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

En cada pequeño objeto hay un universo. Para ti, Mau.

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: 29

Fecha final:
Diciembre 3, 2017

Abstract:

Con el propósito de funcionar con detectores tales como microscopios ópticos de transmisión, fluorescencia o de reflexión, se construye un stage mecánico automatizado capaz de ser controlado desde el computador, con un área de barrido de 13 cm^2, y pasos mínimos de 1.5 μm en todas las direcciones. La velocidad mínima es de 83 μm/s, y la máxima de 176 μm/s en modo continuo. El sistema cuenta además con la posibilidad de ingresar tres puntos en foco, para obtener la ecuación de un plano y de esta forma optimizar el enfoque a lo largo del barrido.

Índice general

Capítulo 1

Introducción

La microscopia óptica representa una fracción muy importante de los desarrollos experimentales en campos de gran importancia en la física desde su invención, y se ha convertido en un instrumento vital para un científico que quiera generar aportes significativos en su campo de estudio. Teniendo en cuenta su importancia, se esperaría que el acceso a microscopios de alta potencia y desempeño estuviera a la mano de cualquiera que estuviera dispuesto a usarlo; sin embargo, existe el problema del costo elevado para el desarrollo de microscopia de alto nivel, y se presenta el reto de desarrollar microscopios de un rendimiento alto que pueda ser accesible a cualquiera en términos económicos. Se han hecho muchos esfuerzos en campos como la biología, la química, la física y la medicina con el fin de lograr el cometido de generar la visualización de alta calidad aplicando técnicas no convencionales (5) o haciendo uso de dispositivos de tamaño reducido (2, 3). Es importante remarcar que un microscopio de buena calidad no sólo se compone de una buena etapa de detección óptica, sino que en el caos de la presencia de campos de visión muy grandes se debe contar con un stage mecánico que le permita desplazarse a través de toda el área de interés, y esto se vuelve mucho más crítico cuando la resolución es alta y, por lo tanto, se debe ser muy cuidadoso en la velocidad del movimiento de esta etapa (1, 4).

Con este fin, este proyecto busca generar un stage con capacidad de muestrear la totalidad del volumen de una muestra y con bajo costo de fabricación que pueda, en conjunto con una etapa de detección óptica que se ajuste a las dimensiones del mismo, cumplir la función de un microscopio de alto desempeño para la obtención de resultados ópticos. Esto se logró a partir del desarrollo de una etapa de movimiento mecánico en tres dimensiones, que funciona gracias a la presencia de un motor encargado de desplazar el dispositivo en cada dirección, y una serie de desarrollos de código que controlan el movimiento del stage y permiten generar un plano de enfoque para tener en cuenta lo posibles desniveles de la muestra posicionada bajo la etapa de detección. El documento presenta detalladamente los

desarrollos anteriormente descritos, presenta un contexto teórico que permite al lector encontrar las bases del desarrollo, presenta resultados en forma de imágenes y métricas de desempeño del movimiento y finaliza con una discusión acerca de cómo esto podría ser implementado y presenta una serie de conclusiones de todo el trabajo realizado.

Marco teórico

Dentro de las piezas de mayor importancia de un microscopio óptico se encuentra el stage. En la gran mayoría de los casos estos no son motorizados, restringiendo la reproducibilidad de estudios que requieran estudiar el área total de la muestra. Por otro lado la manipulación de un stage, en caso de no realizarse con el debido cuidado puede dar lugar a la descalibración de partes sensibles del microscopio. La velocidad con la que un ser humano puede manipular el stage puede ser relativamente alta, sin embargo está lejos de ser constante, razón por la cual cuando se quieren hacer estudios de larga exposición o que requieran el movimiento constante de la muestra es necesario un stage automatizado.

Figura 1.1: Stage mecánico convencional. Modificado de (1).

Si bien la automatización permite resolver varios problemas, también se debe tener en cuenta la susceptibilidad a otros. En el caso de un stage en x y y automatizado, al barrer gran cantidad de área se vuelve necesario una preparación de la muestra exhaustiva para limitar cambios significativos en el grosor de la misma que afecten el enfoque sobre el microscopio. Por esta razón se plantea la construcción del stage con una motorización adicional en la dirección z.

Existen tres tipos de motores de fácil acceso en el mercado local. Cada uno de ellos con ventajas y desventajas respecto a los demás.

