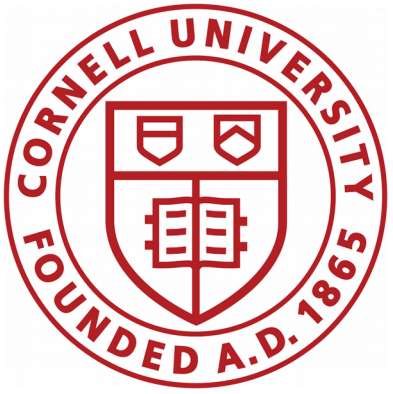


CBPM3 display

Antoine, Jim

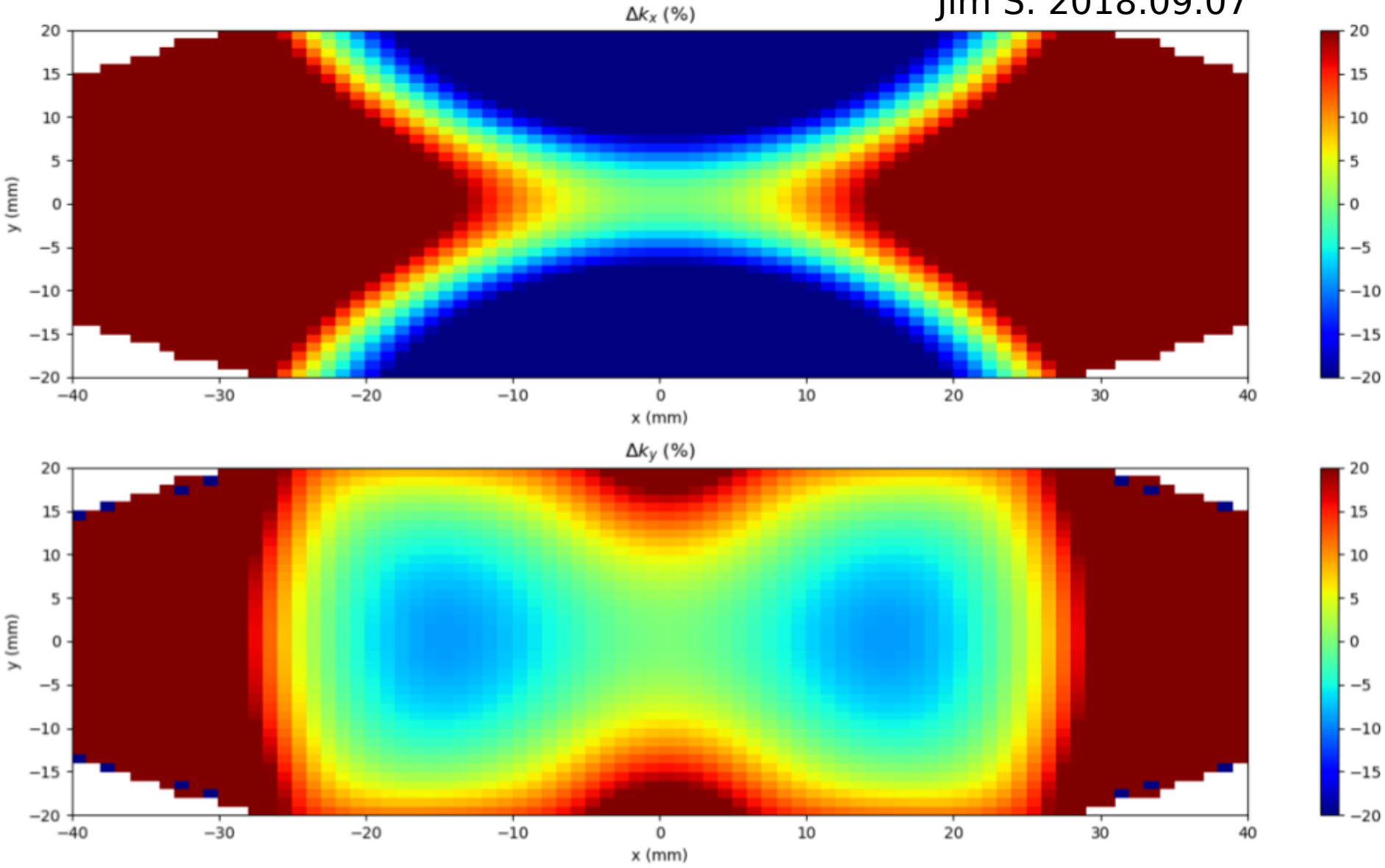
CBPM meeting

January 15, 2021

Non-linear (Poisson) beam position reconstruction

k_x and k_y deviation from linear regime:

Jim S. 2018.09.07



Poisson look-up table

Look-up table containing the non-linear response of button 1 as a function of static charge position (x,y)

/ [ACC] / CESR / CESR_instr / nonlin_bpm / nonlin_bpm_resource

Index of /CESR/CESR_instr/nonlin_bpm/nonlin_bpm_resource

Files shown: 9
Directory revision: 42959 (of 49188)
Sticky Revision: Set

File	Rev.	Age	Author	Last log entry
Parent Directory				
bpm_00_xyp.txt	32754	6 years	mjf7	Moving cesr-related programs out of /trunk/src
bpm_01_xyp.txt	32754	6 years	mjf7	Moving cesr-related programs out of /trunk/src
bpm_02_xyp.txt	32754	6 years	mjf7	Moving cesr-related programs out of /trunk/src
bpm_48_xyp.txt	32754	6 years	mjf7	Moving cesr-related programs out of /trunk/src
bpm_49_xyp.txt	32754	6 years	mjf7	Moving cesr-related programs out of /trunk/src
bpm_arc_xyp.txt	32754	6 years	mjf7	Moving cesr-related programs out of /trunk/src
bpm_chessu_xyp.txt	42959	2 years	js583	Update for CHESS-U BPM indexing; add new CHESS-U quad extrusion table. --JSh
button_coefficients.dat	32754	6 years	mjf7	Moving cesr-related programs out of /trunk/src
detcal.ok	32754	6 years	mjf7	Moving cesr-related programs out of /trunk/src

CESRV Fortran routines

Lots of routines, and many lines of code

/ [ACC] / CESR / CESR_instr / nonlin_bpm / code

Index of /CESR/CESR_instr/nonlin_bpm/code

Files shown: **11**
Directory revision: 44617 (of 49189)
Sticky Revision: [ ] Set

File ▲	Rev.	Age	Author	Last log entry
Parent Directory				
nonlin_bpm_init.f90	44599	21 months	dcs16	Remove debug printout.
nonlin_bpm_interpolate.f90	32754	6 years	mjf7	Moving cesr-related programs out of /trunk/src
nonlin_bpm_minimize.f90	32754	6 years	mjf7	Moving cesr-related programs out of /trunk/src
nonlin_bpm_mod.f90	43827	23 months	sw565	replace bpm calibration file detcal.ok with offset.bpm
nonlin_bpm_real_coords.f90	32754	6 years	mjf7	Moving cesr-related programs out of /trunk/src
nonlin_bpm_set_pointers.f90	44617	21 months	sw565	set 9W to be CHESS-U pipe BPM
nonlin_butcon.f90	32754	6 years	mjf7	Moving cesr-related programs out of /trunk/src
nonlin_mrqmin_mod.f90	32754	6 years	mjf7	Moving cesr-related programs out of /trunk/src
nonlin_orbit.f90	43527	2 years	sw565	fix detector mapping
nonlin_phase.f90	32754	6 years	mjf7	Moving cesr-related programs out of /trunk/src
nonlin_xy_shake_components.f90	44443	21 months	sw565	add comment to indicate cbpm idx

