Mathematical foundations of signal processing

COM-504

Diffuser Cam Project Report

https://github.com/alec-flowers/DiffuserCam

Ludovic CHAN - 300598

Alec FLOWERS - 321786

Albert Kjoller JACOBSEN - 345050

Matthieu VERDET - 272284

Contents

1 Motivation

In the context of the tools presented in the course *COM-504 Mathematical Foundations of Signal Processing* and their application, this project proposes a hands-on introduction to discrete Linear Inverse Problems: a pixelized signal $\alpha \in \mathbb{R}^N$ is probed with a sensing device, resulting in a noisy finite-dimensional signal $\mathbf{y} \in \mathbb{R}^L$. The sensing device is modeled with a linear operator $\mathbf{H} \in \mathbb{R}^{L \times N}$. The goal is to recover the coefficients α from these noisy measurements and get values close to the real signal.
In some cases, it is possible to perform a straight-forward inversion. However, in our situation, a more reliable solution is to regularize the inverse problem with a penalised optimisation:

$$\min_{\alpha \in \mathbf{R}^N} F(\mathbf{y}, \mathbf{H}\alpha) + \lambda \mathcal{R}(\alpha)$$

where $F : \mathbb{R}^L \times \mathbb{R}^L \to \mathbb{R}_+ \cup \{+\infty\}$ is called the *cost functional* or the *data-fidelity functional*, $\mathcal{R} : \mathbb{R}^N \to \mathbb{R}_+ \cup \{+\infty\}$ is called the *regularisation functional* or the *penalty functional*, and $\lambda \in \mathbb{R}_+^*$ is a parameter weighting the influence of the penalty term on the data-fidelity term.

Recovering the α coefficients using this technique avoids several problems. Indeed, when $\mathbf{H}$ is not surjective, there might be no solution; when $\mathbf{H}$ is injective there might be more than one solution. In other words, when $\mathbf{H}$ is not bijective, the existence and the unicity of the solution are not guaranteed. Moreover, $\mathbf{H}$ might be ill-conditioned, leading to numerical instabilities.

This approach offers other advantages such as the flexibility to deal with noisy inputs and the regularity of the original data. According to the situation, we can choose different cost or penalty functionals and decide on their relative influence in the minimization calculation.
Although, good optimization results most often require very heavy computations, taking a significant time.

In the context of lensless photography systems via linear inverse reconstruction optimization, we explore and test various combinations of functionals and algorithms to discuss their performance.

1.1 DiffuserCam

In this project, the original data is a color image probed by a lensless camera: the lens is removed and replaced by a transparent adhesive tape as a *diffuser lens*. With the data collected, an approximation of the original image is reconstructed. Therefore, it is a cheap, lightweight, and very compact camera, as we replace heavy and costly lens systems for a piece of adhesive tape being only a few micrometers thick.

Reconstructing the images coming from the lensless camera data is a linear inverse problem: the original images α are reconstructed using the (flattened) measurements $\mathbf{y}$ of the lensless camera and the operator $\mathbf{H}$. Assuming the system is linear and shift-invariant, the $\mathbf{H}$ operator can be deduced by using only one *point spread function* (PSF). The PSF is measured with a tiny white light source at ≈ 40 cm from the camera. Then the operator $\mathbf{H}$ is built as a 2D convolution taking for filtering the PSF.
This project aims to solve the linear inverse problem for a set of custom chosen images captured with the lensless DiffuserCam by investigating different reconstruction algorithms. A comparison to the ADMM baseline reconstruction approach in terms of four different metrics as well as the processing time of reconstruction will be conducted.

2 Data Acquisition

2.1 Dataset

To evaluate the performances of the algorithms, the following images were chosen (Figures 1 to 7):

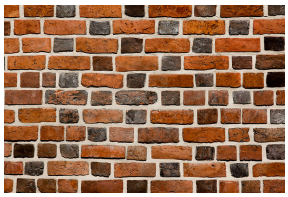


Figure 1: Image1

Figure 2: Image3

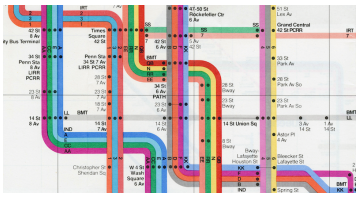


Figure 3: Image4

Figure 4: Image5

Figure 5: Image6

Figure 6: Image7

Figure 7: Image8

We wanted to select images with a wide range of properties: different color sets, contrasts, depth-of-fields, textures, etc... We expect that unique visual properties might impact differently reconstruction algorithms performance. Thus, each of these images was chosen with prior beliefs about penalization in mind as the hypothesized best approach:

- *Image1*: is a brick wall with very clear gaps between bricks. The brick colors vary for each brick but the wall, in general, is considered piece-wise constant. For this reason, the hypothesis was that TV regularisation, i.e. penalizing the ℓ_1 norm, would yield good reconstruction results.
- *Image3*: has gradual changes at the bottom and sharper changes at the top, thus the idea was to see how approaches could deal with this blending and sharp color changes. The initial guess was that using the ℓ_2 norm as the regularisation term would perform well on this image.
- *Image4*: is a subway map with a mixture of big large features and tiny font. The idea was to see if the reconstruction could give readable text. The initial guess was that the Huber norm, which is a *smoothed* ℓ_1 norm (best properties of ℓ_1 and ℓ_2 loss) would be the better choice of regularization in order to detect the sparsity of text, the smooth subway lines and the sharp transition between the background, the text, and the subway lines.
- *Image5*: is a grayscale image of palm trees under a sky of stars. As the night sky is mostly black, with sharp stars that represent a lot of details, the hypothesis was that a sparse solution, penalizing the ℓ_1 norm would be the best candidate for reconstructing the stars in this image. This image is a good test to evaluate the level of detail the algorithms can recover.
- *Image6*: is a grayscale image with sharp transitions between the sky, the mountains, and the boat. The idea was that a sparse solution would be a good candidate for reconstruction. The initial guess was that the ℓ_1 norm would work well here.
- *Image7*: is a natural landscape with color. The idea was to test whether the difference between land and sea could be easily distinguished. The initial guess was that a solution prioritizing smoothness would be good at this, i.e. using the ℓ_2 norm as the penalization term.
- *Image8*: is an image depicting flowers in focus in the foreground with a blurry background. The colors for the focused foreground as well as the blurred background are similar for which reason it was interesting to investigate which reconstruction approach would be the better candidate, ie. the problem that would give the best separation between the in-focus areas and out-of-focus zones. In contrast to the other pictures no strong prior thoughts were made on this image - it was mainly chosen for investigational purposes.

2.2 Measurements

We made two kinds of measurements: 1) to collect the PSF and 2) to capture the chosen images with the DiffuserCam. The setup is described from the following Medium tutorial. The PSF is measured by capturing a LED point light source radiating from a tiny pinhole on the side of a cardboard box. This point source was located 40 cm from the DiffuserCam in a dark room to minimize stray light. To capture the selected images, the only difference was to replace the cardboard box with a computer monitor displaying the images.
Measurements of both PSF and images were raw Bayer captures from the DiffuserCam. This way, we make sure to avoid any type of unwanted pre-processing.

2.3 PSF Analysis

On the left subplot of Figure 8, we see the RGB color-corrected capture of the PSF - the typical caustic pattern is visible. The pixel values of the histogram in Figure 8 show that the white balance of the color correction is fine. We have three color channels with a quasi-similar dynamic range. We also notice no saturation towards high pixel values.

