```
# Programmer: Sergio Proa Coronado
# Version: 2
#Elapsed time to finish iterations: --:--:--
# JPK Script
from __future__ import division
checkVersion('SPM', 4, 2, 61);
import math
import time
start_time=time.time()

#New data type declaration
class Point(object):
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

# Function to calculate angle
def calculateAngle(A = [], B = []):
    """ Given two points (x,y) an angle is calculated
    """
    Time_angleS = time.time()
    if A[0] != B[0]:
        print 'angle calculated: ' + str(math.atan((B[1] -
         A[1])/(B[0]-A[0])))
        Time_angleF = time.time()
        print'Time for calculate angle: ' + str(Time_angleF - Time_angleS)
        return math.atan((B[1] - A[1])/(B[0]-A[0]))


    else:
        Time_angleF = time.time()
        print'Time for calculate angle: ' + str(Time_angleF - Time_angleS)

        return 0

# Function to determine plane equations
def planeEq(Ph, Qh, Rh, P = [], Q = [],R = []):
    """Given three points (coordinates x, y) and its heights calculates the
     plane equation
    """
    coef = []
    coef.append((R.y - P.y) * (Qh - Ph) - (Q.y - P.y) * (Rh - Ph))
    coef.append((Rh - Ph) * (Q.x - P.x) - (Qh - Ph) * (R.x - P.x))
    coef.append((R.x - P.x) * (Q.y - P.y) - (Q.x - P.x) * (R.y - P.y))
    coef.append(-(coef[0] * P.x + coef[1] * P.y + coef[2] * Ph))
    #print 'Zero plane equation: ' + str(coef[0]) + ' X +' + str(coef[1]) +
     ' Y +' + str(coef[2]) + ' Z +' + str(coef[3]) + ' = 0'
    return coef

#Function to recalculate points 1, 2 and 3.
def calculatePoints(ws, a, fN, P = []):
    """ Given the well size and a list variable which stores the average
     central coordinates of a well,
    recalculates these coordinates to be at the center of the well
    """
```

```
Time_centerS = time.time()
dateTemp = time.asctime()
#-------------Variables for global zero plane calculation
global TP
global TH
global Tsd
PC = []
PH = []

#-------------------------------------------------
#Variables to determine weights
zeroPlane = []
tHeights = []
weights = []
distances = []
#-----Variables for the grid
FastLength = ws + 3e-06
SlowLength = ws + 4e-06
XGridCenter = P[0]
YGridCenter = P[1]
NumFastPoints = 5
NumSlowPoints = 5
#--------------------------
#---------------Variables for local zero plane points determination
temp = 0
numX = 0
numY = 0
den = 0
tempX = 0
tempY = 0
tempDen = 0
invtempDen = 0
#-------------------------------------------------------------
if (P[0] + (FastLength / 2)) > 50e-6:
    extraDistance = ((P[0] + FastLength / 2) - 50e-6) + 1e-6
    P[0] -= extraDistance
if (P[1] + (SlowLength / 2)) > 50e-6:
    extraDistance = ((P[1] + SlowLength / 2) - 50e-6) + 1e-6
    P[1] -= extraDistance
ForceSpectroscopy.setPosition(P[0], P[1])
#Saving an image of the initial position of the tip
Snapshooter.saveOpticalSnapshot(path+"initialP-"+ str(dateTemp) +
 ".png")
ForceSpectroscopy.setGridPattern(FastLength, SlowLength, XGridCenter,
 YGridCenter, NumFastPoints, NumSlowPoints, a)
#ForceSpectroscopy.setGridPattern(FastLength, SlowLength, XGridCenter,
 YGridCenter, NumFastPoints, NumSlowPoints, a)
Scanner.approach()
# Iterating through the grid
for h in range(NumFastPoints * NumSlowPoints):
    Scanner.retractPiezo()
    ForceSpectroscopy.moveToForcePositionIndex(h)
    Scanner.approachPiezo()
