



Universitat de les
Illes Balears



Trabajo Fin de Grado

GRAU D'ENGINYERIA ELECTRÒNICA INDUSTRIAL I
AUTOMÀTICA

Dispositivo de posicionamiento 3D de balizas Ultra-Wide-Band

JOSÉ CRESPI VALERO

Tutores

Francisco Bonnín Pascual

Alberto Ortiz Rodríguez

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, 13 de septiembre de 2019

En agradecimiento a toda la gente que me ha apoyado cuando más lo he necesitado, a todas esas personas que me hicieron el camino más sencillo hacia mis objetivos pero especialmente a mi familia, la que estuvo combatiendo conmigo cuando me enfrenté a la lucha más importante de mi vida.
Hoy he cumplido un sueño!

ÍNDICE GENERAL

Índice general	iii
Índice de figuras	v
Índice de tablas	vii
Glosario	ix
Resumen	xi
1 Introducción	1
1.1 Motivación	1
1.2 Objetivos	2
1.3 Estructura del documento	2
2 Dispositivos y herramientas hardware	5
2.1 Medidor de distancias láser	5
2.2 Unidad de Pan y Tilt (en inglés <i>Pan Tilt Unit</i>) (PTU)	6
2.3 Impresora 3D	7
2.4 Otros módulos utilizados	8
2.4.1 Módulo convertidor DC-DC Buck	8
2.4.2 Módulo conversor TTL-USB	9
3 Herramientas software	11
3.1 ROS	11
3.1.1 Introducción	11
3.1.2 Descripción y Funcionamiento	11
3.2 OnShape	12
3.2.1 Introducción	12
3.2.2 Descripción y funcionamiento	13
3.3 Cura	14
3.3.1 Introducción	14
3.3.2 Funcionamiento	15
3.4 Motive	15
3.4.1 Introducción	15
3.4.2 Descripción y funcionamiento	16
4 Implementación	19

4.1	Vista general de la solución escogida	19
4.2	Implementación física del dispositivo de calibración	22
4.2.1	Diseño e impresión 3D del compartimento	22
4.2.2	Conexión de los diferentes módulos	27
4.3	Implementación del software de calibración	28
4.3.1	Interfaz con el módulo láser	28
4.3.2	Software de control del dispositivo	30
4.3.3	Otras funciones	32
5	Experimentos	35
5.1	Metodología	35
5.2	Evaluación de la repetitividad y precisión del dispositivo 1	36
5.3	Evaluación de la repetitividad y precisión del dispositivo 2	38
5.4	Evaluación de la precisión de los servomotores	39
5.5	Simulación de un caso real de una sesión de calibración de balizas UWB	40
6	Conclusiones y trabajo futuro	43
6.1	Conclusiones	43
6.2	Trabajo futuro	44
A	Código software	45
A.1	Código del nodo implementado <i>System Controller</i>	45
A.1.1	Fichero <i>phantom_{tower}.yaml</i>	45
A.1.2	Fichero <i>threeD.launch</i>	45
A.1.3	Fichero <i>tower_{teleop}.py</i>	46
A.2	Código del nodo implementado <i>Laser Drivers</i>	58
A.2.1	Archivo encargado de crear el nodo <i>Laser Drivers</i> de extensión .cpp	58
A.2.2	Archivo .h	58
A.2.3	Archivo .cpp	59
B	Planos del compartimento implementado	65
B.1	Plano de la parte superior	65
B.2	Plano de la parte inferior	67
B.3	Plano del compartimento ensamblado	69
	Bibliografía	71

ÍNDICE DE FIGURAS

2.1	Módulo medidor de distancias láser.	6
2.2	Unidad de Pan-Tilt usada en el proyecto	7
2.3	Impresora 3D Creality 3D® Ender-3 Pro usada en el proyecto.	8
2.4	Módulo reductor de tensión LM2596 usado en el proyecto.	9
2.5	Convertidor TTL a USB PL2303 usado en el proyecto.	10
3.1	Capa de la base de una de una pieza diseñada con OnShape.	13
3.2	Planos de una pieza en OnShape	14
3.3	Diseño dividido en capas por el software Cura.	15
3.4	Entorno gráfico del sistema de captura de movimiento MOTIVE.	16
3.5	Cámara de <i>optitrack</i> utilizada en el proyecto	17
4.1	Vista general del conexionado de la solución escogida.	20
4.2	Funciones del mando de control.	20
4.3	Funciones secundarias del mando de control.	21
4.4	Vista general del software la solución escogida.	21
4.5	Sistema de coordenadas del dispositivo.	23
4.6	Diseño del compartimento realizado con OnShape.	24
4.7	Vista superior del compartimento.	25
4.8	Vista completa del compartimento implementado.	25
4.9	Vista superior del compartimento con CURA.	26
4.10	PINOUT de un cable USB	27
5.1	Vista de las cámaras apuntando a los marcadores utilizando <i>Optitrack</i>	36

