

Drone object tracking with YOLOv4, OpenCV Tracking, Deep SORT and TensorFlow

Objektverfolgung ist mit YOLOv4, OpenCV-Tracker, Deep SORT und TensorFlow implementiert. YOLOv4 ist ein hochmoderner Algorithmus, der tiefe neuronale Faltungsnetzwerke zur Objekterkennung verwendet. Um die Ergebnisse von YOLOv4 zu verbessern, werden Algorithmen aus OpenCV für das Tracking verwendet. Wir können die Ergebnisse von YOLOv4 und OpenCV Tracking in Deep SORT (Simple Online and Realtime Tracking with a Deep Association Metric) einspeisen, um einen hochpräzisen Objekt-Tracker zu erstellen.

Wir können die Bildkoordinaten in Weltkoordinaten umwandeln. Damit ist es möglich den Standort von Objekten und deren IDs auf der Karte darzustellen.

Anhand des Standorts der Objekte und der FPS des Videos kann die Geschwindigkeit jedes Objekts (Fahrzeugs) ermittelt werden.

Object detection

```
$ python object_detector.py
```

YOLOv4 detektiert Objekte (Verkehrsmittel) mit bbox, die von einer Drohne aufgenommen wurden. Letztendlich werden diese Daten als json. Datei abgespeichert. Darüber hinaus benutzen wir Deep SORT um Ids zu Objekten zuzuordnen.

Beschreibung

Wir laden nötige Module (Zeile 1 - 27)

```
object_detector.py
1 import os
2 # comment out below line to enable tensorflow logging outputs
3 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
4 import time
5 import json
6 import tensorflow as tf
7 physical_devices = tf.config.experimental.list_physical_devices('GPU')
8 if len(physical_devices) > 0:
9     tf.config.experimental.set_memory_growth(physical_devices[0], True)
10 from absl import app, flags, logging
11 from absl.flags import FLAGS
12 import core.utils as utils
13 from core.utils import *
14 from core.yolov4 import filter_boxes
15 from tensorflow.python.saved_model import tag_constants
16 from core.config import cfg
17 from PIL import Image
18 import cv2
19 import numpy as np
20 import matplotlib.pyplot as plt
21 from tensorflow.compat.v1 import ConfigProto
22 from tensorflow.compat.v1 import InteractiveSession
23 # deep sort imports
24 from deep_sort import preprocessing, nn_matching
25 from deep_sort.detection import Detection
26 from deep_sort.tracker import Tracker
27 from tools import generate_detections as gdet
28
```

Definieren Parameter für YOLO und Deep SORT (Z. 46 - 68)

```
46     # Definition of the parameters
47     max_cosine_distance = 0.4
48     nn_budget = None
49     nms_max_overlap = 1.0
50
51     # Initialize deep sort
52     model_filename = 'model_data/mars-small128.pb'
53     encoder = gdet.create_box_encoder(model_filename, batch_size=1)
54     # Calculate cosine distance metric
55     metric = nn_matching.NearestNeighborDistanceMetric("cosine", max_cosine_distance, nn_budget)
56     # Initialize tracker
57     tracker = Tracker(metric)
58
59     # Load configuration for object detector
60     config = ConfigProto()
61     config.gpu_options.allow_growth = True
62     session = InteractiveSession(config=config)
63     # STRIDES, ANCHORS, NUM_CLASS, XYSCALE = utils.load_config(FLAGS)
64     ANCHORS = get_anchors(cfg.YOLO.ANCHORS, tiny)
65     XYSCALE = cfg.YOLO.XYSCALE if model == 'yolov4' else [1, 1, 1]
66     NUM_CLASS = len(read_class_names(cfg.YOLO.CLASSES))
67     STRIDES = np.array(cfg.YOLO.STRIDES)
68     input_size = size
69
```

Laden das DL-Model für object detection (Z. 70 - 81)

```
70     # Load tflite model if flag is set
71     if framework == 'tflite':
72         interpreter = tf.lite.Interpreter(model_path=weights)
73         interpreter.allocate_tensors()
74         input_details = interpreter.get_input_details()
75         output_details = interpreter.get_output_details()
76         print(input_details)
77         print(output_details)
78     # Otherwise load standard tensorflow saved model
79     else:
80         saved_model_loaded = tf.saved_model.load(weights, tags=[tag_constants.SERVING])
81         infer = saved_model_loaded.signatures['serving_default']
82
```

Öffnen das Video (Z. 83 - 87) und lesen jeden Videoframe mit while – Schleife (Z. 107)

```
83     # Begin video capture
84     try:
85         vid = cv2.VideoCapture(int(input_video_path))
86     except:
87         vid = cv2.VideoCapture(input_video_path)
88
```

