Master in Computer Vision Barcelona

Project
Module 6
Coordination

Video Surveillance for Road Traffic Monitoring

J. Ruiz-Hidalgo / X. Giró

j.ruiz@upc.edu / xavier.giro@upc.edu

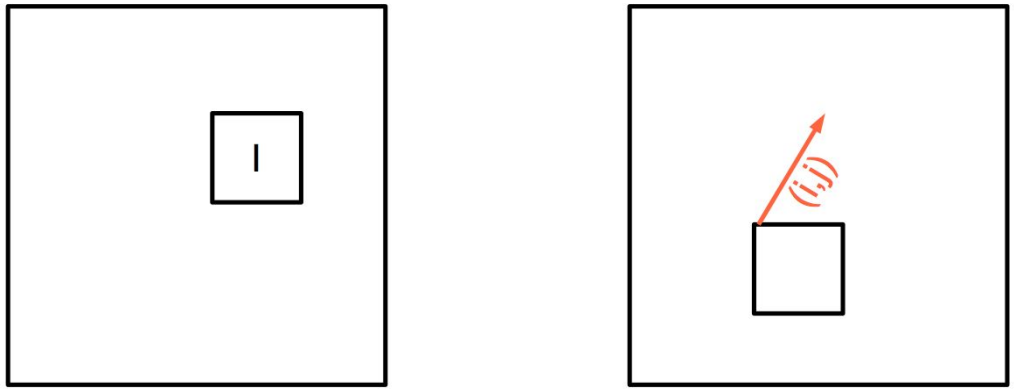
Week 4: Report (part1)

Master in
Computer Vision
Barcelona

UAB UOC UPC upf.

Task 1.1: Optical flow with Block Matching

- Implement a block matching solution for optical flow estimation.
- Configurations and parameters to explore:
  - Forward or Backward compensation.
  - Area of Search.
  - Size of the blocks.
  - Error function.
  - etc.



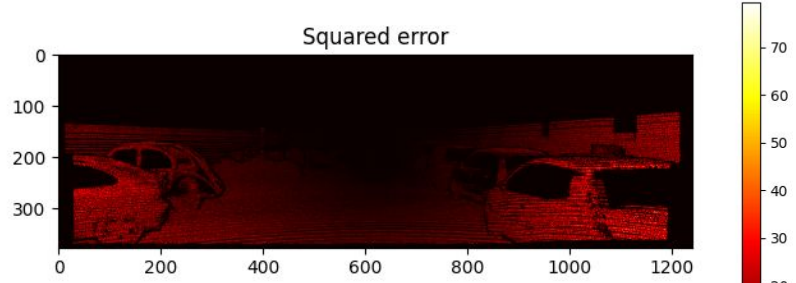
Task 1.1: Optical flow with Block Matching (all teams)

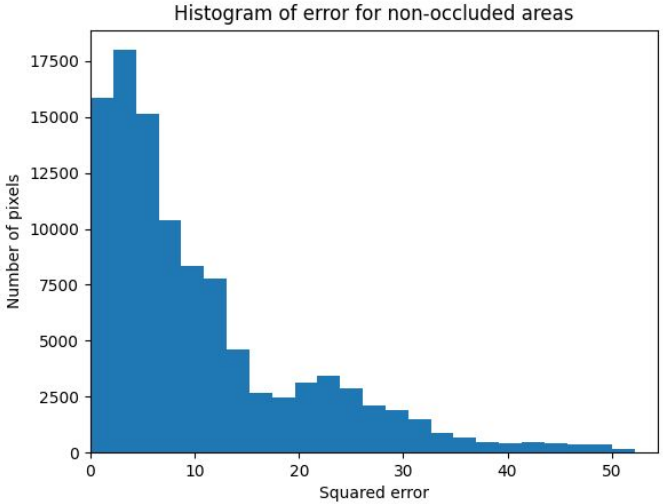
	Report all explored values (use **bold** to indicate the best configuration)					Best configuration	
	Area of Search	Size of Blocks	Step size	Error function	Others	Avg. PEPN	Avg. MSEN
Team 1	32	16	1	SSD	Gaussian smoothing did not improve the results. No optical flow vector smoothing has been used	0.2156	2.7128
Team 2	76	24	1	Normalized Cross Correlation	Postprocessing: - Noise detection + inpainting - Median blur (size 5) - Gaussian blurs (size 55 + size 27 + size 13)	7.03%	1.06
Team 3	4,8,16,**32**,64	12,24,48,96, **192**	We fixed the step size to half the block size	**CCOEFF**,CCORR,SQDIFF		0.125	8.490
Team 4	10, **60**,110	10, **50**, 90, 130,170	1, **3**, 5, 7, 9	**NCC**,SAD,SSD		33.58%	6.56
Team 5	block_side* [**1.1**…2.0]	[3…**40**]	block_side	SSD	• Interpolation: nearest, **cubic** • iterative search depth: [1…**5**]	42.82%	4.52
Team 6	2, 4, 8, 16, **32**, 64, 128	2, 4, 8, **16**, 32, 64, 128	1,2, 4, **8**, 16, 32	**NCC**,SAD,SSD		13.67%	2.71

* Images: 000045_{10, 11}.png

Task 1.1: Optical flow with Block Matching (Team X) (max 4 slides)

Task 1.1: Optical flow with Block Matching (Team 1) - [1/4]
Recap of week1 OF from kitti detection (seq 000045)





Redish pixels indicate where we have **larger errors** in terms of squared error.

The **more red/white** the pixel the **higher** the **difference** between the **ground truth** and the **predicted** optical flow.

On the other hand, **black pixels** indicate that the **motion** of the objects in the sequence have been **accurately predicted.**

The histogram errors show that in the sequence the **majority of pixels have very low squared error**.

From the visual results we can conclude that the given predictions were **good at estimating the optical flow of those points far from the camera**, and as a result did not have much motion. But struggled **(very bad results) at predicting the optical flow of those points closer to the camera** and the motion was more noticeable.

Kitti detections reference:

Image	000045_10.png
MSEN	10.6270
PEPN	0.7856

Task 1.1: Optical flow with Block Matching (Team 1) - [2/4] Block matching algorithm (seq 000045)

Method used:

- Block matching is a technique that **calculates the optical flow of an image by comparing the pixel values of two adjacent frames**.
- The basic idea is to divide the image into small blocks or patches and then **track the movement of each block** from the first frame to the next.
- **For each block in the first frame**, a **corresponding block is searched in the next frame** that best matches it in terms of intensity or feature similarity.
- The search is usually **performed in a small window around the original block**, and the matching process can be done using various metrics such as MSE, MAE, SSD, etc.
    - The max movement observed in the sequence happens on the borders of the image, and **features move up to 30 pixels** approximately. Therefore, **having higher search areas of 32 will not make much sense**. Having **small areas** won't either, as **the sides** of the image will be noisy, as we **won't be able to find a good match**.
- The **displacement** between the original block and the best-matching block is then **considered** as the **optical flow vector for that block**.
- The **resulting flow vectors can be smoothed** or refined using techniques such as Gaussian smoothing to obtain a more accurate and consistent optical flow field.
    - Although visually the result seemed better, gaussian smoothing yield worst mestric results (PEPN and MSEN).
- Block matching is a relatively **simple and fast** method for computing optical flow, but it **has limitations** in handling complex motion patterns or occlusions, and may require careful parameter tuning for optimal performance.

Task 1.1: Optical flow with Block Matching (Team 1) - [3/4] Block Matching hyperparameter search (seq 000045)

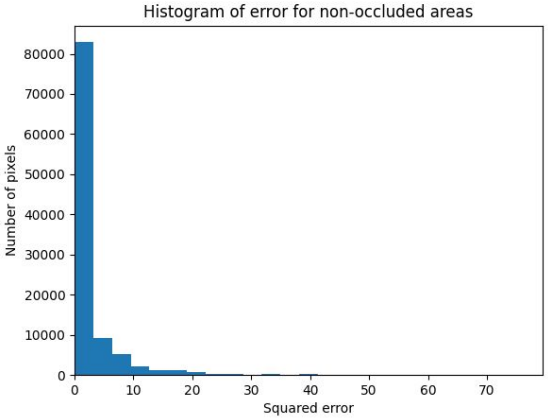
Grid search: **Best results in bold and green highlight (more explanations on description of slide)**

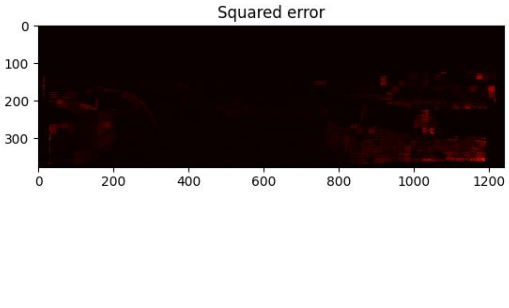
- Block size: 4, 8, **16**, 32, 64
- Search area: 1, 4, 8, 16, **32**
- Step size: **1**, 2, 4
- **Forward** or Backward (Forward: image45_10 as reference and image45_11 as area of search)
- Distances: **SSD**, MSE, SAD, MAD

Execution time of best result: 1:27 min

step size	1	2	4
MSEN	**2.7128**	4.7723	9.9879
PEPN	**0.2156**	0.3123	0.5586

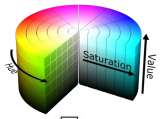
example with block size 16 and search area 32





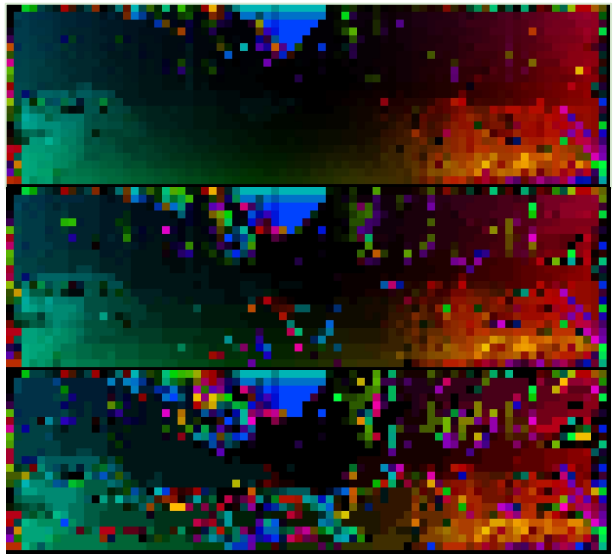
Step size

Best OF: 1



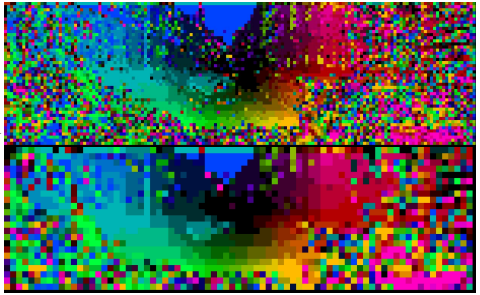
2

4



Task 1.1: Optical flow with Block Matching (Team 1) - [4/4]
Block Matching visual results (seq 000045)

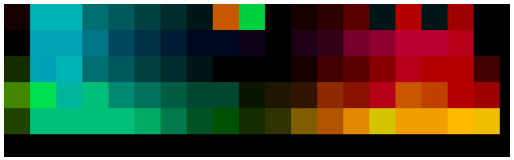
Optical Flow of **small search area** performed **decent vectors on the center of the image**, and very bad on the sides of the image. This experiment has **demonstrated the theory explained in slide [2/4]**. Same happens with a higher block size (bottom image)



search area = 4
block size = 4

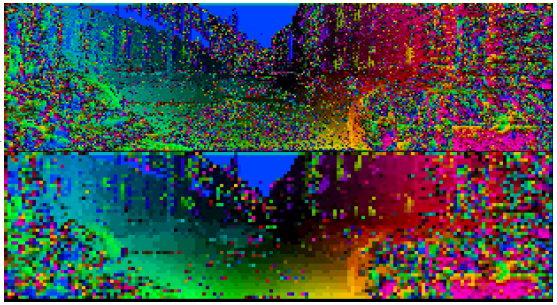
search area = 4
block size = 8

If the **search area is small, but block size is huge**, then optical flow is very simplified, which leads to **bad MSEN and PEPN results**, although visually pleasant.



search area = 64
block size = 8

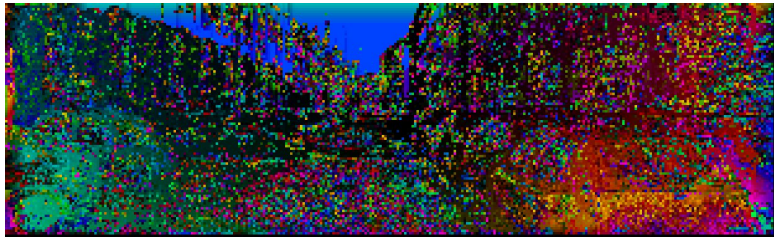
Optical Flow of **higher search areas** with **small block size**, also yield bad optical flows, as there is a **lot of noise**, which traduces to **blocks that didn't find the good correspondence**.



search area = 8
block size = 4

search area = 8
block size = 8

In this last example, we can see that we have **noise because** the **area** is way **higher** than the **block size**, but we see that now, **with a higher search area**, the **optical flow in the sides of the image** can be **correctly estimated**.



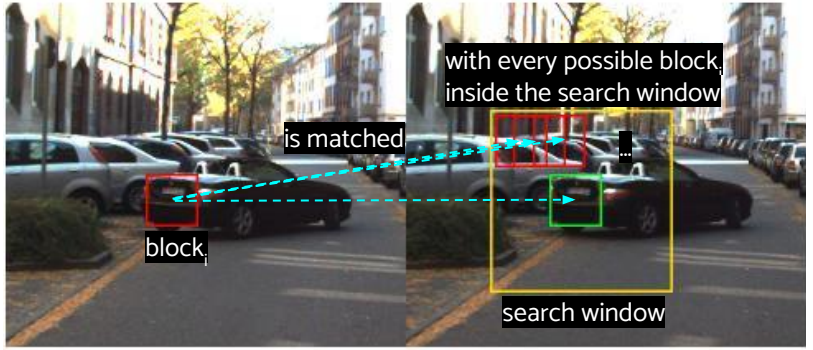
search area = 32
block size = 4

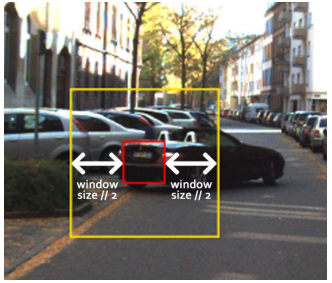
Task 1.1: Optical flow with Block Matching (Team 2) - [1/4]

We implement the Block Matching algorithm to estimate Optical Flow. We set the **error function**, the **estimation type**, the **search window size** and the **block size** as parameters that can be set by the user. We test 4 error functions: Mean Squared Error (MSE), Mean Absolute Error (MAE),,Normalized Cross-Correlation (NCCorr) and the Normalized Correlation Coefficient (NCCoeff). Below, you can find the equations for each of them. We define $\mathbf{Im}_{t-1}$ as the previous image and $\mathbf{Im}_t$ as the current one. We refer to blocks in $\mathbf{Im}_{t-1}$ as $\mathbf{block}_i$ and those in $\mathbf{Im}_t$ as $\mathbf{block}_j$.

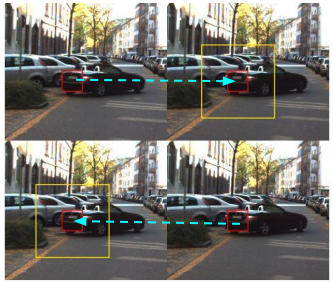
The Block Matching algorithm:

1. Divide $\mathbf{Im}_{t-1}$ into blocks of size *block_size*.
2. If the estimation_type is set to "backward", swap $\mathbf{Im}_{t-1}$ and $\mathbf{Im}_t$.
3. Iterate over every $\mathbf{block}_i$ in $\mathbf{Im}_{t-1}$:
   a. Define a **search_window** around the corresponding $\mathbf{block}_i$ in $\mathbf{Im}_t$.
   b. If the selected *error function* is MSE or MAE:
      i. Compare block_i with every possible $\mathbf{block}_j$ inside the **search_window** in $\mathbf{Im}_t$.
      ii. Return the coordinates of the best matching $\mathbf{block}_j$ with minimum error.
   c. Else:
      i. Run template matching calling cv2.matchTemplate with $\mathbf{block}_i$ and **search_window**, which slides the block through the search window and computes the error function.
      ii. Return the coordinates where there is a maximum.
   d. Compute the optical flow vector for all the pixels in $\mathbf{block}_i$ by subtracting to the coordinates of the matching $\mathbf{block}_j$ in $\mathbf{Im}_t$ the coordinates of the $\mathbf{block}_i$ in $\mathbf{Im}_{t-1}$.
4. If the **estimation_type** is set to "backward", negate the u and v components.





The search window is calculated such that
width=window_size+block_size
height=window_size+block_size.



Estimation types:
First row: Forward estimation
Second row: Backward estimation

MSE

$$\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(I_{m,n}^{(2)} - I_{m,n}^{(1)})^2$$

MAE

$$\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}|I_{m,n}^{(2)} - I_{m,n}^{(1)}|$$

Normalized Cross-Correlation
(check speaker notes)

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x', y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x', y+y')^2}}$$

Normalized Correlation Coefficient

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x', y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x', y+y')^2}}$$

$$T'(x',y') = T(x',y') - 1/(w \cdot h) \cdot \sum_{x'',y''} T(x'',y'')$$

$$I'(x+x', y+y') = I(x+x', y+y') - 1/(w \cdot h) \cdot \sum_{x'',y''} I(x+x'', y+y'')$$

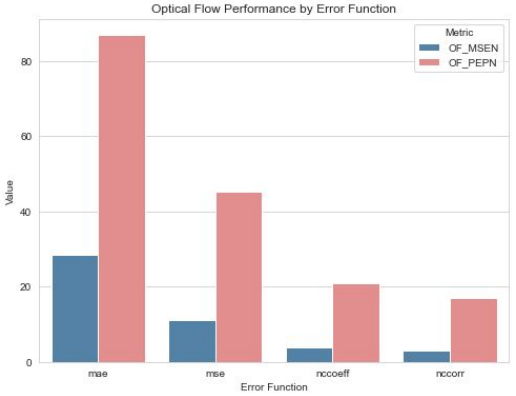
Task 1.1: Optical flow with Block Matching (Team 2) - [2/4]

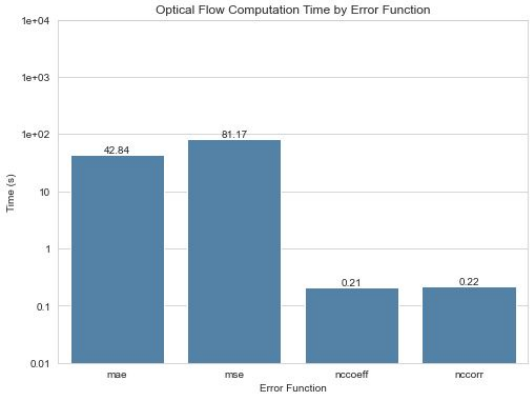
In order to find the best parameters, we conduct an Optuna Search over different values of the parameters explained in the previous slide and look to minimize the MSEN. Moreover, after running 200 trials, we analyze the results.

