Developing a Mirrorless Camera Attachment for Lensless Imaging

Relevant Code Documentation

Jordan Ando, Yolanda Hu, Andrew Olsen
ENGR 030

This document serves as a high-level overview of our code. Each relevant document has been summarized below, with notes on what we adjusted or used.

ADMM Code

This script, adapted from the Waller Lab DiffuserCam GitHub, reconstructs an image using the Alternating Direction Method of Multipliers. This sophisticated optimization technique allows for a decent 2D image reconstruction after very few iterations.

Inputs:

- Images (MUST be in .RAF data format - .JPG or .JPEG files process in nonlinear colorspace)
    - PSF
    - Raw Data
- Parameters (found in configuration file)
    - `f` (downsampling factor)
    - `mu1, mu2, mu3, tau` (mathematical parameters for optimization, adjusting these changes the output image)
    - `iters` (number of iterations)

Output:

- Reconstructed Image (black and white image in matplotlib plot)

Relevant Files (from Waller Lab GitHub):

- ADMM.py (main file)
- admm_config.yml (configuration/parameters)
- ADMM.ipynb (Jupyter notebook version of ADMM.py; has helpful text and links)

GD Code

This script, adapted from the Waller Lab DiffuserCam GitHub, reconstructs an image using the Gradient Descent method. This standard optimization technique of 2D image reconstruction requires more iterations and computing time than ADMM, but we tried this method out for comparison's sake.

Inputs:

- Images (MUST be in .RAF data format - .JPG or .JPEG files process in nonlinear colorspace)
    - PSF
    - Raw Data
- Parameters (found in configuration file)
    - `f` (downsampling factor)
    - `iters` (number of iterations)

Output:
- Reconstructed Image (black and white image in matplotlib plot)

Relevant Files (from Waller Lab GitHub):
- GD.py (main file)
- gd_config.yml (configuration/parameters)
- GD.ipynb (Jupyter notebook version of GD.py; has helpful text and links)

3D Reconstruction Code - MATLAB

The most interesting part of this project was attempting to reconstruct a 3D image from a 2D input captured by our camera attachment diffuser. Adapted from the Waller Lab, this program uses MATLAB to generate a 3D reconstruction.

Inputs:
- Images (MUST be in .RAF data format - .JPG or .JPEG files process in nonlinear colorspace)
    - PSF Array (requires PSFs at different distances to analyze in 3D)
    - Raw Data
- Parameters (found in configuration file, 'admm_config.yml')
    - `psf_bias, image_bias` (accounts for bias of camera or PSF bias)
    - `solverSettings.autotune` (when 1, auto-determines the mu1/mu2/mu3 parameters)
    - `solverSettings.tau, solverSettings.tau_n` (sparsity parameters)
    - `solverSettings.resid_tol` (other parameter to adjust reconstruction)

Outputs:
- XY Reconstructed Image (black and white image in MATLAB figure)
- XZ, YZ plane Reconstructions

Relevant Files (from Waller Lab GitHub):
- DiffuserCam_settings.m (main file containing parameters and main code)
- ADMM3D_solver.m (ADMM algorithm in MATLAB)
- processImage.m (our own file - creates a .mat array of PSFs for input)

3D Reconstruction Code - Python

In an effort to fully integrate our code into one coding language, we converted the MATLAB files to Python.

Inputs:

- Images (MUST be in .RAF data format - .JPG or .JPEG files process in nonlinear colorspace)
    - PSF Array (requires PSFs at different distances to analyze in 3D)
    - Raw Data
- Parameters (found in configuration file, 'admm_config.yml')
    - `psf_bias, image_bias` (accounts for bias of camera or PSF bias)
    - `autotune` (when 1, auto-determines the mu1/mu2/mu3 parameters)
    - `tau, tau_n` (sparsity parameters)
    - `resid_tol` (other parameter to adjust reconstruction)

Outputs:

- XY Reconstructed Image (black and white image in matplotlib figure)
- XZ, YZ plane Reconstructions

Relevant Files (from Waller Lab GitHub):

- DiffuserCam_settings.py (file containing parameters and configs)
- ADMM3D_solver.py (ADMM algorithm in MATLAB)
- DiffuserCam_main.py (main file for running program)