Índice

Objetivo

La característica principal de este proyecto es el poder construir un dispositivo electrónico capaz de generar un haz de luz *láser,* el cual muestre el nivel proyectado en una superficie (ej: una pared).

Para que el usuario no necesite calibrar ni ajustar el nivel de forma manual, el dispositivo lo hará automáticamente. Una vez encendido, éste tomará la información sobre las características del suelo (mediante un giróscopo/acelerómetro), y rotará el *láser* para que el nivel proyectado sea el correcto.

Para mayor facilidad en su uso, de ser necesario, se podrá acoplar el dispositivo a un trípode, el cual se adaptará a las irregularidades del terreno.

Listado de componentes

Para describir en más detalle el funcionamiento del dispositivo, se detallan a continuación las partes que lo componen:

Activos

- Microcontrolador AVR Atmel - 8 bits (ATmega2560)
- Giróscopo/Acelerómetro - MPU-6050
- Módulo *láser* - HLM1230-5mW-CROSS
- Motor *stepper* - 28BYJ-48-5v
- *Driver* para el motor - ULN2003A
- Fuente de alimentación - 9V 800mA FJ-SW7280900800DA

Pasivos (acordados con el profesor a cargo)

- Resistencias de valor 4K7 ohms (X3)
- Transistores 2N3904

Descripción

Ya en funcionamiento, y colocado apuntando a la superficie donde se desea marcar un nivel, el microcontrolador AVR Atmel (en este caso usaremos el que trae la plataforma *Arduino MEGA*), comenzará a obtener los datos suministrados por el giróscopo/acelerómetro. Luego de realizar los cálculos correspondientes para saber cuáles son las correcciones a aplicar, se enviará una señal al motor "*paso a paso*" para corregir la inclinación del *láser*.

Mientras, se seguirá sensando con el giróscopo/acelerómetro para obtener las nuevas coordenadas de inclinación y poder corroborar que las correcciones hechas fueron las correctas. De seguir inclinado, se volverán a realizar los cálculos pertinentes y a mandar al motor la señal para moverlo y terminar de ajustar el *láser.*

Diagrama de conexiones en bloques

Una primera ilustración sobre el diagrama en bloques, se puede encontrar en la siguiente figura (Fig.1), donde se detallan las conexiones principales entre los distintos componentes.

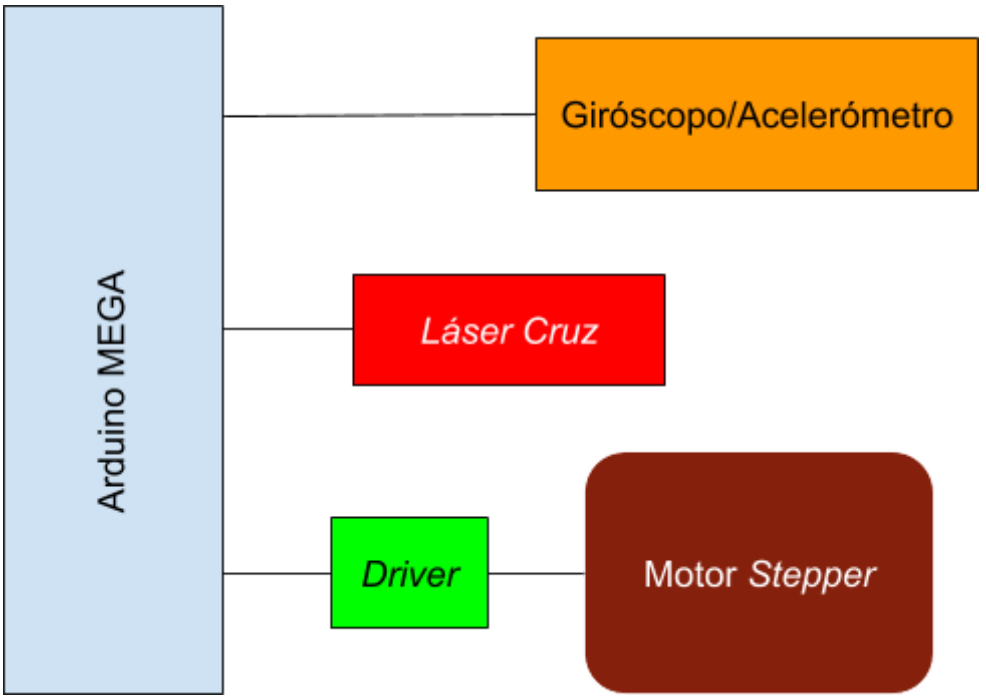

Fig.1

Como se puede apreciar, el microcontrolador se conecta con el dispositivo que contiene los sensores de movimiento (giróscopo y acelerómetro), para obtener las características del terreno. Luego, a partir de cálculos en base a estos datos, se le envían la señal apropiada al *driver* del motor "*paso a paso*", para que éste gire el láser a la posición correcta.