- **Motores de escobillas:** Son los más comunes en el mercado, pues se usan con frecuencia en juguetes, electrodomésticos y accesorios para computadores. Entre sus ventajas se encuentra la velocidad de rotación de los mismos y

el torque que generan. Son operados con voltajes análogos. Su desventaja consiste en el control preciso de la posición.

- **Servo motores:** Los servomotores son usados en aplicaciones donde no se requieren rotaciones completas, pero sí precisión en la ubicación del mismo. Para su manipulación es necesario el uso de frecuencia modulada, es decir realizar pulsaciones sobre salidas digitales. Su gran desventaja es la incapacidad de realizar rotaciones mayores a 180 ° y el poco torque que ejercen.
- **Motores de paso:** Este tipo de motores no presenta escobillas, usando un imán y una serie de bobinas separadas entre sí, es posible determinar la orientación del mismo, logrando que este se quede en esa posición hasta que la bobina vecina sea activada. Son ideales para aplicaciones que requieran control de la posición con torques moderados, sin embargo su velocidad de rotación está restringida a la frecuencia de pulsación de cada bobina.

Figura 1.2: Motores adquiridos para la realización del proyecto. De izquierda a derecha: motor de escobillas, servo y de paso.

Finalmente y con el objetivo de realizar el control de los distintos motores, se debe usar un microcontrolador, el cual permita establecer comunicación con el computador para la recepción de instrucciones, y la habilidad de generar los pulsos necesarios para la activación de determinado motor. Entre los requerimientos que este debe tener se encuentran los puertos UART, los cuales permitirán llevar a cabo la comunicación por puerto serial con el computador.

Capítulo 2

Fabricación y Resultados

Primer modelo

Tres modelos distintos fueron propuestos para la fabricación del stage. El primero hace uso del sistema de lectura propio de los discos compactos, tales como el CD, DVD y Blu-Ray. Este sistema cuenta con movimiento en dos direcciones, propios de coordenadas cilíndricas. El movimiento en θ relacionado con traslación en el plano y en z para realizar el enfoque sobre la muestra.

Usando dos sistemas como el descrito anteriormente y condicionando el experimento a ángulos pequeños es posible obtener un stage aproximado con movimiento x, y y z.

$$\begin{aligned} x = r\cos\theta \approx r\left(1 - \frac{1}{2}\theta^2\right) \\ y = r\sin\theta \approx r\theta \end{aligned} \qquad \text{donde } r \approx 90 \text{ mm} \tag{2.1}$$

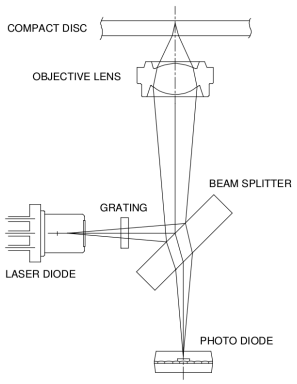

Figura 2.1: Sistema de detección implementado por la primera versión del stage, el cual hace uso del sistema de lectura de un CD.

Según el fabricante la distancia máxima de movimiento sobre el plano es de $\pm$0,5 mm, dando en total 1,0 mm para translación. En el eje z la distancia máxima es de $\pm$ 0.7 mm, para un total de 1,4 mm, sin embargo la sensibilidad es de sólo el 20 %. Usando este stage y aprovechando el interior de los sistemas de lectura de DVDs, se realizó esquema de detección por reflexión, análogo a un microscopio confocal salvo que no existen pinholes el cual se muestra en la Figura 2.1.

