DiffuserCam Project Guide (2021)

1 Introduction

We are excited to incorporate this new project into the course! We hope you find it interesting as it covers a broad scope of topics: hardware prototyping, Python development, dataset collection, camera technology, signal/image processing, and inverse problems. There is a lot of content but our aim is to guide you through the process with blog posts and exercises.

The goal of the project is to perform imaging with a lensless camera known as DiffuserCam [1]. We will use a more rudimentary version of the DiffuserCam where we use a piece of tape instead of the lens and the Raspberry Pi HQ camera sensor.[1] The content of this project is largely based off of the work from Prof. Laura Waller's group at UC Berkeley:

- Build your own DiffuserCam tutorial: `https://waller-lab.github.io/DiffuserCam/tutorial`
- DiffuserCam lensless MIR Flickr dataset [2]: `https://waller-lab.github.io/LenslessLearning/dataset.html`

So a huge kudos to them for the idea and making the tools/code/data available!

2 Organization

You are strongly encouraged to form **groups of 3–4** to help manage the workload. Here is an overview of what the next few weeks will look like so that you can plan accordingly and know what is expected of you.

We **strongly recommend** that you attend the first session on Mon Dec 6 where we will review some background concepts and make sure that the software is properly installed on your laptops. For lab sessions, we will allocate a slot to each group where you can come to the lab (INR 019) for measurements.

Week 1 (Dec 6 – Dec 10)

Objective: install software and measure the DIffuserCam's PSF.

- **Mon Dec 6, 14:15 - 16:00**: introduction to project and lensless imaging with the DiffuserCam, software installation (as described in Section 4.1).
- **Mon Dec 6, 16:15 - 17:00 (LAB)**: PSF measurements in the lab - INR 019 (by group).
- **Fri Dec 10, 15:15 - 17:00 (LAB)**: PSF measurements in the lab - INR 019 (by group).
- **Tasks**
  1. Complete `diffcam.autocorr.autocorr2d` in order to compute the 2-D autocorrelation of an image. Upon successful completion, running the following command should yield autocorrelation plots as shown in Figure 1.

     ```
     python scripts/analyze_image.py --fp data/psf/diffcam_rgb.png --diffcam
     ```

     Listing 1: Analyzing DiffuserCam PSF.

  2. (In the lab - INR 019) measure a DiffuserCam PSF and raw data, as described in Section 4.2.

[1] `https://www.raspberrypi.com/products/raspberry-pi-high-quality-camera/`

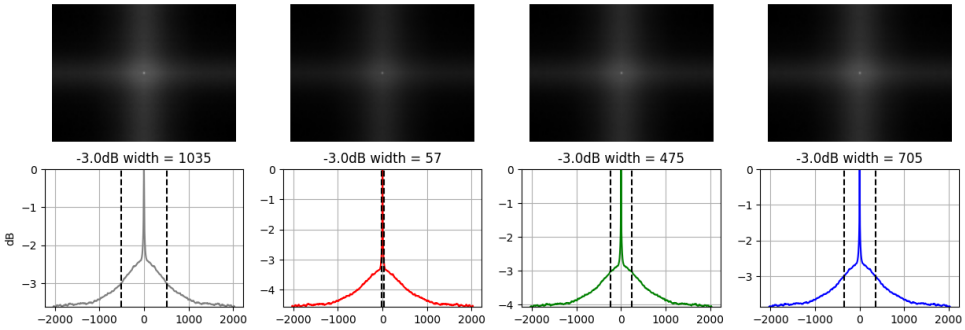


Figure 1: Autocorrelations of grayscale PSF and red, green, and blue channels.

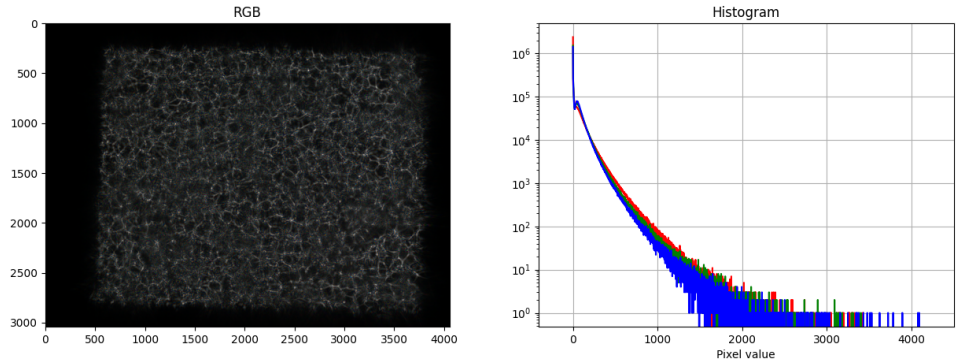


Figure 2: Color corrected DiffuserCam point spread function.

