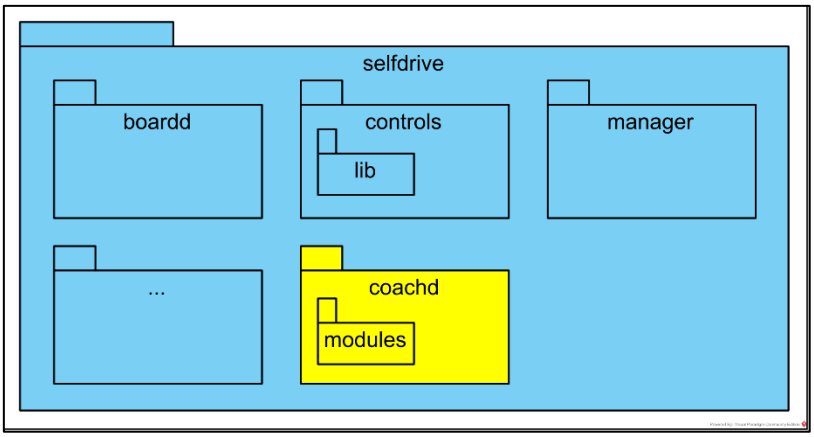
Als laatste moeten de *thw* en *ttc* velden toegevoegd worden aan de dictionary die de functie *get_RadarState_from_vision* teruggeeft. Deze dictionary wordt gevuld met waarden uit *lead_msg*, afkomstig uit het machine learning model. Hierbij worden de *calculate_thw* en *calculate_ttc* functies gebruikt om de velden van waarden te voorzien.

6.2. coachd

De *coachd* service wordt toegevoegd als subpackage aan de selfdrive package. Daarnaast wordt er een thread functie geïmplementeerd en worden de klassen *CoachD* en *CoachModule* in het leven geroepen.

6.2.1. Subpackage selfdrive

Alle hoofdcomponenten en -services (zoals *boardd, controls*, *manager*, et cetera) zijn als subpackages geplaatst in de *selfdrive* package (openpilot/selfdrive) (13). *coachd* zal hier ook als subpackage aan toegevoegd worden. *coachd* bevat zelf ook een subpackage voor de rijcoach modules.



Figuur 10: Structuur van de selfdrive package. Niet alle subpackages zijn weergegeven. In het geel zichtbaar is de nieuw toegevoegde subpackage coachd met daarbinnen de subpackage modules.

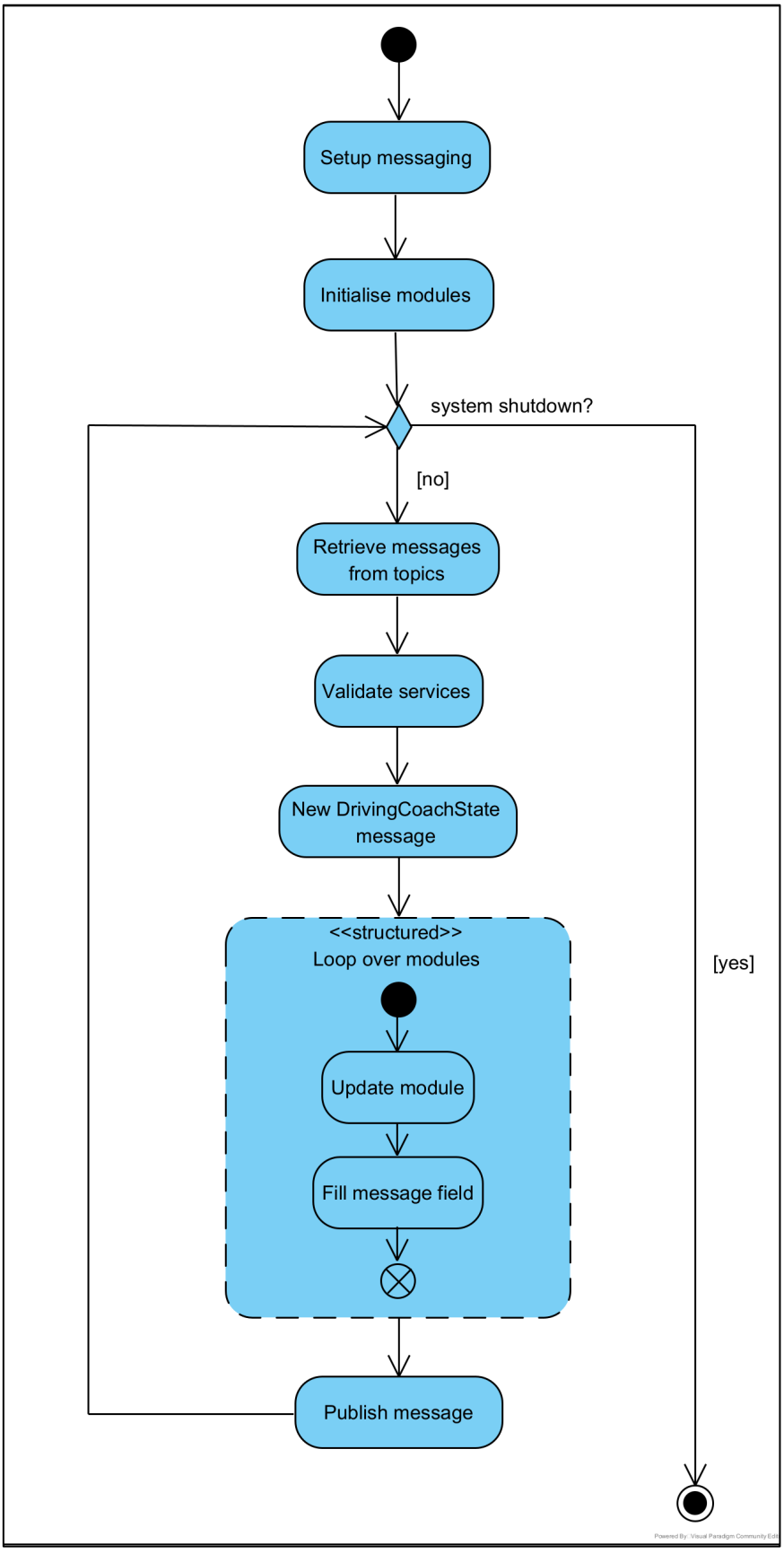
6.2.2. coachd thread

Iedere service draait in een eigen thread. Deze blijft continu draaien, totdat het systeem wordt uitgeschakeld. Een thread wordt gestart en gestopt door de *manager* (beschreven in paragraaf 6.2.6). Voor *coachd* wordt uiteraard ook een eigen thread aangemaakt. Deze moet de volgende stappen uitvoeren:

1) Stel de communicatie in (cereal). *coachd* moet abonneren op de berichten *carState* en *radarState* en publiceert het bericht *drivingCoachState*.
2) Initialiseer de rijcoach modules.
3) Haal de meest recente berichten van *carState* en *radarState* op.
4) Valideer de status van de services die *carState* en *radarState* publiceren.
5) Maak een nieuwe instantie aan van het *DrivingCoachState* bericht.
6) Itereer over de rijcoach modules. Bij elke iteratie wordt:
   a) de module geüpdatet (nieuwe status opgevraagd);
   b) het bijbehorende veld in *driverCoachState* met de nieuwe status gevuld.
7) Publiceer *drivingCoachState* en ga terug naar stap 3.

Figuur 11 geeft de *coachd* thread weer aan de hand van een activiteitendiagram.

Momenteel worden alleen de statussen van *carState* en *radarState* gevalideerd. Ook dit kan wijzigen op het moment dat de rijcoach nieuwe modules gaat bevatten.



Figuur 11: Activiteitendiagram van de coachd thread.

6.2.3. Standaard modules

Een globale variabele *COACH_MODULES* (van het datatype Dictionary) zal gebruikt worden als standaardwaarde om de modules van *CoachD* mee te initialiseren. De key moet overeen komen het veld in de *DrivingCoachState* struct die gekoppeld is aan de module. De value is een verwijzing naar de klasse van de module (toegelicht in paragraaf 6.2.5).

6.2.4. CoachD klasse

Iedere service binnen openpilot is geïmplementeerd als klasse en dus zal er bij *coachd* dit patroon gevolgd worden. De klasse krijgt de naam *CoachD* en is in principe redelijk eenvoudig: het bevat één attribuut, een constructor en twee methoden. Een klasse diagram van *CoachD* is te vinden als figuur 12.

Attributen

CoachD bevat één attribuut: *modules*. Hier worden alle (geactiveerde) rijcoach modules in gezet. De keys komen overeen met velden van het *DrivingCoachState* bericht. De values zijn van het type *CoachModule* (toegelicht in paragraaf 6.2.5).

Constructor

De constructor heeft één optionele parameter: *modules*. Het datatype is een Dictionary en moet aan de zelfde eisen voldoen als bij de *COACH_MODULES* variabele. Wanneer hier geen argument aan meegegeven wordt zal *CoachD* de *modules* attribuut initialiseren met *COACH_MODULES*.

Het doel van deze optionele parameter is om handmatig de modules mee te geven die *CoachD* moet updaten. Hiermee is het mogelijk om bepaalde modules als het ware te deactiveren: velden binnen het *DrivingCoachState* bericht zullen dan niet gevuld worden (ze blijven '0' of zijn leeg). Verder is het hiermee mogelijk om bijvoorbeeld een aangepaste of experimentele variant van een bestaande module uit te proberen en maakt dit *CoachD* makkelijker te testen.

Methoden

Tabel 3 geeft een overzicht van de methoden van de *CoachD* klasse weer.

Tabel 3: Methoden van de CoachD klasse.

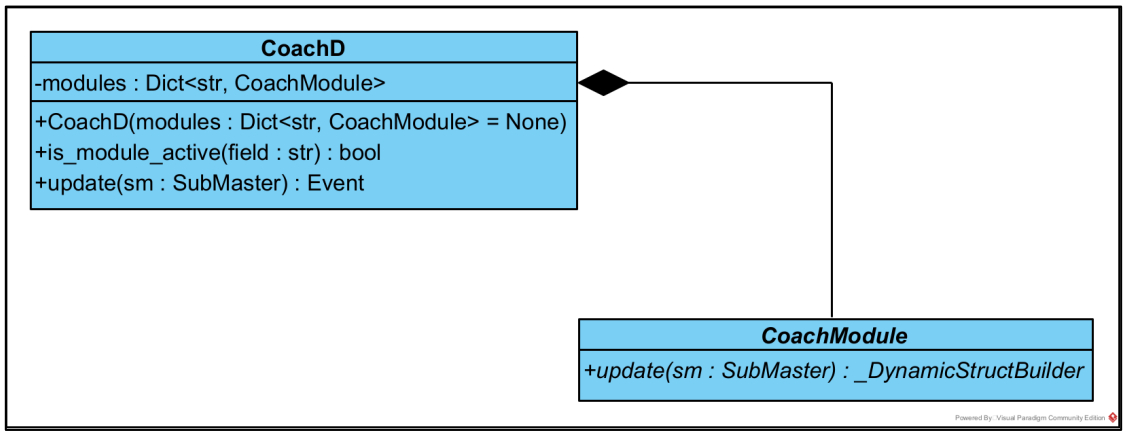
Methode	Doel
is_module_active	Geeft terug of een gegeven module geactiveerd is.
update	Via de *SubMaster* worden de *carState* en *radarState* berichten ontvangen. De *update* methode is verantwoordelijk voor het creëren van een *drivingCoachState* bericht. Het doel van de methode is om over alle modules te itereren en van elk de *update* methode aan te roepen. Op deze manier worden alle velden van *drivingCoachState* gevuld.

6.2.5. CoachModule klasse

CoachModule is een abstracte klasse die geïmplementeerd moet worden door iedere rijcoach module. Het bevat een enkele methode *update* die een *SubMaster* object

ontvangt. Het geeft een *_DynamicStructBuilder* object terug. In paragraaf 6.3.1 is een concreet voorbeeld van een geïmplementeerde *CoachModule* klasse te vinden.