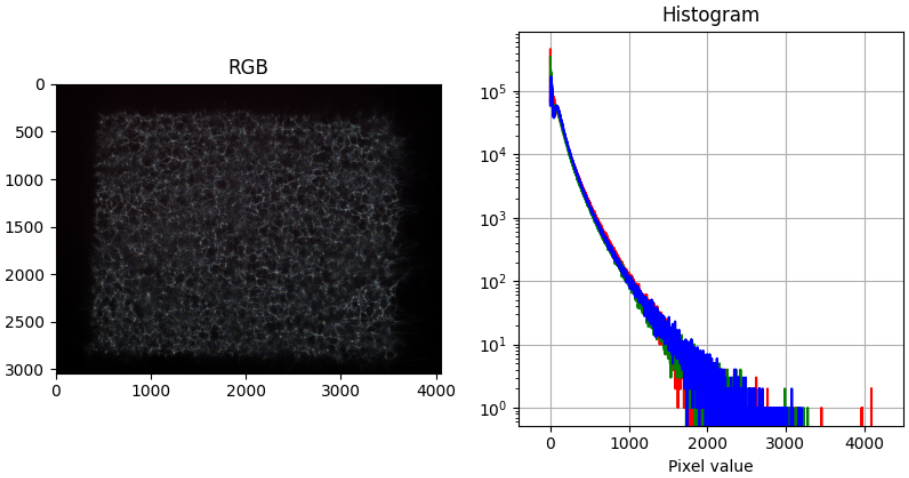


Figure 8: Color corrected Point Spread Function (PSF) taken with DiffuserCam

By analyzing cross-sections of PSF's channels auto-correlation (Figure 9), we obtain a quantitative metric to evaluate PSF focus. This is determined by examining the support/width of the autocorrelations peaks[1]. The closer the dotted horizontal lines are, the sharper the peak is for a certain threshold of decibels. Sub-plots below, show that the PSF could be more focused, although, it might be sufficient for our project. In the worst case, this could harm the reconstruction quality as reducing optimal focus will cause more blur. Another aspect is to check whether the PSF has a sufficient amount of directional filter. Visually, the caustic patterns of the PSF 8 appear sharp and have randomly oriented lines, we can consider it is a proper candidate.

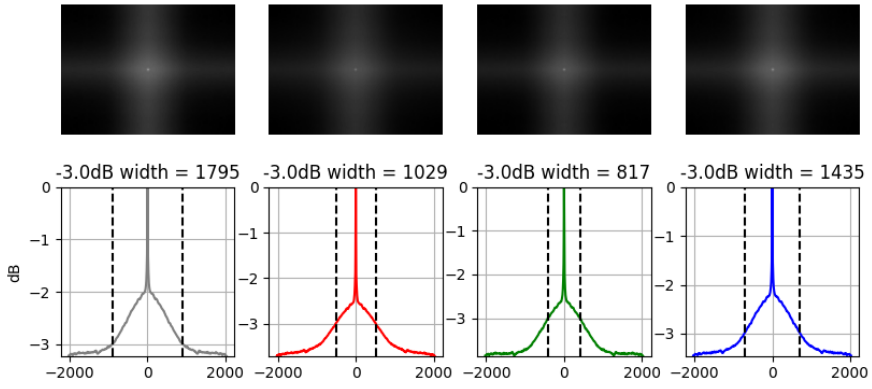


Figure 9: Cross-section plots of PSFs of grayscale, red, green and blue channel autocorrelations

3 Mathematical and Technical Reasoning

3.1 Optimisation problems

Various priors are implemented in order to have the best reconstruction depending on the situation, ie. depending on the regularity of the image to be reconstructed.

3.1.1 Ridge Regression

The Ridge regression is a penalised least-square problem with Tikhonov regularisation, ie. the penalty term is the squared ℓ_2-norm. The data-fidelity term is $F(\mathbf{y}, \mathbf{H}\alpha) = 1/2\|\mathbf{y} - \mathbf{H}\alpha\|_2^2$ and the penalty term is $\mathcal{R}(\alpha) = \|\alpha\|_2^2$:

$$\min_{\alpha \in \mathbb{R}^N} \frac{1}{2}\|\mathbf{y} - \mathbf{H}\alpha\|_2^2 + \lambda\|\alpha\|_2^2$$

This problem is solved using the APGD algorithm and the implemented method is called *ridge*.

3.1.2 LASSO problem

In a LASSO problem, the data-fidelity term is $F(\mathbf{y}, \mathbf{H}\alpha) = 1/2\|\mathbf{y} - \mathbf{H}\alpha\|_2^2$ and the penalty term is $\mathcal{R}(\alpha) = \|\alpha\|_1$:

$$\min_{\alpha \in \mathbb{R}^N} \frac{1}{2}\|\mathbf{y} - \mathbf{H}\alpha\|_2^2 + \lambda\|\alpha\|_1$$

This problem is also solved using the APGD algorithm and the implemented method is called *lasso*.

3.1.3 Non-negative least-square

In a non-negative least-square problem, the regularisation functional forces the solution to be positive. The data-fidelity term is still $F(\mathbf{y}, \mathbf{H}\alpha) = 1/2\|\mathbf{y} - \mathbf{H}\alpha\|_2^2$ but the penalty functional is:

$$\mathcal{R}(\alpha) = \begin{cases} 0, & \alpha \in \mathbb{R}_+^N \\ +\infty, & \text{otherwise} \end{cases}$$

so that the problem becomes:

$$\min_{\alpha \in \mathbb{R}_+^N} \frac{1}{2}\|\mathbf{y} - \mathbf{H}\alpha\|_2^2$$

Again, this problem is solved using the APGD algorithm and the implemented algorithm is called *nnls*.

3.1.4 Generalised LASSO problem

For the Generalised LASSO problem, the ℓ_1-norm penalty is used together with the type-2 Discrete Cosine Transform, ie. $\mathcal{R}(\alpha) = \|\text{DCT2}(\mathbf{x})\|_1$, and the problem becomes:

$$\min_{\mathbf{x} \in \mathbb{R}^N} \frac{1}{2}\|\mathbf{y} - \mathbf{H}\mathbf{x}\|_2^2 + \lambda\|\text{DCT2}(\mathbf{x})\|_1$$

where DCT2 is the type 2 Discrete Cosine Transform orthonormalized.
This problem cannot be solved with the APGD algorithm because $\mathcal{R}(\alpha)$ is not proximable. But the DCT2 operator is orthonormal. Then the change of variable $\tilde{\mathbf{x}} = \text{DCT2}(\mathbf{x})$ is applicable and it yields:

$$\min_{\tilde{\mathbf{x}} \in \mathbb{R}^N} \frac{1}{2}\|\mathbf{y} - \mathbf{H}\left(\text{DCT2}^{-1}(\tilde{\mathbf{x}})\right)\|_2^2 + \lambda\|\tilde{\mathbf{x}}\|_1$$

where DCT2^{-1} is the inverse of the DCT2 operator. This last formulation is a regular LASSO problem. It can be solved using the APGD algorithm for $\tilde{\mathbf{x}}$. Then making the inverse change of variable, the solution of the initial problem is $\mathbf{x} = \text{DCT2}^{-1}(\tilde{\mathbf{x}})$. The implemented method is called *glasso*.

