236781
Deep Learning On Computational Accelerators
Final Project Report

Nitzan Shmuel: 302658331
snitz@campus.technion.ac.il

Itay Shamir: 308068873
itay-shamir@campus.technion.ac.il

Contents

1 Abstract **1**

2 Intro **2**
2.1 The original approach . . . 2
2.2 The TartanVO architecture, and previous works that attacked its components . . . 3
2.3 Our Contributions . . . 3

3 Methods **4**
3.1 Attacking rotation error . . . 4
3.2 Attacking optical flow . . . 4
3.2.1 MSE loss . . . 5
3.2.2 Cosine Similarity loss . . . 5
3.2.3 SSIM loss . . . 5
3.2.4 MS-SSIM loss . . . 7
3.3 APGD . . . 8
3.4 Modified APGD . . . 9

4 Implementation and experiments **9**
4.1 Experiment Notice . . . 9
4.2 Implementation . . . 9
4.2.1 Loss Formulation . . . 10
4.2.2 A note about the changes in the code . . . 10
4.3 Experiments . . . 10

5 Results and discussion **11**
5.1 Comparison of optimization schemes . . . 12
5.2 Comparison of optical flow attacks and their relative importance on the overall results . . . 13
5.3 Hyper Tuning . . . 13
5.4 Conclusions . . . 15
5.5 Future Research . . . 15

1 Abstract

DNN-based Visual Odometry (VO) methods are susceptible to adversarial perturbations. One of these methods, TartanVO, was a case study for a black-box attack on visual odometers. In this work, we improve upon the results of this previous work. This is done by extending the adversarial attack optimizer from PGD to Auto-PGD and extending the loss function to optimize on addition properties other than the deviation from ground-truth trajectory. These additional loss terms include the quaternion product between the perturbed and the ground-truth rotations, and additional losses on the optical flow of the clean and perturbed sequences. Several attempts were made to combine these criteria and regulate them relatively to each other. The result was that replacing PGD with APGD significantly improved the perturbations, while adding the quaternion product *OR* one of the loss functions on optical flow added a less significant additional improvement to the result. However, when we

found out that when not regulated properly, adding a loss function on the optical flow *AND* on the rotation hampered the training process, resulting in a perturbation which is less effective than the perturbations we would get when adding each of the losses alone.

2 Intro

Monocular visual Odometry (VO) aims to infer the relative camera motion (position and orientation) between two or more consecutive viewpoints. Until recent years, such models were mostly based on a common pipeline of "classic" computer vision algorithms, most of them perform the following two stages:

1. Optical flow estimation - optical flow, or dense pixel matching, is a method which tries to estimate the "velocity" (or location change) of image pixels or image features in consecutive frames. In order to solve the relevant equations, assumptions and approximations are usually made, such as the brightness constancy constraint and the small motion constraint, that enable the Taylor approximation of the equations. The performance of these handcrafted methods start to degrade rapidly when there are effects which are hard (or impossible) to model analytically such as occlusions of objects or rapid changes in brightness.

2. A robust pose estimation algorithm that is usually based on the co-planar rule (epipolar geometry). These algorithms perform really well when proper pixel correspondences (or inliers) between the two frames are selected. However, insertion of improper correspondences (or outliers) can degrade the performance of these algorithms, and so many of them now include stages of outlier detection and removal (in addition to various methods for the selection of correspondences). These methods are often based on analytical statistical/probabilistic models, which sometimes break when their assumptions do not hold.

For these reasons, recent research has tried to replace some of these algorithms with deep learning based methods, which can bridge the non-analytical gap and sometimes even improve performance, with the goal of creating an end-to-end approach that performs better than "classical" VO methods. One such method is TartanVO [9], which even showed an ability to generalize from simulated data to real scenes. However, Deep neural networks have been discovered to be susceptible to *adversarial perturbations* - small changes to the input that significantly alter the output of the network.
Adversarial Attacks are methods for generating such perturbations. While some attacks target specific input, *Universal adversarial attacks* aim to produce a perturbation for a set of inputs, with the goal to generalize the perturbation to out-of-sample data. In some cases, perturbations can transfer between similar models or be trained to works on many models simultaneously. Recently, Nemcovsky et al. [5] produced a "black box" method for adversarial attacks on DNN based VO models. The perturbation they produced comes in a form of a patch, which can be transformed to fit in a portion of the scene (a certain percent of the frame's size), and aims to mislead the VO system by disrupting its ability to spatially position itself in the scene. Unlike previous work, in which the perturbation is inserted into a single image or two consecutive frames, their perturbation is inserted into all of the images of numerous scenes, and optimized over the accumulated deviations from ground truth trajectories. In order to do so, they utilized TartanVO as the DNN model, which on its output the perturbations are optimized.

In present work, we aim to improve their result by improving their optimization method, and also by utilizing more outputs of TartanVO - namely rotation and optical flow.

2.1 The original approach

- Nemcovsky et al. [5] optimized their perturbation patch p using the PGD adversarial attack [4], with l_{inf} norm limitation. This attack is considered the benchmark for universal adversarial attacks. However, the method has a few issues such as fixed step size and unawareness of the optimization trend, which the APGD [2] addresses. More on the difference between the methods can be found in 3.3.

- Nemcovsky et al. [5] used the l_{RMS} and l_{MPRMS} between the ground truth and perturbed trajectories as their optimization and evaluation criteria. While the criterion uses both translation and rotation outputs of the network, it considers only the deviation from the trajectory while ignoring other criteria that if corrupted, can also impact this trajectory. Namely, this approach does not include loss functions for the rotation of each frame in the trajectory, and does not include a loss function between the ground truth and perturbed optical flow outputs, which if corrupted can impact the estimation of both rotation and translation.

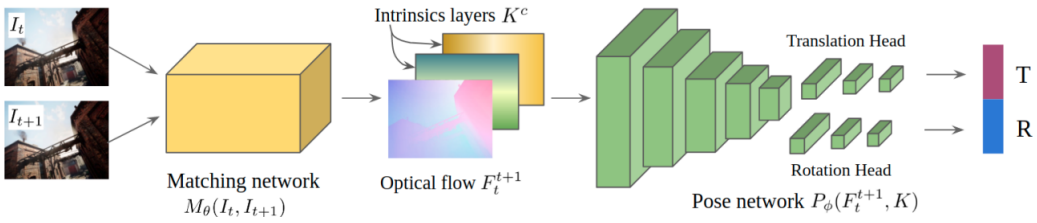


Figure 2.1: Diagram of the two-stage network architecture of TartanVO, taken from [9].

2.2 The TartanVO architecture, and previous works that attacked its components

In order to effectively attack the network, it is important to understand its structure and methods of training.

