SLOVENSKÁ TECHNICKÁ UNIVERZITA

Fakulta elektrotechniky a informatiky

Užívateľská príručka a programová realizácia

Príloha diplomovej práce

Neuroevolúcia autonómneho vozidla

Bratislava, 13.5.2022

Bc. Marko Chylík

OBSAH

1 CARLA

V práci využívame stable verziu 0.9.12 – nájdeme ju na https://carla.org/2021/08/02/release-0.9.12/. V prípade, že je verzia už zastaralá, treba stiahnuť novšiu, pričom neskôr (REF!) netreba zabudnúť nainštalovať korešpondujúci Python API balíček pre danú verziu.

1.1 Pred stiahnutím

Ešte pred stiahnutím je dôležité si pozrieť minimálne požiadavky. Je však odporúčané, aby PC disponoval o dosť väčším výpočtovým výkonom, nakoľko okrem CARLA prostredia bude musieť bežať aj samotný riadiaci program.

1.1.1 Minimálne požiadavky na spustenie CARLA

Názov	*Minimálne*	*Odporúčané*
Operačný systém	Windows/Linux	Windows/Linux
GPU pamäť	6GB	8GB
Miesto na disku	20GB	30GB
RAM	8GB	16GB
Python	2.7	3.8
pip	20.3	najnovšia
Voľné TCP porty	2000, 2001	2000, 2001

1.2 Inštalácia (Windows)

V našom prípade sme použili operačný systém Windows. Bližšie popíšeme postup pri inštalácii pre tento OS. Návody nájdeme na: https://carla.readthedocs.io/en/latest/start_quickstart/

1. Stiahneme CARLA 0.9.12: https://carla-releases.s3.eu-west-3.amazonaws.com/Windows/CARLA_0.9.12.zip
2. Rozbalíme ZIP na disk
3. Pustíme *CarlaUE4.exe* aplikáciu
4. Otvoríme Windows terminál (cmd) – dostaneme sa do miesta, kde sme rozbalili
5. `cd PythonAPI\examples`
6. `python3 -m pip install -r requirements.txt`
7. `python3 manual_control.py`

Po týchto krokoch by sa malo otvoriť druhé okno, kde uvidíme vozidlo, ktoré bude ovládateľné za pomoci klávesnice

2 Projekt

Projekt je možné získať dvomi spôsobmi. Buď je priložený na USB kľúči spolu s touto dokumentáciou ako príloha diplomovej práce, alebo je možné ho stiahnuť na GitHube:

Path - `cesta, kde chceme mať projekt uložený`

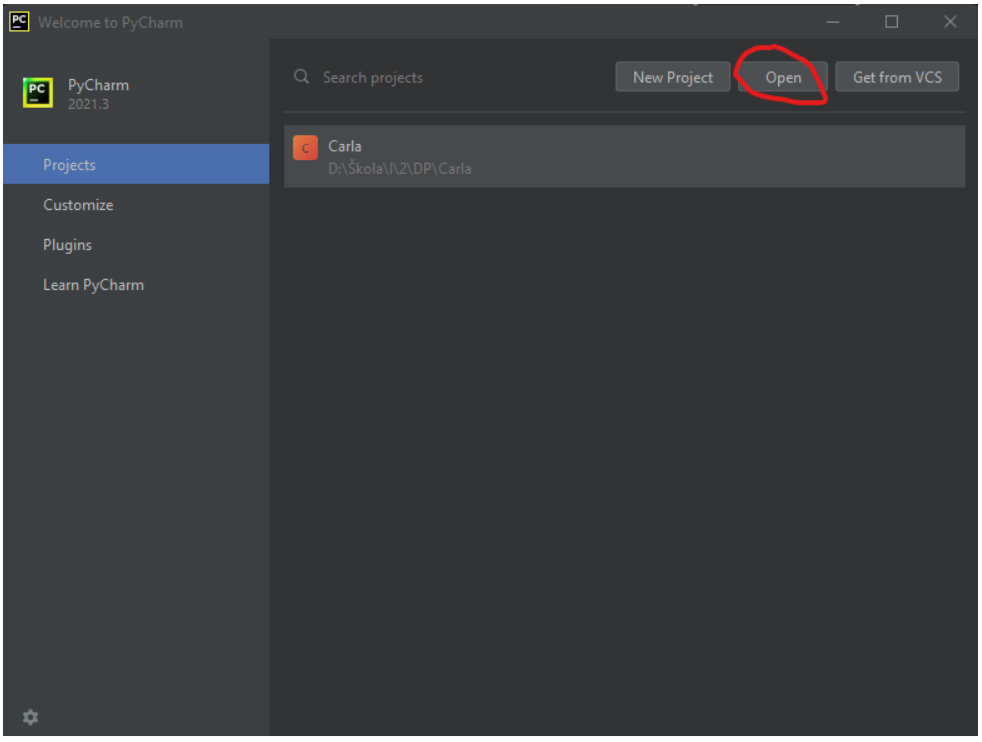
V tomto prípade vieme naklonovať projekt:

1. `cmd`
2. `cd path`
3. `git clone https://github.com/macricek/DT_Carla.git Carla`

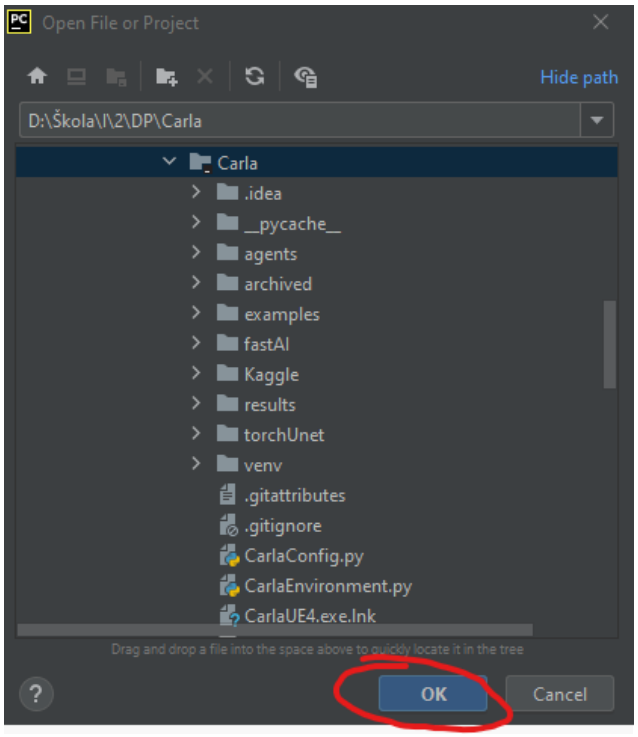
2.1 PyCharm

Ako vývojové prostredie sme použili *PyCharm Community*. Je to voľné dostupné IDE pre Python od českých tvorcov (IntelliJ). Disponuje množstvom výborných funkcionalít, či možností doinštalovania ďalších externých súčastí, čím vieme zrýchliť a zefektívniť našu prácu pri vývoji. **PyCharm je len odporúčeným IDE!**

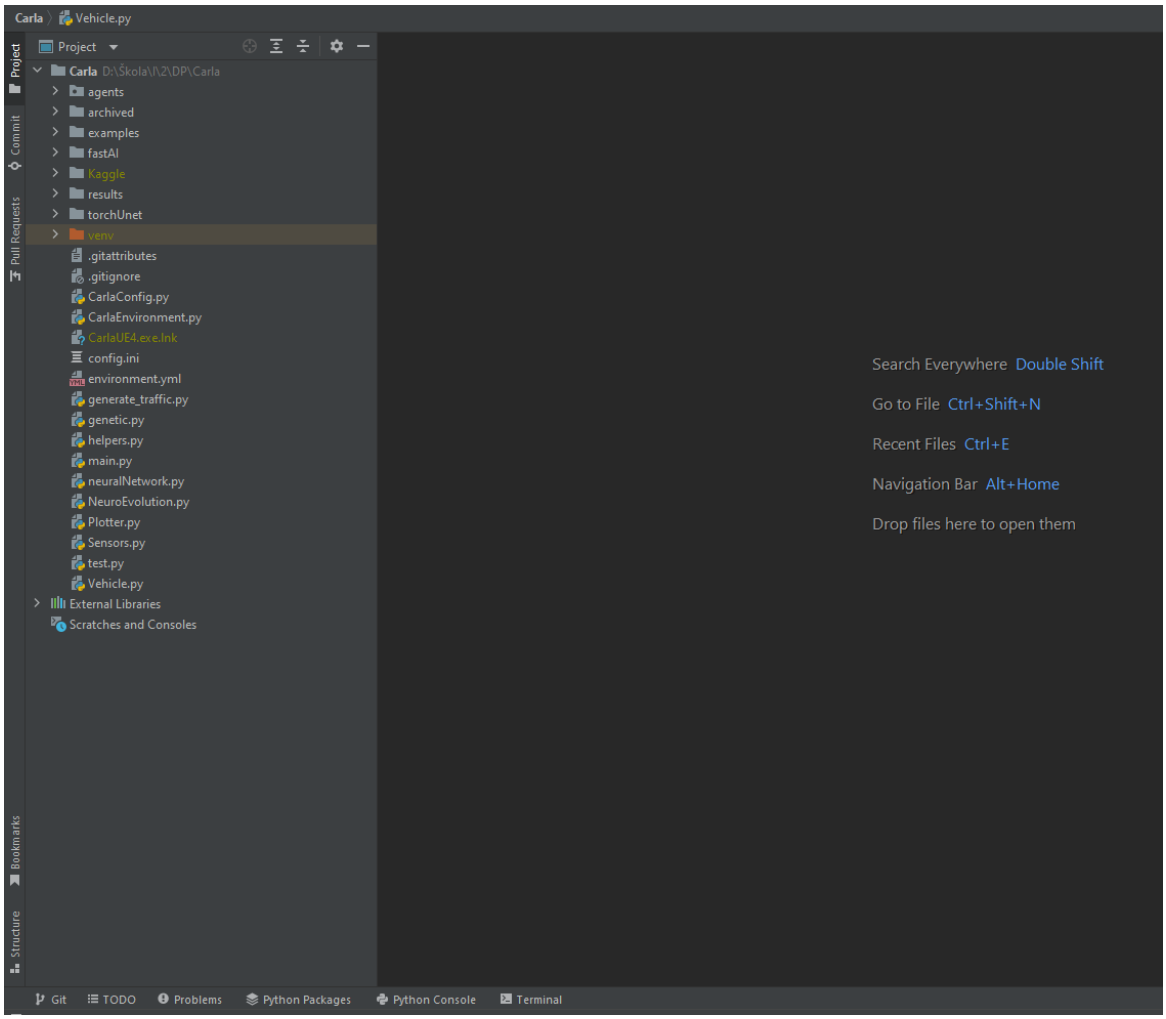
1. `Spustíme PyCharm, uvidíme takéto okno, klikneme na Open`



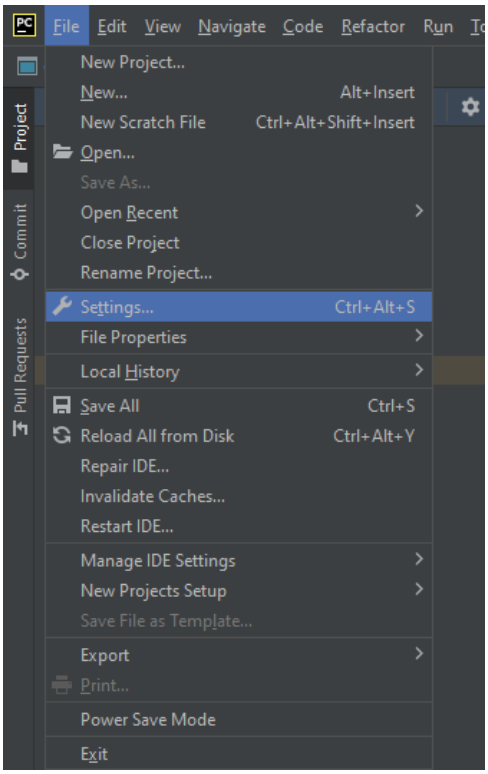
2. Nájdeme cestu k projektu a vyberieme hlavný adresár s názvom Carla, klikneme na OK