3.1.5 Penalised least-square with Total Variation and non-negativity

A Total Variation (TV) $\lambda\|\mathbf{D}\alpha\|_1$ is introduced in the penalty term of the problem in section 3.1.3, where the operator $\mathbf{D}$ is the discrete gradient computed using finite-differences. The penalty term becomes:

$$\mathcal{R}(\alpha) = \begin{cases} 0 + \lambda\|\mathbf{D}\alpha\|_1, & \alpha \in \mathbb{R}_+^N \\ +\infty, & \text{otherwise} \end{cases}$$

and the problem is:

$$\min_{\alpha \in \mathbb{R}_+^N} \frac{1}{2}\|\mathbf{y} - \mathbf{H}\alpha\|_2^2 + \lambda\|\mathbf{D}\alpha\|_1$$

This problem cannot be solved with the APGD algorithm but is solved with the Primal-Dual Splitting (PDS) algorithm instead. The method is called *pls*.

3.1.6 Huber Norm

Here, the ℓ_1-norm is replaced by the Huber norm $\|\cdot\|_{H,\delta}$, where $\delta > 0$ is a parameter. The Huber norm is defined as:

$$\|\mathbf{z}\|_{H,\delta} = \sum_{m=1}^{M} h_\delta(z_m), \quad \forall z_m \in \mathbb{R}^M \qquad \text{with } h_\delta(z_m) = \begin{cases} 1/2 z_m^2, & |z_m| \leq \delta \\ \delta(|z_m| - \delta/2), & \text{otherwise} \end{cases}$$

and the problem, called *pls_huber*, becomes:

$$\min_{\alpha \in \mathbb{R}_+^N} \frac{1}{2}\|\mathbf{y} - \mathbf{H}\alpha\|_2^2 + \lambda\|\mathbf{D}\alpha\|_{H,\delta}$$

Now, let's prove that the Huber norm is differentiable:

$$(\nabla\|\mathbf{z}\|_{H,\delta})_k = \frac{\partial}{\partial z_k}\|\mathbf{z}\|_{H,\delta} = \frac{\partial}{\partial z_k}\sum_{m=1}^{M} h_\delta(z_m), \quad k = 1, ..., M$$

where z_m is the m^{th} component of $\mathbf{z} \in \mathbb{R}^M$. As $h_\delta : \mathbb{R} \to \mathbb{R}$:

$$(\nabla\|\mathbf{z}\|_{H,\delta})_k = \sum_{m=1}^{M} \frac{d}{dz_k} h_\delta(z_m) = \frac{d}{dz_k} h_\delta(z_k)$$

Now, h_δ is defined as:

$$h_\delta(z_k) = \begin{cases} 1/2 z_k^2, & |z_k| \leq \delta \\ \delta(|z_k| - \delta/2), & \text{otherwise} \end{cases}$$

Then it yields :

$$\frac{d}{dz_k} h_\delta(z_k) = \begin{cases} z_k, & |z_k| \leq \delta \\ \delta, & (z_k \geq 0) \cap (|z_k| > \delta) \\ -\delta, & (z_k < 0) \cap (|z_k| > \delta) \end{cases} = \begin{cases} z_k, & |z_k| \leq \delta \\ \delta, & z_k > \delta \\ -\delta, & z_k < -\delta \end{cases}$$

dh_δ/dz_k is continuous on $\mathbb{R}$, particularly at $z_k = \pm\delta$ where the value of the image is z_m. This is true for $k = 1, ..., M$. Then, $\nabla\|\mathbf{z}\|_{H,\delta}$ is differentiable since each of its component is well defined on $\mathbb{R}^M$.
Now, let's prove that $\nabla\|\mathbf{z}\|_{H,\delta}$ is 1-Lipschitz continuous. Let's consider two points $\mathbf{z^{(1)}}, \mathbf{z^{(2)}} \in \mathbb{R}^M$:

$$\left\|\nabla\|\mathbf{z^1}\|_{H,\delta} - \nabla\|\mathbf{z^2}\|_{H,\delta}\right\|_2 = \left\| \begin{matrix} \frac{\partial}{\partial z_1^{(1)}} \sum_{m=1}^{M} h_\delta(z_m^{(1)}) - \frac{\partial}{\partial z_1^{(2)}} \sum_{m=1}^{M} h_\delta(z_m^{(2)}) \\ \vdots \\ \frac{\partial}{\partial z_M^{(1)}} \sum_{m=1}^{M} h_\delta(z_m^{(1)}) - \frac{\partial}{\partial z_M^{(2)}} \sum_{m=1}^{M} h_\delta(z_m^{(2)}) \end{matrix} \right\|_2 = \left\| \begin{matrix} \frac{d}{dz_1^{(1)}} h_\delta(z_1^{(1)}) - \frac{d}{dz_1^{(2)}} h_\delta(z_1^{(2)}) \\ \vdots \\ \frac{d}{dz_M^{(1)}} h_\delta(z_M^{(1)}) - \frac{d}{dz_M^{(2)}} h_\delta(z_M^{(2)}) \end{matrix} \right\|_2$$

The k^{th} line squared writes:

$$\left(\frac{d}{dz_k^{(1)}} h_\delta(z_k^{(1)}) - \frac{d}{dz_k^{(2)}} h_\delta(z_k^{(2)})\right)^2 = \begin{cases} (z_k^1 - z_k^{(2)})^2, & |z_k^{(1)}|, |z_k^{(2)|} \leq \delta \\ (z_k^{(1)} - \delta)^2, & |z_k^{(1)}| \leq \delta, |z_k^{(2)}| > \delta \\ (-z_k^{(2)} + \delta)^2, & |z_k^{(2)}| \leq \delta, |z_k^{(1)}| > \delta \\ 0, & |z_k^{(1)}|, |z_k^{(2)}| > \delta \end{cases} \leq (z_k^{(1)} - z_k^{(2)})^2 \quad \forall z_k^{(1)}, z_k^{(2)} \in \mathbb{R}$$

Finally, as $\sqrt{\cdot}$ is increasing on $\mathbb{R}^+$:

$$\left\|\nabla\|\mathbf{z^{(1)}}\|_{H,\delta} - \nabla\|\mathbf{z^{(2)}}\|_{H,\delta}\right\|_2 \leq \sqrt{\sum_{k=1}^{M} (z_k^{(1)} - z_k^{2})^{(2)}} = 1\|\mathbf{z^{(1)}} - \mathbf{z^{(2)}}\|_2$$

3.2 Operator Optimization

We saw an opportunity to solve problems of both questions 8 and 9 while planning the new convolution class. We explored ADMM and the original ConvolveND implementations to get inspiration.
To correctly implement the Linear Operator, we have to write two operations:

- The main function operation: convolution.
- The adjoint function operation: correlation.

We looked at codes of `scipy.signal.fftconvolve(...)` and `scipy.signal.correlate(...)` present in the pylops Convolve2D class. We realized that `fftconvolve` offers the possibility to run convolution on multiple layers with a single pass thanks to the `axes` argument. Therefore, it solves the RGB problem of question 8. We judged it good to let scipy optimize our computations rather than running convolution for each channel. Conversely, `correlate` doesn't offer such feature, we had to dig deeper in scipy code to break down the code once again:

- The essential functions that uses `fftconvolve` and `correlate` are `scipy.fft.rfftn(...)` and `scipy.fft.irfftn(...)`. These functions have the `axes` argument, thus handle multi-channels images.
- `correlate` calls `fftconvolve` with the reverse conjugate of the second argument. In other words, solving convolve is solving correlate as well.

