Relatório de Iniciação Científica CDTN/CNEN

Programa: PIBIC/CNPq
Período de atividades do bolsista: 10/2021 – 08/2022

Título do Projeto de Pesquisa ao qual o Plano de Trabalho está vinculado:	Investigação numérica e experimental de grades espaçadoras e bocais para elementos combustíveis
Título do Plano de Trabalho:	Desenvolvimento de sistema de automação experimental para o circuito LARANJA
Área do Conhecimento CNPq (nome):	Tecnologia de reatores
Subárea CNPq (nome):	Termo-hidráulica de reatores

Felipe Reis Campanha Ribeiro, felipercr@ufmg.br
Universidade Federal de Minas Gerais - UFMG

Andre Augusto Campagnole dos Santos, aacs@cdtn.br
Serviço de Tecnologia de Reatores - SETRE

Resumo. *Utilizando três motores de passo comandados por um microcontrolador e um código na linguagem python, foi desenvolvido um sistema de posicionamento automático de um LDV utilizado para traçar o perfil de velocidade do escoamento do fluido que escoa através da grade espaçadora de um reator nuclear.*
Palavra-chave*: LDV, Grade Espaçadora, Microcontrolador*

1. INTRODUÇÃO

Tendo em vista a necessidade de validar experimentalmente o comportamento do fluido no núcleo de um reator nuclear ao atravessar uma grade espaçadora, um LDV deve ser utilizado para traçar o seu perfil de velocidade. Para isso, foi construído um sistema de posicionamento de três eixos que tem como objetivo colocar o LDV para medir pontos específicos deste escoamento. A ideia com este trabalho é a de construir a parte elétrica deste sistema, escrever um código para que o processo de aquisição dos dados ocorra de maneira automática e fazer a sua calibração.

2. OBJETIVOS

Este projeto tem como principal objetivo atender à demanda dos pesquisadores do laboratório de termo-hidráulica do CDTN para que possam prosseguir nas suas pesquisas fazendo aquisições experimentais de dados, como citado na seção 1. Além disso, o sistema deverá atender a 3 requisitos mínimos, como é citado no livro "Engenharia de Software [Sommerville, 2018]", ser robusto, confiável e ser facilmente utilizável, uma vez que os dados coletados por ele serão levados em conta em publicações futuras, portanto, devem ser muito precisos.

3. MATERIAIS E MÉTODOS

3.1. Estrutura mecânica

O sistema a ser automatizado já havia sido construído previamente, desta forma, toda a parte mecânica e estrutural já havia sido organizada. O circuito completo conta com um sistema de bombeamento de fluido, que o faz circular ao longo da seção das varetas. Além disso, um sistema de posicionamento de três eixos foi montado de forma que seja possível encaixar nele o LDV que fará as medições na seção. Para fazer os movimentos, são utilizados três motores de passo, cada um para comandar um eixo.

Figura 1. Circuito completo – Vista frontal

Figura 2. Circuito completo – Vista lateral

É interessante ressaltar que a seção das varetas a ser estudada foi construída com borossilicato, um material transparente e com um índice de refração igual ao do fluido utilizado, que, no caso, é o limoneno. Desta forma, ao casarmos os índices de refração, não haverá a necessidade de se preocupar com os lasers do LDV colidindo com alguma vareta, pois os seus feixes não sofrerão desvio algum, como pode ser provado pela lei de Snell aplicada à interface fluido-vareta.

$$n_1 \,.\sin(\theta_1) = \, n_2 \,.\sin(\theta_2) \quad (1)$$

Como os índices de refração tanto do fluido quanto das varetas são muito próximos se considerarmos ambos como sendo praticamente iguais, obtemos que:

$$\sin(\theta_1) = \sin\,(\theta_2) \quad (2)$$

Portanto, como os senos dos ângulos de incidência e de reflexão na interface são iguais, podemos concluir que a luz não sofre reflexão. Desta forma, o LDV poderá medir pontos atrás das varetas sem nenhum problema.