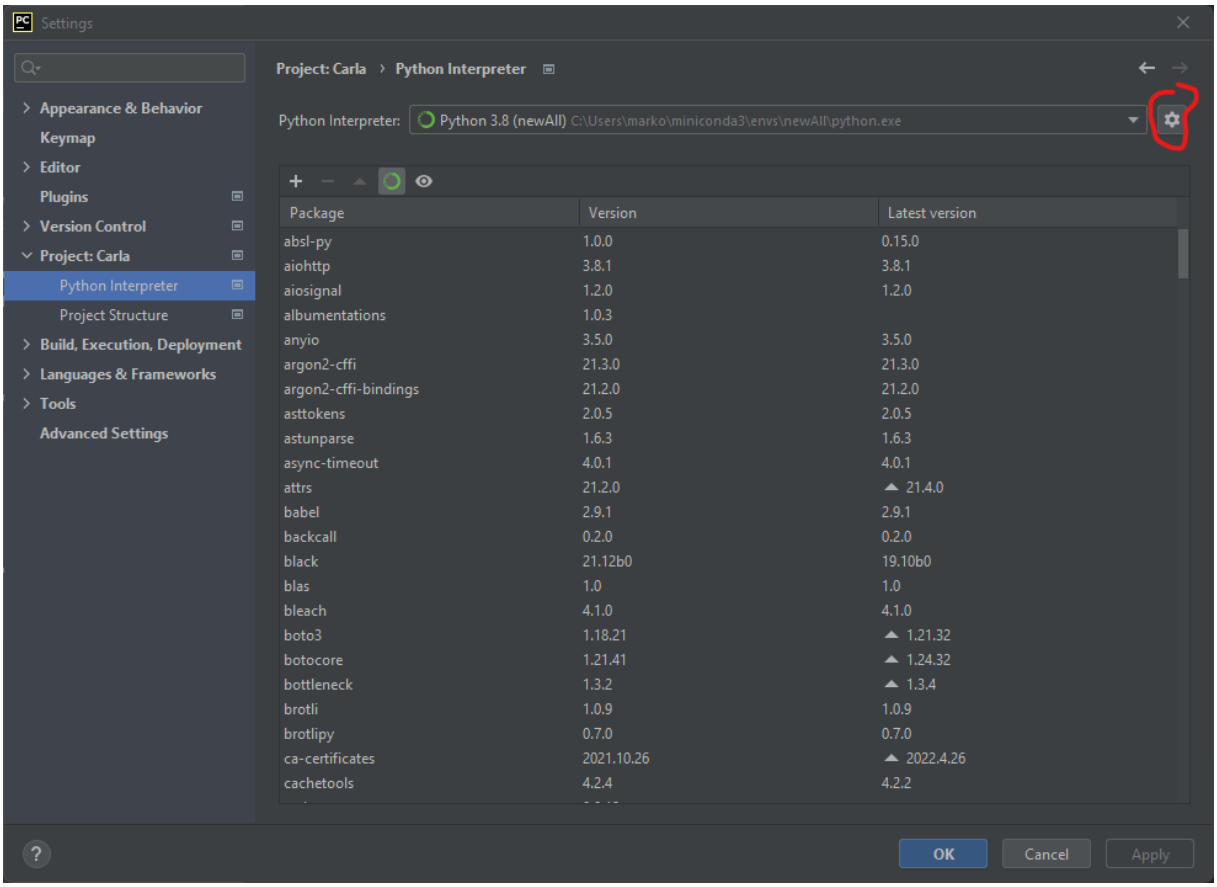


3. Po chvíli sa nám otvorí takáto štruktúra

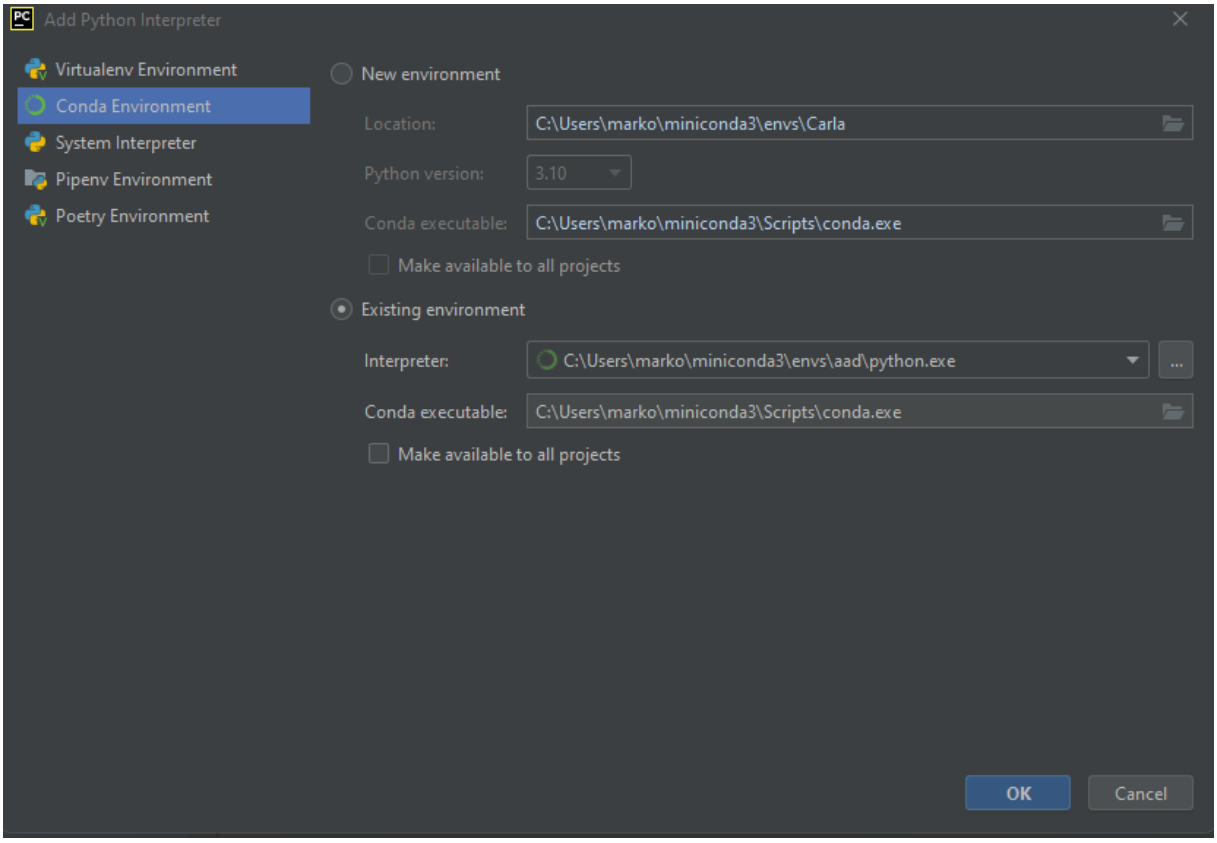


4. Nastavíme virtuálne prostredie (**TENTO KROK je možné urobiť až po dokončení 3.kapitoly**)

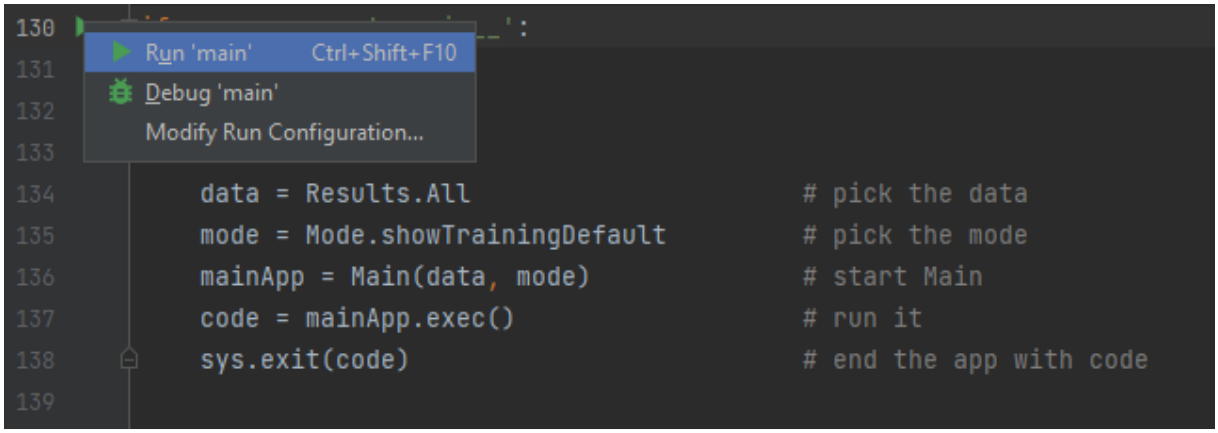




5. Vyberieme existing environment, ako interpreter vyberieme python.exe z Conda prostredia, ktoré sme vytvorili. Klikneme OK



6. Otvoríme main.py, zídeme nižšie:



Virtuálne prostredie sa aktivuje automaticky a všetky súčasti projektu budú spustiteľné.

3 Conda Environment

Na zabezpečenie rovnakej konfigurácie akou disponujeme my pri tvorení tejto diplomovej práce potrebujeme virtuálne prostredie pre Python. Využívame veľa externých súčastí, ktoré by bolo inak potrebné inštalovať ručne. Vhodnou alternatívou bude *Conda Environment.* Dovolí nám vytvoriť prostredie, v ktorom budeme mať nainštalované konkrétne verzie súčastí, ktoré v projekte využívame a malo by to zabezpečiť spätnú kompatibilitu.

Ešte predtým však potrebujeme nainštalovať konkrétnu verziu Pythonu, ktorú budeme chcieť využiť. V našom prípade to je 3.8.0:

https://www.python.org/downloads/release/python-380/

3.1 Anaconda prompt 3 (Miniconda)

Na linku https://docs.conda.io/en/latest/miniconda.html si nájdeme Minicondu podľa OS, nainštalujeme ju a spustíme. Otvorí sa nám miniconda:

1. `cd path/Carla`
2. `conda env create -f environment.yml python=3.8.0`
3. `conda activate newAll`

Po tomto postupe budeme mať vytvorené prostredie **newAll**. Proces tvorby prostredia chvíľku trvá, keďže sa musia posťahovať a nainštalovať všetky potrebné balíčky.

3.2 Inštalácia CARLA API

V originálnom prostredí, ktoré je v projekte a ktoré sme importovali v minulej sekcii je verzia 0.9.12. V prípade, že používateľ chce používať inú verziu CARLA, musí preinštalovať túto verziu, keďže rovnaké verzie sú nutná podmienka. Zoznam verzii nájdeme:

https://pypi.org/project/carla/#history

1. `conda activate newAll`
2. `pip3 uninstall carla`
3. `pip3 install carla==0.9.x (x podľa vybranej verzie)`

4 Segmentačné siete

Ako bolo spomínané aj v samotnej diplomovej práci, v projekte sú dve segmentačné neurónové siete: **Unet++** a **MobileNetV3Small**. Na úvod je dôležité poznamenať, že na optimálnu funkcionalitu týchto súčastí je potrebné stiahnuť nasledujúci dataset:

https://www.kaggle.com/datasets/thomasfermi/lane-detection-for-carla-driving-simulator/download

Potom je potrebné rozbaliť .zip súbor do projektu, konkrétne do základnému súboru s názvom **Kaggle** (čo je teda názov stránky, odkiaľ dataset pochádza). Očakávaná súborová štruktúra bude vyzerať teda nejako takto:

[Pozn. na verzii z USB projekt disponuje aj datasetom]

4.1 Unet++

Nájdeme ju v priečinku *torchUnet*, ktorý obsahuje natrénovanú sieť v *unet_model.pth* a *UnetLineDetection.py*, ktorý slúži na natrénovanie. Používa framework **PyTorch**.

4.1.1 TransformImage

Funkcia na transformáciu obrázku pomocou *Albumentations*, využijeme len operáciu normalizácie. Samozrejme, je potrebné zmeniť obrázok aj masku rovnakou transformáciou.

```
testtransform = A.Compose([
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
def transformImage(image, transformation=testtransform, mask=None):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    if transformation:
        transformed = transformation(image=image, mask=mask)
        img = transformed['image']
        mask = transformed['mask'].long()
    return img, mask
```

4.1.2 LineDetectionDataset

Táto trieda dedí od triedy Dataset, ktorá je priamo vyvinutá v rámci PyTorch na trénovanie NS v tomto frameworku. Pri vytváraní triedy musíme zadať, kde sa nachádza dataset na disku (cestu k nemu), transformáciu za pomoci *Albumentations*.

```
class LaneDetectionDataset(Dataset):
    def __init__(self, path, val=False, transforms=None):
        self.transforms = transforms
```

```
        if not val:
            self.img_path = path + 'train/'
            self.mask_path = path + 'train_label/'
        else:
            self.img_path = path + 'val/'
            self.mask_path = path + 'val_label/'
        self.img_names = [name for name in os.listdir(self.img_path)]
```

Interná funkcia *__getitem__* vyberie z disku príslušný obrázok podľa argumentu *idx*. Ak chceme zobraziť tento obrázok, prepneme prepínač *imShow* na True.

```
def __getitem__(self, idx, imShow=False):
    img_name = self.img_names[idx]
    mask_name = img_name[:-4] + '_label' + img_name[-4:]
    rawImg = cv2.imread(self.img_path + img_name)
    rawMask = cv2.imread(self.mask_path + mask_name, cv2.IMREAD_UNCHANGED)
    if imShow:
        cv2.imshow("Image", rawImg)
        cv2.imshow("Mask", rawMask)
        cv2.waitKey(0)
    img, mask = transformImage(rawImg, self.transforms, rawMask)
    return img, mask
```

Metóda *__len__* nám vráti veľkosť datasetu.

```
def __len__(self):
    return len(self.img_names)
```

4.1.3 CNNLineDetector

Trieda určená na trénovanie / použitie siete, ak nastavíme *from_scratch* na False, bude sa len používať natrénovaná sieť na *path*. Ak to bude True, bude sa sieť trénovať.

```
class CNNLineDetector:
    def __init__(self, from_scratch=True, path='torchUnet/unet_model.pth',
dataPath=data_path):
        self.val_epoch = None
        self.train_epoch = None
        self.optimizer = None
        self.loss = None

        self.train_dataset = LaneDetectionDataset(dataPath, val=False,
transforms=testtransform)
        self.val_dataset = LaneDetectionDataset(dataPath, val=True,
transforms=testtransform)
        self.trainloader = DataLoader(self.train_dataset, batch_size=2,
shuffle=True)
        self.valloader = DataLoader(self.val_dataset, batch_size=2,
shuffle=True)
        self.path = path

        self.model = smp.UnetPlusPlus(encoder_name='resnet34',
                                      encoder_weights='imagenet',
                                      in_channels=3,
                                      classes=3).to(device)
        if from_scratch:
            print('Model initialised from Scratch.')
        else:
            self.model.load_state_dict(torch.load(path))
            print('Loaded saved model at: ', path)
```

V takomto prípade sa nastavia aj trénovacie parametre cez *setTrainingParams* a následne *train* natrénuje a uloží sieť:

```
def setTrainingParams(self):
    self.optimizer = torch.optim.Adam(self.model.parameters(), lr=0.01)
    self.loss = smp.losses.DiceLoss('multiclass')
    self.loss.__name__ = 'DiceLoss'

    self.train_epoch = smp.utils.train.TrainEpoch(
        self.model,
        loss=self.loss,
        optimizer=self.optimizer,
        metrics=[],
        device=device,
        verbose=True
    )

    self.val_epoch = smp.utils.train.ValidEpoch(
        self.model,
        loss=self.loss,
        metrics=[],
        device=device,
        verbose=True
    )

def train(self, num_epochs, save=True):
    for i in range(1, num_epochs+1):
        print("Running epoch {now}/{max}".format(now=i, max=num_epochs))
        logTraining = self.train_epoch.run(self.trainloader)
        logValidation = self.val_epoch.run(self.valloader)
        print("TRAINING STATUS")
        print(logTraining, logValidation)
    if save:
        torch.save(self.model.cpu().state_dict(), self.path)
        print("Model saved to " + self.path)
```

