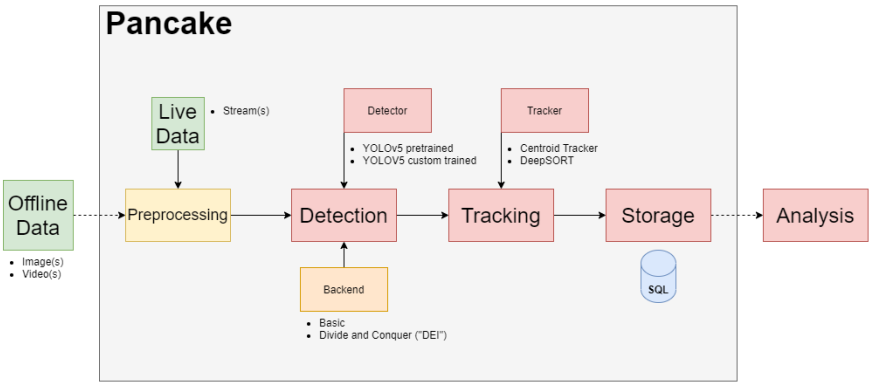
Pancake Modules overview



Data + Preprocessing

Pancake offers various different data sources, both live and offline, as well as multiple sources at once (e.g. multi-camera setups). The preprocessing necessary is handled "under the hood" (for details, see datasets.py), usually the user shouldn't have to delve into this too much though.

For details on how to specify different sources (+ Examples) see Configurations.

Backend

Because the Detection can - depending on the data - not necessarily be run directly, the Backend is responsible for adjusting the data as necessary to make sure the results are in order. All backends are initialized with an instance of a Detector, which is used for the detection.

Click here for more details

Backend	Details	Configuration `NAME:`
Simple	Details	`"simple"`
DEI	Details	`"dei"`

Adding a new Backend

Create your backend_foo.py within detector/backends/ .

Create a Backend class that inherits from the Base Backend.

Implement the detect method.

Add your Backend to the registry (i.e. add from .backend_foo import Foo).

Set your Backend in the configuration (under "BACKEND" -> NAME: "foo").

Important: When implementing your Backend, you need to stick to the Backend API!

Detection

The Detection itself is handled by an instance of a Detector. For pancake, we provide two versions of the YOLOv5 detector.

Click here for more details

Detection	Details	Configuration `NAME:`
YOLOv5 - Simple	Details	`"yolo_simple"`
YOLOv5 - Custom	Details	`"yolo_custom"`

Adding a new Detector

Create your detector_foo.py within detector/ .

Create a Detector class that inherits from the Base Detector.

Implement the detect method.

Add your Detector to the registry (i.e. add from .detector_foo import Foo).

Set your Detector in the configuration (under "DETECTOR" -> NAME: "foo").

Important: When implementing your Detector, you need to stick to the Detector API!

Tracking

Tracking is handled by an instance of a tracker. For pancake, we provide a Centroid Tracker as well as a DeepSORT Tracker, whereat the Trackers have been loosely taken over from previous project groups.

Click here for more details

Tracker	Details	Configuration `NAME:`
Centroid	Details	`"centroid"`
DeepSORT	Details	`"deepsort"`

Adding a new Tracker

Create your tracker_foo.py within tracker/ .

Create a Tracker class that inherits from the Base Tracker.

Implement the update method.

Add your Tracker to the registry (i.e. add from .tracker_foo import Foo).

Set your Tracker in the configuration (under "TRACKER" -> NAME: "foo").

Important: When implementing your Tracker, you need to stick to the Tracker API!

Storage

The collected data can optionally be stored in a SQLite database (this can enabled in the configuration).

Result Processing

If you are not only interested in the raw results data, but also in visualizations of detections or tracks, you can enable and configure this in the configuration.

Analysis

Pancake currently doesn't offer further analysis on the collected data. This is something that could be tackled in the future.

Details

Backends

Why is there a need for a backend in the first place? Can we not just feed the detector with the image(s) and be done with it?

When we first started to think about how to approach the detection, we thought that it would be sufficient to simply run the detection on the stitched panorama image. However it quickly became apparent that this was not going to work, as the detections were very poor. The problem is that the detector is not trained to detect cars that are very small relative to the whole image.