Parameters searched*(check speaker notes):
block size = [8, 128]
window size = [8, 128]
estimation type = [forward, backward]
error function = [MAE, MSE, NCCorr, NCCoeff]

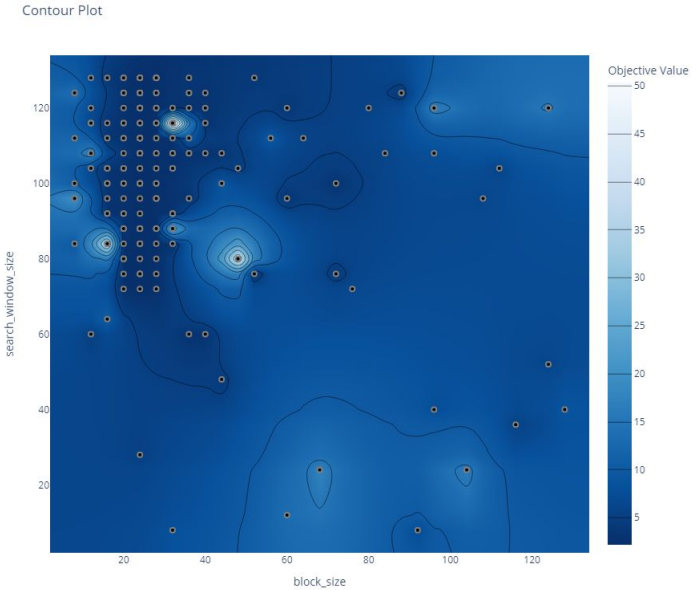
Best parameters (2.16MSEN, 13.07PEPN):
block size = 24
window size = 76
estimation type = Forward
error function = NCCorr

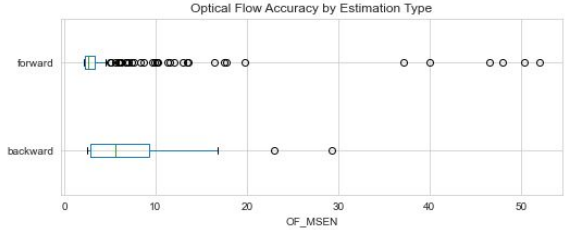
We find that the error functions that run on **Template Matching (NCCorr and NCoeff) outperform the rest in both metrics and execution time**. Both have similar performance, though. We see that most of the promising points are clustered in a ravine that comprises a **window size between 70 and 128** and a **block size between 20 and 32**. This window size makes sense since the maximum GT displacement is u,v=49.38,16.11. Surprisingly, we notice that the forward method achieves slightly better results on average. However, we must note that Optuna tends to explore the most promising paths at the start and stick with them. After checking some of the rows in the results database, we saw that it started exploring the forward estimation paths.



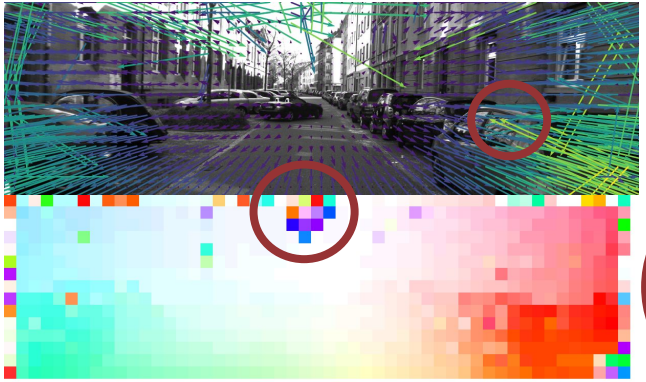


**Mean values over 200 trials used in both tables above.

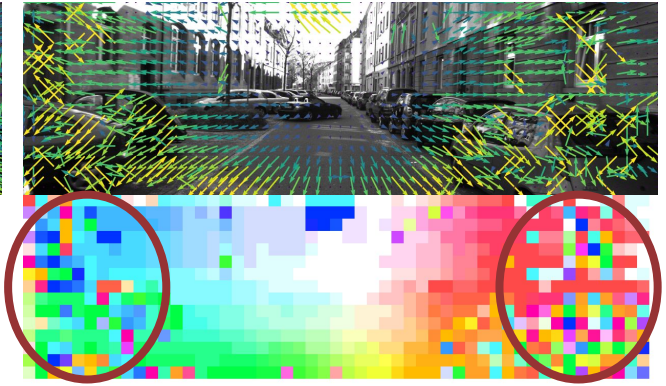




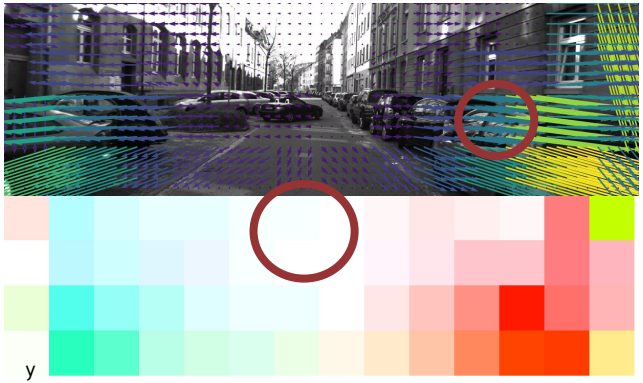
Task 1.1: Optical flow with Block Matching (Team 2) - [3/4]



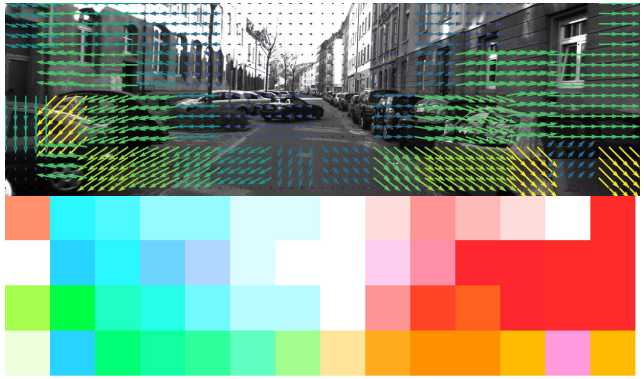
bs[24], ws[76], 0.127s
2.16MSEN, 13.07PEPN



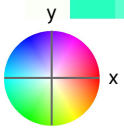
bs[24], ws[12], 0.068s
7.44MSEN, 41.75PEPN



bs[88], ws[76], 0.094s
4.64MSEN, 38.19PEPN



bs[88], ws[12], 0.022s
7.15MSEN, 47.13PEPN



Here, we qualitatively assess the effect of different **block size** and **window size** values using the NCCorr and forward estimation. **Blue** is the best configuration.

In all configurations, we see that the algorithm **struggles with the borders** since the target may be outside of the image. The block size may also play a factor if the dimensions are not divisible by it.

Larger values of the **block size** show a **regularizer effect** since it allows the algorithm to estimate the OF on **textureless areas** and **specularities** (see bs[88], ws[76]). However, **it worsens the metrics, as it is too coarse** and we lose too much detail.

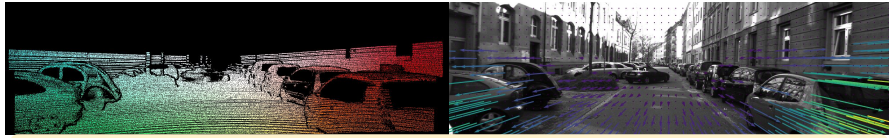
Reducing the **search window size**, although it **speeds up the algorithm**, produces significantly more noise and inaccuracies (see bs[24], ws[12]). This seems to happen when the target block lies outside the search window. Increasing the **block size** as well slightly alleviates this, but still has the previously stated problems (see bs[88], ws[12]).

Task 1.1: Optical flow with Block Matching (Team 2) - [4/4]

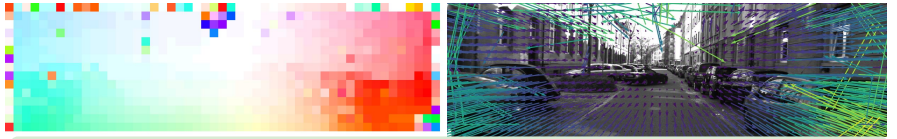
We have identified three areas of improvement from the previous assessment: **borders**, **noise**, and "**superpixels**". To address these problems, we apply the following post-processing:

1. In the borders, interpolate with neighboring block values (only those from the inner part of the image).
2. Slide a window of size block_size*3 with stride=block_size and compute the variance at each block. Create a mask with blocks whose variance is > 100 and dilate with kernel=block_size*3
3. Add those OF blocks whose color difference w.r.t. the average color of its neighboring blocks exceeds a threshold of 8 to the mask
4. Apply an inpaint algorithm for the pixels inside the mask
5. Apply a median filter (size=5) and 3 consecutive Gaussian filters (sizes 55, 27, and 13)

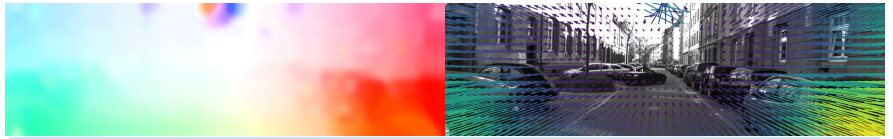
This solves the 3 issues mostly, though we haven't been able to eliminate the sky noise cluster. With this post-processing, we have been able to further improve the results both in metrics and qualitatively, while keeping a low execution time.



Ground truth

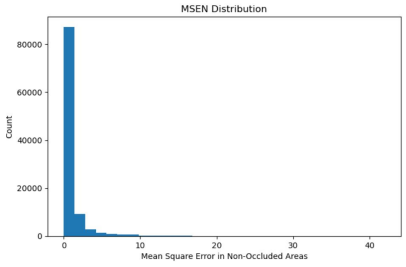


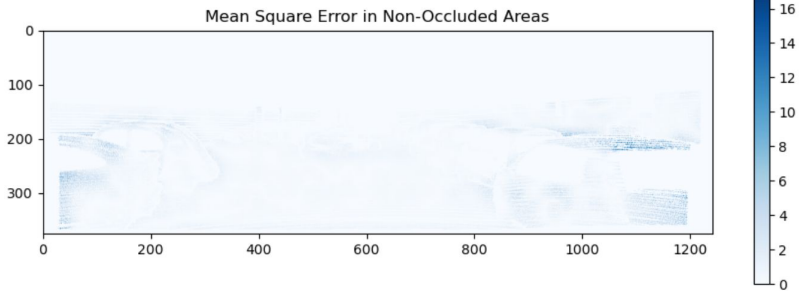
bs[24], ws[76], no-post - 0.127s, 2.16MSEN, 13.07PEPN

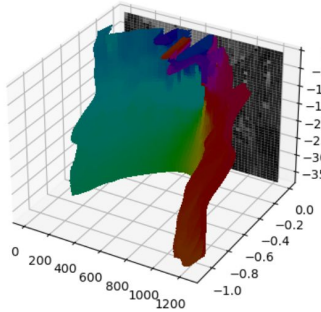


bs[24], ws[76], with post - 0.492s, **1.06MSEN, 7.03PEPN**

Final results:







Task 1.1: Optical flow with Block Matching (Team 3) [1/4]

- Block matching is an effective method for estimating optical flow by comparing the pixel values of two adjacent frames.
- The technique involves searching for a block of the reference image in the target image within a given region.
- The search can be done in various ways, we are going to check every possible block within the region to find the best fit.
- Matching metrics such as sqdiff, ccorr, ccoeff, etc., can be used to determine the degree of similarity or difference between the blocks.
- Block matching is a useful tool for optical flow estimation due to its simplicity and speed. However, it may struggle to handle complex motion patterns or occlusions, leading to less accurate results.
- Parameter tuning is required to achieve optimal performance with block matching.
- Despite its limitations, block matching remains a popular choice in many applications and can still provide useful results as we are going to show in next slides.

Template block

Searching region

Sliding window

Block size: Size of the block you try to match
Quantization: Step size, how often we compare a block (To make the search faster we fixed it to Block size/2)
Search area: Area where we search the block
Motion: If we search the match in the next or previous frame
Template match mode: Type of the template matching operation

000045_10

000045_11

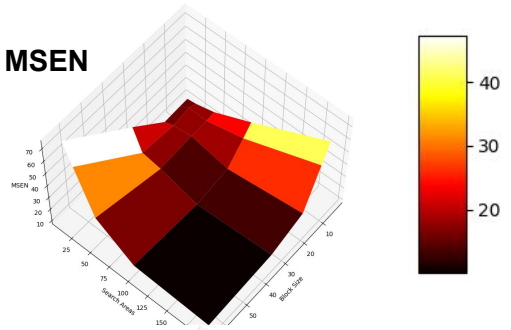
Task 1.1: Optical flow with Block Matching (Team 3) [2/4]

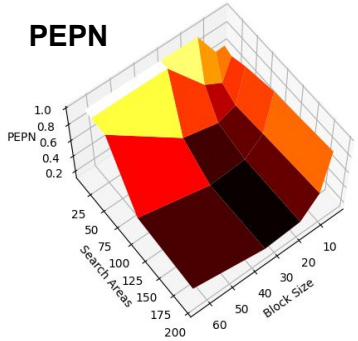
We performed grid search for the following values:
Block size: 4, 8, 16, 32, 64
Search area: 12, 24, 48, 96, 192
Template match mode: TM_CCOEFF_NORMED, TM_CCORR_NORMED, TM_SQDIFF_NORMED

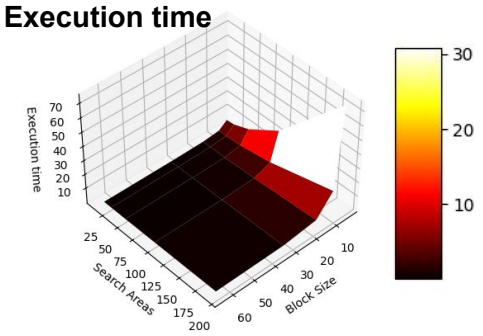
Both sequences can use similar hyperparameters to obtain the best solution (Search area: 192, Block size: 32)

The plots are obtained using ccor normed matching

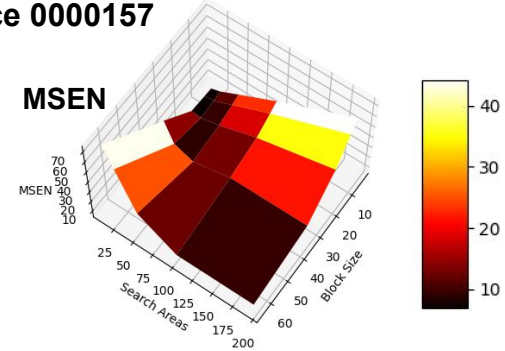
Sequence 000045

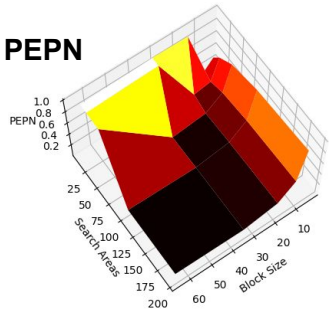


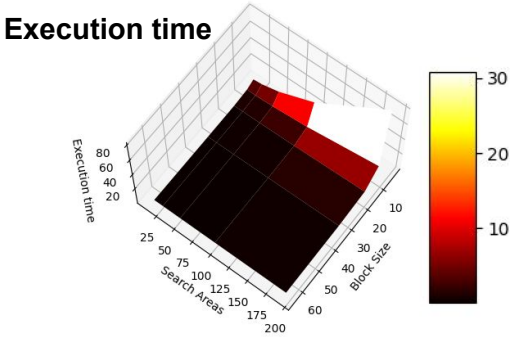




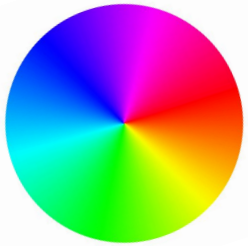
Sequence 0000157

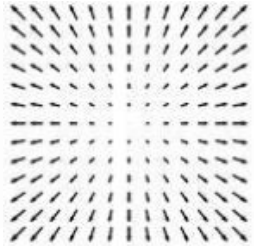






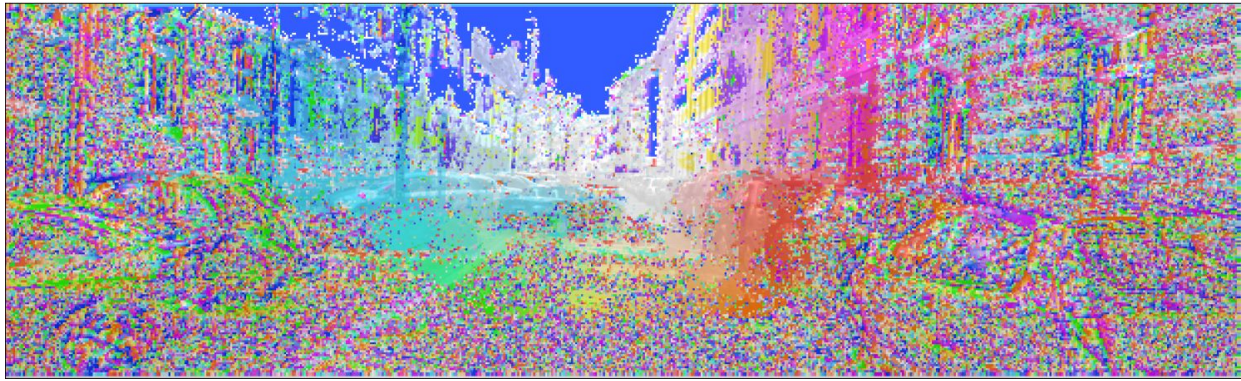
Task 1.1: Optical flow with Block Matching (Team 3) [3/4]





As shown in plots from the previous slide, our performance is not good when using small block sizes. Furthermore, we observed that our results deteriorate even further when using a large search area. This happens because if the block size is too small, we may end up matching the wrong block easily, and if moreover the search area is large, we can mismatch with further blocks.
On the other hand, if the block size is too large, the optical flow will be too coarse.
Images on the right show the contrast between using a small block versus the optimal block size.

