

Índice

Índice	1
Objetivo	2
Listado de componentes	3
Activos	3
Pasivos	3
Descripción	4
Diagrama de conexiones en bloques	5
Circuito esquemático	7
Software - Diagrama de flujo	10
Conclusiones	14
Apéndice	15
Hojas de Datos	15
Compilación/Programación	15
Cálculo de los costos	15
Cálculo de los tiempos en las rutinas de “delay”	16
Configuración del “Timer-1”	18

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

Estos valores fueron acordados con el profesor a cargo del proyecto.

- 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.

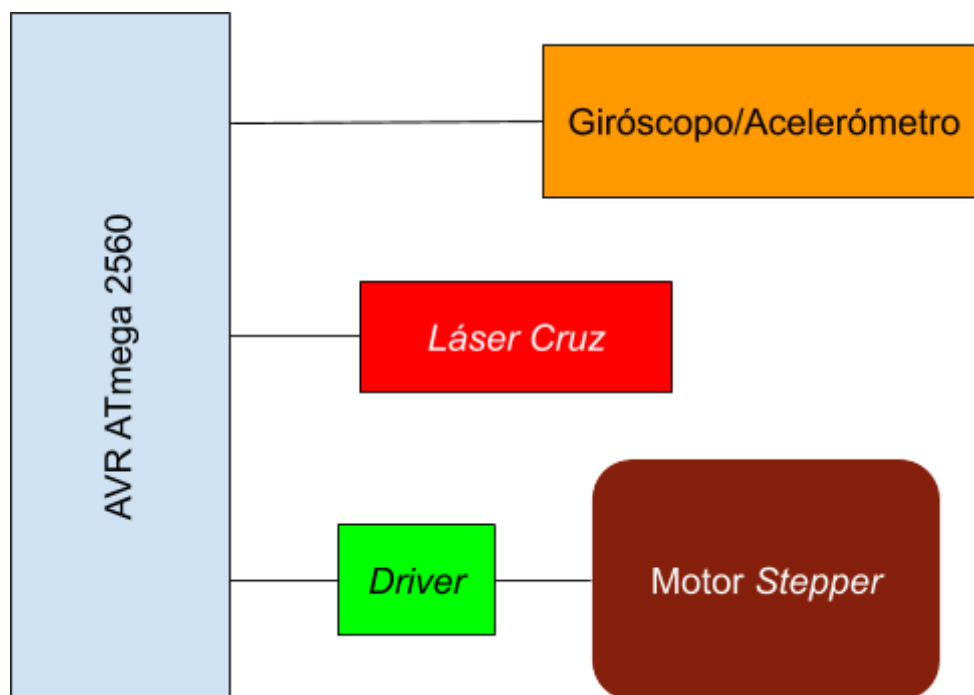


Fig.1

Como se puede apreciar, el microcontrolador ("AVR ATmega2560") se conecta con el dispositivo ("MPU-6050") (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* ("ULN2003A") del motor "paso a paso" ("28BYJ-48"), 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.

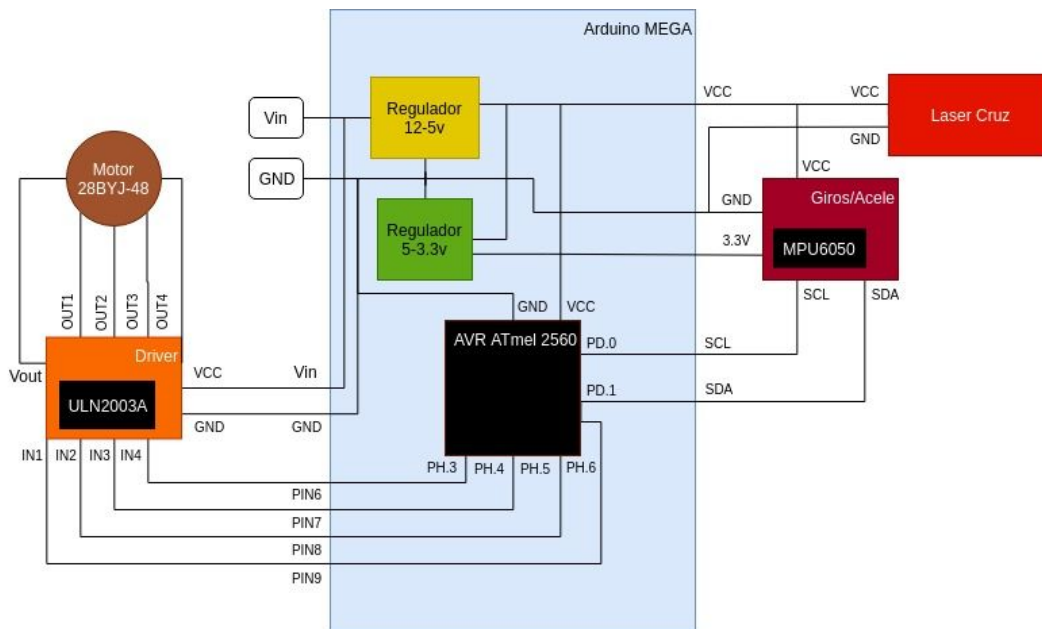


Fig.2

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.