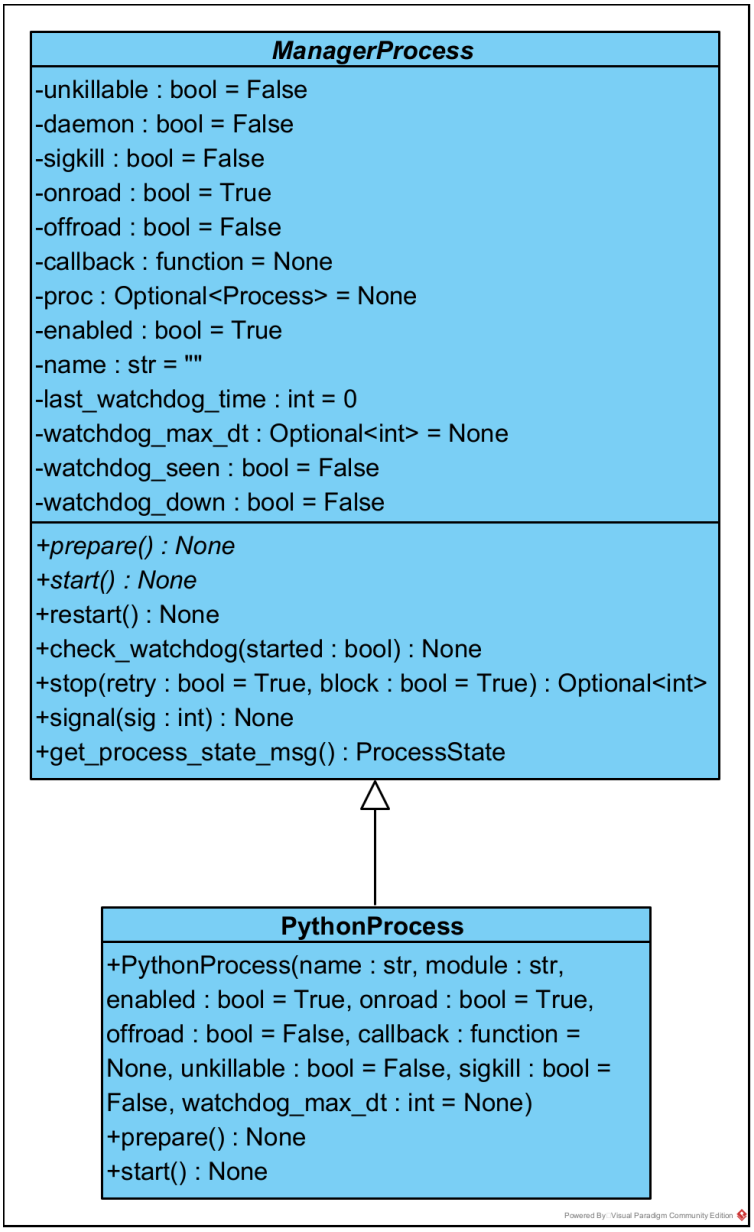
Figuur 12 bevat een klasse diagram van *CoachModule*.



Figuur 12: Definities van de klasse CoachD en abstracte klasse CoachModule.

6.2.6. Manager

Omdat de rijcoach geen tijd kritische service is en om het ontwikkelwerk wat eenvoudiger te houden, is er besloten om *coachd als PythonProcess* toe te voegen. *PythonProcess* is een implementatie van de abstracte klasse *ManagerProcess*. Ze zijn te vinden in het bestand *process.py* (openpilot/selfdrive/manager/process.py) (14). Figuur 13 geeft een beschrijving van deze, redelijk uitgebreide, klassen.



Figuur 13: Definities van de abstracte klasse ManagerProcess en de geïmplementeerde klasse PythonProcess.

Volledige kennis van de klassen is niet nodig, omdat alle logica door de *manager* wordt afgehandeld. Ook zijn veel parameters voor specifieke doeleinden bedoeld die niet van toepassing zijn op *coachd* (en hier ook niet zullen worden behandeld).

coachd service

In tabel 4 is overzicht te vinden van de parameters die van belang zijn voor *coachd* en met welke waarden de argumenten worden voorzien. Voor de overige argumenten kunnen de standaardwaarden aangehouden worden.

PythonProcess instantie

Vervolgens moet er een instantie van een *PythonProcess* met de hierboven genoemde argumenten toegevoegd worden aan de lijst van services die *manager* beheerd. Deze lijst variabele heeft de naam *procs* en is te vinden in het bestand *process_config.py* (openpilot/selfdrive/manager/process_config.py) (15).

Tabel 4: Parameters die van belang zijn voor coachd met beschrijving en waarde van het argument.

