Extended tutorial on running VIPR to calibrate a phase mask

Part 1: Preparation of data and information.

VIPR is a phase retrieval (sensor-less) method which estimates the phase mask from a set of calibration/design images with designed/known positions.

A) The code needs as input the optical parameter of the imaging system:
NA, magnification, focal lens of Fourier transform lens (or tube lens if the experiment is done without an extended 4-f system), emission wavelength, refractive indices of sample and objective immersion media, camera pixel size, Fourier plane pixel size (in real space), estimation of the optical blur in the system (in pixels) and size of the calibration emitter.

B) Calibration images: the software accepts a tiff-stack of calibration images as input, for example ‘’TP_im.tiff’’ which this tutorial works with.

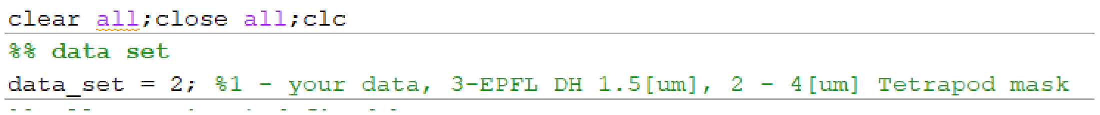
C) Each image needs an associated 4 coordinates to as input:
a) (x,y) positions of the emitters compared to the cantering of the FOV:
The default is (0,0) from the center of the cropped (by the user or by the software) FOV.
b) z is the emitter height from the coverslip. The default is the radius of the emitter. Change this or input as vector if you want to design/retrieve phase information from volumetric data (not beads that are bound to the coverslip).
c) NFP is the shift of the objective from being focused on the coverslip. This is crucial data for z-stack calibration and is usually available from the microscope stage read. To avoid adding a biased NFP estimation to the PR procedure, we suggest focusing (without a phase mask) on the calibration bead, then applying the phase mask and performing the z-stack measurement.

Part 2: Using VIPR

To use the code, there are 2 main steps: User defined parameters and data input.

A) **Open the function ‘’Main.mat’’**

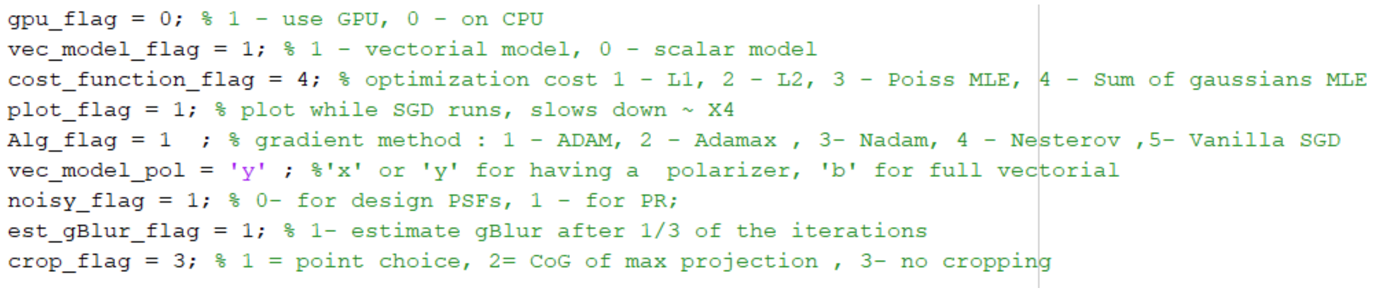
B) **Choose data set (data_set=1 for your data, data_set = 3 for the EPFL challenge Double Helix data and data_set = 2 for demo TetraPod stack).**



```
clear all;close all;clc
%% data set
data_set = 2; %1 - your data, 3-EPFL DH 1.5[um], 2 - 4[um] Tetrapod mask
```