To tackle the problem of question 9, and leverage the unique computation of the Fourier transform of PSF, we had to break down these operations anyway.

Our implementation:

- At initialization, we only ask for the PSF. We store two Fourier transforms: the `rfftn` of PSF and reversed conjugate PSF. We also store all shaping, padding, and slicing attributes necessary to mimic `fftconvolve` and `correlate` computations.
- The methods `__call__` and `adjoint` are receiving a flatten structure having the same flatten size as PSF. We reshape it, apply `rfftn`, multiply it with the respective PSF/reversed-conjugate-PSF, and apply `irfftn` on the result. After slicing, and flattening we return the new structure.

To summary, we observed the code of pycsou, pylops, and scipy through and through, and pruned every possible branch, optional features, or unnecessary code. We verified that we were having the quasi-exact computations values of Convolve2D (with `np.float64`, we get norm and mean differences of at most 1e-8 down to 0.0 . More information on the notebook in the repository). It handles both grayscale and color images assuming that PSF, estimate array, and measurement array are all of the same sizes.
Comparing computations with Convolve2D, just calling `__call__` and `adjoint` don't show significant differences.
However, this class delivers enhanced performance during optimization. We saw speedup on our small benchmark going from 1.3 up to 3.4 depending on the task and the hardware.

4 Evaluation pipeline

All reconstruction approaches were evaluated using the same evaluation pipeline. As a start, each of the images captured with the DiffuserCam was color-corrected in a custom manner. These color-corrected DiffuserCam images were then loaded after which the background of the setting - determined from the PSF measurement - was subtracted. To avoid possibly negative values after subtracting the background, the image was clipped to a range between 0 and the maximum pixel value of the image. The final step of the raw image processing was to normalize the images.

After having processed a DiffuserCam image, it was fed to one of the reconstruction algorithms. All reconstruction approaches had a hyperparameter to tune and the number of iterations was to be determined - more on that in the section 4.2. The area of interest in the reconstructed images was only covering a minor part of the image for which reason cropping was needed. Since the selected reconstruction algorithms were to be compared to ADMM reconstructions, the cropping for each image was found by determining the area of interest as it occurred in the ADMM reconstruction of the respective image.

To compare the various reconstruction approaches to ADMM it was necessary to evaluate reconstructions with respect to the ground truth image. Since the image captures were not done with a physical dual capture setup - meaning that lensed images were not captured when capturing the DiffuserCam images - the image displayed on the screen was chosen as the ground truth image and had to be resized (and thereby also downsampled) to the same shape as its respective cropped reconstruction. To make sure cropped reconstructions and resized ground truth images were also comparable to pixel values, a scaling was applied to all pairs of images when calculating the metrics. This scaling was simply handled by dividing the reconstruction and ground truth images by their respective maximum pixel value.

4.1 Metrics

Reconstructed DiffuserCam images were evaluated using the following four metrics:

- **MSE** is the mean squared error and is computed - as the name reveals - by averaging the squared differences between the estimated pixel values from the reconstructions and the respective pixel values from the ground truth image. It characterizes more the color value than the image structure. Better is closer to 0.
- **PSNR** is the peak signal-to-noise ratio and measures the ratio of the maximum possible power compared to the power of noise influencing the reconstruction. The metric is on the decibel scale and is computed by expressing the ratio between the squared possible maximum value and the MSE on a logarithmic scale. Like MSE, PSNR is a pixel-wise metric and carries similar defaults. Better is higher.
- **SSIM** is the structural similarity index measure and is used to predict how the structural quality of an image is perceived. It uses a change in structural information to determine image deterioration. SSIM differs from MSE-based metrics as it estimates absolute errors and non-pixel-wise. Better is closer to 1.
- **LPIPS** is for Learned Perceptual Image Patch Similarity and differ from the previous metrics by being learned as deep features through a neural network structure. The motivation for using LPIPS is that SSIM and PSNR are simple and may not be able to properly describe various nuances of human perception. Better is closer to 0. [2], [3]

For each reconstructed image, these four metrics as well as the setup and processing time were saved in a log file.

4.2 Hyperparameter Tuning

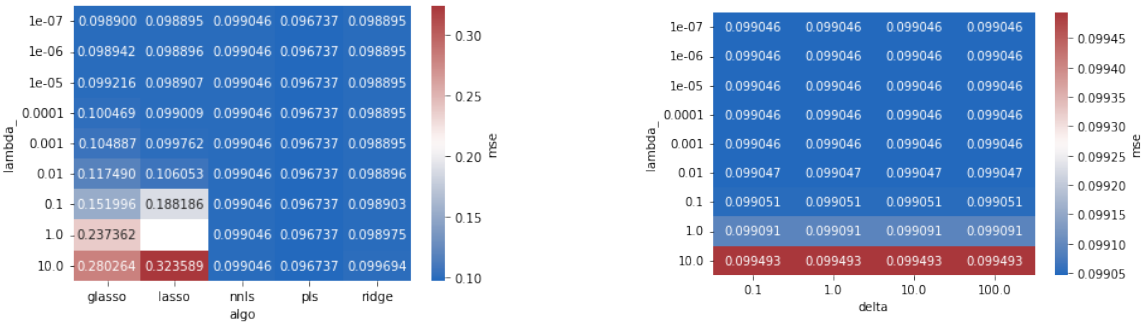
In all the approaches there is a hyperparameter to tune. λ controls the amount of regularization that comes from our penalty term. As λ approaches 0 we get back to the original least square problem. In the PLS HUBER problem, we also have an additional parameter δ to select. δ controls where the Huber norm transitions from an ℓ_1 loss to an ℓ_2 loss. There are other hyperparameters related to APGD and PDS algorithms, however, we rely on the Pycsou framework to choose these optimally for us.

In order to optimally choose λ and δ we used a grid search methodology. We ran the optimization problems multiple times while varying the hyperparameters. In Table 1 we show the combination of algorithms and

parameters that we ran. One limitation of this methodology is the compute power and time involved. We had to run 81 simulations over the 7 photos with an average time of 65 seconds for each photo which totals 10 hours of runtime. Therefore we only ran 100 iterations per optimization problem.

algo	[ridge, lasso, glasso, nnls, pls, pls_huber]
lambda	[1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, .1, 1, 10]
delta	[0.1, 1, 10, 100]
iter	100

Table 1: Set of hyperparams used in the grid search. Note delta was only used when running PLS-HUBER.



(a) All other hyperparameter results

(b) PLS-HUBER hyperparameter results

Figure 10: Grid search results over our approaches looking at the mean MSE over our 7 images.

Our results were not that helpful in choosing our final parameters. We can see from the table that the values are very similar after $1e-1$ One thought is that our range of lambda was too wide and after $1e-1$ the penalty term is so small that it has little to no effect on the optimization problem. This means that we have to narrow in on the proper grid search area to see any significant results. Another thought is that 100 iterations were not enough time for the approaches to differentiate themselves and we would need to run these tests for longer. The charts for the other metrics tell the same story so we omit them for brevity. We decided to re-run a few of these tests while narrowing our focus to address these thoughts.

In this next test we chose to run only on image 1, but for 1000 iterations and over a more compact set of lambda. See Table 2 for the hyperparameters. By focusing on one image we were able to significantly increase the number of iterations.

algo	[pls_huber, pls, lasso]
lambda	[0.1, 0.2, 0.3]
delta	[1]
iter	1000

Table 2: New and narrower grid search. Set of hyperparameters used for reconstruction of image 1.