The architecture of the network, as can be seen in Figure 2.1, cosists of a matching network, which estimates optical flow from two consecutive RGB images, followed by a pose network predicting camera motion from the optical flow. From this structure, we can infer that a successful perturbation should first corrupt the output of the matching network M_θ and then the corrupted optical flow will cause the pose network to deviate from the ground truth.

The Matching Network

Wang et al. [9] use PWC-Net [8] as their dense matching network M_θ. Direct attacks on this network were tried in [6] (black-box attack, circular patch) and [7] (universal attack on the entire image). In both cases, limited degradation was achieved.

Ranjan et al. [6] were among the first to study the robustness of optical flow networks to adversarial attacks. Their method utilizes the cosine similarity loss between the ground-truth optical flow image and the estimated optical flow image. They showed that in some neural networks that estimate the optical flow between consecutive frames, corrupting a small patch in the image can significantly affect optical flow estimates. On some of the optical flow models, their method of attack lead to to noisy flow estimates that extended significantly beyond the region of the attack. They have found that while some models (based on encoder-decoder architectures) are very sensitive to this kind of attacks, other models which use a spatial pyramid architecture (PWC-Net being one of them) are less affected by this attack. It is important to note that in their work, where the attack is the most similar to our scenario, degradation was observed mainly in the area of the patch and some small areas around it. Nevertheless, it is not insignificant, and may be enough to corrupt the pose estimate in this work.

The Pose Network

Wang et al. [9] utilize a modified version of ResNet50 [3] as the pose network P_ϕ. They removed the batch normalization layers from ResNet and added two output heads for the translation and rotation, respectively. The unmodified version of this network is a very popular model for image classification, and several works have tried to analyze its robustness to adversarial perturbations. Notably, Benz et al. [1] analyzed it's robustness to adversarial attacks using PGD and FGSM as optimizers, and classified it as one of the less robust models to adversarial perturbations (which is evident in their results). It is clear, however, that patches that were trained to corrupt the output of this model will not work in this scenario, as they need to pass through M_θ before reaching P_ϕ.

Training scheme

In their training scheme, Wang et al. [9] used a pre-trained version of this model for the first stage of training, where only the pose network was trained. In the second stage of training, both networks were trained together. For this reason, it is unlikely that patches that were trained to degrade PWC-Net in other papers will achieve better results on the PWC-Net that comes with TartanVO.

2.3 Our Contributions

- **Extending the optimizer to APGD** - Nemcovsky et al. [5] utilized the PGD adversarial attack optimizer for their attack on VO systems. They modified it to optimize over a given trajectory containing multiple frames, rather than on two consecutive frames. We have extended their scheme to support Auto-PGD, which properties and algorithm description can be found in Section 3.3. This change has made a significant improvement on the results.

- **Implementation and testing of additional attacks on optical flow** - In addition to the *MSE loss* already implemented in the code of [5] (but was not used), we implemented three more losses on the optical flow produced by the matching network M_θ:

  1. **Cosine Similarity loss** - which measures the angle difference between the flow vectors, and was used in [6] to perturb PWC-Net. See section 3.2.2 for more details.
  2. **SSIM (structure similarity index) loss** - addresses the issue of locality in *MSE* and *Cosine Similarity* losses. Note that the optical flow was needed to be converted to RGB in order to use it. See section 3.2.3 for more details.
  3. **MS-SSIM loss** - A multi-scale approach for the SSIM loss, which gives less weight to changes at a particular scale and more weight to changes at multiple scales.

  When coupled with the translation loss and regulated, these loss functions improved the final result of the training. However, we found out that when not regulated properly, adding a loss function on the optical flow *and* on the rotation hampered the training process, resulting in a perturbation which is less effective than the perturbations we would get when adding each of the losses alone.

- **Utilization and analysis of multiple attack losses** - A loss function which combines and regulates the translation, rotation and optical flow losses was used and multiple configurations were tried and analyzed.

3 Methods

Below we detail the methods we have used or implemented in order to improve the adversarial attack on TartanVO. The details and logic of each implementation are also expanded upon.

3.1 Attacking rotation error

Although not in the scope of their paper, Nemcovsky et al. [5] provided in their code-base a function which can be used as a metric that quantifies the rotation error between the original and perturbed pose estimates. The function is as follows:
Let $R, \tilde{R} \in so(3)$ be the ground-truth rotation and the estimated rotation, respectively. Let $q, \tilde{q} \in \mathbb{H}$ be their respective conversion into quaternions. Note that conversion of rotation matrices or Euler angles into quaternions produce quaternions which are on the unit hyper-sphere with the following properties:

- $\|q\| = 1$
- $q_1 \cdot q_2 \in [-1, 1]$
- $q \cdot q = 1$

Hence, the patch p can be optimized in the following way:

$$\hat{p} = \arg\min_p \{q \cdot \tilde{q}\} = \arg\max_p \{1 - q \cdot \tilde{q}\} \tag{3.1.1}$$

3.2 Attacking optical flow

Consider an optical flow network M_θ that given un-perturbed image inputs (I_t, I_{t+1}) of size $H \times W$ produces the ground-truth optical flow labels (u, v):

$$(u, v) = M_\theta(I_t, I_{t+1}) \tag{3.2.1}$$

Consider a small patch p of a smaller size $h \times w$ that is pasted onto the images to perturb them. Let δ be a transformation that can be applied to the patch. We define the perturbation $A(I, p, \delta, l)$ on the image I, that applies the transformation δ to the patch p and pastes it at a location l in the image, so that:

$$\tilde{I}_t = A(I_t, p, \delta_t, l_t), \quad \tilde{I}_{t+1} = A(I_{t+1}, p, \delta_{t+1}, l_{t+1}) \tag{3.2.2}$$

are the prturbed images. Our goal is to learn a patch p that act as an adversary to the optical flow network F, and is invariant to location l or transformation δ of the patch, so that:

$$(\tilde{u}, \tilde{v}) = M_\theta(\tilde{I}_t, \tilde{I}_{t+1}) \tag{3.2.3}$$

is a corrupted optical flow estimation, which is dissimilar to the ground truth label. The dissimilarity can be measured in a number of ways, some of them (that were tested) will be covered in the next subsections.

3.2.1 MSE loss

Although not used in the scope of their paper, Nemcovsky et al. [5] provided in their code-base a function for computing the mean square error between the flow estimates, which can be used to optimize the perturbation patch p in the following way:

$$\hat{p} = \arg\max_{p} \mathbb{E}_{I,l,\delta} \left\{ ||(u,v) - (\tilde{u},\tilde{v})||^2 \right\} \tag{3.2.4}$$

This loss is sensitive to the difference between the flow vectors, regardless of their direction, and regardless to the locality of the changes in the image (e.g large local changes could have the same MSE loss as small, global changes).