Sequence 000045



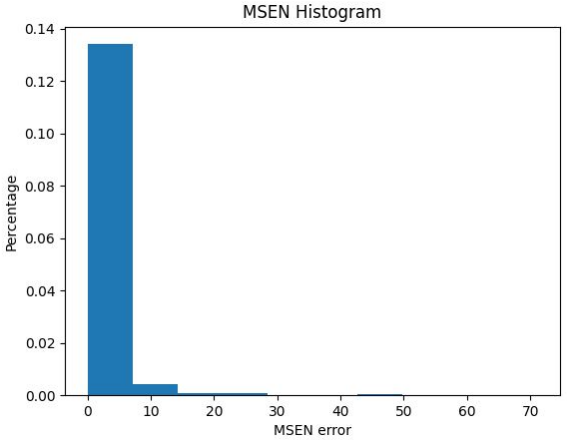
Optical flow with block size of 4

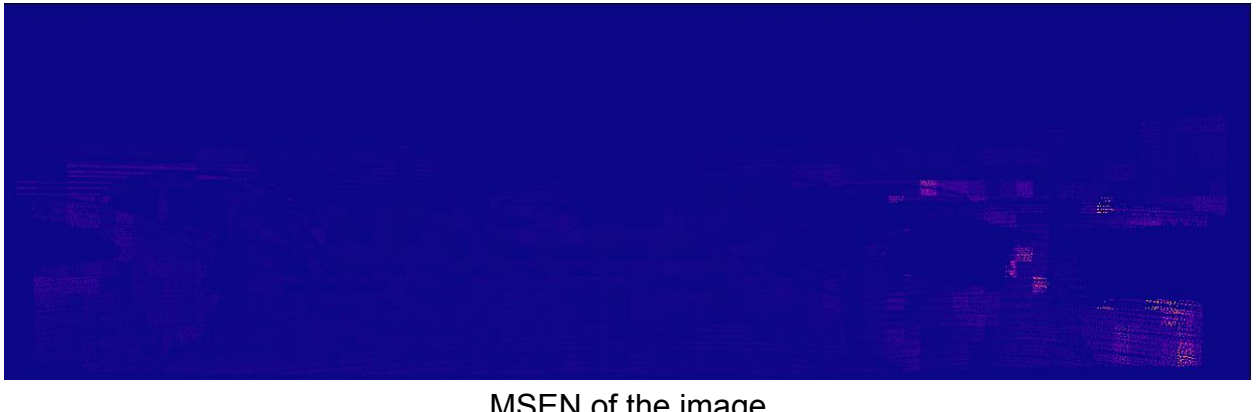
Optical flow with minimum MSEN (block size of 32)

Task 1.1: Optical flow with Block Matching (Team 3) [4/4]

Sequence 000045

The histogram shows that most pixels have a MSEN error under 6 pixels which is incredible knowing how simple it's the algorithm.
The MSEN error figure shows the error for each pixel.



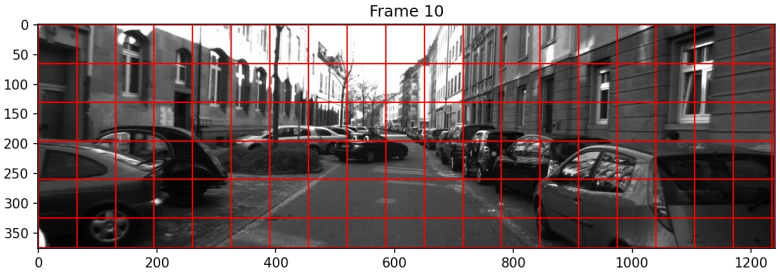


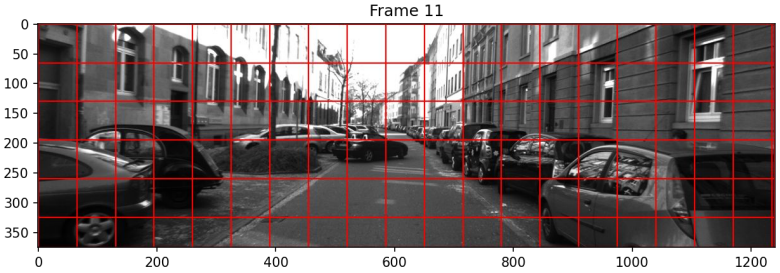
MSEN of the image

Task 1.1: Optical flow with Block Matching (Team 4) - [1/4]

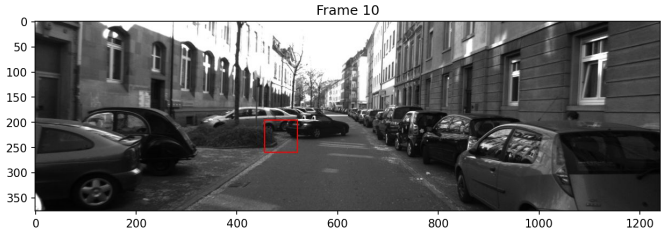
Block matching algorithm can be used for optical flow estimation to track the displacement of blocks of pixels across consecutive frames in a video sequence, following this steps:

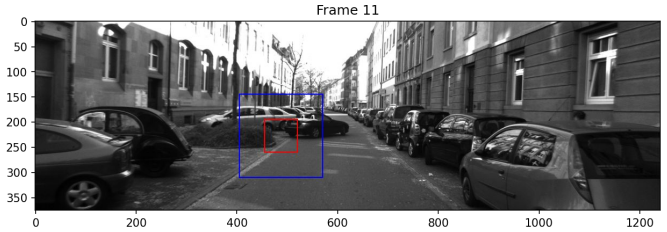
1. Divide the frame in small blocks:





2. Define an area of search around a block:





Task 1.1: Optical flow with Block Matching (Team 4) - [2/4]

Block matching algorithm can be used for optical flow estimation to track the displacement of blocks of pixels across consecutive frames in a video sequence, following this steps:

3. Compare each block of frame *t* to each block inside the search region of frame *t+1*, up to a step size to move the block through the search region.

We keep the most similar block after computing an error function (SAD, SSD, NCC).

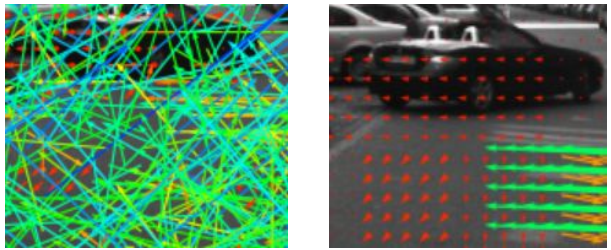
We perform the same for all the blocks of frame *t*, and then we compute the motion vector as the difference between the matched block in frame *t+1*, and the original block in frame *t* coordinates.

Task 1.1: Optical flow with Block Matching (Team 4) - [3/4]

We performed a Grid Search exploring this combinations of values:

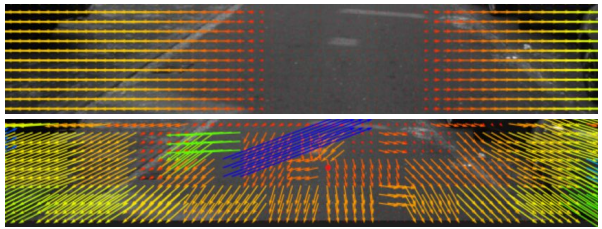
All explored values (**bold** indicates the best configuration after Grid Search)				Best configuration metrics	
Area of Search	Size of Blocks	Step size	Error function	Avg. PEPN	Avg. MSEN
10, **60**, 110	10, **50**, 90, 130, 170	1, **3**, 5, 7, 9	**NCC**, SAD, SSD	33.58%	6.56

Small block sizes produces incorrect motion vectors in most areas, due to the similar appearance of the blocks, using small and large search areas, getting smaller or larger motion vectors respectively, but incorrect anyway. That is why we get better metrics using a big block size but not too big.



Optical flow of a region of the center of the image using: Block size of 10 (left) and Block size of 50 (right). The other parameters are the ones in bold in the table.

Very small search areas, with small and large block sizes, produces incorrect values in most of the areas (top image). In the center it is not able to get a motion vector due to the similar texture of all the ground. In the sides the motion is incorrect because the area is too small to find a correct match. However, bigger areas of search (bottom image) produces a correct optical flow despite some problems in the center due to the similar texture of the ground.



Optical flow of the lowest part of the image using: Area of search of 30 (top) and Area of search of 60 (bottom). The other parameters are the ones in bold in the table.

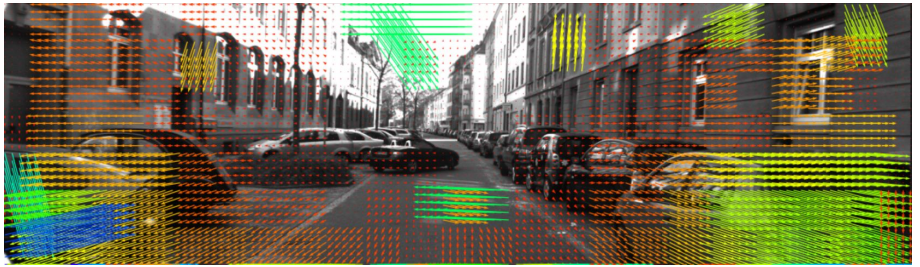
Task 1.1: Optical flow with Block Matching (Team 4) - [4/4]

We obtained better results than the ones obtained in Week 1 using the already provided optical flow values, for both metrics.

Approach	Metrics	
	Avg. PEPN	Avg. MSEN
Week 1: provided optical flow results	78.56%	10.63
Week 4: our block matching implementation	33.58%	6.56

Groundtruth optical flow

Provided result of optical flow used in Week 1 to compute the metrics



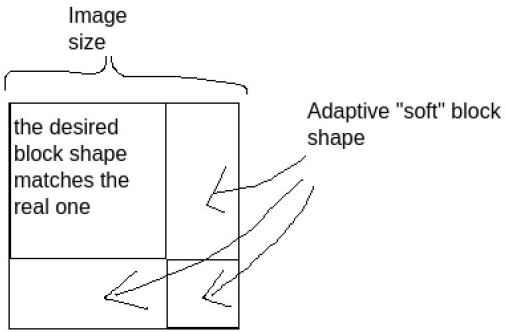
Result of optical flow using our Week 4 block matching implementation

We can see that the results using our approach are visually quite better than the ones provided in week 1, and are visually very similar to the groundtruth, except for some motion vectors due to miss-matched blocks. The main problems are in the texture-less areas like the sky, that produces some bad motion vectors due to the similarities between all blocks in that area. The best optical flow compared to the groundtruth is in the lowest part of the image, that contains more textured objects and the motion vectors of the lower part have almost the same direction than in the groundtruth, in contrast to the week 1 provided values, that were almost (0,0) incorrect vectors.

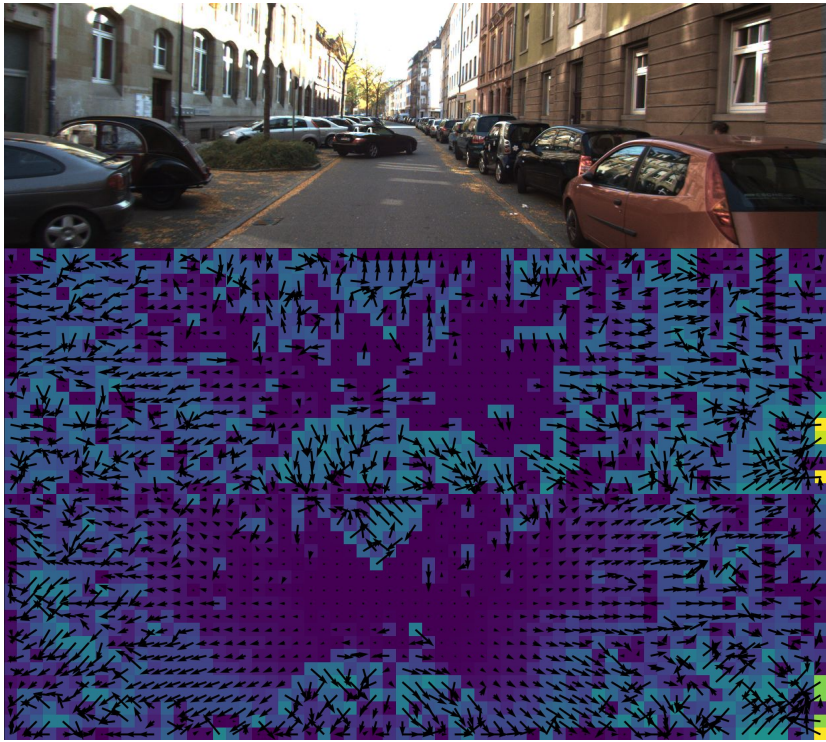
Task 1.1: Optical flow with Block Matching (Team 5) - [1/3]

We implemented the Exhaustive and Logarithmic Block Matching algorithm in Python from scratch, using only NumPy.

Our implementation incorporates “soft” block shape: you specify a desired block shape, and if such block shape is impossible at the current position – smaller block is returned instead.



The same logic is applied for search windows.



From top to bottom: reference frame, logarithmic search, exhaustive search output OF

Task 1.1: Optical flow with Block Matching (Team 5) - [2/3]

To access “candidates” for matching blocks within a search window we use

```
sliding = np.lib.stride_tricks.sliding_window_view(window, block.shape)
```

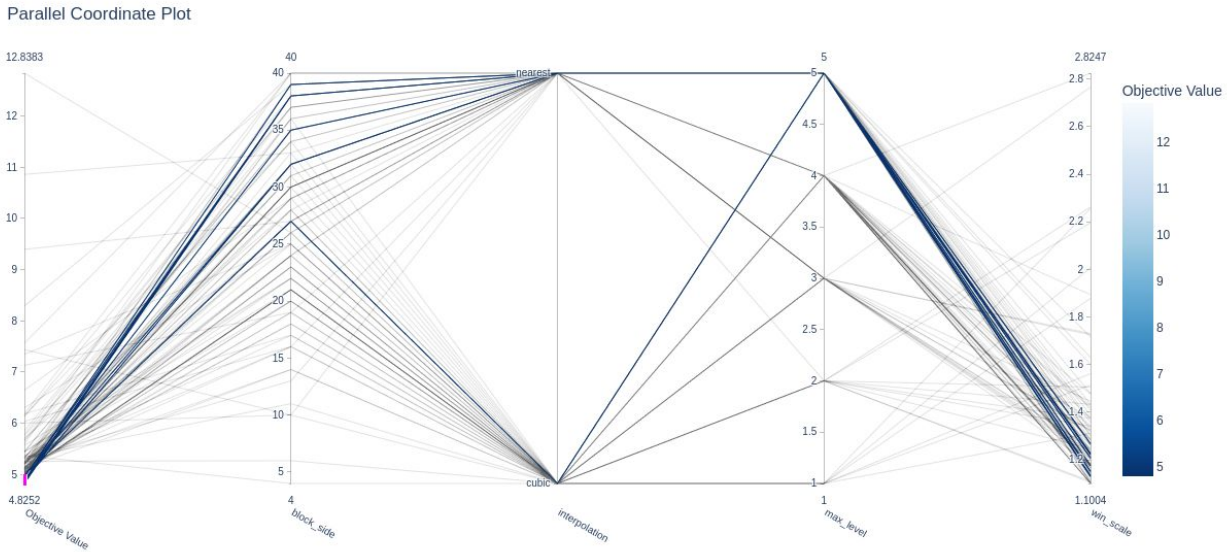
So instead of using inefficient python loops across the window, the job is done by effective NumPy function. Also, all the positions within a search window can be estimated with a single function call:

```
scores = metric_func(sliding, block)
```

In average, it takes 1.69 seconds for our implementation to calculate the OF between a pair of frames, using the exhaustive search, with block size=20, and window size=40 pixels. And **0.56 seconds** using the logarithmic search.

Task 1.1: Optical flow with Block Matching (Team 5) - [3/3]

We used Optuna to find best parameters for the block matching.



Usually, smaller (better) values of objective function are associated with

- bigger block size,
- cubic interpolation for exhaustive search, and nearest for log search

(interpolation is used when upscaling OF to the gt size)

- higher number of iterations in recursive (log) search
- and smaller window sizes*

*it is surprising that smaller search windows are associated with better results. One possible explanation for this, is that smaller windows suppress severe outliers, while preserving valid 1-3 pixel apparent movements

Task 1.1: Optical flow with Block Matching (Team 6) [1/4]

In this task we use Block matching for estimating the optical flow between consecutive frames. The algorithm can be summarized in the following steps:

1. Divide the first frame into small blocks of a fixed **block size** (NxN).
2. Define a **search area** around each block in the second frame where the matching block will be searched for. Note that, the search area should be small enough to avoid excessive computational costs, but large enough to accommodate the expected motion between the frames. Note that the **step** parameter has to be determined and refers to the distance between adjacent blocks in the search area.
3. For each block in the first frame, search for the best matching block in the search area of the second frame by comparing the pixel values or features of the blocks. We will test the following **similarity metrics:**
   a. Sum of absolute differences (SAD)
   b. Sum of squared differences (SSD) → simple and fast but sensitive to noise and illumination changes
   c. Normalized cross-correlation (NCC) → more robust to noise and illumination changes, but more computationally expensive
4. The displacement between the original block and the best matching block in the second frame represents the optical flow for that block.
5. Repeat steps 3-4 for all blocks in the first frame.
6. Optional: refine the optical flow field by applying post-processing techniques, such as Gaussian smoothing, median filtering, or interpolation, to remove noise and smooth out irregularities.

