

Galileo's Finger. Dispositivo de bajo costo para la enseñanza de astronomía de Hardware y Software Libre. Implementación y modificaciones.

Díaz, Javier Francisco
javierd@linti.unlp.edu.ar

Lanfranco, Einar
einar@linti.unlp.edu.ar

Bogado, Joaquín
jbogado@linti.unlp.edu.ar

Traberg, Gastón
gtraberg@cert.unlp.edu.ar

1. Resumen

Este trabajo versa sobre la implementación y las modificaciones realizadas al dispositivo de diseño abierto conocido como “*Galileo's Finger*” [1]. Se trata de un dispositivo que consta de un puntero láser montado sobre un cabezal con dos ejes motorizados, el cual se comunica a una computadora mediante un puerto USB. Un programa de simulación estelar [2] se ejecuta en la computadora y envía comandos al cabezal. Una vez que el operador selecciona el objeto que le interesa en el software este indica a los motores como posicionarse para marcar el objeto con el láser en el firmamento.

Tanto el software que se ejecuta en la computadora como el firmware cargado en la electrónica del dispositivo son Software Libre [3], al igual que el diseño del cabezal es Hardware Libre [4].

Palabras clave: Astronomía, Hardware Libre, Software Libre, Educación

2. Motivación y arte previo

El desconocimiento del cielo nocturno por parte del público general es cada día más patente. La contaminación lumínica de las grandes ciudades tampoco contribuye a que día a día nos preguntemos que es lo que vemos en el firmamento porque simplemente, los objetos más tenues quedan ocultos a nuestros sentidos. Desde el punto de vista tecnológico, existen varios paquetes de software que pueden complementar la enseñanza de conceptos de astronomía básicos, ya sea a niños y adolescentes

como al público general.

Este trabajo se basó en su mayor parte en los trabajos de Keegan Crankshaw, Brendan Ardagh y Sven Steinbauer. Crankshaw y Ardagh presentaron una primera versión del “*Galileo's Finger*” como una publicación en el sitio Hackaday [5]. Allí describen el proyecto como “un puntero láser motorizado en dos ejes, controlado vía Stellarium”. También dejan claro el objetivo educacional del proyecto: “Al ser una herramienta abierta, esperamos sea usada para enseñar las bases de astronomía y que los niños [...] se vean interesados por la ciencia y la tecnología”.

Este trabajo se encuentra enmarcado en el proyecto de investigación del LINTI [6], “Innovación en TICs para el desarrollo de aplicaciones en educación, inclusión, gobierno y salud”.

3. Funcionamiento general

3.1. El software

Stellarium es un programa de simulación de la esfera celeste, multiplataforma, distribuido bajo licencias libres. Una vez configurado el lugar de observación, la fecha y la hora actual, muestra en una ventana una representación gráfica del firmamento tal como se ve desde el lugar de observación. Cuenta con un catálogo de objetos sumamente extenso, el cual puede incrementarse descargando archivos desde Internet.

Stellarium soporta varias personalizaciones, como ser configurar las condiciones del cielo, por ejemplo, indicando el grado de contaminación lumínica, ocultar o mostrar la atmósfera, usar cuadrícula

alt-azimutal o ecuatorial y otras líneas guía, puntos cardinales, etc. También soporta plugins y es a través de uno llamado *Stellarium Scope* [7] que *Stellarium* puede conectarse a un telescopio con montura robotizada utilizando diversos protocolos de comunicación o una conexión de red.

Una vez seleccionado el objeto, *Stellarium* envía sus coordenadas (ascensión recta y declinación¹) a una conexión de red. Un script de Python llamado *tcpTelescope.py* las recibe a través de un socket y las traduce a un sistema de coordenadas absolutas (altitud y azimut), para redirigirlas luego al dispositivo a través de la interfaz USB/Serie. Esta última traducción dependerá del número de pasos por vuelta de los motores.

Por último, un programa residente en la electrónica del dispositivo (basada en *Arduino* [8] e *EasyDriver* [9]) recibe esta posición absoluta y determina la dirección y la cantidad de pasos que deben realizar los motores, dependiendo de la posición actual.



Figura 1: Diagrama de comunicación entre las diferentes componentes de software del proyecto.