Figura 3. Sonda do LDV medindo a velocidade de um fluido.

3.2. Planejamento geral do funcionamento do sistema

Inicialmente, o sistema foi dividido em quatro partes, que deveriam ser acopladas em um único software escrito utilizando a linguagem Python. As partes em questão são:

1. Sistema de controle dos motores de passo para o posicionamento do LDV.

2. Sistema de comunicação entre o software e o sistema de controle dos motores (serve para enviar as posições desejadas ao controlador dos motores).

3. Sistema de comunicação entre o usuário e o programa para escolher as posições que se deseja medir.

4. Sistema de comunicação entre o software e o programa de controle do LDV (serve para iniciar uma nova medição e descobrir se uma medição já terminou).

A primeira etapa do projeto, portanto, foi encontrar uma maneira de fazer o acionamento dos motores de passo. Para isso, foi utilizado um microcontrolador Atmega2560 carregado com um interpretador de G-Code *open source* chamado GRBL. O G-Code nada mais é que um conjunto de códigos que se traduzem em comandos de posição para sistemas de posicionamentos de três eixos, como por exemplo em fresas e impressoras 3D. A função do GRBL é receber informações de G-Code e fazer o acionamento devido dos motores de passo para que o sistema percorra o caminho desejado.

Desta forma, foi montado um circuito elétrico utilizando o microcontrolador Atmega2560 (carregado com o GRBL) e três drivers de motores de passo (um para cada motor). Os drivers têm a função de receber o sinal digital vindo do microcontrolador e converter em sinais de potência para fazer o acionamento dos motores. Uma outra função dos drivers é fazer o que é chamado de *microstepping*. Basicamente, o que isso significa é que cada "passo" do motor é subdividido em mais "passos", desta maneira, conseguimos aumentar a resolução do sistema, que se torna mais preciso. No caso deste sistema, a razão utilizada foi de 1 para 16, ou seja, cada "passo" agora equivale a um dezesseis avos do valor nominal do passo do motor. Além disso, foi utilizada uma fonte de alimentação de 24 volts para alimentar os drivers do motor.

Figura 4. Montagem física do sistema de acionamento dos motores.

Uma outra coisa que também foi incluída ao projeto da máquina foi a instalação de sensores de fim de curso. Eles foram instalados em todos os três eixos da máquina, tendo dois em cada eixo. Desta maneira, a máquina possui um zero mecânico fixo e isto também impede que os eixos saiam dos seus trilhos, o que pode ocasionar uma falha mecânica grave na máquina.

A próxima etapa do projeto foi programar um sistema de comunicação entre o computador (de onde vem a informação dos pontos de medição desejados) e o microcontrolador (que interpretará os comandos vindos do computador e fazer o acionamento dos motores). Para isso, foi necessário estabelecer uma forma de conexão entre o microcontrolador e o computador. Para resolver este problema, foi utilizado a biblioteca pySerial, que permite manipular as portas seriais do computador e estabelecer comunicação com outros sistemas externos "pySerial Documentation Release 3.4 [Liechti, 2021]".

Sendo assim, o software programado envia, através de uma comunicação serial, comandos em G-Code com as coordenadas desejadas para o microcontrolador, que o traduz em movimentos dos motores.

A terceira etapa é fazer um sistema de interface para o usuário. A ideia é que os pontos a serem medidos deveriam ser colocados em uma tabela no formato Excel (.xlsx) e o software deve reconhecer estes pontos, transformá-los em G-Code e enviar para o microcontrolador. Para isso foi utilizada a

biblioteca Pandas do Python, que possui funções de leitura de arquivos .xlsx “Pandas Documentation version 1.5.0 [2022]”. Desta maneira, os pontos podem ser colocados na planilha e o programa pode trabalha-los como variáveis.