Note that it is a simple algorithm but many parameters need to be determined and they affect the resolution of the optical flow field. So, there is a tradeoff between the accuracy of the results and the computational efficiency of the algorithm.

We will use Optuna [1] framework to do the hyperparameter optimization with 100 trials and **TPE sampler** [2] with the following value suggestions:

- **block size** = [2, 4, 8, 16, 32, 64, 128]
- **search area** = [2, 4, 8, 16, 32, 64, 128]
- **step size** = [1,2, 4, 8, 16, 32]
- **similarity** = ['NCC', 'SAD', 'SSD']

To restrict the number of possible combinations we only computed combinations that satisfied:

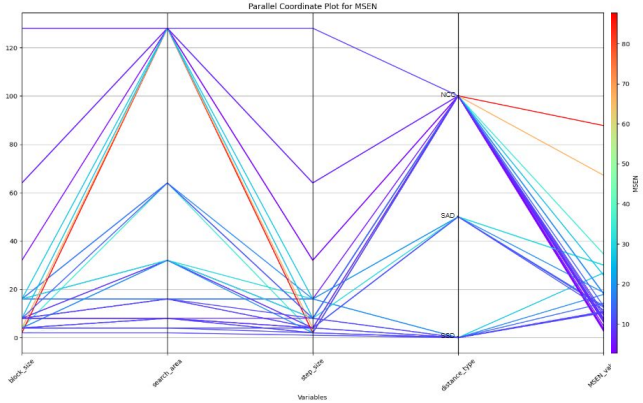
- `search_area` ≥ `block_size`
- `block_size/2` ≤ `step_size` ≤ `block_size`

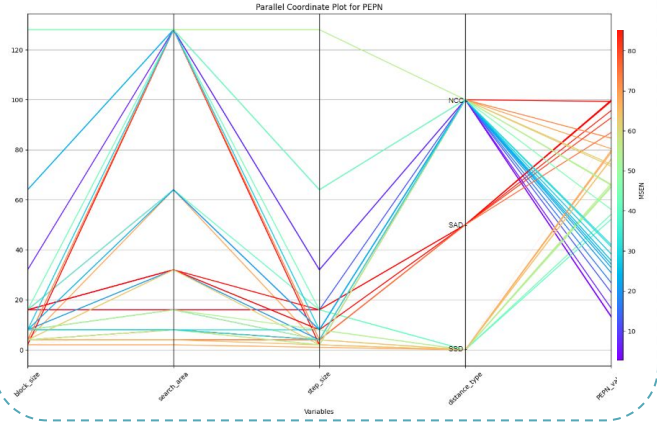
Best results

MSEN	PEPN	Runtime
2.71	13.67	0.76s

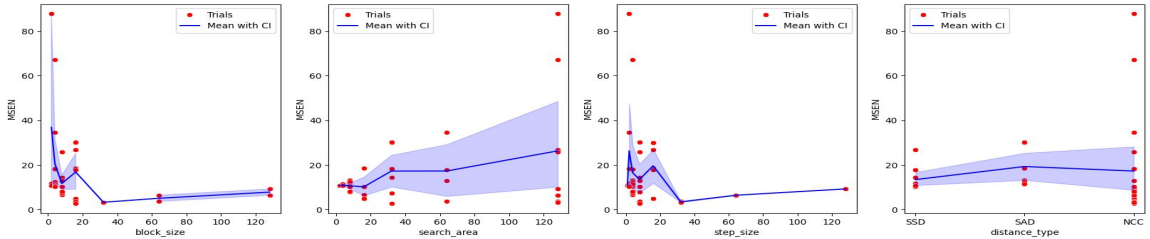
Task 1.1: Optical flow with Block Matching (Team 6) [2/4]

The following graphs show the parameters relationship and the final MSEN and PEPN. Note how the minimum errors are achieved with the combinations that use the NCC distance, which makes sense since we said it was more robust than SAD or SSD. We also see how having a really big search area compared to the block size and step size produces really high errors.



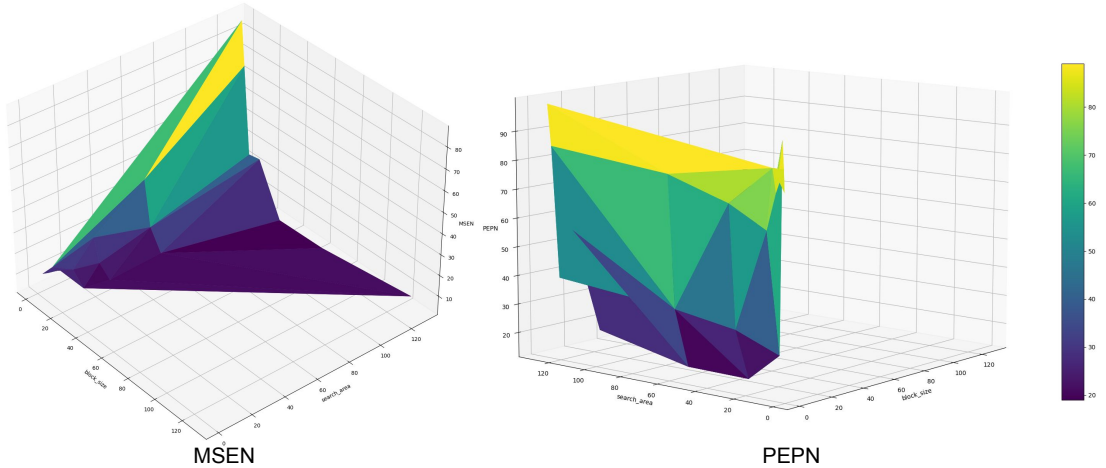


The following plots show separately the relationship of each of the parameters with the MSEN. In red we have the trails that used each parameter value and in blue we have the mean value of the trials for each value of the parameter and the confidence interval (+-standard deviation).



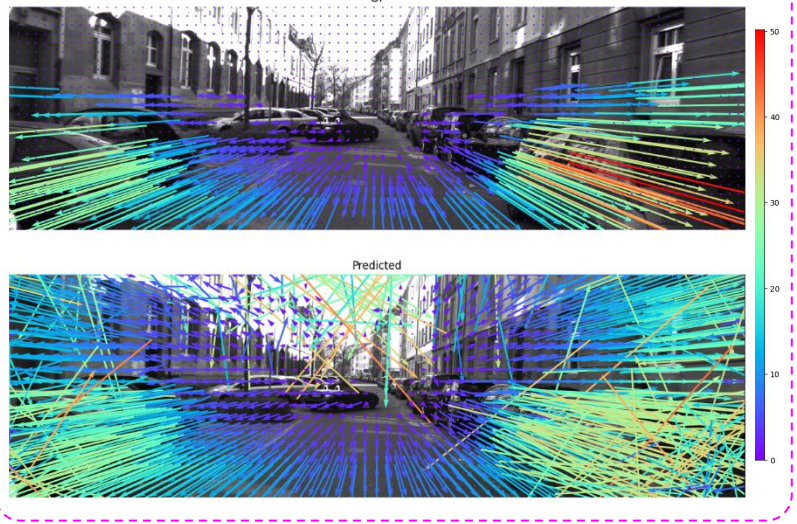
The following 3D surface plots show the MSEN and PEPN for each block size and search area used in the trials.

Note that we are using the mean values of all the trials that used those block size and search area values. Also, see that the axis are flipped for better visualization of the minimas of each 3D shape. We see how the best results are obtained with bock size 16 and search area 32.



Task 1.1: Optical flow with Block Matching (Team 6) [3/4]

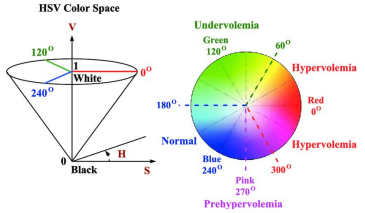
Sequence 45



Using the best results → **block size** = 16 **search area** = 32, **step size** = 8, distance type = **NCC**

As in the first week, we encode the Optical flow with the **quiver graph** and in the **HSV space** as follows:

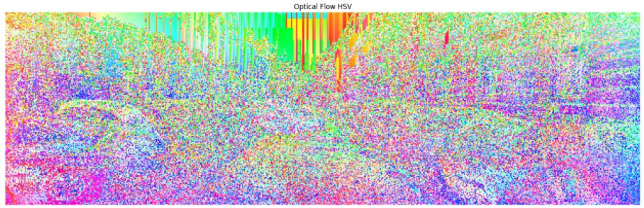
- **Direction of the optical** → Hue (color wheel)
- **Magnitude of the optical** → Saturation
- **Value** (1 [white]: max value, 0 [black]: min value)



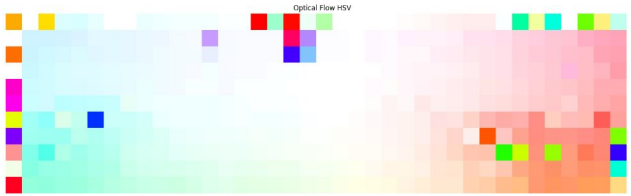
- We see that there the center of the image is quite similar to the GT. However, the sky produces many erroneous optical flow vectors since it is challenging to do block matching with textureless areas.
- Moreover, on the right of the image we see some vectors with wrong orientation maybe due to the reflections produced by the window of the car.
- Note also that we have many errors in the boundaries of the image. This is because the majority of this pixels are missing in the following frame, so it leads to inaccurate optical flow predictions.
- Overall, we can conclude that the results are accurate for the majority of the pixels, and we achieve better results than the ones provided in week 1.

Task 1.1: Optical flow with Block Matching (Team 6) [4/4]

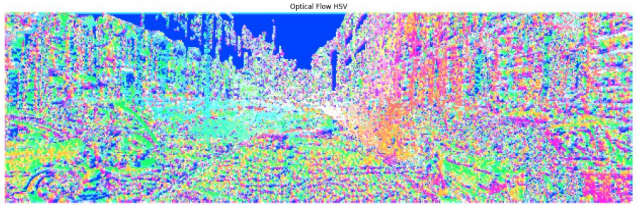
Finally, here we demonstrate visually how some of the parameters affect the results.



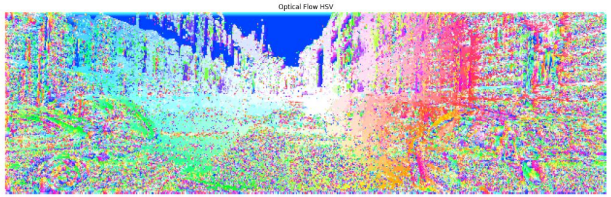
block size = 2, **search area** = 128, **step size** = 2



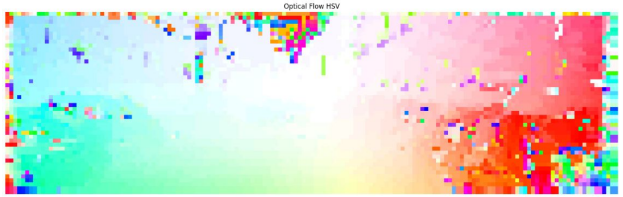
block size = 32 **search area** = 128, **step size** = 32



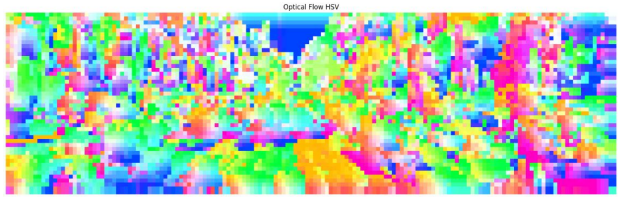
block size = 4, **search area** = 4, **step size** = 2



block size = 4, **search area** = 4, **step size** = 4



block size = 16 **search area** = 32, **step size** = 8 with **NCC**



block size = 16 **search area** = 32, **step size** = 8 with **SAD**

Resolution

Having a search area too big compared to the block size leads to noisy results, while having too big block sizes may produce results not accurate enough. Also, this affects the runtime, which is much longer with higher resolution.

In this case we see how a too small step size may lead to noisier results in some cases. However, when having a big block size it may be interesting to move the window with a step smaller than the block size to have intermediate points and increase the accuracy of the optical flow.

Similarity measure

The similarity distance used is also important. Here we see how using NCC instead of SAD leads to much better results.

Task 1.1: Optical flow with Block Matching (feedback)

Team ID	Feedback
Team 1	4 error functions tested. Clear explanations included. Not clear how parameter search is performed. Reported time, but you need to indicate machine otherwise is meaningless. Implementation used? Good analysis of max. search area. Gaussian smoothing used. No improvement.
Team 2	4 error functions tested. Clear explanations included. Optuna search for parameters. Why do you use estimation forward/backward? Implementation? What stride size did you use? Reported time, but you need to indicate machine otherwise is meaningless. Bonus point for exploring improvements.
Team 3	3 error functions tested. Clear explanations included. Grid search. Nice error figures! Reported time, but you need to indicate machine otherwise is meaningless. Implementation? Why best search area 192 if max motion is 32? (Reported by team1)
Team 4	3 error functions tested. Explanation of block matching included. Grid search for parameters. Implementation? What stride did you use? Good comparison with week1 metrics. Missing computational time report.
Team 5	Two implementations (from scratch): exhaustive and Logarithmic search. Good! Not clear which error functions tested (other than SSD) Parameter search with Optuna. Searched parameters not clear (figure is too small!) Reported time, but you need to indicate machine otherwise is meaningless. Step size == block side???
Team 6	3 error functions tested. Clear explanations included. Optuna search for parameters. Implementation? Missing computational time report.

Task 1.2: Off-the-shelf Optical Flow (all teams)

	MSEN		PEPN	
	PyFlow	Your other best	PyFlow	Your other best
Team 1	0.936	0.7334 (Perceiver IO)	7.429%	4.54% (Perceiver IO)
Team 2	1.00	0.2846 (MaskFlownet)	8.23%	0.5195% (Unimatch)
Team 3	0.936	0.724 (PWC-Net)	7.4 %	0.043 (PWC-Net)
Team 4				
Team 5	0.936	0.236 (GMFlow)	7.429%	0.31% (GMFlow)
Team 6	0.974	0.285 (MaskFlowNet)	7.991%	0.76% (MaskFlowNet)

* Images: 000045_{10, 11}.png

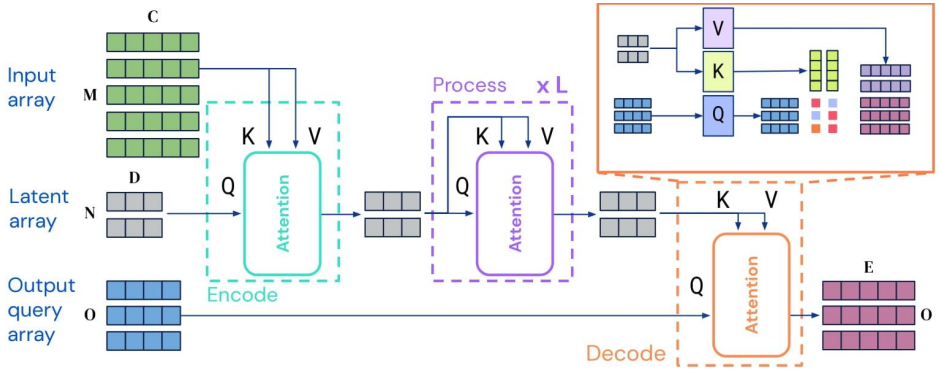
Task 1.2: Off-the-shelf Optical Flow (Team X) [max 3 slides]

Task 1.2: Off-the-shelf Optical Flow (Team 1) [1/3]

The off-the-shelf methods selected have been the Farneback method, Lucas Kanade, the one provided by Pyflow and finally the deep learning model provided by Perceiver IO [1].

Perceiver IO is a transformer encoder model that can be applied on any modality (text, images, audio, video, …).

This model was trained on AutoFlow, a synthetic dataset consisting of 400,000 annotated image pairs for Optical Flow.



Perceiver IO architecture.

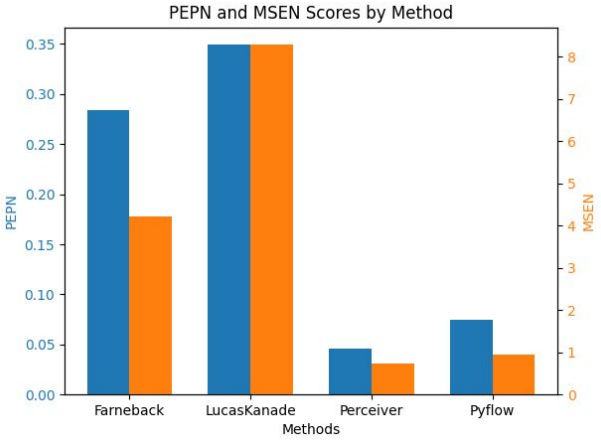
[1] https://arxiv.org/abs/2107.14795

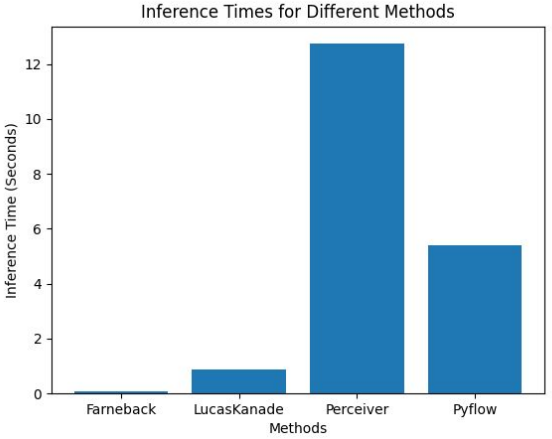
Task 1.2: Off-the-shelf Optical Flow (Team 1) [2/3]

Using the metrics developed in Week1 we have evaluated every model.

It has been observed that the most classical methods like Farneback and Lucas Kanade are obviously the ones producing worse results.