ÍNDICE DE TABLAS

5.1	Posiciones de los marcadores proporcionados por <i>MOTIVE</i>	37
5.2	Resultados repetitividad y precisión por cada marcador.	37
5.3	Resultados repetitividad y precisión global	37
5.4	Resultados repetitividad y precisión por cada marcador para el 2º experimento	38
5.5	Resultados repetitividad y precisión global para el 2º experimento	39
5.6	Resultados repetitividad por cada marcador para el 3º experimento	39
5.7	Resultados repetitividad global para el 3º experimento	40
5.8	Resultados de una sesión de calibración: sin modificar la posición de la PTU.	41
5.9	Resultados de una sesión de calibración: modificando la posición de la PTU.	42

GLOSARIO

- CAD** Diseño asistido por computadora (en inglés *Computer-Aided design*)
- DC** Corriente continua (en inglés *Direct Current*)
- GPS** Sistema de Posicionamiento Global (en inglés *Global Positioning System*)
- MRU** Movimiento Rectilíneo Uniforme
- PCB** Printed Circuit Board
- PTU** Unidad de Pan y Tilt (en inglés *Pan Tilt Unit*)
- PWM** Modulación por Ancho de Pulso (en inglés *Pulse-Width Modulation*)
- ROS** Sistema Operativo Robótico (en inglés *Robot Operating System*)
- STL** siglas provenientes del inglés "Standard Triangle Language"
- TFG** Trabajo Final de Grado
- TTL** Transistor-Transistor Logic
- USB** Universal Serial Bus
- UWB** UltraWideBand

RESUMEN

Uno de los grandes problemas en la robótica móvil es la localización de los robots en el entorno desconocido. La tecnología UltraWideBand (UWB) permite estimar dicha posición, usando un conjunto de balizas que deben ser previamente fijadas en el entorno. Para ello la posición de los dispositivos de localización utilizados debe estar correctamente definida. Sin embargo, el proceso de calibración de las balizas de un sistema UWB puede acumular un error humano importante.

Para facilitar el proceso de medición de éstas así como reducir el error y el tiempo invertido en la calibración, el presente proyecto se centra en el diseño de un dispositivo electro-mecánico capaz de medir la posición 3D de uno o más puntos del entorno. Más concretamente, este dispositivo debe proporcionar la posición de las balizas UWB respecto a una referencia. El manejo del dispositivo se realiza mediante un mando tipo *gamepad*.

En este proyecto se detalla el diseño hardware y software del dispositivo implementado, precisando las funcionalidades de las que ha sido dotado con el objetivo de cumplir con los requisitos funcionales e incrementar su eficacia percibida.

El dispositivo implementado es evaluado en términos de precisión y repetitividad mediante una batería de experimentos, usando para ello un sistema de captura de movimiento. Los resultados de dichos experimentos permiten confirmar el buen rendimiento del dispositivo, limitado únicamente por los componentes electrónicos usados.

Palabras clave: Estimación de una posición 3D, calibración de un sistema UWB, Prototipo de un dispositivo.

INTRODUCCIÓN

Este capítulo abarca la *motivación* puesta en el presente proyecto, los *objetivos* que se esperan de él, así como *la estructura del documento*.

1.1 Motivación

La robótica móvil ha sido un campo de rápido crecimiento. A pesar de que inicialmente su desarrollo fuese impulsado por actividades militares. Paralelamente a estos fines, también se han ideado diferentes tareas menos bélicas como la entrega de suministros, búsqueda, rescate y evaluación de daños.

Para un vehículo autónomo, conocer la localización de éste en un entorno desconocido es una característica de gran importancia para el desarrollo de las funciones para las cuales ha sido diseñado.