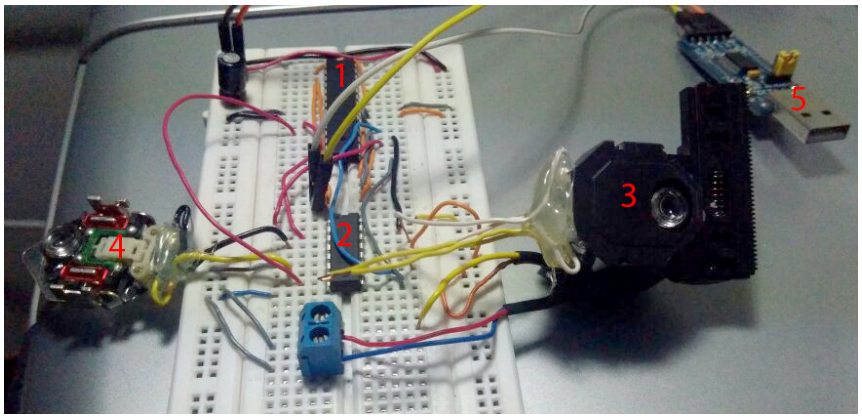

Figura 2.2: Circuito implementado para el primer stage con etapa de detección. (1) Microcontrolador (Atmega328). (2) Puente H, etapa de potencia para los motores. (3) Sistema óptico principal con laser y movimiento en x. (4) Sistema óptico con movimiento en y. (5) Comunicación UART.

El sistema se muestra en la Figura 2.2. Posicionando el elemento 3 sobre el 4 usando como ayuda una prensa, a la altura donde más pequeño se observó el punto del láser fue posible obtener la imagen que se muestra en la Figura 2.3.

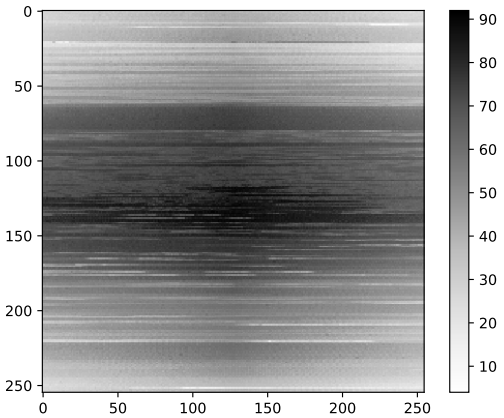

Figura 2.3: Resultados obtenidos usando el primer stage. Se observa la primera línea horizontal de la letra E impresa sobre papel.

El papel con la letra fue tomado de una factura, y se dispuso en la parte superior del lente que se observa en 4, sin ningún tipo de adhesión. De la imagen es importante resaltar que el sistema de detección implementado no cuenta con algún tipo de amplificación, ni de resta del fondo, por lo cual no es posible usar la totalidad de los 10 bits disponibles para la conversión de la señal análoga a digital, y se obtiene únicamente un rango de 0 a 90 valores distintos, lo cual corresponde a 7 bits. Además las condiciones de iluminación del cuarto pudieron alterar los resultados de la medición. En este sentido de implementar un nuevo sistema en el futuro es necesario considerar un sistema de resta en la señal y posterior amplificación de la misma.

Posteriormente y con el objetivo de brindar apoyo en otro proyecto además de simplificar el posicionamiento de la muestra, fueron propuestos dos modelos alternativos.

Segundo modelo

El segundo modelo propuesto carece de movimiento en la dirección z, sin embargo permite un aislamiento del ruido de los motores dado que los mismos se encuentran alejados de la muestra. El movimiento es entonces obtenido al conectar el sistema con los motores usando correas de transporte, los motores serían de paso, conectados al motoreductor que se encuentra a la derecha en la Figura 2.4, con el objetivo de reducir el tamaño del paso de cada uno de ellos.

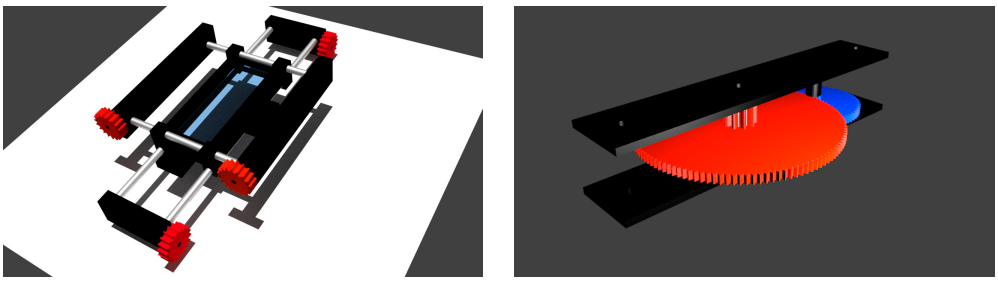

Figura 2.4: Segundo stage propuesto.

El segundo modelo no fue construido en físico dada la dificultad que se encontró para fabricar las piezas del motoreductor, y su diseño fue reemplazado por el último modelo.

