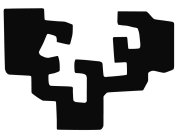
eman ta zabal zazu



Universidad del País Vasco
Euskal Herriko Unibertsitatea

Grado en Ingeniería Informática

Ingeniería de Computadores

Proyecto de Fin de Grado

Desarrollo de una estación de topografía 3D con tecnología LIDAR para cavidades subterráneas

---

Autor

Patxi Llano Vicente

Director

Alex Mendiburu

Co-director

Josu Ceberio

2017

Agradecimientos

Me gustaría dar las gracias a mi familia, por darme la posibilidad de realizar la carrera de Ingeniería Informática y apoyarme para dar siempre lo mejor de mí.

A mi director, Alex Mendiburu, por ayudarme a encaminar el proyecto y buscar soluciones y alternativas a los problemas que hemos encontrado.

A mi co-director Josu Ceberio, por todas las ideas aportadas sin las cuáles nunca habría sido posible desarrollar este proyecto.

A la Unión de Espeleólogos Vascos, que gracias a su apoyo y a la energía de sus miembros (heads-up a Isra) han hecho posible la finalización del trabajo.

Y a todos los compañeros y amigos que me han acompañado a lo largo de la carrera. Especialmente a Ainhoa Havelka y Jon Iker Llorente, por estar siempre a mi lado en los mejores y los peores momentos.

Resumen

Con el fin de conocer en detalle la Tierra, los espeleólogos buscan la forma de dibujar con detalle las diferentes cuevas y simas que son exploradas. Para facilitar esta tarea, este proyecto ha tenido como objetivo crear una prueba de concepto de una estación de topografía portátil, que sea capaz de tomar datos para generar imágenes en tres dimensiones y que resulte asequible en términos económicos, utilizando para ello, entre otros componentes, un láser de pequeñas dimensiones y una placa Arduino.

Índice general

Índice de figuras

Indice de tablas

1. CAPÍTULO

Introducción

El mundo esconde miles de lugares ocultos, inexplorados. Pero el ser humano, en su afán de conocer todo lo que le rodea, sigue explorando, descubriendo y ampliando su conocimiento.

A pesar de tener todo tipo de tecnologías con las que analizar la superficie de nuestro planeta, aún hay sitios que son desconocidos. Si hablamos de lo que se esconde bajo dicha superficie, la única opción posible para conocerlo es explorarlo físicamente.

La espeleología, además de un deporte que podemos practicar por pura afición, es la manera con la que descubrir cómo es la Tierra en sus capas inferiores, a través de sus miles de cuevas y simas. Nos ayuda a conocer con detalle lo que nos oculta la superficie.

Para convertir esta actividad en información útil, los espeleólogos deben recurrir a herramientas que les permitan modelar todos estos lugares. Actualmente es relativamente sencillo realizar imágenes en dos dimensiones. En determinados puntos de una cueva (que interpretamos como puntos en el eje horizontal), se pueden tomas lecturas del eje vertical con un láser de medición. Después, podemos unirlos todos y generar una imagen, que muestra los desniveles y las alturas del lugar.

¿Pero qué ocurre si lo que queremos es un modelo en tres dimensiones, que nos permita ver con alto detalle el interior de una galería concreta? Esto requiere de una cantidad de lecturas láser varias veces superior. Además, dado que necesitamos "dibujar una esfera", el manejo del giro del láser es extremadamente complejo de gestionar de manera manual.

Facilitar esta tarea es precisamente lo que se pretende con este proyecto: crear un disposi-

tivo con la capacidad de recoger datos que nos permita generar imágenes tridimensionales del entorno.

Para ello, se ha realizado una prueba de concepto de un dispositivo basado en Arduino, que incluye un láser para tomar lecturas de distancias, motores para cubrir todos los ángulos y una brújula electrónica necesaria para generar modelos consistentes con el entorno real. Para agrupar todo, se ha diseñado una estructura fabricada por medio de impresión 3D. Por otro lado, existen dos elementos software: el programa principal para tomar y almacenar medidas que ejecuta la placa Arduino, y el software de comunicación y procesamiento de datos a ejecutar desde un equipo informático externo. Más tarde, esos datos deberían ser interpretados por software apropiado, que en teoría generará imágenes tridimensionales.

Dado que practica la espeleología de manera habitual, el co-director Josu Ceberio ha hecho las veces de cliente interesado en el producto.

La memoria del proyecto está estructurada de la siguiente forma:

- En el capítulo 2: Documento de objetivos del proyecto se encuentra la planificación del proyecto, su desglose en tareas y su asignación de horas.
- El capítulo 3: Desarrollo de la estación de topografía 3D analiza las diferentes fases de desarrollo de la estación de topografía 3D.
- El capítulo 4: Desarrollo del software de administración de datos describe el desarrollo de la interfaz de comunicación entre la placa Arduino y un equipo informático externo.
- En el capítulo 5: Resultados se recogen los resultados que hemos obtenido, referentes tanto a la estación de topografía en sí como al software externo.
- Finalmente, en el capítulo 6: Conclusiones se encuentran las conclusiones en relación a diferentes aspectos del proyecto.

Se añade, además, el código implementado en los diferentes programas, a modo de anexo.

2. CAPÍTULO

Documento de objetivos del proyecto

El objetivo principal del proyecto es, como se ha indicado en la introducción, el desarrollo de una estación de topografía 3D. Dicho dispositivo deberá ser portátil, resistente al agua y a los golpes y capaz de realizar lecturas de distancias a su alrededor, para ser procesadas después y generar imágenes simuladas de dicho entorno.

El proyecto consta de dos objetivos principales: la estación de topografía 3D y una aplicación software para el volcado y la gestión de los datos recogidos sobre un equipo informático externo.

2.1. Tareas

Para estructurar el desarrollo del proyecto, se han especificado diferentes tareas

- Estación de topografía 3D con tecnología LIDAR para cavidades subterráneas
  - Análisis de los elementos necesarios
    - T1 - Análisis de placas Arduino
    - T2 - Análisis de láseres de medición
    - T3 - Análisis de otros elementos
  - Intercomunicación de elementos
    - T4 - Conexión Arduino–LIDAR-Lite

        - T5 - Conexión de elementos mecánicos
    - Diseño
        - T6 - Análisis de la estructura externa
        - T7 - Análisis de la estructura interna
        - T8 - Construcción de la estructura en impresora 3D
    - Programación
        - T9 - Programa para la medición y almacenamiento de lecturas
        - T10 - Programa de comunicación Bluetooth para transmisión de datos
        - Librerías
            - T11 - Exploración de librerías Arduino disponibles para los diferentes elementos hardware
- Software de lectura y administración de datos
    - Estructura de datos
        - T12 - Análisis del formato de los datos obtenidos (*raw*)
        - T13 - Estudio de los formatos de ficheros utilizados por el programa Visual Topo
        - T14 - Rutina de transmisión y estructuración de datos
- Pruebas
    - Físicas
        - T15 - Pruebas de resistencia física de la estación
    - Datos
        - T16 - Pruebas de obtención y procesado de datos
- Gestión
    - T17 - Planificación del proyecto
    - T18 - Memoria del proyecto

2.2. Planificación

Las 300 asignadas al proyecto se han distribuido de la siguiente forma entre todas las tareas, con granularidad de 5 horas:

T1	T2	T3	T4	T5	T6	T7	T8	T9
20	20	10	5	15	20	20	20	15

Tabla 2.1: Planificación de horas por tarea 1

T10	T11	T12	T13	T14	T15	T16	T17	T18
15	5	5	10	10	10	10	15	75

Tabla 2.2: Planificación de horas por tarea 2

La figura 2.1 muestra el gráfico Gantt con la distribución del tiempo a invertir:

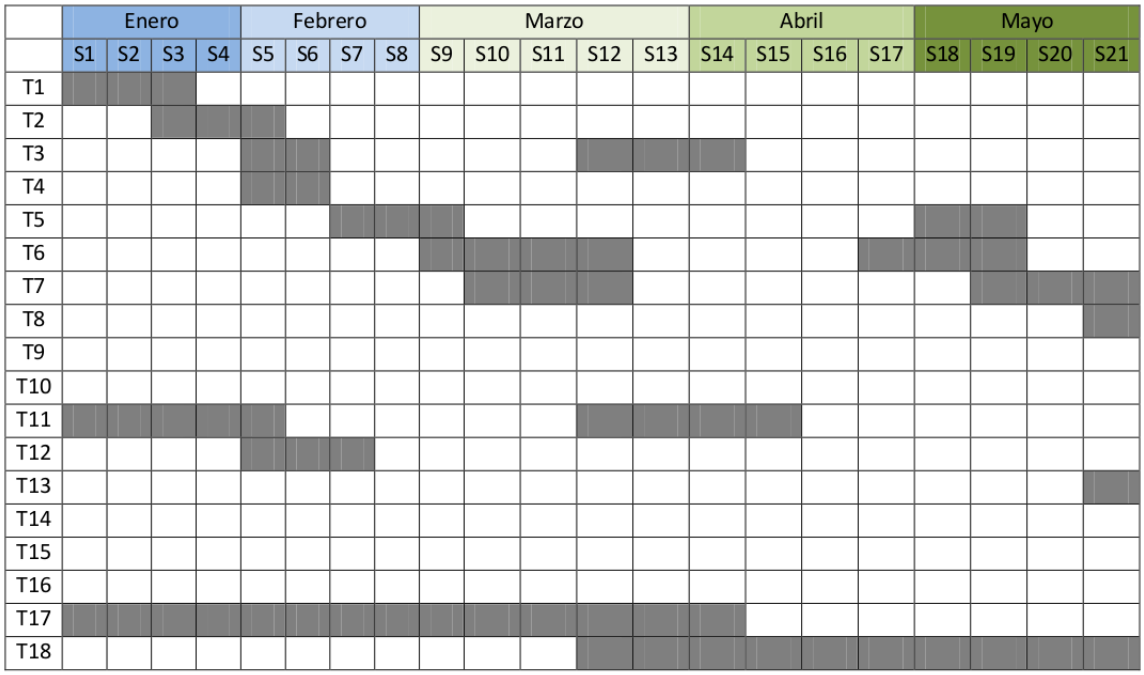


Figura 2.1: Gráfico Gantt 1

Los meses de marzo y abril suelen tener una mayor carga de trabajo por parte de las asignaturas del curso, por lo que se ha planificado poco tiempo de trabajo en estos meses, programando una mayor carga en el mes de junio.

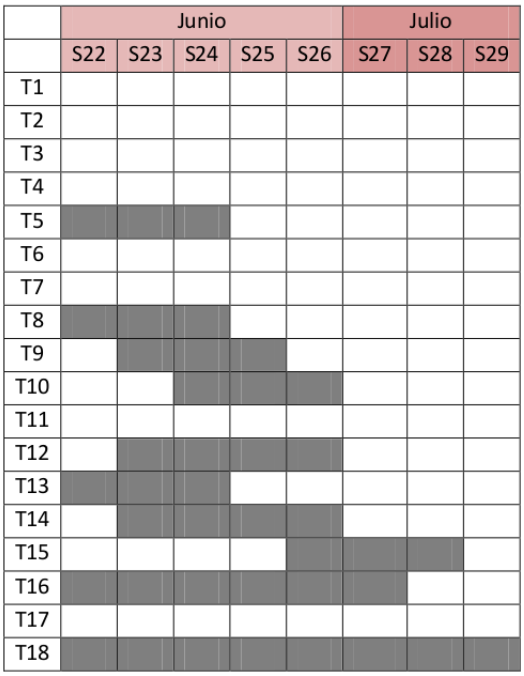


Figura 2.2: Gráfico Gantt 2

2.3. Riesgos

Se han identificado los siguientes riesgos, y se han propuesto medidas para reducir su efecto en cada caso:

- Funcionamiento errático o pérdida del láser LIDAR-Lite
  - Efecto: Se presupone como el más catastrófico, al ser el elemento de mayor valor y con una disponibilidad limitada. Pondría en riesgo la continuidad de todo el proyecto.
  - Medidas de prevención: Analizar con detenimiento todos los requerimientos eléctricos del láser y comprobar en todo momento las conexiones con otros elementos. Tener especial cuidado al transportarlo.
- Funcionamiento errático o pérdida de la placa Arduino 101
  - Efecto: El segundo elemento más crítico. Aunque no se espera que pudiera poner en riesgo la continuidad del proyecto, sí que supondría un retraso muy significativo en el desarrollo.

    - Medidas de prevención: Ídem a las del láser.

- Problemas con la construcción externa

    - Efecto: La construcción de la estructura puede ocurrir en varias fases y con diferentes diseños, lo que puede acabar retrasando su diseño final si ocurren muchos cambios.
    - Medidas de prevención: Ajustar lo mejor posible el diseño a los requerimientos de la estación, y realizar modificaciones adecuadas para reducir el número de cambios en el diseño.

- Cambios de dedicaciones

    - Efecto: Durante el ciclo de vida del proyecto, podrá haber modificaciones en los plazos y las dedicaciones debido a tareas y exámenes del curso académico.
    - Medidas de prevención: No se pueden prevenir dichas modificaciones, pero es necesario ajustar los tiempos de trabajo de la mejor manera posible.

2.4. Metodología

Para realizar el proyecto se ha seguido la siguiente metodología:

- Reuniones semanales de una hora. Se analiza el estado del proyecto, se revisa el progreso de las tareas pendientes y se discuten los objetivos para las siguientes semanas. En caso de no haber completado alguna tarea, se explican y analizan las razones, y se plantean alternativas para la finalización de dicha tarea.

- Desarrollo gradual de la estación de topografía 3D. Su construcción se divide en varias fases, y solo se comienza la siguiente una vez se ha completado la anterior. Se ha distribuido en las siguientes fases: análisis de necesidades, selección de elementos hardware, construcción de su estructura externa y programación.

- Desarrollo por iteraciones del software externo. Al igual que con la estación, se ha desarrollado siguiendo diversas etapas. En este caso, primero se ha programado la conectividad con el dispositivo, después la capacidad de intercambiar información y finalmente la interfaz gráfica para un uso más sencillo.