C) **Open the function ‘’VIPR_user_input.mat’’.**

a) **The first part of this function defines general flags for the VIPR code, most of them should remain constant.**



```
gpu_flag = 0; % 1 - use GPU, 0 - on CPU
vec_model_flag = 1; % 1 - vectorial model, 0 - scalar model
cost_function_flag = 4; % optimization cost 1 - L1, 2 - L2, 3 - Poiss MLE, 4 - Sum of gaussians MLE
plot_flag = 1; % plot while SGD runs, slows down ~ X4
Alg_flag = 1  ; % gradient method : 1 - ADAM, 2 - Adamax , 3- Nadam, 4 - Nesterov ,5- Vanilla SGD
vec_model_pol = 'y' ; %'x' or 'y' for having a  polarizer, 'b' for full vectorial
noisy_flag = 1; % 0- for design PSFs, 1 - for PR;
est_gBlur_flag = 1; % 1- estimate gBlur after 1/3 of the iterations
crop_flag = 3; % 1 = point choice, 2= CoG of max projection , 3- no cropping
```

gpu_flag = 1 if you want to run on CUDA with MATLAB, 0 runs on the CPU.

vec_model_flag : choose optical model, vectorial(recommended) or scalar.

cost_function_flag: we suggest using the Gaussian log-likelihood as defined in our VIPR paper, however, the user can choose to change the cost function to Poisson’s negative log-likelihood or to the L1/L2 norms.

plot_flag: 0 means that no plots are going to be drawn or calculated, good for acceleration.

Alg_flag: choose how to use the gradient to update the phase mask, we suggest leaving ADAM.

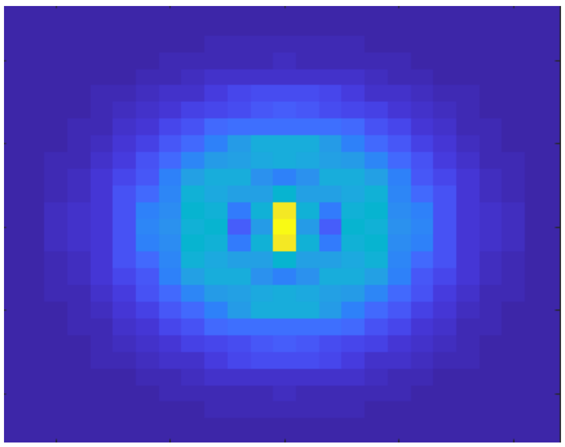
vec_model_pol: the polarization state of the system. Can be ‘x’ , ‘y’ or ‘b’.

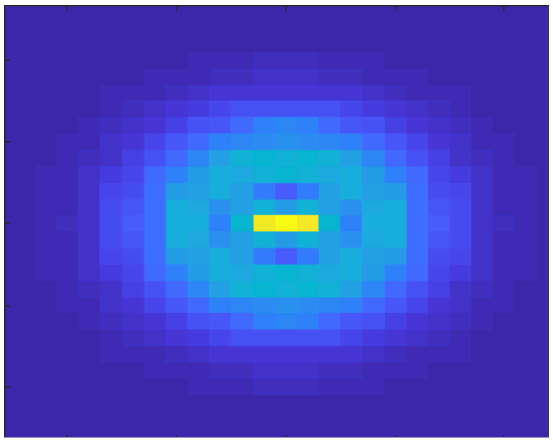
for polarization independent phase mask (e.g. dielectric) use ‘b’.

for polarization independent phase mask (e.g. LC-SLM) use ‘x’ or ‘y’, depending on how the LC-SLM is set-up. In some setups it is hard to know the direction if the system is not as simple as just a 4f system with a polarizer.

If you don’t know the direction of polarization: either simulate with the vectorial model the unmodulated PSF and compare it to your data.

‘y polarization example at z=1um’ ‘x polarization example at z=1um’



