本稿は，2010 年 4 月から 2011 年 6 月にかけて東京大学新領域・三尾研究室でおこなわれた散乱マッピング計測系の制御・計測ツールについてのおぼえがきです．2009 年度まで制御計測ツールは Windows XP + Borland/Inprise C compiler の環境で C 言語を用いて作成していましたが，2010 年に，パラメータ変更の省力化をねらって Python PyVISA ライブラリの利用を試みています．

本稿の LaTeX ソースと付属プログラムは，本 PDF ファイルに PDF 添付されています．

付属プログラムの動作試験をした際に用いた機器は，以下のとおりです．

- Windows XP 搭載ノートパソコン
- Agilent 82357A USB-GPIB アダプタ
- NF LI5640 ロックインアンプ (2 機)
- シグマ光機 OMEC-4BG オプトマイクコントローラ (エンコーダ付き自動ステージコントローラ)

1 GP-IB 通信試験の例

NF 社 LI5640 の GP-IB 通信系は IEEE-488 の IDN コマンドに対応しているので，下記のスクリプトで型番やシリアル番号が取得できます．シグマ光機 OMEC-4BG は IDN コマンドに対応していないので，かわりに問い合わせ指令 N (機器名称), V (ファームウエアバージョン) で得られる情報を記録しています．

idn.py

```
#!/usr/bin/env python
import csv
import visa as vi

fout = open('idn1.csv', 'wb')
csvOut = csv.writer(fout)
csvOut.writerow(['creator', 'idn.py'])
li56int = vi.instrument('GPIB::2')
li56mon = vi.instrument('GPIB::3')
omec4 = vi.instrument('GPIB::8')
li56int.write('*IDN?')
csvOut.writerow(['li56int.idn', li56int.read()])
li56mon.write('*IDN?')
csvOut.writerow(['li56mon.idn', li56mon.read()])
omec4.write('?:N')
csvOut.writerow(['omec.name', omec4.read()])
omec4.write('?:V')
csvOut.writerow(['omec.ver', omec4.read()])
```

2　サンプルステージの原点復帰の例

自動ステージ制御を制御計算機側で中断した場合，ステージの位置は半端な位置でとまったままとなります．また，ステージコントローラが長距離移動の命令を受けた直後に制御計算機側でプログラムを中断した場合，移動が終了するまでステージコントローラに次の指令を送っても無視されます．下記スクリプトを実行すると，ステージが長距離移動中であってもそれが終了するのを待ち，その後に計測原点復帰のコマンドを送出します．

homepos.py

```
#!/usr/bin/env python
import time
import visa as vi

omec4 = vi.instrument('GPIB::8') # Sigma OMEC-4BG stage controller
def wait_move(instr):
    while 1:
        resp = instr.ask('1Q')
        print resp
        if resp.find('B')==-1: # busy if 'B' exists
            break

wait_move(omec4)
omec4.write('1A3:+0,+0')
```

3　複数ロックインアンプのトリガ同期とデータ転送の例

本計測系では，二台のフォトディテクタからの信号を独立したロックインアンプで同時測定し，計算機上で比をとることで光源強度ゆらぎに影響されにくい散乱データを得る方式をとっています．ロックインアンプの同時測定のために，GP-IB バスコマンドのトリガ機能を利用しています．Python プログラムから GP-IB バスコマンドに任意のバイト列を送出するには，visa のインポートだけでは不十分で，pivisa ライブラリの vpp43 モジュールを読み込む必要があります．以下に，二台のロックインアンプでトリガ同期計測をおこない，アンプ内部データメモリから計測データ列を転送し，平均処理を施した後に CSV ファイルにデータを保存します．