3.2.2 Cosine Similarity loss

Ranjan et al. [6] suggested to optimize the patch p using:

$$\hat{p} = \arg\min_{p} \mathbb{E}_{I,l,\delta} \left\{ \frac{(u,v)\cdot(\tilde{u},\tilde{v})}{||(u,v)||\cdot||(\tilde{u},\tilde{v})||} \right\} = \arg\max_{p} \mathbb{E}_{I,l,\delta} \left\{ 1 - \frac{(u,v)\cdot(\tilde{u},\tilde{v})}{||(u,v)||\cdot||(\tilde{u},\tilde{v})||} \right\} \tag{3.2.5}$$

which is the cosine angle between the optical flow estimated by the network for normal and perturbed images. Therefore, minimizing this loss is equivalent to finding the perturbations that reverse the direction of the optical flow estimated by the network. Note that like the MSE loss, this similarity loss does not address the issue of the locality of the corruption.

3.2.3 SSIM loss

Since a robust algorithm for pose estimation could ignore local changes that give a bad estimate, we seek to find a similarity measure that will be able to tell us if the entire structure of the two optical flow images is similar or not. In 2004, Wang et al. [11] introduced the **Structural Similarity Index** (SSIM) metric, that nowadays is often used as a loss function for training neural networks for various computer vision and image processing tasks. The authors of the paper make two essential points:

- Many Image quality assessment techniques rely on quantifying errors between a reference and a sample image. A common metric is to quantify the difference in the values of *each of the corresponding pixels* between the sample and the reference images (By using, for example, Mean Squared Error).
- The Human visual perception system is highly capable of identifying structural information from a scene and hence identifying the *differences between the information* extracted from a reference and a sample scene. Hence, a metric that replicates this behavior will perform better on tasks that involve differentiating between a sample and a reference image.

The comparison between the two images is performed on the basis of three features extracted from the images:

1. **Luminance**, which is compared between two images x and y using the following function:

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{3.2.6}$$

   Where μ represents the mean intensity of a given image and C_1 is a constant to ensure stability when the denominator becomes zero.

2. **contrast**, which is compared between two images x and y using the following function:

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{3.2.7}$$

   Where σ represents the standard deviation of a given image and C_2 is a constant to ensure stability when the denominator becomes zero.

3. **structure**, which is compared between two images x and y using the following function:

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{3.2.8}$$

   Where σ_{xy} represents the co-variance between the two images and C_3 is a constant to ensure stability when the denominator becomes zero.

Finally, the SSIM score is given by:

$$SSIM(x,y) = [l(x,y)]^{\alpha} \cdot [c(x,y)]^{\beta} \cdot [s(x,y)]^{\gamma} \tag{3.2.9}$$

Where α, β, γ are larger than zero scalars that denote the relative importance of each of the metrics. If we combine 3.2.6, 3.2.7, 3.2.8, 3.2.9 and assume $\alpha = \beta = \gamma = 1$, $C_3 = C_2/2$ (to simplify the resulting expression), we get:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{3.2.10}$$

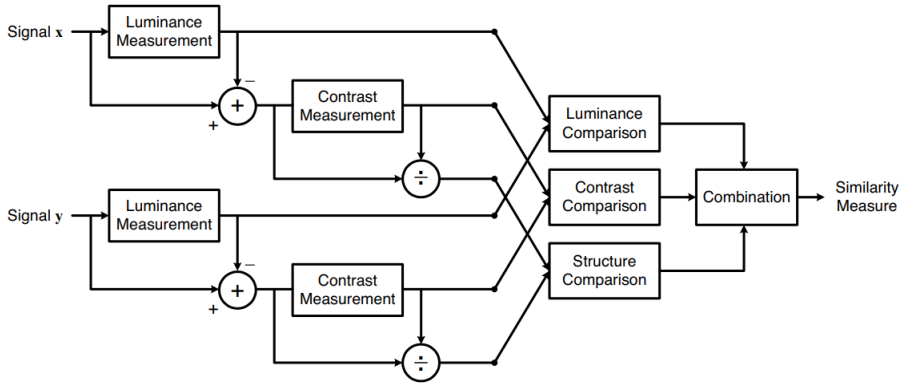


Figure 3.1: Diagram of the structural similarity (SSIM) measurement system, taken from [11]. Signal X and Signal Y refer to the Reference and Sample Images.

The measurement system description can be found in Figure 3.1. The output of this system is a number in $[-1, 1]$ where a value of $+1$ indicates that the given images are very similar or the same while a value of -1 indicates they are very different. Often these values are adjusted to be in the range $[0, 1]$, where the extremes hold the same meaning.

Locality

Wang et al. [11] explain that it is more useful to apply the SSIM index locally (i.e. in small sections of the image and taking the mean overall) rather than globally (all over the image at once), for the following reasons:

1. Image statistical features are usually highly spatially nonstationary.
2. Image distortions, which may or may not depend on the local image statistics, may also be space-variant.
3. At typical viewing distances, only a local area in the image can be perceived with high resolution by the human observer at one time instance.
4. Localized quality measurement can provide a spatially varying quality map of the image, which delivers more information about the quality degradation of the image.

In practice, the authors use an $11x11_{[pixels]}$ window, which moves pixel-by-pixel over the entire image. At each step, the SSIM index is computed for the window, but a circular-symmetric Gaussian Weighing function with $\sigma = 1.5_{[pixels]}$ is used to compute the mean, STD and covariance (instead of the standard formulas). In [11], this method is referred to as the Mean Structural Similarity Index (MSSIM), but since it is the default method we used for this metric, we will still refer to it as SSIM.

Converting optical flow to RGB

Note that the SSIM metric is performed on *intensity values* in $[0, M]$ where M is the maximal intensity level (according to [11], different dynamic ranges will give different C_1, C_2, C_3). RGB images can still fall into this category by, for example, averaging the SSIM score for all three channels. Optical Flow images, however, do not fall into this category. They are represented by a two-channeled floating-point image, which can have negative values (as opposed to an intensity level), where each channel (u, v) represents the magnitude of the flow vector for each axis of the image.

This representation **can be converted to RGB**, where the SSIM metric can be applied, in the following way:

1. **Convert the optical flow to the HSV color-space**
   Flow vector for each pixel can also be represented by *magnitude* and *angle*. The HSV colorspace is cylindrical, where the *Hue* channel represents an angle in $[0, 2\pi]$. This channel can correspond to the angle of the flow vector, while the *Value* channel can correspond to the vector's magnitude. The *Saturation* channel will be constant with an arbitrarily value, which we chose to be the maximal possible value. After the conversion, each flow direction is represented by a different color, while the magnitude of the vector is represented by the color's intensity.