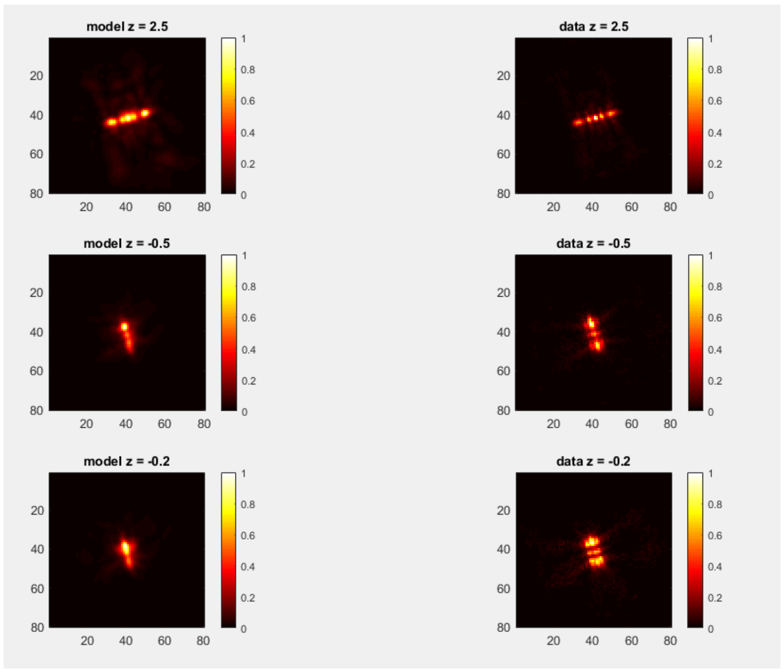


Or run VIPR and see which one works best for you, in all the test we made, the difference was substantial.

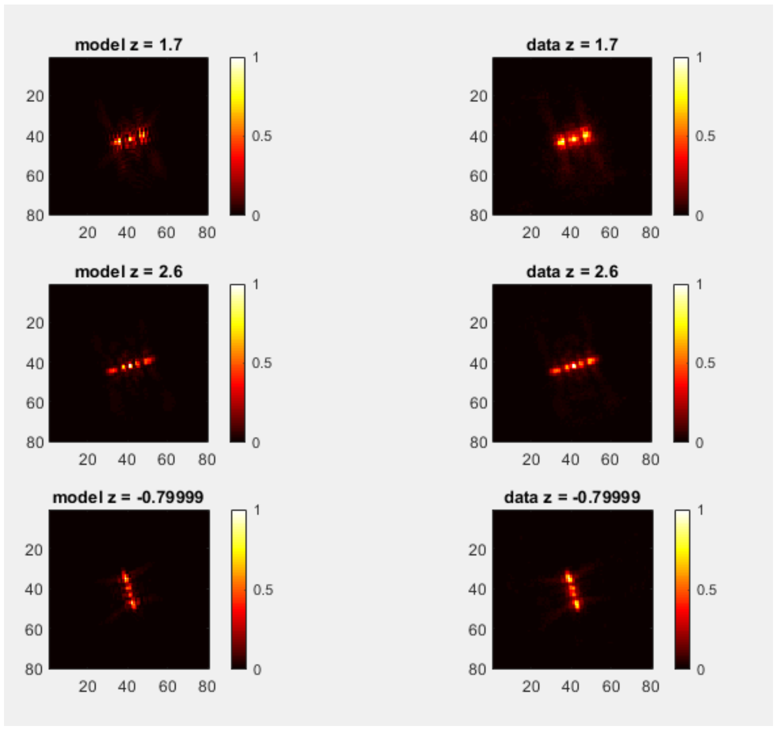
<u>noisy flag:</u> if you use VIPR to design PSFs, there is no need to do noise thresholding and estimation.

<u>est_gBlur flag:</u> the optical blur is an input parameter, default 0.75 pixels for our data. This can vary wildly with changing pixels sizes and optical components. If the flag is 1, the algorithm will estimate the blur after 1/3 of the iterations, although a good initial guess still helps. To get the initial guess, use a simulation (with some phase mask) and see how much you need to blur it , using the function imgaussfilt to match the measured data.

example of how VIPR will perform if the parameter is over-estimated.



example of how VIPR will perform if the parameter is under-estimated.



Crop_flag: choose how to crop the data. Best if you use a centering algorithm which matches best the used phase mask and crop the tiff stack to best fit the (x,y) positions of the input.