3.2. El hardware

La electrónica del dispositivo esta basada en *Arduino*, utilizándose para este trabajo:

- Una placa *Arduino Nano* v3.0 en su versión con procesador ATmega328 con electrónica de 5 voltios. La alimentación de corriente es tomada directamente del puerto USB (5V a 500mA) de la computadora.

¹El sistema de coordenadas de Ascensión Recta y Declinación o RA/DEC por sus siglas en inglés, permite ubicar objetos en el firmamento, independientemente del lugar y fecha de observación. Este sistema se contrapone con el sistema altazimutal o Alt/Az, el cual si depende del lugar, fecha y hora de la observación.

- Dos motores de paso bipolares NEMA 17 de 0.9° por paso son controlados mediante dos *EasyDriver* v 4.4. La alimentación de dichos motores puede estar separada de la alimentación del *Arduino* para darles más potencia. Los controladores *EasyDriver* son controlados a su vez por el firmware que se ejecuta en el procesador del *Arduino*.
- Un cabezal láser verde (532nm) de 50mW. El firmware del *Arduino* también controla el encendido y apagado de un cabezal láser verde (532nm) de 50mW. Esta es una modificación al proyecto original y permite que el láser no esté prendido durante el movimiento del cabezal.
- Una rueda de madera de cedro: El cabezal de dos ejes consta de una rueda cortada en madera de cedro montada directamente sobre uno de los motores. Esta rueda es la responsable del movimiento en azimut. Sobre ella se monta el otro motor y el eje de altitud. Y además, sobre el mismo eje de altitud se monta el láser. Con libertad en ambos ejes, es posible apuntar el láser hacia cualquier dirección sobre el horizonte.

En la figura 2 puede verse un diagrama del conexionado de los diferentes componentes de hardware del proyecto. Este esquema fue realizado utilizando la herramienta de software libre Fritzing [10]. En el sistema real se debe reemplazar el LED por el puntero láser, pero la lógica del sistema es exactamente la misma.

4. Modificaciones al proyecto original

Para este trabajo se realizaron varias modificaciones al proyecto original en varios niveles, incluyendo software, firmware y hardware.

4.1. Modificaciones al software

La primera modificación fue realizada a nivel de software, sobre el programa que recibe los datos de la posición del objeto desde *Stellarium*. Este programa llamado *tcpTelescope.py* [11] esta escrito en *Python* y fue desarrollado originalmente por Sven

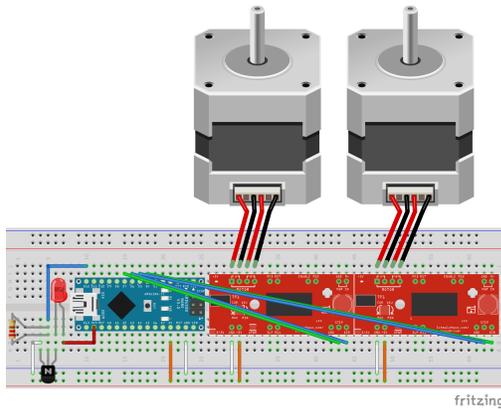


Figura 2: Conexionado de los diferentes componentes. En lugar de un LED se colocan los conectores del puntero láser. El diagrama fue realizado utilizando la herramienta Fritzing App [10].

Steinbauer [12]. Posteriormente Keegan Crankshaw agregó las rutinas para comunicar las coordenadas al *Arduino* a través del puerto USB emulando una conexión serial. El programa original calcula las coordenadas en altitud y azimut a partir de las coordenadas ecuatoriales que envía el *Stellarium*. Dos funciones, *azToArd* y *altToArd* calculan la posición absoluta del objeto en base a la cantidad de pasos de los motores y posteriormente envían por el puerto serial (emulado por medio de la conexión USB) lo siguiente:

- el carácter 'x' o 'y', dependiendo de la función *azToArd* o *altToArd* respectivamente.
- una sucesión de caracteres 's', uno por cada paso que debe dar el motor.
- el carácter 'e', indicando que se terminaron de enviar todos los pasos.

