Leitfaden für Simulation mit

Inhaltsverzeichnis

1. Ziel

Ziel ist es, mit Hilfe der Simulationsumgebung von BeamNG, CAN-Nachrichten abzurufen. Dabei werden die Daten aus BeamNG zunächst an einen MQTT-Broker gesendet. Anhand eines Simulink-Modells sollen die an den Broker gesendeten Fahrzeugdaten übermittelt werden. Die CAN-Schnittstelle, die mit dem Laptop und dem RaspberryPi verbunden ist, sorgt dafür, dass die erforderlichen CAN-Anfragen als Antwort vom MQTT-Broker an die Simulation in Simulink übertragen werden.
Das Fahrspiel BeamNG soll ein reales Fahrzeug simulieren und kann nach dem erfolgreichen Testen von einer realen Umgebung ersetzt werden.

2. Hardware

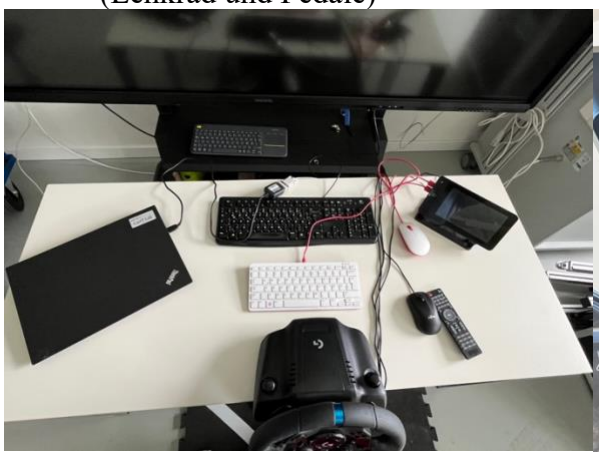
Um die Infrastruktur für die Simulation herzustellen, werden folgende Hardware-Elemente (und Dateien) benötigt:

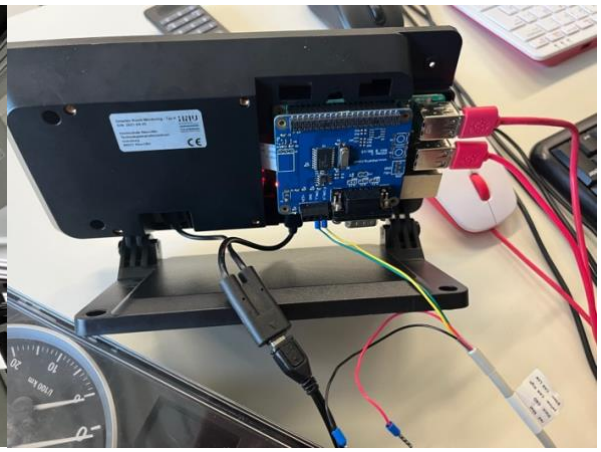
- **Car-IT-Laptop:**
- Benutzername: .\Car-IT-LT0618BE(LaptopNr.) Passwort: 10112022
- Laptop mit LAN verbunden
- Vector Canoe kann man parallel laufen lassen zum Beobachten vom BusTraffic

- **PC mit Bildschirm:**
- Kein Benutzer
- PC mit LAN verbunden
- Py-Script zum Starten der Simulation auf dem Desktop: bng_to_mqtt.py (starten in Visual Studio Code)

- **RaspberryPi**
- Zum Starten das Netzteil am Strom anschließen
- Kein Benutzer
- Datei auf dem Desktop: can_sim -> BeamNG_CarIT_Simulator -> mqtt_to_can.py

- **Weitere Hardware**
- Vector CAN Interface (VN1610)
- Vector CANcableA
- (Lenkrad und Pedale)