Por fim, o ultimo sistema a ser desenvolvido é um que faz a interação do software com o programa que controla o LDV. O controlador do LDV é um software chamado *BSA Flow Software*. Este programa possui uma função capaz de tirar diversas medidas sucessivamente, basta, para isso, passar a lista de pontos a serem medidas para este programa (no caso, estes são os mesmos pontos escolhidos pelo usuário e que são colocados no arquivo .xlsx) e dar início ao processo contínuo de medição “BSA Flow Software Version 4.10 Installation & User’s Guide [Dantec Dynamics, 2006]”.

Nesta função, toda vez que o LDV termina de tirar a medida de um ponto e está pronto para iniciar uma nova medição, aparece uma notificação na tela e pergunta ao usuário se ele deseja realizar a medida do próximo ponto. Se o usuário clicar na opção que confirma o início da medição de um novo ponto, o LDV passará a tirar uma nova medida.

Visando automatizar este processo, foi utilizada a biblioteca PyAutoGUI. Esta biblioteca possui funções que checam as informações contidas na tela do computador, encontram imagens específicas e clicam na posição desejada “PyAutoGUI Documentation [Sweigart, 2021]”. Estas funcionalidades foram utilizadas da seguinte forma: o software espera aparecer a mensagem de confirmação da nova medição, faz a movimentação para o próximo ponto e clica na mensagem confirmando uma nova medição, repetindo este ciclo diversas vezes até terem medido todos os pontos designados pelo usuário.

De forma a garantir maior confiabilidade e robustez ao sistema, o software conta com um arquivo de log. O arquivo em questão é um .txt (arquivo de texto), cuja função é armazenar o estado do sistema a todo momento. Ele registra todos os comandos que foram enviados ao microcontrolador e algumas de suas variáveis de estado, como velocidade (*feedrate*), posição do ponto de coordenadas zero e posição atual da máquina.

Após estar com todo o software devidamente programado, a ultima etapa foi fazer uma análise de sua precisão e, caso necessário, fazer alguma calibração. Para isso, utilizamos um relógio comparador com precisão de décimos de milímetros e o fixamos no lugar da sonda do LDV. Em seguida, colocamos o relógio para encostar 5mm em uma superfície fixa e o zeramos. A partir desta posição, a máquina foi programada para ir para um outro ponto bem afastado e voltar para encostar nesta superfície diversas vezes. A intenção com este experimento seria medir a evolução do erro ao longo de várias medições.

Figura 5. Montagem física do sistema de medição.

Por fim, foi feita a refatoração completa de todo o código, bem como diversos testes. Isto foi feito para garantir uma maior confiabilidade do sistema e para que o mesmo pudesse ter uma maior robustez. Os testes foram feitos de forma manual, forçando situações de erro uma por uma, o que era possível de ser feito, uma vez que o software não era muito grande e não haviam muitos casos para cobrir. Após fazer os testes, aqueles que resultaram em erro foram todos devidamente tratados um a um, e o sistema agora ou não está mais sujeito a alguns desses erros ou, em alguns casos, retorna ao usuário informações sobre o erro ocorrido, por exemplo, quando ocorre o acionamento de algum fim de curso. O código completo pode ser consultado no **Apêndice B**.

4. RESULTADOS E DISCUSSÕES

Após a montagem do relógio comparador na máquina para observar a evolução do erro, duas baterias de testes foram feitas ao todo. No gráfico abaixo, os pontos em azul mostram a primeira seção de testes e em laranja a segunda. O gráfico mostra o erro, dado em centésimos de milímetros, em função do número de vezes que a máquina vai para uma posição afastada da inicial e retorna ao ponto em que ela estava (ciclos).

