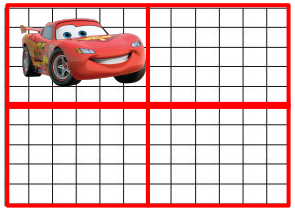
Task 1.1: Optical flow with Block Matching (Team 5) (1/2)

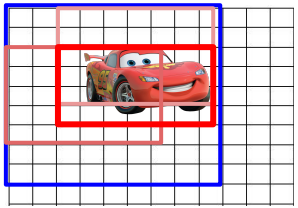
Brief reminder of Block Matching:

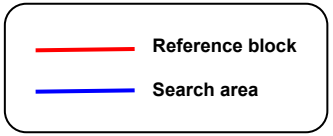
- Split Reference image into blocks of NxN pixels.
- Search block in target Image.
  - Search in window of (2P + N) x (2P + N)
- Take minimal difference as displacement of pixels.
- Repeat for all blocks of reference image.

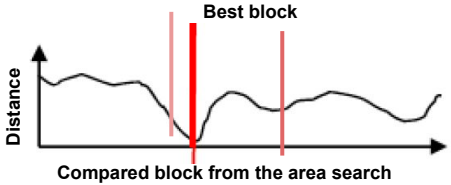
Reference Image



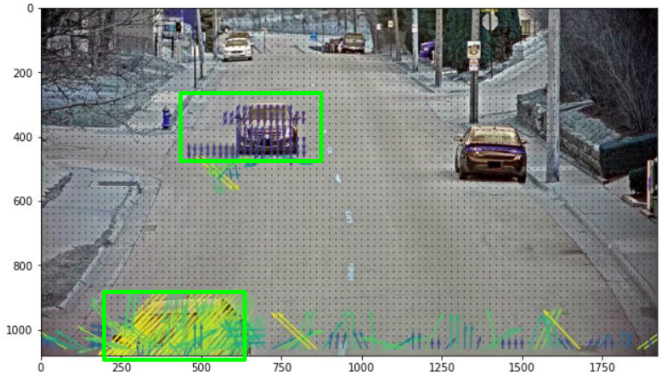
Target Image



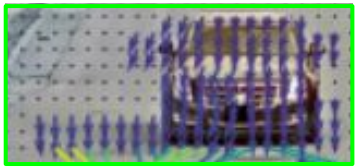


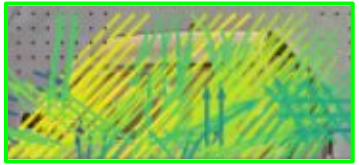


Example of optical flow with block matching



The output matches what could be expected. It's a still scene where there are only two moving cars. So, the optical flow of the background is zero (no displacement of the pixels). And both moving cars showed the proper direction of movement towards the left bottom corner.



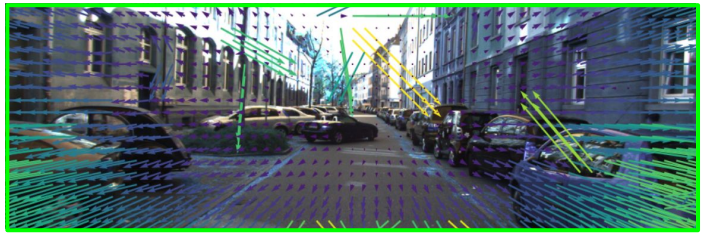


Some errors are probably due to the existence of noise at the bottom of the image. It's visible that there's some repeated error in that section of the image.

Task 1.1: Optical flow with Block Matching (Team 5) (2/2)

There are 5 hyperparameters to adjust: compensation model, block size, area of search, stride, and error function. We have **estimated all the parameters through a grid search***. This option is quite resourcing intensive, but adjusting hyperparameters together should help reach a more optimal setting. In the following table, only the best results for each error function are shown. As we expect, the best results are always given by NCC (Normalized Cross-Correlation) with the downside being more expensive computationally. It takes into account the mean and standard deviation of the pixels values within the two blocks to compare, making it more stable against noise.