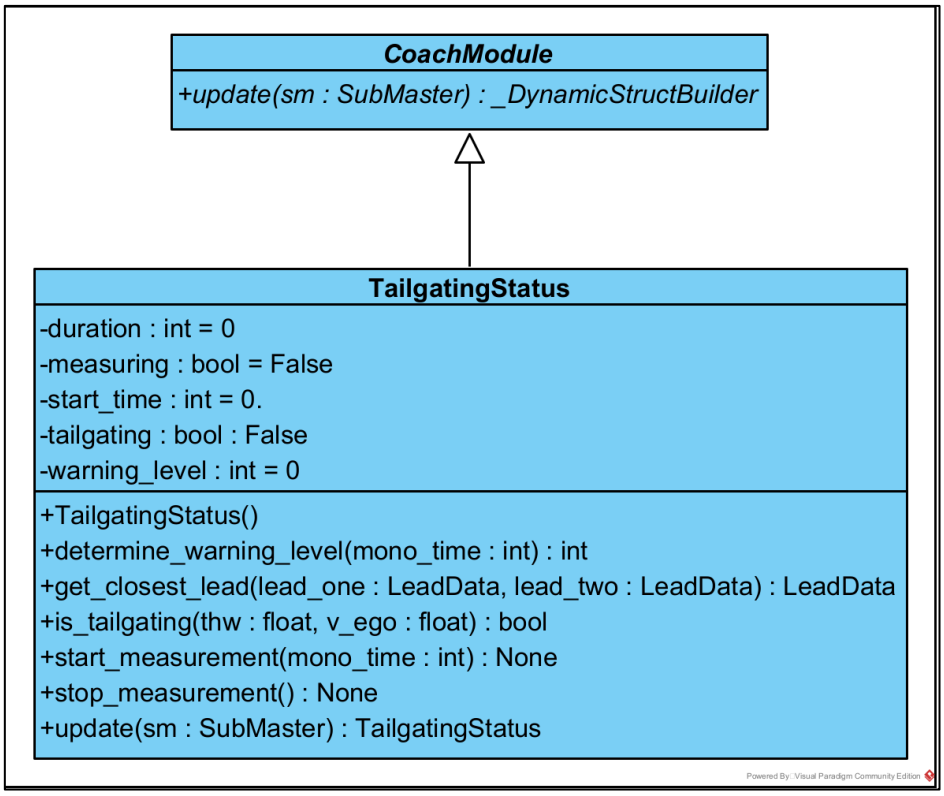
Parameter	Beschrijving	Argument
name	Naam van de service.	“coachd”
module	Pad naar de main functie van de service.	“selfdrive.coachd.coachd”
onroad	Om aan te geven of er gecontroleerd moet worden of de service draait zodra de auto is gestart. Deze check wordt continu uitgevoerd, totdat de auto wordt uitgezet.	True (standaardwaarde)
offroad	Om aan te geven of er gecontroleerd moet worden of de service draait tijdens het opstarten van openpilot. Deze check wordt eenmalig uitgevoerd.	False (standaardwaarde)

6.3. Bumperkleef detectie module

De bumperkleef detectie module wordt geïmplementeerd als *TailgatingStatus* klasse die afgeleid is van de abstracte klasse *CoachModule*. Deze wordt toegevoegd aan de *modules* dictionary van *CoachD* (voor details, zie paragraaf 6.2.4).

6.3.1. TailgatingStatus klasse

Figuur 14 geeft een definitie weer van de *TailgatingStatus* klasse.



Figuur 14: Definities van de abstracte klasse CoachModule en geïmplementeerde klasse TailgatingStatus.

De klasse heeft vijf attributen. Een overzicht hiervan is te vinden in tabel 5.

Tabel 5: Attributen van de TailgatingStatus klasse.

Attribuut	Doel
duration	Hier word de duur van het bumperkleven in opgeslagen. Deze tijd wordt gebruikt voor het bepalen van het waarschuwingsniveau.
measuring	Om aan te geven of er momenteel een bumperkleef meeting plaatsvindt. Zo niet, dan is het waarschuwingsniveau gegarandeerd 0.
start_time	Op het moment dat bumperkleven plaatsvind wordt hier de starttijd ingezet.
tailgating	Om aan te geven of het voertuig momenteel aan het bumperkleven is.
warning_level	Om het actuele waarschuwingsniveau mee aan te geven.

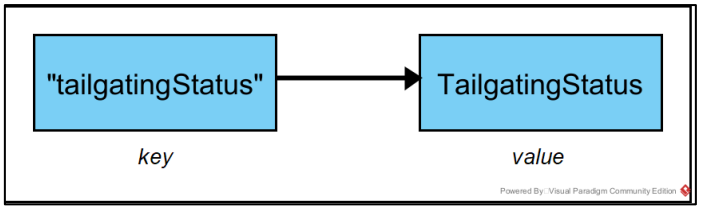
De constructor wordt niet toegelicht. Verder heeft de klasse heeft zes methoden. Tabel 6 geeft hier een overzicht van weer.

Tabel 6: Methoden van de TailgatingStatus klasse.

Methode	Doel
determine_warning_level	Geeft de huidige bumperkleef waarschuwingsniveau terug.
get_closest_lead	Geeft de meest dichtbijzijnste voorligger terug.
is_tailgating	Geeft weer of er de ego momenteel aan het bumperkleven is.
start_measurement	Start een bumperkleef meting.
stop_measurement	Stopt een bumperkleef meting.
update	Geeft de huidige bumperkleef status terug.

6.3.2. Vemelding in modules

Om ervoor te zorgen *tailgatingStatus* wordt geüpdatet, is het nodig om de module te vermelden in de *COACH_MODULES* variabele. Figuur 15 geeft weer hoe *tailgatingStatus* wordt toegevoegd aan de dictionary.



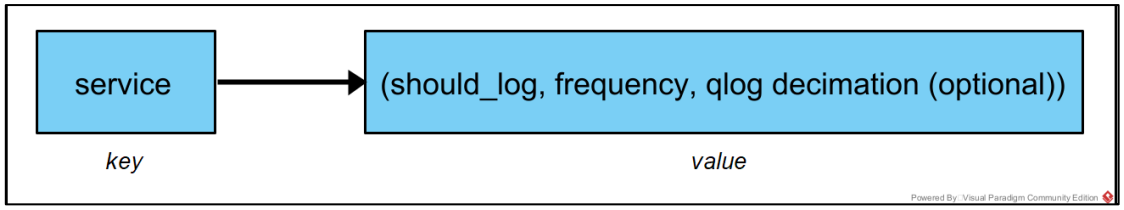
Figuur 15: tailgatingStatus item in de modules dictionary.

6.4. cereal service

Om ervoor te zorgen dat het *drivingCoachState* gepubliceerd kan worden en andere services hierop kunnen abonneren is het nodig om het bericht te vermelden in de *services* dictionary in het *services.py* bestand van cereal (cereal/services.py) (16).

services dictionary

In figuur 16 is de structuur weergegeven van de *services* dictionary. De value is een tuple bestaande uit de twee of drie variabelen.



Figuur 16: Structuur van de services dictionary.