CESRV Fortran routines

Question marks regarding what the code does and how/why, e.g.:

```
49    ! loop over buttons
50    do k = 1, 4
51       c = func_bpm%grid(ix,iy)%b(k)%c
52       ! adapted from NR routine bcuint
53       d(k,:,:) = 0.0
54       ! evaluate polynomials with effecient but incomprehensible method
55       do i = 4, 1, -1
56          d(k,0,0)=t*d(k,0,0)+((c(i,4)*u+c(i,3))*u+c(i,2))*u+c(i,1)
57          d(k,0,1)=t*d(k,0,1)+(3.0*c(i,4)*u+2.0*c(i,3))*u+c(i,2)
58          d(k,0,2)=t*d(k,0,2)+(6.0*c(i,4)*u+2.0*c(i,3))
59          d(k,1,0)=u*d(k,1,0)+(3.0*c(4,i)*t+2.0*c(3,i))*t+c(2,i)
60          d(k,2,0)=u*d(k,2,0)+(6.0*c(4,i)*t+2.0*c(3,i))
61       end do
62       do i = 4, 2, -1
63          d(k,1,1)=t*d(k,1,1)+(3.0*(i-1)*c(i,4)*u+2.0*(i-1)*c(i,3))*u+(i-1)*c(i,2)
64       end do
65    end do
66    d(:,0,0) = d(:,0,0)
67    d(:,1,0) = d(:,1,0)/(func_bpm%x(ix+1)-func_bpm%x(ix))
68    d(:,2,0) = d(:,2,0)/(func_bpm%x(ix+1)-func_bpm%x(ix))**2
69    d(:,0,1) = d(:,0,1)/(func_bpm%y(iy+1)-func_bpm%y(iy))
70    d(:,0,2) = d(:,0,2)/(func_bpm%y(iy+1)-func_bpm%y(iy))**2
71    d(:,1,1) = d(:,1,1)/((func_bpm%x(ix+1)-func_bpm%x(ix)) * &
72         (func_bpm%y(iy+1)-func_bpm%y(iy)))
73
74    d = d * x(3)
75  end subroutine nonlin_bpm_interpolate
```

Python non-linear code

Instead of copying/porting Fortran code: write Python code from “first principles” hoping for clearer, shorter, faster code.

<u>The procedure is understood:</u>

- ✗ fetch button 1 look-up table and create tables for button 2, 3, 4 via reflections
- ✗ apply normalization to look-up tables
- ✗ generate fine map interpolating tables
- ✗ minimize merit function to find (x,y) corresponding to measured b1, b2, b3, b4

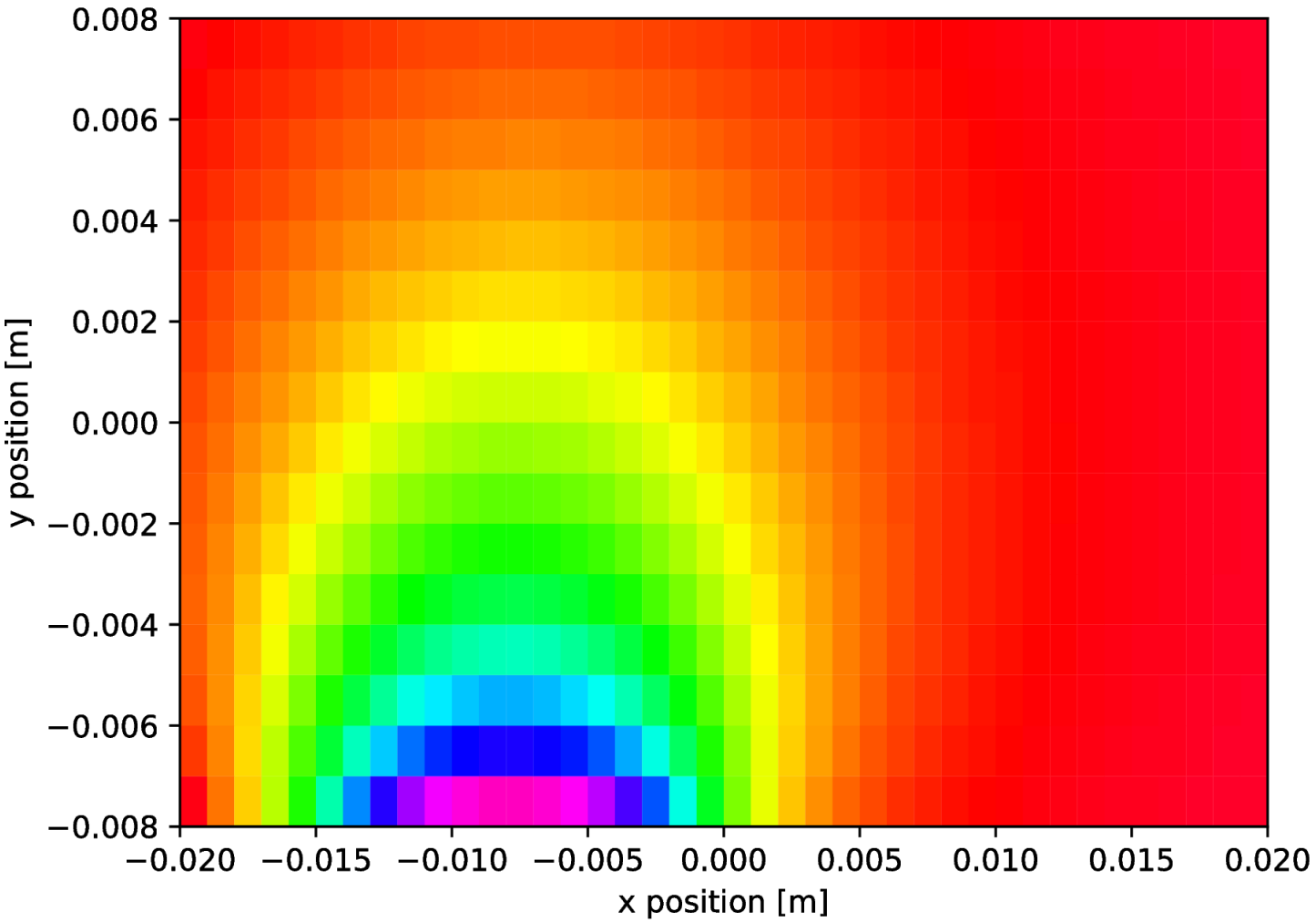
Look-up tables: fetch b1

Read-in and plot *bpm_chessu_xyp.txt* button 1 response:

```
table_name = 'table.txt'

x = []
y = []
q = []

with open(table_name, 'r') as table_file:
    for line in table_file:
        x.append(float(line.split()[0]))
        y.append(float(line.split("    ")[1]))
        q.append(float(line.split("    ")[2]))

x = np.array(x)
y = np.array(y)
q = np.array(q)

b1_x = x
b1_y = y
```

Look-up tables: fetch b1

Read-in and plot *bpm_chessu_xyp.txt* button 1 response:

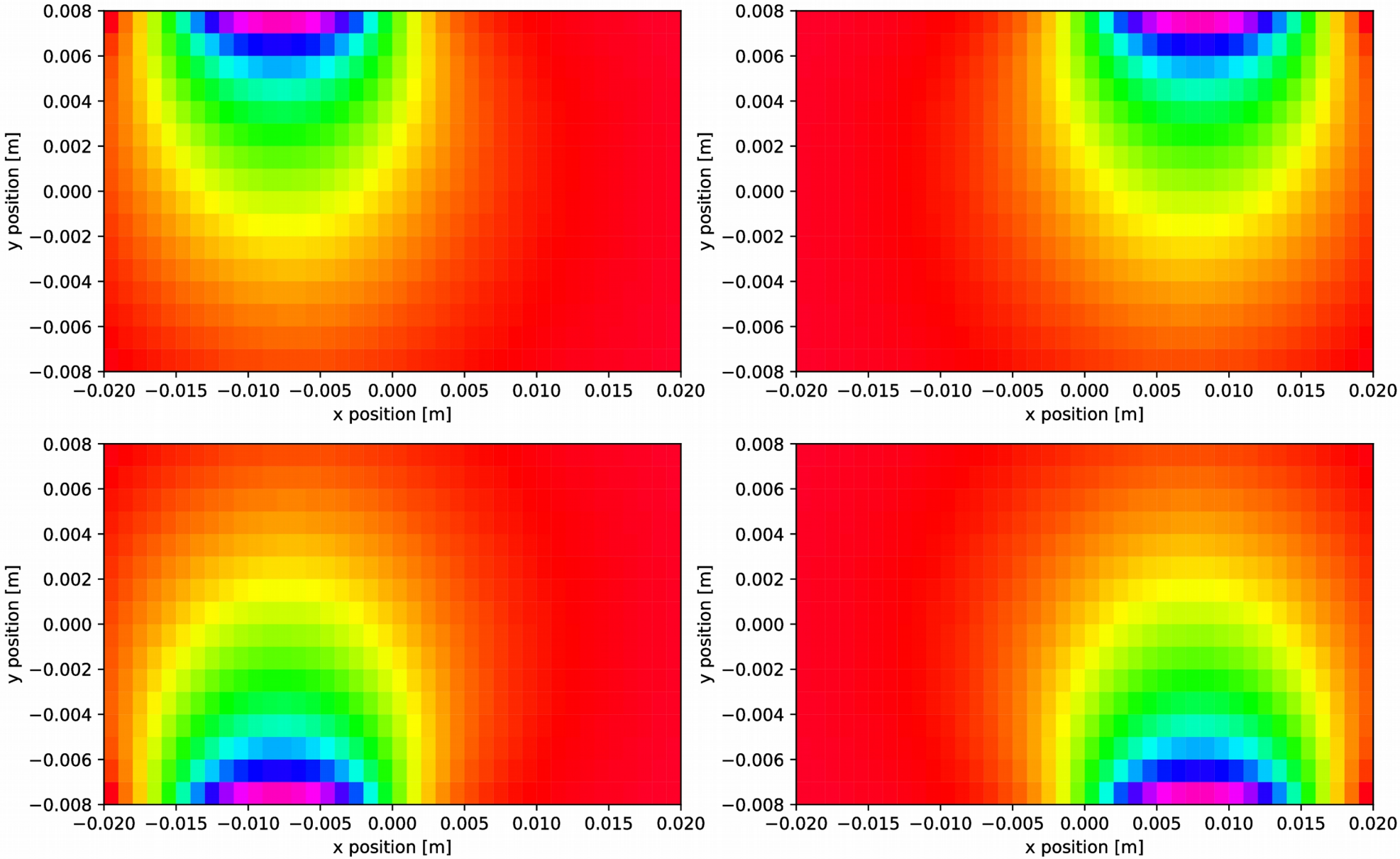


Look-up tables: reflect b1 → b2, b3, b4

Reflect b1 so to obtain b2, b3 and b4:

```
b2_x = x[::-1]
b2_y = y

b3_x = x
b3_y = y[::-1]

b4_x = x[::-1]
b4_y = y[::-1]
```

Look-up tables: reflect b1 → b2, b3, b4



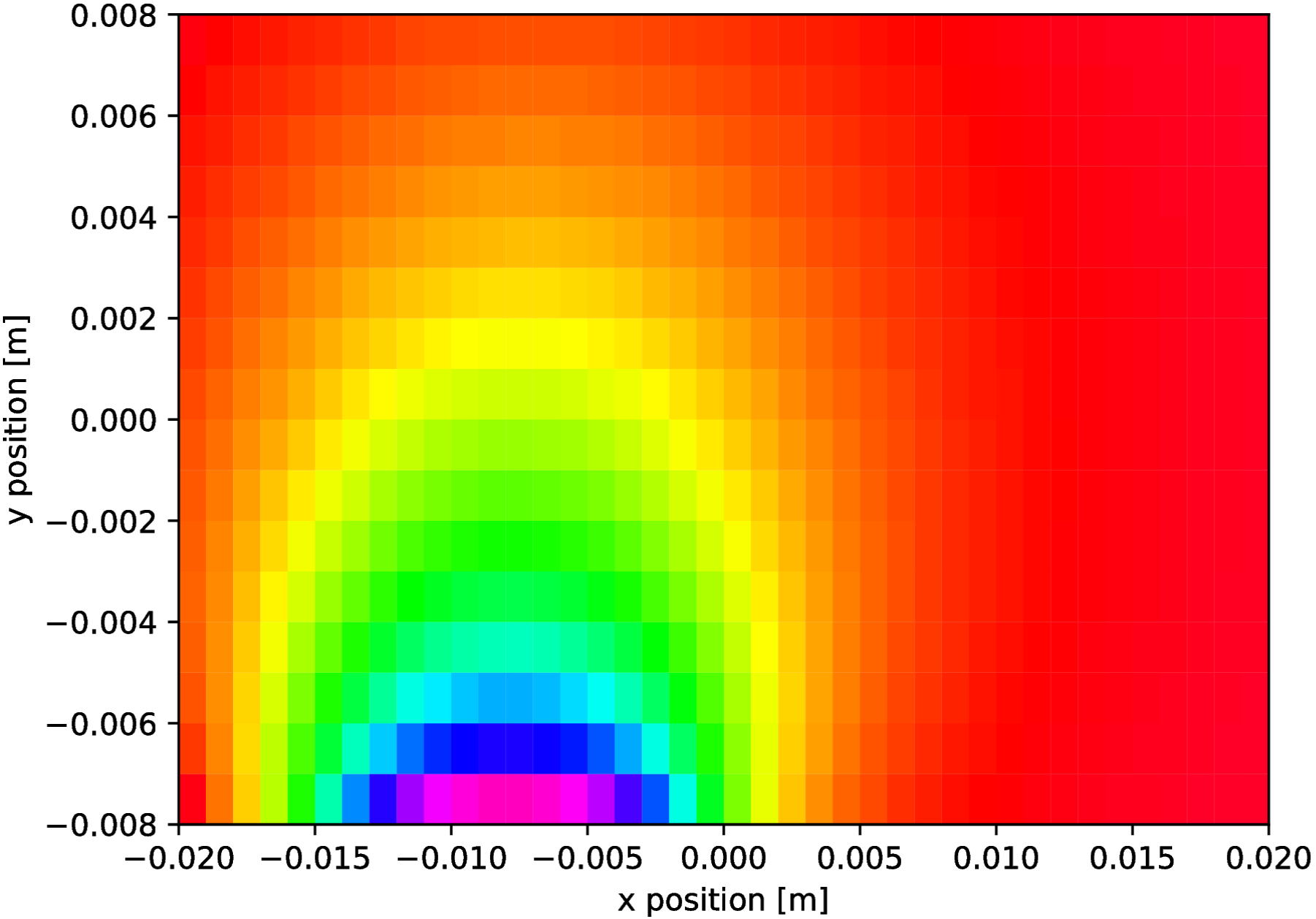
Normalization

Look-up tables are normalized in some way that has nothing to do with actual button amplitudes we measure

