UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE MÁSTER

“Desarrollo e implementación de un escáner tridimensional para recintos cerrados basados en LiDAR”

Titulación: Máster Universitario en Ingeniería de Telecomunicación

Autor: D. Alejandro Santana Pérez

Tutores: Dr. D. Alfonso Medina Escuela

Dr. D. Roberto Esper-Chaín Falcón

Fecha: Julio de 2017

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE MÁSTER

“Desarrollo e implementación de un escáner tridimensional para recintos cerrados basados en LiDAR”
HOJA DE FIRMAS

Alumno/a

Fdo.: Alejandro Santana Pérez

Tutores

Fdo.: Dr. D. Alfonso Medina Escuela Fdo.: Dr. D. Roberto Esper-Chaín Falcón

Fecha: Julio de 2017

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE MÁSTER

“Desarrollo e implementación de un escáner tridimensional para recintos cerrados basados en LiDAR”

HOJA DE EVALUACIÓN

Calificación: _____

Presidente

Fdo.:

Vocal

Fdo.:

Secretario/a

Fdo.:

Fecha: Julio de 2017

INDICE GENERAL

INDICE GENERAL I
INDICE de FIGURAS V
INDICE de TABLAS IX
INDICE de CODIGOS XI
Acrónimos XIII
1 Introducción 1
1.1 Antecedentes 1
1.2 Objetivos 2
1.3 Contenido de la memoria 3
1.4 LiDAR 5
1.4.1 Introducción 5
1.4.2 Principios y fundamentos de la tecnología láser 7
1.4.3 Tipos de LiDAR 8
1.4.4 Formas de medir distancias con la luz 9
1.4.4.1 Triangulación 10
1.4.4.2 Medición basada en pulsos 11
1.4.4.3 Medición basada en fases 13
1.4.5 Errores en las medidas LiDAR 14
1.4.6 Normativa de seguridad en el manejo de láseres 17
1.5 Solución adoptada 18
2 Descripción del Hardware 21
2.1 Introducción 21
2.2 Microcontrolador 21
2.2.1 Introducción 21
2.2.2 Familia STM32 22
2.2.3 Familia Cortex 22
2.2.4 Arquitectura STM32F4 24
2.2.4.1 Mapa de Memoria 24
2.2.4.2 Oscilador y señales de reloj 25
2.2.5 Fuente de alimentación y circuito de reset 25
2.2.6 Periféricos 26
2.3 Tarjeta de desarrollo STM32F4 Discovery 31
2.3.1 Introducción 31
2.3.2 Características generales 31
2.4 LiDAR RPLIDAR A1 32
2.4.1 Introducción 32
2.4.2 Arquitectura del sistema 33
2.4.3 Interfaz de comunicación 35
2.5 Motor stepper 37
2.5.1 Introducción 37
2.5.2 Características principales 37
2.5.3 Driver A4988 38
2.6 Motor servo Parallax 39

2.6.1 Introducción 39
2.6.2 Características principales 39
2.7 Sensor de efecto Hall DRV5053 e imán 40
2.7.1 Introducción 40
2.7.2 Características principales del sensor de efecto Hall 40
2.7.3 Imán 41
3 Diseño del sistema electromecánico 43
3.1 Introducción 43
3.2 Objetivos y consideraciones previas del diseño y fabricación 43
3.3 Piezas del diseño 44
3.3.1 Base inferior 44
3.3.2 Base superior 45
3.3.3 Soporte del stepper 46
3.3.4 Soporte del servo 47
3.3.5 Base giratoria del stepper 48
3.3.6 Base del LiDAR 49
3.3.7 Contrapeso 50
3.4 Integración del sistema electromecánico 50
3.5 Configuración electrónica del sistema diseñado 51
3.5.1 Introducción 51
3.5.2 Motor stepper Nema-17 51
3.5.3 Motor servo Parallax 55
3.5.4 Sensor de efecto Hall 57
4 Integración electrónica 59
4.1 Introducción 59
4.2 Materiales y componentes utilizados 59
4.3 Montaje del prototipo 59
4.4 Etapa electromecánica 61
4.5 Etapa de adquisición y procesamiento 64
4.6 Etapa de representación tridimensional 66
5 Diseño firmware 67
5.1 Introducción 67
5.2 Estructura del código 67
5.3 Configuración de periféricos 69
5.3.1 Introducción 69
5.3.2 GPIO 69
5.3.3 USART 70
5.3.4 Timers 70
5.3.5 NVIC 72
5.3.6 RCC 73
5.4 Descripción general de funcionamiento 73
5.5 Programa principal 77
5.6 Etapa de lectura del USB 78
5.7 Etapa de comunicación LiDAR 79
5.8 Etapa de adquisición y procesamiento de datos 92
5.9 Etapa de envío de datos por USB 99
5.10 Etapa de fin de operación 100

6 Desarrollo de la aplicación de visualización 103
6.1 Introducción 103
6.2 Librerías usadas 104
6.3 Descripción general de la aplicación 105
6.4 Inicialización del programa 108
6.5 Etapa de adquisición y procesamiento 114
6.6 Etapa de representación 119
7 Resultados y revisión de objetivos 123
7.1 Resultados 123
7.2 Revisión de objetivos 128
8 Conclusiones y líneas futuras 129
Presupuesto 131
Recursos materiales 131
Recursos Hardware 131
Recursos Software 132
Trabajo tarifado por tiempo empleado 133
Material fungible 134
Aplicación de impuestos y coste total 134
Bibliografía 137
ANEXO 1 141
ANEXO 2 149
ANEXO 3 155

INDICE de FIGURAS

Figura 1.1. Tipos de LiDAR en función del rango de detección. 7
Figura 1.2. Espectro electromagnético. 8
Figura 1.3. Principios de la triangulación. 10
Figura 1.4. Triangulación en LiDAR. 11
Figura 1.5. Ejemplos de LiDAR de triangulación. 11
Figura 1.6. Medición basada en pulsos. 12
Figura 1.7. Ejemplos de LiDAR basados en pulsos. 13
Figura 1.8. Medición basada en fases. 13
Figura 1.9. Ejemplos de LiDAR basados en fases. 14
Figura 1.10. Reflexión sobre una superficie Lambertiana. 16
Figura 1.11. Efectos de la refracción en materiales semitransparentes. 16
Figura 1.12. Diagrama de bloques de la solución adoptada. 19
Figura 1.13. Esquema de la solución adoptada. 20
Figura 2.1. Dispositivos familia STM32 de alto rendimiento 22
Figura 2.2. Arquitectura ARM Cortex-M4. 23
Figura 2.3. Arquitectura STM32F407. 24
Figura 2.4. Mapa de Memoria. 25
Figura 2.5. Tarjeta de desarrollo. 32
Figura 2.6. Arquitectura del sistema. 33
Figura 2.7. Triangulación láser. 34
Figura 2.8. Cálculo de la distancia y el ángulo. 35
Figura 2.9. Esquema de conexiones RPLIDAR A1. 36
Figura 2.10. Motor Stepper. 37
Figura 2.11. Driver A4988. 38
Figura 2.12. Motor servo y configuración. 40
Figura 2.13. Pines DRV5053. 41
Figura 2.14. Imán. 41
Figura 3.1. Sistema electromecánico final 44
Figura 3.2. Base inferior. 45
Figura 3.3. Dimensiones rodamiento de bolas. 46
Figura 3.4. Base superior. 46
Figura 3.5. Soporte del stepper. 47
Figura 3.6. Soporte del servo. 48
Figura 3.7. Base giratoria del stepper. 49
Figura 3.8. Base del LiDAR. 49
Figura 3.9. Contrapeso. 50
Figura 3.10. Sistema electromecánico final. 51
Figura 3.11. Tipos de microstepping. 52
Figura 3.12. Período de trabajo del servo. 56
Figura 4.1. Esquemático del prototipo. 60
Figura 4.2. Prototipo real. 60

INDICE de FIGURAS

Figura 4.3. Etapa electromecánica. 61
Figura 4.4. Sensor Hall para el stepper. 63
Figura 4.5. Sensor Hall para el start del servo. 63
Figura 4.6. Sensor Hall para el stop del servo. 64
Figura 4.7. Conexiones driver A4988. 65
Figura 4.8. Etapa de adquisición y procesamiento. 65
Figura 4.9. Etapa de representación tridimensional. 66
Figura 5.1. Estructura del código 68
Figura 5.2. Diagrama general de funcionamiento del firmware. 76
Figura 5.3. Diagrama de flujo del programa principal. 78
Figura 5.4. Diagrama de flujo de la etapa de lectura del USB. 79
Figura 5.5. Modo solicitud-única respuesta. 80
Figura 5.6. Modo solicitud-múltiple respuesta. 81
Figura 5.7. Modo solicitud-sin respuesta. 81
Figura 5.8. Solicitudes LiDAR. 82
Figura 5.9. Trama de stop. 82
Figura 5.10. Datos de la trama get-health. 83
Figura 5.11. Descripción de la trama get-health. 83
Figura 5.12. Descripción de la trama get_samplereate. 84
Figura 5.13. Datos de la trama get_samplerate. 84
Figura 5.14. Descripción de la trama get-scan. 85
Figura 5.15. Diagrama sobre la comunicación con el LiDAR. 86
Figura 5.16. Diagrama de la etapa de comunicación con el LiDAR. 91
Figura 5.17. Diagrama de la etapa de adquisición y procesamiento. 92
Figura 5.18. Datos de una medida recibida del LiDAR. 93
Figura 5.19. Sistema de coordenadas esféricas. 98
Figura 5.20. Formato trama de envío por USB. 99
Figura 5.21. Ejemplo de trama de envío por USB. 100
Figura 5.22. Diagrama de la etapa de fin de operación. 101
Figura 6.1. Entorno de programación en Processing 104
Figura 6.2. Diagrama de flujo de la aplicación de procesamiento. 106
Figura 6.3. Diagrama de flujo de la etapa de inicialización. 109
Figura 6.4. Interfaz gráfica de la aplicación. 111
Figura 6.5. Pantalla inicial aplicación. 113
Figura 6.6. Diagrama de la etapa de adquisición y procesamiento. 115
Figura 6.7. Sistema de coordenadas esféricas/cartesianas. 118
Figura 6.8. Diagrama de la etapa de representación. 119
Figura 6.9. Sistema de coordenadas cartesianas de Processing frente al tradicional. 121
Figura 6.10. Rotación sobre el eje X de +90º. 122
Figura 6.11. Rotación sobre el eje Y de +90º. 122
Figura 7.1. Comparativa de medidas LiDAR vs telémetro PLR-50 124
Figura 7.2. Representación tridimensional frente a imagen real de una sala mediana. 125
Figura 7.3. Representación tridimensional frente a imagen real de una sala pequeña. 126
Figura 7.4. Representación tridimensional frente a imagen real de un pasillo. 127
Anexo 1. Figura 1. Plano de la base inferior 141
Anexo 1. Figura 2. Plano de la base superior. 142
Anexo 1. Figura 3. Plano del soporte del stepper. 143

INDICE de FIGURAS

Anexo 1. Figura 4. Plano del soporte del servo. 144
Anexo 1. Figura 5. Plano de la base giratoria. 145
Anexo 1. Figura 6. Plano de la base del LiDAR. 146
Anexo 1. Figura 7. Plano del contrapeso. 147

INDICE de FIGURAS

INDICE de TABLAS

Tabla 2.1. Parámetros característicos del RPLIDAR A1 34
Tabla 2.2. Parámetros del láser..... 35
Tabla 2.3. Parámetros interfaz de comunicación..... 36
Tabla 2.4. Pines del A4988..... 39
Tabla 6.1. Tramas de comunicación USB (start y stop)……..114
Tabla 7.1. Cálculo de errores en las medidas del LiDAR……123
Presupuesto. Tabla 1. Costes de recursos hardware……132
Presupuesto. Tabla 2. Coste de recursos software..... 133
Presupuesto. Tabla 3. Trabajo tarifado por tiempo empleado..... 133
Presupuesto. Tabla 4. Coste de materiales fungibles..... 134
Presupuesto. Tabla 5. Costes totales del TFM. 134

INDICE de TABLAS

INDICE de CODIGOS

Código 2.1. GPIO_InitTypeDef. 27
Código 2.2. Funciones de lectura de pines. 28
Código 2.3. Funciones de escritura y reset. 28
Código 2.4. USART_InitTypeDef. 29
Código 2.5. Funciones de configuración de una USART. 29
Código 2.6. Función GetChar. 30
Código 2.7. Función PutChar. 31
Código 3.1. Función stepperConf 54
Código 3.2. Ejemplos de configuración del stepper. 54
Código 3.3. Función timerPWM. 56
Código 3.4. Función PWM_Init. 57
Código 5.1. Ejemplo de configuración de la GPIO 70
Código 5.2. Configuración de la USART. 70
Código 5.3. Ejemplo de configuración del timer. 71
Código 5.4. Ejemplo de configuración del NVIC. 72
Código 5.5. Ejemplo de rutina de servicio. 72
Código 5.6. Configuración RCC. 73
Código 5.7. Programa principal. 77
Código 5.8. Función lecturaUSB. 79
Código 5.9. Interrupción USART de transmisión. 88
Código 5.10. Interrupción USART de recepción. 88
Código 5.11. Ejemplo de secuencia de transmisión. 89
Código 5.12. Función reSend. 90
Código 5.13. Función capturaDatos. 94
Código 5.14. Función inicialización. 95
Código 5.15. Función calidadAzimut. 96
Código 5.16. Función cAnguloCompl. 97
Código 5.17. Función distancia. 98
Código 5.18. Función resetServo. 101
Código 6.1. Función setup 109
Código 6.2. Función controlEvent. 112
Código 6.3.Función gui (interfaz gráfica). 113
Código 6.4. Función serial. 115
Código 6.5. Función lecturaArchivo. 116
Código 6.6. Función adaptDatos. 117
Código 6.7. Función draw. 120
Anexo 2. Código 1. Función secuenciaComienzo 150
Anexo 2. Código 2. Función verificMove. 151
Anexo 2. Código 3. Función envíoUSB. 153
Anexo 3. Código 1. Función botones 155
Anexo 3. Código 2. Función consola. 155
Anexo 3. Código 3. Función leyendaColores. 156

Anexo 3. Código 4. Colores. 156
Anexo 3. Código 5. Función ONOFF. 157
Anexo 3. Código 6. Función visible. 157
Anexo 3. Código 7. Función cuadrosTexto 158
Anexo 3. Código 8. Función accionTexto 158
Anexo 3. Código 9. Función colorPunto. 159

Acrónimos

3D. Tres dimensiones. *(Three Dimensions)*.

ADC. Conversor analógico-digital. *(Analog-Digital Converter)*.

AHB. Bus de alto rendimiento AMBA. *(AMBA High-performance Bus)*.

ALU. Unidad aritmética lógica. *(Arithmetic Logic Unit)*.

AM. Modulación en amplitud. *(Amplitude Modulation)*.

AMBA. Arquitectura bus de microcontrolador avanzada. *(Advanced Microcontroller Bus Architecture)*.

APB. Bus de periféricos avanzados. *(Advanced Peripheral Bus)*.

ARM. Máquina avanzada RISC. *(Advanced RISC Machine)*.

ASCII. Código estándar estadounidense para el intercambio de información. *(American Standard Code for Information Interchange)*.

CAD. Diseño asistido por ordenador. *(Computer-Aided Design)*.

CMOS. Semiconductor complementario de óxido metálico. *(Complementary Metal-Oxide-Semiconductor)*.

CPU. Unidad central de procesamiento. *(Central Processing Unit)*.

DMA. Acceso directo a memoria. *(Direct Memory Access)*.

DPM. Modelo dato-punto. *(Data-Point Model)*.

DSP. Procesador digital de señales. *(Digital Signal Processor)*.

FIFO. Primero en entrar, primero en salir. *(First In, First Out)*.

FM. Modulación en frecuencia. *(Frequency Modulation)*.

GND. Toma de tierra. *(Ground)*.

GPIO. Entrada/salida de propósito general. *(General Purpose Input/Output)*.

GPS. Sistema de posicionamiento global. *(Global Positioning System)*.

GUI. Interfaz gráfica de usuario. *(Graphical User Interface)*.

HAL. Capa de abstracción hardware. *(Hardware Abstraction Layer)*.

INS. Sistema de navegación inercial. *(Inertial Navigation System)*.

IRQ. Solicitud de interrupción. *(Interrupt Request)*.

ISR. Rutina de servicio de la interrupción. *(Interruption Service Routine)*.

Acrónimos

LiDAR. Detección y medición de distancias por luz. *(Light Detection and Ranging).*

MCU. Unidad del microcontrolador. *(Microcontroller Unit).*

NVIC. Controlador de interrupciones vectorizadas anidadas. *(Nested Vectored Interrupt Controller).*

P3D. Processing tres dimensiones. *(Processing 3D).*

PDF. Formato de documento portátil. *(Portable Document Format).*

PLA. Ácido poliláctico. *(Polylactic acid).*

PLL. Bucles de enganche de fase. *(Phase-Locked Loop).*

PWM. Modulación por ancho de pulsos. *(Pulse-Width Modulation).*

RADAR. Detección y medición de distancias por radio. *(Radio Detection and Ranging).*

RAM. Memoria de acceso aleatorio. *(Random Access Memory).*

RCC. Control de reloj y reset. *(Reset and Clock Control).*

RISC. Ordenador con conjunto reducido de instrucciones. *(Reduced Instruction Set Computer).*

RX. Recepción *(Reception).*

RXE. RX fin. *(RX End).*

SD. Seguridad digital. *(Secure Digital).*

SNR. Relación de señal a ruido. *(Signal to Noise Ratio).*

SPI. Interfaz periférica serie. *(Serial Peripheral Interface).*

SRAM. Memoria estática de acceso aleatorio. *(Static Random Access Memory).*

SYSTICK. Temporizador del sistema. *(System Timer).*

TDC. Conversor de tiempo a señal digital. *(Time to Digital Converter).*

TFM. Trabajo Fin de Máster. *(Final Master Project).*

TTL. Lógica transistor a transistor. *(Transistor-Transistor Logic).*

TX. Transmisión *(Transmission).*

TXE. TX fin. *(TX End).*

URL. Identificador de recursos uniforme. *(Uniform Resource Locator).*

USART. Transmisor-receptor asíncrono universal. *(Universal Asynchronous Receiver-Transmitter).*

USB. Bus universal en serie. *(Universal Serial Bus).*

VCC. Tensión de corriente continua. *(Voltage Continue Current).*

VDD. Tensión entre drenador y puerta. *(Voltage between Drain and Door).*

1 Introducción

1.1 Antecedentes

En la actualidad, el empleo de sistemas *LiDAR* (Light Detection and Ranging), es decir, sistemas que realizan la detección y medición de distancias mediante el uso de la luz se ha incrementado en diversos campos como la gestión de incendios forestales o en la búsqueda de combustibles fósiles. Según los estudios más recientes a nivel mundial la venta de estos sistemas va a incrementarse ininterrumpidamente; se va a pasar de 234,3 millones de euros en 2014 a 824 millones de euros en 2022 [1]. Sin duda, la capacidad de poder detectar el movimiento de una persona, animal, objetos u obstáculos tiene especial importancia gracias al empleo de esta tecnología. Uno de los ejemplos más conocidos y, novedosos, donde esto se hace fundamental es en los nuevos coches autónomos [2]–[4] debido a que el sistema *LiDAR* permite tener una monitorización completa de la región de interés, alrededor del vehículo, en tiempo real. En el mercado, los modelos más reconocidos de vehículos autónomos son tanto el *Tesla* como el *Google self-driving car*. De igual forma, se están analizando sistemas para ayudar a personas con movilidad reducida [5] o para vehículos de gran tamaño [6] para facilitar al conductor su manejo**.**

Un ámbito donde la presencia del *LiDAR* ha incrementado su importancia es en la cartografía terrestre ya que permite crear imágenes reales y detalladas sin necesidad de realizar grandes inversiones económicas pudiendo usarse para labores topográficas, reconstrucciones, entre otras [7]–[9]. De esta forma, se consigue obtener representaciones tridimensionales de un espacio determinado ya sea de tipo urbano o rural. A partir de esta representación tridimensional del entorno se permite nuevas utilidades como, por ejemplo, en el empleo de estos en excavaciones arqueológicas dónde el acceso de un humano no sea recomendable [10], [11] y en otros campos de la robótica [12].

En el mercado ya existen sistemas LiDAR que permiten caracterizar el entorno en tres dimensiones, sin embargo, lo más habitual es que solo trabaje en dos dimensiones. Para poder conseguir tener datos tridimensionales existen varias opciones como: las de realizar reflexiones del haz gracias a un espejo o mediante el uso de motores que permitan mover el sistema en horizontal o vertical para tomar medidas tridimensionales. Es evidente que con los datos obtenidos directamente del sistema no se puede obtener una representación inmediata y precisa de la región de interés. De esta forma, se debe realizar un post-procesado de los datos obtenidos mediante algún procedimiento algorítmico según sea la aplicación que se desee dar. Por ejemplo, para la representación digital tridimensional de espacios, es decir, del escaneo de una determinada región de interés que cubra el LiDAR es normal emplear técnicas como el modelo dato-punto (DPM) [6] o el algoritmo Ramer-Douglas-Peucker [2].

Por último, una de las características más importantes que debe tener este tipo de sistemas es poder representar al usuario la superficie escaneada de la forma más clara y sencilla posible. Para ello, se puede usar distintas plataformas según las necesidades del sistema desde una representación en un teléfono móvil o en una aplicación de ordenador.

1.2 Objetivos

El principal objetivo de este TFM es el desarrollo de un sistema electrónico que permita la visualización tridimensional de regiones de interés que van desde una habitación hasta un paisaje urbano mediante el empleo de tecnología *LiDAR*, el posterior procesamiento de los datos obtenidos y, de acuerdo a ello, su correspondiente representación 3D en los dispositivos de visualización de referencia.

Desde un punto de vista tecnológico este proyecto se basa en la integración electrónica de un sistema de medición de distancias a obstáculos mediante tecnología láser *(LiDAR*) junto con un sistema electromecánico que permita realizar escaneados de 360º. Para el manejo del mismo es necesario un sistema de control mediante un microcontrolador ARM Cortex-M junto con el entorno de procesamiento software adecuado para poder adquirir y tratar los datos. También es necesario desarrollar el entorno de representación de dichos

datos en el dispositivo de visualización que se seleccione. En base a esto, se pretende llevar a cabo los siguientes objetivos:

1. Conocer las ventajas, limitaciones y características principales de los sistemas *LiDAR*.
2. Identificar el sistema electromecánico idóneo que permita efectuar medidas espaciales en tres dimensiones.
3. Disponer de un sistema de almacenamiento de datos que permita adquirir y almacenar datos fiables en tiempo real mediante un sistema de comunicación adecuado.
4. Elegir el entorno de procesamiento más adecuado en función de los datos disponibles.
5. Seleccionar el tipo de procesamiento que se quiere implementar teniendo en cuenta las limitaciones temporales.
6. Desarrollar la interfaz gráfica que permita la representación real del entorno donde se realicen las medidas.
7. Integrar las distintas partes que forman el sistema y realizar las verificaciones, mediante pruebas lo más realistas posibles.

1.3 Contenido de la memoria

A continuación, se describe brevemente cómo se divide el presente Trabajo Fin de Máster y los temas que se tratan en cada uno de los capítulos que lo componen. De esta manera, se pretende dar una visión general de los contenidos tratados en este TFM:

- **Lista de acrónimos.** En este apartado se incluye una descripción completa de los acrónimos usados en este TFM.
- **Capítulo 1. Introducción.** En este capítulo se realiza una pequeña introducción a los sistemas de medida láseres. De todos ellos el caracterizado en este TFM es el *LiDAR*, del cual se realiza una descripción básica explicando sus características

principales, sus limitaciones y principio de funcionamiento. Por último, se indica la solución adoptada al problema planteado.

- **Capítulo 2. Descripción del Hardware.** Este segundo capítulo contiene la descripción de los elementos hardware empleados para el desarrollo de la solución adoptada.
- **Capítulo 3. Diseño del sistema electromecánico.** En este apartado se especifican las piezas realizadas para la obtención de las medidas de acuerdo con la solución adoptada. Además, se incluye un ensamble del conjunto de las piezas desarrolladas. Por otra parte, también se describe el firmware empleado para gestionar los elementos hardware del sistema.
- **Capítulo 4. Integración electrónica.** En este apartado se especifica las conexiones hechas para la integración de las distintas etapas que conforman el sistema completo.
- **Capítulo 5. Desarrollo firmware.** Se dedica este capítulo a describir el software realizado para la etapa de adquisición y preprocesamiento, comentando sus características principales, parámetros de configuración y funciones destacadas.
- **Capítulo 6. Desarrollo de la aplicación de representación.** En este capítulo se describe la aplicación de visualización diseñada como las librerías que se han empleado en ello.
- **Capítulo 7. Resultados y Revisión de Objetivos.** En este apartado se muestran y analizan los resultados obtenidos tras la realización del TFM y se procede a establecer una revisión de los objetivos previamente establecidos.
- **Capítulo 8. Conclusiones y líneas de trabajo futuras.** En este último capítulo de la memoria, se presentan las conclusiones generales del TFM, además de posibles líneas de investigación futuras.
- **Bibliografía.** En esta sección se incluyen todas las referencias bibliográficas usadas a la hora de realizar este TFM.
- **Presupuesto.** En este apartado se establece el precio total de este Trabajo Fin de Máster indicando los precios de los materiales, herramientas, costes de ingeniería, impuestos, etc.

- **Anexos.** En esta sección se incluye información adicional del TFM que, siendo relevante, no tiene cabida en los distintos capítulos de la memoria.

1.4 LiDAR

1.4.1 Introducción

LiDAR (Light Detection And Ranging) es una tecnología óptica de teledetección que puede medir la distancia a objetos lejanos mediante las propiedades del espectro electromagnético usando pulsos de un láser [13]. Los datos capturados pueden ser usados, a posteriori, para realizar reconstrucciones digitales, planos bidimensionales o modelos tridimensionales de un entorno u objeto real en una gran variedad de aplicaciones.

La ventaja del escaneado láser es el hecho de que puede tomar una gran cantidad de puntos con alta precisión en un periodo de tiempo relativamente corto, es como tomar una fotografía con información de profundidad. Además, presenta ventajas respecto a otros sistemas de teledetección como el *RADAR* ya que es más fácil de manejar, transportar y mantener aparte de que, por sí mismo, es más caro que un *LiDAR*[14].

Existen dispositivos ya sean terrestres o aerotransportados que disponen de un láser que mide ángulos, distancias e intensidad de los puntos iluminados, de manera sistemática a una frecuencia elevada y en tiempos reales. El resultado es una nube de puntos 3D que representa el modelo escaneado. La técnica se caracteriza por una alta repetitividad de la medida con precisión y exactitud, haciendo posible la representación fidedigna de la región de interés.

Debido a que puede trabajar en muchos entornos y gracias a su robustez los sistemas *LiDAR* se utilizan actualmente para un número muy amplio de aplicaciones [15]:

- Industria y energía: Análisis de la posible inclusión de nuevas líneas de producción, inspección de líneas eléctricas, red ferroviaria, monitorización y automatización de procesos, etc.
- Ingeniería civil, infraestructuras, minería: Permite una inspección periódica de diferentes infraestructuras para poder determinar su estado.
- Monitorización de espacios: Caracterización de áreas de interés tanto de recintos cerrados como abiertos para análisis o seguridad entre otras aplicaciones.
- Arqueología, patrimonio: Para la representación tridimensional de forma rápida y precisa de cualquier elemento cultural, el láser escáner es la herramienta más potente y fiable.
- Monitorización y estudios medioambientales: Análisis de densidad arbórea, medida de perfiles de aerosoles y nubes, ganadería profesional.

En un primer momento, los escáneres láser eran de corto alcance y se utilizaban principalmente en el diseño automatizado e industrial para facilitar el diseño CAD. Esto ayudó a la producción en masa de productos de consumo. Sin embargo, otros campos de aplicación han sido explotados como consecuencia de la constante evolución tecnológica tal y como se observa en Figura 1.1. Los escáneres láser de medio alcance fueron desarrollados para la industria petroquímica [16]. Otras disciplinas como el patrimonio cultural, la arquitectura, el desarrollo urbanístico, la medicina forense y la industria del entretenimiento están empezando a adoptar esta tecnología.

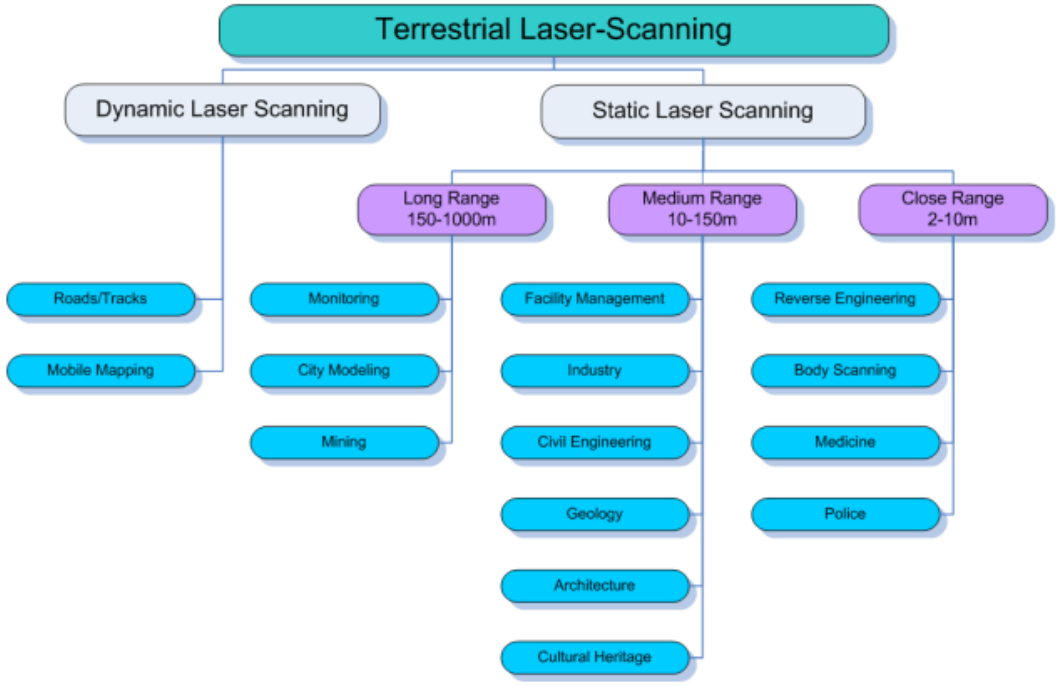


Figura 1.1. Tipos de LiDAR en función del rango de detección.

1.4.2 Principios y fundamentos de la tecnología láser

El primer láser operativo fue publicado en mayo de 1960 por Theodore Maiman en los laboratorios Hughes Research [14]. Un láser normal emite luz en un estrecho y poco divergente haz de longitud de onda limitada (correspondiente a un color particular si el láser si trabaja en el espectro visible) [13]. Sin embargo, esto contrasta con otras fuentes como la bombilla incandescente, que emite en un amplio espectro de longitudes de onda dando lugar al color blanco que incluye todo el espectro visible. La luz tiene algunas propiedades muy especiales que la difieren de otro tipo de luces [17]:

- Luz láser se genera en forma de haz: esto permite su propagación en una dirección bien definida (coherencia) siendo el haz moderadamente divergente. El término coherencia hace referencia a que la señal eléctrica generada presenta una relación de fase constante en distintos puntos a lo largo del haz. Esta propiedad es la razón por la que un haz láser en determinados casos puede propagarse a largas distancias.
- Longitud de onda larga: esto implica una relación fija de fase a través de intervalos de tiempo relativamente largos, correspondientes a amplias distancias de propagación [18].

- No continuidad: esta luz no siempre es continua pudiendo ser emitida en forma de pulsos cortos o ultra cortos. Como consecuencia de esto, los máximos de potencia pueden ser extremadamente altos. De tal forma que los haces permanecen enfocados cuando se proyectan sobre un escenario lejano gracias a las propiedades de coherencia.
- Velocidad de propagación: el haz viaja con una velocidad finita y constante en un determinado medio. Como consecuencia de estas propiedades, la luz láser es muy adecuada para la medición de objetos.

El ancho de banda espectral es estrecho combinando las características descritas anteriormente. Esto significa que los haces de láser visible tienen un cierto color puro, por ejemplo, rojo, verde o azul, pero no blanco o magenta. La mayoría de los láseres usados en mediciones cortas y medias tienen una longitud de onda entre 400 nm (láser azul) y 800 nm (láser rojo), es decir, en el espectro visible tal y como se observa en la Figura 1.2 [13]. En la mayoría de casos, la luz láser está polarizada linealmente. Esto significa que el campo eléctrico oscila en una dirección espacial particular

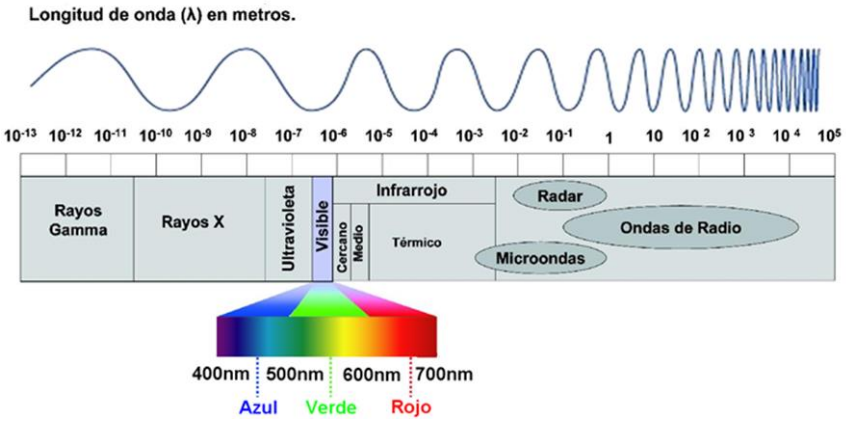


Figura 1.2. Espectro electromagnético.

1.4.3 Tipos de LiDAR

Los *LiDAR* actuales pueden dividirse en dos categorías, estático y dinámico. Cuando el escáner se mantiene en una posición fija durante la toma de datos, se llama *LiDAR* estático. Las ventajas de este método son la alta precisión y la relativa alta densidad de puntos [19].

El láser escáner estático suele ser el método más extendido a la hora de realizar escaneados terrestres.

En los casos de *LiDAR* dinámico, el dispositivo se encuentra montado en una plataforma móvil. Estos sistemas requieren otros sistemas de posicionamiento adicionales tales como *INS* o *GPS*, lo que hace que el sistema completo sea más complejo y caro [18]. Este tipo de escáner laser se encuentran en aviones, sobre vehículos o plataformas aéreas no tripuladas.

1.4.4 Formas de medir distancias con la luz

A partir de los recientes desarrollos en la tecnología de sensores, la luz se ha utilizado de varias maneras para medir objetos. Estas técnicas de medición se pueden dividir en dos categorías: técnicas activas y pasivas.

Las técnicas pasivas no emiten radiación alguna, pero se basan en detectar la radiación ambiental reflejada por los objetos. La mayoría de sistemas de este tipo detectan la luz visible reflejada en los objetos. Los métodos pasivos pueden ser útiles porque no se necesita más hardware que una cámara digital [14], sin embargo, se necesita una fuente de luz que ilumine la zona. El problema con estas técnicas es que necesitan encontrar correspondencias entre imágenes 2D, lo que no siempre tiene una única solución.

Por el contrario, los escáneres activos (*LiDAR*) emiten una radiación láser (espectro visible) controlada y detectan su reflexión con el fin de sondear un objeto o un entorno. Los posibles tipos de radiación utilizados incluyendo la luz visible entre otros. Como estas técnicas de medición activa requieren un transmisor láser y un receptor óptico son mecánicamente más complejos que las técnicas pasivas [13]. Las principales ventajas de estos sistemas son:

- No requieren luz ambiental, porque ellos generan su propia radiación.
- Proporcionan gran cantidad de mediciones de manera automática.
- Capturan mucha información en poco tiempo (100-500.000 ptos/s).

Sin embargo, algunos sistemas activos pueden verse afectados por fuentes de luz externas, reflectividad, color y rugosidad. Existe tres tipos de *LiDAR* activos, dependiendo de la manera en la que el escáner recibe y/o analiza la señal de radiación reflejada.

1.4.4.1 Triangulación

Los triángulos son la base de muchas técnicas de medición. Los fundamentos matemáticos de la trigonometría [20] son utilizados por estos tipos de *LiDAR* para realizar las medidas como se observa en la Figura 1.3. Se dirige un patrón láser sobre el objeto y se emplea una fotosensor para recoger la señal reflejada del objeto [21]. El emisor láser y el dispositivo óptico se instalan con un ángulo constante, creando un triángulo entre ellos y la proyección del láser sobre el objeto. Debido a esta configuración, la proyección del láser cambia el campo de visión del fotosensor en función de la distancia al objeto.

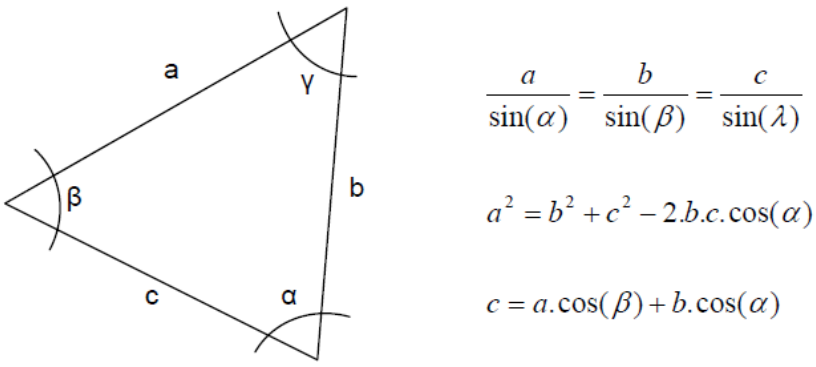


Figura 1.3. Principios de la triangulación.

Analizando la Figura 1.4 se observa que el lado D del triángulo es conocido, es decir, la distancia entre el fotosensor y el emisor láser. El ángulo del emisor láser α, también es conocido. El ángulo de la cámara β puede ser determinado encontrando la localización del haz láser en el fotosensor. Estos tres elementos determinan completamente la forma y el tamaño del triángulo y proporcionan la profundidad exacta hacia el objeto medido que es el objetivo final [22]. Se puede demostrar que, cuanto mayor es el lado D (base), menor es el error en el cálculo de la profundidad del objeto. A pesar de ello, dicha base no puede ser muy grande porque entonces el emisor láser y el dispositivo óptico tendrían un solape en el campo de visión muy reducido y la proyección del láser no siempre podría ser capturada.

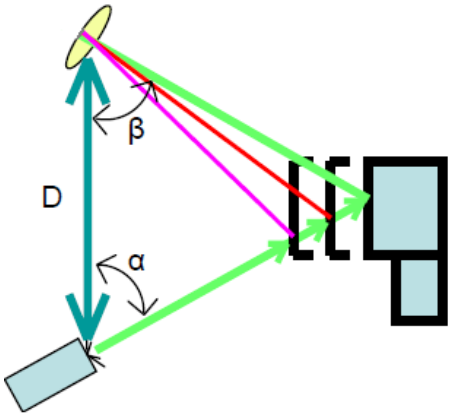


Figura 1.4. Triangulación en LiDAR.

A causa de las limitaciones físicas de usar una base mayor y un campo de visión el fotosensor limitado, los escáneres por triangulación se usan en aplicaciones que generalmente requieren una distancia de uso menor de 10 metros pudiendo tener grandes resoluciones del orden del centímetro o el milímetro dependiendo del modelo [21]. En la Figura 1.5 se pueden observar algunos modelos de *LiDAR* que presentan estas características.

Figura 1.5. Ejemplos de LiDAR de triangulación.

1.4.4.2 Medición basada en pulsos

Los *LiDAR* basados en pulsos son escáneres activos que miden un intervalo de tiempo entre dos eventos (la emisión láser y la recepción de la señal reflejada) ya que las ondas de luz viajan con una velocidad finita y constante a través de un medio [23]. Por consiguiente, cuando puede medirse la demora durante el cual la luz viaja de una fuente a un objeto reflectante y regresa a dicha fuente se puede obtener la distancia al objeto tal y como se observa en la Figura 1.6.

Los escáneres por tiempo no suelen emplear haces continuos, sino pulsos de láser. De esta forma. escanean todo su campo de visión punto a punto cambiando la dirección del haz mediante una unidad de desviación. Hay que destacar que para que no se produzcan mediciones ambiguas, el tiempo medido debe ser mayor que la amplitud del pulso.

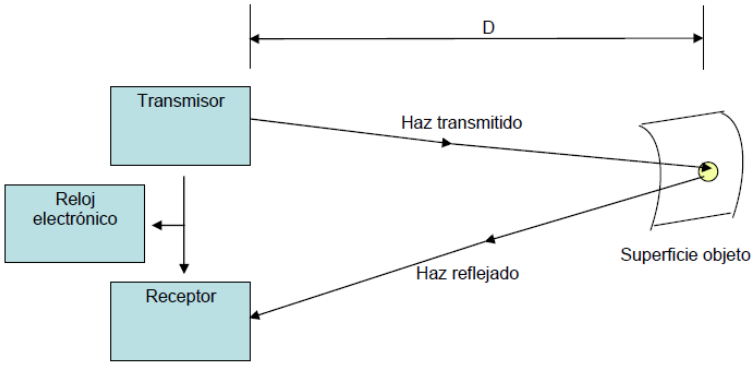


Figura 1.6. Medición basada en pulsos.

En un sistema de medición por tiempo, la máxima frecuencia del pulso está determinada por el hecho de que el emisor no puede enviar un nuevo pulso hasta que no haya recibido el anterior. La finalidad de esta restricción es evitar la confusión en la llegada de los pulsos, lo que recibe el nombre de máximo rango de certeza por lo que se necesitan contadores de alta precisión (TDC) para determinar el tiempo [24].

La principal ventaja de usar un sistema de pulsos para las mediciones de distancia es la alta concentración de la energía transmitida. Esta energía hace posible que se alcance el nivel de relación señal-ruido (SNR) necesario para medidas de precisión en largas distancias (varios cientos de metros). Por el contrario, presenta un inconveniente que es el problema de detectar la llegada exacta de los retornos del pulso debido a la atenuación atmosférica.

En la Figura 1.7 se muestran algunos escáneres basados en el tiempo de vuelo existentes en el mercado como, por ejemplo, Optech o Callidus.

Figura 1.7. Ejemplos de LiDAR basados en pulsos.

1.4.4.3 Medición basada en fases

Otro principio de medición basado en tiempo que evita el uso de contadores de alta precisión es la modulación de la potencia del haz láser. La luz emitida (incoherente) se modula en amplitud y se envía hacia un objeto. La reflexión de dicha señal se captura en el receptor y un circuito mide la diferencia de fase entre las ondas enviada y recibida tal y como se observa en la Figura 1.8. Se puede usar distintos tipos de modulaciones como: modulaciones sinusoidales, amplitud modulada (AM), frecuencia modulada (FM), pseudo ruido o modulación polarizada [25].

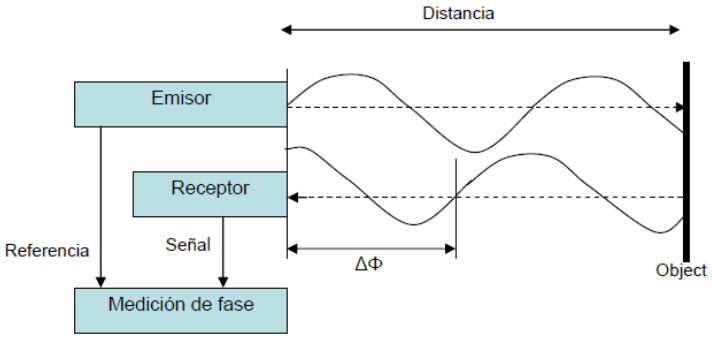


Figura 1.8. Medición basada en fases.

Los escáneres con modulación continua del haz también tienen un máximo rango de certeza, similar al de los sistemas de medición por tiempo. Para estos sistemas el alcance se haya limitado al que corresponde a un retardo de fase de un ciclo de la onda emitida.

Para evitar el inconveniente de la ambigüedad en la medición, se pueden utilizar varias frecuencias de manera que con las frecuencias más bajas (mayor longitud de onda) se localiza el punto a medir y con las altas frecuencias se mida la distancia con precisión [26]. En los últimos modelos se superponen 2 o incluso 3 longitudes de onda. En la Figura 1.9 se muestran algunos escáneres basados en la fase como, por ejemplo, IQSun o Z+F.