Given our earlier results, it is not surprising that lasso performs so poorly with such a high λ. We wanted to see if given more iterations if a higher penalty term would start beating other terms with such small penalties. However, this is not the case, and armed with this information when we attempt the final results we set the λ for GLASSO and LASSO which were sensitive to a high λ very low. PLS is surprising as with 1000 iterations the MSE is larger than the average with only 100 iterations. After closer inspection, this is because image 1 is a tough image for PLS and with only 100 iterations it gives an MSE of .14. This discovery is why we report both average and individual image metrics in results because the average hides a lot of the image-specific details. PLS-HUBER performs the best of the three on this task. It is not surprising it is better than PLS as it is a faster version given that the norm is differential. What is surprising is how λ influences LASSO and PLS-HUBER so differently.

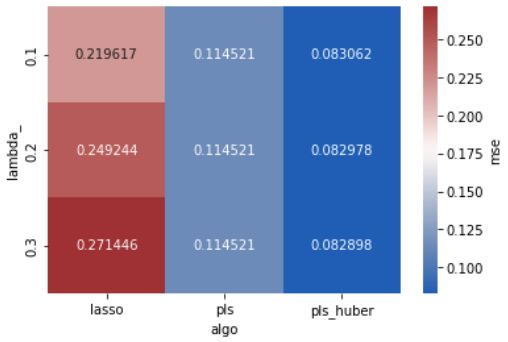


Figure 11: Metrics for our grid search over a narrower set of lambda, and higher iterations, on image 1.

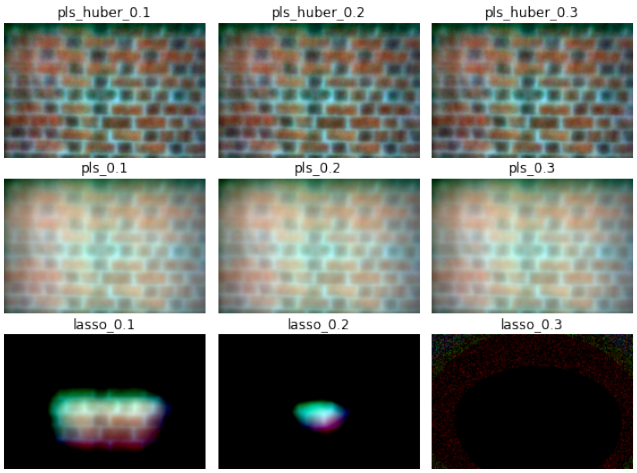


Figure 12: Results of our grid search over a narrower set of lambda, higher iterations, on image 1.

5 Results

For our comparison against the baseline, the final parameters we chose are in Table 3.

	λ
RIDGE	$0.001 = 1 \cdot 10^{-3}$
LASSO	$0.000001 = 1 \cdot 10^{-6}$
GLASSO	$0.0000001 = 1 \cdot 10^{-7}$

Table 3: Final hyperparameter values

Using the optimal values for λ (see Table 3) the reconstruction algorithms were run over 1000 iterations. Based on a pilot run of ADMM it was determined that running ADMM for longer does not help increase its reconstructions - it just ends up alternating between better and worse results. This pilot run revealed that 20 iterations gave the optimal scores across the chosen set of images for which reason ADMM with 20 iterations were used as the baseline. Average results of the baseline reconstructions, as well as reconstructions using RIDGE, LASSO, NNLS, and GLASSO over all 7 of the previously described images, can be seen in Table 4. Also, due to a technical problem discovered too late to account for, the PLS-based approaches were not included in the evaluation run...

Table 4: Average results over the 7 photos with ADMM as a baseline. Best score is highlighted in blue.

	MSE	PSNR	SSIM	LPIPS	Time (s)
ADMM	0.087789	11.509514	0.346748	0.639499	15.1
GLASSO	0.061795	13.204670	0.481097	0.618561	1458.2
LASSO	0.061943	13.195581	0.479841	0.618768	1214.3
NNLS	0.075608	12.422622	0.412996	0.615286	1167.4
RIDGE	0.061894	13.199550	0.480487	0.618882	1402.5

What is clear from the average results is that the ADMM baseline algorithm is outperformed on all metrics by all the other reconstruction algorithms except for the processing time. With an average processing time of only 15.1 seconds, the ADMM approach is approx. 77 times faster at computing 20 iterations than the 1000 iterations that the other reconstruction algorithms use - and yet it obtains recognizable reconstructions (which is seen in Figure 13). Even though the GLASSO algorithm is the best performer wrt. the MSE, PSNR, and SSIM, the other non-baseline algorithms obtain similar results to GLASSO. Only for the LPIPS metric, the NNLS algorithm is the better approach which is also only slightly better than the other non-baseline algorithms.

It is of interest to examine the reconstruction algorithms and their performances on an individual image level to determine if some of our prior beliefs were correct. This is done both qualitatively/subjectively by looking at the reconstructions in Figure 13 and quantitatively/objectively by looking at the metrics for each image in the tables found in Figure 14-20.