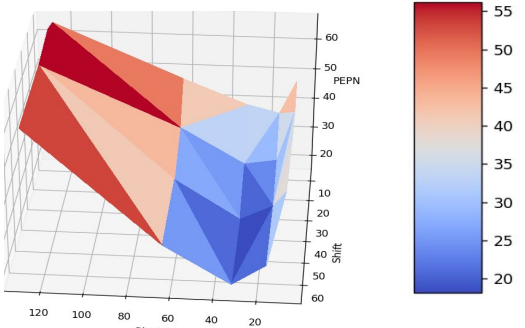
B = Block size S = Shift size	Backwards			Forwards		
	SSD	**SAD**	**NCC**	**SSD**	**SAD**	**NCC**
	B:32 S:64	**B:128 S:8**	**B:32 S:32**	**B:32 S:32**	**B:64 S:8**	**B:32 S:32**
MSNE	14.50	12.76	4.77	7.17	9.35	2.52
PEPN %	40.96	95.40	19.80	35.18	57.11	16.08
Time	2 min 42s	1s	2 min 6s	47s	1s	2 min 7s



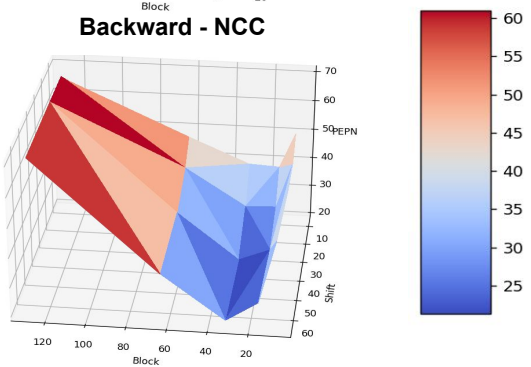
*To check all the results of the grid search they can be checked in git repository

After analyzing the parameter search, we've seen that in both compensations approaches, the results behave in the same way while varying the parameters. And we also check what is expected, when the shift parameter is equal to or bigger than the block size the computation time grows exponentially (Block matching is an inefficient $O(N^2P^2)$ algorithm).

Forward - NCC



Backward - NCC



Task 1.2: Off-the-shelf Optical Flow (Team 5) (1/2)

Methods

Farneback[1], OpenCV's implementation of **dense** optical flow. It is based on Gunner Farneback's algorithm, and it computes the optical flow for all the points in the frame.

Pyflow[2] is based on the implementation of Liu[3], and it uses the concept of **coarse-to-fine warping strategy**[4]. The algorithm's energy functional combines three assumptions: a brightness and a gradient constancy, and a discontinuity-preserving spatio-temporal smoothness constraint.

Perceiver-IO[5] is a general neural network architecture that performs well for structured input modalities and output tasks, which extends to tasks like optical flow. It achieves **state-of-the-art** performance on Sintel optical flow estimation with no explicit mechanisms for multiscale correspondence.

Result from Kitt-mots Frame 45

Metric	Farnebacks	Pyflow	Perceiver-IO
MSNE	4.200	0.966	0.742
PEPN(%)	28.394	7.959	4.075
Time(s)	0.15	7.31	5.97*

* Performance recorded with a different setup.

Frame 45 visualised

[1] Farnebäck, Gunnar. "Two-frame motion estimation based on polynomial expansion." Scandinavian conference on Image analysis. Springer, Berlin, Heidelberg, 2003.
[2] Pathak, Deepak, et al. "Learning features by watching objects move." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.
[3] Liu, Ce. Beyond pixels: exploring new representations and applications for motion analysis. Diss. Massachusetts Institute of Technology, 2009.
[4] Brox, Thomas, et al. "High accuracy optical flow estimation based on a theory for warping." European conference on computer vision. Springer, Berlin, Heidelberg, 2004.
[5] Jaegle, Andrew, et al. "Perceiver io: A general architecture for structured inputs & outputs." arXiv preprint arXiv:2107.14795 (2021).

Task 1.2: Off-the-shelf Optical Flow (Team 5) (2/2)

Frame 45

Farnebacks

Pyflow

Perceiver-IO

Comparison

Farneback provides the **fastest** performance out of the three tested methods, while **Perceiver-IO** achieved the **lowest** MSNE and PEPN**.**

Both **Pyflow** and **Perceiver-IO** are able to provide estimation consistently across the image, whereas **Farneback** fails to detect parts of the large movements as shown by the white patches.

The optical flow estimated by **Perceiver** has more ‘detail’, more refined and sharp compared to **Pyflow**.

