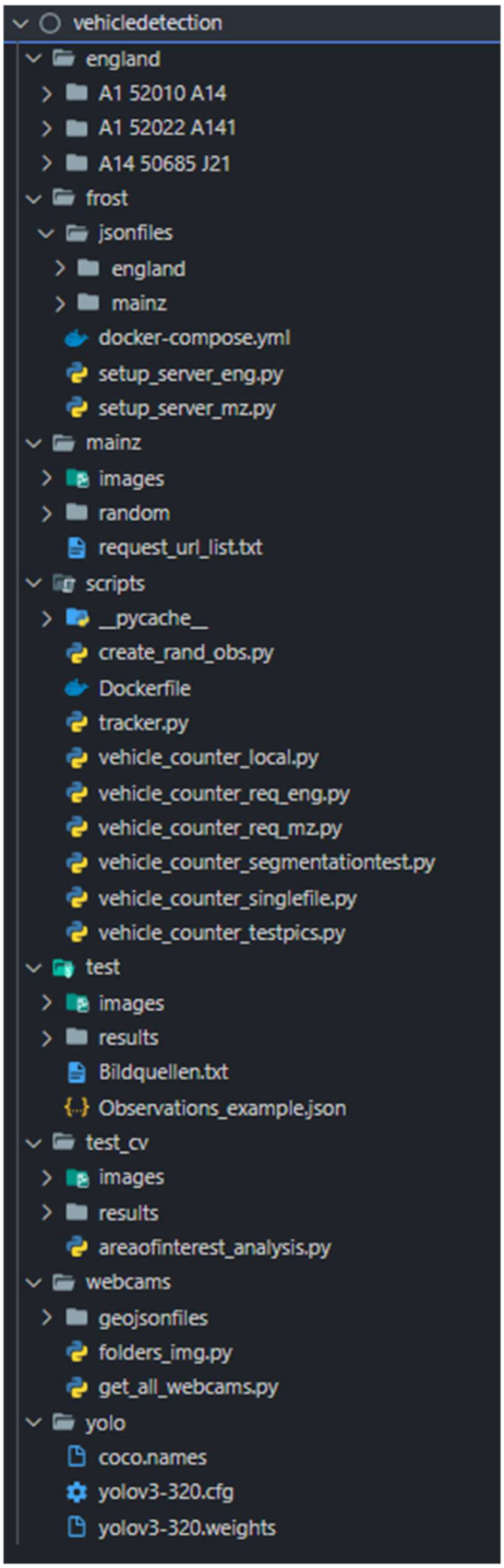
Dokumentation (Ablauf VS Code)

Enthaltene Datenstruktur im Ordner vehicledetection



Konfiguration

1. Im Ordner webcams das Skript **get_all_webcams.py** ausführen: Die Datei webcams.geojson wird im Ordner webcams/geojsonfiles/ gespeichert und enthält alle Webcamstandorte in RLP

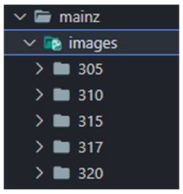
2. Verarbeitung in QGIS (QGIS_Vorgehen.pdf): Ergebnis sind 3 Dateien
   - **Webcams_mz.geojson**
   - **Camera_fahrspuren.geojson**
   - **Fahrtrichtungen_locations.geojson**

   ➔ Speicherung in webcams/geojsonfiles/

   ➔ In den Dateien Ä,ü,ö und ß ersetzen (ae, ue, oe, ss)

   (die notwendigen Daten sind auch im Ordner QGIS im Geopackage data.gpkg zu finden)

3. Im Ordner webcams das Skript folders_img.py ausführen: Anlegen von Ordnern mit dem Namen der Kamera-ID im Ordner mainz/images
   + Datei im Ordner Mainz mit Übersicht der Kamera-ID und Request-Url