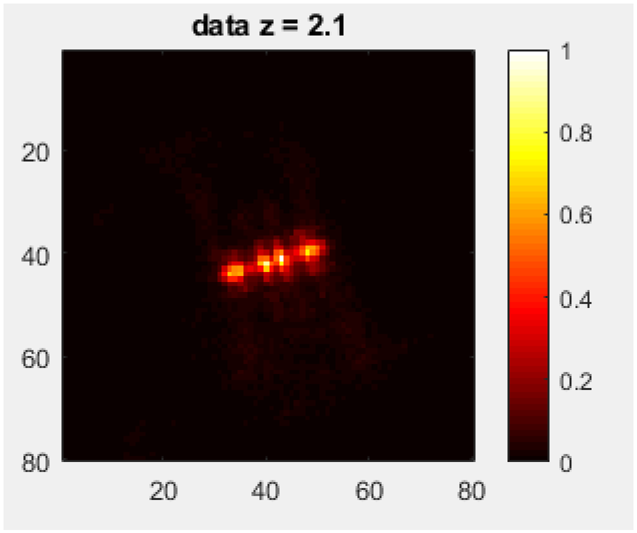
b) Input the optical parameters of the experiment.

```
IS.M=100; % objective magnification
IS.NA=1.45; % objective NA
IS.lambda=[605]*10^-3; % wavelength [um]
IS.SLM_psize = 20; % pixel size of SLM [um]
IS.Cam_psize = 16; % pixel size of camera [um]
IS.gBlur=0.75; %initial guess of the blurring
IS.n_glass=1.518; % RI of immersion oil
IS.n_med=1.33; % RI of sample medium
IS.f_4f = 15e4; % focal length of 4-f  system (if 2D imaging- use tube f)
IS.FOV_size = 70; % size of ROI used
IS.SAF_flag = 1; % include SAF or not (1=include - recommended)
IS.Iris_pNA = 1; % optional iris to limit BFP, in range [0,1] where 1 is the NA
% emitter size
IS.z_emit = 0.023; % emitter radius [um]

% polarization of dipole (0,0,0) is for freely rotating
IS.p_vec = [0,0,0]; % incoherent
```

Key notes:

- FoV size should be chosen to include all the PSF shape and an area around it for noise estimation. In the Tetrapod example: 70-80 is a reasonable pick.



- Iris_pNA is usually 1, unless you use an iris to block some of the high frequencies in your system.

c) Advanced options for VIPR.

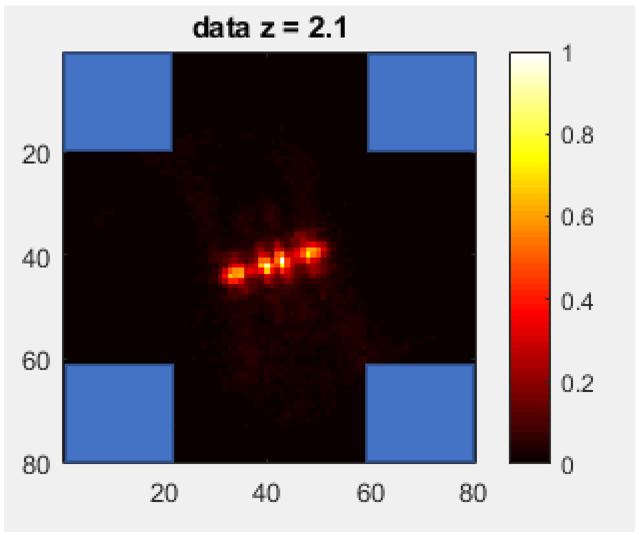
This section has many parameters which control the pre-processing and the optimization.

The first group is image thresholding.

```
% pre-proc parameters
IS.I_thr_flag = 2; % 1- thr above IS.thr*max(I) per image, else - thr above
IS.I_thr = 1; %  threshold parameter
IS.corner_size = 10; % size of corners to  estimate noise [pixels]
```