3. Start collecting a set of images (around 5) that you think would work well for different priors, e.g. those we suggest implementing next week.
4. If you have time, you can already take a look and try next week's exercises using the PSF and raw data already in the repository - `diffcam_rgb.png` and `thumbs_up_rgb.png`.

- **Required reading:**
  - Measuring an optical PSF (Medium article).
  - Measuring a DiffuserCam PSF and raw data (Medium article).
  - Lensless imaging with the DiffuserCam (Medium article).

Week 2 (Dec 13 – Dec 17)

Objective: implement various computational imaging algorithms and collect all raw data.

- **Mon Dec 13, 14:00 - 17:00 (LAB)**: raw data measurement.
- **Fri Dec 17, 15:00 - 17:00 (LAB)**: raw data measurement.
- **Tasks: when testing your implementations below, you can use `diffcam_rgb.png` and `thumbs_up_rgb.png` as PSF and raw data and/or the DiffuserCam MIR Flickr dataset while you wait for your own raw data measurements.**

1. (In the lab - INR 019) collect raw data projected on a monitor, using the procedure described in Section 4.3. The goal is to collect data you think will work well with the various priors / penalties you will implement. Plan to collect around $5-10$ images for this mini-dataset. For

Figure 3: Source: `https://www.flickr.com/photos/thom_astro/51440286414/`.

example, the image in Figure 3 would work well with TV priors which promote piecewise-constant functions.

2. Start with a simple grayscale image reconstruction using a *penalized least-square problem with Tikhonov regularization* (i.e. ridge regression):

$$\min_{\boldsymbol{x}\in\mathbb{R}^N} \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{H}\boldsymbol{x}\|_2^2 \quad + \quad \lambda\|\boldsymbol{x}\|_2^2, \tag{1}$$

   where $\boldsymbol{y} \in \mathbb{R}^N$ is the (flattened) grayscale raw measurements obtained with the DiffuserCam, $\boldsymbol{H} \in \mathbb{R}^{N\times N}$ is a discrete convolution operator modelling the blur introduced by the DiffuserCam's PSF, $\boldsymbol{x} \in \mathbb{R}^N$ is the (flattened) grayscale image to recover, and $\lambda > 0$ is some penalty parameter. Play with various penalty parameters. You may find the following Pycsou classes useful in your implementation:

   - `pycsou.opt.proxalgs.APGD`,
   - `pycsou.func.penalty.SquaredL2Norm`,
   - `pycsou.linop.conv.Convolve2D`.

   You can also use this template for loading the PSF and raw data. Be sure to use the `--gray` flag for grayscale data. See also Pycsou's documentation for an example describing how to use APGD.

3. Perform the grayscale image reconstruction with a *LASSO* problem, i.e. replacing the squared ℓ_2 norm in the penalty term of (1) by an ℓ_1 norm. Again, investigate different penalty parameters. You may find the following Pycsou class useful in your implementation:
   `pycsou.func.penalty.L1Norm`.

4. Perform the grayscale image reconstruction with a *non-negative least-square* problem, i.e. replacing the squared ℓ_2 norm in the penalty term of (1) by a non-negativity prior. You may find the following Pycsou class useful in your implementation:
   `pycsou.func.penalty.NonNegativeOrthant`.

5. Perform the grayscale image reconstruction with a *generalized LASSO* problem with the Type 2 *Discrete Cosine Transform (DCT)* as sparsifying transform, i.e. replacing the square ℓ_2 norm in the penalty term of (1) by an ℓ_1 norm composed with the 2D Type 2 DCT transform $\|\mathrm{DCT}_2(\boldsymbol{x})\|_1$. Pycsou does not support yet the DCT transform, so you will have to implement it yourself by subclassing `LinearOperator` (the base class for linear operators in Pycsou) and implement the `__call__` and `adjoint` methods (see Pycsou's documentation for an example). You may find the following Scipy classes useful in your implementation: `scipy.fft.dctn` and `scipy.fft.idctn`. Note that with the right normalization mode, the DCT of Type 2 is an *orthonormal operator* (see the documentation notes of `scipy.fft.dct` for more information on the topic). Use this fact to propose a change of variable that transforms the generalized LASSO problem with penalty $\|\mathrm{DCT}_2(\boldsymbol{x})\|_1$ into a

regular LASSO problem. Solve the LASSO problem resulting from this change of variable via `pycsou.opt.proxalgs.APGD`.
Note: Most natural images have a sparse DCT transform*, making it a good transform to use with a sparsity prior. This fact is notably leveraged in JPEG compression to represent complex images with a few DCT coefficients only.*

