Perception

statisch

Verwendete Sensoren

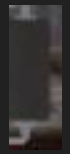
psaf 1

- **Kamera RGB**, 3200x1200, 120°, 13 fps
  bei x=2m, z=2m
- 3rd person Kamera 20 fps (nicht verw.)
- **Lidar segm.** bei z=3m
  -9°=>0°, 24x, 80k pts/s, 20 rot/s, 50m
- **Lidar segm.** bei z=3m
  -37°=>-9.5°, 16x, 40k pts/s, 20 rot/s, 50m
- kein Radar
- **Kamera segm**. 1600x600, 120°, 13 fps
- **Kamera depth** 1600x600, 120°, 13 fps
- keine DVS-Kamera
- **GPS** (GNSS) und **IMU** (Accelerometer)
- **Kollision** und **Lane Invasion Sensor**

psaf 2

- **Kamera RGB**, 800x600, 90°, 20 fps
  bei x=2m, z=2m
- 3rd person Kamera 20 fps (nicht verw.)
- **Lidar normal** bei z=0.5m
  -5°=>1°, 32x, 20k pts/s, 20 rot/s, 50m
- **Lidar segm.** bei z=2.4m (gleiches Topic?!)
  -26.8°=>2°, 32x, 320k pts/s, 20 rot/s, 50m
- **Radar** “vorne” 400x300, 90°, 20 fps
- **Kamera segm.** 600x110, 90°, 20 fps
- **Kamera depth** 400x300, 90°, 20 fps
- **Kamera DVS** 400x70, 90°, 20 fps (!?)
- **GPS** (GNSS) und **IMU** (Accelerometer)
- **Kollision** und **Lane Invasion Sensor**

Ampel- und Schildererkennung




Ausschneiden des Bildausschnitts in der Kamera m.H. Segmentation Image, zusätzliche Information zur Entfernung durch die depth-Kamera (Berechnung teuer)

psaf1:

- Ampel- und Schildererkennung mit YOLOv3 und pytorch
- Einteilung in folgende Klassen: (ausreichend für die neuen Karten?)
  Rückseite Schild, EU 30, EU 60, EU 90, amerik. 30, amerik. 40, amerik. 60, Stoppschild

psaf2:

- Ampelerkennung durch Bildmanipulation, erkennen der dominanten Farbe
- Schilder lesen durch Bibliothek tesseract OCR LSTM (EasyOCR + GPU ggf. besser)

Die Geschwindigkeit der beiden Methoden sollte verglichen werden, vor allem tesseract und yolo

psaf1.psaf_perception, psaf2.Perception.street_object_detector, Bilder aus den Training-Set von psaf1

psaf 2 - Evaluation mit Test-Daten

RED: correct: 97.05%, fps: 9.91, {'red': 263, 'yellow': 7, 'green': 1}

YELLOW: correct: 98.28%, fps: 72.9, {'yellow': 171, 'red': 3}

GREEN: correct: 96.08%, fps: 29.11, {'green': 343, 'back': 1, 'yellow': 12, 'red': 1}

BACK: correct: 36.08%, fps: 161.01, {'back': 57, 'green': 25, 'red': 61, 'yellow': 15}

mit Parametern von psaf2

Linienerkennung Stopplinien

Ausschneiden des Bildausschnitts in der Kamera m.H. Segmentation Image, zusätzliche Information zur Entfernung durch die depth-Kamera

psaf1:

- Stoplinienerkennung durch Bildmanipulation (cv2.Canny+cv2.HoughLinesP)
  Grenzwerte werden direkt der cv2 Funktion mitgegeben.
- Zudem können mit yolov3 STOP-Schriften auf der Straße erkannt werden.

psaf2:

- Stoplinienerkennung durch Bildmanipulation (drawContours+cv2.LINE_AA), Auslesen der Winkel.
  Die erkannten Linien müssen eine Mindestlänge und einen Maximalwinkel aufweisen.

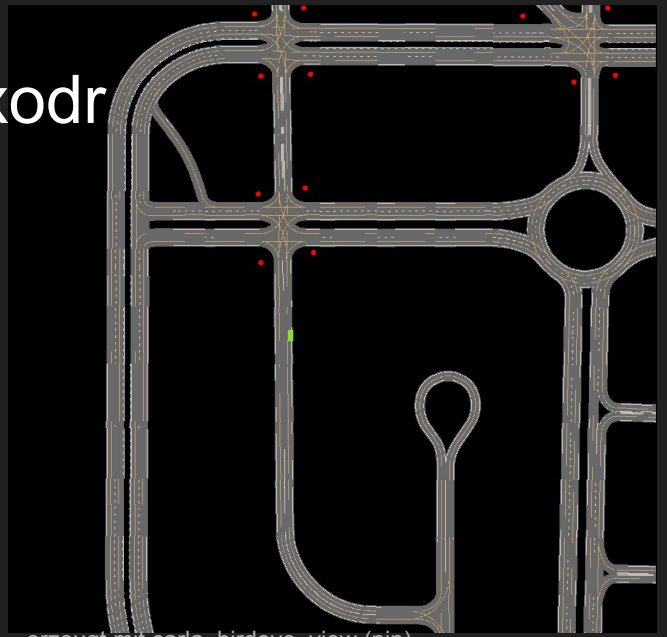
Die Linienerkennung von Stopplinien ist ähnlich.
Die Erkennung von Stop-Schriften kann eventuell vernachlässigt werden, da Informationen zu Schildern in der openDrive-/commonroad-Karte stehen

psaf1.psaf_perception, psaf2.Perception.line_detection

Karteninformationen aus carla bzw. xodr

Zum Beispiel (https://carla.readthedocs.io/en/latest/python_api/#carla.LandmarkType)

- Stoppschilder
- Geschwindigkeit pro Straßenabschnitt + Geschw. Einheit
- Ampel-Position und zugehörige Spur
- Vorfahrt-Gewähren Schilder
- Spur-Informationen



erzeugt mit carla_birdeye_view (pip)

XODR-Code

```
<road>
        <link></link>
        <type><speed max=”40” unit=”mph”/></type>
        < …. >
        <lanes>
                <center><lane><roadMark type=”broken” laneChange=”both”/></lane></center>
        </lanes>
        <objects></objects>
        <signals>
                <signal name=”Sign_Stop” type=”206” dynamic=”no”></signal>
        </signals>
</road>
```

Nützliche Hinweise zur Carla World

```
client = carla.Client('127.0.0.1', 2000)
client.get_world().get_map().get_topology()

is_vehicle = lambda actor: "vehicle" in actor.type_id
is_pedestrian = lambda actor: "walker" in actor.type_id
is_traffic_light = lambda actor: "traffic_light" in actor.type_id

carla.TrafficLightState.Green (enum für Ampelphasen)

all_actors = carla_birdeye_view.actors.query_all(world=self._world)
zum beispiel Actor x:  x.get_location() und x.state (carla.Actor)

agent_vehicle_loc = agent_vehicle.get_location()
```