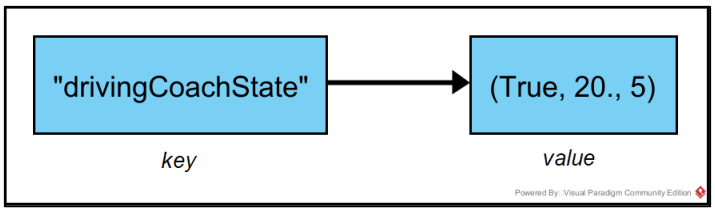
Tabel 7 geeft een overzicht van alle variabelen inclusief datatype en beschrijving.

Tabel 7: Variabelen van een item in de services dictionary.

Variabele	Datatype	Beschrijving
service	String	Naam van het bericht.
should_log	Boolean	Om aan te even dat het bericht gelogd moet worden.
frequency	Float	Hoe vaak het bericht geüpdatet dient te worden.
qlog_decimation	Integer	Optionele waarde om aan te geven in hoeverre floating point getallen afgerond moeten worden (indien van toepassing).

Vermelding in services dictionary

Figuur 17 geeft weer hoe *drivingCoachState* wordt toegevoegd aan de *services* dictionary. Het bericht wordt twintig keer per seconde geüpdatet en floating point getallen worden tot 5 decimalen afgerond.



Figuur 17: drivingCoachState item in de services dictionary.

6.5. Events

De bumperkleef waarschuwingen worden toegevoegd als events. Events worden geïmplementeerd als een *Alert* instantie. Verder bestaan er meerdere soorten event types (*ET* klasse) en dus moet er een geschikte uitgekozen worden. Tot slot is de *controlsd* service verantwoordelijk voor het aanmaken van events en dus moet er hier logica voor aan toegevoegd worden.

6.5.1. EventName enum

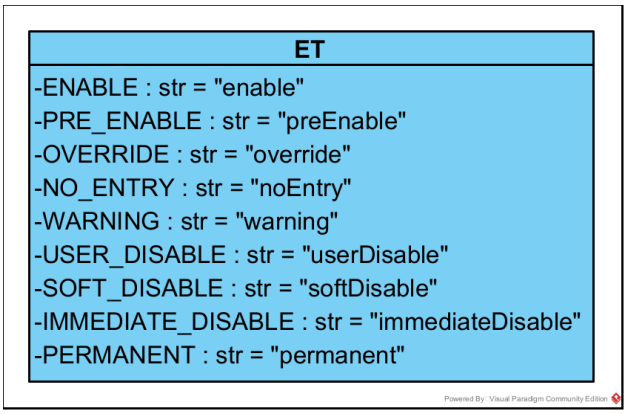
De namen van events zijn toegevoegd aan de *EventName* enum die zich bevindt in de *CarEvent* struct binnen cereal. Dit is toegelicht in paragraaf 5.1.4.

6.5.2. Event types (ET klasse)

Een overzicht van de event types is te vinden als figuur 18. Het volgende is hierover bekend:

- Events van het type *WARNING* zijn waarschuwingen die alleen afgevuurd worden wanneer openpilot is geactiveerd of 'soft disabled' is (?).
- Events van het type *PERMANENT* zijn waarschuwingen die afgevuurd worden, ongeacht wat de status van openpilot is (geactiveerd of gedeactiveerd).

Verder is er helaas weinig bekend over het doel van de verschillende types. Wel zijn er tests uitgevoerd waaruit bleek dat waarschuwingen van het type *PERMANENT* als enige afgevuurd worden wanneer openpilot gedeactiveerd is. De types hoeven dus ook verder uitgezocht te worden: *PERMANENT* is het type die nodig is voor de bumperkleef waarschuwingen; bumperkleef waarschuwingen moeten immers altijd getoond worden, ook wanneer de bestuurder zelf aan het rijden is (en openpilot gedeactiveerd is).



Figuur 18: Overzicht van alle event types, geïmplementeerd als ET klasse.