Este método es bastante ineficiente puesto que si se deben dar 300 en una dirección y 200 en otra dirección, no deben enviarse menos de 500 bytes por el puerto serie. Además, dado que son funciones diferentes, hasta que no se terminan de enviar todos los pasos de un motor, no se envían los del otro, lo que se traduce en que primero se mueve un motor y luego el otro y no los dos a la vez.

Para solucionar esto, se reescribió el protocolo de comunicación de forma tal que siempre se envían la misma cantidad de bytes. Además se reemplazaron

las funciones de comunicación *azToArd* y *altToArd* por una única función que calcula la posición absoluta del objeto en relación a la cantidad de pasos por vuelta de los motores y retorna el arreglo de bytes a enviar. El mensaje esta compuesto por un arreglo de 6 bytes con la siguiente estructura:

- el carácter 'x'
- dos bytes indicando la posición absoluta en azimut en formato little endian.
- el carácter 'y'
- dos bytes indicando la posición absoluta en altitud en formato little endian.

Esto independiza el movimiento de los motores del envío de las coordenadas, permitiendo otras optimizaciones del lado del firmware, como la posibilidad de mover ambos motores a la vez, de detener el movimiento de los motores ante la llegada de una nueva coordenada o de una señal de parada.

Además se agregaron controles para impedir que el láser sea apuntado a objetos por debajo del horizonte.

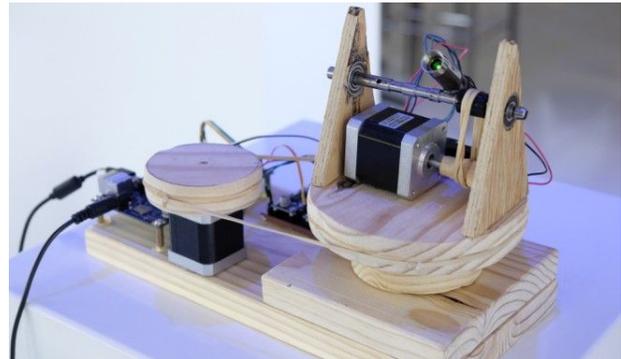


Figura 3: Fotografía del proyecto original por Keegan Crankshaw y Brendan Ardagh.

4.2. Modificaciones al firmware

Dadas las modificaciones al protocolo de comunicación, fue necesario modificar el firmware que se ejecuta en el *Arduino* para que reciba y procese los datos de manera acorde. Es responsabilidad del firmware recibir e interpretar los datos enviados por el puerto serie (emulado sobre USB), determinar la dirección del movimiento para los motores a partir de la posición absoluta

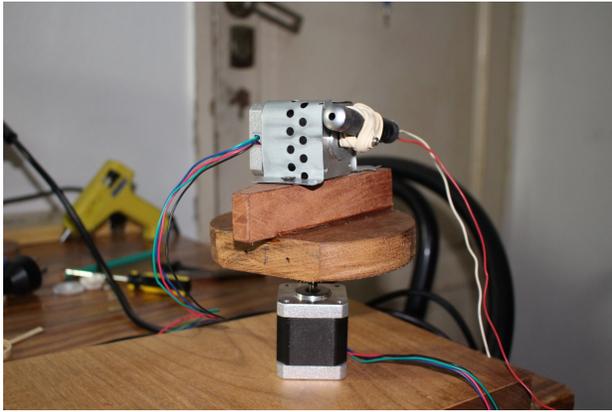


Figura 4: Las modificaciones realizadas al cabezal.

enviada y la posición actual y de enviar las señales a los controladores *EasyDriver* para mover los motores efectivamente.

La versión original del firmware se distribuye en formato de proyecto para el IDE de *Arduino* (.ino) y fue escrito originalmente por Keegan Crankshaw. Tiene dos funciones además de `setup()` y `loop()`. La función `analyzeInput()` determina la dirección de movimiento para una serie de pasos de acuerdo a la información recibida por el puerto serial. Dicha recepción se realiza en la función `loop()`. Luego, `analyzeInput()` invoca a la función `stepMotor()` que mueve un motor. Esta función recibe como parámetros el motor identificador del motor, la dirección del movimiento y la cantidad de pasos.