get_sig.py

```
#!/usr/bin/env python
import csv
import time
import visa as vi
from pyvisa import vpp43
import tkFileDialog as tkf
import numpy as np

creator = 'get_sig.py'
revision = '2011-06-28'
```

```
fnameout = tkf.asksaveasfilename()
csvOut = csv.writer(open(fnameout, 'wb'))
colnames = ['#int_re', '#int_im',\
    '#mon_re', '#mon_im',\
    '#rat_re', '#rat_im', '', '']
ncol = len(colnames)
pad_data = ['#']*(ncol-2)
pad_dict = ['']*2
csvOut.writerow(pad_data+['creator', creator])
csvOut.writerow(pad_data+['revision', revision])
def vrange(strVSEN):
    k = int(strVSEN)-25
    d = k/3
    b = k%3-1
    return 10**d*2**b
intfc = vpp43.open(vi.resource_manager.session, 'GPIB::INTFC')
li56int = vi.instrument('GPIB::2') # LI5640 for integration sphere PD
li56mon = vi.instrument('GPIB::3') # LI5640 for monitor PD
li56int.write('*CLS')
li56mon.write('*CLS') # clear status
def rec_stat(vi, strGPIB, recName, csvOut):
    vi.write(strGPIB)
    str = vi.read()
    csvOut.writerow(pad_data+[recName, str])
    return str
rec_stat(li56int, '*IDN?', 'li56int.idn', csvOut)
vsen_int =\
    rec_stat(li56int, 'VSEN?', 'li56int.vsen', csvOut) # voltage sensitivity
rec_stat(li56int, 'TCON?', 'li56int.tcon', csvOut) # time constant
rec_stat(li56int, 'SLOP?', 'li56int.slop', csvOut) # filter slope
rec_stat(li56mon, '*IDN?', 'li56mon.idn', csvOut)
vsen_mon =\
    rec_stat(li56mon, 'VSEN?', 'li56mon.vsen', csvOut)
rec_stat(li56mon, 'TCON?', 'li56mon.tcon', csvOut)
rec_stat(li56mon, 'SLOP?', 'li56mon.slop', csvOut)
li_setting = 'DTYP 2;' # data memory type => DATA1,DATA2
li_setting += 'DSIZ 0;' # data memory size => 2048 records
li_setting += 'DNUM 0;' # data memory number => 0 (first block)
li_setting += 'DSMP 9' # data memory sampling period => 20msec
li56int.write(li_setting)
li56mon.write(li_setting)
n = 16 # number of samples for averaging
```

```
li56int.write('STRT')
li56mon.write('STRT') # start (waiting for a trigger)
cmd_trig = chr(0x3f) # unlisten (UNL)
cmd_trig += chr(0x20+0x02) # adding GPIB::2 to the group
cmd_trig += chr(0x20+0x03) # adding GPIB::3 to the group
cmd_trig += chr(0x08) # group execution trigger (GET)
vpp43.gpib_command(intfc, cmd_trig)
time.sleep(n*0.020)
li56int.write('STOP')
li56mon.write('STOP')
li56int.write('DASC? 1,%d' % n)
li56mon.write('DASC? 1,%d' % n) # ASCII data transfer
lcint, lcmon, lcrat = [], [], []
for i in range(n):
    sint = li56int.read().split(',')
    smon = li56mon.read().split(',')
    cint = 1.2/(2**15)*float(sint[0])*vrange(vsen_int)
    cint *= np.exp(np.pi*2j*float(sint[1])/(2**16))
    cmon = 1.2/(2**15)*float(smon[0])*vrange(vsen_mon)
    cmon *= np.exp(np.pi*2j*float(smon[1])/(2**16))
    crat = cint/cmon
    lcint.append(cint)
    lcmon.append(cmon)
    lcrat.append(crat)
mcint = np.average(lcint)
mcmon = np.average(lcmon)
mcrat = np.average(lcrat)
csvOut.writerow(colnames)
row = ['%.6g' % mcint.real, ' %.6g' % mcint.imag]
row += [' %.6g' % mcmon.real, ' %.6g' % mcmon.imag]
row += [' %.6g' % mcrat.real, ' %.6g' % mcrat.imag]
csvOut.writerow(row+pad_dict)
```

vpp43 モジュールには，GP-IB インターフェースの初期化 (IFC, interface clear) の関数も用意されており，制御中断後の測定器との通信回復に役立つこともあります．以下にその利用例を挙げています．

ifc.py

```
#!/usr/bin/env python
""" Test for gpib_ifc (interface clear).
tested with pyvisa-1.3 for windows and Python-2.6.2. """
import visa as vi
from pyvisa import vpp43
#from visa import *
```

```
intfc = vpp43.open(vi.resource_manager.session, 'GPIB::INTFC')
#interf.send_ifc()
#vi.Gpib().send_ifc()
#Gpib().send_ifc()
vpp43.gpib_send_ifc(intfc)
```