Metóda *predict* funguje len na použitie siete na segmentáciu:

```
def predict(self, image):
    predictedMask = self.model(image.unsqueeze(0).to(device))
    predictedMask = torch.argmax(predictedMask.squeeze(), axis=0)
    return predictedMask
```

4.1.4 Spustiteľná časť

Najprv nastavíme trénovací a validačný dataset:

```
train_dataset = LaneDetectionDataset(data_path, val=False,
transforms=testtransform)
val_dataset = LaneDetectionDataset(data_path, val=True,
transforms=testtransform)
```

Následne si vyberieme pomocou prepínača *best*, či chceme trénovať sieť, alebo ukázať výsledky najlepšej. Podľa toho vytvoríme inštanciu detektora a nastavíme parametre:

```
best = False
if best:
    model = CNNLineDetector(from_scratch=not best)
else:
    model = CNNLineDetector(from_scratch=not best)
    model.setTrainingParams()
    model.train(num_epochs=2, save=True)
```

Na záver vykreslíme výsledky:

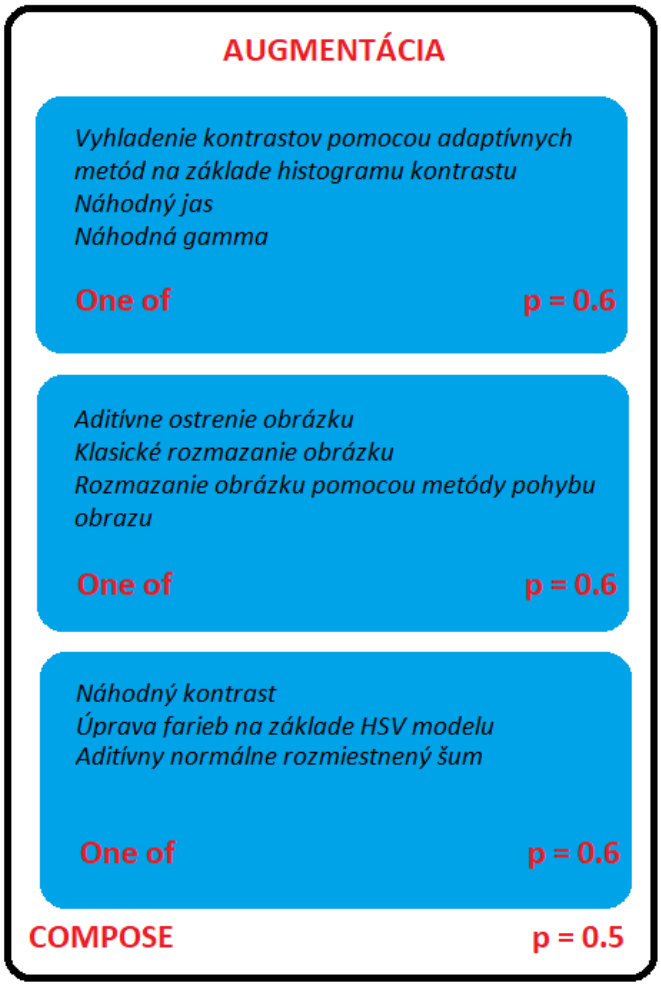
```
showDataset(num_imgs=4, dataset=val_dataset, model=model)
```

4.2 MobileNetV3Small

Ako sme spomínali aj v práci samotnej, táto sieť vyšla ako víťazná, preto taktiež obsahuje viacero funkcionalít, pričom väčšina je spojená s jej ďalším využitím pri riadení vozidla. Nájdeme ju v súbore *fastAI*, čo indikuje názov frameworku, v ktorom bude sieť trénovaná.

4.2.1 SegmentationAlbumentationsTransformation

Trieda, kde sa nachádza augmentácia pre túto NS. Používame tu augmentáciu na základe:



Obrázok 1: Augmentácia pri MobileNetV3Small

Princíp je obdobný, ako pri predchádzajúcej hlbokej sieti, opäť vytvoríme triedu, ktorá dedí od triedy, ktorá má za úlohu riešiť augmentáciu v rámci frameworku. V tomto prípade sa odporúča na načítavanie obrázkov používať knižnica PIL.

Pri vytváraní inštancie si používateľ zadá buď vlastnú transformáciu, alebo nechá parameter *trasnform* nastavený na None. Vtedy sa využije transformácia z Obrázka 1.

```
class SegmentationAlbumentationsTransformation(ItemTransform):
    split_idx = 0

    def __init__(self, transform=None):
        if transform is None:
```

```
            self.aug = self.useDefaultTransform()
        else:
            self.aug = transform
```

Metóda *encodes* vytvára augmentované obrázky na základe definovanej transformácie.

```
    def encodes(self, x):
        img, mask = x
        aug = self.aug(image=np.array(img), mask=np.array(mask))
        return PILImage.create(aug["image"]), PILMask.create(aug["mask"])
```

Transformáciu podľa obrázku 1 zabezpečí statická metóda *useDefaultTransform:*

```
@staticmethod
def useDefaultTransform():
    albuList = [albu.IAAAdditiveGaussianNoise(p=0.2),
                albu.OneOf(
                    [
                        albu.CLAHE(p=1),
                        albu.RandomBrightness(p=1),
                        albu.RandomGamma(p=1),
                    ],
                    p=0.6,
                ),
                albu.OneOf(
                    [
                        albu.IAASharpen(p=1),
                        albu.Blur(blur_limit=3, p=1),
                        albu.MotionBlur(blur_limit=3, p=1),
                    ],
                    p=0.6,
                ),
                albu.OneOf(
                    [
                        albu.RandomContrast(p=1),
                        albu.HueSaturationValue(p=1),
                    ],
                    p=0.6,
                ),
                ]
    transform = albu.Compose(albuList)
    return transform
```

4.2.2 Trénovanie

Proces trénovania máme v tomto prípade riešený mimo vytvorenia špeciálnej triedy. Je to klasická sekvencia inštrukcií na základe ktorej natrénujeme sieť. V tomto prípade sú zadefinované dve funkcie, ktoré kopírujú originálne názvy v rámci fastAI:

```
def get_image_array_from_fn(fn):
    image = cv2.imread(fn)
    return cv2.cvtColor(image, cv2.COLOR_BGR2RGB)


def label_func(fn):
    return str(fn).replace(".png", "_label.png").replace("train",
"train_label").replace("val\\", "val_label\\")
```

Keď sa pozrieme na hlavný kód, najprv vyberieme grafickú kartu ako zariadenie a overíme, že to tak skutočne je:

```
if __name__ == '__main__':
    torch.cuda.device(0)
    print(torch.cuda.get_device_name(0))
```

Rozdelíme obrázky podľa názvu na trénovacie a validačné:

```
my_get_image_files = partial(get_image_files, folders=["train", "val"])
```

Zadefinujeme názvy objektov, ktoré budeme hľadať pomocou segmentácie. Pre nás je to pozadie, ľavá a pravá čiara:

```
codes = np.array(['back', 'left', 'right'], dtype=str)
```

Na načítanie obrázkov a ich másk použijeme DataBlock. Ten bude dávkovať potrebné dáta:

```
carla = DataBlock(blocks=(ImageBlock, MaskBlock(codes)),
                  get_items=my_get_image_files,
                  get_y=label_func,
                  splitter=FuncSplitter(lambda x:
str(x).find('validation_set') != -1),
                  item_tfms=[SegmentationAlbumentationsTransformation()])
```

Z datablocku si vytvoríme objekt dataloaders, ktorý umožňuje načítavať dáta priamo do siete. Definujeme aj veľkosť dávky (batch size) na 2. Ukážku aj vykreslíme:

```
dls = carla.dataloaders(Path(DATA_DIR), path=Path("."), bs=2)
dls.show_batch(max_n=6)
plt.show()
```

Vytvoríme predtrénovaný model, ktorý bude segmentovať 3 triedy a používať 8 filtrov:

```
model = MobileV3Small(num_classes=3, use_aspp=True, num_filters=8)
```

Na jeho natrénovanie použijeme *Learner*. Ten potrebuje dávkovač dát, model a metriku. Následne zavoláme jeho metódu *fine_tune*, pričom nastavíme 10 epoch. Po natrénovaní uložíme model v dvoch formách – ako fastAI model a torch model:

```
learn = Learner(dls, model, metrics=[DiceMulti()], cbs=ShowGraphCallback())
learn.fine_tune(10)
learn.export('seg_aug.pkl')
torch.save(learn.model, './fastai_model_aug.pth')
```

4.3 FALineDetector

Trieda, ktorá je vytvorená na neskoršie, efektívnejšie použitie natrénovanej MobileNetV3Small. Pri inicializácii importujeme natrénované modely pomocou:

```
def importModels(self, aug, ismain):
    if ismain:
        self.torchModel = os.path.abspath('fastai_model.pth')
        self.fastAiModel = os.path.abspath('seg.pkl')
    else:
        self.torchModel = os.path.join('fastAI/fastai_model.pth')
        self.fastAiModel = os.path.join('fastAI/seg.pkl')
    if aug:
        self.torchModel = self.torchModel.replace("model", "model_aug")
        self.fastAiModel = self.fastAiModel.replace("seg", "seg_aug")
```

Pri vytváraní inštancie teda okrem importovania modelov nastavíme geometriu kamery a aktivujeme detektor:

```
def __init__(self, aug=True, isMain=False):
    self.importModels(aug, isMain)
    self.device = "cuda"
```

```
    self.learner = load_learner(self.fastAiModel)
    self.model = torch.load(self.torchModel).to(self.device)
    self.time = time.time()
    self.treshold = 0.3
    self.cg = CameraGeometry()
    self.cut_v, self.grid = self.cg.precompute_grid()
    self.init(isMain)
    warnings.filterwarnings("error")
```

Aktiváciu detektoru myslíme zavolaním metódy *init*. V prípade, že je táto trieda volaná ako súčasť celého projektu (segmentačná sieť je pripevnená ku konkrétnemu vozidlu), musíme zmeniť cestu k obrázku, ktorý bude uvádzať sieť do behu.

Ako indikuje komentár nad deklaráciou, prvá detekcia trvá vždy 2-3 sekundy. Preto ju prvý krát urobíme „falošne“. Rozsegmentujeme preddefinovaný obrázok a následné segmentácie budú trvať už len približne 30ms. Objekt je potrebné po celú dobu udržať nažive, aby sme sa vyhli vytváraniu nových inštancí, ktoré by opätovne vyžadovali takýto proces.

```
# First detection takes 2-3s so make "fake" on beginning
def init(self, isMain):
    self.loadImage(path="im.png" if isMain else "fastAI/im.png")
    self.predict()
    self.model.eval()
```

Ďalšou je metóda *predict*. Tá segmentuje vstupný obraz a nastavuje členov triedy left a right – čo sú samozrejme masky jednotlivých čiar.

```
def predict(self):
    with torch.no_grad():
        image_tensor = self.image.transpose(2, 0, 1).astype('float32') /
255
        x_tensor = torch.from_numpy(image_tensor).to("cuda").unsqueeze(0)
        _, self.left, self.right = F.softmax(self.model.forward(x_tensor),
dim=1).cpu().numpy()[0]

    self.left = self.filter(self.left, 1)
    self.right = self.filter(self.right, 1)
```

Ako vidíme, tento výstup najprv prechádza filtráciou. Tá spočíva v kombinácii binárnych obrazových operácii, erózii a následnej dilatácii (inak aj binárne uzavretie):

```
@staticmethod
def filter(inputImage, it=1) -> np.ndarray:
    kernel = np.ones((5, 5), np.uint8)
    retVal = inputImage
    eroded = cv2.erode(retVal, kernel, iterations=it)
    dilated = cv2.dilate(eroded, kernel, iterations=it)
    retVal = dilated
    return retVal
```

Metóda *integrateLines* má za úlohu vložiť do aktuálneho obrazu čerstvo vysegmentované masky a označiť ich modrou pre ľavú a červenou pre pravú čiaru:

```
def integrateLines(self):
    self.image[self.left > self.treshold, :] = [0, 0, 255]  # blue for left
    self.image[self.right > self.treshold, :] = [255, 0, 0]  # red for
right
```

visualizeLines slúži na zobrazenie obrázka s integrovanými čiarami. Taktiež ponúka parameter delay, ktorý rozhoduje, ako dlho bude obrázok na obrazovke, pokým sa automaticky zavrie:

```
def visualizeLines(self):
    cv2.imshow("Left Line", self.left)
    cv2.imshow("Right Line", self.right)
    cv2.waitKey()
```

extractPolynomials: táto metóda vytvorí kubické aproximácie pre ľavú aj pravú čiaru, pričom práve tieto aproximácie sú návratovou hodnotou. Ak je problém, vracia nulové koeficienty.

```
def extractPolynomials(self):
    try:
        leftPolynomial = self.fit(self.left)
        rightPolynomial = self.fit(self.right)
    # in specific conditions, LineDetector is not relevant, so we rather
not use this detected lines
    # numpy will return Warning, which we change to an Error to be able
catching it.
    except:
        leftPolynomial = np.poly1d(np.array([0., 0., 0., 0.]))
        rightPolynomial = np.poly1d(np.array([0., 0., 0., 0.]))
    return leftPolynomial, rightPolynomial
```