This is why we designed the *DEI* backend to make sure we can have decent detections.

This however, comes with its own set of problems, mainly 1. Performance overhead 2. Duplication problems

The duplication problems arise, as the individiual subframes (see below) are overlapping (they have to be, else some cars might not get detected at all), which can cause multiple detections for the same car. We already have deduplication/merging strategies in play, however they are by no means perfect. Having more sophisticated strategies in turn would mean an even bigger performance overhead.

The ideal solution would be to have a detector specifically designed and trained for the data at hand, which would make the need of a backend obsolete.

Simple

The Simple Backend simply takes the input image(s), and runs the detection on each image. It returns the detections as well as a stitched image of all input images. (Note that this is currently hard coded to be horizontally stitched)

DEI (Divide and Conquer)

The DEI Backend is specifically designed for the detection on the Strasse des 17. Juni, using a panorama image (made up by three images).

Because the detections would be very poor if it was run one the panorama directly, the Backend first splits the panorama image into partial images (blue squares):

These then get rotated, depending on the proximity to the center (no rotation in the center, more rotation on the outer sides).

This is done as the angle of the cars gets quite skewed on the outer sides, which hinders a successful detection.

The actual detection is now run on the partial images, after which the rotation und splitting are reversed to produce the final results.

Detector

YOLOv5 Simple

A very simple detector using a pretrained model provided by YOLOv5.

Configuration options: (under YOLO_SIMPLE:)

Parameter	Example Values	Description
size	“s”, “m”, “l” or “x”	Yolo model size

YOLOv5 Custom

A detector based on YOLOv5, custom trained with data provided by a previous project group.

Configuration options: (under YOLO_CUSTOM:)

Parameter	Example Values	Description
model	“yolov5”	Yolo custom class from registry
weights	“yolov5m.pt”	Weights to be loaded
img_size	640	Image size, applies to standard and trt
conf_thres	0.65	Confidence threshold (Confidence will be in the range 0-1.0)
iou_thres	0.6	IoU (Intersection over Union) threshold
classes	[0, 1, 2, 3, 5, 7]	Filtered classes: Person(0), Bicycle(1), Car(2), Motorcycle(3), Bus(5), Truck(7)
agnostic_nms	True, False	Agnostic nms (Non-maximum suppression)
max_det	20	Maximum detections per infered frame
trt	True, False	Enable trt engine for inference
trt_engine_path	“yolov5s6.engine”	Path to locally compiled engine

Parameter	Example Values	Description
trt_plugin_library	“libmyplugins.so”	Path to locally compiled lib

YOLOv5 TensorRT

> NVIDIA® TensorRT™ is an SDK for high-performance deep learning inference. It includes a deep learning inference optimizer and runtime that delivers low latency and high throughput for deep learning inference applications.

TensorRT bear the potential of significantly speeding up the inference. For that purpose, we investigated the usage of a TensorRT engine for predict the bounding boxes instead of the standard model.

In order to be able to enable TensorRT inference with YOLOv5, you have to generate a specific engine with respective library with this repository. A detailed How-To is given in the repo's description.

After you have successfully transformed the original model, you need to specify the engine's and plugin library's path and set `trt` to *True*.