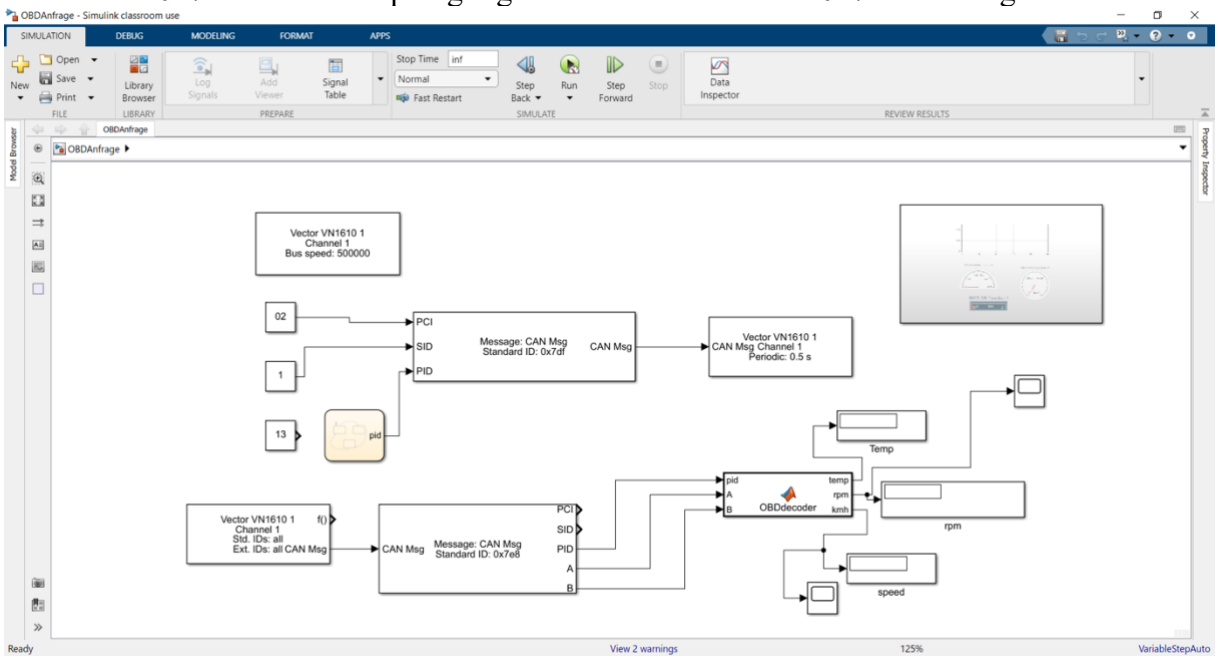


3. Funktionsweise

3.1 Simulink

Hinweis: Die Simulation kann über den Run-Button gestartet werden.

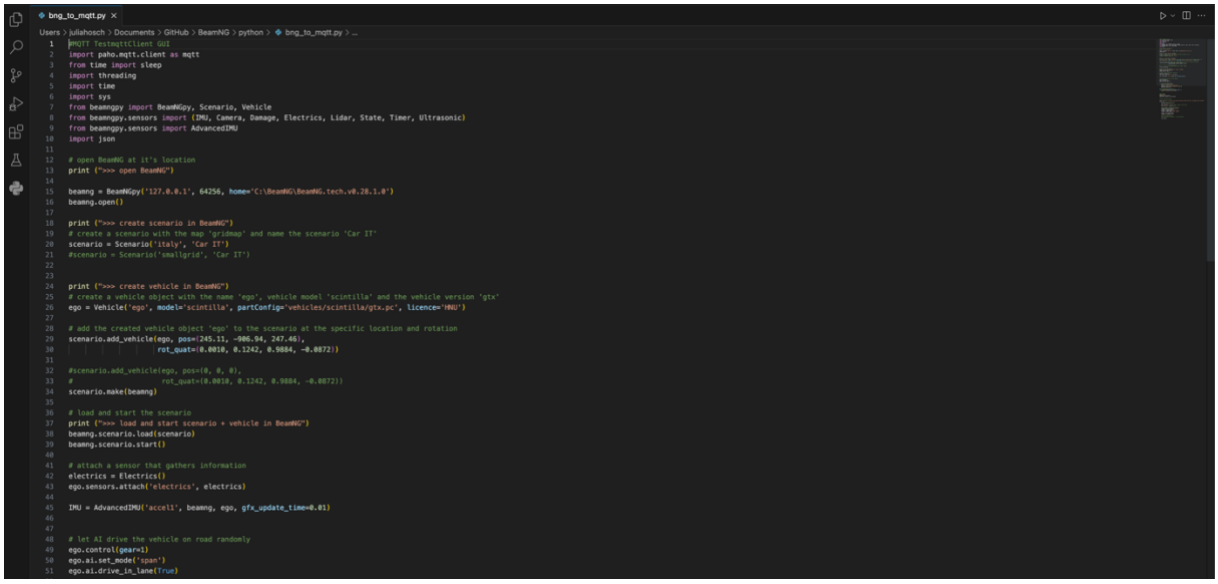
Mithilfe der Simulationsumgebung in Simulink werden CAN-Nachrichten angefragt und empfangen. Die Beispieldatei „OBDAnfrage.slx“ liegt auf dem Desktop des Car-IT-Laptops. Die Kommunikation erfolgt über die CAN-Schnittstelle VN610. Im Folgenden werden die Werte für Temperatur, Drehzahl und Geschwindigkeit abgefragt. Dabei wird eine Nachricht über die ID 0x7DF an alle Empfänger gesendet und von der ID 0x7EF zurückgeschickt.