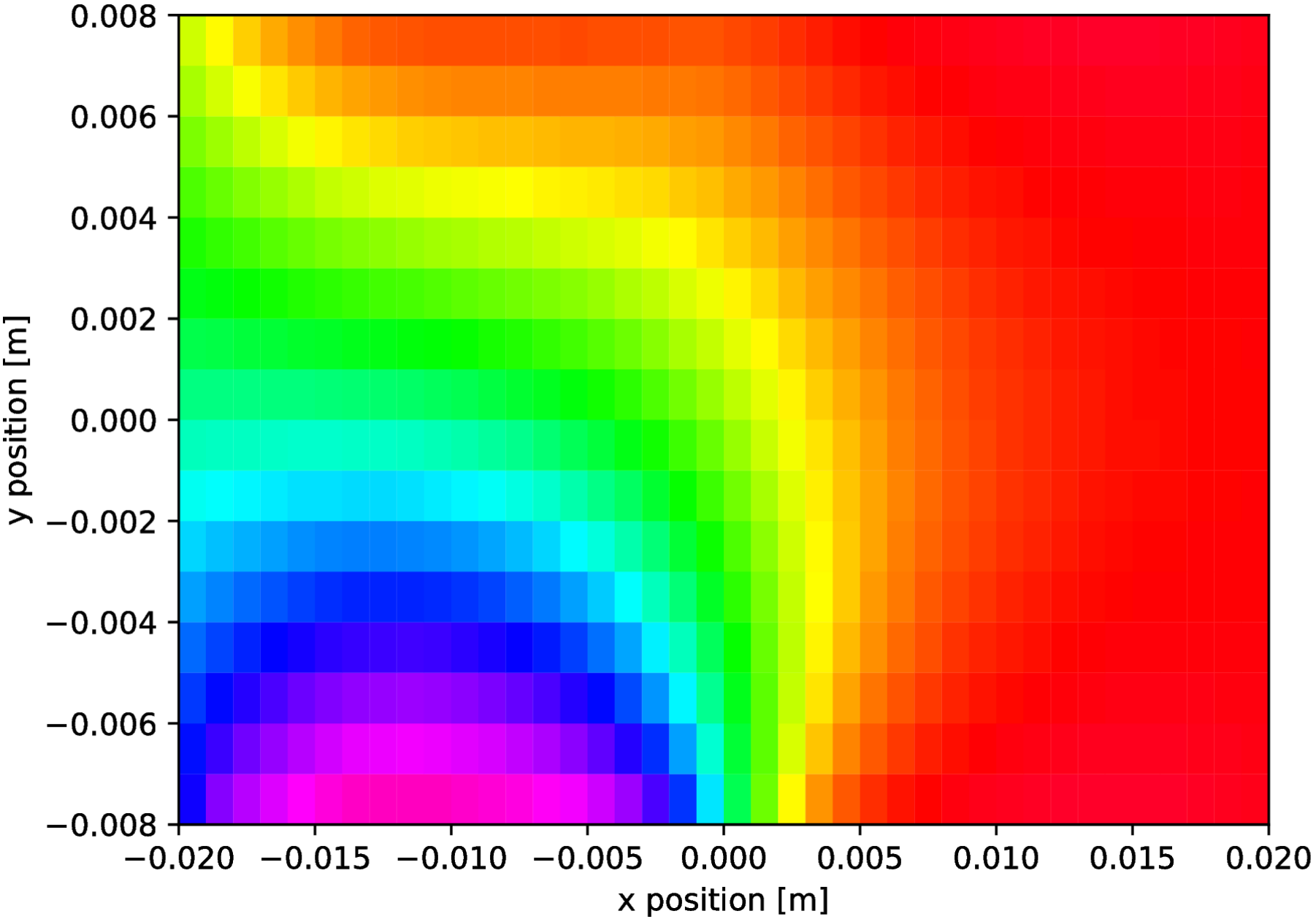
A straightforward normalization to allow measured button amplitudes and look-up table to be compared: normalize individual button amplitude by the sum of the 4 buttons

```
count = 0

q_norm = np.copy(q)

for b1x, b1y, q1 in zip(b1_x,b1_y,q):
    sum_q = q1
    for b2x, b2y, q2 in zip(b2_x,b2_y,q):
        if b2x == b1x and b2y == b1y:
            sum_q += q2
    for b3x, b3y, q3 in zip(b3_x,b3_y,q):
        if b3x == b1x and b3y == b1y:
            sum_q += q3
    for b4x, b4y, q4 in zip(b4_x,b4_y,q):
        if b4x == b1x and b4y == b1y:
            sum_q += q4

    q_norm[count] = q1/sum_q
    count += 1
```

Button 1 **before** normalization



Button 1 **after** normalization



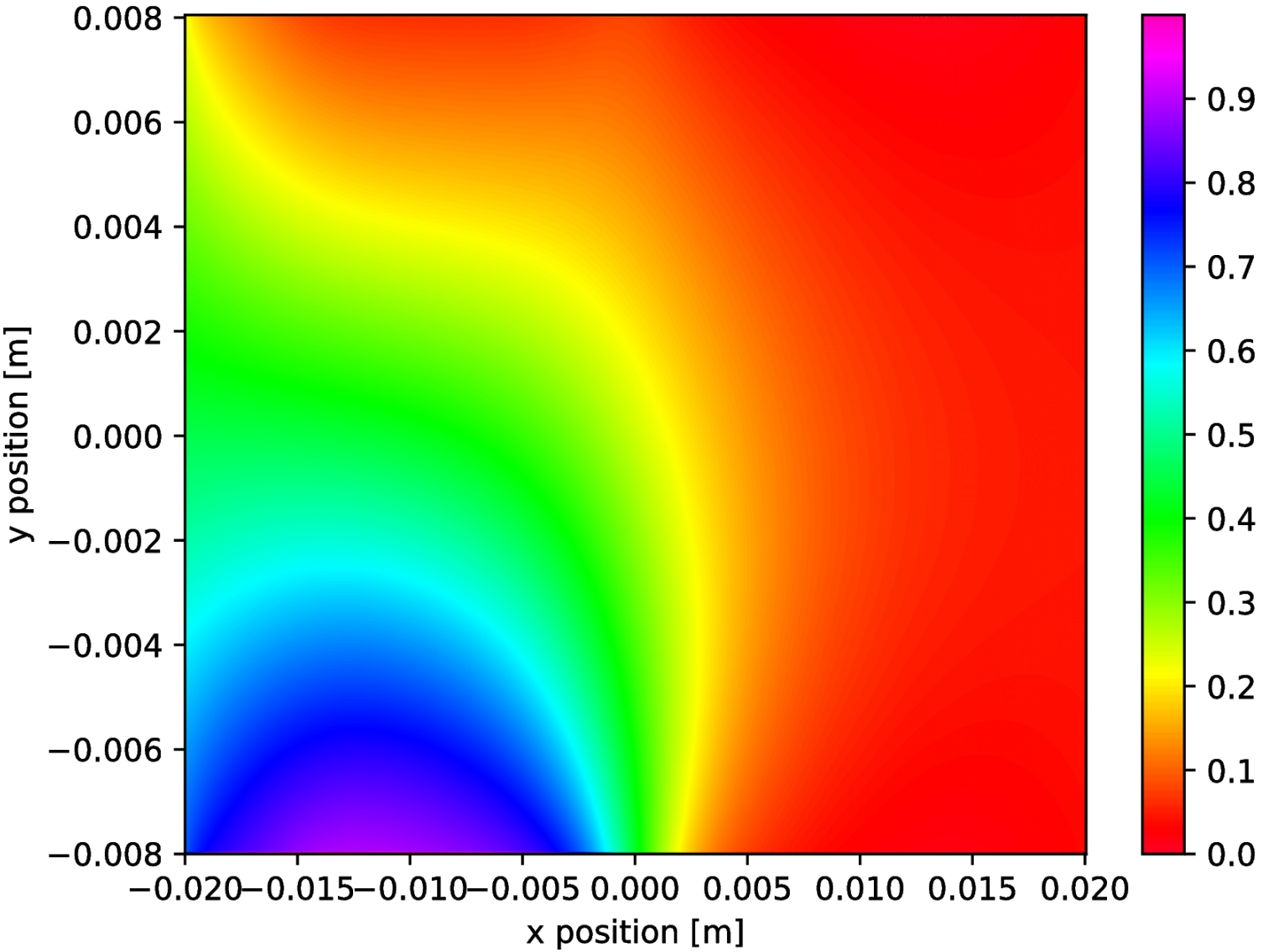
2D Interpolation

Quintic (order 5) 2D interpolation:

```
b1_X = np.unique(b1_x)
b1_Y = np.unique(b1_y)

b2_X = np.unique(b2_x)[::-1]
b2_Y = np.unique(b2_y)

b3_X = np.unique(b3_x)
b3_Y = np.unique(b3_y)[::-1]

b4_X = np.unique(b4_x)[::-1]
b4_Y = np.unique(b4_y)[::-1]

q = q.reshape(17,41)
q_norm = q_norm.reshape(17,41)

# TO DO: compare RectBivariateSplit vs interp2d
f_b1 = interp2d(b1_X, b1_Y, q, kind='quintic')
f_b2 = interp2d(b2_X, b2_Y, q, kind='quintic')
f_b3 = interp2d(b3_X, b3_Y, q, kind='quintic')
f_b4 = interp2d(b4_X, b4_Y, q, kind='quintic')

f_norm_b1 = interp2d(b1_X, b1_Y, q_norm, kind='quintic')
f_norm_b2 = interp2d(b2_X, b2_Y, q_norm, kind='quintic')
f_norm_b3 = interp2d(b3_X, b3_Y, q_norm, kind='quintic')
f_norm_b4 = interp2d(b4_X, b4_Y, q_norm, kind='quintic')
```

2D Interpolation

Quintic (order 5) 2D interpolation:



Minimization to extract (x,y)

The goal is to match normalized measured amplitudes to normalized look-up table amplitudes to extract best (x,y). One merit function to minimize can be:

$$\sqrt{\Sigma_i (b_i - f_i(x,y))^2}$$

sum over 4 buttons

measured i^{th} button amplitude

2D interpolation of look-up table

two unknowns

Sum of square is required to avoid cancellation between different buttons as we want to minimize the difference between measured/look-up for all the buttons. The overall square root is used to go back to the “natural” unit of button amplitude.