    # Storing grid coordinates
```

```
    PC.append(Point(ForceSpectroscopy.getForcePosition(h).getX(),
     ForceSpectroscopy.getForcePosition(h).getY()))
    # Storing grid coordinates heights
    PH.append(Scanner.getCurrentHeight())

    temp += Scanner.getCurrentHeight()

#Calculating  average coordinate for the global zero plane
for h in range(5):
    tempX += PH[h] * PC[h].x
    tempY += PH[h] * PC[h].y
    tempDen += PH[h]
invtempDen = 1/tempDen
TP = Point(tempX * invtempDen, tempY * invtempDen)


# zero plane calculation
zeroPlane = planeEq(PH[0], PH[4], PH[24], PC[0], PC[4], PC[24])
logFile = open(fN, "a")
logFile.write("Distances to the zero plane:\n")
logFile.close()
# Storing the heigh for the global zero plane
ForceSpectroscopy.setPosition(TP.x, TP.y)
TH = Scanner.getCurrentHeight()
#Assigning weights
for h in range(len(PC)):
    invden = 1 / math.sqrt(zeroPlane[0] ** 2 + zeroPlane[1] ** 2 +
     zeroPlane[2] ** 2)
    # Formula taken from:
     http://mathworld.wolfram.com/Point-PlaneDistance.html
    d=(zeroPlane[0] * PC[h].x + zeroPlane[1] * PC[h].y +zeroPlane[2] *
     PH[h] + zeroPlane[3]) * invden
    distances.append(d)
    print str(h) + ': ' + str(d)
# Storing the heights to calculate the zero distance
tDistances = distances[:5]
tDistances[6:6] = distances[20:25]
# Determination of zero distance
zDmax = max(tDistances)
zDmin = min(tDistances)
for h in range(len(PC)):
    if distances[h] >= (5 * zDmax):
        weights.append(Point(h, 5)) # Using class Point we store index
         (x position) and weight (y position) for that coordinate
    elif distances[h] >= (4 * zDmax):
        weights.append(Point(h, 4))
    elif distances[h] >= (3 * zDmax):
        weights.append(Point(h, 3))
    elif distances[h] >= (2 * zDmax):
        weights.append(Point(h, 2))
    elif distances[h] >= zDmax:
        weights.append(Point(h, 1))
    elif distances[h] <= (5 * zDmin):
        weights.append(Point(h, -5))
    elif distances[h] <= (4 * zDmin):
```

```
            weights.append(Point(h, -4))
        elif distances[h] <= (3 * zDmin):
            weights.append(Point(h, -3))
        elif distances[h] <= (2 * zDmin):
            weights.append(Point(h, -2))
        elif distances[h] < (zDmin):
            weights.append(Point(h, -1))
        else:
            weights.append(Point(h, 0))
    # Discarding points which weight = 0
    weights = [value for value in weights if value.y != 0]

    #Select the apropiate point to be the new center
    #minimum value
    mi = min([value.y for value in weights])

    #Obtaining the heightes weight
    ma = max([value.y for value in weights])
    #Only the maximum heights are considered
    weights = [value for value in weights if value.y == ma]
    for h in range(len(weights)):
        logFile = open(fN, "a")
        logFile.write(str(weights[h].x) + ', ' + str(weights[h].y) + "\n")
        logFile.close()
        print str(weights[h].x) + ', ' + str(weights[h].y)
    #Calculating new central point
    if len(weights) > 1:
        for h in range(len(weights)):
            numX += distances[weights[h].x] * PC[weights[h].x].x
            numY += distances[weights[h].x] * PC[weights[h].x].y
            den += distances[weights[h].x]
        invden = 1/den
        P[0] = numX * invden    # PX recalculated
        P[1] = numY * invden    # PY recalculated