3.2 BeamNG mit PyPi und MQTT

Hinweis: Alle Python-Skripte können über den Play Button oben rechts in der Ecke gestartet werden.

Um Fahrzeugdaten aus BeamNG abfragen zu können, sollte das Fahrspiel mit folgendem Py-Script über Visual Studio Code gestartet werden: bng_to_mqtt.py (liegt auf dem Desktop). Dieses Skript setzt ein Szenario voraus. In diesem Beispiel haben wir ein Fahrzeug namens „ego“ des Modells „Scintilla“ mit dem Kennzeichen „HNU“. Wir befinden uns in Italien und nennen das Szenario „Car IT“. Bei BeamNG lassen sich zahlreiche Sensordaten abfragen. In diesem Skript ist dies auf Electrics beschränkt, da dies schon eine ausreichende Anzahl an Informationen liefert. Weitere Sensorik kann unter der Seite https://beamngpy.readthedocs.io/en/latest/beamngpy.html#id2 eingesehen werden.
Es ist möglich, das Fahrzeug mit einer AI fahren zu lassen (Zeile 34-36). Falls man selbst fahren möchte, kann man die Zeilen 34 und 35 auskommentieren.



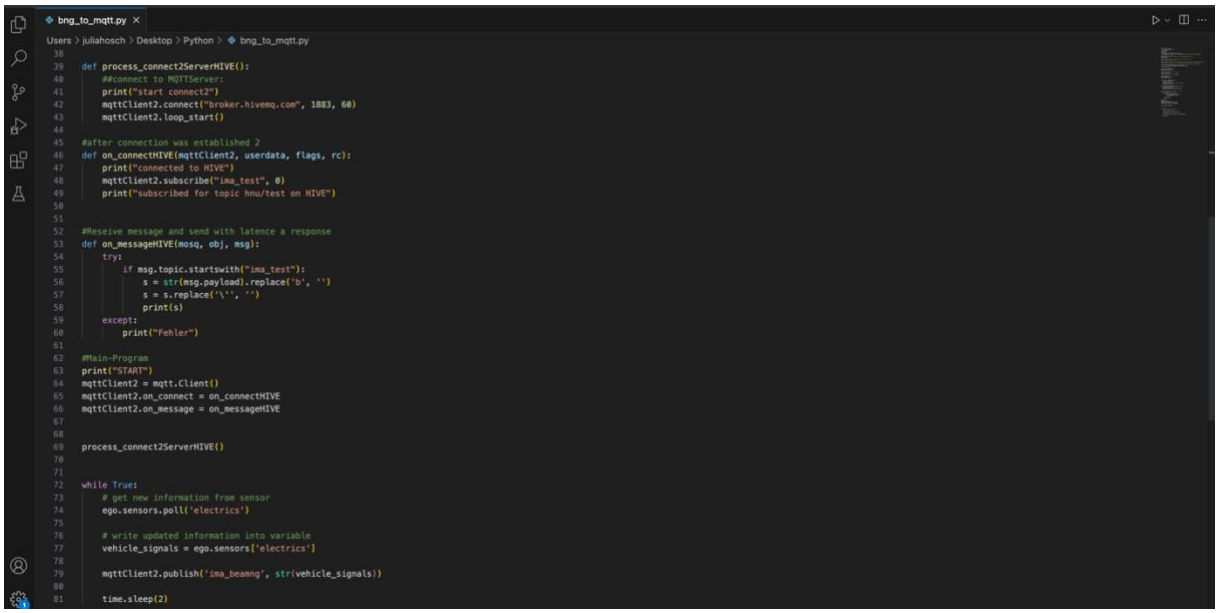
```
#MQTT TestmqttClient GUI
import paho.mqtt.client as mqtt
from time import sleep
import threading
import time
import sys
from beamngpy import BeamNGpy, Scenario, Vehicle
from beamngpy.sensors import (IMU, Camera, Damage, Electrics, Lidar, State, Timer, Ultrasonic)
from beamngpy.sensors import AdvancedIMU
import json

# open BeamNG at it's location
print (">>> open BeamNG")

beamng = BeamNGpy('127.0.0.1', 64256, home='C:\BeamNG\BeamNG.tech.v0.28.1.0')
beamng.open()

print (">>> create scenario in BeamNG")
# create a scenario with the map 'gridmap' and name the scenario 'Car IT'
scenario = Scenario('italy', 'Car IT')
#scenario = Scenario('smallgrid', 'Car IT')


print (">>> create vehicle in BeamNG")
# create a vehicle object with the name 'ego', vehicle model 'scintilla' and the vehicle version 'gtx'
ego = Vehicle('ego', model='scintilla', partConfig='vehicles/scintilla/gtx.pc', licence='HNU')

# add the created vehicle object 'ego' to the scenario at the specific location and rotation
scenario.add_vehicle(ego, pos=(245.11, -986.94, 247.46),
                     rot_quat=(0.0010, 0.1242, 0.9884, -0.0872))

#scenario.add_vehicle(ego, pos=(0, 0, 0),
#                     rot_quat=(0.0010, 0.1242, 0.9884, -0.0872))
scenario.make(beamng)

# load and start the scenario
print (">>> load and start scenario + vehicle in BeamNG")
beamng.scenario.load(scenario)
beamng.scenario.start()

# attach a sensor that gathers information
electrics = Electrics()
ego.sensors.attach('electrics', electrics)

IMU = AdvancedIMU('accel1', beamng, ego, gfx_update_time=0.01)


# let AI drive the vehicle on road randomly
ego.control(gear=1)
ego.ai.set_mode('span')
ego.ai.drive_in_lane(True)
```