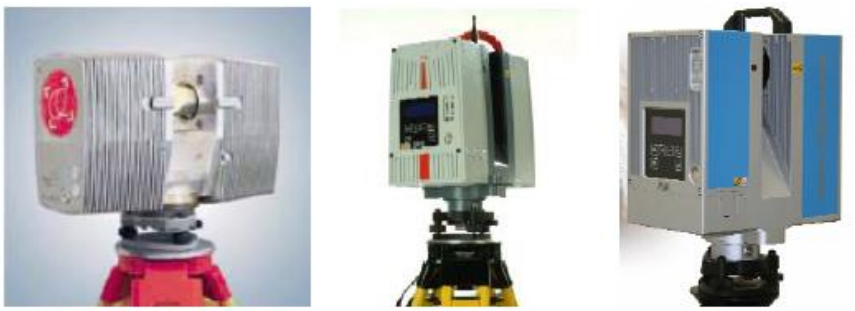


Figura 1.9. Ejemplos de LiDAR basados en fases.

1.4.5 Errores en las medidas LiDAR

Las empresas que producen dispositivos *LiDAR* publican las precisiones de sus equipos para ilustrar las ventajas de su producto. Sin embargo, la experiencia demuestra qua algunas veces esto no se debería tomar como valor real y que la precisión de los instrumentos varía de uno a otro depende de la calibración individual. Muchas veces, la nube de puntos (distancias medidas) contiene un considerable número de puntos que presentan grandes errores [27]. La precisión de estas medidas no se puede garantizar de la misma manera que con instrumentos topográficos convencionales. Se han realizado diversos estudios que dividen estos problemas en tres categorías principales: errores instrumentales, errores relacionados con el objeto, errores por el entorno.

Los errores instrumentales pueden ser sistemáticos o aleatorios y se deben al diseño del sistema. Los errores aleatorios afectan principalmente a la precisión de la medida y la localización del ángulo [28]. Por otra parte, los errores sistemáticos pueden ser generados por la variación de temperatura en la electrónica de medición del tiempo provocando problemas en la medida de la distancia

La divergencia del haz es la anchura que alcanza el haz con la distancia recorrida. La divergencia del haz tiene una gran influencia con la resolución. Una de las consecuencias más importantes de la divergencia del haz es el problema del borde partido. Cuando un haz incide contra un borde de un objeto, éste se divide en dos. Un parte se refleja mientras que la otra parte va más lejos hasta que alcanza otra superficie. El resultado de ello es que la información que llega al receptor llega desde dos puntos diferentes produciendo una distorsión en la medida.

Como los escáneres miden la reflexión del haz láser sobre la superficie, se debe tener en cuenta la reflexión y las propiedades ópticas de los materiales. La reflexión de luz monocromática normalmente muestra rayos reflejados en muchas direcciones. Este tipo de reflexión isotrópica (difusa) tal y como se observa en la Figura 1.10 se conoce como la ley del coseno de Lambert [29].

Esta ley implica que el haz láser está afectado por la absorción de la señal que se transporta por el aire, la reflexión del material sobre el que se mide y el ángulo de incidencia entre la superficie medida y el haz láser. Esto significa que para superficies muy oscuras (negras) que absorben la mayor parte del espectro, la señal reflejada será muy débil e incluso nula, por lo que la precisión de la medida se hallará afectada por el ruido. Superficies con alta reflectividad (superficies brillantes, entre otras) dan unas mediciones más fiables y precisas. Sin embargo, si la reflectividad del objeto es demasiado elevada, por ejemplo, en superficies metálicas el haz láser es desviado totalmente y chocará en otras superficies o se propagará por el aire. Esta desviación provoca que el punto medido no sea el punto al que apunta el láser, sino otro o ninguno. Este tipo de ruido se le denomina ruido speckle [30].

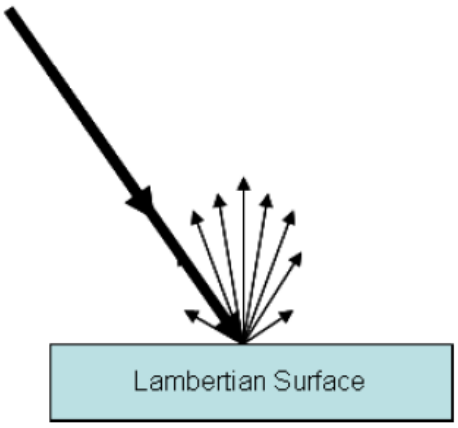


Figura 1.10. Reflexión sobre una superficie Lambertiana.

El registro de superficies con reflectividad diferente también ocasiona errores sistemáticos en la distancia. Al igual que las propiedades de reflexión de la superficie, las propiedades del color también afectan a la precisión. Se ha observado que existen diferencias significativas en la distancia que, en términos generales, se pueden relacionar con el color de cada superficie en relación y la longitud de onda del láser utilizado [29].

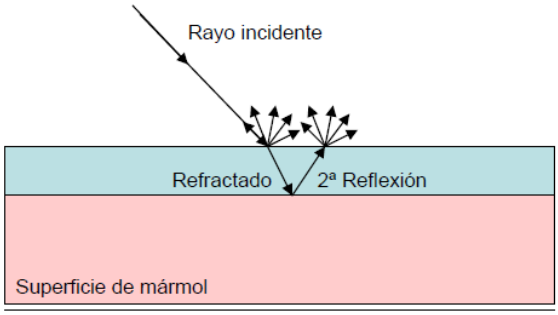


Figura 1.11. Efectos de la refracción en materiales semitransparentes.

Aparte de los efectos de reflectividad, existen ciertos materiales que presentan una capa semitransparente que permite pasar el haz láser refractándose y reflejándose en el propio material, como, por ejemplo, la madera o el mármol tal y como se observa en la Figura 1.11.

Otro de los errores que comúnmente afectan a los sistemas *LiDAR* son las condiciones ambientales donde se realizan las mediciones, tales como, la temperatura, la atmósfera, la interferencia de radiación y la dispersión por movimiento.

Por una parte, la temperatura dentro del sistema puede ser bastante más alta que la temperatura de la atmósfera alrededor del equipo debido al calor interno o al calor resultante de la radiación externa (por ejemplo, el sol) [31]. No solo la temperatura del equipo es importante, sino también la temperatura del área a analizar. Cuando se escanea algo que se encuentra a altas temperaturas la radiación causada por las superficies calientes reduce la relación señal ruido y, por tanto, la precisión de las mediciones.

Los *LiDAR* solo funcionan adecuadamente cuando se utilizan dentro de un cierto rango de temperatura. Incluso dentro de este rango se pueden observar desviaciones en las distancias [27]. Los errores naturales provienen principalmente de las variaciones atmosféricas de temperatura, presión y humedad, lo que afecta al índice de refracción y modifica la longitud de onda electromagnética. Esto significa que la velocidad de la luz del láser depende de la densidad del aire.

Como estos sistemas trabajan en una banda de frecuencia muy estrecha, la precisión en la distancia puede estar influenciada por una radiación externa, como por ejemplo las fuentes fuertes (luz fluorescente, por ejemplo) de iluminación externa.

1.4.6 Normativa de seguridad en el manejo de láseres

Para permitir determinar el riesgo potencial, todos los láseres e instrumentos que hacen uso de éste están etiquetados y clasificados dependiendo de la longitud de onda y de la potencia de energía que produce el láser ya que puede tener riesgos para la salud ocular de las personas que se encuentren dentro de su rango de acción. El estándar europeo IEC 60825-1 proporciona información sobre las distintas clases y precauciones asociadas. Se describen siete clases de láser de las que se explicarán las más importantes [32]:

- Clase 1. Son seguros en condiciones de utilización razonablemente previsibles, incluyendo el uso de instrumentos ópticos para visión intrahaz.
- Clase 2. Láseres que normalmente producen un reflejo ciego para proteger al ojo. Esta reacción puede proporcionar la adecuada protección en condiciones de utilización razonablemente previsibles.

- Clase 3B. Normalmente son peligrosos si ocurre exposición directa intrahaz, aunque la visión de reflexiones difusas es normalmente segura. Generalmente, esta clase de láser no es adecuada en aplicaciones de campo.
- Clase 4. Si se observan directamente causarán daños en los ojos o en la piel. Los láseres de esta clase también pueden producir reflexiones peligrosas. Esta clase de láser no es adecuada en aplicaciones de campo.

Las precauciones más relevantes que incluye esta norma IEC para los escaneados láser son [32]:

- Sólo el personal cualificado y preparado debe ser asignado para instalar, ajustar y utilizar el equipamiento láser.
- Las áreas donde se utiliza estos láseres deberían estar señalizadas con la señal de advertencia apropiada.
- Tomar las precauciones necesarias (uso de gafas de protección láser) que permitan no mirar el haz láser directamente.

1.5 Solución adoptada

El objetivo de este Trabajo Fin de Master es diseñar e implementar un escáner tridimensional para entornos cerrados basados en *LiDAR*. Además, se realizará una aplicación en ordenador que permita visualizar la representación de dichos entornos. En la Figura 1.12 se observa un diagrama de bloques que representa el sistema al completo. En base a esto se estiman necesarios los siguientes componentes: escáner *LiDAR*, motor tipo servo y stepper junto con sus drivers si los hubiera, sistema electro mecánico, sensores de posición, tarjeta de desarrollo con el microcontrolador adecuado y comunicación USB.

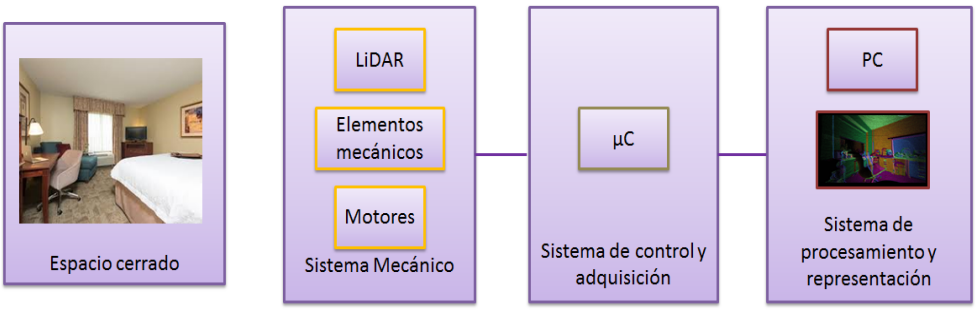


Figura 1.12. Diagrama de bloques de la solución adoptada.

El escáner *LiDAR* seleccionado es el RPLIDAR A1 de la marca Slamtec ya que presenta las condiciones adecuadas para caracterizar espacios cerrados en base a sus especificaciones y lo visto en la parte introductoria de este capítulo [33]. Por otra parte, al disponer de un motor incorporado permite por sí mismo obtener medidas en dos dimensiones. Teniendo en cuenta las limitaciones económicas, se dispone de un escáner que compensa precio y prestaciones técnicas.

Para poder realizar el escaneado en tres dimensiones es necesario disponer de un motor stepper que permita realizar movimientos muy pequeños siendo estable y robusto. El seleccionado para este TFM es un motor bipolar híbrido Nema 17 que son de los más comunes en este campo [34]. A su vez, para su manejo es necesario disponer de un driver de comunicación siendo el seleccionado el A4988 de Pololu ya que permite a bajas tensiones de alimentación llegar a pasos del motor muy pequeños [35]. También se considera necesario la presencia de un motor de tipo servo, manejado por PWM, para permitir realizar distintos ángulos de barrido siendo el escogido el motor servo Parallax por sus características de funcionamiento [36].

Por otra parte, es necesaria la presencia de sensores de posición que permitan detectar el paso del sistema por determinados puntos para poder sincronizar la adquisición de datos de forma adecuada. De esta forma, el sensor seleccionado es de tipo Hall, el DRV5053VAQLPG de Texas Instruments [37]. Al seleccionar este sensor es necesaria la presencia de imanes ferromagnéticos en punto estratégicos del diseño [38].

Teniendo en cuenta todos los componentes electromecánicos seleccionados es necesario diseñar e implementar un sistema mecánico que permita integrarlos en un único

prototipo que permita realizar medidas en tres dimensiones. En la Figura 1.13 se ilustra un ejemplo del sistema desarrollado con el sistema electromecánico, la adquisición de datos y manejo del sistema y su posterior representación.

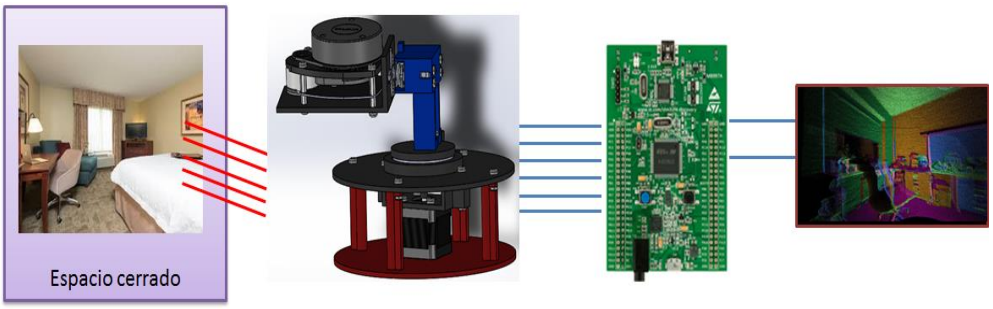


Figura 1.13. Esquema de la solución adoptada.

El microcontrolador ARM Cortex-M4 seleccionado es el procesador STM32F407VGT6 de STMicroeletronics siendo la arquitectura de habitual uso en el grupo de investigación (División de Equipos y Sistemas de Comunicación (COM) del IUMA). El microprocesador M4 incorporado permite soportar el trabajo con grandes volúmenes de datos (nube de puntos) a una velocidad de trabajo máxima de 168 MHz y presenta periféricos adecuados en este caso como pueden ser el timer o la USART [39]. Por último, la tarjeta de desarrollo seleccionada la STM32F4 Discovery dispone del microcontrolador descrito anteriormente además de otros periféricos tales como un USB bidireccional que permite enviar y recibir datos de otro dispositivo [40].

2 Descripción del Hardware

2.1 Introducción

En este capítulo se procederá a detallar los distintos componentes hardware empleados en el desarrollo del trabajo fin de máster. Los componentes hardware que se usarán en este TFM son:

- *LiDAR* RPLIDAR A1 de Smlatec
- Microcontrolador STM32F407VGT6 de STMicroelectronics
- Tarjeta de desarrollo STM32F4 Discovery
- Motor stepper Nema-17 y driver A4988 de Pololu
- Motor servo de rotación continua Parallax
- Sensor de efecto Hall DRV5053VAQLPG e imán ferromagnético

2.2 Microcontrolador

2.2.1 Introducción

Es un microcontrolador de la marca STMicroelectronics de la familia STM32 concretamente el STM32F407VGT6 [39]. Los dispositivos de la familia STM32 son dispositivos de baja potencia con un microcontrolador de alto rendimiento. Los micros de la serie F4 tiene una memoria Flash máxima de hasta 2 MB y una memoria máxima RAM de hasta 384 K.

2.2.2 Familia STM32

El microcontrolador usado para este proyecto, como se observa en la Figura 2.1, pertenece a la gama de alto rendimiento dentro de la familia STM 32. Dentro de esta, sin embargo, se encuentra en el rango intermedio entre los 3 modelos disponibles en las que sus principales diferencias radican en la frecuencia de trabajo del microprocesador, la memoria flash y la memoria SRAM disponible. La serie F4 puede llegar a trabajar a 180 MHz según el modelo concreto empleado.

De acuerdo con las especificaciones concretas del modelo utilizado, la frecuencia de trabajo máximo del microprocesador es de 168 MHz, una memoria Flash máxima de 1 MB y una memoria máxima RAM de 192 K [41].

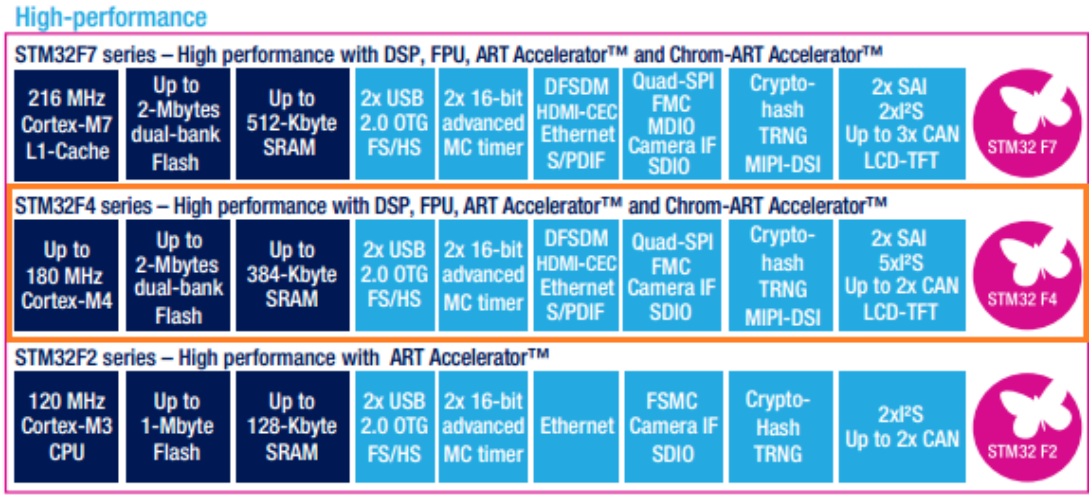


Figura 2.1. Dispositivos familia STM32 de alto rendimiento.

2.2.3 Familia Cortex

Los dispositivos Cortex son la actual generación de núcleos integrados de ARM. Este circuito es un procesador de núcleo completo formado por la CPU del Cortex y los distintos periféricos situados alrededor dando lugar a lo que es el corazón del dispositivo.

Los microprocesadores ARM Cortex son empleados para múltiples usos. Estos usos vienen delimitados por la letra que acompaña al nombre [42]. Existen tres tipos de Cortex:

- Cortex-A: Se emplea para aplicaciones de procesadores para complejos Sistemas Operativos y aplicaciones de usuario. Soporta los juegos de instrucciones ARM, Thumb y Thumb-2.
- Cortex-R: Se emplea para perfiles de sistemas en tiempo real. Soporta los juegos de instrucciones ARM, Thumb y Thumb-2.
- Cortex-M: Perfil para microcontrolador optimizado para aplicaciones sensibles al coste. Soporta sólo el juego de instrucciones Thumb-2.

Por otra parte, el número que se encuentra a continuación de la letra, descrita anteriormente, indica el nivel de rendimiento relativo del micro siendo 1 el más bajo y 8 el más alto. Teniendo en cuenta estas consideraciones podemos determinar que el microprocesador de la familia STM32 seleccionado es un ARM Cortex-M4. El Cortex-M4 proporciona sistemas de interrupción, temporizadores, sistemas de depuración y mapas de memoria [42]. El espacio de direcciones de 4 Gbytes se divide en distintas regiones según su función: código, SRAM y periféricos. Presenta una arquitectura Harvard lo que le permite tener varios buses realizando operaciones en paralelo aumentando su rendimiento global. En la Figura 2.2 se puede observar cómo es la estructura de un Cortex-M4 dónde se representa el sistema de interrupciones (NVIC), la ALU de 32 bits o el juego de instrucciones que soporta (Thumb-2).

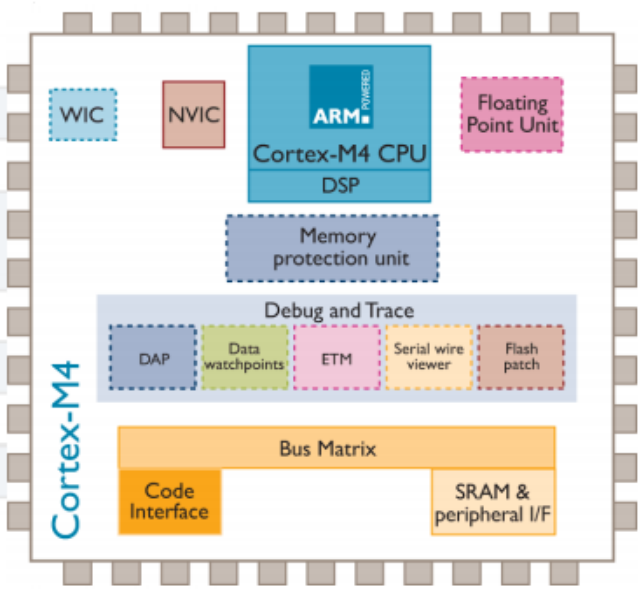


Figura 2.2. Arquitectura ARM Cortex-M4.

2.2.4 Arquitectura STM32F4

El STM32F407VGT6 se compone de un núcleo Cortex que se encuentra conectado a la memoria Flash mediante el bus de instrucciones. Los buses de datos y sistemas del Cortex son conectados a la matriz de ARM mediante los buses de alta velocidad avanzada (AHB) [43]. En la Figura 2.3 se puede observar como los periféricos son conectados a la matriz mediante dos buses de periféricos avanzados (APB). La matriz de bus AHB presenta la misma velocidad que el núcleo del Cortex (como máximo 168 MHz). Sin embargo, el bus APB2 se puede ejecutar a 82 MHz mientras que el APB1 solo a 42 MHz. El arbitraje de la matriz de buses garantiza 2/3 del tiempo de acceso para el DMA y 1/3 para la CPU del Cortex. Al bus AHB1 se encuentran conectados los pines de propósito general (GPIO), mientras que al bus APB1 se hayan los periféricos usados: USART y Timers. A su vez, el USB se encuentra conectado al núcleo a través del bus AHB2.

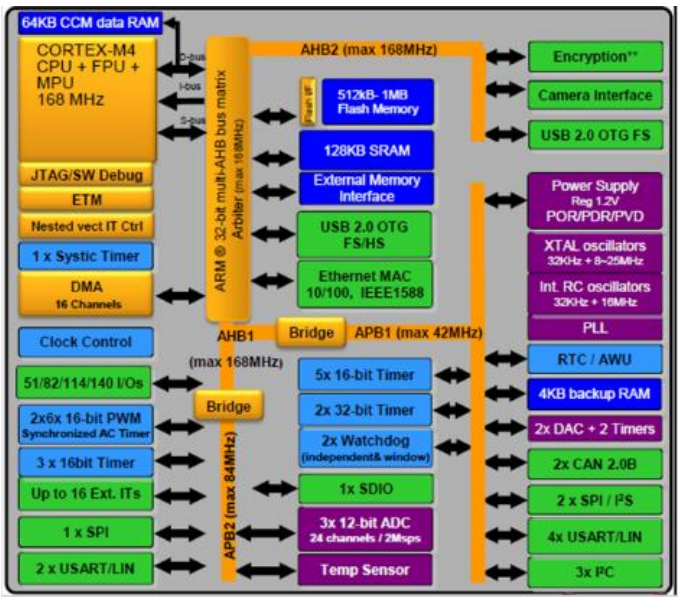


Figura 2.3. Arquitectura STM32F407.

2.2.4.1 Mapa de Memoria

Desde un punto de vista la capacidad de memoria el STM32F407 ofrece un espacio de direcciones de 4 Gbytes. El mapa de memoria debe ajustarse a la disposición estándar del Cortex. Como se ilustra en la Figura 2.4, existen áreas de memoria asignadas para el código

de usuario (512 MB), periféricos (512 MB), memorias externas (1 GB) o para dispositivos externos como tarjetas SD (1GB).

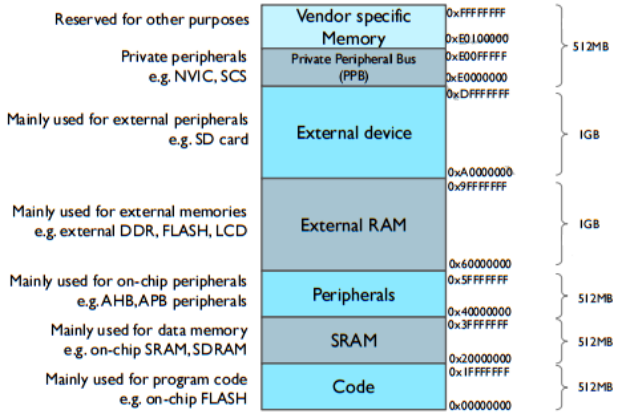


Figura 2.4. Mapa de Memoria.

2.2.4.2 Oscilador y señales de reloj

Existen dos tipos de osciladores: los de alta velocidad y los de baja velocidad. Los osciladores de alta velocidad se suelen usar para derivar procesos del Cortex y de relojes periféricos. Por el contrario, los de baja velocidad se emplean para el reloj en tiempo real principalmente.

El primer oscilador interno es de alta velocidad funcionando a 8 MHz mientras que el segundo oscilador es de baja velocidad funcionando a 32,768 KHz. A su vez, la señal de reloj que se genera puede ser adaptada según las necesidades (prescaler) de los perifériosque se configuren.

2.2.5 Fuente de alimentación y circuito de reset

El STM32F103 requiere de una fuente de alimentación que debe estar en el rango de 2,0 a 3,6 V. Es necesario la presencia de un regulador interno para generar un suministro de 1,8 V para el núcleo del Cortex. Por otra parte, el STM32 presenta otras dos fuentes de alimentación opcionales. La primera es una batería de respaldo para conservar datos de interés (reloj en tiempo real y registros de especial importancia). La segunda fuente de

alimentación se emplea para dar alimentación al ADC cuya tensión se encuentra limitada entre 2,4 V y 3,6 V [41]. También posee pines que permiten dar una tensión de 3,3 o 5 V.

El STM32F407 contiene un circuito de reset interno que coloca al dispositivo en modo reset siempre y cuando VDD esté por debajo de 2 V con una histéresis de 40 mV. La potencia de encendido o apagado del reset garantiza que el procesador solo funcionará con una tensión de la fuente de alimentación estable. En general no se requiere ningún circuito de reset externo.

2.2.6 Periféricos

En este apartado se comentarán los periféricos del microprocesador usados para el desarrollo del proyecto, se realizará una introducción general y se comentará las principales funciones de sus librerías de desarrollo software. Los periféricos que se tratarán son: GPIO, USART y USB.

2.2.6.1.1 GPIO (General Purpose Input/Output)

2.2.6.1.1.1 Descripción general

El microcontrolador STM32 está bien comunicado con pines de propósito general ya que tiene más de 80 pines de entrada/salida bidireccional. Los pines son dispuestos como 9 puertos cada uno con 15 líneas de entrada/salida.

Cada pin puede estar configurado como GPIO o como una función alternativa. También puede, simultáneamente, estar configurado una de las 15 líneas de interrupción externa. Estos 9 puertos son llamados desde la A hasta la I y presentan una tolerancia de tensión de 5V. Por ejemplo, si se quiere configurar el pin 4 del puerto B hay que referirse a él como PB4. Muchos de los pines externos pueden cambiar de estar asignados como propósito general a que sirva de entrada/salida para algún periférico de usuario.

Cada puerto GPIO puede configurar pines de manera individual tanto de entrada como de salida independientemente de la configuración del controlador. Posee registros para escribir o para realizar operaciones de bit atómicas. Una vez que una configuración se define

estos pines pueden ser bloqueados, es decir, no se permite cambiar su configuración por el programador [40].

El campo modo de los registros del GPIO permite al programador establecer el pin como entrada o como salida mientras que el campo de configuración define las características de la unidad. Se puede conectar una resistencia interna a la entrada de cada pin configurable como pull up o de drenador de puerta abierto. Cada pin de salida puede tener varias velocidades de salida máximas: 2 MHz, 25 MHz, 50 MHz y 100 MHz.

2.2.6.1.1.2 Librerías stm32f4xx_gpio.c y stm32f4xx_gpio.h

En el desarrollo del firmware para la configuración de los pines de la GPIO se han usado las librerías provistas por el fabricante. A continuación, se procederá a comentar las funciones, usadas en el proyecto, más importantes:

- Función de inicialización: se encarga de inicializar un determinado pin de un puerto concreto con la configuración establecida por el usuario como podemos ver en el Código 2.1. Para ello se hace uso de una estructura llamada GPIO_InitTypeDef. En esta estructura se puede configurar el tipo de pin que será (entrada o salida) se puede configurar su velocidad, su modo (input, output, analog o alternate function.), el tipo de salida y si es pull-up o pull-down.

```
void GPIO_Init(GPIO_TypeDef*GPIOx,GPIO_InitTypeDef*GPIO_InitStruct);
typedef struct
{
  uint32_t GPIO_Pin;
  GPIOSpeed_TypeDef GPIO_Speed;
  GPIOMode_TypeDef GPIO_Mode;
  GPIOOType_TypeDef GPIO_OType;
  GPIOPuPd_TypeDef GPIO_PuPd;
}GPIO_InitTypeDef;
```

Código 2.1. GPIO_InitTypeDef.

- Funciones de lectura de pines: la GPIO es capaz de leer datos tanto de pines de entrada como de salida. De esta forma existen dos tipos de funciones de lectura, una para los pines de salida y otra para los de entrada. Las 4 funciones, que se describen en el Código 2.2, permiten leer los pines dependiendo si son de salida o, de entrada. Mientras las funciones del tipo *ReadData* leen los pines del puerto

completo, las funciones del tipo *ReadDataBit* lee un pin determinado de un puerto.

```
uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx);
uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
uint16_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx);
```

Código 2.2. Funciones de lectura de pines.

- Funciones de Set y Reset: mientras la función de Set se encarga de poner un determinado pin de un puerto a "1", la función de Reset se encarga de poner un "0" en dicho pin. Las dos primeras funciones que se observan en el Código 2.3 utilizan el registro mencionado anteriormente, mientras las dos siguientes realizan la misma operación, pero sin usar dicho registro.

```
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction
BitVal);
void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal);
```

Código 2.3. Funciones de escritura y reset.

2.2.6.1.2 USART (Universal Synchronous/Asynchronous Receiver/Transmitter)

2.2.6.1.2.1 Descripción general

El microcontrolador STM32F4 presenta hasta 5 USARTs que se encargan de la transmisión y recepción serie asíncrona de datos. Cada uno de ellos presenta una velocidad de hasta 4.5 Mbps.

Cada USART presenta una interfaz serie independiente y completamente programable. Se puede configurar el formato de la trama (bits de paridad, stop), velocidad de transmisión, tamaño de la trama, etc. Normalmente una trama se compondrá de: 1 bit de start, una palabra de datos de 8 o 9 bits y 0,5, 1, 1,5 o 2 bits de stop [39].

2.2.6.1.2.2 Librerías stm32f4xx_usart.c y stm32f4xx_usart.h

En el desarrollo del firmware para la configuración de la USART se usan las librerías dadas por el fabricante. A continuación, se explican las funciones usadas en el proyecto más importantes.

- Función de inicialización: se encarga de establecer la configuración de una USART en función de los parámetros que se deseen. Como se puede ver en el Código 2.4, toda esta configuración se realiza mediante la estructura USART_InitTypeDef. Esta estructura permite definir distintos aspectos como la velocidad de transmisión, la longitud del paquete de datos a enviar, bit de stop, paridad, modo (recepción o transmisión), etc.

```
void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef*
USART_InitStruct);

typedef struct
{
  uint32_t USART_BaudRate;
  uint16_t USART_WordLength;
  uint16_t USART_StopBits;
  uint16_t USART_Parity;
  uint16_t USART_Mode;
  uint16_t USART_HardwareFlowControl;
} USART_InitTypeDef;
```

Código 2.4. USART_InitTypeDef.

- Funciones de configuración: las 4 funciones que se observan en el Código 2.5 se emplean para habilitar/deshabilitar la USART, configurar si funciona por interrupción y las funciones de envío y recepción de datos.

```
void USART_Cmd(USART_TypeDef* USARTx, FunctionalState NewState);
void USART_ITConfig(USART_TypeDef* USARTx, uint16_t USART_IT,
FunctionalState NewState);
void USART_SendData(USART_TypeDef* USARTx, uint16_t Data);
uint16_t USART_ReceiveData(USART_TypeDef* USARTx);
```

Código 2.5. Funciones de configuración de una USART.

2.2.6.1.3 USB (Universal Serial Bus)

2.2.6.1.3.1 Descripción general

El microcontrolador STM32F4 dispone de USB bidireccional por lo que se debe configurar el dispositivo para que funcione de forma adecuada. Permite transmitir de 5 a 16 bits, bit de paridad y bits de stop si lo hubiera. La tasa binaria de comunicación es adaptable según las necesidades presentando velocidades de 9600 o 115200 bps según sean las necesidades.

2.2.6.1.3.2 Librerías usbd_cdc_if.c y usb_cdc_if.h

Para poder trabajar con el USB se debe configurar el USB-OTG como Device-Only para que actúe solo como dispositivo tanto en cuento el otro punto de la conexión deberá ejercer como maestro. Debido a problemas que se observaron en la configuración base del mismo a la hora de mandar gran volumen de datos (se colapsaba) se realizaron modificaciones en la forma de comunicación.

La implementación realizada para solucionarlo utiliza colas de transferencia de 2KB. En transmisión, los datos se escriben en la cola de salida a medida que entran. En recepción los datos son descargados a una FIFO intermedia cada vez que la función de servicio (*USB_CDC_Service*) es llamada. Esta función debe ser ejecutada periódicamente cada milisegundo en el archivo de interrupciones. En cada llamada verifica si hay datos pendientes en recepción y los añade a la FIFO correspondiente y realiza un flush de la salida. Todas las comunicaciones son bufferizadas de tal forma que se solventa el problema descrito al comienzo del apartado, gestionadas por interrupción y, potencialmente, se pueden alcanzar tasas de transferencia de hasta 16Mbps.

La función, que se ilustra en el Código 2.6, que permite leer los datos recibidos se llama *GetChar()* y devuelve en un entero sin signo de 16 bits el byte recibido en los 8 bits con menos peso. En caso de que el buffer se encuentre vacío se envía como respuesta un 0xFFFF.

```
uint16_t GetChar()
{
    if (RX_FIFO_AVAIL) {
        char ch;
        if (RX_FIFO_EMPTY)
            return 0;
        ch = UserRxFIFO.data[UserRxFIFO.tail++];
        UserRxFIFO.tail %= APP_RX_DATA_SIZE;
        return ch;
        }
    return 0xFFFF;
}
```

Código 2.6. Función GetChar.

A su vez, para transmitir datos desde el microcontrolador hacia el otro dispositivo existen dos variantes; por una parte con la función *PutChar(unsigned char)* que permite

enviar un byte en cada ejecución tal y como se observa en el Código 2.7 o por otra usar la función *printf()* que permite enviar un string completo sin tener que ir llamando a la función descrita anteriormente carácter a carácter.

```
void PutChar(char ch)
{
     if (UserTxBufferCount < APP_TX_DATA_SIZE) {
         UserTxBufferFS[UserTxBufferCount++]=ch;
     }
}
```

Código 2.7. Función PutChar.

2.3 Tarjeta de desarrollo STM32F4 Discovery

2.3.1 Introducción

La tarjeta de desarrollo STM32F4 Discovery es la seleccionada entre las disponibles en el mercado con el microcontrolador STM32F407VGT6 tanto por los periféricos que presenta como por su precio. Este microcontrolador presenta una memoria flash de 1 MB, 192 KB de RAM y un encapsulado LQFP100 [41].

2.3.2 Características generales

Cabe destacar entre sus características principales la presencia de una interfaz de comunicación serie mediante la presencia de un USB to RS232 que permite enviar datos por USB hacia un dispositivo compatible. Además, presenta otras características que le dan un valor añadido a esta tarjeta de desarrollo [40]:

- Conector ST-LINK/V2 que puede trabajar con un ST-LINK/V2 independiente (con conector SWD para programación y depuración).
- Alimentación a la tarjeta a través del bus USB o desde una fuente de alimentación externa de 5V.
- Sensor de movimiento ST MEMS LIS302DL que incluye un acelerómetro con salida digital de 3 ejes.

- Sensor de audio ST MEMS MP45DT02 con micrófono digital omnidireccional.
- Audio DAC CS43L22 con controlador integrado de altavoz clase D.
- Ocho leds
- Dos pulsadores (usuario y reset).
- USB-OTG con conector microUSB.

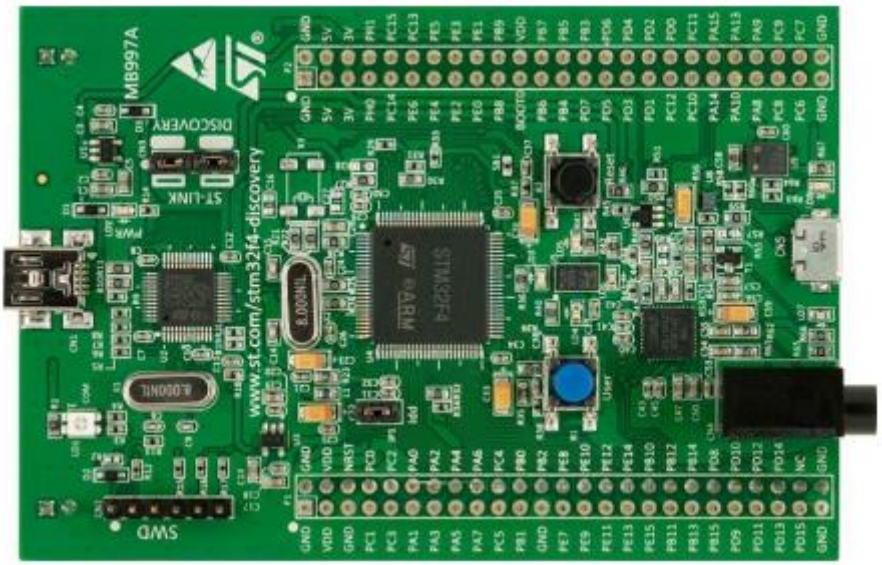


Figura 2.5. Tarjeta de desarrollo.

2.4 LiDAR RPLIDAR A1

2.4.1 Introducción

RPLIDAR A1 es una solución basada en un escáner láser 2D (*LiDAR*) de 360 grados y de bajo coste desarrollada por Slamtec. El sistema puede realizar la exploración 360 grados dentro del rango permitido (6 metros). Con los datos de los puntos 2D obtenidos pueden emplearse en mapeo, localización o modelado de objeto y entornos. La frecuencia de barrido de RPLIDAR A1 normal es de 5.5 Hz pudiendo estar configurada a 10 Hz como máximo. Este dispositivo se caracteriza por ser un sistema de medición láser por triangulación. Puede

trabajar en todas las clases de ambiente interiores y al aire libre sin luz del sol directa. Se ha seleccionado esta opción *LiDAR* porque de acuerdo al análisis previo es el que más se adapta a las exigencias del entorno y a las limitaciones económicas del proyecto.

2.4.2 Arquitectura del sistema

RPLIDAR A1 contiene un escáner láser por triangulación junto con un sistema motor que permite rotar el láser para poder obtener medidas en dos dimensiones. Después de encender cada subsistema, el *LiDAR* comienza a girar y escanear en el sentido de las agujas del reloj como se ilustra en la Figura 2.6 [33]. Se pueden obtener datos de la exploración a través de la interfaz de comunicación serie (UART).

Figura 2.6. Arquitectura del sistema.

Además, dispone de un sistema de detección de velocidad adaptativo en el que sistema cambia la frecuencia del explorador del láser automáticamente según la velocidad del motor. Por otra parte, el sistema receptor puede obtener la velocidad real del RPLIDAR A1 a través de la interfaz de comunicación [44]. En la Tabla 2.1 se ilustran los parámetros que caracterizan al RPLIDAR destacando la posibilidad de medir distancia desde 0,5 a los 6 metros con una resolución con un error típico de 0,5 mm para objetos blancos. Por otro lado, presenta una resolución angular menor o igual a 1º y una frecuencia de movimiento del motor de entre 1 y 10 Hz.

Item	Unit	Min	Typical	Max	Comments
Distance Range	Meter(m)	TBD	0.15 - 6	TBD	White objects
Angular Range	Degree	n/a	0-360	n/a	
Distance Resolution	mm	n/a	<0.5 <1% of the distance	n/a	<1.5 meters All distance range*
Angular Resolution	Degree	n/a	≤1	n/a	5.5Hz scan rate
Sample Duration	Millisecond(ms)	n/a	0.5	n/a	
Sample Frequency	Hz	n/a	≥2000	2010	
Scan Rate	Hz	1	5.5	10	Typical value is measured when RPLIDAR A1 takes 360 samples per scan

Tabla 2.1. Parámetros característicos del RPLIDAR A1

RPLIDAR se basa en el principio de la triangulación láser (en distintas posiciones por el movimiento del motor) y utiliza hardware de adquisición y procesamiento de visión de alta velocidad desarrollado por Slamtec. El sistema mide datos de distancia más de 2000 veces por segundo y con alta resolución (<1% de la distancia) y una distancia máxima detectable de 6 metros [33]. Normalmente, RPLIDAR emite una señal láser infrarroja (a través de un diodo láser) modulada y dicha señal es reflejada por el objeto a detectar como se observa en la Figura 2.7. De esta forma cuando la señal rebotada llega al *LiDAR* converge mediante una lente y se realiza la medición empleando un detector CMOS.

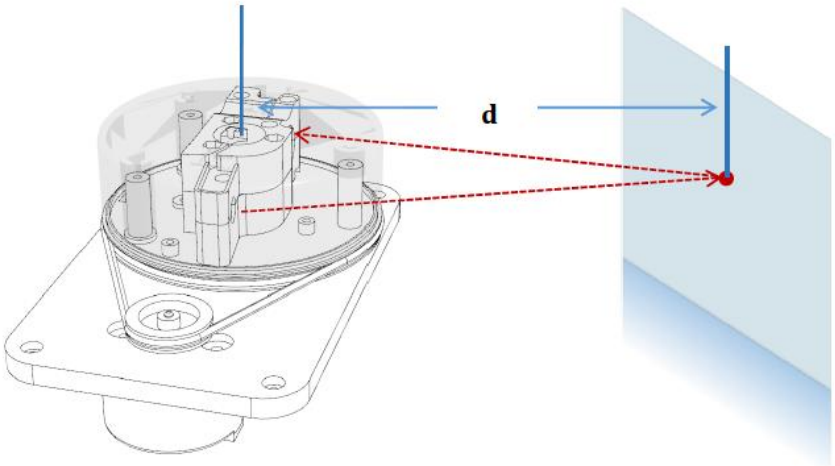


Figura 2.7. Triangulación láser.

La señal de retorno es muestreada por el sistema de adquisición de visión y el DSP incorporado. Dicha señal converge en la lente de recepción que con ayuda de un sensor de

posición que permite medir el lugar de la luz en dos dimensiones. Posteriormente, se procesan los datos de la muestra con el valor de distancia de salida y el valor de ángulo entre el objeto y el sistema de adquisición tal y como se muestra en la Figura 2.8 a través de la interfaz de comunicación. El sistema de escáner de alta velocidad se monta en una estructura giratoria con un sistema de codificación angular incorporado. Durante la rotación, por tanto, se realiza una exploración de 360 grados del entorno.

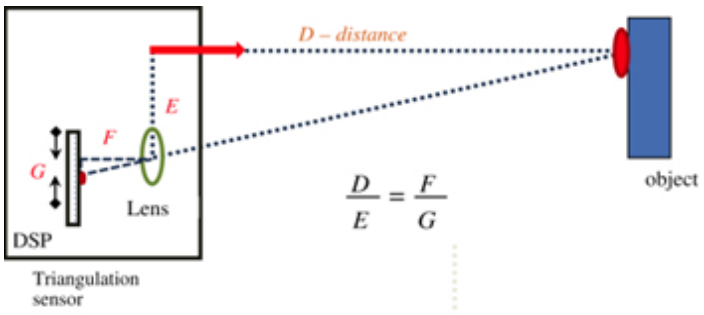


Figura 2.8. Cálculo de la distancia y el ángulo.

El sistema RPLIDAR A1 utiliza como fuente de luz un láser de infrarrojos de baja potencia (<5mW) funcionando mediante impulsos modulados. En la Tabla 2.2 se identifican las principales características del láser empleado [33] como su longitud de onda (785 nm). El láser emite en un marco de tiempo muy corto manteniendo la seguridad para humanos y mascotas (clase I de seguridad). La principal ventaja del láser modulado es que es inmune a las perturbaciones producidas tanto por la luz artificial como la luz solar durante el proceso de exploración de alcance. Por tanto, puede funcionar en todo tipo de ambientes interiores y exteriores sin luz solar.

Parámetro	Valor	Unidad
Longitud de onda	775-795 (785 típico)	nm
Potencia	3-5 (3 típico)	mW
Longitud del pulso	110	µS

Tabla 2.2. Parámetros del láser.

2.4.3 Interfaz de comunicación

RPLIDAR A1 utiliza un puerto serie 3.3V-TTL (UART) como interfaz de comunicación. La otra interfaz de comunicación, USB, se puede modificar para requisitos particulares según las necesidades. La Tabla 2.3 describe la especificación para la interfaz

de puerto serie donde destaca que tanto el motor como el núcleo de procesamiento debe ser alimentado con una tensión de 5 V [44]. Además, en la Figura 2.9 se observa la configuración de los pines de las señales en el propio dispositivo.