Minimization to extract (x,y)

First pass:

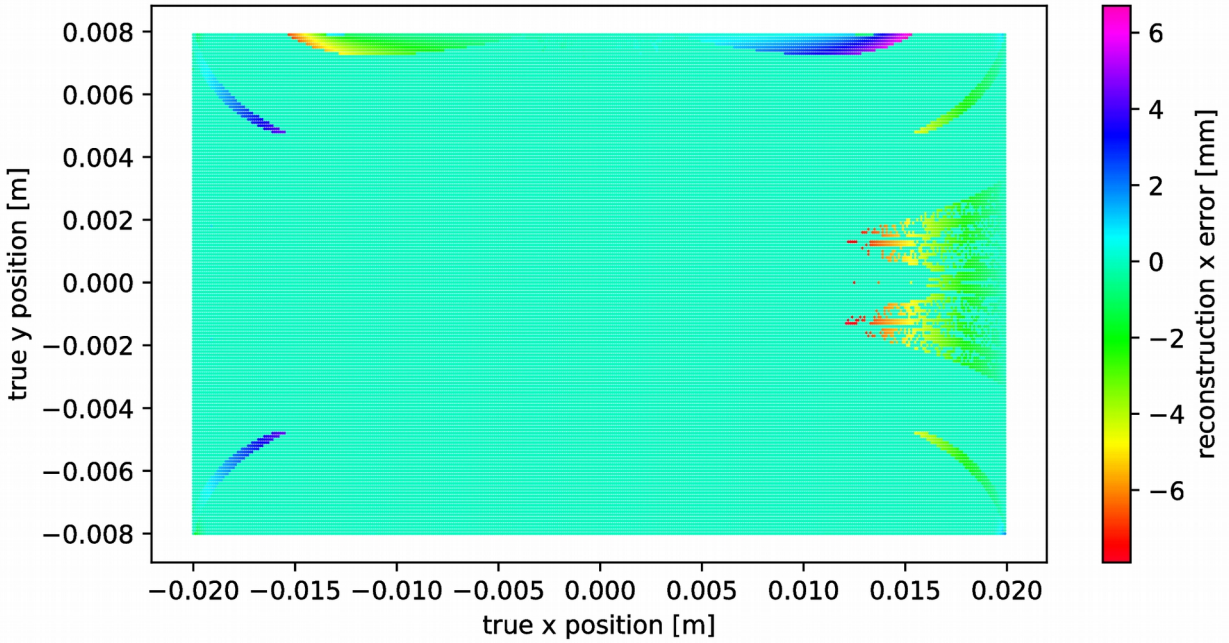
- ✗ use (x,y) computed with k_x=0.0102 m, k_y=0.0104 m as initial guesses
- ✗ bound minimization to entire 2D space (x),(y)=(-0.02,0.02),(-0.008,0.008)
- ✗ default minimizer configurations

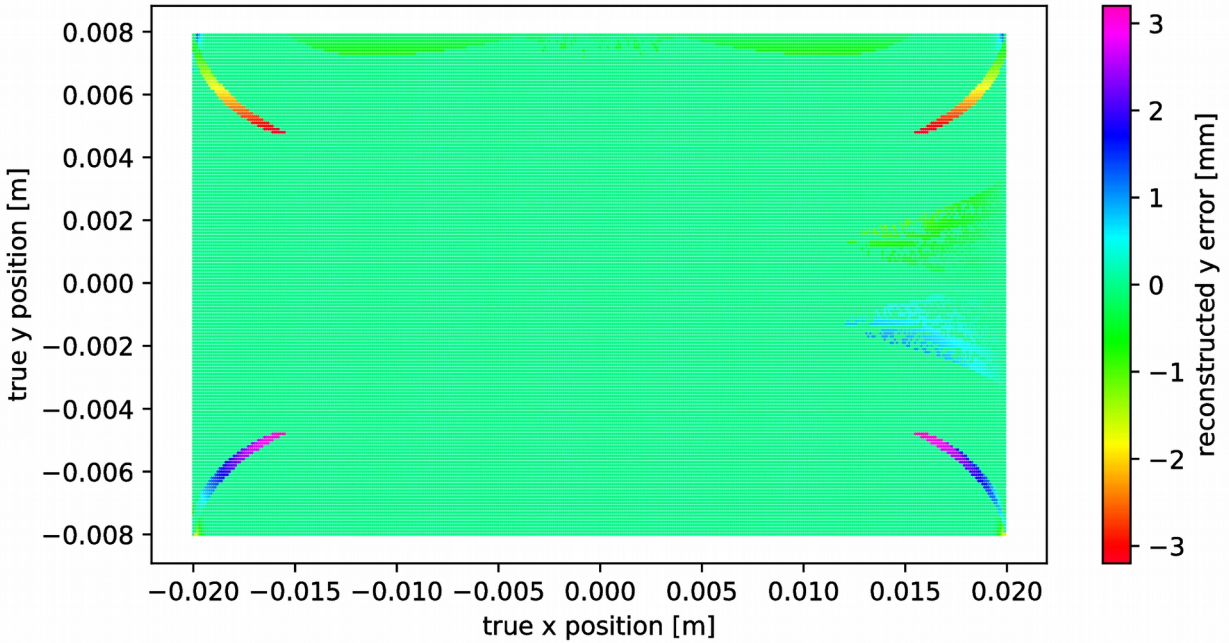
```
def pos(params, *args):
    x, y = params
    return np.sqrt((args[0]-f_norm_b1(x, y))**2+(args[1]-f_norm_b2(x, y))**2+(args[2]-f_norm_b3(x, y))**2+(+args[3]-f_norm_b4(x, y))**2)
```

```
result = optimize.minimize(
                           pos,
                           initial_guess,
                           args=(b1, b2, b3, b4),
                           method='SLSQP',
                           bounds=((-0.02, 0.02), (-0.008, 0.008))
                          )
```

Minimization to extract (x,y)

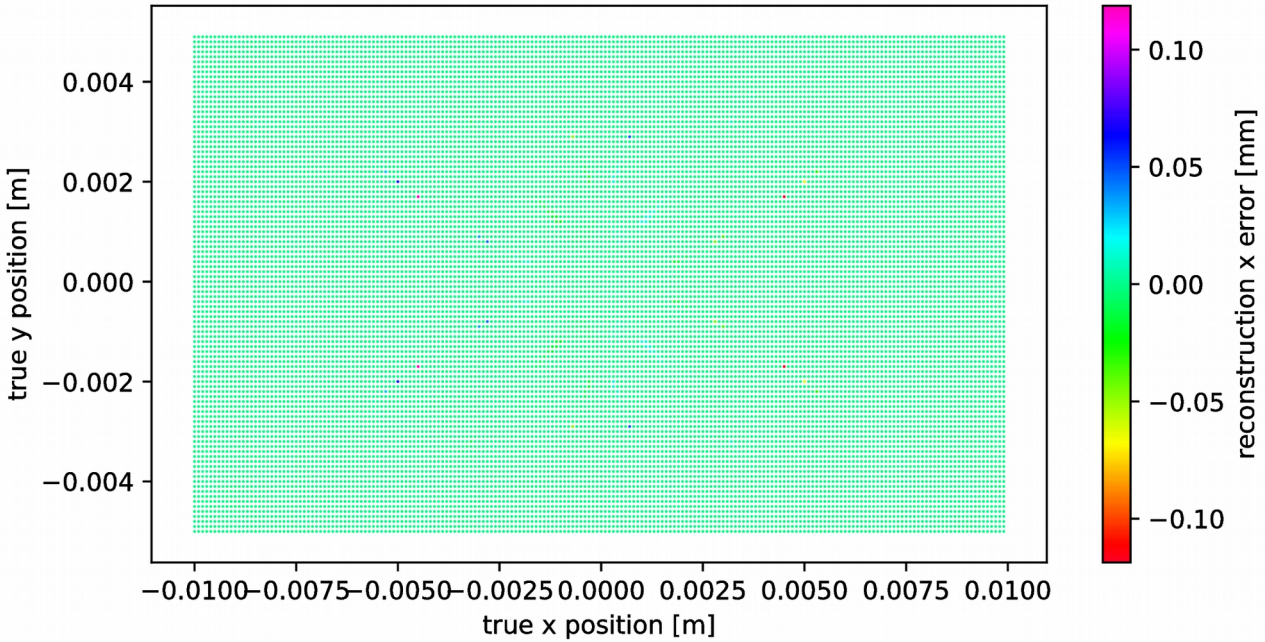
First pass:

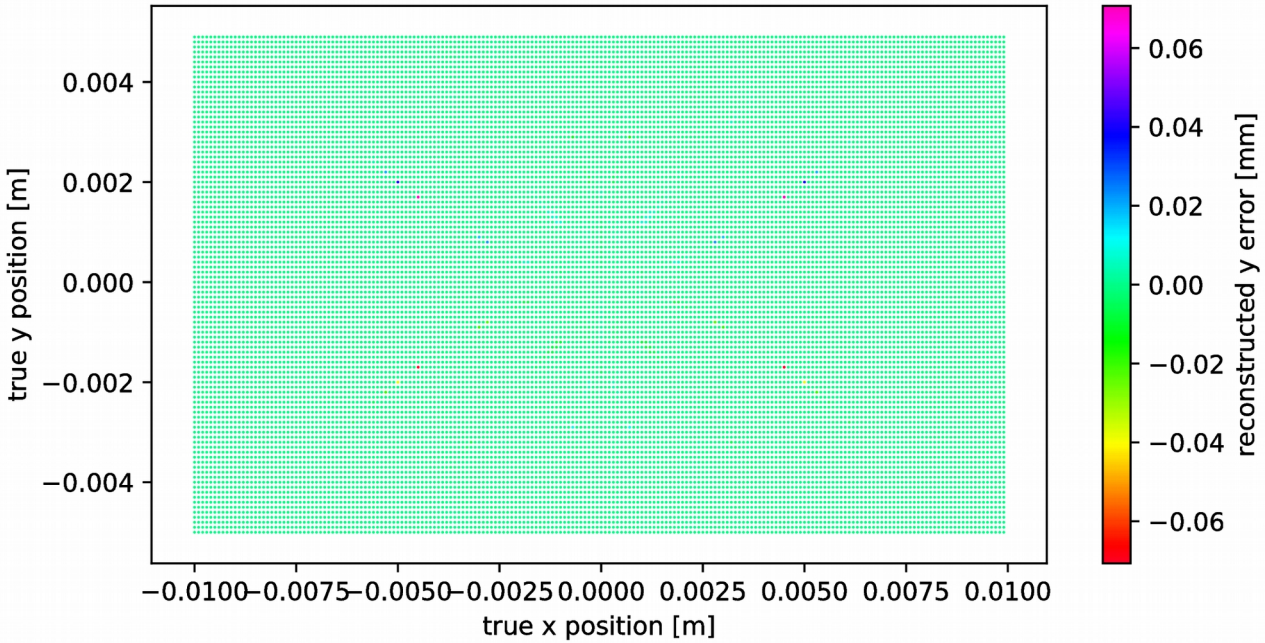




Minimization to extract (x,y)

First pass:





Outlook

First pass at the procedure is promising:

✗ refinements, checks and optimization (accuracy and speed) needed

Once the Python code is fully understood and trusted, results will be compared with Fortran routine

Additional materials

Minimization to extract (x,y)

Second path:

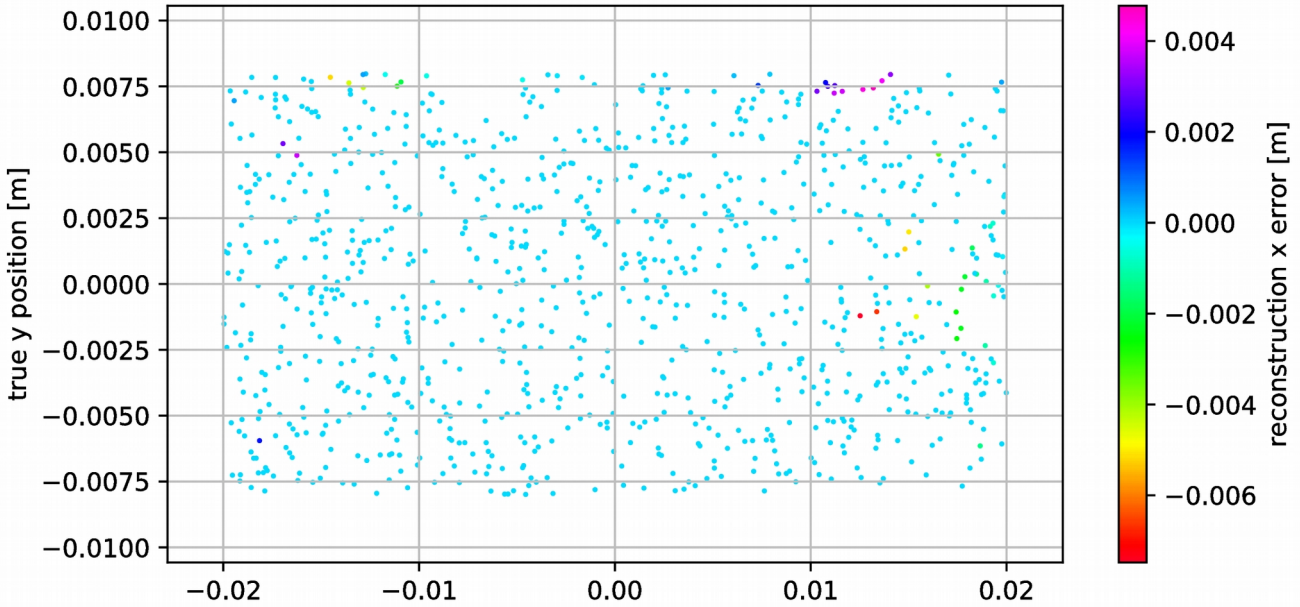
- ✗ use (x,y) computed with k_x=0.0102 m, k_y=0.0104 m as initial guesses
- ✗ bound minimization to entire 2D space (x),(y)=(-0.02,0.02),(-0.008,0.008)
- ✗ tweaked minimizer configurations