En una segunda ilustración (Fig.2), se puede encontrar un diagrama en bloques que muestra en detalle las conexiones que se realizaron entre los distintos componentes, teniendo en cuenta la alimentación, microprocesadores y sensores.

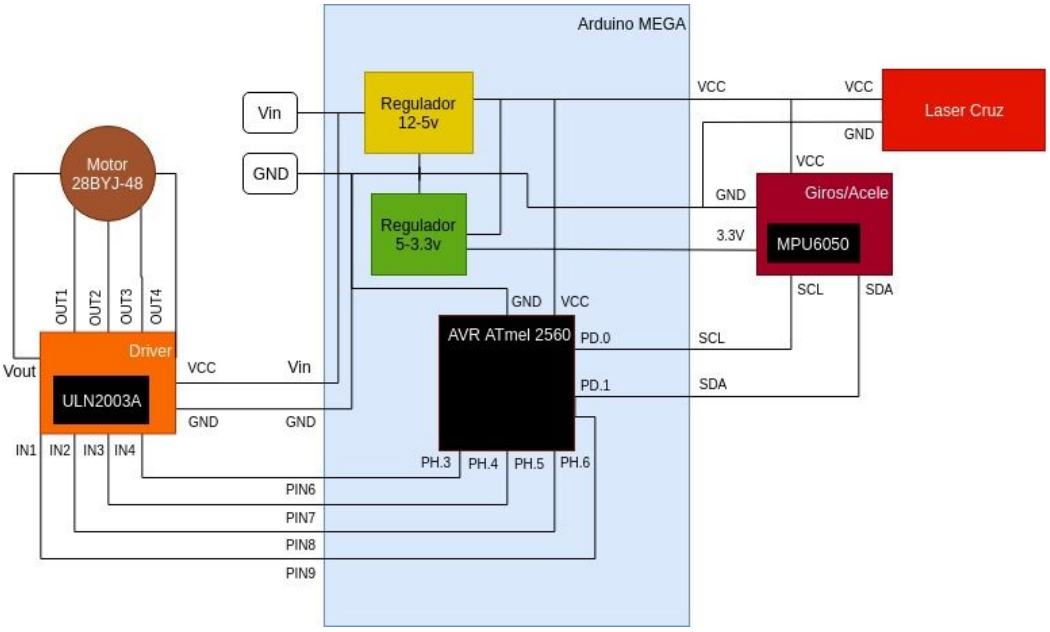

Fig.2

En esta imagen, se puede ver cómo el *Arduino*, utiliza una fuente externa de alimentación (en este caso es de 9V - 800mA max). EL driver que controla al motor *stepper* 28BYJ-48-5v, recibe la secuencia correspondiente de los pines 6 al 9, los cuales se mapean los pines 3 al 6 del puerto H (PORTH). Por otro lado, el controlador del giróscopo/acelerómetro funciona con la salida de 5V proporcionado por el *Arduino*, y también con la salida de 3.3V (ver esquemático para mayor detalle). Como utiliza el protocolo *I2C* para comunicarse con el microcontrolador, recibe SCL y SDA de los correspondientes pines del *Arduino*, los cuales están mapeados a los pines 0 y 1 del puerto D (PORTD) del AVR-ATmega2560, correspondientemente.

Cabe mencionar que físicamente el motor está acoplado tanto al giróscopo/acelerómetro como al láser, para que cuando se detecta una variación en el sensor del MPU-6050, el microcontrolador pueda darle la señal correspondiente al motor hasta que el desnivel inicial sea eliminado por completo. En otras palabras, se mueven solidariamente el láser y el giróscopo/acelerómetro debido al accionar del motor.