Color wheel

Example with Pyflow

T1.3 Object Tracking with Optical Flow (Team 5) (1/2)

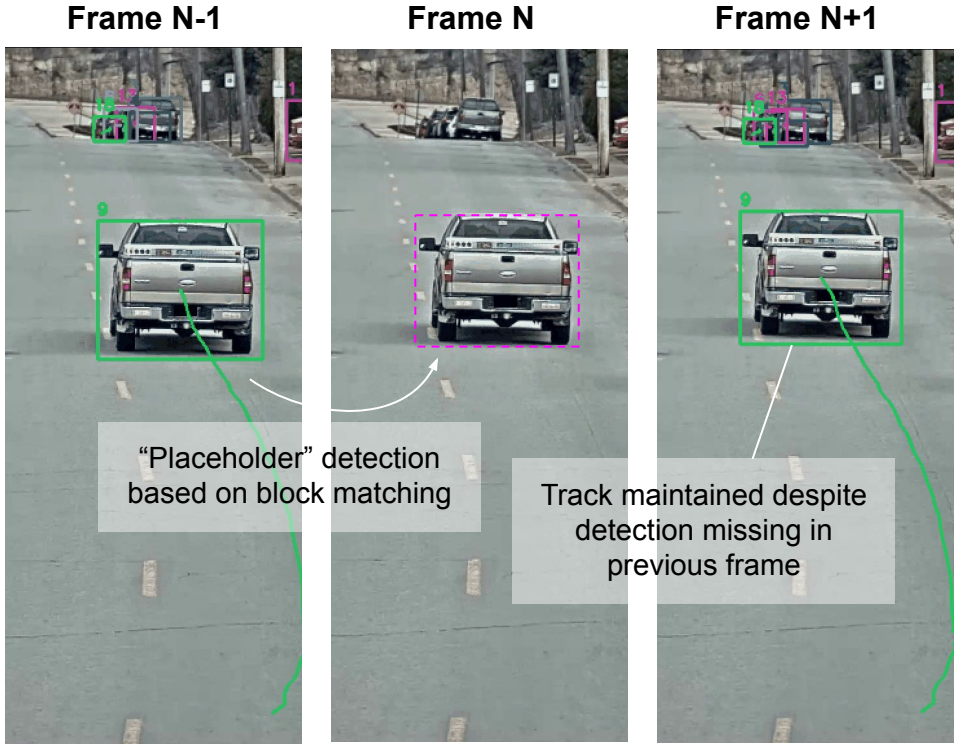
Approach #1 - Compensate missing detections w/ block matching predictions

Since tracks are lost as soon as an object is not detected in one frame, the intention here is to identify detections in the previous frame N-1 that do not have a match in the current frame N and use block matching to estimate where they should be located. This should in principle help maintain tracks alive despite sporadic missing detections.

To avoid using this recourse too many times and keeping tracks alive for too long, a counter is associated to them that increases every time it is applied. Tracks that have used this more than 2 times in a row are discarded.

	IDF1	Computation time
No optical flow	0.716	2m 52s
Approach #1*	0.675	6m 02s

Despite involving longer computation times, this approach does not lead to an increased performance. The IDF1 instead decreases, which could have to do with FPs from keeping tracks alive for 2 extra frames whenever vehicles disappear.



Carried out with a search area of 10px and a stride of 1px and SSD.

T1.3 Object Tracking with Optical Flow (Team 5) (2/2)

Approach #2 - Correction of frame N-1 bounding box detections w/ block matching

The idea is to compute the estimated displacement of box detections from the previous frame N-1 to the current frame N with block matching, and apply it to these so that they overlap better with current detections.

When “correcting” box detections from the previous frame N-1 by displacing them according to the forward block matching, it can be appreciated how these stick to the current frame N detections better.

However, when the distance between detections from frames N and N-1 becomes larger than the search area of the block matching, this effect stops.

	IDF1	Computation time
No optical flow	0.716	2m 52s
Approach #2*	0.719	38m 58s

The improvement in IDF1 resulting from better IoUs is clearly not worth the way higher computation times.

No optical flow | **Block matching correct.**

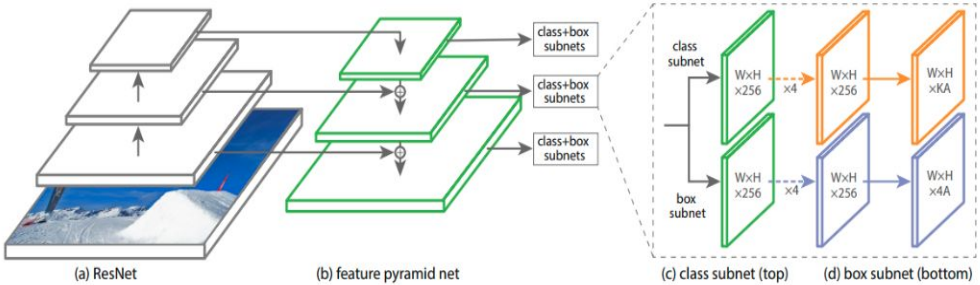
Frame N detects. Frame N-1 detects. Corrected frame N-1 detects.