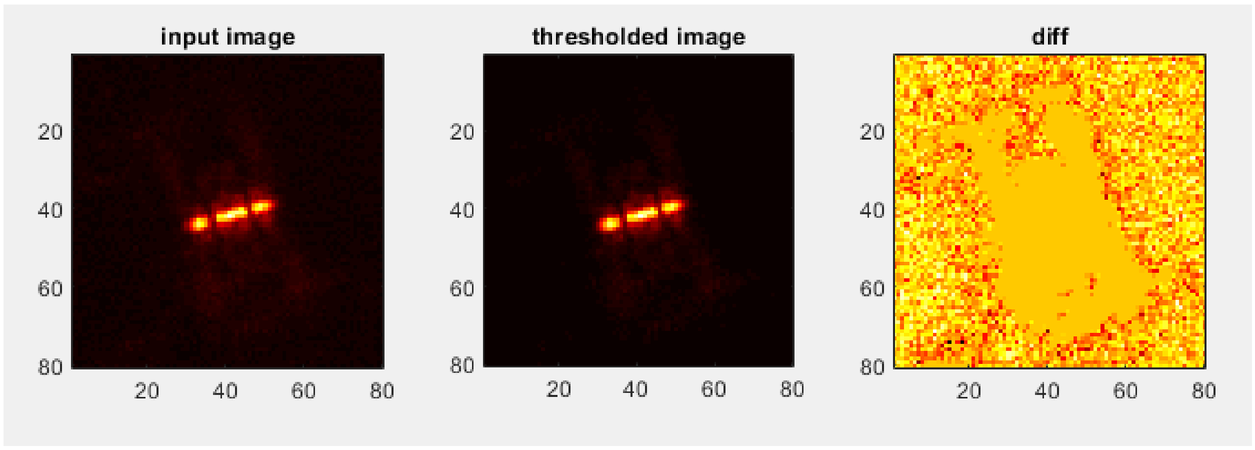
There are 2 options, as explained with I_thr_flag and it's associated parameter I_thr.

The corner size controls how large the corners are cropped to estimate the background noise from (blue squares). Try not to have any PSF data in them.



The thresholding process is plotted before the process begins so you can view if your choice was good. Try increasing the threshold in you have a lot of background SNR and you notice that the PR procedure starts to estimate the noise.

Example of a desired scenario:



<u>The second group of parameters control how many iterations and step size:</u>

```
IS.SGDiter = 250; %  how  many iterations to SGD
IS.step_size = 5e-1; % step  size (try 3e-1 for ADAM and 3e-8 for SGD)
IS.point_num = 3; % size of mini-batch per SGD iteration
```

Change this depending on how well your data converges.

Note: an initial stabilization of the cost is usually not enough for a perfect convergence, the final results should be evaluated.

<u>The third group of parameters are more advanced options, depending on how well the PSF converges:</u>

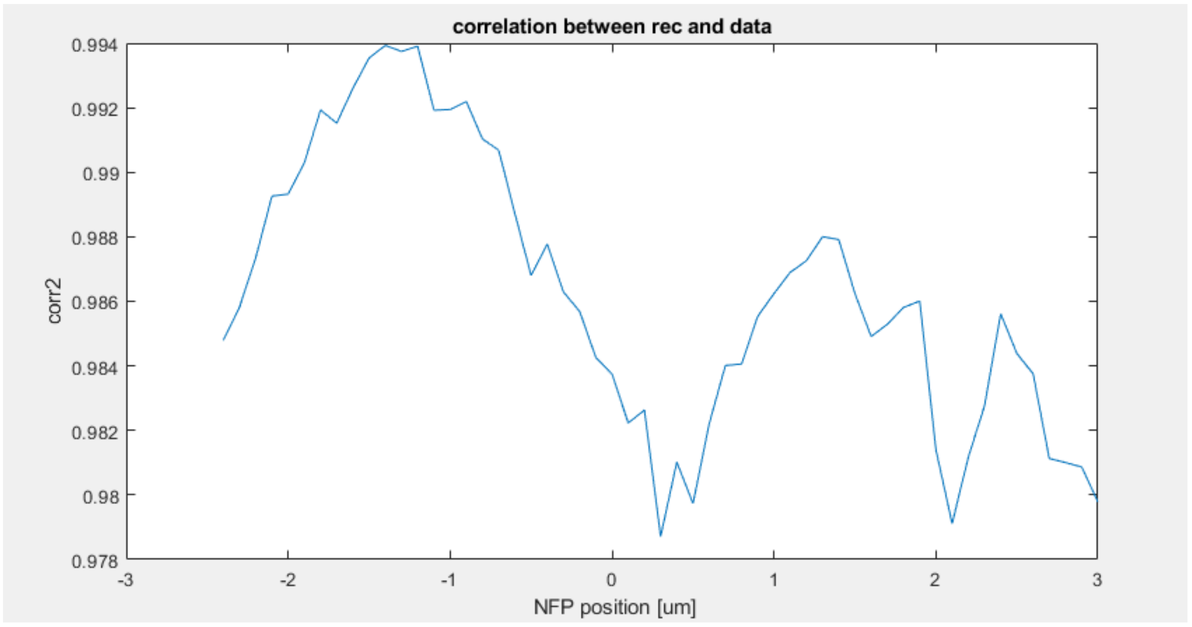
```
% additional options
IS.gBlur_cost = 2; % cost to estimate gBlur if est_gBlur_flag=1, (1-4 same as cost_function_flag , 5 - by
% option not to use SGD
IS.last_iter = 50; % how many iterations  to run not with SGD (at end of optimization)
IS.last_iter_flag = 1; % 1 - contuine SGD, 2 - global gradient, 3- batch the lowest correlation points,
%                     4- adaptive sampling with side info on corr
IS.thr_corr = 0.01; % threshold for correlation calc (used if last_iter_flag = 3)
IS.upsample_fact = 1; % how much to upsample the data (usually leave at 1)
IS.update_Signal = 0; % 1 - update signal at second half of iterations
IS.Photobleach_cost_mult = 0; % add to cost the SNR consideration
% plot sizes for PSF
IS.plotsize = 99 ; % size of psf plots [pixels]
```