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.

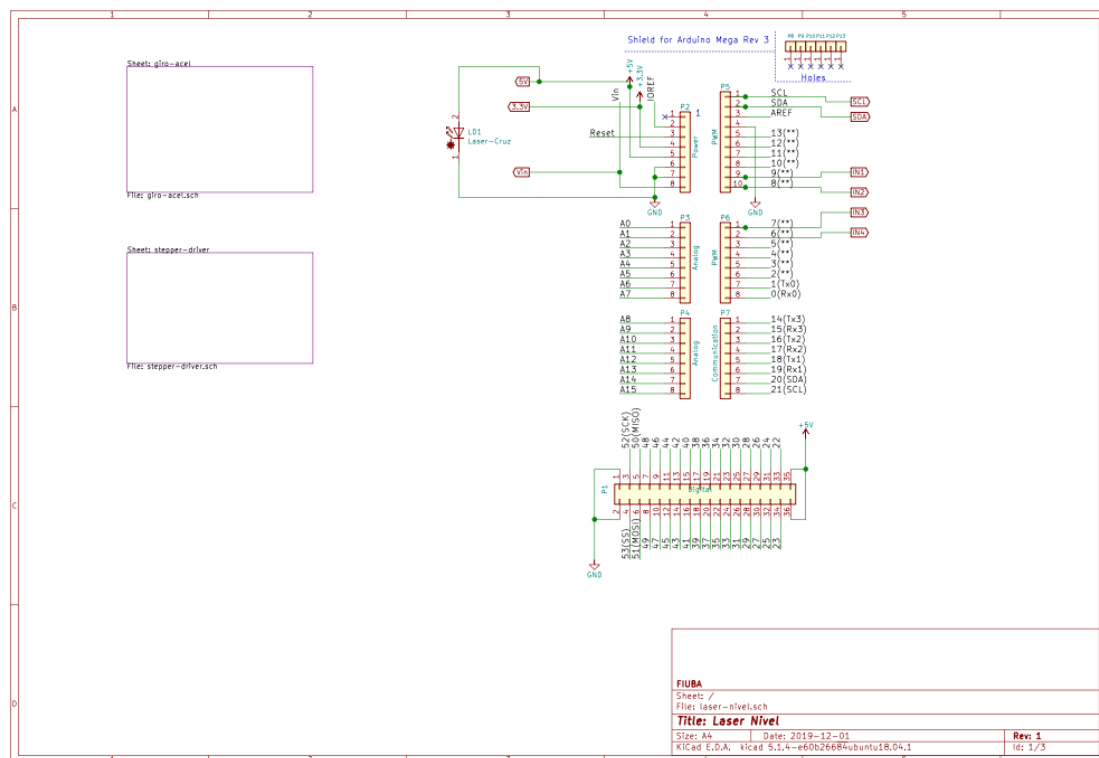


Fig.3 Esquemático de las conexiones realizadas con el Arduino MEGA

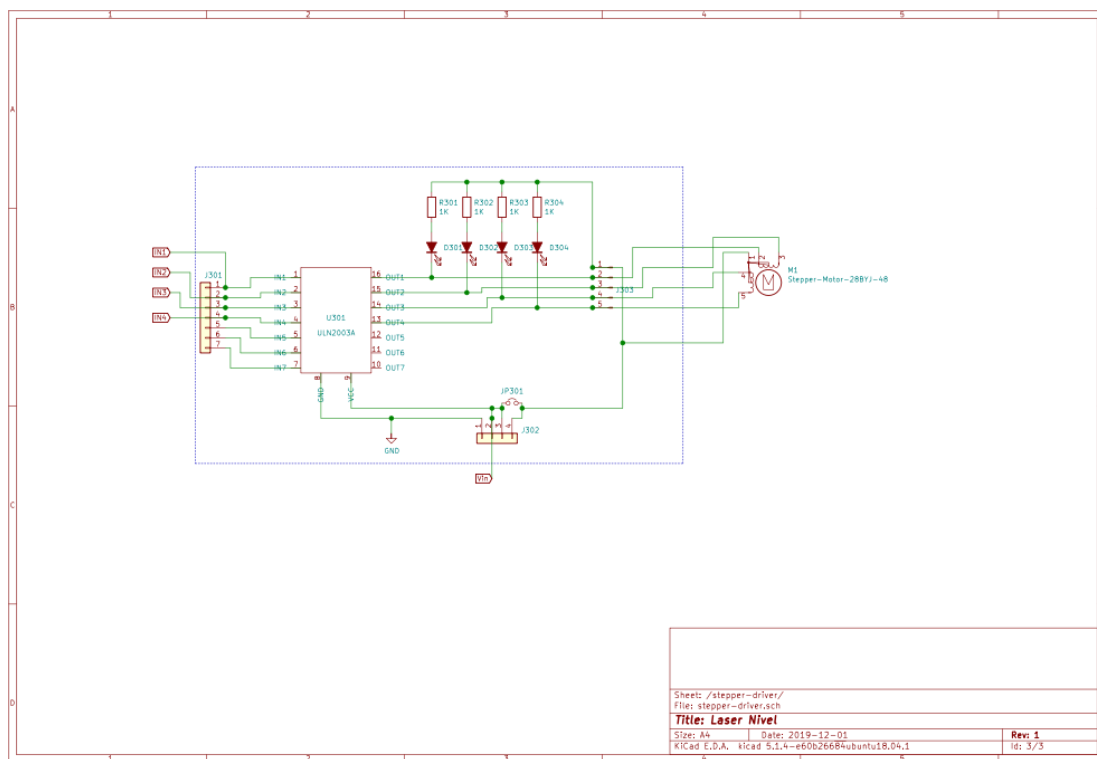


Fig.5 Esquemático del circuito que contiene las conexiones con el integrado ULN2003A