2. **Convert the HSV image to RGB**
   After the conversion, we can apply the SSIM metric to the image.

Note that this means gradients for both conversions should be saved in order to use the SSIM metric as a loss for the optical flow, so each conversion was implemented as a layer.
We'll denote:

$$I_{RGB}^{flow} = RGB\Big(HSV\Big(M_\theta(I_t, I_{t+1})\Big)\Big), \quad \tilde{I}_{RGB}^{flow} = RGB\Big(HSV\Big(M_\theta(\tilde{I}_t, \tilde{I}_{t+1})\Big)\Big) \tag{3.2.11}$$

The optimization for the patch p will then be:

$$\hat{p} = \arg\min_p \mathbb{E}_{I,l,\delta,\text{channel}} \left\{SSIM\left(I_{RGB}^{flow}, \tilde{I}_{RGB}^{flow}\right)\right\} = \arg\max_p \mathbb{E}_{I,l,\delta,\text{channel}} \left\{1 - SSIM\left(I_{RGB}^{flow}, \tilde{I}_{RGB}^{flow}\right)\right\} \tag{3.2.12}$$

3.2.4 MS-SSIM loss

In a different work, Wang et al. [10] proposed a multi-scale structural similarity method, which they claim supplies more flexibility than SSIM in incorporating the variations of viewing conditions. In practice, the single-scale method as described in the previous section may be appropriate only for specific settings, while a multi-scale method is a convenient way to incorporate image details at different resolutions.

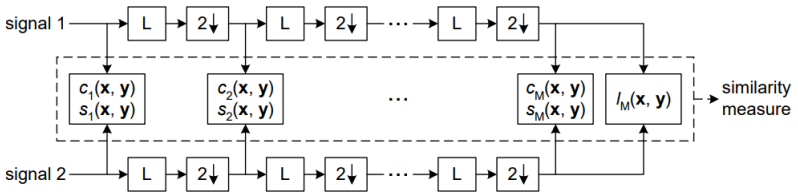


Figure 3.2: Diagram of the Multi-scale structural similarity (MS-SSIM) measurement system, taken from [10]. L: low-pass filtering; 2↓: downsampling by 2.

The metric is calculated by iteratively downsampling both images by a factor of 2 in each iteration (after low-pass filtering to avoid aliasing). The scale of the original image is indexed as 1, and the highest scale (lowest image size) is indexed as M, which is obtained after $M-1$ iteration. At each iteration j, the contrast comparison (Eq. 3.2.7) and the structure comparison 3.2.8 are calculated and denoted as $c_j(x,y)$ and $s_j(x,y)$ respectively. The luminance comparison (Eq. 3.2.6) is computed only for scale M and is denoted $l_M(x,y)$. The overall score is obtained by combining the measurements at different scales using the following equation:

$$MSSSIM(x,y) = [l_M(x,y)]^{\alpha_M} \cdot \prod_{j=1}^{M} [c_j(x,y)]^{\beta_j} [s_j(x,y)]^{\gamma_j} \tag{3.2.13}$$

where $\alpha_M, \beta_j, \gamma_j$ are used to adjust the relative importance of the different components, in a similar way to the corresponding constants in 3.2.9. Figure 3.2 illustrates the process.

In order to simplify parameter selection, two steps were made by Wang et al. [10]:

1. Constraining the parameters in two ways:

   - $\alpha_j = \beta_j = \gamma_j$ for all j's.
   - $\sum_{j=1}^{M} = 1$

   This makes all the constants equal at each scale, and normalize the cross-scale settings such that the relative values across different scales are comparable.

2. Cross-scale calibration of the parameters with the above constraints, using an algorithm defined in the paper. The resulting parameters the authors obtained (which are used in this project) are:

$$\begin{aligned} &\beta_1 = \gamma_1 = 0.0448, \quad \beta_2 = \gamma_2 = 0.2856, \quad \beta_3 = \gamma_3 = 0.3001 \\ &\beta_4 = \gamma_4 = 0.2363, \quad \alpha_5 = \beta_5 = \gamma_5 = 0.1333 \end{aligned} \tag{3.2.14}$$

Optimization for path p is the same as in Eq. 3.2.12.

3.3 APGD

Croce and Hein [2] pointed a few issues with the PGD [4] optimization scheme, which they have considered suboptimal:

- **Fixed step size** - performance of PGD is highly influenced by the choice of this value, and convergence is not guaranteed.
- **Unawareness of trend** - PGD does not consider whether the optimization is evolving successfully and is not able to react to this.
- **Agnostic of budget** - after a number of iterations, PGD optimization loss plateaus, except for extremely small step sizes which do not translate into better results. As a consequence, judging the strenth of an attack by the number of iterations is misleading.

They proposed an attack called Auto-PGD (APGD), which extends PGD in the following ways:

- **Gradient step** - The update step of APGD is similar to that of the PGD algorithm, only adding a momentum term. Let $\eta^{(k)}$ be the step size at iteration k. Then the step size is:

$$\begin{aligned} z^{(k+1)} &= P_S\left(x^{(k)} + \eta^{(k)} \nabla f\left(x^{(k)}\right)\right) \\ x^{(k+1)} &= P_S\left(x^{(k)} + \alpha \cdot \left(z^{(k+1)} - x^{(k)}\right) + (1-\alpha) \cdot \left(x^{(k)} - x^{(k-1)}\right)\right) \end{aligned} \tag{3.3.1}$$

  Where $\alpha \in [0, 1]$ (a default value of $\alpha = 0.75$ is used) regulates the influence of the previous update on the current one.

- **Step size selection** - In order to prevent the optimization from getting stuck in cycles around the extremum, the step size is changed during the optimization according to the following scheme:
  Let $\eta^{(0)} = 2\epsilon$ be the step size at iteration 0. Given a budget of N_{iter} iterations, we compute checkpoints $\omega_0 = 0, \omega_1, ..., \omega_n$ at which the algorithm decides whether it is necessary to halve the current step size. There are two conditions for halving the step size:

  1. Count in how many cases since the last checkpoint ω_{j-1} the update step has been successful in increasing f. If this happened for at least a fraction ρ of the total update steps, then the step size is kept as the optimization is proceeding properly (default value is $\rho = 0.75$. Else, the step size is halved. Formally, the step size is halved if:

$$\sum_{i=\omega_{j-1}}^{\omega_j - 1} \mathbf{1}_{f\left(x^{(i+1)}\right) > f\left(x^{(i)}\right)} < \rho \cdot (\omega_j - \omega_{j-1}) \tag{3.3.2}$$

  2. If the step size was not reduced at the last checkpoint *and* there has been no improvement in the best found objective value since the last checkpoint, then the step size is halved. Formally, the step size is halved if:

$$\eta^{(\omega_{j-1})} \equiv \eta^{(\omega_j)} \quad \textbf{and} \quad f_{max}^{(\omega_{j-1})} \equiv f_{max}^{(\omega_j)} \tag{3.3.3}$$

- **Restart from the best point** - If at checkpoint ω_j the step size gets halved, the perturbation is restarted to the one attaining the highest objective f_{max} so far. Formally, we set:

$$x^{(\omega_j + 1)} := x_{max} \tag{3.3.4}$$

  This makes sense as reducing η leads to a more localized search, and this should be done in a neighborhood of the current best candidate solution.

