Hanzehogeschool Groningen
University of Applied Sciences
digital society hub

comma.ai

OPENPILOT TAILGATING WARNING

Een bumperkleef detectie- en waarschuwingsservice voor het openpilot systeem

TOELICHTING REALISATIE

Auteur
Jeroen Lammersma

Publicatiedatum
10 september 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

TOELICHTING REALISATIE

Publicatiedatum
10 september 2022

Versie
1.0

Hanzehogeschool Groningen University of Applied Sciences | digital society hub

comma.ai

Inhoudsopgave

1. Inleiding

In dit document wordt de realisatie van het project *openpilot Tailgaiting Warning* toegelicht. De bumperkleef detectie- en waarschuwingsservice is geïmplementeerd binnen het *openpilot* systeem.

openpilot is open-source en te vinden op GitHub onder het account van comma.ai (1). Voor het project is een fork gemaakt van deze repository. De default branch is verwisseld met de branch: *driving-coach*. De fork is te bereiken via de volgende link:

github.com/jeroenlammersma/openpilot

Voor het afstudeerproject is de versie van openpilot (inclusief de bumperkleef detectie- en waarschuwingsservice) die opgeleverd is aan de opdrachtgever getagd onder de naam ‘graduation-project’. Controleer bij het bekijken van de repository of deze tag geselecteerd is (of klik op de bovenstaande link).

1.1. Vooruitblik

Hoofdstuk 2 toont een overzicht van de commits die gepusht zijn naar de openpilot fork. Vervolgens wordt er in hoofdstuk 3 de realisatie van de bumperkleef detectie- en waarschuwingsservice beschreven. Hoofdstuk 4 behandeld de aanpassingen die gemaakt zijn aan de scripts voor de simulator. Daarna gaat hoofdstuk 5 in op continues integration en hoe deze geïmplementeerd is binnen de forks. Hoofdstuk 6 beschrijft de contributies die gedaan zijn aan openpilot en in het laatste hoofdstuk komt de openpilot-dev repository aan bod.

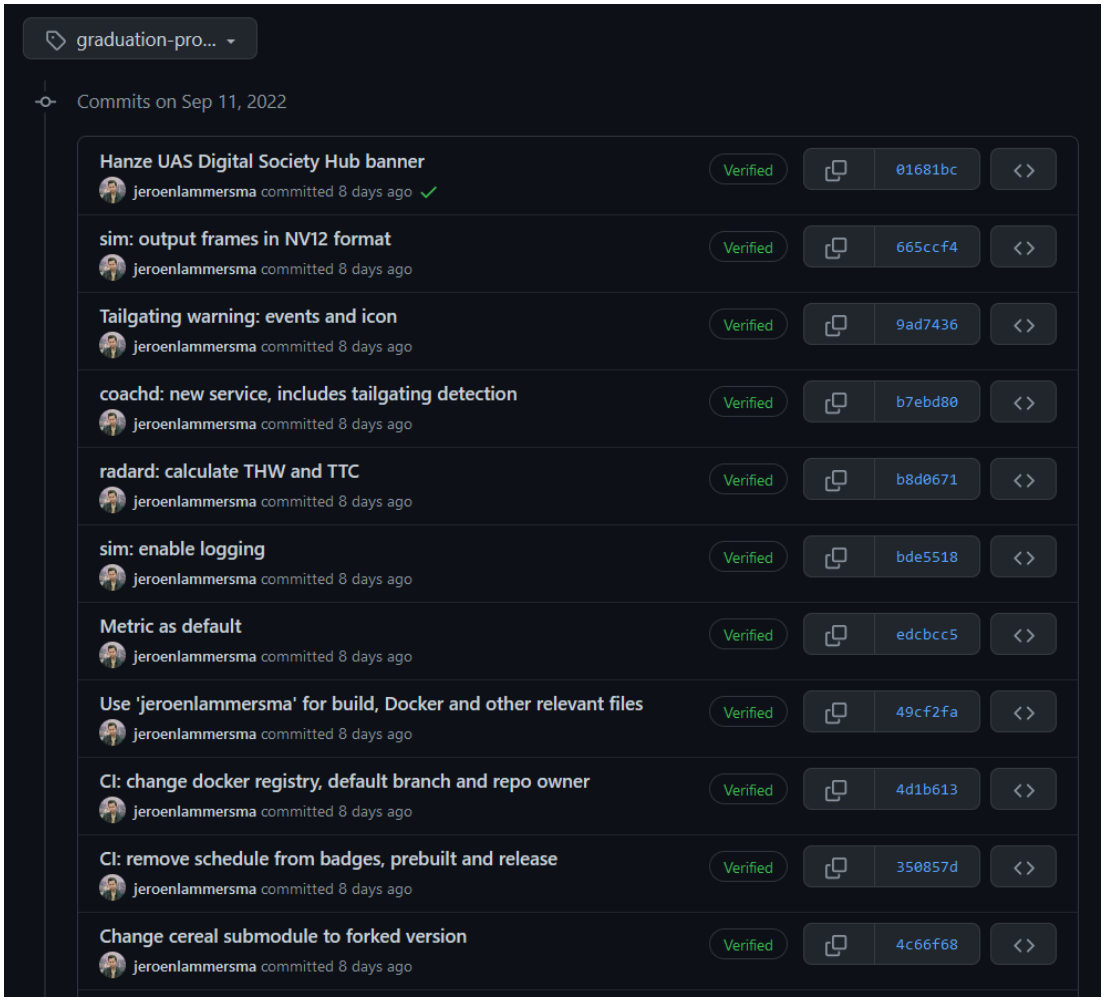
1.2. Markering

Groene markering wordt gebruikt om veranderingen aan code mee aan te duiden, mits de verandering plaatsvindt binnen een reeds bestaand element (zoals een functie of een klasse). Compleet nieuwe toevoegingen zullen dus niet gemarkeerd zijn.

2. Commits

- github.com/jeroenlammersma/openpilot/commits

Er zijn elf commits gepusht naar openpilot:



Figuur 1: Commits die naar de openpilot repository zijn gepusht.

Van onder naar boven:

- De eerste commit is het wijzigen van de cereal submodule naar de geforkte versie.
- Vervolgens zijn er wat aanpassingen gemaakt in de CI en andere relevante bestanden met drie commits.
- De volgende twee commits stellen het metrisch systeem als standaard in en zorgen ervoor dat logging is ingeschakeld voor de simulator.
- Dan volgen de commits die de bumperkleef- en waarschuwingsservice implementeren met drie commits.
- De laatste twee commits zorgen voor een verbetering aan de simulator en een banner op de hoofdpagina.

3. Bumperkleef detectie- en waarschuwingsservice

De implementatie van de bumperkleef detectie- en waarschuwingsservice is grotendeels geïmplementeerd binnen selfdrive. Het detectie deel is toegevoegd als rijcoach module aan de *coachd* service en het waarschuwingsdeel is in de user interface en *controls* service geïmplementeerd (toegelicht in paragraaf 3.4). Verder was het nodig de submodule cereal uit te breiden (toegelicht in paragraaf 3.1).

Voor de code geschreven in Python is zoveel mogelijk de Google Python Style Guide aangehouden (2). Er is echter niet gekozen voor een inspringing van vier spaties, omdat openpilot een inspringing van 2 spaties hanteert.