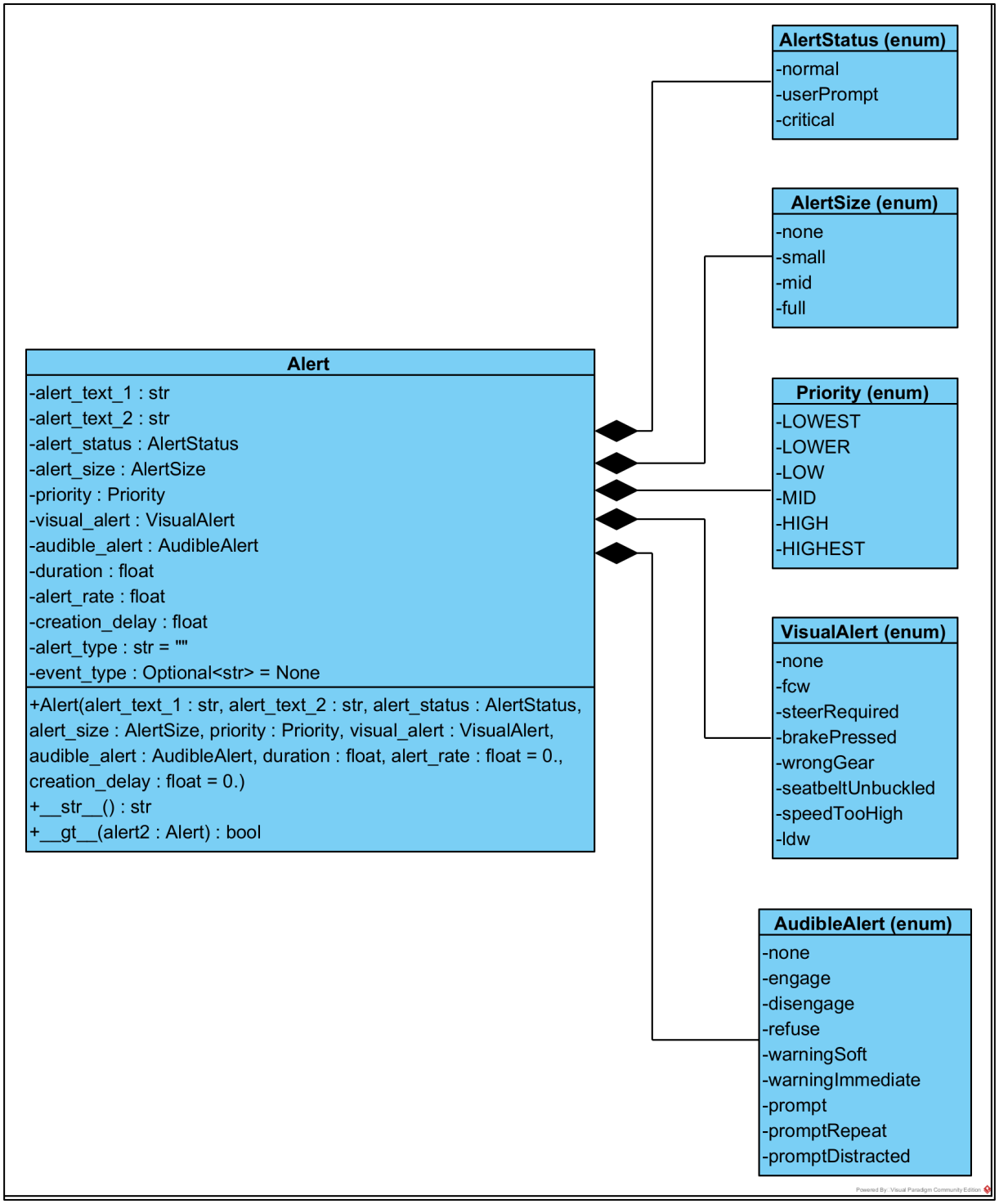
6.5.3. Alert klasse

De (hoofd) klasse waarmee events geïmplementeerd worden is *Alert*. Een definitie is te vinden als figuur 19. Het bestaat uit twaalf attributen, een constructor en twee methoden.

Enkele attributen hebben een enum als type en bevinden zich niet in hetzelfde bestand als *Alert*:

- *Alert* en *Priority* zijn te vinden in het bestand *events.py* (openpilot/selfdrive/controls/lib/events.py) (17).
- *AlertStatus* en *AlertSize* zijn te vinden in het bestand *log.capnp* (cereal/log.capnp) (9).
- *VisualAlert* en *AudibleAlert* zijn te vinden in het bestand *car.capnp* (cereal/car.capnp) (11).

In tabel 8 is een overzicht te vinden van alle attributen en doel van de *Alert* klasse.



Figuur 19: Definitie van de Alert klasse inclusief AlertStatus, AlertSize, Priority, VisualAlert en AudibleAlert enums.

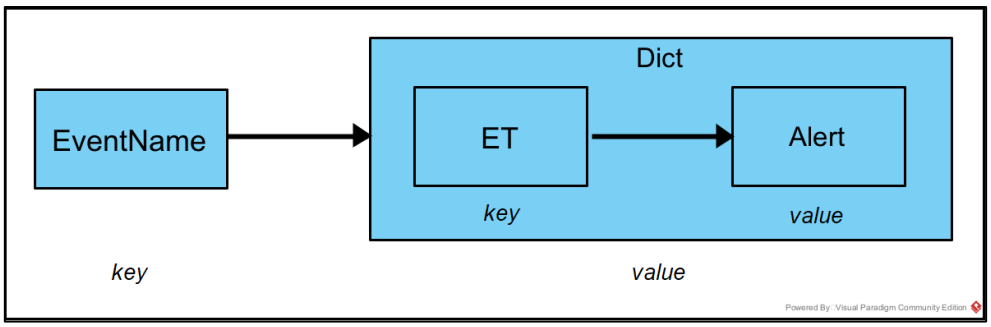
Tabel 8: Attributen van de Alert klasse.

Attribuut	Doel
alert_text_1	De eerste (hoofd)tekst van een alert. Alleen zichtbaar vanaf *alert_size small*.
alert_text_2	De tweede tekst van een alert. Alleen zichtbaar vanaf *alert_size normal*.
alert_status	De status van de alert. *normal* is bedoeld voor lage prioriteit voor gemak van de gebruiker, *userPrompt* is bedoeld voor gemiddelde prioriteit waarvoor mogelijk tussenkomst van gebruiker vereist is en *critical* is bedoeld voor hoge prioriteit die onmiddellijke tussenkomst van de gebruiker vereist.
alert_size	De grootte van de alert. De grootte varieert tussen niks tonen (*none*), klein (*small*), half scherm (*mid*) en volledig scherm (*full*).
priority	De prioriteit van de alert. Een alert met een hogere prioriteit krijgt voorrang.
visual_alert	Visuele waarschuwingen die getoond kunnen worden door de auto (bijvoorbeeld op het dashboard). Deze alerts komen van Honda en moeten (zo goed mogelijk) gekoppeld worden aan de alerts van andere (merk/type) auto's. Deze alerts worden alleen getoond wanneer deze koppeling gemaakt is. Er kan natuurlijk ook gekozen worden om geen te tonen (*none*).
audible_alert	Auditieve waarschuwingen/alarmeringen. Ze variëren tussen geen (*none*), in- en uitschakelen van openpilot (*disengage* en *engage*), niet toestaan (*refuse*), (heftige) waarschuwing (*warning*) en geven van een seintje (*prompt)*.
duration	De duur dat een alert zichtbaar is in de logs, nadat deze is aangemaakt. Dit heeft dus niks te maken met de duur van de alert op de user interface of iets dergelijks.
alert_rate	*Onbekend*. Het lijkt erop dat deze attribuut (momenteel) nog geen doel heeft. De standaardwaarde is '0.'.
creation_delay	Hoe lang het creëren van de alert uitgesteld dient te worden. De standaardwaarde is '0.'.
alert_type	Het type alert. Combinatie van *EventName* (zie paragraaf 6.5.1) en event type (zie paragraaf 6.5.2).
event_type	Het event type (zie paragraaf 6.5.2).

De constructor en methoden worden niet verder toegelicht.