Tercer modelo

Este último fue construido a partir de los planos del Manipulador de BackyardBrains, el cual es de acceso libre, con la posibilidad de realizar modificaciones por los usuarios. Para esto fue necesario la construcción de las piezas del eje y y z, dado que las tuercas que se adquirieron eran de dimensiones mucho mayores. Además las bases de los motores fueron diseñadas con el objetivo de obtener un manipulador motorizado. Las piezas fueron diseñadas en Blender, software de uso libre.

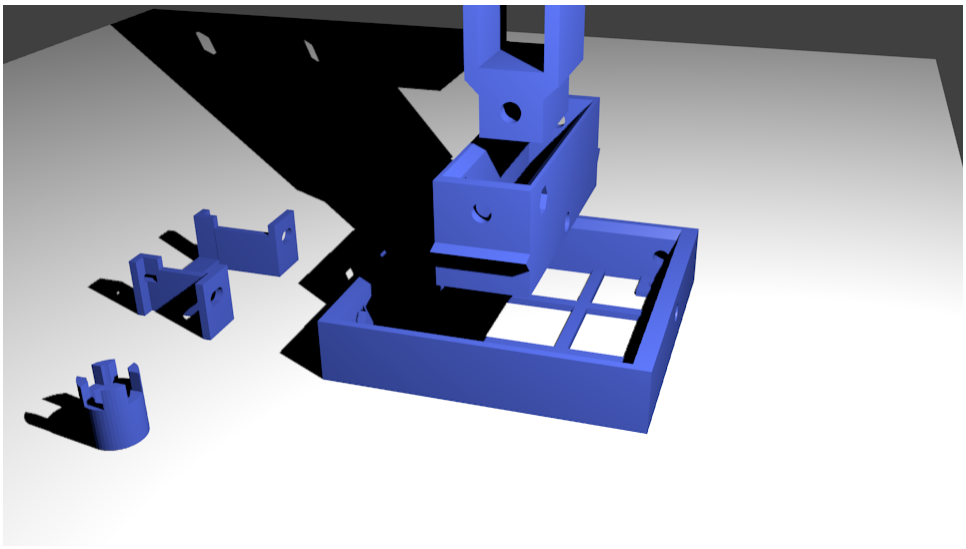

Figura 2.5: Tercer stage propuesto.

Además de las piezas de impresión, también fue necesario adquirir:

- 12 Tornillos M3 de 10 mm.
- 2 Tornillos 15/32"de 100 mm.
- 1 Tornillo 15/32"de 50 mm.
- 12 Tuercas M3.
- 6 Tuercas 15/32".

Una vez montado el sistema puede moverse las siguientes distancias sobre cada eje, las cuales fueron medidas usando un calibrador Vernier:

Cuadro 2.1: Capacidad en cada dirección del stage.

Eje	Distancia (cm)
x	$3{,}42 \pm 0{,}01$
y	$3{,}90 \pm 0{,}01$
z	$1{,}31 \pm 0{,}01$

La electrónica es simple, y únicamente fueron necesarios 3 arreglos de transistores Darlington ULN2003, un adaptador de 12 V, y tres motores de paso 28BYJ. El microcontrolador es el mismo usado en el primer modelo, el cual tiene las siguientes tareas:

1. Comunicación con el computador del usuario.
2. Control de los motores.

El protocolo de comunicación UART establecido envía un mensaje en cada lado de la transmisión de 8 bits, a 4800 baudios. A cada instrucción que envía el computador, el microcontrolador responde `0xFF` (255 en hexadecimal). Usando el sistema hexadecimal, se usan como primer dígito letras para determinar la identidad del canal, como se muestra en la Tabla 2.2.

Cuadro 2.2: Identificación de los canales en el protocolo UART.

Valor	Descripción
A	Canal X
B	Canal Y
C	Canal Z

Para cada eje es necesario conocer la dirección en la que se desea hacer rotar al motor, la cual es obtenida con el segundo dígito del número hexadecimal.

Cuadro 2.3: Asignación de la tarea por UART.