Práve metóda *fit* reálne vytvára túto aproximáciu. Využíva na to inverzné mapovanie, ktoré sme prebrali od M. Theersa. Tým získame pravdepodobnostné triplety, na základe ktorých vytovríme spomínané aproximácie pomocou funkcie polyfit v rámci numpy:

```
def fit(self, probs):
    probs_flat = np.ravel(probs[self.cut_v:, :])
    mask = probs_flat > 0.3
    if mask.sum() > 0:
        coeffs = np.polyfit(self.grid[:, 0][mask], self.grid[:, 1][mask],
deg=3, w=probs_flat[mask])
    else:
        coeffs = np.array([0., 0., 0., 0.])
    return np.poly1d(coeffs)
```

Poslednou, no zrejme kľúčovou metódou je *loadImage*. Táto funkcia umožňuje načítať ako obrázky z disku (ak definujeme path), tak aj obrázky priamo z kamery v real time – v tom prípade vkladáme tento obrázok do argumentu *numpyArr*:

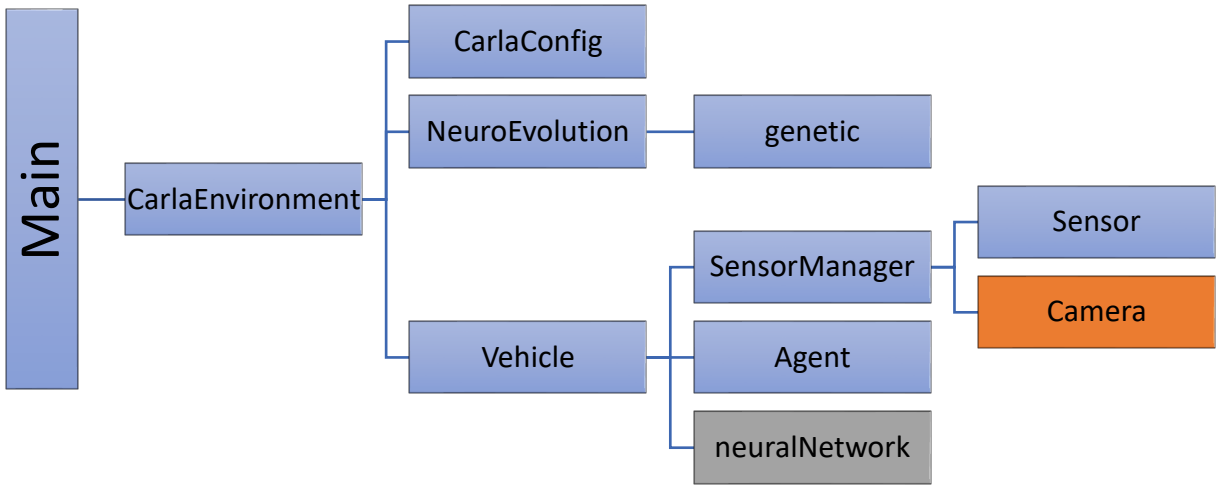
```
def loadImage(self, path=None, numpyArr=None):
    if path is not None:
        self.image = cv2.imread(path)
        self.image = cv2.cvtColor(self.image, cv2.COLOR_BGR2RGB)
    else:
        self.image = numpyArr
```

sinceLast je bonusová metóda, ktorá len počíta čas medzi poslednou detekciou čiar a aktuálnou. Jej využitie bolo pri meraní rýchlosti aproximácii a v real-time aplikácii ju nepoužívame.

```
def sinceLast(self):
    since = time.time() - self.time
    self.time = time.time()
    return since
```

5 Architektúra programu

Architektúre sme sa venovali aj v samotnej diplomovej práci, v sekcii Implementácia. V tomto prípade však budeme popisovať architektúru čisto z programátorského hľadiska, preto sú tu názvy tried v ich hierarchii:



Obrázok 2: Architektúra

Všetky súčasti hlavného programu (na obrázku vyššie) disponujú popisom metód priamo pri ich deklarácii v angličtine. Dôležité časti spomenieme v tejto kapitole.

5.1 Main (main.py)

Aktuálne je podporovaných 7 módov – na základe názvu v rámci enumerácie je zrejme jasné, ako sa ktorý mód bude správať.

```
class Mode(enum.IntEnum):
    '''
    Enum of all current supported modes
    '''
    trainingDefault = 0
    showTrainingDefault = 1
    runTestDefault = 2
    trainingTraffic = 3
    showTrainingTraffic = 4
    runTestTraffic = 5
    runTestSecondPath = 6
```

a akú konfiguráciu chce použiť. Ak chceme trénovať novú konfiguráciu, potrebujeme použiť *none*. V takom prípade sa na základe config.ini súboru vytvorí nová konfigurácia. Aby

sme ju vedeli použiť, musíme ju samozrejme pridať medzi ostatné konfigurácie (vymyslieť názov a hodnotu nastaviť podľa názvu novovytvoreného súboru v rámci *results*).

```
class Results(enum.IntEnum):
    '''
    Enum of all current results
    '''
    none = -1
    All = 2
    Navigation = 5
    Lines_Navigation = 22
    Radar = 204
    Lines = 300
    Lines_Radar = 301

    def __str__(self):
        return self.value

    def title(self):
        return self.name.replace("_", ", ")
```

Hlavná trieda. Vytvorenie inštancie spúšťa program – užívateľ si vopred navolí aký mód a aká konfigurácia bude použitá. Na základe toho sa vytvorí aj rozhranie medzi programom a simulátorom.

```
class Main(QCoreApplication):
    '''
    Main Class. This should be turned on to see the functionality!
    @author: Marko Chylik
    @Date: May, 2022
    '''
    def __init__(self, data, mode):
        '''
        Start the program.
        :param data: which data will be used -> from Results enum
        :param mode: which mode we want -> from Mode enum
        '''
        super(Main, self).__init__([])
        self.time = time.time()
        self.data = data
        self.mode = mode

        self.carlaEnvironment = CarlaEnvironment(self, data=data.value,
debug=False)
```

V prípade, že nastal problém, je potrebné ukončiť správne program:

```
def terminate(self):
    '''
    terminate the program
    :return: None
    '''
    print("Terminating MAIN!")
    try:
        self.carlaEnvironment.terminate()
    finally:
        sys.exit(0)
```

Beh funkcionalít má na starosti oddedená metóda, *exec* (vráti návratový kód):

```
def exec(self):
    '''
    Run defined mode by self.mode
    :return: exec method from QCoreApplication
    '''
    # DEFAULT scenarios
    if self.mode == Mode.trainingDefault:
        self.training(False)
    elif self.mode == Mode.showTrainingDefault:
        self.showTrainingResult(False)
    elif self.mode == Mode.runTestDefault:
        self.runTest(False, 1)
    # ADVANCED scenarios
    elif self.mode == Mode.trainingTraffic:
        self.training(True)
    elif self.mode == Mode.showTrainingTraffic:
        self.showTrainingResult(True)
    elif self.mode == Mode.runTestTraffic:
        self.runTest(True, 1)
    elif self.mode == Mode.runTestSecondPath:
        self.runTest(False, 2)

    return super().exec()
```

Ako aj vidíme vyššie, spášťame trénovacie/testovacie scenáre s určitými parametrami. Všetky bool parametre v nasledujúcich metódach znamenajú, či chceme, aby bol použitý pokročilý mód – s vygenerovanou dopravou v simulácii. Testovací scenár vyžaduje aj voľbu testovacej dráhy, keďže ich je v projekte viacero:

```
def training(self, advanced):
    '''
    run training with traffic, if advanced
    :param advanced: bool
    :return: None
    '''
    self.carlaEnvironment.train(advanced)

def showTrainingResult(self, advanced):
    '''
    Show how vehicle performed on training path.
    :param advanced: bool
    :return: None
    '''
    if self.data == Results.none:
        print("Need to pick some results")
        return
    self.carlaEnvironment.replayTrainingRide(data.value, advanced)

def runTest(self, advanced, path):
    '''
    Run testing path
    :param advanced: bool
    :param path: 1 or 2
    :return: None
    '''
    if self.data == Results.none:
        print("Need to pick some results")
        return
    self.carlaEnvironment.testRide(data.value, advanced, path)
```

5.1.1 Hlavný kód

V hlavnom kóde si navolíme, čo vlastne chceme spustiť. Vyzerá takto:

```
if __name__ == '__main__':
    '''
    Main code!
    '''
    data = Results.All                          # pick the data
    mode = Mode.showTrainingDefault             # pick the mode
    mainApp = Main(data, mode)                  # start Main
    code = mainApp.exec()                       # run it
    sys.exit(code)                              # end the app with code
```

V aktuálnom prípade teda použijeme konfiguráciu všetkých vstupov pre NS a pozrieme si, ako prejde vozidlo s touto konfiguráciou trénovacou dráhou.

5.2 CarlaEnvironment

Táto trieda je hlavné premostenie medzi simulátorom a programom. Pri vytvorení inštancie sa vytvorí spojenie so simulátorom – *client*. Okrem toho inicializujeme detektor čiar, ktorý dodávame jednotlivým vozidlám, či sa nastavujú ďalšie stavové premenné tejto triedy.

```
def __init__(self, main, data=-1, debug=False):
    '''
    Create an object of Carla environment.
    :param main: reference to main (QCoreApplication)
    :param data: default = -1 -> create new directory based on "rev" from
config.ini
                    if param is > 0 -> use the directory with the same name
as data
    :param debug: bool
    '''
    super(CarlaEnvironment, self).__init__()
    self.id = 0
    self.debug = debug
    self.main = main
    self.clock = pygame.time.Clock()
    self.loadedData = True if data>0 else False
    self.whichPath = 0

    self.client = carla.Client('localhost', 2000)
    path = "config.ini" if data == -1 else
os.path.join(f"results/{data}/config.ini")
    self.config = CarlaConfig(self.client, path)
    self.config.apply()
    self.NE = NeuroEvolution(self.config.loadNEData())
    self.faLineDetector = FALineDetector()
    self.MAX_ID = self.NE.popSize
    self.trainingMode = False
    self.trafficManager = self.client.get_trafficmanager()
    self.trafficManager.set_synchronous_mode(self.config.sync)
    self.world = self.client.get_world()
    self.blueprints = self.world.get_blueprint_library()
    self.map = self.world.get_map()

    self.traffic = False

    print("CARLA ENVIRONMENT DONE")
```

5.2.1 tick

Pošle signál serveru a vykonanie simulačného kroku.

```
def tick(self):
    '''
    TICKING the world
    :return: num of tick
    '''
    self.clock.tick()
    tickNum = self.world.tick()
    return tickNum
```

5.2.2 replayTrainingRide

Ukážka trénovacej jazdy. Ako argumenty očakáva číslo konfigurácie a či chceme aby bola použitá premávka:

```
def replayTrainingRide(self, numRevision, traffic):
    '''
    Spawn one car and "simulate" ride through waypoints of training path
    :param numRevision: Which model will be used
    :param traffic: bool -> use traffic?
    :return: None
    '''
    self.trainingMode = False
    self.generateTraffic(traffic)
    spawnPoints = self.map.get_spawn_points()
    start = spawnPoints[99]
    self.whichPath = 0
    self.spawnVehicle(True, start, numRevision)
    print("Starting replay of training ride")
    if self.loop():
        self.main.terminate()
```

5.2.3 testRide

Odsimuluje prechod vozidla testovacou dráhou (podľa argumentu path ktorou):

```
def testRide(self, numRevision, traffic, path):
    '''
    Run test ride on one of the test paths.
    :param numRevision: Which model will be used
    :param traffic: bool
    :param path: which path is going to be used (1,2)
    :return:
    '''
    self.trainingMode = False
    self.generateTraffic(traffic)
    spawnPoints = self.map.get_spawn_points()
    point = 334 if path == 1 else 258

    start = spawnPoints[point]
    self.whichPath = path

    self.spawnVehicle(True, start, numRevision)
    if self.loop():
        self.main.terminate()
```

5.2.4 train

Zabezpečuje celý priebeh trénovania z najvyššej vrstvy. Riadi počet epoch, komunikuje s neuro-evolučnou jednotkou a volá funkciu trainingRide.

```
def train(self, traffic):
    '''
    Run training process, don't show anything at all, just cover whole
training process
    :param traffic: bool
    :return:
    '''
    self.trainingMode = True
    if not self.loadedData:
        self.config.incrementNE()
    self.generateTraffic(traffic)
    for i in range(self.NE.startCycle, self.NE.numCycle):
        print(f"Starting EPOCH {i+1}/{self.NE.numCycle}")
        # run one training epoch
        self.trainingRide(i)
        self.NE.perform()
        self.NE.finishNeuroEvolutionProcess()  # will probably block the
thread
    self.main.terminate()
```

5.2.5 trainingRide

Zabezpečuje jednu celú epochu trénovania – vygeneruje vozidlo a pridelí mu NS. Následne počká, dokým vozidlo skončí činnosť a cez neuroevolučnú triedu zabezpečí ohodnotenie kvality jazdy. Najlepšieho jedinca z minulej populácie ignoruje, aby sme ušetrili čas.

```
def trainingRide(self, epoch):
    '''
    Handles one epoch of training
    :return: None
    '''
    print("Starting training")
    for i in range(self.MAX_ID):
        if epoch != 0 and i == 0:
            # do not run best solutions again!
            continue
        self.id = i
        spawnPoints = self.map.get_spawn_points()
        start = spawnPoints[99]
        self.whichPath = 0
        self.spawnVehicle(False, start)
        self.loop()
```

5.2.6 loop

Táto funkcia zabezpečuje čo sa deje po odsimulovaní jedného kroku na serveri. Spúšťa funkciu runStep, ktorá zabezpečuje, že každé ovládané vozidlo vykoná svoj hlavný kód.

```
def loop(self):
    '''
    main loop for any paths/scenario -> send tick to server and call
runStep from vehicle
    :return:    True, if any vehicle ended
```

```
                False, if any error has occured
    '''
    while True:
        try:
            tickNum = self.tick()
            listV = self.runStep(tickNum)
            if len(listV) > 0:
                for veh in listV:
                    if self.trainingMode:
                        self.NE.singleFit(veh)
                    else:
                        self.storeVehicleResults(veh)
                    print(f"Vehicle {veh.vehicleID} done!")
                    self.deleteVehicle(veh)
                return True
        except Exception as e:
            print(e)
            self.main.terminate()
            return False
```