Las modificaciones realizadas implementan el funcionamiento de la función `analyzeInput()` directamente en `loop()`. Esta función se encarga de recibir e interpretar el mensaje de 6 bytes enviado desde la computadora con la posición absoluta del objeto a apuntar. Luego determina las distancias desde la posición actual a la nueva para cada motor y envía estos datos a la función `moveMotors()`. Esta función mueve ambos motores a la vez, intercalando un paso para el eje de altitud y otro para el eje de azimut. Además esta función incluye el comportamiento que permite apagar el láser durante el movimiento de los motores, característica que fue agregada para este proyecto. Modificaciones a la electrónica

En la versión original de la electrónica, ambos motores se conectan a sendos *EasyDriver* en los pines correspondientes (cuatro pines etiquetados motor A -dos pines- y B -dos pines-). Los pines de

step y dir del *EasyDriver* que controla el motor en eje de azimut se conectan a las salidas digitales D4 y D5 del *Arduino*. Los pines de step y dir del control de altitud, se conectan a los pines D6 y D7. Todos estos pines están configurados como salidas desde el firmware. La electrónica de los *EasyDriver* se alimenta directamente a 5V tomados del pin de 5V del *Arduino*, mientras que la electrónica de potencia para los motores se conecta de manera separada.

El proyecto original no incluye el control para el láser por medio del *Arduino*. Para este trabajo se conectó el pin D12 del *Arduino* al pin BASE de un transistor NPN bipolar. Los pines EMISOR y COLECTOR se conectan a GND y al pin negativo del láser respectivamente. El pin positivo del láser se conecta directamente a la salida de 3.3V del *Arduino*. De esta manera, cuando el transistor recibe una señal de la salida digital D12 del *Arduino*, cierra el circuito y el láser se enciende.

4.3. Modificaciones al cabezal altazimutal

En el proyecto original, el eje X (azimut) es impulsado por uno de los motores de paso. Este mueve una polea con reducción 2:1 sobre la que se monta el motor para el eje Y (altitud), más el eje propiamente dicho, sobre el que se monta el cabezal láser. Este eje no tiene reducción. Determinar las reducciones de los ejes resulta importante debido a que *tcpTelescope.py* [11] utiliza esta información para realizar las transformaciones de coordenadas ecuatoriales a altazimutales.

Este sistema presenta el inconveniente de utilizar correas para transmitir el movimiento de los motores. Estas pueden ser difíciles de conseguir y pueden no ser confiables para un sistema sin ruedas dentadas. En el proyecto original, estas correas se reemplazaron por bandas de goma, las cuales introducen backlash, es decir, como la banda de goma se estira, el movimiento en un sentido y luego en otro, no es inmediato, sino que el segundo movimiento debe realizar pasos extra para contrarrestar el estiramiento de la banda de goma producido en el primer movimiento.

Para el presente trabajo, se realizaron modificaciones al cabezal para prescindir del uso de reducciones y correas. Dado que el peso del cabezal no es muy grande, se montó la rueda del

eje X directamente sobre el motor. El puntero láser también se montó directamente sobre el motor del eje Y. De esta manera, el dispositivo resulta más compacto, sencillo y preciso.

5. Pruebas realizadas

Durante las pruebas realizadas destacó la repetibilidad del dispositivo, no así la precisión, probablemente debido a la imposibilidad de nivelarlo correctamente y a la inestabilidad de la base. Las mejoras realizadas al firmware y al software se comportaron de manera estable y la comunicación entre *Stellarium* y *tcpTelescope.py* parece robusta.

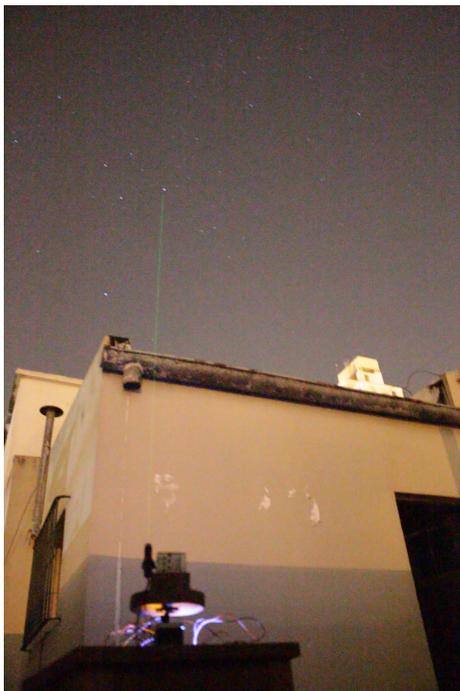


Figura 5: Primera luz del dispositivo, señalando a Acrux en la constelación de la Cruz del Sur.

