CPU Temperature, Utilization, and Frequency Log

Department of Electrical and Computer Engineering

ECE 785: Spring 2024 (Dr.Dean)

Project #1 Report: Basic Performance Analysis and Display

by

Loyda Yusufova

(2/9/2024)

Introduction

This project aimed to modify a simple Python script which can be run as a background command on the Raspberry Pi terminal to measure the SoC temperature. The script was modified to additionally measure the CPU frequency and utilization periodically, log them and display them in a live graph with matplotlib.

The starter code for this project is from Raspberry Pi Organization's Temperature Log example project which already has the CPU temperature log implemented. The starter code uses the system on chip (SoC) of the Raspberry Pi's temperature sensor to measure its temperature from the command line.

In this project, the Temperature Log example program was restructured and then modified so it also monitors CPU frequency and utilization.

Starter Code

The starter code was found here: https://projects.raspberrypi.org/en/projects/temperature-log

The code was obtained by scrolling to the bottom of the page and clicking on "completed project here" as shown in the highlighted section of the following image:

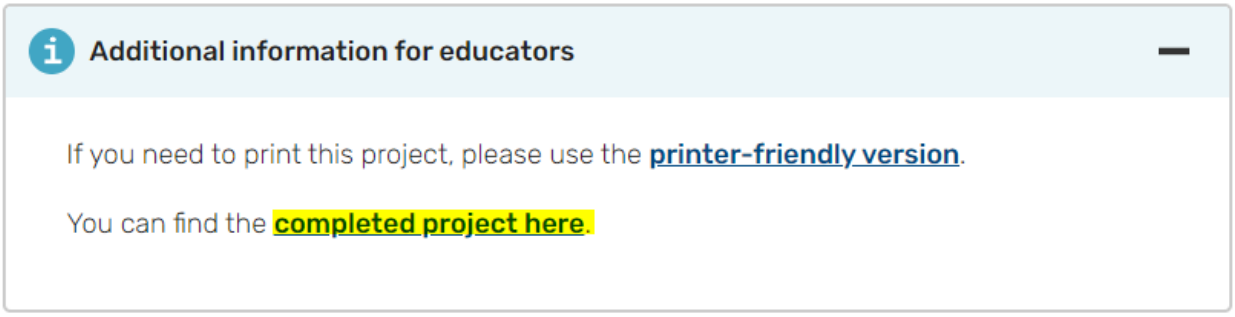


Figure 1: Screenshot of temperature-log tutorial.

Additionally, a screenshot of the starter code is provided for illustration:

```
from gpiozero import CPUTemperature
from time import sleep, strftime, time
import matplotlib.pyplot as plt

cpu = CPUTemperature()

plt.ion()
x = []
y = []

def write_temp(temp):
    with open("/home/pi/cpu_temp.csv", "a") as log:
        log.write("{0},{1}\n".format(strftime("%Y-%m-%d %H:%M:%S"),str(temp)))

def graph(temp):
    y.append(temp)
    x.append(time())
    plt.clf()
    plt.scatter(x,y)
    plt.plot(x,y)
    plt.draw()

while True:
    temp = cpu.temperature
    write_temp(temp)
    graph(temp)
    plt.pause(1)
```

Figure 2: Starter code.

Restructuring the Starter Code: Temperature Measurement Graph in Real-Time

The starter code was re-structured so that once the temperature is sampled the temperature graph is updated immediately. This restructuring allows the temperature graph to be updated in real-time.

To achieve this restructuring, I followed the tutorial "Graph Sensor Data with Python and Matplotlib" written by Shawn Hymel where he creates a real-time plot using the animation module in Matplotlib.