3. CAPÍTULO

Desarrollo de la estación de topografía 3D

3.1. Análisis inicial

La estación de topografía 3D está enfocada para usarse en cuevas y simas, las cuales serán escaneadas con dicho dispositivo. Existen diferentes alternativas tecnológicas para realizar mediciones de distancia. Generalmente, las basadas en láser tienen mayor fiabilidad y rango de medición. Pero además, esta categoría incluye dispositivos con diferentes técnicas y métodos.

Si nos referimos al escaneo de objetos y entornos, destaca la LIDAR, considerado acrónimo de *Light Detection And Ranging* aunque en realidad es la simple combinación de las palabras *light* y *radar*. Este método envía una onda lumínica (normalmente no visible al ojo humano) y mide el tiempo de propagación y la degradación de dicha onda en su retorno. Este proceso se repite miles de veces para obtener una nube de puntos, con los que poder generar modelos virtuales del entorno.

Actualmente es utilizado para todo tipo de aplicaciones en geografía, geología, físicas de la atmósfera e incluso conducción autónoma, entre otros.

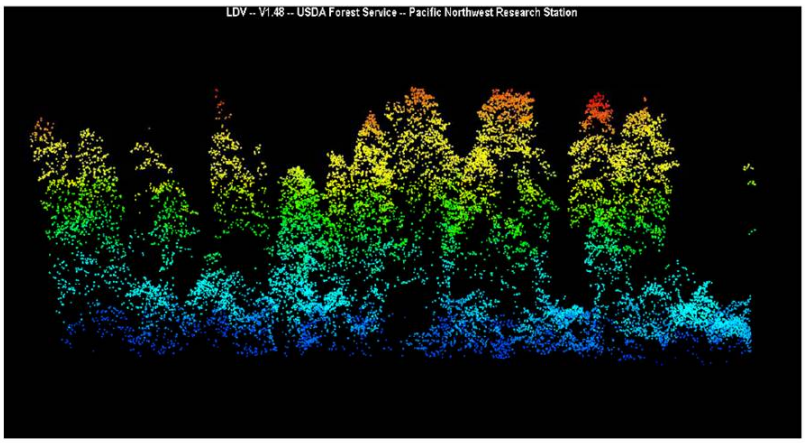


Figura 3.1: Imagen con LIDAR del bosque H.J. Andrews. Imagen de la Oregon State University. Bajo licencia CC-BY-SA

Existen dispositivos que implementan LIDAR de tamaños muy variados, en función de las necesidades de distancia y precisión que necesitemos. Los de menor tamaño, de venta al público, incluso vienen preparados para su uso con placas Arduino. Estas placas integran un microcontrolador y una serie de periféricos en un espacio muy reducido.

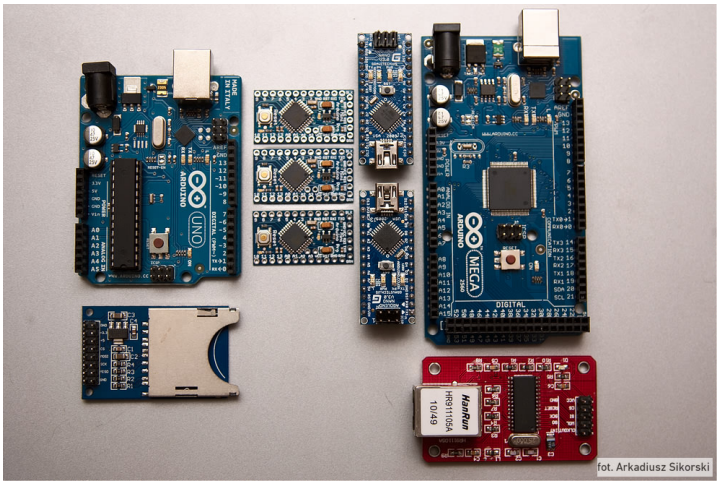


Figura 3.2: Múltiples modelos de placas Arduino. Foto de Arkadiusz Sikorski. Bajo licencia CC-BY

Además, permiten manejar múltiples dispositivos externos mediante la conexión de éstos a los pines del microcontrolador. Gracias a estos sistemas existen millones de proyectos y utilidades particulares para casi cualquier aplicación o problema.

Si combinamos un láser pequeño con tecnología LIDAR con el control de una placa Arduino, podemos crear un sistema de tamaño reducido para crear modelos del entorno. Por supuesto, sus prestaciones serán más limitadas que las de un equipo de escaneo profesional, pero suficientes para su uso en entornos donde las distancias no sean excesivamente grandes.

Teniendo en cuenta todos estos factores, se han determinado los siguientes requerimientos para la estación de topografía 3D:

- La estación se basará en la tecnología láser LIDAR, preparada para su uso en medición de distancias.
- Su tamaño debe ser lo suficientemente reducido como para ser transportada con facilidad.
- Debe tener una autonomía suficiente para realizar un número aceptable de lecturas antes de necesitar una recarga.
- Al ser un dispositivo concebido para su uso en espeleología, debe ser resistente a golpes y humedad; lo más estanco posible.
- Su conexión con dispositivos externos se realizará mediante tecnología Bluetooth, para minimizar las conexiones externas y facilitar su estanquidad.

Adicionalmente, los datos de medición necesitan indicadores de clino y orientación (brújula) para generar modelos adecuados de las galerías subterráneas. Por ello, se requieren dispositivos capaces de ofrecer dichos valores.

3.2. Componentes

Una vez identificadas las necesidades de la estación de topografía, fue necesario explorar el mercado para encontrar elementos que cubrieran dichos factores. A la hora de decidir qué elementos concretos iban a a ser adquiridos, se consideraban varios factores: especificaciones del producto, facilidad de uso, impacto en la funcionalidad de la estación, precio, ...

Finalmente, la estación de topografía 3D cuenta con los siguientes elementos hardware:

- Láser Garmin LIDAR-LiteV3

Figura 3.3: Láser Garmin LIDAR-LiteV3

Elemento central de la estación, ya que se encarga de realizar las lecturas de distancias del entorno. Aunque existen diversos modelos de láseres LIDAR, era necesario escoger uno de coste bajo. Entre las opciones restantes, el modelo de Garmin escogido es sin duda el más popular entre los consumidores, pues ofrece una resolución y frecuencia de lectura muy buenas por un precio muy competitivo. De hecho, esta es la tercera versión presentada de dicho láser, gracias a la popularidad de las versiones anteriores.

En este modelo en concreto, destacamos las siguientes especificaciones:

- Rango: 0.05 - 40 metros
- Resolución: 1 cm
- Precisión: +/- 2.5 cm en distancias superiores a 1 metro
- Tasa de refresco: hasta 500 Hz

- Placa Arduino 101

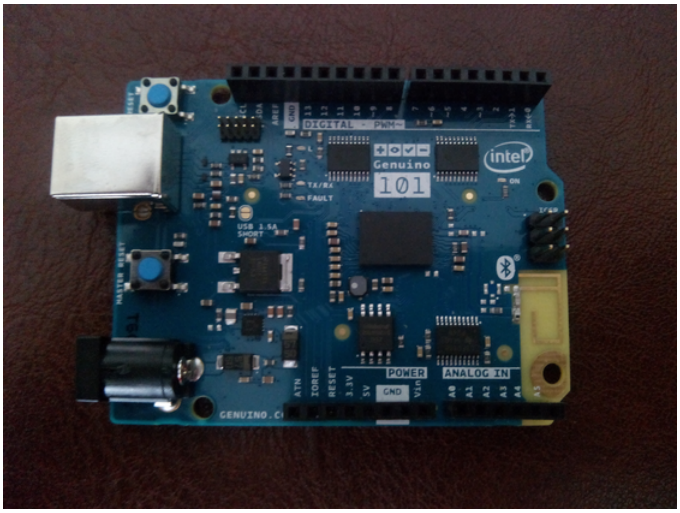


Figura 3.4: Arduino 101

Segundo elemento más importante de la estación. Se encarga de manejar todos los dispositivos, recoger y almacenar la información de las lecturas, y comunicarse con equipos externos para la transmisión de los datos. Existe un amplio catálogo de placas Arduino. Cada una incluye diferentes funcionalidades y periféricos específicos, en función de las necesidades del proyecto a realizar.

Para nuestra estación de topografía 3D se escogió el modelo "101". Esta placa integra un chip de bajo consumo Intel® Curie™. Está compuesto por dos núcleos con arquitecturas diferentes, uno x86 y el otro ARC, ambos de 32 bits, funcionando a una frecuencia de reloj de 32 MHz y un voltaje de 3.3V. Aunque otros modelos incluyen prestaciones similares, se decidió elegir esta por integrar giroscopio (para valores de clino) y Bluetooth (para conexión con equipos externos y transmisión de datos).

El 17 de julio de 2017, Intel® anunció el fin de producción(*End-of-Life*) su familia Curie™, un mes después de hacer lo propio con otras de sus plataformas de productos *maker*, como la Edison. Por lo tanto, en caso de continuar el desarrollo de este dispositivo, se recomienda utilizar una placa distinta o migrar a otro sistema, como puede ser una Raspberry Pi, para evitar problemas de soporte.

- Shield SD de SparkFun para placas Arduino

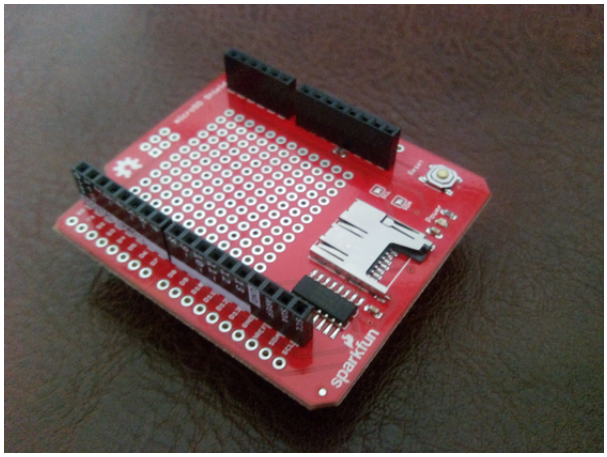


Figura 3.5: Shield SD

Periférico adicional para la placa Arduino, que añade capacidad para tarjetas microSD. Necesario para almacenar grandes cantidades de datos en memoria no volátil.

- Magnetómetro HMC5883L de Honeywell (montado por Adafruit)

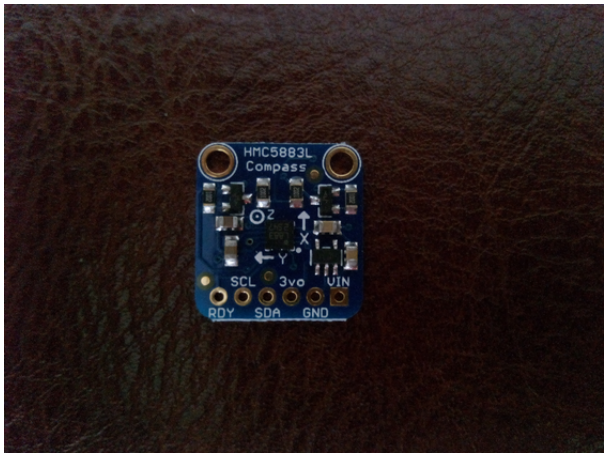


Figura 3.6: Magnetómetro HMC5883L

Dispositivo para obtener valor de brújula. Mide la intensidad de los campos magnéticos para estimar la orientación. Tiene una precisión de +/- 1-2 grados.

- Servomotor de rotación continua

Figura 3.7: Servomotor SpringRC SM-S4303R

Con capacidad de giro ilimitada (360º). Se encarga del giro del láser en el eje horizontal (en adelante *servomotor horizontal*).

- Servomotor estándar

Figura 3.8: Servomotor SpringRC SM-S2309S

Con capacidad de giro limitada (180º). Se encarga del giro del láser en el eje vertical (en adelante *servomotor vertical*).

- Batería LiPo de 1500 mAh de capacidad

Figura 3.9: Batería LiPo AeroEnergy de 1500mAh

Funete de alimentación de todos los elementos. En uso continuo de la estación de topografía, su duración se estima en unas 2 horas. Su adquisición incluye la necesidad de un cargador específico para baterías de polímero de litio.

Figura 3.10: Cargador SkyRC IMAX B6 Mini

Para distribuir la electricidad desde la batería hasta el resto de elementos, se ha utilizado una placa electrónica personalizada. Incluye:

- Regulador de voltaje 7805 para entregar 5V
- Resistencias de 100K Ω para la absorción de sobrecargas
- Condensadores de 100 y 680 μF para mantener la corriente y evitar caídas de tensión
- Relés Finder 30 Series para controlar el paso de la energía dirigida a los servomotores y el láser

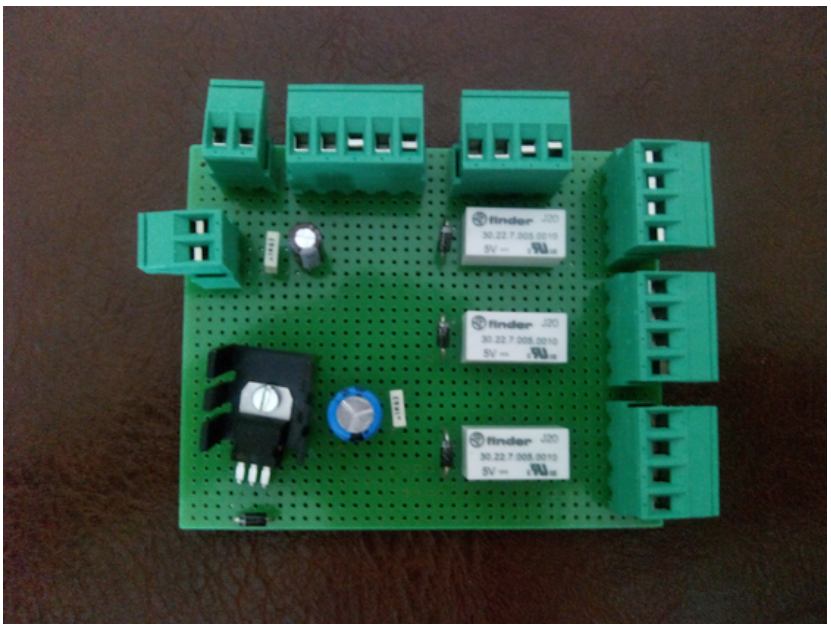


Figura 3.11: Placa electrónica para distribuir la electricidad

Por supuesto, es necesario agrupar todos los elementos en uno solo para obtener un objeto único, la estación de topografía en sí misma. Esto se traduce en diseñar y construir una estructura que cumpla lo mejor posible con los requerimientos de tamaño y aguante especificados anteriormente

3.3. Diseño de la estructura

A la hora de realizar diseños de la carcasa para la estación de topografía, era necesario mantener un equilibrio entre los dos factores base: la funcionalidad del dispositivo y las comodidades para el usuario (uso/transporte). Desde un inicio se planeó utilizar la impresión 3D como método de fabricación, ya que ofrece una gran flexibilidad tanto en diseños como en materiales. Además, disponíamos una impresora adecuada a nuestro servicio.