Tracker

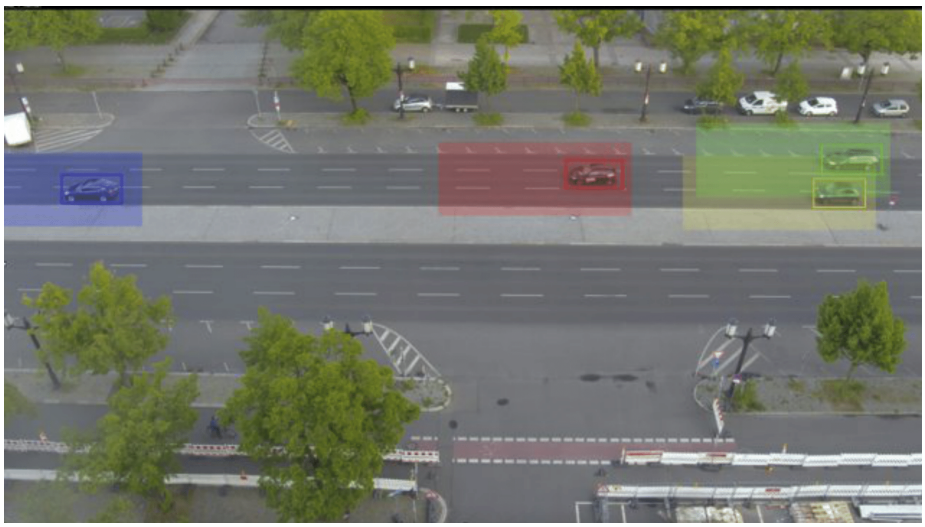
Centroid

Functionality

The Centroid-Tracker is a basic but very fast and reliable tracker. It operates using the euclidean distances between centroids of objects. Further information about the Centroid-Tracker used as a basis for this work can be found here. Moreover, the algorithmic background presented by previous groups was adjusted and extended to fit the new concept.

In this work the Centroid-Tracker was enhanced to allow for tracking of objects with unreliable detections and over the range of a stitched panorama generated from different camera streams. This results in special care being taken with wide camera angles due to the perspective size decrease and the transition regions between the stitched images.

The possible tracking match cases of objects between frames was limited to their surroundings due to their limited speed. This resulted in a rectangle around each object representing an area of possible movement between frames. This rectangle was further adjusted to allow for rotations due to the non-horizontal movement of objects especially on the side cameras. This resulted in these match area rectangles to be generated by the predicted movement of objects, their last known position and some padding. The padding was added to allow for acceleration or lane swapping of the object or even frame drops in the video source. The possible match regions for different objects can be seen in the image below.



To compensate for unreliable detections - e.g. caused by trees blocking the view or the change of view of the object due to the prespective shift - objects that are not being detected are held in memory for a certain amount of frames and are additionally actively premoved to account for their position after a possible redetection. This movement prediction is based on the past known movement of the object and thus allows for a very fast handling. However, due to the added time objects are stored for a possible redetection, objects that leave the recorded area are needlessly held on to. This problem was addressed by adding deregistration zones at the end of the recorded area where object are immediately removed from memory. These deregistration zones are present only in the direction of travel, so the separator line - defining the middle of the road - makes sure that these zones do not thread into the cameras incoming traffic.

Since non-moving objects, especially partially occluded non-moving objects, resulted in unreliable detections, a registration zone was added for incoming traffic into the cameras are of view. Only during the very first frame or afterwards only inside the registration zone, objects are being added to the list of tracked centroids. This allows for a more robust tracking and less matching errors between centroids.

At last, a transition zone was added between the different camera streams. This zone enables a better continued movement of objects in these regions which allows for higher temporal inconsistencies between camera streams, making the tracking as a whole more reliable.

Limitations

The achieved results using the Centroid-Tracker are very promissing. However, objects that are bunched up in a small area, especially in the wide angles of the cameras, pose some track matching errors. Due to the dynamically adjustable match area rectanlgles these errors where reduced to a minimum, but in some special edge cases they are still present (e.g. close driving cars right after a mutual lane swap). This problem could be tackled by implementing even stricter movement prediction rules and/or taking the actual image into account; But this would most likely deminish the computational performance lead of the Centroid-Tracker.

Furthermore, in very specific edge cases close to each other, partially occluded, parked cars combined with alternating detection leads to a track being assumed. This problem could be handled by adjusting the region of interest (ROI) to cut out the parked cars completely. However, this would require a non axis aligned ROI which makes implementation harder.

Configuration options: (under CENTROID:)

Parameter	Example Values	Description
TRACKER_CFG_PATH	"../configs/tracker/centroid.yaml"	Centroid config path

centroid.yaml:

Parameter	Example Values	Description
MAX_ID	100000	Highest possible id for tracked entities
MAX_DISAPPEARED	10	Maximum time (in frames) an object will be premoved after disappearance
USE_BETTER_RECTS	True	Use rotated match rectangles based on predicted positions for better tracking
USE_DYNAMIC_SCALING	False	Scale the match rectangles according to the perspective size decrease objects