4 ナイフエッジによるビームプロファイル測定の 1D スキャン

scan_horiz.py (horizontal scan の意) は，前節のスクリプトにステージ移動のループを組み合わせ，一次元の空間スキャンをしてデータを CSV ファイルに保存するスクリプトの例です．2 本の自動ステージヘッドの第 1 軸が水平方向に対応するよう X-Z ステージを設置しています．

scan_horiz.py

```
#!/usr/bin/env python
import os, csv, time
import visa as vi
from pyvisa import vpp43
import Tkinter as tk
import tkFileDialog as tkf
import numpy as np
#import matplotlib.pyplot as plt

colnames = ['#x/mm', '#y/mm', '#int_re', '#int_im', '#mon_re', '#mon_im']
colnames += ['#rat_re', '#rat_im', '', '']
ncol = len(colnames)
pad_data = ['#']*(ncol-2)
pad_dict = ['']*2
intfc = vpp43.open(vi.resource_manager.session, 'GPIB::INTFC')
li56int = vi.instrument('GPIB::2') # LI5640 for integration sphere PD
li56mon = vi.instrument('GPIB::3') # LI5640 for monitor PD
omec4 = vi.instrument('GPIB::8') # Sigma OMEC-4BG stage controller
cmd_trig = chr(0x3f) # unlisten (UNL)
cmd_trig += chr(0x20+0x02) # adding GPIB::2 to the group
cmd_trig += chr(0x20+0x03) # adding GPIB::3 to the group
cmd_trig += chr(0x08) # group execution trigger (GET)
nSamp = 16 # number of samples for averaging at a point
def rec_stat(vi, strGPIB, recName, csvOut):
    vi.write(strGPIB)
    csvOut.writerow(pad_data+[recName, vi.read()])
def wait_move(vi):
    while 1:
        resp = vi.ask('1Q')
        print resp
        if resp.find('B')==-1: # busy if 'B' exists
            break

class App(tk.Frame):
    def __init__(self, master=None):
        tk.Frame.__init__(self, master)
        self.pack()
        self.lfSave = tk.LabelFrame(self, text='filename for save')
        self.lfSave.pack(side=tk.TOP)
        self.entSave = tk.Entry(self.lfSave, width=64)
        self.entSave.pack(side=tk.LEFT)
```

```
        self.entSave.insert(0, os.path.realpath('untitled.csv'))
        self.bSel = tk.Button(self.lfSave, text='select...', command=self.sel)
        self.bSel.pack()
        self.frPar = tk.Frame(self)
        self.frPar.pack()
        self.lfStep = tk.LabelFrame(self.frPar)
        self.lfStep['text'] ='mapping step in um (integer)'
        self.lfStep.pack(side=tk.LEFT)
        self.entStep = tk.Entry(self.lfStep)
        self.entStep.pack()
        self.entStep.insert(0, '50') # in um
        self.entStep.bind('<Key-Return>', self.showLen)
        self.lfSize = tk.LabelFrame(self.frPar)
        self.lfSize['text'] = 'point number for scan (integer)'
        self.lfSize.pack(side=tk.LEFT)
        self.entSize = tk.Entry(self.lfSize)
        self.entSize.pack()
        self.entSize.insert(0, '200')
        self.entSize.bind('<Key-Return>', self.showLen)
        self.lfLen = tk.LabelFrame(self.frPar)
        self.lfLen['text'] = 'length of square side in mm'
        self.lfLen.pack()
        self.labLen = tk.Label(self.lfLen)
        self.labLen.pack()
        self.frProc = tk.Frame(self)
        self.frProc.pack()
        self.bStart = tk.Button(self.frProc, text='start')
        self.bStart['command'] = self.new_scan
        self.bStart.pack(side=tk.LEFT)
        self.lfProgress = tk.LabelFrame(self.frProc, text='progress')
        self.lfProgress.pack(side=tk.LEFT)
        self.count = tk.Label(self.lfProgress)
        self.count.pack()
        self.bExit = tk.Button(self.frProc, text="exit")
        self.bExit['command'] = self.quit
        self.bExit.pack(side=tk.LEFT)
    def sel(self):
        fnameout = tkf.asksaveasfilename()
        if fnameout:
            self.entSave.delete(0, tk.END)
            self.entSave.insert(0, fnameout)
    def showLen(self, event):
        self.labLen['text'] = \
            float(self.entStep.get())*float(self.entSize.get())*1e-3
    def new_scan(self):
        self.fOut = open(self.entSave.get(), 'wb')
        self.csvOut = csv.writer(self.fOut)
        self.csvOut.writerow(pad_data+['creator.name', 'scan_horiz.py'])
        rec_stat(li56int, '*IDN?', 'li56int.idn', self.csvOut)