Interface	Signal Name	Type	Description	Min	Typical	Max
Motor Interface	VMOTO	Power	Power for RPLIDAR A1 Motor	-	5V	9V
	MOTOCTL	Input	Enable signal for RPLIDAR A1 Motor/PWM Control Signal	0V	-	VMOTO
	GND	Power	GND for RPLIDAR A1 Motor	-	0V	-
Core Interface	VCC_5	Power	Power for RPLIDAR A1 Range Scanner Core	4.9V	5V	6V
	TX	Output	Serial output for Range Scanner Core	0V	-	5V
	RX	Input	Serial input for Range Scanner Core	0V	-	5V
	GND	Power	GND for RPLIDAR A1 Range Scanner Core	-	0V	V5.0

Tabla 2.3. Parámetros interfaz de comunicación.

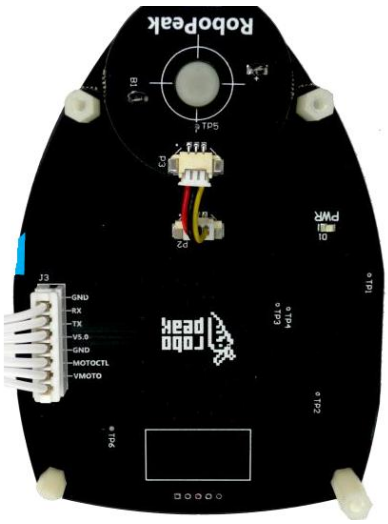


Figura 2.9. Esquema de conexiones RPLIDAR A1.

2.5 Motor stepper

2.5.1 Introducción

Los motores paso a paso (stepper) se usan generalmente en una variedad de aplicaciones en las que es deseable un control de posición preciso a un bajo coste. Por lo tanto, es óptimo para la realización de un movimiento con mayor resolución del *LiDAR* que permita una adquisición de datos más fiable. En este TFM, es necesaria su presencia en el sistema electromecánico para poder realizar un escaneo con precisión en 3 ejes (en este caso, el eje azimutal)

2.5.2 Características principales

Este motor, que se puede observar en la Figura 2.10, presenta dos fases que trabajan a una corriente de 1,7 A a 2,8 V [34]. Por otra parte, indicar que el ángulo de paso del motor es de 1,8 º lo que implica una frecuencia de 200 pasos /vuelta. Además, el par motor dispone de la fuerza necesaria (3.7 kg por cm) para poder mover el peso que soporta que se estima en torno a 400 gramos.

Figura 2.10. Motor Stepper.

2.5.3 Driver A4988

Para el control y manejo del motor no se puede directamente atacar a las bobinas proporcionándole la corriente necesaria sino que es necesario la presencia de un controlador que permita manejar el stepper (corrientes) de forma correcta [45]. El seleccionado para este proyecto es el A4988 de Pololu (con integrado de Allegro), que se observa en la Figura 2.11, que permite trabajar con el motor.

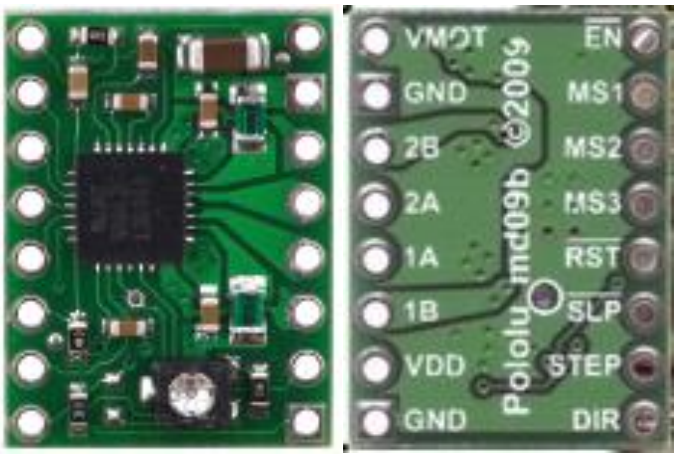


Figura 2.11. Driver A4988.

La principal ventaja de este driver es que permite una limitación de corriente ajustable, protección contra sobretemperatura y sobrecorriente y seis resoluciones de micropaso (hasta 1/16-paso). Funciona de 8 V a 35 V y puede entregar hasta aproximadamente 2 A por fase sin un disipador de calor o flujo de aire [35]. Es preciso indicar que para poder alcanzar las resoluciones más pequeñas es necesario limitar la corriente. Para ello, haciendo uso del potenciómetro se ajusta la tensión a la dada por la siguiente expresión: $V_{REF} = 8 * I_{MAX} * R_{CS}$ donde I_{MAX} es la corriente máxima por fase que admite el motor y R_{CS} puede ser 50 o 68 mΩ dependiendo de la resistencia del sensor. Por último, en la Tabla 2.4 se puede observar la funcionalidad de cada uno de los pines disponibles [35].

Pin	Tipo	Función
VMOT	Entrada	Tensión alimentación: 8-35 V
GND	Tierra	Tierra del driver
B2-B1	Salida	Par de cables de una bobina del motor
A2-A1	Salida	Par de cables de una bobina del motor

VDD	Entrada	Tensión continua (3-5.5 V)
EN	Entrada	Habilita el driver (Activo a nivel bajo)
MS1-MS2-MS3	Entrada	M2 M1 M0 Full Step 0 0 0 Half Step (1/2) 1 0 0 1/4 Step 0 1 0 8 microsteps/step 1 1 0 16 microsteps/step 1 1 1
RST	Entrada	Reset del sistema (Activo a nivel bajo)
SLP	Entrada	Situación de bajo consumo del sistema (Activo a nivel bajo)
STEP	Entrada	Señal de reloj que permite movimiento del motor
DIR	Entrada	Señal de dirección de giro (1-sentido antihorario, 0-sentido horario)

Tabla 2.4. Pines del A4988.

2.6 Motor servo Parallax

2.6.1 Introducción

El Parallax Standard Servo es un tipo de motor que permite agregar rotación continua bidireccional a los sistemas mecánicos. La principal diferencia de este tipo de motor respecto al anterior es que este tipo de motor no tiene la precisión en el movimiento que el anterior; por lo tanto, es eficaz en entornos donde la precisión no sea un factor clave. En el sistema electromecánico su funcionalidad corresponderá a un movimiento en el ángulo de elevación que permita pasar de la posición de inicio a reposo y viceversa y, si fuera necesario, realizar barridos en otros ángulos y no en la vertical.

2.6.2 Características principales

Este tipo de motor que se observa en la Figura 2.12 normalmente es manejado mediante una señal PWM. El punto de estabilidad del servo se encuentra en una señal de pulso de 1,5 ms dentro de un rango de 1,3 ms para el sentido horario y de 1,7 ms para el sentido antihorario [36].

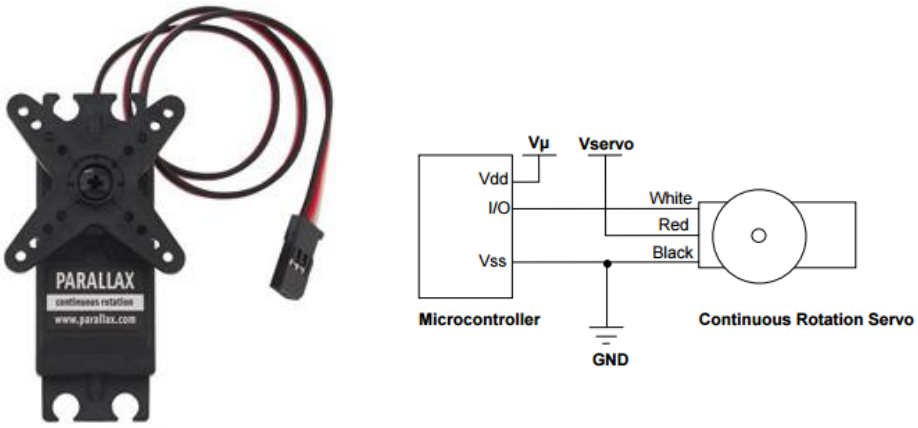


Figura 2.12. Motor servo y configuración.

2.7 Sensor de efecto Hall DRV5053 e imán

2.7.1 Introducción

El dispositivo DRV5053VAQLPG es un integrado de efecto Hall estabilizado por chopper que permite controlar el paso de un objeto a través de su campo de acción gracias a la detección magnética [37]. La salida analógica de respuesta es de 0 a 2 V respondiendo linealmente a la densidad de flujo magnético aplicada y, además, distinguiendo la polaridad de la dirección del campo magnético [37]. Debido a la salida analógica limitada a 2 V asegura que no exista sobre tensión a la entrada donde se vaya a aplicar. Por otro lado, es necesaria la presencia de un imán ferromagnético que permita establecer un campo electromagnético entre sí mismo y el sensor.

2.7.2 Características principales del sensor de efecto Hall

Presenta, un amplio rango de voltaje de operación de 2,5 a 38 V lo que hace que sea adecuado para una amplia gama de aplicaciones industriales y de consumo. En la Figura 2.13, se observan los pines del componente destacando que el modelo usado es el de inserción debido a la aplicación que se le va a dar en este TFM.

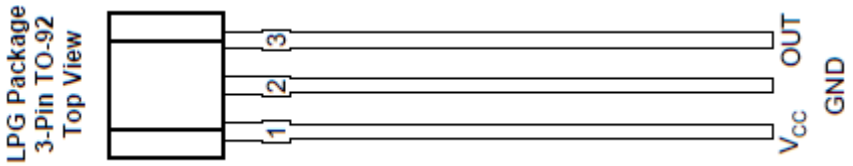


Figura 2.13. Pines DRV5053.

Al ser un sensor de efecto Hall, la tensión de salida depende del campo magnético perpendicular al dispositivo. La ausencia de un campo magnético dará a la salida una tensión constante de 1 V. Un campo magnético que se encuentre en el radio de acción del dispositivo hará que la tensión de salida cambie linealmente con el campo magnético.

La polaridad de campo se define tal y como sigue: un polo sur cerca de la parte frontal del dispositivo es un campo magnético positivo (tensión disminuye por debajo de 1 V), mientras que un polo norte (tensión aumenta por encima de 1 V) cerca de dicha parte frontal es un campo magnético negativo [46].

2.7.3 Imán

El imán seleccionado es de la empresa Radial Magnet Inc. el cual presenta forma circular (4.78 mm x 1.60 mm) que se puede visualizar en la Figura 2.14 [47]. Está compuesto de una aleación de neodimio-hierro-boro (NdFeB) con terminación de níquel-cobre-níquel (NiCuNi) y, además, se encuentra magnetizado axialmente [38].

Figura 2.14. Imán.

Capítulo 3

3 Diseño del sistema electromecánico

3.1 Introducción

En este capítulo se documenta el diseño del sistema electromecánico desarrollado, el entorno de desarrollo empleado y la funcionalidad que cumple. Además, se especifica la forma de fabricación del mismo y los materiales empleados. En el anexo de esta memoria se encuentran de forma detallada los planos del diseño de cada una de las piezas.

3.2 Objetivos y consideraciones previas del diseño y fabricación

El principal objetivo que se persigue con este diseño es obtener un sistema compacto, estable y manejable que permita realizar las mediciones en tres dimensiones. Para ello, se debe conseguir que el sistema pueda efectuar (con ayuda de los dispositivos mecánicos para ello) movimientos completos (360 grados), tanto, en el plano horizontal en forma circular como en el plano vertical. Una condición importante que debe cumplir el diseño es la robustez del mismo para que sea capaz de soportar el peso de los dispositivos sin producir movimientos bruscos del sistema que puedan invalidar las medidas del sistema.

Para el diseño de este sistema se ha usado la herramienta de diseño SolidWorks 2014. A su vez, para la impresión del diseño se ha empleado una impresora 3D disponible en el laboratorio (Prusa i3) con capacidad para imprimir las piezas diseñadas. Para ello, emplea el software de gestión de impresión Slic3r. Como material de impresión se usa PLA (poliácido láctico) que es el tipo de termoplástico habitual que se usa en este tipo de impresiones.

3.3 Piezas del diseño

En función de los objetivos explicados anteriormente y, teniendo en cuenta, las limitaciones físicas de la impresora disponible, se diseña un sistema electromecánico, el cual se puede observar en la Figura 3.1, que permite realizar una adquisición tridimensional de 360º. Las piezas que forman este diseño son:

- Base inferior
- Base superior
- Soporte del stepper
- Soporte del servo
- Base giratoria del stepper
- Base del *LiDAR*
- Contrapeso *LiDAR*

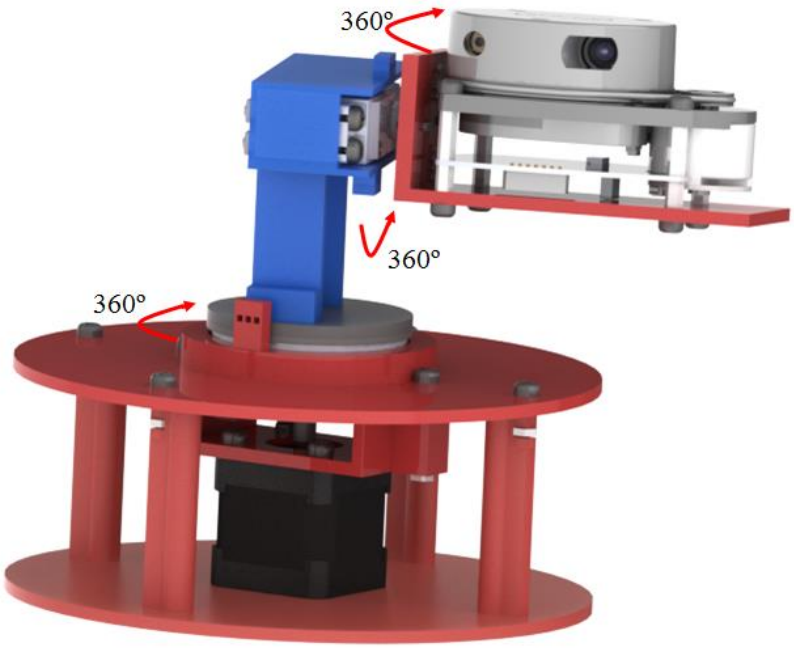


Figura 3.1. Sistema electromecánico final.

3.3.1 Base inferior

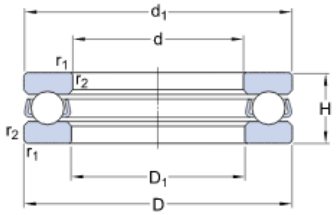
Esta pieza que se puede observar en la Figura 3.2 tiene como principal funcionalidad soportar el peso de todo el sistema descargando de ello al stepper. El plano con las medidas

que caracterizan esta pieza se puede analizar en el Anexo 1. Figura 1. Para ello dispone de columnas (de la altura adecuada en base al tamaño del stepper) que permiten trasladar el peso del sistema. La sujeción del sistema se realiza mediante puntos habilitados en los soportes de esta base que se ajustan a la base superior.

Figura 3.2. Base inferior.

3.3.2 Base superior

La funcionalidad principal de la base superior, cuyas dimensiones se pueden consultar en el Anexo 1. Figura 2, es la de dar soporte y sujeción al stepper como a la base giratoria que se encuentra dentro de su eje de rotación que se puede observar en la Figura 3.3. Indicar, además que para facilitar el movimiento de esta pieza y con la intención de evitar el rozamiento producido se introduce un rodamiento axial de bolas 51108 como se ilustra en la Figura 3.4 donde se adjuntan sus características métricas [48]. Para asegurar el movimiento adecuado del rodamiento este se encuentra dentro de una estructura circular de altura igual a 3/4 partes de la altura de dicho rodamiento. Esta pieza irá sujeta tanto al soporte del stepper como de la base inferior. Adicionalmente, en este soporte existe un saliente que fijará el sensor de efecto Hall (inserción) que debe disponer de tres orificios a pequeñas distancias para poder introducir el componente y de esta forma quedarse sujeto.



d		40	mm
D		60	mm
H		13	mm
d_1	≈	60	mm
D_1	≈	42	mm
$r_{1,2}$	min.	0.6	mm

Figura 3.3. Dimensiones rodamiento de bolas.

Figura 3.4. Base superior.

3.3.3 Soporte del stepper

El objetivo del soporte del stepper es dar rigidez y estabilidad a este motor evitando su movimiento cuando se encuentre en funcionamiento. Para ello, como se representa en la Figura 3.5 habrá cuatro puntos de sujeción sobre el motor y dos adicionales uno por cada lado a la base superior para conseguir que por acción del peso del sistema evitar el movimiento de este motor. El plano con las dimensiones que describen a esta pieza se encuentra en el Anexo 1. Figura 3.

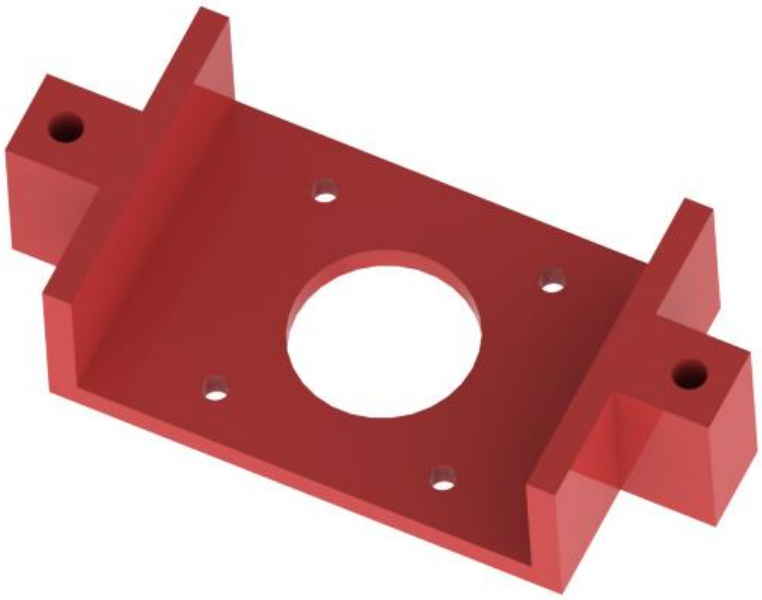


Figura 3.5. Soporte del stepper.

3.3.4 Soporte del servo

La principal funcionalidad de esta pieza radica en tener un punto de anclaje robusto para el servo cuando se encuentre en movimiento. Para dar mayor estabilidad al sistema, se diseña la pieza teniendo en cuenta que este motor se colocará de forma horizontal. Además, este se debe encontrar a una altura tal que permita realizar el movimiento del *LiDAR* sin chocar con ningún obstáculo. Por una parte, el motor servo se anclará a esta pieza mediante tornillos situados en la parte superior, como se observa en la Figura 3.6 y esta pieza se sujetará a la base giratoria del stepper. Además, debe incluir dos piezas que permitan sujetar los sensores de efecto Hall (montaje de inserción) por lo que deberán disponer de tres orificios a pequeñas distancias entre ellos para que puedan entran las patas del componente. Las dimensiones que caracterizan a esta pieza se pueden observar en el Anexo 1. Figura 4.

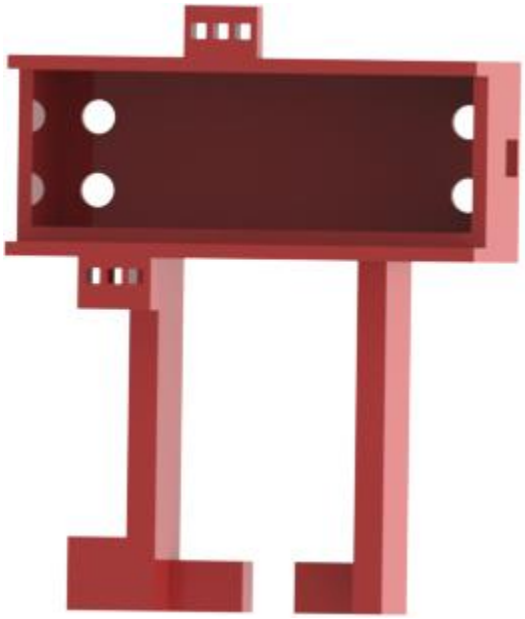


Figura 3.6. Soporte del servo.

3.3.5 Base giratoria del stepper

La base giratoria del stepper, cuyas dimensiones se ilustran en el Anexo 1. Figura 5, debe cumplir dos funcionalidades principales: la primera es ajustarse perfectamente a la forma del eje de rotación del motor (reforzado con soportes) y la segunda es dar estabilidad al rodamiento gracias al saliente de la parte superior tal y como se ilustra en la Figura 3.7. Por otra parte, indicar que a su vez que a esta pieza se sujeta el soporte del servo y que, gracias a la presencia del rodamiento, el movimiento circular que se realiza no produce ninguna fricción.

Figura 3.7. Base giratoria del stepper.

3.3.6 Base del LiDAR

La base del stepper tiene como objetivo principal trasladar el movimiento del servo hacia el *LiDAR*. Para ello, la huella de la hélice se sujeta con la propia hélice del servo mediante los tornillos adecuados como se ilustra en la Figura 3.8. A su vez, a la parte plana de esta base se sujeta el *LiDAR* que gracias a la fuerza de este tipo de motor se podrá mover. Para conocer concretamente las dimensiones de esta pieza se debe acudir al Anexo 1. Figura 6.

Figura 3.8. Base del LiDAR.

3.3.7 Contrapeso

Debido a la ligera inclinación que se observa en el sistema cuando se monta se opta por diseñar una pequeña caja, como la que se observa en la Figura 3.9, que se pueda atornillar al soporte del servo por su parte posterior para ejercer la función de contrapeso. De esta forma, y con el peso adecuado en dicho soporte se consigue equilibrar el sistema. Sus dimensiones se ilustran en el Anexo 1. Figura 7.

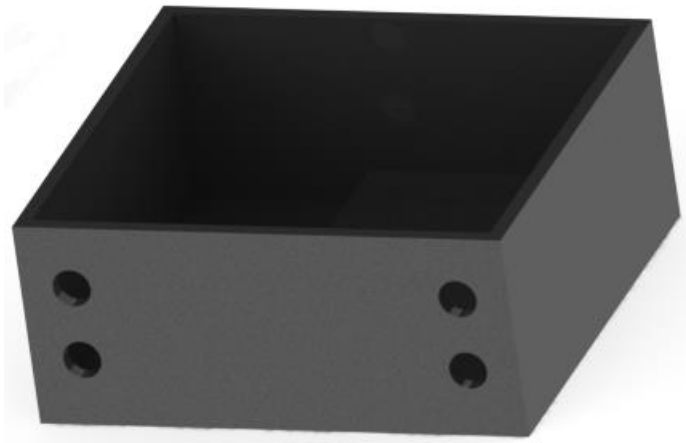


Figura 3.9. Contrapeso.

3.4 Integración del sistema electromecánico

Para el montaje del sistema completo se deben usar los tornillos de longitud y métrica adecuada, además, de las tuercas necesarias para ello. Además, para la simulación del sistema mecánico se han usado modelos disponibles (del tipo setp) por los fabricantes del *LiDAR*, del servo y del stepper seleccionados. El sistema electromecánico completo en distintos ángulos del RPLIDAR se observa en la Figura 3.10 donde adicionalmente se indica la forma en que realice un escaneado tridimensional ya que tanto en azimut como en elevación podrá hacer una rotación de 360º. A partir de esta imagen y sabiendo que el LiDAR da como datos ángulo y distancia se puede determinar que las medidas obtenidas siguen un patrón de coordenadas esféricas. Por último, con el movimiento del servo también se consigue que el escaneado del espacio en estudio se pueda hacer en distintos ángulos (360º).

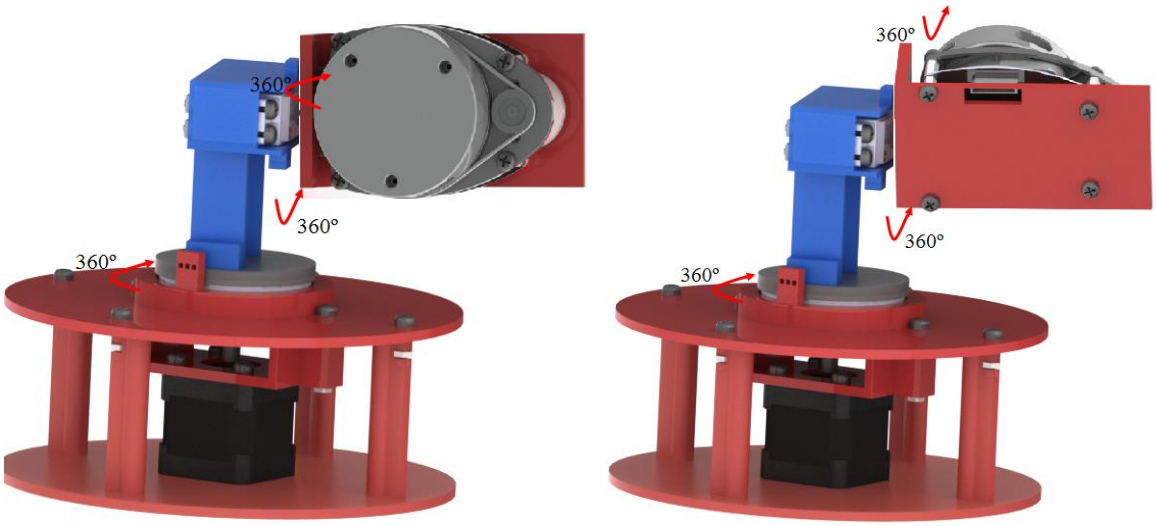


Figura 3.10. Sistema electromecánico final.

3.5 Configuración electrónica del sistema diseñado

3.5.1 Introducción

En este apartado se describe la funcionalidad de los dispositivos electrónicos empleados y la configuración firmware utilizada para poder manejarlos correctamente. Los dispositivos electrónicos descritos en este apartado son: el motor stepper Nema-17, el motor servo y los sensores de movimiento.

3.5.2 Motor stepper Nema-17

La primera comprobación que se debe realizar con este motor es si dispone de la fuerza adecuada para mover el sistema completo. Conociendo que el par de mantenimiento (holding torque) del motor es de 3,7 kg por cm y sabiendo que el diámetro del eje es de 0,5 cm; este motor tendrá una fuerza resultante de 1,85 kg [49]. Haciendo una estimación al alza del peso total que soportaría dicho motor (600 gramos), éste se encuentra muy por debajo

del límite de peso establecido anteriormente por lo que sería capaz de mover el sistema con total seguridad.

Una característica muy importante que hace que este tipo de motores sea de gran interés para estos sistemas es el microstepping. El microstepping se caracteriza por conseguir movimientos del motor por debajo de un paso normal pudiendo resoluciones del motor de hasta 32 veces por debajo de un paso normal lo que da una aproximación de su gran precisión y suavidad en el movimiento. En la Figura 3.11 se puede observar de forma analítica cómo funciona el microstepping [50]. Mientras que en una configuración estándar existen 4 movimientos dando lugar a una señal cuadrada a medida que se aumenta el número de mircopasos la señal se deforma en pequeñas señales cuadradas dando lugar a una sinusoidal como se observa en la última imagen. A niveles eléctricos esto implica que la corriente que el driver deriva a cada una de las bobinas va variando en menor grado de un paso a otro a medida que se aumenta el nivel del microstepping [51]. Llegar hasta grandes factores de reducción en el paso del motor vendrá dado por las limitaciones del driver asociado; en este caso el A4988 llega a una resolución de 16 micropasos por paso [45].

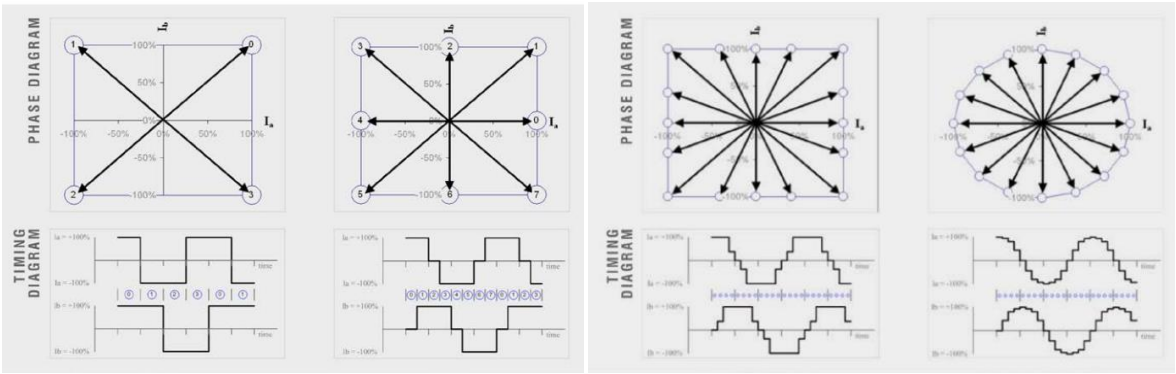


Figura 3.11. Tipos de microstepping.

En líneas generales es habitual que en este tipo de motores el paso estándar sea de 1,8º; por lo tanto; en un modo full step (modo normal) se necesitarían de 200 pasos para poder alcanzar una vuelta completa [34]. Por lo tanto, si se quisiera dar una vuelta completa en una configuración de 1/4 de micropaso (0,45º) por paso se necesitarían de 800 pasos Otra señal importante que se debe tener en cuenta a la hora de trabajar con este tipo de motores es la señal de reloj (step) que debe introducirse al driver que tiene como función establecer

la velocidad del movimiento de la rotación dentro de una misma configuración de paso y la señal de dirección del giro del eje.

A partir de estas consideraciones se desarrolla la función en C que permita controlar el stepper tal y como se ilustra en el Código 3.1. Esta función se caracteriza por tener cuatro parámetros que coinciden con los descritos previamente. El parámetro "period" indica el período de la señal de reloj que se introduce al driver para indicar la velocidad del giro. A su vez, el parámetro "step" indica el número de pasos que se desean dar en una vuelta valor que puede ir desde 1 hasta el valor máximo de pasos por vuelta en cada configuración (200*microstepping). La configuración de microstepping seleccionada se establece con el parámetro "config" que puede tomar los valores: 1, 2, 4, 8 y 16. Por último, con ayuda del parámetro "dir" se establece el sentido del giro siendo 0 el sentido horario y 1 el antihorario. El bucle que ejecuta la señal de reloj se encontrará activo mientras todavía existan pasos que dar según la configuración establecida y se produzcan ciertas condiciones adicionales en función de las necesidades del sistema tales como si ha llegado a la posición del sensor o si estando en esta posición se encuentra dentro de un rango de pasos ya dado.

```
void stepperConf(uint32_t period, uint32_t steps, uint8_t config, uint8_t dir){

     //Pin dir
     if(dir==0){
       GPIO_WriteBit(GPIOB, GPIO_Pin_7, Bit_RESET);
     }else if(dir==1){
       GPIO_WriteBit(GPIOB, GPIO_Pin_7, Bit_SET);
     }

      //Mode MicroStepping
     if(config==1){//Full Step (000)
       GPIO_WriteBit(GPIOE, GPIO_Pin_2, Bit_RESET);
       GPIO_WriteBit(GPIOE, GPIO_Pin_4, Bit_RESET);
       GPIO_WriteBit(GPIOE, GPIO_Pin_6, Bit_RESET);
    }else if(config==2){//Half Step (100)
       GPIO_WriteBit(GPIOE, GPIO_Pin_2, Bit_SET);
       GPIO_WriteBit(GPIOE, GPIO_Pin_4, Bit_RESET);
       GPIO_WriteBit(GPIOE, GPIO_Pin_6, Bit_RESET);
     }else if(config==4){//1/4 Step (010)
       GPIO_WriteBit(GPIOE, GPIO_Pin_2, Bit_RESET);
       GPIO_WriteBit(GPIOE, GPIO_Pin_4, Bit_SET);
       GPIO_WriteBit(GPIOE, GPIO_Pin_6, Bit_RESET);
     }else if(config==8){//1/8 Step (110)
       GPIO_WriteBit(GPIOE, GPIO_Pin_2, Bit_SET);
       GPIO_WriteBit(GPIOE, GPIO_Pin_4, Bit_SET);
       GPIO_WriteBit(GPIOE, GPIO_Pin_6, Bit_RESET);
     }else if(config==16){//Full Step (111)
       GPIO_WriteBit(GPIOE, GPIO_Pin_2, Bit_SET);
       GPIO_WriteBit(GPIOE, GPIO_Pin_4, Bit_SET);
       GPIO_WriteBit(GPIOE, GPIO_Pin_6, Bit_SET);
```

```
    }


while(((countStep<steps)&&(((GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_15)==1)&&((pos_stepp
er>0)&&(pos_stepper<100)))||((pos_stepper>=0)&&((GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_
15)!=1))))){
       countStepper=0;
       for(j=0;j<1;j++){
           while(countStepper<(period/2)){
                GPIO_WriteBit(GPIOB, GPIO_Pin_5, Bit_SET);
             }
           while(countStepper>=(period/2)){
                GPIO_WriteBit(GPIOB, GPIO_Pin_5, Bit_RESET);
                if(countStepper==delay){
                    countStepper=0;
                }
           }
       }
       countStep++;
    }
    flagHall=1;
    j=0;
}
```

Código 3.1. Función stepperConf.

En este TFM, el movimiento del stepper tendrá dos configuraciones estándar tal y como se indica en el Código 3.2. La configuración del microtepping es fija en ambas casos siendo establecida la de 1/4 de micropaso por paso, es decir, 0,45º por movimiento ya que se considera una resolución adecuada que permite llegar al compromiso entre una correcta caracterización del espacio frente al tiempo de escaneado.

```
stepperConf(30,800,4,0);

stepperConf(2,1,4,flag_antihorario);
```

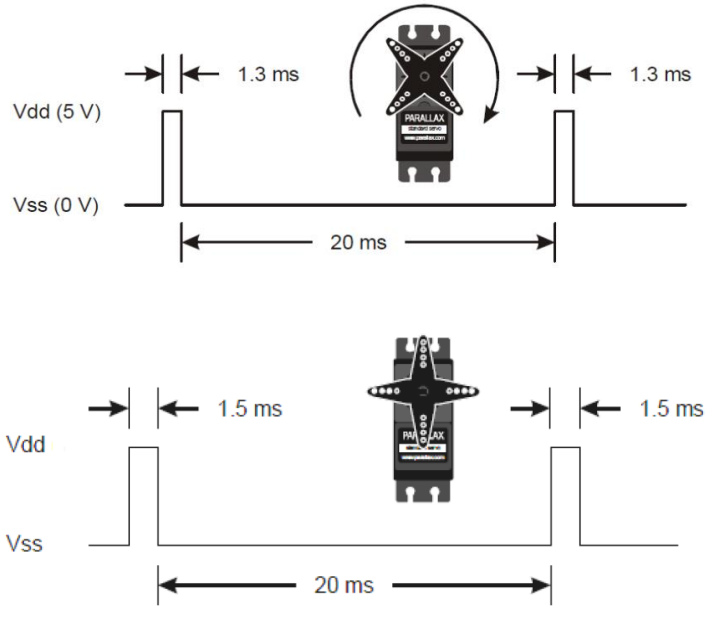
Código 3.2. Ejemplos de configuración del stepper.

Un primer movimiento de búsqueda del punto inicial se caracteriza por establecer una configuración de vuelta completa (800 pasos) debido a que no se sabe, a priori, cuantos pasos se tarda en llegar a la posición inicial establecida junto con una señal de reloj con un período lo suficientemente elevado (30 ms) que permita girar el sistema de forma suave y una dirección de giro fija en el sentido horario (0). Por otro lado, el segundo movimiento se establece para ejecutar el movimiento paso a paso necesario en el plano horizontal para poder caracterizar el espacio. De esta forma, es necesario que el movimiento presente una configuración de un solo pulso, una señal de reloj bastante más pequeña que la anterior (2 ms) y una dirección de rotación adaptable según las necesidades del sistema.

3.5.3 Motor servo Parallax

El motor servo Parallax, en las condiciones típicas de funcionamiento, puede soportar hasta 1076,7 gramos por lo que puede mover adecuadamente la parte mecánica que tenga asignada [36]. Los motores servo presentan como gran ventaja su robustez y fuerza frente a su precisión lo que es óptimo para el diseño realizado.

Este tipo de motores suelen trabajar con PWM (Pulse-Width Modulation), es decir, el movimiento viene basado por el ancho del pulso enviado durante el ciclo de trabajo. De esta forma, es necesario conocer el ciclo de trabajo y el ancho del pulso de los tres eventos que ocurre en este tipo de motores. Tal y como se ilustra en la Figura 3.12 la duración del pulso para que el motor esté parado es de 1,3 ms, para que se mueve en sentido horario de 1,5 ms (período máximo) y, por último para que rote en sentido antihorario de 1,7 ms (período máximo) [36]. Teniendo en cuenta que esto se debe producir en los tres casos cada 20 ms el período estándar del sistema es de 21,7 ms.



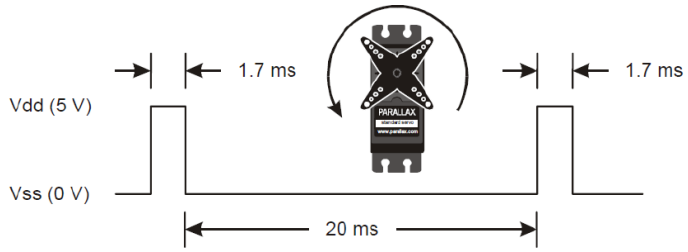


Figura 3.12. Período de trabajo del servo.

Conociendo el período de trabajo del motor se puede comenzar a diseñar la modulación empleada para el control de dicho motor. Primero se debe configurar el timer que dará lugar a la señal PWM adecuada (además de habilitar el pin correspondiente de la tarjeta como PWM) tal y como se ilustra en el Código 3.3.

```
void timerPWM() {
    TIM_TimeBaseInitTypeDef TIM_BaseStruct;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

    TIM_BaseStruct.TIM_Prescaler = 168;
    TIM_BaseStruct.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_BaseStruct.TIM_Period = 10804.24;
    TIM_BaseStruct.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_BaseStruct.TIM_RepetitionCounter = 0;
    TIM_TimeBaseInit(TIM4, &TIM_BaseStruct);
}
```

Código 3.3. Función timerPWM.

Para ello, se debe conocer la frecuencia por defecto del timer (timer 4) que debida la configuración establecida es de 84 MHz (con un prescaler fijado a 168) y la frecuencia de la señal PWM deseada, que viene establecida por $^{1}/_{Tmotor}$ donde el período es el de un ciclo del motor, es decir, 21,7 ms, es de 46 Hz. En base a estos datos la frecuencia del timer queda establecida por la siguiente expresión:

$$Frecuencia = \frac{\frac{Reloj\ del\ sistema}{Prescaler + 1}}{Frecuencia\ PWM\ deseada} - 1 = \frac{\frac{84 * 10^6}{168 + 1}}{46} - 1 = 10804{,}24\ Hz$$

Teniendo fijado el timer ya se puede pasar a configurar el ancho del pulso propiamente dicho. Para ello se debe tener .la frecuencia del timer establecida anteriormente y conocer el ciclo de trabajo en cada uno de los tres casos de movimiento del motor. Éste viene definido como el porcentaje del período (21,7 ms) de trabajo en el que el pulso se

encuentra a nivel alto. Por lo tanto, la duración del pulso vendrá marcada por los siguientes valores en los tres diferentes casos:

$$Ancho\ del\ pulso\ (1{,}3\ ms) = \frac{(Frecuencia + 1) * ciclo\ de\ trabajo\ (\%)}{100 - 1}$$

$$= \frac{(10804{,}24 + 1) * (\frac{1{,}3}{21{,}7} * 100)}{100 - 1} = 654$$

$$Ancho\ del\ pulso\ (1{,}5\ ms) = 764$$

$$Ancho\ del\ pulso\ (1{,}7\ ms) = 873$$

Conociendo el ancho del pulso para los tres casos en estudio ya se puede configurar el PWM de dicho timer tal y como se describe en el Código 3.4. Sin embargo, no se establece el ancho del pulso en esta configuración, sino que se atacará este registro (TIM4->CCR3) cuando sea necesario en función de las necesidades de sistema. A pesar de conocer los límites de velocidad del sistema en ambos giros se necesita una velocidad de movimiento mucho más lenta para que no afecte al sistema de forma significativa. De esta forma, los valores quedan establecidos de la siguiente forma: el ancho del pulso cuando se gire en sentido horario será de 755, cuando se encuentre quieto se fijará en 764 y cuando se gire en sentido antihorario se establecerá en 770.

```
void PWM_Init() {
    TIM_OCInitTypeDef TIM_OCStruct;
    TIM_OCStruct.TIM_OCMode = TIM_OCMode_PWM2;
    TIM_OCStruct.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCStruct.TIM_OCPolarity = TIM_OCPolarity_Low;
    TIM_OCStruct.TIM_Pulse = 0;
    TIM_OC3Init(TIM4, &TIM_OCStruct);
    TIM_OC3PreloadConfig(TIM4, TIM_OCPreload_Enable);
}
```

Código 3.4. Función PWM_Init.

3.5.4 Sensor de efecto Hall

El sensor de posición seleccionado es un sensor de tipo Hall de Texas Instruments, concretamente, el DRV5053VAQLPG por lo que se necesita que mediante un imán

ferromagnético se pueda establecer un campo magnético que avise cuando el sistema (imán) ha pasado por una determinada posición.

En base a lo visto en el apartado 2.7.2 cuando se produce un campo positivo esta señal aumenta hasta los 2 V y si es negativo desciende hasta los 0 V [37]. Disponer de un sensor que limite su salida entre los 0 y 2 V es óptimo para evitar sobretensiones sobre los pines de entrada de la tarjeta (3,6 V tensión máxima de entrada) [43].

Esto es de gran utilidad para poder gestionar desde el firmware los eventos en torno a estos sensores. Debido a que no es necesaria ningún tipo de comunicación entre el microcontrolador y el sensor no es necesario disponer de una función o un código específico del dispositivo a excepción de una gestión adecuada de la tensión de entrada a cada uno de los pines asignados y su correspondencia a nivel lógico. Teniendo en cuenta que la tensión de alimentación interna de la tarjeta de desarrollo (V_{DD}) es de 3V se pueden establecer los límites de decisión lógica (0, 1 o alta impedancia). La tensión máxima para que el microcontrolador establezca que una determinada tensión de entrada corresponde a un 0 viene dada por: $0{,}1 * V_{DD} = 0{,}1 * 3 = 0.33\ V$ [43]. De igual forma la tensión mínima para que una tensión de entrada corresponda a un 1 se establece por: $0{,}45 * V_{DD} + 0{,}3 = 0{,}45 * 3 + 0{,}3 = 1{,}65\ V$ [43]. Teniendo en cuenta esto, el valor estándar de 1 V dado por el sensor es a todos los efectos considerado por el microcontrolador como alta impedancia ya que no puede decidir a cuál de los dos valores lógicos corresponde. De forma experimental, se comprueba que el microcontrolador se comporta mejor detectando un nivel alto que un nivel bajo por lo que se opta por una configuración sensor-imán que dé lugar a un campo magnético positivo. La corriente que da el sensor es muy pequeña (2,3 mA) y en ningún caso producirá sobrecorriente sobre el pin de entrada que puede soportar hasta 25 mA.

4 Integración electrónica

4.1 Introducción

En este capítulo se detallan los pasos seguidos para llevar a cabo la integración electrónica de las distintas etapas que forman parte del diseño. Para ello se realiza una descripción de los materiales y componentes utilizados.

4.2 Materiales y componentes utilizados

Para la integración final del sistema se deben usar materiales y componentes (Introducción) que permiten llevar a cabo esta misión. A continuación, se realiza un listado de los materiales y dispositivos empleados:

- Cables de prototipado
- Tiras de pines.
- Termoretráctil
- Tornillos y tuercas
- Fuente de alimentación Promax FAC-662B
- Cable USB-microUSB.
- Cable USB-miniUSB.

4.3 Montaje del prototipo

En la Figura 4.1 se puede ver el esquemático del prototipo montado; en ella se observa las distintas etapas que lo conforman: en la parte izquierda se encuentra la etapa

electromecánica (mecánica, motores y sensores), en la parte central la etapa de adquisición y preprocesamiento y en la parte derecha la etapa de representación tridimensional (software). Además, en la Figura 4.2 se observa el montaje del prototipo implementado que se usará para realizar las medidas tridimensionales.

