

Drone object tracking with YOLOv4, Deep SORT and TensorFlow

Objektverfolgung ist mit YOLOv4, Deep SORT und TensorFlow implementiert. YOLOv4 ist ein hochmoderner Algorithmus, der tiefe neuronale Faltungsnetzwerke zur Objekterkennung verwendet. Wir können die Ergebnisse von YOLOv4 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, tracking

```
$ python object_tracker.py
```

YOLOv4 detektiert Objekte (Verkehrsmittel), die von einer Drohne aufgenommen wurden. Dann wird Deep SORT eingesetzt, um die detektierten Objekte zu verfolgen und entsprechende IDs zuzuordnen. Letztendlich werden die entsprechenden Daten abgespeichert.

Beschreibung

Wir laden nötige Module (Zeile 1 - 27)

```
object_tracker.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
```

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
70
71
72
73
74
75
76
77
78
79
80
81
82
```

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 (Z. 123 - 216)

```
123     # Run detections on tflite if flag is set
124     if framework == 'tflite':
125         interpreter.set_tensor(input_details[0]['index'], image_data)
126         interpreter.invoke()
127         pred = [interpreter.get_tensor(output_details[i]['index']) for i in range(len(output_details))]
128     # Run detections using yolov3 if flag is set
129     if model == 'yolov3' and tiny == True:
130         boxes, pred_conf = filter_boxes(pred[1], pred[0], score_threshold=0.25,
131                                         input_shape=tf.constant([input_size, input_size]))
132     else:
133         boxes, pred_conf = filter_boxes(pred[0], pred[1], score_threshold=0.25,
134                                         input_shape=tf.constant([input_size, input_size]))
135     else:
136         batch_data = tf.constant(image_data)
137         pred_bbox = infer(batch_data)
138         for key, value in pred_bbox.items():
139             boxes = value[:, :, 0:4]
140             pred_conf = value[:, :, 4:]
141
142         boxes, scores, classes, valid_detections = tf.image.combined_non_max_suppression(
143             boxes=tf.reshape(boxes, (tf.shape(boxes)[0], -1, 1, 4)),
144             scores=tf.reshape(
145                 pred_conf, (tf.shape(pred_conf)[0], -1, tf.shape(pred_conf)[-1])),
146             max_output_size_per_class=200,
147             max_total_size=200,
148             iou_threshold=iou,
149             score_threshold=score
150     )
```

Weiter benutzen wir Deep SORT um Objekte zu verfolgen und Indexes zuzuordnen. Letztendlich speichern wir die Daten von Deep SORT als eine json. Datei (Z. 219 - 284).