- **Exploration vs exploitation** - The checkpoints ω_j are defined in a way that allows the algorithm to transit gradually from exploring the whole physical set of perturbation into a local optimization. In practice, the initial exploration phase should be relatively long, since the improvements with smaller step sizes are more frequent but much lower in magnitude. In addition, the importance of taking advantage of the whole input space is testified by the success of random restarts in the original PGD attack. The checkpoints are calculated as follows:

$$\begin{aligned} &\omega_j = \lceil p_j N_{iter} \rceil \leq N_{iter} \\ &p_j \in [0,1], \quad p_0 = 0, p_1 = 0.22 \\ &p_{j+1} = p_j + max\left\{p_j - p_{j-1} - 0.03, 0.06\right\} \end{aligned} \tag{3.3.5}$$

Note that while the optimization scheme has a few parameters which could be adjusted, most of them are fixed to the default values indicated above, so that **the only free parameter is the budget** N_{iter}. A description for the method is described in Algorithm 1.

3.4 Modified APGD

We tried to implement a minor change to the APGD scheme - if the process reaches a checkpoint and the step size is not halved, it is incremented by 10% of its last value. As we will see in the results section, this change aids the process to converge faster. If the step size is too small, it will increase and improve the convergence rate. If the step size is too large, a smaller percentage of the iterations will improve the results, increasing the probability of halving the step size in the next checkpoint. Results that show exactly that can be found in Figure 5.6. It also shows that while the convergence is faster, the resulting perturbation is not necessarily better that the one optimized with the regular version of APGD.

Algorithm 1 APGD

Input: $f, S, x^{(0)}, \eta, N_{iter}, W = \{\omega_0, ..., \omega_n\}$
Output: x_{max}, f_{max}

$x^{(1)} \leftarrow P_S\left(x^{(0)} + \eta \nabla f\left(x^{(0)}\right)\right)$
$f_{max} \leftarrow max\left\{f\left(x^{(0)}\right), f\left(x^{(1)}\right)\right\}$
$x_{max} \leftarrow x^{(0)}$ **if** $f_{max} \equiv f\left(x^{(0)}\right)$ **else** $x_{max} \leftarrow x^{(1)}$
for $k = 1$ to $N_{iter} - 1$ **do**
    $z^{(k+1)} \leftarrow P_S\left(x^{(k)} + \eta^{(k)} \nabla f\left(x^{(k)}\right)\right)$
    $x^{(k+1)} \leftarrow P_S\left(x^{(k)} + \alpha \cdot \left(z^{(k+1)} - x^{(k)}\right) + (1-\alpha) \cdot \left(x^{(k)} - x^{(k-1)}\right)\right)$
    if $f\left(x^{(k+1)}\right) > f_{max}$ **then**
        $x_{max} \leftarrow x^{(k+1)}$; $f_{max} \leftarrow f\left(x^{(k+1)}\right)$
    end if
    if $k \in W$ **then**
        if `<condition 1>` **or** `<condition 2>` **then**
            $\eta \leftarrow \eta/2$; $x^{(k+1)} \leftarrow x_{max}$
        end if
    end if
end for

4 Implementation and experiments

4.1 Experiment Notice

Due to the fact our account on the lambda server was deleted on the 15th of septemer. some experiments data was lost and since each run takes a considerable amount of time. we werent able to reproduce all of the initial eperiemnts.

4.2 Implementation

Nemcovsky et al. [5] defined the adversarial setting in their paper. Since we aim to improve their result, we shall use the same formulations they used for defining our own loss function and optimization schemes below.

4.2.1 Loss Formulation

$$l_{TOT} = \lambda_1 \cdot l_{\text{trajectory}} + \lambda_2 \cdot l_{\text{rotation}} + \lambda_3 \cdot l_{\text{optical flow}} \tag{4.2.1}$$

The loss is comprised of three terms:

1. **Trajectory loss term $l_{\text{trajectory}}$** - we decided to use l_{RMS} as defined in [5] as it gave us significantly better result than l_{MPRMS}.
2. **Rotation loss term l_{rotation}** - the quaternion product loss, as described in 3.1, is our only option.
3. **Optical Flow loss term $l_{\text{optical flow}}$** - Here we have four options: MSE (3.2.1), Cosine Similarity (3.2.2), SSIM (3.2.3) and MS-SSIM (3.2.4).

Figure 4.1 describes the process of obtaining l_{TOT}.