Valor	Descripción
0	Derecha
1	Izquierda
2	Pasos: 1
3	Pasos: 64
4	Pasos: 128
5	Pasos: 192
6	Pasos: 256
7	Pasos: 320
8	Pasos: 384
9	Pasos: 448
A	Pasos: 512

Adicionalmente es posible decirle al microcontrolador que realice varias pasos al motor sin requerir comunicación adicional, es importante decir que el número de pasos para que el motor complete una vuelta es de 512. Con el objetivo de obtener resultados sobre el sistema se realizan mediciones del tiempo requerido para ir y volver hasta el máximo de cada dirección del stage, para varios valores de la resolución (inversos del número de pasos).

En todos los casos se observaron variaciones inferiores al 0.2 % para resoluciones menores a 9, para el caso de la máxima resolución las variaciones son menores a 1.3 % en todas las dimensiones, siguiendo la misma forma en todos los casos. Para cada medición se realizaron triplicados, los cuales mostraron desviaciones menores al 0.01 %.

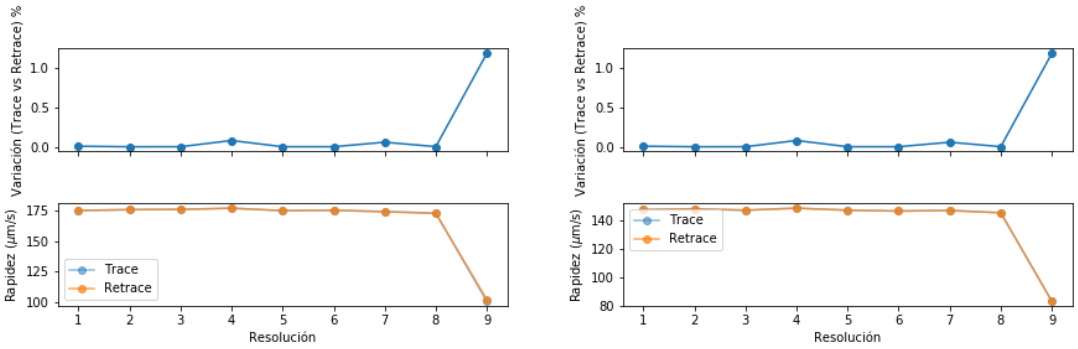

Figura 2.6: Variación y rapidez en función de la resolución para el eje x y y, correspondientemente.

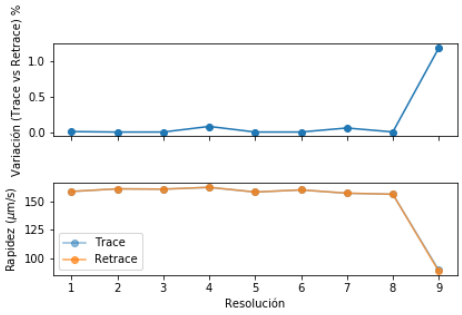

Figura 2.7: Variación y rapidez en función de la resolución para el eje z.

Finalmente y con el objetivo de optimizar el enfoque de un detector sobre una muestra se incluyó en la librería la posibilidad de ingresar 3 puntos (x, y, z) en donde se cumple que la muestra está enfocada, de esta forma es posible obtener un plano que optimiza el enfoque a lo largo de un barrido. Para cada punto del barrido el sistema calcula si debe o no modificarse la altura, por lo que el plano termina en una función de escalones como los que se observan en la Página 13, en donde cada cuadro representa las posiciones de medición, el circulo rojo muestra la posición actual del stage.