<u>gBlur_cost : </u>which cost function to estimate the optical blur with

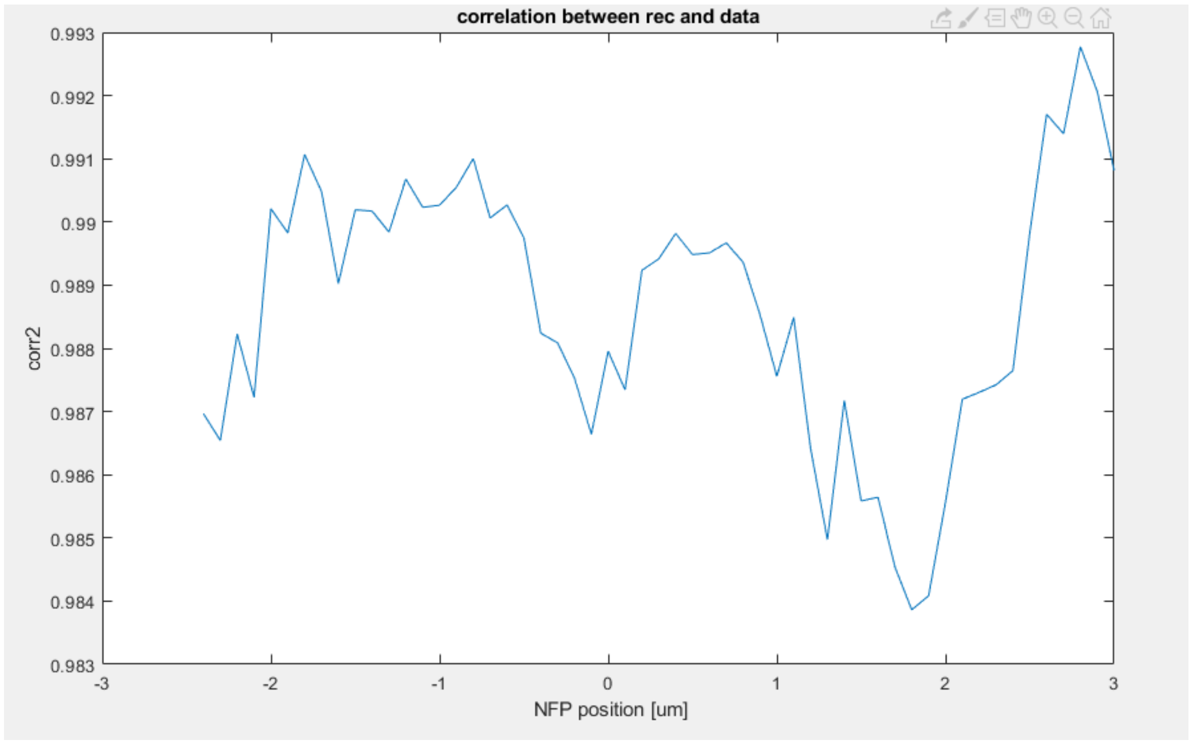
(if est_gBlur_flag = 1). Use L2(=2) or the gaussian cost (=4) was found to be the most consistent.