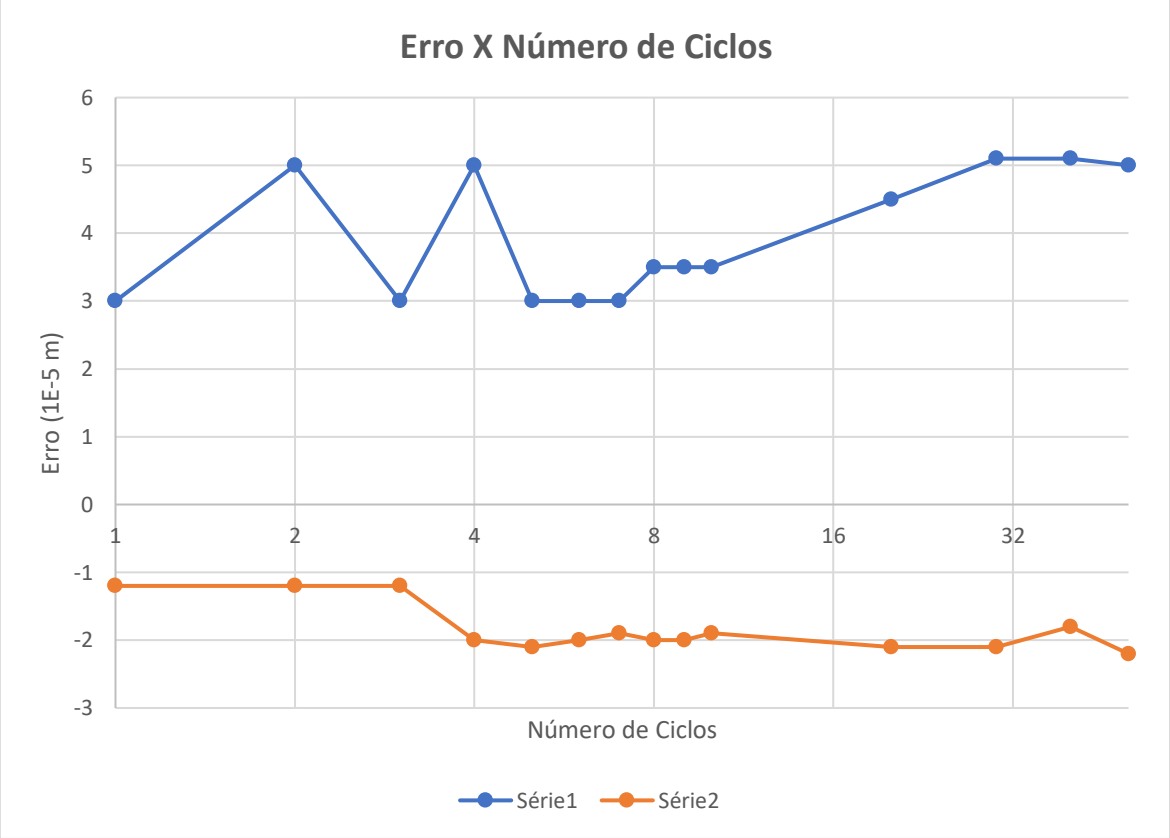

Figura 6. Evolução do erro ao longo de diversas repetições.

Observando estes valores, podemos concluir que a máquina possui uma precisão de décimos de milímetros, tendo apresentado pequenos erros da ordem de centésimos de milímetros. Vemos ainda que estes erros tendem a se acomodar aos poucos em uma certa faixa de valores.

Um outro ponto a ser abordado é quanto ao seguimento do plano de trabalho proposto para o projeto. No caso, não houveram atrasos e o plano foi seguido conforme era planejado.

5. CONCLUSÕES

Após a conclusão do projeto e ter sido feita uma avaliação sobre a precisão da máquina, chegamos a duas conclusões. A primeira delas é que o software está de fato com uma robustez bastante satisfatória. Todos os erros encontrados e mais comuns foram devidamente tratados, o que pode ajudar muito os usuários finais a utilizarem o sistema.

A segunda conclusão é de que a máquina possui uma precisão muito boa tendo em vista os fins para os quais ela foi pensada. Como dificilmente haverá a necessidade de se medir dois pontos com distâncias muito pequenas entre si (da ordem de centésimos de milímetros), podemos concluir que a precisão está bastante satisfatória.