3.1. cereal submodule

github.com/jeroenlammersma/cereal

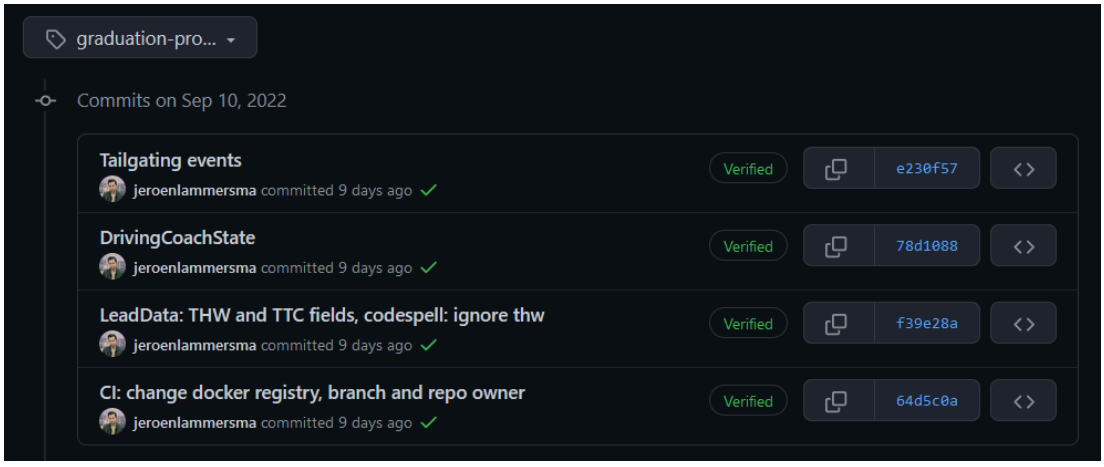
cereal is een submodule binnen de openpilot repository (3). Het maakt gebruikt van Cap'n Proto (4). Om onder andere *drivingCoachState* berichten te versturen is het nodig om het Cap'n Proto schema uit te breiden

Hiervoor is ook van de cereal repository een fork gemaakt. Vervolgens is de cereal submodule op de openpilot fork omgewisseld met deze fork. Verder is ook hier de default branch verwisseld met de branch: *driving-coach*.

3.1.1. Commits

github.com/jeroenlammersma/cereal/commits

Er zijn vier commits gepusht naar cereal:



Figuur 2: Commits die naar de cereal repository zijn gepusht.

Van onder naar boven:

- De eerste commit zijn wat aanpassingen voor de CI.
- De volgende commit is het toevoegen van *THW* en *TTC* velden aan *LeadData*.
- Dan wordt de DrivingCoachState bericht toegevoegd met één commit.
- De laatste commit is het toevoegen van de tailgating events aan *CarEvent*.

3.1.2. car.capnp

openpilot/cereal/car.capnp

Om te zorgen dat de bumperkleefwaarschuwingen afgevuurd kunnen worden, worden deze toegevoegd aan de *EventName* enum binnen de *CarEvent* struct. De waarschuwingen zijn: *tailgating*, *promptTailgating* en *persistentTailgating*.

```
  8 struct CarEvent @0x9b1657f34caf3ad3 {
 22   enum EventName @0xbaa8c5d505f727de {
116     # driving coach
117     tailgating @114;
118     promptTailgating @115;
119     persistentTailgating @116;
143   }
144 }
```

3.1.3. log.capnp

openpilot/cereal/log.capnp

Bevat Cap'n Proto definities.

LeadData

Aan de *LeadData* struct worden twee velden toegevoegd: *thw* en *ttc*. In *thw* wordt de time-headway geplaatst en in *ttc* is de time-to-collision, beiden in seconden.

```
491 struct RadarState @0x9a185389d6fdd05f {
501   struct LeadData {
516     thw @15 :Float32;  # seconds
517     ttc @16 :Float32;  # seconds
520   }
529 }
```

DrivingCoachState

Verder wordt er nieuwe struct geïntroduceerd: *DrivingCoachState*. Deze bevat momenteel één veld: *tailgatingStatus*. Hierin wordt de actuele bumperkleef status in geplaatst.

Dit wordt gerealiseerd door de geneste struct *TailgatingStatus*. Deze bevat vier velden:

- *active*, om aan te geven of de module actief is;

- *isTailgating*, om aan te geven of er momenteel sprake is van bumperkleven;
- *duration*, waarin de duur van het bumperkleven wordt gezet (in nanoseconden);
- *warningLevel*, voor het huidige bumperkleef waarschuwingsniveau.

Het rijcoach bericht is ook eenvoudig uit te breiden; nieuwe velden zijn namelijk gemakkelijk toegevoegd aan de *DrivingCoachState* struct.

```
1876 struct DrivingCoachState {
1877   tailgatingStatus @0 :TailgatingStatus;
1878
1879   struct TailgatingStatus {
1880     active @0 :Bool;
1881     isTailgating @1 :Bool;
1882     duration @2 :UInt64;  # nanoseconds
1883     warningLevel @3 :UInt16;
1884   }
1885 }
```

Event

Om te zorgen dat de rijcoach-staat als bericht gecreëerd en verstuurd kan worden wordt het veld *drivingCoachState* toegevoegd aan de union binnen de *Event* struct.

```
1887 struct Event {
1891   union {
1957     # driving coach
1958     drivingCoachState @93 :DrivingCoachState;
2004   }
2005 }
```

3.1.4. services.py

openpilot/cereal/services.py

Als laatst wordt *drivingCoachState* vermeld in de *services* dictionary. De frequency is twintig keer per seconde en floats worden afgerond met maximaal vijf cijfers na de komma. Ook zal *drivingCoachState* nu zichtbaar zijn in de logs.

```
21 services = {
22   # service: (should_log, frequency, qlog decimation (optional))
73   # driving coach
74   "drivingCoachState": (True, 20., 5),
82 }
```

Nu is het mogelijk om een *drivingCoachState* bericht te publiceren of hierop te abonneren.

3.1.5. Gegenereerde code

Wanneer openpilot opnieuw gebuild wordt met SCons, dan wordt er automatisch C++ code gegeneerd voor de nieuwe toevoegen aan cereal. Deze kunnen gebruikt worden in componenten geschreven in C++, zoals de user interface.

Op dit proces zal niet uitgebreid in worden gegaan. Echter, om een idee te geven volgt hieronder een voorbeeld van de gegenereerde getter en setter voor het *isTailgating* veld:

```
39593 inline bool DrivingCoachState::TailgatingStatus::Builder::
      getIsTailgating() {
39594   return _builder.getDataField<bool>(
39595       ::capnp::bounded<1>() * ::capnp::ELEMENTS);
39596 }
39597 inline void DrivingCoachState::TailgatingStatus::Builder::
      setIsTailgating(bool value) {
39598   _builder.setDataField<bool>(
39599       ::capnp::bounded<1>() * ::capnp::ELEMENTS, value);
39600 }
```

Voor Python is dit zelfs nog makkelijker, omdat zowel cereal als de *pycapnp* package (capnp bindings voor Python) beiden een eenvoudige API beschikbaar stellen (5). Deze worden gebruikt door de geïmplementeerde module en dus zullen voorbeelden hiervan in het volgende hoofdstuk volgen.

3.2. assets

openpilot/selfdrive/assets/tailgating_warning.svg

Het bumperkleef waarschuwingsicoon wordt toegevoegd aan de assets als SVG bestand: *tailgating_warning.svg*. Hiervoor wordt een nieuwe map aangemaakt binnen assets genaamd: *coach*.

3.3. radard

De *radard* service moet de THW en TTC gaan berekenen, wanneer nodig, en deze waarden plaatsen in de *leadOne* en *leadTwo* velden van het *radarState* bericht.

3.3.1. radar_helpers.py

openpilot/selfdrive/controls/lib/radar_helpers.py

Bevat hulpfuncties voor *RadarD*.

FLOAT32_INF

Allereerst wordt er een constante gedefinieerd: *FLOAT32_INF*. Deze bevat de hoogste waarde dat een Float32 kan bevatten en representeert 'oneindig':

```
18 FLOAT32_INF = 3.4028235e+38   # largest value Float32 can hold
```

Het is niet mogelijk om de Python variant van infinity te gebruiken. Gebruik hiervan zorgt voor ongedefinieerd gedrag, omdat de bit-breedte groter is dan Float32.

```
float('inf')  # causes undefined behaviour!
```

calculate_thw

Dan volgt de functie *calculate_thw* die de time headway berekend. Als de snelheid van het ego voertuig groter is dan nul, dan wordt de THW teruggeven. In het andere geval wordt 'oneindig' teruggeven.

```
21 def calculate_thw(d_rel: float, v_ego: float) -> float:
22   # time headway = distance / velocity (in m/s), only when velocity of ego
     > 0
23   return d_rel / v_ego if v_ego > 0 else FLOAT32_INF
```

calculate_ttc

De functie *calculate_ttc* berekent de time-to-collision. Als zowel de snelheid van het ego voertuig én de relatieve snelheid ten opzichte van de voorligger groter zijn dan nul wordt de TTC teruggeven. In het andere geval wordt 'oneindig' teruggeven.

```
26 def calculate_ttc(d_rel: float, v_rel: float, v_ego: float) -> float:
27   # time-to-collision = distance / relative velocity (in m/s)
28   # only when velocity of ego > 0 and relative velocity > 0 (gaining on
     lead vehicle)
29   # for openpilot, relative velocity needs to be negated
30   return d_rel / -v_rel if v_ego > 0 and -v_rel > 0 else FLOAT32_INF
```