5.2.7 path

Načíta cestu v podobe radu z configu.

```
def path(self):
    '''
    load path based on whichPath
    :return: list of waypoints
    '''
    return self.config.loadPath(self.whichPath)
```

5.2.8 generateTraffic

Ak je nastavený parameter na true, na začiatku behu programu sa vygeneruje za pomoci externe prebraného skriptu cez nové vlákno premávka. V prípade ukončenia je potrebné ako prvé tento skript ukončiť.

```
def generateTraffic(self, generate):
    '''
    generate Traffic if desired
    :param generate: bool
    :return: None
    '''
    if not generate:
        return
    self.traffic = True
    self.trafficThread = threading.Thread(target=generateTraffic,
args=(self.trafficGenerated,))
    self.trafficThread.start()
```

5.2.9 trafficGenerated

Vráti pravdivostnú hodnotu, či je vygenerovaná premávka.

```
def trafficGenerated(self):
    '''
    :return: bool: is traffic generated?
    '''
    return self.traffic
```

5.2.10 spawnVehicle

Vygeneruje vozidlo na základe žiadaných parametrov. Potrebuje štartovací bod, či sme v testovacom móde (v tomto prípade sa myslí, či chceme vidieť výsledok) a ktorú štruktúru používame. Ak ide o trénovaciu jazdu, neuroevolučná trieda pridelí sieť na základe populácie.

```
def spawnVehicle(self, testRide, start, numRevision=0):
    '''
    Spawn vehicle to starting spot (start) and create it.
    :return: None
    :param testRide: is it test ride? (bool)
    :param start: point on map, where vehicle should be spawned
    :param numRevision: in case of test ride, which NN model should be used
    '''

    if not testRide:
        neuralNetwork = self.NE.getNeuralNetwork(self.id)
    else:
        weightsFile = f'results/{numRevision}/best.csv'
        weights = np.loadtxt(weightsFile, delimiter=',')
        neuralNetwork = self.NE.getNeuralNetworkToTest(weights)
    vehicle = Vehicle(self, spawnLocation=start, id=self.id,
neuralNetwork=neuralNetwork)
    vehicle.applyConfig(testRide)
    self.vehicles.append(vehicle)
```

5.2.11 runStep

Zabezpečí chod jedného simulačného kroku pre všetky ovládané vozidlá.

```
def runStep(self, tickNum):
    '''
    Ask all available vehicles to do their job
    :return: list of ended vehicles
    '''
    endedVehicles = []
    for vehicle in self.vehicles:
        if not vehicle.run(tickNum):
            endedVehicles.append(vehicle)
    return endedVehicles
```

5.2.12 storeVehicleResults

Po skončení jazdy vozidla na testovacej dráhe (prípadne ukážke najlepšej jazdy na trénovacej) chceme, aby sa uložili pozície vozidla počas behu. Ukladá sa pozícia vozidla, pozície čiar a optimálna dráha vypočítaná navigáciou. Všetky tieto dáta sú zbierané samotným vozidlom a táto funkcia ich len spracuje a uloží na disk vo vhodnom formáte.

```
def storeVehicleResults(self, vehicle: Vehicle):
    '''
    We can expect that vehicle will have 4 lists. Save this lists on disk
with current revision
    :param vehicle: Vehicle object
    :return: None
    '''
    print("Storing results")
    pos, ll, rl, op = vehicle.returnVehicleResults()
    numberOfPositions = len(pos)
```

```
    x = np.zeros((4, numberOfPositions))
    y = np.zeros((4, numberOfPositions))
    for idx in range(numberOfPositions):
        x[0, idx] = pos[idx].x
        y[0, idx] = pos[idx].y

        x[3, idx] = op[idx].x
        y[3, idx] = op[idx].y

    for idx in range(len(ll)):
        x[1, idx] = ll[idx].x
        y[1, idx] = ll[idx].y

        x[2, idx] = rl[idx].x
        y[2, idx] = rl[idx].y

    rev = self.config.parser.get("NE", "rev")
    pathX = os.path.join(f"results/{rev}/X{self.whichPath}.csv")
    pathY = os.path.join(f"results/{rev}/Y{self.whichPath}.csv")
    np.savetxt(pathX, x, delimiter=',')
    np.savetxt(pathY, y, delimiter=',')
```

5.2.13 deleteVehicle

Zmaže auto z prostredia CARLA (aj všetky jeho súčasti – senzory, kamery,...)

```
def deleteVehicle(self, vehicle):
    '''
    When vehicle ends, we need to delete it also from our list
    :param vehicle: Vehicle
    :return: None
    '''
    for v in self.vehicles:
        if v == vehicle:
            try:
                print(f"Deleting vehicle {vehicle.vehicleID}")
                v.destroy()
                self.vehicles.remove(vehicle)
            except:
                print("Vehicle already out")
```

5.2.14deleteAll

Zmaže všetko vygenerované aktuálnym programom z prostredia.

```
def deleteAll(self):
    '''
    In case of error/ending the whole simulation, we wants to delete all
vehicles.
    :return: None
    '''
    for vehicle in self.vehicles:
        try:
            vehicle.destroy()
            self.vehicles.remove(vehicle)
            del vehicle
            print("Removing vehicle!")
        except:
            print("Already deleted!")
```

5.2.15 terminate

V prípade nečakaného konca programu sa zavolá táto metóda pre zabezpečenie všetkých krokov potrebných k bezchybnému ukončeniu, najmä z hľadiska simulátora.

```
def terminate(self):
    '''
    In case of error/ending simulation, gives signal to main to end the
program
    :return: None
    '''
    self.deleteAll()

    if self.traffic:
        self.traffic = False
        for _ in range(3):
            self.tick()
            time.sleep(0.5)

    print("Turning off sync mode")
    self.trafficManager.set_synchronous_mode(False)
    self.config.turnOffSync()
```

5.3 CarlaConfig

Trieda zabezpečuje nastavenie simulácie na základe zvoleného .ini súboru (parameter path).

```
def __init__(self, client=None, path="config.ini"):
    '''
    Init the config object with carla.Client reference and path to config
on hard drive
    :param client: carla.Client
    :param path: path towards config file on hard drive
    '''
    self.client = client
    self.parser = configparser.ConfigParser()
    self.path = path
    self.parser.read(self.path)
```

5.3.1 apply

Metóda volaná z CarlaEnvironment. Aplikuje nastavenia ohľadne simulátora pri štarte programu – najmä fixný krok, synchrónny mód, počasie či mapu.

```
def apply(self):
    '''APPLY CONFIG SETTINGS TO SIMULATOR'''
    self.sync = self.client.get_world().get_settings().synchronous_mode
    self.client.set_timeout(30)
    weather = self.parser.get("CARLA", "weather")
    map = self.parser.get("CARLA", "map")
    fixed = self.parser.get("CARLA", "ts")
    sync = self.parser.get("CARLA", "sync")
    world = self.client.get_world()
    curMap = world.get_map().name

    if map not in curMap:
        world = self.client.load_world(map)
    else:
        world = self.client.reload_world()
    world.set_weather(eval(weather))
```

```
    if len(fixed) > 0:
        settings = world.get_settings()
        settings.fixed_delta_seconds = eval(fixed)
        world.apply_settings(settings)

    self.turnOnSync() if eval(sync) else self.turnOffSync()
    self.client.set_timeout(5)
```

5.3.2 readSection

Pomocná metóda, ktorá prečíta sekciu podľa hlavičky a vráti vo forme dictionary.

```
def readSection(self, section):
    '''
    read the section and return dictionary
    :param section: name of section in ini file
    :return: dict
    '''
    return dict(self.parser.items(section))
```

5.3.3 loadNEData

Načíta dáta potrebné pre neuroevolúciu – na základe zvolených vstupov pre NS sa vypočíta počet vstupov a prepíše sa v config súbore.

```
def loadNEData(self) -> dict:
    '''
    load data for neuro-evolution: rewrite num of inputs based on which of
them are asked
    :return: dictionary of data
    '''
    listIns, count = self.loadAskedInputs()
    self.parser.set("NE", "nInput", str(count))
    self.rewrite(self.path)
    expDict = self.readSection("NE")
    return expDict
```

5.3.4 rewrite

Prepíše konfiguračný súbor na umiestnení path podľa aktuálne načítaného súboru.

```
def rewrite(self, path):
    '''
    rewrite current configfile to new one on PATH
    :param path: where config should be stored
    :return: None
    '''
    with open(path, 'w') as configfile:
        self.parser.write(configfile)
```

5.3.5 incrementNE

Aby sme zabezpečili, že každý výsledok trénovania bude mať unikátny identifikátor, vždy pri spustení trénovania inkrementujeme počítadlo pre neuroevolúciu o 1.

```
def incrementNE(self):
    '''
    when running new configuration, we need to create file for it and
increment rev
    :return: None
    '''
    expDict = self.loadNEData()
```

```
    base = str(expDict.get("base"))

    if not os.path.exists(base):
        os.mkdir(base)
    old = int(self.parser.get("NE", "rev"))
    file = base + str(old) + "/"

    if not os.path.exists(file):
        os.mkdir(file)
    self.rewrite(file + "config.ini")

    self.parser.set("NE", "rev", str(old + 1))
    self.rewrite(self.path)
```

5.3.6 loadAskedInputs

Podľa konfiguračného súboru sa načíta list žiadaných vstupov pre NS a zároveň sa vypočíta, aký bude ich počet dohromady.

```
def loadAskedInputs(self) -> (list, int):
    '''
    load which of inputs are asked based on config
    :return: list of inputs, number of inputs
    '''
    nnIns = self.readSection("NnInputs")
    expectedList = []
    count = 0

    for key, val in nnIns.items():
        if convertStringToBool(val):
            inputElement = InputsEnum.elem(key)
            expectedList.append(inputElement)
            count += inputElement.getValue()

    return expectedList, count
```

5.3.7 turnOffSync

Vypne synchrónny mód.

```
def turnOffSync(self):
    '''
    when simulation ends, we want to turn off sync mode
    :return:
    '''
    world = self.client.get_world()
    if self.sync:
        settings = self.client.get_world().get_settings()
        settings.synchronous_mode = False
        self.sync = False
        world.apply_settings(settings)
        print("Sync mode turned off!")
```

5.3.8 turnOnSync

Zapne synchrónny mód.

```
def turnOnSync(self):
    '''
    when simulation starts, we want to turn on the sync mode
    :return:
    '''
```

```
    world = self.client.get_world()
    if not self.sync:
        settings = self.client.get_world().get_settings()
        settings.synchronous_mode = True
        self.sync = True
        world.apply_settings(settings)
        print("Sync mode turned on!")
```

5.3.9 loadPath

Načíta žiadanú cestu podľa parametra. Hodnoty sú tu ručne vpisované do listov. Prípadné rozšírenie o novú cestu treba pridať obdobne.

```
@staticmethod
def loadPath(which=0) -> queue.Queue:
    '''
    load waypoints of asked path:
    :param which: 0 -> training, 1 -> testing1, 2 -> testing 2
    :return: Queue
    '''
    path = queue.Queue()
    if which == 0:
        x = [-8.6, -121.8, -363.4, -463.6, -490]
        y = [107.5, 395.2, 406, 333.6, 174]
    elif which == 1:
        x = [98.6, 272.5, 409.1, 410.7, 211.2]
        y = [34.7, 37.5, -37.2, -228.7, -392.1]
    elif which == 2:
        x = [258.9, 242, 200, 220.5]
        y = [-186, -249.7, -229.6, -169.5]
    for i in range(len(x)):
        loc = carla.Location()
        loc.x = x[i]
        loc.y = y[i]
        loc.z = 0.267
        path.put(loc)
    return path
```

5.3.10 InputsEnum

Síce nie je súčasťou CarlaConfig triedy, je však dôležitá. Je to enumerácia, ktorá ukladá názov vstupu a taktiež koľko vstupov v prípade jeho použitia je potrebných pre NS. Je to vo forme enumerácie. Vstupy sú popísané v diplomovej práci.

```
class InputsEnum(enum.Enum):


    linedetect = 6
    radar = 3
    agent = 1
    metrics = 2
    binaryknowledge = 4
    navigation = 0

    @staticmethod
    def elem(name):
        for element in InputsEnum:
            if name == element.name:
                return element
```

```
    def getValue(self):
        if self == InputsEnum.navigation:
            return 2
        else:
            return self.value
```

5.4 NeuroEvolution

Trieda, ktorá zabezpečuje proces neuro – evolúcie, od hodnotenia jednotlivých členov populácie až po vytváranie NS na základe jedincov:

```
def __init__(self, nnConfig: dict):
    super(NeuroEvolution, self).__init__()

    # neural network params
    nInputs, nHidden, nOutputs = loadNNParamsFromConfig(nnConfig)
    self.nn = NN(nInputs, nHidden, nOutputs)
    self.w, self.b = self.nn.getNumOfNeededElements()
    self.numParams = self.w + self.b

    # genetic algorithm params
    self.popSize = int(nnConfig.get("popsize"))
    self.startCycle = 0
    self.numCycle = int(nnConfig.get("numcycles"))
    self.calculateParamsOfGeneticAlgorithm(nnConfig)
    bottom = np.ones((1, self.numParams)) * -3
    upper = np.ones((1, self.numParams)) * 3
    self.space = np.concatenate((bottom, upper), axis=0)
    self.amp = upper * 0.1
    initSpace = self.space * 0.1

    # for saving
    self.base = str(nnConfig.get("base"))
    self.rev = str(nnConfig.get("rev"))
    self.fileBest = self.base + f'{self.rev}/best.csv'
    self.fileEvol = self.base + f"{self.rev}/evol.csv"

    # initial params
    if os.path.exists(self.fileEvol):
        minFit = np.loadtxt(self.fileEvol, delimiter=',')
        self.minFit = list(minFit)
        self.numCycle += len(self.minFit)
        self.startCycle += len(self.minFit)
    else:
        self.minFit = []

    self.fit = np.ones((1, self.popSize)) * 100000
    self.pop = genetic.genrpop(self.popSize, initSpace)

    if os.path.exists(self.fileBest):
        best = np.loadtxt(self.fileBest, delimiter=',')
        self.pop[0, :] = best
        self.fit[0, 0] = self.minFit[-1]
```

