Diseño y construcción de un stage de translación en x, y y z automatizado.

Documento final

Juan Barbosa

Universidad de los Andes
Departamento de Física

Escrito en Python y C. El software de código abierto es un software con código fuente que cualquier persona puede inspeccionar, modificar y mejorar.

Universidad de los Andes
Departamento de Física

Título:
Diseño y construcción de un stage de translación en x, y y z automatizado.

Periodo del projecto:
Segundo periodo 2017

Participantes:
Juan Barbosa

Supervisores:
Manu Forero Shelton, PhD

Copias: 2

Número de páginas: 20

Fecha final:
Diciembre 3, 2017

Abstract:

Here is the abstract

Índice general

Capítulo 1

Introducción

Estado del arte

Marco teórico

Capítulo 2

Resultados

Here is chapter 2. If you want to leearn

Capítulo 3

Conclusion

In case you have questions, comments, suggestions or have found a bug, please do not hesitate to contact me. You can find my contact details below.

Jesper Kjær Nielsen
jkn@es.aau.dk
http://kom.aau.dk/~jkn
Fredrik Bajers Vej 7
9220 Aalborg Ø

Apéndice A

Código

El código fuente del projecto, tanto para el uso del usuario para la obtención de datos, como para la manipulación del microcontrolador al interior del mismo, se encuentra de manera libre en GitHub [1]. El código es libre, cualquier persona lo puede inspeccionar, modificar y mejorar.

Librería Python (mauscope)

Disponible en PyPI [2](Python Package Index), la instalación se puede llevar a cabo usando el mánager de paquetes de Python (pip) de la siguiente forma:

```
pip install mauscope
```

También es posible descargar la versión de desarrollo desde GitHub seguido de su instalación:

```
python setup.py install
```

constants.py

```
BAUDRATE = 4800
TIMEOUT = 7

#SLEEP_TIME = 1e-4

XLEFT = 0xA0
XRIGHT = 0xA1
X1 = 0xA2
X64 = 0xA3
X128 = 0xA4
```

[1] https://github.com/jsbarbosa/miniscope
[2] https://pypi.python.org/pypi/mauscope

```
X192 = 0xA5
X256 = 0xA6
X320 = 0xA7
X384 = 0xA8
X448 = 0xA9
X512 = 0xAA

YLEFT = 0xB0
YRIGHT = 0xB1
Y1 = 0xB2
Y64 = 0xB3
Y128 = 0xB4
Y192 = 0xB5
Y256 = 0xB6
Y320 = 0xB7
Y384 = 0xB8
Y448 = 0xB9
Y512 = 0xBA

ZLEFT = 0xC0
ZRIGHT = 0xC1
Z1 = 0xC2
Z64 = 0xC3
Z128 = 0xC4
Z192 = 0xC5
Z256 = 0xC6
Z320 = 0xC7
Z384 = 0xC8
Z448 = 0xC9
Z512 = 0xCA

TURN = 512

LEFT = XLEFT
RIGHT = XRIGHT

FORWARD = YLEFT
BACKWARD = YRIGHT

UP = ZLEFT
DOWN = ZRIGHT

XTURNS = 51
YTURNS = 43
ZTURNS = 16

# meters
XLENGTH = 0.01 * 3.90
YLENGTH = 0.01 * 3.42
ZLENGTH = 0.01 * 1.31
```

```
XSTEP = XLENGTH / (XTURNS * TURN)
YSTEP = YLENGTH / (YTURNS * TURN)
ZSTEP = ZLENGTH / (ZTURNS * TURN)

SYSTEM_RETURN = 0xFF
```