Circuito esquemático

En las siguientes figuras (Fig.3, Fig.4 y Fig.5) se pueden observar los circuitos esquemáticos correspondientes a tres partes que conforman el proyecto: el Arduino MEGA, el controlador del motor “paso a paso”, y el circuito que contiene al giróscopo/acelerómetro.

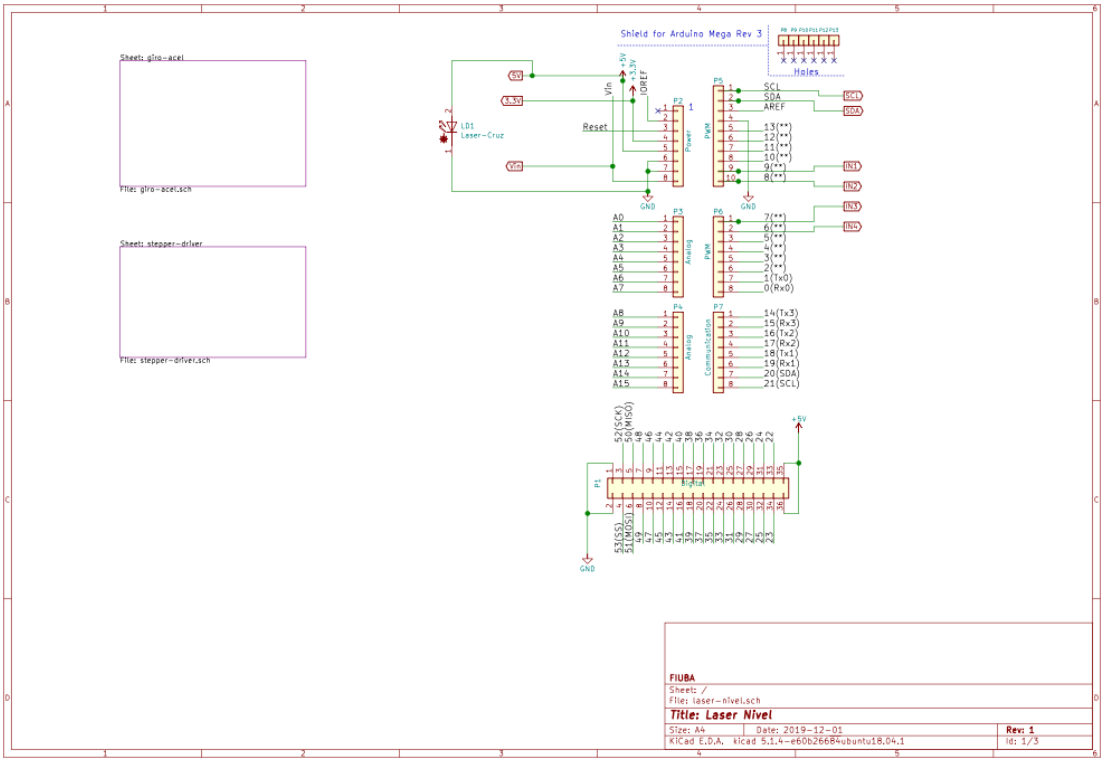

Fig.3 Esquemático de las conexiones realizadas con el Arduino MEGA

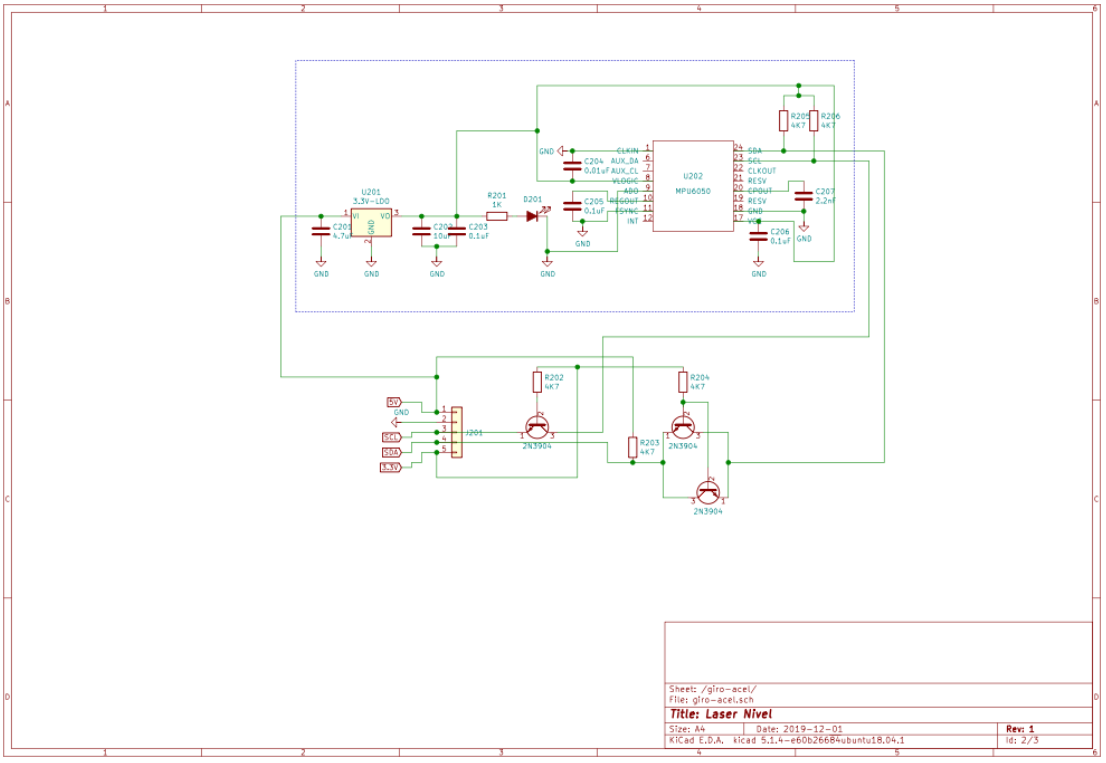

Fig.4 Esquemático del circuito que contiene el microcontrolador MPU-6050

Los valores de las resistencias (R202, R203 y R304) utilizadas por fuera de las que ya vienen en la placa comercial del giróscopo/acelerómetro MPU-6050, son de 4k7 ohms. Los transistores utilizados son 2N3904. Todos estos valores fueron validados y aconsejados por el profesor a cargo del proyecto.

La motivación de este circuito es la de regular el voltaje suministrado por cada uno de los pines del Arduino (SCL y SDA) que en este caso son de 5V, para convertirlos a 3.3V que es el voltaje que acepta internamente el microcontrolador MPU-6050. La placa comercial ya trae uno, pero este sólo afecta a la alimentación del mismo y no así a las entradas SCL y SDA para la comunicación con el microcontrolador AVR ATmega2560 de la placa Arduino.