Software - Diagrama de flujo

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

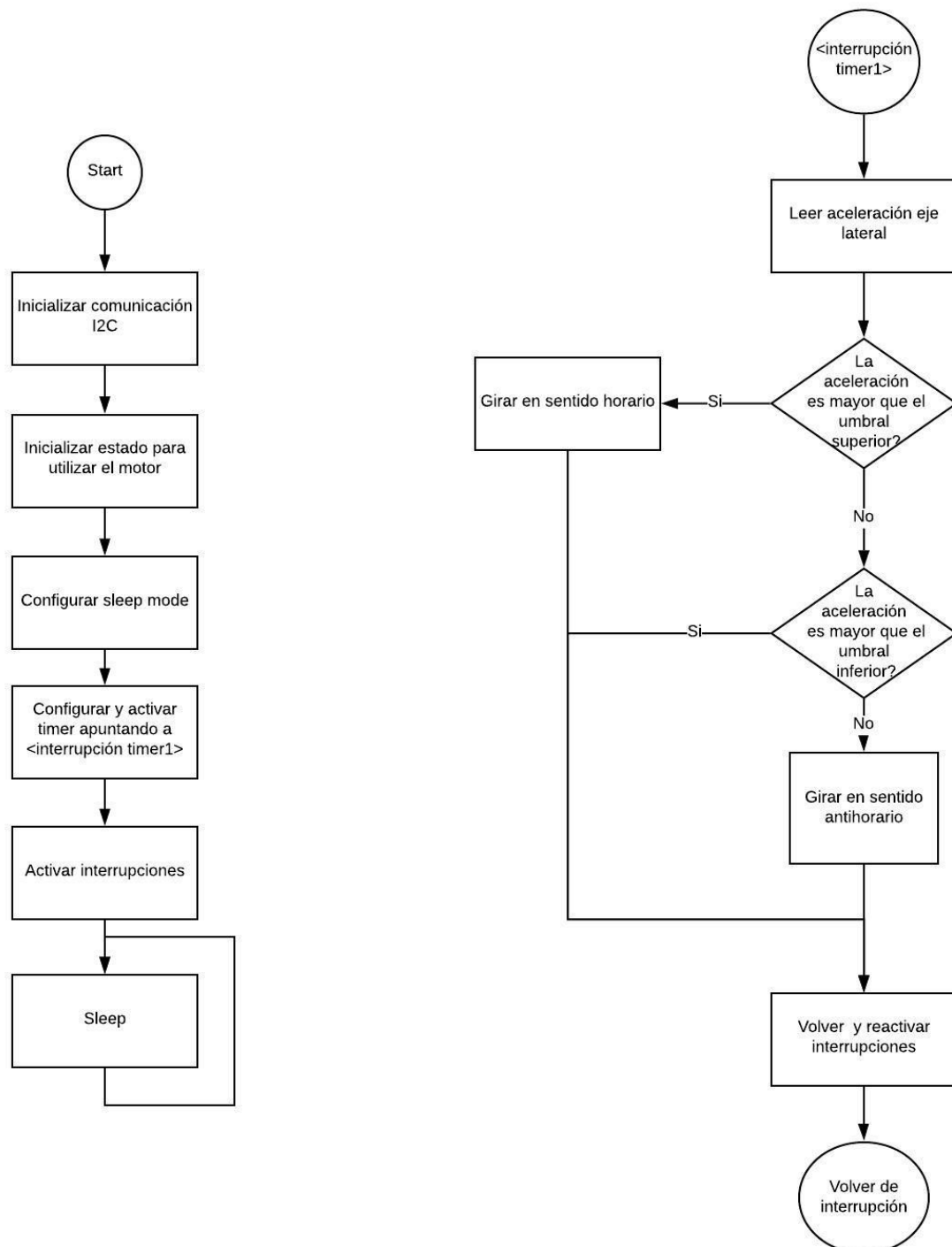


Fig. 6. Diagrama de flujo del programa utilizado.

Como se puede ver, el programa inicializa la comunicación I2C con el giróscopo/acelerómetro para luego poder hacer la lectura de los valores medidos. También, se realiza una inicialización de los registros a utilizar para el control del motor. Finalmente, se configura el modo de la instrucción “*sleep*” (en este caso usamos el modo: “*idle*”, que nos permite utilizar el *timer-1*) al cual se va a pasar cuando no se esté moviendo al motor; se configura el *timer-1* (ver Apéndice, sección *Configuración del Timer-1*) para que genere una interrupción por *overflow* (cada vez que se llega al máximo de cuentas), se activan las interrupciones, y por último se coloca al microcontrolador en modo “*sleep*”.

Luego, cada vez que se genere una interrupción del *timer-1* el flujo de ejecución de la ISR (Interrupt Service Routine) será el que se muestra en la parte derecha de la Fig.6, y que se describe a continuación con mayor detalle:

Se mide la aceleración en el eje que es horizontal y perpendicular al láser y dependiendo del valor y signo que la componente de 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 esté 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 esté suficientemente nivelado (qué es considerado suficientemente nivelado depende de los umbrales utilizados, por lo que es posible modificarlo a través de la configuración de los umbrales. Donde umbrales muy cercanos resultaron en una nivelación más estricta y menos laxa pero que puede resultar en que el sistema nunca quede en reposo), en cuyo caso simplemente se opta por no realizar otra acción y terminar la rutina de interrupción.

Finalmente, si la aceleración gravitatoria está sobre el mismo eje pero en el otro sentido (a niveles prácticos resultando en que la aceleración medida sea 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.