6. AGRADECIMENTOS

Agradeço, inicialmente, ao CNPq por terem apoiado financeiramente o projeto, o que permitiu que ele pudesse ser realizado.

Em segundo lugar, agradeço a todos os membros do laboratório de termo-hidráulica do CDTN por toda a ajuda que foi dada e pelos ensinamentos passados.

7. REFERÊNCIAS

Sommerville, Ian. “Engenharia de Software”, 10. Ed, S.Paulo, Brasil, Pearson, 2018.

Lieichti, “pySerial Documentation Release 3.4”, Liechti, 2021.

Sweigart, “PyAutoGUI Documentation”, Sweigart, 2021.

Pandas, “Pandas Documentation version 1.5.0”, em: https://pandas.pydata.org/docs/index.html, acessado em 21 de novembro de 2022.

Dantec Dynamics, “BSA Flow Software Version 4.10 Installation & User’s Guide” Dantec Dynamics, 2006.

8. ANÚNCIO DE RESPONSABILIDADE

Os autores são os únicos responsáveis pelo material incluído neste trabalho.

APÊNDICE A – ESTRUTURA DE ARQUIVOS DO CÓDIGO DO SISTEMA DE POSICIONAMENTO

```
/
|_ _ _ _ /auto_clicker
|                 |_ _ _ _ /imgs
|                 |          |_ _ _ _ acquire.jpg
|                 |
|                 |_ _ _ _ clicker.py
|
|_ _ _ _ gcode_sender.py
|_ _ _ _ main.py
|_ _ _ _ menu.py
|_ _ _ _ points_matrix.py
```

APÊNDICE B – CÓDIGO DO SISTEMA DE POSICIONAMENTO

Arquivo main.py

```
from gcode_sender import *
from points_matrix import *
from menu import menu
import ctypes, sys

def is_admin():
    try:
        return ctypes.windll.shell32.IsUserAnAdmin()
    except:
        return False

def main():
    m = movement()

    while True: menu(m)

if __name__ == "__main__":
    if is_admin(): main()
    else: ctypes.windll.shell32.ShellExecuteW(None, "runas", sys.executable, "
".join(sys.argv), None, 1)
```

Arquivo menu.py

```
from os import system
from gcode_sender import *

def menu(m):

    system('cls')

    while True:
        print('Menu:')
        print('  1. Full cycle (Remember to open BSA Flow)')
        print('  2. Home machine')
        print('  3. Set zero')
        print('  4. Set feedrate\n')
        opt = input('Select option: ')

        if opt.isnumeric():
            if int(opt) > 4 or int(opt) == 0:
                system('cls')
                print('Invalid option\n')
            else: break

        system('cls')
        print('Invalid option\n')

    system('cls')

    match opt:
        case '1':
            m.full_cycle()
        case '2':
            m.home()
        case '3':
            m.set_zero()
        case '4':
            m.set_feedrate()
```

Arquivo points_matrix.py

```
import pandas as pd
import numpy as np


#################### EXCEL ######################

#     A     B     C
# 1   x     y     z    ->  A1 = x, B1 = y, C1 = z
# 2   12    20    17   ->  Generic example
# 3   ...

# Use . as a decimal point

#################################################

def points_matrix():
    matrix = pd.read_excel('../matrix.xlsx')
    return(matrix.to_numpy())
```

Arquivo clicker.py

```
import pyautogui as pag
import time

# auto_clicker/imgs/acquire.png

button = 'acquire.png'

def acquire(last):

    location = None
    while location == None: location =
pag.locateCenterOnScreen(f'auto_clicker/imgs/{button}', confidence=0.7)

    pag.click(location)

    time.sleep(0.2)

    if last == True: return

    location = None
    while location == None: location =
pag.locateCenterOnScreen(f'auto_clicker/imgs/{button}', confidence=0.7)

    return
```