Para ello se utilizan diversos métodos de localización dependiendo de diferentes factores, por ejemplo el espacio por el cual se mueve el vehículo, la misión que debe llevar a cabo, etc. Actualmente los dispositivos Sistema de Posicionamiento Global (en inglés *Global Positioning System*) (GPS) son de uso común en la localización de vehículos. Sin embargo, debido a sus limitaciones, no son apropiados en interiores o entornos urbanos. Recientemente, la tecnología Ultra-Wide Band (UWB) está siendo considerada para la localización de vehículos en interiores, obteniéndose resultados favorables [1].

Este Trabajo Final de Grado (TFG) se centra en el uso de la tecnología UWB. Este tipo de sistemas permite determinar la posición del vehículo en un entorno restringido mediante el emplazamiento en el escenario de navegación de un determinado número de balizas de posición conocida ("beacons"). Aunque puede entenderse que este proceso conlleva la percepción del entorno, la posición no se estima a partir del análisis o interpretación del entorno percibido, sino que es determinado de una forma más o menos directa en base al principio de triangulación, bien a partir de medidas de distancias, de ángulos o combinaciones de los dos. El número mínimo de balizas

requeridas dependerá del tipo de sistema empleado [2]. Concretamente, el sistema empleado en el presente proyecto requiere de cuatro o más balizas colocadas a la superficie del entorno, emisoras de un señal que recibe un dispositivo receptor llamado trazador que permanece acoplado al robot. No obstante, este método requiere de un conocimiento preciso de la posición de éstas balizas.

Hoy en día conocer la posición de éstas se realiza manualmente y ésta supone una tarea tediosa, repetitiva, pudiendo ser larga dependiendo del número de balizas utilizadas y además se pueden producir errores con facilidad. Para ello, una de las posibles soluciones es el diseño de un dispositivo capaz de medir las coordenadas 3D de las balizas. Esta solución reduciría considerablemente el tiempo del proceso de medición de éstas, el tiempo que proporciona la comodidad de no tener que recorrer toda la zona para ir a la posición de cada baliza a realizar su medición manualmente y evitaría hacer los cálculos pertinentes para obtener las distancias respecto a un punto escogido como referencia.

1.2 Objetivos

Este proyecto tiene como objetivos principales el diseño hardware y software, la implementación y la evaluación de un dispositivo capaz de medir posiciones 3D respecto a una referencia escogida previamente. Con ello se pretende que el proceso de calibración sea rápido, cómodo, y se disminuya la posibilidad de introducir errores humanos. Además, el dispositivo diseñado debe de ser fácil de montar y desmontar en la zona de trabajo.

Para cumplir los objetivos de este Trabajo Final de Grado (TFG) se utilizarán: un módulo medidor de distancias láser, una Unidad de Pan y Tilt (en inglés *Pan Tilt Unit*) (PTU) que se encargará de realizar el movimiento angular de éste, diferentes componentes electrónicos que permitan establecer una correcta conexión entre ambos y, por último, un mando que facilitará el manejo del sistema.

Para evaluar el dispositivo diseñado se realizarán una serie de experimentos usando un sistema de captura de movimiento para determinar *la precisión y repetitividad* del sistema.

1.3 Estructura del documento

Con la finalidad de cumplir los objetivos explicados en el apartado 1.2, a partir de aquí, esta memoria explicará los siguientes capítulos.

- En el capítulo 2 se explican los dispositivos hardware y herramientas utilizadas en el proyecto. Destacamos el módulo medidor de distancias, del cual se explican las funciones que nos puede aportar y la PTU, la cual nos permite realizar el movimiento angular del dispositivo.
- En el capítulo 3 se explican las diferentes herramientas software utilizadas en el desarrollo del TFG. Destacando el Sistema Operativo Robótico (en inglés *Robot Operating System*) (ROS), el cual realiza los servicios estándar de un sistema operativo permitiendo entre otras cosas el control de dispositivos hardware y el

programa *OnShape*, el cual permite realizar diseños en 3D de una forma rápida y completa.

- En el capítulo 4 se presenta la implementación realizada en el presente proyecto, la cual se ha dividido en dos grandes bloques: la implementación física del dispositivo de calibración y la implementación del software de calibración.
- Posteriormente, en el capítulo 5 se exponen los experimentos realizados para evaluar la precisión y repetitividad del dispositivo, así como los resultados que se han obtenido de éstos.
- Por último, en el capítulo 6, se exponen las conclusiones extraídas del presente proyecto así como el planteamiento de posibles trabajos futuros que mejoren los resultados de éste.

DISPOSITIVOS Y HERRAMIENTAS HARDWARE

Este capítulo habla de los diversos dispositivos y herramientas hardware que se han utilizado en el proyecto.