        rec_stat(li56int, 'VSEN?', 'li56int.vsen', self.csvOut) # voltage sensitivity
        rec_stat(li56int, 'TCON?', 'li56int.tcon', self.csvOut) # time constant
        rec_stat(li56int, 'SLOP?', 'li56int.slop', self.csvOut) # filter slope
        rec_stat(li56mon, '*IDN?', 'li56mon.idn', self.csvOut)
        rec_stat(li56mon, 'VSEN?', 'li56mon.vsen', self.csvOut)
        rec_stat(li56mon, 'TCON?', 'li56mon.tcon', self.csvOut)
        rec_stat(li56mon, 'SLOP?', 'li56mon.slop', self.csvOut)
        rec_stat(omec4, '?:N', 'omec.name', self.csvOut)
        rec_stat(omec4, '?:V', 'omec.ver', self.csvOut)
```

```
        li_setting = 'DTYP 2;' # data memory type => DATA1,DATA2
        li_setting += 'DSIZ 0;' # data memory size => 2048 records
        li_setting += 'DNUM 0;' # data memory number => 0 (first block)
        li_setting += 'DSMP 9' # data memory sampling period => 20msec
        li56int.write(li_setting)
        li56mon.write(li_setting)
        self.csvOut.writerow(colnames)
        self.lStep = int(self.entStep.get())
        self.nSide = int(self.entSize.get())
        self.nTot = self.nSide
        self.kx = -self.nSide/2
        self.ky = 0
        self.k = 0
        #self.kxmax, self.kxmin, self.kymax, self.kymin = 0, 0, 0, 0
        self.dx = 1
        self.dy = 0
        self.count['text'] = '%d/%d' % (self.k, self.nTot)
        print 'k: %d, kx: %d' % (self.k, self.kx)
        #plt.ion()
        self.after(1, self.meas_and_move)
    def meas_and_move(self):
        li56int.write('STRT')
        li56mon.write('STRT') # start (waiting for a trigger)
        vpp43.gpib_command(intfc, cmd_trig)
        time.sleep(nSamp*0.020+0.1)
        li56int.write('STOP')
        li56mon.write('STOP')
        time.sleep(0.1)
        li56int.write('DASC? 1,%d' % nSamp)
        li56mon.write('DASC? 1,%d' % nSamp) # ASCII data transfer
        lcint, lcmon, lcrat = [], [], []
        for i in range(nSamp):
            sint = li56int.read().split(',')
            smon = li56mon.read().split(',')
            cint = 1.2/(2**15)*float(sint[0])
            cint *= np.exp(np.pi*2j*float(sint[1])/(2**16))
            cmon = 1.2/(2**15)*float(smon[0])
            cmon *= np.exp(np.pi*2j*float(smon[1])/(2**16))
            crat = cint/cmon
            lcint.append(cint)
            lcmon.append(cmon)
            lcrat.append(crat)
        mcint = np.average(lcint)
        mcmon = np.average(lcmon)
        mcrat = np.average(lcrat)
        row = ['%.6g' % (self.kx*self.lStep*1e-3)]
        row += [' %.6g' % (self.ky*self.lStep*1e-3)]
        row += [' %.6g' % mcint.real, ' %.6g' % mcint.imag]
        row += [' %.6g' % mcmon.real, ' %.6g' % mcmon.imag]
        row += [' %.6g' % mcrat.real, ' %.6g' % mcrat.imag]
        self.csvOut.writerow(row+pad_dict)
        self.fOut.flush()
        self.count['text'] = '%d/%d' % (1+self.k, self.nTot)
        self.k += 1
        if self.k==1:
            print 'move to beginning point...'
            omec4.write('1M1:%+d' % (self.lStep*self.kx))
            wait_move(omec4)
```

```
            print 'done.'
        if self.k==self.nTot:
            self.fOut.close()
            wait_move(omec4)
            print 'move to initial point...'
            omec4.write('1A3:+0,+0')
            wait_move(omec4)
            print 'done.'
            return
        if self.dx!=0:
            wait_move(omec4)
            omec4.write('1M1:%+d' % (self.lStep*self.dx))
            print 'x===> %d' % (self.kx*self.lStep)
            self.kx += self.dx
        if self.dy!=0:
            wait_move(omec4)
            omec4.write('1M2:%+d' % (self.lStep*self.dy))
            print 'y===> %d' % (self.ky*self.lStep)
            self.ky += self.dy
        self.after(1, self.meas_and_move)

root = App()
root.master.title('scan_horiz')
root.mainloop()
```