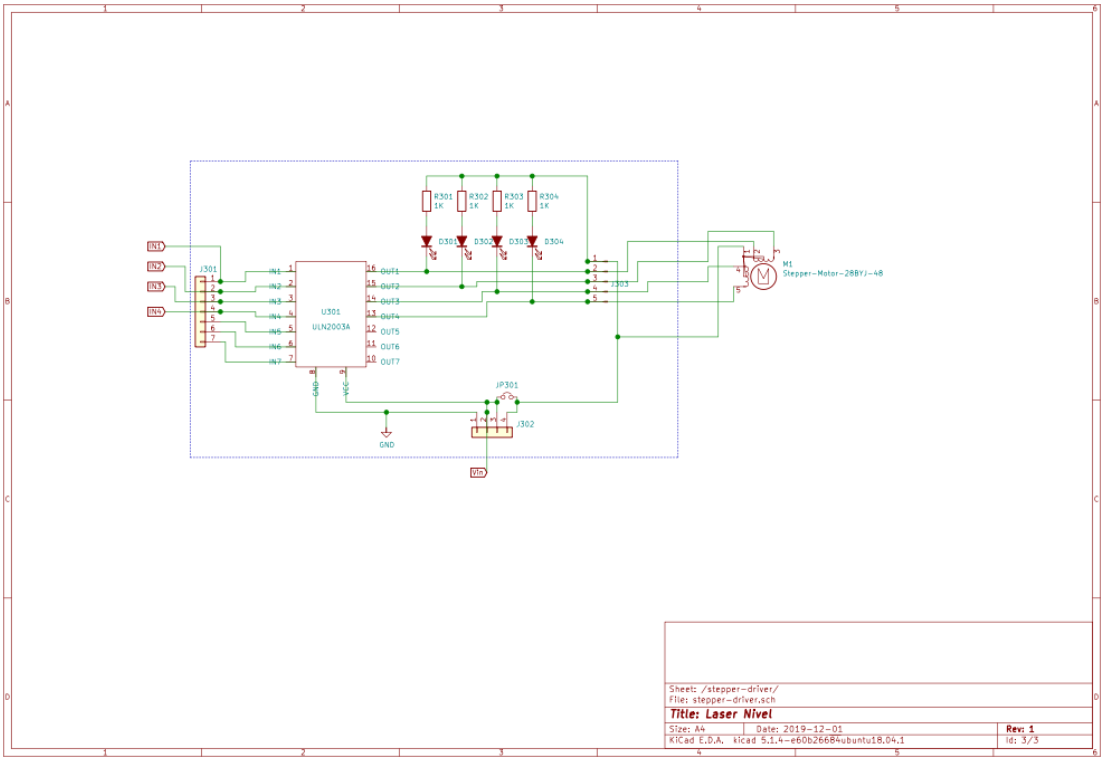

Fig.5 Esquemático del circuito que contiene las conexiones con el integrado ULN2003A

Software - Diagrama de flujo

En el la figura 6 se ve un diagrama de flujo donde se muestra a alto nivel el programa implementado para la solución de este trabajo práctico.

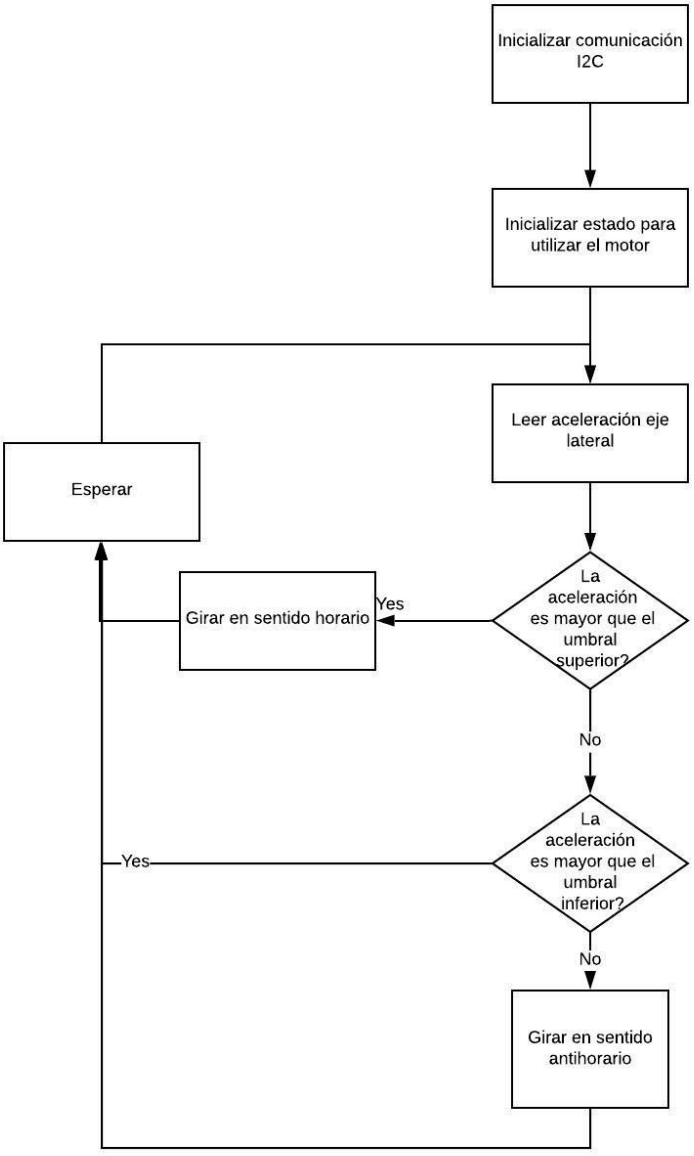

Fig. 6. Diagrama de flujo del programa utilizado.

Como se puede ver el programa inicializa la comunicación I2C con el giróscopo para luego poder hacer la lectura de los valores medidos por el ultimo y tambien se realiza una inicialización de los registros a utilizar para el control del motor.