6. Perform the grayscale image reconstruction with a *penalized least-square* problem with composite penalty term involving a *TV* and non-negativity prior. This means replacing the squared ℓ_2 norm in the penalty term of (1) by the sum of a non-negativity prior and a weighted ℓ_1 norm composed with the finite-difference *2D gradient operator*: $\iota_{[0,+\infty)}(\boldsymbol{x}) + \lambda\|\boldsymbol{D}\boldsymbol{x}\|_1$. You may find the following Pycsou classes useful in your implementation: `pycsou.linop.diff.Gradient` and `pycsou.opt.proxalgs.PDS` (see Pycsou's documentation for an example). Note that PDS is much slower than APGD, and may require at least 5000 iterations to converge.
7. To accelerate the previous reconstruction, replace the ℓ_1 norm in the TV penalty by a *smoothed* ℓ_1 norm known as the *Huber norm*. This means replacing the penalty term $\iota_{[0,+\infty)}(\boldsymbol{x}) + \lambda\|\boldsymbol{D}\boldsymbol{x}\|_1$ by $\iota_{[0,+\infty)}(\boldsymbol{x}) + \lambda\|\boldsymbol{D}\boldsymbol{x}\|_{H,\delta}$ where

$$\|\boldsymbol{z}\|_{H,\delta} = \sum_{m=1}^{M} h_\delta(\boldsymbol{z}_m), \qquad \forall \boldsymbol{z} \in \mathbb{R}^M,$$

and $h_\delta : \mathbb{R} \to \mathbb{R}_+$ is the Huber function:

$$h_\delta(z) = \begin{cases} \dfrac{1}{2}z^2 & \text{for } |z| \leq \delta \\ \delta(|z| - \delta/2) & \text{otherwise.} \end{cases}$$

Show that the Huber norm is differentiable and with 1-Lipschitz continuous gradient. Compute this gradient. Implement the Huber norm by subclassing `DifferentiableFunctional` (the base class for differentiable functionals in Pycsou) and providing the `__call__` and `jacobianT` methods. Do not forget to provide the Lipschitz constant of the derivative via the `diff_lipschitz_cst` argument of the `DifferentiableFunctional.__init__` method. See Pycsou's documentation for an example.
8. Extend your implementations to RGB such that, without the `--gray` flag, your scripts perform RGB reconstruction. You can take inspiration from `ADMM` which updates all channels simultaneously using array operations.
9. *(Bonus)* `pycsou.linop.conv.Convolve2D` *relies on* `pylops.signalprocessing.ConvolveND` *that recomputes the FFT of the filter/PSF each time the convolution is called. We can save some computation by pre-computing the FFT of the PSF (this will save quite a bit of time when performing reconstructions for a dataset in the next week!). Implement a convolution operator that does this and benchmark the computational time reduction. We've noticed a 25% reduction, which can significantly speed up the algorithm development process! Hint: you can take inspiration from the* forward operator *of ADMM.*

- **Required Reading**
  - Lensless imaging with the DiffuserCam (Medium article).
  - Solving inverse problems with Pycsou (Documentation).
  - Examples of how to use Pycsou (Documentation).

Week 3 (Dec 20 – Dec 24)

Objective: continue implementing various techniques, evaluate the approaches on datasets, and writing report.

- **Mon Dec 20, 14:00 - 17:00**: algorithm / code debugging.
- **Tasks**

  - Implement metrics to evaluate the accuracy/performance of the various approaches on collected mini-datasets and the MIR Flickr dataset (like this).

- **Reading**
  - Evaluating your DiffuserCam reconstructions (Medium article).

Week 4 (Dec 27 – Dec 30)

Objective: wrapping up, packaging data + code.

- **Submission deadline: December 30**

3 Grading

You will be evaluated on the following criteria:

- Quality of report: explicitly state what optimization problem(s) you are solving and justify the various choices of algorithms, priors and penalties. Proceed scientifically: be rigorous about your assumptions/calculations, compare and comment the results obtained, be critical.
- Quality of code (should be easy to run and documented with docstrings). Consider using a command line parser like `click` or `argparse`. Your implementations of the reconstruction algorithms should make use of this template.
- Technical correctness of optimization problem specification and code.