YOLO wird zum object detection eingesetzt. Als output bekommen wir bboxes, classes, scores, die als eine json. Datei in *outputs/* gespeichert werden (Z. 109 - 271).

```

109      # While video is running
110     while True:
111         return_value, frame = vid.read()
112         if return_value:
113             frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
114             image = Image.fromarray(frame)
115         else:
116             print('Video has ended or failed, try a different video format!')
117             break
118         frame_num +=1
119         print('Frame #: ', frame_num)
120         frame_size = frame.shape[:2]
121         image_data = cv2.resize(frame, (input_size, input_size))
122         image_data = image_data / 255.
123         image_data = image_data[np.newaxis, ...].astype(np.float32)
124         start_time = time.time()
125
126         # Run detections on tflite if flag is set
127         if framework == 'tflite':
128             interpreter.set_tensor(input_details[0]['index'], image_data)
129             interpreter.invoke()
130             pred = [interpreter.get_tensor(output_details[i]['index']) for i in range(len(output_
131             # Run detections using yolov3 if flag is set
132             if model == 'yolov3' and tiny == True:
133                 boxes, pred_conf = filter_boxes(pred[1], pred[0], score_threshold=0.25,
134                                         input_shape=tf.constant([input_size, input_size]))
135             else:
136                 boxes, pred_conf = filter_boxes(pred[0], pred[1], score_threshold=0.25,
137                                         input_shape=tf.constant([input_size, input_size]))
138         else:
139             batch_data = tf.constant(image_data)
140             pred_bbox = infer(batch_data)
141             for key, value in pred_bbox.items():
142                 boxes = value[:, :, 0:4]
143                 pred_conf = value[:, :, 4:]
144
145             boxes, scores, classes, valid_detections = tf.image.combined_non_max_suppression(
146                 boxes=tf.reshape(boxes, (tf.shape(boxes)[0], -1, 1, 4)),
147                 scores=tf.reshape(
148                     pred_conf, (tf.shape(pred_conf)[0], -1, tf.shape(pred_conf)[-1])),
149                 max_output_size_per_class=200,
150                 max_total_size=200,
151                 iou_threshold=iou,
152                 score_threshold=score
153             )

```

Problem

Nicht in jedem Frame erkennt YOLO die Objekte. D.h. die bbox fehlen bei einigen Videoframes für einige Objekte.

Aus diesem Grund tracket (verfolgt) Deep SORT einige Objekte falsch (IDs falsch zuordnet).



Lösung

Weiter werden klassische computer vision Ansätze eingesetzt, wie Tracking von OpenCV, um Objekte mit bbox zu erkennen und zu vervollständigen, wo YOLO nicht erkannt hat.

Object tracking mit OpenCV

```
$ python cv_preprocessor.py
```

In diesem Skript werden klassische computer vision Ansätze eingesetzt, wie Tracking von OpenCV, um Objekte mit bbox zu erkennen und zu vervollständigen, wo YOLO nicht erkannt hat. Im Vergleich zu *branch y_CV_deepsort* funktioniert das Programm viel schneller, sagt aber viel mehr falsche bbox vorher.

Beschreibung

Das Programm liest die Daten von YOLO (Z. 16) ein.

```
cv_preprocessor.py
1  import cv2
2  import time
3  import math
4
5  from cv_tracker.utils import *
6
7
8
9  input_video_path = r"inputs\input.mp4"
10 input_json_path=r"outputs\yolo_detection.json"
11 output_json_path=r"outputs\cvTracker_summary.json"
12
13 edge_score = 0.15
14
15
16 data_dict = data_reader(input_json_path)
17 new_data_dict = data_dict.copy()
18
```

Wir initialisieren einen Multi Tracker von OpenCV (Z. 26).

```
24
25 # Generate a MultiTracker object
26 multi_tracker = cv2.legacy.MultiTracker_create()
27
28 false_cvTrackon = s1
```

Bearbeiten jeden Videoframe mit while-Schleife (Z. 36)

```
35
36 ˜ while True:
37      ret, frame = cap.read()
38      output_frame = frame.copy()
39 ˜ if not ret:
40      | break
41
```

Wir benutzen die vorher bekommenen Daten von YOLO (Z. 43).

Wenn die Nummer der aktuellen Frame gleich 1 ist, benutzen wir bbox von YOLO für den Tracker von OpenCV. Wir erstellen MultiTracker. Dann fangen wir an, mit dem Tracker von OpenCV die Objekte (Z. 46 – 71) zu verfolgen.