El concepto inicial constaba de un contenedor cúbico, posteriormente cilíndrico, con una estructura interna de tres bases sostenidas directamente sobre la carcasa exterior.

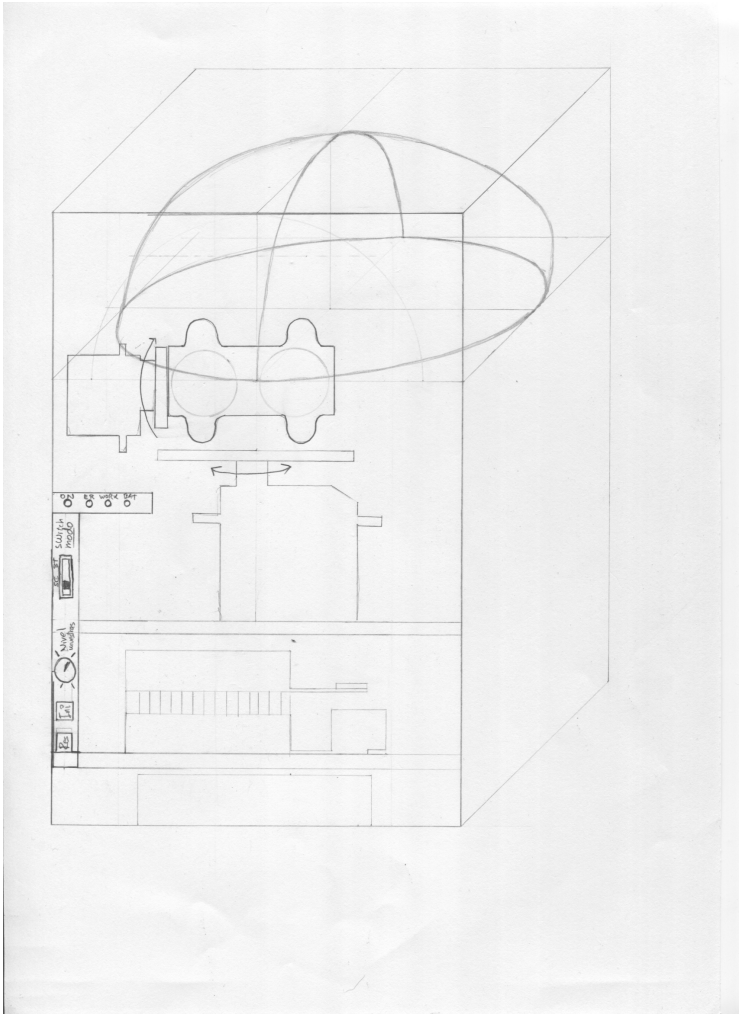


Figura 3.12: Primer concepto de la estructura inicial

Se barajó la posibilidad de utilizar materiales translúcidos en la zona superior, para permitir que el láser realizara lecturas siempre cubierto, favoreciendo su estanquidad. Pero la refracción de dichos materiales resultó ser demasiado alta como para ofrecer resultados fiables, por lo que no quedó mas remedio que diseñar una protección desmontable; esto es, una tapa.

Dado que la batería necesita ser conectada para su recarga, era necesario poder extraerla o, al menos, contar con un puerto accesible desde la carcasa exterior. Finalmente, se decidió hacer la estructura interna extraíble, ya que favorece a la estanquidad de la estación al tener un único posible punto de fallo en este aspecto. Por otra parte, con el fin de ahorrar complejidad en el diseño exterior, se decidió colocar los botones, switches y LEDs en la base superior, aprovechando espacio libre y accesible al estar al descubierto.

Se trató de ajustar el tamaño al mínimo posible, teniendo en cuenta las dimensiones de los diferentes elementos, la necesidad de paso de cables entre niveles y el diámetro necesario para hacer girar el láser y el servomotor del eje vertical. Con todo, se determinó un diámetro de 14,4 centímetros internos, más el grosor de 1 centímetro de la carcasa. Sin la tapa, la altura era de 10,5 centímetros. Las bases de la estructura interna eran cuadradas, con lados de 11 centímetros y sostenidas por columnas cuadradas de 5 milímetros en las esquinas, y encajaba en unas muescas de la estructura cilíndrica exterior. De esta forma, los cables podrían pasar ente bases sin ningún problema. Este fue el primer modelo construido.

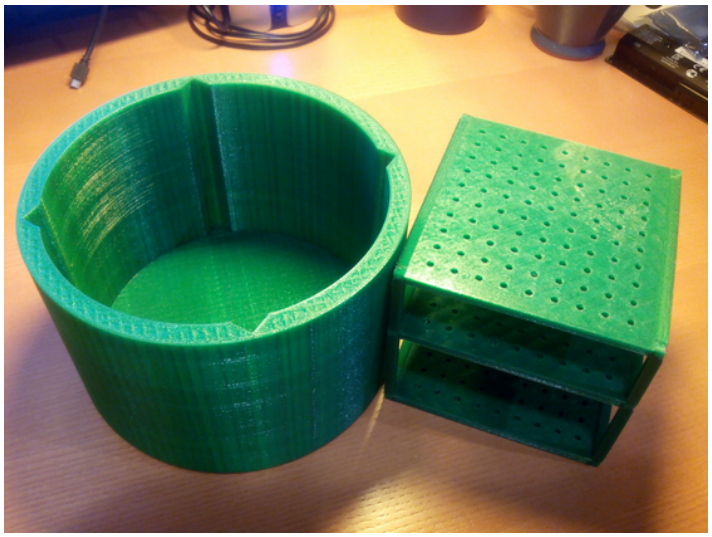


Figura 3.13: Primer modelo de la carcasa. Imagen 1

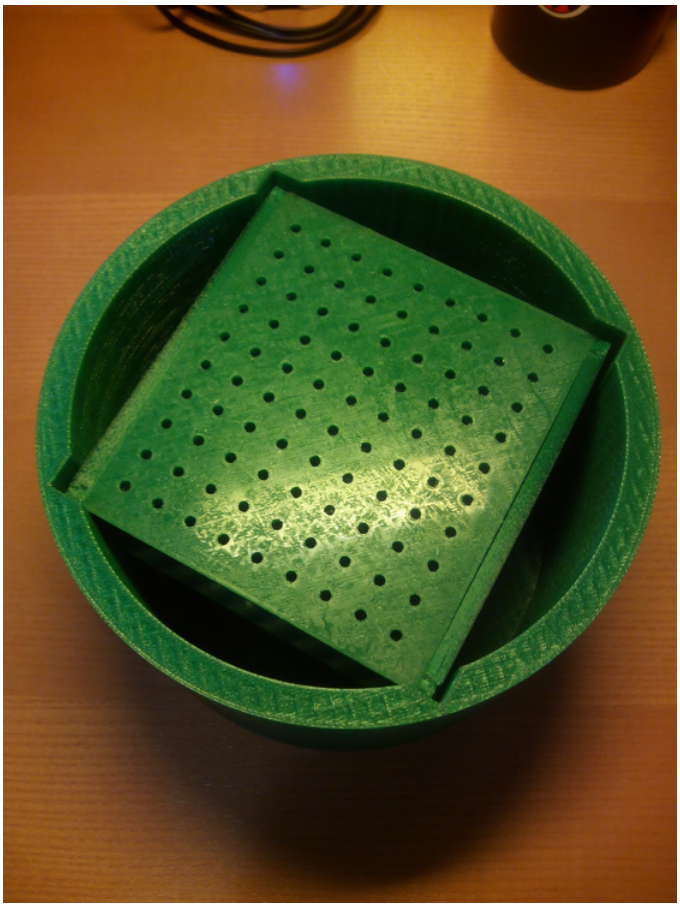


Figura 3.14: Primer modelo de la carcasa. Imagen 2

Con la posibilidad de poder colocar y realizar mediciones sobre un modelo físico, se descubrieron nuevas opciones de mejora. La batería no necesitaba una base completa, ya que era más compacta de lo esperado; la carcasa exterior podía ajustarse al cuadrado interior, con una pequeña abertura en uno de los lados para permitir el paso de cables; y el grosor de la carcasa resultaba demasiado grande. Por lo tanto, se remodeló el diseño a una estructura interna de solo dos bases, y una carcasa cuasi-cuadrada con paredes de 5 milímetros de grosor. La anchura máxima de la carcasa se redujo a los 12,8 centímetros, y su altura a 9,5 centímetros

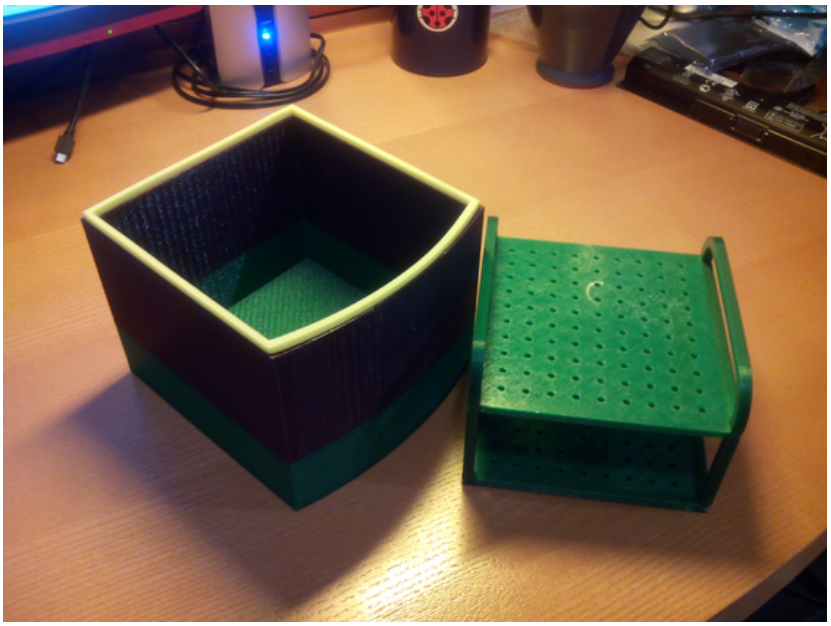


Figura 3.15: Segundo modelo de la carcasa. Imagen 1

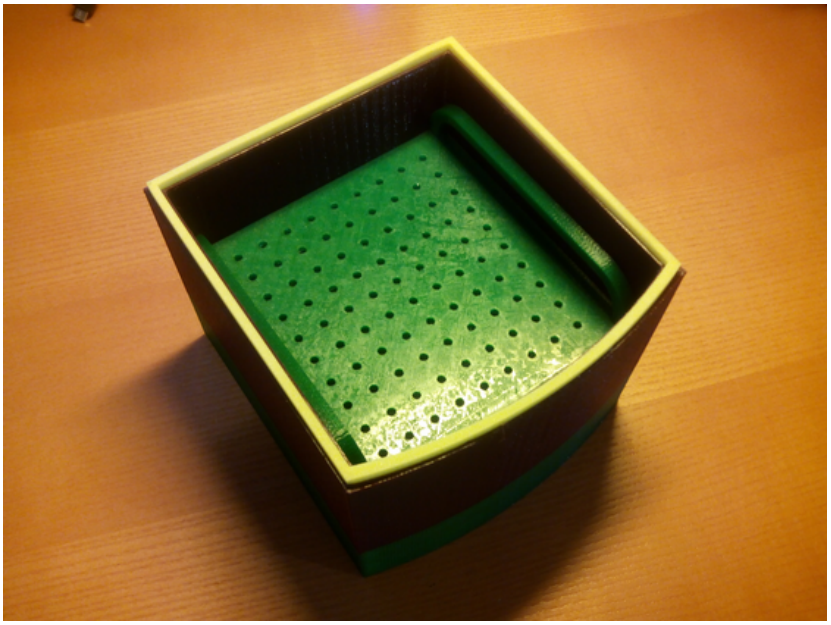


Figura 3.16: Segundo modelo de la carcasa. Imagen 2

Se decidió crear piezas personalizadas para ensamblar los servos y el láser. Se necesitaba una base para el servomotor horizontal con algún lugar para encajar el servo vertical, una base para colocar el láser y una pieza auxiliar para mantener el equilibrio del este. Estas fueron finalmente las piezas fabricadas:

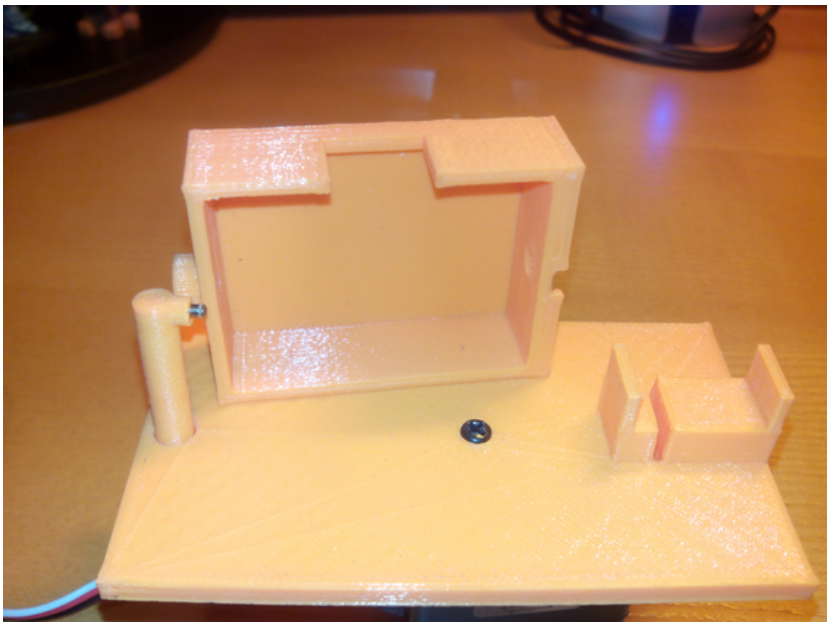


Figura 3.17: Piezas de sujeción de elementos. Imagen 1

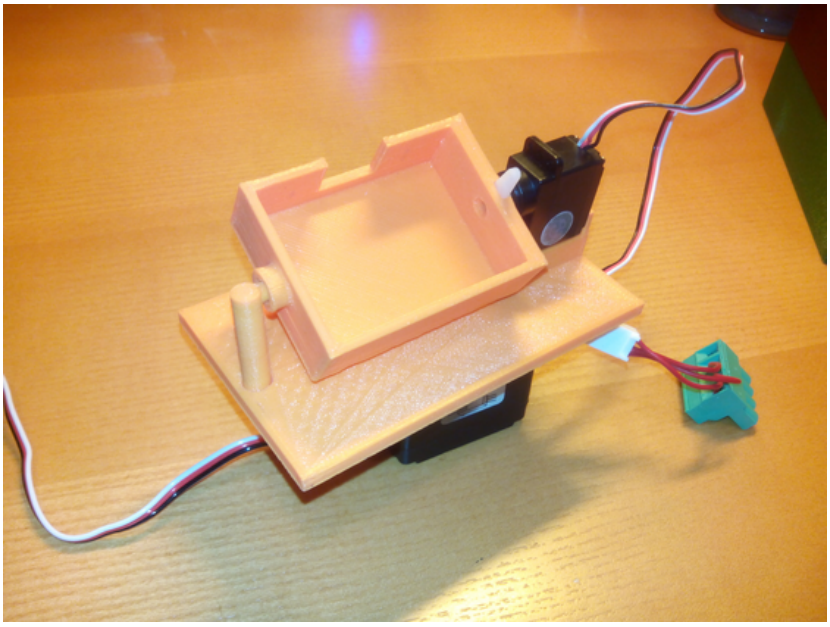


Figura 3.18: Piezas de sujeción de elementos. Imagen 2

Poniendo juntos todos los elementos, obtenemos una visión de cómo luce la estación de topografía

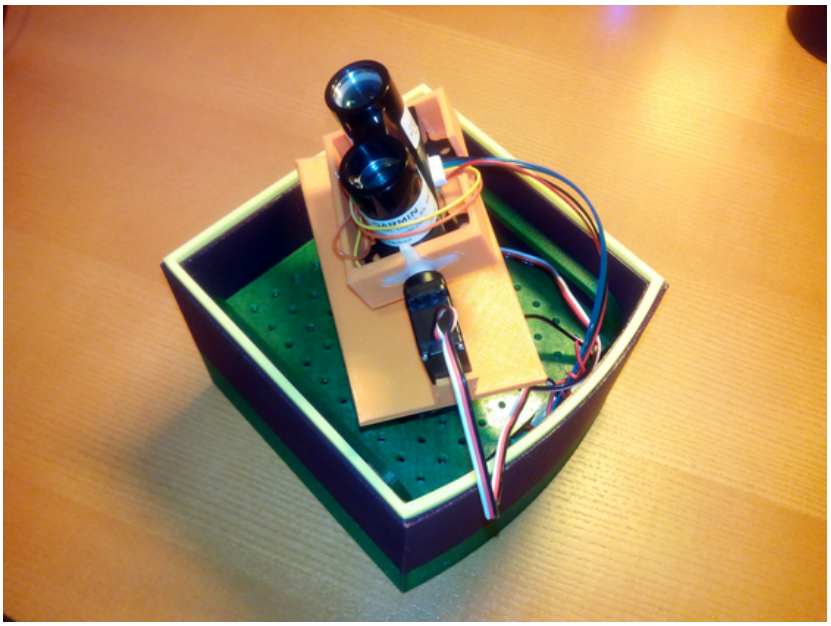


Figura 3.19: Estación de topografía 3D

Tras un nuevo análisis del último diseño, se identificó la posibilidad de atravesar la base superior con el cuerpo del servo del eje horizontal, dándole mayor estabilidad. Los elementos colocados debajo serían cambiados a una posición vertical para evitar obstrucciones. Además, los cables podían pasar a través del espacio sin utilizar de la base superior, pudiendo ajustar la carcasa exterior a la estructura interior. Sin embargo, debido a la escasez de tiempo, este modelo no llegó a diseñarse.

3.4. Programación

El dispositivo tiene dos modos de trabajo:

- Modo de escaneo del entorno
- Modo de volcado y gestión de datos.

Dado que es la única posibilidad, se ha programado en C++, simplificado para Arduino, utilizado el Arduino IDE, necesario para compilar y cargar los programas en la placa.

3.4.1. Planteamiento inicial

Antes de iniciar el dispositivo, el usuario debería poder elegir entre uno de estos modos mediante algún elemento físico. Dada la falta de algún botón o switch, los programas se han cargado y probado por separado.

El funcionamiento de cada programa se plantea de la siguiente manera:

- Modo de escaneo

    1. El usuario escoge la densidad de barrido que desea. Los valores posibles estarán preconfigurados. Tendrá hasta 3 densidades, y 3 LEDs indicarán cuál está seleccionado en ese momento (ordenados de menor a mayor densidad). Se puede alternar el valor mediante un pulsador.
    2. El usuario confirma su elección mediante otro pulsador y se inicia el proceso de lectura de distancias.
    3. En función de la densidad deseada, se configuran la velocidad de rotación de los servos. Para ajustarse a la densidad seleccionada, tras cada una de las lecturas del láser, el servo horizontal girará un tiempo determinado, calculado al inicio del programa. Se aplicará el mismo criterio a los movimientos del servo vertical.
    4. Se genera un nuevo fichero en la tarjeta microSD, donde se irán almacenando los datos recogidos. Los nombres de los ficheros irán numerados de manera ascendente, siendo este último fichero creado el que tenga el valor más alto. Se escriben los valores de clino y brújula iniciales.
    5. El servomotor horizontal empieza a girar y comienza la recogida de datos de distancias. Por cada nuevo dato, se almacena el valor devuelto por el láser en centímetros y sus valores de clino y brújula, calculados en función de la velocidad de rotación del servomotor.
    6. Por motivos de cableado, cada vez que el servomotor horizontal realiza 360º de giro, su dirección se invertirá. En este momento, el servomotor vertical realizará un cambio de ángulo hacia una posición más vertical.
    7. Una vez que el láser alcance la posición perpendicular con respecto al eje horizontal, se detiene el proceso y los servomotores vuelven a sus posiciones de partida. Un LED situado en la parte superior de la estación de topografía

parpadeará durante 30 segundos, indicando que el proceso ha finalizado. Se cierra el fichero de escritura y se vuelve al estado de inicio.

- Volcado y gestión de datos

  1. Se realiza la conexión entre la estación de topografía y el equipo informático vía Bluetooth Low Energy (en adelante *BLE*). Se dispondrá de una interfaz gráfica que facilite este procedimiento al usuario.
  2. El dispositivo se mantiene a la espera de recibir órdenes. El usuario puede entonces realizar diversas acciones:
     - Para explorar ficheros se dispondrá de una lista con lo que contiene el directorio actual. Se podrán seleccionar ficheros para ser transmitidos o borrados.
     - Si se decide transmitirlos, los datos se almacenarán en ficheros locales en el equipo. Los datos son inmediatamente procesados para su posterior uso en Visual Topo. En la memoria microSD de la placa, estos ficheros transmitidos se almacenarán en una carpeta secundaria, a modo de backup. Solo serán borrados si el usuario así lo indica, a través de la interfaz gráfica.
     - Si se decide modificar las densidades de los barridos, el usuario dispondrá de tres cuadros de texto sobre los que indicar el número de puntos deseado para cada slot.
  3. La placa Arduino cierra la conexión Bluetooth cuando se lo indica la aplicación externa, y se vuelve al estado de inicio.

3.4.2. Versiones finales

En ambos casos, las funcionalidades finales se han visto reducidas por problemas encontrados durante el desarrollo:

- Modo de escaneo

  Desde un inicio, se esperaba que el control preciso del servomotor horizontal fuese a suponer el mayor problema. Dado que, a diferencia de un servomotor convencional, los de rotación continua no pueden detenerse en posiciones concretas, la única manera para controlar la rotación, sin hardware adicional, es el tiempo.

En el momento de ponerlo a prueba, el servomotor se comporta de manera muy irregular en cada ejecución del programa, con velocidades de rotación ligeramente diferentes con cada escritura que se realiza sobre su señal. No afecta en gran medida en los primeros giros, pero se acentúa a medida que avanza la ejecución del programa. Debido a esto, las lecturas en el eje horizontal no son precisas. Para obtener rotaciones de exactamente 360º es necesario algún tipo de sensor, que indique cuándo se ha realizado el giro completo, y el tiempo necesario en cada giro seguiría siendo variable. Por ello, el número de lecturas seguiría sin poder controlarse de manera adecuada, lo que hace imposible establecer densidades variables y procesar los datos correctamente.

Por lo tanto, el programa se ha simplificado a un número fijo de puntos, y los parámetros de control de tiempo se han ajustado lo mejor posible a un giro regular. Dado que no resultaban esenciales, tampoco se han tenido en cuenta los valores de brújula y clino en esta prueba de concepto, aunque serían imprescindibles en un uso real para construir la ruta de exploración del espeleólogo. El resto del funcionamiento se mantiene según el planteamiento inicial.
En la implementación final, se realizan 112 lecturas en cada giro horizontal (una por cada 3,2º) por 90 posiciones diferentes en el eje vertical (un grado de diferencia por cambio de posición). Se obtienen un total de más de 10.000 puntos, lo que aporta una densidad muy alta para crear modelos detallados. Estos datos deben ser escritos en el formato adecuado para que Visual Topo los interprete más tarde.

Para el manejo del láser, ha sido necesario analizar la librería para Arduino proporcionada por el fabricante, la cuál podemos descargar directamente desde el gestor de librerías de IDE Arduino. Este dispositivo dispone de múltiples opciones de configuración, que permiten ajustar su uso a las necesidades del proyecto. Para nuestra estación, no se requería de ninguna configuración especial, por lo que se han utilizado los valores base.
La comunicación puede realizarse tanto por medio del protocolo I^2C como por PWM (*Pulse-Width Modulation*). Se ha seleccionado la primera por tener una mayor capacidad de transmisión de información y una implementación más trivial. Este protocolo se basa en un modelo maestro-esclavo, puede conectarse con múltiples dispositivos de manera asíncrona al mismo tiempo y permite velocidades de reloj de entre 100 y 400KHz. Requiere de tan solo dos líneas de conexión para su funcionamiento: *SCL* para la señal de reloj y *SDA* para la de datos. Utilizando I^2C,

es necesario conectar un condensador electrolítico de 680μF en la conexión eléctrica del láser.

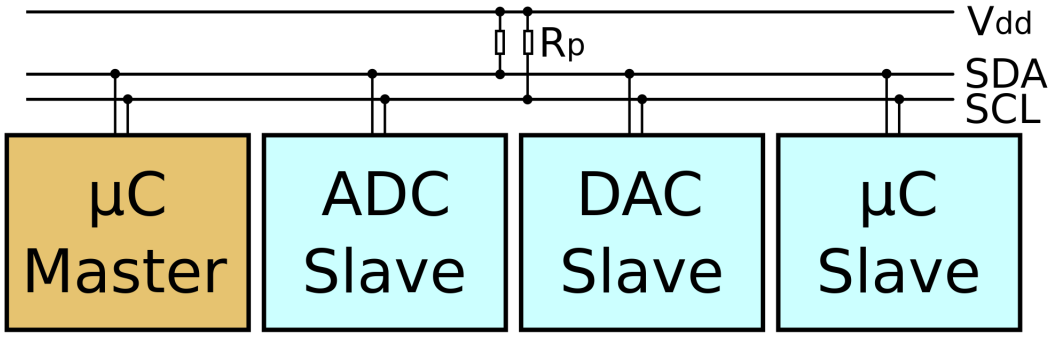


Figura 3.20: Representación de bus I^{2}C con un maestro y tres esclavos. Imagen de Cburnett. Bajo licencia CC-BY-SA

En cuanto a los servomotores, todos utilizan PWM para la transmisión de señal.

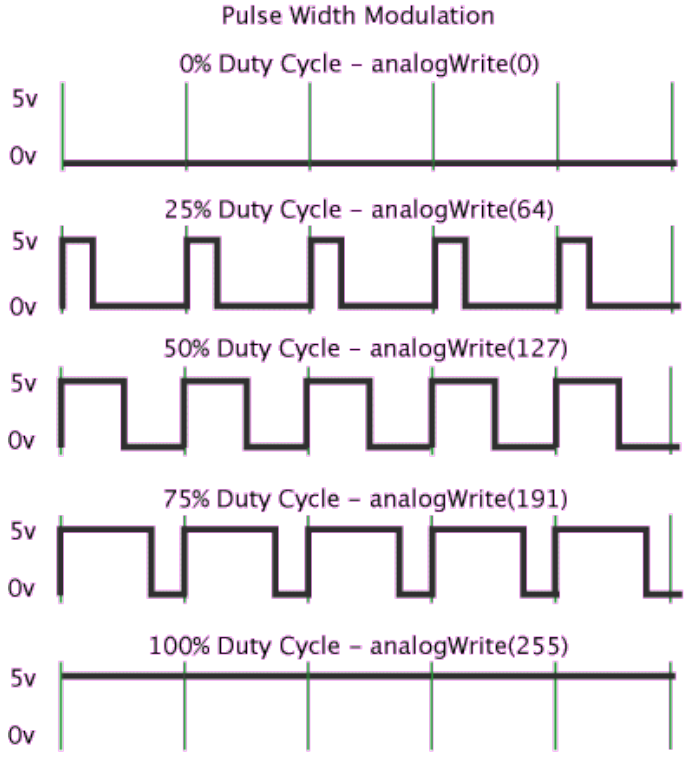


Figura 3.21: Representación de PWM en 5 etapas diferentes. Imagen del equipo de arduino.cc. Bajo licencia CC-BY-SA

Aunque en teoría los valores límite del ancho de pulso son 1000μs y 2000μs, cada servo estándar único establece un ángulo diferente incluso con el mismo tiempo de pulso. De manera similar, los servos de rotación continua giran a diferentes velocidades. Debido a esto, ha sido necesario realizar pruebas para determinar los valores límite y el intervalo disponible.
Para manejarlos, se ha hecho uso de la librería *Servo*, incluida en el Arduino IDE. Gracias a su operación *writeMicroseconds*, establecer el tiempo de pulsos resulta trivial, lo que agiliza el proceso de pruebas y la implementación final.