```

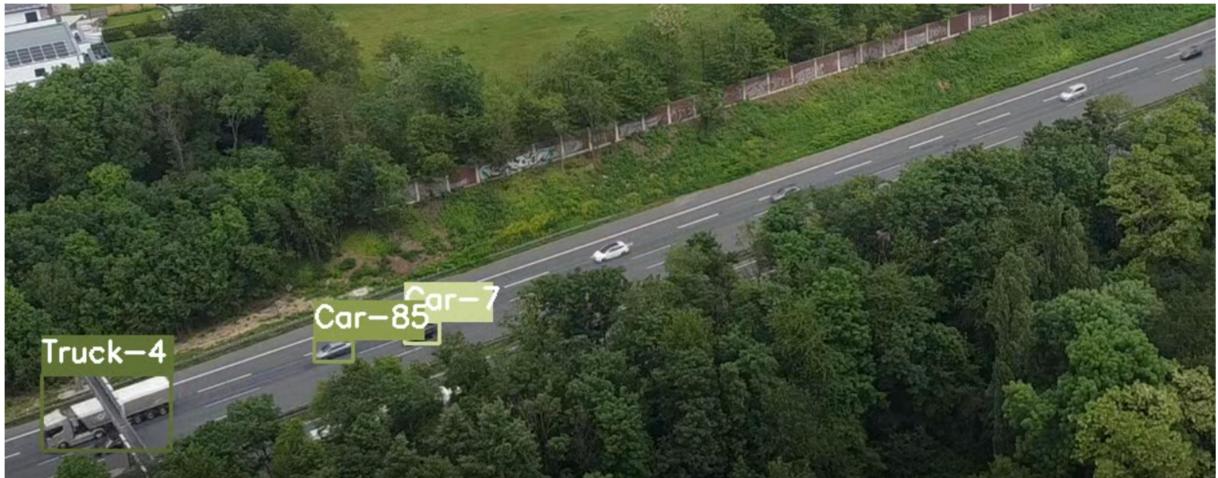
219  # Update tracks
220  for track in tracker.tracks:
221      if not track.is_confirmed() or track.time_since_update > 1:
222          continue
223      bbox = track.to_tlbr()
224      class_name = track.get_class()
225
226      # bbox coordinate
227      x1, y1, x2, y2, = int(bbox[0]), int(bbox[1]), int(bbox[2]), int(bbox[3])
228
229      if x1 < 0: x1 = 0
230      if y1 < 0: y1 = 0
231
232      if x2 > frame_size[1]: x2 = frame_size[1]
233      if y2 > frame_size[0]: y2 = frame_size[0]
234
235      w = int(x2 - x1)
236      h = int(y2 - y1)
237
238      obj_bbox = [x1,y1,w,h]
239
240      # Append obj data to json data
241      json_frame_data['bboxes'].append(obj_bbox)
242      json_frame_data['classes'].append(class_name)
243      json_frame_data['idx'].append(track.track_id)
244
245      # Draw bbox on screen
246      color = colors[int(track.track_id) % len(colors)]
247      color = [i * 255 for i in color]
248      cv2.rectangle(frame, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])), color, 2)
249      cv2.rectangle(frame, (int(bbox[0]), int(bbox[1]-30)), (int(bbox[0])+(len(class_name)+len(str(track.track_id)))*17, int
250      cv2.putText(frame, class_name + "-" + str(track.track_id),(int(bbox[0]), int(bbox[1]-10)),0, 0.75, (255,255,255),2)
251
252      # If enable info flag then print details about each track
253      if info:
254          print('Tracker ID: {}, Class: {}, BBox Coords (xmin, ymin, xmax, ymax): {}'.format(str(track.track_id), class_name,
255
256      # Calculate frames per second of running detections
257      fps = 1.0 / (time.time() - start_time)
258      print('FPS: %.2f' % fps)
259      result = np.asarray(frame)
260      result = cv2.cvtColor(result, cv2.COLOR_RGB2BGR)
261
262      output_data_dict[frame_num] = json_frame_data

```

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

1. Man kann das YOLO-Netz trainieren. Dafür muss ein neuer Dataset erstellt und die entsprechenden Bilder annotiert werden.

Das kann sehr Zeitaufwendig und teuer sein, bis gute Ergebnisse gekriegt sein werden.

Andererseits hat das ein großes Potenzial und funktioniert das.

2. Man kann klassische computer vision Ansätze einsetzen, wie z.B Tracking von OpenCV (schau andere Branches).

Als Nachteil ist es nicht bekannt, ob bessere Ergebnisse im Vergleich zu YOLO hingekriegt werden können.

Transformation of image coordinates to world coordinates

```
$ python coordinate_transform.py
```

Für die Objektanalyse 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

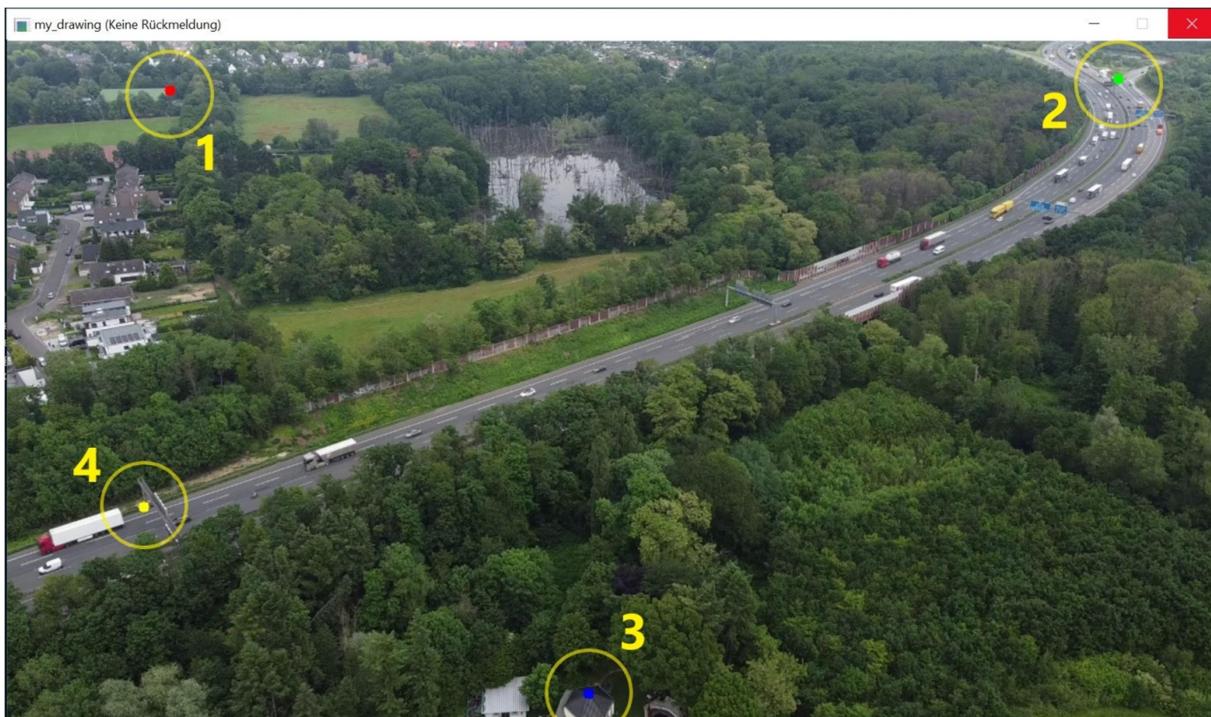
Wir laden nötige Module. Definieren die Paths, wo das input Video, die json-Tracking-Daten. Am Ende wird die json-Datei mit den umgewandelten Koordinaten für jeden Frame in *outputs/* gespeichert (Z. 1 - 12).