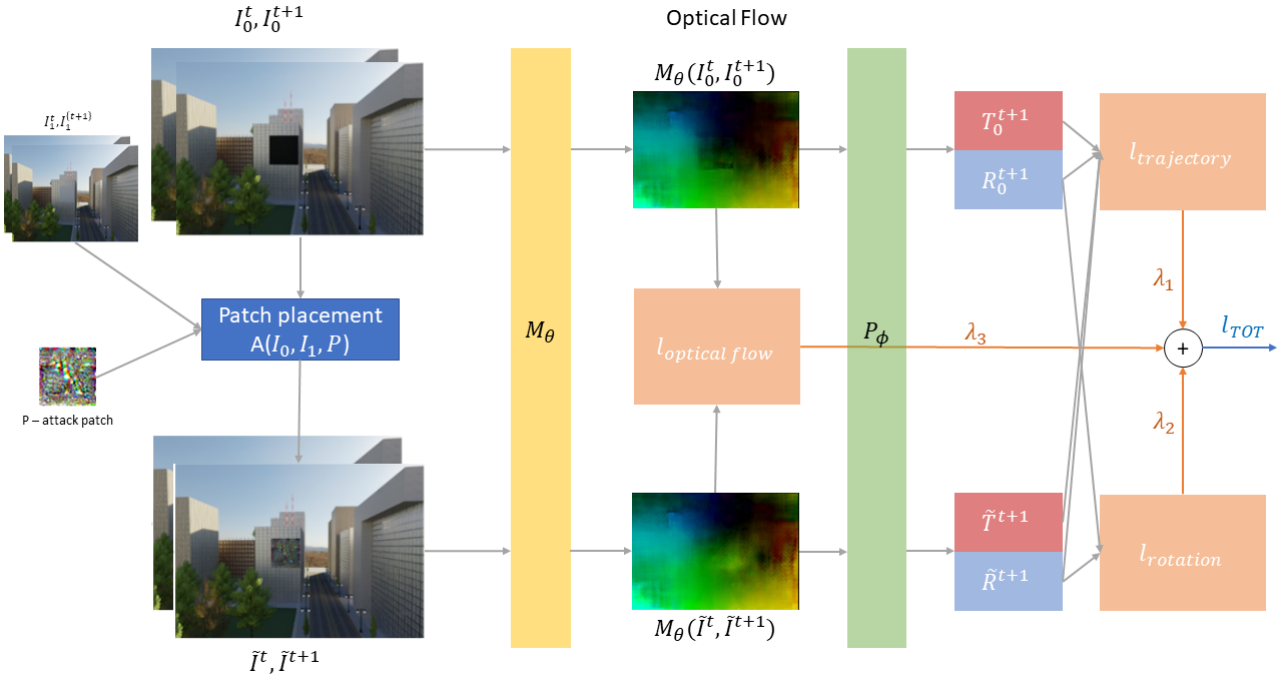


Figure 4.1: Diagram of the loss components.

4.2.2 A note about the changes in the code

Below we detail our implementations in the code and where to find them. Note that some of them require additional packages, which will also be mentioned.

- Both of the APGD implementations can be found in "attacks/apgd.py" and "attacks/our_apgd.py". They do not require any additional packages. Changes to "utils.py" had to be made in order to incorporate these attacks.
- All additional loss and auxilary functions can be found "loss.py". They SSIM and MSSSIM loss functions require the package "pytorch-msssim" which can be installed using pip.

4.3 Experiments

The Evaluation metric used in our experiments is l_{RMS}. The experiments can be divided into categories:

- First batch of experiments were in order to determine the benefits of APGD over PGD.
  Here we made similar runs using the PGD provided to us and our implementation of APGD with the following configurations
  - attack-k : 100, optical flow loss : MSE/CosineSimilarity/SSIM

  - attack-k : 200, optical flow loss : CosineSimilarity/SSIM/MSSSIM, rotation loss : QuatProdcut

- Second batch of runs were made with attack k value of 200 in order to check if the optimization scheme converged.

  - attack-k : 200, optical flow loss : MSE/CosineSimilarity/SSIM

- The next experiment tested the effects of the rotation criterion and optical flow criterion on each other. The tests preformed are:
  - Baseline run of APGD, QuatProduct for 200 iterations without an attack on optical flow.
  - APGD run with 200 iterations and optical flow SSIM/MSSSIM/CosineSimilarity without an attack on rotation.
  - APGD run with 200 iterations with both attacks: on optical flow and on rotation with same factor.
  - APGD run for 200 iterations with both optical flow criterion and rotation criterion with different factors. using quatProduct and different optical flow criterions.
    * rotation factor: 0.5, flow factor : 2
    * rotation factor: 2, flow factor : 0.5
    * rotation factor: None, flow factor : 2

- We proposed another option for optical flow criterion which uses multiple criterions. for example
  - MSSSIM+CosineSimilarity
  - MSSSIM+CosineSimilarity+MSE

5 Results and discussion

Below, in Figure 5.1, we can clearly see the improvement made to the previous work, which utilized only l_{RMS} and PGD. It is clear that most of the improvement came from utilizing APGD, but the improvement that came from adding the quaternion product loss term in not insignificant.

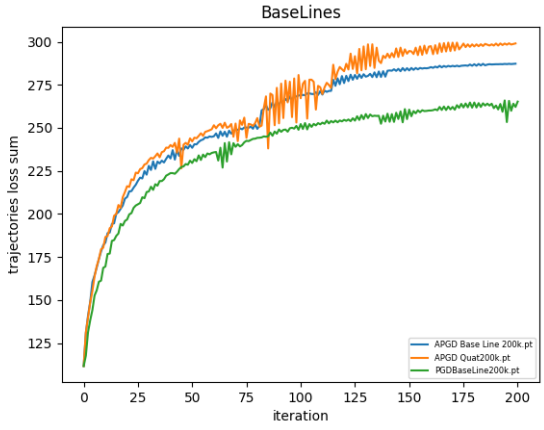


Figure 5.1: The optimization process of our best result (orange, l_{RMS}+quaterion product, with $\lambda_1 = 1$ and $\lambda_2 = 1$, without an additional loss term on the optical flow), compared with two baseline results: 1. l_{RMS} optimized with PGD (green); 2. l_{RMS} optimized with APGD (blue).

Best Overall Result

As seen in the figures above, our pertubation ,which was generated by APGD with Quat Product as rotation ciretiorn, has more of a knit pattern compared to the pertubation from the article [5] and from the pretubation to the right which was made using PGD with Cosine Similarity as optical flow criterion. There is higher similarity between 5.3 pertubation to 5.4 pertubation

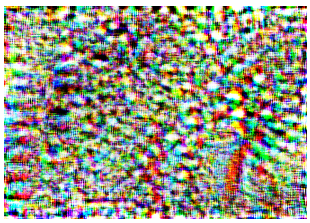


Figure 5.2: Our Best Pertubation Optimization RMS eval RMS

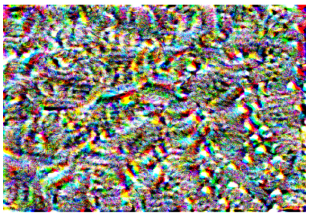


Figure 5.3: optimization RMS eval RMS [5]

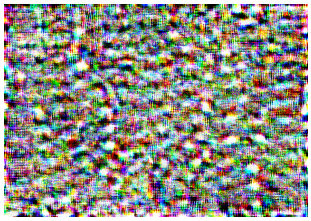
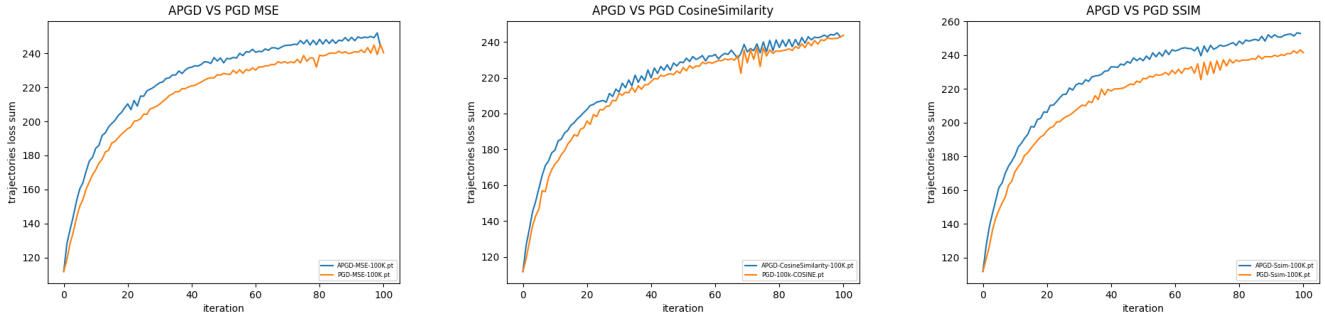