    else:
        P[0] = PC[weights[0].x].x # PX recalculated
        P[1] = PC[weights[0].x].y # PY recalculated
    Scanner.retractPiezo()
    Time_centerF = time.time()
    logFile = open(fN, "a")
    logFile.write('Time for centering: ' + str(Time_centerF - Time_centerS)
     + "\n")
    logFile.close()
    print'Time for centering: ' + str(Time_centerF - Time_centerS)

#****************************************Inputs
 block*************************************************************

#-------------The working area is defined as 100 x 100
 micrometers----------------------------------
#Points 1 and 2, take this coordinates from the center of each well, MANUAL
 INPUT

P1=[42.893e-6 , 36.031e-6]
```

```
P2=[44.425e-6 , -36.655e-6]


#-------------Pitch------------
pitch= 10.5e-6

#Well dimensions assuming square wells
Ws=4.5e-6

#Path for saving directory
path="~/jpkdata/Automatip/20190307/"

# Pattern area
totalArea = 300e-6

# ForceScan matrix for the wells
numScans=[3,3]

#*****************************************************************************
 ***************************


#Temporal variables for global zero plane
TP = Point(0,0)
#temporal piezo hight for the zero plane
TH = 0
#temporal zero plane error
Tsd = 0
# Creating the log file to store the messages displayed on screen
#You can find this log file in the Automatip folder
now = time.asctime(time.localtime(time.time()))
fileName = "LogFile-" + str(now) + ".txt"
logFile = open(fileName, "w")
logFile.close()
#used for the calculation of motor stage coordinates
g = 0
# Temporal variable for timer
parcialTimeMS = 0
#Calculating the angle, two points distance equation
angle = math.fabs(calculateAngle(P1, P2))
logFile = open(fileName, "a")
logFile.write("angle calculated: " + str(angle) + "\n")
logFile.close()
print 'Angle calculated: ' + str(angle)
#Defining distance in order to know the number of wells
da = math.sqrt((P2[0]-P1[0])**2 + (P2[1]-P1[1])**2)
nFP = int(da/pitch+0.4)
nSP = int(da/pitch+0.4)
logFile = open(fileName, "a")
logFile.write(str(nFP) + ' x ' + str(nSP) + "\n")
```

```
logFile.close()
print str(nFP)+' x '+str(nSP)
if P1[0] > P2[0]:
    option = 0
else:
    option = 1
logFile = open(fileName, "a")
logFile.write('option = ' + str(option) + "\n")
logFile.close()
print'option = ' + str(option)
#variables needed for loop
i=0
j=-1
e=0

#path2 = "~/jpkdata/forceScans/20180725/"
#Redefining P1
calculatePoints(Ws,angle, fileName, P1)
logFile = open(fileName, "a")
logFile.write('P1: ' + str(P1[0]) + ', ' + str(P1[1]) + "\n")
logFile.close()
print 'P1: ' + str(P1[0]) + ', ' + str(P1[1])
ForceSpectroscopy.setPosition(P1[0], P1[1])
Scanner.approachPiezo()
Snapshooter.saveOpticalSnapshot(path+"P1-Corrected.png")
Scanner.retractPiezo()
zP1 = TP
zH1 = TH
#Tsd1 = Tsd

#Redefining P2
calculatePoints(Ws,angle, fileName, P2)
logFile = open(fileName, "a")
logFile.write('P2: ' + str(P2[0]) + ', ' + str(P2[1]) + "\n")
logFile.close()
print 'P2: ' + str(P2[0]) + ', ' + str(P2[1])
ForceSpectroscopy.setPosition(P2[0], P2[1])
Scanner.approachPiezo()
Snapshooter.saveOpticalSnapshot(path+"Test1-25p-afterP2.png")
Scanner.retractPiezo()
zP2 = TP
zH2 = TH
#Tsd2 = Tsd

#Redefining P3
#calculatePoints(Ws,angle, P3)
#ForceSpectroscopy.setPosition(P3[0], P3[1])
#Scanner.approachPiezo()
#Snapshooter.saveOpticalSnapshot(path2+"Test1-25p-afterP3.png")