```
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
```

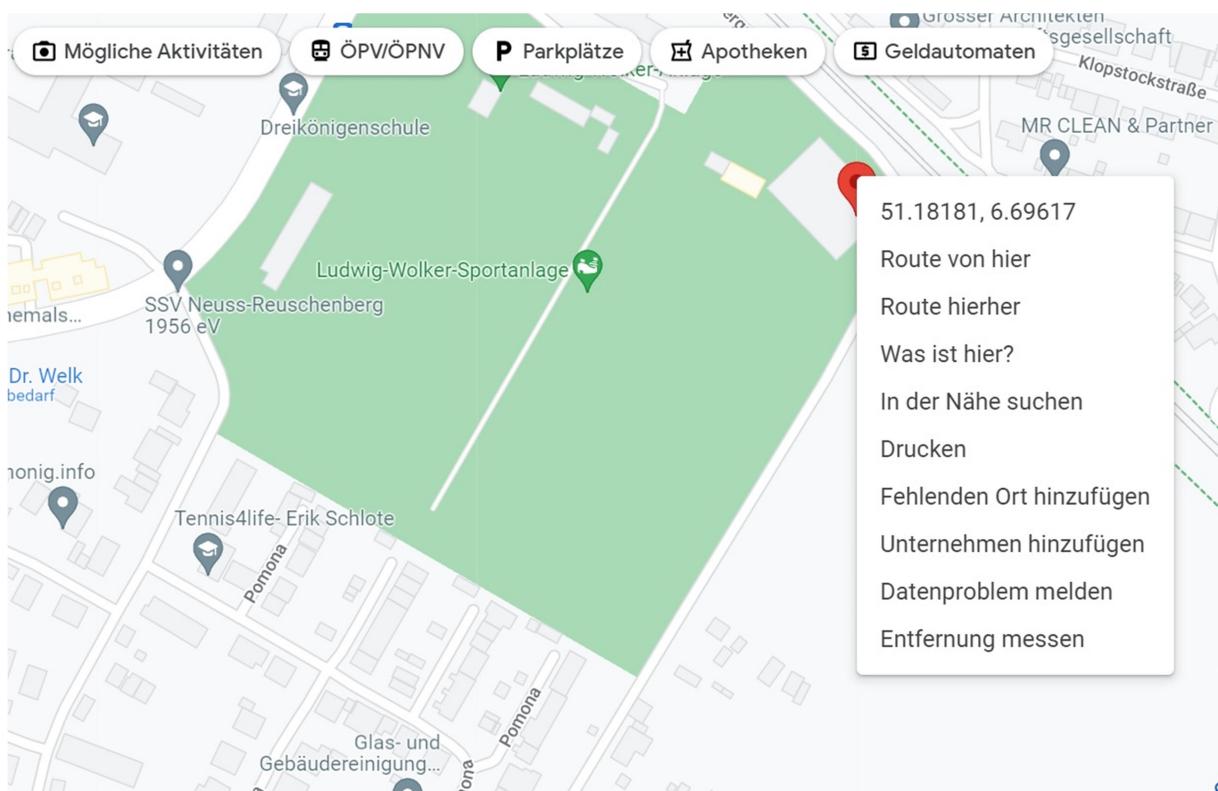
Das Video wird geöffnet und der dritte Frame wird uns gezeigt. Beim Bedarf kann der Nummer vom Frame *frame_no* geändert werden.

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 libcuda.so.1; dlerror: /usr/local/cuda/lib64/libcuda.so.1: version `CU_1_0' not found (required by tensorflow/stream_executor/platform/default/dso_loader.cc)  
2022-12-04 13:44:06.746932: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a CUDA GPU.  
