```
import time
import picamera
import io
import os
import RPi.GPIO as GPIO
from datetime import datetime
from PIL import Image

camera = picamera.PiCamera()

#Change difference and pixels to adjust sensitivity. Difference is
#a threshold that the pixel must exceed to be counted as a changed pixel
#Pixel says how many pixels must change to count as a detection
difference = 50 #norm is 50
pixels = 100    #norm is 100

width = 1280
height = 960

count = 0
neb = 13
shutter = 15
galvo = 17

settleTime = 40 #Set to 20 for actual experiments
puffNeeded = 1  #1 means that we need to puff
desiredNumberOfTests = 3 #Change this depending on how many tests you want the program to run
testNum = 0
```

```
startTime = 0  #Used for determining the length of the hold
endTime = 0    #Used for determining the length of the hold
arrayOfDurations = [] # Used to store all the durations to get statistical values after the tests are run
log = ''
GPIO.setmode(GPIO.BOARD)
GPIO.setup(neb, GPIO.OUT)
GPIO.setup(shutter, GPIO.OUT)



def compare():
        camera.resolution = (100, 75)
        stream = io.BytesIO()
        camera.capture(stream, format = 'bmp')
        stream.seek(0)
        im = Image.open(stream)
        buffer = im.load()
        stream.close()
        return im, buffer

def newimage(width, height):
        time = datetime.now()
        filename = "motion-%04d%02d%02d-%02d%02d%02d.jpg" %(time.year, time.month, time.day,
time.hour, time.minute, time.second)
        camera.resolution = (width, height)
        camera.capture(filename)
        print "Captured %s" % filename

def puff():
        global neb
```

```
        GPIO.output(neb,0)
        time.sleep(1)
        GPIO.output(neb,1)
        return

def takePic():
        global shutter
        GPIO.output(shutter, 1)
        time.sleep(2)
        GPIO.output(shutter, 0)
        return

def startGalvo():
        GPIO.output(galvo, 1)
        time.sleep(1)
        GPIO.output(galvo, 0)
        return

def evaluateTest():
        global log
        global arrayOfDurations
        duration = endTime - startTime
        seconds = duration % 60
        timeInMinutesMinusSeconds = (duration - seconds)/60
        minutes = timeInMinutesMinusSeconds % 60
        hours = (timeInMinutesMinusSeconds-minutes)/60
        arrayOfDurations.append(duration)
        log = log + 'Test #' + str(testNum) + ' ran for ' + str(hours) + ' hours, ' + str(minutes) + ' minutes
and ' + str(round(seconds,0)) + ' seconds \n'
```

```
        return


def getStats():
        global arrayOfDurations
        global log
        maxValue = round(max(arrayOfDurations), 2)
        minValue = round(min(arrayOfDurations), 2)
        meanValue = round(sum(arrayOfDurations)/len(arrayOfDurations), 2)
        statsMessage = str(desiredNumberOfTests) + ' tests have been completed.  The longest was ' + str(maxValue) + ' seconds and the shortest was ' + str(minValue) + ' seconds with an average time of ' + str(meanValue) + ' seconds. \n'
        finalMessage = '\n \nThe test is done. \n' + statsMessage + log
        print(finalMessage)
        return


image1, buffer1 = compare() #Takes a control image to compare against following images.


#timestamp = time.time()
#print 'Do it now!'        #Agitate particles now to allow time for settling. Move to while loop later.
#puff()
#time.sleep(20)
#print 'Times up'          #differences detected now will be counted as a particle trap.


while testNum < desiredNumberOfTests:
        #time.sleep(5)
        ######################
        #    Nebulize
        #           Wait 5-10 s
        #################
```

```
        print 'Start detection'     #differences detected now will be counted as a particle trap.
        if puffNeeded == 1:
                puff()
                print 'Settling'
                time.sleep(40)
                print 'Detecting'
                puffNeeded = 0

        image2, buffer2 = compare() #takes second image to compare against the control.
        changedpixels = 0                       #keeps track of how many pixels have changed beyond the allowed threshold.
        for x in xrange(0, 100):
                for y in xrange(0,75):
                        pixdiff = abs(buffer1[x,y][1] - buffer2[x,y][1]) #The [1] signifies the green pixel only is being compared since it is the most intense
                        if pixdiff > difference:
                                changedpixels += 1

        if changedpixels > pixels:          #Change code inside statement to change what program does once it detects trapping
                print 'Particle Trapped'
                startTime = time.time()
                newimage(width, height)
                # start recording
                takePic()

                # start drawing
                #startGalvo()
```

```
        image3, buffer3 = compare()
        changedpixels = 0                       #keeps track of how many pixels have changed beyond the allowed threshold.
        for x in xrange(0, 100):
                for y in xrange(0,75):
                        pixdiff = abs(buffer1[x,y][1] - buffer3[x,y][1]) #The [1] signifies the green pixel only is being compared since it is the most intense
                        if pixdiff > difference:
                                changedpixels += 1

        while changedpixels > pixels:
                image3, buffer3 = compare()
                changedpixels = 0                       #keeps track of how many pixels have changed beyond the allowed threshold.
                for x in xrange(0, 100):
                        for y in xrange(0,75):
                                pixdiff = abs(buffer1[x,y][1] - buffer3[x,y][1]) #The [1] signifies the green pixel only is being compared since it is the most intense
                                if pixdiff > difference:
                                        changedpixels += 1

        endTime = time.time()   #takes a picture if enough pixels have changed to be counted
        evaluateTest()
        testNum = testNum + 1
        count += 1
        puffNeeded = 1          #Used to get the next particle in there
getStats()
GPIO.cleanup()
```