Hinweis: Falls Visual Studio Code die Python-API-Bibliothek nicht erkennen sollte, muss diese noch einmal über die Konsole neu installiert bzw. geupdatet werden:

```
pip install beamngpy
```

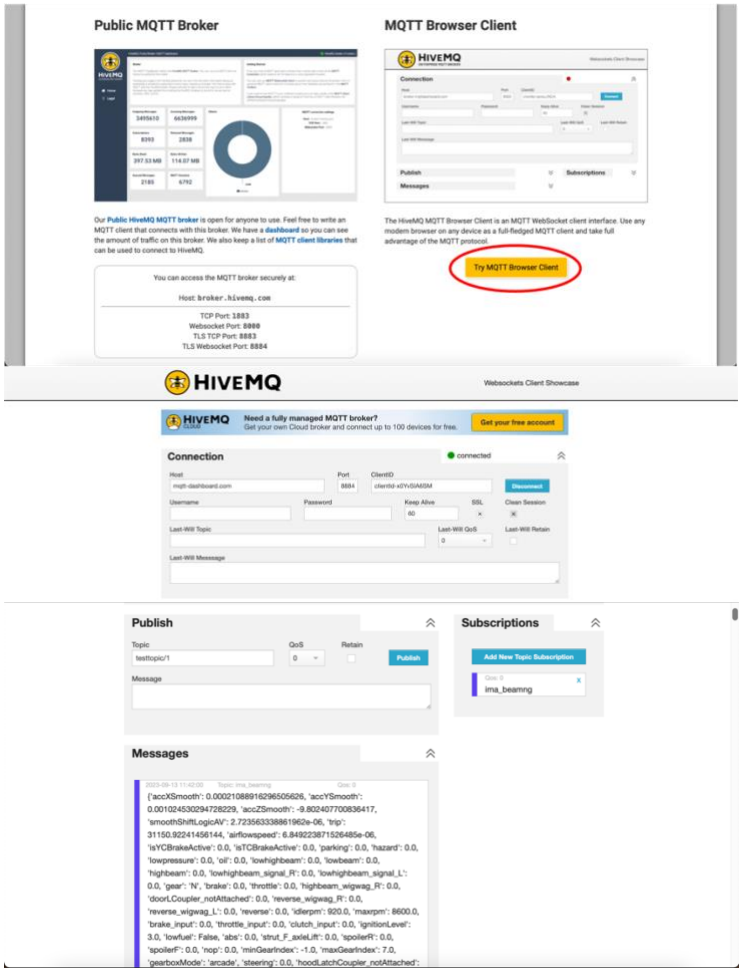
```
pip install --upgrade beamngpy
```

Nachdem die benötigten Daten abgerufen wurden, werden diese nun an den Broker HiveMQ gesendet. Dazu wird zunächst eine Verbindung mit dem Hostnamen „broker.hivemq.com“ auf Port 1883 hergestellt. Hier werden alle Sensordaten aus dem Feld „Electrics“ formatiert, in der Variablen „vehicle_signals“ gespeichert und an das Topic „ima_beamng“ gesendet.



```
def process_connect2ServerHIVE():
    ##connect to MQTTServer:
    print("start connect2")
    mqttClient2.connect("broker.hivemq.com", 1883, 60)
    mqttClient2.loop_start()

#after connection was established 2
def on_connectHIVE(mqttClient2, userdata, flags, rc):
    print("connected to HIVE")
    mqttClient2.subscribe("ima_test", 0)
    print("subscribed for topic hnu/test on HIVE")


#Reseive message and send with latence a response
def on_messageHIVE(mosq, obj, msg):
    try:
        if msg.topic.startswith("ima_test"):
            s = str(msg.payload).replace('b', '')
            s = s.replace('\'', '')
            print(s)
    except:
        print("Fehler")

#Main-Program
print("START")
mqttClient2 = mqtt.Client()
mqttClient2.on_connect = on_connectHIVE
mqttClient2.on_message = on_messageHIVE


process_connect2ServerHIVE()


while True:
    # get new information from sensor
    ego.sensors.poll('electrics')

    # write updated information into variable