Last_iter : how many iterations to add in the end of the optimization that are not regular stochastic sampling (chosen by the following flag). A good way to know if such measures are required is by looking (at the end of the optimization) at the 2D correlation between the model and the data. Good for ‘’flattening’’ the PR matching over the z-range.

Example: doing only SGD (200 iterations) – not a bad result at all.



Example: adding extra 50 directed sampling steps after SGD (200 iterations)



Finally, 3 extra options are added for more challenging data.

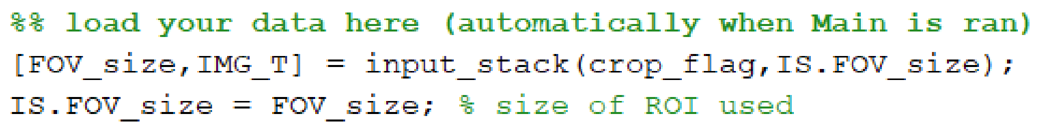
Upsample: usually leave 1, this will imresize the data and reduce the Camera pixel size (with the same factor). Use if the pixel size (object plane) is too big (can happen with low NA air objectives).

Update_signal: adds an adaptive signal estimation. If it is 0, the signal is estimated as the sum of the image. If it is 1, at each iteration, the signal is estimated by what factor the image needs to multiply by the match the data better. In the tetrapod/double helix examples it is not necessary (and it doesn’t change the results much). Consider using this as a ‘’last resort’’ if all other options fail.

Photobleach_cost_multi: if enabled, this adds weights to the gradients according to the image sum (per slice). Use this if the photobleaching effect is significant in your data.

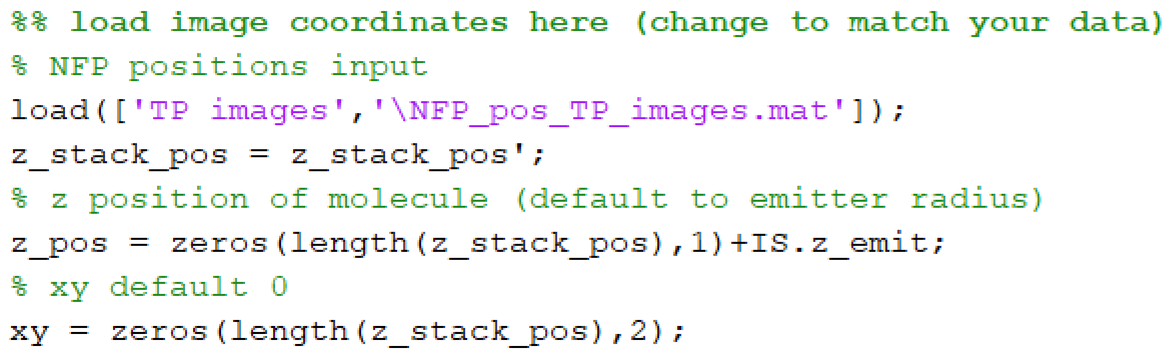
D) Input your data.

a) Open the function ‘’VIPR_load_data.mat’’
   The first part runs automatically, no need to change it.



```
%% load your data here (automatically when Main is ran)
[FOV_size,IMG_T] = input_stack(crop_flag,IS.FOV_size);
IS.FOV_size = FOV_size; % size of ROI used
```

b) The second part is your input of the coordinates (x,y,z,NFP) for each image, just make sure it’s the same size of amount of images.



```
%% load image coordinates here (change to match your data)
% NFP positions input
load(['TP images','\NFP_pos_TP_images.mat']);
z_stack_pos = z_stack_pos';
% z position of molecule (default to emitter radius)
z_pos = zeros(length(z_stack_pos),1)+IS.z_emit;
% xy default 0
xy = zeros(length(z_stack_pos),2);
```

The default loads NFP positions for the tetrapod, change it in any way you want

E) <u>Run ‘’main.mat’’.</u>

On The github, you can view the plots that are presented during optimization.

The output of the function is:

maskRec – the recovered phase mask.
gB – the gaussian blur estimation (changed from the initial guess if enabled).
Nph – the estimated signal sum of each stack.
I_mod – a generated z-stack, use to compare to the input data.