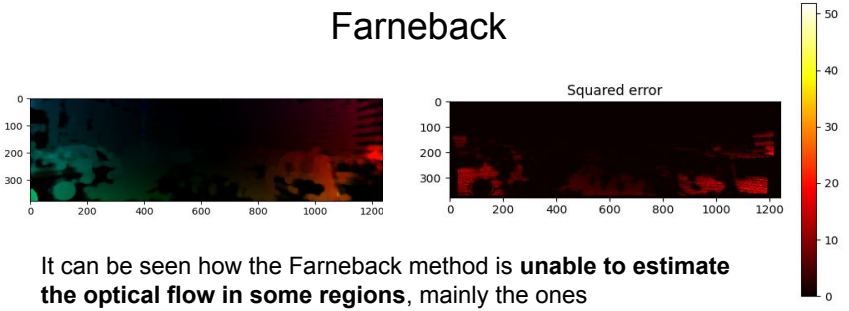
While the most recent ones perform better, and the transformer model from Perceiver IO being the best one with a **PEPN=4.54%** and **MSEN=0.7334**.
However, such model is also the one that takes the most time at inference (results are extracted using a Nvidia GTX 1660).





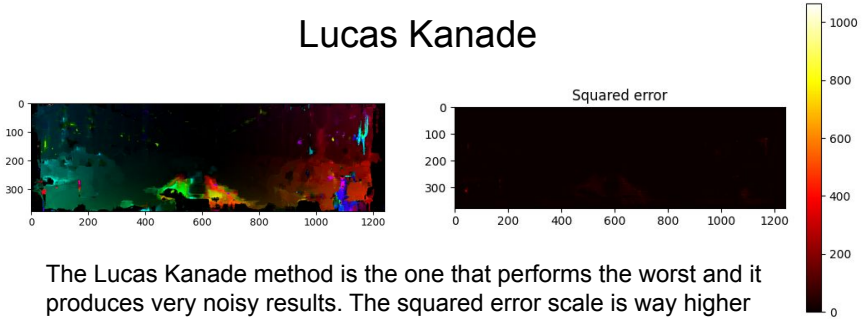
Task 1.2: Off-the-shelf Optical Flow (Team 1) [3/3]

Farneback



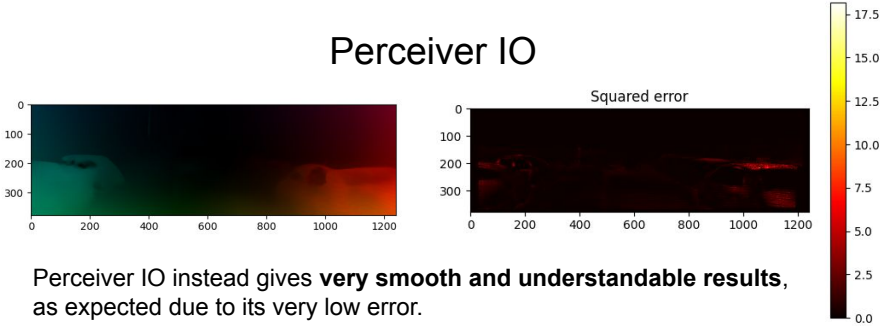
It can be seen how the Farneback method is **unable to estimate the optical flow in some regions**, mainly the ones **edge/corner-less areas**.

Lucas Kanade



The Lucas Kanade method is the one that performs the worst and it produces very noisy results. The squared error scale is way higher compared to the rest of plots, that is why there are **less reddish pixels** but the **error is higher**.

Perceiver IO



Perceiver IO instead gives **very smooth and understandable results**, as expected due to its very low error.

Task 1.2: Off-the-shelf Optical Flow (Team 2) [1/3]

In this section, we test the method introduced in *High Accuracy Optical Flow Estimation Based on a Theory for Warping*[1] and implemented in the *PyFlow* library. This method works by minimizing a functional with the following constraints: Grey value constancy assumption, Gradient constancy assumption, Smoothness assumption and Multiscale approach (for more details, please check the paper). The final functional can be found below this text.

$$E_{Data}(u,v) = \int_\Omega \Psi\left(|I(\mathbf{x}+\mathbf{w}) - I(\mathbf{x})|^2 + \gamma|\nabla I(\mathbf{x}+\mathbf{w}) - \nabla I(\mathbf{x})|^2\right) \mathbf{dx}$$

$$E_{Smooth}(u,v) = \int_\Omega \Psi\left(|\nabla_3 u|^2 + |\nabla_3 v|^2\right) \mathbf{dx}.$$

$$E(u,v) = E_{Data} + \alpha E_{Smooth}$$

*Check speaker notes for references.

We conduct an Optuna Search over different values of its parameters. This time, we minimize the MSEN and time, as we want to keep the execution time at bay.

Parameters searched:
alpha = [0.001, 0.5]
ratio = [0.5, 0.9]
minWidth = [10, 50]
nOuterFPIterations = [1, 10]
nInnerFPIterations = [1, 10]]
nSORIterations = [1, 40]

Best parameters:
alpha = 0.008
ratio = 0.698
minWidth = 12
nOuterFPIterations = 6
nInnerFPIterations = 7
nSORIterations = 28

Parameter explanation:
alpha: the regularization weight
ratio: the downsample ratio
minWidth: the width of the coarsest level
nOuterFPIterations: the number of outer fixed point iterations
nInnerFPIterations: the number of inner fixed point iterations
nSORIterations: the number of SOR iterations

Qualitatively, we obtain smoother results than with Block Matching, although we get similar metrics. This method still struggles with specularities but improves on flat areas. However, there seem to be some issues with repetitive patterns. Timewise, this method is not worth it.

Block Matching - 0.492s, 1.06MSEN**, 7.03PEPN**

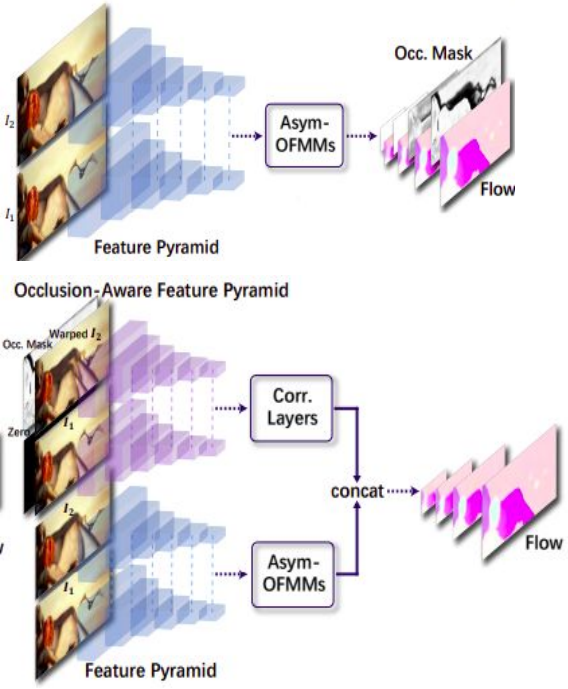
PyFlow - 23.527s, **1.00MSEN,** 8.23PEPN

Task 1.2: Off-the-shelf Optical Flow (Team 2) [2/3]

MaskFlownet

MaskFlownet: Asymmetric Feature Matching with Learnable Occlusion Mask. ***Zhao, Shen, et al.***

Code - Paper



Feature warping followed by a correlation layer is the common practice to compute the cost volume for non-local feature matching.

When aligning (warping) an image and the subsequent corrected frame, **occlusions induce ambiguity**.

MaskFlownet is a two-stage net. Right before the 2nd stage, an **occlusion mask** for occluded areas is **learned** in an unsupervised way. This **reduces the ambiguity** when warping

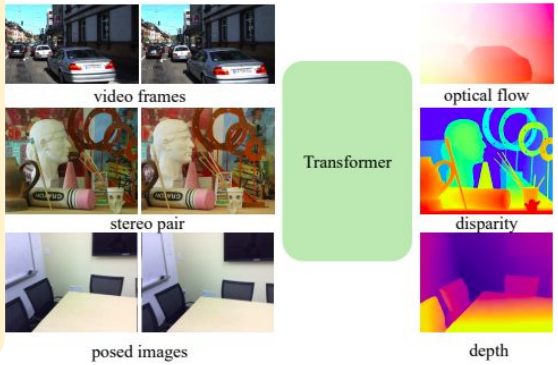
UniMatch

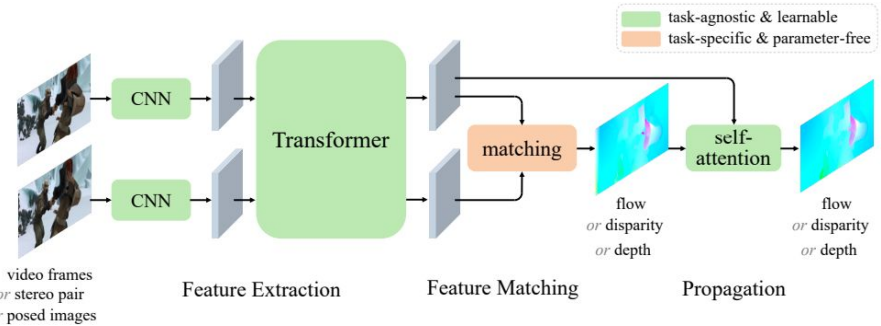
Unifying Flow, Stereo and Depth Estimation. ***Xu, Zhang, et al.***

Code - Paper

Unified dense correspondence matching approach for **jointly** solving optical flow, rectified stereo, and depth estimation.

Generate discriminative **features** with a **task-agnostic** Transformer. Then **match** features in a **task-specific** manner. Finally, **refine** through self-attention.





Task 1.2: Off-the-shelf Optical Flow (Team 2) [3/3]

MaskFlownet

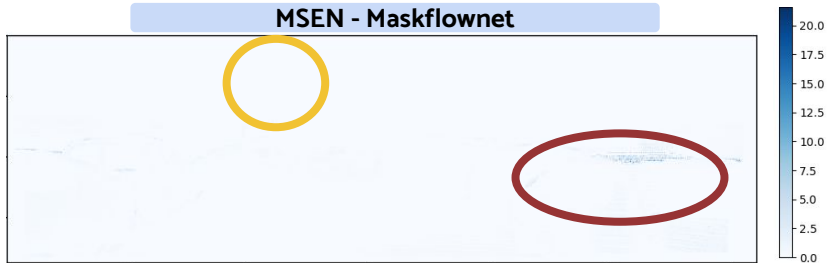
The per-pixel error and estimated motion (right) are highest on the left and right sides of the image. This is probably due to the fact that cars parked on the sideway occlude one another (introducing ambiguity), and due to the parallax effect, whereby objects to the sides have higher apparent velocity.

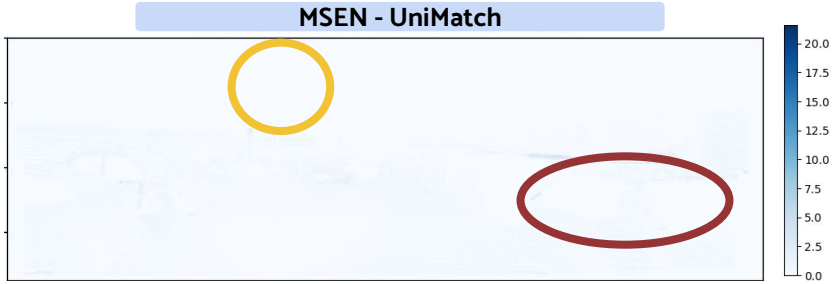
UniMatch

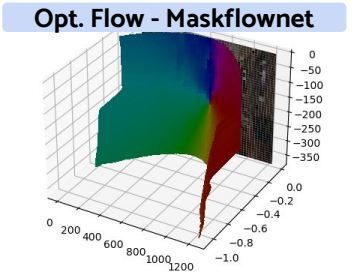
The optical flow plot and the error plot for UniMatch look very much like MaskFlownet's. This indicates that SOTA methods are actually producing very similar results, and the improvement must come from cases where there are occlusions (in this example, the cars occlude one another), the parallax effect...

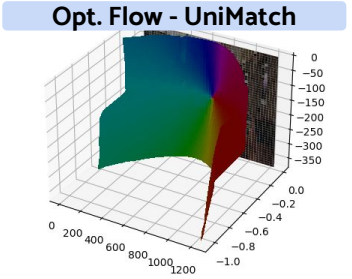
Both UniMatch and MaskFlownet can resolve **specularities** and **reflections** much better than block matching and PyFlow. We can see that now the error on the window and roof (red circle in the plots) of one of the cars in the righ-hand side is basically null. Nevertheless, MaskFlownet still shows some error on the car's roof.

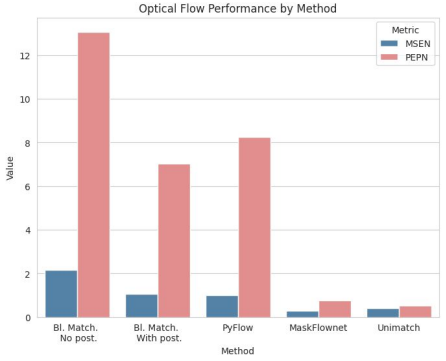
Moreove, **textureless** areas such as the sky (yellow circle) are now handled properly, unlike in the case of PyFlow.

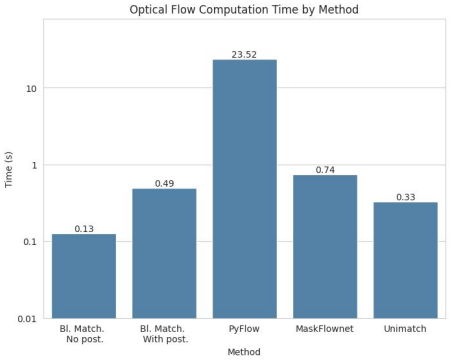












Clearly, both MaskFlownet and Unimatch are much **faster** than PyFlow, and even faster than Block Matching with postprocessing.

Current SOTA methods outperform classic ones both in terms of accuracy (left bar chart) and run time (right bar chart).

Task 1.2: Off-the-shelf Optical Flow (Team 3) [1/3]

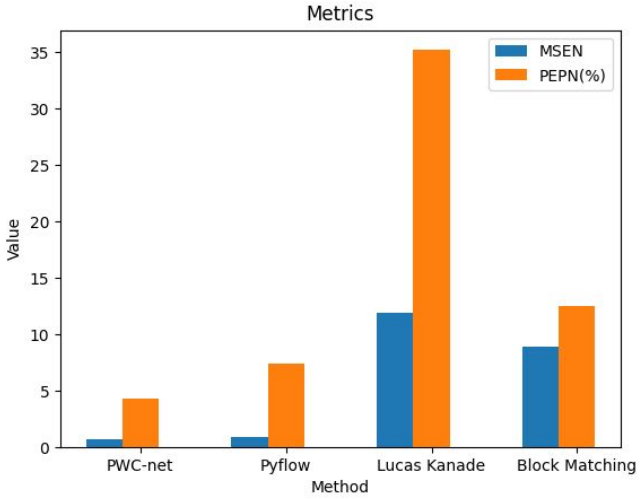
We have work with three different optical flow implementations to compute the motion in the consecutive scenes.

- **PyFlow**: This is super fast and accurate optical flow implementation based on Coarse2Fine method from Thomas Brox[1] work. This is an optimization based algorithm that minimize the functional energy to perform the estimation. It has three main assumptions:
  - Brightness constancy.
  - Gradient constancy.
  - Spatio-temporal smoothness.
- **Lucas-Kanade**: We have used the OpenCV implementation of the iterative Lucas-Kanade method with pyramids [2]. The OpenCV implementation is the sparse one, to make it dense we have build a matrix of interest point to match each pixel.
- **PWC-Net**: PWC-Net stands for "Pyramid, Warping, and Cost Volume Network"[3]. The architecture is designed to be computationally efficient while achieving high accuracy in optical flow estimation.The main idea behind PWC-Net is to use a multi-scale approach to capture both global and local motion in the scene. This is achieved by building a pyramid of image scales and computing the optical flow at each level of the pyramid.
  The implementation is based on the popular tensorflow library

[1] Brox, Thomas, et al. "High accuracy optical flow estimation based on a theory for warping." European conference on computer vision. Springer, Berlin, Heidelberg, 2004.
[2] Bouguet, Jean-Yves. "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm." Intel Corporation 5.1-10,2001.
[3] Deqing Sun et al, "PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume,"(CVPR 2018)

Task 1.2: Off-the-shelf Optical Flow (Team 3) [2/3]

	PWC-Net	PyFlow	Lucas-Kanade	Block Matching
MSEN	**0.724**	0.936	11.963	8.490
PEPN	**0.043**	0.074	0.352	0.125
Runtime (seg)	7.07*	4.34	0.46	**0.388**



- The table shows that **the best algorithm** of the three exposes **is the PWC-net being nearly 25 % better in the MSEN and 40 %** in PEPN(0.5 threshold) better than the second best algorithm, the Pyflow.
- We can also see that the fast algorithm is the Block Matching, however its score is really low. PWC-net could run faster on a GPU.
- We have also seen that our block matching approach with grid search perform better and runs faster than the Lucas-Kanade ones, although is really far from the learning methods.

1. PWC-Net run on CPU

Task 1.2: Off-the-shelf Optical Flow (Team 3) [3/3]

Direction index

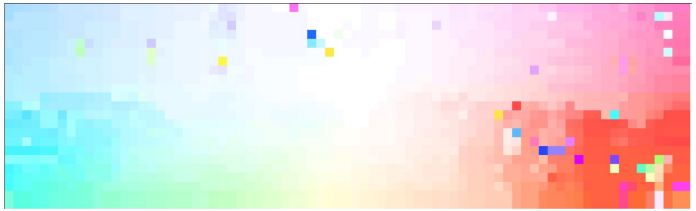
Pyflow and PWC-NET show smooth transition between neighbors, this can be seen in the first column plots where smother color maps are shown. In the block matching approach there heavy changes at some points showing error in the optical flow computation. They are especially seen in the bottom-right corner of the image where parts of the car become occluded between frames. In the Lukas-Kanade case the algorithm nearly no detect the movement. There are areas where no optical flow is observed. Accordingly to the quantitative results shown in the previous slide where the Kanade Algorithm score really low.