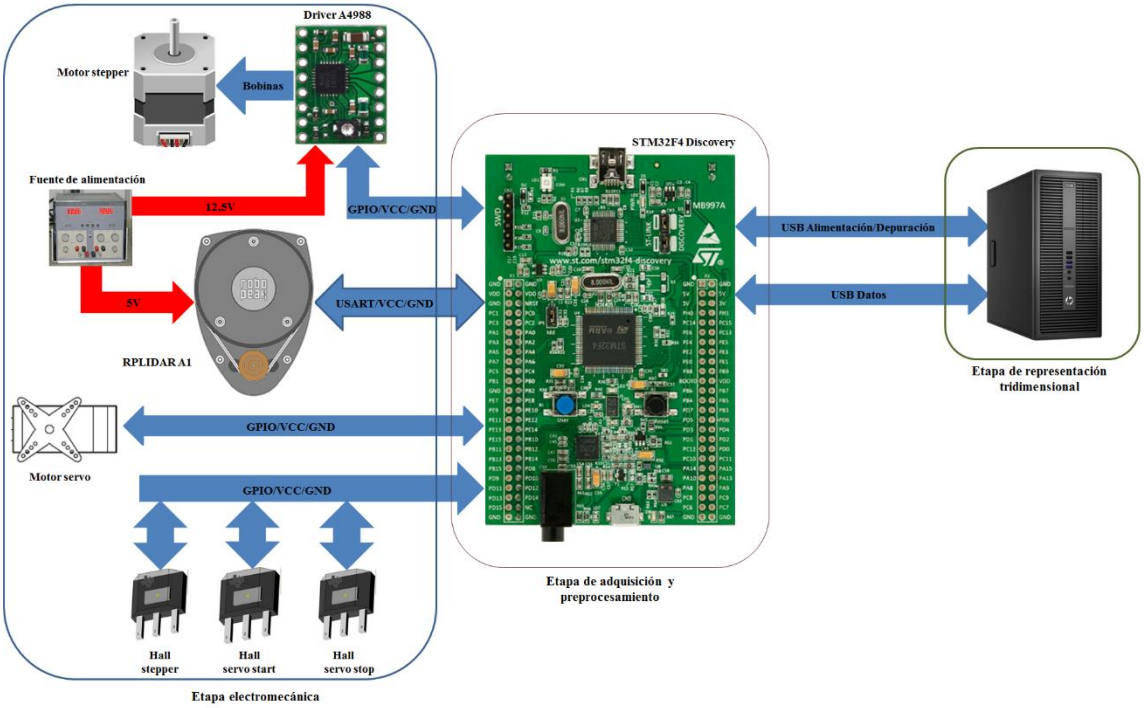


Figura 4.1. Esquemático del prototipo.

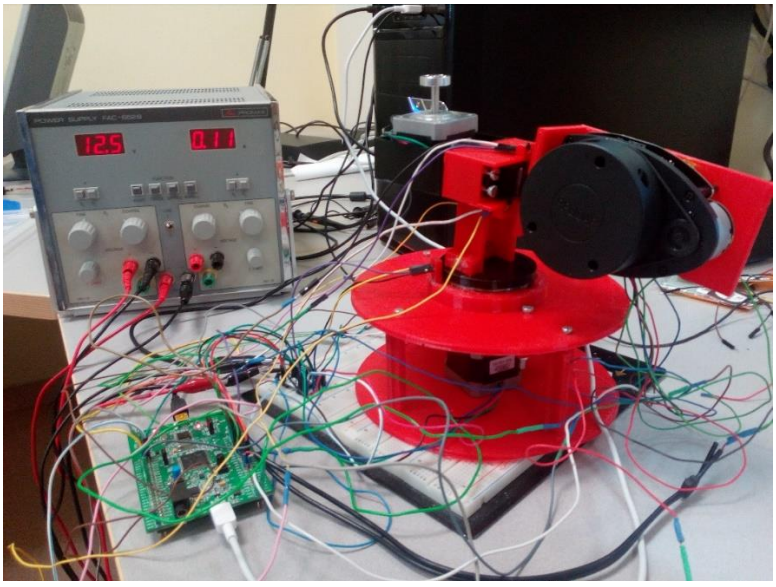


Figura 4.2. Prototipo real.

4.4 Etapa electromecánica

Corresponde a la primera parte del prototipo diseñado disponer de un sistema hardware que permita realizar los movimientos tanto en el plano horizontal como en el vertical que faciliten la realización de las medidas por el *LiDAR* tal y como se ha explicado en el capítulo 3. A partir de ello, se debe proveer de las conexiones necesarias tanto de alimentación, tierra como de funcionamiento de los motores. Se encuentra formado por tres componentes: el motor Stepper (Nema-17) con su correspondiente driver de control A4988, el motor servo Parallax y los tres sensores de efecto Hall DRV5053VAQLPG con su dispositivo de señalización asociado (imán ferromagnético), aparte del propio RPLIDAR A1 tal y como se ilustra en la Figura 4.3.

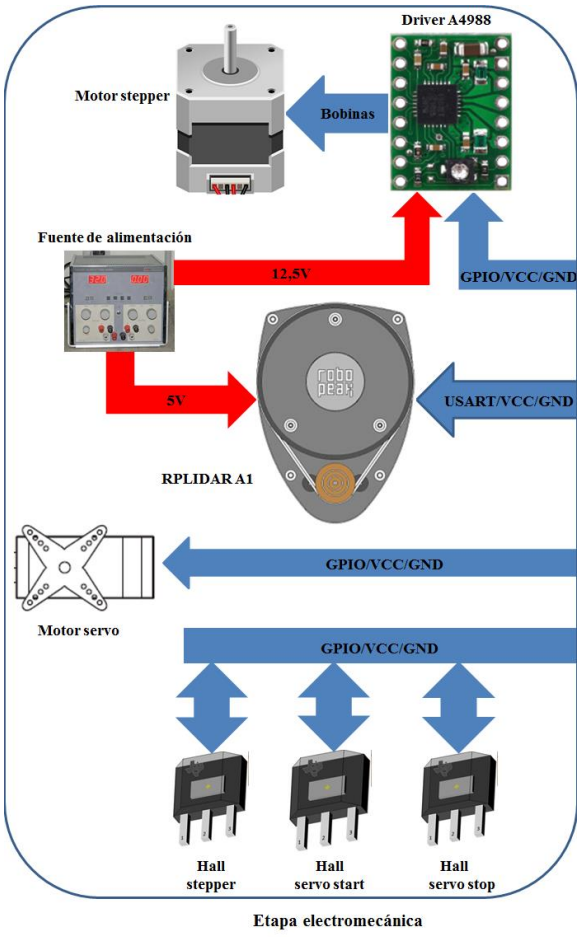


Figura 4.3. Etapa electromecánica.

Es importante resaltar que para la configuración del stepper de forma correcta es necesario limitar la corriente que el driver A4988 proporcionar a las dos bobinas del motor [45]. La tensión con la que se alimenta aeste dispositivo es de 12,5 V consumiendo 120 mA. Para ello, haciendo uso del potenciómetro se ajusta a la tensión dada por la expresión: $V_{REF} = 8 * I_{MAX} * R_{CS}$ donde I_{MAX} es la corriente máxima por fase que admite el motor y R_{CS} puede ser 50 o 68 mΩ dependiendo de la resistencia del sensor. En este caso, gracias al driver seleccionado la R_{CS} será de 50 mΩ [35] y la I_{MAX} que puede soportar cada una de las bobinas es de 1,68 A [34]; por lo tanto la tensión de referencia a la que se debe ajustar el potenciómetro es de 0,672 V. Sin embargo, realizando pruebas de funcionamiento se comprueba que el driver se calienta en exceso y hay un elevado nivel de ruido; por lo que experimentalmente se ajusta el potenciómetro (bajando la tensión de referencia) hasta el mínimo indispensable para que el motor pueda moverse de forma adecuada. Esto se consigue a una tensión de referencia de 0,421 V consumiendo 120 mA.

El motor Servo Parallax en comparación con el anterior no presenta un driver previo que controle el movimiento del mismo, sino que este se puede realizar directamente desde la tarjeta de desarrollo. Además, no es necesaria la conexión del mismo a ninguna fuente de alimentación, sino que la propia tarjeta le da la tensión necesaria (5V).

Además, debido al diseño del RPLIDAR A1 se debe conectar el pin de VMOTO a una tensión fija de 5 V y limitada en corriente a 2 A, aunque su consumo realmente sea de 10 mA. Con esta tensión aplicada se consigue una resolución en el ángulo obtenido de 1,2º que se considera adecuado para la realización de las medidas, aunque se pueden llegar a resoluciones más finas (reduciendo la tensión) o mayores (aumentando la tensión).

Por último, el sensor de efecto Hall DRV5053VAQLPG [37] colocados de forma adecuados y con sus correspondientes puntos de señalización (imanes) [38] situados en posiciones críticas del diseño tal y como se observan en la Figura 4.4 correspondiente a la señalización de la posición inicial de paso de vuelta del stepper, en la Figura 4.5 que se encarga de indicar la posición inicial del *LiDAR* (motor servo) en vertical para poder realizar las medidas de forma correcta y, por último en la Figura 4.6 que representa la posición del *LiDAR* en reposo cuando no se vayan a realizar medidas. De esta forma, cada vez que el

imán pase por el campo de acción donde se encuentre el sensor se podrá conocer la situación actual tanto en el movimiento circular (stepper) como en el vertical (servo).

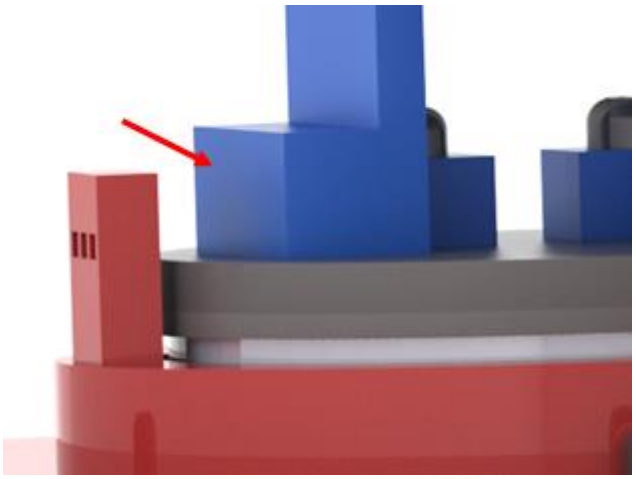


Figura 4.4. Sensor Hall para el stepper.

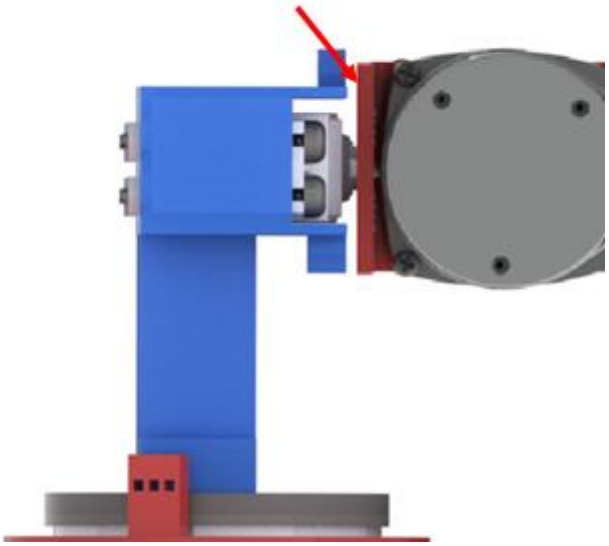


Figura 4.5. Sensor Hall para el start del servo.

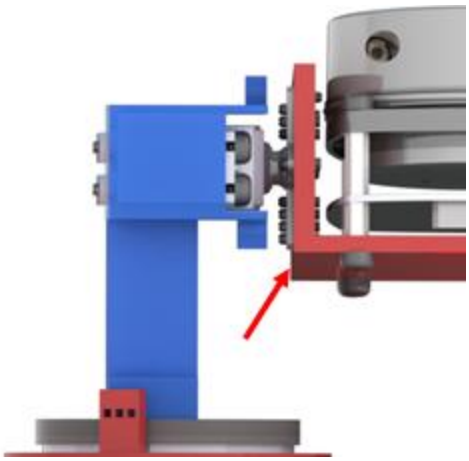


Figura 4.6. Sensor Hall para el stop del servo.

4.5 Etapa de adquisición y procesamiento

La etapa de adquisición y procesamiento comprende la tarjeta de desarrollo STM32F4 Discovery que tiene como función obtener las medidas del *LiDAR* de forma adecuada. Para ello a través de esta tarjeta se manejan de forma coordinada los movimientos de los distintos motores, las indicaciones del sensor hall y las medidas obtenidas por el *LiDAR*. Por un lado, en la Figura 4.7 se ilustran las conexiones que se realizan sobre el driver A4988 para un funcionamiento básico; en este diseño sería necesario además poder comunicarse con la interfaz del microstepping (MS1, MS2 y MS3). Por otro lado, en la Figura 4.8 se observa el esquemático de la etapa electromecánica indicando con qué dispositivos se comunica.

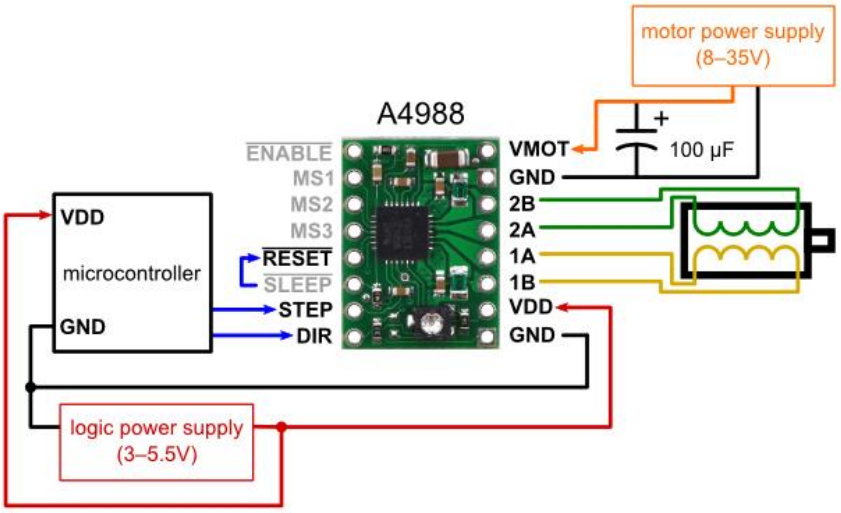


Figura 4.7. Conexiones driver A4988.

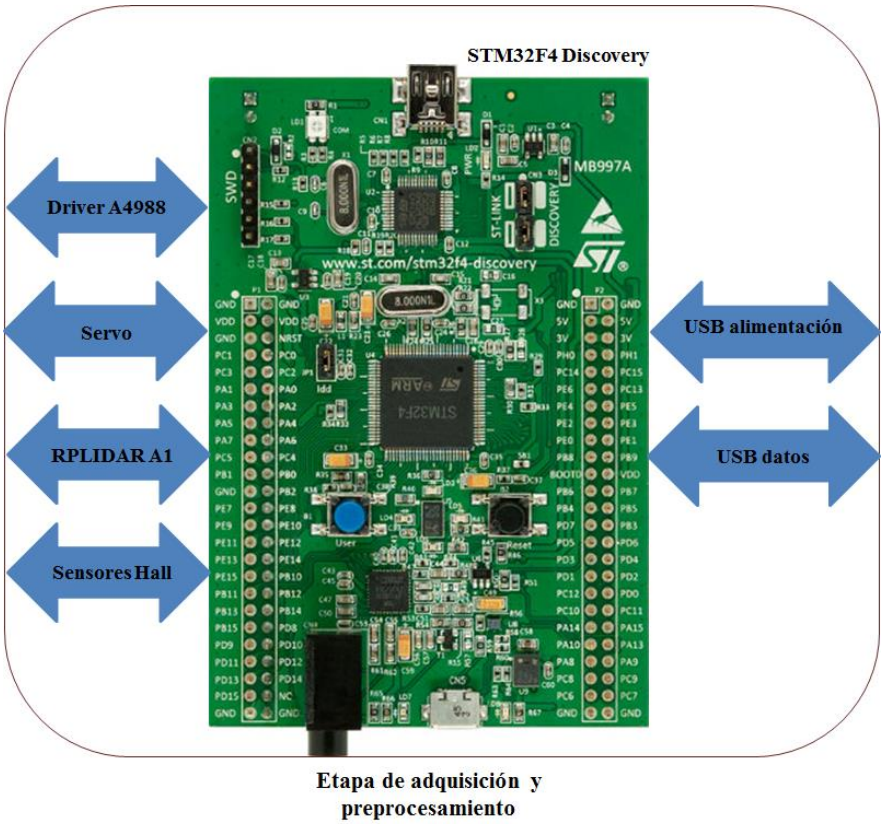


Figura 4.8. Etapa de adquisición y procesamiento.

4.6 Etapa de representación tridimensional

La etapa de representación tridimensional se basa, principalmente, en realizar el procesado de los datos y su posterior representación tridimensional en el ordenador, por lo tanto, se necesita una conexión vía USB para la comunicación tal y como se observa en la Figura 4.9. Además, la tarjeta de desarrollo no tiene una fuente de tensión independiente que la alimente por lo que es necesario conectar la tarjeta y el PC a través del USB de alimentación y depuración.

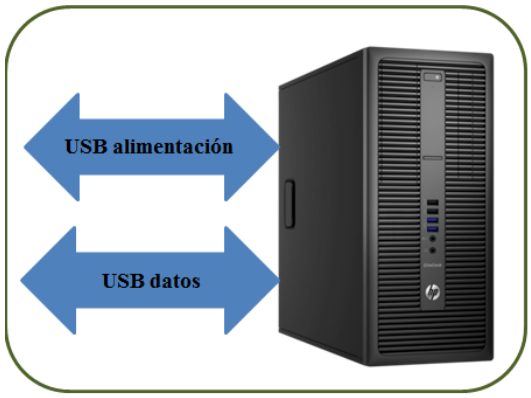


Figura 4.9. Etapa de representación tridimensional.

5 Diseño firmware

5.1 Introducción

En este capítulo se describe el diseño firmware implementado para la gestión del sistema electromecánico, es decir, gestión del *LiDAR*, motores y comunicación USB para que se puedan adquirir y transmitir los datos de forma adecuada. De por sí, este firmware no ejecuta funciones autónomas, sino que debe primero recibir la indicación de comienzo de programa de la aplicación software de representación de los datos.

Primero, se realiza una visión general de la estructura y formato del código. A continuación, se hace hincapié en algunas configuraciones de periféricos que son necesarias para la ejecución del programa y, por último, se describe cual es la secuencia de funcionamiento del programa principal y se realiza la descripción de las distintas etapas que conforman el diseño.

5.2 Estructura del código

Para la realización del firmware de control y adquisición se utiliza el entorno de programación Keil uvision4 que usa lenguaje C. Además, de las librerías provistas por el fabricante, se han implementado librerías propias dependiendo de la aplicación que se necesite. Además, también existe un archivo de texto que comenta la funcionalidad general del programa. La estructura de ficheros generada se puede visualizar en la Figura 5.1. Por una parte, se encuentran las librerías propias del sistema relacionadas con periféricos específicos (tipo HAL, por ejemplo) y las que se encuentran relacionadas con el USB.

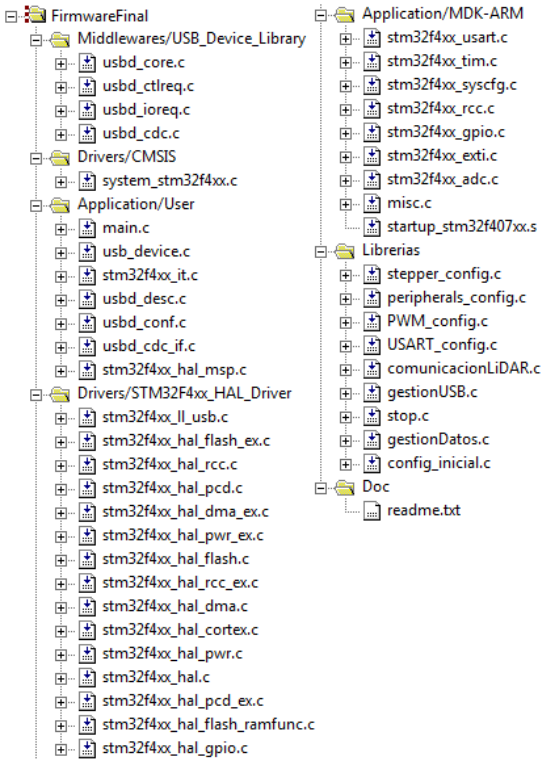


Figura 5.1. Estructura del código.

Para la generación inicial de este programa se empleó el software STM32Cube de ST [52] que permite configurar, inicialmente a más alto nivel, periféricos tales como el USB y los relojes del sistema. Por otra parte, se encuentran librerías propias que presentan la siguiente funcionalidad:

- Stepper: contiene la configuración inicial y las funciones que permiten manejar el stepper.
- Peripherals_config: alberga las funciones que permiten inicializar ciertos periféricos como pueden ser los timers o la GPIO.
- PWM: tiene como misión disponer de las funciones de configuración y manejo de la señal PWM que permita controlar el motor servo.
- USART: contiene la configuración y la gestión de la USART que se empleará para comunicarse con el *LiDAR*.
- ComunicacionLiDAR: se encarga de manejar la transmisión y recepción de datos entre el *LiDAR* y la tarjeta de desarrollo.

- GestionUSB: contiene las funciones que permiten recibir y tratar las tramas llegadas desde la aplicación. Además, se encarga de preparar las tramas de datos que se enviarán a la aplicación con las medidas obtenidas.
- Stop: tiene funciones que se encargan de ejecutar las acciones necesarias para parar el sistema.
- GestionDatos: tiene como misión procesar los datos recibidos del *LiDAR* para que puedan ser enviados por USB.
- ConfigInicial: acoge las funciones que permiten inicializar el sistema tales como la configuración de los relojes y USB iniciales.

5.3 Configuración de periféricos

5.3.1 Introducción

En este apartado, se incluyen todas las configuraciones de periféricos que se utilizan en el desarrollo del software: GPIO, ADC, Timers, etc. En esta sección se indica los parámetros que caracterizan a cada uno de ellos y algunas características especiales.

5.3.2 GPIO

Sin duda, una de las partes más importantes de cualquier diseño son las entradas y salidas al microprocesador, en este caso, se necesitan habilitar pines para: driver A4988, USART, servo, *LiDAR* y sensores. Debido a la gran cantidad de pines a configurar en este apartado solo se incluye un ejemplo de configuración de un pin normal y de uno que tiene una funcionalidad especial tal y como se ilustra en el Código 5.1. En él se ilustra como un pin (3) de la GPIOB es habilitado como entrada, configurado sin pull-down ni pull-up y a una frecuencia de 50 MHz. A su vez, se observa en la segunda configuración como un pin (8) de la GPIOB es asignado a un propósito específico (TIM4) y por lo tanto su modo cambia.

```
GPIO_InitStructure.Pin = GPIO_Pin_3;
GPIO_InitStructure.Mode = GPIO_Mode_IN;
GPIO_InitStructure.Pull = GPIO_PuPd_NOPULL;
GPIO_InitStructure.Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);

GPIO_PinAFConfig(GPIOB, GPIO_PinSource8, GPIO_AF_TIM4);

GPIO_InitStruct.Pin = GPIO_Pin_8;
GPIO_InitStruct.Pull = GPIO_PuPd_NOPULL;
GPIO_InitStruct.Mode = GPIO_Mode_AF;
GPIO_InitStruct.Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStruct);
```

Código 5.1. Ejemplo de configuración de la GPIO.

5.3.3 USART

La comunicación entre el microcontrolador y el *LiDAR* se debe implementar bajo esta forma de comunicación debido a que es la que soporta el sistema láser. En el Código 5.2 se puede observar dicha configuración. Esta comunicación se puede resumir como: velocidad de transmisión de 115200 bps, 8 bits de tamaño de palabra, 1 bit de stop, sin paridad ni control de flujo y modo transmisión y recepción. Destacar que la comunicación tanto de transmisión como de recepción se hace mediante interrupciones.

```
USART_InitStruct.USART_BaudRate =115200;
USART_InitStruct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStruct.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
USART_InitStruct.USART_Parity = USART_Parity_No;
USART_InitStruct.USART_StopBits = USART_StopBits_1;
USART_InitStruct.USART_WordLength = USART_WordLength_8b;
USART_Init(USART2, &USART_InitStruct);
USART_Cmd(USART2, ENABLE);
```

Código 5.2. Configuración de la USART.

5.3.4 Timers

En este diseño se han usado varios timers para la realización de actividades correspondientes al movimiento del stepper, PWM (servo), para la verificación de la pérdida de datos y para establecer el tiempo entre datos enviados al *LiDAR*. Los dos timers que se han usado para propósito general son el timer 3 y el timer 5, los cuales son temporizadores de propósito general por lo que no presentan características esenciales diferentes a cualquier

otro tipo de timer mientras que el empleado para el PWM (timer 4) presenta una configuración distinta tal y como se explicó en el apartado del Motor servo Parallax. Como se observa en el Código 5.3 correspondiente al timer 3, para la configuración de un determinado timer se necesita determinar dos aspectos esenciales: la base de tiempos y el modo de trabajo de dicho periférico.

Centrándose en la base de tiempos si se especifican los parámetros de periodo, prescaler y sabiendo la frecuencia del reloj del timer se puede establecer cada cuanto debe "saltar" el timer. Teniendo en cuenta que el reloj del timer se encuentra configurado a 84 MHz, como se explicará en el apartado correspondiente, y sabiendo que este timer debe configurarse a 2 ms, los valores de periodo y prescaler se determinan como:

- Frecuencia Timer (Hz) = $\frac{\frac{TImer\_CLK}{Prescaler}}{Periodo}$
- Se establece a un valor fijo el prescaler, dentro de los rangos del tipo de variable, siendo este de 2000.
- Por lo tanto, sabiendo la frecuencia que se requiere del timer se puede calcular el periodo del mismo, Periodo= $\frac{\frac{TImer\_CLK}{Prescaler}}{Frecuencia\ timer}=\frac{\frac{84\ MHz}{2000}}{1000}=42$

```
TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
TIM_TimeBaseStructure.TIM_Period = 42;
TIM_TimeBaseStructure.TIM_Prescaler = 2000;
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Active;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 1;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
TIM_OC3Init(TIM3, &TIM_OCInitStructure);
TIM_ITConfig(TIM3, TIM_IT_CC3, ENABLE);
```

Código 5.3. Ejemplo de configuración del timer.

La forma de contar del temporizador es en modo ascendente y se selecciona el modo de comparación de salida, es decir, que cuando valor del contador sea igual al valor fijado (en este caso 2 ms) salta la rutina de servicio del timer correspondiente. Se usa este modo ya que es el adecuado para determinar cuándo ha expirado una temporización [39]. Para ello se debe configurar una captura de comparación de las disponibles, en este caso, la del canal 3.

Para finalizar se habilita la interrupción del timer y la señal del reloj del timer se conformará con polaridad baja y de 1 pulso.

Para el caso del timer 5 es completamente semejante a excepción, como es lógico, de su frecuencia de funcionamiento. Se encuentra configurado a una frecuencia de 1000 Hz, es decir, 1 ms. Siguiendo un proceso análogo al anterior se fija el prescaler a un valor 1000 y, por tanto, el valor del periodo será de 42.

5.3.5 NVIC

El NVIC es el controlador de interrupciones vectorizadas que presenta el microprocesador empleado en este diseño. Es necesario usarlo porque se deben habilitar las IRQs correspondientes a cada uno de los dos timers usados. De tal forma que para cada timer se debe configurar un canal del NVIC como se ve en el Código 5.4. En concreto, para el timer 3 el canal 3 (TIM3_IRQn). Se le asigna la prioridad 0 y subprioridad 2, es decir, una de la más alta (quién tiene la máxima es la USART).

```
NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x2;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

Código 5.4. Ejemplo de configuración del NVIC.

Otro aspecto a señalar es como deben ser las rutinas de servicios que atienden a estas interrupciones. Atendiendo al Código 5.5 se comprueba esto. Primero se debe verificar si el flag de interrupción correspondiente al timer, en este caso el 3, se encuentre a nivel alto lo que significa que se ha cumplido la comparación. En ese caso se ejecutan las sentencias que sean necesarias siendo, en este caso, incrementar un contador. A continuación, se debe limpiar el flag de interrupción para poder atender, cuando llegue, otra interrupción.

```
void TIM3_IRQHandler(void){
    if( TIM_GetITStatus(TIM3 , TIM_IT_CC3) != RESET ){
        countStepper++;
        TIM_ClearITPendingBit(TIM3 , TIM_FLAG_CC3);
    }
}
```

Código 5.5. Ejemplo de rutina de servicio.

5.3.6 RCC

El Control de reloj y reset (RCC) es el encargado de configurar tanto el reloj del sistema (SYSCLK) como el reloj que afecta a cada periférico en concreto. Lo primero que se debe tener claro, como se especificó en el apartado de la Arquitectura STM32F4, es la frecuencia de reloj de trabajo del sistema. Este valor es de 168 MHz, la cual no se ha cambiado a lo largo del diseño firmware. Sin embargo, lo que sí ha variado es la frecuencia de reloj que afectan a ciertos periféricos.

```
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                            |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV4;
```

Código 5.6. Configuración RCC.

Se puede comprobar, como se observa en el Código 5.6 la habilitación de los relojes para la GPIO se realiza a través del APB1 la cual se encuentra a la frecuencia de 42 MHz ya que se encuentra afectado por el prescaler del APB1(4). Por otra parte, también se necesita habilitar los relojes correspondientes a los timers 3, 5, PWM y USART2 que se encuentran conectados a través del mismo bus de periféricos. Mientras que la USART2 sigue la misma configuración de reloj que la GPIO para los timers y el PWM esta señal de reloj se duplica llegando a los 84 MHz finales.

5.4 Descripción general de funcionamiento

Como se ha explicado anteriormente, el firmware diseñado, por sí mismo, no puede ponerse en funcionamiento de forma automática, sino que debe esperar a la llegada de la trama necesaria, por USB, desde la aplicación de representación tal y como se ilustra en la Figura 5.2.

Cuando esto ocurre, primero se comprueba si ha llegado la cabecera (0xA5) y, en caso afirmativo, se comprueba si se ha recibido el dato de start (0x41) o de stop (0x52); en el resto de casos que se pudieran dar el sistema permanecería a la espera. En caso de que llegue el start, se comienza la etapa de comunicación *LiDAR* que consiste en el envío de

tramas de comunicación a éste mediante USART y la posterior respuesta del RPLIDAR y su procesamiento en la tarjeta de desarrollo.

En un primer momento se envía una solicitud del tipo "get_health" que le pide al *LiDAR* que le devuelva el estado actual del dispositivo. En caso de que todo llegue de forma correcta se comprueba si es la primera vez que esto ocurre y, en caso afirmativo, se procede a transmitir al sistema láser una solicitud del tipo "get_freq" que devuelve la frecuencia de muestreo del dispositivo tal y como se ilustra en la Figura 5.2. Cuando esto se recibe de forma correcta ya se puede determinar que el *LiDAR* está en el estado adecuado para comenzar a trabajar.

Sin embargo, para ello es necesario colocar al sistema electromecánico en las posiciones iniciales de trabajo (Figura 4.4, Figura 4.5 y Figura 4.6). De esta forma, primero se debe comprobar con la ayuda del sensor que se encuentra en la base si el stepper se encuentra en la posición inicial. En caso negativo se procede a realizar el movimiento del stepper (sentido horario) hasta que se produzca un campo magnético entre el imán y el sensor que indique que se encuentra en la posición inicial. De forma análoga, a continuación, se debe comprobar la posición inicial del servo y cuando la base del *LiDAR* no se encuentra en dicha posición moverlo (sentido antihorario) hasta que llegue a la posición marcada por el sensor.

De esta forma, ya el sistema se encuentra en las posiciones de partida para que se comience la adquisición de datos por lo que desde la tarjeta de desarrollo se envía al *LiDAR* la trama "get scan" que indica al sistema que debe comenzar a medir distancias. Cuando esto ocurre, el *LiDAR* comienza a enviar los datos medidas, siendo los más importantes el ángulo, la distancia y el flag de comienzo de nuevo escaneo de 360º. Como al comienzo de la toma de medidas se desconoce la posición desde donde se empieza los datos recibidos hasta la llegada del nuevo flag de comienzo deben ser descartadas. De esta forma, en una segunda vuelta se consiguen de forma completa un escáner de 360 º. A su vez, cada vez que se disponga de una medida de un ángulo del *LiDAR* (5 bytes completos recibidos) y se tengan todos los datos correspondientes a la misma se prepara la trama de comunicación necesaria para que se envíe por USB a la aplicación de representación. Entre otros datos que se envían,

se consigue que para cada medida existan un trío de datos: azimut (stepper), ángulo de elevación y distancia (*LiDAR*).

Cuando este flag de nueva vuelta llegue por tercera vez se considera que se ha completado las medidas en una determinada posición horizontal (se halla por software a partir la posición del stepper y la resolución de su movimiento) que corresponde al azimut en un sistema de coordenadas esféricas. Cuando esto se logra, se envía una trama de stop al sistema láser que para la adquisición de datos y el movimiento del motor del RPLIDAR hasta que sea necesario. A continuación, se debe mover el stepper un determinado ángulo (0,45º) cuya dirección de giro vendrá determinada por la posición global del mismo. Si se ha llegado a dar una vuelta entera (360º) se debe cambiar el sentido del movimiento hasta la llegada del siguiente giro completo. Posteriormente, se vuelve a comenzar el ciclo descrito anteriormente, pero sin enviar la trama de tipo "get freq" ni mover los motores; solo es necesario verificar el estado del *LiDAR* y, a continuación, enviar de nuevo la trama "get scan".

Este procedimiento seguirá funcionando de forma ininterrumpida hasta que, desde la aplicación de representación, por USB, llegue la trama de stop. Cuando esto ocurre, como primer paso; se debe parar la adquisición de datos enviando la trama de stop correspondiente. A continuación, se debe verificar si el servo se encuentra en la posición final; en caso negativo moverlo hasta que llegue a dicha y, posteriormente, proceder a resetear el firmware de tal forma que se encuentre listo para que vuelva a funcionar cuando llegue la siguiente trama de start desde la aplicación.

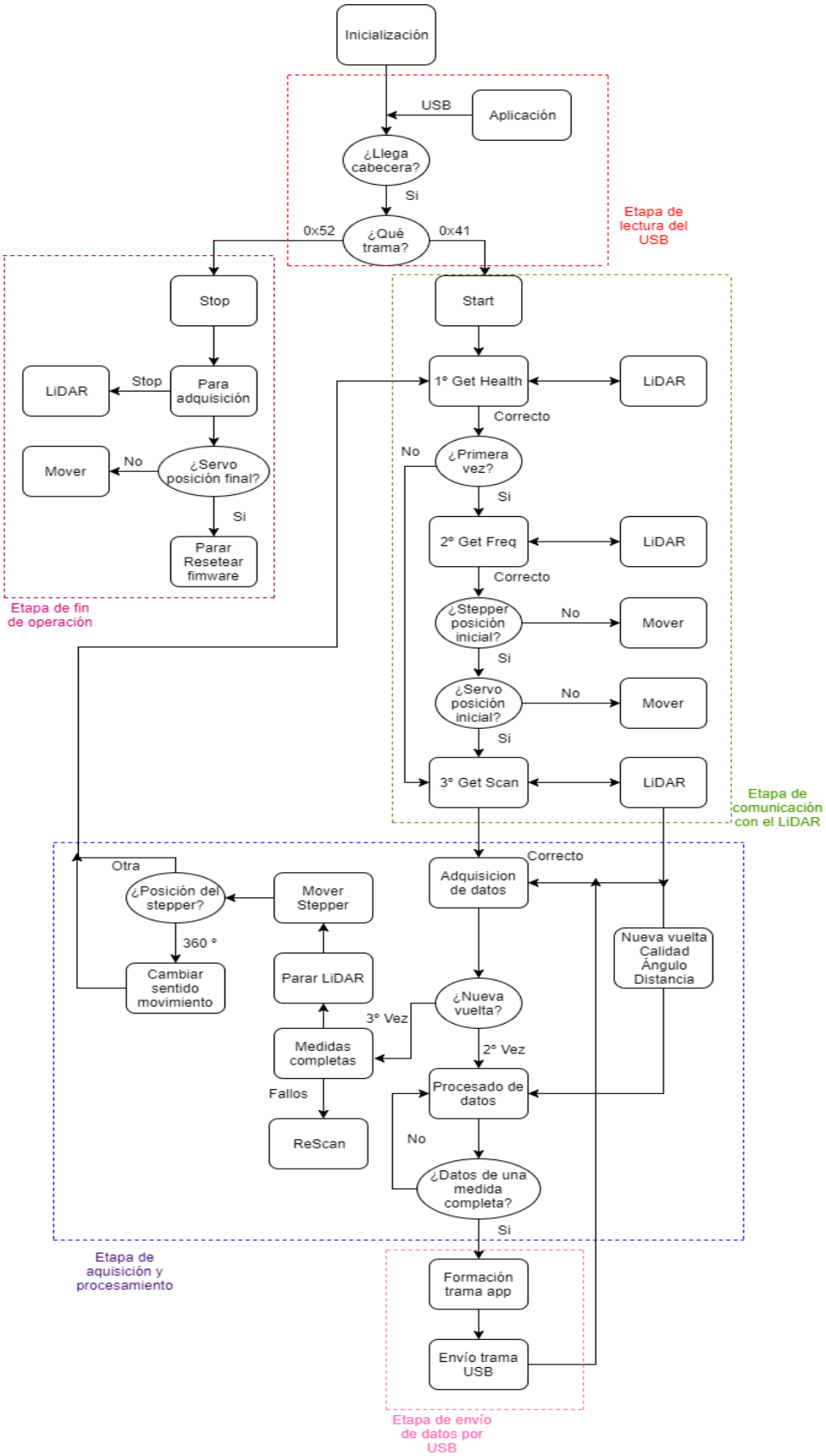


Figura 5.2. Diagrama general de funcionamiento del firmware.

5.5 Programa principal

La función main() se divide en dos partes, la primera corresponde a la inicialización de periféricos y ciertas funciones que sólo es necesario que se efectúen una vez y la segunda dentro del bucle infinito donde se encuentra el resto del código que se ejecuta continuamente de manera secuencial.

```
while (1)
{
 lecturaUSB();
 USARTstate();
 reSend();
 stopLIDAR();

 if((dato==0x41)||start){
    start=1;
    stop=0;
    reset_servo=1;
    secuenciaComienzo();
    capturaDatos();
    envioUSB();
 }
 if((dato==0x52)||stop){
    start=0;
    stop=1;

    stopReset();
    resetServo();
 }
}
```

Código 5.7. Programa principal.

Lo que se ejecuta en el bucle infinito tal y como se ilustra en el Código 5.7 corresponde a la lectura del USB y la comprobación de qué tipo de trama es y, a continuación, ejecutar una serie de funciones en función si dicha trama es de start o de stop. Además, existe un conjunto de funciones que serán llamadas de forma independiente, cuando sea necesario, en este programa principal siendo estas las correspondientes al stop y a la habilitación de la USART tal y como se ilustra en la Figura 5.3. En las siguientes iteraciones, posteriores a la llegada de una trama, se siguen ejecutando las funciones correspondientes a la trama recibida, en caso de start, se ejecuta la etapa de comunicación *LiDAR*, la etapa de adquisición y procesamiento y la etapa de envío de datos por USB. A su vez, vez cuando se recibe un stop se pone en funcionamiento la etapa de finalización.

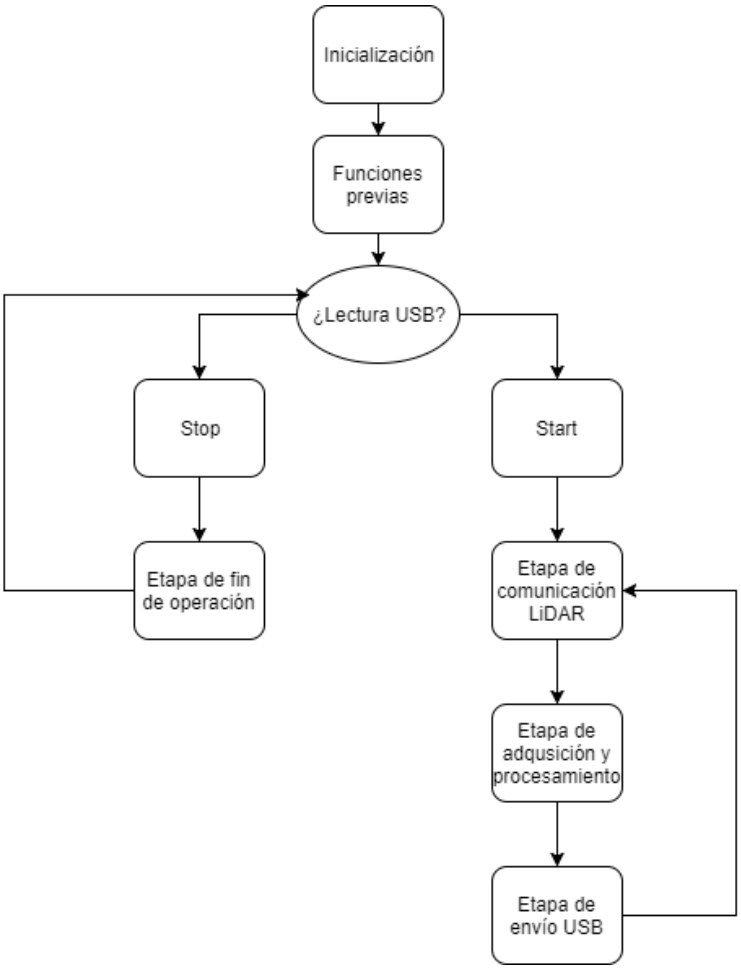


Figura 5.3. Diagrama de flujo del programa principal.

5.6 Etapa de lectura del USB

Esta etapa comprende la lectura y gestión de los datos procedentes de la aplicación enviadas a la tarjeta de desarrollo mediante USB. Primero se comprueba si existe algún dato en la cola del buffer de recepción. En caso afirmativo se comprueba, si el primer dato que ha llegado es la cabecera (0xA5). Si esto es así, se comprueba si el siguiente dato corresponde a alguno de los relacionados con la secuencia de start (0x41) o con la secuencia de stop (0x52) habilitando un flag dependiendo del caso como se observa e n la Figura 5.4.

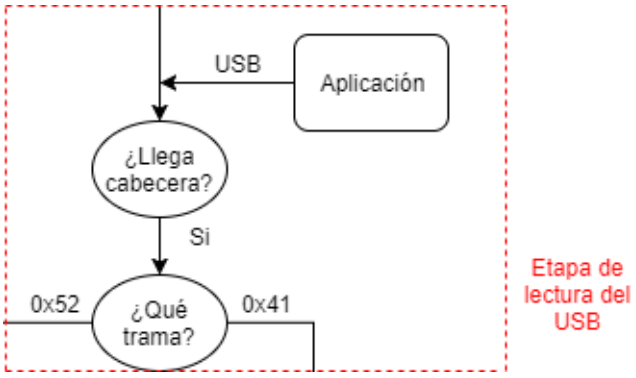


Figura 5.4. Diagrama de flujo de la etapa de lectura del USB.

Tal y como se observa en el Código 5.8 para que se considere como trama válida primero debe llegar la cabecera en caso contrario se desecha. De igual forma, cuando el segundo dato recibido no coincide con los descritos anteriormente se procede al reinicio de la lectura USB esperando la cabecera de nuevo.