Para operar con el Shield SD ha sido necesario analizar la librería *SD* incluida de serie con el Arduino IDE. Hace uso del protocolo *Serial Peripherial Interface* (SPI), utilizado para realizar comunicaciones síncronas entre dispositivos. Al igual que I^2C, se basa en un modelo maestro-esclavo, y aunque permite transmisiones full-duplex con velocidades de reloj superiores a los 10MHz, requiere de 4 líneas de conexión(*Clock*, *Master Out/Slave In*, *Master In/Slave Out* y *Slave Selector*), lo que complica la interconexión de dispositivos. Podemos ajustar parámetros como la velocidad de reloj y el tipo de desplazamiento de bits, generalmente establecidos por los periféricos.

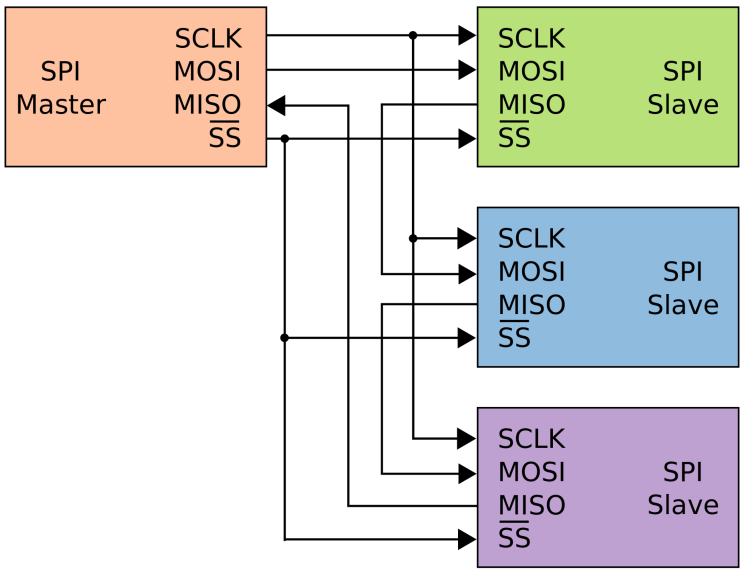


Figura 3.22: Representación de protocolo SPI con un maestro y tres esclavos. Imagen de Cburnett. Bajo licencia CC-BY-SA

En la mayoría de placas Arduino, esta interfaz emplea los pines digitales 11, 12 y 13 para el intercambio de información, además del pin adicional para habilitar o deshabilitar dispositivos, que varía según el periférico con el que se va a realizar la comunicación. En el caso concreto del Shield SD de Sparkfun, corresponde al pin número 8.

- Volcado y gestión de datos

  Como consecuencia de problemas expuestos anteriormente para realizar lecturas, el programa para el intercambio de datos con un equipo externo también se ha simplificado.

  La opción de establecer densidades variables se ha suprimido por la imposibilidad de aplicarlo de manera eficaz, mientras que la exploración de ficheros ha quedado sin implementar por falta de tiempo en el desarrollo del software externo.

  Por lo tanto, el programa se dedica exclusivamente a la transmisión de los datos recogidos, por medio de BLE. Para utilizar este protocolo de comunicación ha sido necesario el estudio del mismo, los métodos proporcionados por la librería *CurieBLE* (parte del *suite* incluido con el controlador de la placa) y su correcta implementación.
  Como parte del estándar Bluetooth 4.0, el sistema BLE ofrece un servicio de menor consumo y velocidad (aproximadamente 1Mbit/s) que la versión estándar. Las opciones que ofrece el dispositivo se presentan en servicios, y estos se dividen en características. Los sistemas externos, también conocidos como *centrales*, pueden acceder a cada servicio y característica mediante su UUID, y el intercambio de información ocurre en base a los cambios en las características. Podemos compararlo con un tablón de anuncios: cada servicio representa una hoja en el tablón, y cada característica un párrafo dentro de la hoja.

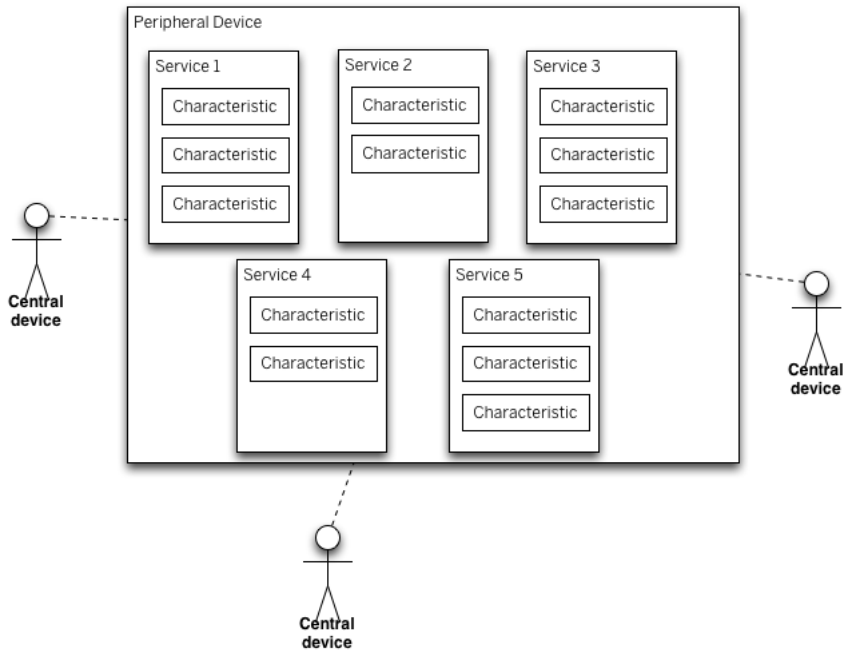


Figura 3.23: Modelo tablón de anuncios seguido por BLE. Imagen del equipo de arduino.cc. Bajo licencia CC-BY-SA

Las características pueden almacenar un máximo de 20 bytes, una restricción clave a la hora de diseñar las interfaces de comunicación. En función de la configuración que escojamos al inicializarlas, pueden ser de solo escritura para el periférico y de solo lectura para los dispositivos externos, u ofrecer lectura y escritura para ambos. También es posible dotarlas de notificaciones, para comunicar a las centrales conectadas de cambios en alguna característica, o ser dichos dispositivos externos los que soliciten este servicio de manera individual.

Para realizar la operación de envío de datos, se inicializa el periférico y se añaden los servicios y las características necesarias; en este caso, un único servicio con dos características. La primera cuenta con la opción de escritura para centrales habilitada para que estas, mediante el envío de cualquier valor distinto de 0, realicen una solicitud de información, a modo de *flag*. La segunda, configurada como solo lectura para dispositivos externos y con notificaciones habilitadas, es la encargada de transmitir los datos solicitados.

Cada vez que la central escribe sobre la característica *flag*, la estación lee 20 bytes

del fichero de datos almacenado en la tarjeta microSD y los escribe en la segunda característica. Estos datos son leídos y almacenados por la central, que repite la primera operación. Una vez se ha transmitido todo el fichero, el periférico escribe una secuencia de ceros, fácilmente identificable por el sistema externo, lo que permite detener el proceso.

4. CAPÍTULO

Desarrollo del software de administración de datos

4.1. Analisis inicial

Es imprescindible procesar los datos que hayamos recogido con la estación de topografía, para lo que es necesario utilizar un sistema informático externo. Cualquier usuario debería ser capaz de conectar el dispositivo y transmitir la información a dicho equipo, para poder después interpretarla con el programa apropiado.

Por ello, se requiere crear un programa que cumpla con los siguientes requisitos:

- Se creará una interfaz para la conexión entre un equipo informático y el dispositivo, que servirá para la transmisión de datos.
- Se requerirá de una rutina que organice los datos obtenidos de acuerdo con el formato requerido por el software anterior.
- Se utilizará el software Visual Topo para generar los modelos 3D a partir de los datos recogidos.

La elección de Visual Topo para obtener las imágenes finales viene dada por el equipo de espeleología de la Unión de Espeleólogos Vascos, puesto que es el programa que utilizan habitualmente para estas tareas.

A la hora de realizar este trabajo, hubo que decidir el lenguaje de programación a utilizar. El elegido finalmente fue Java gracias a su facilidad para crear interfaces gráficas, en especial cuando se complementa con el *addon* **WindowBuilder** del IDE Eclipse.

A pesar de utilizar Java, la librería utilizada para gestionar el servicio de Bluetooth hace uso de librerías nativas escritas en C++, lo que la hacen dependiente del sistema operativo. En este caso, se ha programado sobre un sistema Linux Ubuntu 17.04, por lo que los comandos para la compilación e instalación de la librería que aparecen más adelante solo funcionarán en sistemas Linux.

4.2. Funciones de comunicación

4.2.1. Planteamiento inicial

Al iniciar el programa, el usuario debe solicitar la conexión con la estación de topografía. Se realiza una búsqueda para encontrar todos los dispositivos BLE disponibles, y comparamos los identificadores para ver si alguno corresponde con el de la placa, cuyo UUID es conocido de antemano. Si está disponible, detenemos la búsqueda y nos comunicamos directamente utilizando su identificador.

Nuestra estación de topografía tan solo ofrece tres servicios: *densities*, *browse* y *transmit*. El primero sirve para la modificación de densidades, las cuales se encontrarán escritas en un fichero; el segundo nos permite explorar los ficheros que se encuentran en la tarjeta SD y recibirlos o borrarlos; y el último hará las veces de orden *express* para transmitir los últimos ficheros creados con la estación, sin pasar por el explorador de archivos.

Una vez recibidos los ficheros en nuestro sistema, el propio programa se encarga de gestionar su formato para que sea importado por Visual Topo. Además, se comunica a la estación que mueva dichos ficheros a un directorio nuevo junto con la fecha actual, para poder así tener ordenados los ficheros antiguos, a modo de backup.

4.2.2. Instalación de la librería TinyB

Para utilizar la interfaz de Bluetooth con dispositivos BLE ha sido necesario descargar una API personalizada. Aunque existen diferentes proyectos disponibles, se ha seleccionado la librería TinyB, ya que es parte del kit de desarrollo de Intel® para sus dispositivos IoT.

Esta librería hace uso de los servicios de BlueZ, el sistema de control de Bluetooth utilizado habitualmente en la mayoría de distribuciones Linux y que, al menos en Ubuntu, viene instalado por defecto. Para compilarla, necesitaremos instalar CMake:

```
sudo apt install cmake
```

Una vez instalado, establecemos algunas variables de entorno para ayudar a dicha utilidad a localizar los ficheros Java necesarios:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export JAVA_AWT_LIBRARY=/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/
            amd64/libawt.so
export JAVA_JVM_LIBRARY=/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/
            amd64/server/libjvm.so
export JAVA_INCLUDE_PATH=/usr/lib/jvm/java-8-openjdk-amd64/include
export JAVA_INCLUDE_PATH2=/usr/lib/jvm/java-8-openjdk-amd64/include/linux
export JAVA_AWT_INCLUDE_PATH=/usr/lib/jvm/java-8-openjdk-amd64/include
```

Aunque en teoría CMake debería ser capaz de localizarlos por sí mismo si los ficheros se encuentran en las localizaciones por defecto, la mayoría de usuarios suelen experimentar problemas, y eso es precisamente lo que ocurrió en este caso. Por lo tanto, es más que recomendable almacenar dicha información con antelación para evitar errores.

También es posible que algunos ficheros Java no se encuentren presentes en la jerarquía de directorios. Por razones desconocidas, los archivos *javac*, *javah*, *jni.h* y *jni_md.h*, necesarios para hacer uso de la interfaz nativa de Java, no venían incluidos en la instalación base. Los dos primeros se pueden obtener con el siguiente comando de instalación:

```
sudo apt install openjdk-8-jdk-headless
```

El paquete *headless*, a diferencia del paquete estándar, sí incluye dichos ficheros. En cuanto a los otros dos, fue necesario generarlos manualmente, copiando su contenido de la página web de Oracle® referente a la interfaz nativa de Java.

Una vez tenemos todos los ficheros necesarios presentes y correctamente referenciados, procedemos a su compilación e instalación. Desde el directorio base de la librería, ejecutamos la siguiente secuencia de comandos:

```
mkdir build
cd build
cmake -DBUILDJAVA=ON ..
make
sudo make install
```

Se generarán todos los ficheros necesarios, tanto los objeto de C++ como el JAR de Java. De manera opcional, podemos copiarel fichero *tinyb.jar* al directorio de librerias adicionales, en este caso:

```
sudo cp java/tinyb.jar /usr/lib/jvm/java-8-openjdk-amd64/jre/lib/ext/
```

Para poder utilizarla en Eclipse, es necesario modificar las propiedades del proyecto. Accedemos a la sección *Java Build Path*, en la que podemos editar los módulos de Java con los que podemos compilar nuestar aplicación. Si hemos realizado el comando de copia, comprobamos que se encuentra registrada en la librería general de Java, aunque siempre podemos añadirma como librería independiente si no la detecta o no hemos realizado la copia:

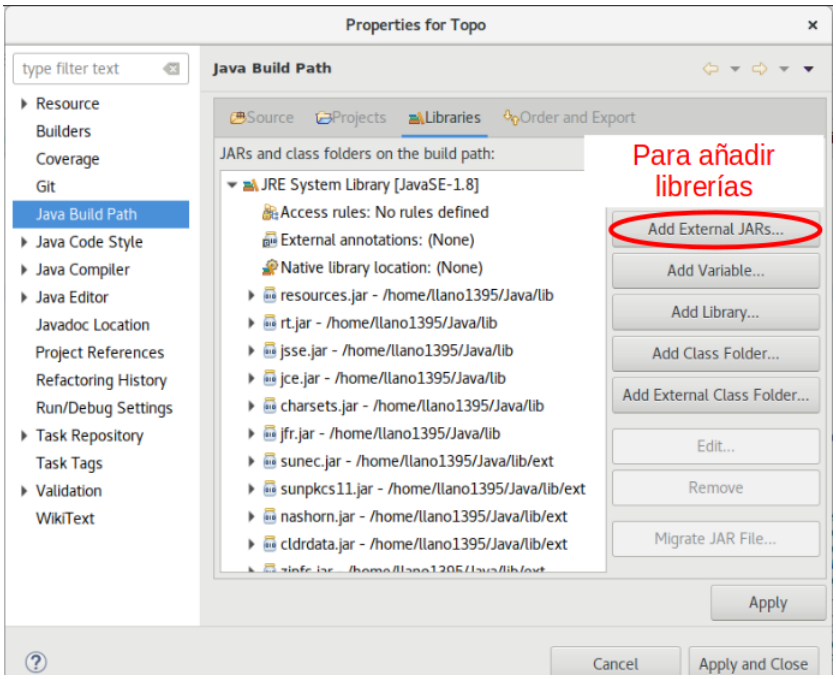


Figura 4.1: Ventana de propiedades del proyecto. Pestaña de librerías

Una vez tenemos la librería añadida, solo hace falta referenciar la librería nativa de la que

hace uso; es decir, los ficheros objeto (con extensión *.o*) compilados a partir del código C++. Por defecto, los encontraremos en la ruta */usr/local/lib*:

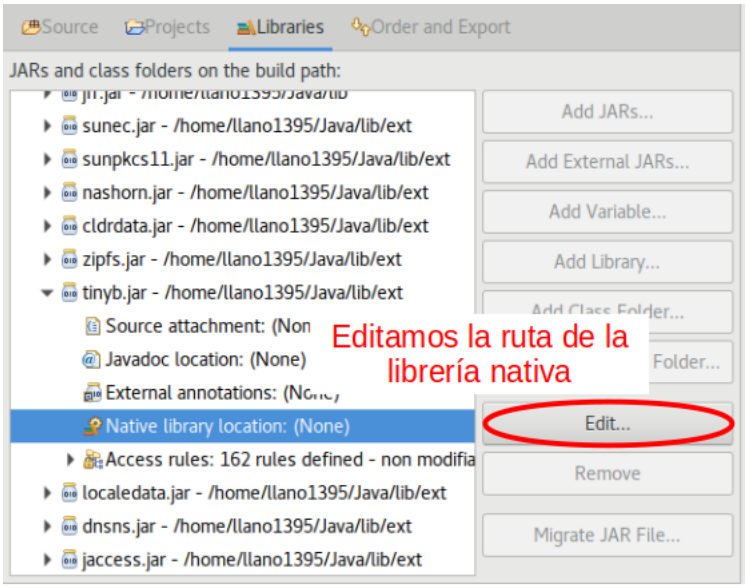


Figura 4.2: Edición de propiedades de módulo: librería nativa

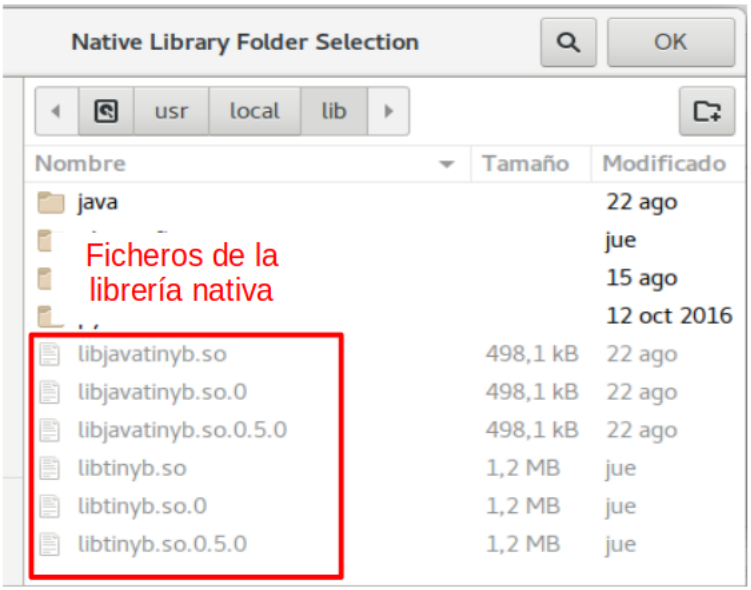


Figura 4.3: Ficheros compilados de la librería nativa

De esta forma, tendremos la capacidad de utilizar Java para la gestión de dispositivos con Bluetooth LE, haciendo uso de la librería TinyB.

4.2.3. Versión final

En el capítulo 3 se ha descrito una reducción de servicios en el programa de volcado de datos. Por lo tanto, las funciones del software externo han sido reducidas en consonancia.

Tras establecer la conexión con la estación de topografía, el usuario tan solo dispondrá de una opción: recibir el fichero de lecturas de distancias. Aunque no han existido problemas técnicos para el desarrollo del explorador de archivos, se ha consideraddo una funcionalidad de baja prioridad, y finalmente no ha habido tiempo para su desarrollo.

Para realizar la conexión, se inicializa un objeto *BluetoothManager* encargado de gestionar la interfaz de Bluetooth de nuestro equipo. Se realiza un barrido para identificar los dispositivos BLE visibles y, si está disponible, establecemos conexión con la placa Arduino. Dado que necesitamos conocer el UUID que presenta el módulo BLE para establecer dicha conexión, se necesitó realizar una prueba de exploración previa y almacenar dicho dato.

Una vez conectados, el programa espera a que el usuario ejecute la función de transmisión de datos. Este proceso funciona tal y como se explicó en el capítulo anterior: se escribe sobre la característica *flag* cualquier valor distinto de 0 y se leen los 20 bytes escritos por la placa en la otra característica, los cuáles serán escritos en un fichero en el equipo, situado en la carpeta en la que se encuentra el ejecutable del programa.
El proceso se repite hasta que se identifica una cadena de al menos cuatro caracteres '0' consecutivos, indicativo de que el fichero ha sido leído por completo y momento en el que el programa termina.

En cuanto al formato para ser interpretado por Visual Topo, a pesar de no poseer precisión suficiente para obtener un modelo fiable, se ha implementado una función para intentar obtener un fichero con formato similar a los que genera dicha aplicación. Pero debido a su complejidad, además de no contar con todos los datos necesarios, ha quedado sin llegar a funcionar correctamente.

4.3. Interfaz gráfica

Para facilitar todas las tareas anteriores a los usuarios, se ha diseñado una interfaz gráfica. Originalmente se componía de tres ventanas: menú principal, modificación de densidades

y explorador de ficheros. Pero debido a la reducción de servicios, tan solo era necesaria la primera de ellas.

El menú principal consta de dos botones. Uno para conectar y desconectar el equipo con el dispositivo, y otro para iniciar la secuencia de transmisión de datos. Para utilizar el segundo es necesario disponer de una conexión activa.

Al comenzar la transmisión, aparecerá un mensaje informando de que el paso de datos se está llevando a cabo. Durante este tiempo, los botones quedan deshabilitados. Cuando termina el paso del fichero, se informa de ello al usuario y se desbloquean los elementos de la ventana.

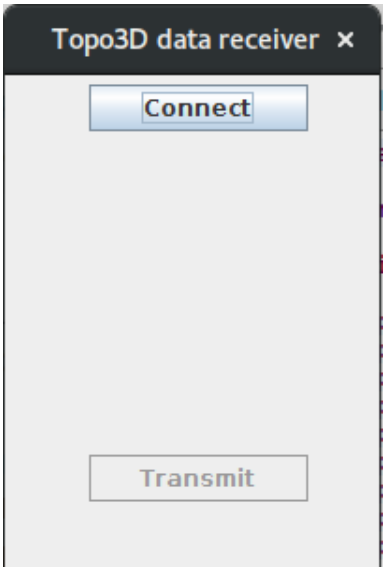


Figura 4.4: Menú principal de la interfaz, previa conexión

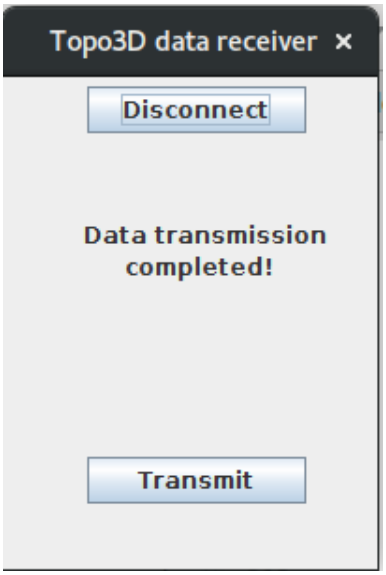


Figura 4.5: Transmisión completada

5. CAPÍTULO

Resultados

5.1. Resumen

- No se ha conseguido un programa eficaz para realizar los barridos, debido a la imprecisión del servomotor de rotación continua.
- El láser LIDAR-LiteV3 de Garmin aporta medidas muy precisas y rápidas, resulta muy fiable.
- El resto de elementos ofrecen prestaciones adecuadas para cubrir las neecesidades de la estación de topografía.
- La placa Arduino 101 maneja con solvencia todos los dispositivos.
- La interfaz de Bluetooth implementada permite comunicar y transmitir información entre dispositivos BLE.

En resumen: a pesar de no haber logrado una estación de topografía funcional debido a un único elemento, el resto de elementos e implementaciones funcionan de manera satisfactoria.

5.2. Detalles

5.2.1. Estación de topografía 3D

Aunque la implementación del programa de barridos no es complicada, la irregularidad en el comportamiento del servomotor de rotación continua ha impedido obtener un programa eficaz. No afecta especialmente a una sola ronda de lecturas, pero es realmente significativo tras varios giros completos. Por contra, se ha comprpbado que el uso del servomotor estándar para el eje vertical es más que adecuado, gracias a su precisión en ángulos pequeños.

Se han podido probar las capacidades del láser seleccionado para este proyecto, con resultados más que satisfactorios. En las pruebas realizadas, se han obtenido lecturas precisas sin necesidad de detener el servomotor horizontal, obteniendo un valor de distancia cada aproximadamente 3º de giro. Su alta velocidad para tomar lecturas de distancias y su precisión de +/- 1cm lo convierten en un elemento más que aceptable para una estación de topografía real, siempre que no se requiera medir espacios con distancias superiores a los 40 metros.
Como dato de referencia, si se toma una lectura por cada grado vertical y horizontal (360 * 90) obtenemos 32.400 puntos. Teniendo en cuenta que la frecuencia media del láser es de unos 270Hz, podemos recoger esa información en un tiempo de entre 2 y 3 minutos con un motor lo suficientemente rápido. Incluso si preferimos que opere más lentamente para asegurar datos fiables, obtener una densidad tan alta de puntos en algo más de 5 minutos supone un gran avance con respecto a la actual toma de puntos manual.

Se destaca también el buen rendimiento del Shield SD de Sparkfun. Teniendo en cuenta que las operaciones sobre memorias siempre son relativamente lentas, la escritura de un punto recogido por el láser apenas supone un par de milisegundos. Aunque es un tiempo significativo, es lo suficientemente bajo como para no dispara el tiempo necesario para realizar un barrido.

En cuanto a la batería seleccionada, sus prestaciones resultan adecuadas para la estación de topografía. Proporciona intensidad y voltaje suficientes para alimentar sin problemas todos los dispositivos a la vez. Además, tras comprobar el gasto de energía tras los barridos completos, se estima que una carga completa puede ofrecer, como mínimo, 30 secuencias. En total, se pueden recoger mas de 300.000 puntos, una cantidad alta para la gran mayoría de cavidades subterráneas.

5.2.2. Software de administración de datos

Con respecto al software externo a la placa, tan solo se ha implementado la interfaz de comunicación de BLE, de forma satisfactoria. Todos los pasos son ejecutados de manera correcta: toma de control del dispositivo de Bluetooth a traves de la interfaz de BlueZ, localización y conexión con la placa Arduino y transmisión de datos. Por lo tanto, disponemos de la capacidad de enviar información a un equipo externo de manera inalámbrica.

Como punto negativo, el tiempo necesario para la transmisión de ficheros resulta bastante alta. BLE fue diseñado con el objetivo de comunicar dispositivos mediante mensajes cortos con un gasto de energía inferior. Por ello, la transmisión de algo más de 50KBs necesita algo más de 10 minutos para completarse.

6. CAPÍTULO

Conclusiones

6.1. Técnicas

Las conclusiones finales del proyecto se presentan a continuación:

- A nivel técnico, el proyecto no ha sido excesivamente complejo. Trabajar con Arduino resulta relativamente fácil hoy en día, gracias a la multitud de librerías ya existentes. Además, los fabricantes de los distintos componentes se ocupan en la mayoría de los casos de proporcionar APIs para sus productos, evitando al usuario el estudio y la manipulación de las variables y configuraciones disponibles, o tener que buscar soluciones de terceros.

  Gracias a ello, el programa base de la estación ha sido bastante sencilla de programar. Por desgracia, el servomotor de rotación continua ha impedido obtener un dispositivo capaz de realizar lecturas precisas. Aunque el resto de componentes satisfacen con creces las necesidades que cubren, es necesario buscar un sistema alternativo para el giro del eje horizontal si se desea continuar con el desarrollo de la estación de topografía.

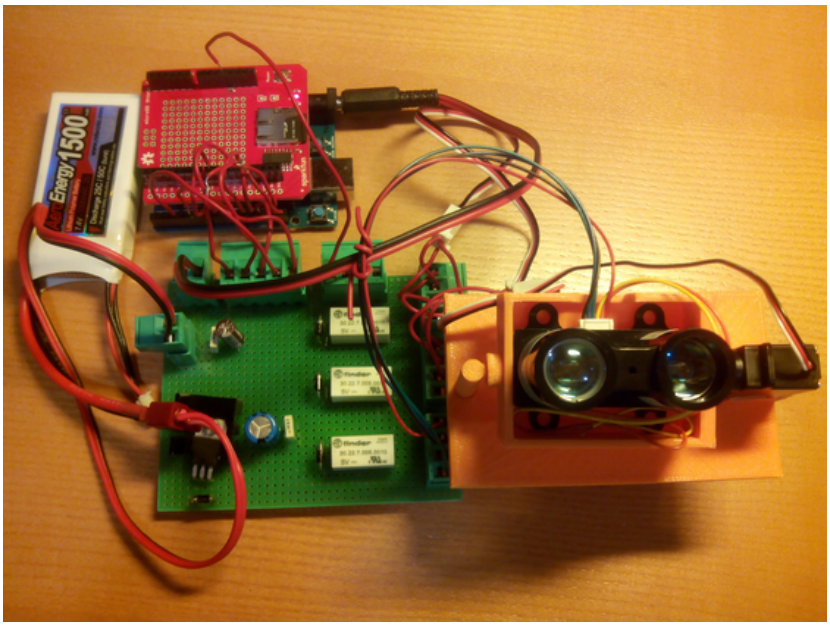


Figura 6.1: Interconexión de todos los elementos eleéctricos mientras se realizan pruebas

- En cuanto al software de administración de datos, el hecho de escoger Java ha facilitado y agilizado mucho su desarrollo gracias a su sencillez. Además, como ya se mencionó en el capítulo 4, la IDE Eclipse ofrece herramientas y opciones que simplifican la creación de interfaces gráficas.

  La necesidad de trabajar con Bluetooth Low Energy supuso sin duda el mayor *hándicap*. Aunque por fortuna la propia Intel ® dispone de una librería apropiada para ello, su compilación requirió de bastante tiempo debido a las malas distribuciones de Java en sistemas Linux y a los problemas de funcionamiento de CMake para explorar las jerarquías de ficheros. Además, fue necesario estudiar el funcionamiento de Bluetooth LE y de la librería.

  Con todo, el desarrollo no ha resultado complicado, aunque ha requerido de una inversión de tiempo bastante superior a lo esperado. Se ha conseguido implementar con éxito la transmisión inalámbrica de los ficheros, y se ha demostrado que puede también ser utilizado para transmitir cantidades relativamente grandes de datos, a pesar de su limitación de velocidad. Con el auge de los *wearables* y los dispositivos IoT, es sin duda un protocolo de gran utilidad y casi imprescindible para la comunicación entre dispositivos.

En conclusión, la creación de una estación de topografía eficaz y de bajo coste es posible. Tan solo es necesario cambiar el servo horizontal por un sistema preciso de giro para lograr un dispositivo que funcione de acuerdo con las necesidades de los espeleólogos. En cuanto a software a desarrollar, dado que las interfaces de comunicación están creadas y funcionan sin problemas tanto en la estación como en dispositivos externos, se limitaría a introducir funcionalidades adicionales; por ejemplo, la posibilidad de añadir densidades personalizadas o configurar el envío de avisos durante la ejecución del barrido a algún dispositivo con el que cuente el espeleólogo.

6.2. Planificación

Si comparamos la planificación inicial con el trabajo finalmente realizado, existen ciertos puntos que conviene destacar.

En términos generales, el desarrollo de las diferentes tareas se ha seguido según el orden que fue establecido en su momento. En cuanto al tiempo invertido, aunque el total de horas no supera las 300 que corresponden al proyecto, se ha distribuido de manera diferente a la esperada.
Las tareas relacionadas con la estructura externa han consumido aproximadamente 80 horas, un valor bastante por encima de las 60 estimadas. En cambio, el análisis de los diferentes elementos que componen la estación ha requerido de no mas de 25 horas, lo que compensa el exceso anterior.
La distribución de dichas horas a lo largo de las semanas, sin embargo, se ha visto retrasada en todos los casos. Las tareas asignadas en las diferentes asignaturas del curso lectivo han supuesto posponer trabajo del proyecto a fechas posteriores. El proyecto ha sido finalizado un mes después de lo planeado.

Durante el desarrollo del las últimas fases del proyecto, que según se estableció corresponde a la programación de los diferentes elementos, ha quedado patente que el orden de tareas establecido en la metodología no era el adecuado. La estructura de la estación debería haber sido el último elemento a desarrollar. Adelantar la implementación de los programas habría ayudado a identificar la decisión errónea de utilizar un servo de rotación continua con tiempo suficiente para encontrar una solución. En cambio, la estación ha quedado incompleta por falta de tiempo para cubrir esta deficiencia.

En resumen, la planificación se ha seguido con relativa normalidad en horas invertidas pero con un desfase importante en cuanto al número de semanas ha utilizar. Además, el

ruta de desarrollo escogida no ha sido la adecuada para identificar problemas nucleares a tiempo. Para futuros proyectos, es recomendable analizar con más detalle los posibles problemas que puedan encontrarse en cada tarea y ajustar las prioridades de desarrollo en consonancia.

6.3. Trabajo futuro

El alcance del proyecto no contempla el análisis de los siguientes apartados, aunque puede resultar de interés para quién desee continuar desarrollando o mejorando este dispositivo:

6.3.1. Raspberry Pi

Aunque Arduino ofrece suficientes recursos para este proyecto, un usuario experimentado en el uso de Raspberry Pi podría preferir el uso de uno de estos dispositivos. A nivel de cómputo, la segunda ofrece capacidades muy superiores, y podría utilizarse para procesar los datos en la propia placa, sin necesidad de recurrir a una aplicación de escritorio.

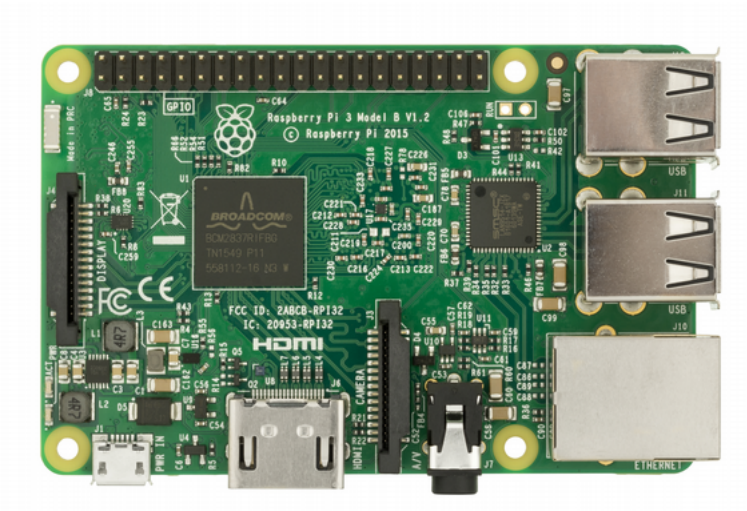


Figura 6.2: Raspberry Pi 3

El uso de una Raspberry Pi podría simplificar las tareas del usuario, ahorrando el paso de gestionar la información con un software intermedio, aunque sería necesario analizar la repercusión que tendría en la estación, en su diseño y en la interacción entre componentes.

6.3.2. Mejora de componentes

La estación de topografía actual cuenta con componentes modestos. Uno de los objetivos era el ahorro de costes para obtener un producto lo más asequible posible con unas prestaciones mínimas, suficiente para crear una prueba de concepto. Recordemos que existen equipos profesionales de este tipo en el mercado, pero a precios que superan con creces el presupuesto total de este proyecto.

Si analizamos una situación real, las especificaciones de la estación deberían ser superiores. Por ejemplo, algunas de las cavidades subterráneas que exploran los grupos de espeleólogos cuentan con galerías grandes, en las que se encuentran distancias muy por encima de los 40 metros, la máxima que puede registrar nuestra estación.

El giro de los ejes también podría ser mejorado, manejado por mecanismos mas complejos que solo un par de motores; un sistema de engranajes, por ejemplo, que ofrezca un precisión mucho más alta y permita escaneos muchos más precisos al usuario.

Existen muchas maneras de aumentar las capacidades del dispositivo, ya que lo desarrollado en este proyecto no es más que un concepto inicial, por lo que no estaría de más plantear una continuación para crear una versión mejorada y renovada.

Anexo A. Código fuente

full_sequence.ino