The tutorial can be found here: https://learn.sparkfun.com/tutorials/graph-sensor-data-with-python-and-matplotlib/update-a-graph-in-real-time

The contents of the resulting python script are shown here:

```python
# Starter code based on https://projects.raspberrypi.org/en/projects/temperature-log
# and https://github.com/raspberrypilearning/temperature-log
from gpiozero import CPUTemperature
from time import sleep, strftime, time
import matplotlib.pyplot as plt
import datetime as dt
import matplotlib.animation as animation

# Create figure for plotting
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
xs = []
ys = []

# Initialize communication cpu_temperature
cpu = CPUTemperature()
#tmp102.init()

# This function is called periodically from FuncAnimation
def animate(i, xs, ys):
    # Read temperature (Celsius) from gpiozero
    temp=cpu.temperature

    # Add x and y to lists
    xs.append(dt.datetime.now().strftime('%H:%M:%S.%f'))
    ys.append(temp)

    # Limit x and y lists to 20 items
    xs = xs[-20:]
    ys = ys[-20:]

    # Draw x and y lists
    ax.clear()
    ax.plot(xs, ys)

    # Format plot
    plt.xticks(rotation=45, ha='right')
    plt.subplots_adjust(bottom=0.30)
    plt.title('Time (Hours:Minutes:Seconds)')
    plt.ylabel('Temperature (deg C)')

# Set up plot to call animate() function periodically
ani = animation.FuncAnimation(fig, animate, fargs=(xs, ys), interval=1000)
plt.show()
```

Figure 3: Modified baseline code that displays CPU temperature in real-time.

The resulting python script allows us to see SoC temperature in real-time. Below is a snapshot of the temperature over-time plot:

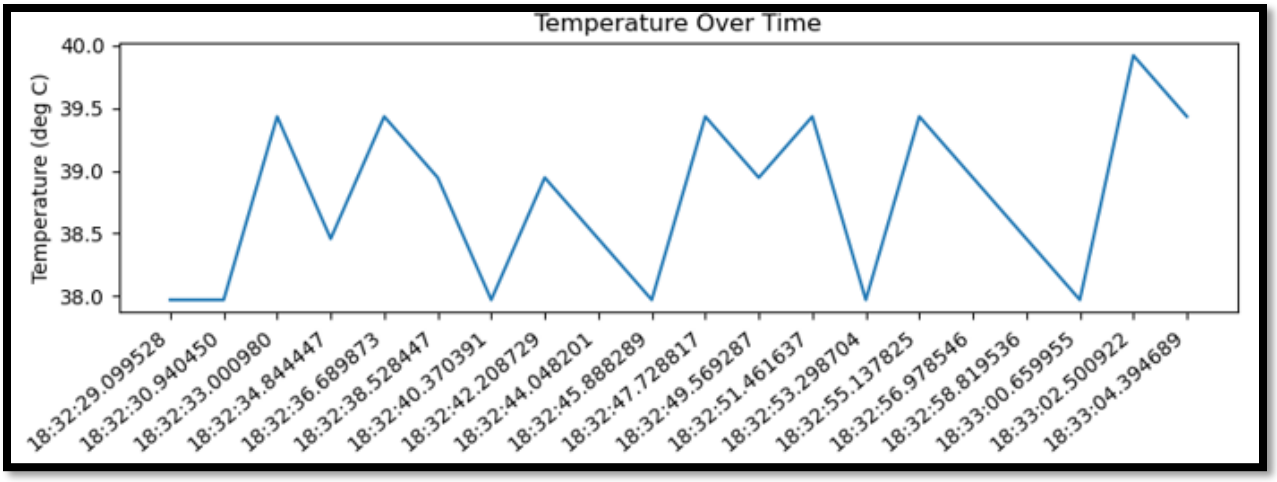


Figure 4: Snapshot of temperature over-time plot.

Enhance: CPU Utilization Measurement Graph in Real-Time

The resulting python script from the previous section was enhanced to measure the CPU utilization percentage and to display the CPU utilization over-time.

There are many ways to measure the CPU utilization of the Raspberry Pi. There is a very good tutorial written by Tamiko Lattimore which can be found here: https://robots.net/tech/how-do-i-check-my-cpu-usage-on-raspberry-pi/. This tutorial goes over many ways in which we can measure the CPU utilization on the Raspberry Pi. For this particular project, I was advised to use Method 5 described in Tamiko's tutorial.

I followed the tutorial's steps to be able to get the CPU utilization. I first imported the Python's psutil module. However, unlike the tutorial, I did not specify an interval when calling 'psutil.cpu_percent()'. I also added a new plot to the existing figure to display the graph of the CPU utilization over time.

The contents of the new python script with the additional code to measure and display the CPU utilization are shown here:

```python
# Starter code based on https://projects.raspberrypi.org/en/projects/temperature-log
# and https://github.com/raspberrypilearning/temperature-log
#modified by Loyda Yusufova 2/6/2024

from gpiozero import CPUTemperature
from time import sleep, strftime, time
import matplotlib.pyplot as plt
import datetime as dt
import matplotlib.animation as animation
import psutil
import os

# Create figure for plotting
fig = plt.figure()
temp_plt = fig.add_subplot(2, 1, 1)
util_plt = fig.add_subplot(2, 1, 2)

xtime = []
ytemp = []
yutil = []


plt.subplots_adjust(top=0.97, bottom=0.14, left=0.30, right=0.7, hspace=0.7,wspace=0.035)

# Initialise communication cpu_temperature
cpu = CPUTemperature()

# This function is called periodically from FuncAnimation
def animate(i, xtime, ytemp, yutil, yfreq):
    # Add x to lists
    xtime.append(dt.datetime.now().strftime('%H:%M:%S.%f'))

    #-----------------------------------------------------------------------
    # Read temperature (Celsius) from gpiozero
    temp=cpu.temperature
    ytemp.append(temp)

    # Limit x and y lists to 20 items
    xtime = xtime[-20:]
    ytemp = ytemp[-20:]

    # Draw x and y lists
    temp_plt.clear()
    temp_plt.plot(xtime, ytemp)

    # Format plot
    for label in temp_plt.get_xticklabels():
        label.set_rotation(40)
        label.set_horizontalalignment('right')
    temp_plt.set_title('Temperature Over Time')
    temp_plt.set_ylabel('Temperature (deg C)')

    #-----------------------------------------------------------------------
    cpu_usage = int(psutil.cpu_percent())

    # Add y to lists
    yutil.append(cpu_usage)

    # Limit x and y lists to 20 items
    yutil = yutil[-20:]

    # Draw x and y lists
    util_plt.clear()
    util_plt.plot(xtime, yutil)

    # Format plot
    for label in util_plt.get_xticklabels():
        label.set_rotation(40)
        label.set_horizontalalignment('right')
    util_plt.set_title('CPU Utilization Over Time')
    util_plt.set_ylabel('CPU Utilization (percentage)')

# Set up plot to call animate() function periodically
ani = animation.FuncAnimation(fig, animate, fargs=(xtime, ytemp, yutil), interval=5000)
plt.show()
```

Figure 5: Modified baseline code that displays CPU temperature and utilization in real-time.

The updated plot figure now displays the CPU utilization in a separate plot. The screenshot is shown here:

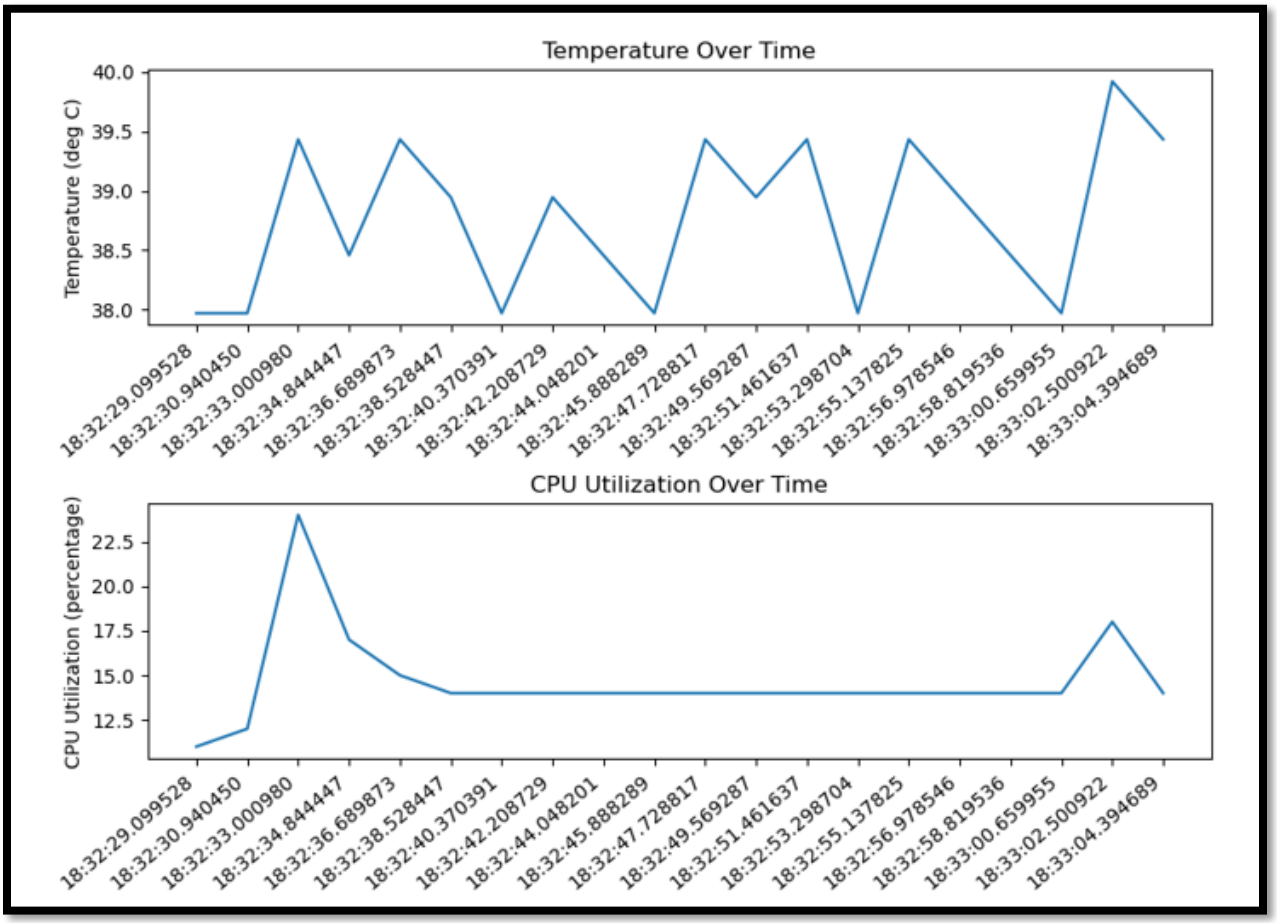


Figure 6: Snapshot of temperature and CPU utilization plots over-time.

Enhance: CPU Frequency Measurement Graph in Real-Time

The resulting python script from the previous section was enhanced once again, this time to measure the CPU frequency and to display the CPU frequency over-time.

To be able to read the CPU frequency, I use the Python's subprocess module. I use 'os.popen()' to create a new child process. This child process then runs a command to read the frequency information of the processor. The program used to read the frequency information is 'cpufreq-info' which can be executed after installing the 'cpufrequtils' utility. This utility is very useful as it provides many useful programs to control and monitor the frequency of all the CPU cores of the raspberry Pi. For instance, it provides a program to modify the frequency scaling daemon of the Raspberry Pi. There is also a program that allows the user to check the status of the frequency of each CPU core of the Raspberry Pi.

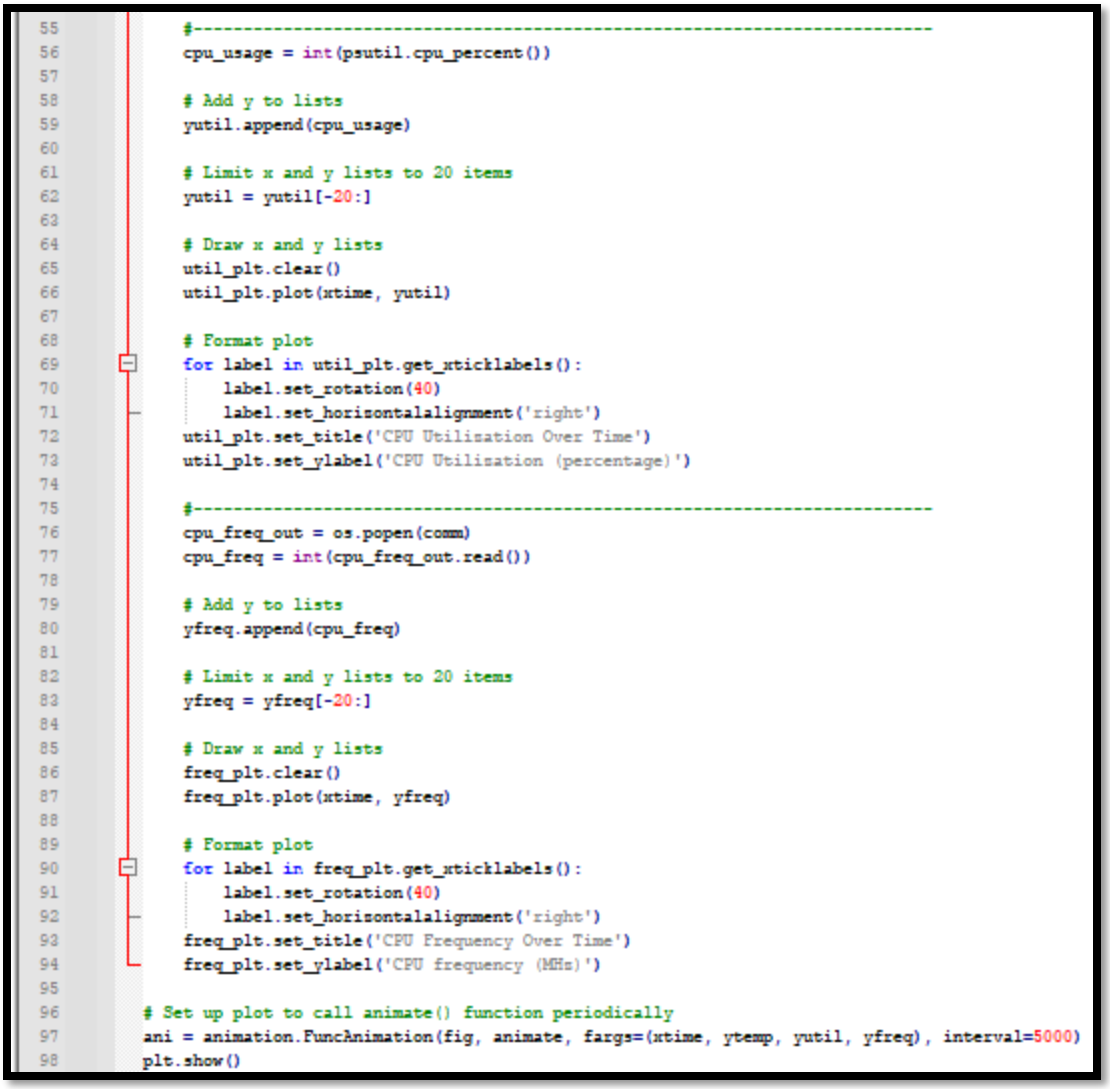
In this particular project, I needed to extract the frequency information of the CPU cores of the Raspberry Pi. So, I used the 'cpufreq-info' program. One thing to mention is that the raspberry Pi 4 has 4 cores, and, at any given time, they all are running at the same frequency. Therefore, I only needed to extract the frequency of one of the CPU cores. I randomly chose to read the CPU frequency information of core number 0 which I specified as follows: "cpufreq-info -f -c 0".

To summarize, I let os.open() execute the command “cpufreq-info -f -c 0” which returned the CPU frequency of CPU core number 0. The frequency was then appended to a list containing a list of frequencies. These frequencies are then plotted such as in the case of the temperature and CPU utilization. The plot is added to the existing plot figure.

The following image is a screenshot of the final Python script:

```
# Starter code based on https://projects.raspberrypi.org/en/projects/temperature-log
# and https://github.com/raspberrypilearning/temperature-log
#modified by Loyda Yusufova 2/6/2024

from gpiozero import CPUTemperature
from time import sleep, strftime, time
import matplotlib.pyplot as plt
import datetime as dt
import matplotlib.animation as animation
import psutil
import os

# Create figure for plotting
fig = plt.figure()
temp_plt = fig.add_subplot(3, 1, 1)
util_plt = fig.add_subplot(3, 1, 2)
freq_plt = fig.add_subplot(3, 1, 3)
xtime = []
ytemp = []
yutil = []
yfreq = []

plt.subplots_adjust(top=0.97, bottom=0.14, left=0.30, right=0.7, hspace=0.7,wspace=0.035)

# Initialize communication cpu_temperature
cpu = CPUTemperature()

comm = 'cpufreq-info -f -c 0'

# This function is called periodically from FuncAnimation
def animate(i, xtime, ytemp, yutil, yfreq):
    # Add x to lists
    xtime.append(dt.datetime.now().strftime('%H:%M:%S.%f'))

    #-----------------------------------------------------------------------
    # Read temperature (Celsius) from gpiozero
    temp=cpu.temperature
    ytemp.append(temp)

    # Limit x and y lists to 20 items
    xtime = xtime[-20:]
    ytemp = ytemp[-20:]

    # Draw x and y lists
    temp_plt.clear()
    temp_plt.plot(xtime, ytemp)

    # Format plot
    for label in temp_plt.get_xticklabels():
        label.set_rotation(40)
        label.set_horizontalalignment('right')
    temp_plt.set_title('Temperature Over Time')
    temp_plt.set_ylabel('Temperature (deg C)')

```

Figure 7: First part of final Python script.



```
55          #--------------------------------------------------------------------------
56          cpu_usage = int(psutil.cpu_percent())
57
58          # Add y to lists
59          yutil.append(cpu_usage)
60
61          # Limit x and y lists to 20 items
62          yutil = yutil[-20:]
63
64          # Draw x and y lists
65          util_plt.clear()
66          util_plt.plot(xtime, yutil)
67
68          # Format plot
69          for label in util_plt.get_xticklabels():
70              label.set_rotation(40)
71              label.set_horizontalalignment('right')
72          util_plt.set_title('CPU Utilization Over Time')
73          util_plt.set_ylabel('CPU Utilization (percentage)')
74
75          #--------------------------------------------------------------------------
76          cpu_freq_out = os.popen(comm)
77          cpu_freq = int(cpu_freq_out.read())
78
79          # Add y to lists
80          yfreq.append(cpu_freq)
81
82          # Limit x and y lists to 20 items
83          yfreq = yfreq[-20:]
84
85          # Draw x and y lists
86          freq_plt.clear()
87          freq_plt.plot(xtime, yfreq)
88
89          # Format plot
90          for label in freq_plt.get_xticklabels():
91              label.set_rotation(40)
92              label.set_horizontalalignment('right')
93          freq_plt.set_title('CPU Frequency Over Time')
94          freq_plt.set_ylabel('CPU frequency (MHz)')
95
96      # Set up plot to call animate() function periodically
97      ani = animation.FuncAnimation(fig, animate, fargs=(xtime, ytemp, yutil, yfreq), interval=5000)
98      plt.show()
```

Figure 8:Second part of final Python script.

The updated plot figure now displays the CPU frequency in a separate plot. The screenshot of the plot figure is shown here:

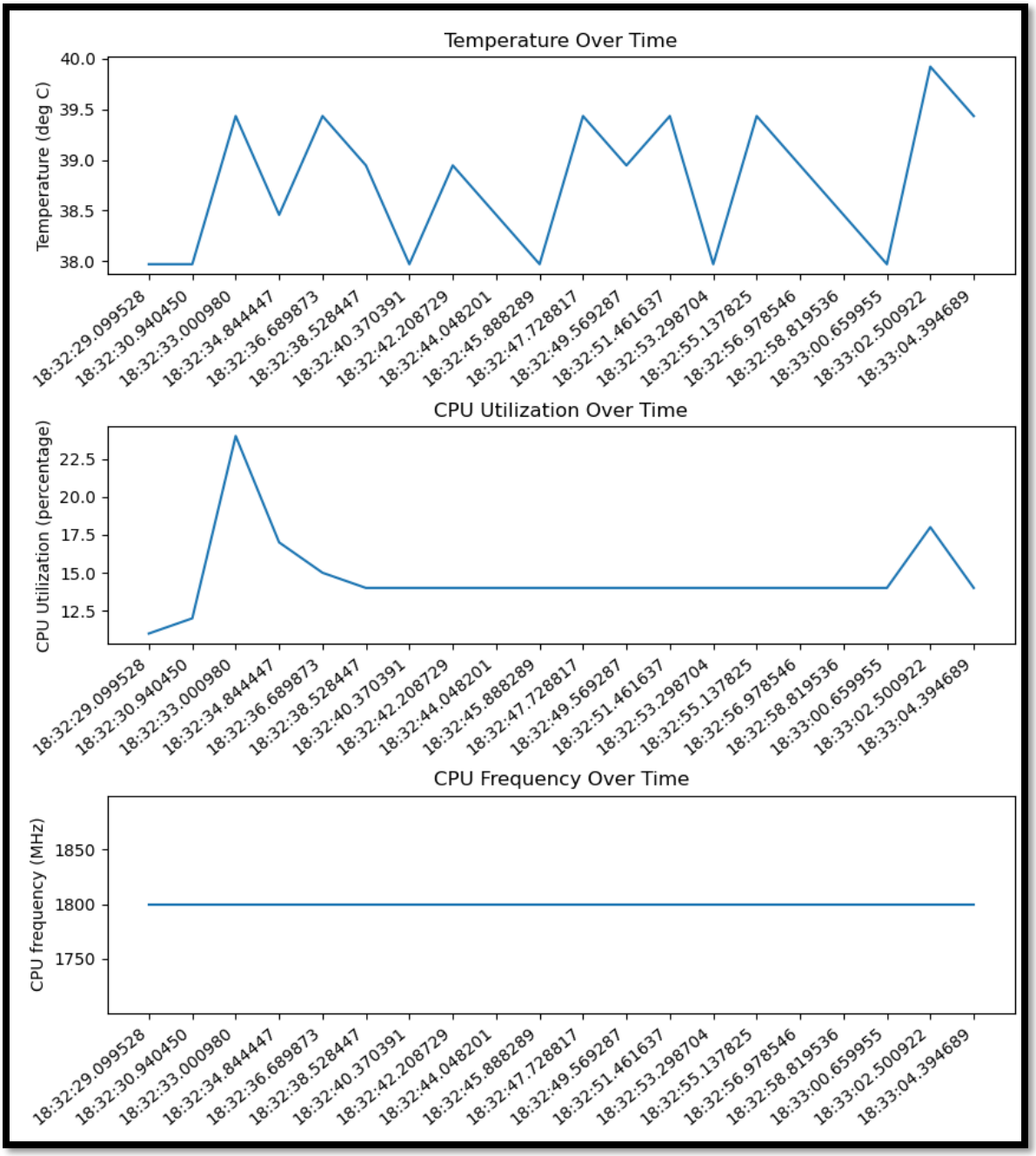


Figure 9: Snapshot of final figure with temperature, CPU frequency and utilization plots.

Conclusion: Monitor Program

This resulting python script is useful for monitoring the temperature, CPU utilization and frequency of the Raspberry Pi in real-time. This script can be run in the background. The plot figure will be updated every 5 seconds which is specified inside the call to animation.FuncAnimation().The script opens a new window in which it displays the 3 plots. The plots can then be observed to monitor the temperature, CPU utilization and frequency.