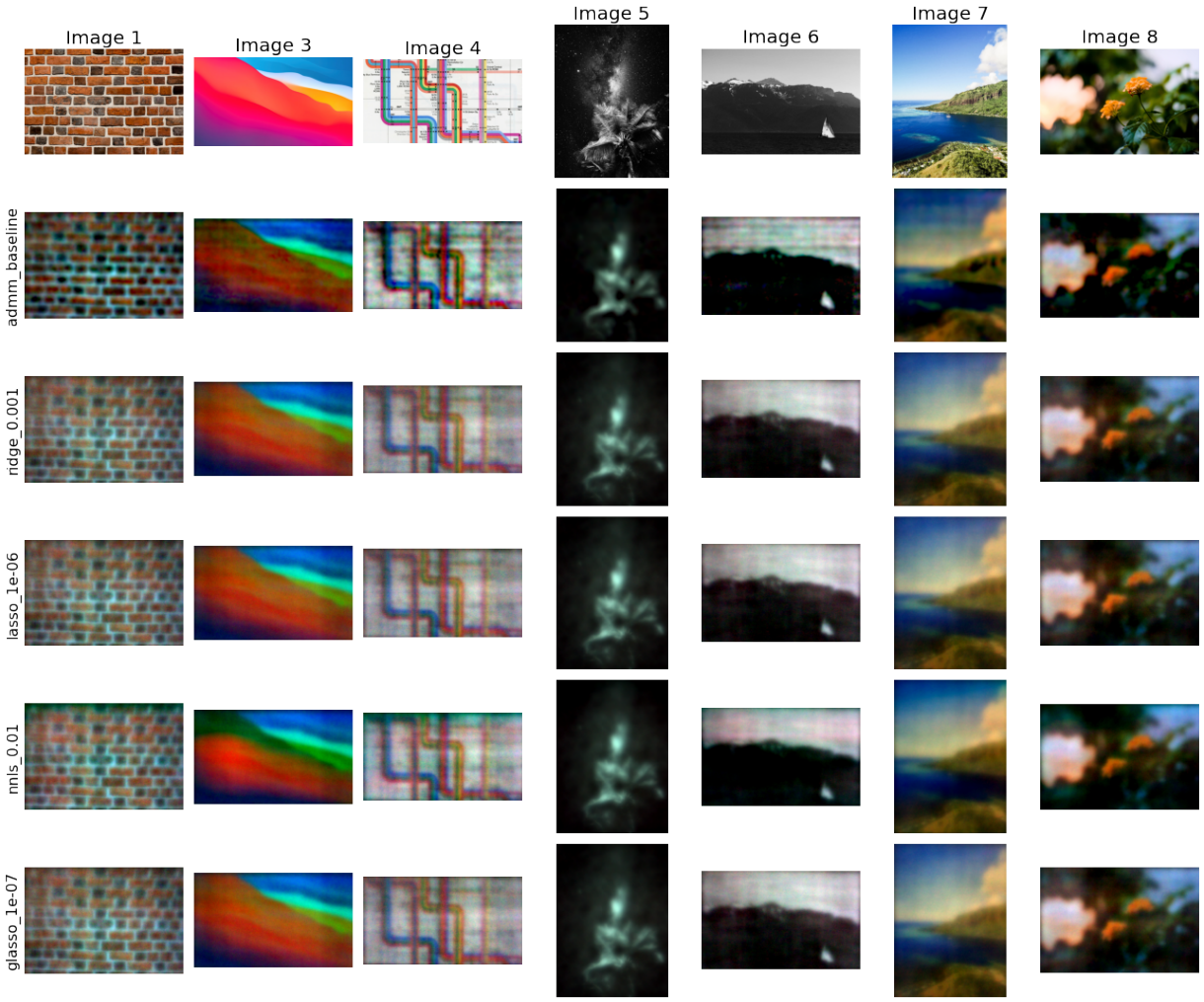


Figure 13: Final results over the 7 photos with ADMM as a baseline.

As seen in Figure 13 all reconstruction approaches seem to be able to reconstruct the DiffuserCam unlensed images to an extend that is recognizable as a bad quality/blurred version of the original image for all images except Image5. The blurry tendency is most probably due to the quality of the PSF measurement that was analyzed as not being completely optimally focused. Even though Image5 is originally a high-quality image with clearly predictable features (by the human eye) it is hard to determine from the reconstructions that it is a palm tree under a night sky of stars. This might be due to the sparsity of the original image and that the features of the palm tree such as the leaves are very small and detailed. As seen before in the average results, the tendency of ADMM to create worse reconstructions seem to be valid - for instance, it tends to overestimate the contrast on Image1 of the brick wall whereas there seems to be a noticeable overlay blur on some of the other images (like Image4 and Image6 in the white homogeneous areas of the original images). It is hard to distinguish the reconstructions created by the non-baseline algorithms based on a visual inspection from one another - the only clear difference that is visible is that the reconstruction of color and lines in Image3 varies with the different approaches. For instance, the orange "hill" in the NNLS reconstruction is perceived as an artificial "hill" compared to the one in the original image. Notice that the tiny text on the subway map in Image4 has not been reconstructed by any of the approaches.
It is important to note that since we cropped manually for each image, the consistency of the metric values might not be optimal. Since the reconstruction gives rather poor results (for the human eye at least) it is also possible that small differences in the metrics don't necessarily mean so much. We interpret our result with these precautions in mind.
To determine the validity of the reconstruction approaches in a more measurable way, the metrics and processing time are also considered on the individual image level. This is seen in Figures 14-20 where the best performing reconstruction algorithm on each metric is again highlighted in blue.

algo	mse	psnr	ssim	lpips	time
admm	0.117690	9.292622	0.12805994	0.873704	15
glasso	0.078173	11.069438	0.16418634	0.696275	1456
lasso	0.078213	11.067237	0.16405377	0.695550	1228
nnls	0.083433	10.786625	0.15875842	0.715656	1167
ridge	0.078225	11.066565	0.16403149	0.695271	1408

Figure 14: Image1 Results

algo	mse	psnr	ssim	lpips	time
admm	0.141268	8.499566	0.44460192	0.544492	15
glasso	0.096266	10.165265	0.5492192	0.512656	1458
lasso	0.096546	10.152680	0.54764	0.513778	1205
nnls	0.154612	8.107573	0.46549568	0.561599	1166
ridge	0.096341	10.161901	0.548291	0.513776	1401

Figure 15: Image3 Results

algo	mse	psnr	ssim	lpips	time
admm	0.137045	8.631361	0.33291325	0.773305	14
glasso	0.120027	9.207197	0.3687112	0.816915	1456
lasso	0.120436	9.192424	0.367802	0.817692	1197
nnls	0.122116	9.132273	0.36335394	0.820062	1171
ridge	0.120424	9.192886	0.36767578	0.817484	1403

Figure 16: Image4 Results

algo	mse	psnr	ssim	lpips	time
admm	0.018275	17.381526	0.35484466	0.737210	15
glasso	0.011584	19.361311	0.58466315	0.677491	1457
lasso	0.011595	19.357439	0.5837877	0.676732	1198
nnls	0.012438	19.052504	0.5110291	0.689870	1176
ridge	0.011586	19.360729	0.58462644	0.676566	1401

Figure 17: Image5 Results

algo	mse	psnr	ssim	lpips	time
admm	0.032462	14.886201	0.33837232	0.581236	15
glasso	0.020424	16.898487	0.6059361	0.583860	1455
lasso	0.020485	16.885714	0.60174626	0.584176	1198
nnls	0.025966	15.855897	0.41384748	0.510280	1156
ridge	0.020436	16.895981	0.60483295	0.584947	1401

Figure 18: Image6 Results

algo	mse	psnr	ssim	lpips	time
admm	0.107223	9.697138	0.4357892	0.546180	15
glasso	0.064940	11.874884	0.571815	0.502231	1460
lasso	0.065130	11.862187	0.5711222	0.502017	1199
nnls	0.082970	10.810810	0.50409764	0.504615	1165
ridge	0.065054	11.867286	0.57130593	0.502014	1401

Figure 19: Image7 Results

algo	mse	psnr	ssim	lpips	time
admm	0.107223	9.697138	0.4357892	0.546180	15
glasso	0.064940	11.874884	0.571815	0.502231	1460
lasso	0.065130	11.862187	0.5711222	0.502017	1199
nnls	0.082970	10.810810	0.50409764	0.504615	1165
ridge	0.065054	11.867286	0.57130593	0.502014	1401

Figure 20: Image8 Results

What is clear from analyzing the performance on individual images is that the average trend of GLASSO creating better reconstructions is also the case when locally investigating individual images. Furthermore, it is observed that the LPIPS metric that on average is dominated by the NNLS reconstruction approach is only the better approach for Image6. Thus, the best reconstruction approach determined from the LPIPS score is

not super easy to determine as it varies for most of the images. It could also mean LPIPS is more discriminant on the resulting quality and could be a good indicator. Since all metrics but LPIPS predicate GLASSO as the best for all of the images, the prior beliefs specified in section 2 do not have the best conditions to be verified but let's see:

- *Image1*: that *GLASSO* is the better reconstruction is in line with our prior belief since it uses the ℓ_1 norm. Wrt. the best LPIPS algorithm *RIDGE* is the best - it uses the squared ℓ_2 norm and is thus not in line with our prior belief.
- *Image3*: here *GLASSO* is the best algorithm on all metrics. This is not in line with our prior belief about prioritizing smoothness using the ℓ_2 norm.
- *Image4*: that *GLASSO* is the better approach somehow makes sense as it uses the ℓ_1 norm that is good for piece-wise constant images which the subway line is. However, the text is not reproduced on any of the approaches and unfortunately, the Huber norm was not applied, due to technical problems when running the evaluation script.
- *Image5*: is in general quite bad. It makes sense that *GLASSO* is dominant as it prioritizes sparsity but the best LPIPS approach is *RIDGE*, which uses the ℓ_2 norm.
- *Image6*: here our initial guess of ℓ_1 norm being good was true due to *GLASSO* being the majority best algorithm.
- *Image7*: Again, *GLASSO* being the best was not as hypothesized. In the LLPIPS best algorithm is *RIDGE* which was as the prior belief reasoned with the smoothness of the sea and landscape.
- *Image8*: the prior beliefs for this image were unspecified - instead we hoped to find out which penalization term would be better for distinguishing between a focused foreground and a blurry background. Unfortunately, the results do not yield a clear conclusion as they are biased by the *GLASSO* algorithm.