Visualisieren wir bbox von Trackern (Z. 71). Score benutzen wir von YOLO.

```
36 ˜ while True:
37      ret, frame = cap.read()
38      output_frame = frame.copy()
39 ˜ if not ret:
40      | break
41
42      # Get Yolo data from current frame
43      countYOLO, classesYOLO, scoresYOLO, bboxesYOLO, center_pointsYOLO, new_data_dict = get_YOLObbox()
44
45      start_time = time.time()
46      if count == 1:
47          for bbox in bboxesYOLO:
48              bbox = (int(bbox[0]), int(bbox[1]), int(bbox[2]), int(bbox[3]))
49
50              tracker = cv2.legacy.TrackerCSRT_create()
51              multi_tracker.add(tracker, frame, bbox)
52
53      if ret:
54          # Update the location of the bounding boxes
55          success, bboxesTRACKER = multi_tracker.update(frame)
56
57          # If a tracker is not successful, initialize new tracker on the corresponding image
58          if success:
59              pass
60          else:
61              trackers_arr = np.array(multi_tracker.getObjects())
62              idx = np.where(trackers_arr.sum(axis=1)!=0)[0]
63              trackers_arr=trackers_arr[idx]
64
65              multi_tracker = cv2.legacy.MultiTracker_create()
66              for i in trackers_arr:
67                  tracker = cv2.legacy.TrackerCSRT_create()
68                  multi_tracker.add(tracker, frame, tuple(i))
```

Wenn die Nummer der aktuellen Frame NICHT gleich 1 ist, müssen wir prüfen, ob neue bbox von YOLO beim aktuellen Frame aufgetreten sind, die noch nicht in

OpenCV Tracker sind. Wenn bbox von YOLO neu ist, wird ein neuer Tracker für dieses Objekt erstellt.

Dafür benutzen wir die Mittelpunkte von YOLO-bbox für einen aktuellen Frame (Z. 77).

Berechne die Entfernung von der ausgewählten YOLO-bbox zu jeder Tracker bbox (Z. 80).

Wenn die Distanz ≥ 15 ist, nehmen wir an, dass die aktuelle YOLO-bbox neu ist und nicht in OpenCV Tracker ist. Dann initialisieren wir einen neuen Tracker und fügen wir diese Koordinaten von bbox in den OpenCV-Tracker hinzu (Z. 86 - 92).

OpenCV-Multitracker sagt die bbox für den nächsten Frame basierend auf den vorherigen bbox vorher.

```
cv_preprocessor.py
72     else:
73         # If the object that has a yolo bbox does not have a tracker bbox that belongs to the same object,
74         # then the yolo bbox is new and create a tracker for it.
75
76         # Check the distance from each yolo bbox to each tracker bbox
77         for yolo_count, pointsYOLO in enumerate(center_pointsYOLO):
78
79             # Calculate the distance from the selected yolo bbox to each tracker bbox
80             dist_to_trackers = np.array([math.dist([pointsYOLO[0], pointsYOLO[1]], [pointsTRACKER[0], pointsTRACKER[1]])) for pc
81
82             # If at least one distance is less than 15, then skip,
83             if any(dist_to_trackers<15):
84                 pass
85             else:
86                 # A new YOLO bbox that appeared
87                 current_bboxesYOLO = bboxesYOLO[yolo_count]
88                 bbox = (current_bboxesYOLO[0], current_bboxesYOLO[1], current_bboxesYOLO[2], current_bboxesYOLO[3])
89
90                 tracker = cv2.legacy.TrackerCSRT_create()
91                 multi_tracker.add(tracker, frame, bbox)
92
93             if ret:
94                 # Update the location of the bounding boxes
95                 success, bboxesTRACKER = multi_tracker.update(frame)
96
97                 if success:
98                     pass
99                 else:
100                     # https://github.com/opencv/opencv\_contrib/issues/2377
101                     prev_frame_no = int(count - 2)
102
103                     cap.set(1,prev_frame_no);
104                     prev_ret, prev_frame = cap.read() # Read the frame
105
106                     trackers_arr = np.array(new_data_dict["summary"][str(count-1)]["bboxes"])
107
108                     multi_tracker = cv2.legacy.MultiTracker_create()
109                     for i in trackers_arr:
110                         tracker = cv2.legacy.TrackerCSRT_create()
111                         multi_tracker.add(tracker, prev_frame, tuple(i))
112
113
```

Wir prüfen `multi_tracker.update(frame)` und wenn das Ergebnis `False` ist, initialisieren wir einen neuen `MultiTracker` für alle bbox (Z. 100 - 115)

Wenn wir die bbox von YOLO visualisieren möchten, müssen wir `yolo_data = True` machen (Z. 126).

