ANALYSIS OF LIVE-PLOTTING SENSOR READINGS WITH RASPBERRY PI

Plasticity Model: Team 8

Dillon C. Edlund

4-7-19

Introduction

The problem posed to the team was to design and build a model which could replicate the region of plastic deformation of a material using a block and spring. This model will be used in a classroom for educational purposes. Modeling and teaching plasticity theory is notoriously difficult and hard to depict. To the client's best knowledge, and the team's research, no such model has been created to aid in teaching this subject. The model theory is based on the relationship between plastically deformed regions of a material measured by Stress versus Strain. The plots of stress vs. strain can be accurately replicated using force and displacement plots. One component of the solution to this problem involves the ability to obtain force and displacement readings from the movement of the block and to display them in a graph, replicating the curves of stress versus strain plot during plastic deformation. This analysis will focus on the use of a Raspberry Pi system running Python script to read, record, and display sensor data in real-time, thus satisfying the respective engineering requirement.

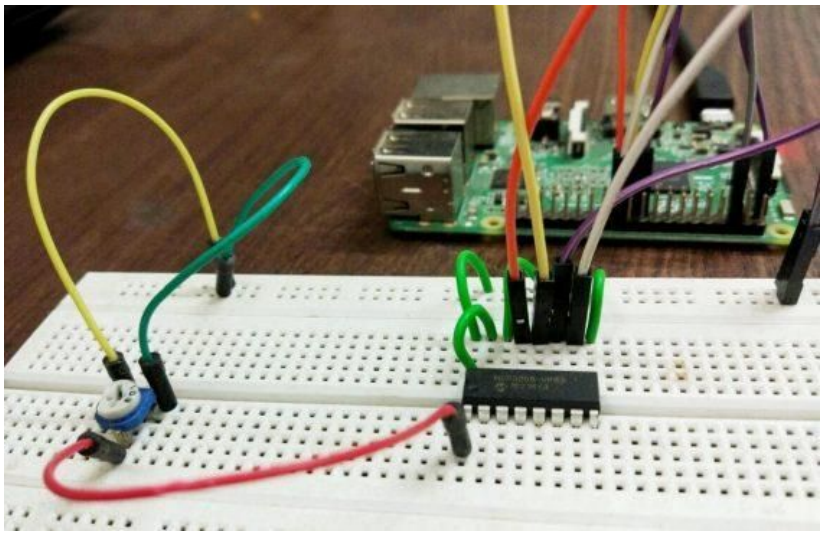
Assumptions

The design the team has chosen will use a potentiometer to measure displacement of the block within the plane. A specific potentiometer has not been chosen at the time of this analysis, so the Python script that was written is based on the code used to read an arbitrary analog sensor. The use of an analog potentiometer was assumed based on the team's development of design to date but will possibly be altered in the future. The code format can be easily modified in the event a different type of sensor is chosen.

- Theoretical Potentiometer
- Analog Sensor
- Display data in real-time (as opposed to record-then-display)
- Code is written to display Displacement Vs. Time but will be updated in the future to display Force Vs. Displacement when model constants are known.

Schematic

The potentiometer will be wired to the Raspberry Pi and will run through an Analog to Digital Convertor (ADC) chip. This will allow the analog output of the potentiometer to be read by the digital GPIO inputs on the Raspberry Pi. A depiction like that which will be implemented is shown below.



Source: https://radiostud.io/sensing-analog-signal-raspberrypi/

The potentiometer will likely be mounted directly to the block tracking system. The ADC chip and the Raspberry pi will be mounted on the outside of the model itself.

Python code

The language used to program the Raspberry Pi is Python which comes pre-loaded on the Pi. To use any analog sensors with the Raspberry Pi the Serial Peripheral Interface (SPI) bus port must be enabled [1]. SPI is a synchronous serial communication interface and is the method that the Pi will use to communicate with the sensor [2]. Once the SPI bus is enabled, the code can be written.

To integrate the sensor with Python, the sensor's specific "library" must be downloaded to the Pi. Once downloaded it can be called in the code using the "import" function in Python [3]. The program is then written to save the incoming data from the sensor as a Comma Separated Value (CSV) file which will be called later to plot the data.

```
import " " #Import Specific Potentiometer Library (NOT YET CHOSEN!)
import time
import csv  # save pot data to csv file (comma seperated value)
import sys
csvfile = "displacement.csv"  #CSV file name to call for plotting
als = True # This loop is always True
```

A "while" loop is iterated to set parameters for collecting input from the potentiometer. This loop is set to always be true. This is also where the specific pin of the GPIO where the potentiometer is connected is indicated. This tells the Raspberry Pi where to look for the potentiometer. The incoming values of voltage are rounded to 2 decimal points and are "printed" in increments of 0.01 inches. If there is an unusual sensor reading, or a reading outside of the normal voltage range the program will display "no sensor reading". The data is then cataloged in columns, the first as distance and the second as time.

```
while als:
    displacement = " # pot sensor".read_retry("potsensor", "pin number") #  GPIO pin used
    if displacement is not None
        displacement = round(displacement,2) # rounding distance measurment to 2 decimal places
        print 'Displacement = {0,0.01f}inch'.format(displacement)
    else:
        print 'no sensor reading'  # verify that the sensor is reading properly
    timeD = time.strftime("%I")+':' +time.strftime("%M")+':'+time.strftime("%S") # display time of readings
    data = [displacement, timeD]

    with open(csvfile, "a")as output:   # saving data to csv file as column of values to be imported later
        writer = csv.writer(output, delimiter=",", lineterminator = '\n')
        writer.writerow(data)
    time.sleep(0.25)  # the script will be updated every 0.25 seconds, may need to be adjusted later
```

In the line beginning "displacement =..." the entries "# pot sensor", "potsensor", and "pin number" are placeholders which will be replaced/updated once a specific potentiometer is chosen and installed.

The CSV file with the displacement ant time data is then transitioned to an output which will be continuously called by the code that will plot the data. The program will update with new sensor data every 0.25 seconds. This value was chosen arbitrarily and will likely be updated when prototype testing takes place.

```
with open(csvfile, "a")as output:   # saving data to csv file as column of values to be imported later
    writer = csv.writer(output, delimiter=",", lineterminator = '\n')
    writer.writerow(data)
time.sleep(0.25)  # the script will be updated every 0.25 seconds, may need to be adjusted later
```

This is the end of the program for collecting and saving the potentiometer data. A second program was written to run concurrently with this program that will actively plot the data as it is updated in the first program. The program that will be used to plot the data is matplotlib. It is an open source data analytics program suited well for this project. The program initiates with importing the matplotlib software as well as the formatting options package. The animate function is defined. This tells the plot to update in real time, creating a moving curve. The CSV file is then called to import the displacement and time data.

```
import matplotlib.pyplot as plt  # importing the Matplotlib software to plot data
import matplotlib.dates as mdates
import matplotlib.animation as animation # importing stylization and formatting
from datetime import datetime

fig = plt.figure()
rect = fig.patch
rect.set_facecolor('#0079E7') # designating color of plot
def animate(i):
    fdisp = 'displacement.csv' # pulling data from the sensor data saved to CSV file
    fh = open(fdisp) # open CSV file
    disp = list() # list displacement values
    timeD = list() # list corresponding time values
```

A for loop is then initiated for formatting, labeling, and categorizing the incoming data from the CSV file. It also scales the values for graphing.

```
for line in fh # formating for each line in the listed values above
    pieces = line.split(',')
    inch = pieces[0] # labeling displacement values with unit
    timeB = pieces[1]
    timeA = timeB[:8] # labeling/scaling time values
        #print timeA
    time_string = datetime.strptime(timeA, '%H:%M:%S')
        #print time_string
    try:
        disp.append(float(inch)) # updates the values in the CSV file and plot (live plotting)
        timeD.append(time_string) # updates values in the time file
    except:
        pring "Not Sure" # if files not reading correctly display not sure
```

The final section of code is formatting the plot, labeling the axes, scaling the axes, and printing the plot. It is also calling the animation function of matplotlib.

```
        ax1 = fig.add_subplot(1,1,1,axisbg='white')  # formatting the x-axis
        ax1.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M:%S'))
        ax1.clear()
        ax1.plot(timeD,disp, 'in.', linewidth = 3) # labeling axis
        plt.title('Displacement')
        plt.xlabel('Time')

ani = animation.FuncAnimation(fig, animate, interval = 6000)

plt.show() # plot the graph
```

Once the code is finalized it will be made executable. This means that the end user will be able to start and run both programs, and therefore the model, by double clicking one icon on the home screen. The entirety of the code can be found in the appendix.

Results

Unfortunately, without the potentiometer and other hardware available the code has not been tested fully. Although debugging was done, and no syntax errors were found this program and its code were written based on the best available resources at the time. This means there will likely be updates and tweaking to the code once hardware is chosen and assembled. The process of writing the program brought to light several considerations which need to be discussed. The team needs to decide whether to use a digital or analog potentiometer as this will affect the coding process and the budget. It will also need to decide on whether to measure force directly with a strain gauge or to calculate force based on the displacement data alone.

If the program runs as it should, it would complete a major component of the project. It would allow the client to plug the Raspberry Pi directly into the projector in the classroom, run the program, and display the Force Vs. Displacement plots for the class to see in real-time as the move the block. The program

format is such that it would also allow the team to add sensors in the future, such as strain gauges, and display the force and displacement data directly.

Bibliography

[1] Takaitra, "Controlling an SPI device with the Raspberry Pi," Takaitra.com, 20 July 2018. [Online]. Available: https://www.takaitra.com/spi-device-raspberry-pi/. [Accessed 6 April 2019].

[2] Anonymous, "Serial Peripheral Interface," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface. [Accessed 7 April 2019].

[3] L. Ada, "SPI Sensors & Devices," Adafruit, 30 June 2018. [Online]. Available: https://learn.adafruit.com/circuitpython-on-raspberrypi-linux/spi-sensors-devices. [Accessed 7 April 2019].

[4] S. Igor, "Sensing Analog Signal using Raspberry Pi," Radio Studio, 29 November 2018. [Online]. Available: https://radiostud.io/sensing-analog-signal-raspberrypi/. [Accessed 7 April 2019].

[5] Python Packaging Authority , "Python Packaging Authority," Python Software Foundation, 20 March 2019. [Online]. Available: https://www.pypa.io/en/latest/. [Accessed 7 April 2019].

Appendix

Python Code for Program 1: Saving Data from Potentiometer

```
# Code to record displacement data and plot it vs. time using matplotlib.  By: Dillon Edlund


import " " #Import Specific Potentiometer Library (NOT YET CHOSEN!)
import time
import csv  # save pot data to csv file (comma seperated value)
import sys
csvfile = "displacement.csv"  #CSV file name to call for plotting
als = True # This loop is always True

while als:
    displacement = " # pot sensor".read_retry("potsensor", "pin number") #  GPIO pin used
    if displacement is not None
        displacement = round(displacement,2) # rounding distance measurment to 2 decimal places
        print 'Displacement = {0,0.01f}inch'.format(displacement)
    else:
        print 'no sensor reading'  # verify that the sensor is reading properly
    timeD = time.strftime("%I")+':' +time.strftime("%M")+':'+time.strftime("%S") # display time of readings
    data = [displacement, timeD]

    with open(csvfile, "a")as output:   # saving data to csv file as column of values to be imported later
        writer = csv.writer(output, delimiter=",", lineterminator = '\n')
        writer.writerow(data)
    time.sleep(0.25)  # the script will be updated every 0.25 seconds, may need to be adjusted later
```

Python Code for Program 2: Displaying Potentiometer Data in Real-Time Plot (matplotlib)

```
#Script to read data from the CSV file and display it in a real-time plot

import matplotlib.pyplot as plt  # importing the Matplotlib software to plot data
import matplotlib.dates as mdates
import matplotlib.animation as animation # importing stylization and formatting
from datetime import datetime

fig = plt.figure()
rect = fig.patch
rect.set_facecolor('#0079E7') # designating color of plot
def animate(i):
    fdisp = 'displacement.csv' # pulling data from the sensor data saved to CSV file
    fh = open(fdisp) # open CSV file
    disp = list() # list displacement values
    timeD = list() # list corresponding time values
    for line in fh # formating for each line in the listed values above
        pieces = line.split(',')
        inch = pieces[0] # labeling displacement values with unit
        timeB = pieces[1]
        timeA = timeB[:8] # labeling/scaling time values
            #print timeA
        time_string = datetime.strptime(timeA, '%H:%M:%S')
            #print time_string
        try:
            disp.append(float(inch)) # updates the values in the CSV file and plot (live plotting)
            timeD.append(time_string) # updates values in the time file
        except:
            pring "Not Sure" # if files not reading correctly display not sure
```

```
        ax1 = fig.add_subplot(1,1,1,axisbg='white')  # formatting the x-axis
        ax1.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M:%S'))
        ax1.clear()
        ax1.plot(timeD,disp, 'in.', linewidth = 3) # labeling axis
        plt.title('Displacement')
        plt.xlabel('Time')

ani = animation.FuncAnimation(fig, animate, interval = 6000)

plt.show() # plot the graph
```