PWC-NET

Lukas-Kanade

Pyflow

Block matching



Task 1.2: Off-the-shelf Optical Flow (Team 4) (1/3)

Overview of the libraries used

We have evaluated the following optical flow estimators:

Pyflow [1]: PyFlow is a Python library that implements the Horn and Schunck method for optical flow estimation. It utilizes a variational approach, minimizing an energy functional that accounts for brightness constancy and flow field smoothness. PyFlow employs an iterative optimization process, utilizing numerical methods like Gauss-Seidel or Jacobi to solve partial differential equations (PDEs), and a pyramidal approach with multiple resolution levels to handle large displacements and varying motion scales in image sequences

GMFlow [2]: GMFlow presents a unified formulation and model for three motion and 3D perception tasks, including optical flow, rectified stereo matching, and unrectified stereo depth estimation from posed images. Unlike task-specific architectures, all three tasks are formulated as a unified dense correspondence matching problem, which is solved with a single model by directly comparing feature similarities. Discriminative feature representations are achieved using a Transformer, specifically the cross-attention mechanism, which allows for integration of knowledge from another image via cross-view interactions and improves feature quality. The model enables cross-task transfer as the model architecture and parameters are shared across tasks.

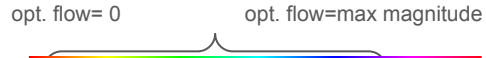
RAFT [3]: Recurrent All-Pairs Field Transforms (RAFT) for optical flow estimation extracts per-pixel features and builds multi-scale 4D correlation volumes for all pairs of pixels. It then iteratively updates a flow field through a recurrent unit that performs lookups on the correlation volumes.

MaskFlownet [4]: MaskFlownet is an asymmetric occlusion-aware feature matching module that learns a rough occlusion mask to filter out occluded areas immediately after feature warping, without the need for explicit supervision. This module can be seamlessly integrated into end-to-end network architectures, boosting performance with minimal computational overhead.The learned occlusion mask is then used in a subsequent network cascade with dual feature pyramids.

[1] C. Liu. "Beyond Pixels: Exploring New Representations and Applications for Motion Analysis." Doctoral Thesis, Massachusetts Institute of Technology (2009)
Implementation (python wrapper of C. Liu's implementation)
[2] Xu, Haofei, Jing Zhang, Jianfei Cai, Hamid Rezatofighi, Fisher Yu, Dacheng Tao, and Andreas Geiger. "Unifying Flow, Stereo and Depth Estimation." arXiv preprint arXiv:2211.05783 (2022). Implementation (official repository)
[3] Teed, Zachary, and Jia Deng. "RAFT: Recurrent All Pairs Field Transforms for Optical Flow." ECCV (2020) Implementation (official repository)
[4] Zhao, Shengyu, Yilun Sheng, Yue Dong, Eric I-Chao Chang, and Yan Xu. "MaskFlownet: Asymmetric Feature Matching with Learnable Occlusion Mask." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2020)
Implementation (official repository)

Task 1.2: Off-the-shelf Optical Flow (Team 4) (2/3)

Quiver plot interpretation



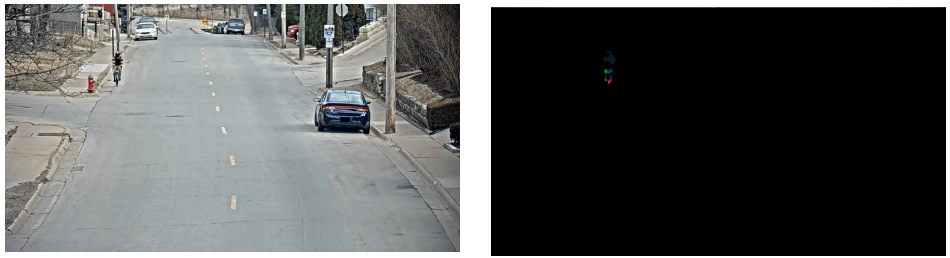
[image source]

- Direction represented by the arrow angle
- Magnitude represented by the arrow length and colour

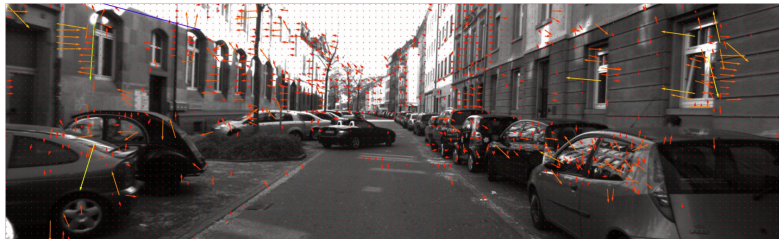
Pyflow vs provided Lucas Kanade (week 1)

Unlike the results provided on the first week, Pyflow manages to obtain an estimate that is much closer to the ground truth as we can see in the examples with image 000045_10.png. The camera motion matches the one of the ground truth: small on the background pixels increasing with the closest ones. We can observe that there is still room for improvement as some of the arrows got inaccurate angles, especially the ones located on the foreground cars. Furthermore, the areas where the pavement is have an optical flow that is way smaller than the ground truth one.

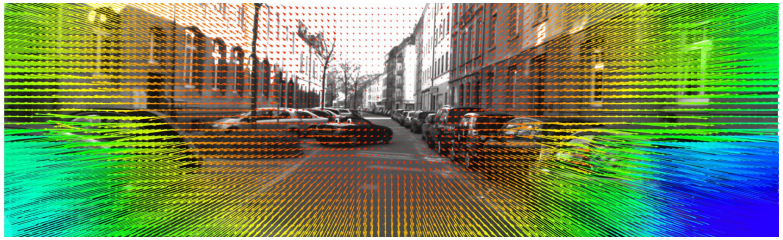
Below we can see how it performs in video. Pyflow is able to capture really fine details, as even the pedalling motions of the cyclist have been represented properly in the optical flow estimate.



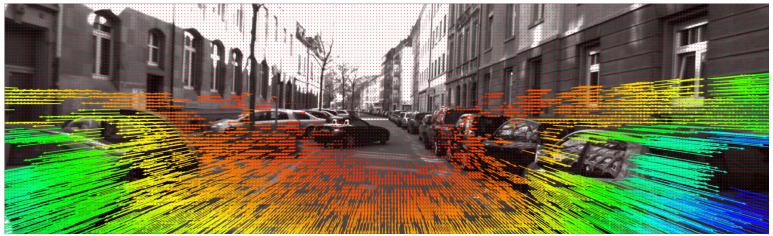
Pyflow estimate on AICity_data video (train/s03/c010)



Provided week 1 estimate on '000045_10.png'



Pyflow estimate on '000045_10.png'



Ground truth of '000045_10.png'

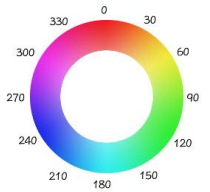
Task 1.2: Off-the-shelf Optical Flow (Team 4) (3/3)

Off-the-shelf methods comparison

We can see that more recent methods manage to surpass the Pyflow results. In particular, with the image 000045_10.png, the best performing method is GMFlow. We can see that, qualitatively, its optical flow prediction is also the smoothest one. MaskFlownet and GMFlow get really similar results. However, visually, MaskFlownet has slightly rougher edges of the car silhouettes.

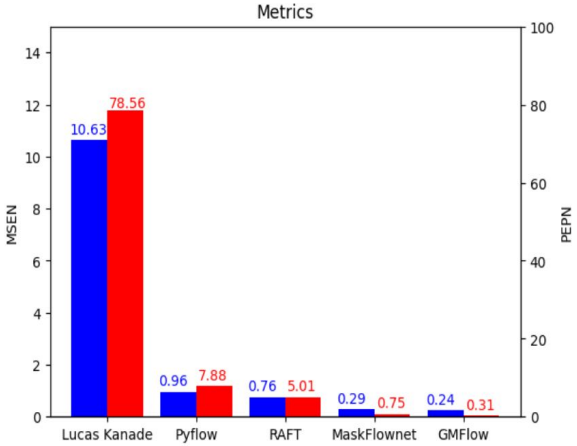
An observation is that some of the approaches barely predict any optical flow in the windows of the cars (Pyflow, RAFT), whereas MaskFlownet and GMFlow do predict the optical flow of the entire car. This is not something that affects the MSEN/PEPN results, as, in particular, the ground truth does not consider the windows as valid regions. However, we believe that it makes more sense to predict it, making GMflow and MaskFlownet more accurate.

Plots of this slide represent direction and magnitude of the optical flow using the following:



angle [image source]

normalized magnitude [image source]



MSEN and **PEPN** of the evaluated approaches

Pyflow estimate

MaskFlownet estimate

GMflow estimate

RAFT estimate

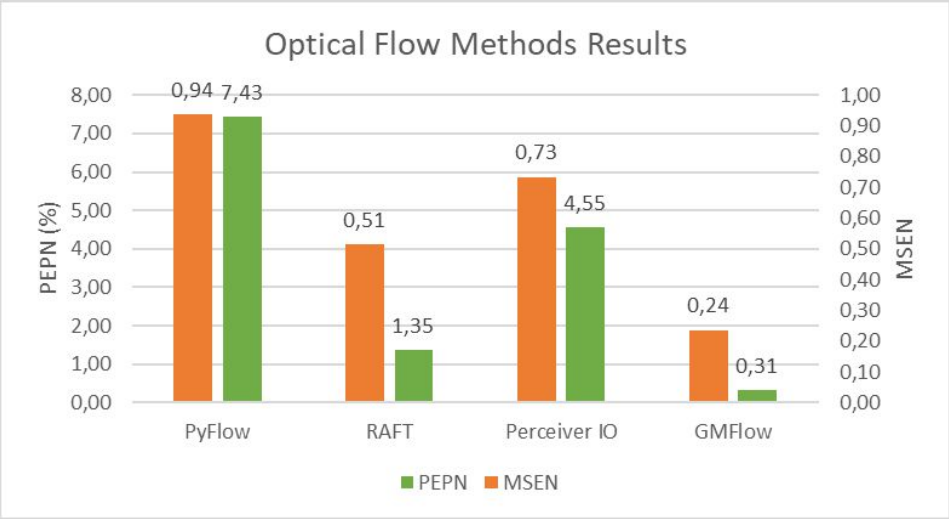
Task 1.2: Off-the-shelf Optical Flow (Team 5) [1/3]

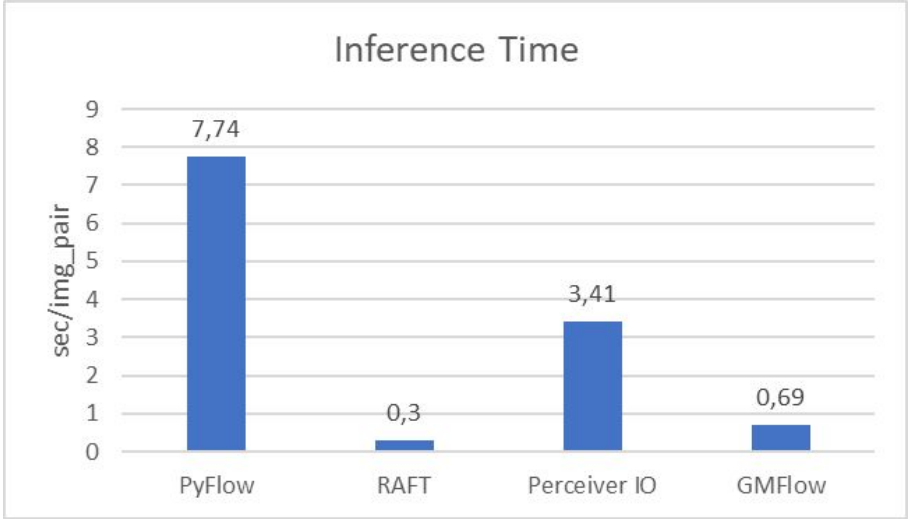
Off-the-shelf methods used to estimate the optical flow:

- Classic:
  - **Pyflow (2009) (reference, implementation):** Coarse2Fine optical flow method, which is a classic optical flow method. It does not use neural networks. The method is based on the iterative refinement of an initial optical flow estimate using a coarse-to-fine approach. At each scale, the method uses the Lucas-Kanade method to estimate the optical flow field and refines it using a smoothing filter. The process is repeated at each scale until the finest resolution is reached.
- Modern:
  - **RAFT (2020) (reference, implementation):** RAFT (Recurrent All-Pairs Field Transforms) is a modern optical flow method that uses a neural network to estimate the optical flow field. It is a recurrent neural network that processes pairs of images iteratively and predicts the optical flow field between them. The network uses a multi-scale approach to estimate the optical flow field at different resolutions and a correlation layer to aggregate information from the pairs of images.
  - **Perceiver IO (2021) (reference, implementation):** Perceiver IO is a recent optical flow method that uses a neural network with an attention mechanism to estimate the optical flow field. The method uses a multi-scale approach and estimates the optical flow field at different resolutions. At each scale, the method uses a convolutional neural network with a spatial attention mechanism to weight the contribution of each pixel to the optical flow estimation. The attention mechanism allows the method to handle complex motion patterns, such as non-rigid deformations and rotations.
  - **GMFlow (2022) (reference, implementation):** The presented method is a unified model that can solve three motion and 3D perception tasks: optical flow, rectified stereo matching, and unrectified stereo depth estimation. The method uses a single model to solve the three tasks by directly comparing feature similarities. Discriminative feature representations are achieved using a Transformer with cross-attention mechanism, which enables integration of knowledge from another image via cross-view interactions. (Top 1 Sintel benchmark)

Task 1.2: Off-the-shelf Optical Flow (Team 5) [2/3]

- Different parameter values for the PyFlow method were tested, but the default values provided the best results.
- All methods were tested using the original image resolution of 1241x376.
- The network-based methods (RAFT, Perceiver IO, GMFlow) were tested using pre-trained weights from KITTI or Sintel.
- The methods were executed on a RTX 3060.

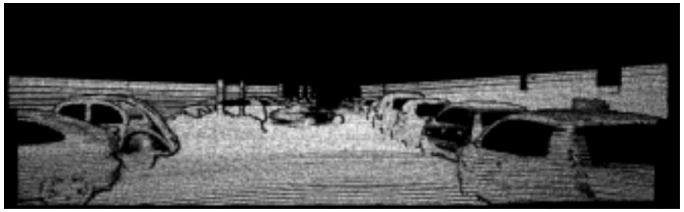




- All modern methods outperformed PyFlow in terms of both results and inference time.
- The best results were obtained using GMFlow, followed by RAFT.
- Although RAFT had a faster inference time, GMFlow achieved better results in a competitive inference time.
- Perceiver IO was the worst-performing method in terms of both results and inference time among the modern methods.

Task 1.2: Off-the-shelf Optical Flow (Team 5) [3/3]

000045_{10, 11}.png

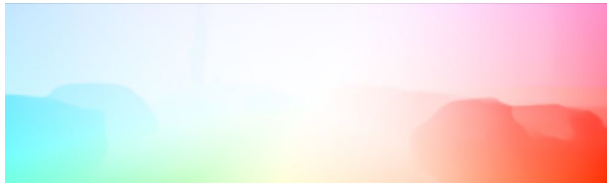


Valid pixels

Not valid

GT

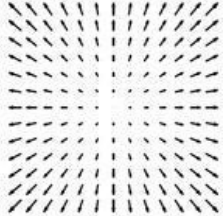
PyFlow



RAFT

Perceiver IO

GMFlow



- Qualitative results show that the optical flow estimated by PyFlow does not appear to be as smooth as other methods, as can be observed in the optical flow of the car in the bottom right corner.
- GMFlow produces the best optical flow results, as seen in the qualitative evaluation. It accurately predicts the shapes of all objects and their optical flow, even in areas that are not valid, such as car windows or building facades. No other method could replicate this level of performance.

Task 1.2: Off-the-shelf Optical Flow (Team 6) [1/3]

Classical methods

Lucas-Kanade [1]: It assumes that the motion is small and constant in a local neighborhood of pixels, and uses this assumption to solve a system of linear equations to estimate the optical flow.

Pyflow [2]: is a Python implementation of the Coarse2Fine Optical Flow algorithm based on the work of Chen et al. and it uses a pyramidal approach to estimate the flow at multiple scales, allowing for better accuracy and robustness to large displacements.

[1] [2001] Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. Intel Corporation, 5, 2001.
[2] [CVPR 2017] Unsupervised deep learning using unlabelled videos on the web
[3] [CVPR 2018] LiteFlowNet: A Lightweight Convolutional Neural Network for Optical Flow Estimation
[4] [CVPR 2020, Oral] MaskFlownet: Asymmetric Feature Matching with Learnable Occlusion Mask
[5] [ECCV 2020] RAFT: Recurrent All Pairs Field Transforms for Optical Flow
[6] [ICLR 2022] Perceiver IO: A General Architecture for Structured Inputs & Outputs
[7] [CVPR 2022] Deep Equilibrium Optical Flow Estimation

Learning-based methods

LiteFlowNet [3]: lightweight version of the FlowNet deep learning architecture. It uses a compact network architecture and reduces the computational cost of the FlowNet architecture while maintaining similar performance.

MaskFlowNet [4]: deep learning-based algorithm for dense optical flow estimation in videos with occluded or partially visible objects, using semantic segmentation masks to improve accuracy in such regions. It uses a modified version of the FlowNet architecture with an additional occlusion mask branch that predicts occlusion masks indicating which pixels in the image are occluded or partially visible

RAFT [5]: (Recurrent All-Pairs Field Transforms) is a deep learning-based algorithm for estimating dense optical flow in videos. It uses a recurrent neural network architecture to iteratively refine the flow estimation.

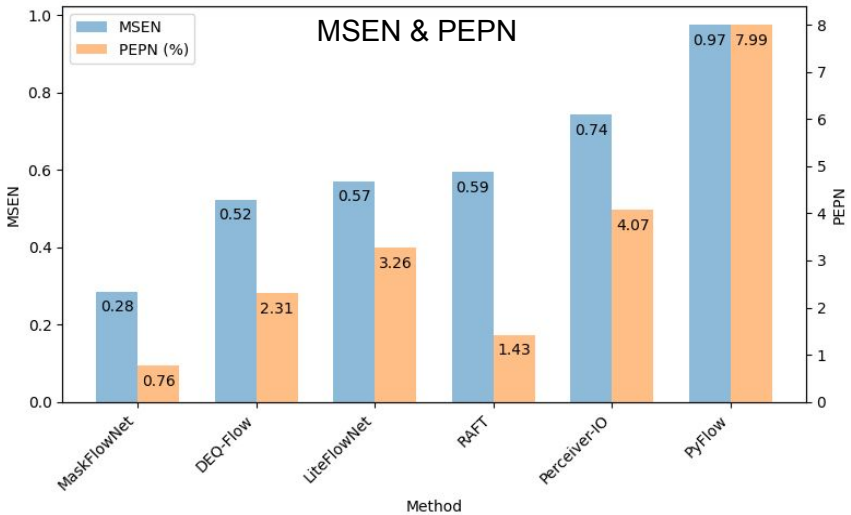
Perceiver-io [6]: deep learning architecture that combines the Perceiver and Transformer models to enable end-to-end dense optical flow estimation. It can process high-dimensional inputs and has shown promising results on optical flow benchmarks. PerceiverIO can be trained in a single forward pass, making it computationally efficient.