Figure 5.4: optimization RMS eval RMS PGD Cosine

5.1 Comparison of optimization schemes

In our project, we've compared three optimization schemes, PGD, APGD, and our own modified APGD. We've experimented using similar arguments in order to check which of the optimization schemes, APGD or PGD will result in better performance.



as seen in the graphs. The APGD optimization scheme has provided better loss in all test cases. In addition, we can see that the step size is not halved in the graph (usually marked by a "jump" in the loss). This shows that the number of iterations needs to be raised from 100 to 200 enable the optimization scheme to produce better results.

In their paper, Croce and Hein [2] compare the performance of PGD with and without momentum vs. APGD. The result of the comparison can be seen in Figure 5.5. Our results are in accordance with the observed behavior.

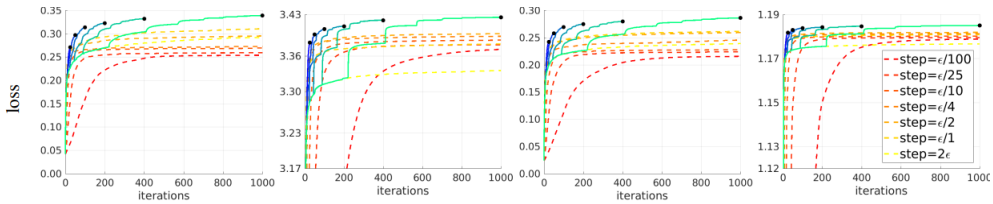


Figure 5.5: Comparison between APGD (solid lines) and PGD (dashed lines) behaviour, taken from [2]. The authors tested PGD with a momentum term, as used in APGD, always with a budget of 1000 iterations. APGD was tested with different budgets of iterations. It can be seen that APGD outperforms PGD with momentum for every budget of iterations in achieved loss.

Another result we've seen is that our Modified APGD converges faster to the same value. As seen in Figure 5.6, The Modified APGD uses the checkpoint to increase its step size and thus learning faster than the regular APGD.

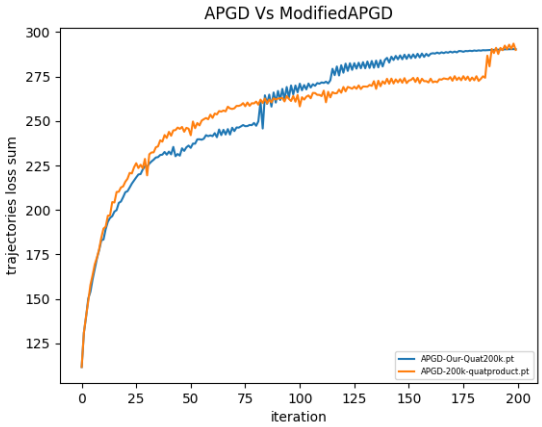
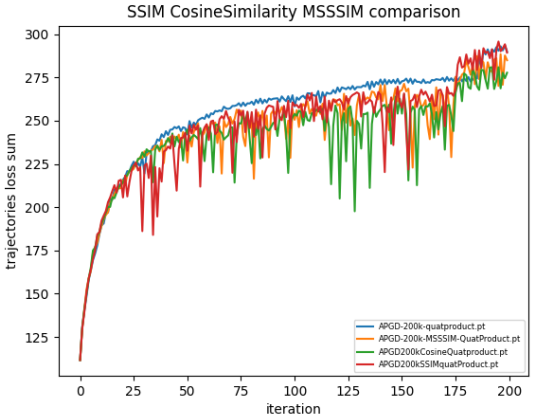


Figure 5.6: Our Modified APGD uses the checkpoint to increase its step size and thus learning faster than the regular APGD

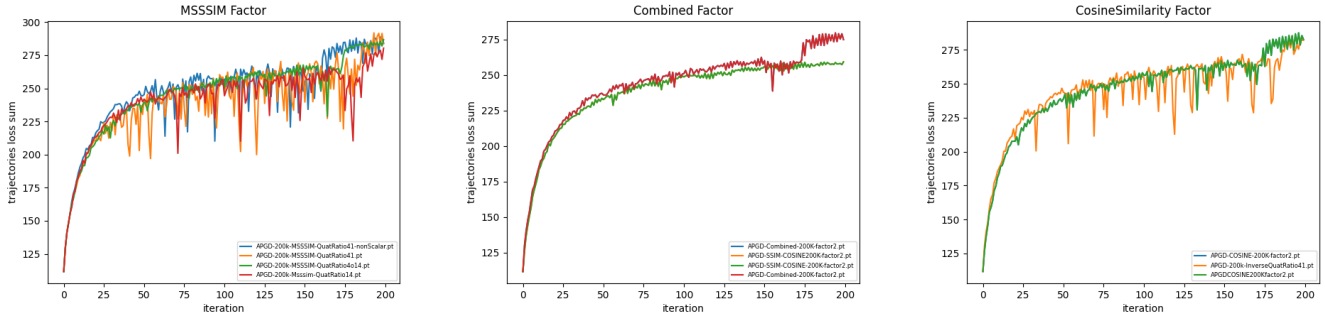
5.2 Comparison of optical flow attacks and their relative importance on the overall results



The three main optical flow attack loss functions we've checked are Cosine Similarity loss, SSIM loss and MSSSIM loss functions. as seen in the graph. there is an improvement when using all three of the optical flow attacks, but the biggest improvement is when attacking the rotation.