2.1 Medidor de distancias láser

Un medidor de distancias láser, también llamado distanciómetro, es un dispositivo electrónico de medición que calcula la distancia desde el instrumento hasta la proyección del haz láser. A diferencia de otros sensores utilizados con este fin como un sensor ultrasonido, este es normalmente utilizado para medir distancias mayores.

Para la realización de este TFG se ha utilizado el módulo medidor láser *LDB6-X6N3* de la marca *KKmoon* como el que se puede observar en la figura 2.1, el cual puede medir hasta una distancia de 50m con una precisión de $\pm 2\text{mm}$. El módulo requiere de una tensión de alimentación regulada comprendida dentro del rango de tensiones de 2,9V-3,2V. El módulo está compuesto por un diodo láser y una celda receptora de éste, además de un sensor de temperatura interno y un control de la tensión de entrada. Para calcular la distancia, el módulo emite un haz de luz láser y calcula el tiempo que tarda en recibirlo la celda receptora, una vez calculado el tiempo, conocida la velocidad de la luz en un medio de propagación como es el aire, y utilizando la fórmula de Movimiento Rectilíneo Uniforme (MRU), se consigue calcular la distancia. Este dispositivo utiliza una tecnología de comunicación Transistor-Transistor Logic (TTL).

Una vez establecida la comunicación, el módulo láser se queda a la espera de recibir una de las diferentes consignas asignadas por el fabricante las cuales ordenan al dispositivo realizar una determinada función. Las funciones de las que dispone el módulo láser *LDB6-X6N3* son las siguientes:

- Encender el láser,
- Apagar el láser,
- Medir la distancia en modo lento,

- Medir la distancia en modo rápido,
- y Conocer la tensión de entrada y la temperatura del dispositivo.



Figura 2.1: Módulo medidor de distancias láser. Imagen tomada de: <https://i.ebayimg.com/images/g/ggAAASwQPtauILm/s-l1600.jpg>

Se ha aportado como material complementario a la memoria del presente proyecto el *datasheet* del dispositivo explicado en esta sección.

2.2 Unidad de Pan y Tilt (en inglés *Pan Tilt Unit*) (PTU)

La PTU es un dispositivo electro-mecánico que permite realizar una rotación entorno a dos ejes de rotación perpendiculares normalmente utilizado en las cámaras fotográficas para controlar la orientación de éstas.

La PTU está equipada con dos servomotores conectados mecánicamente uno en cada eje con la finalidad de realizar el movimiento sobre dos planos, uno horizontal y otro vertical (en inglés *pan* y *tilt* respectivamente). Éstos dispositivos electrónicos son muy utilizados hoy en día en la robótica ya que permiten controlar la orientación del elemento terminal y por lo tanto cualquier dispositivo fijado en éste.

Concretamente para este TFG se ha utilizado el modelo *PhantomX Turret* tal y como puede visualizarse en la figura 2.2. Este modelo consta de dos servomotores *Dynamixel AX-18A* los cuales tienen un paso de $0,29^\circ$ y disponen de una rotación máxima de 300° . La comunicación con los servomotores se realiza mediante Modulación por Ancho de Pulso (en inglés *Pulse-Width Modulation*) (PWM). Con la acción combinada de los dos servomotores consigue un volumen de trabajo muy cercano al esférico.