Wenn wir die bbox von OpenCV visualisieren möchten, müssen wir `cv2_tracker = True` machen (Z. 129).

```
120     print("--- %s seconds ---" % (time.time() - start_time))

121
122
123     ##### Data visualization #####
124
125     #yolo_data = True      # Activate to show predict of yolo
126     yolo_data = False
127
128     #cv2_tracker = True    # Activate to show predict of cv tracker
129     cv2_tracker = False
130
131     summary = True         # Activate to show summary predict of yolo and cv tracker ### BEST RESULT
132     #summary = False
133
134     show_data( new_data_dict, output_frame, count, yolo_data=yolo_data, cv2_tracker=cv2_tracker, summary=summary )
135
136
137     # # Activate to visualize the path of vehicles
```

Problem

Wenn der Tracker das Objekt „verloren“ hat, verschwindet der Tracker nicht und gibt die falschen Koordinaten von bbox, obwohl das Objekt nicht da ist. Im Vergleich zu *branch y_CV_deepsort* zeigt das Programm viel mehr falsche bbox, ist aber schneller.

Object tracking mit Deep SORT

```
$ python deepsort_tracker.py
```

In diesem Skript benutzen wir Deep SORT, um die Objekte zu verfolgen und zu jedem Objekt den entsprechenden ID zuzuordnen.

Transformation of image coordinates to world coordinates

```
$ python coordinate_transform.py
```

Für die Objektnalyse ist wichtig, wo und bei welchem Zeitpunkt sich die Objekte befinden. Um genaue Weltkoordinaten (Koordinaten des Breitengrads und des Längengrads) von Objekten zu bestimmen, müssen wir wissen, wo dieser Standort sich bei Aufnahme war. Dann können wir die Koordinate auf dem Frame in die Weltkoordinate umwandeln.

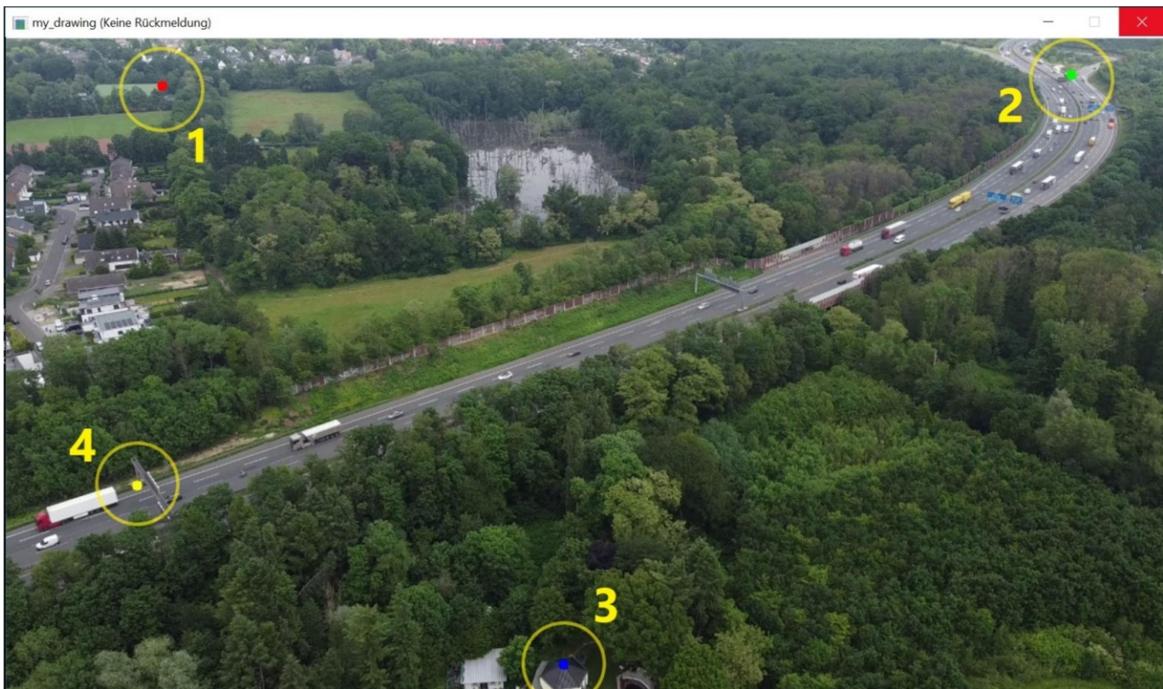
Beschreibung

Wir laden nötige Module. Definieren die Paths, wo das input Video, die json-Tracking-Daten. Am Ende wird die json-Datei mit den umgewandelten Koordinaten für jeden Frame in *outputs/* gespeichert (Z. 1 - 12).

