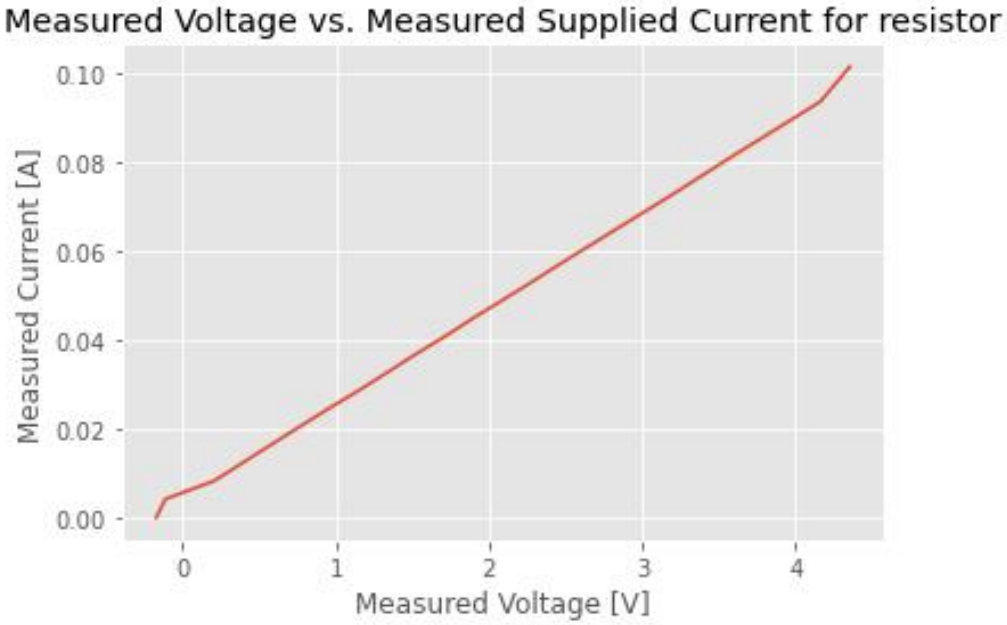
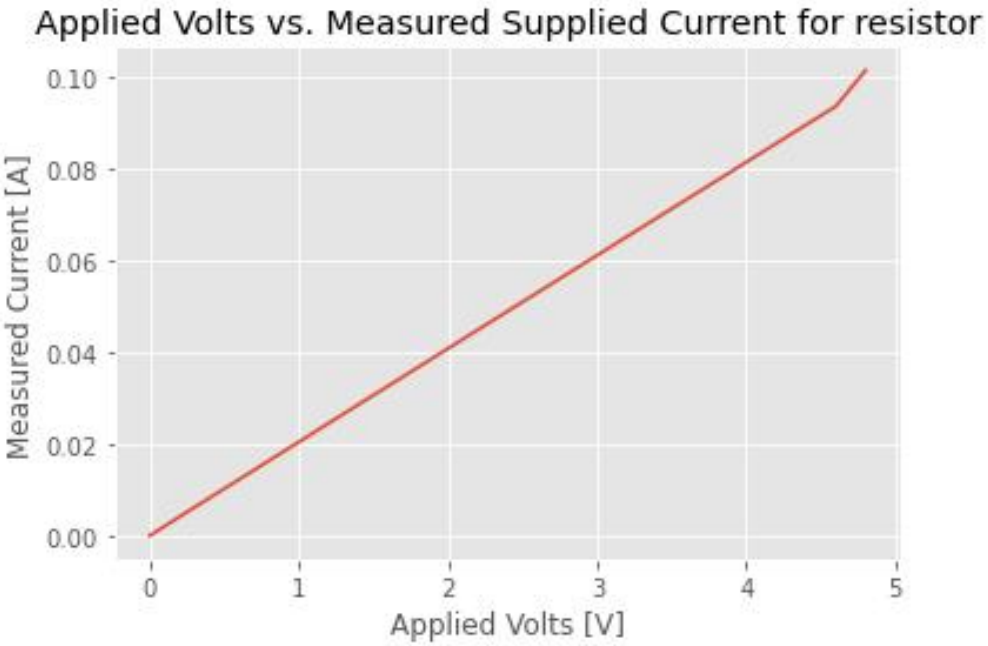
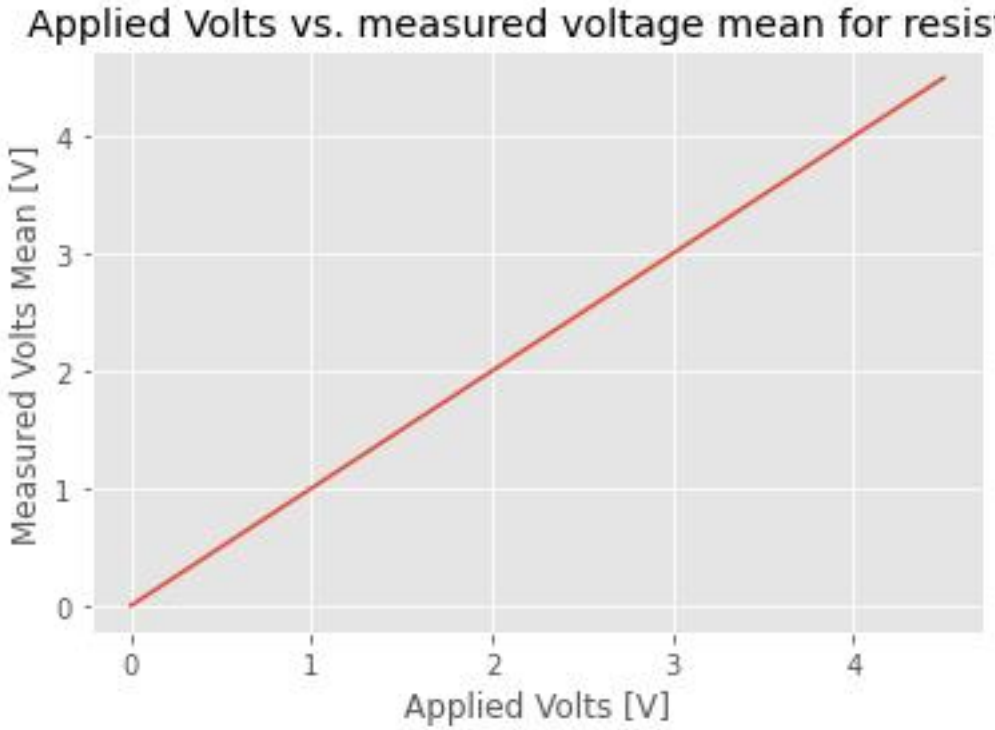
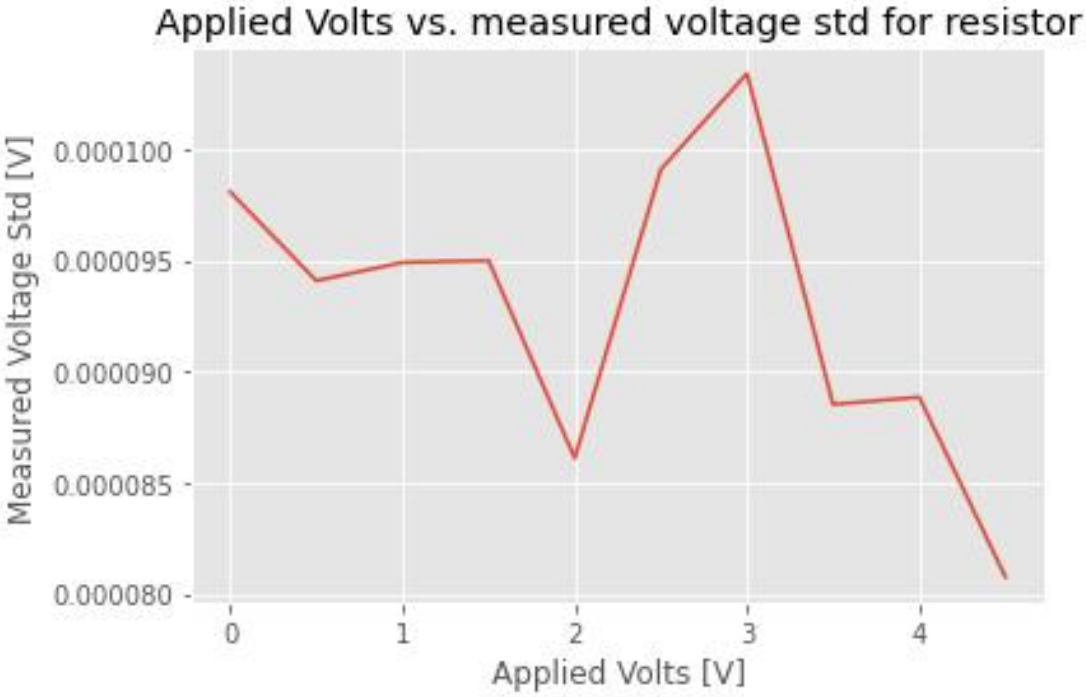
Postlab questions:

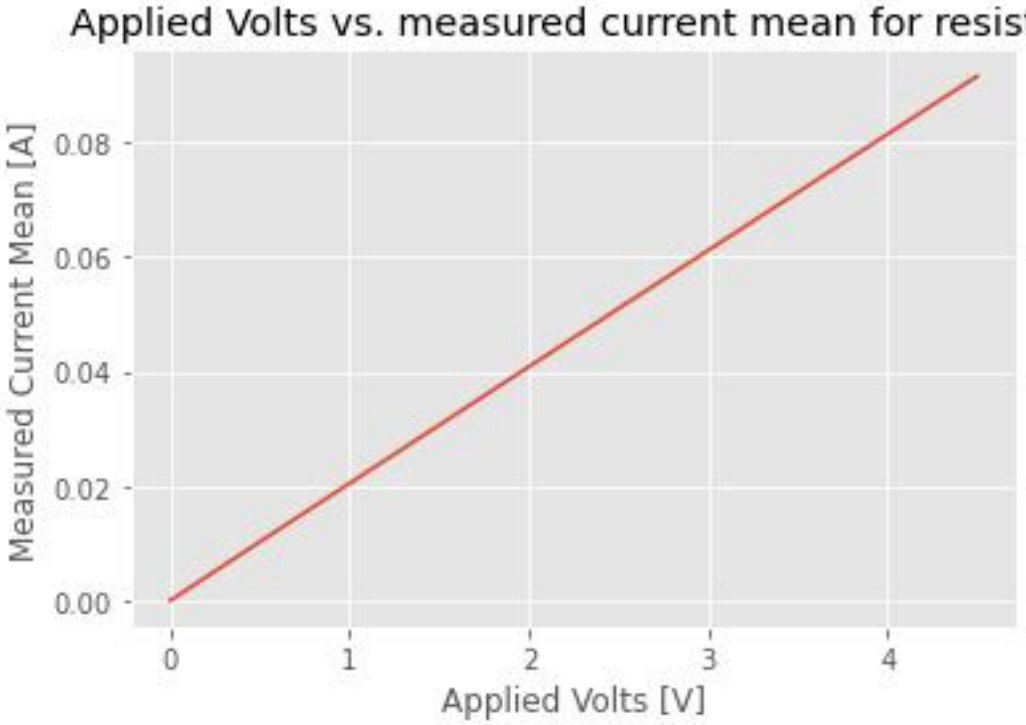
MS1 graphs:

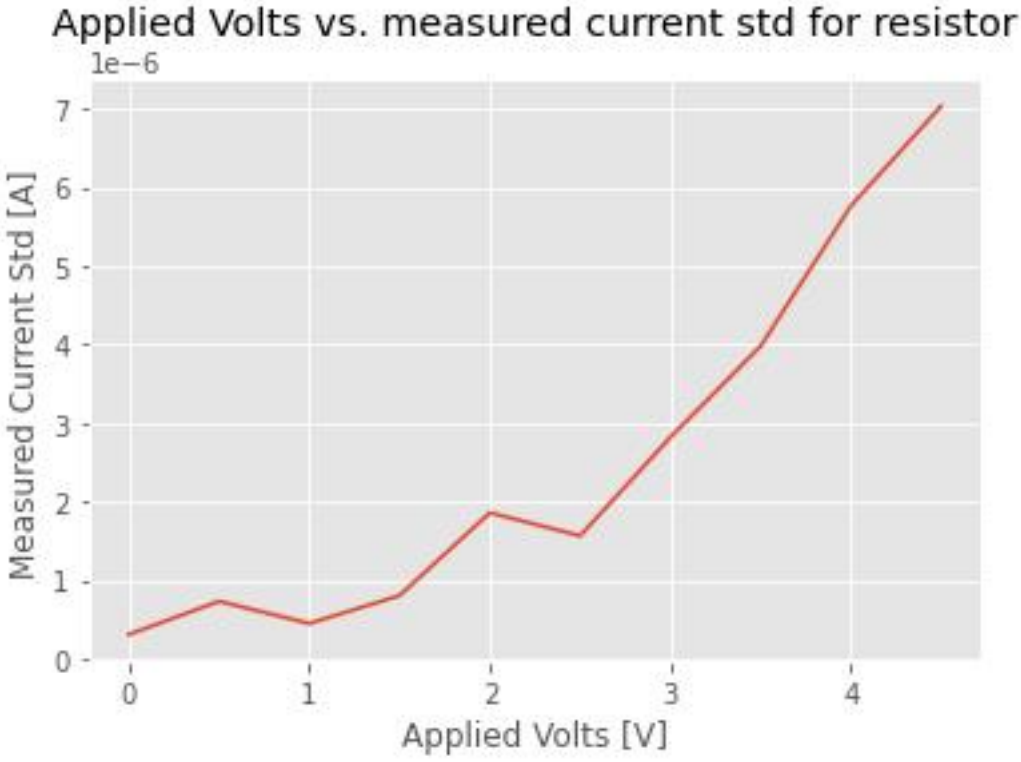


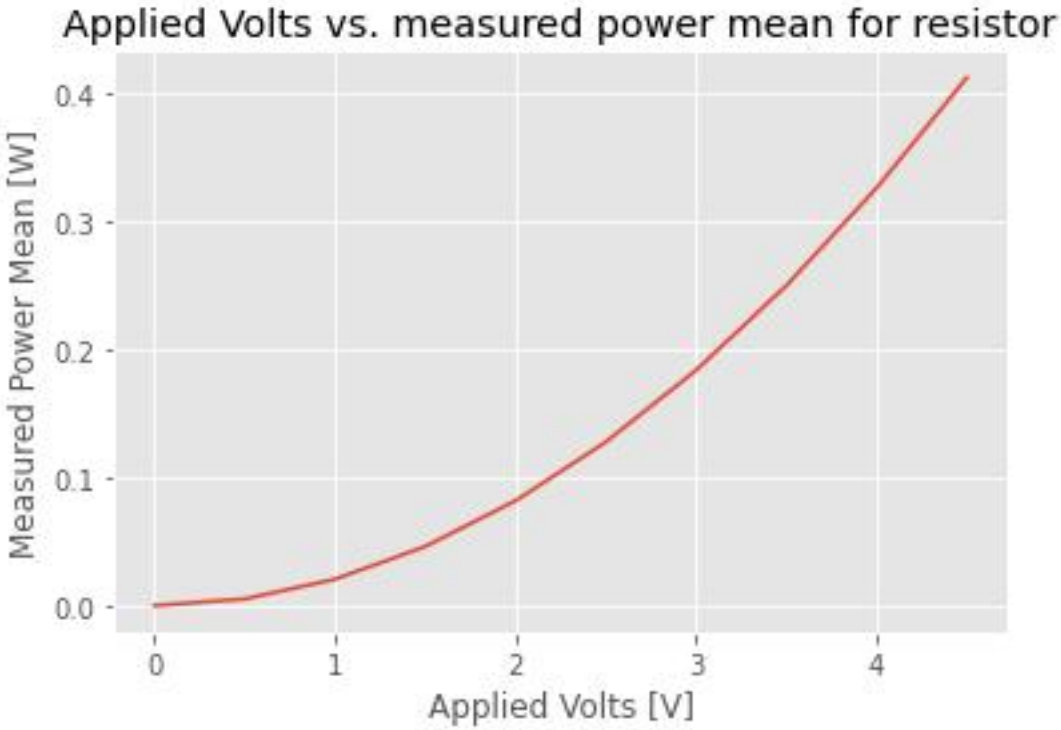


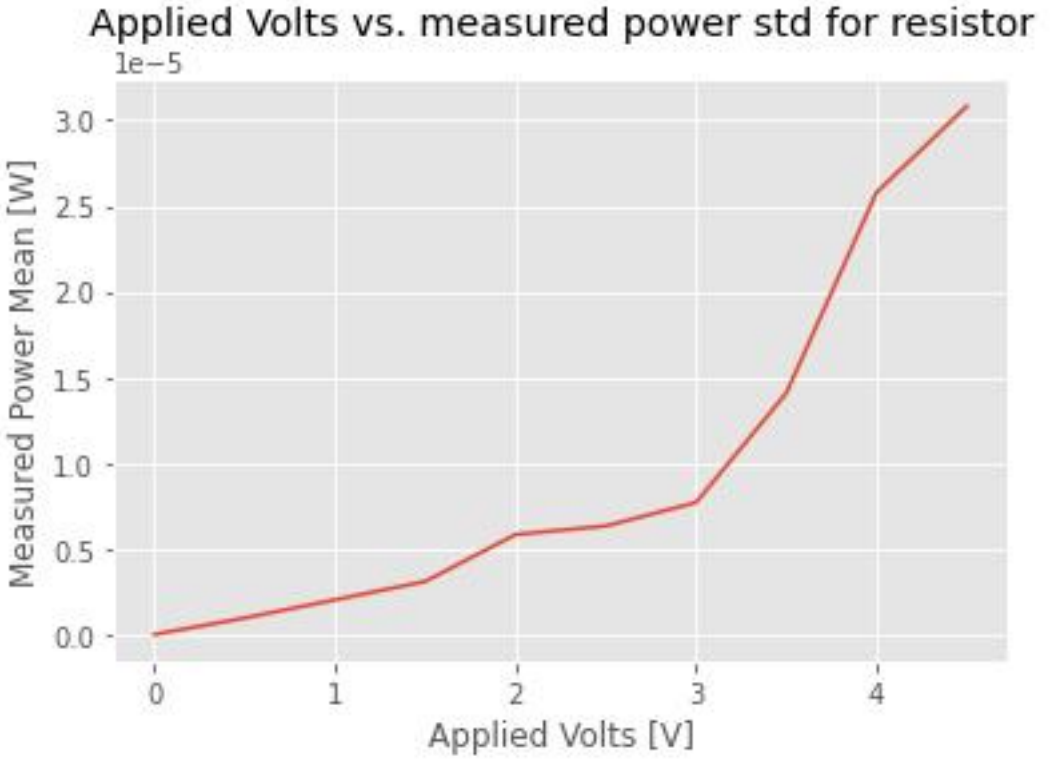

Applied Volts vs. measured voltage mean for resistor
Measured Volts Mean [V]
0
1
2
3
4
Applied Volts [V]


Applied Volts vs. measured voltage std for resistor
Measured Voltage Std [V]
0.000100
0.000095
0.000090
0.000085
0.000080
0
1
2
3
4
Applied Volts [V]


Applied Volts vs. measured current mean for resistor
Measured Current Mean [A]
0.00
0.02
0.04
0.06
0.08
0
1
2
3
4
Applied Volts [V]


Applied Volts vs. measured current std for resistor
1e−6
Measured Current Std [A]
0
1
2
3
4
5
6
7
0
1
2
3
4
Applied Volts [V]



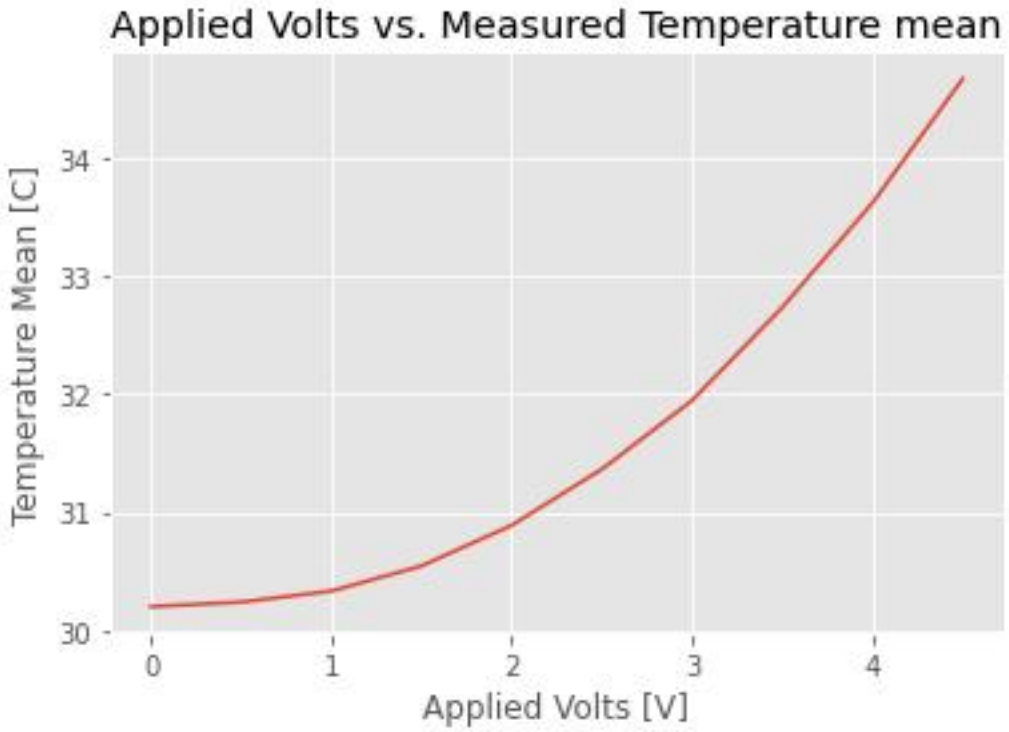


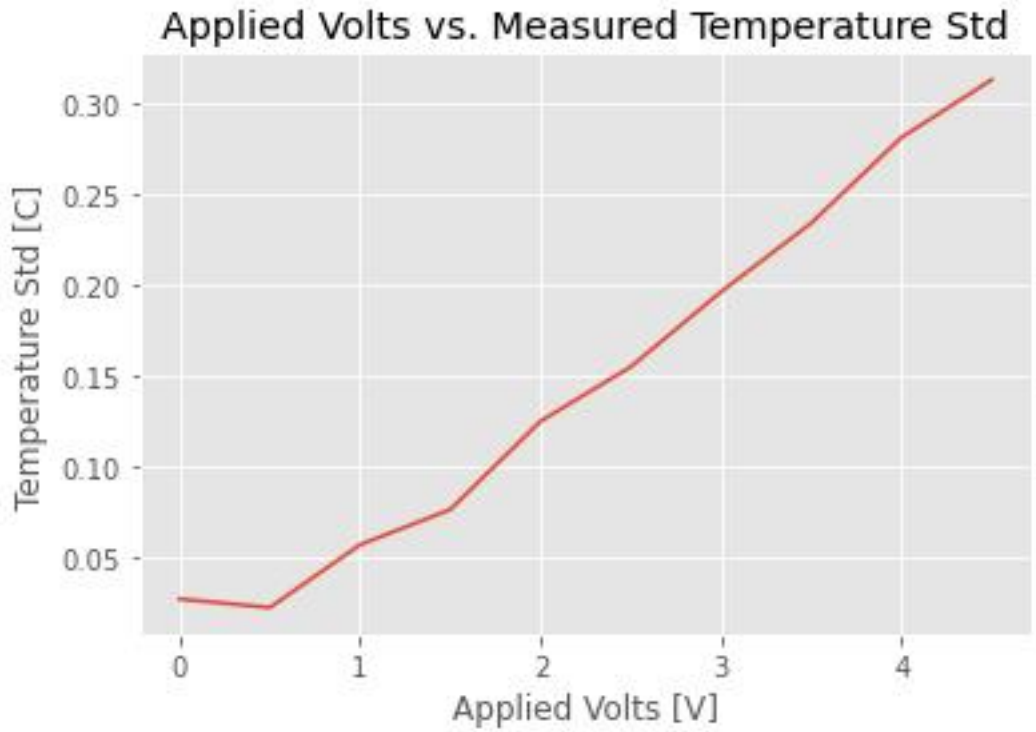
MS1:

I see the mean of measured voltage and measured currents are linearly correlated with the applied voltage, while the mean of measured power

is quadratic correlated with the applied voltage. For the standard deviation, while the standard deviation of measure voltage stay stable, the standard deviation of measured current and measured power increase quadratic while the applied voltage is increase. I will model the measured them with Vmeasured = Vapplied, A = Vappiled / R (47 ohms), and P = V * C = Vappiled^2 / R, and the mean of these measurement follow the theoretical model very well.

MS2 graphs:





MS2:

Yes, the standard deviation is different for different temperature/Applied voltage. The higher the temperature, the higher the standard deviation. This is because as the temperature increases, the accuracy decreases for the temperature sensor. This is also shown in the datasheet of the sensor:

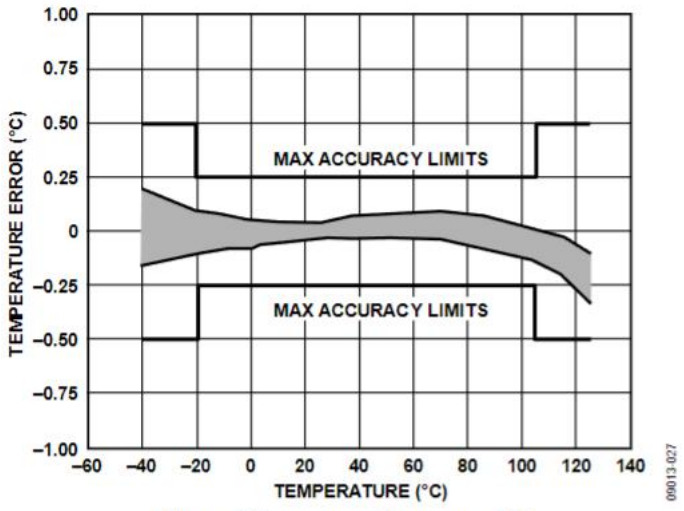


Figure 4. Temperature Accuracy at 3 V

As we can see from the graph, as the temperature increase from 30 to 35, the shaded area increases, implying the error will be higher so the standard deviation will increases, which explains why the measured temperature standard deviation increases as the temperature increases. The graph we get from measurement correspond with the figure from datasheet very well. Differences between our grade and this graph from datasheet are that it has a wider x-axis range for different error from -60 to 140 and that it shows the absolute error range from negative to positive while our measurement just shows the standard deviation.

Code:

MS1:

```
# This code reads data from the temperature sensor and outputs the results on the screen.
# The bit file programs OpalKelly's XEM7310 board with a finite state machine that implements
# I2C protocol. With this protocol, temperature data is received from the temperature sensor
# to the FPGA. Then the FPGA transfers the data from the two registers containing
# the temperature data to the PC using OKWireOut.

# import various libraries necessary to run your Python code
import pyvisa as visa # You should pip install pyvisa and restart the kernel.
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import time      # time related library
import sys,os        # system related library
ok_sdk_loc = "C:\\Program Files\\Opal Kelly\\FrontPanelUSB\\API\\Python\\x64"
ok_dll_loc = "C:\\Program Files\\Opal Kelly\\FrontPanelUSB\\API\\lib\\x64"
mpl.style.use('ggplot')
sys.path.append(ok_sdk_loc)      # add the path of the OK library
os.add_dll_directory(ok_dll_loc)

import ok          # OpalKelly libraryy

#%%
```

```
# Define FrontPanel device variable, open USB communication and
# load the bit file in the FPGA
dev = ok.okCFrontPanel();    # define a device for FrontPanel communication
SerialStatus=dev.OpenBySerial("");          # open USB communication with the OK board
ConfigStatus=dev.ConfigureFPGA("../Provided_bit/I2C_Temperature.bit"); # Configure the FPGA with this bit file

# Check if FrontPanel is initialized correctly and if the bit file is loaded.
# Otherwise terminate the program
print("----------------------------------------------------")
if SerialStatus == 0:
    print ("FrontPanel host interface was successfully initialized.")
else:
    print ("FrontPanel host interface not detected. The error code number is:" + str(int(SerialStatus)))
    print("Exiting the program.")
    sys.exit()

if ConfigStatus == 0:
    print ("Your bit file is successfully loaded in the FPGA.")
else:
    print ("Your bit file did not load. The error code number is:" + str(int(ConfigStatus)))
    print ("Exiting the progam.")
    sys.exit()
print("----------------------------------------------------")
print("----------------------------------------------------")

#%%
# This section of the code cycles through all USB connected devices to the computer.
# The code figures out the USB port number for each instrument.
# The port number for each instrument is stored in a variable named “instrument_id”
# If the instrument is turned off or if you are trying to connect to the
# keyboard or mouse, you will get a message that you cannot connect on that port.
device_manager = visa.ResourceManager()
devices = device_manager.list_resources()
number_of_device = len(devices)

power_supply_id = -1
waveform_generator_id = -1
digital_multimeter_id = -1
oscilloscope_id = -1

# assumes only the DC power supply is connected
for i in range (0, number_of_device):
```

```
# check that it is actually the power supply
    try:
        device_temp = device_manager.open_resource(devices[i])
        print("Instrument connect on USB port number [" + str(i) + "] is " + device_temp.query("*IDN?"))
        if (device_temp.query("*IDN?") == 'HEWLETT-PACKARD,E3631A,0,3.2-6.0-2.0\r\n'):
            power_supply_id = i
        if (device_temp.query("*IDN?") == 'HEWLETT-PACKARD,E3631A,0,3.0-6.0-2.0\r\n'):
            power_supply_id = i
        if (device_temp.query("*IDN?") == 'Agilent Technologies,33511B,MY52301259,3.03-1.19-2.00-52-00\n'):
            waveform_generator_id = i
        if (device_temp.query("*IDN?") == 'Agilent Technologies,34461A,MY53208026,A.01.10-02.25-01.10-00.35-01-01\n'):
            digital_multimeter_id = i
        if (device_temp.query("*IDN?") == 'Keysight Technologies,34461A,MY53212931,A.02.08-02.37-02.08-00.49-01-01\n'):
            digital_multimeter_id = i
        if (device_temp.query("*IDN?") == 'KEYSIGHT TECHNOLOGIES,MSO-X 3024T,MY54440318,07.50.2021102830\n'):
            oscilloscope_id = i
        device_temp.close()
    except:
        print("Instrument on USB port number [" + str(i) + "] cannot be connected. The instrument might be powered of or you are trying to connect to a mouse or keyboard.\n")

#%%
# Open the USB communication port with the power supply.
# The power supply is connected on USB port number power_supply_id.
# If the power supply ss not connected or turned off, the program will exit.
# Otherwise, the power_supply variable is the handler to the power supply

if (power_supply_id == -1):
    print("Power supply instrument is not powered on or connected to the PC.")
else:
    print("Power supply is connected to the PC.")
    power_supply = device_manager.open_resource(devices[power_supply_id])

#%%
# Open the USB communication port with the power supply.
# The power supply is connected on USB port number power_supply_id.
# If the power supply ss not connected or turned off, the program will exit.
# Otherwise, the power_supply variable is the handler to the power supply
```

```
if (digital_multimeter_id == -1):
    print("Digital multimeter instrument is not powered on or connected to the PC.")
else:
    print("Digital multimeter is connected to the PC.")
    digital_multimeter = device_manager.open_resource(devices[digital_multimeter_id])

#%%
# Open the USB communication port with the power supply.
# The power supply is connected on USB port number power_supply_id.
# If the power supply ss not connected or turned off, the program will exit.
# Otherwise, the power_supply variable is the handler to the power supply

if (oscilloscope_id == -1):
    print("Oscilloscope instrument is not powered on or connected to the PC.")
else:
    print("Oscilloscope is connected to the PC.")
    oscilloscope = device_manager.open_resource(devices[oscilloscope_id])

#%% Press control-C in the console window to stop the loop
print(power_supply.write("OUTPUT ON"))
output_voltage = np.arange(0, 5, 0.5)
measured_voltage = np.array([]) # create an empty list to hold our values
measured_current = np.array([]) # create an empty list to hold our values
measured_voltage_mean = np.array([])
measured_voltage_std = np.array([])
measured_current_mean = np.array([])
measured_current_std = np.array([])
measured_power_mean = np.array([])
measured_power_std = np.array([])
try:
    for v in output_voltage:
        power_supply.write("APPLy P25V, %0.2f, 0.1" % v)
        time.sleep(0.5)
        same_volt_power_measurement = np.array([])
        same_volt_voltage_measurement = np.array([])
        same_volt_current_measurement = np.array([])
        measured_voltage_tmp = oscilloscope.query("MEASure:VAVERAGE? DISPLAY,
CHANNEL1")
        measured_voltage = np.append(measured_voltage, measured_voltage_tmp)
            # read the output current on the 6V power supply
        measured_current_tmp = digital_multimeter.query("MEASure:CURRent:DC?")
        measured_current = np.append(measured_current, measured_current_tmp)
        for i in range(20):
```

```
                measured_voltage_tmp = power_supply.query("MEASure:VOLTage:DC? P25V")
                # read the output current on the 6V power supply
                measured_current_tmp = digital_multimeter.query("MEASure:CURRent:DC?")
                measured_current = np.append(measured_current, measured_current_tmp)
                power_consumption = float(measured_voltage_tmp) *
float(measured_current_tmp)
                if power_consumption > 0.5:
                    print("Exceeding 0.5W")
                    break
                same_volt_power_measurement = np.append(same_volt_power_measurement,
float(power_consumption))
                same_volt_voltage_measurement = np.append(same_volt_voltage_measurement,
float(measured_voltage_tmp))
                same_volt_current_measurement = np.append(same_volt_current_measurement,
float(measured_current_tmp))
                time.sleep(0.2)
            measured_power_mean = np.append(measured_power_mean,
np.mean(same_volt_power_measurement))
            measured_power_std = np.append(measured_power_std,
np.std(same_volt_power_measurement))
            measured_voltage_mean = np.append(measured_voltage_mean,
np.mean(same_volt_voltage_measurement))
            measured_voltage_std = np.append(measured_voltage_std,
np.std(same_volt_voltage_measurement))
            measured_current_mean = np.append(measured_current_mean,
np.mean(same_volt_current_measurement))
            measured_current_std = np.append(measured_current_std,
np.std(same_volt_current_measurement))
except KeyboardInterrupt:
    pass
print(power_supply.write("OUTPUT OFF"))

#%% Plot measured data. First convert the data from strings to numbers (ie floats)
power_mean_list = np.zeros(np.size(output_voltage))
power_std_list = np.zeros(np.size(output_voltage))
voltage_mean_list=np.zeros(np.size(output_voltage))
voltage_std_list=np.zeros(np.size(output_voltage))
current_mean_list=np.zeros(np.size(output_voltage))
current_std_list=np.zeros(np.size(output_voltage))
for i in range(len(output_voltage)):
    voltage_mean_list[i]= float(measured_voltage_mean[i])
    voltage_std_list[i]= float(measured_voltage_std[i])
    current_mean_list[i]= float(measured_current_mean[i])
    current_std_list[i]= float(measured_current_std[i])
```

```
        power_mean_list[i] = float(measured_power_mean[i])
        power_std_list[i] = float(measured_power_std[i])

# plot results (applied voltage vs measured voltage mean)
plt.figure()
plt.plot(output_voltage, voltage_mean_list)
plt.title("Applied Volts vs. measured voltage mean for resistor")
plt.xlabel("Applied Volts [V]")
plt.ylabel("Measured Volts Mean [V]")
plt.draw()
# plot results (applied voltage vs measured voltage std)
plt.figure()
plt.plot(output_voltage, voltage_std_list)
plt.title("Applied Volts vs. measured voltage std for resistor")
plt.xlabel("Applied Volts [V]")
plt.ylabel("Measured Voltage Std [V]")
plt.draw()
# plot results (applied voltage vs measured current mean)
plt.figure()
plt.plot(output_voltage, current_mean_list)
plt.title("Applied Volts vs. measured current mean for resistor")
plt.xlabel("Applied Volts [V]")
plt.ylabel("Measured Current Mean [A]")
plt.draw()
# plot results (applied voltage vs measured current std)
plt.figure()
plt.plot(output_voltage, current_std_list)
plt.title("Applied Volts vs. measured current std for resistor")
plt.xlabel("Applied Volts [V]")
plt.ylabel("Measured Current Std [A]")
plt.draw()
# plot results (applied voltage vs measured power mean)
plt.figure()
plt.plot(output_voltage, power_mean_list)
plt.title("Applied Volts vs. measured power mean for resistor")
plt.xlabel("Applied Volts [V]")
plt.ylabel("Measured Power Mean [W]")
plt.draw()
# plot results (applied voltage vs measured power std)
plt.figure()
plt.plot(output_voltage, power_std_list)
plt.title("Applied Volts vs. measured power std for resistor")
plt.xlabel("Applied Volts [V]")
plt.ylabel("Measured Power Mean [W]")
```

```
plt.draw()
plt.show()
```

MS2:

```
# This code reads data from the temperature sensor and outputs the results on the screen.
# The bit file programs OpalKelly's XEM7310 board with a finite state machine that implements
# I2C protocol. With this protocol, temperature data is received from the temperature sensor
# to the FPGA. Then the FPGA transfers the data from the two registers containing
# the temperature data to the PC using OKWireOut.

# import various libraries necessary to run your Python code
import pyvisa as visa # You should pip install pyvisa and restart the kernel.
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import time      # time related library
import sys,os        # system related library
ok_sdk_loc = "C:\\Program Files\\Opal Kelly\\FrontPanelUSB\\API\\Python\\x64"
ok_dll_loc = "C:\\Program Files\\Opal Kelly\\FrontPanelUSB\\API\\lib\\x64"
mpl.style.use('ggplot')
sys.path.append(ok_sdk_loc)      # add the path of the OK library
os.add_dll_directory(ok_dll_loc)

import ok          # OpalKelly libraryy

#%%
# Define FrontPanel device variable, open USB communication and
# load the bit file in the FPGA
dev = ok.okCFrontPanel();    # define a device for FrontPanel communication
SerialStatus=dev.OpenBySerial("");           # open USB communication with the OK board
ConfigStatus=dev.ConfigureFPGA("../Provided_bit/I2C_Temperature.bit"); # Configure the FPGA
with this bit file

# Check if FrontPanel is initialized correctly and if the bit file is loaded.
# Otherwise terminate the program
print("------------------------------------------------------")
if SerialStatus == 0:
      print ("FrontPanel host interface was successfully initialized.")
else:
      print ("FrontPanel host interface not detected. The error code number is:" +
str(int(SerialStatus)))
      print("Exiting the program.")
      sys.exit()
```

```
if ConfigStatus == 0:
    print ("Your bit file is successfully loaded in the FPGA.")
else:
    print ("Your bit file did not load. The error code number is:" + str(int(ConfigStatus)))
    print ("Exiting the progam.")
    sys.exit()
print("-----------------------------------------------------")
print("-----------------------------------------------------")

#%%
# This section of the code cycles through all USB connected devices to the computer.
# The code figures out the USB port number for each instrument.
# The port number for each instrument is stored in a variable named “instrument_id”
# If the instrument is turned off or if you are trying to connect to the
# keyboard or mouse, you will get a message that you cannot connect on that port.
device_manager = visa.ResourceManager()
devices = device_manager.list_resources()
number_of_device = len(devices)

power_supply_id = -1
waveform_generator_id = -1
digital_multimeter_id = -1
oscilloscope_id = -1

# assumes only the DC power supply is connected
for i in range (0, number_of_device):

# check that it is actually the power supply
    try:
        device_temp = device_manager.open_resource(devices[i])
        print("Instrument connect on USB port number [" + str(i) + "] is " +
device_temp.query("*IDN?"))
        if (device_temp.query("*IDN?") == 'HEWLETT-PACKARD,E3631A,0,3.2-6.0-2.0\r\n'):
            power_supply_id = i
        if (device_temp.query("*IDN?") == 'HEWLETT-PACKARD,E3631A,0,3.0-6.0-2.0\r\n'):
            power_supply_id = i
        if (device_temp.query("*IDN?") == 'Agilent
Technologies,33511B,MY52301259,3.03-1.19-2.00-52-00\n'):
            waveform_generator_id = i
        if (device_temp.query("*IDN?") == 'Agilent
Technologies,34461A,MY53208026,A.01.10-02.25-01.10-00.35-01-01\n'):
            digital_multimeter_id = i
        if (device_temp.query("*IDN?") == 'Keysight
Technologies,34461A,MY53212931,A.02.08-02.37-02.08-00.49-01-01\n'):
```

```
                digital_multimeter_id = i
            if (device_temp.query("*IDN?") == 'KEYSIGHT TECHNOLOGIES,MSO-X 3024T,MY54440318,07.50.2021102830\n'):
                oscilloscope_id = i
            device_temp.close()
        except:
            print("Instrument on USB port number [" + str(i) + "] cannot be connected. The instrument might be powered of or you are trying to connect to a mouse or keyboard.\n")

#%%
# Open the USB communication port with the power supply.
# The power supply is connected on USB port number power_supply_id.
# If the power supply ss not connected or turned off, the program will exit.
# Otherwise, the power_supply variable is the handler to the power supply

if (power_supply_id == -1):
    print("Power supply instrument is not powered on or connected to the PC.")
else:
    print("Power supply is connected to the PC.")
    power_supply = device_manager.open_resource(devices[power_supply_id])

#%%
# Open the USB communication port with the power supply.
# The power supply is connected on USB port number power_supply_id.
# If the power supply ss not connected or turned off, the program will exit.
# Otherwise, the power_supply variable is the handler to the power supply

if (digital_multimeter_id == -1):
    print("Digital multimeter instrument is not powered on or connected to the PC.")
else:
    print("Digital multimeter is connected to the PC.")
    digital_multimeter = device_manager.open_resource(devices[digital_multimeter_id])

#%%
# Open the USB communication port with the power supply.
# The power supply is connected on USB port number power_supply_id.
# If the power supply ss not connected or turned off, the program will exit.
# Otherwise, the power_supply variable is the handler to the power supply

if (oscilloscope_id == -1):
    print("Oscilloscope instrument is not powered on or connected to the PC.")
else:
    print("Oscilloscope is connected to the PC.")
    oscilloscope = device_manager.open_resource(devices[oscilloscope_id])
```

```
#%% Press control-C in the console window to stop the loop
print(power_supply.write("OUTPUT ON"))
output_voltage = np.arange(0, 5, 0.5)
measured_voltage = np.array([]) # create an empty list to hold our values
measured_temp_mean = np.array([])
measured_temp_std = np.array([])
try:
    for v in output_voltage:
        power_supply.write("APPLy P25V, %0.2f, 0.1" % v)
        time.sleep(0.5)
        same_volt_temp_measurement = np.array([])
        measured_voltage_tmp = oscilloscope.query("MEASure:VAVERAGE? DISPLAY, CHANNEL1")
        measured_voltage = np.append(measured_voltage, measured_voltage_tmp)
        for i in range(20):
            dev.SetWireInValue(0x00, 1); # Sending 1 at memory location 0x00 starts the FSM
            dev.UpdateWireIns();    # Update the WireIns
            time.sleep(0.5)
            dev.UpdateWireOuts()    # Receive the temperature data
            temperature_msb = dev.GetWireOutValue(0x20)    # MSB temperature register
            temperature_lsb = dev.GetWireOutValue(0x21)    # LSB temperature register
            temperature = float(((temperature_msb<<8) + temperature_lsb))/8*0.0625; # Put the temperature data together
            same_volt_temp_measurement = np.append(same_volt_temp_measurement, temperature)
            time.sleep(0.5);
            print ("Temperature is:" + str((temperature))); # print the results
        measured_temp_mean = np.append(measured_temp_mean, np.mean(same_volt_temp_measurement))
        measured_temp_std = np.append(measured_temp_std, np.std(same_volt_temp_measurement))
except KeyboardInterrupt:
    pass
print(power_supply.write("OUTPUT OFF"))

#%% Plot measured data. First convert the data from strings to numbers (ie floats)
#power_mean_list = np.zeros(np.size(measured_power_mean))
#power_std_list = np.zeros(np.size(measured_power_std))
voltage_list=np.zeros(np.size(output_voltage))
temp_mean_list=np.zeros(np.size(output_voltage))
temp_std_list=np.zeros(np.size(output_voltage))
for i in range(len(measured_voltage)):
    voltage_list[i]= float(measured_voltage [i])
```

```
        temp_mean_list[i] = float(measured_temp_mean[i])
        temp_std_list[i] = float(measured_temp_std[i])

# plot results (applied voltage vs measured supplied current)
plt.figure()
plt.plot(output_voltage, temp_mean_list)
plt.title("Applied Volts vs. Measured Temperature mean")
plt.xlabel("Applied Volts [V]")
plt.ylabel("Temperature Mean [C]")
plt.draw()
# plot results (applied voltage vs measured supplied current)
plt.figure()
plt.plot(output_voltage, temp_std_list)
plt.title("Applied Volts vs. Measured Temperature Std")
plt.xlabel("Applied Volts [V]")
plt.ylabel("Temperature Std [C]")
plt.draw()
plt.show()
```