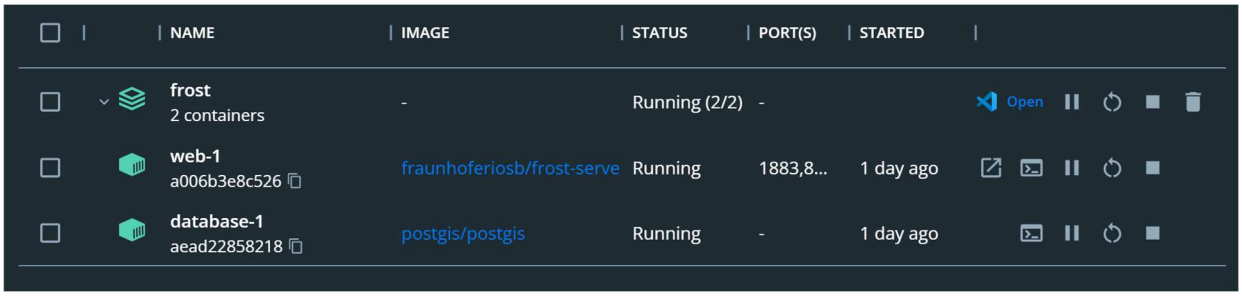



```
305;https://verkehr.rlp.de/api/webcams/305/pic/640x480?t=1653908021526
310;https://verkehr.rlp.de/api/webcams/310/pic/640x480?t=1653908021526
315;https://verkehr.rlp.de/api/webcams/315/pic/640x480?t=1653908021526
317;https://verkehr.rlp.de/api/webcams/317/pic/640x480?t=1653908021526
320;https://verkehr.rlp.de/api/webcams/320/pic/640x480?t=1653908021526
```

4. Nach der Installation von Docker Desktop und dem Download der docker-compose.yml Datei (https://github.com/FraunhoferIOSB/FROST-Server/blob/v2.x/scripts/docker-compose.yaml) kann der FROST-Server über den Befehl „docker compose up“ im Dateinpfad „…/vehicledetection/frost“ eingerichtet werden

   ➔ Mit dem Befehl „docker update --restart unless-stopped *container_id*“ wird ein automatischer Neustart der Container nach einem Reboot vorgenommen

   ➔ Zugang über http://localhost:8080/FROST-Server/v1.0 (oder anstatt localhost, eine andere Severadresse)



(Ansicht in Docker Desktop der laufenden Container)

5. Mit dem Skript frost/setup_server_mz.py (und setup_server_eng.py für die Demonstration mit Kameras aus England) werden dann die Objekte in den einzelnen Klassen angelegt

   ➔ Things, Sensors, ObservedProperty, Locations, Datastreams

   ➔ Unter Verwendung der Dateien: webcams_mz.geojson; camera_fahrspuren.geojson; fahrtrichtungen_locations.geojson

   ➔ Für England wurden 3 selektierte Kameras verwendet: Namen und Koordinaten in Array eintragen

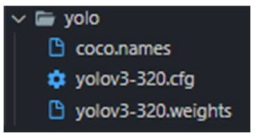
   ➔ Gleichzeitig werden die geposteten Objekte in JSON-Dateien im Ordner frost/jsonfiles/mainz oder /england gespeichert

Automatisierter Workflow

3 verschiedene Skripte

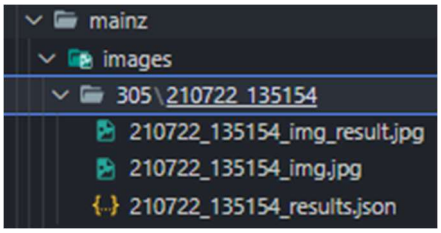
Bilddetektionsalgorithmus:

https://techvidvan.com/tutorials/opencv-vehicle-detection-classification-counting/