    vehicle_signals = ego.sensors['electrics']

    mqttClient2.publish('ima_beamng', str(vehicle_signals))

    time.sleep(2)
```

Verbindet man sich unter https://www.hivemq.com/demos/websocket-client/ mit dem Websocket-Client auf HiveMQ und abonniert das Topic „ima_beamng“, kann man auf die jeweiligen Daten aus BeamNG zugreifen.



3.3 RaspberryPi mit MQTT und Simulink

In dem Codesnippet der Datei *mqtt_to_can.py* werden Anfragen aus dem Simulink-Modell *OBDAnfrage.slx* angenommen, die Antwort aus MQTT geholt und an Simulink über eine CAN-Schnittstelle übermittelt.

Zuerst werden die benötigten Bibliotheken importiert. Daraufhin müssen die Werte, die man abfragen möchte (in dem Fall Geschwindigkeit, Temperatur und Drehzahl) auf „0“ gesetzt werden. Damit wird vorübergehend ein sinnvoller Wert definiert, bevor reale Daten von MQTT empfangen werden. Des Weiteren verhindert man einen Error, falls keine Nachrichten empfangen werden können, da ein sinnvoller Wert definiert ist. Danach wird der Name des Topics für das spätere Abonnieren deklariert.
Zudem wird die CAN-Schnittstelle namens „can0“ als Typ can mit einer Bandbreite von 500.000 Bits pro Sekunde und einer Verzögerung von 0.1 Sekunden eingerichtet.

```python
# Bridge from MQTT to CAN-Bus using in RaspPi

import can
import os
import time

import sys
import struct
import socket
import os

import paho.mqtt.client as mqtt
from time import sleep
import threading
import time
import json

velocity = 0
rpm = 0
temp = 0

topic_str = "ima_beamng"


os.system("sudo /sbin/ip link set can0 up type can bitrate 500000")
time.sleep(0.1)