```
void lecturaUSB(){
  //Lectura del USB (entrada)
  if(RX_FIFO_AVAIL){
     dato_1=(GetChar()&0x0FF);
     //Start: 0x41 (A)
     //Stop: 0x52 (R)
     if((dato_1==0x5F)&&(flag_cab_USB==0)){
        flag_cab_USB=1;
     }else if ((dato_1!=0x5F)&&(flag_cab_USB==1)){
        flag_cab_USB=0;
        dato=dato_1;
     }else{
        flag_cab_USB=0;
     }
  }
}
```

Código 5.8. Función lecturaUSB.

5.7 Etapa de comunicación LiDAR

Para establecer comunicación con el RPLIDAR A1 primero se debe analizar el funcionamiento de su protocolo de comunicación. Lo primero que se debe destacar es que para que se produzca el envío de datos desde el *LiDAR* la tarjeta de desarrollo debe comenzar la comunicación. Esta se caracteriza por seguir una transmisión de tipo USART con una tasa binaria de 115200 bps, 8 bits de transmisión, sin paridad y 1 bit de stop.

El RPLIDAR A1 dependiendo de la trama de comunicación que se envíe presenta tres modos de funcionamiento: solicitud-única respuesta, solicitud-múltiple respuesta o solicitud-sin respuesta [53]. El modo solicitud-única respuesta se caracteriza, tal y como se observa en la Figura 5.5, por enviar a la tarjeta de desarrollo una solicitud y, pasado un tiempo, recibir una respuesta del láser pudiendo, a continuación, enviar otra petición.

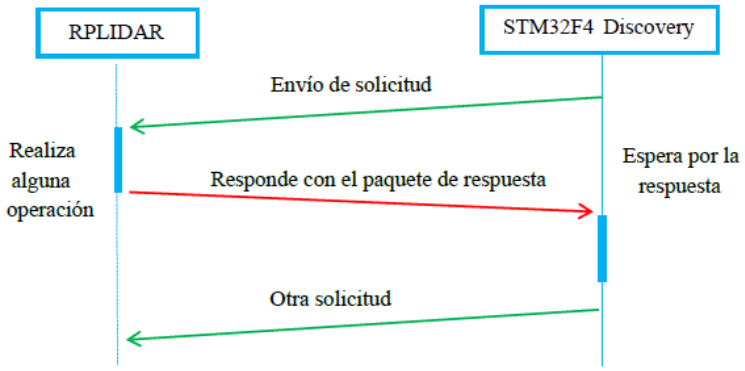


Figura 5.5. Modo solicitud-única respuesta.

Por otro lado, el modo solicitud-múltiple respuesta se utiliza cuando se pide al RPLIAR que realice la operación de exploración (scan). Con ello se consigue que continuamente se envían los paquetes de datos; sin embargo, previo al inicio de la comunicación se envía la trama de respuesta como se observa en la Figura 5.6. El host solo necesita enviar un único paquete de solicitud para recibir datos continuamente. El microcontrolador puede interrumpir la adquisición enviando una solicitud de stop o cualquier otro paquete de solicitud. En el caso de que se envíe una petición alternativa al stop esta se atenderá y, posteriormente, se continuará en modo scan.

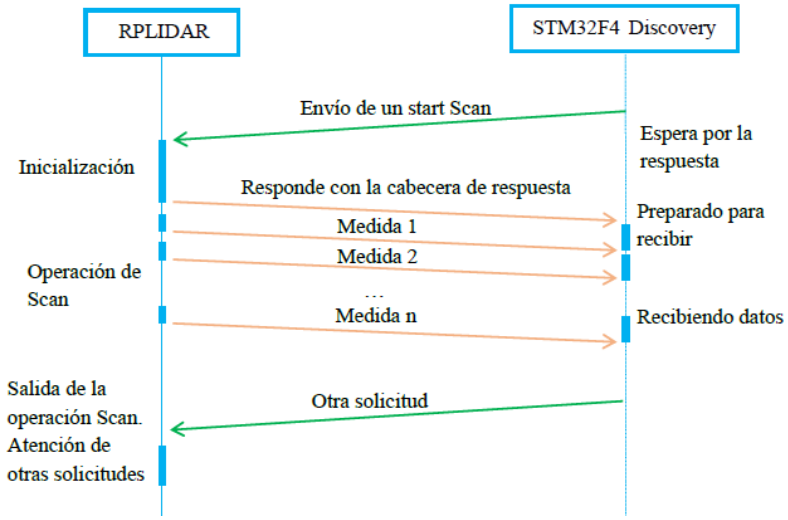


Figura 5.6. Modo solicitud-múltiple respuesta.

Por último, el modo solicitud-sin respuesta es el característico para solicitudes como "stop" y "reset" ya que no hay necesidad de responder según estima RPLIDAR tal y como se representa en la Figura 5.7. La tarjeta de desarrollo deberá esperar un periodo de tiempo antes de enviar otra solicitud, ya que RPLIDAR necesita procesar la operación de solicitud. De lo contrario, la petición puede ser desechada por la pila de protocolos.

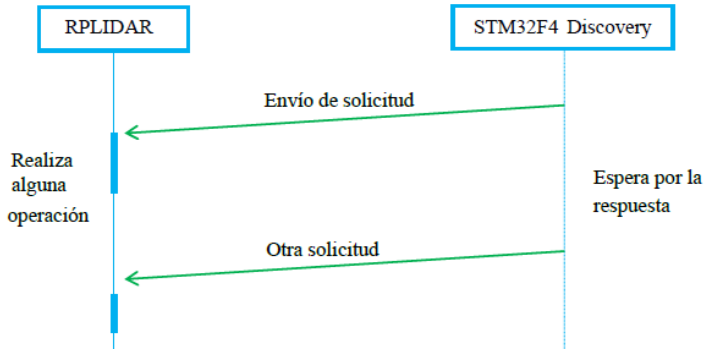


Figura 5.7. Modo solicitud-sin respuesta.

En la Figura 5.8 se ilustra el conjunto de las solicitudes que se pueden mandar al RPLIDAR y el tipo de respuesta que presenta cada una de ellas con una pequeña descripción de su funcionalidad [53]. En este TFM se usarán las tramas de "stop", "scan", "get_health" y "get_samplerate" que se irán detallando a continuación. Todas las solicitudes enviadas deberán llevar como cabecera el indicador 0xA5.

Request Name	Value	Payload	Response Mode	RPLIDAR Operation
STOP	0x25	N/A	No response	Exit the current state and enter the idle state
RESET	0x40	N/A		Reset(reboot) the RPLIDAR core
SCAN	0x20	N/A	Multiple response	Enter the scanning state
EXPRESS_SCAN	0x82	YES		Enter the scanning state and working at the highest speed
FORCE_SCAN	0x21	N/A		Enter the scanning state and force data output without checking rotation speed
GET_INFO	0x50	N/A	Single response	Send out the device info (e.g. serial number)
GET_HEALTH	0x52	N/A		Send out the device health info
GET_SAMPLERATE	0x59	N/A		Send out single sampling time

Figura 5.8. Solicitudes LiDAR.

Cuando se recibe la petición de "stop" (0x25) RPLIDAR sale del estado de escaneo. Esto implica que el diodo láser y el sistema de medición se desactivarán y el sistema pasará a estar en estado de reposo. Esta solicitud se ignorará cuando RPLIDAR esté en estado inactivo o de parada por fallo. Dado que RPLIDAR no enviará ningún paquete de respuesta para esta solicitud, el sistema emisor debe esperar al menos 1 milisegundo (ms) antes de enviar otra solicitud tal y como se ilustra en la Figura 5.9.

Figura 5.9. Trama de stop.

También se puede enviar la solicitud "get_health" (0x52) para consultar el estado del RPLIDAR. Si el RPLIDAR ha entrado en el estado de parada por protección causado por un fallo de hardware, se enviará el código de error asociado al fallo tal y como se ilustra en la Figura 5.10. De esta forma, los datos de respuesta lo conforman 3 bytes; el primer byte indica el estado del sistema (0 bueno, 1 precaución, pero puede seguir trabajando, 2 error). Cuando ocurre un error en los otros dos bytes se incluye el código del mismo.

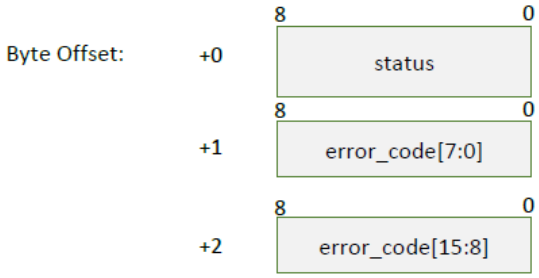


Figura 5.10. Datos de la trama get-health.

A su vez, la trama de respuesta que se envía a la tarjeta de desarrollo antes de los datos de respuesta se puede observar en la Figura 5.11 que destaca por tener una longitud de 7 bytes que es común al resto de cabeceras de respuesta.

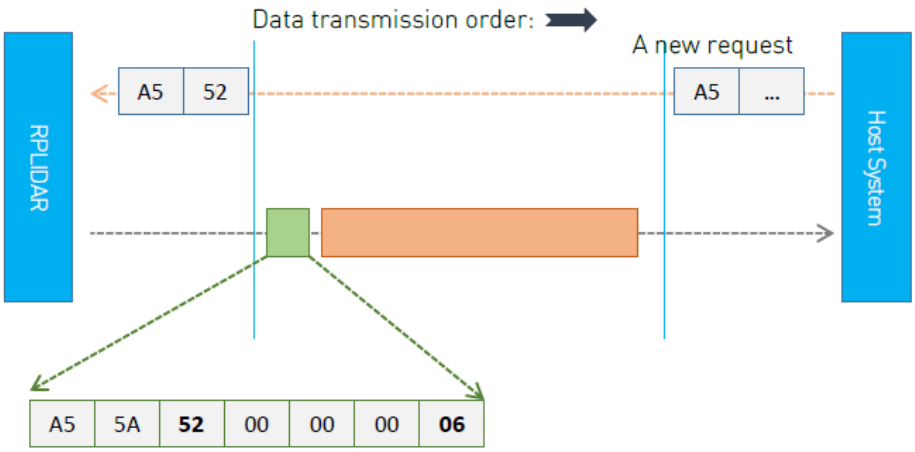


Figura 5.11. Descripción de la trama get-health.

Cuando se envía la solicitud “get_samplerate” (0x59) se desea consultar la frecuencia de muestreo del sistema, es decir, el tiempo de duración desde que el emisor emite el haz hasta que el rebote llegue al fotosensor. En la Figura 5.12 se ilustra la forma de la trama de respuesta que es de 7 bytes como en el caso anterior.

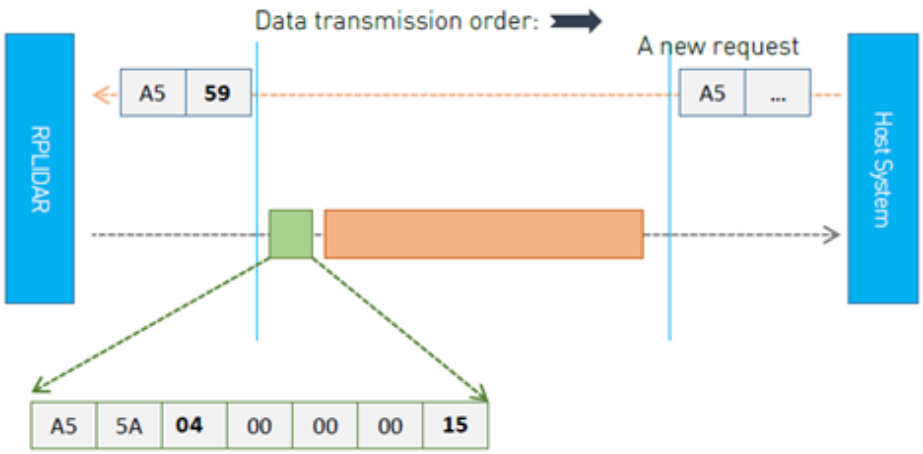


Figura 5.12. Descripción de la trama get_samplereate.

Los datos de respuesta (4 bytes) que se obtienen se pueden observar en la Figura 5.13 donde los dos primeros bytes indica el tiempo de muestreo en el modo scan normal y los otros dos en el modo scan express. Ambas medidas se representan en microsegundos.

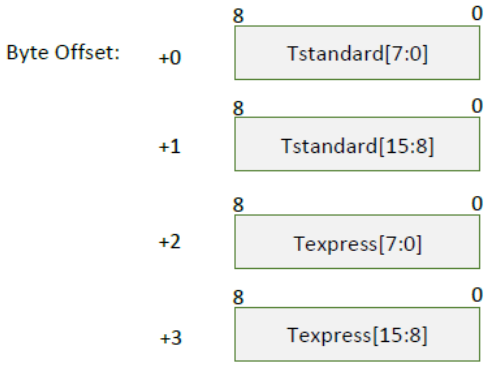


Figura 5.13. Datos de la trama get_samplerate.

Por último, para realizar un escaneado se debe enviar la trama “scan” (0x20) que a no ser que se encuentre en parada de seguridad, deberá entrar en este modo de funcionamiento. Tal y como se ilustra en la Figura 5.14 la cabecera de recepción de esta solicitud es también de 7 bytes. A su vez destacar que cada paquete de datos lo forman 5 bytes que se detallará en la etapa de adquisición y procesamiento.

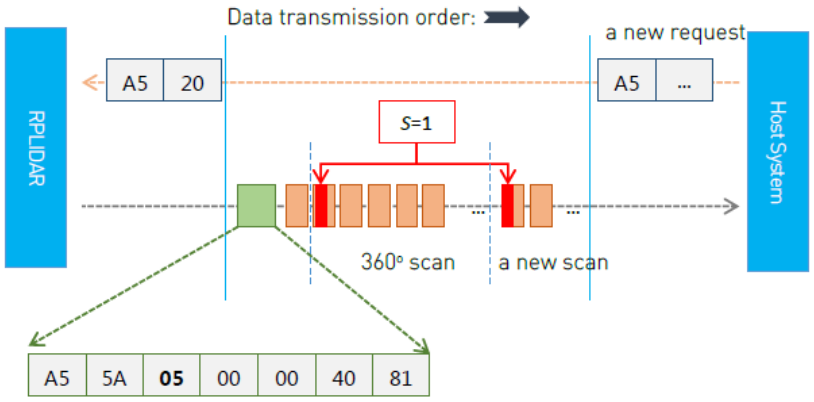


Figura 5.14. Descripción de la trama get-scan.

Como paso previo a comentar la secuencia de transmisión que se sigue en esta etapa es necesario explicar el funcionamiento a nivel software de la comunicación USART (transmisión y recepción) desde el punto de vista del firmware desarrollado ya que es una etapa común que debe seguir cualquier trama que se envíe.

En la Figura 5.15 se ilustra en un diagrama de flujo la secuencia de pasos que se siguen tanto a la hora de enviar como recibir datos al sistema láser. Como se indicó al comienzo de este capítulo, las comunicaciones USART se encuentran controladas por interrupciones por lo que es necesario disponer de una rutina de servicio que permita atender tanto la transmisión como recepción de datos.

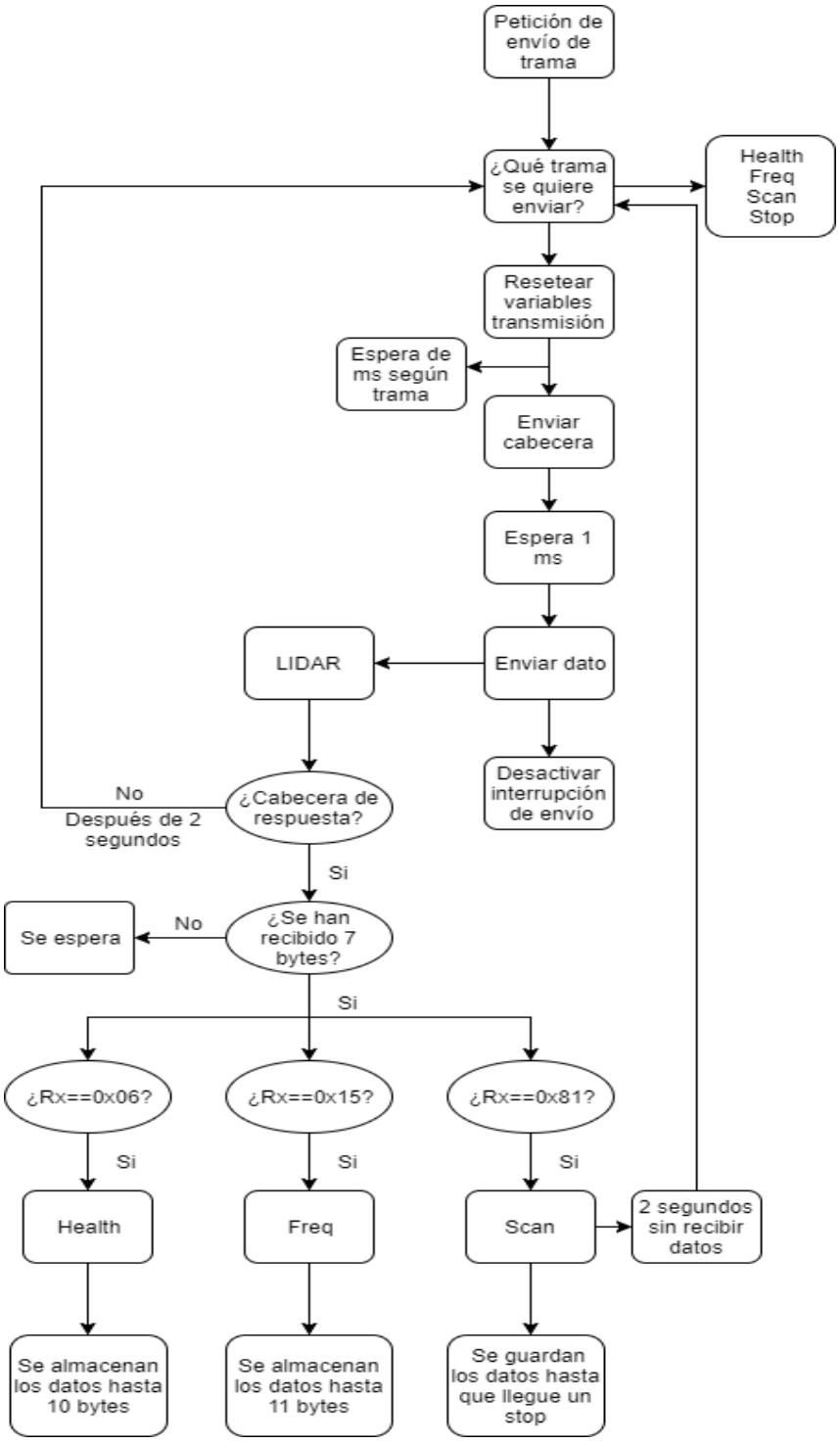


Figura 5.15. Diagrama sobre la comunicación con el LiDAR.

Cuando comienza esta etapa lo primero que se debe ejecutar es el envío de una solicitud. Para ello, primero se deben resetear las variables de funcionamiento y a, continuación esperar un determinado espacio de tiempo hasta enviar la trama. Esto es

necesario debido a que empíricamente se ha comprobado que debe de haber cierto espacio entre una trama y otra. Este valor se establece de media a 10 ms.

A continuación, ya se habilita la transmisión USART por lo que se debe enviar tanto cabecera como dato tal y como se observa en el Código 5.9. Cómo se ha indicado anteriormente la cabecera es común a todas las tramas siendo esta 0xA5. Además, se comprueba que entre cabecera y dato debe existir un espacio de tiempo para que el *LiDAR* pueda procesarlo correctamente. Este gag de tiempo se establece en 1 ms para todas las tramas enviadas. El dato que se envía dependerá de la posición de la secuencia en la que se encuentre: 0 (health), 1 (freq), 2 (scan) y 3 (stop). Posteriormente al enviar el dato se activa un flag que permite en el programa principal deshabilitar la interrupción de transmisión de la USART. Además, para la comprobación de los datos que se mandan estos se guardan tramas en un buffer por si fuera necesario realizar un depurado. Además, se inicializa la cuenta para establecer si se no ha recibido respuesta en un tiempo determinado (2 segundos). Por último, como en caso de que se envíe una trama de stop no hay respuesta por prate del RPLIDAR se opta por cambiar de paso de la secuencia en este punto del código.

```
if(count_USART==1){//Espera 1 ms para enviar el dato después de cabecera
   count_id=2;
   count_USART=0;
}
if(count_id==0){
   TxCabecera=0xA5;
   if(sec==0){
      TxId=0x52;
   }else if (sec==1){
      TxId=0x59;
   }else if (sec==2){
      TxId=0x20;
   }else if (sec==3){
      TxId=0x25;
   }
   USART_SendData(USART2,TxCabecera);
   count_id=1;
   buffer_Tx[pos_buffer_Tx]=TxCabecera;
   pos_buffer_Tx++;
}else if(count_id==2){
   count_id++;
   USART_SendData(USART2,TxId);
   tim_noenvio=0;
   buffer_Tx[pos_buffer_Tx]=TxId;
   pos_buffer_Tx++;
   if(sec==3){
      sec=0;
   }
}
if(pos_buffer_Tx==199){
```

```
    pos_buffer_Tx=0;
}
```

Código 5.9. Interrupción USART de transmisión.

A partir de que se envía una trama de comunicación hacia el RPLIDAR el programa se encentra esperando a que comience la llegada de datos. Estos datos deben comenzar con la trama de respuesta o de cabecera que como se vio anteriormente presenta una parte común la cabecera como se observa en el Código 5.10. A partir de su llegada se debe contar la llegada de los siguientes 6 bytes para poder determinar cuál es el séptimo byte que servirá como indicador de qué tipo de dato se está recibiendo. Si lo que se recibe es un 0x06 es una trama de tipo "health", si se recibe un 0x15 es una respuesta de tipo "freq" y, por último, si llega un 0x81 es una trama de tipo "scan".

```
Rx=USART_ReceiveData(USART2);

if(Rx==0xA5){
   count_cabecera=1;
   count_bytes++;
   tim_noenvio=0;

}else if ((((Rx==0x06)&&(sec==0))||flag_health)&&(count_cabecera==1)&&(count_bytes>=7
&& count_bytes<10)){
   if(flag_health){
      buffer_health[pos_buffer_health]=Rx;
      pos_buffer_health++;
   }
   flag_health=1;
}else if ((((Rx==0x15)&&(sec==1))||flag_freq)&&(count_cabecera==1)&&(count_bytes>=7 &&
count_bytes<11)){
   if(flag_freq){
      buffer_freq[pos_buffer_freq]=Rx;
      os_buffer_freq++;
   }
   flag_freq=1;
}
if(((Rx==0x81)&&(sec==2))||flag_cont_scan){
   if(flag_cont_scan){
      flag_scan=1;
      buffer_datos[pos_buffer_datos]=Rx;
      pos_buffer_datos++;
      if(pos_buffer_datos==2000){
         pos_buffer_datos=0;
      }
   }
   flag_cont_scan=1;
   flag_health=0;
   pos_buffer_health=0;
   flag_freq=0;
   pos_buffer_freq=0;
}
```

Código 5.10. Interrupción USART de recepción.

A partir de esto los siguientes bytes que se envíen, posterior a las cabeceras, se debe almacenar en unos bufferes (debido a que la USART funciona por interrupciones) para que cuando continúe la ejecución normal del código se puedan comprobar los datos recibidos dependiendo de qué solicitud se haya mandado. Por ejemplo, en el caso que se ilustra en el Código 5.11para el envío y recepción de una solicitud de tipo "health" se comprueba cómo los 3 primeros bytes almacenados en el buffer correspondiente tienen el valor 0 que implica que todo se encuentra correcto. Cuando esto ocurre se activa el paso de la siguiente trama a enviar siguiendo un proceso análogo al explicado. La secuencia completa se puede revisar en el Anexo 2. Código 1.

```
if(sec==0){ // 1º Get_Health
 if(flag_count_reset==0){
    count_Reset=0;
    flag_count_reset++;
    count_bytes=0;
    count_cabecera=0;
    tim_noenvio=0;
 }else if ((buffer_health[0]==0x00)&&(count_bytes==9)&&(flag_health)){
    count_bytes=0;
    flag_health=0;
    pos_buffer_health=0;
    count_cabecera=0;
    sec++;
    countStep=0;
    if(stop_stepper==1){//Cuando ya ha escaneado directo a SCAN
       stop_stepper=0;
       sec=2;
       flag_count_reset=2;
       flag_USART=2;
    }
 }else if (count_Reset==10){
    if(flag_USART==0){
       flag_int=1;
       flag_USART++;
       count_id=0;
    }
    count_USART=0;
    count_bytes=0;
    count_cabecera=0;
 }
}
```

Código 5.11. Ejemplo de secuencia de transmisión.

Por último, cuando se verifica que pasado el tiempo de 2 segundos establecido anteriormente se comprueba que no se hayan recibido nuevos datos y verificando si anteriormente se había mandado una cabecera se reenvía la trama de comunicación solicitada siguiendo la función *resend* que se observa en el Código 5.12. Además, esta función también

se usará cuando se compruebe que ciertas medidas obtenidas por el láser no son correctas y deban volver a realizarse.

```
void reSend(){
  if(((buffer_Tx[pos_buffer_Tx-1]==0xA5)&&(tim_noenvio>2000))||(flag_reScan==1)){
     if(TxId==0x52){
        flag_count_reset=0;
        stop_stepper=1;
        sec=0;
        flag_USART=0;
     }else if(TxId==0x59){
        flag_count_reset=1;
        sec=1;
        flag_USART=1;
     }else if(TxId==0x20){
        flag_count_reset=2;
        sec=2;
        flag_USART=2;
        flag_reScan=0;
     }
     tim_noenvio=0;
  }
}
```

Código 5.12. Función reSend.

De forma global, el funcionamiento de la etapa de comunicación *LiDAR* se puede observar en la Figura 5.16 y se basa en realizar peticiones desde la tarjeta de desarrollo hacia el sistema láser de forma previa a llevar a la adquisición de datos. Esto se hace con dos objetivos, primero asegurar el funcionamiento correcto del sistema en cada medición y a la vez se usa para en un estadio inicial realizar los movimientos de los motores cuando sea necesario.

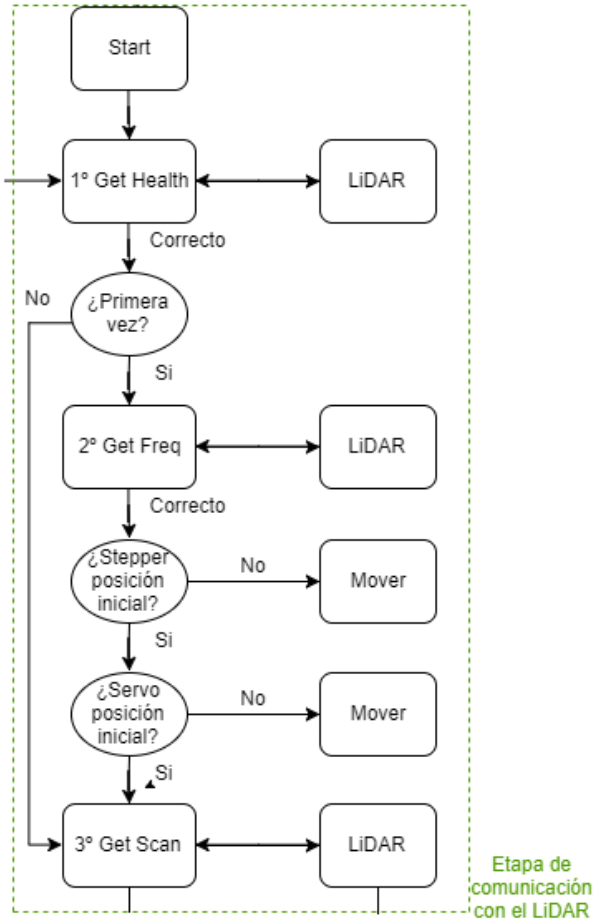


Figura 5.16. Diagrama de la etapa de comunicación con el LiDAR.

Siguiendo el proceso explicado anteriormente primero se debe enviar una trama "health". Cuando se reciban los datos correctos, enviar una trama de tipo "freq" y, a partir de que se reciba la confirmación se procede a mover los motores tanto servo como stepper si fuera necesario tal y como se explica en el capítulo 3. Por último, cuando el stepper y el servo se encuentren en la posición correcta para iniciar el escaneado se comienza enviando la trama correspondiente (scan). En un segundo escaneado de las medidas ya no será necesario mover los motores por lo que el paso de 2 (freq) de la secuencia se saltará. El proceso de manejo de estos datos corresponde ya a la etapa de adquisición y procesamiento de datos.

5.8 Etapa de adquisición y procesamiento de datos

En esta etapa tal y como se describe en la Figura 5.17 corresponde a la etapa de adquisición y procesamiento que se encarga de recoger los datos recibidos de cada medida, procesarlos para su posterior envío por USB y realizar los movimientos necesarios en el stepper para continuar cogiendo medidas en otras posiciones con el objetivo de lograr en el eje horizontal (azimut) un movimiento de 360 grados. Debido a la forma en que se envían los datos las primeras medidas que se obtengan hasta la llegada del primer flag de nueva vuelta deben ser descartadas y cuando llegue la segunda vez se considerará que se han completado las medidas (a no ser que se detecten fallos y se deba reenviar al RPLIDAR la trama de "scan" de nuevo). Cuando esto ocurra se parará la adquisición de datos (mediante la trama de stop) y se desactivará desde el firmware el movimiento del motor del RPLIDAR, se moverá el stepper lo que se haya estipulado y comenzará la secuencia de adquisición de datos de nuevo.

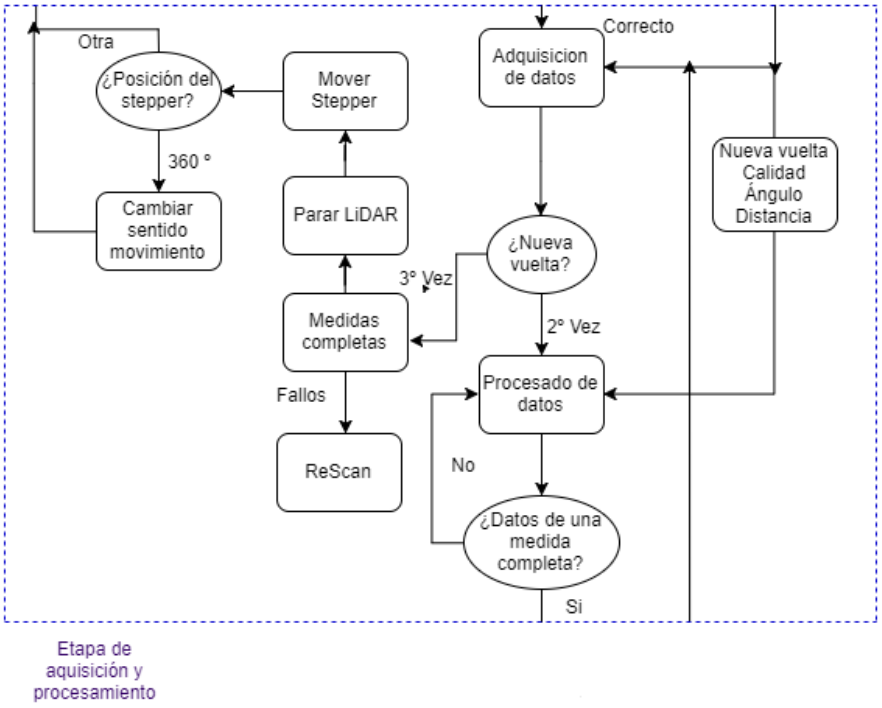


Figura 5.17. Diagrama de la etapa de adquisición y procesamiento.

Ya que esta etapa se encarga del procesamiento de los datos después del envío de una trama de tipo "scan" es necesario conocer el conjunto de datos que envía el RPLIDAR en

cada medida. Cada medida láser completa (ángulo y distancia) se envía en 5 bytes tal y como se representa en la Figura 5.18. En ella se puede observar como en el primer byte se envían la calidad y dos flags (S y $\bar{S}$). El flag S y $\bar{S}$ correspondientes al primer y segundo bit respectivamente del primer byte indican que a partir de esta medida comenzarán a llegar datos de un escaneo de 360 grados completo (S=1) [53]. Desde el tercer bit hasta el final de este byte se envía la calidad de la medida que representa lo buena que es la medida (en caso de que no haya medida de distancia este valor será 0). El segundo byte alberga los bits menos significativos del ángulo [6:0] y un flag C (en el primer bit) que puede servir como comprobación de trama ya que se encuentra constantemente a 1. A continuación, el tercer byte al completo alberga los bits más significativos del ángulo [14:7] de 0 a 360º y, por último, el cuarto y el quinto byte albergan los bits menos [7:0] y más significativos de la distancia [15:8] respectivamente que se representa en mm.

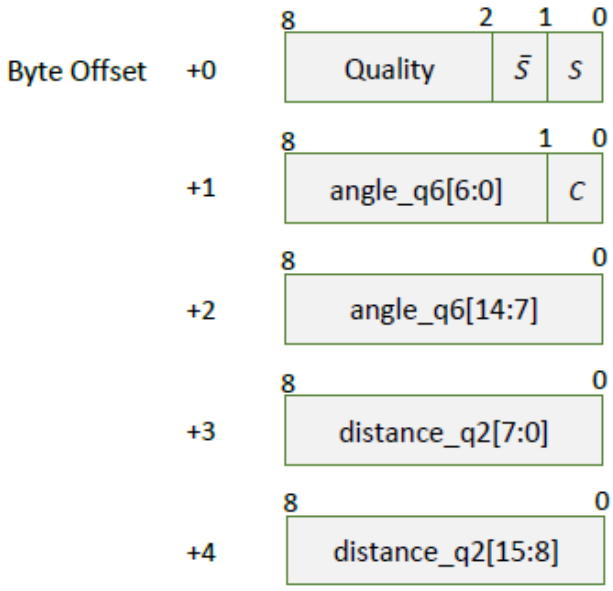


Figura 5.18. Datos de una medida recibida del LiDAR.

Es importante destacar que según expresa el fabricante del RPLIDAR las medidas tanto de distancias como de ángulo vienen expresada en formato de punto fijo, en concreto, en formato Q. Este es un tipo de formato de representación de un número decimal como entero para poder ser enviado por un bus de comunicaciones. En concreto, este formato se basa en codificar el número con una serie de números fraccionales. En este caso, para el ángulo se usa un formato Q6 y para la distancia un formato Q2. La forma que se usa para

transformar este dato a float consiste en pasar este dato hexadecimal a su entero correspondiente y posteriormente multiplicarlo por la potencia $2^{-números\ fraccionales}$ [54]. A continuación, se describe un ejemplo a partir del dato de un ángulo:

$$Dato\ hexadecimal\ recibido = 4B00$$

$$Dato\ entero = 19200$$

$$Dato\ float = 19200 * 2^{-6} = 300\ grados$$

Para conseguir almacenar los datos de cada medida es necesario disponer de una función que permita guardar cada uno de los valores y en la secuencia que son recibidos para poder tratarlos de forma adecuada tal y como se ilustra en la función *capturaDatos* en el Código 5.13. Cuando llega el flag de llegada de dato (desde la interrupción de la USART) es necesario determinar que posición ocupa (de 0 a 4) para poder saber cómo tratarlo.

```
void capturaDatos(){
if(((flag_scan)||(move_stepper))){ //Recepción
  if((contar_dato==0)||(move_stepper)){ //Coger S y Quality
    inicializacion();
    verificMove();
    calidadAzimut();
    }else if(contar_dato==1||contar_dato==2){ //Coger C y angle completo
        cAnguloCompl();
    }else if(contar_dato==3||contar_dato==4){ //Coger distancia
        distancia();
    }
    buffersReset();

  }
}
```

Código 5.13. Función capturaDatos.

En el momento que llega el primer byte de una nueva medida tal y como indica el código anterior se ejecutan una serie de funciones. La primera de ella *inicialización*, que se ilustra en el Código 5.14, sirve para recoger los datos recibidos por la USART (se usa un buffer intermedio para evitar sobreescrituras cuando lleguen nuevos datos sin haber atendido a los anteriores) y guardar los flags que indican nueva vuelta.

```
void inicializacion(){
  if(move_stepper==0){
    Rx_dato=buffer_datos[pos_buffer_datos_1];
    pos_buffer_datos_1++;
    if(pos_buffer_datos_1==2000){
```

```
        pos_buffer_datos_1=0;
    }
    dato_S=Rx_dato&0x01;
    dato_S_neg=Rx_dato&0x02;
  }
}
```

Código 5.14. Función inicialización.

A continuación, debe ejecutarse la función *verificMove*, cuyo funcionamiento se encuentran descrito en el Anexo 2. Código 2, que como principal misión tiene controlar el estado del flag S. De esta forma, se puede indicar cuando actuar, en este caso hasta que no llegue el flag S a nivel alto no se guardará ningún dato. Sin embargo, cuando se comience la primera vuelta completa se habilitará el almacenamiento de los datos recibidos correspondientes a cada medida. A su vez, cuando se vaya a comenzar la segunda vuelta completa y no se han producido fallos en las medidas se procede a parar la adquisición reseteando las variables correspondientes y procediendo al movimiento del stepper un paso (establecido previamente en 0,45º). Además, se debe comprobar, a través del sensor del stepper, si se ha alcanzado una vuelta horizontal completa (azimut) y, en caso afirmativo, se procederá a girar en el sentido contrario. Posteriormente, ya se podrá comenzar secuencia de nuevo con el envío de un "get_health". Sin embargo, si se detecta la presencia de fallo en las medidas debido a que han llegado muy pocas (menor que 4, por ejemplo) o los datos de ángulos no comienzan a llegar desde las cercanías del ángulo 0º se procede al reenvío de la trama "scan" al RPLIDAR.

Cuando en la recogida de medidas en una vuelta del *LiDAR* (vuelta vertical) se detecta que ya se comienza un escaneo de 360º se procede a guardar los datos recibidos empezando por el de calidad tal y como se indica en la función *calidadAzimut* que se ilustra en el Código 5.15. Con ayuda de esta función se guardan varios tipos de datos. Primero se almacena la calidad que es un dato recibido por RPLIDAR; pero también se almacenan otros como el número de medida (referida a los datos recibidos en una vuelta vertical del *LiDAR*) y el ángulo horizontal (azimut) que se obtiene en base a la multiplicación del ángulo de paso por el número de movimientos que el stepper haya realizado. Se debe tener en cuenta que este azimut tendrá en cuenta el sentido del giro para establecer el ángulo horizontal (por ejemplo, si se llega al tope en un giro en el sentido horario las medidas en el sentido antihorario comenzarán con un azimut de 360 grados y no 0º). Ambos datos son generados por el firmware y no son enviados por el RPLIDAR.

```
void calidadAzimut(){
  if(flag_scan){//Siempre y cuando el flag_scan este a 1 es que hay nuevo dato
     contar_dato++;
     nuevo_dato++;
     if(nueva_vuelta==2){
        //Guarda dato de nuevo medida del LiDAR
        buffer_scan[pos_buffer_scan]= nuevo_dato;
        pos_buffer_scan++;
        if(pos_buffer_scan==2000){
           pos_buffer_scan=0;
        }
        //2 primeras condiciones son los puntos limites cuando los flags_antihorario
todavía no han cambiado pero el ángulo debe cambiar
        if((flag_antihorario==1)&&(pos_stepper==0)){
           buffer_move_stepper[pos_buffer_move_stepper]= pos_stepper*angulo_step;
        }else if((flag_antihorario==0)&&(pos_stepper==0)&&(flag_primeravez==1)){
           buffer_move_stepper[pos_buffer_move_stepper]= angulo_tope-
(pos_stepper*angulo_step);
        }else if(flag_antihorario==0){
           buffer_move_stepper[pos_buffer_move_stepper]= pos_stepper*angulo_step;
        }else if(flag_antihorario==1){
           buffer_move_stepper[pos_buffer_move_stepper]= angulo_tope-
(pos_stepper*angulo_step);
        }

        if((pos_stepper*angulo_step)==angulo_tope){//Para que llegue a 360 grados u
otro ángulo que se establezca
           flag_45grados=1;
        }
        pos_buffer_move_stepper++;
        if(pos_buffer_move_stepper==2000){
           pos_buffer_move_stepper=0;
        }

        //Dato Calidad
        dato_quality=(Rx_dato&0xFC)>>2;//Se desplazan a la derecha para tenerlo en la
posición correcta
        buffer_scan[pos_buffer_scan]= dato_quality;
        pos_buffer_scan++;
    }
    flag_scan=0;
  }
}
```

Código 5.15. Función calidadAzimut.

Posteriormente, con la llegada del segundo y tercer byte se almacena el ángulo al completo recibido teniendo en cuenta las especificaciones relatadas al comienzo de este apartado respecto a los bits más y menos significativos tal y como se ilustra en la función *cAnguloCompl* en el Código 5.16. Además, en función a lo explicado anteriormente relacionado con el formato Q de punto fijo (Q6) se convertirá dicho valor a decimal (float). También, se almacena el bit C por si fuera necesario usarlo en algún momento. El dato de este ángulo de elevación tiene una resolución de 1,2 º que se considera una buena distancia

entre medidas para una correcta representación. Esto se consigue variando la tensión de alimentación que se le da al motor del RPLIDAR, en este caso de 5 V.

```
void cAnguloCompl(){
  contar_dato++;
  contar_angulo++;
  Rx_dato=buffer_datos[pos_buffer_datos_1];
  pos_buffer_datos_1++;
  if(contar_angulo==1){ //Angulo por partes
     dato_C=Rx_dato&0x01;   // Flag C
     dato_angulo_1=(Rx_dato&0xFE)>>1; //Bits menos significativos ángulo
     flag_scan=0;
  }else if(contar_angulo==2){
     dato_angulo_2=Rx_dato; //Bits más significativos ángulo
     dato_angulo= ((dato_angulo_2<<7)|dato_angulo_1); // Se junta los 15 bits del
angulo

     if(nueva_vuelta==2){

        angle_float= (dato_angulo*const_angle_q);
        buffer_angle_dist[pos_buffer_angle_dist]=angle_float;

        pos_buffer_angle_dist++;

        buffer_angle[pos_buffer_angle]=angle_float;
        pos_buffer_angle++;
     }
     flag_cont_angle++;
if(((angle_float<2)||((angle_float>359)&&(angle_float<360)))&&(flag_cont_angle==1)){
        flag_correct=1;
     }else
if(((angle_float>2)&&((angle_float<357)||(angle_float>360)))&&(flag_cont_angle==1)){
       flag_correct=0;
    }
    contar_angulo=0;
    flag_scan=0;
  }
}
```

Código 5.16. Función cAnguloCompl.

Por último, se procesa la distancia con la llegada del cuarto y quinto byte siguiendo un proceso análogo al explicado para el caso del ángulo como se observa en la función *distancia* en el Código 5.17 siendo, en este caso, el punto fijo de tipo Q2. En este momento cuando se recoge una medida completa se procede a resetear la función para la llegada de la siguiente medida y proceder a habilitar el envío de los datos completos recibidos por USB hacia la aplicación.