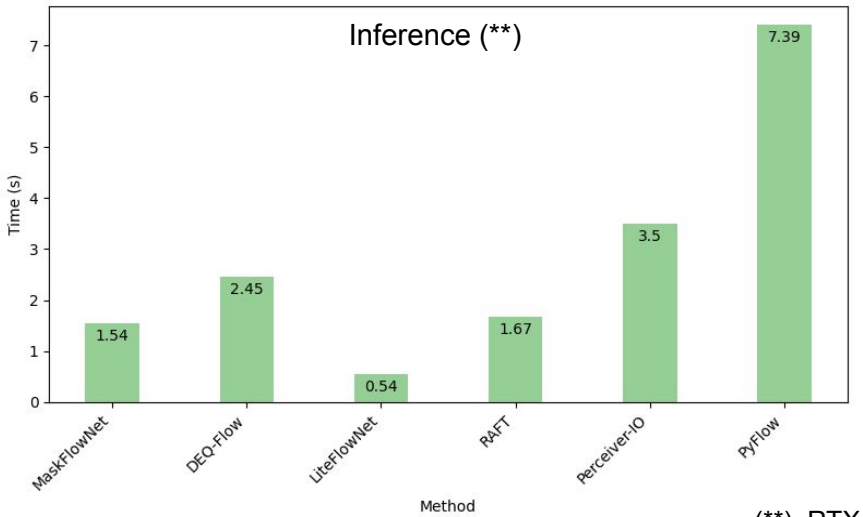
DEQ-Flow [7]: deep learning method that combines a hierarchical coarse-to-fine approach with a differential equation solver. It uses a hierarchical network architecture to estimate the flow at multiple scales and uses a differential equation solver to refine the flow estimates.

Task 1.2: Off-the-shelf Optical Flow (Team 6) [2/3]

We implemented all the previously explained methods to compute the Optical Flow of the two frames (*) belonging to the Kitti dataset. We show all metrics results (MSEN and PEPN) and the inference time for all methods except for Lucas-Kanade (LK). The values obtained for MSEN and PEPN by LK were significantly higher than the other methods, distorting the histogram and making it difficult to compare between the different techniques. This is likely because LK is a very basic and simple method and its assumptions may not fit in real-world scenarios.

- MaskFlowNet method outperformed all other methods with the lowest MSEN and PEPN values, indicating better accuracy, and the second-fastest runtime.
- MSEN results for DEQ-Flow, LiteFlowNet, and RAFT are very similar, but RAFT achieves better results in terms of PEPN.
- From these first four methods, the fastest is LiteFlowNet, which achieves reasonably good accuracy.
- Perceiver-IO and PyFlow are the slowest methods and obtain worse results. These results may be because the Perceiver-IO method requires a large number of parameters, and the PyFlow method is based on an iterative algorithm that may converge slowly, leading to reduced accuracy.





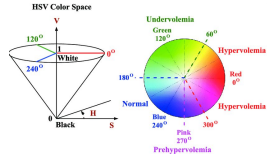
(*) Images: 000045_{10, 11}.png

(**) RTX 3090

Task 1.2: Off-the-shelf Optical Flow (Team 6) [3/3]

GT

MaskFlowNet



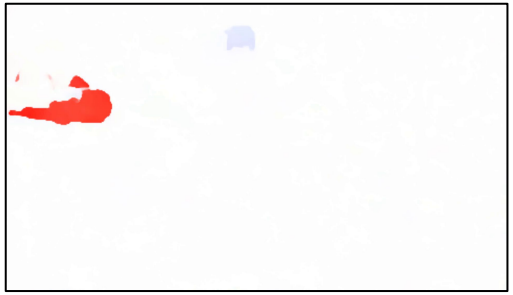
PyFlow

Here we present the visual results of the top-performing methods from the previous slide, along with PyFlow. Notably, PyFlow and LiteFlowNet can accurately detect the car windows, which is a challenging task due to their reflective nature. Furthermore, the deep learning-based methods exhibit smoother results, with flow magnitudes very similar to the ground truth.

RAFT

LiteFlowNet

Here we present the results of applying **MaskFlowNet** to one subset of the AICity Dataset. The performance of this method in estimating optical flow of partially occluded objects, such as the first bicycle, is noteworthy.

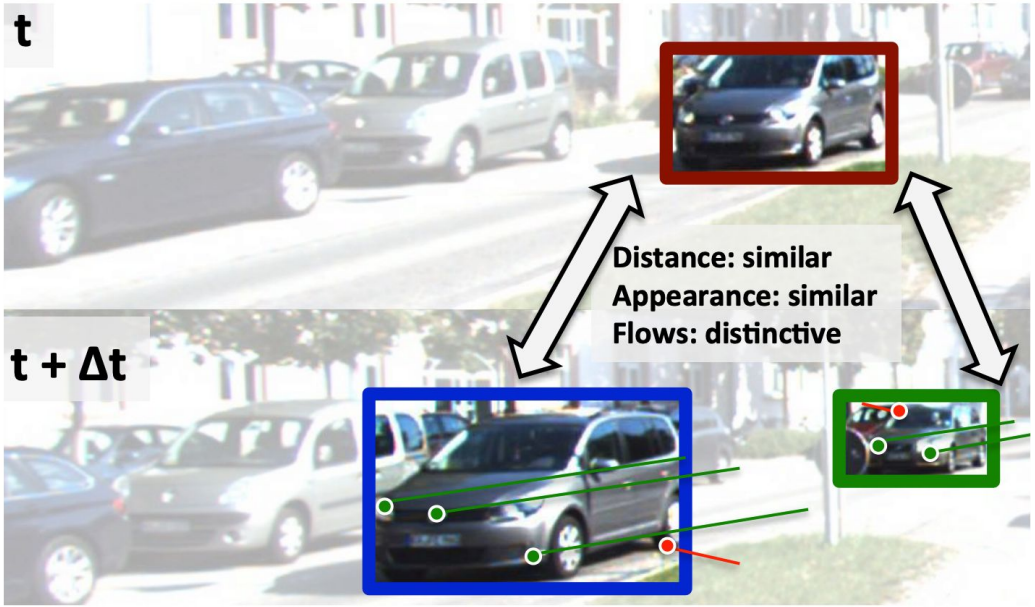


Task 1.2: Off-the-shelf Optical Flow (Feedback)

Team ID	Feedback
Team 1	Perceiver IO + 3 classical methods. Perceriver much better results Time results provided with HW information. Good! Which version of LK? Good discussions
Team 2	Pyflow. Optuna to tune hyperparameters. Results comparable to block matching MasFlowNet & UniMatch. Bonus points for clear summary of both methods. Results much better → great! Training dataset of methods?
Team 3	PWC net + classical methods Best results: PWC Implementations of methods provided Good discussions
Team 4	Missing reports in table, why? Pyflow. How did you fine-tune parameters? GMFlow, RAFT and MaskFlownet. Much better results → great! Good discussions. Training dataset of methods?
Team 5	RAFT, PerceiverIO, GMFlow + pyflow.. Learning methods trained on Sinter, Kitty Best is GMFlow. Fastest is RAFT Good quantitative and qualitative results. Time results with HW info. Good.
Team 6	Pyflow. How did you test the parameters? LiteFlowNet, RAFT, Perceiver IO, MaskflowNet, DEQ-Flow. Much better results → great! Good discussions. Training dataset of methods?

T1.3 Object Tracking with Optical Flow

Explore whether optical flow improves your results on object tracking from Week 3.



Choi, Wongun. "Near-online multi-target tracking with aggregated local flow descriptor." ICCV 2015.

T1.3 Object Tracking with Optical Flow (all teams)

		IDF1	
Team ID	**Your best algorithm (brief description)**	**Without optical flow**	**Using optical flow**
Team 1	Overlap with IOU threshold of 0.5 and using YOLOv8 detections	81.392(Overlap)	76.898
Team 2	Kalman filter SORT with OF (modification of the matching score adding Optical Flow vectors)	Kalman filter SORT (faster-finetune): → 0,6940	Kalman filter SORT with OF (faster-finetune): → 0,5009
Team 3	Overlap tracking with optical flow(pyflow) and Yolov8 detections	82.47	89.0
Team 4	Algorithm based on SORT (IoU threshold of 0.4) from week 3	79.571 (SORT)	77.449
Team 5	Best results using an algorithm based on Optical Flow instead of IoU (explained in the slides) using DETR	93.53 (SORT)	93.99
Team 6	Tracking by maximum overlap (IoU threshold = 0.5) with Optical Flow (based on RAFT) using fasterRCNN detections.	87.79	87.94

T1.3 Object Tracking with Optical Flow (Team X) [max 3 slides]

T1.3 Object Tracking with Optical Flow (Team 1)[1/3]

Our tracking algorithm uses overlap form past week (out best model), and makes use of the optical flow information in the following way.

1. At first frame every detected object is assigned an ID and added as a track
2. From frame 2 to N the boxes coordinates from frame N-1 plus the mean, median or argmax of the optical flow for the box area in each direction are used to compute the overlap with the detections in the frame N.
3. The detections from the frame N that achieve an overlap greater than 0.5 with a track from the previous frame (N-1) are assigned that track id.
4. If the overlap is lower than 0.5 the object is assigned another track id.

Applying the mean, median or max of the optical flow to the bounding boxes in the frame N-1 we can compute where is expected the bounding box from frame N-1 to be in the frame N. Therefore the overlap between the expected bounding box in N and the detected bounding boxes in N should be higher than just using the bounding box from the previous frame and we may avoid mistakes in the assignment of the bounding box when the detector is not accurate enough.

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

We used the detections from YOLOv8 model fine-tuned with 25% of the sequence

We tried to use:

- Mean Optical Flow of the bbox
- Median Optical Flow of the bbox
- Max value Optical Flow of the bbox

Tracking method	IDF1	HOTA
Farneback OF mean	76.898	72.025
Farneback OF median	76.891	72.023
Farneback OF max	77.394	72.028
Overlap (week 3)	**85.7**	**83.07**

Quantitative analysis

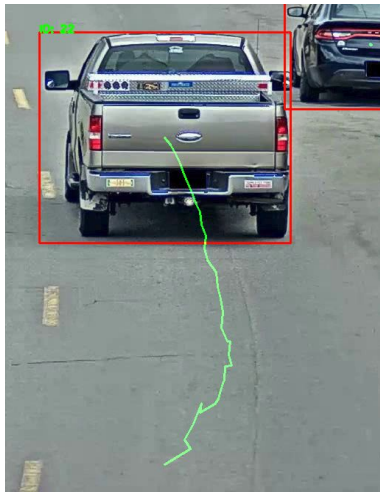
- The tracking algorithm using the Optical Flow information is not able to surpass the performance of the best algorithm from the past week that used the overlap algorithm.
- The difference in the performance when using the mean median of max value from the OF show minimal variance.

T1.3 Object Tracking with Optical Flow (Team 1) [3/3]

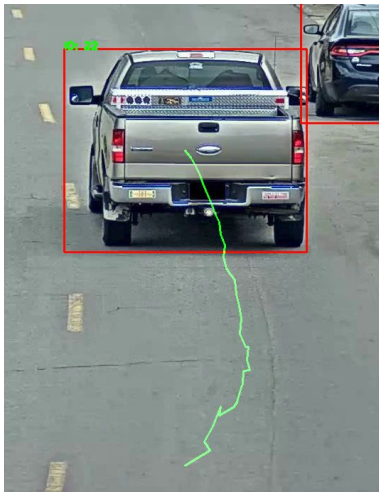
OF obtained with Farneback method

Week 3 Overlap method

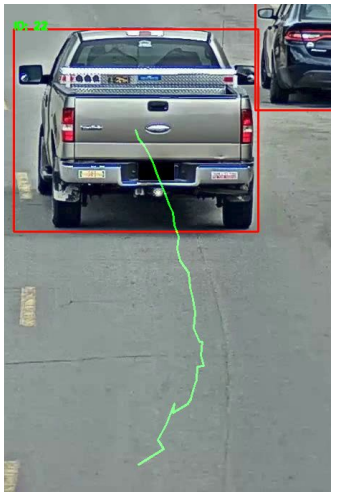
Overlap OF mean



Overlap OF median



Overlap OF max



Qualitative analysis

- As shown in the above image the tracks for the detected objects are in general less noisy when using the Optical Flow information.
- The difference in the metrics is due to the mismatch in the track id with the distant cars where there happens to be a lot of overlapping with the parked cars at the end of the street.

Conclusions

The implemented method was not successful and the smoother tracks are not sufficient to justify the extra time needed to compute the optical flow of the images

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

We try to improve our best tracking method from Week 3, **tracking with a Kalman filter**, by leveraging **optical flow** information.
- **Data Association**: The Kalman filter requires associating detected objects across frames to maintain tracking. SORT uses the Hungarian algorithm to perform data association, which finds the optimal assignment of detections to existing tracks based on their predicted states.
- **Track Management**: SORT uses heuristics, such as the number of frames since the last update and the overlap between the predicted state and the newly associated detection, to determine whether to initialize a new track or terminate an existing track.

In previous week, we used the IoU as a similarity metric to associate predicted bounding boxes with existing tracks in the SORT algorithm.
This week, we introduce optical flow information to compute the matching score between bounding boxes, which can be used as an additional similarity metric for data association in the SORT algorithm.

Tracking by Kalman filter with optical flow
Modification of the SORT algorithm to compute the matching pixels between two sets of bounding boxes using optical flow information between two frames.
It is used in the part of the algorithm where we assign detections to tracked objects (both represented as bounding boxes).

1. **Compute** (or preferably pre-compute and save, as in our case) the **optical flow between two frames**. OF is represented as a 3-channel float32 image with optical flow vectors (u, v, 1).
2. For each predicted bbox, **compute a new bbox** in the 2nd frame **by adding the optical flow vectors** to the coordinates of the bbox in the 1st frame.
3. **Calculate a matching score** for each pair of bboxes. The score is computed as the number of pixels that are present in both the updated bbox and the bbox in the 2nd frame, and stored in a distance matrix. It is used to determine the similarity or dissimilarity between the two bboxes; a higher score indicates a higher spatial similarity.

We performed the following experiments for the **object tracking** of sequence **S03-c010**:

- **Models**: [faster-finetune, mask-rcnn, ssd512, yolo3]
- **Optical flow** prediction:
  - [with **best Block Matching**: → estimation_type = 'forward' / error_function = 'nccorr' / block_size = 24 / search_window_size = 76,
  - with **(off-the-shelf) Unimatch**]
- **Tracking** method:
  - [**Kalman filter** with SORT, BEST Kalman filter Tracking parameters (found in WK 3) → max_age = 50 / min_hits = 3 / min_iou = 0.3
  - Maximum **Overlap** BEST Overlap Tracking parameters (found in WK 3) → max_frame_skip = 10 / min_iou = 0.5]

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

Tracking	Model	Optical Flow	HOTA	IDF1	Time [FPS]
Kalman	faster-finetune	-	**0,6758**	**0,6940**	42.1
Kalman	faster-finetune	Block Matching	0,4889	0,5009	1.9
Kalman	faster-finetune	Unimatch	0,4888	0,5008	2.3
Kalman	mask-rcnn	-	0,4866	0,4377	47.0
Kalman	mask-rcnn	Unimatch	0,3357	0,3340	2.7
Kalman	mask-rcnn	Block Matching	0,3356	0,3340	3.0
Kalman	ssd512	-	0,3895	0,3882	40.9
Kalman	ssd512	Unimatch	0,2499	0,2266	4.3
Kalman	ssd512	Block Matching	0,2499	0,2265	4.3
Kalman	yolo3	-	0,4127	0,4419	46.1
Kalman	yolo3	Block Matching	0,2827	0,3039	4.8
Kalman	yolo3	Unimatch	0,2825	0,3038	4.2
Overlapping	faster-finetune	-	**0,6626**	**0,6387**	47.0
Overlapping	mask-rcnn	-	0,5158	0,4579	38.2
Overlapping	ssd512	-	0,3742	0,3377	41.1
Overlapping	yolo3	-	0,3812	0,3689	43.4

Based on the results we obtained, it appears that using optical flow for object tracking with the Kalman filter does not lead to improvements, but rather decreases the overall tracking performance as measured by the HOTA and IDF1 metrics. Moreover, when using OF we suffer a lower processing of frames: **4FPS**; when not using it the processing time goes up to **40FPS** for Kalman and Overlap.

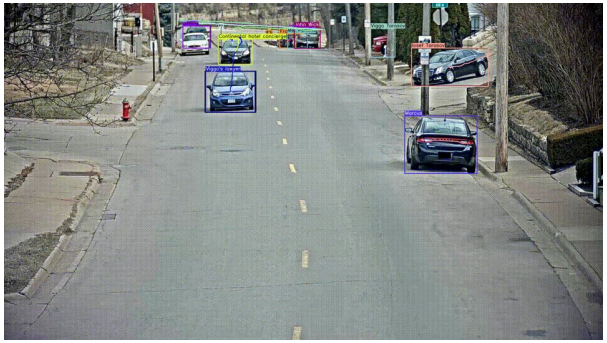
HOTA measures the overlap between predicted and GT bboxes, taking into account localization, FPs, missed detections, and identity switches. IDF1 is the f1 score specifically for object detection, which considers both precision and recall.

We can observe that across different models and optical flow estimations the HOTA and IDF1 scores are lower when optical flow is used compared to when it is not used.