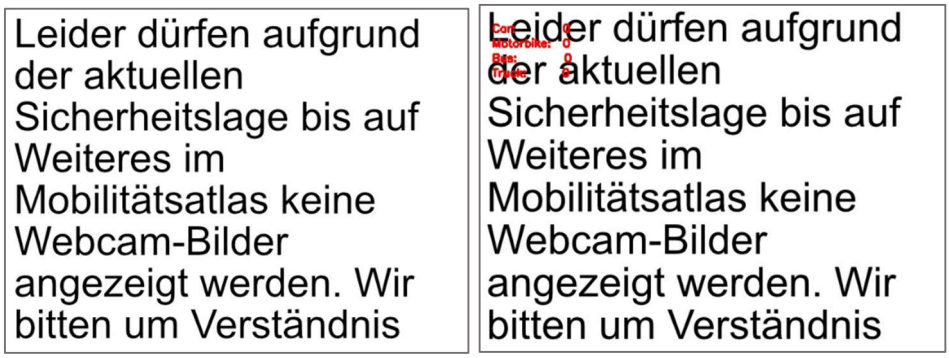


➔ Die für den YOLOv3 Algorithmus benötigten Dateien finden sich im Ordner yolo

I. Scripts/vehicle_counter_req_mz.py:
- Schleife über Kamera IDs
- Greift auf die API der Mainzer Webcams zu, um das aktuelle Bild abzufragen („https://verkehr.rlp.de/api/webcams/"+id+"/pic/640x480?t=1653908021526")
- Objektdetektion im Bild
- Gleichzeitig wird ein Ordner mit dem aktuellen Timestamp angelegt (mainz/images/id/timestamp), in dem das Originalbild, das Detektionsergebnis und die erstellte Observation gespeichert sind
  - Die Speicherung kann zukünftig gelöscht oder auskommentiert werden, um Speicherplatz zu sparen, hier nur zur Demonstration
- Bei Wiederbereitstellung der Bilder direkter Post auf FROST-Server, hier auskommentiert

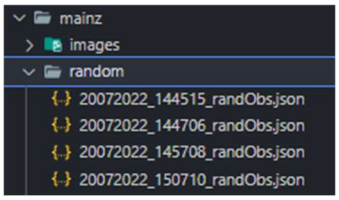


➔ Da die Bilder aktuell nicht verfügbar sind, kann keine sinnvolle Auswertung gemacht werden, zur Demonstration des Workflow wurde trotzdem das Bild angefragt, gespeichert, eine Objektdetektion gemacht und eine Observation für die entsprechende



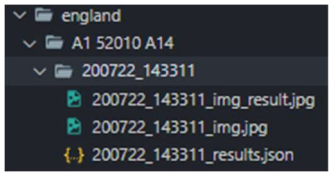
II. Scripts/create_rand_obs.py:

- Abfrage aller Datastreams und Schleife über diese
- Erzeugen von Observations für jeden Datastream mit random Zählwert
- Speicherung der JSON-Datei im Ordner mainz/random mit dem aktuellen Timestamp als Dateinamen
  - Die Speicherung kann zukünftig gelöscht oder auskommentiert werden, um Speicherplatz zu sparen, hier nur zur Demonstration
- Direkter Post auf FROS-Server
- Hier mit time.sleep(600) Befehl in While Schleife, um ein dauerhaftes Ausführen alle 10 Minuten zur Demonstration zu gewährleisten
  ➔ Alternative zu nicht funktionierendem Dockercontainer oder Scheduled Tasks in Windows



III. Scripts/vehicle_req_eng.py:

- Selektion von den Kameras A1 52010 A14, A1 52022 A141, A14 50685 J21 von https://uktraffic.live/england/
  - URLs der Bilder in Array eintragen
- Schleife über Kameras
- Abfrage des aktuellen Bilds über URL
- Objektdetektion im Bild
- Gleichzeitig werden Ordner mit den Namen den „IDs“ der Kameras im Ordner england angelegt
- Während dem Ablauf wird ein Ordner mit dem aktuellen Timestamp angelegt (england/name/timestamp), in dem das Originalbild, das Detektionsergebnis und die erstellte Observation gespeichert sind
  - Die Speicherung kann zukünftig gelöscht oder auskommentiert werden, um Speicherplatz zu sparen, hier nur zur Demonstration