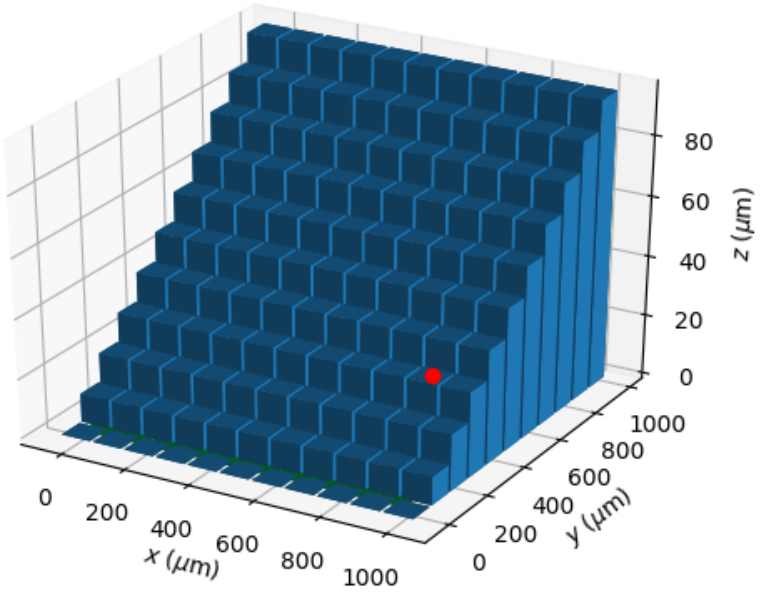

Figura 2.8: Variación y rapidez en función de la resolución para el eje z.

Capítulo 3

Conclusión

Como conclusión de todo el proceso desarrollado se puede entonces asegurar que se construyó un stage de microscopia capaz de hacer barridos en tres dimensiones de una muestra determinada de manera constante y teniendo en cuenta consideraciones de resolución según la etapa de detección a usar. Adicionalmente, el código mediante el cual se pone ene ejecución el proceso considera la toma de tres puntos distribuidos en el área de interés para calcular una ecuación del plano, y por lo tanto tener en cuenta cualquier clase de desnivel en el plano de muestra. Como importante característica se puede remarcar que el stage funciona de tal manera que las medidas se pueden tomar con una velocidad que permita que se tome punto por punto de manera cuidadosa o que se haga un recorrido continuo en las tres dimensiones para abarcar una muestra de tamaño superior en una sola ejecución. Se espera a futuro poder generar una mayor cantidad de pruebas que permitan validar su funcionamiento, mejorar la velocidad de barrido probando una denominación de motor diferente, y como aporte más importante se espera hacer un acople exitoso de la parte mecánica con una etapa de detección con el fin de obtener resultados ópticos y tener un microscopio de bajo costo funcionando en su totalidad. Adicionalmente el proyecto cuenta con una librería para Python, la cual es de acceso libre y puede ser modificada por cualquiera interesado en la misma.

Juan Barbosa
js.barbosa10@uniandes.edu.co
https://github.com/jsbarbosa
Universidad de los Andes
Bogotá, Colombia

Bibliografía

[1] Mortimer. Abramowitz and Michael W. Davidson. Microscope Stages, 2015. URL https://micro.magnet.fsu.edu/primer/anatomy/stage.html.

[2] Ahmet F. Coskun, Ting-Wei Su, and Aydogan Ozcan. Wide field-of-view lens-free fluorescent imaging on a chip. *Lab on a Chip*, 10(7):824, 2010. ISSN 1473-0197. doi: 10.1039/b926561a. URL http://xlink.rsc.org/?DOI=b926561a.

[3] Alon Greenbaum, Wei Luo, Ting-Wei Su, Zoltán Göröcs, Liang Xue, Serhan O Isikman, Ahmet F Coskun, Onur Mudanyali, and Aydogan Ozcan. Imaging without lenses: achievements and remaining challenges of wide-field on-chip microscopy. *Nature Methods*, 9(9):889–895, aug 2012. ISSN 1548-7091. doi: 10.1038/nmeth.2114. URL http://www.nature.com/doifinder/10.1038/nmeth.2114.

[4] Oliver Kim. What are the advantages of a mechanical stage?, 2008. URL Whataretheadvantagesofamechanicalstage?

[5] Yu Shrike Zhang, Jae-Byum Chang, Mario Moisés Alvarez, Grissel Trujillo-de Santiago, Julio Aleman, Byambaa Batzaya, Vaishali Krishnadoss, Aishwarya Aravamudhan Ramanujam, Mehdi Kazemzadeh-Narbat, Fei Chen, Paul W. Tillberg, Mehmet Remzi Dokmeci, Edward S. Boyden, and Ali Khademhosseini. Hybrid Microscopy: Enabling Inexpensive High-Performance Imaging through Combined Physical and Optical Magnifications. *Scientific Reports*, 6 (1):22691, sep 2016. ISSN 2045-2322. doi: 10.1038/srep22691. URL http://www.nature.com/articles/srep22691.

