To connect to the RPI to modify the scan range or if you want to manipulate the resolution (if for instance you wanted high res vertical/ low res horizontal), first you must access the desktop of the RPI. I typically do this using SSH, however, initial setup for each computer/internet connection will require a monitor the first go around (to enter wifi password etc.). I used the wifi hotspot on my laptop, which then gave me the IP address of the RPI, then in command prompt I would enter: ssh [pi@192.168.XXX.XXX](mailto:pi@192.168.XXX.XXX) (using the known IP address of course). See Figure 1 for reference. Then it will prompt for the password (1234). Then you should see green text for the virtual command prompt of the RPI where you need to kick off the VNC server to access the RPI remotely, and without having started the RPI with a monitor (if it is powered on without a monitor it will not render a desktop). Simply type “vncserver” into the RPI command prompt and it will render a virtual desktop for you to access via a common internet connection. You will need VNC viewer on whichever computer you plan to use to connect to this. See figure 2. Then in VNC viewer you just need to open this virtual desktop using the “new” IP address provided by the command prompt, again in figure 2. On the desktop you will find the target file which will run automatically when the RPI gets powered on. It is called “HFVLidar\_V8”. The program is reasonably well commented, Dr. D knows how to get a hold of me if need be. To transfer data: use a flashdrive or VNC to transfer data remotely, click on VNC button in upper right corner of RPI desktop, three bar button and file transfer, Figure 3. It saves as a .csv, but matlab is currently set to run them as .txt files. Runs will show up in /home/pi, the target file MUST stay on the desktop or reconfigure the boot code logic. I will attach a copy of the HFV script and matlab script below. Appendix 1, 2.

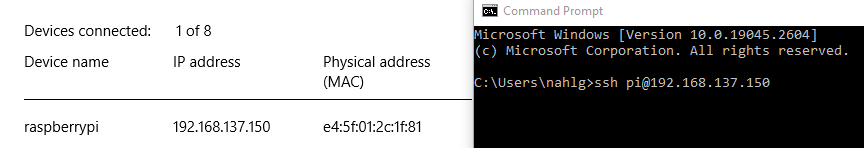


Figure 1



Figure 2.

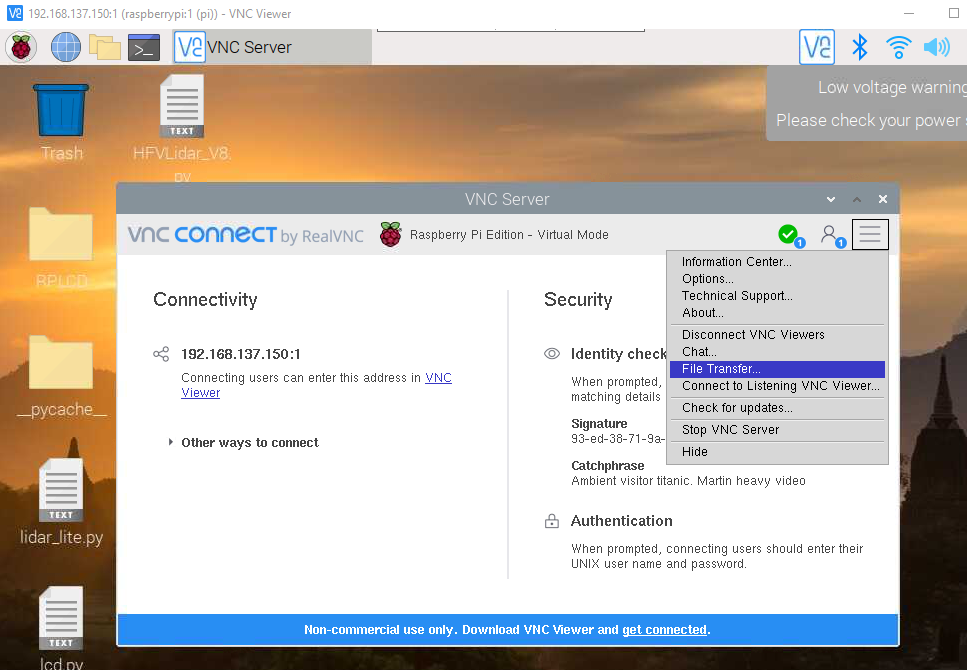


Figure 3.

Appendix 1: HFVLidar\_V8:

#!/usr/bin/python3

#Nate Ahlgren Lidar Gen4 dev code V8

from lidar\_lite import Lidar\_Lite

lidar = Lidar\_Lite()

import math

connected = lidar.connect(1)

import time

from datetime import datetime

import os

import RPi.GPIO as GPIO

import lidar\_lite

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM)

while(True):

hsweep = int(45) #Degrees Change this for horizontal sweep angle

vsweep = int(45) #Degrees Change this for vertical sweep angle

#Initial location values

hstep = 0

vstep = 0

hangle = 0

vangle = 0

hanglepos = 0

#Pin locations/definitions

Vertax = [6,13,19,26]

Horzax = [20,21,4,5]

for pin in Horzax:

GPIO.setup(pin,GPIO.OUT)

GPIO.output(pin,0)

for pin in Vertax:

GPIO.setup(pin,GPIO.OUT)

GPIO.output(pin,0)

halfstepf = [[1,0,0,0], #Clockwise, halfstep logic

[1,1,0,0],

[0,1,0,0],

[0,1,1,0],

[0,0,1,0],

[0,0,1,1],

[0,0,0,1],

[1,0,0,1]]

fullstepf = [[1,0,0,0], #Full step logic

[0,1,0,0],

[0,0,1,0],

[0,0,0,1]]

threestepf = [[1,0,0,0], #1/3 step logic

[0,1,1,0],

[0,0,0,1]]

halfstepr = [[0,0,0,1], #same but for counter clockwise

[0,0,1,1],

[0,0,1,0],

[0,1,1,0],

[0,1,0,0],

[1,1,0,0],

[1,0,0,0],

[1,0,0,1]]

fullstepr = [[0,0,0,1],

[0,0,1,0],

[0,1,0,0],

[1,0,0,0]]

threestepr = [[0,0,0,1],

[0,1,1,0],

[1,0,0,0]]

forward = [halfstepf, #Fun indexing#

fullstepf,

threestepf]

rev = [halfstepr, #gnixedni nuF#

fullstepr,

threestepr]

GPIO.setup(25, GPIO.IN, pull\_up\_down=GPIO.PUD\_DOWN) #button the first

GPIO.setup(7, GPIO.IN, pull\_up\_down=GPIO.PUD\_DOWN) #button the second

GPIO.setup(23, GPIO.IN, pull\_up\_down=GPIO.PUD\_DOWN) #button the third

GPIO.setup(24, GPIO.OUT) #LED

GPIO.output(24, GPIO.HIGH) #ready light on

easybutton = 0

try:

while easybutton<1:

if GPIO.input(25) == GPIO.HIGH:

ssl = 0 #ssl: step size logic, fun indexing and all that

easybutton = easybutton+1

if GPIO.input(7) == GPIO.HIGH:

ssl = 1

easybutton = easybutton+1

if GPIO.input(23) == GPIO.HIGH:

ssl = 2

easybutton = easybutton+1

except:

pass

GPIO.output(24,GPIO.LOW)

print(ssl)

stepsizelogic = [8,4,3] #how many steps in sequence

thresholdlogic = [7,3,2] #how many steps if step 1 is step 0

anglesizelogic = [.0879, .176, .234] #step sizes

forward = forward[ssl]

rev = rev[ssl]

hsweep1 = math.ceil(hsweep/anglesizelogic[ssl]/stepsizelogic[ssl]) #Convert angle to loop count

vsweep1 = math.ceil(vsweep/anglesizelogic[ssl]/stepsizelogic[ssl])

hsweep2 = math.ceil(hsweep1/2) #So lidar starts pointed at target, moves to start of sweep

vsweep2 = math.ceil(vsweep1/2)

hsweep3 = math.ceil(hsweep/2) #this logic should take the angles defined earlier, divide them by 2 and round up

#print(anglesizelogic[ssl]) debugging

ti = datetime.now()

filename1 = "HVF7Lidar\_1a.csv"

dir\_contents = os.listdir("/home/pi/LidarRuns7") #this can be changed to modify where the data is saved

if filename1 in dir\_contents: #run number = filename + # of files in folder +1

dir\_contents.sort()

max\_index = 1

split = dir\_contents[-1].split("\_")

if len(split)>1:

max\_index = len(dir\_contents) +1

filename1 = "HFV7Lidar\_" +str(max\_index) + ".csv"

#filename2 = "HFV7Lidar\_" +str(max\_index) + ".csv"

with open("/home/pi/LidarRuns7/"+str(filename1),"w") as file1:#, open("/home/pi/LidarRuns7/"+str(filename2),"w") as file2:

for i in range(int(hsweep2)): #half sweep to center about zero, no DAQ

for halfstep in range(stepsizelogic[ssl]): #This allows you to point the lidar device at the target

print(halfstep,stepsizelogic[ssl],rev[halfstep],Horzax)

GPIO.output(Horzax, rev[halfstep])

hstep = hstep-1

time.sleep(.05)

for i in range(int(vsweep2)):

for halfstep in range(stepsizelogic[ssl]):

GPIO.output(Vertax, rev[halfstep])

vstep = vstep+1

time.sleep(.05)

while hstep\*anglesizelogic[ssl] <= hsweep3: #vertax takes a half step each time horzax ends a CW/CCW sweep

for i in range(int(vsweep1)): #horzax loops

for halfstep in range(stepsizelogic[ssl]): # 8 halfstep sequence # 4 coils on the motor

distance = lidar.getDistance()

file1.write(str(hstep\*anglesizelogic[ssl])+"," +str(vstep\*anglesizelogic[ssl])+ "," + str(distance) +"\n")

GPIO.output(Vertax, forward[halfstep])

vstep = vstep-1

#time.sleep(.05)

if hanglepos < (thresholdlogic[ssl]): # if statement loop to allow vangle to step continuously

distance = lidar.getDistance() # without looping through 8 step sequence

file1.write(str(hstep\*anglesizelogic[ssl])+"," +str(vstep\*anglesizelogic[ssl]-.7605)+ "," + str(distance) +"\n")#f2 #dcvstep

GPIO.output(Horzax,forward[int(hanglepos)]) #"hanglepos" is how I made this work with individaul steps in the horizontal direction

hstep = hstep+1

hanglepos= hanglepos+1

#time.sleep(.05)

print("Azimuth =" + str(hstep\*anglesizelogic[ssl])+ "of" + str(hsweep))

else:

distance = lidar.getDistance()

file1.write(str(hstep\*anglesizelogic[ssl])+"," +str(vstep\*anglesizelogic[ssl]-.7605)+ "," + str(distance) +"\n")#f2

GPIO.output(Horzax,forward[int(hanglepos)])

hstep = hstep+1

hanglepos= hanglepos-(thresholdlogic[ssl])

time.sleep(.05)

for j in range(int(vsweep1)):

for halfstep in range(stepsizelogic[ssl]): # reverse side of sweep

distance = lidar.getDistance()

file1.write(str(hstep\*anglesizelogic[ssl])+","+str(vstep\*anglesizelogic[ssl]-.7605)+"," +str(distance)+ "," +"\n") #f2 #Single direction data acquisition

GPIO.output(Vertax, rev[halfstep])

vstep = vstep+1

time.sleep(.05)

#print("vstep = %s" % (vstep))

if hanglepos < (thresholdlogic[ssl]): # if statement loop to allow vangle to step continuously

distance = lidar.getDistance() # without looping through 8 step sequence

file1.write(str(hstep\*anglesizelogic[ssl])+"," +str(vstep\*anglesizelogic[ssl])+ "," + str(distance) +"\n")

GPIO.output(Horzax,forward[int(hanglepos)])

hstep = hstep+1

hanglepos= hanglepos+1

#time.sleep(.05)

print("Azimuth =" + str(hstep\*anglesizelogic[ssl])+ "of" + str(hsweep))

else:

distance = lidar.getDistance()

file1.write(str(hstep\*anglesizelogic[ssl])+"," +str(vstep\*anglesizelogic[ssl])+ "," + str(distance) +"\n")

GPIO.output(Horzax,forward[int(hanglepos)])

hstep = hstep+1

hanglepos= hanglepos-(thresholdlogic[ssl])

time.sleep(.05)

tf = datetime.now() #this was to determine the data acquisition rate

tdelta = tf-ti

filename1 = "HFV7Lidartimelog.csv"

with open("/home/pi/LidarRuns7/HFV7Lidartimelog.csv","a") as file: #really dont need this

file.write(filename1 +","+ str(tdelta)+"\n")

if hanglepos!=0: #return to origin section

hanglepos2 = (stepsizelogic[ssl])-hanglepos

else:

hanglepos2 = 0

hhse = hstep/(stepsizelogic[ssl]) #hor half step loop equivalent

hcf = math.ceil(hhse) #Horizontal correction factor

if hstep>0:

for k in range(int(hanglepos2)):

GPIO.output(Horzax,forward[int(hanglepos)])

hstep = hstep+1

hanglepos= hanglepos+1

time.sleep(.05)

print("hstep #2 = %s" % (hstep))

for f in range(int(hcf)):

for halfstep in range(stepsizelogic[ssl]):

GPIO.output(Horzax,rev[halfstep])

hstep = hstep-1

time.sleep(.05)

print("hstep #1 = %s" % (hstep))

vhse = vstep/(stepsizelogic[ssl])

if vstep >0:

for l in range(int(vhse)):

for halfstep in range(stepsizelogic[ssl]):

GPIO.output(Horzax,rev[halfstep])

vstep = vstep - 1

print("vstep #1 = %s" % (vstep))

GPIO.cleanup() #good luck

#except:

#GPIO.cleanup()

Appendix 2: Matlab script:

clear

clc

Configuration=input('Run Configuration ');

switch Configuration %0-5: config sweep, 6: cleaner C1, 7: single full cycle

case 0

%lidar configuration 0: Balanced Performance

rawdata=dlmread('HFV7Lidar\_36c.txt');

low=0;

high=8000;

phi=0;

name='Tree and Runoff';

xlim([3.2,3.9]);

case 1

%lidar configuration 1: short range, high speed, max acquisition count

rawdata=dlmread('convergingfencelines.txt');

low=0;

high=32000;

phi=0;

name='Converging Fencelines';

%xlim([3.2,3.9]);

case 2

%lidar configuration 1: short range, high speed, max acquisition count

rawdata=dlmread('HFV7Lidar\_92c.txt');

low=0;

high=8000;

phi=0;

name='Parking Lot';

case 3

%lidar configuration 2: Default range, higher speed short range. Turns on quick termination detection

%for faster measurements at short range(with decreased accuracy)

rawdata=dlmread('HFV7Lidar\_26c.txt');

low=0;

high=800;

phi=0;

name='Pothole';

xlim([0,1]);

case 4

%lidar configuration 3: Maximum range. Uses 0xff maximum acquisition count.

rawdata=dlmread('HFV7Lidar\_94c.txt');

low=10;

high=1000;

phi=0;

name='Gen 4 HFV sighting range';

case 5

%lidar configuration 4: High sensitivity detection. Overrides default valid measurement detection

%algorithm, and uses a threshold value for high sensitivity and noise.

rawdata=dlmread('HFV7Lidar\_20cminus.TXT');

low=10;

high=1000;

phi=-5;

name='VW BUG';

case 6

%lidar configuration 5: Low sensitivity detection. Overrides default valid measurement detection

%algorithm, and uses a threshold value for low sensitivity and noise.

rawdata=dlmread('HFV6Lidar\_27.TXT');

low=10;

high=1000;

phi=-20;

name='Configration 5';

case 7

rawdata=dlmread('Sighting range c11.TXT');

low=10;

high=1000;

phi=-20;

name='Gen 3 Configration 1';

case 8

rawdata=dlmread('LidarGen4\_10.TXT');

low=0;

high=1000;

phi=0;

name='Leading edge sweep';

end

rawdata(rawdata(:,3)>high,:)=[];

rawdata(rawdata(:,3)<low,:)=[];

azimuth=-deg2rad(rawdata(:,1));%/.18\*360/2048);

elevation=deg2rad(rawdata(:,2));%/.18\*360/2048+phi);

distance=rawdata(:,3)/100;

[x,y,z]=sph2cart(azimuth,elevation,distance);

figure(1)

switch Configuration;

case {0,1,2,3,4,5,6,7}

scatter3(x,y,z,15,distance,'o','filled');

%meshgrid(x,y)

% case 6

% scatter3(y,z,-x,10,-x,'o')

%case {7,8}

%scatter3(x,y,z,10,z,'o','filled')

end

xlim([0,6]);

%ylim([-11,-5]);

%zlim([-1,1])

xlabel('x-axis')

ylabel('y-axis')

zlabel('z-axis')

zlim([-1,1.5]);

%xlim([10,40]);

%zlim([zl(1)-(mean(x)-mean(z)) zl(2)-(mean(x)-mean(z))])

xlabel('X [m]');

ylabel('Y [m]');

zlabel('Z [m]');

title(name);

colormap();%colorcube);%

%caxis([2.4,5.6]);

%caxis([24,29]);

%caxis([8,9]);

%caxis([24,35]);

colorbar;

%% Depcik contourf plot

% x => this is the distance data (typically z-direction for contour plots)

% y => typically the x-direction for contour plots

% z => typically the y-direction for contour plots

%%%Xc = y; %Start here

% Yc = z;

% Zc = x;

% resX = 1000; % How many datapoints to plot in the contour X-direction

% resY = 1000; % How many datapoints to plot in the contour Y-direction

% resC = 10; % How many contour lines to show

% Xi = linspace(min(Xc),max(Xc),resX);

% Yi = linspace(min(Yc),max(Yc),resY);

% Zg = griddata(Xc,Yc',Zc,Xi,Yi');

% figure(2)

% contourf(Xi,Yi',Zg,resC);

% xlabel('Y')

% ylabel('Z')

% zlabel('X')

% colorbar %Allows for a color legends for distance

% colormap(jet) %Creates a more detailed colorbar.

%