En la figura 5 puede verse el dispositivo en acción, señalando hacia Acrux, la estrella más brillante de la constelación de la Cruz del Sur.

6. Costos del prototipo

Al momento de escribir este trabajo, el prototipo está valuado en unos \$2500 pesos. La mayor parte de del costo esta relacionada con los motores, representando más de un tercio del valor total.



Figura 6: Durante las pruebas realizadas destacó la falta de estabilidad de la base.

Estos motores pueden ser reemplazados por motores de impresoras o escáneres en desuso, siempre y cuando estos sean motores de paso bipolares. Hay que tener en cuenta que estos motores deben tener el torque suficiente y que es probable que sea necesario modificar el software de acuerdo a la cantidad de pasos de cada motor. Todas las partes pueden conseguirse en el mercado local.

7. Retorno a la comunidad

Todo el trabajo realizado para este proyecto se basa en trabajos previos de software y hardware libre. Debido a esto, quienes realizamos este desarrollo creemos conveniente poner a disposición de la comunidad las mejoras realizadas.

Para ello compartimos el código fuente en uno de los repositorios de software más populares en la comunidad hoy en día, GitHub [13]. Este puede obtenerse desde el repositorio disponible en <https://github.com/jwackito/Galileo-s-Finger> [14]. Los esquemas de conexionado también están disponibles allí.

8. Trabajo a Futuro

8.1. Software

En cuanto al software, resta implementar los mecanismos de parada y cambio de dirección durante el movimiento de los motores. Esto permitiría realizar una parada de emergencia o

cambiar de objeto mientras los motores se están moviendo al previo.

También deben realizarse algunas mejoras en cuanto a los mecanismos de puesta en estación, ya que los disponibles hasta el momento son imprecisos.

8.2. Hardware

El prototipo resultó bastante robusto ante el manipuleo en la parte del cabezal. Sin embargo, la electrónica está armada sobre un protoboard. En un futuro cercano será necesario realizar un circuito impreso dedicado sobre el cual soldar todos los componentes de manera apropiada.

También cabe la posibilidad de modificar el prototipo del cabezal para incluir partes fabricadas mediante la técnica de impresión 3D. Un soporte para el puntero láser es indispensable.

También será necesario montar el cabezal en una base más estable y que permita nivelar y alinear el dispositivo correctamente, ya que de esto depende la precisión de la marca con láser.

- [9] EasyDriver - Stepper Motor Driver <https://www.sparkfun.com/products/retired/13226>
- [10] Fritzing <http://fritzing.org/home/>
- [11] Galileo-s-FingeR - An open source astronomy learning tool. <https://github.com/kcranky/Galileo-s-Finger>
- [12] PYSCOPE -A Stellarium compatible telescope control server module written in Python <https://github.com/Svenito/Pyscope>
- [13] GitHub <http://www.github.com>
- [14] <https://github.com/jwackito/Galileo-s-Finger>

Referencias

- [1] El proyecto original Galileo Finger <https://hackaday.io/project/4846-galileos-finger>
- [2] Proyecto Stellarium <http://www.stellarium.org>
- [3] ¿Qué es el software libre? - Proyecto GNU - Free Software Foundation <http://www.gnu.org/philosophy/free-sw.html>
- [4] Hardware Libre <http://www.iearobotics.com/personal/juan/publicaciones/art4/html/node1.html>
- [5] Hackaday <https://hackaday.io/>
- [6] Laboratorio de Investigación en Nuevas Tecnologías Informáticas - LINTI <http://www.linti.unlp.edu.ar/linti>
- [7] Telescope Control http://www.stellarium.org/wiki/index.php/Telescope_Control
- [8] Arduino Website <http://www.arduino.cc/>