**Carried out with a search area of 10px and a stride of 1px and SSD.*

Task 2.1: Multi-target single-camera (MTSC) tracking (Team 5) (1/2)

Methodology

- Detections used were extracted with the **RetinaNet** with **ResNet-101-FPN** backbone from *Detrectron2*[1].
- Training used **SEQ01** and **SEQ04** with **1/30 of the frames of each camera** in the sequence.
  - Limited training due to memory limitations.
- Evaluation of **each camera in SEQ03** with **IDF1** metric from *py-motmetrics*[2].
  - Detections of parked cars were removed as ground truth did not contemplate them.
  - SORT Kalman filter applied to predictions to improve the tracking.



RetinaNet model architecture[3]

Finetune Pretrained RetinaNet w/ subset of train → Post-Process Detections Parked Cars + Kalman → Evaluation w/ different cameras and sequences

[1] Detectron2 from Facebook Research.
[2] Py-motmetrics python library.
[3] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He: “Focal Loss for Dense Object Detection”, 2017

Task 2.1: Multi-target single-camera (MTSC) tracking (Team 5) (2/2)

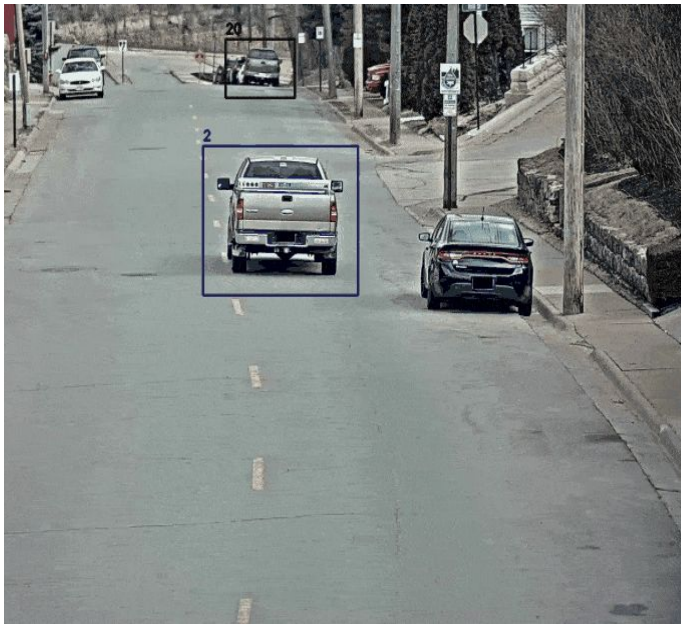
The ground truth provided for these new sequences does not include parked cars, which forces us to disregard detections that remain static throughout frames. We do so by assigning a *"parked"* = *True* label to any detections from a *"track"* = *i* whenever the IoU between said detection and its corresponding detection from the previous frame is larger than 0.95. We later ignore these cases when computing the IDF1.

Example: SEQ 3 - c10

IDF1	
All detections	**Ignoring parked cars**
0.221	0.278

While using the described approach to ignore parked cars slightly increases the resulting IDF1 score, this score is still very low. Based on the tracking animation to the right, this is obviously related to poor detections and tracking.

The problem therefore most likely lies on the training needing to be better optimized or on the used model. Detections carried out with *RetinaNet* displayed an AP score of only 0.593 when trained on these new sets, which is significantly lower than the results obtained last week.



Task 2.1: Multi-target single-camera (MTSC) tracking (Team 5) (2/2)

The ground truth provided for these new sequences does not include parked cars, which forces us to disregard detections that remain static throughout frames. We do so by assigning a *“parked”* = *True* label to any detections from a *“track”* = *i* whenever the IoU between said detection and its corresponding detection from the previous frame is larger than 0.95. We later ignore these cases when computing the IDF1.

Example: SEQ 3 - c10

IDF1	
All detections	**Ignoring parked cars**
0.221	0.278

While using the described approach to ignore parked cars slightly increases the resulting IDF1 score, this score is still very low.