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 descrito 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 I²C, 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 giróscopo 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.
- **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.
- **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 $\pm 2g$ hasta $\pm 16g$ (la cual es programable por el usuario, actualmente seteada por default en $\pm 2g$).

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 al microcontrolador a través de 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

Total: U\$S 38.- (equivalente a \$ 2500 aprox)

Cálculo de los tiempos en las rutinas de “*delay*”

Tendiendo en cuenta que el microcontrolador AVR *ATMega2560* 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:

- mili_delay_1:
 - Parámetros: X=2, Y=51, Z=51
 - Resultado: 16022 y dividido por 16.000.000 se obtiene 1.0013 milisegundos.

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

Configuración del “*Timer-1*”

La configuración del “timer-1” es la más sencilla que se puede establecer haciendo uso del prescaler y activando las interrupciones cuando se supera la cuenta máxima, es decir, cuando se genera un overflow del contador. Para realizar ésto se utilizaron los siguientes registros:

- TCCR1B - *Timer/Counter 1 Control Register B*
 - Bits 2:0 - *CS1 2:0: Clock Select*

Utilizamos la combinación en binario “011” para seleccionar el *prescaler* de 64.

- TIMSK1 - *Timer/Counter 1 Interrupt Mask Register*
 - Bit 0 - *TOIE1: Timer/Counter 1, Overflow Interrupt Enable*

Luego, teniendo en cuenta la selección del *prescaler*, el período de ejecución de cada una de las interrupciones se puede calcular de la siguiente forma:

$$T = \frac{presc * TCNT1}{F_{ck}} \Leftrightarrow \frac{64 * 65536}{16 * 10^6} \Leftrightarrow T = 262 \text{ mseg}$$