core.py

```
import serial
import numpy as np
import time
from time import sleep
from .constants import *
from threading import Thread
import serial.tools.list_ports
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

class Serial(serial.Serial):
    def __init__(self, port):
        serial.Serial.__init__(self, port = port, baudrate = BAUDRATE,
                               stopbits = serial.STOPBITS_ONE, parity =
                                   serial.PARITY_NONE,
                                  bytesize = serial.EIGHTBITS, timeout =
                                      TIMEOUT)

        self.flush()

    def send(self, hex_int):
        self.write([hex_int])
        ans = int.from_bytes(self.read(1), byteorder='big')
        if ans != SYSTEM_RETURN:
            raise Exception()

def printPorts():
    ports = list(serial.tools.list_ports.comports())
    ports = "\n".join([str(port) for port in ports])

    print("Currently␣avaiable␣ports␣are:\n%s"%ports)

def choosePort():
    while True:
        try:
            port = input("Please␣choose␣port:␣")
            serial = Serial(port)
            return serial
        except Exception as e:
            print(e)

def commandLoop(serial):
    while True:
```

```
        try:
            command = input("Command: ")
            if command == "exit":
                break
            exec("serial.send(%s)"%command)
        except Exception as e:
            pass

class Table():
    def __init__(self, port, plane = None):
        self.port = Serial(port)
        self.plane = plane

        self.ax = None
        self.fig = None
        self.dot = None
        self.currentx = 0
        self.currenty = 0
        self.currentz = 0

    def setOrigin(self):
        self.port.send(X512)
        self.port.send(Y512)
        self.port.send(Z512)

        print("Setting x...")
        for i in range(XTURNS):
            try:
                self.port.send(LEFT)
            except KeyboardInterrupt:
                break
        print("Setting y...")
        for i in range(YTURNS):
            try:
                self.port.send(BACKWARD)
            except KeyboardInterrupt:
                break
        print("Setting z...")
        for j in range(ZTURNS):
            try:
                self.port.send(DOWN)
            except KeyboardInterrupt:
                break

    def getNSteps(self, res):
        if (not type(res) is int) or (res > 9) or (res < 1):
            raise(Exception("Resolution is not an int number. Min value is
                1, max is 9."))

        return int(64 * (9 - res))
```

```
    def plot(self, plane, xmax, ymax):
        self.fig = plt.figure()
        x = np.linspace(0, xmax, 10)
        y = np.linspace(0, ymax, 10)
        x, y = np.meshgrid(x, y)
        z = plane.getZ(x, y)

        self.ax = self.fig.gca(projection='3d')
        self.ax.plot_surface(x, y, z)
        self.dot, = self.ax.plot([self.currentx], [self.currenty], [self.
            currentz], marker="o")

    def scan(self, xlength_cm, ylength_cm, xres = 1, yres = 1, zres = 5,
        plane = None):
        self.thread = Thread(target = self.scanThread, args=(xlength_cm,
            ylength_cm, xres, yres, zres, plane))
        # self.thread.daemon = True
        self.thread.start()

    def plotThread(self):
        while self.dot == None:
            sleep(1)
        plt.ion()
        while True:
            plt.pause(0.01)
            self.dot.set_data([self.currentx], [self.currenty])
            self.dot.set_3d_properties([self.currentz])
            plt.draw()

    def updatePlot(self, i):
        self.dot.set_data([self.currentx], [self.currenty])
        self.dot.set_3d_properties([self.currentz])
        return self.dot,

    def scanThread(self, xlength_cm, ylength_cm, xres = 1, yres = 1, zres
        = 5, plane = None):
        x = xlength_cm * 0.01
        y = ylength_cm * 0.01

        xsteps = self.getNSteps(xres)
        ysteps = self.getNSteps(yres)
        zsteps = self.getNSteps(zres)
        self.port.send(eval("X%d"%xsteps))
        self.port.send(eval("Y%d"%ysteps))
        self.port.send(eval("Z%d"%zsteps))

        nx = int(round((x / XSTEP) / xsteps, 0))
        ny = int(round((y / YSTEP) / ysteps, 0))

        z = 0
```

```
        xdir = RIGHT

        if self.plane != None and plane == None:
            plane = self.plane

        self.plane = plane

        if self.plane != None:
            self.plot(self.plane, x, y)

        for i in range(ny):
            if i %2 == 0: xdir = RIGHT
            else: xdir = LEFT

            for j in range(nx):
                if xdir == LEFT:
                    self.currentx = (nx - j)*xsteps*XSTEP
                self.currentx = j*xsteps*XSTEP
                self.currenty = i*ysteps*YSTEP

                if plane != None:
                    self.currentz = plane.getZ(self.currentx, self.
                        currenty)
                    z = self.currentz - z
                    nz = int(round((abs(z) / ZSTEP) / zsteps, 0))
                    for k in range(nz):
                        if z > 0:
                            self.port.send(UP)
                        else:
                            self.port.send(DOWN)
                    z = self.currentz
                self.port.send(xdir)
            self.port.send(FORWARD)
```

commandLine.py

```
from .core import printPorts, choosePort, commandLoop

printPorts()
serial = choosePort()
commandLoop(serial)
serial.close()
```

plane.py

```
import numpy as np
from .constants import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

class Plane():
```

```
    def __init__(self, n, d):
        self.n = n
        self.d = d
        self.a, self.b, self.c = self.n

    def getZ(self, x, y):
        z = self.a*x + self.b*y - self.d
        if self.c == 0:
            return -z
        else:
            return -z/self.c

def findPlane(p1, p2, p3):
    p1 = np.array(p1)
    p2 = np.array(p2)
    p3 = np.array(p3)
    p1 = p1 - p3
    p2 = p2 - p3
    n = np.cross(p1, p2)
    d = n.dot(p3)
    return Plane(n, d)
```