5.4.1 singleFit

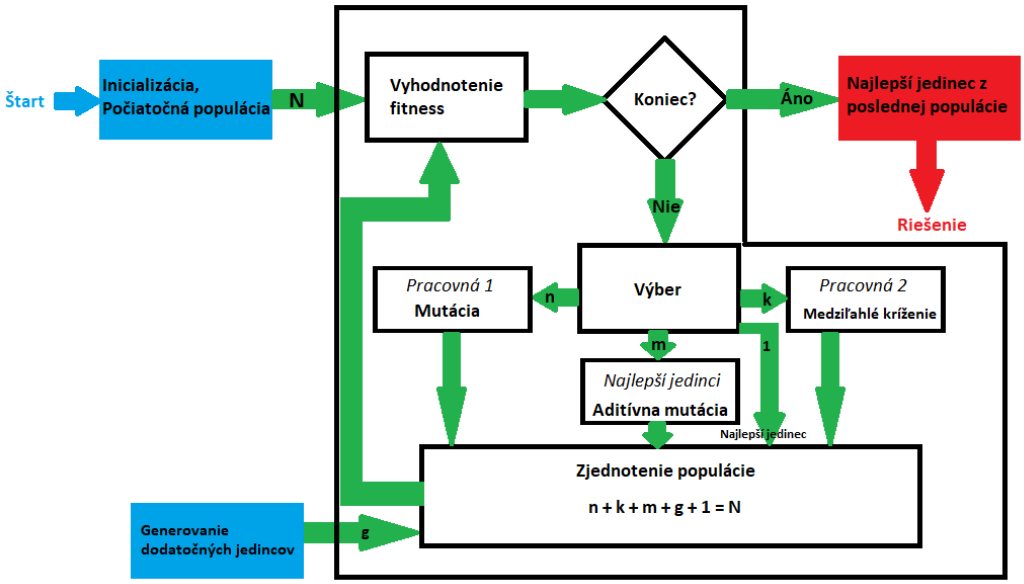
Vypočíta hodnotu účelovej funkcie jedinca – vozidla, ktoré dokončilo svoju jazdu. To dá vo forme directory výsledok všetkých potrebných súčastí. Tie sa tu rozdelia a vypočíta sa finálna hodnota. Na záver sa uloží do premennej triedy na základe ID vozidla.

Po ohodnotení sa vypíše postup daného vozdila aj do konzoly (celková fitness aj jej čiastkové hodnoty).

```
def singleFit(self, vehicle: Vehicle.Vehicle):
    '''
    Calculate fit value of single vehicle solution
    :param vehicle: Vehicle object
    :return: Nothing

    loadedDict = {'crossings': self.__crossings,
                  'collisions': self.__collisions,
                  'inCycle': self.__inCycle,
                  'rangeDriven': self.__rangeDriven,
                  'reachedGoals': self.__reachedGoals,
                  'error': self.__errDec}
    '''
    at = vehicle.vehicleID
    loadedDict = vehicle.record()

    crossings = loadedDict.get('crossings')
    collisions = loadedDict.get('collisions')
    inCycle = loadedDict.get('inCycle')
    rangeDriven = loadedDict.get('rangeDriven')
    reachedGoals = loadedDict.get('reachedGoals')
    error = loadedDict.get('error')

    cCrossings = 5
    cCollisions = 5000
    cInCycle = 5000
    cError = 0
    cRangeDriven = -3
    cReachedGoals = -2500

    fitValue = cCrossings * crossings + cError * error + cCollisions *
collisions + \
               cInCycle * inCycle + cRangeDriven * rangeDriven +
cReachedGoals * reachedGoals

    self.fit[0, at] = fitValue

    print('\n-----------------------------------------------------------------
---------\n')
    print(f"Vehicle {at} finished with fitness: {fitValue}")
    print(f"Crossings: {crossings}, fit: {cCrossings * crossings}")
    print(f"Collisions: {collisions}, fit: {cCollisions * collisions}")
    print(f"In cycle: {inCycle}, fit: {cInCycle * inCycle}")
    print(f"Error: {error}, fit: {cError * error}")
    print(f"Range: {rangeDriven}, fit: {cRangeDriven * rangeDriven}")
    print(f"Reached goals: {reachedGoals}, fit: {cReachedGoals *
reachedGoals}")
    print('\n-----------------------------------------------------------------
---------\n')
```

5.4.2 perform

Proces genetického algoritmu medzi cyklami. Genetické operácie sú na základe obrázka:



Obrázok 3: Genetický algoritmus

```
def perform(self):
    if len(self.minFit) > 0:
        self.fit[0, 0] = self.minFit[-1]
    self.minFit.append(np.min(self.fit))
    print(f"Done epochs: {len(self.minFit)}/{self.numCycle}, BestFit:
{np.min(self.fit)}")
    Best = genetic.selsort(self.pop, self.fit, 1)
    BestPop = genetic.selsort(self.pop, self.fit, self.nBest)
    WorkPop1 = genetic.selrand(self.pop, self.fit, self.nWork1)
    WorkPop2 = genetic.seltourn(self.pop, self.fit, self.nWork2)
    NPop = genetic.genrpop(self.nGenerate, self.space)

    SortPop = genetic.around(WorkPop1, 0, 0.75, self.space)
    WorkPop = genetic.mutx(WorkPop2, 0.15, self.space)
    BestPop = genetic.muta(BestPop, 0.01, self.amp, self.space)
    self.pop = np.concatenate((Best, SortPop, BestPop, WorkPop, NPop),
axis=0)
```

5.4.3 getNeuralNetwork

Na základe jedinca sa dosadia váhy a biasy do reálnej MLP siete a tento objekt je vrátený:

```
def getNeuralNetwork(self, at):
    '''
    Set weights to NN at argument
    :param at: which part of population will fill the weights
    :return: NN object
    '''
    weights = self.pop[at, :]
    self.nn.setWeights(weights)
    return self.nn
```

5.4.4 getNeuralNetworkToTest

Obdobná metóda, avšak pre testovanie. Je potrebné vložiť konkrétne načítané váhy a biasy (najlepšieho jedinca), na jeho základe sa vytvorí MLP, ktorá bude vrátená

```
def getNeuralNetworkToTest(self, weights):
    self.nn.setWeights(weights)
    return self.nn
```

5.4.5 calculateParamsOfGeneticAlgorithm

Vypočíta na základe konfigurácie, koľko bude treba členov pre jednotlivé subpopulácie pri genetickom algoritme. Percentá sú v konfiguračnom súbore v sekci NE.

```
def calculateParamsOfGeneticAlgorithm(self, conf: dict):
    '''
    Loads config and sets values of pop
    :return:Nothing
    '''
    bestPecentage = float(conf.get("best"))
    work1Percentage = float(conf.get("work1"))
    work2Percentage = float(conf.get("work2"))

    popSizeWithoutBest = self.popSize - 1
    self.nBest = int(bestPecentage * popSizeWithoutBest)
    self.nWork1 = int(work1Percentage * popSizeWithoutBest)
    self.nWork2 = int(work2Percentage * popSizeWithoutBest)
    self.nGenerate = popSizeWithoutBest - self.nBest - self.nWork1 -
self.nWork2
```

5.4.6 finishNeuroEvolutionProcess

Metóda, ktorá uloží najlepšieho jedinca a proces trénovania. Pôvodne bola volaná až na konci, avšak je dobré ju používať po každom dokončenom cykle v prípade nečakaného spadnutia programu tak získame backup.

```
def finishNeuroEvolutionProcess(self):
    Best = genetic.selsort(self.pop, self.fit, 1)
    np.savetxt(self.fileBest, Best, delimiter=',')
    evol = np.asarray(self.minFit)
    np.savetxt(self.fileEvol, evol, delimiter=',')
```

5.5 Vehicle

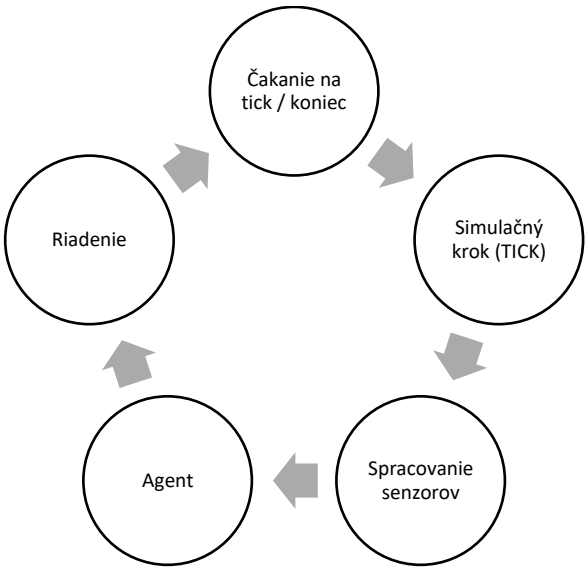
Trieda, ktorá zabezpečuje chod vozidla. Je najrozsiahlejšou triedou v rámci celého projektu. Na svoju inicializáciu potrebuje referenciu na prostredie, kde má nastať spawn vozidla, riadiacu NS pre vozidlo a ID vozidla. Vozidlo je schopné si pamätať svoju historickú pozíciu, či pozíciu čiar, ktorá je po jeho konci dostupná pomocou jednej z metód.

```
def __init__(self, environment, spawnLocation, neuralNetwork, id):
    '''
    Init vehicle and spawn it
    :param environment: reference to CarlaEnvironment
    :param spawnLocation: point on map, where Vehicle should be spawned
    :param neuralNetwork: NeuralNetwork object
    :param id: which vehicle from population of vehicles this is
    '''
    super(Vehicle, self).__init__()
    self.vehicleID = id
    self.environment = environment
    self.path = environment.path()
    self.askedInputs = environment.config.loadAskedInputs()[0]
    self.debug = self.environment.debug
```

```
    self.fald = self.environment.faLineDetector

    self.me = None
    while not self.me:
        self.me =
self.environment.world.try_spawn_actor(self.environment.blueprints.filter('
model3')[0], spawnLocation)
        if not self.me:
            for _ in range(10):
                self.environment.tick()

    self.nn = neuralNetwork

    self.speed = 0
    self.vehicleStopped = 0
    self.steer = 0
    self.limit = 0.8
    self.defaultSteerMaxChange = 0.1
    self.numMeasure = 10
    self.lastLocation = self.getLocation()

    self.initAgent(spawnLocation.location)
    self.sensorManager = SensorManager(self, self.environment)
    currentDt = str(datetime.datetime.now())
    print("Vehicle {id} ready at {dt}".format(id=self.vehicleID,
dt=currentDt))
    self.startTime = time.time()

    self.toGoal = deque(maxlen=10)
    self.metrics = deque(maxlen=2)

    self.__positionHistory = []
    self.__leftLinePlanner = []
    self.__rightLinePlanner = []
    self.__optimalPath = []
```

5.5.1 run

Hlavná metóda, ktorá je volaná z vonku – konkrétne z CarlaEnvironment. Vozidlo vykoná všetky činnosti podľa schémy:



```
def run(self, tickNum):
    '''
    Do a tick response for vehicle object
    :param tickNum: current tickNum from server
    :return: True, if vehicle is alive; False, if ending conditions were
MET
    '''
    if not self.me or self.sensorManager.isCollided() or self.checkGoal():
        print("Collision or Goal")
        return False
    if not self.debug: #TRAINING MODE
        if self.standing() or self.inCycle():
            print("Standing/In cycle, penalization!")
            self.__inCycle += 1
            return False

    # there will NN decide
    self.sensorManager.processSensors()
    control = self.getControl()
    self.me.apply_control(control)
    if tickNum % self.numMeasure == 0:
        self.print(f"TN {tickNum}: {self.diffToLocation(self.goal)}")
        self.toGoal.append(self.diffToLocation(self.goal))
        self.metrics.append(control)
        self.storeCurrentData()
    return True
```

5.5.2 agentAction

Pomocou integrovaného agenta získame riadenie rýchlosti a najbližší bod podľa lokálnej navigácie:

```
def agentAction(self):
    '''
    Usage of agent to reach expected speed and navigation implementation
    :return: carla.Control, current waypoint from navigation
    '''
    if self.agent.done():
        self.print(f"New waypoint for agent: {self.goal}")
        self.agent.set_destination(self.goal)
        self.toGoal.clear()
    control = self.agent.run_step()
    control.manual_gear_shift = False
    waypoints = self.agent.get_waypoints()[0]
    waypoint = waypoints.transform.location if waypoints else None
    self.print(f"Control: {control}")
    return control, waypoint
```

5.5.3 initAgent

Táto metóda zabezpečí štart agenta naviazaného na vozidlo pri štarte vozidla. Je potrebné počkať, dokým sa vozidlo spawne na žiadanú lokalitu – v opačnom prípade nebude lokálna navigácia presná a nebudeme ju môcť použiť.

```
def initAgent(self, spawnLoc):
    '''
    Init BasicAgent - we will use agent's PID to regulate speed.
    :param spawnLoc: carla.Location -> starting Location of agent
    :return: None
    '''
```

```
    self.getLocation()
    while self.diffToLocation(spawnLoc) > 1:
        self.environment.tick()
        self.getLocation()
        time.sleep(0.01)
    self.agent = BasicAgent(self.me, target_speed=50)
    self.goal = self.path.get()
    self.agent.set_destination(self.goal)
```