As previously mentioned, the ADMM is more efficient in its reconstructions than the other algorithms. Since the GLASSO was both on average and on the individual image level the better reconstruction algorithm, the following investigation will solely compare GLASSO to the ADMM baseline. An example of reconstruction for ADMM across iters as well as GLASSO reconstruction across iters is Figure 21-22

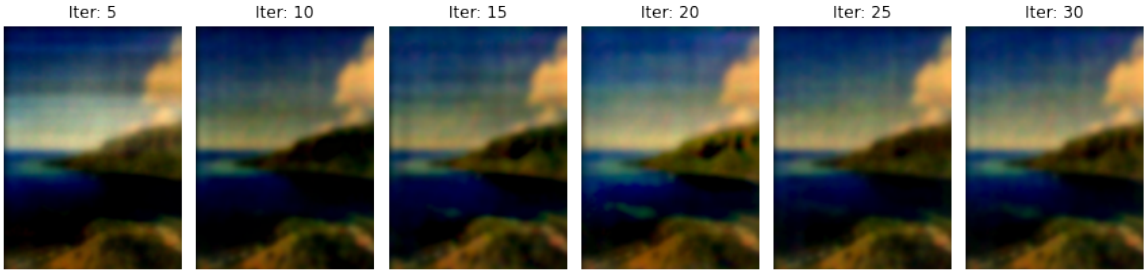


Figure 21: *ADMM* reconstruction for Image 7 by iteration.

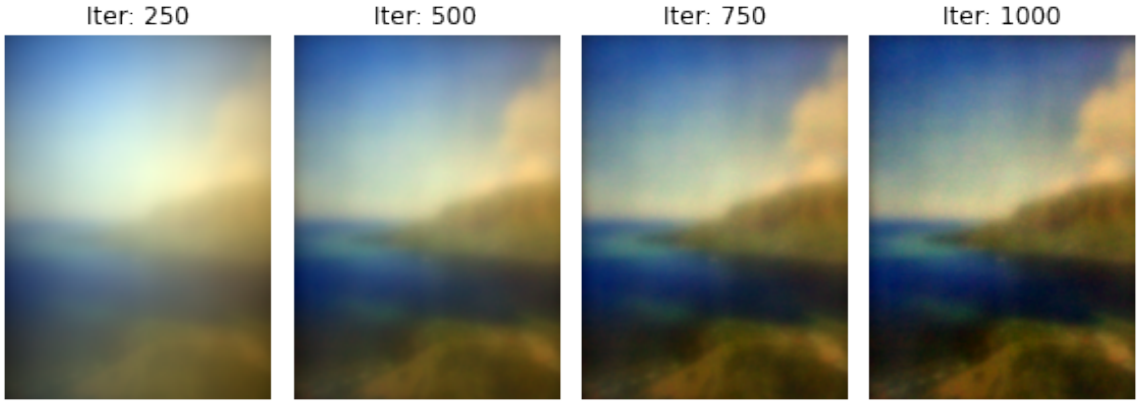
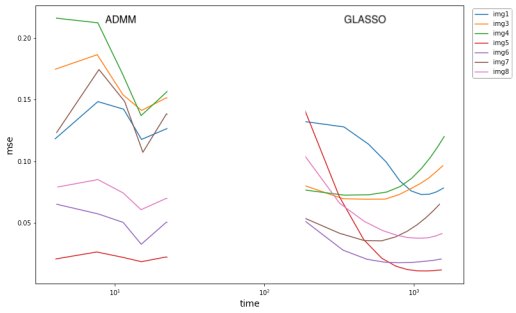
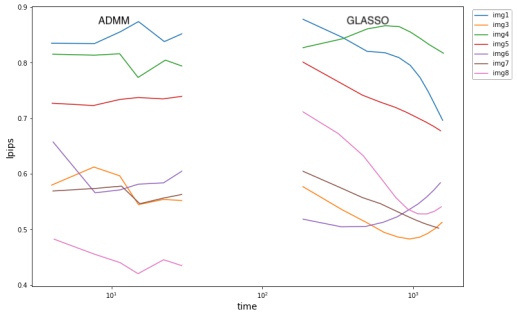


Figure 22: *GLASSO* reconstruction for Image 7 by iteration.

We also wanted to look at how the metrics we calculate evolve over time. In Fig 23-24 we plot all of the metrics over time on an exponential scale. These plots highlight that the algorithms are not always improving with iterations. For some images, you can see GLASSO decline over time. For example metric SSIM for images 1, 3, 4, 6, 7, 8. Either λ is not set correctly for this image or we indeed have reached the optimal performance that GLASSO can offer on such images. It is also apparent that some images are "tough" images for these two reconstructions. Image 1 for example is one that GLASSO has trouble with as it is consistently near the bottom (or top) of the metrics. The first time GLASSO is plotted is after 250 iterations it has sometimes already overtaken ADMM from a reconstruction metric standpoint. This means the gap in time between the two may be a bit smaller than we originally believed.

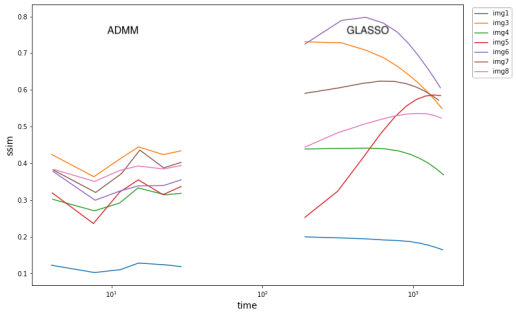


(a) MSE over time for ADMM and GLASSO.

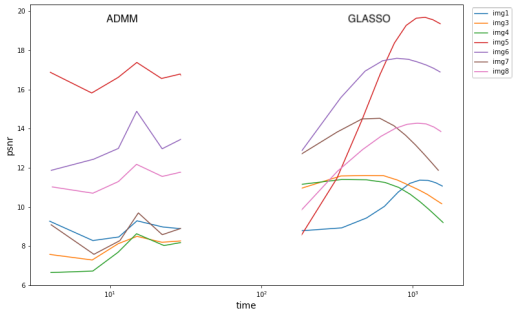


(b) LPIPS over time for ADMM and GLASSO

Figure 23: The lower the curve the better for these metrics. The x axis scale is exponential to better visualize ADMM near GLASSO. ADMM has datapoints for 5, 10, 15, 20 , 30 and 40 iterations. GLASSO has datapoints for 250, 500, 750 and 1000 iterations.