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).

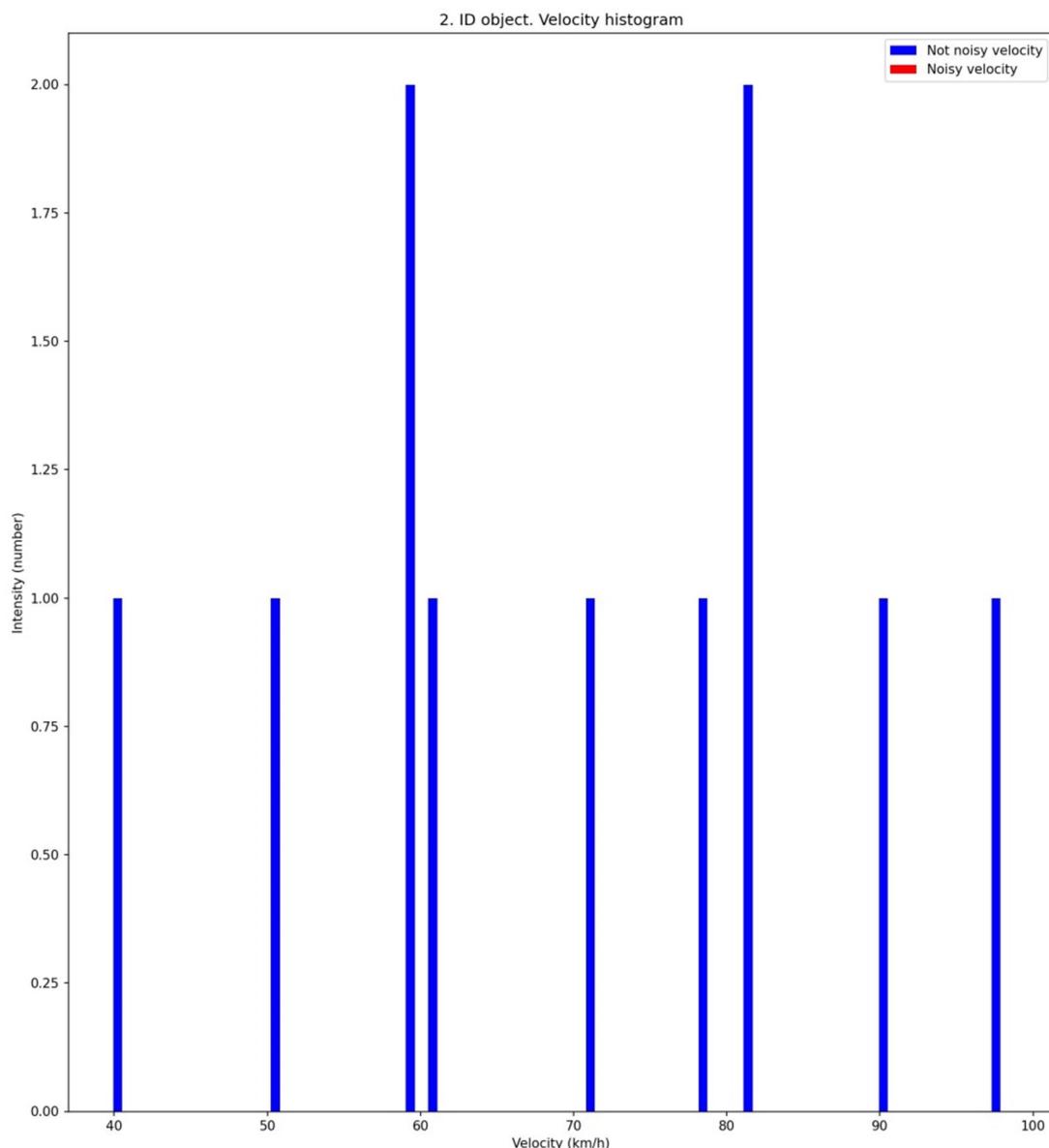
```
object_velocity.py
1  from core.utils import *
2  from math import sin, cos, sqrt, atan2, radians
3  import cv2
4  import matplotlib.pyplot as plt
5
6  input_video_path = r'inputs\input.mp4'
7  input_json_path = r'outputs\coordinate_transforming.json'
8  velocity_dir = 'outputs/velocity_statistic/'
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.