第一軸を動かさず第二軸でスキャンするスクリプト scan_vert.py (vertical scan の意) も用意しています．scan_horiz.py, scan_vert.py ともに，らせん状スキャンのプログラムを改変して作成したため，ループ構造が無用に複雑になっており，簡略化の余地があります．

5 散乱マップ作成用 2D スキャン (未完成)

自動ステージの現在位置を中心に，正方形領域をらせん状にスキャンして計測を行うスクリプトの例です．

scan_spiral.py

```
#!/usr/bin/env python
import os, csv, time
import visa as vi
from pyvisa import vpp43
import Tkinter as tk
import tkFileDialog as tkf
import numpy as np
import matplotlib.pyplot as plt

colnames = ['#x/mm', '#y/mm', '#int_re', '#int_im', '#mon_re', '#mon_im']
colnames += ['#rat_re', '#rat_im', '', '']
ncol = len(colnames)
pad_data = ['#']*(ncol-2)
pad_dict = ['']*2
intfc = vpp43.open(vi.resource_manager.session, 'GPIB::INTFC')
li56int = vi.instrument('GPIB::2') # LI5640 for integration sphere PD
li56mon = vi.instrument('GPIB::3') # LI5640 for monitor PD
omec4 = vi.instrument('GPIB::8') # Sigma OMEC-4BG stage controller
cmd_trig = chr(0x3f) # unlisten (UNL)
cmd_trig += chr(0x20+0x02) # adding GPIB::2 to the group
cmd_trig += chr(0x20+0x03) # adding GPIB::3 to the group
```

```
cmd_trig += chr(0x08) # group execution trigger (GET)
nSamp = 16 # number of samples for averaging at a point
def rec_stat(vi, strGPIB, recName, csvOut):
    vi.write(strGPIB)
    csvOut.writerow(pad_data+[recName, vi.read()])
def wait_move(vi):
    while 1:
        resp = vi.ask('1Q')
        print resp
        if resp.find('B')==-1: # busy if 'B' exists
            break
def show_map(fname, lStep, nSide):
    fIn = open(fname, 'rb')
    csvIn = csv.reader(fIn)
    lx, ly, lrreal, lrimag = [], [], [], []
    for row in csvIn:
        if not row[1].find('#')==0:
            lx.append(float(row[0]))
            ly.append(float(row[1]))
            lrreal.append(float(row[6]))
            lrimag.append(float(row[7]))
    fIn.close()
    kmax = round((nSide-1)/2.0)
    akx = np.arange(nSide)-kmax
    aky = np.arange(nSide)+kmax
    mkx, mky = np.meshgrid(akx, aky)
    v = mkx*0.0+1e-9
    for i in range(len(lx)):
        print lx[i], ly[i]
        ix = round(lx[i]/lStep*1e3)+kmax
        iy = round(ly[i]/lStep*1e3)+kmax
        print 'ix,iy: ', ix, iy
        v[ix, iy] = np.log(lrreal[i]**2+lrimag[i]**2)
    plt.imshow(v, cmap=plt.cm.spectral)
    #plt.colorbar()
    plt.draw()

class App(tk.Frame):
    def __init__(self, master=None):
        tk.Frame.__init__(self, master)
        self.pack()
        self.lfSave = tk.LabelFrame(self, text='filename for save')
        self.lfSave.pack(side=tk.TOP)
        self.entSave = tk.Entry(self.lfSave, width=64)
        self.entSave.pack(side=tk.LEFT)
        self.entSave.insert(0, os.path.realpath('untitled.csv'))
        self.bSel = tk.Button(self.lfSave, text='select...', command=self.sel)
        self.bSel.pack()
        self.frPar = tk.Frame(self)
        self.frPar.pack()
        self.lfStep = tk.LabelFrame(self.frPar)
        self.lfStep['text'] ='mapping step in um (integer)'
        self.lfStep.pack(side=tk.LEFT)
        self.entStep = tk.Entry(self.lfStep)
        self.entStep.pack()