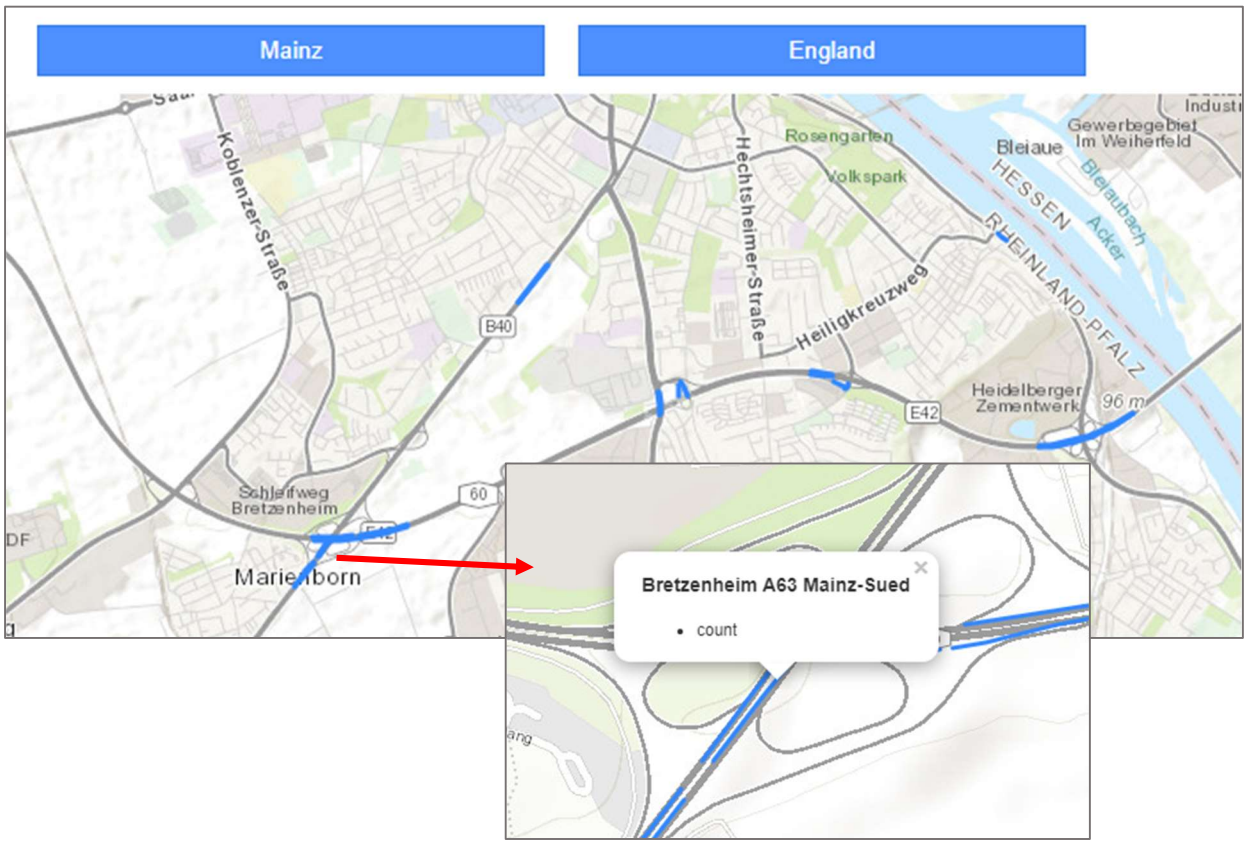


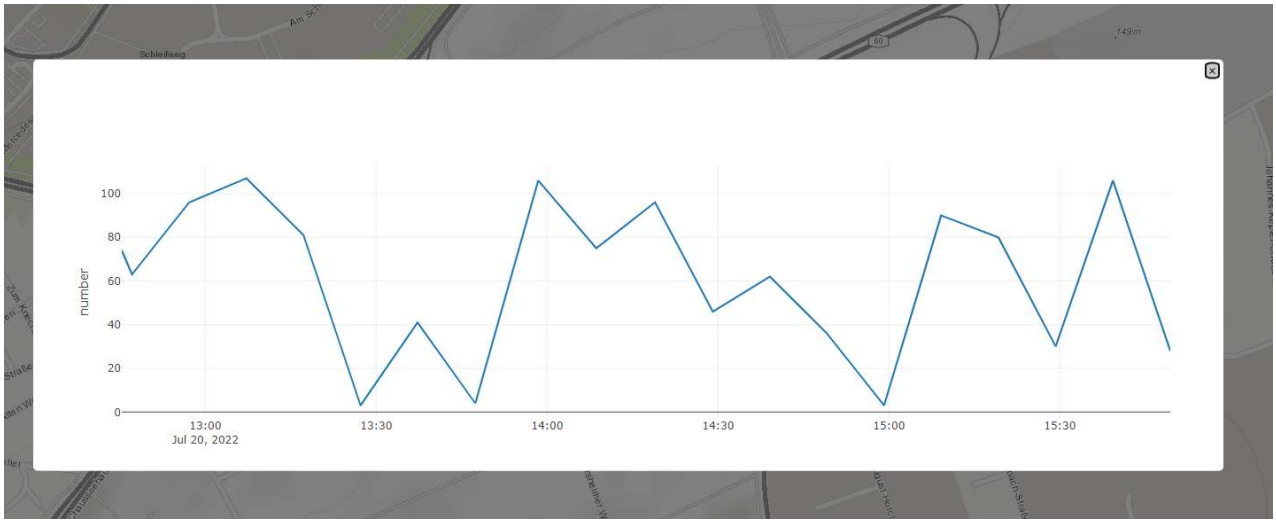
➔ Die Auswertung von Kamerabilder in England dient der Demonstration an echten Bildern, bei denen die Detektion funktioniert und sinnvolle Ergebnisse liefert

- Hier mit time.sleep(36000) Befehl in While Schleife, um ein dauerhaftes Ausführen alle 3 Stunden zur Demonstration zu gewährleisten
  ➔ Alternative zu nicht funktionierendem Dockercontainer oder Scheduled Tasks in Windows

Visualisierung

1. STAM Anwendung von GitHub herunterladen: https://github.com/DataCoveEU/STAM
2. Im Ordner example wird nur die leaflet.html Datei benötigt → example hier in webpage umbenannt & Lizenzdateien in Ordner licence extrahiert
3. Hintergrundkarte einfügen
4. baseUrl und die Zentrumskoordinaten der Karte anpassen
5. 2 Buttons einfügen, um leichter zwischen Mainz und England zu wechseln → mit Zoomfunktion auf Ort





Testbilder & Experimente

- **Beispielbilder**: Mit dem Skript scripts/vehicle_counter_testpics.py wird eine Objektdetektion von den Bildern im Ordner test/images vorgenommen und die Ergebnisse im Ordner test/results gespeichert
    - Die Bildquellen der Bilder sind in der Datei Bildquellen.txt gespeichert
    - Zudem wird von den Beispielbildern eine Observations-Datei angelegt, die aus jedem Bild die Zählung im JSON-Format speichert
- **Segmentation**:
    1. Im Ordner test_cv kann mit dem Skript areaofinterest_analysis.py eine manuelle Selektion von Bildbereichen schrittweise vorgenommen werden. Am Beispiel der Bilder im Ordner test_cv/images wird manuell jeweils der Bereich der dargestellten Fahrspuren selektiert
    2. und diese jeweils in einem eigenen Bild im Ordner test_cv/results gespeichert
    3. durch Ausführen des Skripts scripts/vehicle_counter_segmentationtest.py wird dann eine Objektdetektion über diese Bildausschnitte gemacht und die Ergebnisse in test_cv/results/objdetect gespeichert