Track

De signatuur van methode *update* van de klasse *Track* is veranderd, omdat het nu ook de snelheid van het ego voertuig, *v_ego*, vereist voor het bereken van de THW en TTC. Verder worden de zojuist benoemde functies gebruikt om de attributen *thw* en *ttc* van waarden te voorzien:

```
33 class Track():
42   def update(self, d_rel, y_rel, v_rel, v_lead, measured, v_ego):
63     # computed THW and TTC
64     self.thw = calculate_thw(self.dRel, v_ego)
65     self.ttc = calculate_ttc(self.dRel, self.vRel, v_ego)
```

Cluster

Aan de klasse *Cluster* worden twee properties toegevoegd, *thw* en *ttc*. Beiden bereken het gemiddelde van de THW of TTC waarden in de tracks en geeft deze terug:

```
79 class Cluster():
138   @property
139   def thw(self):
140     return mean([t.thw for t in self.tracks])
141
142   @property
143   def ttc(self):
144     return mean([t.ttc for t in self.tracks])
145
```

get_RadarState

De functie *get_RadarState* geeft een *LeadData* datastructuur terug die wordt geplaatst in *leadOne* of *leadTwo* van het *radarState* bericht. Omdat *thw* en *ttc* als velden zijn toegevoegd aan de *LeadData* struct is het nodig om deze van de juiste waarden te voorzien. Hiervoor worden de zojuist benoemde properties gebruikt:

```
146   def get_RadarState(self, model_prob=0.0):
147     return {
159       "thw": float(self.thw),
160       "ttc": float(self.ttc)
161     }
162
```

get_RadarState_from_vision

Bij de functie *get_RadarState_from_vision* wordt in principe dezelfde wijziging aangebracht. Om de functie netjes te houden worden *dRel* en *vRel* gedefinieerd als variabelen: zo wordt repetitieve code voorkomen.

```
163   def get_RadarState_from_vision(self, lead_msg, v_ego):
164     dRel = lead_msg.x[0] - RADAR_TO_CAMERA
165     vRel = lead_msg.v[0] - v_ego
166     return {
167       "dRel": float(dRel),
168       "yRel": float(-lead_msg.y[0]),
      ...
178       "thw": float(calculate_thw(dRel, v_ego)),
179       "ttc": float(calculate_ttc(dRel, vRel, v_ego))
180     }
```

Nu is het mogelijk om THW en TTC informatie te verkrijgen van de *radard* service.

3.3.2. radard.py

openpilot/selfdrive/controls/radard.py

De signatuur van de functie *update* van de klasse *Track* is veranderd. Daarom moet er een kleine wijziging plaatsvinden binnen de functie *update* van de klasse *RadarD*. De variabele *v_ego* moet als argument toegevoegd worden bij het updaten van de tracks:

```
      ...
 92 class RadarD():
      ...
105   def update(self, sm, rr):
      ...
123     # *** compute the tracks ***
124     for ids in ar_pts:
      ...
133       self.tracks[ids].update(rpt[0], rpt[1], rpt[2], v_lead, rpt[3],
          self.v_ego)
      ...
```

3.4. coachd

De rijcoach service, *coachd*, wordt als nieuwe service toegevoegd aan openpilot.

3.4.1. modules

De package *modules* binnen *coachd* is bedoeld voor de rijcoach modules. Aan de package wordt de bumperkleef detectie toegevoegd, samen met unit tests.

3.4.2. base.py

openpilot/selfdrive/coachd/modules/base.py

Bevat de abstracte klasse *CoachModule*.

Imports

Eerst een aantal imports:

- *ABC (Abstract Base Class)* en *abstractmethod* worden gebruikt om van *CoachModule* een abstracte klasse te maken (6);
- *capnp* en *messaging* worden gebruikt voor optionele type hints.

```
1 from abc import ABC, abstractmethod
2
3 import capnp
4
5 from cereal import messaging
```

CoachModule

De klasse *CoachModule* is afgeleid van *ABC* en heeft één abstracte methode: *update*. Deze methode ontvangt een *SubMaster* object als argument en geeft een *_DynamicStructBuilder* object terug.

SubMaster wordt gebruikt om toegang te krijgen de services waarop de module op geabonneerd is. *_DynamicStructBuilder* is het object dat gecreëerd wordt bij het initialiseren van een *capnp* struct (wat dit precies is zal later helder worden).

```
8 class CoachModule(ABC):
9
10   @abstractmethod
11   def update(
12       self, sm: messaging.SubMaster) -> capnp.lib.capnp.
         _DynamicStructBuilder:
13     pass
14
```

3.4.3. tailgating_detection.py

openpilot/selfdrive/coachd/modules/tailgating_detection.py

Bevat de implementatie van de bumperkleef detectie module en heeft het doel om de *tailgatingStatus* te berekenen.

Imports

Eerst de imports:

- *log* en *messaging* worden gebruikt voor optionele type hints;
- *CoachModule* wordt geimplementeerd door de *TailgatingDetection* klasse.

```
1 from cereal import log, messaging
2 from selfdrive.coachd.modules.base import CoachModule
```

Constanten

Dan een achttal constanten:

- *THW_THRESHOLD*, de grenswaarde in meter waar THW onder moet vallen om te spreken van bumperkleven. Deze is gezet op 1;
- *MINIMUM_VELOCITY*, de minimale snelheid in m/s dat het ego voertuig moet rijden voordat er sprake kan zijn van bumperkleven. Deze is gezet op 5 (18 km/u);
- *TIME_TILL_LEVEL_X*, de tijd in seconden wanneer een waarschuwingsniveau bereikt is (opeenvolgend voor niveau één, twee en drie: 5, 10 en 20 seconden);
- *LEVEL_X_THRESHOLD*, de tijd in nanoseconden wanneer een waarschuwingsniveau bereikt is (seconden vermenigvuldigd met 10 tot de macht 9).

```
5  THW_THRESHOLD = 1.  # in m, ego is tailgating when THW is below threshold
6  MINIMUM_VELOCITY = 5.  # in m/s, ego not tailgating when velocity below
   minimum
7
8  # all in s
9  TIME_TILL_LEVEL_1 = 5
10 TIME_TILL_LEVEL_2 = 10
11 TIME_TILL_LEVEL_3 = 20
12
13 # all in ns
14 LEVEL_1_THRESHOLD = int(TIME_TILL_LEVEL_1 * 1e+9)
15 LEVEL_2_THRESHOLD = int(TIME_TILL_LEVEL_2 * 1e+9)
16 LEVEL_3_THRESHOLD = int(TIME_TILL_LEVEL_3 * 1e+9)
```

get_closest_leaad

De functie *get_closest_lead* ontvangt beide leads, berekend welke zich het meest dichtbij het ego voertuig bevindt en geeft deze vervolgens terug.

```
19 def get_closest_lead(
20     lead_one: log.RadarState.LeadData, lead_two: log.RadarState.LeadData
21 ) -> log.RadarState.LeadData:
22   # return nearest lead relative to ego
23   return lead_one if lead_one.dRel <= lead_two.dRel else lead_two
```

is_tailgating

De functie *is_tailgating* berekend of het ego voertuig aan het bumperkleven is. Dit is alleen het geval als *thw* binnen nul en de *THW_THRESHOLD* valt én als het snelheid van het ego voertuig boven het gedefinieerde minimum ligt.

```
26 def is_tailgating(thw: float, v_ego: float) -> bool:
27   # ego is tailgating when thw of lead is between 0 and threshold,
28   # and velocity of ego is greater than minimum
29   return 0 < thw < THW_THRESHOLD and v_ego >= MINIMUM_VELOCITY
```

TailgatingDetection

De klasse *TailgtatingDetection** implementeert *CoachModule*. Het bevat vier attributen:

- *measuring*, om aan te geven of er momenteel een bumperkleef meting plaatsvindt;
- *tailgating*, om aan te geven of er momenteel sprake is van bumperkleven;
- *start_time*, waar de starttijd van het bumperkleven in geplaatst wordt (in nanoseconden);
- *duration*, waar de duur van het bumperkleven in geplaatst wordt (in nanoseconden).