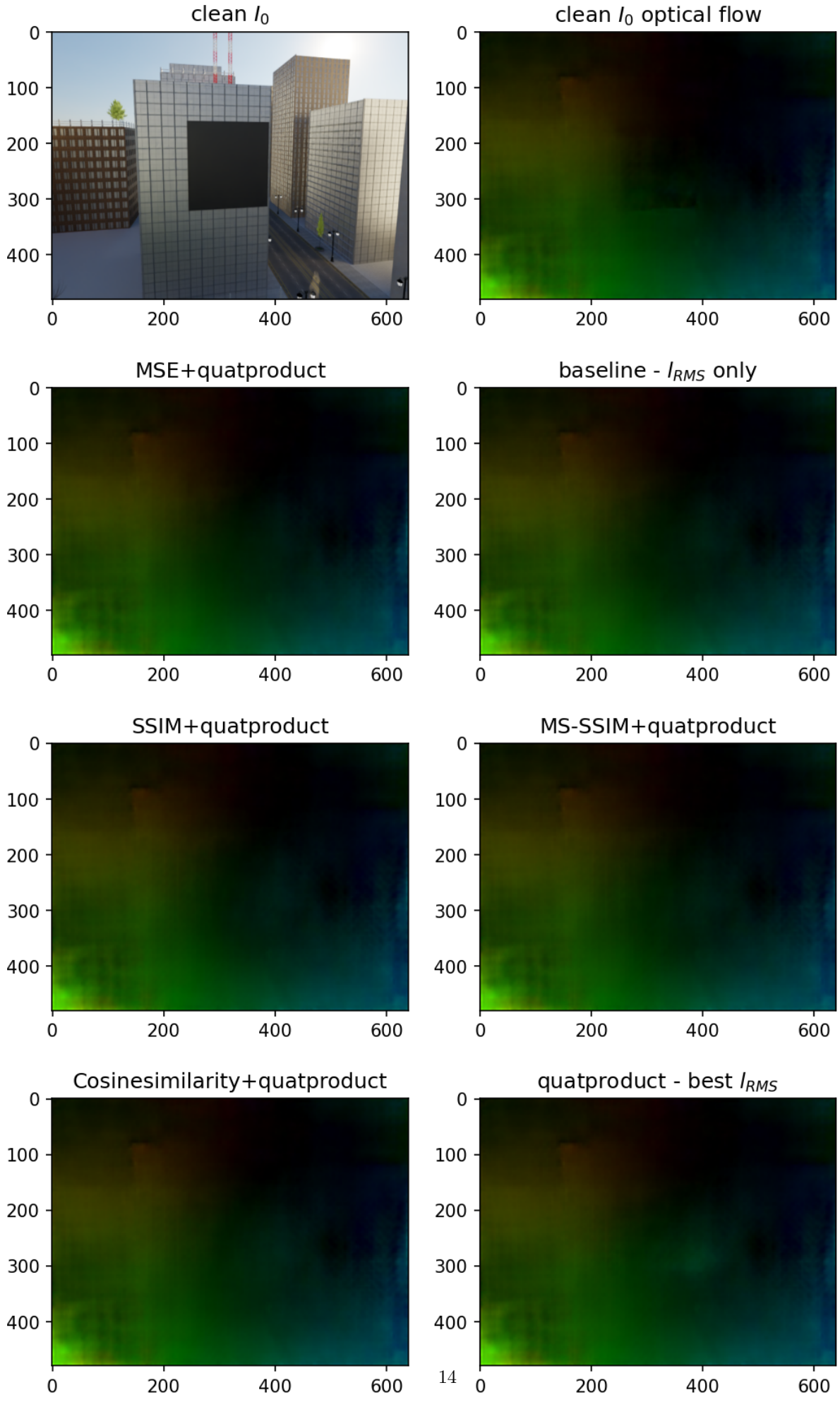
in hindsight. the use of attacks on optical flow in combination with an attack on the rotation backfires and does not improve the loss compared to using an attack on the rotation by itself.

5.3 Hyper Tuning



As seen from the experiments that used different factors for optical flow and rotation. Increasing the optical flow factor resulted in a more stable stable optimization scheme. (as seen in the CosineSimilarity Factor graph). In addition, incresaing the ratio between the optical flow factor and rotation factor gave a more stable result (as seen in Graph MSSSIM Factor by the green line that describes a ratio of $4:1$ between optical flow factor and rotation factor)

Optical Flow comparison



It can be seen that there is a difference between the optical flow produced using different criterion arguments. The optical flow produced by MS-SSIM+quadproduct is the closest one to the best perturbation received by using quad product as an attack on rotation without any attack on the optical flow. The effects of attacks on optical flow are negligible compared to the attack on rotation. Furthermore, the effects on the optical flow are limited to the area around the patch which doesn't effect all of the optical flow provided and thus limits its effect on the outcome. This effect was also seen in [6] and was expected.

5.4 Conclusions

- Extending the optimization scheme from PGD to APGD improved the adversarial perturbation by a large margin. Its impact is great when compared to PGD without momentum, as seen in the graphs and in [2].
- Attacking rotation using the quaternion product loss also improves the perturbations. The attack on the rotation made a bigger difference than the attack on the optical flow. as seen in the optical flow comparison, both criterion affect the optical flow and as it seems there is no subset of variable that improve both the rotation criterion and the optical flow criterion simultaneously
- Attacking optical flow directly makes negligible difference. Since the only attack we can make are inside the patch there is low effect on the total optical flow seen by the drone.

5.5 Future Research

Possible future research directions we would like to explore

- Add a loss function on the layers of P_ϕ, e.g on the outputs of the layers of Resnet50, which is more susceptible to attacks. Benz et al. [1]
- Preprocess a batch instead of using a randomly generated batch for the start of the run,
  The preprocessed batch will try to mimic the illusion of "depth" like the drawing of Wild Coyote.

- Adjust the checkpoints and step size adjustment in the ModifiedAPGD, to see if we can get better results.
- Check why the attack on both optical flow and rotation hampered the training process. A reasonable guess is that a perturbation that will maximize the various optical flow losses may not necessarily maximize l_{RMS}. E.g the sample space of perturbations that can maximize the optical flow loss is larger than the sample space of perturbations that can maximize l_{RMS}.
- Research new ways to improve the effect of the optical flow loss on the adversarial attack on TartanVO.

References

[1] P. Benz, C. Zhang, and I. S. Kweon. Batch normalization increases adversarial vulnerability and decreases adversarial transferability: A non-robust feature perspective supplementary material. *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2020.

[2] F. Croce and M. Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. *ICML*, 2020.

[3] K. He, X. Zhang, S. Ren, and J. Sum. Deep residual learning for image recognition. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *ICLR*, 2017.

[5] Y. Nemcovsky, M. Yaakoby, A. M. Bronstein, and C. Baskin. Physical passive patch adversarial attacks on visual odometry systems. *ArXiv*, 2022.

[6] A. Ranjan, J. Janai, A. Geiger, and M. J. Black. Attacking optical flow. *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[7] S. Schrodi, T. Saikia, and T.Brox. Towards understanding adversarial robustness of optical flow networks supplementary material. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[8] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[9] W. Wang, Y. Hu, and S. Scherer. Tartanvo: A generalizable learning-based vo. *CoRL*, 2020.

[10] Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multi-scale structural similarity for image quality assessment. *The Thrity-Seventh Asilomar Conference on Signals, Systems and Computers*, 2003.

[11] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 2004.