```
#include <Servo.h>
#include <Wire.h>
#include <LIDARLite.h>
#include <SPI.h>
#include <SD.h>

File myFile;
LIDARLite myLidarLite;
bool stop = false;
unsigned long sec, sec2;
int j = 0;
Servo myservo;  // create servo object to control a servo

Servo myservo2;  // create servo object to control a servo

int ms = 1320;    // variable to store the Y servo position
int ms2 = 1600;    // variable to store the X servo rotation speed



void setup() {
  myservo.attach(3);  // attaches the servo on pin 3 to the Y servo
  myservo2.attach(5);  // attaches the servo on pin 5 to the X servo

  myservo.writeMicroseconds(ms);              // tell servo to go to position in variable 'ms'
  myservo2.writeMicroseconds(1510);              // set servo initial rotation speed, 0 at this
     point
  delay(3000);

  // initialize SD Shield
  if (!SD.begin(8)) {
    return;
  }

  // open the file to write at
```

```
35    myFile = SD.open("realtest.txt", FILE_WRITE);
36
37
38    myFile.position();
39
40    // initialize laser
41    myLidarLite.begin(0, true); // Set configuration to default and I2C to 400 kHz
42
43    myFile.position();
44
45    // choose standard (0) preset for the laser
46    myLidarLite.configure(0); // Change this number to try out alternate configurations
47
48    myFile.position();
49
50    sec = millis();
51  }
52
53
54  // mutiple position(); operations are present. This keeps the SD Shield synchronized and avoids
        possible locks
55
56  // even though the cause of the locks is unknown, it may be because of how the SD library is
        implemented relaying upon
57  // old versions of sdfatlib.
58
59  void loop() {
60
61    if (!stop) {
62      myFile.position();
63      // while Y servo hasn't reached full vertical position
64      while (ms > 562) {
65        if (myFile) {
66          // measurement with bias correction. Recommended at least one for each 100 standard
        measurements
67          // write the data straight into the file
68          myFile.println(myLidarLite.distance());
69        } else {
70        }
71        myservo2.writeMicroseconds(1600);              // tell X servo to start rotating
72
73        for (int i = 0; i < 111; i++) {
74          if (myFile) {
75            // standard distance measurements written into the file for 360 rotation in axis X
76            myFile.println(myLidarLite.distance(false));
77          } else {
78          }
79          // delay added to give the servo time to rotate an appropriate amount of degrees
80          delay(59);
81        }
82
83        myFile.position();
```

```
84
85        //rotate axis X 360 in the opposite direction
86        myservo2.writeMicroseconds(1470);
87        for (int j = 0; j < 71; j++){
88          myFile.position();
89          // delay added to give the servo time to rotate an appropriate amount of degrees
90          delay(100);
91        }
92        // stop X servo
93        myservo2.writeMicroseconds(1505);
94
95        // decrease pulse width value for Y servo, to give approximately 1 rotation
96        ms -= 8;
97        // tell Y servo to move to the new position
98        myservo.writeMicroseconds(ms);
99        // gives times for the Y servo to reach its new position
100       delay(20);
101     }
102
103     // end of program; close file and stop process
104     myFile.close();
105     stop = true;
106   }
107
108 }
```

CallbackSD.ino

```
#include <CurieBLE.h>

#include <SPI.h>
#include <SD.h>

int pos, count, count2 = 0;
char cr;
char str[20], str2;
bool term = false;

// create service
BLEService dataService("19B10000-E8F2-537E-4F6C-D104768A1214"); // create service

// create characteristics and configure remote devices access
BLECharCharacteristic switchChar("19B10001-E8F2-537E-4F6C-D104768A1214", BLERead | BLEWrite);
BLECharacteristic fileinfo("19B10002-E8F2-537E-4F6C-D104768A1214", BLERead | BLENotify, 20);

File dataFile;

void setup() {
  // begin initialization
  BLE.begin();

  // set the local name peripheral advertises
  BLE.setLocalName("LEDCB");
  // set the UUID for the service this peripheral advertises
  BLE.setAdvertisedService(dataService);

  // add the characteristic to the service
  dataService.addCharacteristic(switchChar);
  dataService.addCharacteristic(fileinfo);


  // add service
  BLE.addService(dataService);


  // assign event handler for characteristic
  switchChar.setEventHandler(BLEWritten, switchCharacteristicWritten);
  // set an initial value for the characteristic
  switchChar.setValue(0);

  // start advertising
  BLE.advertise();

}

void loop() {
  // poll for BLE events
```

```
50    BLE.poll();
51    // keeps the SD Shield active and synchronized
52    dataFile.position();
53  }
54
55
56
57  // mutiple position(); operations are present. This keeps the SD Shield synchronized and avoids
        possible locks
58
59  // even though the cause of the locks is unknown, it may be because of how the SD library is
        implemented relaying upon
60  // old versions of sdfatlib.
61
62
63  // if a value is written to characteristic switchChar
64  void switchCharacteristicWritten(BLEDevice central, BLECharacteristic characteristic) {
65    if (switchChar.value()) {
66      if (!dataFile) {
67        //Initialize SD Shield
68        if (!SD.begin(8)) {
69          return;
70        }
71        // open file to be read
72        dataFile = SD.open("realtest.txt");
73      }
74      // while the file hasn't been read to its last character
75      if (!term) {
76        if (dataFile) {
77          // place cursor after the last byte read
78          pos = count2 * 20;
79          dataFile.seek(pos);
80          while (dataFile.available() && count < 20) {
81            cr = dataFile.read();
82            str[count] = cr;
83            count++;
84          }
85          count2++;
86          // covers a lack of bytes written whenever the file ends
87          if (!dataFile.available()) {
88            for (int i = count; i < 20; i++) {
89              str[i] = ' ';
90            }
91            term = true;
92          }
93          dataFile.position();
94          count = 0;
95          // value conversión for proper byte value of the character
96          for (int i = 0; i < 8; i++)
97          {
98            str2 |= str[i] << (7 - i);
99          }
```

```
100
101         dataFile.position();
102         // write the bytes read from the file into the characteristic
103         fileinfo.setValue((unsigned char *)str, 20);
104         dataFile.position();
105       }
106     } else {
107       dataFile.position();
108       // write and convert an array full of '0's
109       for (int i = 0; i < 20; i++) {
110         str[i] = '0';
111       }
112       for (int i = 0; i < 8; i++)
113       {
114         str2 |= str[i] << (7 - i);
115       }
116       fileinfo.setValue((unsigned char *)str, 20);
117       dataFile.position();
118     }
119   } else {
120   }
121 }
```

MainMenu.java