*In het ontwerp document had deze klasse de naam ‘TailgatingStatus’.

```
32 class TailgatingDetection(CoachModule):
33
34   def __init__(self) -> None:
35     self.measuring = False
36     self.tailgating = False
37     self.start_time = 0  # nanoseconds
38     self.duration = 0  # nanoseconds
39
```

warning_level

De property *warning_level* berekend het huidige waarschuwingsniveau en geeft deze terug. Het niveau wordt berekend door de duur te vergelijken met de gedefinieerde grenswaarden.

```
40   @property
41   def warning_level(self) -> int:
42     if self.duration >= LEVEL_3_THRESHOLD:
43       return 3
44     if self.duration >= LEVEL_2_THRESHOLD:
45       return 2
46     if self.duration >= LEVEL_1_THRESHOLD:
47       return 1
48     return 0
49
```

update

Binnen de methode *update* wordt het meeste werk verricht. Het ontvangt een *SubMaster* object en haalt hier eerst *radarState*, de huidige tijd (sinds het opstarten) en

carState uit*. Dan wordt de meest dichtstbijzijnde lead berekend en wordt er gekeken of het ego voertuig momenteel aan het bumperkleven is.

Als dit zo is én er is nog geen meting gestart, dan wordt deze gestart. Als er geen sprake is van bumperkleven, maar er is nog wel een meting gaande, dan wordt de meting gestopt.

Vervolgens wordt de duur bepaald door het verschil tussen de huidige tijd en de starttijd, mits er een meting gaande is. Zo niet, dan is de duur nul. Als laatst wordt de *TailgatingStatus* gecreëerd en wordt deze teruggeven.

Het is uiteraard nodig dat de *SubMaster* geabonneerd is op *carState* en *radarState* om ervoor te zorgen dat deze methode correct werkt.

*Momenteel zijn de eerste twee regels uitgecommentarieerd. Dit is een check om het creëren van een nieuwe *tailgatingStatus* over te slaan wanneer *radarState* niet geüpdatet is. Deze check bevindt zich op dit moment in de methode *update* van de klasse *CoachD*.

```
50   def update(
51       self, sm: messaging.SubMaster) -> log.DrivingCoachState.
       TailgatingStatus:
52     # if not sm.updated['radarState']:
53     #   return self.create_tailgating_status()
54
55     radar_state = sm['radarState']
56     current_time = sm.logMonoTime['radarState']
57     v_ego = sm['carState'].vEgo
58
59     # get closest lead
60     lead_one = radar_state.leadOne
61     lead_two = radar_state.leadTwo
62     closest_lead = get_closest_lead(lead_one, lead_two)
63
64     # determine if ego is tailgating
65     self.tailgating = is_tailgating(closest_lead.thw, v_ego)
66
67     # start measurement if tailgating and not measuring yet
68     if self.tailgating and not self.measuring:
69       self.start_measurement(current_time)
70     # stop measurement if not tailgating and still measuring
71     elif not self.tailgating and self.measuring:
72       self.stop_measurement()
73
74     # calculate duration (0 when not measuring)
75     self.duration = current_time - self.start_time if self.measuring else
       0
76
77     return self.create_tailgating_status()
78
```

create_tailgating_status

De methode *create_tailgating_status* creëert een *TailgatingStatus* datastructuur*, vult de velden met waarden en geeft deze terug.

*Eigenlijk creëert het een dictionary met dezelfde structuur als *TailgatingStatus*.

```
79   def create_tailgating_status(self) -> log.DrivingCoachState.
     TailgatingStatus:
80     return {
81       "active": True,
82       "isTailgating": bool(self.tailgating),
83       "duration": int(self.duration),
84       "warningLevel": int(self.warning_level)
85     }
86
```

start_measurement

De methode *start_measurement* start de meting en stelt de starttijd in.

```
87   def start_measurement(self, mono_time: int) -> None:
88     self.measuring = True
89     self.start_time = mono_time
90
```

stop_measurement

Met de methode *stop_measurement* wordt de meting gestopt en wordt de starttijd gereset.

```
91   def stop_measurement(self) -> None:
92     self.measuring = False
93     self.start_time = 0
94
```

Nu is het mogelijk om de huidige *tailgatingStatus* op te vragen.

3.4.4. test_tailgating_detection.py

openpilot/selfdrive/coachd/modules/tests/test_tailgating_detection.py

tailgating_detection.py wordt zorgvuldig getest aan de hand van een set unit tests. Er wordt onder andere getest op grenswaarden van THW en snelheid van het voertuig, of de *TailgatingStatus* met de juiste waarden wordt gevuld en of de meting correct wordt uitgevoerd. De *SubMaster* wordt hiervoor gemockt.

De unit tests worden hier verder niet behandeld, bekijk hiervoor de klasse *TestTailgatingDetection*.

3.4.5. coachd.py

openpilot/selfdrive/coachd/coachd.py

Bevat de implementatie van de initiële rijcoach. Deze heeft het doel om het *drivingCoachState* bericht te publiceren.

Shebang en imports

Eerst een 'shebang', vervolgens een aantal imports:

- *Dict*, *List*, *Optional* en *Type* worden gebruikt voor optionele type hints;
- *log* en *messaging* idem;
- *CoachModule* idem;
- *TailgatingDetection* voor het uitvoeren van de bumperkleef detectie.

```
1  #!/usr/bin/env python3
2  from typing import Dict, List, Optional, Type
3
4  from cereal import log, messaging
5  from selfdrive.coachd.modules.base import CoachModule
6  from selfdrive.coachd.modules.tailgating_detection import
   TailgatingDetection
```

Constanten

De dictionary *COACH_MODULES* bevat de standaard modules die uitgevoerd moeten worden door de rijcoach. De sleutel moet overeenkomen met een veld uit het *DrivingCoachState* capnp schema.

VALIDATED_SERVICES is een lijst met de standaard services die gevalideerd moeten worden tijdens het publiceren van een *drivingCoachState* bericht.

```
9  # key must match name of coresponding field in DrivingCoachState schema
10 COACH_MODULES: Dict[str, Type[CoachModule]] = {
11   # fieldname: module
12   "tailgatingStatus": TailgatingDetection,
13 }
14
15 # services that need to be validated to make a DrivingCoachState message
   valid
16 VALIDATED_SERVICES = ['carState', 'radarState']
```

CoachD

De klasse *CoachD* is het hart van de rijcoach. De constructor stelt de modules en services in die gevalideerd moeten worden. Als er geen *modules* wordt meegegeven, dan wordt hiervoor *COACH_MODULES* gebruikt. Idem voor *validated_services*: wanneer deze leeg is wordt *VALIDATED_SERVICES* gebruikt. Op deze manier is het mogelijk om deze dependencies te injecteren (bijvoorbeeld voor test doeleinden).

Ook wordt gebruik gemaakt van 'lazy instantiation': de modules worden pas geïnitialiseerd tijdens het creëren van een *CoachD* object.

```
↑
19 class CoachD(object):
20
21   def __init__(
22       self,
23       modules: Optional[Dict[str, Type[CoachModule]]] = None,
24       validated_services: Optional[List[str]] = None
25   ):
26     # initialize modules
27     if modules is None:
28       modules = COACH_MODULES
29     self.modules = {field: module() for field, module in modules.items()}
30
31     # set services that need validation
32     if validated_services is None:
33       validated_services = VALIDATED_SERVICES
34     self.validated_services = validated_services
35
```

is_module_active

De methode *is_module_active* geeft aan of een gegeven module momenteel actief is:

```
36   def is_module_active(self, module: str) -> bool:
37     return module in self.modules
38
```

update

Een *drivingCoachState* bericht wordt gecreëerd en van waarden voorzien door de methode *update*. Eerst maakt het een nieuw bericht. Dit is een instantie van de *Event* struct* en bevat:

- *logMonoTime*, met de huidige tijd (since het opstarten);
- *valid*, om aan te geven of het bericht geldig is;
- *drivingCoachState*, welke een instantie van de *DrivingCoachState* struct bevat.