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
X192 = 0xA5
X256 = 0xA6
```

[1] https://github.com/jsbarbosa/miniscope

[2] https://pypi.python.org/pypi/mauscope

```
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 = 43
YTURNS = 51
ZTURNS = 16

# micrometers
XLENGTH = 1e4 * 3.42
YLENGTH = 1e4 * 3.90
ZLENGTH = 1e4 * 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 Stage():
    def __init__(self, port, plane = None):
        if type(port) is str:
            self.port = Serial(port)
        else:
            self.port = port
        self.plane = plane

        self.ax = None
        self.fig = None
```

```
        self.dot = None
        self.currentx = 0
        self.currenty = 0
        self.currentz = 0

        self.break_ = False

    def setOrigin(self):
        self.port.send(X64)
        self.port.send(Y64)
        self.port.send(Z64)

        print("Setting x (Ctrl c if set)...")
        for i in range(8*XTURNS):
            try:
                self.port.send(LEFT)
            except KeyboardInterrupt:
                break
        print("Setting y (Ctrl c if set)...")
        for i in range(8*YTURNS):
            try:
                self.port.send(BACKWARD)
            except KeyboardInterrupt:
                break
        print("Setting z (Ctrl c if set)...")
        for j in range(8*ZTURNS):
            try:
                self.port.send(DOWN)
            except KeyboardInterrupt:
                break

    def move(self, Nsteps, dir_):
        for i in range(Nsteps):
            self.port.send(dir_)

    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."))

        val = int(64 * (9 - res))
        if val == 0:
            return 1
        return val

    def plot(self, plane, x, y, z):
        self.fig = plt.figure()

        self.ax = self.fig.gca(projection='3d')
        self.ax.plot_surface(x, y, z, color = "g")

        xstep = x[0, 1] - x[0, 0]
        ystep = y[1, 0] - y[0, 0]

        x = x.ravel()
        y = y.ravel()
        z = z.ravel()
        bottom = np.zeros_like(z)
```

```
        if len(z) < 200:
            self.ax.bar3d(x - 0.5*xstep, y- 0.5*ystep, bottom, xstep*0.9, ystep
                *0.9, z, shade=True, alpha = 0.9)
        self.dot, = self.ax.plot([self.currentx], [self.currenty], [self.currentz],
             marker="o", color = "r")

        self.ax.set_xlabel("$x$ ($\mu$m)")
        self.ax.set_ylabel("$y$ ($\mu$m)")
        self.ax.set_zlabel("$z$ ($\mu$m)")

    def scan(self, xlength_um, ylength_um, xres = 8, yres = 8, zres = 9, plane =
        None):
        self.thread = Thread(target = self.scanThread, args=(xlength_um, ylength_um
            , xres, yres, zres, plane))
        self.thread.start()

    def plotThread(self):
        while self.dot == None:
            sleep(1)
        while self.thread.is_alive():
            try:
                ani = FuncAnimation(self.fig, self.updatePlot)
                plt.show()
            except KeyboardInterrupt:
                self.break_ = True
                break
            else:
                pass

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

    def setXStepsPerCall(self, n):
        self.port.send(eval("X%d"%n))

    def setYStepsPerCall(self, n):
        self.port.send(eval("Y%d"%n))

    def setZStepsPerCall(self, n):
        self.port.send(eval("Z%d"%n))

    def scanThread(self, x, y, xres, yres, zres, plane):
        xsteps = self.getNSteps(xres)
        ysteps = self.getNSteps(yres)
        zsteps = self.getNSteps(zres)

        self.setXStepsPerCall(xsteps)
        self.setYStepsPerCall(ysteps)
        self.setZStepsPerCall(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:
            xmesh = np.arange(0, x, xsteps*XSTEP)
            ymesh = np.arange(0, y, ysteps*YSTEP)
            xmesh, ymesh = np.meshgrid(xmesh, ymesh)
            zmesh = plane.getZ(xmesh, ymesh)
            self.plot(self.plane, xmesh, ymesh, zmesh)

        for i in range(ny):
            if i%2 == 0: xdir = RIGHT
            else: xdir = LEFT
            self.currenty = i*ysteps*YSTEP
            for j in range(nx):
                jj = j
                if xdir == LEFT: jj = nx - j - 1
                if plane != None:
                    self.currentz = zmesh[i, jj]
                    z = zmesh[i, jj] - 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)

                        if self.break_:
                            return None

                    z = self.currentz
                self.currentx = jj*xsteps*XSTEP
                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;
}
```