(a) SSIM over time for ADMM and GLASSO.



(b) PSNR over time for ADMM and GLASSO

Figure 24: The higher the curve the better for these metrics. The x axis scale is exponential to better visualize ADMM near GLASSO. ADMM has datapoints for 5, 10, 15, 20 , 30 and 40 iterations. GLASSO has datapoints for 250, 500, 750 and 1000 iterations.

6 Conclusion

This project was an introduction to linear inverse problems. Multiple reconstruction algorithms were implemented for different situations, ie. for different images characteristics. We analyzed how each of these minimization problems performs using different metrics.

Overall, our reconstructions quality was poor. The blur occurring in the reconstructed images is most likely due to imprecise measurements of the PSF. We suspect that using adhesive tape as a visual filter would have induced blur anyway. There might be mathematical approaches to reach different reconstruction focuses based on the shape of the tape (like multi-lens cameras which propose different focus levels). Down-sampling used in our experiments hurts the quality since it reduces fine details of the PSF/measurement data. However, it would be computationally too expensive to run the original image size.

Performance-wise, ADMM definitely has the best trade-off if we aim to get fast results. Often, we can identify the elements of the image estimate and the color quality is generally sufficient.
If we aim for the best reconstruction though, we saw that ADMM was not the most relevant. It can be aggressive and create artifacts where the other techniques produce softer reconstructions after many iterations. In our case, GLASSO has shown the best metric results overall, but it would be hardly fair saying it is the best for several reasons: we couldn't run PLS-based approaches as we intended to compare, manual cropping and color correction might induce serious inconsistencies in the measures, it was hard to fine-tune the number of iterations for so many heavy algorithms.
Nonetheless, the computation time of non-ADMM being prohibitive, we can conclude that the most useful algorithm remains ADMM.

Finally, we thought that coding implementations for GPUs could give new interesting performance/quality trade-offs concerning the down-sampling and the algorithm design since we evade many optimization/image-related computation overheads. It could be an interesting work extension for this topic.

References

[1] Camille Biscarrat and Shreyas Parthasarathy. How to Build a (Pi) DiffuserCam. URL: https://waller-lab.github.io/DiffuserCam/tutorial/building_guide.pdf.

[2] Spencer Woo. Perceptual simularity. URL: https://wiki.spencerwoo.com/perceptual-similarity.html#similarity-measures.

[3] Richard Zhang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. URL: https://arxiv.org/abs/1801.03924.

7 Appendix

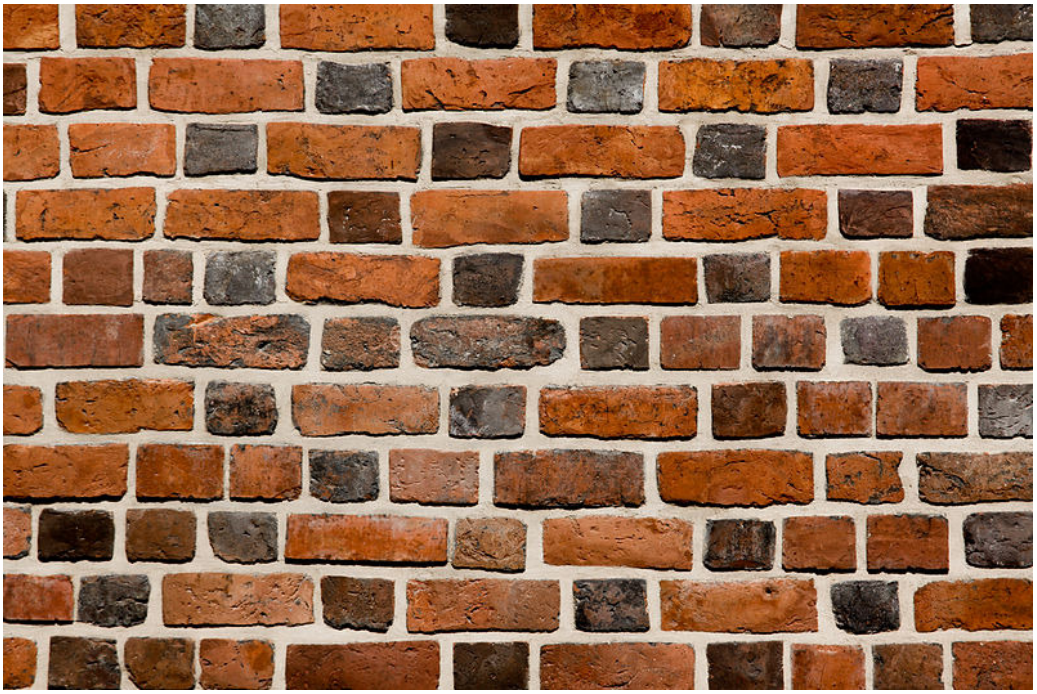


Figure 25: Image 1

Figure 26: Image 3

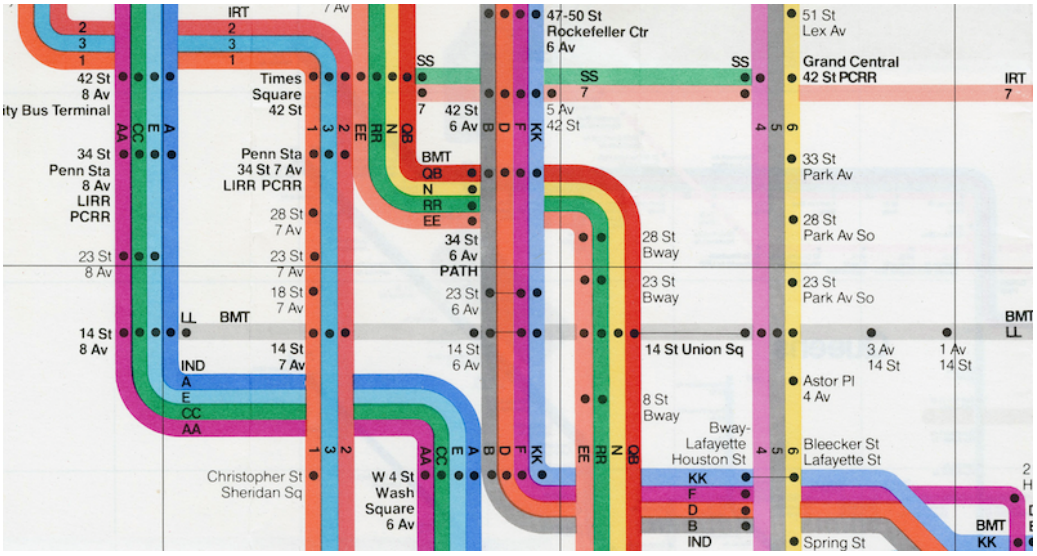


Figure 27: Image 4

Figure 28: Image 5

Figure 29: Image 6

Figure 30: Image 7

Figure 31: Image 8

algo	mean mse	mean psnr	mean ssim	mean lpips	time (s)
admm	0.087789	11.509514	0.346748	0.639499	15.1
ridge	0.061894	13.199550	0.480487	0.618882	1402.5
lasso	0.061943	13.195581	0.479841	0.618768	1214.3
nnls	0.075608	12.422622	0.412996	0.615286	1167.4
glasso	0.061795	13.204670	0.481097	0.618561	1458.2

Figure 32: Final results over the 7 photos with ADMM as a baseline. (Identical to Table 4)