```
void distancia(){
  contar_dato++;
  contar_distancia++;
  Rx_dato=buffer_datos[pos_buffer_datos_1];
  pos_buffer_datos_1++;
  if(contar_distancia==1){  // Distancia por partes
```

```
      dato_distancia_1=Rx_dato; //Bits menos significativos distancia
      flag_scan=0;
   }else if(contar_distancia==2){
      //Resetear flags porque viene una nueva trama
      contar_dato=0;
      flag_scan=0; // Se termina de recibir y se cambia el flag
      dato_distancia_2=Rx_dato; //Bits más significativos distancia
      dato_distancia=(dato_distancia_2<<8)|dato_distancia_1; // Se junta los 16 bits
      contar_distancia=0;
      if(nueva_vuelta==2){
         distance_float= (dato_distancia*const_dist_q);
         buffer_angle_dist[pos_buffer_angle_dist]=distance_float;
         pos_buffer_angle_dist++;

         buffer_dist[pos_buffer_dist]=distance_float;
         pos_buffer_dist++;

         if((nueva_vuelta==2)&&(flag_correct==1)){
            flag_USB=1;
         }
      }
   }
}
```

Código 5.17. Función distancia.

En este momento ya se dispone del conjunto de tres datos que permiten describir el movimiento del sistema de forma tridimensional: el azimut (dado por el stepper), el ángulo de elevación (dado por el ángulo del RPLIDAR) y la distancia (dado por el láser igualmente). Tal y como se caracteriza en la Figura 5.19 el sistema de datos obtenidos corresponde a una representación de unas coordenadas esféricas [55].

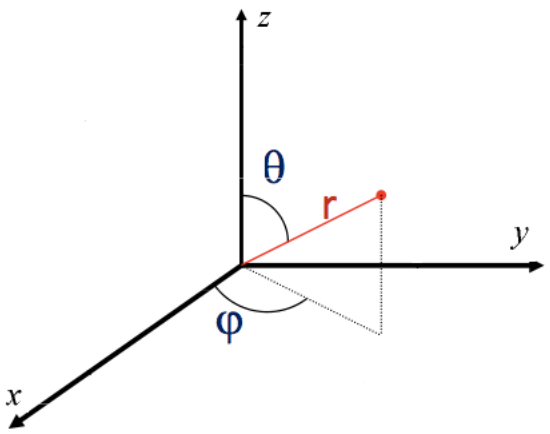


Figura 5.19. Sistema de coordenadas esféricas.

5.9 Etapa de envío de datos por USB

Esta etapa consiste en preparar la trama de comunicación en base a los datos consolidados de cada medida y enviarla por el USB a la aplicación. El formato de la trama cuya longitud es de 31 bytes se caracteriza por enviar una cabecera, el conjunto de los datos y, por último, el fin de trama tal y como se ilustra en la Figura 5.20. A su vez en la Figura 5.21 se encuentra un ejemplo de una trama real de transmisión. Los datos que conforman una trama completa son:

- Cabecera [0]: corresponde al valor en hexadecimal de 0x5A (Z)
- Número de medida [1:4]: especifica el número de la medida (entero) en una vuelta vertical del *LiDAR*. Como máximo tomará el valor de 300 muestras.
- Azimut [5:10]: indica el ángulo de desplazamiento horizontal (decimal) que puede variar entre 0 y 360º.
- Flag antihorario [11]: indica el sentido del giro horizontal del sistema (horario (0), antihorario (1)).
- Calidad [12:15]: representa lo buena que es la medida obtenida. Si es 0 es que la medida es inválida.
- Ángulo de elevación [16:22]: representa el ángulo de la medida (decimal) provisto por el *LiDAR* y puede ir de 0 a 360º. Se envía el dato con una precisión de como mínimo 3 decimales.
- Distancia [23:29]: indica la distancia medida respecto a un punto tomando valores desde 0 a 6000 (mm). Se envía el dato con una precisión de mínimo 2 decimales.
- Fin de trama [30]: corresponde al valor en hexadecimal de 0x23 (#).

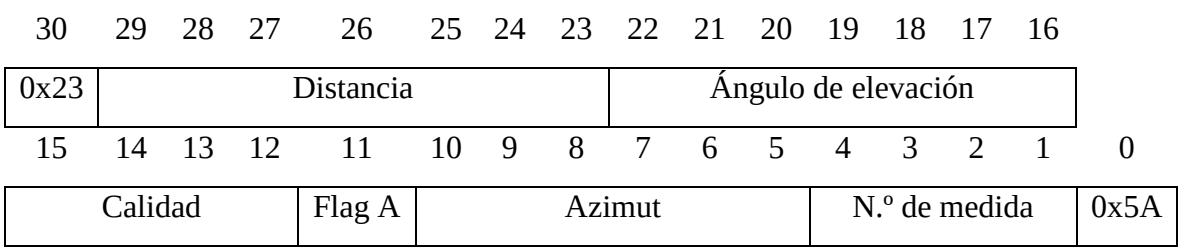


Figura 5.20. Formato trama de envío por USB.

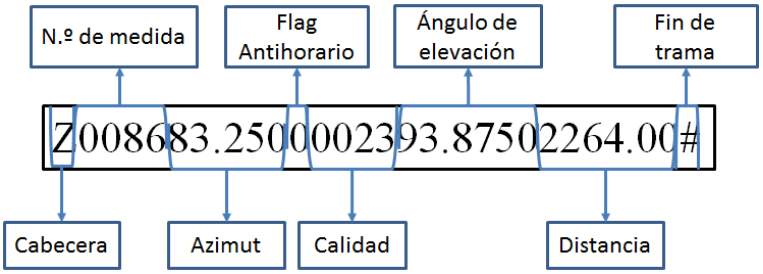


Figura 5.21. Ejemplo de trama de envío por USB.

En el Anexo 2. Código 3 se observa la forma en como se preparan los datos para enviarlos por USB a través de la función *printf()*. Como esta función envía caracteres es necesario transformar los datos disponibles a hexadecimal y covnertirlos en caracteres ASCII [56]. Independientemente si el dato a transmitir se encuentra en hexadecimal o en float primero es necesario pasarlo a dato de tipo carácter (char) con ayuda de la función *snprintf()* [57] y posteriormente almacenarlo en un buffer intermedio de enteros de 8 bits sin signo (uint8_t) que es lo que se envía por USB. Además, destacar que en ciertas conversiones cuando, por ejemplo, para un espacio de transmisión de tres bytes solo se ocupan dos, manualmente, se añade el símbolo en hexadecimal del carácter 0 (0x30). A continuación, ya se puede guardar cada dato en el buffer de transmisión en la posición que le corresponda. Es necesario que cuando se produce una transmisión deban limpiarse los bufferes intermedios para evitar corromper posteriores tramas de comunicación.

5.10 Etapa de fin de operación

Esta etapa corresponde como se representa en la Figura 5.22 a la parada del sistema cuando se envía la trama de stop desde la aplicación (0x52). Cuando esto ocurre, primero se debe enviar una trama de stop al RPLIDAR A1 para que pare la adquisición de medidas a la vez que se para el motor de rotación de dicho dispositivo. A continuación, se debe verificar la posición del servo (RPLIDAR A1) respecto al sensor de parada y en caso de que no se encuentra en dicho lugar proceder a moverlo tal y como se ilustra en la función *resetServo* en el Código 5.18. Por último, es necesario resetear el sistema para que vuelva a las condiciones iniciales de funcionamiento en caso de que llegue una nueva trama de start.

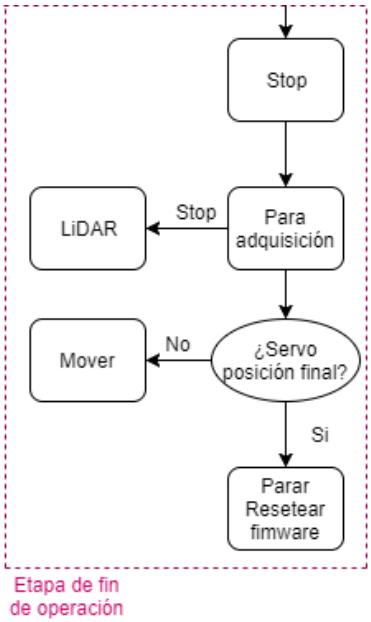


Figura 5.22. Diagrama de la etapa de fin de operación.

```
void resetServo(){
  if(reset_servo){
    TIM4_Enable();
    TIM4->CCR3=755;
    if(first_stop){//Si es la primera vez se manda un stop
      sec=3;
      first_stop=0;
      flag_count_reset=3;
      flag_USART=3;
    }
  }
}
```

Código 5.18. Función resetServo.

6 Desarrollo de la aplicación de visualización

6.1 Introducción

Para la realización de la aplicación de visualización se ha empleado la herramienta Processing ya que permite diseñar aplicaciones estables (representación tridimensional) de forma sencilla siendo de uso libre [58]. Además, presenta gran cantidad de librerías disponibles y abundante bibliografía sobre el entorno que hacen recomendable su uso. Processing fue creado por Casey Reas y Ben Fry en la primavera de 2001 con la intención de proveer una plataforma que permitiera probar las ideas gráficas (interfaces de usuario o geometría) de forma sencilla sin malgastar demasiado tiempo en la programación C++ [59]. Su principal objetivo era obtener un entorno de programación sencillo (por ejemplo, una línea represente un círculo y la siguiente línea implemente el movimiento de dicha figura en función del movimiento del ratón) que permita escribir software para imágenes, animaciones e interacciones sin excesivo código.

La versión 3.0 de Processing, que es la que se usa para la realización de este TFM, presenta como gran ventaja la depuración del código. El lenguaje usado está basado en Java siendo la sintaxis de éste casi idéntica, pero agregando características personalizadas a los gráficos y la interacción [60]. Los elementos gráficos de Processing están relacionados con PostScript (una base de PDF) y OpenGL (una especificación de gráficos 3D) [61].

El entorno de programación tal y como se representa en la Figura 6.1 contiene semejanzas con el entorno de otros sistemas de programación como Arduino. Por otro lado, a lo largo de ese tiempo se ha desarrollado un sistema de bibliotecas que permite realizar multitud de procedimientos gráficos (en la actualidad hay más de 100 bibliotecas disponibles, desde elementos gráficos hasta de comunicación serie) [58].

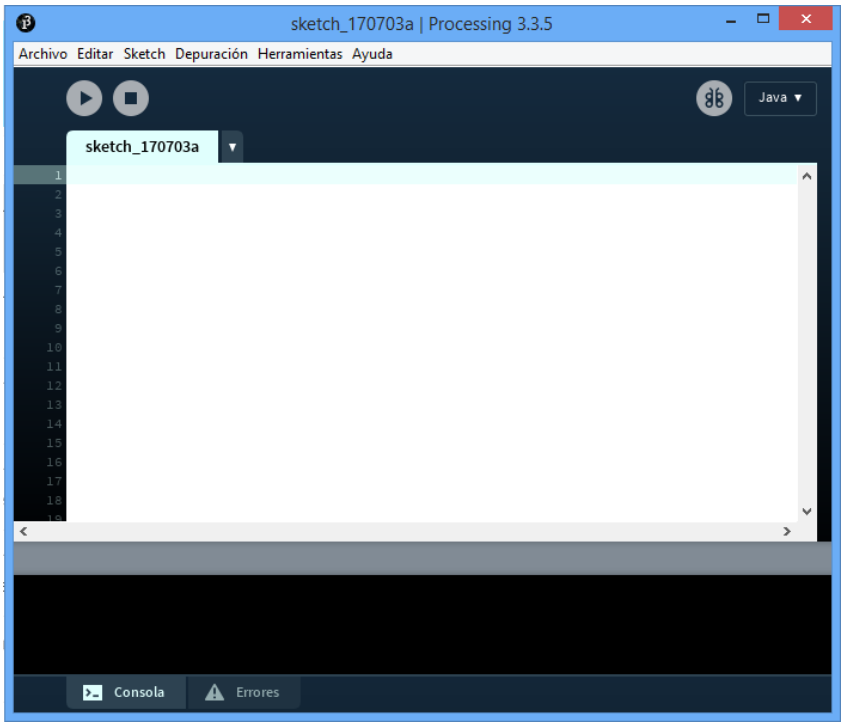


Figura 6.1. Entorno de programación en Processing.

6.2 Librerías usadas

Para el desarrollo de esta aplicación se han usado una serie de librerías, que soporta el programa Processing, de código abierto siendo estas:

- ControlP5: sirve para diseñar la interfaz gráfica de usuario de la aplicación (botones, cuadros de texto, displays, etc.). Fue diseñada por Andreas Schlegel y la versión usada en este proyecto es la 2.2.6 [62]. La principal ventaja del uso de este tipo de librería frente a otras es que cada elemento gráfico puede tener un controlador diferente o hacer agrupaciones de elementos para un único controlador. Por otra parte, permite de forma sencilla detectar y actuar en consecuencia en función de los eventos producidos en cada uno de los elementos gráficos.
- PeasyCam: es una librería, diseñada por Jonathan Feinberg, que permite implementar una cámara en una determinada posición del espacio de

representación para la correcta visualización del objeto. La versión utilizada en este proyecto es la v202 que se distribuye bajo Licencia Pública Apache [63]. La principal ventaja de esta librería es que la cámara es independiente a la representación y dispone tanto de distintos modos de funcionamiento (total o de alguno de los ejes) como de funciones ya implementadas como el giro o el zoom de la cámara en función de distintas acciones del ratón.

- HE_Mesh: permite crear y representar tridimensionalmente figuras preestablecidas o nubes de puntos Fue creada por Frederik Vanhoutte siendo la versión usada para este TFM la 5.0.3; además es de acceso libre para cualquier usuario [64]. Su principal ventaja es la forma sencilla que tiene de almacenamiento y tratamiento de datos, permitiendo trabajar con grupos de puntos o de forma individual según sean las necesidades.

6.3 Descripción general de la aplicación

Cuando se arranca la aplicación se carga la pantalla inicial y se representan el botón "on-off" y dos campos de texto (archivo a leer y archivo a escribir). Además, de forma invisible al usuario, se inicializan todas las funcionalidades de la aplicación para que puedan ejecutarse cuando sea necesario.

Lo primero que se debe hacer es introducir por teclado el nombre de un archivo en alguno de los dos cuadros dependiendo de si se quiere realizar un escaneado (escribir) o una visualización de datos ya guardados (leer). En base a esto, se enseñará por pantalla si estas operaciones se han realizado de forma correcta o no. Es necesario que se hayan seguido alguna de estas dos acciones para que el programa pueda continuar su ejecución tal y como se ilustra en la Figura 6.2

Posteriormente, cuando se cliquea en el botón "on-off" (habiéndose realizado el paso previo) se envía la trama de comienzo (start) hacia el sistema hardware implementado, y explicado en el capítulo anterior, en el caso de que se haya creado un archivo nuevo para una representación en tiempo real.

En el caso que se realice un escaneado en directo, se espera ya en la ventana principal la llegada de datos y su representación. Para ello, se va almacenando cada byte que llega a la aplicación en un string y cuando se detecte la llegada del comando "#" se comprueba si corresponde a la posición 30 de dicho string. Realizando esta comprobación se considera que ha llegado una trama completa correctamente siguiendo la descripción de la trama explicada en el capítulo anterior.

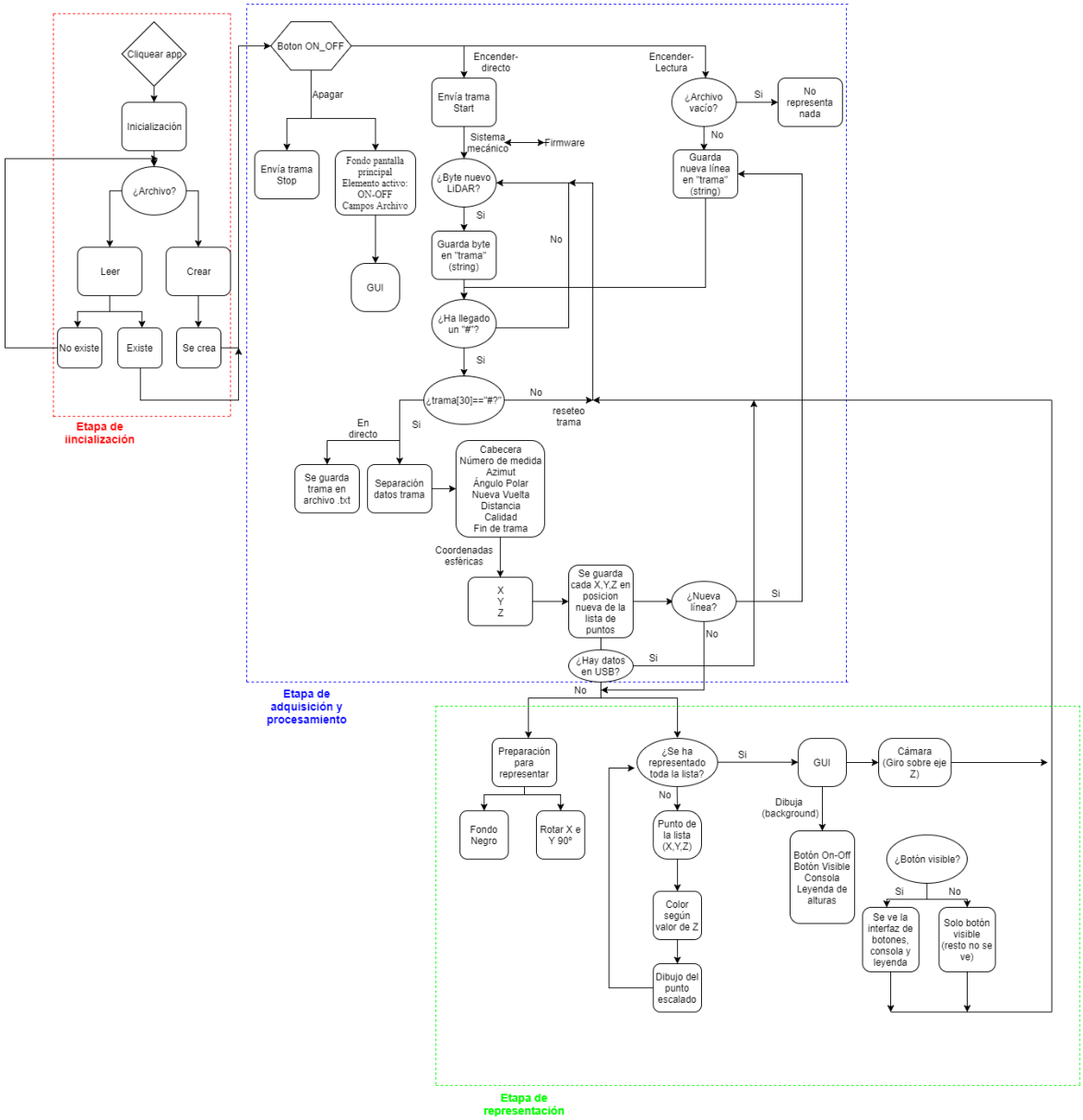


Figura 6.2. Diagrama de flujo de la aplicación de procesamiento.

Cuando esto sucede se procede a dividir la trama en los diferentes campos de datos necesarios, por ejemplo, azimut, ángulo de elevación o la distancia entre otros y, con ellos, se procede a realizar la conversión entre el sistema de coordenadas esféricas de la adquisición con el cartesiano en tres dimensiones de la representación que se explicará más detalladamente en los siguientes apartados. A medida que se tenga el trío de puntos (‘x’, ‘y’ y ‘z’) para cada una de las tramas recibidas se procede a guardar en una nueva posición de la lista de puntos (empleada posteriormente para su representación). Se seguirá guardando datos hasta que el puerto serie indique que no hay cola de espera; esto ocurre cuando se dé un escaneo completo en el plano vertical. De forma adicional, cada trama recibida se irá guardando en un archivo externo por si se desea realizar un análisis offline del entorno.

Sin embargo, cuando lo que se quiere es leer un archivo completo no hace falta realizar la comunicación serie ni esperar datos del sistema electromecánico por lo que únicamente se debe leer cada una de las tramas y proceder a su división de forma análoga a lo explicado en el párrafo anterior hasta que se finalice el archivo.

Posteriormente, se procederá a representar tridimensionalmente (etapa de representación) los conjuntos de puntos almacenados en dicha lista. Para ello, primero se deben realizar una serie de rotaciones del espacio para poder observar el área de forma correcta y, a continuación, se volcará toda la lista de puntos en la pantalla (que se irá incrementando a medida que vayan llegando nuevos datos) de tal forma que se podrá ver la reconstrucción del espacio a medida que se obtienen nuevos datos. Destacar, que cada punto representado (escalado proporcionalmente) tendrá un color u otro en función de su coordenada Z (para la percepción de la altura). Además, indicar que en el entorno de visualización se visualiza la consola de tramas recibidas en la parte izquierda como la leyenda de colores que indica la altura de cada punto. Adicionalmente, cliqueando el botón “visible” se pueden ver estas interfaces o esconderlas cuando solo se necesite ver la representación tridimensional.

En el caso de que se esté representado en tiempo real, cuando se termina de representar la lista de puntos se espera a la llegada de la siguiente trama completa repitiendo el ciclo de funcionamiento detallado en este apartado. Sin embargo, en el caso de que se lea

un archivo existente lo que se observa, desde el primer momento, es la representación completa del conjunto de puntos almacenados en el archivo de origen

Por último, cuando de nuevo se cliquea el botón de "on-off" (apagado, en este caso) se carga la pantalla inicial, reseteando las interfaces de usuario a posiciones iniciales (cámara) y desactivando la visualización de la consola y la leyenda de colores. Además, se envía la trama de stop al sistema mecánico (firmware) para que pare la adquisición de datos y vuelva al estado de reposo en el caso de que fuera necesario. Para que volviera a iniciarse se debe seguir el procedimiento explicado al comienzo de este apartado.

6.4 Inicialización del programa

La etapa de inicialización parte desde el comienzo de la aplicación hasta que se observa la pantalla de inicialización. En esta fase se carga la función "setup" tal y como se puede observar en el Código 6.1 que en Processing tiene la propiedad de ser la función que se ejecuta la primera vez que se activa la aplicación. En ella al comienzo se define el tamaño de la pantalla de visualización del programa (800x600 píxeles) y el formato de representación, en este caso, P3D (representación tridimensional). A su vez, en Figura 6.3 también se puede observar el diagrama de flujo de esta etapa.

```
void setup() {

  //Tamaño de la pantalla
  size(800, 600, P3D);

  //Inicialización de la cámara
  g3 = (PGraphics3D)g;
  cam = new PeasyCam(this,0,-100,0, 400);
  cam.setYawRotationMode();
  cam.setWheelScale(0.1);
  state = cam.getState();

  //Inicialización de los controladores de la interfaz de usuario
  MyController1 = new ControlP5(this);
  MyController2 = new ControlP5(this);
  MyController3 = new ControlP5(this);
  MyController4 = new ControlP5(this);
  MyController5 = new ControlP5(this);
  MyController6 = new ControlP5(this);

  botones();
```

```
  consola();
  leyendaColores();
  cuadrosTexto();

  //Inicialización de elementos gráficos todos apagados
  MyController1.setAutoDraw(false);
  MyController2.setAutoDraw(false);
  MyController3.setAutoDraw(false);
  MyController4.setAutoDraw(false);
  MyController5.setAutoDraw(false);
  MyController6.setAutoDraw(false);

  render= new WB_Render3D(this);

  points=new ArrayList<WB_Point>();

  bg = loadImage("FondoPrograma.png");

  //Inicialización del puerto serie
  String portName = Serial.list()[1];
  myPort = new Serial(this, portName, 115200);
}
```

Código 6.1. Función setup.

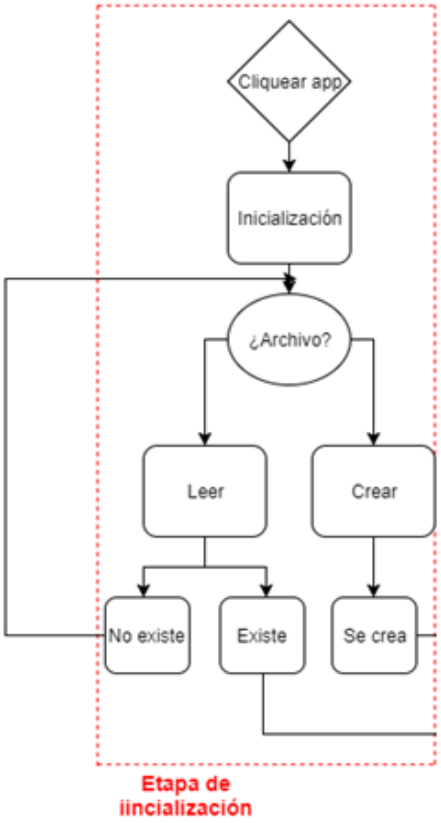


Figura 6.3. Diagrama de flujo de la etapa de inicialización.

A continuación, se inicializa la cámara de visualización de la representación tridimensional; se entrará más en detalle de su funcionamiento en la etapa de representación

que se explicará en los siguientes apartados. Además, se deben inicializar los controladores de cada uno de los elementos gráficos de la interfaz de usuario que automáticamente no se representarán hasta que sea necesario:

- Controlador 1: tiene como función el manejo del botón de encendido/apagado de la aplicación. Cuando se cliquea por primera vez se activa la aplicación y la siguiente vez que se realice dicha acción se desactiva.
- Controlador 2: su misión principal es el la caracterización del botón visible que permite la visualización de ciertos elementos de la interfaz o no según estime el usuario.
- Controlador 3: se encarga de dirigir el funcionamiento de la consola donde se visualizan las tramas de datos recibidas (destacar que permite observar desde la primera a la última trama recibida).
- Controlador 4: tiene como misión el manejo de la leyenda de colores en función de la altitud de los puntos.
- Controlador 5: se encarga de detectar los datos introducidos en los campos de texto y actuar en consecuencia.
- Controlador 6: su misión se fundamenta en la presentación por pantalla de un mensaje relacionado con la acción de gestión de archivos usada.

A su vez, las funciones *botones*, *consola* y *leyendaColores* se encargan de diseñar la interfaz de usuario tal y como se puede observar en el Anexo 3. Código 1, Anexo 3. Código 2, y Anexo 3. Código 3, respectivamente, además en el Anexo 3. Código 4 se describe la configuración hexadecimal de cada uno de los colores empleados. En ellos se puede observar en qué posición de la pantalla se coloca cada elemento, su letra y tamaño, color, etc. El resultado a nivel de representación de la interfaz gráfica en la pantalla quedaría tal y como se ilustra en la Figura 6.4.

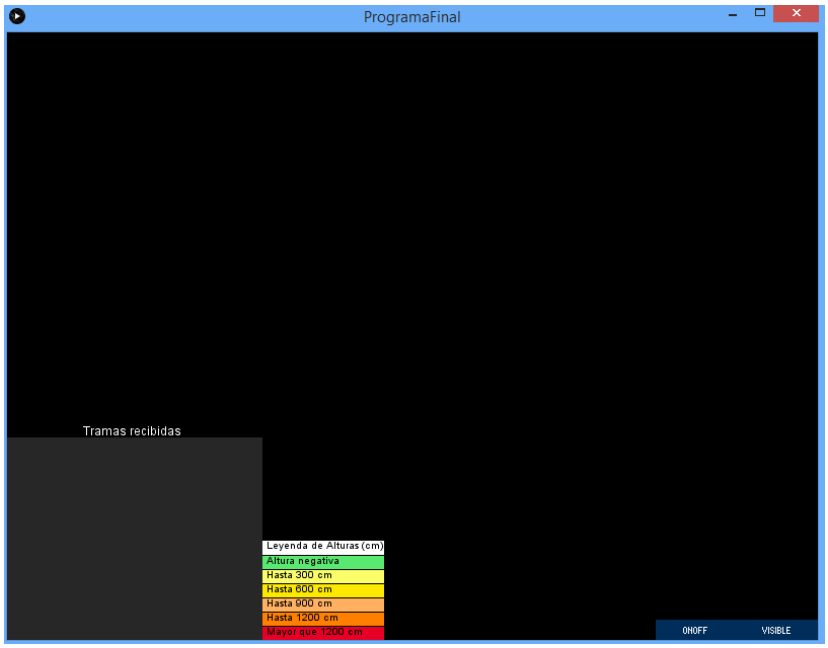


Figura 6.4. Interfaz gráfica de la aplicación.

Por otro lado, destacar que las funciones de gestión de los botones que se encuentran en el Anexo 3. Código 5 y Anexo 3. Código 6, solo se activan cuando se producen un evento en alguno de ellos (presionar). En particular, en el caso del botón de "on-off" tiene como misión fundamental iniciar una representación en tiempo real y la configuración del puerto. Para ello se debe realizar primero una búsqueda del mismo que, normalmente, coincide con el COM14 que debe ser configurado a la misma velocidad que el puerto USB de la tarjeta de desarrollo (115200 bps). Las tramas empleadas para la comunicación se explicarán en el siguiente apartado de este capítulo. Además, se encarga del envío de la trama de comienzo (start) el de final (stop) según sea el estado de dicho botón, inicializar objetos (render, lista, etc.) y de resetear el programa cuando sea necesario

Posteriormente a realizar la inicialización de la aplicación se procede a ejecutar el *draw()* que en esencia funciona como el bucle infinito de otros lenguajes como C y que, por tanto, continuamente se estará ejecutando. Esta función se explicará en detalle en el apartado de la representación de los datos. Sin embargo, cuando se inicializa se ejecuta una parte reducida del mismo; en este caso inicial debido a que el botón "on-off" está apagado se desactiva la visualización de ciertos elementos como la leyenda de colores y la consola además de añadir el fondo de pantalla (background) de la aplicación; y siendo el único botón visible en ese momento es el de "on-off" además de los campos de datos.

Para estos campos de datos se diseña una interfaz que se encuentra dentro de la función *cuadrosTexto* que tiene como misión la representación en pantalla de estos campos tal y como se ilustra en el Anexo 3. Código 7. Como se observa, se crean dos campos de texto uno para archivos que se van a leer y otro para archivos que se van a crear y, además, un mensaje por pantalla que indica el resultado de la gestión de los archivos.

```
void controlEvent(ControlEvent eventEnter) {
  if(eventEnter.isAssignableFrom(Textfield.class)) {
   if(eventEnter.getName().equals("Archivo a leer")){
     textValue=eventEnter.getStringValue()+".txt";
     file= new File(dataPath(textValue));
     flagArchivo=1;
     accionTexto();
   }else if(eventEnter.getName().equals("Archivo a crear")){
     textValue=eventEnter.getStringValue()+".txt";
     file= new File(dataPath(textValue));
     output = createWriter(dataPath(textValue));
     flagArchivo=2;
     accionTexto();
   }
  }
}
```

Código 6.2. Función controlEvent.

Para la gestión de estos de campos de texto se encuentra la función *controlEvent* que tiene como misión adquirir los datos introducidos en los espacios habilitados cuando se pulse la tecla *enter* tal y como se ilustra en el Código 6.2. Además, se observa que no es necesario poner el archivo completo, sino que el programa lo completa con la extensión .txt. A continuación, se comprueba la existencia de éste o se crea en la ruta especificada (carpeta data). Posteriormente, se llama a la función *accionTexto* que en dependiendo de la existencia o no del archivo muestra un mensaje por pantalla u otro tal y como se ilustra en el Anexo 3. Código 8. Los mensajes que se podrán visualizar son: el archivo existe, el archivo no existe o archivo creado. El resultado de la interfaz gráfica de usuario cuando se produce la inicialización se representa en la Figura 6.5.

Figura 6.5. Pantalla inicial aplicación.

Para que todo esto se disponga en pantalla ha sido necesario que se ejecute la función *gui* que implementa el dibujo de cada uno de los elementos gráficos y la cámara de visualización (que en este momento se encuentra invisible) tal y como se describe en el Código 6.3.

```
void gui() {
    currCameraMatrix = new PMatrix3D(g3.camera);
    camera();//Función que permite realizar todos los movimientos sobre el objeto 3D
    g3.camera = currCameraMatrix;
    //dibujo de cada elemento gráfico
    MyController1.draw();
    MyController2.draw();
    MyController3.draw();
    MyController4.draw();
    MyController5.draw();
    MyController6.draw();
}
```

Código 6.3.Función gui (interfaz gráfica).

6.5 Etapa de adquisición y procesamiento

La etapa de adquisición y procesamiento comienza cuando se pulsa el botón "on-off" de la pantalla principal que en líneas generales como se explicó anteriormente se encarga de enviar las tramas de start o de stop según corresponda. La comunicación serie entre la aplicación y el sistema electromecánico siempre se comienza desde esta aplicación con la trama de inicialización y se termina de igual forma con la trama de finalización en ASCII que se observan en la Tabla 6.1. Debido a la comunicación serie establecida por Processing lo máximo que se puede transmitir son 8 bits en cada comunicación por lo que primero se debe enviar la cabecera y a continuación el dato que se desea. Además, debe haber un retardo de 1 ms entre el envío de un dato y otro para que el USB de la tarjeta de desarrollo pueda procesarlo.

Start	95 (0x5F)	65 (0x41)
Stop	95 (0x5F)	82 (0x52)

Tabla 6.1. Tramas de comunicación USB (start y stop).

Posteriormente al envío de la trama de comienzo el sistema electromecánico, mediante el puerto USB, habilitado para ello empieza a enviar tramas con el formato especficado en el capítulo anterior (Figura 5.20). Tener en cuenta que como se revisa la llegada de dato siempre que se ejecuta el *draw()* (while (1)) se puede determinar que es una comunicación por "polling". En el Código 6.4 y en la Figura 6.6 se ilustra la forma en que se adquieren los datos procedentes del LiDAR. Para ello, primero se debe comprobar si existe algún byte para leer en el puerto serie (port.availaible>0); en caso afirmativo este se guarda en un string denominado "trama" que se termina de llenar cuando llega el comando "#" que indica fin de trama. Además, como comprobación adicional por si llegara algún valor nulo se verifica que el string trama no comprenda ninguno de estos tres supuestos: que el string no esté vacío, que no tenga el valor null como string o que sea declarado como null. A la vez que se guardan y procesan los datos también se guardan en un archivo externo (txt) todas las tramas que van llegando por si fuera de interés realizar una representación offline.

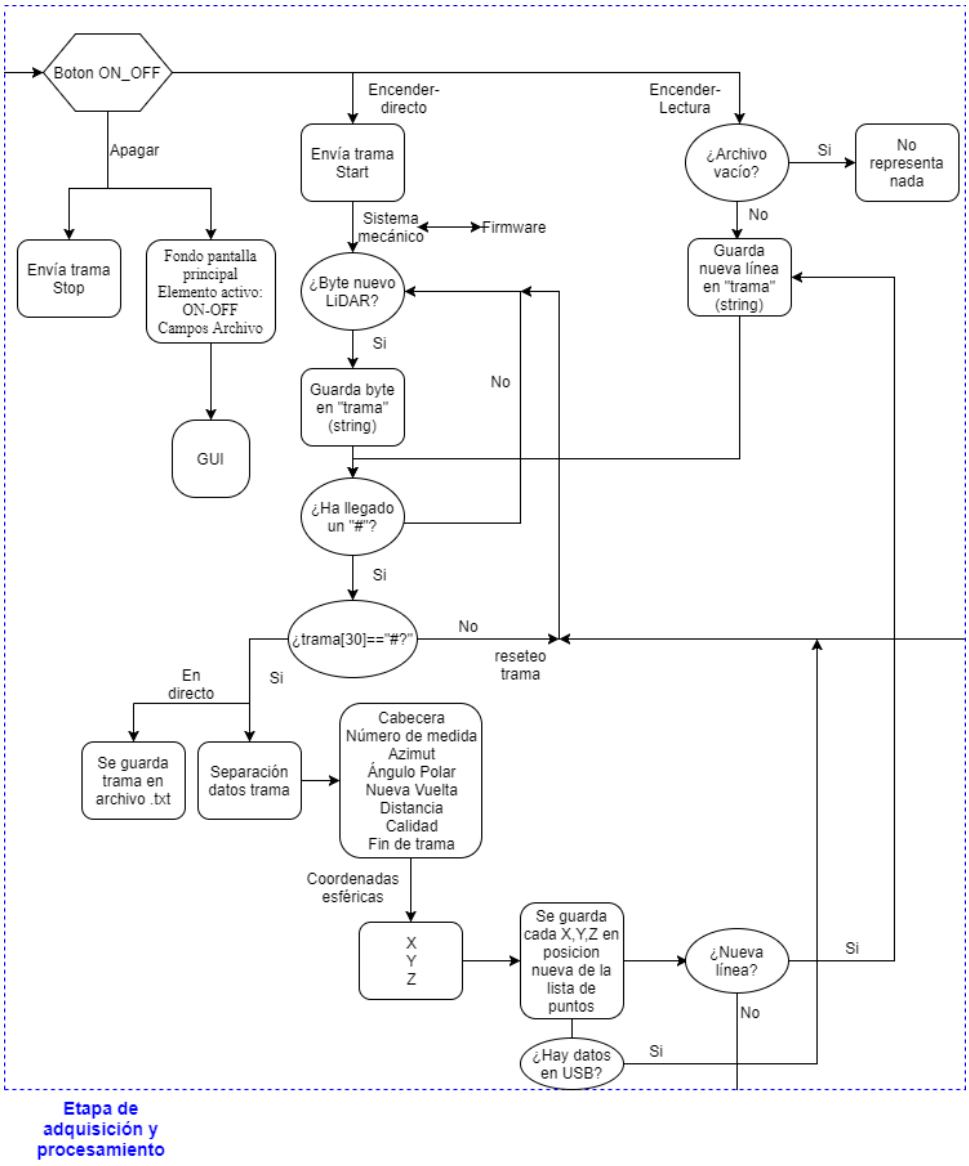


Figura 6.6. Diagrama de la etapa de adquisición y procesamiento.

```
void serial(){
   while(myPort.available()>0){
     trama=myPort.readStringUntil('#');
     if ((trama!="null")&&trama!=null&&trama!=""){
        output.println(trama);//se guarda trama en archivo
        AdaptDatos();
     }
   }
}
```

Código 6.4. Función serial.

Adicionalmente, si se quiere realizar una visualización de un entorno offline ya que se dispone de un archivo que contenga la totalidad de las tramas enviadas se ejecuta la función *lecutraArcivo*. Este algoritmo se encarga leer cada línea y cuando esto ocurra

analizar dicha trama como se explicó anteriormente con la función *adaptDatos*. La lectura completa del archivo debe realizarse cuando se cliquea sobre el botón "on-off".

```
void lecturaArchivo(){
  while(trama!=null){
  try {
    trama = reader.readLine();
  } catch (IOException e) {
    e.printStackTrace();
    trama = null;
  }
    if(trama==null){
       try{
         reader.close();
         flagDato=0;
       } catch (IOException e) {
         e.printStackTrace();
       }
    }else{
       AdaptDatos();
    }
  }
}
```

Código 6.5. Función lecturaArchivo.

A continuación, para ambos casos, ya se puede realizar el procesado de la trama subdividiendo dicho string en los distintos campos según el formato de la trama explicado en el capítulo anterior tal y como se describe en el Código 6.6. Indicar que esto se realiza de forma individual con cada una de las tramas completes recibidas.

```
void adaptDatos(){
  fintrama2=trama.substring(30);
  if(fintrama2.equals("#")){
    flagRec=1;
  }

  if(flagRec==1){
    //Cabecera
    cabecera = trama.substring(0,1);
    //Desplazamiento Vertical LiDAR
    desplVertLidar=int(trama.substring(1,5));
    //Desplazamiento Horizontal Stepper (azimut)
    desplHoriStepper=float(trama.substring(5,11));
    //Indica como es la medida horizontal si 0-360 (0) o 360-0(1)
    flagHorizontal=int(trama.substring(11,12));
    //Calidad
    calidad=int(trama.substring(12,16));
    //Ángulo (ángulo de elevación)
    angulo=float(trama.substring(16,23))*0.5;
    //Distancia
    distancia=float(trama.substring(23,30));
    //Fin de Trama
    finTrama=trama.substring(30);
```

```
    //Conversión a cartesianas
    x=(distancia)*sin(radians(angulo))*cos(radians(desplHoriStepper));
    y=(distancia)*sin(radians(angulo))*sin(radians(desplHoriStepper));
    z=(distancia)*cos(radians(angulo));


    //Se resetea flag hasta que llegue nueva trama
    flagRec=0;

    if((flagHorizontal!=1)&&(flagPrimeraVuelta==0)){
       points.add(new WB_Point(x,y,z));
    }else if(flagHorizontal!=1){
       points.add(new WB_Point(x,y,z));
    }else if(flagHorizontal==1){
       points.add(new WB_Point(x,y,z));
       flagPrimeraVuelta=1;
    }
    flagDibujar=1;
  }
  stext = myTextarea.getText();
  stext += trama + "\n";//Nueva trama más salto de línea
  trama="";//Reseteo trama
  myTextarea.setText(stext);
}
```

Código 6.6. Función adaptDatos.

Posteriormente, con la ayuda de los datos de azimut (desplHoriStepper), ángulo de elevación (ángulo) y la distancia se puede pasar de un sistema de coordenadas esféricas al sistema cartesiano tridimensional (x,y,z) [55]. En esta conversión es importante destacar que se debe respetar los convenios de representación de cada una de las componentes esféricas, es decir, azimut (de 0º a 360º), ángulo de elevación (de 0º a 180º), y la distancia (de 0 a ∞). A pesar de que los datos de azimut y del ángulo de elevación se adquieran en un rango de 360º se debe seguir el convenio especificado anteriormente para evitar solapamientos de la señal ya que, por ejemplo, el ángulo de elevación nunca va a poder tomar ángulos negativos ni mayor que a 180º lo que tiene relación con el principal uso de estas coordenadas que es el posicionamiento de satélites en el espacio. Esto de forma gráfica se puede observar en la Figura 6.7 que da lugar a las expresiones matemáticas (en radianes) descritas en el Código 6.6, con el que se representan los puntos en esta aplicación y que se pueden observar a continuación:

$$x = r * \sin(\theta) * \cos(\varphi)$$

$$y = r * \sin(\theta) * \sin(\varphi)$$

$$z = r * \cos(\theta)$$

El siguiente paso consiste en guardar el trio de puntos dentro de la lista de puntos habilitada para ello (WB_Point tal y como lo describe la librería He_Mesh). A la vez que se guardan los datos en la lista también se guardan en la consola que los pasará por pantalla cuando se representen dichos puntos.

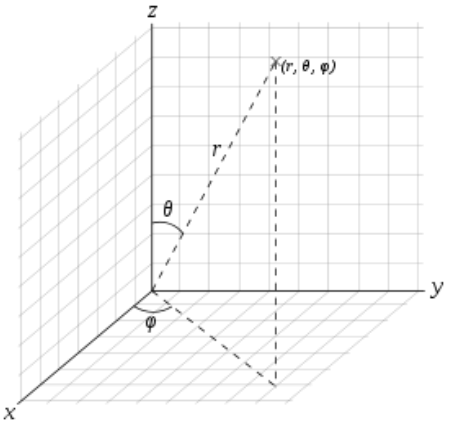


Figura 6.7. Sistema de coordenadas esféricas/cartesianas.

Aunque en esta versión del programa todos los datos que lleguen se guardan como datos nuevos es importante determinar a qué momento del LiDAR corresponden: la primera vuelta cuando se inicializa por vez primera (de 0º a 360º), cuando vuelve al punto de partida (360º a 0º) y cuando comienza la secuencia de nuevo. La aplicación siempre se encontrará en esta etapa de adquisición mientras haya datos en el puerto serie que leer, es decir, se adquirirán y procesarán primero todos los datos de una vuelta del LiDAR en un determinado azimut para posteriormente, al completo, realizar su representación tridimensional en la siguiente etapa. Por último, se ha incluido en esta etapa el proceso de apagado de la aplicación ya que únicamente consiste en enviar la trama de stop especificada anteriormente y resetear todas aquellas variables que sean necesarias como, por ejemplo, la cámara a su estado inicial.

6.6 Etapa de representación

La etapa de representación se encarga de dibujar en el entorno de visualización la nube de puntos almacenada en la lista explicada anteriormente. Esto se realiza en la función *draw()* que como se explicó al comienzo de este capítulo hace la función de bucle infinito o while(1) de otros lenguajes de programación por lo que debido a ello la función *serial* y *lecturaArchivos* se hayan ahí. Por lo tanto, en ella se encuentra tal y como se ve en el Código 6.7 y en la Figura 6.8, las acciones a realizar con la interfaz gráfica (en función de los botones) junto con la representación de los puntos formalmente dicha. El entorno de representación de dicha nube de puntos puede visualizarse en la Figura 6.4 en la que se observa tanto la consola de tramas, la leyenda de colores y los dos botones disponibles (los campos de texto en esta etapa se encuentran invisibles para el usuario).

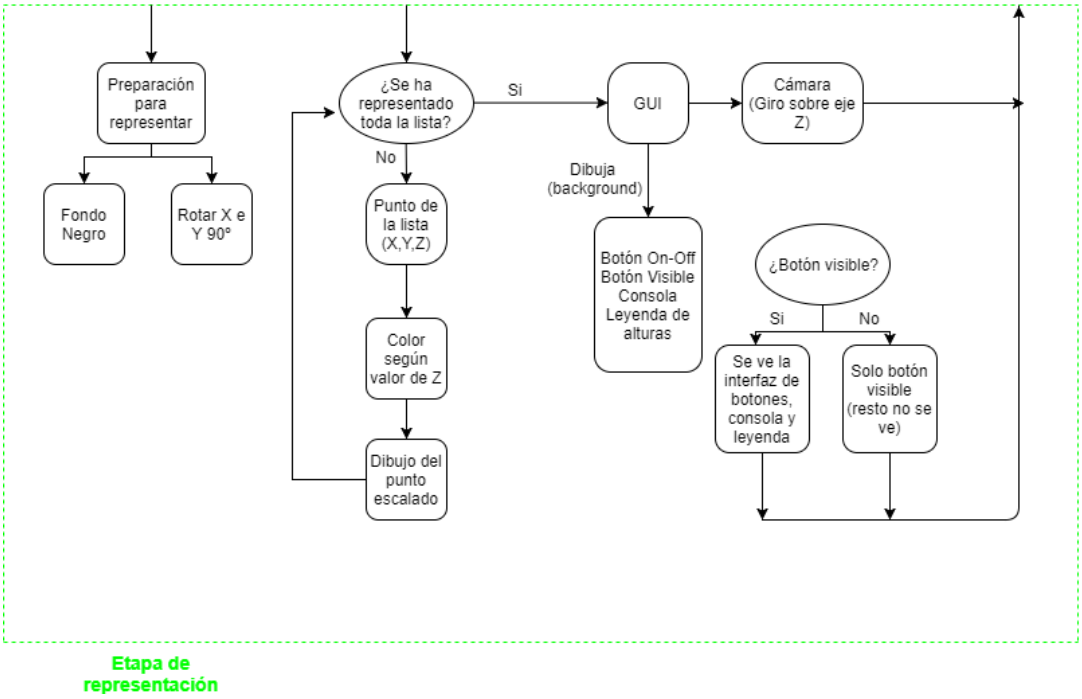


Figura 6.8. Diagrama de la etapa de representación.