Arquivo gcode_sender.py

```
from os import system, path
import serial
import serial.tools.list_ports
import time
from points_matrix import points_matrix
from auto_clicker.clicker import acquire

class movement():
    def __init__(self):

        self.log_path = '../log.txt'

        system('cls')

        #Serial communication
        while True:
            print('Connecting...\n')

            ports = serial.tools.list_ports.comports()
            for port, desc, hwid in sorted(ports):
                print(f"{port}")

            self.com_number = input('Select a serial port: ')
            if self.com_number.isnumeric() == False:
                system('cls')
                print('Invalid port, insert a numeric value\n')
            else: break


        self.g = serial_begin(self.com_number)
        print('\nGRBL connected\n')

        time.sleep(1)

        system('cls')

        if path.exists(self.log_path) == False:
            with open(self.log_path, 'w') as log:
                log.write(f'Zero (x,y,z): 0 0 0\n')
                log.write(f'Feedrate (mm/s): 1000\n')
                log.write(f'Last Position (x,y,z): 0 0 0\n')
                log.write('\n-------------------------------------\n\n')
                log.write('Given of commands: \n\n')

        with open(self.log_path, 'r') as log:
            lines = log.readlines()
            lines = lines[1]
            lines = lines.split()
```

```
            self.fdr = lines[2]


    def move(self, point, message=None):

        x = float("{:.3f}".format(float(point[0])))
        y = float("{:.3f}".format(float(point[1])))
        z = float("{:.3f}".format(float(point[2])))

        self.g.flushInput()
        self.g.flushOutput()
        #time.sleep(1)

        if message: print(f'{message}')
        print(f'Command: X{x} Y{y} Z{z}')

        with open(self.log_path, 'r') as log:
            lines = log.readlines()

        feedrate = lines[1]
        feedrate = feedrate.split()
        feedrate = feedrate[2]
        self.fdr = feedrate

        lines.append(f'X{x} Y{y} Z{z}\n')

        with open(self.log_path, 'w') as log:
            for line in lines:
                log.write(line)

        self.g.write(f'$J = G21 X{x} Y{y} Z{z} F{self.fdr}\n'.encode())

        #time.sleep(1)

        self.g.flushOutput()

        inp = self.g.readline().decode().strip()
        if inp == 'ok': print('ok')
        else: print(f'Error: {inp}' + '\n\n' )

        while self.check_pos(point) == False: pass

    def check_pos(self, point):
        self.g.flushOutput()
        self.g.flushInput()
        self.g.write('?'.encode())

        pos = self.g.readline().decode()

        #If a hard limit stop occurs, GRBL will send
```

```
        #a message -> [MSG:Reset to continue]
        if pos == '[MSG:Reset to continue]' or pos == 'ALARM 1':
            print('\nHard limits\n')
            print('Re-homing is highly recommended!\n')
            while True: pass

        pos = pos.split('|')
        try:
            pos = pos[1]
        except:
            print('\nError, probably due to hard limits')
            print('Re-homing is highly recommended!\n')
            while True: pass
        pos = pos.split(':')
        pos = pos[1]
        pos = pos.split(',')

        for item in range(len(pos)): pos[item] = float(pos[item])

        x = float("{:.3f}".format(float(point[0])))
        y = float("{:.3f}".format(float(point[1])))
        z = float("{:.3f}".format(float(point[2])))

        with open(self.log_path, 'r') as log:
            lines = log.readlines()

        with open(self.log_path, 'w') as log:
            for n, line in enumerate(lines):
                if n == 2:
                    log.write(f'Last Position (x,y,z): {pos[0]} {pos[1]}
{pos[2]}\n')
                else:
                    log.write(line)

        if pos[0] == x and pos[1] == y and pos[2] == z: return True
        else: return False

    def home(self):
        print('Homing cycle enabled...')
        self.g.write('$H\n'.encode())
        #time.sleep(1)

        self.g.flushOutput()

        while self.g.readline().decode().strip() != 'ok': pass

        #Close and reopen communication to reset MPos
        self.g.close()
        #time.sleep(1)
        self.g = serial_begin(self.com_number)
```

```
        with open(self.log_path, 'r') as log:
            lines = log.readlines()

        lines.append('Homing cycle\n')
        lines.append('Reset MPos\n')

        with open(self.log_path, 'w') as log:
            for line in lines:
                log.write(line)