```
package gui;

import java.awt.BorderLayout;
import java.awt.Dialog.ModalityType;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.event.WindowEvent;
import java.nio.file.FileSystems;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardOpenOption;
import java.util.List;
import java.awt.event.WindowAdapter;
import tinyb.*;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import java.awt.GridLayout;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JTextField;
import javax.swing.JLabel;

public class MainMenu extends JFrame {

    private JPanel contentPane = null;
    private JTextField percent = null;
    private JButton btnConnect = null;
    private JButton btnDisconnect = null;
    private JButton btnDensities = null;
    private JButton btnBrowse = null;
    private JButton btnTransmit = null;
    private BluetoothDevice arduino;
    private JLabel lblRecievingData;
    private JLabel lblThisMayTake;
    private JLabel lblSomeMinutes;
    private JLabel lblDataTransmission;
    private JLabel lblCompleted;
    static boolean running = true;
    Path path = FileSystems.getDefault().getPath("Data1.tro");


    static void printDevice(BluetoothDevice device) {
        System.out.print("Address = " + device.getAddress());
        System.out.print(" Name = " + device.getName());
        System.out.print(" Connected = " + device.getConnected());
        System.out.println();
    }
```

```
50      /**
51       * Launch the application.
52       */
53      public static void main(String[] args) {
54          EventQueue.invokeLater(new Runnable() {
55              public void run() {
56                  try {
57                      MainMenu frame = new MainMenu();
58                      frame.setVisible(true);
59                  } catch (Exception e) {
60                      e.printStackTrace();
61                  }
62              }
63          });
64      }
65
66      /**
67       * Create the frame.
68       */
69      public MainMenu() {
70          setTitle("Topo3D data receiver");
71          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
72          setBounds(100, 100, 200, 300);
73          contentPane = new JPanel();
74          contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
75          setContentPane(contentPane);
76          contentPane.setLayout(null);
77
78          contentPane.add(getConnectButton(), null);
79          contentPane.add(getDisconnectButton(), null);
80  //      contentPane.add(getDensitiesButton(), null);
81  //      contentPane.add(getBrowseButton(), null);
82          contentPane.add(getTransmitButton(), null);
83
84          lblRecievingData = new JLabel("Recieving data");
85          lblRecievingData.setBounds(55, 62, 107, 15);
86          contentPane.add(lblRecievingData);
87
88          lblThisMayTake = new JLabel("This may take");
89          lblThisMayTake.setBounds(55, 121, 107, 15);
90          contentPane.add(lblThisMayTake);
91
92          lblSomeMinutes = new JLabel("some minutes");
93          lblSomeMinutes.setBounds(55, 148, 107, 15);
94          contentPane.add(lblSomeMinutes);
95
96          lblDataTransmission = new JLabel("Data transmission");
97          lblDataTransmission.setBounds(43, 77, 133, 15);
98          contentPane.add(lblDataTransmission);
99
100         lblCompleted = new JLabel("completed!");
101         lblCompleted.setBounds(65, 94, 86, 15);
```

```
102         contentPane.add(lblCompleted);
103 
104         lblRecievingData.setVisible(false);
105         lblThisMayTake.setVisible(false);
106         lblSomeMinutes.setVisible(false);
107         lblDataTransmission.setVisible(false);
108         lblCompleted.setVisible(false);
109 
110     }
111     private JButton getConnectButton() {
112         if (btnConnect == null) {
113             btnConnect = new JButton();
114             btnConnect.setText("Connect");
115             btnConnect.setEnabled(true);
116             btnConnect.setVisible(true);
117             btnConnect.setBounds(45, 5, 117, 25);
118             btnConnect.addActionListener(new java.awt.event.ActionListener() {
119                 @Override
120                 public void actionPerformed(java.awt.event.ActionEvent e) {
121                     try {
122                     /*
123                      * Initialization of the TinyB library.
124                      */
125                     BluetoothManager manager = BluetoothManager.getBluetoothManager();
126 
127                     /*
128                      * Enable device discovery mode
129                      */
130                     boolean discoveryStarted = manager.startDiscovery();
131 
132                     arduino = getDevice("98:4F:EE:10:BA:A0");
133 
134                     /*
135                      * After we find the device, stop looking for other devices.
136                      */
137                     try {
138                         manager.stopDiscovery();
139                     } catch (BluetoothException er) {
140                         System.err.println("Discovery could not be stopped.");
141                     }
142 
143                     if (arduino == null) {
144                         System.err.println("Arduino 101 not found for that address.");
145                         System.exit(-1);
146                     }
147 
148                     System.out.print("Found device: ");
149                     printDevice(arduino);
150 
151                     if (arduino.connect())
152                         System.out.println("Connected to Arduino 101");
153                     else {
```

```
154                         System.out.println("Could not connect device.");
155                         System.exit(-1);
156                     }
157                     btnTransmit.setEnabled(true);
158                     btnConnect.setEnabled(false);
159                     btnConnect.setVisible(false);
160                     btnDisconnect.setEnabled(true);
161                     btnDisconnect.setVisible(true);
162                     } catch (Exception ex) {
163                         ex.printStackTrace();
164                     }
165                 }
166             });
167         }
168         return btnConnect;
169     }
170 
171     private JButton getDisconnectButton() {
172         if (btnDisconnect == null) {
173             btnDisconnect = new JButton();
174             btnDisconnect.setText("Disconnect");
175             btnDisconnect.setBounds(45, 5, 117, 25);
176             btnDisconnect.setEnabled(false);
177             btnDisconnect.setVisible(false);
178             btnDisconnect.addActionListener(new java.awt.event.ActionListener() {
179                 @Override
180                 public void actionPerformed(java.awt.event.ActionEvent e) {
181                         arduino.disconnect();
182                         btnTransmit.setEnabled(false);
183                         btnConnect.setEnabled(true);
184                         btnConnect.setVisible(true);
185                         btnDisconnect.setEnabled(false);
186                         btnDisconnect.setVisible(false);
187                         lblDataTransmission.setVisible(false);
188                         lblCompleted.setVisible(false);
189                 }
190             });
191         }
192         return btnDisconnect;
193     }
194 
195     private JButton getTransmitButton() {
196         if (btnTransmit == null) {
197             btnTransmit = new JButton();
198             btnTransmit.setText("Transmit");
199             btnTransmit.setEnabled(false);
200             btnTransmit.setBounds(45, 203, 117, 25);
201             btnTransmit.addActionListener(new java.awt.event.ActionListener() {
202                 @Override
203                 public void actionPerformed(java.awt.event.ActionEvent e) {
204                     try {
205                         btnTransmit.setEnabled(false);
```

```
206                         btnDisconnect.setEnabled(false);
207                         lblRecievingData.setVisible(true);
208                         lblThisMayTake.setVisible(true);
209                         lblSomeMinutes.setVisible(true);
210                         lblDataTransmission.setVisible(false);
211                         lblCompleted.setVisible(false);
212                     BluetoothGattService sdService= getService(arduino, "19b10000-e8f2-537e-4f6c-
    d104768a1214");
213
214                     if (sdService == null) {
215                         System.err.println("This device does not have the data service we are
    looking for.");
216                         arduino.disconnect();
217                         System.exit(-1);
218                     }
219                     System.out.println("Found service " + sdService.getUUID());
220
221                     BluetoothGattCharacteristic flagValue = getCharacteristic(sdService, "19
    b10001-e8f2-537e-4f6c-d104768a1214");
222                     BluetoothGattCharacteristic dataValue = getCharacteristic(sdService, "19
    b10002-e8f2-537e-4f6c-d104768a1214");
223
224                     if (flagValue == null || dataValue == null) {
225                         System.err.println("Could not find the correct characteristics.");
226                         arduino.disconnect();
227                         System.exit(-1);
228                     }
229
230                     System.out.println("Found the data characteristics");
231
232
233
234                     /*
235                      *
236                      */
237                     byte[] config = { 0x01 };
238                     flagValue.writeValue(config);
239                     Thread.sleep(20);
240                     byte[] dataRaw = dataValue.readValue();
241                     char[] endChecker = {'1', '1', '1', '1'};
242
243                     while (running) {
244                         flagValue.writeValue(config);
245                         Thread.sleep(20);
246                         dataRaw = dataValue.readValue();
247                         for(int i = 0; i < 4; i++) {
248                             endChecker[i] = (char) (dataRaw[i] & 0xFF);
249                         }
250                         if(endChecker[0] == '0' && endChecker[1] == '0' && endChecker[2] == '0'
    && endChecker[3] == '0')
251                             running = false;
252                         else
```

```
253
254                         Files.write(path, dataRaw,StandardOpenOption.CREATE,
    StandardOpenOption.APPEND);
255                     }
256                 btnTransmit.setEnabled(true);
257                 btnDisconnect.setEnabled(true);
258                 lblRecievingData.setVisible(false);
259                 lblThisMayTake.setVisible(false);
260                 lblSomeMinutes.setVisible(false);
261                 lblDataTransmission.setVisible(true);
262                 lblCompleted.setVisible(true);
263
264                 } catch (Exception ex) {
265                     ex.printStackTrace();
266                 }
267             }
268         });
269     }
270     return btnTransmit;
271 }
272
273 static BluetoothDevice getDevice(String address) throws InterruptedException {
274     BluetoothManager manager = BluetoothManager.getBluetoothManager();
275     BluetoothDevice arduino = null;
276     for (int i = 0; (i < 15) && running; ++i) {
277         List<BluetoothDevice> list = manager.getDevices();
278         if (list == null)
279             return null;
280
281         for (BluetoothDevice device : list) {
282             printDevice(device);
283             /*
284              * Here we check if the address matches.
285              */
286             if (device.getAddress().equals(address))
287                 arduino = device;
288         }
289
290         if (arduino != null) {
291             return arduino;
292         }
293         Thread.sleep(4000);
294     }
295     return null;
296 }
297
298 /*
299  * Our device should expose a data transmission service,
300  */
301 static BluetoothGattService getService(BluetoothDevice device, String UUID) throws
    InterruptedException {
302     System.out.println("Services exposed by device:");
```

```
303         BluetoothGattService tempService = null;
304         List<BluetoothGattService> bluetoothServices = null;
305         do {
306             bluetoothServices = device.getServices();
307             if (bluetoothServices == null)
308                 return null;
309 
310             for (BluetoothGattService service : bluetoothServices) {
311                 System.out.println("UUID: " + service.getUUID());
312                 if (service.getUUID().equals(UUID))
313                     tempService = service;
314             }
315             Thread.sleep(4000);
316         } while (bluetoothServices.isEmpty() && running);
317         return tempService;
318     }
319 
320     static BluetoothGattCharacteristic getCharacteristic(BluetoothGattService service, String
      UUID) {
321         List<BluetoothGattCharacteristic> characteristics = service.getCharacteristics();
322         if (characteristics == null)
323             return null;
324 
325         for (BluetoothGattCharacteristic characteristic : characteristics) {
326             if (characteristic.getUUID().equals(UUID))
327                 return characteristic;
328         }
329         return null;
330     }
331 
332 /*  private JButton getDensitiesButton() {
333         if (btnDensities == null) {
334             btnDensities = new JButton();
335             btnDensities.setText("Densities");
336             btnDensities.setBounds(45, 82, 117, 25);
337             btnDensities.addActionListener(new java.awt.event.ActionListener() {
338                 @Override
339                 public void actionPerformed(java.awt.event.ActionEvent e) {
340                     DensitiesGUI.main(null);
341                 }
342             });
343         }
344     return btnDensities;
345     }
346 
347     private JButton getBrowseButton() {
348         if (btnBrowse == null) {
349             btnBrowse = new JButton();
350         btnBrowse.setText("Browse");
351             btnBrowse.setBounds(45, 143, 117, 25);
352             btnBrowse.addActionListener(new java.awt.event.ActionListener() {
353                 @Override
```

```
354                 public void actionPerformed(java.awt.event.ActionEvent e) {
355                     BrowserGUI.main(null);
356                 }
357             });
358         }
359         return btnBrowse;
360     }
361 */
362 
363 }
```

Bibliografía

[1] Manual de uso y Especificaciones Técnicas del láser Garmin LIDAR-LiteV3 http://static.garmin.com/pumac/LIDAR_Lite_v3_Operation_Manual_and_Technical_Specifications.pdf

[2] Página principal Genuino 101 https://store.arduino.cc/genuino-101

[3] Sparkfun microSD Shield https://www.sparkfun.com/products/12761

[4] SpringRC SM-S2309S 180º Analog Servo Specifications https://servodatabase.com/servo/springrc/sm-s2309s

[5] SpringRC SM-S4303R Continuous Rotation Servo Datasheet https://www.sparkfun.com/datasheets/Robotics/servo-360_e.pdf

[6] Página principal Adafruit Compass Board HMC5883L https://www.adafruit.com/product/1746

[7] Tutorial I^2C https://learn.sparkfun.com/tutorials/i2c

[8] Tutorial SPI https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/

[9] Referencia librería SD de Arduino https://www.arduino.cc/en/Reference/SD

[10] Referencía librería CurieBLE de Arduino https://www.arduino.cc/en/Reference/CurieBLE

[11] Referencia librería CurieIMU de Arduino https://www.arduino.cc/en/Reference/CurieIMU

[12] Demostración PWM con servomotor SM-S4303R https://www.khanacademy.org/computer-programming/pulse-width-demo/4961611581489152

[13] Proyecto TinyB en GitHub https://github.com/intel-iot-devkit/tinyb/

[14] Referencia de todas las variables de entorno necesarias para compilar la librería TinyB https://github.com/eclipse/kura/issues/1164

[15] Página principal d la Interfaz Nativa de Java. Incluye el código de **jawt.h** y **jawt_md.h** https://docs.oracle.com/javase/8/docs/technotes/guides/awt/AWT_Native_Interface.html

[16] 7805 voltage regulator datasheet https://www.sparkfun.com/datasheets/Components/LM7805.pdf

[17] Finder 30 Series relays datasheet https://www.finder-relais.net/en/finder-relays-series-30.pdf

[18] Intel® Curie™ products End-of-Life Notification https://qdms.intel.com/dm/i.aspx/43A2E4E9-B349-4FF6-AB6B-237BE2F07D2C/PCN115578-00.pdf