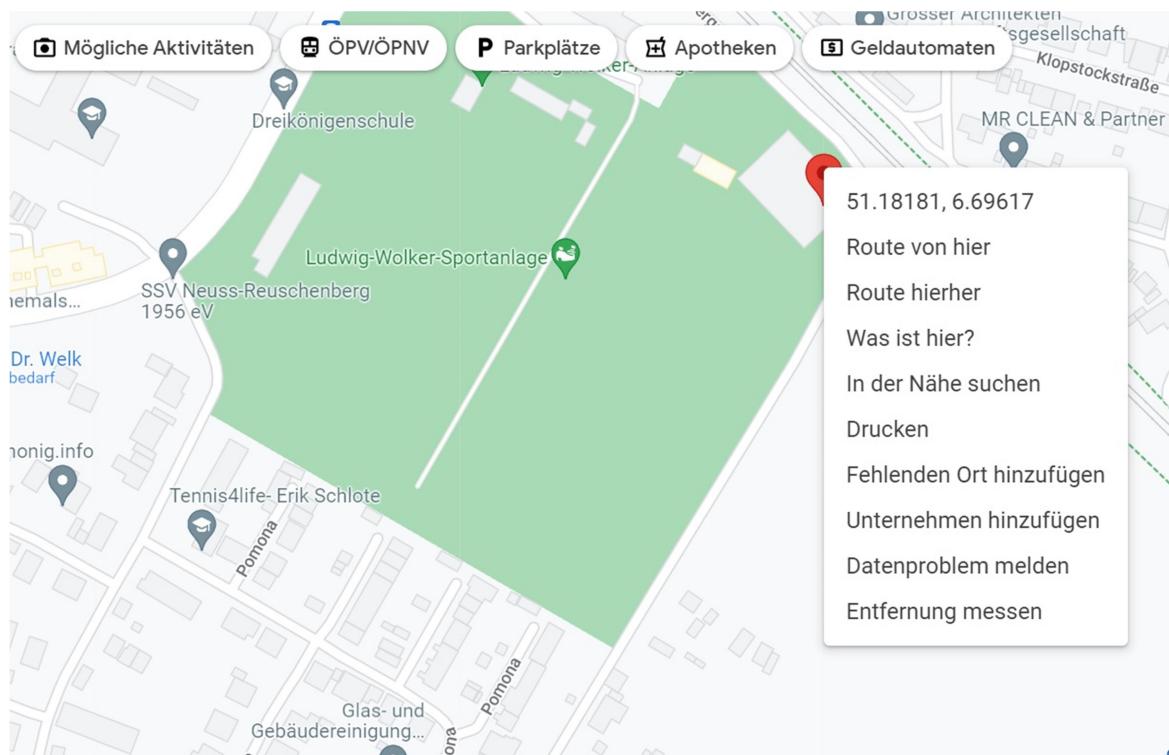
```
coordinate_transform.py
1 import cv2
2 import time
3 from core.utils import *
4
5
6 video_path = r'inputs\input.mp4'
7
8 input_json_path=r'outputs\tracking.json'
9 output_json_path=r'outputs\coordinate_transforming.json'
10
11 # Select a third frame
12 frame_no = 3
13
```

Das Video wird geöffnet und der dritte Frame wird uns gezeigt. Beim Bedarf kann der Nummer vom Frame *frame_no* geändert werden. Wir müssen vier Orte (rot, grün, blau, gelb) auf dem Frame markieren.

Für das Demo Video markieren wir entsprechende Orte zuerst auf dem Bild (Reihenfolge einhalten).



Die Weltkoordinaten (Koordinaten des Breitengrads und des Längengrads) können wir aus google maps entnehmen.



Wir müssen die entsprechenden Weltkoordinaten (Koordinaten des Breitengrads und des Längengrads) von diesen vier Punkten aus dem Bild in der Kommandozeile eingeben.

```
$ 51.181623, 6.696171  
$ 51.176709, 6.703237  
$ 51.175804213549, 6.692105166693712  
$ 51.17688327019584, 6.691930451309815
```

```
$ python coordinate_transform.py  
2022-12-04 13:44:06.745624: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library  
2022-12-04 13:44:06.746932: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do n  
To transform the coordinates in the image into geographic coordinates you need to mark only four points in the image.  
Then in a special field you need to enter GIS coordinates to them  
Mark four points in the image ...  
  
Mark the first point(RED) in the image ...  
x (px), y (px): 258 78  
  
Mark the second point(GREEN) in the image ...  
x (px), y (px): 1765 60  
  
Mark the third point(BLUE) in the image ...  
x (px), y (px): 923 1036  
  
Mark the fourth point(YELLOW) in the image ...  
x (px), y (px): 216 740  
  
Write GIS coordinates to the first point,  
for example: 48.40228307637909, 9.795953379174994  
1. point(RED): 51.181623, 6.696171  
  
Write GIS coordinates to the second point,  
for example: 48.40228307637909, 9.795953379174994  
2. point(GREEN): 51.176709, 6.703237  
  
Write GIS coordinates to the third point,  
for example: 48.40228307637909, 9.795953379174994  
3. point(BLUE): 51.175804213549, 6.692105166693712  
  
Write GIS coordinates to the fourth point,  
for example: 48.40228307637909, 9.795953379174994  
4. point(YELLOW): 51.17688327019584, 6.691930451309815|
```