```
void draw(){
  if(VISIBLE_OFF== 1){//Cuando se pulsa visible (Para taparse)
    MyController1.setVisible(false);
    MyController3.setVisible(false);
    MyController4.setVisible(false);
    background(bg);
  }else{//Cuando se pulsa visible (Para enseñar)
    MyController1.setVisible(true);
```

```
    MyController2.setVisible(true);
    MyController3.setVisible(true);
    MyController4.setVisible(true);
    background(bg);
  }

  if(ONOFF==1){
      if(flagArchivo==2){
        serial();
      } if(flagArchivo==1){
        lecturaArchivo();
      }
  }

  if((ONOFF==1)&&(flagDibujar==1)){
      MyController5.setVisible(false);
      MyController6.setVisible(false);
      background(0, 0, 0);
      strokeWeight(2);
      rotateY(PI/2);
      rotateX(PI/2);
      for(int w=0; w<points.size();w++){
          point=points.get(w);
          //A cada punto se le da un color en función del valor de la Z
          colorPunto(point.zf());
          //Se pasa al render cada uno de los puntos escalados
          render.drawPoint(new
WB_Point((point.xf())*escaladoX,(point.yf())*escaladoY,(point.zf())*escaladoZ));
      }
  }else{
      MyController2.setVisible(false);
      MyController3.setVisible(false);
      MyController4.setVisible(false);
      MyController5.setVisible(true);
  }
  gui();
}
```

Código 6.7. Función draw.

Cuando ha llegado el conjunto de tramas completas del LiDAR para un determinado azimut se procede a su representación. Para ello, primero se debe quitar la imagen de fondo inicial y pasar a un background negro (color de fondo adecuado para la representación de puntos); a continuación, se determina el tamaño de cada uno de los puntos (en este caso 2) y realizar una serie de rotaciones espaciales de la representación para una correcta visualización. Por último, un bucle recorre la totalidad de la lista de puntos, le da a cada uno un color en función de la componente Z (función *colorPunto* que se encuentra en el Anexo 3. Código 9) y se representa cada uno de ellos en el render (figura de representación tridimensional) habiendo previamente escalado cada uno de esos puntos a un 50% de su valor nominal.

La representación tridimensional en Processing (imagen de la izquierda, sistema de representación tridimensional de la mano izquierda) tiene cambios respecto a la concepción tradicional de un espacio tridimensional cartesiano (mano derecha) como se puede observar en la Figura 6.9. Esto indica que en Processing la representación del eje Y es desde el extremo superior al inferior y la tridimensionalidad que representa el eje Z es hacia el "fondo" del sistema. A efectos prácticos, esto no representa un fallo de la representación sino una disparidad de criterios en torno a la visualización de la representación. En los siguientes párrafos se desarrollarán los efectos trigonométricos realizados que permiten disponer de una representación en el sistema cartesiano tridimensional habitual. Es necesario indicar que los cambios de ejes de representación que se realizarán son a efectos visuales manteniendo el conjunto de la aplicación el sistema original de Processing.

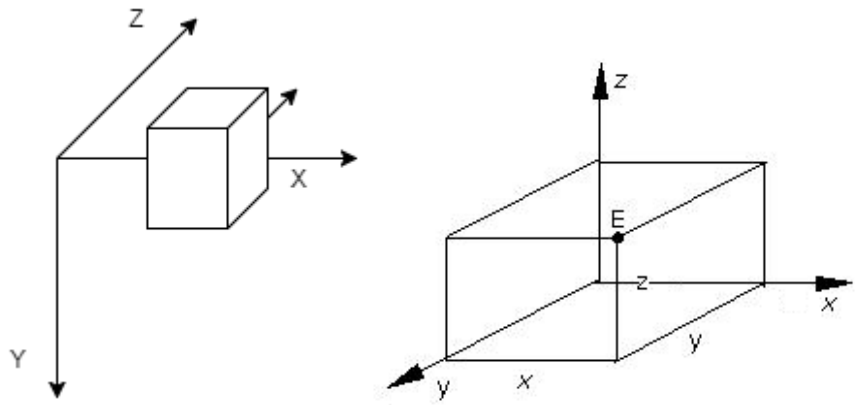


Figura 6.9. Sistema de coordenadas cartesianas de Processing frente al tradicional.

Lo primero que se quiere lograr es que el eje Z corresponda la vertical del sistema de coordenadas cartesianas habitual. Para ello, se debe girar 90º sobre el eje X tal y como se ilustra en la Figura 6.10 donde se pueden ver sus resultados que implican que a efectos de la visualización (por la forma de las medidas) la nueva representación en el eje X se incrementa de mayor a menor y no al revés. Esto comporta que se deba hacer una última rotación de la representación de 90º pero sobre el eje Y original (en código); sin embargo, en la nueva representación sería girar sobre el eje Z tal y como se ilustra en la Figura 6.11. El resultado es un sistema de coordenadas semejante al buscado con la diferencia que la componente Y positiva se desarrolla hacia el interior del sistema de coordenadas.

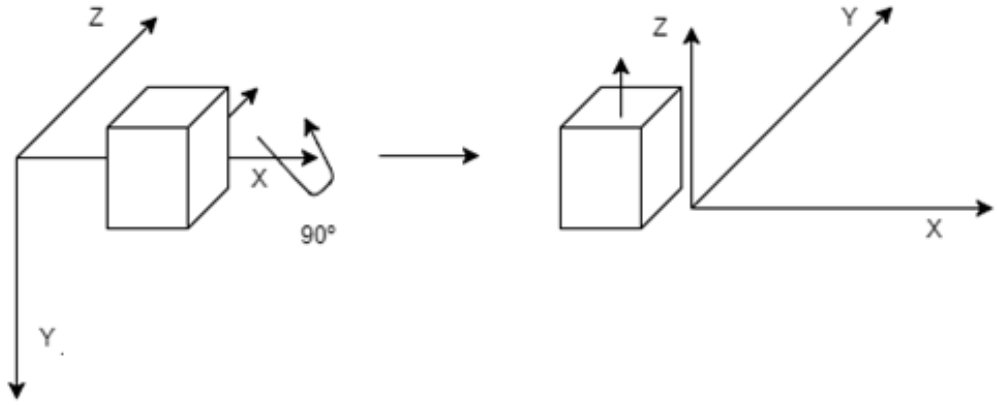


Figura 6.10. Rotación sobre el eje X de +90º.

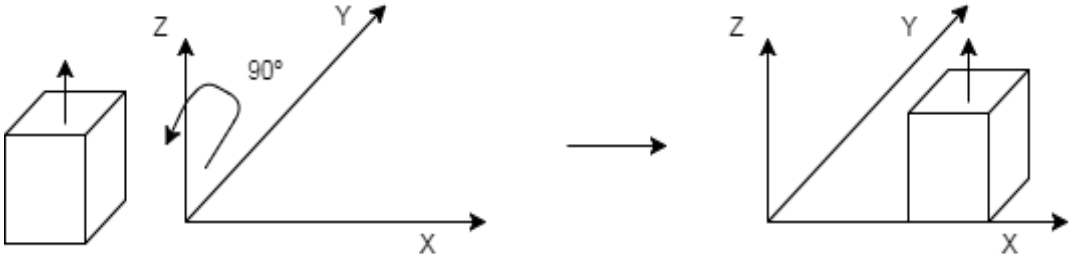


Figura 6.11. Rotación sobre el eje Y de +90º.

Para finalizar, indicar que la posición de la cámara de visualización de la representación tal y como se definió en la inicialización se encuentra en una posición central (origen de la representación) a excepción de la coordenada 'y' que se encuentra ligeramente desplazada verticalmente desde el origen (-100 teniendo en cuenta el eje de representación de Processing, a ojos del usuario sería un incremento positivo) a una cierta distancia espacial en el eje Z (400 pixeles para poder tener un ángulo espacial mayor). Los únicos movimientos permitidos a esta cámara es un zoom (botón derecho y rueda del ratón) como el movimiento en el eje Y de Processing, es decir, "eje Z" a ojos del usuario.

7 Resultados y revisión de objetivos

7.1 Resultados

Para poder realizar un análisis de resultados del sistema diseñado primero es necesario analizar la calidad de las datos, resolución, del *LiDAR*. Para ello, con ayuda del telémetro Bosch PLR-50 (clase 2) que se toma como base (resolución de 2 mm en 50m) se realizan una serie de medidas hacia un objeto que se encuentra en posiciones desde 1 metro hasta 6 metros de distancia (máxima distancia medible por el RPLIDAR) (Tabla 2.1) tal y como se ilustra en la Tabla 7.1.

Datos telémetro Bosch PLR-50	Datos *LiDAR*	Error relativo
6	5,9543	0,76%
5	4,9754	0,49%
4	3,9831	0,42%
3	2,9893	0,36%
2	1,9945	0,28%
1	0,9982	0,18%

Tabla 7.1. Cálculo de errores en las medidas del LiDAR.

A partir de estos datos se haya su error relativo tal y como se demuestra en la tabla anterior y en la Figura 7.1 se observa como presenta un error menor al 1% siendo el máximo del 0,76% cuando se mide en el fondo de escala del láser. Atendiendo a estos resultados se demuestra que las medidas obtenidas cumplen los parámetros de diseño por lo que se puede representar espacios tridimensionalmente de forma correcta.

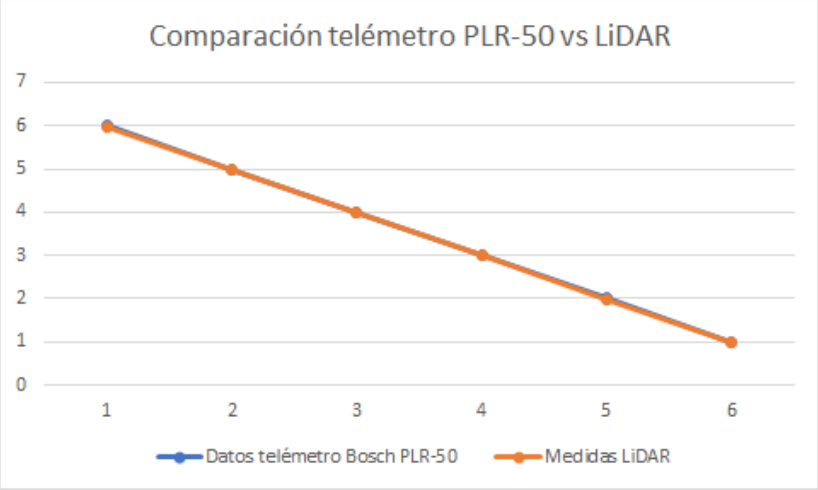


Figura 7.1. Comparativa de medidas LiDAR vs telémetro PLR-50.

En la Figura 7.2 se puede representa una parte de la representación tridimensional de una habitación de tamaño medio (laboratorio D de la División COM del IUMA; Edificio Central del Parque Científico 1º Planta) que alberga desde muebles o mesas hasta ventanas pudiendo hacerse una idea de su altura en función de la escala de colores con la que se representa cada punto. Comprobando su representación real con la que se observa en la aplicación desarrollada se puede distinguir la presencia de ciertos objetos como las sillas (cuadro lila), focos de luz (cuadro naranja), las estanterías (cuadro gris) y las ventanas. Destacar como en el caso de las ventanas cuando solo se encuentra el cristal se representa un espacio vacío ya que no encuentra punto de medida por la resolución del láser (cuadro azul y amarillo) y, por ejemplo, cuando a las persianas les afecta el sol ocurre un efecto parecido al anterior (cuadro rojo). Sin embargo, cuando las persianas cubren la totalidad de la ventana sin dejar pasar ningún rayo de sol estas son detectadas por el *LiDAR* (cuadro verde).

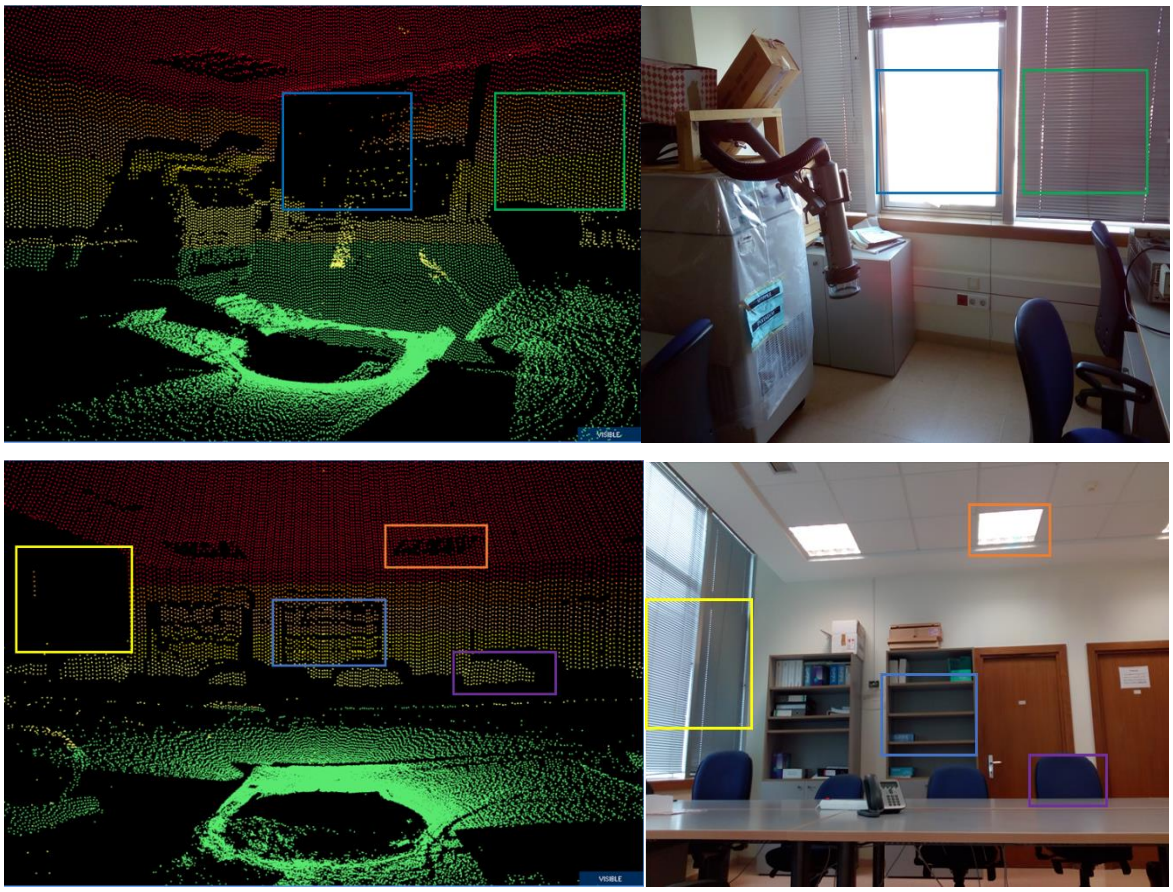


Figura 7.2. Representación tridimensional frente a imagen real de una sala mediana.

Por otro lado, en una habitación más pequeña (laboratorio A de la División COM del IUMA; Edificio Central del Parque Científico 1º Planta) a la anterior tal y como se observa en la Figura 7.3 se puede distinguir de mejor forma en la representación tridimensional algunas características del espacio que en el caso anterior. Por ejemplo, se observa con claridad la presencia de la pantalla de ordenador (cuadro lila) o un carro de medidas (cuadro gris) e incluso es capaz de representar la puerta entornada y la presencia del fondo de la otra habitación (cuadro amarillo). Además, se observa como la presencia de un cristal no produce difracciones en los haces de luz lo cual es positivo e incluso representa el techo de esa habitación (cuadro naranja). Por otra parte, como era de prever ante objetos negros o muy oscuros el *LiDAR* no puede obtener medidas de su posición (cuadro azul). Como en los casos anteriores se observa, en base a los colores de cada punto de representación, como la altura máxima es de aproximadamente 1,5 metros.

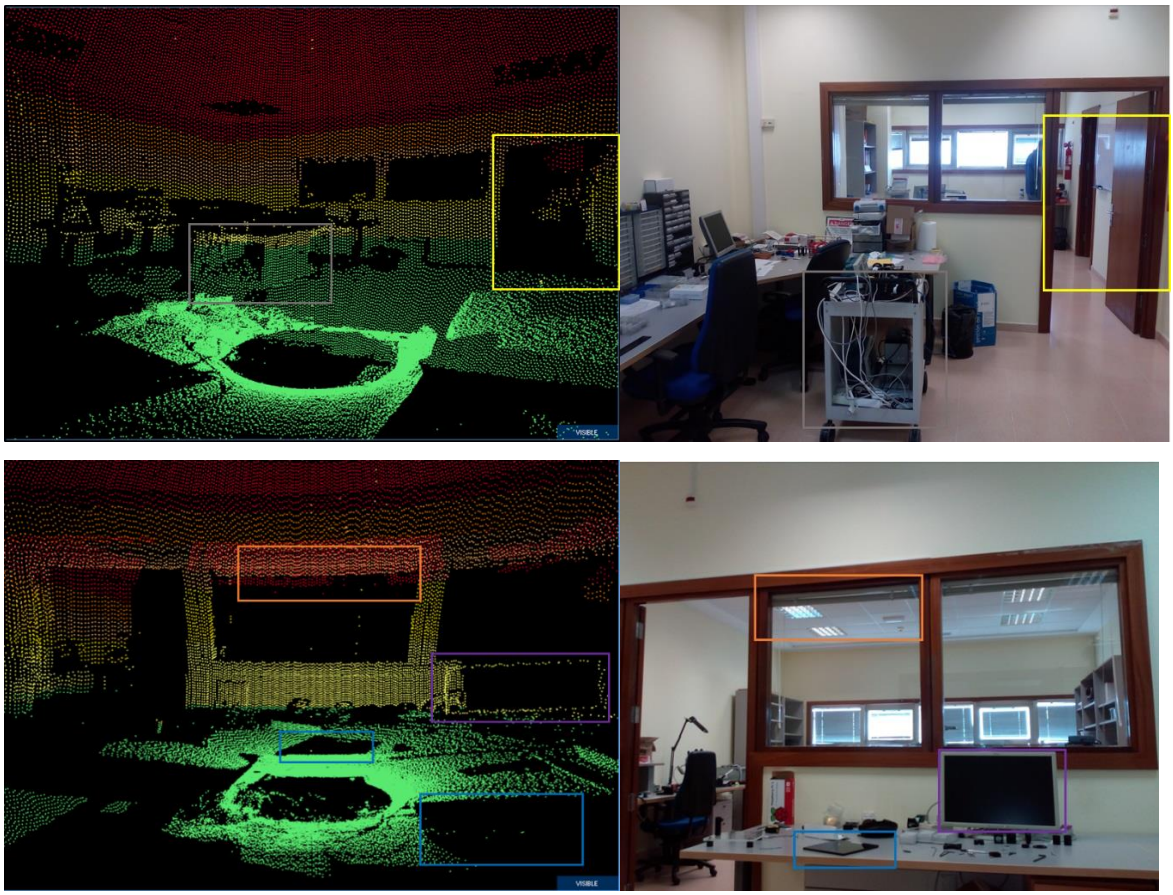


Figura 7.3. Representación tridimensional frente a imagen real de una sala pequeña.

Por último, en un pasillo que es un espacio más abierto (pasillo de entrada de la División COM del IUMA; Edificio Central del Parque Científico 1º Planta) y de mayor longitud (más de 6 metros) tal y como se ilustra en la Figura 7.4 no puede representar el fondo de la misma porque sobrepasa su rango (cuadro naranja). De igual forma se observa como representa los tablones de anuncio sin producirse difracción en los haces del láser (cuadro lila). Por último, indicar como debido al color de los asientos y el cubo de basura no puede representarlos, pero se observa claramente la figura que dejan (cuadro amarillo).

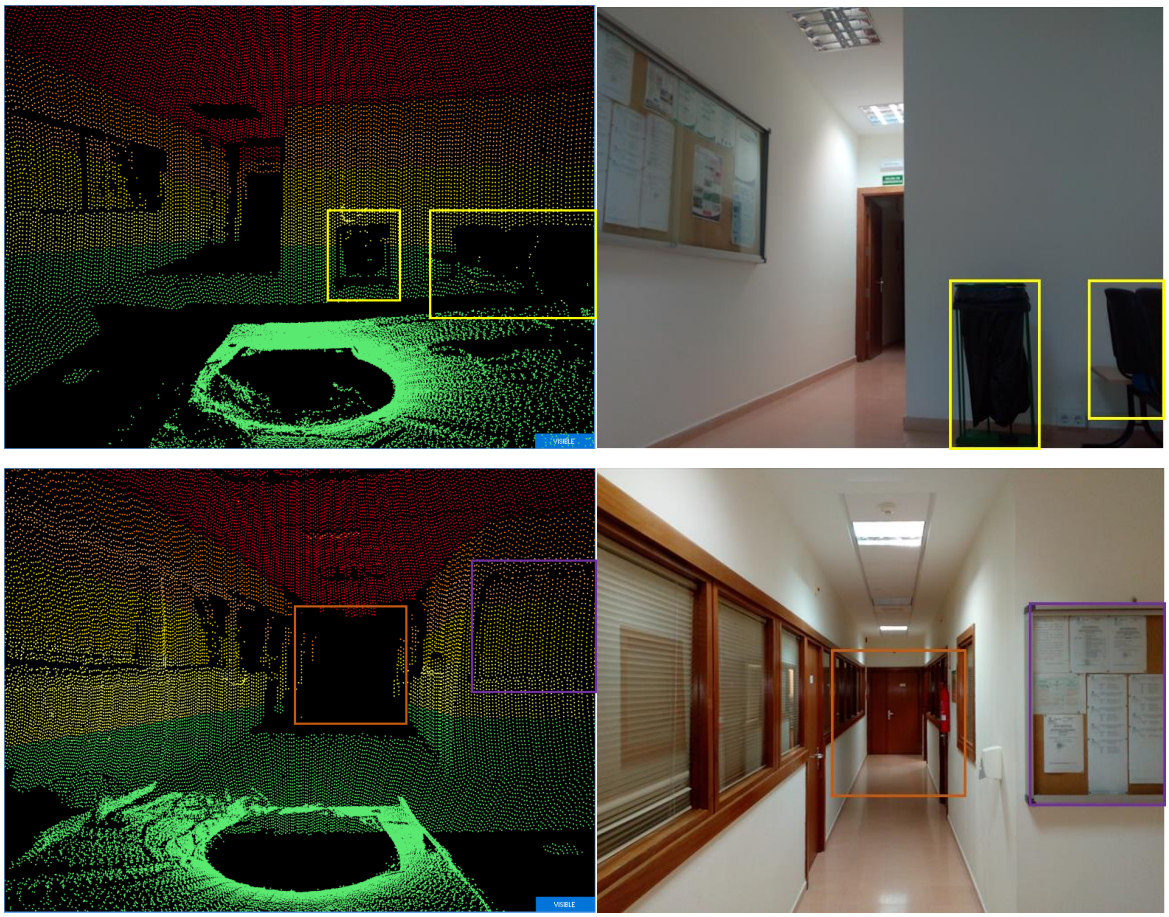


Figura 7.4. Representación tridimensional frente a imagen real de un pasillo.

En base a la representación de estos tres espacios diferentes se pueden sacar algunas conclusiones sobre el sistema láser diseñado:

- La representación por colores de las alturas da una sensación de tridimensionalidad y permite al usuario conocer la situación espacial de cada punto.
- La mejor representación de los espacios corresponde a figuras homogéneas y de gran densidad como pueden ser techos, paredes o el suelo.
- Cuando el objeto a escanear es blanco su representación es bastante cercana a la realidad, sin embargo, a medida que los objetos se van oscureciendo su representación es más difusa llegando a ser nula cuando es negro o muy cercano al mismo.
- Se comprueba como la presencia de cristales no produce difracción en el haz láser.

- Se verifica que la presencia o no de focos de luz activa (fluorescente) no afecta a la adquisición de medidas.

7.2 Revisión de objetivos

Con la realización de este proyecto se ha obtenido el conocimiento sobre los principios de medición láser, sus limitaciones y la normativa de seguridad que se debe tener en cuenta dependiendo del sistema láser que se emplee. Además, se han conocido las ventajas, fundamentos de funcionamiento y características principales de los sistemas *LiDAR* lo que ha sido muy útil para el posterior diseño del sistema de medición.

A partir de ello, se ha logrado diseñar un sistema mecánico que permite con lo ayuda de distintos tipos de motores poder realizar una adquisición de 360 º de un espacio. Para ello, ha sido necesario conocer las dimensiones de los distintos componentes (motores y RPLIDAR) para poder verificar su correcto funcionamiento teniendo en cuenta sus limitaciones mecánicas.

Por otra parte, se selecciona de forma adecuada un entorno de programación que permita recoger y almacenar (dando la oportunidad a una visualización offline del espacio) los datos *LiDAR* en tiempo real gracias al protocolo de comunicación implementado. A su vez, se diseña una interfaz gráfica que permite visualizar los datos recibidos y representar los puntos con distintos colores para poder interpretar su altura.

Por último, se hace una integración de las distintas etapas que, a pesar de ser un prototipo, es compacto pudiendo trasladarse para realizar las verificaciones y representaciones en distintos espacios cerrados.

Con todo ello, atendiendo a los objetivos planteados en el anteproyecto de este TFM y repasados al inicio de esta memoria se han cumplido satisfactoriamente los objetivos planteados.

8 Conclusiones y líneas futuras

Conseguir caracterizar un espacio cerrado con componentes electrónicos de precios reducidos como, por ejemplo, el *LiDAR* de medición (RPLIDAR A1) o el microcontrolador ARM Cortex-M4 permite lograr un diseño que, sin ser el que más prestaciones presenta, permite realizar representaciones tridimensionales y caracterizar un espacio.

Trabajar con sistemas láseres requiere un estudio previo que, se efectúa en este TFM, de las características de esta tecnología para permitir saber su rango de trabajo, su forma de trabajo, peligros para la vista humana o fuentes de errores que se deben conocer para poder actuar en este ámbito.

En general, esta forma de medición incrementa la dificultad del diseño electromecánico ya que se deben evitar las interferencias en su campo de visión (de otras piezas del diseño, por ejemplo) y, además, dicho sistema debe estar calibrado para poder realizar las medidas de forma adecuada. Sin embargo, con un conocimiento inicial limitado se diseña un sistema que permite, con ayuda de motores, realizar exploraciones de 360º sin producirse fricciones en ninguna de sus piezas. Con todo ello se logra disponer de un prototipo compacto con el que poder realizar las mediciones necesarias.

A su vez, a partir del firmware desarrollado se diseñan las tramas de comunicación adecuadas que permiten transmitir de forma eficiente los datos capturados a la aplicación de representación y realizar la comunicación entre ésta y el sistema mecánico.

El entorno de representación seleccionado presenta características que lo hacen interesante para este tipo de proyectos tales como su disponibilidad, su escalabilidad o su eficiencia. Conociendo esto, se diseña una aplicación que permite comunicarse con el sistema electromecánico para comenzar un escaneado y representar los datos a la vez que van llegando. Además, su representación en escala de colores permite situar las alturas de los distintos objetos.

La aplicación de visualización permite a medida que se reciben los datos guardarlos en un archivo para una posterior visualización offline lo que abre un abanico de posibilidades dependiendo de la funcionalidad que se le quiera dar como, por ejemplo, realizar un procesado software más exhaustivo para detectar regiones de interés. Para ello, también sería de interés disponer de un *LiDAR* con mayores prestaciones que permita caracterizar mejor los espacios. Para ello sería útil, con la ayuda de la aplicación de representación, disponer de una interfaz gráfica de fácil manejo que permita dimensionar la calidad de su escaneado pudiendo cambiar la resolución azimutal (stepper) o del ángulo de elevación (RPLIDAR).

Por último, una mejora que le daría un valor añadido al sistema desarrollado sería la posibilidad de ser manejado en otras plataformas. Para ello, la herramienta de representación presenta modos de trabajo para un desarrollo web (p5.js [65] y Python [66]) o para su representación en terminales móviles mediante su adaptación para Android [67] (incluso introduciéndose en el entorno de trabajo de Android Studio).

Presupuesto

En este capítulo se desarrolla el presupuesto de este Trabajo Fin de Máster, que permite valorar los costes que han supuestos los materiales y herramientas utilizados durante su elaboración, además del coste del trabajo realizado. Para ello, los puntos tratados en este capítulo en relación al cálculo del coste total que supone este trabajo son los siguientes: recursos materiales (tanto hardware como softwares), material fungible, trabajo tarifado por tiempo empleado y aplicación de impuestos y coste total.

Recursos materiales

Para la ejecución de este Trabajo Fin de Máster han sido necesarios tanto herramientas hardware como herramientas software para configurar los dispositivos o el entorno office para la redacción de la memoria.

Recursos Hardware

En la tabla Presupuesto. Tabla 1 se incluye pormenorizado el coste de los recursos hardware que se han usado para este proyecto tanto los de uso exclusivo para este TFM como los que son compartidos en otro tipo de investigaciones tales como aparatos de medida u ordenadores.

Indicar que el período estimado de amortización de un equipo electrónico es de 24 meses. Para el desarrollo de este TFM se emplean 4 meses, sin embargo, algunos de recursos se emplean menos tiempo. Señalar que otros como la tarjeta de desarrollo, motores u otros dispositivos que se hayan comprado expresamente para el desarrollo de este TFM no presentan período de amortización.

Hardware	Precio unitario	Unidades	Período de amortización (meses)	Tiempo de uso (meses)	Importe
STM32F4 discovery board	18,5 €	1	-	-	18,5 €
Motor stepper Nema-17	14,86 €	1	-	-	14,86 €
Driver A4988 Pololu	5,92 €	1	-	-	5,92 €
Sensor de efecto Hall DRV5053VAQLPG	1,22 €	3	-	-	3,66 €
Rodamiento axial de bolas 51108 SKF	11,01 €	1	-	-	11,01 €
Iman radial magnetic 8189	0,2 €	3	-	-	0,6 €
RPLIDAR A1	174,5 €	1	-	-	174,5 €
Motor servo Parallax	12,28 €	1	-	-	12,28 €
Cable mini-USB	1,26 €	1	-	-	1,26 €
Cable micro-USB	2,5 €	1	-	-	2,5 €
Cable de prototipado	-	-	-	-	5,3 €
Tornillería de varias métricas	-	-	-	-	3,2 €
Protobard	4,52	1	24	4	0,75 €
Osciloscopio PicoScope 3224	590,66 €	1	24	4	98,44 €
Impresora 3D Prusa i3	1.050 €	1	24	3	131,25 €
Fuente de alimentación Promax FAC-662B	556,9 €	1	24	4	92,81
Sonda CP-260 60 MHz	19,55 €	2	24	4	6,51 €
Ordenador Intel i3-220 con tarjeta gráfica ASUSP8875V	1.230,45 €	1	24	4	205,07 €
Pantalla Sun Microsystems 22" LCD	284,25 €	1	24	4	47,38 €
TOTAL	**835,8 €**				

Presupuesto. Tabla 1. Costes de recursos hardware.

Recursos Software

Las herramientas software que han sido necesarias para la ejecución de este proyecto se desgranan en la tabla Presupuesto. Tabla 2. Indicar que el período estimado de amortización de un software electrónico es de 12 meses ya que el pago de las licencias es anual. Al igual que para los recursos hardware existen ciertas herramientas software que no son empleadas a lo largo de la totalidad del proyecto.

Software	Coste	Período de amortización (meses)	Tiempo de uso (meses)	Importe
Keil uVision 4.73	8.538,10 €	12	4	2.846,03 €
Solidworks 2014	6.230 €	12	3	1.557,5 €
Microsoft Office 2016	539 €	12	4	179,6 €
Processing 3	0 €	-	-	0 €
TOTAL	**4.583,13 €**			

Presupuesto. Tabla 2. Coste de recursos software.

Trabajo tarifado por tiempo empleado

Para calcular el salario que debe percibir el autor de este Trabajo Fin de Máster, se toma como referencia la Tabla de Contrataciones de Investigadores con cargo a Proyectos de Investigación de la Universidad de Las Palmas de Gran Canaria, aprobado el 21 de julio de 2010 y publicada posteriormente en el Boletín Oficial de la ULPGC [68]. Se asume que el autor de este TFM se enmarca dentro de la categoría de investigador en proyecto a tiempo parcial (20 horas semanales), con una titulación de ingeniero, de modo que su salario mensual es de 1.192,99 €.

Dado que este proyecto tiene una duración estipulada de 300 horas, en la tabla Presupuesto. Tabla 3 se muestra el sueldo total que debe ganar el autor de este trabajo en función del número de horas dedicadas y el sueldo establecido por la institución mencionada.

Categoría	Sueldo mensual base	Número de horas trabajadas	Coste de hora trabajada	Coste final
Ingeniero	1.192,99 €	300	14,91 €	**4.473 €**

Presupuesto. Tabla 3. Trabajo tarifado por tiempo empleado.

Material fungible

Además de los recursos hardware y software, en este proyecto, se han utilizado otros materiales como es: plástico de impresión 3D (PLA), los folios, impresión y encuadernación tal y como se observa en la tabla Presupuesto. Tabla 4.

Materiales	Costes
Plástico PLA 1,75 mm (negro y rojo)	50 €
Pegamento	2,1 €
Folios	12 €
Impresión	40 €
Encuadernación	8 €
TOTAL	**112,1 €**

Presupuesto. Tabla 4. Coste de materiales fungibles.

Aplicación de impuestos y coste total

Dado que este Trabajo Fin de Máster se realiza en una institución integrada en la Comunidad Autónoma de Canarias, sobre los costes totales se debe aplicar el Impuesto General Indirecto Canario (I.G.I.C.), que se establece en un 7% sobre los costes asociados al mismo.

El coste total del proyecto, antes de aplicarle los correspondientes impuestos, asciende 10.004,03 € a lo que hay que sumarle el 7% de IGIC tal y como se observa en la tabla Presupuesto. Tabla 5.

Descripción	Coste
Recursos hardware	835,8 €
Recursos software	4.583,13 €
Trabajo tarifado por tiempo empleado	4.473 €
Material fungible	112,1 €
Subtotal	10.004,03 €
Aplicación de impuestos (IGIC 7%)	700,28 €
TOTAL PRESUPESTO	**10.704,31 €**

Presupuesto. Tabla 5. Costes totales del TFM.

Presupuesto

El presupuesto total asciende a la cantidad de *diez mil setecientos cuatro euros con treinta y un céntimos (**10.704,31 €**)*.

Las Palmas de Gran Canaria a 17 de Julio del 2017.

Fdo.: Alejandro Santana Pérez.

Bibliografía

[1] «Global automotive LiDAR sensor sales 2020 | Statistic». [En línea]. Disponible en: https://www.statista.com/statistics/430086/automotive-sales-of-automotive-lidar-systems-worldwide/.

[2] R. Domínguez, E. Onieva, J. Alonso, J. Villagra, y C. González, «LIDAR based perception solution for autonomous vehicles», *Int. Conf. Intell. Syst. Des. Appl. ISDA*, pp. 790-795, 2011.

[3] A. Ibisch, S. Stumper, H. Altinger, M. Neuhausen, M. Tschentscher, M. Schlipsing, J. Salinen, y A. Knoll, «Towards autonomous driving in a parking garage: Vehicle localization and tracking using environment-embedded LIDAR sensors», *IEEE Intell. Veh. Symp. Proc.*, n.º Iv, pp. 829-834, 2013.

[4] J. Choi, S. Ulbrich, B. Lichte, y M. Maurer, «Multi-Target Tracking using a 3D-Lidar sensor for autonomous vehicles», *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, n.º Itsc, pp. 881-886, 2013.

[5] D. Jeon, C. K. Kim, y J. Kim, «Dynamic object tracking system», *2015 12th Int. Conf. Ubiquitous Robot. Ambient Intell. URAI 2015*, n.º Urai, pp. 512-515, 2015.

[6] M. Bui, V. Frémont, D. Boukerroui, y P. Letort, «Multi-sensors people detection system for heavy machines», pp. 867-872, 2014.

[7] M. Quigley, S. Batra, S. Gould, E. Klingbeil, Q. Le, A. Wellman, y A. Y. Ng, «High-accuracy 3D sensing for mobile manipulation: Improving object detection and door opening», *Robot. Autom. 2009. ICRA '09. IEEE Int. Conf.*, pp. 2816-2822, 2009.

[8] J. Counsell, S. Smith, y a. Richman, «Overcoming some of the issues in maintaining large urban area 3D models via a web browser», *Tenth Int. Conf. Inf. Vis.*, pp. 2-7, 2006.

[9] M. Bosse y R. Zlot, «Place recognition using keypoint voting in large 3D lidar datasets», *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 2677-2684, 2013.

[10] D. Stott, D. S. Boyd, A. Beck, y A. Cohn, «Hyperspectral detection dynamics of archaeological vegetation marks and enhancement using full waveform LiDAR data», *Int. Geosci. Remote Sens. Symp.*, pp. 2829-2831, 2013.

[11] A. M. Richter, F. Kuester, T. E. Levy, y M. Najjar, «Terrestrial laser scanning (LiDAR) as a means of digital documentation in rescue archaeology: Two examples from the Faynan of Jordan», *Proc. 2012 18th Int. Conf. Virtual Syst. Multimedia, VSMM 2012 Virtual Syst. Inf. Soc.*, pp. 521-524, 2012.

[12] P. Wu, S. Xie, H. Liu, J. Luo, y Q. Li, «A Novel Algorithm of Autonomous Obstacle - avoidance for Mobile Robot Based on LIDAR Data», n.º 1, pp. 2377-2382, 2015.

[13] B. Chu, *Laser light scattering : basic principles and practice*. Dover Publications, 2007.

[14] Hamamatsu, «Introduction of Light», *Opto-Semiconductor Handb.*, pp. 2-7, 2014.

[15] «An Overview of LIDAR for Urban Applications».

[16] J. C. Suarez, «LiDAR Applications», 2005.

[17] M. U. Khasenov, «Luminescence spectra of active media of lasers on visible and near infrared transitions of inert gases under ion beam excitation», en *2016 IEEE 7th International Conference on Advanced Optoelectronics and Lasers (CAOL)*, 2016,

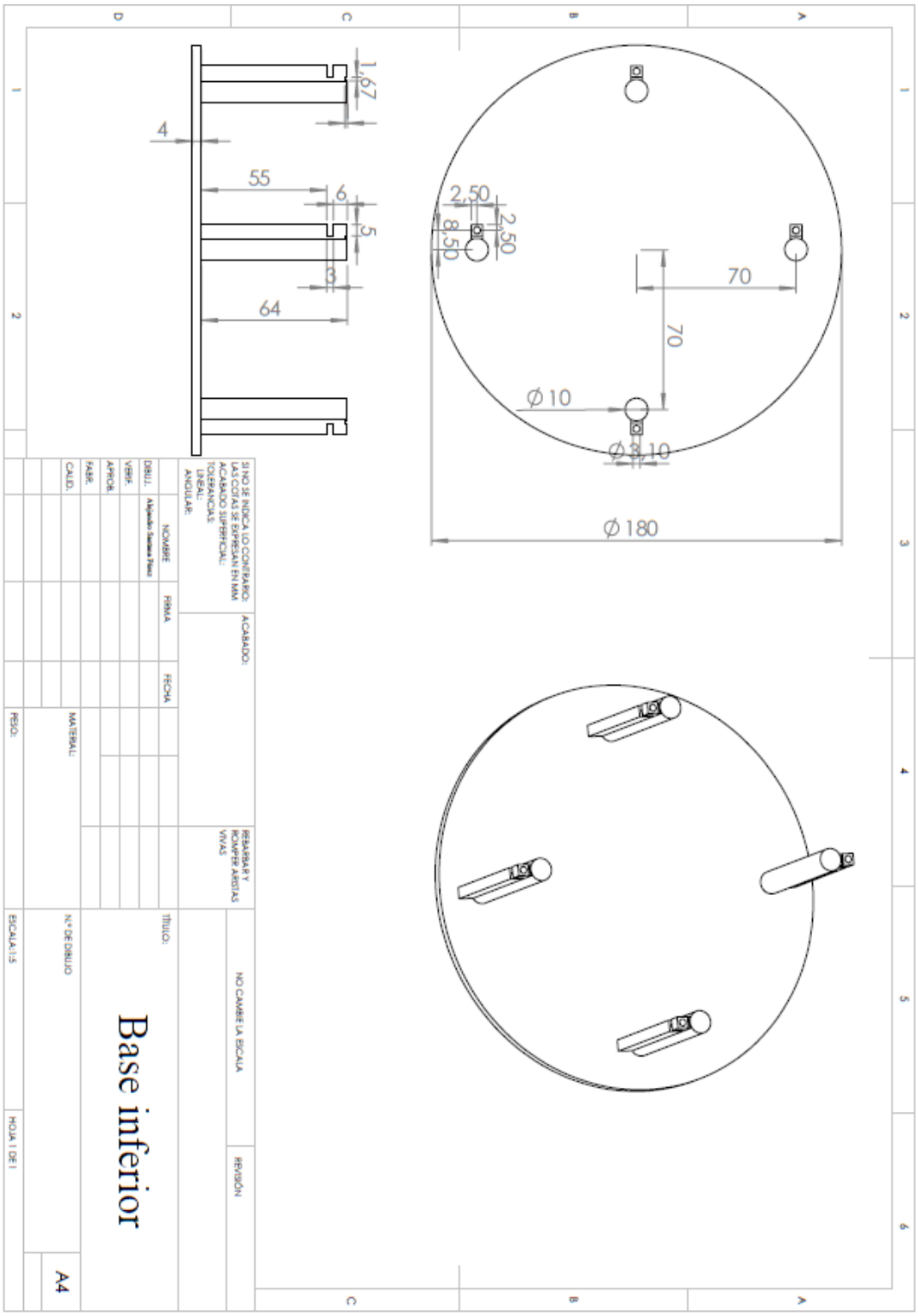
pp. 93-95.
[18] D. N. Wang, G. Dick, S. Chen, K. T. V. Grattan, y A. W. Palmer, «The use of short coherence length laser light for eye length measurement», en *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 1992, pp. 340-341.
[19] N. Xiaolong y L. Zhi, «A method to adjust the focusing and coherence characteristics of laser beam complexly using a liquid crystal spatial light modulator», en *2015 International Conference on Optoelectronics and Microelectronics (ICOM)*, 2015, pp. 38-42.
[20] E. Hernández y UAM, «El teorema de Thales y el Teorema de Pitágoras», 2010.
[21] Y. Li, Y. Ruichek, y C. Cappelle, «3D triangulation based extrinsic calibration between a stereo vision system and a LIDAR», *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, pp. 797-802, 2011.
[22] M. Azizi y E. Tarshizi, «Autonomous control and navigation of a lab-scale underground mining haul truck using LiDAR sensor and triangulation - Feasibility study», *IEEE Ind. Appl. Soc. 52nd Annu. Meet. IAS 2016*, pp. 1-6, 2016.
[23] M. U. Piracha, D. Nguyen, y P. J. Delfyett, «High resolution chirped pulse lidar with spectral phase modulation for two fold improvement in range resolution», en *IEEE Avionics, Fiber-Optics and Photonics Digest CD*, 2012, pp. 42-43.
[24] G. Kim, J. Eom, y S. Korea, «Analysis on the characteristics of mutual interference between pulsed terrestrial LiDAR scanners», pp. 2151-2154, 2015.
[25] L. Wu, J. Xu, X. Yang, W. Lv, Q. Zhang, J. Yan, Y. Zhang, y Y. Zhao, «Range resolution improvement of range gated lidar system by phase coded method», en *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 2016, pp. 6205-6208.
[26] H. Kawata, A. Ohya, S. Yuta, W. Santosh, y T. Mori, «Development of ultra-small lightweight optical range sensor system», *2005 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS*, n.º 42 50, pp. 3277-3282, 2005.
[27] W. Boehler y A. Marbs, «Investigating Laser Scanner Accuracy».
[28] J. Clark y S. Robson, «Accuracy of measurements made with a Cyrax 2500 laser scanner against surfaces of known colour».
[29] D. D. Lichti y S. J. Gordon, «Error Propagation in Directly Georeferenced Terrestrial Laser Scanner Point Clouds for Cultural Heritage Recording FIG Working Week 2004 Error Propagation in Directly Georeferenced Terrestrial Laser Scanner Point Clouds for Cultural Heritage Recording».
[30] Jorge Márquez Flores ; UNAM, «Ruido en datos».
[31] H. Weichel, *Laser beam propagation in the atmosphere*. SPIE Optical Engineering Press, 1990.
[32] International Electrotechnical Commission, «International Standard - IEC 60825-1 - Safety of Laser Products», p. 122, 2001.
[33] Slamtec, «Rplidar A1 Introduction and Datasheet», pp. 1-15, 2016.
[34] Pololu, «Stepper Motor: Bipolar, 200 Steps/Rev, 42×38mm, 2.8V, 1.7 A/Phase». [En línea]. Disponible en: https://www.pololu.com/product/2267.
[35] Pololu, «A4988 Stepper Motor Driver Carrier». [En línea]. Disponible en: https://www.pololu.com/product/1182. [Accedido: 30-jun-2017].
[36] Parallax Inc, «Parallax Continuous Rotation Servo Datasheet», n.º 916, pp. 5-10, 2009.

[37] Texas Instruments, «DRV5053 Analog-Bipolar Hall Effect Sensor», 2014.
[38] Radial Magnet Inc, «Iman». [En línea]. Disponible en: https://www.digikey.es/product-detail/es/radial-magnet-inc/8189/469-1059-ND/5400498. [Accedido: 30-jun-2017].
[39] STMicroelectronics, «Reference Manual STM32F4XX», n.º October 2015, 2012.
[40] STMicroelectronics, «Guía de iniciación al kit de evaluación St STM32F4 Discovery», pp. 1-24, 2012.
[41] STMicroelectronics, «User manual Getting started with software and firmware environments for the STM32F4DISCOVERY Kit», n.º September, pp. 1-46, 2011.
[42] Hitex, «Insider ' s Guide STM32 To The STM32 ARM Based Microcontroller».
[43] STMicroelectronics, «STM32F407xx Spec Sheet», n.º October, 2015.
[44] Slamtec, «Rplidar A1 Development Kit User Manual», pp. 1-16, 2016.
[45] Allegro, «A4988 DMOS Microstepping Driver with Translator And Overcurrent Protection».
[46] R. Eisenbeis, «Understanding & Applying Hall Effect Sensor Datasheets», n.º July, pp. 1-7, 2014.
[47] Radial Magnet Inc, «Iman», p. 1.
[48] SKF, «Dimensiones del rodamiento axial», p. 102, 2017.
[49] Changzhou Songyang Machinery & Electronics, «High torque hybrid stepping motor specifications», 2009.
[50] P. Acarnley, «Stepping Motors and practice».
[51] D. W. Jones, *Control of Stepping Motors A Tutorial*. 2001.
[52] STMicroelectronics, «STM32CubeF4», 2017.
[53] Slamtec «Interface Protocol and Application Notes», pp. 1-32, 2016.
[54] M. Z. Hussain y K. N. Parvin, «Q-Format Data Representation and Its Arithmetic», *IJECT*, vol. 7, n.º 2.
[55] U. of Kansas, «Spherical Coordinates», vol. 122, n.º 1, pp. 1-7.
[56] «Códigos ASCII - Tabla de caracteres y simbolos ascii». [En línea]. Disponible en: http://ascii.cl/es/. [Accedido: 12-jul-2017].
[57] «snprintf - C++ Reference». [En línea]. Disponible en: http://www.cplusplus.com/reference/cstdio/snprintf/. [Accedido: 12-jul-2017].
[58] «Processing.org». [En línea]. Disponible en: https://processing.org/. [Accedido: 03-jul-2017].
[59] Casey Reas and Ben Fry, *Getting Started with Processing*. 2010.
[60] C. Reas y B. Fry, *Processing: A Programming Handbook for Visual Designers and Artists*. 2007.
[61] D. Shiffman, *Learning Processing A Beginner's Guide to Programming Images, Animation, and Interaction*. 2008.
[62] «processing GUI, controlP5». [En línea]. Disponible en: http://www.sojamo.de/libraries/controlP5/. [Accedido: 03-jul-2017].
[63] «peasycam». [En línea]. Disponible en: http://mrfeinberg.com/peasycam/. [Accedido: 03-jul-2017].
[64] «HE_Mesh – W:Blut – Creative Coding». [En línea]. Disponible en: http://www.wblut.com/he_mesh/. [Accedido: 03-jul-2017].
[65] «p5.js | home». [En línea]. Disponible en: https://p5js.org/. [Accedido: 16-jul-2017].
[66] «Python Mode for Processing». [En línea]. Disponible en: http://py.processing.org/. [Accedido: 16-jul-2017].

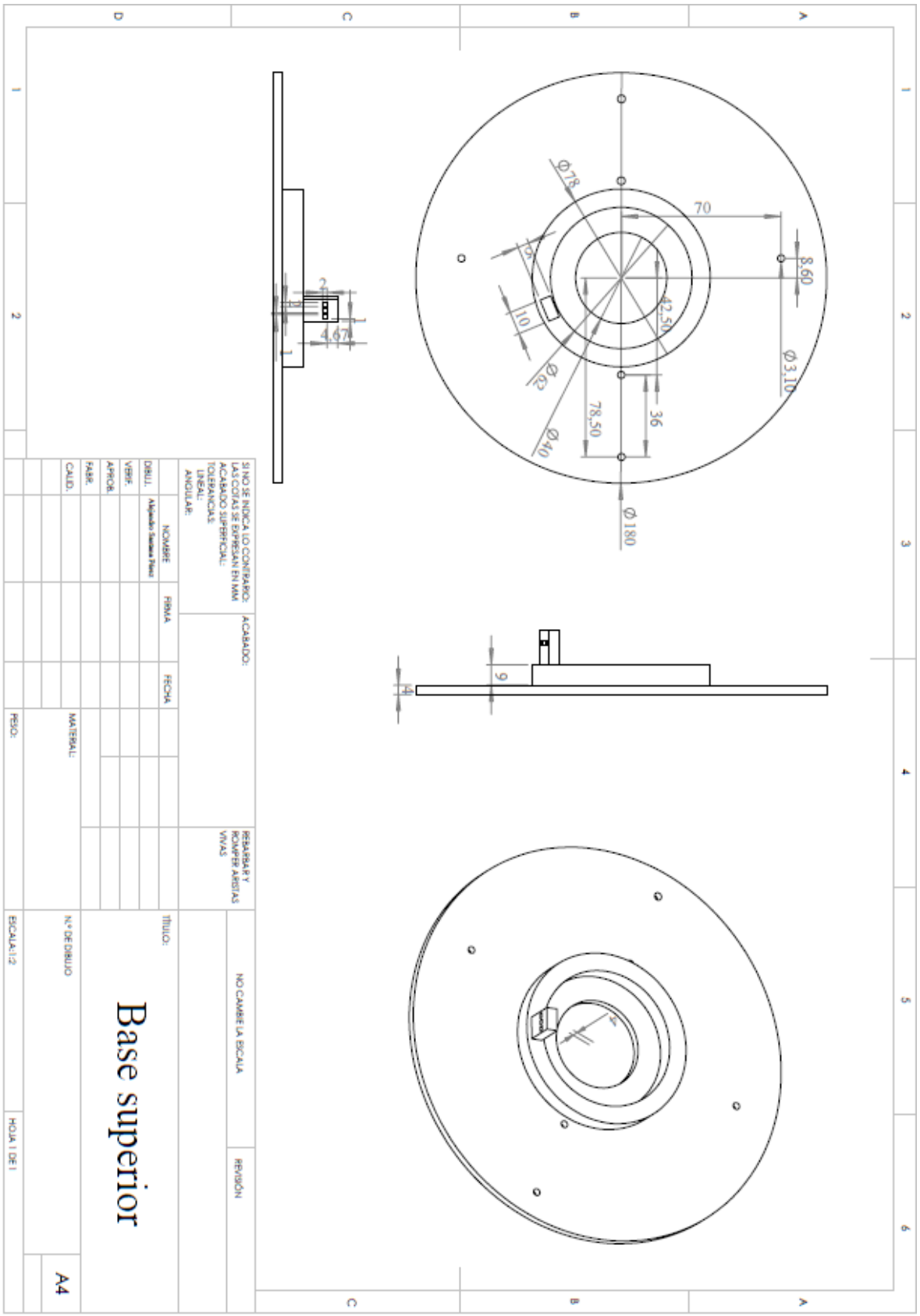
[67] «Processing for Android». [En línea]. Disponible en: http://android.processing.org/. [Accedido: 16-jul-2017].

[68] Universidad de Las Palmas de Gran Canaria, «Tabla de Contrataciones de Investigadores con cargo a Proyectos de Investigación de la Universidad de Las Palmas de Gran Canaria», en *BOULPGC*, 2010.

ANEXO 1

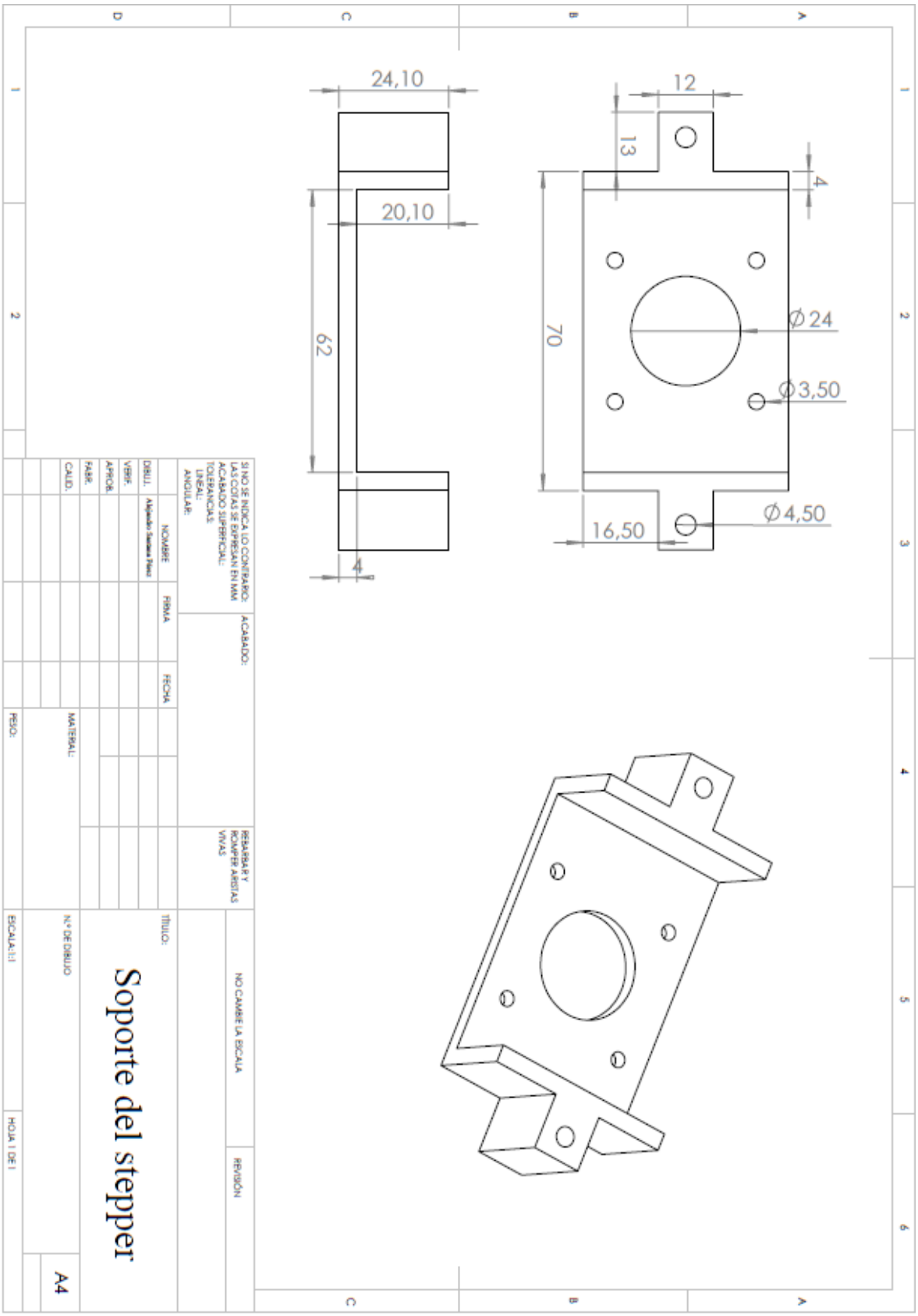


Anexo 1. Figura 1. Plano de la base inferior.

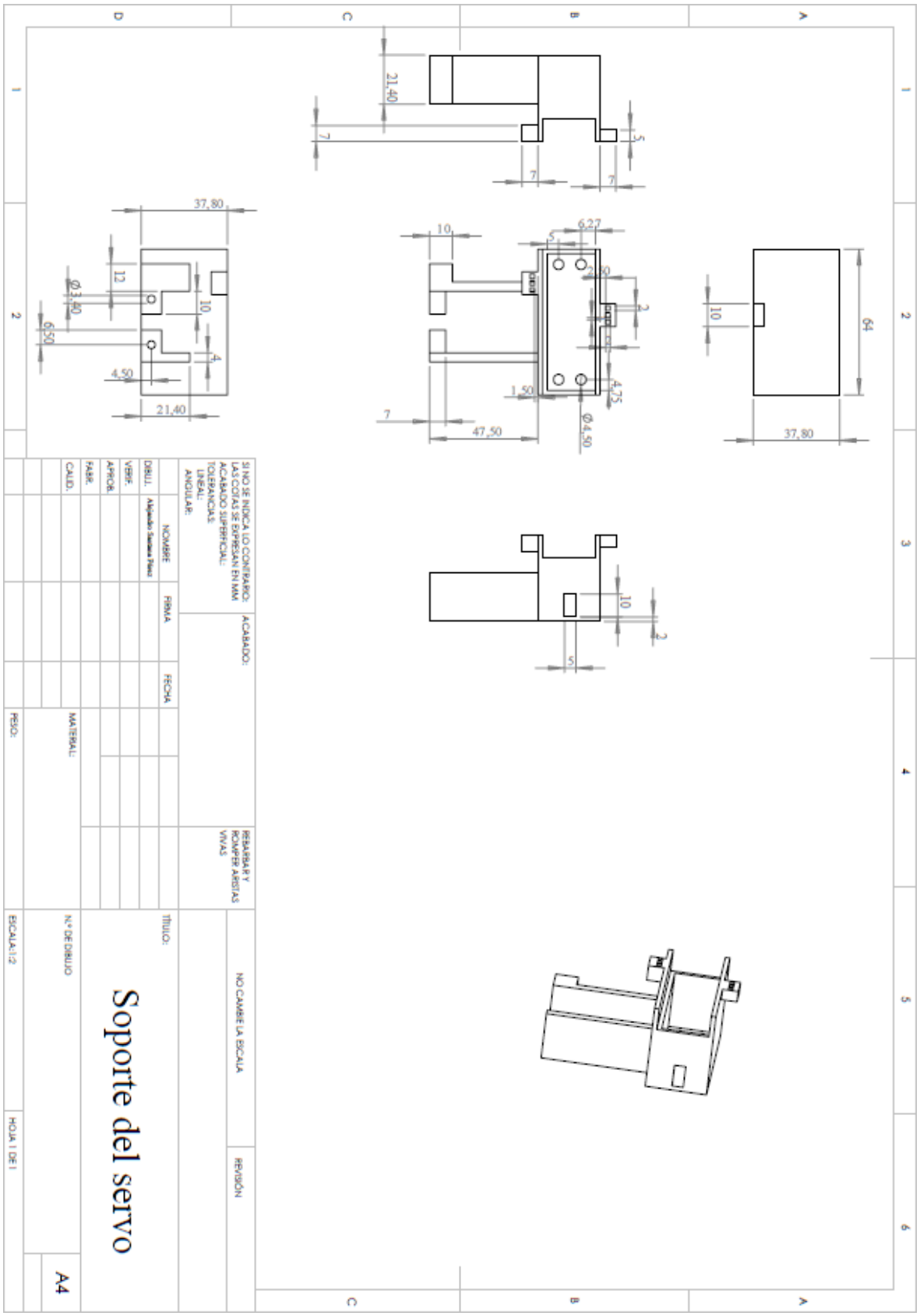


Anexo 1. Figura 2. Plano de la base superior.

ANEXO 1

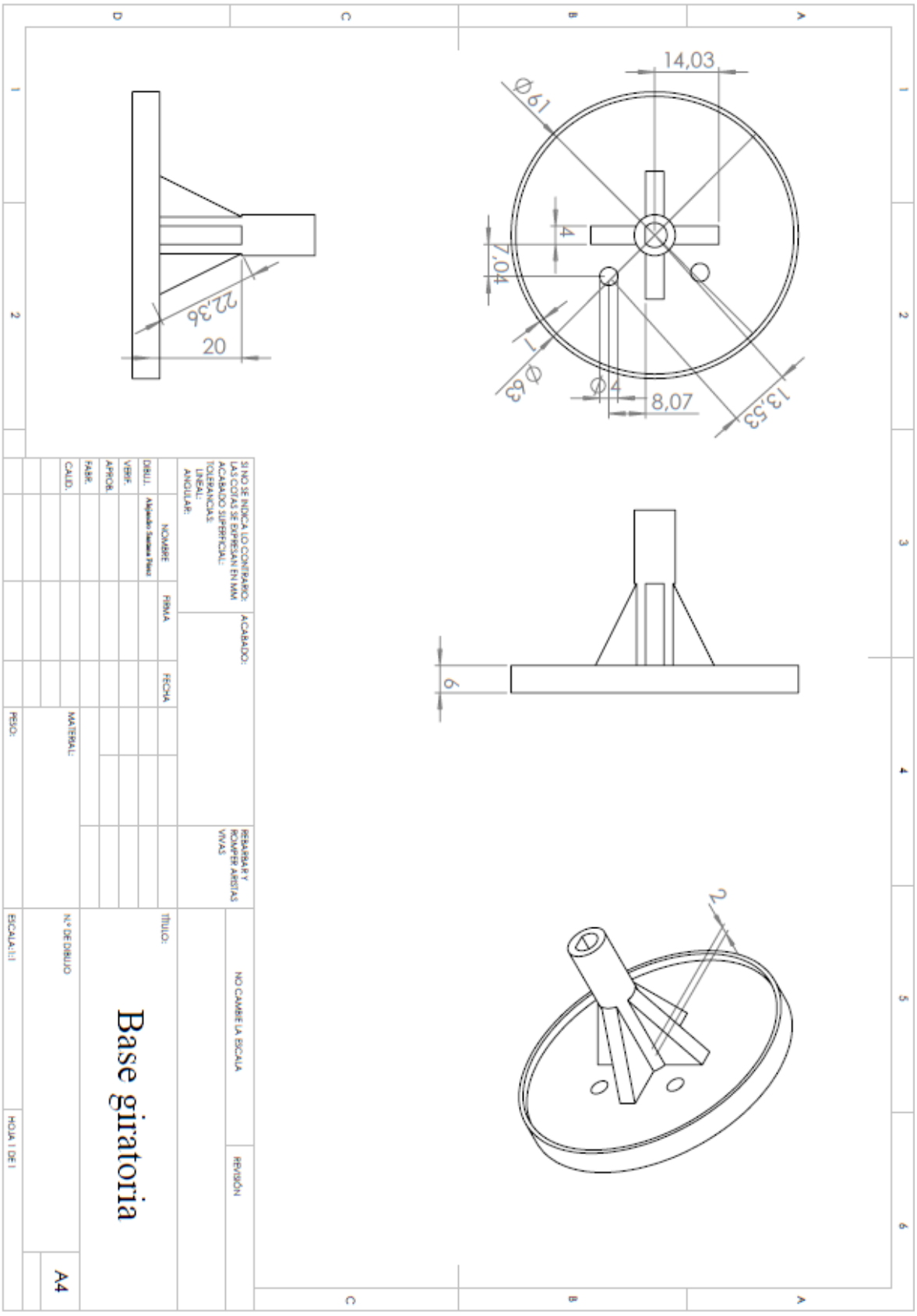


Anexo 1. Figura 3. Plano del soporte del stepper.

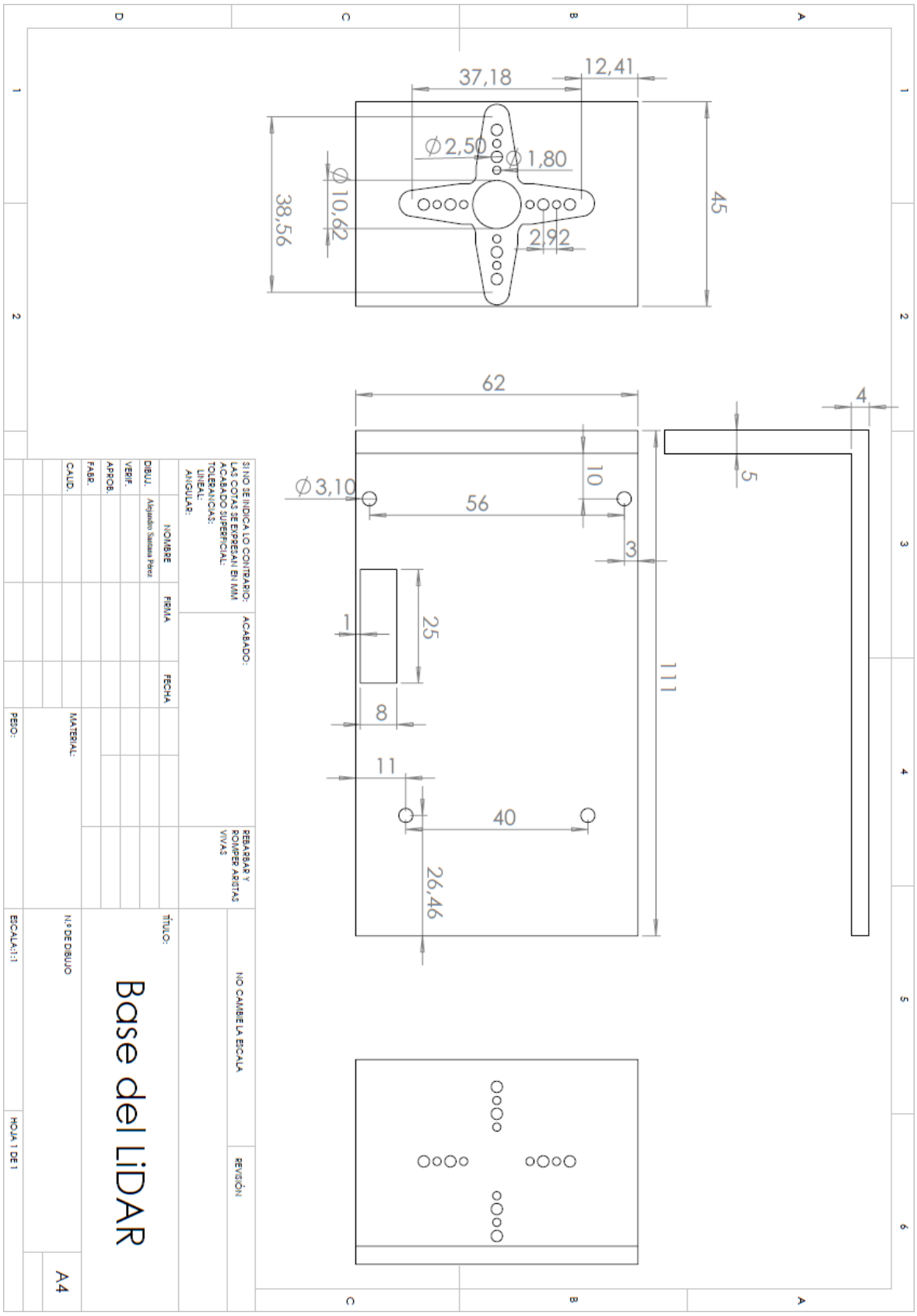


Anexo 1. Figura 4. Plano del soporte del servo.

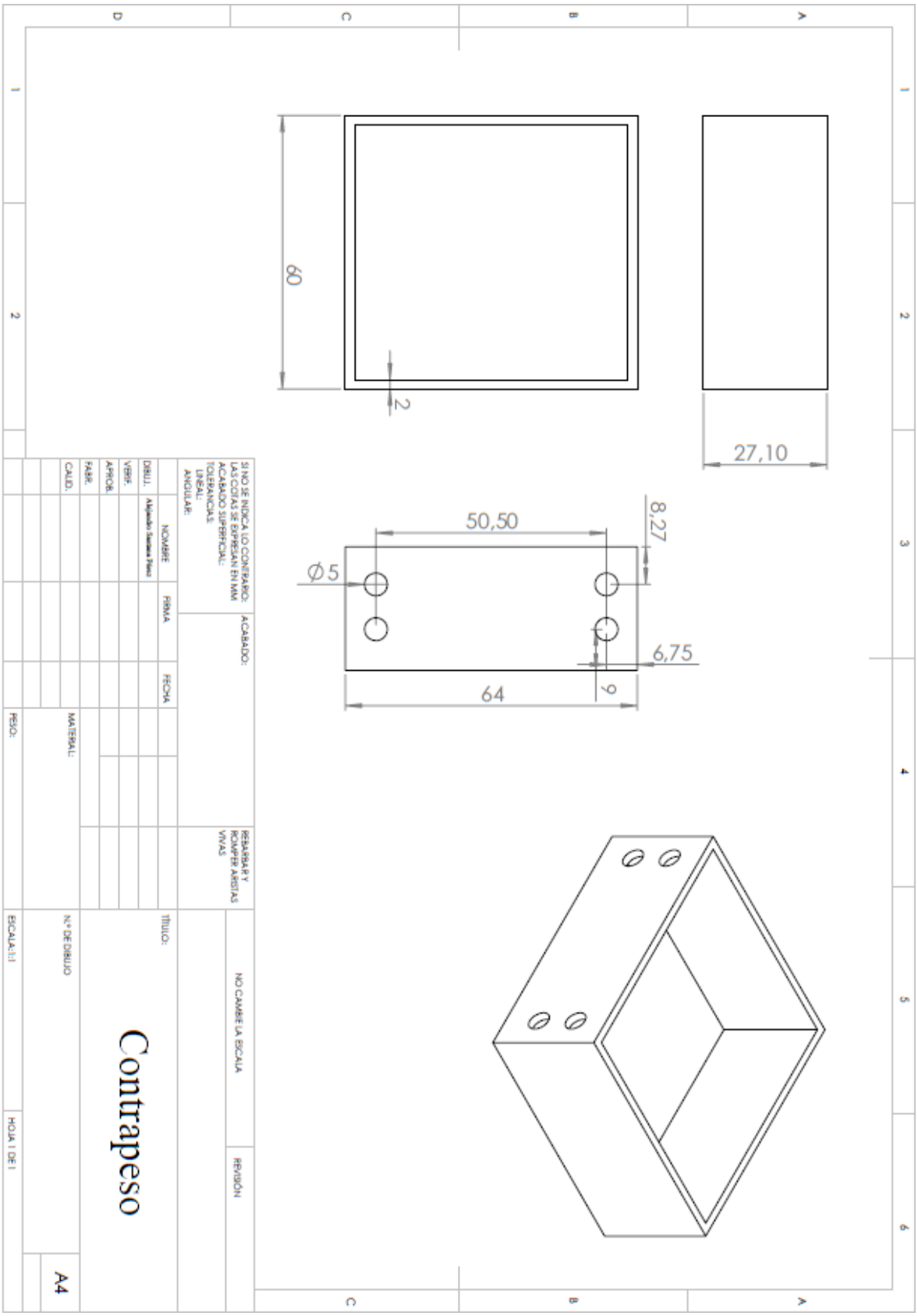
ANEXO 1



Anexo 1. Figura 5. Plano de la base giratoria.



Anexo 1. Figura 6. Plano de la base del LiDAR.



Anexo 1. Figura 7. Plano del contrapeso.

ANEXO 1

ANEXO 2