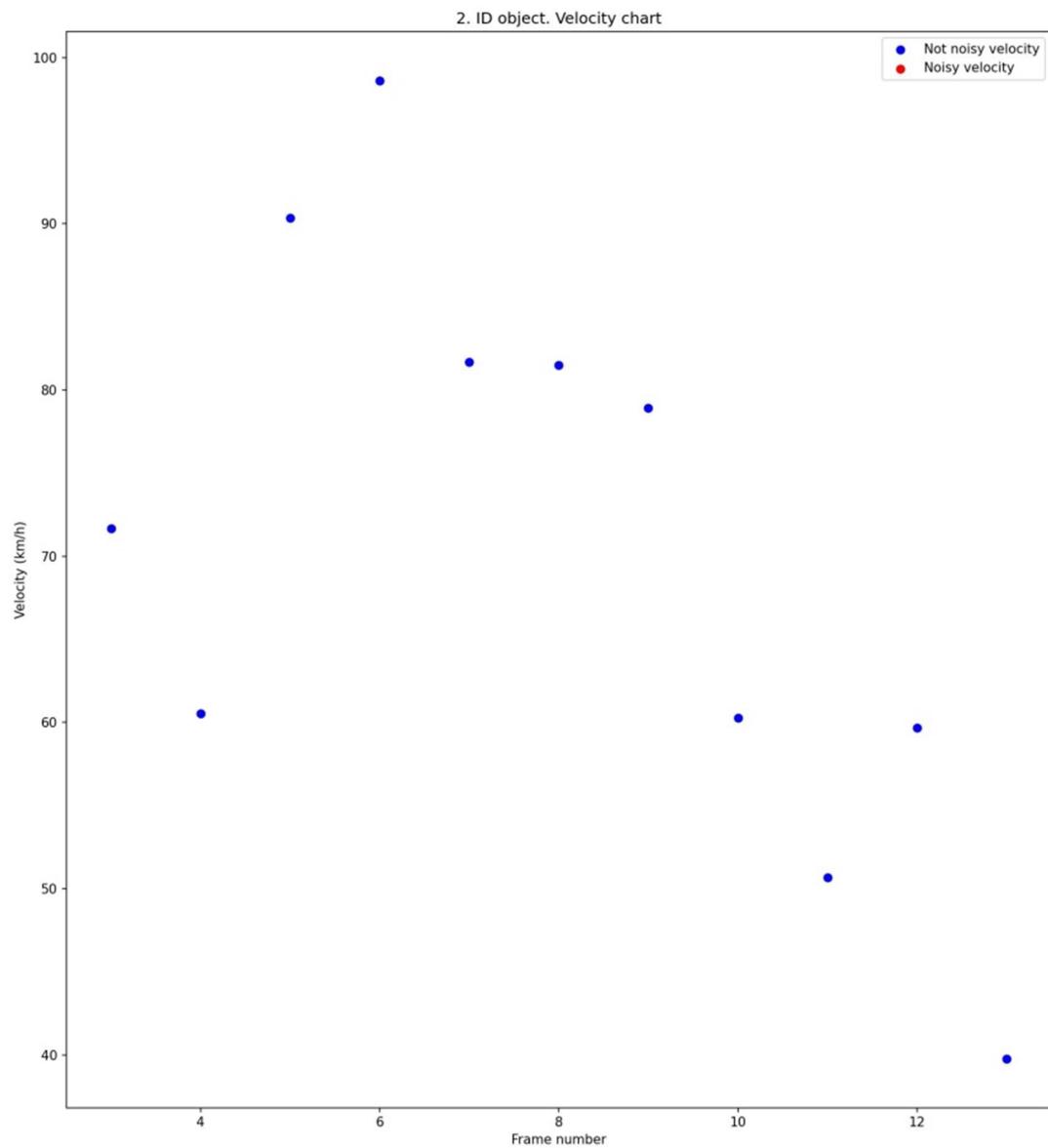
```
--  
16  # Transform two points from world coordinates to distance in km  
17  def coordinate2dist(start_pt, end_pt):  
18      R = 6373.0  
19      lat1 = radians(start_pt[0])  
20      lon1 = radians(start_pt[1])  
21      lat2 = radians(end_pt[0])  
22      lon2 = radians(end_pt[1])  
23  
24      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 Histogram 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