Luego de esto el programa simplemente mide la aceleración en el eje que se horizontal perpendicular al láser y dependiendo del valor y signo de la componente que

tenga la aceleración gravitatoria tenga sobre dicho eje el programa determina en qué posición está el giróscopo, y por consiguiente el láser. Esa posición podemos separarla en tres grandes categorías y actuar en base a esto. Por un lado podría suceder que el láser está inclinado en sentido anti-horario, si esto sucede, la componente de la aceleración en el eje de interés será grande y positivo(es decir, desde un punto de vista práctico sería mayor que un umbral superior), en cuyo caso el microcontrolador indicará al motor que debe corregir esta desviación girando en sentido horario. Por otro lado, podría suceder que el acelerómetro está suficientemente nivelado(que es considerado suficientemente nivelado depende de los umbrales utilizados, por lo que es posible modificarlo a través de la modificación los umbrales, donde umbrales muy cercanos resultaron en una nivelación más estricta y menos laxa pero puede resultar en que el sistema nunca quede en reposo), en cuyo caso simplemente se opta por no realizar otra acción y reiniciar el loop. Finalmente, si la aceleración gravitatoria está sobre el mismo eje pero en el otro sentido(a niveles prácticos resultando en que la aceleración medida resulte menor que el umbral inferior) significa que el giróscopo y el láser están inclinados en sentido horario por lo que el motor, por orden del microcontrolador, deberá corregirlo girando en sentido antihorario.

Por no tener la presencia de eventos asincrónicos la necesidad de utilizar interrupciones es nula y por lo tanto decidimos no utilizarlas ya que complejizan el proyecto en vano. Los posibles motivos o funciones que sugieren la posible utilidad de una interrupción son, a nuestro entender, solamente dos, sin embargo estas terminan siendo inválidas por los motivos desarrollados a continuación.

Por un lado, se podría pensar la lectura como algo asincrónico y esperar que ante un cambio en la medición de los valores de intereses el microcontrolador reciba una interrupción, sin embargo esto no es posible ya que el giróscopo/acelerómetro utilizado no envía los valores leídos directamente a través de una serie de pines sino que los comunica, a pedido del microcontrolador, a través de la interfaz I2C, como ya se mencionó. Por lo que la utilización de interrupciones son impracticables o muy poco útiles ya que antes de detectar el cambio se requiere el trabajo previo del microcontrolador, resultando en que si la interrupción debiera dispararse esto se conocería una vez terminado el trabajo de este resultando en que el evento no sea asincrónico, finalmente haciendo que las ventajas de utilizar una interrupción desaparezcan.

Por otro lado, podríamos haberla utilizada en lugar de las subrutinas de espera a través de los timers del microcontrolador. Una de las ventajas de la utilización del timer es cuando se buscan tener tiempos prolongados donde no se realiza cómputo, se requiere mucha autonomía (por lo que se debe utilizar intensamente el modo sleep del microcontrolador) o mientras el tiempo transcurre queremos seguir utilizando el microcontrolador para algún otro propósito. En nuestro caso, ninguno de los tres

escenarios se presenta por lo que la utilización de los timers tampoco traería ninguna ventaja significativa.

Las subrutinas utilizadas son las siguientes:

- **setup**: Esta función no recibe parámetros ni devuelve ningún resultado. Esta función debe ser definida con el trabajo necesario a realizar al inicio del sistema, como por ejemplo seteo de registros como entrada o salida. Esta idea fue tomada de la plataforma de desarrollo de Arduino, y en nuestro caso inicializa la comunicación con el giróscopo y el estado interno para trabajar con el Stepper motor.
- **loop**: Esta función no recibe parámetros ni devuelve ningún resultado. Esta función debe ser definida con el trabajo necesario a realizar una vez terminada con la fase de inicio, siendo esta función llamada repetidas veces hasta que el microcontrolador es apagado. Esta idea fue tomada de la plataforma de desarrollo de Arduino, y en nuestro caso lee los valores pertinentes del giróscopo y hace movimientos a través del stepper motor, si fuera necesario, como fue descripto en esta misma sección.
- **i2c_init**: Esta función no recibe parámetros ni devuelve ningún resultado. Se encarga de iniciar la comunicación con el giróscopo a través de la interfaz I^2C, en particular seteando los pines correspondientes para habilitar la comunicación, realizar un chequeo WHO_AM_I con el giróscopo y finalmente habilitar el giróscopo para la lectura de datos.
- **i2c_start**: Esta función no recibe parámetros ni devuelve ningún resultado. Se encarga de iniciar una comunicación volátil y temporal con el giroscopo, seteando los registros necesarios para que el microcontrolador sea quien escribe los datos por el momento.
- **i2c_connect**: Esta función recibe como parámetro el registro del cual se desea leer información. No devuelve ningún resultado. Establece la conexión con el dispositivo enviándole dicha dirección para que éste sepa de qué registro leer el próximo byte.
- **i2c_init_read:** Esta función no recibe parámetros ni devuelve ningún resultado. Se encarga de inicializar la lectura del giròscopo al enviarle al giroscopo un flag indicando que a partir de ese momento el microcontrolador solo actuará como lector de los datos.
- **i2c_read:** Esta función devuelve a través de un registro el byte leído y recibe de igual manera un parámetro que indica si esta es la última lectura a realizar o se debe continuar. Con este parámetro, la subrutina manda un flag u otro al giróscopo para indicar si deberá preparar más datos para enviar o el que se recibió fue el último.
- **get_rotation**: Esta función no recibe parámetros. Se encarga de establecer una conexión temporal con el giroscopo, leer los valores de rotación alrededor de los tres ejes, cada valor formado por parte alta y parte baja, y guardar dichos valores en un lugar reservado para esto en la memoria.

- **get_acceleration**: Esta función no recibe parámetros. Se encarga de establecer una conexión temporal con el giroscopo/acelerómetro, leer los valores de aceleración de los tres ejes, cada valor formado por parte alta y parte baja. y guardar dichos valores en un lugar reservado para esto en la memoria.
- **delay_1**: Esta función no recibe parámetros ni devuelve ningún resultado. La única función que tiene esta rutina es la de lograr que el microcontrolador no continúe con la ejecución del resto del programa sino hasta que haya pasado un segundo, logrando esto con loops anidados para ejecutar muchas veces las mismas pocas líneas.
- **mili_delay_1**: Esta función no recibe parámetros ni devuelve ningún resultado. La única función que tiene esta rutina es la de lograr que el microcontrolador no continúe con la ejecución del resto del programa sino hasta que haya pasado un milisegundo, logrando esto con loops anidados para ejecutar muchas veces las mismas pocas líneas.
- **mili_delay_100**: Esta función no recibe parámetros ni devuelve ningún resultado. La única función que tiene esta rutina es la de lograr que el microcontrolador no continúe con la ejecución del resto del programa sino hasta que haya pasado cien milisegundo, logrando esto con loops anidados para ejecutar muchas veces las mismas pocas líneas.
- **stepper_init**: Esta función no recibe parámetros ni devuelve ningún resultado. En particular, esta subrutina se encarga de inicializar los pines que se utilizarán para controlar el motor como pines de salida.
- **stepper_move**: Esta función no devuelve ningún resultado. Esta función se encarga de girar el motor una cantidad de pasos recibido como parámetros y en el sentido que indique el otro parámetro recibido. Esta función simplemente contiene un loop que llama repetidas veces a la siguiente.
- **one_step**: Esta función no devuelve ningún resultado. Esta función se encarga de girar el motor solo un paso en la dirección que reciba como parámetro. Esto lo logra a través de recorrer una tabla de símbolos que mandarle al stepper motor para que este se mueva, manteniendo una variable en la memoria para recordar el paso en el que quedó la anterior llamada a one_step. La tabla mencionada puede ser recorrida en un sentido o en el otro resultando en que el motor gire en un sentido o en el contrario.

Conclusiones

Los resultados obtenidos son satisfactorios en relación a que efectivamente el láser se autonivela cuando los terrenos de prueba están inclinados con respecto a la horizontal. Y aún más, experimentalmente se puede apreciar que el error cometido por el microcontrolador del MPU-6050 (giróscopo/acelerómetro) no es perceptible a simple vista.