6.5.4. Event implementaties

De implementaties van alle events zijn te vinden in het bestand *events.py* (openpilot/selfdrive/controls/lib/events.py) (17). Ze zijn allemaal geplaatst in de globale variabele *EVENTS*, een Dictionary met *EventName* enum als keys en als value een geneste Dictionary. De geneste Dictionary heeft ET (event type) datatype als keys en *Alert* datatype als values. Figuur 20 geeft de structuur van de *EVENTS* dictionary weer.



Figuur 20: Structuur van de EVENTS dictionary.

Het is dus mogelijk meerdere event types te koppelen aan een event (*EventName*), met verschillende soorten waarschuwing implementaties. Aan deze dictionary worden dus de implementaties van de bumperkleef events toegevoegd.

Bumperkleef events

Voor de bumperkleef waarschuwing worden de events *tailgating*, *promptTailgating* en *persistentTailgating* geïmplementeerd. Deze zijn ieder een instantie van de *Alert* klasse. Tabel 9 geeft een overzicht van de waarden van de attributen van waarmee iedere event mee geïnitialiseerd wordt.

De events zullen stapsgewijs intensiveren aan de hand van het waarschuwingsniveau. Bij elke stap krijgt het event een hogere prioriteit, waardoor deze voorrang krijgt op de vorig afgevuurde bumperkleef event.

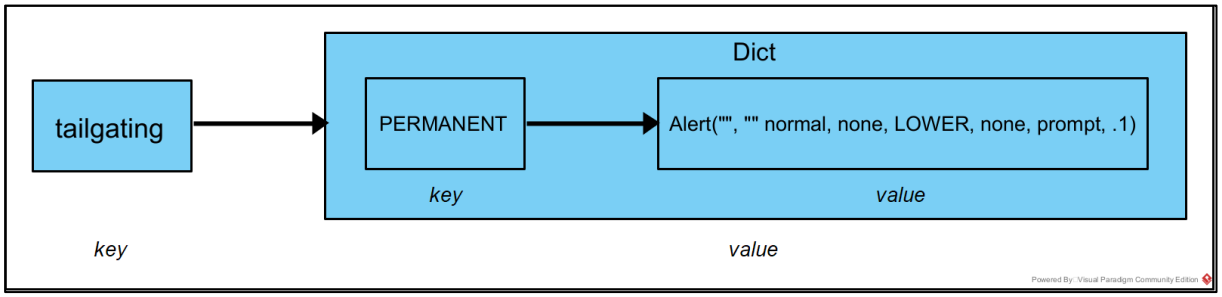
Tabel 9: Overzicht van de waarden van de attributen waar de tailgating, promptTailgating en persistentTailgating events geïmplementeerd mee worden.

	tailgating	promptTailgating	persistentTailgating
alert_text_1	“”	“Tailgating”	“TAILGATING”
alert_text_2	“”	“”	“”
alert_status	normal	normal	userPrompt
alert_size	none	small	small
priority	LOWER	LOW	MID
visual_alert	none	none	fcw
audible_alert	prompt	prompt	promptRepeat
duration	.1	.1	.1

tailgating

Het doel van het *tailgating* event is om een eenmalige auditieve waarschuwing te geven, om de bestuurder op bumperkleven te wijzen. Daarnaast zal het waarschuwingsteken zichtbaar worden op de user interface (toegelicht in paragraaf 6.6) en blijft zichtbaar bij iedere opvolgende event.

Figuur 21 geeft als voorbeeld weer hoe *tailgating* wordt toegevoegd aan de *EVENTS* dictionary. Voor de andere twee events zullen dergelijke figuren worden weggelaten; ze zijn immers nagenoeg een kopie van figuur 21.



Figuur 21: tailgating item in de EVENTS dictionary.

promptTailgating

Bij het *promptTailgating* event zal er een kleine waarschuwing op het scherm verschijnen met de tekst "Tailgating". De status is *normal*, dus de waarschuwing zal subtiel zijn. Wederom zal er een eenmalige auditieve waarschuwing te horen.

persistentTailgating

Wanneer na langere tijd de bestuurder onophoudelijk aan het bumperkleven is zal de *persistentTailgating* event afgevuurd worden. Het blijft een kleine waarschuwing, maar de status is *userPrompt* wat de visuele waarschuwing iets heftiger maakt. De tekst zal in hoofdletters getoond worden : "TAILGATING". Verder zal de auditieve waarschuwing herhaaldelijk te horen zijn en zal er een FCW melding op het dashboard zichtbaar worden (indien ondersteund).

6.5.5. Aanpassingen aan controlsd

Het enige wat er nog te doen staat is zorgen dat de *controlsd* service de bumperkleef events aanmaakt, wanneer nodig. De service is te vinden in het bestand *controlsd.py* (openpilot/selfdrive/controls/controlsd.py) (18).

Abonneren op drivingCoachState

De klasse *Controls* moet tijdens de initialisatie abonneren op *drivingCoachState*. Hiervoor moet het toegevoegd worden aan de lijst van topics waarop de service moet abonneren.

Aanpassen update_events methode

De methode *update_events* moet logica worden toegevoegd zodat bumperkleef events worden afgevuurd wanneer een bepaald waarschuwingsniveau is bereikt.

6.6. User interface

Op het user interface moet het bumperkleef waarschuwingsteken, wanneer nodig, zichtbaar zijn. Hiervoor moet het icoon toegevoegd worden als asset en er moeten aanpassingen plaatsvinden aan de *ui* service. Voor de user interface maakt openpilot gebruik van het *Qt Framework* (19).

6.6.1. Icoon toevoegen als asset

Het waarschuwingsteken zal als een SVG (Scalable Vector Graphics) bestand toegevoegd worden aan de assets (openpilot/selfdrive/assets) (20).

6.6.2. Aanpassingen aan ui

De user interface moet zo aangepast worden dat het bumperkleef waarschuwingsteken zichtbaar is wanneer het waarschuwingsniveau de waarde 1 of hoger heeft.