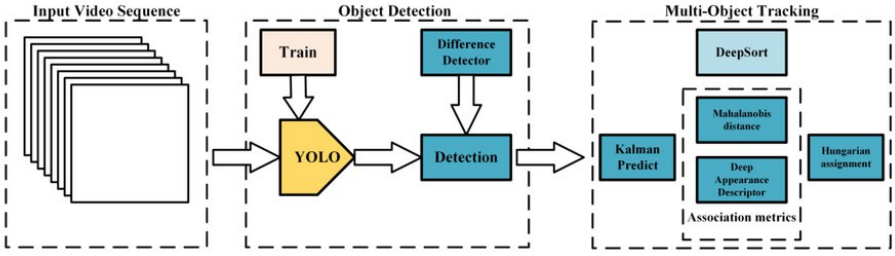
Parameter	Example Values	Description
DISTANCE_TOLERANCE	500	Maximum distance to allow for an object tracking match (only active if USE_BETTER_RECTS is False)
VERTICAL_TOLERANCE	100	Maximum vertical distance to allow for an tracking tracking match (only active if USE_BETTER_RECTS is False)
FRONT_DISTANCE_TOLERANCE	80	Padding added to the direction of movement of the object for tracking matches (only active if USE_BETTER_RECTS is True)
BACK_DISTANCE_TOLERANCE	150	Padding added to the back of an object for tracking matches (only active if USE_BETTER_RECTS is True)
SIDE_DISTANCE_TOLERANCE	50	Padding added to the sides of an object to allow for tracking matches (only active if USE_BETTER_RECTS is True)
FRAME_WIDTH	11520	Total image width
TRANSITION_WIDTH	200	Transition region width between camera images
LANE_SEPARATOR_LL	1117	y-coordinate of the separator line (middle of the road parking area) - left
LANE_SEPARATOR_LC	925	y-coordinate of the separator line - left-center
LANE_SEPARATOR_CR	925	y-coordinate of the separator line - right-center

Parameter	Example Values	Description
LANE_SEPARATOR_CR	1151	y-coordinate of the separator line - right
DEREG_ZONE_L	1600	Deregistration zone x-boundary left
DEREG_ZONE_R	10500	Deregistration zone x-boundary right
REG_ZONE_L	2750	Registration zone x-boundary left
REG_ZONE_R	9750	Registration zone x-boundary right

DeepSORT

Functionality

DeepSORT is a tracking-by-detection algorithm which considers both the bounding boxes of the detection results and the information about appearance of the tracked objects. Therefore, DeepSORT is very sophisticated but also complex and slower algortihm compared to the Centroid-Tracker. Thereby making real-time tracking very hard. The algorithm is implemented as provided by Wojke, Nicolai and Bewley, Alex and was only adjusted to accept a common interface between trackers. A functional diagram of the internal workings of DeepSORT can be seen below.



Limitations

Even though DeepSORT offers very good results, also with stiched panoramic images, the feature extractor - being the main part of the “Deep” side of DeepSORT - needs further training on the objects being tracked. Especially since objects appear first as a frontal upper view, convert to a fully upper side view (in the center camera image) and exit seen from the upper back; there was no training to compensate for these disturbances. Furthermore, the ROI could be adjusted to completely cut out parked objectes to allow for a more reliable tracking. However, this would require a non axis aligned ROI which makes implementation harder.

Configuration options: (under DEEPSORT:)

Parameter	Example Values	Description
TRACKER_CFG_PATH	"../configs/tracker/deep_sort.yaml"	DeepSORT config path

deep_sort.yaml:

Parameter	Example Values	Description
REID_CKPT	"../weights/tracker/deepsort/ckpt.t7"	Path to the feature extractor. Path is relative to the feature extractor to allow for relative addressing
MAX_DIST	0.6	Maximum cosine distance to nearest neighbor to allow for a tracking match
MIN_CONFIDENCE	0.4	Minimum confidence for an object to be considered
NMS_MAX_OVERLAP	0.7	Non-maximum Suppression max overlap fraction. ROIs that overlap more than this values are suppressed
MAX_IOU_DISTANCE	0.75	Max intersection over union distance. Associations with cost larger than this value are disregarded
MAX_AGE	70	Maximum number of missed timesteps before a track is deleted
N_INIT	3	Number of consecutive detections before the track is confirmed
NN_BUDGET	10000	Maximum number of samples. Removes the oldest samples when the budget is reached
MAX_ID	100000	Highest possible id for tracked entities