We are only working with a piece of tape, so it is important to keep expectations in check when evaluating your reconstructions! We are primarily interested in (1) how you formulate the inverse problem, and (2) how you go about solving it with the data you collect and code you implement.

Deliverables include:

1. Report (5-15 pages) which should include:
   - Analysis of PSF (plot and autocorrelations).
   - Reconstruction results.
   - Clear benchmark of different approaches, as done in Table 2 of [2] (timing can be done on CPU and number of training images not needed). Include ADMM and two other approaches of your choice in your benchmark.
2. Code + mini-dataset.

4 Code examples

4.1 Installation and setup

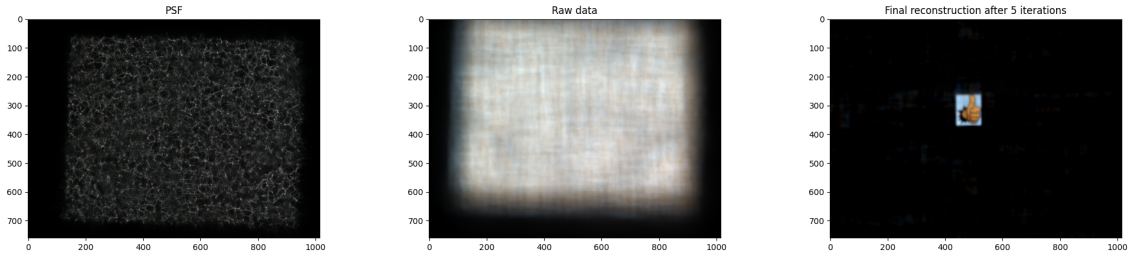
Install the `diffcam` library – `https://github.com/LCAV/DiffuserCam` – e.g. by entering the following commands from a terminal.

```
# download from GitHub
git clone git@github.com:LCAV/DiffuserCam.git

# install in virtual environment
cd DiffuserCam
python3.9 -m venv diffcam_env
source diffcam_env/bin/activate
pip install -e .
```

Listing 2: Installation from terminal.

Let's run a couple scripts to make sure things installed properly. First a script to reconstruct data that has already been measured with our prototype camera, which should produce plots like those in Figure 4.



(a) PSF (gamma corrected). (b) Raw data. (c) ADMM reconstruction.

Figure 4: Output of Listing 3.

```
python scripts/admm.py --psf_fp data/psf/diffcam_rgb.png \
--data_fp data/raw_data/thumbs_up_rgb.png \
--n_iter 5
```

Listing 3: Image reconstruction via ADMM.

The algorithm applied is called *Alternating Direction Method of Multipliers* (ADMM) [3]. This method was not presented during the lectures, but we will use it as a comparison baseline for the various approaches you will explore.

Now we will run a script that applies ADMM reconstruction to a dataset and computes some metrics. You can download the dataset here and run the following command to perform an evaluation on 10 (of 200) files[2] and save the reconstruction results.

```
python scripts/evaluate_mirflickr_admm.py --data DiffuserCam_Mirflickr_200_3011302021_11h43_seed11 \
--n_files 10 --n_iter 10 --save
```

Listing 4: Evaluate part of DiffuserCam MIR Flickr dataset with ADMM reconstruction.

You will be expected to produce a similar script to evaluate the performance of your algorithm implementations.

DiffuserCam available?

If you are connected to the same network as a Raspberry Pi with the DiffuserCam software installed, you can try the scripts to remotely take a picture and display one for measurements.

First we need to set up passwordless SSH'ing (Medium article). In short:

1. If you don't have a private/public SSH key pair yet, you can see how to create one here (until "Adding your SSH key to the ssh-agent" and **don't set a password!**).

2. Run the following command from the Terminal and enter the Raspberry Pi's password.

   ```
   cat <PUBLIC_KEY_PATH> | ssh pi@<HOSTNAME> "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"
   ```

   Listing 5: Passwordless SSH setup.

   where `<PUBLIC_KEY_PATH>` is the path to your **public** SSH key (e.g. `~/.ssh/id_ed25519.pub`) and `<HOSTNAME>` is the hostname (e.g. `raspberrypi.local` on a default installation) or the IP address of the Raspberry Pi (can be determined with `ifconfig`).

We can now run the following command to test the image capture software.

```
python scripts/remote_capture.py --exp 0.1 --iso 100 --hostname <HOSTNAME>
```

Listing 6: Image capture script.

If you have a display connected to the Raspberry Pi, you can also test the software to display images remotely.

[2]The original dataset [2] contains 25′000 files so if you feel like you need more, do not hesitate to ask!