*Van het type *_DynamicStructBuilder*, een wrapper voor de Cap'n Proto C++ *DynamicStruct::Builder* (7).

Daarna worden de *carState* en *radarState* services gevalideerd met *all checks* (*service_alive*, *frequency_ok* en *service_valid*): als één van deze checks mislukt is dus een *drivingCoachState* bericht niet geldig.

Vervolgens wordt er geïtereerd over de modules (momenteel alleen de bumperkleef detectie). Het bijbehorende *drivingCoachState* veld wordt gevuld met de waarde die teruggegeven wordt door het aanroepen van de *update* methode van de module.

Als laatst wordt het bericht teruggeven.

```
39   def update(self, sm: messaging.SubMaster) -> log.Event:
40     dat = messaging.new_message('drivingCoachState')
41
42     # validate services
43     dat.valid = sm.all_checks(service_list=self.validated_services)
44
45     # fill drivingCoachState fields
46     drivingCoachState = dat.drivingCoachState
47     for field, module in self.modules.items():
48       setattr(drivingCoachState, field, module.update(sm))
49
50     return dat
```

coachd_thread

De functie *coach_thread* begint met het instellen van de communicatie. Er wordt geabonneerd op *carState* en *radarState*, en *drivingCoachState* zal worden gepubliceerd.* Daarna wordt *CoachD* geïnitialiseerd (met de standaard *COACH_MODULES* en *VALIDATED_SERVICES*).

*Tenzij *SubMaster* en/of *PubMaster* zijn geïnjecteerd, bijvoorbeeld voor test doeleinden.

Vervolgens zal de thread continue blijven draaien, totdat openpilot wordt gestopt. Bij elke stap zal de *SubMaster* de staat van *carState* en *radarState* updaten. Als radarState in de tussentijd geen update heeft gehad, heeft het geen nut een nieuw bericht te publiceren; dat is immers verspilling van resources (Deze stap zal wellicht verwijderd of aangepast moeten worden als er nieuwe modules worden toegevoegd).

Wanneer *radarState* wél is geüpdatet, dan wordt de *drivingCoachState* geüpdatet en wordt dit bericht gepubliceert met de *PubMaster*.

```
53 def coachd_thread(
54     sm: Optional[messaging.SubMaster] = None,
55     pm: Optional[messaging.PubMaster] = None
56 ):
57   # *** setup messaging ***
58   if sm is None:
59     sm = messaging.SubMaster(['carState', 'radarState'])
60   if pm is None:
61     pm = messaging.PubMaster(['drivingCoachState'])
62
63   CD = CoachD()
64
65   while True:
66     sm.update()
67
68     # not necessary to publish new message if radarState not updated
69     # (if this check needs to be renoved, move it to 'update' method of
       TailgatingDetection)
70     if not sm.updated['radarState']:
71       continue
72
73     # *** publish drivingCoachState ***
74     dat = CD.update(sm)
75     pm.send('drivingCoachState', dat)
```

main

Wanneer het *coachd* Python script uitgevoerd wordt zal de functie *main* aan geroepen worden. Bij een normale opstart van openpilot zullen de argumenten voor *SubMaster* en *PubMaster* leeg zijn.

De functie *coachd_thread* wordt vervolgens aangeroepen: de *coachd* service zal nu starten.

```
78 def main(
79     sm: Optional[messaging.SubMaster] = None,
80     pm: Optional[messaging.PubMaster] = None
81 ):
82   coachd_thread(sm, pm)
83
84
85 if __name__ == "__main__":
86   main()
87
```

Nu is het mogelijk om de rijcoach met de bumperkleef detectie te starten als proces.

3.4.6. test_coachd.py

openpilot/selfdrive/coachd/tests/test_coachd.py

Ook *coachd.py* wordt zorgvuldig getest aan de hand van een set unit tests. Er wordt onder andere getest of *DrivingCoachState* en alle keys van de *COACH_MODULES* dictionary zijn gedefinieerd in het *log.capnp* Cap'n Proto schema, of alle standaard modules de methode *update* implementeren en of het initialiseren van *CoachD* met andere modules correct wordt uitgevoerd.

De unit tests worden hier verder niet behandeld, bekijk hiervoor de klasse *TestCoachD*.

3.5. manager

De manager zorgt ervoor dat de services gestart en gestopt worden. de *coachd* service moet hier dus aan toegevoegd worden.

3.5.1. process_config.py

openpilot/selfdrive/manager/process_config.py

coachd wordt als *PythonProcess* toegevoegd met het pad naar de module waar de main functie zich bevindt.

```
20 procs = [
60   # Driving coach
61   PythonProcess("coachd", "selfdrive.coachd.coachd"),
65 ]
```

3.5.2. manager.py

openpilot/selfdrive/manager/manager.py

Naast het toevoegen van de *coachd* service wordt er ook een extra standaard parameter toegevoegd. Deze parameters worden ingeladen wanneer openpilot start (mits de gebruiker deze waarde niet zelf handmatig heeft aangepast).

IsMetric zorgt ervoor dat standaard het metrisch systeem gebruikt wordt in plaats van het Brits-Amerikaanse (Imperial) maatsysteem.

```
29 def manager_init() -> None:
39   default_params: List[Tuple[str, Union[str, bytes]]] = [
43     ("IsMetric", "1"),
46   ]
```

Nu wordt de bumperkleef detectie uitgevoerd bij het starten van openpilot en is mogelijk om te abonneren op het *drivingCoachState* bericht.

3.6. controlsd

De *controlsd* service is onder andere verantwoordelijk voor het afvuren van events. De service moet er dus voor zorgen dat de bumperkleefwaarschuwingen afgevuurd worden wanneer nodig.

3.6.1. events.py

openpilot/selfdrive/controls/lib/events.py

De drie waarschuwingen worden gedefinieerd binnen *EVENTS*. Dit zijn *tailgating*, *promptTailgating* en *persistentTailgating*.

```
334 EVENTS: Dict[int, Dict[str, Union[Alert, AlertCallbackType]]] = {
943   # ********** events for driving coach **********
944
945   EventName.tailgating: {
946     ET.PERMANENT: Alert(
947       "",
948       "",
949       AlertStatus.normal, AlertSize.none,
950       Priority.LOWER, VisualAlert.none, AudibleAlert.prompt, .1),
951   },
952
953   EventName.promptTailgating: {
954     ET.PERMANENT: Alert(
955       "Tailgating",
956       "",
957       AlertStatus.normal, AlertSize.small,
958       Priority.LOW, VisualAlert.none, AudibleAlert.prompt, .1),
959   },
960
961   EventName.persistentTailgating: {
962     ET.PERMANENT: Alert(
963       "TAILGATING",
964       "",
965       AlertStatus.userPrompt, AlertSize.small,
966       Priority.MID, VisualAlert.fcw, AudibleAlert.promptRepeat, .1),
967   },
968
969 }
970
```

3.6.2. controlsd.py

openpilot/selfdrive/controls/controlsd.py

Constructor

drivingCoachState moet toegevoegd worden aan de lijst van services waar de klasse *Controls* op abonneert.

```
63  class Controls:
64    def __init__(self, sm=None, pm=None, can_sock=None, CI=None):
99      if self.sm is None:
105       self.sm = messaging.SubMaster(['deviceState', 'pandaStates',
          'peripheralState', 'modelV2', 'liveCalibration',
106                                      'driverMonitoringState',
                                         'longitudinalPlan', 'lateralPlan',
                                         'liveLocationKalman',
107                                      'managerState', 'liveParameters',
                                         'radarState', 'drivingCoachState'] +
                                          self.camera_packets +
                                          joystick_packet,
108                                      ignore_alive=ignore, ignore_avg_freq=
                                         ['radarState', 'longitudinalPlan'])
```

update_events

De methode *update_events* moet aangepast worden, zodat het de waarschuwingen toevoegt aan *events*, wanneer nodig. Deze actie wordt alleen uitgevoerd als de bumperkleef module actief is.