        print('Homing cycle completed\n')

    def full_cycle(self):
        with open(self.log_path, 'r') as log:
            lines = log.readlines()

        with open(self.log_path, 'w') as log:
            lines = lines[0:8]
            for line in lines:
                log.write(line)

        self.home()
        self.move_to_zero()

        #time.sleep(1)

        matrix = points_matrix()

        n_points = len(matrix)
        completed = 0

        last = False

        for line in matrix:
            completed = completed + 1
            if completed == n_points: last = True

            self.move(line, f'{completed} / {n_points}')
            while self.check_pos(line) == False: pass

        ##################LASER#################
            acquire(last)
        ##################LASER#################

            print(f'Done\n')
            time.sleep(0.2)

        print('Full cycle completed!')
```

```
        with open(self.log_path, 'r') as log:
            lines = log.readlines()

        lines.append('Full cycle completed!\n')

        with open(self.log_path, 'w') as log:
            for line in lines:
                log.write(line)

        time.sleep(2)

    def set_zero(self):

        system('cls')
        while True:
            x = input('Select x: ')

            try:
                if float(x) < 0:
                    system('cls')
                    print('Select a valid number\n')
                else: break

            except:
                system('cls')
                print('Select a valid number\n')

        system('cls')
        while True:
            y = input('Select y: ')

            try:
                if float(y) < 0:
                    system('cls')
                    print('Select a valid number\n')
                else: break

            except:
                system('cls')
                print('Select a valid number\n')

        system('cls')
        while True:
            z = input('Select z: ')

            try:
                if float(z) < 0:
                    system('cls')
                    print('Select a valid number\n')
                else: break
```

```
            except:
                system('cls')
                print('Select a valid number\n')

        with open(self.log_path, 'r') as log:
            lines = log.readlines()

        with open(self.log_path, 'w') as log:
            for n, line in enumerate(lines):
                if n == 0:
                    log.write(f'Zero (x,y,z): {x} {y} {z}\n')

                else:
                    log.write(line)

    def set_feedrate(self):

        system('cls')
        while True:
            f = input('Select a Feedrate (mm/s): ')
            try:
                if float(f) < 0:
                    system('cls')
                    print('Select a valid number\n')
                else: break

            except:
                system('cls')
                print('Select a valid number\n')

        with open(self.log_path, 'r') as log:
            lines = log.readlines()

        with open(self.log_path, 'w') as log:
            for n, line in enumerate(lines):
                if n == 1:
                    log.write(f'Feedrate (mm/s): {f}\n')

                else:
                    log.write(line)

    def last_loc(self):
        pass

    def move_to_zero(self):
        with open(self.log_path, 'r') as log:
            lines = log.readlines()
            lines = lines[0]
            lines = lines.split()
```

```
            x = lines[2]
            y = lines[3]
            z = lines[4]

            point = (x, y, z)

        self.move(point, 'Move to zero and reset MPos')

        #Close and reopen communication to reset MPos
        self.g.close()
        #time.sleep(1)
        self.g = serial_begin(self.com_number)

        with open(self.log_path, 'r') as log:
            lines = log.readlines()

        lines.append('Reset MPos\n')

        with open(self.log_path, 'w') as log:
            for line in lines:
                log.write(line)

        print('Done\n')

    #Close serial communication
    def finish(self):
        self.g.close()

def serial_begin(com_number):

    try:
        g = serial.Serial(f'COM{com_number}', 115200, timeout = 2)

    except:
        print(f'\nConnection error, please check COM{com_number} and restart the
program!\n')
        while True: pass


    g.write('\r\n\r\n'.encode())          #Wake up GRBL
    time.sleep(1)

    g.write('$X\n'.encode())              #Unlock
    time.sleep(1)

    g.flushInput()

    return g
```