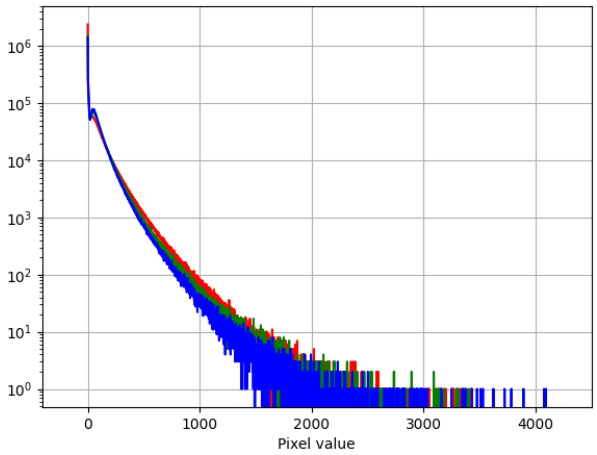


Figure 5: Color-corrected PSF.

```
python scripts/remote_display.py --fp <FILE> --hostname <HOSTNAME> \
--pad 80 --vshift 10 --brightness 90
```

Listing 7: Image display script.

where `<FILE>` is the path to a local file you would like to display on the Raspberry Pi's screen.

4.2 Measuring a DiffuserCam PSF and raw data

For a more in-depth description, you can refer to this Medium article.

Set up a point source around 40 cm and measure the PSF with this command:

```
python scripts/remote_capture.py --exp 0.1 --iso 100 --bayer --fn psf --hostname <HOSTNAME> \
--nbits_out 12
```

Listing 8: Measuring Bayer data.

where `<HOSTNAME>` is the Raspberry Pi's hostname or IP address and you should adjust the exposure and ISO in order to avoid clipping and to use as much of the dynamic range as possible. We use the flag `--bayer` in order to collect raw Bayer data and adjust the color balancing by hand.[3] ***For Raspberry Pi Bullseye, the Bayer data will be saved as a DNG file, while earlier versions of Raspberry Pi OS will save the Bayer data as a PNG file.***

You can determine the best red and blue gains by adjusting their values until the red, green, and blue histograms line up (as shown in Figure 5), and use the `--save` flag to save a color-corrected version.

```
python scripts/analyze_image.py --bayer --fp psf.dng --bg <BLUE_GAIN> --rg <RED_GAIN> --save psf_rgb.png
```

Listing 9: Determine red and blue gains.

Note that the Bayer file will be saved as a PNG file for OS' before Bullseye. You can follow the same procedure for collecting raw data: capture data (use a backlit source like a phone) and then color-correct manually. You can try reconstructing with ADMM, as below, to make sure the red and blue gains for the raw data make sense.

```
python scripts/admm.py --psf_fp psf_rgb.png --data_fp data_rgb.png --n_iter 5
```

Listing 10: Check white balancing with a reconstruction.

4.3 Displaying images on the screen

Make sure the screen is at the same distance at which you measured your PSF. More in-depth instructions can be found here.

From the Raspberry Pi, launch the image viewer.

[3] You can read about Bayer data here.

```
feh DiffuserCam_display --scale-down --auto-zoom -R 0.1 -x -F -Y
```

Listing 11: Launch image viewer.

Then from your laptop, you can use the following command to display a local file on the display connected to the Raspberry Pi.

```
python scripts/remote_display.py --fp <FILE> --hostname <HOSTNAME> --pad 80 --vshift 10 --brightness
```

Listing 12: Display image remotely on Raspberry Pi.

where `<HOSTNAME>` is the hostname / IP address of the Raspberry Pi, `<FILE>` is the local file path to the image you would like to display, and the other arguments can be used to adjust the positioning and brightness of the image.

You can take pictures of the screen with the same command as before. Collect raw Bayer data so that you can color-correct afterwards.

```
python scripts/remote_capture.py --exp 0.1 --iso 100 --bayer --fn <FILENAME> --hostname <HOSTNAME> \
--nbits_out 12
```

Listing 13: Measuring Bayer data.

References

[1] Nick Antipa, Grace Kuo, Reinhard Heckel, Ben Mildenhall, Emrah Bostan, Ren Ng, and Laura Waller. Diffusercam: lensless single-exposure 3d imaging. *Optica*, 5(1):1–9, 2018.

[2] Kristina Monakhova, Joshua Yurtsever, Grace Kuo, Nick Antipa, Kyrollos Yanny, and Laura Waller. Learned reconstructions for practical mask-based lensless imaging. *Optics express*, 27(20):28075–28090, 2019.

[3] Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.