Als de module actief is wordt er aan de hand van het waarschuwingsniveau de correcte waarschuwing toegevoegd aan *events*.

```
221   def update_events(self, CS):
425     # create tailgating events only when module is active
426     tailgating_status = self.sm['drivingCoachState'].tailgatingStatus
427     if (tailgating_status.active
428         and (level := tailgating_status.warningLevel) != 0):
429       switch = {1: EventName.tailgating,
430                 2: EventName.promptTailgating,
431                 3: EventName.persistentTailgating}
432       self.events.add(switch.get(level))
```

Nu zullen er auditieve en visuele waarschuwingen hoor- en zichtbaar zijn wanneer het waarschuwingsniveau minimaal niveau één heeft bereikt.

3.7. ui

De *ui* service is verantwoordelijk voor het presenteren van de user interface. Het is geïmplementeerd aan de hand van het *Qt Framework* (8). De service moet ervoor zorgen dat het waarschuwingsteken zichtbaar is als dat nodig is.

3.7.1. ui.cc

openpilot/selfdrive/ui/ui.cc

drivingCoachState moet toegevoegd worden aan de lijst van services waar de klasse *UIState* op abonneert:

```
229 UIState::UIState(QObject *parent) : QObject(parent) {
230   sm = std::make_unique<SubMaster, const std::initializer_list<const char
      *>>({
231     "modelV2", "controlsState", "liveCalibration", "radarState",
      "deviceState", "roadCameraState",
232     "pandaStates", "carParams", "driverMonitoringState", "sensorEvents",
      "carState", "liveLocationKalman",
233     "wideRoadCameraState", "managerState", "navInstruction", "navRoute",
      "gnssMeasurements", "drivingCoachState",
234   });
244 }
```

3.7.2. onroad.h

openpilot/selfdrive/ui/qt/onroad.h

Eerst volgen de declaraties in de *onroad* header file. Aan de klasse *NvgWindow* wordt een nieuwe *Q_PROPERTY* toegevoegd: *showTailgatingWarning* (9). Verder bevat het een *QPixmap* voor het waarschuwingsteken en is de grootte gedefinieerd (10).

```
28 class NvgWindow : public CameraViewWidget {
45   // driving coach
46   Q_PROPERTY(bool showTailgatingWarning MEMBER showTailgatingWarning);
52 private:
75   // driving coach
76   QPixmap tailgating_img;
77   const int tailgating_img_size = img_size / 1.5 * 1.65;
78   bool showTailgatingWarning = false;
94 };
```

3.7.3. onroad.cc

openpilot/selfdrive/ui/qt/onroad.cc

De implementatie is te vinden in de *onroad* source file.

Constructor

In de constructor van *NvgWindow* wordt het waarschuwingsteken ingeladen met het pad naar de asset en gedefinieerde grootte:

```
173 NvgWindow::NvgWindow(VisionStreamType type, QWidget* parent) : fps_fil-
    ter(UI_FREQ, 3, 1. / UI_FREQ), CameraViewWidget("camerad", type, true,
    parent) {
176   tailgating_img = loadPixmap("../assets/coach/tailgating_warning.svg",
      {tailgating_img_size, tailgating_img_size});
177 }
```

updateState

De methode *updateState* wordt uitgebreid. Eerst wordt *tailgatingStatus* opgevraagd van *drivingCoachState* via de *SubMaster*. Wanneer deze actief is wordt de property *showTailgatingWarning* geüpdatet. De waarde wordt afgeleid door te kijken naar het waarschuwingsniveau. Wanneer deze niet nul is, dan moet het waarschuwingsteken zichtbaar zijn.

```
179 void NvgWindow::updateState(const UIState &s) {
236   // driving coach
237   // show tailgating warning icon only when module is active
238   auto TS = sm["drivingCoachState"].getDrivingCoachState().
      getTailgatingStatus();
239   if (TS.getActive())
240     setProperty("showTailgatingWarning", TS.getWarningLevel() != 0);
241 }
```

drawHud

De methode *drawHud* wordt ook uitgebreid. Wanneer het waarschuwingsteken zichtbaar moet zijn wordt deze getekend op het scherm.

```
243 void NvgWindow::drawHud(QPainter &p) {
404   // tailgating warning icon
405   if (showTailgatingWarning) {
406     int x = rect().right() - radius / (1.5 * 1.65) - (bdr_s * 2);
407     int y = rect().top() + header_h;
408     p.drawPixmap(x - radius / 2, y - radius / 2, tailgating_img);
409   }
410 }
```

Nu zal het waarschuwingsteken, wanneer minimaal niveau één is bereikt zichtbaar zijn op het scherm, aan de rechterkant, onder het *engageable* icoon.

4. Simulator

Om openpilot te testen wordt gebruik gemaakt van een simulator: CARLA (11). Hier is niet veel aan veranderd: twee scripts zijn iets aangepast.

4.1. launch_openpilot.sh

openpilot/tools/sim/launch_openpilot.sh

Normaliter wordt de *loggerd* service geblokkeerd wanneer openpilot in simulatiemodus wordt gestart. Door deze uit de environment variabele *BLOCK* te halen worden er nu wel logs gegenereerd:

```
 8 export BLOCK="camerad,encoderd"
```

4.2. start_carla.sh

openpilot/tools/sim/launch_openpilot.sh

start_carla.sh is een script dat de CARLA simulator in een docker container start. Normaliter wordt hier de standaard image gebruikt die beschikbaar is op Docker hub (12). Echter, deze image bevat niet de extra assets en maps die je kan importeren (13). Daarom wordt er een image opgehaald die deze wel bevat: *carla-complete* (14).

```
18 docker pull ghcr.io/jeroenlammersma/carla-complete:0.9.13

26 docker run \

33   ghcr.io/jeroenlammersma/carla-complete:0.9.13 \
```

5. Continuous integration

github.com/jeroenlammersma/openpilot/actions

Naast het implementeren van de bumperkleef detectie- en waarschuwingsservice is ook nagenoeg de volledige continuous integration (CI) setup van de openpilot en cereal repositories overgenomen. Hiervoor wordt gebruikt gemaakt van *GitHub Actions* (15). Deze moeten bij het aanmaken van een fork wel eerst handmatig ingeschakeld worden.

5.1. Workflows

CI wordt bij GitHub Actions geïmplementeerd met *workflows*, een configureerbaar en automatisch proces dat één of meerdere taken (jobs) uitvoert op een virtual machine. Ze kunnen op meerdere manier worden gestart, zowel handmatig als via events.

Van beide forks zal iedere workflow kort worden beschreven, zonder inhoudelijk in te gaan op de exacte werking en configuratie.

5.1.1. openpilot

De openpilot repository bevat vijf workflows: *badges*, *prebuilt*, *release*, *selfdrive* en *tools*.

badges, *prebuilt* en *release worden* door middel van *Cron Jobs* elk uur gestart. *cron* is een command-line programma waarmee taken gepland mee kunnen worden (job scheduler) (16). Daarnaast zijn ze ook handmatig te starten.

selfdrive en *tools* worden gestart wanneer een commit of tag gepusht wordt of wanneer er activiteit plaatsvindt in een pull request.

badges

openpilot/.github/workflows/badges.yaml

Deze workflow creëert SVG badges in de branch *badges*, waarmee de progressie van de vertalingen mee getoond kan worden.

prebuilt

openpilot/.github/workflows/prebuilt.yaml

De pre-built workflow bouwt en pusht de Docker image *openpilot-prebuilt* naar de GitHub Container Registry.

release

openpilot/.github/workflows/release.yaml

Deze workflow bouwt *master-ci* in de branch *master-ci*.

selfdrive

openpilot/.github/workflows/selfdrive_tests.yaml

De selfdrive workflow voert de volgende taken uit:

- Bouwen van openpilot en uitvoeren van checks.
- Bouwen van openpilot met alle options (flags).

