```
#!/usr/bin/env python
#
# Neato XV-11 Scanning LIDAR via Raspberry Pi
#
# Adapted by Noah Z Brock from:
# # XV-11 Lidar Test
# Copyright 2010 Nicolas "Xevel" Saugnier
#
# Licensed under the Apache License, Version 2.0 (the "License");
#You may not use this file except in compliance with the License.
#You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
#Unless required by applicable law or agreed to in writing, software
#distributed under the License is distributed on an "AS IS" BASIS,
#WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#See the License for the specific language governing permissions and
#limitations under the License.

#for communication with the LIDAR: PySerial
# for pwm output from RaspberryPi: wiringpi2

import math, threading, thread, time, sys, traceback, wiringpi2 as wiringpi

#--SETTINGS------------------
low_dist = 150
high_dist = 2975
off_dist = 700
distance_threshold = 15
pwm_high = 1023
pwm_low = 0
last_pwm = 0
reverse_pwm = True
last_minAngle = 0
remove_angle_low = 100
remove_angle_high = 115
remove_angle_range = range(remove_angle_low,remove_angle_high)
remove_angle_distance = 2500
last_speed = 0
increment = 1
show_data = False
show_speed = False
show_angle = False
max_buffer = 1000
ease_up = .001
ease_down = .0005
com_port = "/dev/ttyAMA0"
ser = None
baudrate = 115200
init_level = 0
index = 0
distlist = [i for i in range(360)]
masterlist = []
print "Scanning PiDAR by NZB"
print "Stay out of range for first 10 seconds to acquire room data"
print "press ctrl+c to pause and check data"
#------------------------------

#--WIRINGPI-SETTINGS---------
wiringpi.wiringPiSetup()
wiringpi.pinMode(1,2) #set GPIO pin 18 to pwm mode, this is the LED pin
wiringpi.setMode(0) #set pwm mode to marksspace
wiringpi.pwmWrite(1,0) #write 0 to GPIO pin 18
wiringpi.pwmSetRange(pwm_high+1)
wiringpi.pwmSetClock(200) #200 is default aka smooth
#-----------------------------


def checksum(data):
    #Compute and return the checksum as an int.
    #data -- list of 20 bytes (as ints), in the order they arrived in.

    # group the data by word, little-endian
    data_list = []
    for i in range(10):
        data_list.append( data[2*i] + (data[2*i+1]<<8) )
    # compute the checksum on 32 bits
    chk32 = 0
    for d in data_list:
        chk32 = (chk32 << 1) + d

    # return a value wrapped around on 15bits, and truncated to still fit i
    checksum = (chk32 & 0x7FFF) + ( chk32 >> 15 ) # wrap around to fit into
    checksum = checksum & 0x7FFF # truncate to 15 bits
    return int( checksum )

def compute_speed(data):
    current_speed = float(data[0]|(data[1]<<8)) / 64
    return current_speed

def compute_distance(angle, data):
    global low_dist, high_dist, off_dist, distlist

    d0= data[0]
    d1= data[1]
    d2= data[2]
    d3= data[3]
    dist = d0 | ((d1 & 0x3f) << 8)

    #if distance is new, edit list
    if distlist[angle]!=dist:
        distlist[angle]=dist

    #if distance is less than low distance, make it high distance
    if distlist[angle]<=low_dist:
        distlist[angle]=high_dist

def compute_min(list):
    global low_dist, high_dist

    if min(list)>low_dist and min(list)<off_dist:
        return high_dist
    else:
        return min(list)

def test_int(response):
    try:
        int(response)
        return True
    except:
        return False

def compute_pwm(input_value):
    global distlist,low_dist,high_dist,pwm_low,pwm_high

    pwm=translate(input_value,low_dist,high_dist,pwm_low,pwm_high)
    rev_pwm=abs(pwm-pwm_high)
    if reverse_pwm==True:
        return rev_pwm
    else:
        return pwm

def compare_master():
    global masterlist, distlist, distance_threshold, high_dist
    for i,j in zip(masterlist,distlist):
        if j+distance_threshold < i:
            return True
        else:
            return False

def pwm_led(value):
    global pwm_high, pwm_low

    #keep pwm in range
    if value>pwm_high:
        value=pwm_high
    if value<pwm_low:
        value=pwm_low

    wiringpi.pwmWrite(1,value)

def remove_angle(list,new_element,indices):
    return[new_element if i in indices else e for i, e in enumerate(list)]


def easing(last_value, current_value):
    global increment, ease_up, ease_down

    if current_value==0:
        pwm_led(0)
    else:
        while current_value<last_value:
            pwm_led(last_value)
            last_value -= increment
            if show_data==True: print last_value
            #difference = last_value-current_value
            time.sleep(ease_down)
        while current_value>last_value:
            pwm_led(last_value)
            last_value += increment
            if show_data==True: print last_value
            #difference = current_value-last_value
            time.sleep(ease_up)
    if current_value==last_value:
        pwm_led(current_value)
        return current_value


def translate(x,in_min,in_max,out_min,out_max):
    return (x-in_min)*(out_max-out_min)/(in_max-in_min)+out_min

def read_v_2_4(ser):
    global init_level, angle, index
    global off_dist, low_dist, high_dist, distance_threshold, distlist, rem
    global pwm_high, pwm_low, last_pwm, last_minAngle, last_speed
    global increment, max_buffer, ease_up, ease_down
    global show_data, show_speed, show_angle

    while True:
        try:
            elapsed=time.clock()-start
            time.sleep(0.0001) # do not hog the processor power
            if init_level == 0:
                b = ord(ser.read(1))
                # start byte
                if b == 0xFA :
                    init_level = 1
                else:
                    init_level = 0
                if ser.inWaiting()>max_buffer:
                    ser.flushInput()
            elif init_level == 1:
                # position index
                b = ord(ser.read(1))
                if b >= 0xA0 and b <= 0xF9  :
                    index = b - 0xA0
                    init_level = 2
                elif b != 0xFA:
                    init_level = 0
                    if ser.inWaiting()>max_buffer:
                        ser.flushInput()
            elif init_level == 2:
                # motor speed
                b_speed = [ ord(b) for b in ser.read(2)]

                # distance data
                b_data0 = [ ord(b) for b in ser.read(4)]
                b_data1 = [ ord(b) for b in ser.read(4)]
                b_data2 = [ ord(b) for b in ser.read(4)]
                b_data3 = [ ord(b) for b in ser.read(4)]

                # for the checksum, we need all the data of the packet...
                # this could be collected in a more elegant fashion...
                all_data = [ 0xFA, index+0xA0 ] + b_speed + b_data0 + b_data

                # checksum
                b_checksum = [ ord(b) for b in ser.read(2) ]
                incoming_checksum = int(b_checksum[0]) + (int(b_checksum[1])

                # verify that the received checksum is equal to the one com
                if checksum(all_data) == incoming_checksum:

                    #get motor rpm
                    current_speed = int(compute_speed(b_speed))

                    #toggle print speed
                    if show_speed and current_speed != last_speed:
                        print current_speed, 'RPM'
                        last_speed=current_speed

                    #send all distance data to distance function
                    compute_distance(index * 4 + 0, b_data0)
                    compute_distance(index * 4 + 1, b_data1)
                    compute_distance(index * 4 + 2, b_data2)
                    compute_distance(index * 4 + 3, b_data3)

#                    if elapsed > 10 and elapsed < 11:
#                        masterlist = list(distlist)
#                        high_dist = max(masterlist)
#
                    if elapsed > 11:
                        if compare_master():

                    #minimum distance from master list in MM
                    closest_distance = compute_min(distlist)

                    #find angle (0-359) of the closest object
                    current_minAngle = distlist.index(min(distlist))

                    current_pwm = compute_pwm(closest_distance)

                    #compute difference in pwm value
                    difference_pwm = abs(last_pwm-current_pwm)

                    #test if new value is different from las value and grea
                    if difference_pwm>distance_threshold:

                        easing(last_pwm, current_pwm)

                        #set last_pwm to current pwm
                        last_pwm = current_pwm

                    if current_minAngle != last_minAngle:
                        last_minAngle = current_minAngle

                    #flush the buffer
                    if ser.inWaiting()>max_buffer:
                        ser.flushInput()

                # reset and wait for the next packet
                init_level = 0

            # default, should never happen...
            else:
                init_level = 0

        except KeyboardInterrupt:
            show_data = False
            show_speed = False
            show_angle = False
            print '\nCurrent Data:'
            print 'closest_distance:',closest_distance,'MM'
            print 'current_pwm:',current_pwm, 'PWM'
            print 'current_minAngle',current_minAngle, 'DEG'
            print 'high_dist:',high_dist, 'MM'
            print 'current_speed:',current_speed, 'RPM'
            print 'Pausing...'
            time.sleep(.5)
            print "type a value 0-1023 to set the minimum pwm"
            print "type 'set current' to set current distance to high dista
            print "type 'data' to toggle show data values"
            print "type 'speed' to toggle show motor speed"
            print "type 'quit' to exit"
            print "hit ENTER to continue"
            try:
                response = raw_input()
                if response == 'quit':
                    wiringpi.pwmWrite(1,0)
                    sys.exit()
                    break
                elif response == 'set current':
                    high_dist = closest_distance
                elif response == 'data':
                    show_data = not show_data
                elif response == 'speed':
                    show_speed = not show_speed
                elif test_int(response) and int(response)<=pwm_high:
                    pwm_low = int(response)
                    print 'setting pwm_low to', pwm_low
                    print 'Resuming...'
            except KeyboardInterrupt:
                print 'Resuming...'
                continue
        except:
            traceback.print_exc(file=sys.stdout)
            wiringpi.pwmWrite(1,0)

import serial
ser = serial.Serial(com_port, baudrate)
ser.flushInput()
start=time.clock()
th1 = thread.start_new_thread(read_v_2_4(ser), ())
sys.exit()
```