        self.entStep.insert(0, '200') # in um
        self.entStep.bind('<Key-Return>', self.showLen)
        self.lfSize = tk.LabelFrame(self.frPar)
```

```
        self.lfSize['text'] = 'point number for square side (integer)'
        self.lfSize.pack(side=tk.LEFT)
        self.entSize = tk.Entry(self.lfSize)
        self.entSize.pack()
        self.entSize.insert(0, '9')
        self.entSize.bind('<Key-Return>', self.showLen)
        self.lfLen = tk.LabelFrame(self.frPar)
        self.lfLen['text'] = 'length of square side in mm'
        self.lfLen.pack()
        self.labLen = tk.Label(self.lfLen)
        self.labLen.pack()
        self.frProc = tk.Frame(self)
        self.frProc.pack()
        self.bStart = tk.Button(self.frProc, text='start')
        self.bStart['command'] = self.new_scan
        self.bStart.pack(side=tk.LEFT)
        self.lfProgress = tk.LabelFrame(self.frProc, text='progress')
        self.lfProgress.pack(side=tk.LEFT)
        self.count = tk.Label(self.lfProgress)
        self.count.pack()
        self.bExit = tk.Button(self.frProc, text="exit")
        self.bExit['command'] = self.quit
        self.bExit.pack(side=tk.LEFT)
    def sel(self):
        fnameout = tkf.asksaveasfilename()
        if fnameout:
            self.entSave.delete(0, tk.END)
            self.entSave.insert(0, fnameout)
    def showLen(self, event):
        self.labLen['text'] = \
            float(self.entStep.get())*float(self.entSize.get())*1e-3
    def new_scan(self):
        self.fOut = open(self.entSave.get(), 'wb')
        self.csvOut = csv.writer(self.fOut)
        self.csvOut.writerow(pad_data+['creator.name', 'scan_spiral.py'])
        rec_stat(li56int, '*IDN?', 'li56int.idn', self.csvOut)
        rec_stat(li56int, 'VSEN?', 'li56int.vsen', self.csvOut) # voltage sensitivity
        rec_stat(li56int, 'TCON?', 'li56int.tcon', self.csvOut) # time constant
        rec_stat(li56int, 'SLOP?', 'li56int.slop', self.csvOut) # filter slope
        rec_stat(li56mon, '*IDN?', 'li56mon.idn', self.csvOut)
        rec_stat(li56mon, 'VSEN?', 'li56mon.vsen', self.csvOut)
        rec_stat(li56mon, 'TCON?', 'li56mon.tcon', self.csvOut)
        rec_stat(li56mon, 'SLOP?', 'li56mon.slop', self.csvOut)
        rec_stat(omec4, '?:N', 'omec.name', self.csvOut)
        rec_stat(omec4, '?:V', 'omec.ver', self.csvOut)
        li_setting = 'DTYP 2;' # data memory type => DATA1,DATA2
        li_setting += 'DSIZ 0;' # data memory size => 2048 records
        li_setting += 'DNUM 0;' # data memory number => 0 (first block)
        li_setting += 'DSMP 9' # data memory sampling period => 20msec
        li56int.write(li_setting)
        li56mon.write(li_setting)
        self.csvOut.writerow(colnames)
        self.lStep = int(self.entStep.get())
        self.nSide = int(self.entSize.get())
        self.nTot = self.nSide**2
        self.kx = 0
        self.ky = 0
        self.k = 0
```

```
        self.kxmax, self.kxmin, self.kymax, self.kymin = 0, 0, 0, 0
        self.dx = 1