#Scanner.retractPiezo()
#zP3 = TP
#zH3 = TH
#Tsd3 = Tsd

#Global_zeroPlane = planeEq(zH1, zH2, zH3, zP1, zP2, zP3)
```

```
#time.sleep(10)

#Distance for the MotorStage
delta = pitch*nFP
logFile = open(fileName, "a")
logFile.write('Total distance to cover on X axis: ' + str(delta)+' m' +
 "\n")
logFile.close()
print'Total distance to cover on X axis: ' + str(delta)+' m'

# MSstep is the number of motorstage steps
MSstep = int(totalArea / delta)
logFile = open(fileName, "a")
logFile.write('The pattern was divided into: ' + str(MSstep) + " scanning
 areas\n")
logFile.close()
print 'MSstep = ' + str(MSstep)
#Variable WellPositions stores the coordinates for the WellGrid
WellPositions=[]

#Variable MSCoord stores MotorStage coordinates
MSCoord = []

# Create a MSCoord list full of zeros
for f in range(MSstep * MSstep):
    MSCoord.append(Point(0,0))

# Assign the first motor stage coordinate to the first position
MSCoord[0].x = MotorizedStage.getPosition().getX()
MSCoord[0].y = MotorizedStage.getPosition().getY()
#Scanner.approach()
if (MSCoord[0].x > 0):
    # Calculating MotorStage coordinates
    for f in range(1,MSstep):
        MSCoord[f].x = MSCoord[f-1].x - delta * math.cos(angle)
        MSCoord[f].y = MSCoord[f-1].y + delta * math.sin(angle)

    i = (MSstep*2) -1
    for f in range(0, len(MSCoord) - MSstep):
        if i < 1:
            i = (MSstep*2) -1
        MSCoord[f+i].x = MSCoord[f].x - delta * math.sin(angle) - (Ws / 5)
        MSCoord[f+i].y = MSCoord[f].y - delta * math.cos(angle)
        i = i-2
else:
    # Calculating MotorStage coordinates
    for f in range(1,MSstep):
        MSCoord[f].x = MSCoord[f-1].x + delta * math.cos(angle)
        MSCoord[f].y = MSCoord[f-1].y + delta * math.sin(angle)

    i = (MSstep*2) -1
    for f in range(0, len(MSCoord) - MSstep):
        if i < 1:
            i = (MSstep*2) -1
        MSCoord[f+i].x = MSCoord[f].x - delta * math.sin(angle) - (Ws / 5)
```

```
        MSCoord[f+i].y = MSCoord[f].y - delta * math.cos(angle)
        i = i-2
#Iterating through the MotorStage coordinates
logFile = open(fileName, "a")
logFile.write('Going to the motor stage initial coordinate: ' +
 str(MSCoord[0].x) + ' , ' + str(MSCoord[0].y) + "\n")
logFile.close()
print 'Going to the motor stage initial coordinate: '+str(MSCoord[0].x)+' ,
 '+str(MSCoord[0].y)

while g < len(MSCoord):
    initialMStime=time.time()

    #Retract the scanner to avoid damages
    #Scanner.retract()
    Scanner.retractPiezo()

    #Moving the scanner to the initial point
    ForceSpectroscopy.setPosition(P1[0], P1[1])

    if g > 0:
        #Engaging MotorStage
        MotorizedStage.engage()

        #Moving the MotorStage to a particular coordinate
        MotorizedStage.moveToAbsolutePosition(MSCoord[g].x, MSCoord[g].y)
        logFile = open(fileName, "a")
        logFile.write('Motor stage current coordinate '+ str(g)+':
         '+str(MSCoord[g].x)+str(MSCoord[g].y) + "\n")
        logFile.close()
        print 'Motor stage current coordinate '+ str(g)+':
         '+str(MSCoord[g].x)+str(MSCoord[g].y)
        MotorizedStage.disengage()
        calculatePoints(Ws,angle,fileName, P1)
    logFile = open(fileName, "a")
    logFile.write('Calculating well coordinates...' + "\n")
    logFile.close()
    print 'Calculating well coordinates...'