```
void secuenciaComienzo(){
  if(sec==0){ // 1 Get_Health
     if(flag_count_reset==0){
        count_Reset=0;
        flag_count_reset++;
        count_bytes=0;
        count_cabecera=0;
        tim_noenvio=0;
     }else if ((buffer_health[0]==0x00)&&(count_bytes==9)&&(flag_health)){
        count_bytes=0;
        flag_health=0;
        pos_buffer_health=0;
        count_cabecera=0;
        sec++;
        countStep=0;
        if(stop_stepper==1){//Cuando ya ha escaneado directo a SCAN
           stop_stepper=0;
           sec=2;
           flag_count_reset=2;
           flag_USART=2;
        }
     }else if (count_Reset==10){
        if(flag_USART==0){
           flag_int=1;
           flag_USART++;
           count_id=0;
        }
        count_USART=0;
        count_bytes=0;
        count_cabecera=0;
     }
  }else if (sec==1){ // 2 Get_Sample_Rate
     if(GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_15)!=1){
        StepperConf(30,800,4,0);
        tim_noenvio=0;
     }else
if(GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_15)==1&&GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_1
4)!=1){
        countStep=0;    //Reseteo variable stepper
        if(GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_14)!=1){
           TIM4_Enable(); //Se activa timer del PWM del servo
           TIM4->CCR3=770;//Antihorario
           tim_noenvio=0;
        }
     }else if((flag_count_reset==1)&&GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_14)==1){
        count_Reset=0;
        flag_count_reset++;
        count_bytes=0;
        count_cabecera=0;
        tim_noenvio=0;
        TIM4_Disable(); //Se desactiva timer del PWM del servo
        //1.5 ms (quieto)
        TIM4->CCR3=764;
```

```
    }else if ((count_bytes==11)&&(flag_freq)){
       count_bytes=0;
       flag_freq=0;
       pos_buffer_freq=0;
       count_cabecera=0;
       sec++;
    }else if (count_Reset==10){
       if(flag_USART==1){
          flag_int=1;
          flag_USART++;
          count_id=0;
       }
       count_USART=0;
    }
  }else if (sec==2){ // Get_Scan
    if(flag_count_reset==2){
       count_Reset=0;
       flag_count_reset++;
       count_bytes=0;
       count_cabecera=0;
    }else if (count_Reset==2){
       if(flag_USART==2){
          flag_USART++;
          flag_int=1;
          GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_SET); // Activar movimiento motor
          count_id=0;
       }
       count_USART=0;
    }
  }
}
```

Anexo 2. Código 1. Función secuenciaComienzo.

```
void verificMove(){
  if(((dato_S==1)&&(dato_S_neg==0))||(move_stepper)){
    nueva_vuelta++;
    if(nueva_vuelta==2){
       count_Freq_Lidar=0;
       nuevo_dato=0;
       flag_cont_angle=0;
    }else if(((nueva_vuelta>2)&&(flag_correct==0))||(flag_cont_angle<40)){
       //Se reinicia secuencia cuando la primera medida estuviera fuera del rango
establecido o menos de 40 medidas
       nueva_vuelta=0;
       flag_cont_scan=0;
       flag_scan=0;
       pos_buffer_datos_1=pos_buffer_datos;
       flag_reScan=1;
       flag_cont_angle=0;

    }else if((nueva_vuelta>2)&&(flag_correct==1)){
       if(flag_scan==1){
          countStep=0;  //Reseteo variable del stepper la primera vez que entre
          flagHall=0;

          flag_scan=0;
          StepperConf(2,1,4,flag_antihorario);
          vuelta_float=(float)vuelta_horizontal/2;
```

```
          vuelta_uint=vuelta_horizontal/2;
          if(vuelta_uint==vuelta_float){//Vueltas pares y primera
             //Bit de dirección motor
             //Reset--->Sentido horario
             //Set---> Sentido antihorario
             //GPIO_WriteBit(GPIOB, GPIO_Pin_7, Bit_RESET);
             flag_antihorario=0;
          }else if(vuelta_uint!=vuelta_float) {//Vueltas impares
             flag_antihorario=1;
          }
       }
       if(stepper_new==0){
          move_stepper=1;//A partir de ahora se entra porque hay movimiento del
stepper ya no datos de scan
          sec=3;//Stop al LidAR
          stepper_new=1;
          pos_stepper++;//Se incrementa una posicion del Stepper que se almacenara
(nueva medida)
       }else
if((((GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_15)!=1)||(pos_stepper<=200))&&(flag_45grado
s==0))&&(flagHall==1)){//Cuando NO se llega a una nueva vuelta del stepper

nueva_vuelta=0;
          stepper_new=0;
          stop_stepper=1;
          move_stepper=0;
          flag_cont_scan=0;
          flag_scan=0;
          pos_buffer_datos_1=pos_buffer_datos;
          flag_cont_angle=0;
          flag_correct=0;
          flag_45grados=0;
          countStep=0;
          flagHall=0;

       }else
if((((GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_15)==1)&&(pos_stepper>200))||(flag_45grados
==1))&&(flagHall==1)){//Cuando SI se llega a una nueva vuelta del stepper.
          nueva_vuelta=0;
          pos_stepper=0;/
          stop_stepper=1;
          stepper_new=0;
          move_stepper=0;
          flag_cont_scan=0;
          flag_scan=0;
          pos_buffer_datos_1=pos_buffer_datos;

          vuelta_horizontal++;
          flag_correct=0;
          flag_cont_angle=0;
          flag_45grados=0;
          countStep=0;
          flagHall=0;

          flag_primeravez=1;
       }
    }
  }
}
```

Anexo 2. Código 2. Función verificMove.

```
void envioUSB(){
  if((flag_USB)&&(nueva_vuelta==2)){
     //1º Cabecera
     txbuffer[0]=0x5A;
     //2º Desplazamiento vertical LiDAR
     //Se convierte a string (char)
     snprintf(buffer_vert_LIDAR_USB,4,"%d",buffer_scan[pos_buffer_scan-2]);
     //Se pasa de char a uint8_t

if(((buffer_vert_LIDAR_USB[0]!=0x00)||(buffer_vert_LIDAR_USB[0]!=0x00))&&(buffer_vert_
LIDAR_USB[1]==0x00)&&(buffer_vert_LIDAR_USB[2]==0x00)&&(buffer_vert_LIDAR_USB[3]==0x00
)){
        txbuffer[1]=0x30;
        txbuffer[2]=0x30;
        txbuffer[3]=0x30;
        txbuffer[4]=buffer_vert_LIDAR_USB[0];
     }else if((buffer_vert_LIDAR_USB[2]==0x00)&&(buffer_vert_LIDAR_USB[3]==0x00)){
        txbuffer[1]=0x30;
        txbuffer[2]=0x30;
        txbuffer[3]=buffer_vert_LIDAR_USB[0];
        txbuffer[4]=buffer_vert_LIDAR_USB[1];
     }else if(buffer_vert_LIDAR_USB[3]==0x00){
        txbuffer[1]=0x30;
        txbuffer[2]=buffer_vert_LIDAR_USB[0];
        txbuffer[3]=buffer_vert_LIDAR_USB[1];
        txbuffer[4]=buffer_vert_LIDAR_USB[2];
     }else {
        for(m=0; m<sizeof(buffer_vert_LIDAR_USB);m++){
           txbuffer[m+1]=buffer_vert_LIDAR_USB[m];
        }
     }
     //Limpieza buffer LIDAR_USB
     for(k=0;k<sizeof(buffer_vert_LIDAR_USB);k++){
        buffer_vert_LIDAR_USB[k]=0x00;
     }
     //3º Desplazamiento horizontal Stepper

snprintf(buffer_horizontal_USB,7,"%f",buffer_move_stepper[pos_buffer_move_stepper-1]);
     for(m=0; m<sizeof(buffer_horizontal_USB);m++){
        txbuffer[m+5]=buffer_horizontal_USB[m];
     }
     //Limpieza buffer horizontal_USB
     for(a=0;a<sizeof(buffer_horizontal_USB);a++){
        buffer_horizontal_USB[a]=0x00;
     }
     //4º Vuelta horizontal (horaria (0-360 grados) o antihoraria (360-0 grados))
     if(flag_antihorario==0){
        txbuffer[11]=0x30;
     }else if(flag_antihorario==1){
        txbuffer[11]=0x31;
     }
     //5º Calidad
     snprintf(buffer_calidad_USB,5,"%d",buffer_scan[pos_buffer_scan-1]);

if(((buffer_calidad_USB[0]!=0x00)||(buffer_calidad_USB[0]!=0x00))&&(buffer_calidad_USB
[1]==0x00)&&(buffer_calidad_USB[2]==0x00)&&(buffer_calidad_USB[3]==0x00)){
        txbuffer[12]=0x30;
        txbuffer[13]=0x30;
        txbuffer[14]=0x30;
```

```
        txbuffer[15]=buffer_calidad_USB[0];
     }else if((buffer_calidad_USB[2]==0x00)&&(buffer_calidad_USB[3]==0x00)){
        txbuffer[12]=0x30;
        txbuffer[13]=0x30;
        txbuffer[14]=buffer_calidad_USB[0];
        txbuffer[15]=buffer_calidad_USB[1];
     }else if(buffer_calidad_USB[3]==0x00){
        txbuffer[12]=0x30;
        txbuffer[13]=buffer_calidad_USB[0];
        txbuffer[14]=buffer_calidad_USB[1];
        txbuffer[15]=buffer_calidad_USB[2];
     }else {
        for(m=0; m<sizeof(buffer_calidad_USB);m++){
           txbuffer[m+12]=buffer_calidad_USB[m];
        }
     }
     //Limpieza buffer horizontal_USB
     for(b=0;b<sizeof(buffer_calidad_USB);b++){
        buffer_calidad_USB[b]=0x00;
     }
     //6º Dato nulo
     snprintf(buffer_angle_USB,9,"%f",buffer_angle_dist[pos_buffer_angle_dist-2]);
     for(m=0; m<sizeof(buffer_angle_USB);m++){
        txbuffer[m+16]=buffer_angle_USB[m];
     }
     //7º Dato distancia
     snprintf(buffer_distance_USB,9,"%f",buffer_angle_dist[pos_buffer_angle_dist-1]);
     for(m=0; m<sizeof(buffer_distance_USB);m++){
        txbuffer[m+23]=buffer_distance_USB[m];
     }
     //Limpieza buffer distance_USB y angle_USB
     for(c=0;c<sizeof(buffer_calidad_USB);c++){
        buffer_distance_USB[c]=0x00;
        buffer_angle_USB[c]=0x00;
     }
     //Como fin de trama se envía un 0x23 (#)
     txbuffer[30]=0x23;
     //Envío por USB al PC
     printf("%s", txbuffer);
     //Reseteo flag_USB
     flag_USB=0;
  }
}
```

Anexo 2. Código 3. Función envíoUSB.

ANEXO 2

ANEXO 3

```
void botones () {
  //Botón ON_OFF
  MyController1.addButton("ONOFF”, 10,640,580,80,20);
  //Botón Visible
  MyController2.addButton("Visible”, 10,720,580,80,20);
}
```

Anexo 3. Código 1. Función botones.

```
void consola(){
  //Área de texto
  Textarea = MyController3.addTextarea("txt")
                  .setPosition(0,400)
                  .setSize(252,200)
                  .setFont(createFont("arial",15))
                  .setLineHeight(14)
                  .setColor(color(152, 152, 152))
                  .setColorBackground(color(255,100))
                  .setColorForeground(color(255,100));
                  ;
  Textarea.scroll(height);

  //Texto encabezado consola
  Textlabel = MyController3.addTextlabel("label")
                   .setText("Tramas recibidas")
                   .setPosition(71,385)
                   .setColorValue(Blanco)
                   .setColorBackground(color(255,100))
                   .setFont(createFont("arial",12))
                   ;
}
```

Anexo 3. Código 2. Función consola.

```
void leyendaColores(){
  leyColor = MyController4.addListBox("myList")
         .setPosition(252, 502)
         .setSize(120, 120)
         .setItemHeight(14)
         .setBarHeight(14)
         .setColorBackground(color(Blanco))
         .setFont(createFont("arial",10))
         .lock()
         ;

  leyColor.getCaptionLabel().toUpperCase(false);
```

```
  leyColor.getCaptionLabel().set("Leyenda de Alturas (cm)");
  leyColor.getCaptionLabel().setColor(Negro);

  //Generalidades para todo los items de la tabla
  //Letra de color negro y en minúscula
  leyColor.setColorValueLabel(Negro).getValueLabel().toUpperCase(false);

  //Verde Claro
  leyColor.addItem("Altura negativa", 0);
  leyColor.getItem("Altura negativa").put("color", new
CColor().setBackground(VerdeClaro));

  //Amarillo Claro
  leyColor.addItem("Hasta 300 cm", 1);
  leyColor.getItem("Hasta 300 cm").put("color", new
CColor().setBackground(AmarilloClaro));

  //Amarillo Oscuro
  leyColor.addItem("Hasta 600 cm", 2);
  leyColor.getItem("Hasta 600 cm").put("color", new
CColor().setBackground(AmarilloOscuro));

  //Naranja Claro
  leyColor.addItem("Hasta 900 cm", 3);
  leyColor.getItem("Hasta 900 cm").put("color", new
CColor().setBackground(NaranjaClaro));

  //Naranja Oscuro
  leyColor.addItem("Hasta 1200 cm", 4);
  leyColor.getItem("Hasta 1200 cm").put("color", new
CColor().setBackground(NaranjaOscuro));

  //Rojo
  leyColor.addItem("Mayor que 1200 cm", 5);
  leyColor.getItem("Mayor que 1200 cm").put("color", new
CColor().setBackground(Rojo));
}
```

Anexo 3. Código 3. Función leyendaColores.

```
int Verde=0xFF009150;
int VerdeClaro=0xFF59E970;
int AmarilloClaro=0xFFFBFE69;
int AmarilloOscuro=0xFFFDE900;
int NaranjaClaro=0xFFFFB060;
int NaranjaOscuro=0xFFFF8000;
int Rojo=0xFFE60026;
int Negro=0xFF000000;
int Blanco=0xFFFFFFFF;
```

Anexo 3. Código 4. Colores.

```
public void ONOFF(){
    if(ONOFF == 0){//Cuando se inicia el programa
       if(flagArchivo==1||flagArchivo==2){
          ONOFF = 1;
          Textarea.setText("");//Se limpia la consola
          flagDibujar=1;
       }
```

```
        if(flagArchivo==2){
          String portName = Serial.list()[1];
          myPort = new Serial(this, portName, 115200);
          myPort.write(95);//Cabecera
          delay(10);
          myPort.write(65);//START 0x41 en HEX, símbolo A
        }
    }else{
        ONOFF = 0;
        if(flagArchivo==2){//Tiempo Real
          myPort.write(95);//Cabecera
          delay(10);
          myPort.write(82);//STOP 0x52 en HEX, símbolo R
          myPort.stop();//Se cierra el puerto
        }
        cam.setState(state);//Carga estado inicial de la camara

        if(flagArchivo==2){
          output.flush();
          output.close();
        }
        points.clear();

        stext="";

        //Se resetean todas las variables cuando se apaga el programa
        trama="";
        cabecera="";
        desplVertLidar=0;
        desplHoriStepper=0;
        flagHorizontal=0;
        calidad=0;
        angulo=0;
        distancia=0;
        finTrama="";
        fintrama2="";
        flagRec=0;
        flagDibujar=0;
        x=0;
        y=0;
        z=0;
        flagPrimeraVuelta=0;
        VISIBLE_OFF=0;
        flagArchivo=0;

    }
}
```

Anexo 3. Código 5. Función ONOFF.

```
public void visible(){
    if(VISIBLE_OFF == 0){
        VISIBLE_OFF = 1;
    }else{
        VISIBLE_OFF = 0;
    }
}
```

Anexo 3. Código 6. Función visible.

```
void cuadrosTexto(){
  Textfield crearArchivo =MyController5.addTextfield("Archivo a leer")
     .setPosition(437,580)
     .setSize(100,20)
     .setFont(createFont("arial",12))
     ;
  Textfield leerArchivo =MyController5.addTextfield("Archivo a crear")
     .setPosition(539,580)
     .setSize(100,20)
     .setFont(createFont("arial",12))
     ;
  //Minúsculas y color negro
  crearArchivo.getCaptionLabel().toUpperCase(false).setColor(Negro);;
  leerArchivo.getCaptionLabel().toUpperCase(false).setColor(Negro);;
  //Centrar texto label
  crearArchivo.getCaptionLabel().align(ControlP5.CENTER,0);
  crearArchivo.getCaptionLabel().getStyle().setPaddingTop(-24);
  leerArchivo.getCaptionLabel().align(ControlP5.CENTER,0);
  leerArchivo.getCaptionLabel().getStyle().setPaddingTop(-24);

  textLabelFile = MyController6.addTextlabel("label")
     .setPosition(645,553)
     .setFont(createFont("arial",20))
     ;
}
```

Anexo 3. Código 7. Función cuadrosTexto.

```
void accionTexto(){
  if(flagArchivo==1){//Lectura Archivo
    boolean exists =file.exists();
    if(exists){
      textLabelFile.setText("El archivo existe").setColorValue(Verde);
      reader = createReader(textValue);
      MyController6.setVisible(true);
     }else{
      textLabelFile.setText("     No existe").setColorValue(Rojo);
      MyController6.setVisible(true);
      flagArchivo=3;
    }
  }
  if(flagArchivo==2){//Crear Archivo
    textLabelFile.setText("Archivo creado").setColorValue(Verde);
    MyController6.setVisible(true);
  }
}
```

Anexo 3. Código 8. Función accionTexto.

```
void colorPunto(float point){
  if(point<0){
```

```
      //VERDE CLARO
      stroke(VerdeClaro);//Colorea perímetro
      fill(VerdeClaro);//Colorea interior
   }else if(point<300){
      //AMARILLO CLARO
      stroke(AmarilloClaro);
      fill(AmarilloClaro);
   }else if(point<600){
      //AMARILLO OSCURO
      stroke(AmarilloOscuro);
      fill(AmarilloOscuro);
   }else if(point<900){
      //NARANJA CLARO
      stroke(NaranjaClaro);
      fill(NaranjaClaro);
   }else if(point<1200){
      //NARANJA OSCURO
      stroke(NaranjaOscuro);
      fill(NaranjaOscuro);
   }else if(point<1600||point>=1600){
      //ROJO
      stroke(Rojo);
      fill(Rojo);
   }
}
```

Anexo 3. Código 9. Función colorPunto.