        self.dy = 0
        self.count['text'] = '%d/%d' % (self.k, self.nTot)
        plt.ion()
        self.after(1, self.meas_and_move)
    def meas_and_move(self):
        li56int.write('STRT')
        li56mon.write('STRT') # start (waiting for a trigger)
        vpp43.gpib_command(intfc, cmd_trig)
        time.sleep(nSamp*0.020+0.1)
        li56int.write('STOP')
        li56mon.write('STOP')
        time.sleep(0.1)
        li56int.write('DASC? 1,%d' % nSamp)
        li56mon.write('DASC? 1,%d' % nSamp) # ASCII data transfer
        lcint, lcmon, lcrat = [], [], []
        for i in range(nSamp):
            sint = li56int.read().split(',')
            smon = li56mon.read().split(',')
            cint = 1.2/(2**15)*float(sint[0])
            cint *= np.exp(np.pi*2j*float(sint[1])/(2**16))
            cmon = 1.2/(2**15)*float(smon[0])
            cmon *= np.exp(np.pi*2j*float(smon[1])/(2**16))
            crat = cint/cmon
            lcint.append(cint)
            lcmon.append(cmon)
            lcrat.append(crat)
        mcint = np.average(lcint)
        mcmon = np.average(lcmon)
        mcrat = np.average(lcrat)
        row = ['%.6g' % (self.kx*self.lStep*1e-3)]
        row += [' %.6g' % (self.ky*self.lStep*1e-3)]
        row += [' %.6g' % mcint.real, ' %.6g' % mcint.imag]
        row += [' %.6g' % mcmon.real, ' %.6g' % mcmon.imag]
        row += [' %.6g' % mcrat.real, ' %.6g' % mcrat.imag]
        self.csvOut.writerow(row+pad_dict)
        self.fOut.flush()
        self.count['text'] = '%d/%d' % (1+self.k, self.nTot)
        show_map(self.entSave.get(), self.lStep, self.nSide)
        self.k += 1
        if self.k==self.nTot:
            self.fOut.close()
            wait_move(omec4)
            print 'move to initial point...'
            omec4.write('1A3:+0,+0')
            wait_move(omec4)
            print 'done.'
            return
        if self.dx!=0:
            wait_move(omec4)
            omec4.write('1M1:%+d' % (self.lStep*self.dx))
            print 'x===> %d' % (self.kx*self.lStep)
            self.kx += self.dx
        if self.dy!=0:
            wait_move(omec4)
            omec4.write('1M2:%+d' % (self.lStep*self.dy))
            print 'y===> %d' % (self.ky*self.lStep)
```

```
            self.ky += self.dy
        if self.kx>self.kxmax:
            self.kxmax = self.kx
            (self.dx, self.dy) = (0, 1)
        if self.ky>self.kymax:
            self.kymax = self.ky
            (self.dx, self.dy) = (-1, 0)
        if self.kx<self.kxmin:
            self.kxmin = self.kx
            (self.dx, self.dy) = (0, -1)
        if self.ky<self.kymin:
            self.kymin = self.ky
            (self.dx, self.dy) = (1, 0)
        self.after(1, self.meas_and_move)

root = App()
root.master.title('scan_spiral')
root.mainloop()
```

上記スクリプトは一応動作しデータが取得できますが，ロックインアンプから新しいデータが送られてくるたびに出力ファイルストリームを閉じて読み込みなおし，matplotlib で画面にマッピング像を描画するという構造をとっているのがあだとなって，データ取得速度がスローダウンしてゆくため，実用に耐えません．メモリ開放にからんだ，何らかの修正が必要と思われます．

改版日/担当者	変更要点
2011-07-21/森脇	初稿