Am Ende wird die json-Datei mit den umgewandelten Koordinaten für jeden Frame in *outputs/* gespeichert.

Visualizing objects on the map

```
$ python map_plot.py
```

Mit diesem Skript visualisieren wir die Objekte auf der Karte.

Bei der Visualisierung können wir sehen, bei welchem Frame, welches Objekt (Id) und wo genau war.



Beschreibung

Die Module werden geladen. Wir benutzen outputs/coordinate_transforming.json als Input Datei (Z. 1 - 10).

```
map_plot.py
1 import pandas as pd
2 import plotly.express as px
3 from core.utils import *
4
5
6 # https://plotly.com/python/reference/scattergeo/
7 # https://plotly.com/python/mapbox-layers/
8
9
10 input_json_path=r'outputs\coordinate_transforming.json'
11
```

Benutzen plotly für Datenvisualisierung

```
12 | 
13 < if __name__ == '__main__':
14     class_list = []
15     frame_list = []
16     id_list    = []
17     lat_list   = []
18     lon_list   = []
19
20     # Read the data from coordinate transformation
21     data_dict = data_reader(input_json_path)
22
23     frame_data = data_dict.keys()
24
25 < for count, frame_count in enumerate(frame_data):
26
27         obj_class = data_dict[frame_count]['classes']
28         obj_id    = [ 'id_'+str(current_id) for current_id in data_dict[frame_count]['i
29         obj_lat   = [ coordinate[0] for coordinate in data_dict[frame_count]['world_coo
30         obj_lon   = [ coordinate[1] for coordinate in data_dict[frame_count]['world_coo
31         obj_frame = [ int(frame_count) for current_id in range(len(obj_id)) ]
32
33         class_list.extend(obj_class)
34         id_list.extend(obj_id)
35         lat_list.extend(obj_lat)
36         lon_list.extend(obj_lon)
37         frame_list.extend( obj_frame )
38
39         customdata = [ custom_id  for custom_id in range(len(class_list)) ]
40
41 <     obj_data = pd.DataFrame({
42         'lat'      : lat_list,
43         'lon'      : lon_list,
44         'customdata': customdata,
45         'id'       : id_list
46     })
47
48 <     fig = px.scatter_mapbox(obj_data, lat='lat', lon='lon', custom_data=['customdata'],
49                               color='id', zoom=3)
50
51     fig.update_layout(mapbox_style='open-street-map')
52     fig.update_layout(margin={'r':0,'t':0,'l':0,'b':0})
53     fig.show()
```

Problem

Aus dem Bild sehen wir, dass der Trajektorienweg von Objekten leicht verschoben ist.

Lösung

Bei der Koordinatentransformierung kann man andere vier Punkte wählen.

Idealerweise müssen die Orte gewählt werden, die nah zu Objekten liegen. Beim demo Video können vier Punkte gewählt werden, die entlang der Straße sind.

Object velocity statistics

```
$ python object_velocity.py
```

Da die Weltkoordinate von Objekten bekannt sind und das Video die Daten von FPS hat, können wir die Zeit zwischen zwei Frames ausrechnen und anschließend die Geschwindigkeit von Objekten finden.