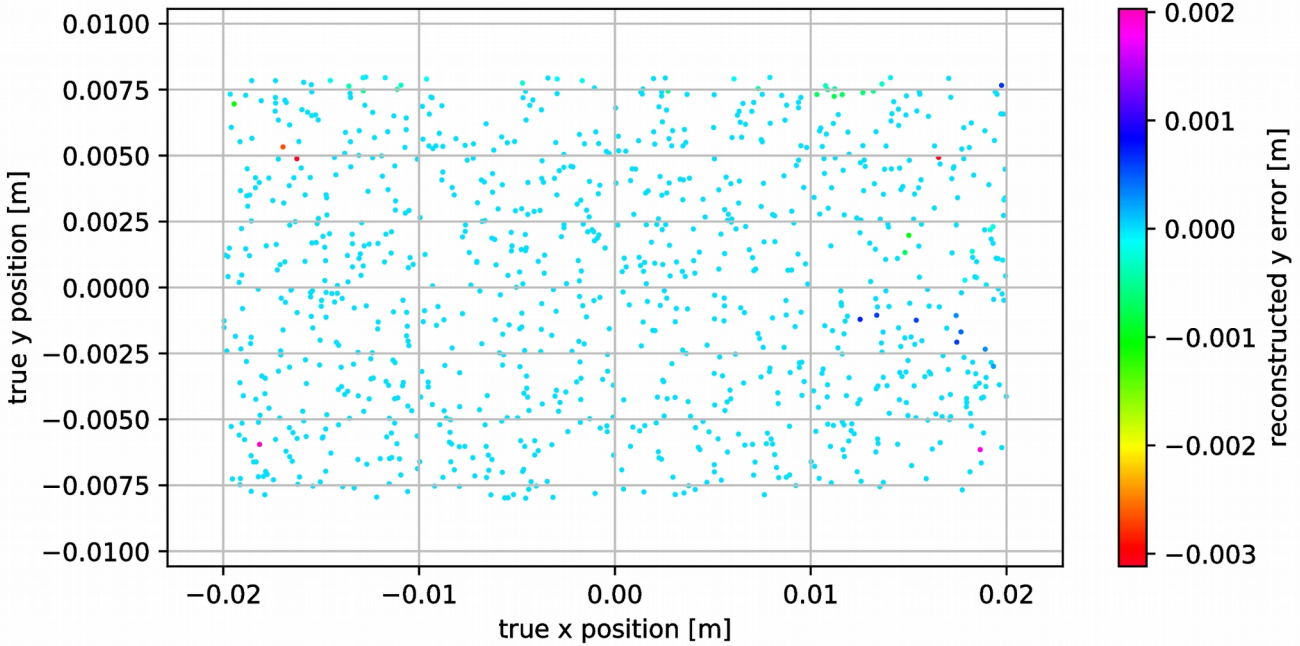
```
def pos(params, *args):
    x, y = params
    return np.sqrt((args[0]-f_norm_b1(x, y))**2+(args[1]-f_norm_b2(x, y))**2+(args[2]-f_norm_b3(x, y))**2+(+args[3]-f_norm_b4(x, y))**2)
```

```
result = optimize.minimize(
                           pos,
                           initial_guess,
                           args=(b1, b2, b3, b4),
                           method='SLSQP',
                           bounds=((-0.02, 0.02), (-0.008, 0.008))
                          )
```

Minimization to extract (x,y)

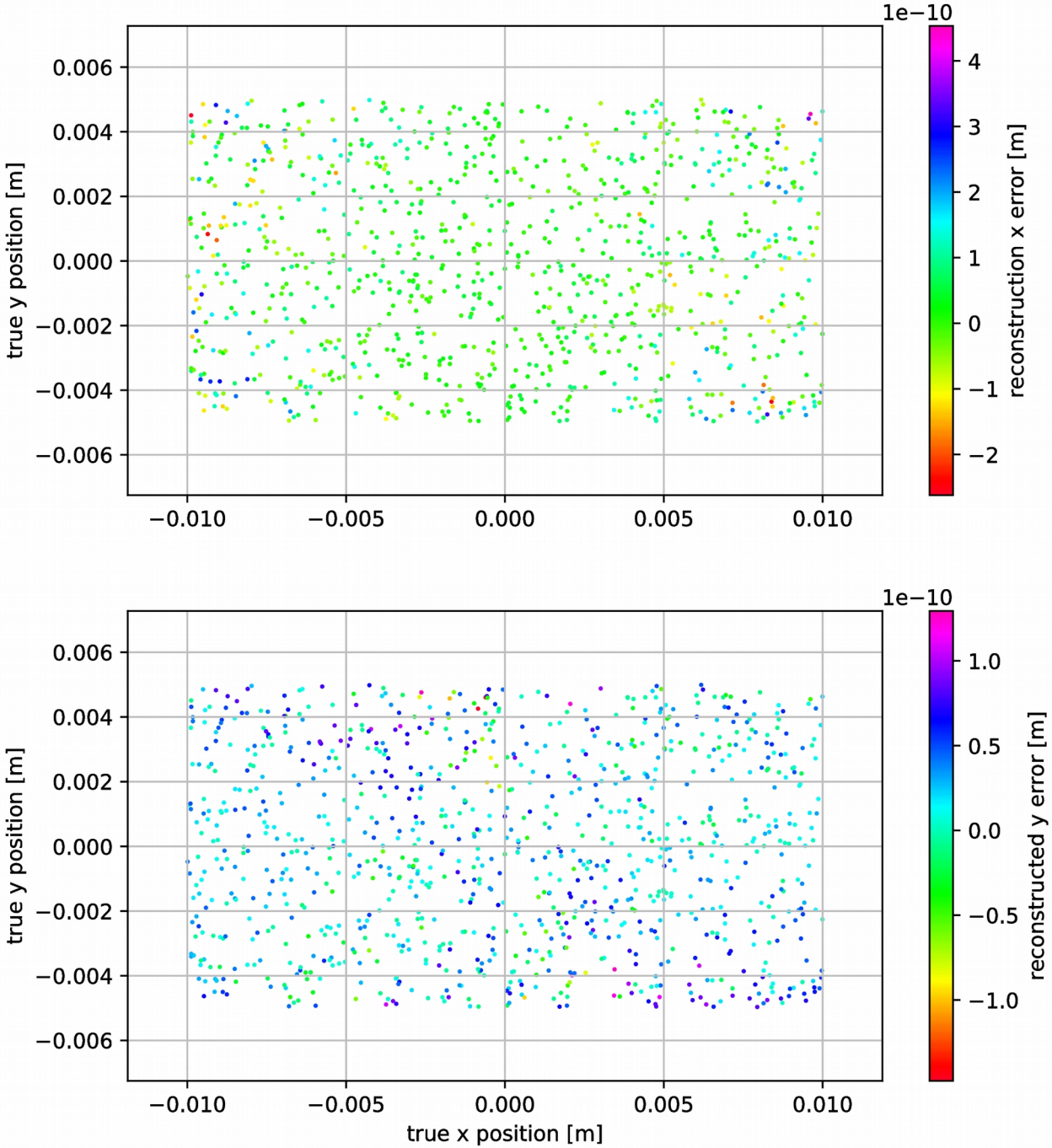
Second path:





Minimization to extract (x,y)

Second path:



Minimization to extract (x,y)

Third path:

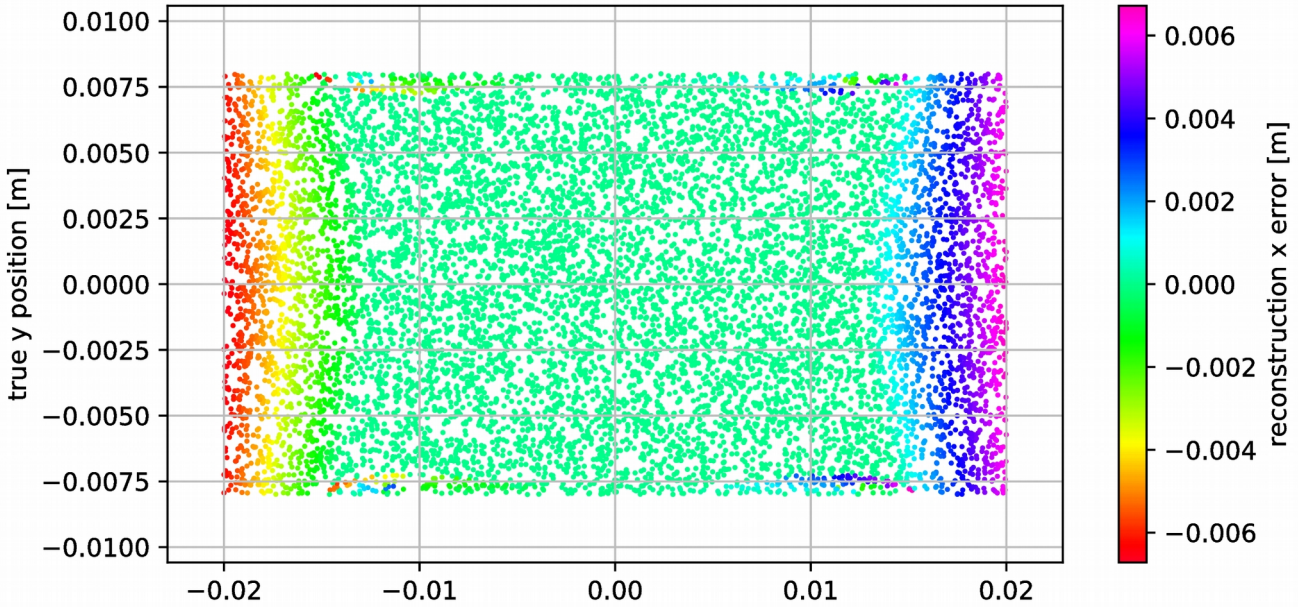
- ✗ use (x,y) computed with k_x=0.0102 m, k_y=0.0104 m as initial guesses
- ✗ bound minimization around ±5mm and ±2mm of initial guesses
- ✗ tweaked minimizer configurations to increase expected precision

