Richard Ziegahn

June-July 2018

ricky.ziegahn@gmail.com

X-Ray Diffraction

X-Ray Diffraction Software

The software for the X-Ray Diffractometer is composed of two parts: The Arduino code and the python (3.6) code.

Arduino

General Overview

There are a series of menus which are controlled by logic gates that determine what this program will do. The first gate is whether a run will be performed, or manual commands will be performed. The manual commands will be described first.

The first of the manual commands is accept_drive(), what this does is it receives a number of steps to move a specific motor, a positive amount of steps moves counter clockwise and a negative amount moves anti clockwise. Driving is done by switching on a pin for a certain amount of time (pulse length), then switching off for a certain amount of time (pulse cycle – pulse length), this is dependent on what the motors need to switch on and move a step. The accept_drive() command works only in steps so no conversions are being made and exact amounts are guaranteed.

The second of the manual commands does nothing at all, it is an update to the location on the python side of the code.

The third command is another drive commands, but the setup is different from the python side.

The forth manual command accepts a time amount to count pulses, counts, then prints it to the serial port. The counting is done by setting up a pin as an interrupt reader, and incrementing the count each time the power rises. The count is reset to zero before the counting begins, and waits until the time period has passes to report the time.

What the machine does is read pulses per time period from a start to degree to an end degree, and do this every degree increment until the end degree is reached. A start degree is accepted, and converted to steps then rounded to the nearest degree. The pulses per time period is then read and reported. The degree is then incremented, and the same process happens to convert it to steps, pulses are read and reported. This reconversion is done each time to prevent the locations from lagging behind and creating inaccuracies. There are two motors, one motor moves half the degrees of the other, and both are moved before the pulses are measured. This process is repeated until it reaches or passes the end degree. There is currently no way to interrupt the process and it must be ran until the end.

Function Overview

wait_for_input(): Waits for input from Python before trying to read the serial and proceeding.

degrees_to_steps(degree): Accepts a degree, converts it to degrees.

steps_to_degrees(steps): Accepts a step amount, converts it to degrees.

confirm_flag(): Prints a flag so python will know that an operation is complete.

inc_count(): Increments the count by one. This is performed each time there is a pin interrupt.

accept_length(): Receives the time period to read pulses for.

accept_parameters(): Accepts the starting degree, ending degree, degree increment, and the time period to read pulses for. This function is done before the run begins.

count(): Waits the time period and lets the count increase (inc_count is always run in the background). The count is always reset to zero before this function is run, if the count is reset to within the function, it will only print zero. At the end of the time interval, the counts in that time period is printed.

drive(steps, motor): Accepts an amount of steps to move a motor.

choose_direction(steps, motor): Run from the drive command, it turns a pin to high or low to let the motor know which direction it will be moving, whether its high or low depends on the sign of the steps.

move_steps(steps, motor): Run from within the drive command, sends the correct amount of pulses to move the amount of steps wanted.

accept_drive(): Receives the motor and amount of steps to move from python, then moves it. Prints a confirmation flag when complete.

Python

General Overview

The python (3.6) code brings the user through a series of menus to determine what wants to be achieved. It also plots the data (2 theta vs time) in real time and saves it to a text file.

The libraries used are serial, time, and PyQtGraph (which can be found at http://pyqtgraph.org).

PyQtGraph is used for live graphing of the temperature. To initialize a plot, do as follows (in the python code, each of these objects is instead part of a list of objects):

```
from pyqtgraph.Qt import QtGui

import pyqtgraph as pg

app = QtGui.QApplication([])

p = pg.plot()

curve = p.plot()
```

To set or update the data and the graph, include

```
curve.setData(x = <list>, y = <list>)

app.processEvents()
```

This does not update the range, the range can be updated with one of the following (required before app.processEvents())

```
p.setXRange(<x1>,<x2>)
```

```
p.setYRange(<y1>,<y2>)
```

When performing a run, the python code will first zero the machine. The location, in steps, for each motor is stored in a text file. That step amount, multiplied by negative 1, will be sent to the Arduino for each motor so the motors will be moved to the zero location. The user must then verify that it is at zero (and make any adjustments if it is not). A name and detailed title will then have to be entered (detailed meaning more than 10 characters). The user will then be prompted to enter the parameters for the run (starting degree, ending degree, degree increment, milliseconds per reading). The parameters are all passed to the Arduino and the Arduino will begin to send values. Python runs in parallel to the Arduino, calculating each step and degree locations. The count received, and the degree location are plotted against each other. The plot immediately sets the x bounds to go from the starting degree to the ending degree, and the y bounds go from zero to the maximum received count. This is done until the location reaches the end degree.

The manual commands are as described in the Arduino code documentation.

Function Overview

confirm_flag(): Waits for the confirmation flag from the Arduino.

response_time(): Waits the amount of time the Arduino needs before it can receive the next parameter.

done(): Prints a confirmation for the user.

degrees_to_steps(degree): Converts a degree to steps, and returns it.

steps_to_degrees(step): Converts a step amount to degrees, and returns it.

current_degree(): Looks are the starting degree and the amount of terms received, and uses the degree increment to determine the current location.

file_chooser(motor): Returns the name of the text file storing the location of a motor.

read_location(motor): Returns the location of a motor.

set_location(motor): Sets the location of the motor, does not do any driving (is an override).

increment_location(steps, motor): Increments the location of a motor by an amount of steps.

give_drive(steps, motor): Passes an amount of steps to pass to a specific motor to the Arduino.

goto(location, motor): Reads the current location, accepts where the user wants to go, takes the difference in steps and moves the motor that many steps to go to that location.

check_limit(steps, motor): Reads the current location and checks that the motor will not move past its limit if does it moves the amount of steps requested.

receive_count(): Accepts the amount of pulses counted by the Arduino.

X-Ray Diffraction Hardware

The X-Ray Diffraction machine was found to be totally broken thus this project has been abandoned.

Github

https://github.com/RickyZiegahn/X-Ray_Diffraction

Version 1.1