Beschreibung

Wir laden die Module. Definieren input und output Dateien. Definieren *max_velocity* mit dem Wert 150 km/h. D.h. alle Geschwindigkeiten, die größer als 150 km/h werden, werden rot auf den Grafiken gekennzeichnet.

dpi_value bestimmt die Qualität der gespeicherten Bilder (Z. 1 - 13).

object_velocity.py

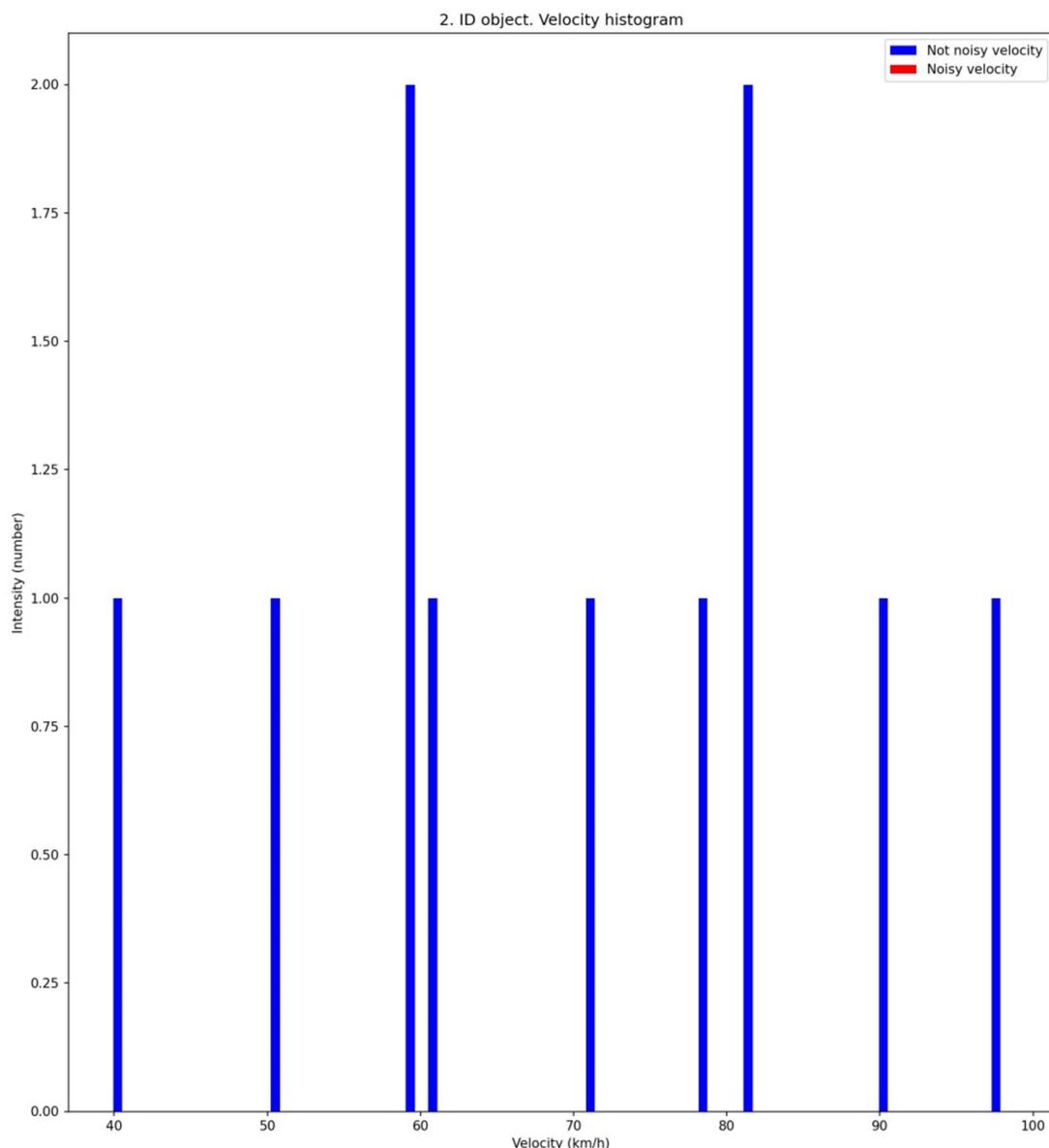
```
1  from core.utils import *
2  from math import sin, cos, sqrt, atan2, radians
3  import cv2
4  import matplotlib.pyplot as plt
5
6  input_video_path = r'inputs\input.mp4'
7  input_json_path = r'outputs\coordinate_transforming.json'
8  velocity_dir = 'outputs/velocity_statistic/'
9
10 # Maximum speed limit between noisy and not noisy value
11 max_velocity=150 # km/h
12
13 dpi_value = 150 # dpi
14
```

Mit `def coordinate2dist()` finden wir die Distanz in km zwischen zwei Punkten aus dem Bild.