Abonneren op drivingCoachState

De *ui* service moet geabonneerd zijn op *drivingCoachState*, zodat het bumperkleef waarschuwingsniveau opgehaald kan worden. Hiervoor moet het toegevoegd zijn aan de lijst waarop de service moet abonneren. Deze is te vinden in het bestand *ui.cc* (openpilot/selfdrive/ui/ui.cc) (21).

Logica

Alle ui logica dat moet plaatsvinden wanneer de auto is gestart bevindt zich in het bestand *onroad.cc* (openpilot/selfdrive/ui/qt/onroad.cc) en de bijbehorende header file *onroad.h* (openpilot/selfdrive/ui/qt/onroad.h) (22) (23).

De volgende logica moet geïmplementeerd worden:

- Het icoon moet ingeladen worden.
- Het bumperkleef waarschuwingsniveau moet opgehaald worden uit *drivingCoachState*.
- Het icoon moet zichtbaar zijn wanneer het bumperkleef waarschuwingsniveau 1 of hoger is.

6.6.3. Plaatsing van icoon

Het icoon zal aan de rechterkant geplaatst worden, onder het *engageable* icoon en zal nagenoeg even groot als deze zijn. Figuur 22 geeft een schets weer hoe dit er ongeveer uit zal komen te zien op de ui.

Figuur 22: Schets van de plaatsing van het bumperkleef waarschuwingsteken op de user interface.

Literatuurlijst

1. comma.ai. commaai/openpilot: openpilot is an open source driver assistance system. openpilot performs the functions of Automated Lane Centering and Adaptive Cruise Control for over 150 supported car makes and models. GitHub. Beschikbaar via: https://github.com/commaai/openpilot. Geraadpleegd 2022 juni 5.
2. SWOV. Volgtijd en verkeersveiligheid. Leidschendam: SWOV; 2012.
3. Vogel K. A comparison of headway and time to collision as safety indicators. Accident Analysis & Prevention. 2003 mei: p. 427-433.
4. Carnetsoft B.V. Data sampling and data storage for research simulators. Carnetsoft Driving Simulators. 2013. Beschikbaar via: https://cs-driving-simulator.com/downloads/DataStorage.pdf. Geraadpleegd 2022 juni 2.
5. Dijksterhuis C. over. Dijksterhuis Onderzoek & Advies. Beschikbaar via: https://www.chrisdijksterhuis.nl/over. Geraadpleegd 2022 juni 2.
6. comma.ai. How openpilot works in 2021. 2021. Beschikbaar via: https://blog.comma.ai/openpilot-in-2021/. Geraadpleegd 2022 april 8.
7. comma.ai. commaai/cereal: capnp struct definitions and messaging used in comma ecosystem. GitHub. Beschikbaar via: https://github.com/commaai/cereal/. Geraadpleegd 2022 juni 6.
8. Cap'n Proto. Cap'n Proto: Introduction. Beschikbaar via: https://capnproto.org/. Geraadpleegd 2022 juni 6.
9. comma.ai. commaai/cereal: log.capnp. GitHub. Beschikbaar via: https://github.com/commaai/cereal/blob/master/log.capnp. Geraadpleegd 2022 juni 6.
10. comma.ai. Introduction to openpilot · commaai/openpilot Wiki. GitHub. Beschikbaar via: https://github.com/commaai/openpilot/wiki/Introduction-to-openpilot. Geraadpleegd 2022 juni 8.
11. comma.ai. commaai/cereal: car.capnp. GitHub. Beschikbaar via: https://github.com/commaai/cereal/blob/master/car.capnp. Geraadpleegd 2022 juni 6.
12. comma.ai. commaai/openpilot: radar_helpers.py. GitHub. Beschikbaar via: https://github.com/commaai/openpilot/blob/master/selfdrive/controls/lib/radar_helpers.py. Geraadpleegd 2022 juni 7.
13. comma.ai. commaai/openpilot: selfdrive. GitHub. Beschikbaar via: https://github.com/commaai/openpilot/tree/master/selfdrive. Geraadpleegd 2022 juni 7.

14. comma.ai. commaai/openpilot: process.py. GitHub. Beschikbaar via: https://github.com/commaai/openpilot/blob/master/selfdrive/manager/process.py. Geraadpleegd 2022 juni 7.

15. comma.ai. commaai/openpilot: process_config.py. GitHub. Beschikbaar via: https://github.com/commaai/openpilot/blob/master/selfdrive/manager/process_config.py. Geraadpleegd 2022 juni 7.

16. comma.ai. commaai/cereal: services.py. GitHub. Beschikbaar via: https://github.com/commaai/cereal/blob/master/services.py. Geraadpleegd 2022 juni 7.

17. comma.ai. commaai/openpilot: events.py. GitHub. Beschikbaar via: https://github.com/commaai/openpilot/blob/master/selfdrive/controls/lib/events.py. Geraadpleegd 2022 juni 8.

18. comma.ai. commaai/openpilot: controlsd.py. GitHub. Beschikbaar via: https://github.com/commaai/openpilot/blob/master/selfdrive/controls/controlsd.py. Geraadpleegd 2022 juni 8.

19. The Qt Company. Cross-platform software development for embedded & desktop. Qt. Beschikbaar via: https://www.qt.io/. Geraadpleegd 2022 juni 8.

20. comma.ai. commaai/openpilot: assets. GitHub. Beschikbaar via: https://github.com/commaai/openpilot/tree/master/selfdrive/assets. Geraadpleegd 2022 juni 8.

21. comma.ai. commaai/openpilot: ui.cc. GitHub. Beschikbaar via: https://github.com/commaai/openpilot/blob/master/selfdrive/ui/ui.cc. Geraadpleegd 2022 juni 8.

22. comma.ai. commaai/openpilot: onroad.cc. GitHub. Beschikbaar via: https://github.com/commaai/openpilot/blob/master/selfdrive/ui/qt/onroad.cc. Geraadpleegd 2022 juni 8.

23. comma.ai. commaai/openpilot: onroad.h. GitHub. Beschikbaar via: https://github.com/commaai/openpilot/blob/master/selfdrive/ui/qt/onroad.h. Geraadpleegd 2022 juni 8.