5.5.4 record

Na účely neuroevolúcie všetky zozbierané dáta potrebné pre vyhodnotenie účelovej funkcie odovzdáme NeuroEvolution triede pomocou tejto metódy vo forme dict:

```
def record(self):
    '''
    Record the training ride and make a summary of it into dictionary.
    :return: dictionary
    '''
    self.__crossings = self.sensorManager.laneInvasionDetector.crossings
    self.__collisions = 1 if self.sensorManager.isCollided() else 0
    retDict = {'crossings': self.__crossings,
               'collisions': self.__collisions,
               'inCycle': self.__inCycle,
               'rangeDriven': self.__rangeDriven,
               'reachedGoals': self.__reachedGoals,
               'error': self.__errDec}

    return retDict
```

5.5.5 recordEachStep

Niektoré údaje je potrebné merať každý krok, aby boli relevantné, táto metóda to zabezpečí.

```
def recordEachStep(self, agentSteer):
    '''
    Some of the parameters needs to be recorded at every step.
    :param agentSteer: steer suggested by agent
    :return: None
    '''
    self.__errDec += abs(agentSteer - self.steer)

    distLast = self.errInLocation(self.lastLocation, self.goal)
    self.lastLocation = self.getLocation()
    distNow = self.errInLocation(self.location, self.goal)
    difference = distLast - distNow
    self.__rangeDriven += difference if 10 > difference > -5 else 0
```

5.5.6 getControl

Fúzia riadenia. Z agenta máme údaj o rýchlosti. Natáčanie získame pomocou neurónovej siete, ktorá je používaná priamo v tejto metóde. Metóda podporuje aj priame použitie agentového riadenia (autopilot z CARLA) v prípade, že by užívateľ chcel len niečo skontrolovať. V projekte reálne tento autopilot nie je využitý.

```
def getControl(self, useAutopilot=False):
    '''
    calculate the steering:
    :param useAutopilot: use autopilot
    :return: carla.Control
```

```
    '''
    maxSteerChange = self.dynamicMaxSteeringChange()
    control, waypoint = self.agentAction()
    agentSteer = control.steer

    if useAutopilot:
        self.steer = control.steer
        return control

    radar = self.sensorManager.radarMeasurement()
    left, right = self.sensorManager.lines()

    inputs = self.processInputs(left, right, radar, agentSteer, waypoint)
    outputNeural = self.nn.run(inputs, maxSteerChange)[0][0]
    self.steer = self.limitSteering(outputNeural)

    self.recordEachStep(agentSteer)
    control.steer = self.steer
    return control
```

5.5.7 limitSteering

Zabráni nečakane veľkým zmenám, v prípade, že nastanú (nemali by nastať).

```
def limitSteering(self, askedChange):
    '''
    block bigger changes - this should never happen, but we need to ensure
it
    :param askedChange: change of steering
    :return: real change of steering
    '''
    actualSteer = self.steer
    askedSteer = actualSteer + askedChange

    if askedSteer > self.limit:
        askedSteer = self.limit
    elif askedSteer < -self.limit:
        askedSteer = -self.limit

    return askedSteer
```

5.5.8 processInputs

Na základe config súboru sa vyberie, ktoré vstupy chceme použiť.

```
def processInputs(self, left, right, radar, agentSteer, waypoint):
    '''
    based on config, we want to use just some of inputs. This function will
pick them for us
    :param left: left line from detector
    :param right: right line from detector
    :param radar: radar measures
    :param agentSteer: suggested agent steering
    :param waypoint: the next waypoint from navigation
    :return: ndarray of all inputs
    '''
    inputs = np.array([])

    for asked in self.askedInputs:
        if asked == InputsEnum.linedetect:
            inputs = np.append(inputs, self.nn.normalizeLinesInputs(left,
right))
        elif asked == InputsEnum.radar:
```

```
            inputs = np.append(inputs, self.nn.normalizeRadarInputs(radar))
        elif asked == InputsEnum.agent:
            inputs = np.append(inputs, self.nn.normalizeAgent(agentSteer))
        elif asked == InputsEnum.metrics:
            inputs = np.append(inputs,
self.nn.normalizeMetrics(self.metrics, self.limit))
        elif asked == InputsEnum.binaryknowledge:
            inputs = np.append(inputs,
self.nn.normalizeBinary(self.getBinaryKnowledge(left, right, agentSteer,
radar)))
        elif asked == InputsEnum.navigation:
            inputs = np.append(inputs,
self.nn.normalizeNavigation(self.location, waypoint))

    return inputs
```

5.5.9 storeCurrentData

Uloženie histórie pohybu vozidla, pravej, ľavej čiary a ideálnej trajektórie na základe navigácie.

```
def storeCurrentData(self):
    '''
    when we are running test scenarios / showing the best on training, we
want to store the history of movement of
    the vehicle, left, right lane and optimal path based on navigation -
store it in global class members
    :return: None
    '''
    if self.debug:
        waypoints = self.agent.get_waypoints()[0]
        if waypoints:
            self.__positionHistory.append(self.location)
            ll = waypoints.get_left_lane()
            if ll:
                self.__leftLinePlanner.append(ll.transform.location)
            rl = waypoints.get_right_lane()
            if rl:
                self.__rightLinePlanner.append(rl.transform.location)
            self.__optimalPath.append(waypoints.transform.location)
```

5.5.10 returnVehicleResults

Uložené dáta z predošlej metódy vráti vo forme 4 výstupov.

```
def returnVehicleResults(self):
    '''
    when we are running test scenarios / showing the best on training,
return all eligible measures when vehicle
     ends
    :return: position, left line, right line, navigation path
    '''
    return self.__positionHistory, self.__leftLinePlanner,
self.__rightLinePlanner, self.__optimalPath
```

5.5.11 dynamicMaxSteeringChange

Prepočet maximálnej zmeny natáčania kolies podľa NS.

```
def dynamicMaxSteeringChange(self):
    '''
    Max steering change based on the current speed of vehicle.
    If speed is more than 10km/h, we will use 0,8/speed; else just 0,8.
    :return: float
```

```
    '''
    # Speed will be 0 - 50, so max division will be 5
    speed = self.getSpeed() / 10
    return self.defaultSteerMaxChange / speed if speed > 1 else
self.defaultSteerMaxChange
```

5.5.12 checkGoal

Zistí, či je vozidlo v (čiastkovom) cieli.

```
def checkGoal(self):
    '''
    Checks, how far is vehicle from current goal. If goal is reached, new
waypoint is loaded. When no more waypoints
    it will return True as signing that current task is done.
    :return: bool
    '''
    dist = self.diffToLocation(self.goal)
    self.print(f"Distance to goal is: {dist}")
    if dist < 0.2 or self.agent.done():
        self.__reachedGoals += 1
        if not self.path.empty():
            self.goal = self.path.get()
        else:
            return True
```

5.5.13 standing

V procese trénovania kontrolujeme, či vozidlo náhodou nestojí (ak je jeho rýchlosť 100 po sebe idúcich krokov menšia ako 5km/h)

```
def standing(self):
    '''
    check, if vehicle is moving or standing. If standing for longer period
(100+ticks) returns True
    :return: bool
    '''
    speed = self.getSpeed()
    if speed < 5:
        self.vehicleStopped += 1
    else:
        self.vehicleStopped = 0

    if self.vehicleStopped >= 100:
        print("VEHICLE IS STOPPED!")
        return True
    else:
        return False
```

5.5.14 inCycle

Obdobne, pri trénovaní kontrolujeme aj či vozidlo nechodí stále v jednom kruhu.

```
def inCycle(self):
    '''
    determine, that vehicle is not moving towards the goals and probably
are stucked in cycle. It's used just in
    training scenarios.
    :return: bool
    '''
    now = time.time()
    timeBool = now > 300 + self.startTime  # gives timeout 5 min
```

```
    if len(self.toGoal) < self.toGoal.maxlen:
        dequeBool = False
    else:
        left = self.toGoal.popleft()
        right = self.toGoal.pop()
        self.toGoal.append(right)
        dequeBool = True if abs(left - right) < 5 else False
    if timeBool or dequeBool:
        return True
    else:
        return False
```

5.5.15 applyConfig

V prípade testovania chceme aby sa program správal mierne inak. To zabezpečí táto metóda:

```
def applyConfig(self, testing):
    '''
    If we are running test config, we want to change a few things - Record
the vehicle's path more frequently
                                                                    apply
test config for sensors/cameras
    :param testing: bool
    :return: None
    '''
    self.debug = testing
    if testing:
        self.numMeasure = 8
        self.sensorManager.applyTesting()
```

5.5.16 getLocation

Vráti aktuálnu pozíciu vozidla na mape (a zároveň ju pridelí do self.location)

```
def getLocation(self):
    '''
    FILLS self.location with location of VEHICLE
    :return: carla.Location
    '''
    self.location = self.me.get_location()
    if self.debug:
        self.print3Dvector(self.location, "Location")
    return self.location
```

5.5.17 diffToLocation/errInLocation

Chyba medzi aktuálnou pozíciou vozidla a žiadanou, prípadne dvoch pozícii na mape.

```
def diffToLocation(self, location: carla.Location):
    '''
    difference to location from current location of vehicle
    :param location: carla.Location
    :return: distance [float]
    '''
    self.getLocation()
    dist = location.distance(self.location)
    return dist

@staticmethod
def errInLocation(l1: carla.Location, l2: carla.Location):
    '''
    error from L1 to L2
```

```
    :param l1: carla.Location
    :param l2: carla.Location
    :return: distance [float]
    '''
    dist = l1.distance(l2)
    return dist
```

5.5.18 getBinaryKnowledge

Výpočet binary knowledge vstupov pre použitie NS.

```
def getBinaryKnowledge(self, left, right, agentSteer, radar) -> list:
    '''
    Process all needed datas and calc binary knowledge. More information in
CarlaConfig.py
    :return: list(len=4) of binary knowledges (-1,0, or 1)
    '''
    try:
        left1 = np.abs(left[0])
        right1 = np.abs(right[0])
        '1: Based on lines detected (left line is closer than right -> turn
right: 1)'
        if left1 > right1:
            one = 1
        elif left1 < right1:
            one = -1
        else:
            one = 0
    except:
        one = 0
    '2: Based on difference between agent steering and actual steering'
    if self.steer < agentSteer:
        two = -1
    elif self.steer > agentSteer:
        two = 1
    else:
        two = 0
    '3: Which radar measurement has the shortest range [-1 for left, 0 for
center, 1 for right]'
    minRadar = np.min(radar)
    if minRadar == radar[0]:
        three = -1
    elif minRadar == radar[1]:
        three = 0
    else:
        three = 1
    '4: Based on actual speed towards goal'
    x = abs(self.velocity.x)
    y = abs(self.velocity.y)
    diffX = abs(self.goal.x - self.location.x)
    diffY = abs(self.goal.y - self.location.y)
    # We can expect that direction will be ok, so just use abs values
    if diffX > diffY and y > x:
        four = 1
    elif diffY > diffX and x > y:
        four = -1
    else:
        four = 0
    return [one, two, three, four]
```

5.5.19 getSpeed

Vypočíta aktuálnu rýchlosť vozidla v km/h.

```
def getSpeed(self):
    '''
    FILLS   self.velocity with 3d vector of velocity
            self.speed with speed in km/h of vehicle
    :return: self.speed
    '''
    self.velocity = self.me.get_velocity()
    self.speed = 3.6 * math.sqrt(self.velocity.x ** 2 + self.velocity.y **
2 + self.velocity.z ** 2)
    return self.speed
```

5.5.20 ref

Vráti to referenciu na objekt Vehicle v rámci CARLA simulátora (vozidlo v simulácii)

```
def ref(self):
    '''
    :return: carla.Vehicle object of Vehicle
    '''
    return self.me
```

5.5.21 destroy

Zabezpečí zničenie vozidla a všetkých jeho vnorených súčastí zo simulácie.

```
def destroy(self):
    '''
    Destroy all sensors attached to vehicle and then vehicle on it's own
    :return: none
    '''
    self.print("Destroying Vehicle {id}".format(id=self.vehicleID))
    try:
        self.sensorManager.destroy()
        self.me.destroy()
        self.environment.tick()
    except:
        print(f"Error in destroying of {self.vehicleID}")
```

5.6 SensorManager

Manažér vytvorí všetky potrebné senzory a uloží ich do listov. Následne bude používať tie, ktoré užívateľ žiada. Základný element je objekt Sensor, ktorý tvorí základ každého senzoru/kamery a zabezpečuje jednoduchú manipuláciu s objektami.

```
def __init__(self, vehicle, environment):
    '''
    Create a SensorManager object.
    :param vehicle: reference to Vehicle's object
    :param environment: reference to CarlaEnvironment's object
    '''
    super().__init__()
    self._queues = []
    self.sensors = []
    self.cameras = []
    self.count = 0
    self.readySensors = 0
    self.vehicle = vehicle

    self.environment = environment
    self.config = environment.config
    self.debug = not environment.trainingMode

    self.ldCam = LineDetectorCamera(self, False, False)
    self.rgbCam = Camera(self, False, False)
    self.segCam = SegmentationCamera(self, False, False)

    self.collision = CollisionSensor(self, False)
    self.radar = RadarSensor(self, False)
    self.lidar = LidarSensor(self, False)
    self.obstacleDetector = ObstacleDetector(self, False)
    self.laneInvasionDetector = LaneInvasionDetector(self, False)

    self.addToSensorsList()
    self.activate()
```