    # Loops in order to store the coordinates;
    for i in range(nFP*nSP):
        WellPositions.append(Point(0,0))

    # Assign the first coordinate (P1) to the first position
    WellPositions[0].x = P1[0]
    WellPositions[0].y = P1[1]

    # Calculating Well positions
    if option == 0:
        for f in range(1, nFP):
            WellPositions[f].x = WellPositions[f-1].x - pitch *
             math.cos(angle)
            WellPositions[f].y = WellPositions[f-1].y - pitch *
             math.sin(angle)

        i = (nFP*2) -1
```

```
    for f in range(0, len(WellPositions) - nFP):
        if i < 1:
            i = (nFP*2) -1
        WellPositions[f+i].x = WellPositions[f].x - pitch *
         math.sin(angle)
        WellPositions[f+i].y = WellPositions[f].y + pitch *
         math.cos(angle)
        i = i-2
else:
    for f in range(1, nFP):
        WellPositions[f].x = WellPositions[f-1].x + pitch *
         math.cos(angle)
        WellPositions[f].y = WellPositions[f-1].y - pitch *
         math.sin(angle)

    i = (nFP*2) -1
    for f in range(0, len(WellPositions) - nFP):
        if i < 1:
            i = (nFP*2) -1
        WellPositions[f+i].x = WellPositions[f].x - pitch *
         math.sin(angle)
        WellPositions[f+i].y = WellPositions[f].y + pitch *
         math.cos(angle)
        i = i-2
# activate spectroscopy gui mode (show the force scan oscilloscope)
ForceSpectroscopy.activateGUIMode()

# activate autosaving
ForceSpectroscopy.setAutosave(1)

# Variable for date
date = time.asctime()

#Path to store optical images
pathR = path+"/Area"+str(g) + '--' + str(date)

#Iterating through the Wellgrid
for a in range(nSP):
    for b in range(nFP):
        #Timer
        initialParcialWtime=time.time()

        #Compute index position in WellGrid and use it to move there
        index=a*nFP+b

        #Move to the center of each square
        ForceSpectroscopy.setPosition(WellPositions[index].x,
         WellPositions[index].y)

        # Print the WellGrid current position
        logFile = open(fileName, "a")
        logFile.write('Current Well: '+ str(index) + "\n")
        logFile.close()
        print 'Current position on Wellgrid: '+ str(index)
```

```
        # set the directory where the force scans should be stored
         HAVE FUN
        date = time.asctime()

         ForceSpectroscopy
         .setOutputDirectory(pathR+"/WellGrid"+str(index) + '--' +
         str(date))

        #Cleaning the position list
        ForceSpectroscopy.clearPositions()

        #Defining the measurement grid
        FastLength=Ws - 1.5e-6
        SlowLength=Ws - 1.5e-6
        XGridCenter=WellPositions[index].x
        YGridCenter=WellPositions[index].y
        NumFastPoints=numScans[0]
        NumSlowPoints=numScans[1]
        try:
            ForceSpectroscopy.setGridPattern(FastLength, SlowLength,
             XGridCenter, YGridCenter, NumFastPoints, NumSlowPoints,
             angle)
            Scanner.approachPiezo()
            #Iterating through the measurement grid
            ForceSpectroscopy.startScansPerPosition(1)
            Scanner.retractPiezo()
        except:
            print 'This well has some points outside the scanning
             area\nProceeding to the next one'
            logFile = open(fileName, "a")
            logFile.write("This well has some points outside the
             scanning area\nProceeding to the next one\n")
            logFile.close()
        #Timers to show the time consumed on 100x100 um area
        endingParcialWtime=time.time()
        partialTime=endingParcialWtime - initialParcialWtime
        logFile = open(fileName, "a")
        logFile.write('Time for one well: ' + str(partialTime) + "\n")
        logFile.close()
        print'Time for one well: ' + str(partialTime)
        timeLeft = (partialTime * (len(WellPositions))) - ( partialTime
         * (index + 1))
        logFile = open(fileName, "a")
        logFile.write('Time left for this area: ' + str(timeLeft/60) +
         "\n")
        logFile.close()
        print 'Time left for this area: ' + str(timeLeft/60) + ' min'

# Returning the scanner to the first well position
#Scanner.retractPiezo()
ForceSpectroscopy.setPosition(WellPositions[0].x, WellPositions[0].y)

#Saving optical images
Snapshooter.saveOpticalSnapshot(path+"image"+ str(date)+str(g))
g += 1
```

```
    # This gives info about how much time will take for the remaining areas
    endingMStime = time.time()
    parcialTimeMS = endingMStime-initialMStime

#Retract the piezo after finish
Scanner.retract()
MotorizedStage.moveToAbsolutePosition(MSCoord[0].x, MSCoord[0].y)
end_time=time.time()
print 'Elapsed time: '+ str((end_time-start_time)/60)+' min'
```