Ejemplos

Internal C

motor.c

```
#include <avr/io.h>
#include <stdint.h>                    // needed for uint8_t
#include <util/delay.h>

#define Mx0 _BV(PB0)
#define Mx1 _BV(PB1)
#define Mx2 _BV(PB2)
#define Mx3 _BV(PB3)

#define My0 _BV(PB4)
#define My1 _BV(PB5)
#define My2 _BV(PB6)
#define My3 _BV(PB7)

#define Mz0 _BV(PD2)
#define Mz1 _BV(PD3)
#define Mz2 _BV(PD4)
#define Mz3 _BV(PD5)

#define XLEFT 0xA0
#define XRIGHT 0xA1
```

```
#define YLEFT 0xB0
#define YRIGHT 0xB1

#define ZLEFT 0xC0
#define ZRIGHT 0xC1

#define RIGHT 1
#define LEFT 0

// UART
#define FOSC 1000000     // Clock Speed
#define BAUD 4800
#define MYUBRR (FOSC/16/BAUD -1)
#define TIMEOUT 100

uint16_t XROT, YROT, ZROT;

void delay(void)
{
    _delay_ms(2);
    _delay_us(500);
}

void rotateX(uint8_t direction)
{
    uint16_t i;
    for(i = 0; i < XROT; i++)
    {
        if(direction == RIGHT)
        {
            PORTB = Mx0;
            delay();
            PORTB = Mx1;
            delay();
            PORTB = Mx2;
            delay();
            PORTB = Mx3;
            delay();
        }
        else
        {
            PORTB = Mx3;
            delay();
            PORTB = Mx2;
            delay();
            PORTB = Mx1;
            delay();
            PORTB = Mx0;
            delay();
        }
    }
```

```
}

void rotateY(uint8_t direction)
{
    uint16_t i;
    for(i = 0; i < YROT; i++)
    {
        if(direction == RIGHT)
        {
            PORTB = My0;
            delay();
            PORTB = My1;
            delay();
            PORTB = My2;
            delay();
            PORTB = My3;
            delay();
        }
        else
        {
            PORTB = My3;
            delay();
            PORTB = My2;
            delay();
            PORTB = My1;
            delay();
            PORTB = My0;
            delay();
        }
    }
}

void rotateZ(uint8_t direction)
{   uint16_t i;
    for(i = 0; i < ZROT; i++)
    {
        if(direction == RIGHT)
        {
            PORTD = Mz0;
            delay();
            PORTD = Mz1;
            delay();
            PORTD = Mz2;
            delay();
            PORTD = Mz3;
            delay();
        }
        else
        {
            PORTD = Mz3;
            delay();
```

```
                PORTD = Mz2;
                delay();
                PORTD = Mz1;
                delay();
                PORTD = Mz0;
                delay();
            }
        }
}

void initUART(void)
{
    /*Set baud rate */
    UBRR0H = (MYUBRR >> 8);
    UBRR0L = MYUBRR;

    UCSR0B = (1 << TXEN0)| (1 << TXCIE0) | (1 << RXEN0) | (1 << RXCIE0); ;
              // Enable receiver and transmitter
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);     // Set frame: 8data, 1 stp

    DDRD &= 0b11111110; //set all of port D as inputs except for TX
}

uint8_t getChar(void)
{
    uint16_t i;
    for(i = 0; i < TIMEOUT; i++)
    {
        if (UCSR0A & (1<<RXC0)) return (char) UDR0;
    }
    return 0;
}

void sendChar(uint8_t tosend)
{
    while (( UCSR0A & (1<<UDRE0))  == 0){};
    UDR0 = tosend;
}

int main(void)
{
    DDRB = 0xFF; // all B as output
    PORTB = 0x00; // all low

    DDRD = 0xFF; // all B as output
    PORTD = 0x00; // all low

    XROT = 1;
    YROT = 1;
    ZROT = 1;
```

```
initUART();

uint8_t command;

while(1 == 1)
{
    command = getChar();

    if(command != 0)
    {
        if((command >= XLEFT) & (command < YLEFT))
        {
            command -= XLEFT;
            if(command <= 1)
            {
                rotateX(command);
            }
            else if(command == 2)
            {
                XROT = 1;
            }
            else
            {
                XROT = 64*(command - 2);
            }
        }
        else if((command >= YLEFT) & (command < ZLEFT))
        {
            command -= YLEFT;
            if(command <= 1)
            {
                rotateY(command);
            }
            else if(command == 2)
            {
                YROT = 1;
            }
            else
            {
                YROT = 64*(command - 2);
            }
        }
        if(command >= ZLEFT)
        {
            command -= ZLEFT;
            if(command <= 1)
            {
                rotateZ(command);
            }
            else if(command == 2)
            {
```

```
                    ZROT = 1;
                }
                else
                {
                    ZROT = 64*(command - 2);
                }
            }
            PORTB = 0x00;
            PORTD = 0x00;
            sendChar(0xFF);
        }
    }
    return 0;
}
```