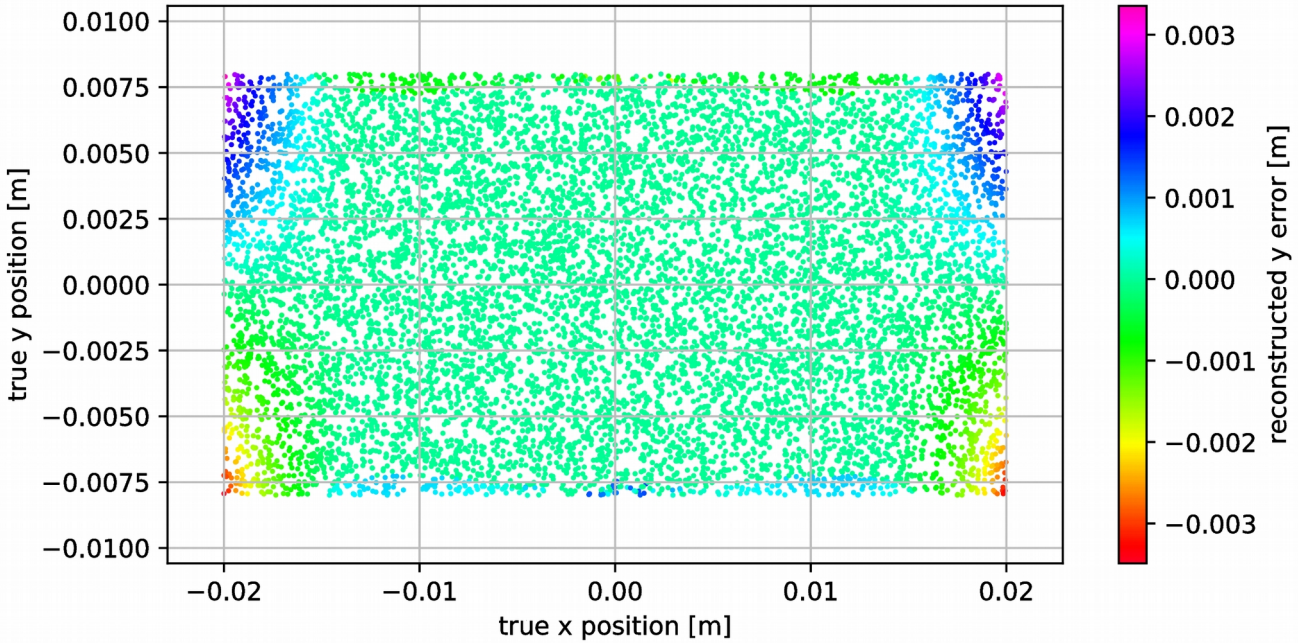
```
def pos(params, *args):
    x, y = params
    return np.sqrt((args[0]-f_norm_b1(x, y))**2+(args[1]-f_norm_b2(x, y))**2+(args[2]-f_norm_b3(x, y))**2+(+args[3]-f_norm_b4(x, y))**2)
```

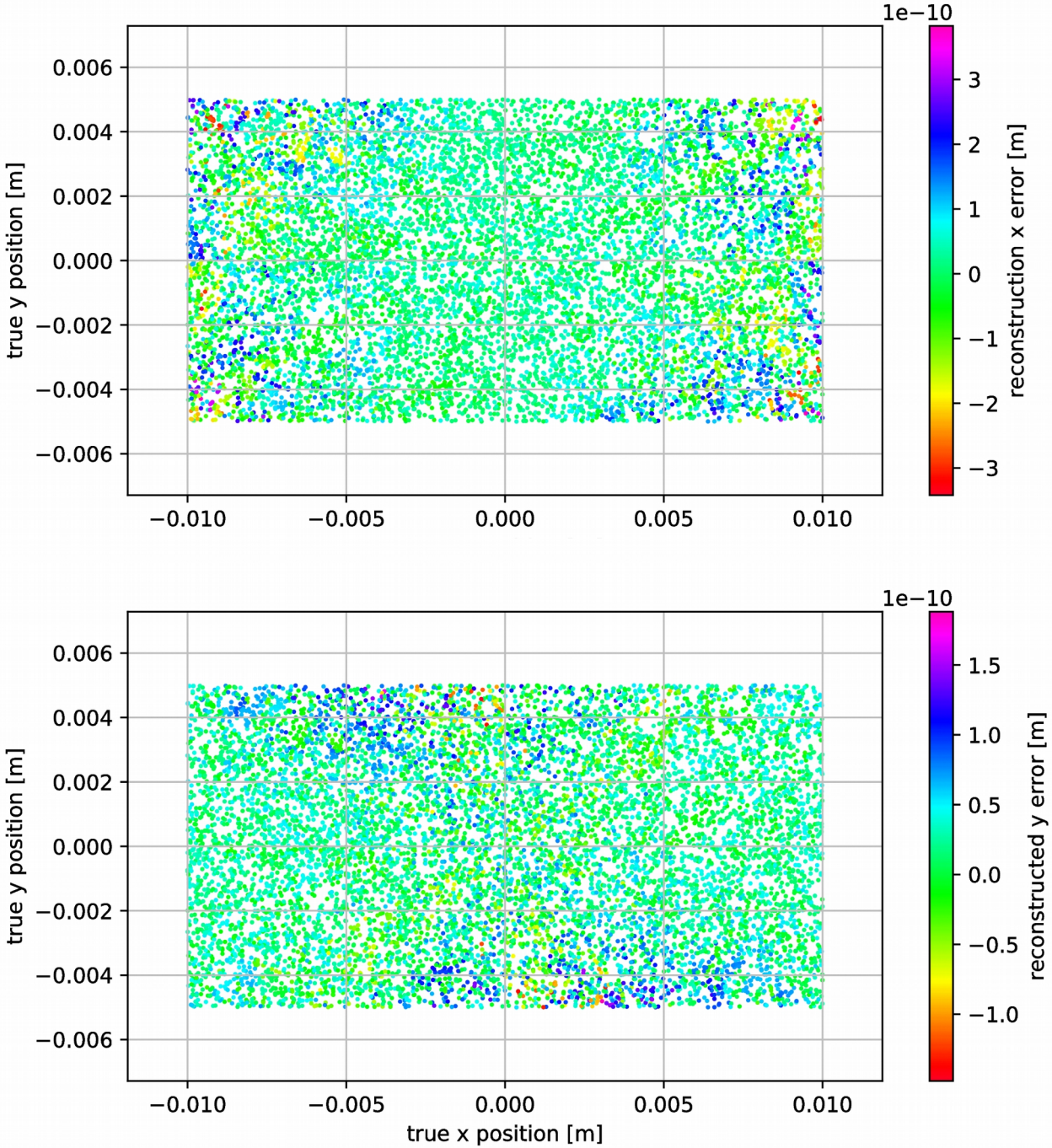
```
result = optimize.minimize(
                           pos,
                           initial_guess,
                           args=(b1, b2, b3, b4),
                           method='SLSQP',
                           bounds=((-0.02, 0.02), (-0.008, 0.008))
                          )
```

Minimization to extract (x,y)

Third path:





Minimization to extract (x,y)

Third path: