Developing a Data Acquisition System for use in Cold Neutral Atom Traps

A Senior Project

presented to

the Faculty of the Physics Department

California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

of the Requirements of the Degree

Bachelor of Science

Physics

by

Jonathan Fuzaro Alencar

June 2022

Table of Contents

Table of Figures

Abstract

The rising interest in quantum computing has led to new quantum systems being developed and researched. Among these are trapped neutral atoms which have several desirable features and may be configured and operated on using lasers in an optical lattice. This work describes the development of a new data acquisition system for use in tuning lasers near the precise hyperfine transition frequencies of ^{87}Rb atoms, a crucial step in the functionality of a neutral atom trap. This improves on previous implementations that were deprecated and limited in laser frequency sweep range. Integration into the experiment was accomplished using an Arduino microcontroller and Python for real-time data acquisition and visualization.

1. Introduction

A quantum computer uses properties of quantum systems such as superposition and entanglement to perform a select number of calculations faster than a conventional classical computer. Like a classical computer uses circuitry to retain information in the form of binary digits (bits), a quantum computer requires a system that encapsulates information as quantum bits (qubits) that are manipulated with quantum algorithms to perform a desired calculation. The criteria that make a quantum computer have been identified [1], some of which are ideally satisfied by neutral atoms where scalability and long coherence times (stability) are achievable [2].

Neutral atom quantum computing experiments may realize a qubit as the hyperfine ground states of the valence electron in alkali atoms. For example, atoms of Rubidium-87 (^{87}Rb) may be placed in an optical lattice which would form an ensemble of qubits that may be used as a quantum computer. To achieve, this ^{87}Rb atoms must first be cooled significantly and localized to a central point in space where they may then be trapped in the optical lattice. Cooling and localizing are experimentally accomplished using Doppler cooling and the Zeeman effect in a magneto-optical trap (MOT) [3]. This experiment attempts to trap ^{87}Rb atoms in a MOT

Paramount to the function of both the optical lattice and magneto-optical trap is a precise tuning of laser light which must first be accomplished using a dichroic-atomic vapor laser lock (DAVLL) and saturated absorption spectroscopy to determine the laser frequency relative to the hyperfine transition frequencies of ^{87}Rb. Prior electronics and software comprising the data acquisition (DAQ) configuration were limited in the range of its laser frequency sweep and had become deprecated by out-of-

date device drivers. This work describes the development of a new DAQ and visualization framework implemented using the Arduino microcontroller and the Python programming language.

2. Theoretical Background

2.1. The Magneto Optical Trap

The MOT is a device that uses light (lasers) and magnetic fields to trap and cool neutral atoms in a vacuum chamber. The MOT uses a process called Doppler cooling which exploits the Doppler effect to lower the temperature of an atomic specimen. Magnetic fields are used to exploit the Zeeman effect coupled with radiation pressure from polarized laser light to aggregate atoms toward a central location. In conjunction, these two processes allow the MOT to reduce the temperature of an atomic specimen to very low temperatures in a centralized location within the MOT chamber.

Doppler cooling makes use of the relativistic doppler effect which occurs when the frequency of light changes due to the relative motion of the interacting atom. When light is incident on an atom that is moving towards it, the Doppler effect indicates that the light will become blueshifted in the atom's reference frame. Likewise, light incident on an atom that is moving away from it will appear redshifted in the atom's reference frame. To induce radiation pressure on an atom that is not moving, the laser frequency must be tuned near the excitation frequency of the atom. Therefore, to slow down an atom that is moving, incident laser light must be red-detuned slightly below resonance to induce pressure against its axis of motion. This idea may apply to all three axes of motion with six lasers, two counterpropagating lasers for each axis. The result

is an optical molasses that cools atoms within the intersecting laser beams using the trap transition shown in Figure 1.

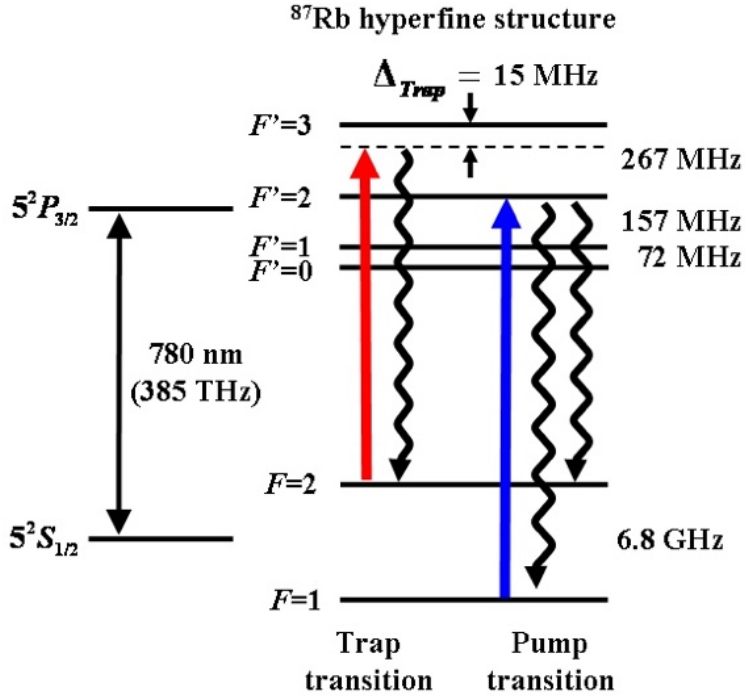

Figure 1: Pump and trap transitions of ^{87}Rb. The MOT uses the pump transition to optically pump ^{87}Rb to the cycling trap transition where it may be cooled [2].

The MOT also consist of anti-Helmholtz coils that create a quadrupole magnetic field in the center of the vacuum chamber. Near the center of the chamber, the magnetic field strength varies linearly in space and induces splitting in the atomic energy levels of ^{87}Rb by the Zeeman effect. Figure 2 displays the configuration in 1 dimension; due to atomic transition rules, atoms that deviate from the center of the trap will have their energy levels shifted closer towards the frequency of polarized

laser light that points inward. This will induce a converging radiation pressure that will drive ^{87}Rb atoms towards the center of the vacuum chamber.

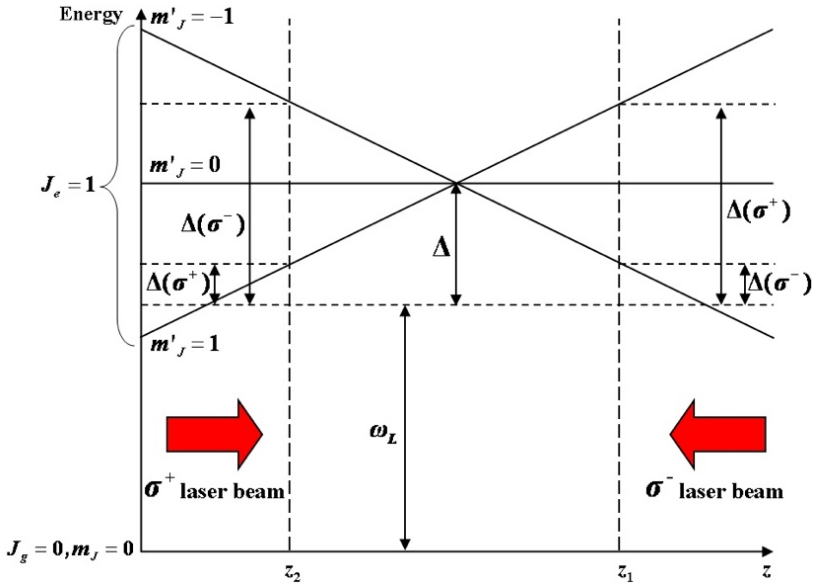

Figure 2: Configuration of Zeeman shift due to MOT in 1D. Atoms that deviate from the center will have their energies shifted to interact with the corresponding beam pointing inwards (σ_+ or σ_- circularly polarized) [2].

2.2. Laser Spectroscopy & Tuning

Together, both Doppler cooling and the polarized light radiation on the Zeeman shifted atoms will work to cool and collect atoms in the MOT. Subsequently, an optical lattice may then be configured by loading the atoms into dipole traps. Critical to this entire process is the precise tuning of laser light towards the hyperfine transition frequencies of ^{87}Rb shown in Figure 1. This experiment uses a near infrared (NIR) external cavity diode laser (ECDL) utilizing the Littrow configuration for mechanical feedback as shown in Figure 3. In this configuration, the diffraction grating serves as a

retro-reflector and frequency tuner where the first order diffracted beam is reflected back into the laser diode. The angle of the diffraction grating is controllable via a piezoelectric transducer (PZT) which results in changing the seed laser frequency and therefore ECDL output frequency. By applying a voltage ramp to the grating PZT, the laser frequency may thus be swept.

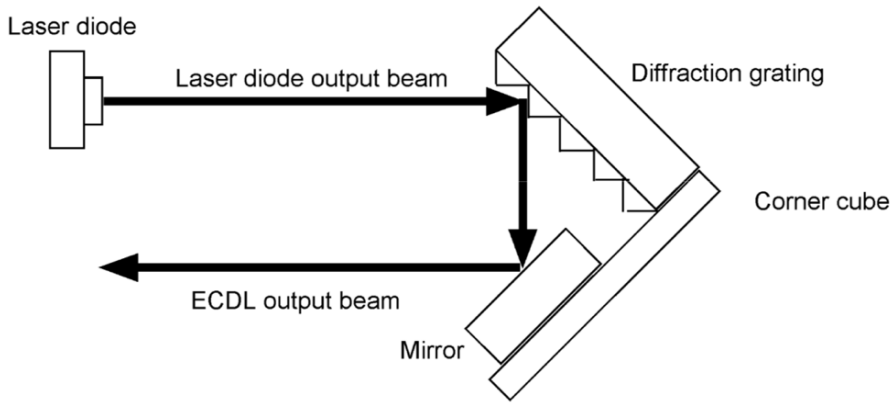

Figure 3: Littrow configuration of ECDL Laser [4].

The laser is locked to a reference frequency using a dichroic-atomic-vapor laser lock (DAVLL). The DAVLL functions by measuring the intensity of linearly polarized laser light which passes through a Rb vapor cell subject to a magnetic field pointing along the beam axis. The light is a superposition of right (σ_+) and left (σ_-) circularly polarized light and due to a Zeeman shift in the Rb atoms by the magnetic field, the σ_+ and σ_- light is absorbed at different frequencies. This light subsequently passes through a quarter-wave plate and then beam splitter producing two beams, each consisting of the former σ_+ and σ_- light of which are fed into separate photodiodes (PD) used to measure light intensity. If the laser frequency is swept, the result is two

roughly identical Doppler broadened absorption signals each separated by an amount relative to the magnetic field strength. These two absorption signals are subtracted electronically to produce an error signal whose zero point represents the intersection of the two peaks and a good reference point for grating locking. The error signal also provides a useful reference point from which to observe other signals and as such, is one of the streams of data collected by the DAQ [2].

To resolve precise hyperfine transitions of ^{87}Rb that are indistinguishable in the Doppler broadened signals, this experiment utilizes saturated absorption (SA) spectroscopy. This technique functions by applying counter-propagating beams from the same laser, a “pump” and “probe” beam, into a Rb vapor cell. The pump beam is much more intense than the probe beam, and when tuned near resonance, will excite a group of atoms moving along its axis because of the Doppler effect. Similarly, a different group of atoms moving in the opposite direction will be excited by the probe beam. When the laser is tuned precisely on a hyperfine transition frequency, the pump and probe beams will now interact with the same group of atoms, i.e., those moving perpendicular to the beam axis. The pump beam optically pumps and thus saturates this group resulting in a lack of absorption from the probe beam. If the laser frequency is swept and the probe beam measured with a PD, hyperfine transitions will appear as small peaks in probe beam intensity. The probe beam intensity is relatively small and a lock-in amplifier is needed to amplify the PD signal. The frequency at which to amplify the pump beam is modulated using an optical chopper. The saturated absorption photodiode provides crucial information regarding the laser frequency and is thus another stream of data actively collected by the DAQ during frequency sweeps [2].

In addition to the DAVLL error and SA signals, the experiment also utilizes a Fabry-Perot (FP) interferometer. The FP is used by measuring the transmission intensity with a PD across the FP cavity. This provides information on when the laser light undergoes resonance with the cavity and is useful in distinguishing cavity mode hops during a sweep. FP signals will consist of single sharp peaks at a free spectral range 300 MHz apart.

Together, signals from the DAVLL error, saturated absorption PD, and Fabry-Perot PD provide the necessary data for precise laser characterization, tuning, and feedback. This project developed a new data acquisition framework for acquiring, saving, and visualizing data in real-time during laser frequency sweeps and is discussed in the next section.

3. Development of New DAQ System

Previous implementation of the data acquisition system utilized the National Instruments (NI) PCI-6014 multifunction DAQ. The necessary replacement of ramping electronics that could easily interface to a newer and more flexible DAQ system resulted in the development of a new DAQ implementation using the Arduino Portenta H7 microcontroller and Python programming language. The Arduino Portenta was selected for its ease of use and ability to recreate the grating PZT ramp voltage using its built-in digital to analogue converter (DAC). Likewise, Python was selected for its ease of use, and mature package ecosystem.

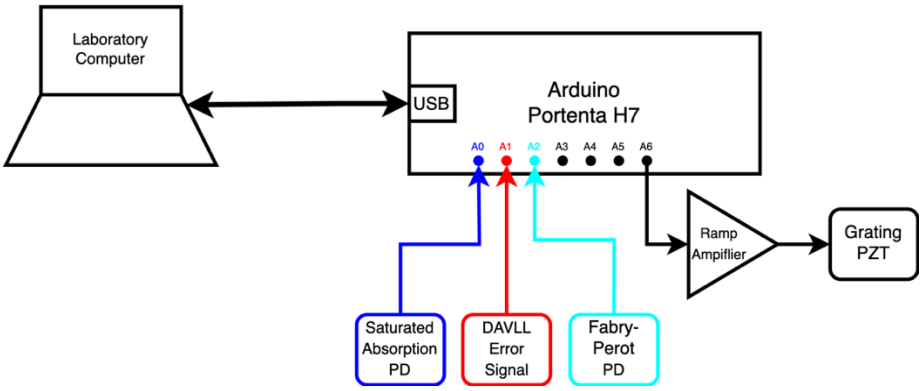

Figure 4: Simplified Arduino DAQ configuration with labeled pin connections. The Portenta H7 pin connections are highly simplified.

The DAQ Arduino configuration is outlined in the diagram in Figure 4. Here the Arduino is responsible for mediating input and output (I/O) between the laboratory computer and optical system including the lasers and PDs. The pinout is simplified showing only analog pins labeled A0 through A6. The analog pins are capable of reading voltage signals ranging from 0 to 5 volts which are converted into digital (binary) form via the built-in analog to digital converter (ADC). The built-in ADC defaults to a 10-bit read resolution but may be configured higher on the Portenta H7 up to 16 bits [5]. This allows for a theoretical measuring resolution of $\frac{5}{2^{16}}$ V though in practice it is likely much higher. The Portenta H7's built-in DAC may be accessed exclusively on the A6 pin and is configured as the ramp voltage output that is amplified and sent to control the grating PZT for laser frequency sweeps. The Arduino is powered using the laboratory computer using a USB connection (USB-C on the Portenta H7) though may also be powered externally.

In order to read data collected on the Arduino and issue commands related to data acquisition or the grating PZT ramp, the computer and Arduino must have a

means of communication. This is currently accomplished with the USB connection to serve as a port for serial communication. On the lab computer, software written in Python is run to interface with the serial port device (USB) drivers using the pySerial package [6]. The speed of serial communication is limited by baud (bits/s), which for serial to USB is typically 115200 baud (~14 kB/s). Thus, in tandem with the Arduino, Python software on the computer uses pySerial to save signal data fed through the serial port and plots it in real time.

The visualization of data in real-time is accomplished using Python's matplotlib package [7]. Matplotlib is typically used for generating and displaying figures of data plots. Real-time plotting is accomplished using the package's Animation class which facilitates the process of repeatedly drawing and updating plot figures. A figure in matplotlib is typically distinguished by its own desktop window. A figure can consist of several axes that may either share a common axis or use their own axis to yield several subplots on one figure. For laser frequency sweeps, it is desirable to have the DAVLL error and SA PD signals share an axis on a single figure, with their signal voltages plotted against the ramp voltage. A second figure consisting of the FP PD signal voltage is also plotted against the ramp voltage. The result are two figures with all three signals plotted in real time against the ramping voltage of the grating PZT.

The appendix contains a functional Python script (.py file) making use of the packages and techniques just described. It also contains corresponding simplified Arduino C++ (.ino file) code used for reading and streaming data across the serial port. The Python script works by defining a Python class called "serialPlot" to facilitate data collection, timing, and threading. In essence, the serialPlot class initializes by opening the serial port and creating a background thread (process) that actively saves data that

is streamed in by the Arduino while the program is running. In the “main” function, the program configures the figures to be used for plotting and attempts to automatically detect which serial port the Arduino is connected to. It then instantiates a serialPlot object to be used with the animation module from matplotlib to begin streaming and plotting data. Before the program is ended, the program waits for the background thread to terminate and then finally saves all collected data in the form of a csv file. Up to date source code for these files may be found in [8]. Examples of recorded SA and FP PD signals plotted against a grating PZT voltage sweep are shown in Figure 5 and Figure 6 respectively. The peaks in the SA signal correspond to locations at which the grating (laser frequency) is at a Rb hyperfine transitions.

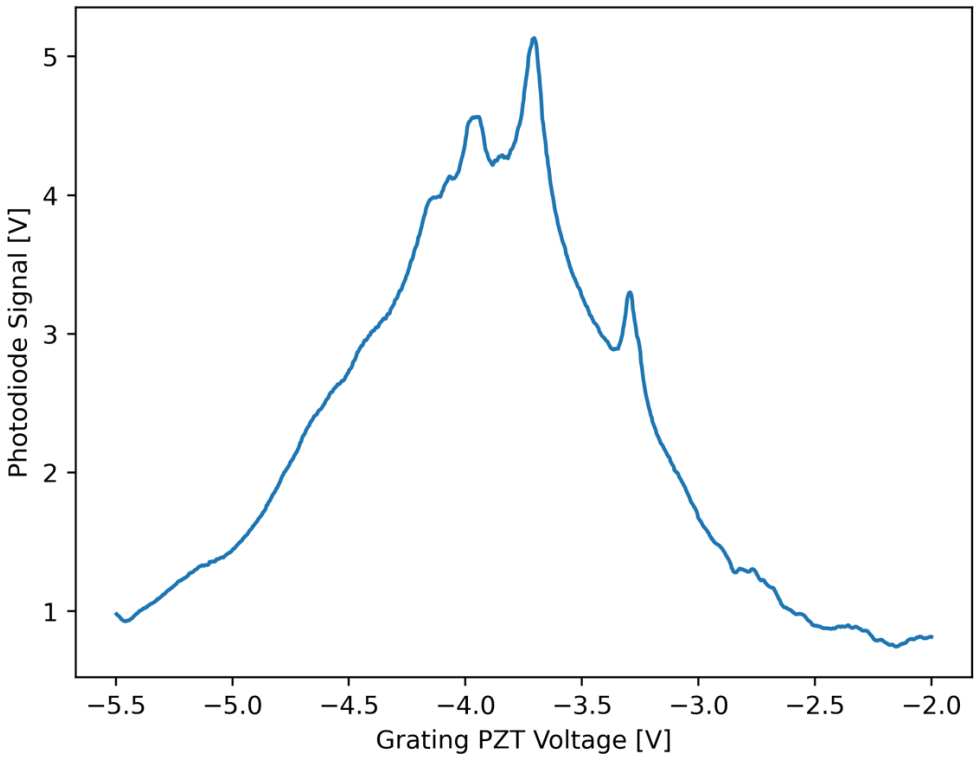

Figure 5: Saturated absorption PD plotted against the grating PZT voltage.

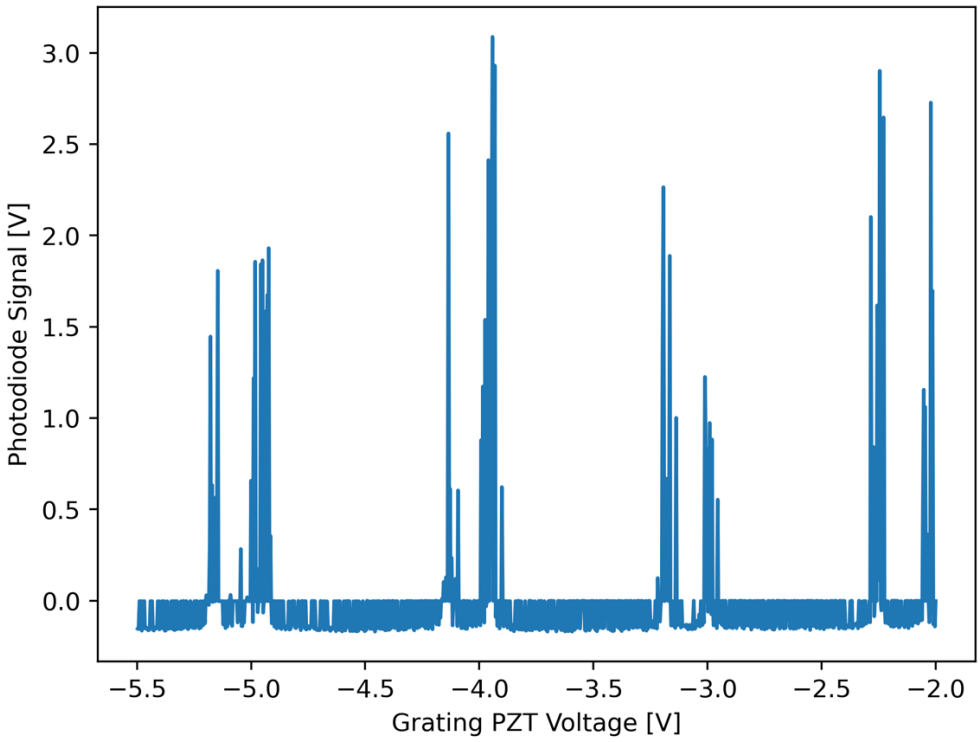

Figure 6: Fabry-Perot PD signal plotted against the grating PZT voltage.

4. Conclusion

Using the Arduino Portenta H7 microcontroller and Python programming language, a new DAQ system was successfully implemented into the existing experiment. Acquisition of the saturated absorption and Fabry-Perot photodiode signals was accomplished for laser frequency sweeps whilst simultaneously rendering data plots of the corresponding streams in real-time. At time of writing, functionality may be improved in various ways through the addition of a graphical user interface (GUI) used to control and configure the DAQ and voltage ramp systems. Additionally, the Python and Arduino source code may be generalized and improved to ease future development.

References

[1] D. P. DiVincenzo, "The Physical Implementation of Quantum Computation," *Fortschritte der Physik,* vol. 48, no. 9-11, pp. 771-783, 2000.

[2] K. Christandl, "Advancing neutral atom quantum computing: Studies of one-dimensional and two-dimensional optical lattices on a chip," The Ohio State University, 2005.

[3] H. J. Metcalf and P. Van, Laser Cooling and Trapping, New York: Springer, 1999.

[4] K. Christandl, "A Compact, Grating-stabilized Diode Laser for Atomic Spectroscopy," The Ohio State University, 2000.

[5] "analogReadResolution()," Arduino, 20 October 2020. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/zero-due-mkr-family/analogreadresolution/. [Accessed 9 June 2022].

[6] C. Liechti, "pySerial," 2020. [Online]. Available: https://pyserial.readthedocs.io/en/latest/. [Accessed 9 June 2022].

[7] "Matplotlib: Visualization with Python," 2021. [Online]. Available: https://matplotlib.org/. [Accessed 9 June 2022].

[8] "AtomTrapLab," GitHub, Inc., 2022. [Online]. Available: https://github.com/KGAtomTrapLab/AtomTrapLab_auto. [Accessed 9 June 2022].

Appendix

Python Source Code

```
from threading import Thread
from serial.tools import list_ports
import serial
import time
import collections
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import struct
import copy
import numpy as np

class serialPlot(Thread):
    def __init__(self, serialPort='/dev/ttyUSB0', serialBaud=38400, plotLength=100, dataNumBytes=2, numPlots=1):
        self.port = serialPort
        self.baud = serialBaud
        self.plotMaxLength = plotLength
        self.dataNumBytes = dataNumBytes
        self.numPlots = numPlots
        self.rawData = bytearray(numPlots * dataNumBytes)
        self.dataType = None
        if dataNumBytes == 2: self.dataType = 'h' # 2 byte integer
        elif dataNumBytes == 4: self.dataType = 'f' # 4 byte float
        elif dataNumBytes == 8: self.dataType = 'd' # 8 byte float (double)
        self.data = []
        self.privateData = None     # for storing a copy of the data so all plots are synchronized
        for i in range(numPlots):   # give an array for each type of data and store them in a list
            self.data.append(collections.deque([0] * plotLength, maxlen=plotLength))
        self.isRun = True
        self.isReceiving = False
        self.thread = None
        self.plotTimer = 0
        self.previousTimer = 0

        if serialPort == None:
            for ports in list_ports.comports():
                if 'arduino' in str(ports.manufacturer).casefold():
                    self.port = ports.device
        print('Trying to connect to: ' + str(serialPort) + ' at ' + str(serialBaud) + ' BAUD.')
        try:
            self.serialConnection = serial.Serial(serialPort, serialBaud, timeout=4)
            print('Connected to ' + str(serialPort) + ' at ' + str(serialBaud) + ' BAUD.')
        except:
            print("Failed to connect with " + str(serialPort) + ' at ' + str(serialBaud) + ' BAUD.')

    def readSerialStart(self):
        if self.thread == None:
            self.thread = Thread(target=self.backgroundThread)
            self.thread.start()
            # Block till we start receiving values
            while self.isReceiving != True:
                time.sleep(0.1)

    def getSerialData(self, frame, lines, lineValueText, lineLabel, timeText, pltNumber):
        if pltNumber == 0:  # in order to make all the clocks show the same reading
            currentTimer = time.perf_counter()
            self.plotTimer = int((currentTimer - self.previousTimer) * 1000)     # the first reading will be erroneous
            self.previousTimer = currentTimer
        self.privateData = copy.deepcopy(self.rawData)    # so that the 3 values in our plots will be synchronized to the same
sample time
        timeText.set_text('Plot Interval = ' + str(self.plotTimer) + 'ms')
        data = self.privateData[(pltNumber*self.dataNumBytes):(self.dataNumBytes + pltNumber*self.dataNumBytes)]
        value,  = struct.unpack(self.dataType, data)
        self.data[pltNumber].append(value)    # we get the latest data point and append it to our array
        lines.set_data(range(self.plotMaxLength), self.data[pltNumber])
        lineValueText.set_text('[' + lineLabel + '] = ' + str(value))

    def backgroundThread(self):    # retrieve data
        time.sleep(1.0)  # give some buffer time for retrieving data
        self.serialConnection.reset_input_buffer()
        while (self.isRun):
            self.serialConnection.readinto(self.rawData)
            self.isReceiving = True

    def close(self):
        self.isRun = False
```

```
        self.thread.join()
        self.serialConnection.close()
        np.savetxt('out.csv', np.array(self.data), delimiter=',', fmt='%8f')
        print('Disconnected...')


def makeFigure(xLimit, yLimit, title):
    xmin, xmax = xLimit
    ymin, ymax = yLimit
    fig = plt.figure(figsize=[16, 5])
    ax = plt.axes(xlim=(xmin, xmax), ylim=((ymin - (ymax - ymin) / 10),(ymax + (ymax - ymin) / 10)))
    ax.set_title(title)
    ax.set_xlabel("Time")
    ax.set_ylabel("Photodiode Signal [V]")
    # ax.set_yticks(xrange(ymin, ymax))
    return fig, ax

def findPort() -> str:
    print('Searching for DAQ serial port...', flush=True)
    ports = list_ports.comports()

    for port in ports:
        for info in [port.description, port.manufacturer, port.product]:
            if isinstance(info, str):
                if any(match in info.casefold() for match in ('arduino', 'envie')):
                    return port.device
    print('Cannot find Arduino! Port must be specified', flush=True)
    return None

def main():
    portName = findPort()
    baudRate = 115200
    maxPlotLength = 1000 # number of points in x-axis of real time plot
    dataNumBytes = 4     # number of bytes of 1 data point
    numPlots = 2         # number of plots in 1 graph
    s = serialPlot(portName, baudRate, maxPlotLength, dataNumBytes, numPlots)   # initializes all required variables
    s.readSerialStart()                                               # starts background thread

    # plotting starts below
    pltInterval = 42    # Period at which the plot animation updates [ms]
    lineLabelText = ['Saturated Absorption', 'Fabry-Perot']
    title = ['Saturated Absportion', 'Fabry-Perot']
    xLimit = [(0, maxPlotLength), (0, maxPlotLength), (0, maxPlotLength)]
    yLimit = [(-0.05, 1), (-0.05, 5)]
    style = ['b-', 'c-']  # linestyles for the different plots
    anim = []
    for i in range(numPlots):
        fig, ax = makeFigure(xLimit[i], yLimit[i], title[i])
        lines = ax.plot([], [], style[i], label=lineLabelText[i])[0]
        timeText = ax.text(0.50, 0.95, '', transform=ax.transAxes)
        lineValueText = ax.text(0.50, 0.90, '', transform=ax.transAxes)
        anim.append(animation.FuncAnimation(fig, s.getSerialData, fargs=(lines, lineValueText, lineLabelText[i], timeText, i),
interval=pltInterval))  # fargs has to be a tuple
        plt.legend(loc="upper left")
    plt.show()

    s.close()


if __name__ == '__main__':
    main()
```

Arduino Source Code (C++)

```
/**
 * @file simple_daq.ino
 * @author Jonathan Fuzaro Alencar (jfuzaroa@outlook.com)
 * @brief Arduino program to test piezo ramp and data acquisition.
 * @date 2022-06-12
 */

#if defined(LEDR) && defined(LEDG) && defined(LEDB)
    #define PORTENTA
    #define LED_BUILTIN LEDB
#else
    #define LEDR LED_BUILTIN
    #define LEDG LED_BUILTIN
    #define LEDB LED_BUILTIN
#endif

#ifndef PORTENTA
  auto const kTestOut = 3; // digital (PWM) output pin for voltage ramp
  auto constexpr aReadRes = 10; // [bits] analogRead bit resolution
  auto constexpr aWriteRes = 8; // [bits] analogWrite bit resolution
#else
  auto const kTestOut = A6; // analog (DAC) output pin for voltage ramp
  auto constexpr aReadRes = 16; // [bits] analogRead bit resolution
  auto constexpr aWriteRes = 12; // [bits] analogWrite bit resolution
#endif

auto constexpr kBaud = 115200; // serial baud rate (bits per second)
auto constexpr kSatAbsPin = A0; // analog input pin for saturated absorption PD
auto constexpr kErrorPin  = A1; // analog input pin for error signal
auto constexpr kFabPerPin = A2; // analog input pin for Fabry-Perot PD

auto const aReadMax  = pow(2, aReadRes)  - 1; // max read value
auto const aWriteMax = pow(2, aWriteRes) - 1; // max write value

/**
 * @brief Configures built-in LED including RGB leds of Arduino Portenta.
 */
void setupLED() {
    pinMode(LED_BUILTIN, OUTPUT);
    #ifdef PORTENTA
        pinMode(LEDR, OUTPUT);
        pinMode(LEDG, OUTPUT);
        pinMode(LEDB, OUTPUT);
        digitalWrite(LEDR, HIGH);
        digitalWrite(LEDG, HIGH);
        digitalWrite(LEDB, HIGH);
    #endif
}

/**
 * @brief Maps value (x) from one floating point range (in_min, in_max) to another another (out_min, out_max).
 *
 * @param in Input value in range between `in_min` and `in_max`.
 * @param in_min Minimum input value.
 * @param in_max Maximum input value.
 * @param out_min Minimum output value.
 * @param out_max Maximum output value.
 * @return double Mapped output value.
 */
double mapD(double in, double in_min, double in_max, double out_min, double out_max) {
  return (in - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

/**
 * @brief Streams data out through serial as raw byte values.
 *
 * @param data1 First data value to be sent.
 * @param data2 Second data value to be send.
 */
void sendSerial(float* data1, float* data2)
{
  byte* byteData1 = (byte*)(data1);
  byte* byteData2 = (byte*)(data2);
  byte buf[8] = {byteData1[0], byteData1[1], byteData1[2], byteData1[3],
                 byteData2[0], byteData2[1], byteData2[2], byteData2[3]};
  Serial.write(buf, 8);
}

void setup() {
  pinMode(kSatAbsPin, INPUT);
```

```
  pinMode(kErrorPin, INPUT);
  pinMode(kFabPerPin, INPUT);
  setupLED();

  Serial.begin(kBaud); // begin serial communication via USB
}

void loop() {
  int satAbsRaw = analogRead(kSatAbsPin);
  int fabPerRaw = analogRead(kFabPerPin);
  float SA = mapD(satAbsRaw, 0, aReadMax, 0.0, 5);
  float FP = mapD(fabPerRaw, 0, aReadMax, 0.0, 5);
  sendSerial(&SA, &FP);
}
```