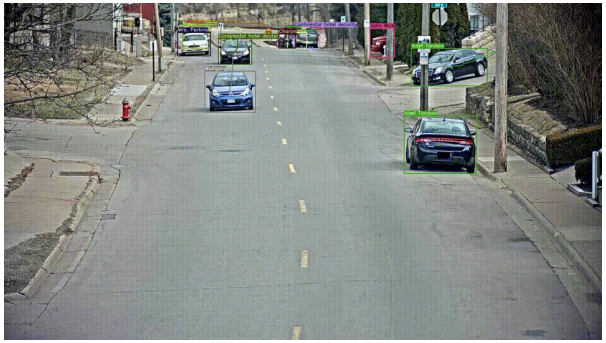
We think there could be some reasons why optical flow in this case does not improve tracking performance. Here are some possible explanations:

1. **Inaccurate optical flow**: it can be affected by various factors such as occlusions, illumination changes, and camera motion. If the OF estimates are not accurate, it can result in incorrect predictions of object motion, leading to decreased tracking performance.
2. **Mismatched motion models**: the Kalman filter assumes a linear motion model for object tracking. However, objects in the video sequence may exhibit non-linear or complex motion patterns (eg. cars that turn). If the motion model used in the Kalman filter does not match the actual motion, incorporating OF may not improve tracking.
3. **Increased computational complexity and numerical errors**: adding OF to the tracking requires additional computations, such as computing flow vectors, save them (with some precision loss!) and updating the bboxes based on the flow estimates. This can increase complexity and induce additional sources of error to the tracking.
4. **Lack of robustness to noise**: optical flow estimates can be noisy, especially in regions with low textures or areas with rapid motion. If the OF estimates are noise, it can introduce error in the tracking and lead to decreased performance.

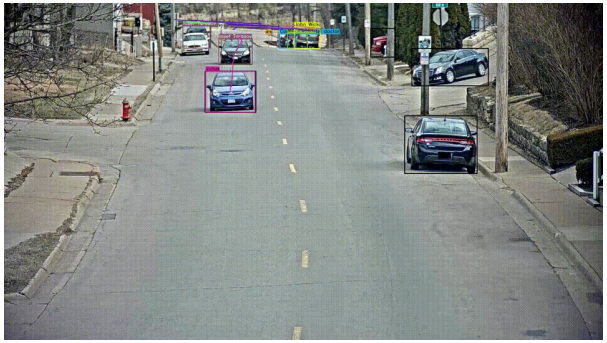
T1.3 Object Tracking with Optical Flow (Team 2) [3/3]



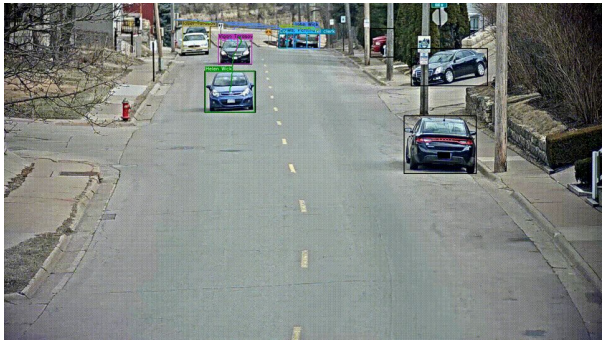
Kalman without optical flow
HOTA: 0,6758 IDF1: 0,6940



Maximum Overlap without optical flow
HOTA: 0,6626 IDF1: 0,6387



Kalman with Block Matching optical flow
HOTA: 0,4889 IDF1: 0,5009



Kalman with Unimatch optical flow
HOTA: 0,4888 IDF1: 0,5008

Initially, we thought that incorporating optical flow information could provide a more accurate estimate of the spatial similarity between bboxes, and could improve the tracking performance in certain scenarios where objects undergo significant motion or deformation between frames. However, quantitative results indicate the opposite.

In this qualitative comparison using the **Faster-Finetuned** model the same conclusion is observed: adding the OF does not improve the tacking. One clear example is the parked car on the right, which is correctly "tracked" with both Kalman and Max. Overlap approaches, but this car tracking disappears when using OF. This could be due to having a noisy OF in this region that wipes out the predicted bbox.

Apart from this obvious difference, there are no more significant qualitative differences between the 3 Kalman filter approaches. Only the tracking by Max. Overlap experiments some duplications of instances when cars leave the scene (as seen in WK3), but even the quantitative performance is higher than when using OF.

Task 1.3: Object Tracking with Optical Flow (Team 3) 1/2

Algorithm:

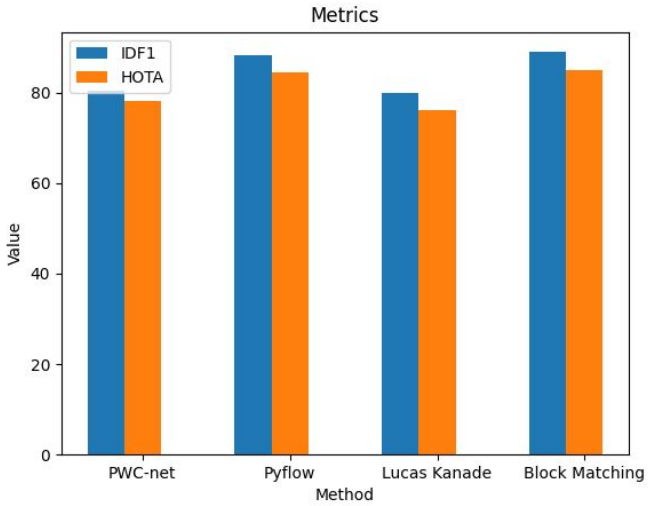
- Collect all the bounding boxes for each frame obtained using the trained model.
- Loop over all the frames, excluding the last one. Assign a new ID to each unassigned block.
- **Predict the position of the bounding box in the next frame using optical flow. All the pixels in the window will be assigned to the predominant flow.**
- For each track, compute IOU for each detection for next frame.
- Set threshold of IoU ≥ 0.5 to determine the correspondence of consecutive bounding boxes
- Assign the bounding box to the closest track which has the maximum overlap or the maximum iou.

Without Optical Flow

Optical Flow - Block Matching

Task 1.3: Object Tracking with Optical Flow (Team 3) 2/2

Method	IDF1	HOTA	Runtime
Without OF	82.47	78.211	**1 min**
Block Matching	88.119	84.512	65 mins
Lucas Kanade	80.009	76.033	28 mins
Pyflow	**89.0**	**85.1**	6 hrs



- As seen in the table, pyflow and block matching perform best for tracking in terms of IDF1 and HOTA accuracy. Block Matching is able to capture fine grained details related to motion of the object and hence performs better. Pyflow is robust to large displacements and hence performs well in tracking.
- Lucas Kanade method does not perform well if there is rapid or larger displacement and other such errors in optical flow algorithms may harm the overall execution of the algorithm.
- Adding optical flow increases the accuracy in some cases, but it is computationally very expensive and not always worth the increase in accuracy.

T1.3 Object Tracking with Optical Flow (Team 4) [1/3]

Tracking + Optical Flow Algorithm

Algorithm used

1. Detect objects
2. Assign the object to an existent or a new track depending on the scenario:
    a. If we are in the first frame, assign each object to a different track
    b. If not, compute the optical flow of the bounding boxes of the objects detected in frame N-1 and select the mean/median in each direction.
    c. Compare the bounding boxes of the objects detected in the frame N with the bounding boxes in the frame N-1 + mean/median in each direction of the optical flow computed in step b.
        i. Pick the candidate with the maximum overlap, and if the IoU is greater than 0.4, assign the detected bounding box to the corresponding track ID of frame N
        ii. If the candidate with the maximum overlap doesn't have a IoU greater than 0.4 or no candidate has been found, assign the object detected to a new track

T1.3 Object Tracking with Optical Flow (Team 4) [2/3]

We used Faster R-CNN model fine-tuned in Week 3 to detect the cars

To compute the optical flow, we used Lucas-Kanade implementation from OpenCV. We use goodFeaturesToTrack() function to decide which points to track, using Shi-Tomasi corner detection. If no good points are found in the bounding box of frame N-1, then we pick all points in the bounding box.
Then, we compute the mean/median in each direction of the optical flow and we add it to the bounding box to estimate the bounding box in frame N

Method	IDF1	HOTA
Overlap (week 3)	76.938	80.878
SORT (week 3)	79.571	81.06
Lucas-Kanade (mean)	77.449	81.289
Lucas-Kanade (median)	77.449	81.291

- Using optical flow improves slightly IDF1 and HOTA performance with respect to the overlap method of week 3
- However, the SORT method implemented in week 3 still performs better in the IDF1 metric
- There is no difference between using the mean or the median

T1.3 Object Tracking with Optical Flow (Team 4) [3/3]

Lucas-Kanade optical flow implementation using the mean

Detected car by the model
Estimated bounding box using optical flow

Conclusions

- The results are dependent on having good detections from the model
- There are still problems with occlusions
- The algorithm loses track of cars that are far from the camera and that are moving fast (0:42, 1:12 in the video)
- The optical flow method is much more slower due to the extra computation required

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

Our tracking algorithm uses Optical Flow estimation in the following way:

1. Assign an ID to every object detected in the first frame (one new track per ID).
2. Repeat from frame 2 until the sequence finishes:
    a. Compute the optical flow between the previous frame and the current frame (we used the GMFlow method, explained in Task 1.2 slides).
    b. For each of the tracks, compute the mean or median optical flow for the pixels inside the track's previous frame BBox.
    c. Assign current frame detections taking the previous BBox center and mean or median optical flow as a reference. The minimum distance to the actual BBox center detections will be assigned (only if the estimate new BBox center with the optical flow is inside the new detection BBox).
    d. If there were detections in the current frame that are not assigned to any track, create a new track with a new object ID.
    e. If one of the tracks was not assigned to any of the actual detections, estimate a new BBox with the old BBox and the mean or median optical flow predicted.

Details:

- We did not use Intersection over Union of BBoxes, only the optical flow and BBox center comparison.
- To assign old tracks to new detections, we used the mean or median optical flow from the previous 5 frames (if available).
- If a track is not assigned to any of the next 5 frames' detections, the track will be removed. We did this to make the tracking more robust to bad detections and avoid sudden loss of object detection affecting the tracking.

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

We tried four different approaches in our new algorithm:

- Using the mean Optical Flow (OF) of the BBox.
- Using the median OF of the BBox.
- Using the pretrained DETR object detector.
- Using the DETR object detector fine-tuned last week.

We compare it with the last week tracking algorithms:

- SORT algorithm (based on Kalman Filter)
- Overlap (based on the IoU of BBoxes)

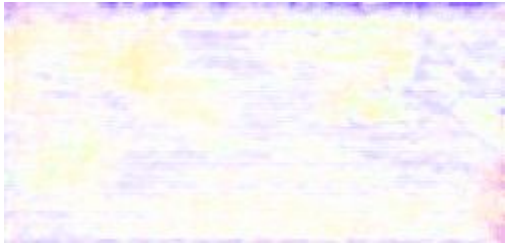
DETR detector	Tracking method	IDF1	HOTA
Fine-tuned	OF mean	93.81	86.63
	OF median	**93.99**	**86.7**
	SORT	93.53	86.66
	Overlap	85.7	83.07
Pretrained	OF mean	39.49	45.86
	OF median	39.51	45.87
	SORT	**40.79**	**47.77**
	Overlap	33.95	43.43

CONCLUSIONS:

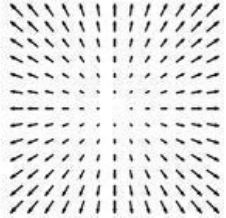
- Our optical flow (OF) algorithm outperformed the IoU-based Overlap algorithm in all scenarios.
- Using the median OF instead of the mean OF resulted in better performance. This could be due to objects not completely filling their bounding boxes, as they are not segmented.
- The SORT algorithm performed similarly to our OF algorithm, with slightly worse performance when using the fine-tuned object detector and better performance when using the pretrained detector. This indicates that the SORT algorithm is more robust to imperfect object detections.

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

Original sequence

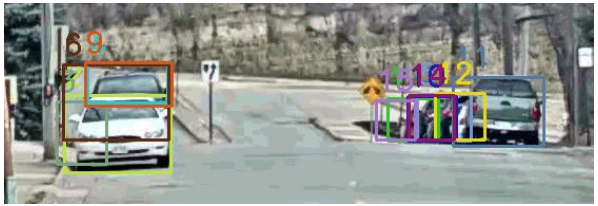


Estimated Optical Flow

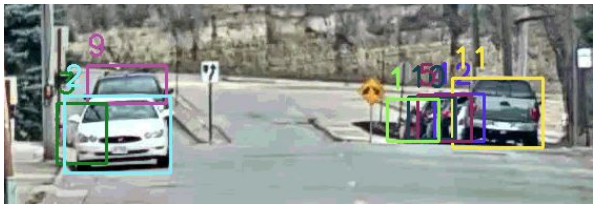


Optical flow estimation was less accurate for objects farther away.

Pretrained detector



OF median



SORT

Fine-tuned detector

OF median

SORT

CONCLUSIONS:

- Good quality detections significantly impacted the final results for both methods, with the fine-tuned detector producing better results.
- The SORT algorithm lost track of some cars (example: left initially with ID 4 and 6) when there was overlap with other cars using the fine-tuned detector, whereas the optical flow method tracked these cars correctly.
- Although both methods yielded similar results, the optical flow method was slower due to the need for optical flow estimation per frame.

T1.3 Object Tracking with Optical Flow (Team 6) [1/3]

For this task we used the same implementation as in task 2.1 of previous week. First we excluded the boxes with an overlap greater than 0.9, and we added a filter to remove the bounding boxes corresponding to bikes. This filter discarded those bounding boxes with a value greater than 230 on the y-axis and with a ratio (h/w) greater than 1.2. Detections were obtained with **FasterRCNN** and we set a **confidence value of 0.8**, since it was one of the best configurations obtained in week 3.

Maximum Overlap algorithm (week 3):

We compare the maximum overlap between bounding boxes of frame N to frame N_{n+1}. We took into account the following conditions:

- In frame N_0 we initialize all the IDs of the objects present at N_0.
- When comparing objects present in N_{n+1} with those present in N, we set a threshold of **iou>0.5**.
- If there is no maximum overlap above 0.5, a new ID is added to that object. The same for those boxes with maximum overlap with the same object in the previous frame.
- We added a memory function that considered static and moving elements separately. So, if they appeared consecutively, their IDs were kept, if not, we dropped them. This was done with a frequency of 5 frames.

Maximum Overlap algorithm with Optical Flow:

- In each frame N_{n+1}, the optical flow is computed with respect to the previous one N.
- We identify the optical flow of each object previously tracked.
- The **displacement is defined as: delta * bounding_box_flow**. Where the bounding_box_flow is the average of the optical flow of all pixels composing the bounding box, both in the vertical and horizontal component. Delta is the inverse of fps.
- The object motion is estimated by adding the displacement to each coordinate of the bounding box in frame N.
- Finally, the iou was computed for each object in frame N_{n+1} with respect to the objects in frame N, with the motion estimate added.

T1.3 Object Tracking with Optical Flow (Team 6) [2/3]

We evaluated the tracking algorithm with optical flow using 4 different methods, as shown in the table. We chose these four from the task 1.2 comparison, where we saw that these were the best in terms of accuracy or runtime. In our case, the addition of optical flow did not improve the results quantitatively, in fact, better results were obtained without optical flow. Even so, the differences between the different methods are very small, and the results were already good without optical flow. We used TrackEval repository to evaluate our analysis [1].

Method	HOTA [%]	IDF1 [%]
max_overlap + Lucas-Kanade	84.22	87.78
max_overlap + RAFT	84.27	**87.94**
max_overlap + LiteFlowNet	84.25	87.81
max_overlap + MaskFlowNet	84.26	87.91
max_overlap (week 3) *	**84.28**	87.79

* These results are corrected from previous week. We realized that we did not correctly enter the ground truth in TrackEval.

The reason why optical flow does not improve our tracking algorithm may be due to several factors:

- Because the objects do not show much motion between frames and adding optical flow can generate noisy motion vectors.
- Some objects are partially or completely occluded, as in the case of the cars in the background. In this case, the optical flow may not be able to estimate the motion correctly.
- For the simple fact that maybe the implementation is not the most suitable, and therefore, the same errors we found last week, are also present this week.

The fact that we obtained a better IDF1 with RAFT, and a higher HOTA with last week's method, indicates that RAFT is better in terms of overall tracking accuracy, whereas the other method has a higher tracking quality. This could be due, for example, to RAFT being more robust to occlusions, while the other method more robust to identity switches. Even so, the differences are minimal and we cannot confirm this either.

T1.3 Object Tracking with Optical Flow (Team 6) [3/3]

As the results obtained did not show large differences between metrics, in this slide we will only comment on the qualitative results of tracking without optical flow, with RAFT and with MaskFlowNet. Since the best results were obtained with these three methods, and also MaskFlowNet proved to be the method with the shortest execution time, this is why the rest of the tasks were performed with MaskFlowNet.

max_overlap

max_overlap + RAFT

max_overlap + maskflownet

It is very difficult to find differences that characterize the different methodologies, since both qualitative and quantitative results are quite similar. What we can confirm is that we reached the same failures as in week 3, and therefore the same conclusion:

The method is effective for tracking objects without partial occlusions, since these can generate identity changes by mistake. Or for objects that do not move at high speed over short distances, considering that the algorithm is not capable of correctly updating the memory in these cases.

T1.3 Object Tracking with Optical Flow (Feedback)

Team ID	Feedback
Team 1	No improvement mean, median or argmax of the OF in BB. Good analysis
Team 2	Adapt SORT to include OF: How do you move the BB with the OF? Good quantitative and qualitative presentation of results. NO improvements
Team 3	Several OF methods tested Predominant vector used Improvement Conclusion: Improvement maybe does not compensate the computational cost.
Team 4	New tracking implementation from OF. Did you try better OF algorithms? why Lucas-Kanade? Use mean/media of OF in the BB. Small improvement in HOTA.
Team 5	Mean or median OF GMFLow used Small improvement
Team 6	Adapt Maximum overlap with OF. Further explain the delta parameter? How does it affect results? Use mean of OF in the BB. Tried 4 OF algorithms NO improvements (you said “better results are obtained without but the table reflects the opposite)?

… continue in part2