Finalmente, si se quisiera mejorar la sensibilidad/precisión en la detección de la inclinación se podría optar por lo siguiente: la primera podría ser mejorar el algoritmo de detección actual, el cual consiste en comparar el valor de una de las componentes del acelerómetro contra un umbral (*threshold*) y así decidir si realizar un paso del motor (ya sea izquierda o derecha), o no; la segunda, y teniendo en cuenta la primera, sería utilizar un dispositivo de medición más preciso que el MPU-6050, el cual posee una precisión de ±2g hasta ±16g (la cual es programable por el usuario, actualmente seteada por default en ±2g).

Como nota al margen sobre la placa de desarrollo utilizada, *Arduino MEGA,* se podría utilizar el *Arduino UNO* sin cambio alguno (en cuanto al software y también las conexiones del hardware, sólo habría que modificar los puertos a utilizar), pero se decidió utilizar el primero ya que uno de los integrantes del equipo poseía uno.

Apéndice

- **Hojas de Datos**
  - Microcontrolador: AVR ATmega2560 (*datasheet*, *instruction-set*)
  - Placa de desarrollo: Arduino MEGA (*datasheet, register-mapping*)
  - Giróscopo/Acelerómetro: MPU 6050
  - Driver motor: ULN2003A
  - Motor: Stepper 28BYJ-48
  - Láser cruz: HLM1230

- **Compilación/Programación**

El presente proyecto fue desarrollado para el sistema operativo *Linux*, utilizando las herramientas de desarrollo proporcionadas por los repositorios oficiales de la distribución utilizada (en este caso *Ubuntu/debian*) y se puede compilar mediante el comando ***make***. También, se puede realizar la carga a la placa de desarrollo *Arduino* mediante el comando ***make upload***.

- **Cálculo de los costos**
  - Placa de desarrollo: Arduino MEGA
    - U$S 15.00
  - Giróscopo/Acelerómetro: MPU 6050
    - U$S 5.00
  - Driver motor: ULN2003A + Motor: Stepper 28BYJ-48
    - U$S 5.00
  - Láser cruz: HLM1230
    - U$S 6.00
  - Fuente de alimentación
    - U$S 7.00

- **Cálculo de los tiempos en las rutinas de "*delay*"**

Tendiendo en cuenta que la placa de desarrollo *Arduino MEGA* funciona a una frecuencia de 16 MHz, los cálculos se realizarán en base a este valor. La estructura del algoritmo de *delay* por la cual se optó es la siguiente:

```
delay:
    push r17
    push r18
    push r19

    ldi r17, X
for1:
    ldi r18, Y
for2:
    ldi r19, Z
for3:
    dec r19
    brne for3
endfor3:
    dec r18
    brne for2
endfor2:
    dec r17
    brne for1
endfor1:
    pop r19
    pop r18
    pop r17
    ret
```

Luego, teniendo en cuenta los parámetros constantes X, Y y Z y que las instrucciones utilizadas consumen las siguientes cantidades de ciclos:

- *dec*: 1 ciclo
- *brne*: 1 ciclo si la condición es **Falsa**, 2 ciclos si la condición es **Verdadera**

Se analiza el patrón de ejecución de las instrucciones previas, de la siguiente forma:

- "X" brne (for1) - salida por *True*
- "X" dec (for1)
- "X" brne (for2) - salida por *False*
- "X" x "Y" brne (for2) - salida por *True*
- "X" x "Y" dec (for2)
- "X" x "Y" brne (for3) - salida por *False*
- "X" x "Y" x "Z" dec (for3)
- "X" x "Y" x "Z" brne (for3) - salida por *True*

Las salidas por *True* se deben multiplicar por 2 como se mencionó anteriormente ya que consumen 2 ciclos.

Ahora, de acuerdo a los valores empleados en los *delays*, realizando los cálculos mencionados y sumando cada uno de los resultados parciales, se va a tener lo siguiente:

- *delay_1*:
  - Parámetros: X=80, Y=255, Z=255
  - Resultado: 15.687.920 y dividido por 16.000.000 se obtiene 0.98 segundos.
- *mili_delay_1*:
  - Parámetros: X=2, Y=51, Z=51
  - Resultado: 16022 y dividido por 16.000.000 se obtiene 1.0013 milisegundos.
- *mili_delay_100*:
  - Parámetros: X=200, Y=51, Z=51
  - Resultado: 1602200 y dividido por 16.000.000 se obtienen 100,1375 milisegundos.

Que son tiempos aceptables para los que se manejan en el proyecto.