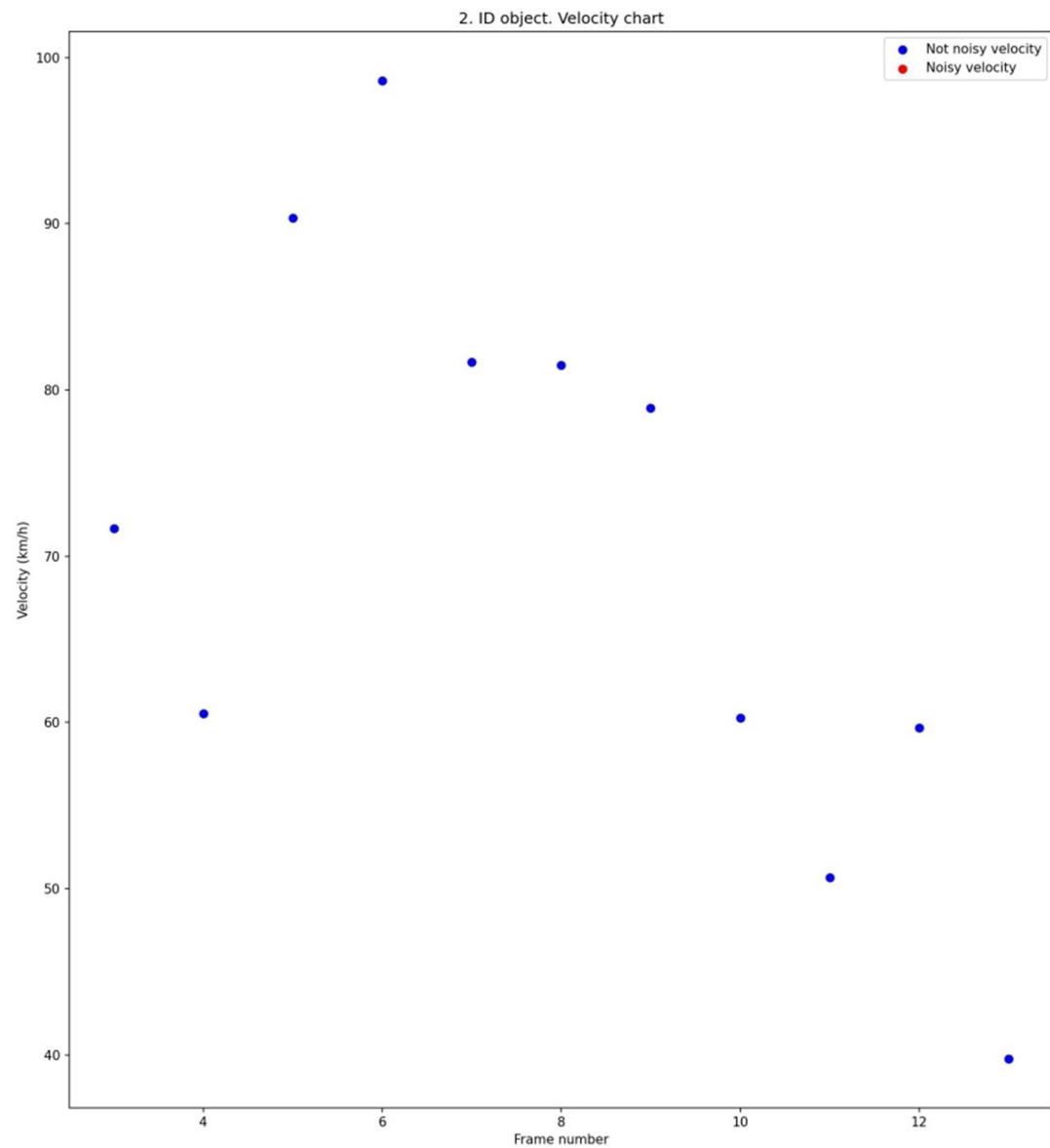
```
-- 
16 # Transform two points from world coordinates to distance in km
17 def coordinate2dist(start_pt, end_pt):
18     R = 6373.0
19     lat1 = radians(start_pt[0])
20     lon1 = radians(start_pt[1])
21     lat2 = radians(end_pt[0])
22     lon2 = radians(end_pt[1])
23
24     dlon = lon2 - lon1
25     dlat = lat2 - lat1
26
27     a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
28     c = 2 * atan2(sqrt(a), sqrt(1 - a))
29
30     distance = R * c  # km
31
32     return distance
33
```

Im Endeffekt werden eine Histogramm und eine Grafik für jedes Objekt gespeichert.

Aus dem Histogramm können wir ablesen, mit welcher Geschwindigkeit und wie lange das Objekt mit dieser Geschwindigkeit war.



Die Grafik zeigt, bei welchem Frame und welche Geschwindigkeit das Objekt hatte.



Problem

Aus den Bildern sehen wir, dass die Geschwindigkeiten unterschiedlich sind.

Lösung

Um das zu beheben, müssen wir die oben beschriebenen Probleme lösen