- Bouwen en pushen van de volgende Docker images naar de GitHub Container Registry:
  1. *openpilot-base*;
  2. *openpilot-base-ci*;
  3. *openpilot-docs*.
- Uitvoeren van een statische analyse met de pre-commit hooks (zoals *pylint*, *cppcheck*, et cetera).
- Uitvoeren van Valgrind, een tool dat memory leaks en memory errors kan detecteren.
- Uitvoeren van unit tests.
- Uitvoeren van *process replay* en *model_replay_onyx* waarbij het rijden van routes gesimuleerd worden en gekeken wordt of de huidige build afwijkt.
- Testen van longitudinale manoeuvres.
- Uitvoeren van testen op de door openpilot ondersteunde automodellen.

tools

openpilot/.github/workflows/tools_tests.yaml

Deze workflow voert unit tests uit op PlotJuggler, een tool dat openpilot gebruikt om de logs mee te plotten (17). Verder bouwt en pusht het de Docker image *openpilot-sim* naar de GitHub Container Registry.

5.1.2. cereal

github.com/jeroenlammersma/cereal/actions

cereal bevat een enkele workflow: *tests*. Deze wordt gestart wanneer een commit of tag gepusht wordt of wanneer er activiteit plaatsvindt in een pull request.

tests

openpilot/cereal/.github/workflows/tests.yaml

De tests workflow voert de volgende taken uit:

- Bouwen en pushen van de Docker image *cereal* naar de GitHub Container Registry.
- Uitvoeren van unit tests.
- Uitvoeren van een statische analyse met de pre-commit hooks (zoals *pylint*, *cppcheck*, et cetera).

5.2. Aanpassingen

De workflows en andere relevante bestanden zijn iets aangepast om ervoor te zorgen dat de CI correct wordt uitgevoerd op de forks.

- De geplande acties (*badges*, *prebuilt* en *release*) worden niet meer via een Cron Job gestart, omdat ze (op dit moment) niet nodig zijn. Ze zijn eventueel nog wel handmatig te starten.
- De username 'commaai' is veranderd naar '{{ github.repository_owner }}' (environment variabele) in:
  1. de url van de Docker registry, zodat de images worden gepusht naar de eigenaar van de repository;

2. de string 'commaai/openpilot', tijdens de check of de workflow in deze repository is gestart, zodat de Docker push actions worden uitgevoerd.

- Omdat de naam van de default branch is gewijzigd wordt 'master' veranderd naar 'driving-coach' in de string 'refs/heads/master' tijdens de check door welke branch de workflow is gestart. Hierdoor worden onder andere de Docker push actions uitgevoerd op de default branch.
- In de Dockerfiles, scripts en andere relevante bestanden is de username veranderd van 'commaai' naar 'jeroenlammersma'.
- In het *check_modules.sh* script, dat van iedere submodule controleert of deze uitgecheckt is op een commit dat ook in de master branch van deze submodule aanwezig is, wordt ervoor gezorgd dat er bij cereal gecontroleerd wordt op de branch 'driving-coach', in plaats van 'master'.

6. Contributies

Er zijn ook een aantal pull requests gedaan naar de openpilot repository van comma.ai. Hiervan zijn ook al een paar gemerged met de master branch

6.1. Update CARLA versie

Updated CARLA to v0.9.13

Een kleine aanpassing: de CARLA versie is van versie 0.9.12 geüpdatet naar versie 0.9.13 (op het moment van schrijven de meest recente uitgebrachte versie).

Deze pull request is gemerged met commaai:master.

6.2. ScenarioRunner attribuut

sim: role_name attribute used by ScenarioRunner for CARLA

Wederom een kleine aanpassing. Aan het ego voertuig (in de simulator) is het attribuut *role_name* toegevoegd en is voorzien van de waarde 'hero'. Zonder dit attribuut kan ScenarioRunner het ego voertuig niet herkennen.

Deze pull request is gemerged met commaai:master.

6.3. Nederlandse vertalingen

Add Dutch translations (Nederlands)

Een significantere aanpassing: het toevoegen van Nederlandse vertalingen. Hiervoor wordt gebruik gemaakt van Qt Linguist (18). Nu is het mogelijk om de taal voor openpilot te wijzigen naar Nederlands.

Deze pull request is gemerged met commaai:master.

6.4. Videoframe formaat conversie

sim: outputting frames in NV12 format

openpilot zet camerabeelden om van RGB naar YUV NV12. YUV is een video pixel formaat: Y staat voor het helderheidscomponent (luminance) en U en V voor de kleurcomponenten (chrominance) (19).

Tijdens het ontwikkelen van de bumperkleef detectie- en waarschuwingsservice was dit formaat echter gewijzigd van I420 naar NV12 (varianten van het YUV formaat).

I420

Bij I420 worden de Y, U en V waarden in vectoren opeenvolgend opgeslagen:

YYYYYYYY UU VV (voor een *n*-pixel I420 frame: Y×8×*n* U×2×*n* V×2×*n*)

NV12

Dit formaat lijkt erg op I420, het begint ook met de Y vector, maar vervolgens volgt hierop een enkele vector met de UV waarden, waarbij de waarden van U en V verweven zijn:

YYYYYYYY UVUV (Voor een *n*-pixel NV12 frame: Y×8×*n* (UV)×2×*n*)

Verkeerde conversie

Momenteel is het probleem dat de simulator nog steeds RGB naar I420 converteert. Als gevolg hiervan worden beelden weergegeven in grijsschalen. Om dit op te lossen moet de *bridge* (middleware tussen CARLA en openpilot) aangepast worden zodat de frames omgezet worden naar NV12.

OpenCL

De *bridge* maakt gebruik van een *OpenCL* programma om de conversie van RGB naar I420 uit te voeren (20). Dit zorgt voor erg hoge performance, omdat hierbij de kracht van parallellisatie goed wordt benuttigd. Er is geprobeerd een nieuw *OpenCL* programma te schrijven die de conversie naar NV12 uitvoert. Door tijdgebrek en compilatieproblemen is dit werk (helaas) gestaakt.

NumPy

Om toch de simulator beelden terug in kleur te krijgen is er een (tijdelijke) oplossing voorgesteld dat gebruik maakt van *NumPy* (21). In de *bridge* wordt een I420 frame al opgeslagen in een numpy array (als rijvector), alvorens deze wordt gestuurd naar de *vision interprocess communication server* (VisionIPCServer), die de uitwisseling van visuele data verzorgt (onderdeel van cereal).

Door nu wat slimme vector berekeningen uit te voeren op deze array is een frame in principe redelijk eenvoudig omgezet naar het NV12 formaat; de ordering van de U en V waarden hoeft namelijk alleen aangepast te worden. Wel is het hierbij belangrijk om alleen gebruik te maken van de ingebouwde functies van NumPy, omdat deze geoptimaliseerd zijn voor operaties op vectoren en matrices (en daarnaast ook parallel uitgevoerd).

Op het moment van schrijven is deze pull request nog open en dus nog niet gemerged met commaai:master.

bridge.py

openpilot/tools/sim/bridge.py

Om een I420 frame om te zetten naar NV12 met NumPy moeten de volgende stappen worden uitgevoerd:

1. Bereken de grens tussen Y en U vectoren in de YUV array.*
2. Extraheer de Y vector als sub-array Y (alle waarden tot de grens).
3. Extraheer de U en V vectoren als sub-array UV (alle waarden ná de grens):
4. Vorm UV om naar een $2 \times n$ matrix, waarbij n wordt afgeleid uit de lengte van UV en aantal rijen (door -1 gebruiken als tweede argument bij *reshape* wordt dit automatisch berekend). Nu zullen alle U waarden zich in de eerste rij bevinden en alle V waarden in de tweede.
5. Vorm de matrix om naar een aaneengesloten rijvector (flattened) en gebruik hierbij *Fortran-style* ordering (column-major order) (22). Hierdoor worden de U en V waarden telkens 'om en om' geplaatst.
6. Maak het NV12 frame door de Y en UV samen te voegen als een aaneengesloten rijvector.

*Deze wordt berekend door de lengte en de breedte van het originele RGB frame met elkaar te vermenigvuldigen.

De Python code van bovenstaande instructies:

```
68  class Camerad:
99    def _cam_callback(self, image, frame_id, pub_type, yuv_type):
112     # convert YUV frame to NV12
113     uv_offset = H * W
114     y = yuv[:uv_offset]
115     uv = yuv[uv_offset:].reshape(2, -1).ravel('F')
116     nv12 = np.hstack((y, uv))
```

De simulator zal nu weer beelden in kleur tonen.

Op het moment van schrijven is deze pull request nog niet gemerged met commaai:master. De aanpassing is al wel doorgevoerd op de openpilot fork.