5.6.1 addToSensorsList

Pridá do globálneho listu senzorov tie, ktoré majú byť pridané na základe config súboru.

```
def addToSensorsList(self):
    '''
    Based on config's settings add sensors to list.
    :return: None
    '''
    settings: dict
    settings = self.config.readSection("Sensors")
    # SENSORS
    if convertStringToBool(settings.get("radarsensor")):
        self.sensors.append(self.radar)
    if convertStringToBool(settings.get("collisionsensor")):
        self.sensors.append(self.collision)
    if convertStringToBool(settings.get("obstacledetector")):
        self.sensors.append(self.obstacleDetector)
    if convertStringToBool(settings.get("lidarsensor")):
        self.sensors.append(self.lidar)
    if convertStringToBool(settings.get("laneinvasiondetector")):
        self.sensors.append(self.laneInvasionDetector)
```

```
    # CAMERAS
    if convertStringToBool(settings.get("linedetectorcamera")):
        self.sensors.append(self.ldCam)
        self.cameras.append(self.ldCam)
    if convertStringToBool(settings.get("defaultcamera")):
        self.sensors.append(self.rgbCam)
        self.cameras.append(self.rgbCam)
    if convertStringToBool(settings.get("segmentationcamera")):
        self.sensors.append(self.segCam)
        self.cameras.append(self.segCam)
```

5.6.2 activate

Aktivuje všetky senzory, ktoré sú v globálnom liste senzorov.

```
def activate(self):
    '''
    Activate all sensors from self.sensors list
    :return: None
    '''
    for sensor in self.sensors:
        self.print(f"Activating {self.count}: {sensor.name}")
        self.count += 1
        sensor.activate()
```

5.6.3 processSensors

Vykoná sa obslužná funkcia po prijatí nových dát pre senzor.

```
def processSensors(self):
    '''
    When tick came from server, process all sensors.
    :return: None
    '''
    for sensor in self.sensors:
        sensor.on_world_tick()
```

5.6.4 isCollided

Rýchla odpoveď z kolízneho senzora o stave kolízii:

```
def isCollided(self):
    '''
    Fast reference to collision sensor's method
    :return: True if vehicle have collided, False if not
    '''
    return self.collision.isCollided()
```

5.6.5 lines

Výber čiar z LineDetectionCamera.

```
def lines(self) -> (np.ndarray, np.ndarray):
    '''
    Fast reference to LineDetection camera's left and right line
    :return: left [ndarray], right line [ndarray]
    '''
    retLeft = copy.deepcopy(self.ldCam.left)
    retRight = copy.deepcopy(self.ldCam.right)
    self.ldCam.resetLines()
    return retLeft, retRight
```

5.6.6 radarMeasurement

Návrat spracovaných radarových meraní z RADAR objektu.

```
def radarMeasurement(self) -> np.ndarray:
    '''
    Fast reference to Radar's averages ranges
    :return: average ranges of measures [ndarray]
    '''
    return self.radar.returnAverageRanges()
```

5.6.7 destroy

Zabezpečí odstránenie všetkých senzorov z prostredia.

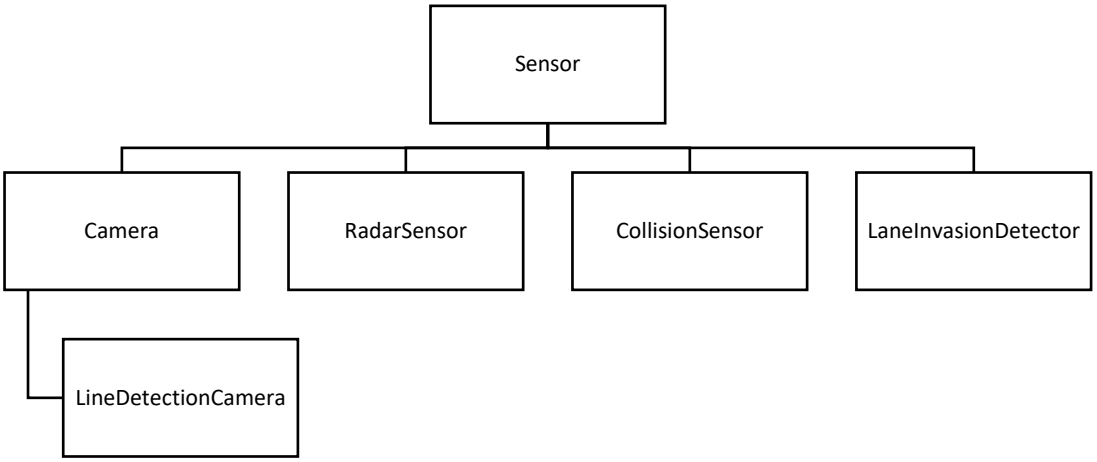
```
def destroy(self):
    '''
    Destroy all current sensors in self.sensors
    :return: None
    '''
    self.print(f"Invoking deletion of sensors of {self.vehicle.vehicleID}
vehicle!")
    for sensor in self.sensors:
        self.print(f"Deleting sensor {sensor.name}")
        sensor.destroy()
```

5.6.8 applyTesting

V prípade testovacej konfigurácie chceme dodatočné správy písať do konzoly a taktiež vykresľovať kamery.

```
def applyTesting(self):
    '''
    Apply test configuration:
    Show all attached cameras + print debug messages to console
    :return: None
    '''
    self.debug = True
    for camera in self.cameras:
        camera.show = True
```

5.7 Sensor



Sensor je hlavný objekt, ktorý je dedený všetkými integrovanými senzormi z CARLA. Ako sme spomínali v predošlej kapitole, je to najmä na jednoduchú a rýchlu manipuláciu so snímačmi z manažéra. V prípade, že chceme vytvoriť nový senzor, stačí postupovať podľa tejto triedy, prípadne si vytvoriť isté pomocné metódy, ktoré budú potrebné.

```
def __init__(self, manager, debug):
    '''
    Create sensor object
    :param manager: reference to SensorManager
    :param debug: bool
    '''
    super(Sensor, self).__init__()
    self.sensor = None
    self.bp = None
    self.where = None
    self.manager = manager
    self.name = "Sensor"
    self.queue = queue.Queue()
    self.vehicle = manager.vehicle
    self.debug = debug
```

5.7.1 callBack

Metóda, ktorá spracuje dáta. Volaná po odsimulovaní jedného kroku, pričom ho zabezpečuje SensorManager. Každý senzor má svoj tvar dát, ktoré je potrebné odčítať z dokumentácie: https://carla.readthedocs.io/en/latest/ref_sensors/, pri každom senzore je to sekcia output.

```
def callBack(self, data):
    '''
    Function that will handle data from sensor/camera later on
    :param data: sensor's data (blueprint)
    :return: None
    '''
    pass
```

5.7.2 activate

Metóda, ktorá aktivuje senzor, je volaná taktiež zo SensorManagera. Spawne senzor na vozidlo a taktiež nastaví aby sa každé príchodzie dáta uložili do queue daného senzora.

```
def activate(self):
    '''
    activate current sensor:
     - spawn it on world, attached to vehicle's BP
     - add any new incoming data to queue
    :return: None
    '''
    self.sensor = self.world().spawn_actor(self.bp, self.where,
attach_to=self.reference())
    self.sensor.listen(lambda data: self.queue.put(data))
```

5.7.3 on_world_tick

Zabezpečí, že sa zavolá handler pre daný senzor, len ak v queue senzora už čakajú nejaké dáta.

```
def on_world_tick(self):
    '''
    call handler of data when any data are ready in queue
    :return: None
    '''
    if self.debug:
        print(f"[{self.name}] on world tick")
    if self.queue.qsize() > 0:
        self.callBack(self.queue.get())
```

5.7.4 reference

Vráti referenciu na konkrétny blueprint, ktorý zabezpečuje senzor.

```
def reference(self):
    '''
    :return: Reference to vehicle's BP (real vehicle model in carla)
    '''
    return self.vehicle.ref()
```

5.7.5 setVehicle

Nastaví vozidlo, ku ktorému je senzor pripevnený

```
def setVehicle(self, vehicle):
    '''
    :param vehicle: Vehicle object
    :return: None
    '''
    self.vehicle = vehicle
```

5.7.6 blueprints

Vráti knižnicu pre jednoduchšiu manipuláciu s blueprints.

```
def blueprints(self):
    '''
    Fast reference to blueprints library
    :return: carla.Blueprints library
    '''
    return self.vehicle.environment.blueprints
```

5.7.7 world

```
def world(self):
    '''
    Fast reference to world
    :return: carla.World
    '''
    return self.vehicle.environment.world
```

5.7.8 lineDetector

Objekt detektora čiar, ktorý je využívaný najmä pre kameru.

```
def lineDetector(self):
    '''
    Fast reference to FALineDetector object
    :return: FALineDetector object
    '''
    return self.vehicle.fald
```

5.7.9 config

Referencia na CarlaConfig

```
def config(self):
    '''
    Fast reference to CarlaConfig object
    :return: CarlaConfig
    '''
    return self.vehicle.environment.config
```

5.7.10 destroy

Zabezpečí zmazanie senzora.

```
def destroy(self):
    '''
    Destroy self from world
    :return: None
    '''
    if self.sensor is not None:
        try:
            self.sensor.destroy()
        except:
            pass
```

5.8 Camera

Základný objekt pre kamery. Dedí senzor, takže základná kostra je rovnaká, ponúka však nové funkcionality spojené najmä s vykresľovaním kamier. V prípade integrácie kamery sa odporúča použitie Camera ako základnej triedy. Má uložený dictionary options, v ktorom sú prednastavené odporúčané transformácie obrazu z kamery aby sme dosiahli očakávaný výsledok. Každá kamera musí mať meno.

```
options = {
    'Main': ['sensor.camera.rgb', cc.Raw],
    'Depth': ['sensor.camera.depth', cc.LogarithmicDepth],
    'Semantic Segmentation': ['sensor.camera.semantic_segmentation',
cc.CityScapesPalette],
    'LineDetection': ['sensor.camera.rgb', cc.Raw]
```

```
}

def __init__(self, manager, debug=False, show=True):
    '''
    Init Camera.
    :param manager: SensorManager
    :param debug: bool
    :param show: bool -> if true, camera's image will be displayed

    Set height and width of image based on config
    Set bounding box around the car to calculate proper middle of the 3D
area
    '''

    super().__init__(manager, debug)
    d = self.config().readSection('Camera')
    self.camHeight = int(d["height"])
    self.camWidth = int(d["width"])
    self.image = None
    self.show = show
    self.drawingThread = None
    self.stop = False
    self.bound_x = 0.5 + self.reference().bounding_box.extent.x
    self.bound_y = 0.5 + self.reference().bounding_box.extent.y
    self.bound_z = 0.5 + self.reference().bounding_box.extent.z
    self.name = 'Main'
    self.create()
```

5.8.1 create

Vytvorí kameru a na základe mena použije nastavenia.

```
def create(self):
    '''
    Create camera based on name of the camera, then apply default settings
from options dictionary
    :return: None
    '''
    typeOfCamera = self.options.get(self.name)[0]
    self.bp = self.blueprints().find(typeOfCamera)
    self.bp.set_attribute('image_size_x', f'{self.camWidth}')
    self.bp.set_attribute('image_size_y', f'{self.camHeight}')
    self.bp.set_attribute('fov', '110')
    self.where = carla.Transform(carla.Location(x=-2.0 * self.bound_x,
y=+0.0 * self.bound_y, z=2.0 * self.bound_z),
                                 carla.Rotation(pitch=-8.0))
```

5.8.2 callBack

Pretvorí dáta na obrázok, aplikuje transformáciu. V prípade, že ide o základnú kameru, pokúsi sa vykresliť obrázok (závisí na nastevení parametra show).

```
def callBack(self, data):
    '''
    handle Camera's output - convert image data, if we are using some
special camera
    :param data: carla.Image
    :return:
    '''
    if self.debug:
        print(f"Entering {self.name} callback!")
    data.convert(self.options.get(self.name)[1])
```

```
    i = np.array(data.raw_data)
    i2 = i.reshape((self.camHeight, self.camWidth, 4))
    self.image = i2[:, :, :3]
    if self.isMain():
        self.invokeDraw()
```

5.8.3 draw

Zabezpečuje vykreslenie kamerového obrazu pomocou openCV, pozor, táto metóda je spúšťaná v separátnom threade a nemožno ju volať v hlavnom – zablokovala by ostatné funkcionality programu. ak nastavíme self.stop na True, vykresľovanie končí – po ukončení jazdy sa ukončí vykresľovanie automaticky.

```
def draw(self):
    '''
    !IN SEPARATE THREAD!
    Show the image from camera using openCV imshow.
    When stop is coming from SensorManager, we need to break the infinite
while
    :return: None
    '''
    print(f"[{self.name}]Starting drawing process")
    while True:
        drawingImg = copy.copy(self.image)
        if self.stop:
            print("STOP")
            self.stop = False
            break
        cv2.imshow("Vehicle {id}, Camera
{n}".format(id=self.vehicle.vehicleID, n=self.name), drawingImg)
        cv2.waitKey(1)
```

5.8.4 invokeDraw

Metóda, ktorá zabezpečí vytvorenie vlákna na vykreslenie kamier.

```
def invokeDraw(self):
    '''
    Invoke draw - start separate thread. Ensure, that could happen just
once per every camera!
    :return: None
    '''
    if self.show and self.drawingThread is None:
        self.drawingThread = threading.Thread(target=self.draw)
        self.drawingThread.start()
```

5.8.5 isMain

Na základe mena sa určí či ide o základnú kameru.

```
def isMain(self):
    '''
    This is determined by camera's name
    :return: bool
    '''
    return self.name == 'Main'
```

5.8.6 destroy

Preťaženie základnej destroy metódy, ktorá musí zabezpečiť aj ukončenie separátneho vlákna, v ktorom beží vykresľovanie obrazu.

```
def destroy(self):
    '''
    destroy Camera, but first wait until thread that is drawing the
camera's image is finished
    :return:
    '''
    if self.drawingThread is not None:
        self.show = False
        self.stop = True  # break showing threads
        for _ in range(10):
            if self.stop:
                self.manager.environment.tick()
        self.drawingThread = None
    super(Camera, self).destroy()
```

5.9 Na záver

V prípade nejasností ohľadom spúšťania programu, či nejakých iných otázok ma môžete kontaktovať na:

xchylik@stuba.sk

marko.chylik@gmail.com