```

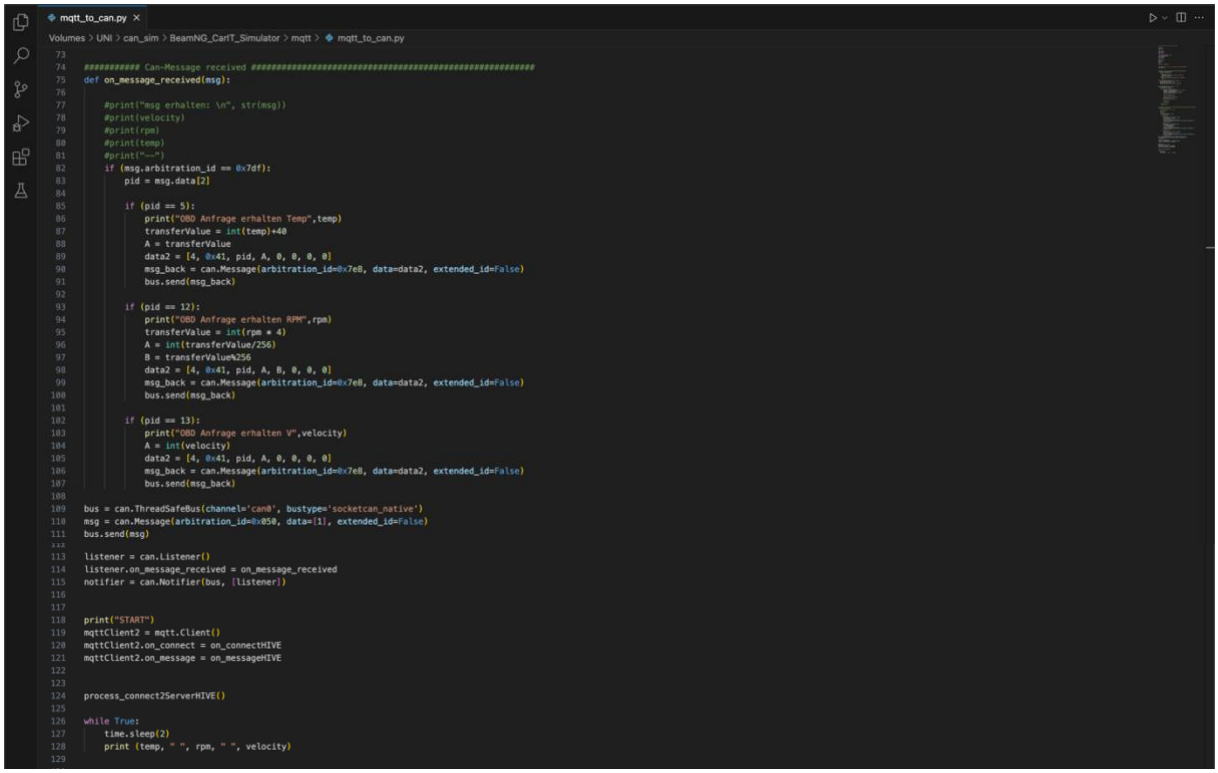
Im Folgenden wird eine Verbindung zu "broker.hivemq.com" auf Port 1883 hergestellt und sich für das Topic „ima_beamng“ abonniert.

```python
######## MQTT callback-methods ###################################
def process_connect2ServerHIVE():
    ##connect to MQTTServer:
    print("start connect2")
    mqttClient2.connect("broker.hivemq.com", 1883, 60)
    mqttClient2.loop_start()

#after connection was established 2
def on_connectHIVE(mqttClient2, userdata, flags, rc):
    print("connected to HIVE")
    mqttClient2.subscribe(topic_str, 0)
    print("subscribed for topic on HIVE: ", topic_str)
```

Nach dem Abonnieren werden die Nachrichten des jeweiligen Topics empfangen. Die darin enthaltenen Daten werden in den globalen Variablen gespeichert. Der Payload wird im JSON-Format empfangen.

```python

#Receive message and send with latence a response
def on_messageHIVE(mosq, obj, msg):
    global temp, rpm, velocity
    try:
        if msg.topic.startswith(topic_str):
            m_decode = str(msg.payload.decode("utf-8","ignore"))
            m_decode = m_decode.replace('\'', '"')
            m_decode = m_decode.replace('False', 'false')
            m_decode = m_decode.replace('True', 'true')

            #print(m_decode)
            m_in = json.loads(m_decode)

            velocity = m_in["wheelspeed"]*3.6
            rpm = m_in["rpm"]
            temp = m_in["oil_temperature"]

            #print(velocity)
            #print(rpm)
            #print(temp)
            #print("------")
    except:
        print("Fehler")
```

Beim Empfang einer CAN-Nachricht gibt es verschiedene Bedingungen, die für die angefragte PID erforderlich sind. Dabei wird der Transfer-Wert, der hinter der jeweiligen PID steckt, in einen physikalischen umgewandelt. Anschließend wird den Nachrichten die CAN-Schnittstelle „can0“, über den die Nachrichten gesendet werden sollen, zugeordnet.
Es wird sich mit dem MQTT-Server verbunden. Ein Listener reagiert nun auf neue CAN-Nachrichten, die alle 2 Sekunden für die angefragten PIDs ausgegeben werden.



4. Nützliche Links

https://beamngpy.readthedocs.io/en/latest/index.html#features

https://github.com/BeamNG/BeamNGpy/blob/master/examples/annotation_bounding_boxes.ipynb

https://beamngpy.readthedocs.io/en/latest/beamngpy.html#id2

https://beamngpy.readthedocs.io/_/downloads/en/latest/pdf/

Bei weiteren Fragen:

Frau Dr.-Ing. Prof. Dany Meyer
dany.meyer@hnu.de

Herr Dietmar Gräf
dietmar.graef@hnu.de

Letzte Änderung am 14.09.2023