6.5. Condtioneel nvme bootlog commando

Add nvme bootlog command only when device is TICI

Tijdens het ontwikkelen is er aan de bootlog een vector van commando's toegevoegd. Deze wijziging zorgt ervoor dat openpilot blijft hangen bij de bootlog stap, tijdens het opstarten voor de simulator.

De oorzaak ligt bij het *nvme* commando, die uitgevoerd moet worden als superuser (sudo) voor de comma three hardware. Voor de simulator is dit commando niet relevant en daarom is ervoor gezorgd dat dit commando alleen wordt uitgevoerd als openpilot gestart is op de comma three (TICI).

Deze pull request is gemerged met commaai:master.

6.6. Verwijderen overbodige environment variables

CI: remove redundant env variables in 'openpilot env setup'

Eén van de workflow bestanden bevatte een drietal environment variabelen die overbodig gekopieerd waren uit een ander workflow bestand. Deze variabelen zijn op één plek verwijderd. Dit is beter, omdat ze tijdens een wijziging niet meer op twee plekken aangepast hoeven te worden: dit verminderd de kans op fouten.

Deze pull request is gemerged met commaai:master.

7. Ondersteunende repository

github.com/jeroenlammersma/openpilot-dev

Wat ooit begonnen is als een simpel bash script die het clonen van openpilot, het uitvoeren van de ubuntu setup, het bouwen van openpilot met SCons en het installeren van PlotJuggler automatiseerde, is uiteindelijk uitgegroeid tot een heuse ondersteunende repository: *openpilot-dev*.

7.1. openpilot-dev

Het doel van openpilot-dev is om een gebruiksvriendelijke ontwikkelomgeving te realiseren voor openpilot. Iemand die wil starten met het ontwikkelen aan openpilot (zoals een opvolger van het rijcoach project) kan hierdoor direct beginnen.

openpilot-dev bevat een interactieve setup dat het opzetten de ontwikkelomgeving versimpeld en automatiseert. Daarnaast bevat het ook een collectie aan handige scripts. Lees de README voor meer informatie (bevindt zich op de hoofdpagina van de GitHub repository).

Figuur 3: Impressie van de interactieve openpilot-dev setup.

Ook voor deze repository is de versie die opgeleverd is getagd onder de naam ‘graduation-project’. Controleer bij het bekijken van de repository of deze tag geselecteerd is (de link aan het begin van het hoofdstuk wijst naar deze versie).

7.2. Workflows

github.com/jeroenlammersma/openpilot-dev/actions

De repository bevat twee workflows: *build carla* en *run setup*.

7.2.1. build carla

openpilot-dev/.github/workflows/build_carla.yaml

Deze workflow moet handmatig worden gestart. Het bouwt en pusht de Docker image *carla-complete* naar de GitHub Container Registry (14). Dit is de image die *start_carla.sh* gebruikt (zoals beschreven in paragraaf 4.2).

7.2.2. run setup

openpilot-dev/.github/workflows/run_setup.yaml

run setup wordt gestart wanneer een commit of tag gepusht wordt of wanneer er activiteit plaatsvindt in een pull request. Het heeft als doel om te testen of de setup zonder fouten wordt uitgevoerd. Hiervoor voert het op meerdere omgevingen *setup.sh* uit, telkens met een andere option (flag).

Bij de *run_all* taak (met de flag --all) voert het de setup tweemaal achterelkaar uit, om te controleren of er bij het opnieuw uitvoeren ook geen fouten ontstaan.

7.3. Work in progress…

De ontwikkelomgeving is nog niet volledig afgemaakt: hier was geen tijd meer voor. Er is een *todo.txt* bestand in de root van de repository geplaatst waar nog wensen en vervolgstappen staan beschreven. Daarnaast staan her en der wat 'TODO' comments.

De setup is daarentegen wel gebruiksklaar en kan direct worden ingezet (los van eventuele verbeteringen).

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 augustus 19.
2. Google. Google Python Style Guide. GitHub. Beschikbaar via: https://google.github.io/styleguide/pyguide.html. Geraadpleegd 2022 augustus 10.
3. comma.ai. commaai/cereal: capnp struct definitions and messaging used in comma ecosystem. GitHub. Beschikbaar via: https://github.com/commaai/cereal/. Geraadpleegd 2022 augustus 19.
4. Cap'n Proto. Cap'n Proto: Introduction. Beschikbaar via: https://capnproto.org/. Geraadpleegd 2022 augustus 19.
5. Cap'n Proto. capnproto/pycapnp: Cap'n Proto serialization/RPC system - Python bindings. GitHub. Beschikbaar via: https://github.com/capnproto/pycapnp. Geraadpleegd 2022 augustus 19.
6. Python Software Foundation. Abstract Base Classes. Python 3.10.6 documentation. Beschikbaar via: https://docs.python.org/3.10/library/abc.html. Geraadpleegd 20 augustus 2022.
7. Paryani J, Alexander J. API Reference. capnp 1.0.0 documentation. Beschikbaar via: https://capnproto.github.io/pycapnp/capnp.html?highlight=_dynamicstructbuilder#capnp._DynamicStructBuilder. Geraadpleegd 2022 augustus 21.
8. The Qt Company. Cross-platform software development for embedded & desktop. Qt. Beschikbaar via: https://www.qt.io/. Geraadpleegd 2022 augustus 21.
9. The Qt Company. The Property System. Qt Documentation (Core 5.15.10). Beschikbaar via: https://doc.qt.io/qt-5/properties.html. Geraadpleegd 2022 augustus 21.
10. The Qt Company. QPixmap Class. Qt Documentation (GUI 5.15.10). Beschikbaar via: https://doc.qt.io/qt-5/qpixmap.html. Geraadpleegd 2022 augustus 21.
11. CARLA Team. CARLA Simulator. Beschikbaar via: https://carla.org/. Geraadpleegd 2022 augustus 21.
12. carlasim. carlasim/carla - Docker Image. Docker Hub. Beschikbaar via: https://hub.docker.com/r/carlasim/carla. Geraadpleegd 2022 augustus 21.
13. CARLA Team. Quick start package installation. CARLA Documentation. Beschikbaar via: https://carla.readthedocs.io/en/latest/start_quickstart/#import-additional-assets. Geraadpleegd 2022 augustus 21.

14. Lammersma J. jeroenlammersma/openpilot-dev: Package carla-complete. GitHub. Beschikbaar via: https://github.com/jeroenlammersma/openpilot-dev/pkgs/container/carla-complete. Geraadpleegd 2022 augustus 21.

15. GitHub, Inc. GitHub Actions Documentation. GitHub Docs. Beschikbaar via: https://docs.github.com/en/actions. Geraadpleegd 2022 augustus 21.

16. Canonical Ltd. cron - daemon to execute scheduled commands (Vixie Cron). Ubuntu Manpage. Beschikbaar via: https://manpages.ubuntu.com/manpages/focal/en/man8/cron.8.html. Geraadpleegd 2022 augustus 20.

17. Faconti D. facontidavide/PlotJuggler: The Time Series Visualization Tool that you deserve. GitHub. Beschikbaar via: https://github.com/facontidavide/PlotJuggler. Geraadpleegd 2022 augustus 21.

18. The Qt Company. Qt Linguist Manual. Qt Documentation. Beschikbaar via: https://doc.qt.io/qt-5/qtlinguist-index.html. Geraadpleegd 2022 augustus 21.

19. VideoLan. YUV. VideoLAN Wiki. Beschikbaar via: https://wiki.videolan.org/YUV. Geraadpleegd 2022 augustus 21.

20. The Khronos® Group Inc. OpenCL Overview. The Khronos Group Inc. Beschikbaar via: https://www.khronos.org/opencl/. Geraadpleegd 2022 augustus 19.

21. NumPy Developers. What is NumPy? NumPy v1.23 Manual. Beschikbaar via: https://numpy.org/doc/1.23/user/whatisnumpy.html. Geraadpleegd 2022 augustus 20.

22. Fortran Community. Multidimensional Arrays. Fortran Programming Language. Beschikbaar via: https://fortran-lang.org/learn/best_practices/multidim_arrays. Geraadpleegd 2022 augustus 21.