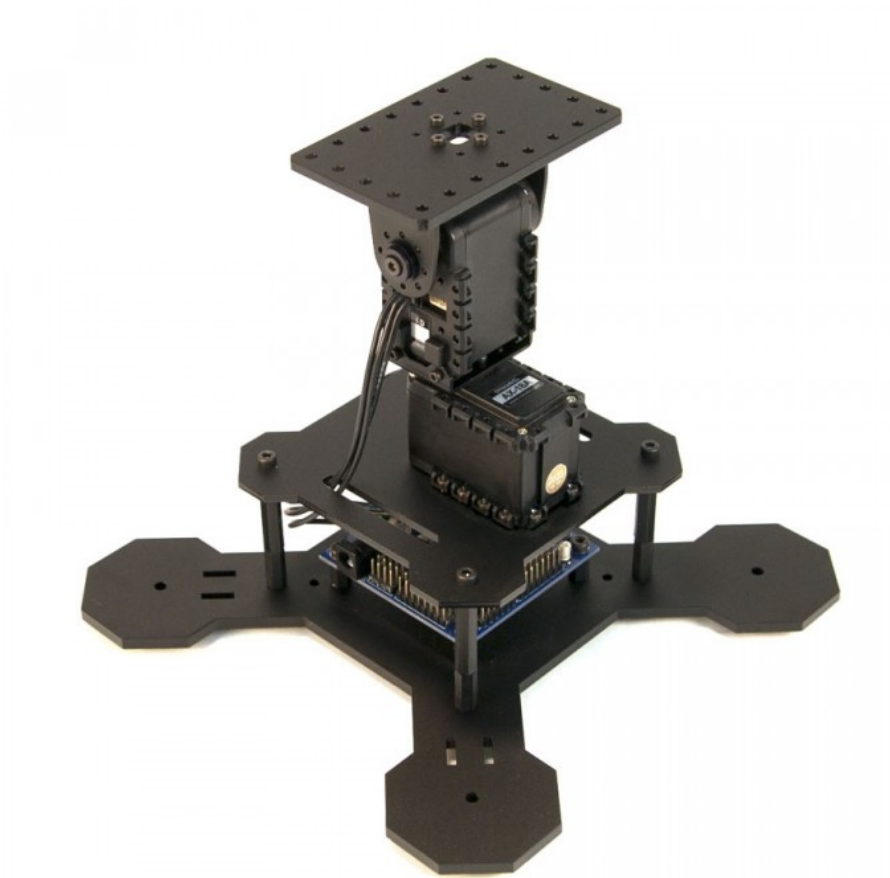


Figura 2.2: Unidad de Pan-Tilt usada en el proyecto. Imagen tomada de: https://www.roscomponents.com/466-thickbox_default/phantomx-robot-turret-kit.jpg

En el enlace añadido en el pie de pagina ¹ puede observarse el *datasheet* de los servomotores utilizados en el dispositivo explicado en esta sección.

2.3 Impresora 3D

Una impresora 3D es una maquina que tiene la capacidad de imprimir diseños en 3D. Comúnmente se utiliza en el prefabricado de piezas y componentes. Existen de varios tipos según el método utilizado: de tinta, de láser o de inyección de polímeros.

La impresora utilizada en el presente proyecto es la *Creality 3D® Ender-3 Pro* la cual tiene un espacio de trabajo de $220 \times 220 \times 250$ m y imprime con la técnica de inyección

¹http://support.robotis.com/en/product/actuator/dynamixel/ax_series/ax-18f.htm

de polímeros. Ésta se ha utilizado para imprimir la caja que acopla los diferentes dispositivos electrónicos necesarios de este TFG con la PTU utilizando el polímero llamado PLA de 1.75 mm de grosor. En la figura 2.3 puede observarse el modelo de impresora utilizado en este TFG



Figura 2.3: Impresora 3D Creality 3D® Ender-3 Pro usada en el proyecto. Imagen tomada de: <https://images-na.ssl-images-amazon.com/images/I/61pgpT7JHfL.5X425.jpg>

2.4 Otros módulos utilizados

2.4.1 Módulo convertidor DC-DC Buck

El reductor de tensión es un dispositivo electrónico convertidor de potencia que obtiene en su salida la reducción de la tensión de entrada de Corriente continua (en inglés *Direct Current*) (DC).

El módulo utilizado en el proyecto es un conversor BUCK LM2596 como el que puede observarse en la figura 2.4, el cual reduce la tensión de entrada aportando así la tensión apropiada para alimentar el módulo láser.

En el presente proyecto la tensión subministrada por la PTU es de 14V la cual el conversor reduce y regula a 3,1 V, proporcionando hasta un máximo de 3 A, siendo estos valores adecuados para alimentar el distanciómetro.

En el enlace añadido en el pie de pagina² puede observarse el *datasheet* del dispositivo explicado en esta sección.

²<https://www.alldatasheet.com/datasheet-pdf/pdf/134372/ETC1/LM2596.html?>



Figura 2.4: Módulo reductor de tensión LM2596 usado en el proyecto. Imagen tomada de: <http://www.geekbotelectronics.com/wp-content/uploads/2015/01/Lm2596.jpg>

2.4.2 Módulo conversor TTL-USB

Es bien sabido que hoy en día hay muchas tecnologías de comunicación diferentes entre dispositivos, a consecuencia de ello existen varios dispositivos que permiten la conversión de una tecnología a otra.

Para la realización del presente proyecto se ha requerido de un módulo conversor PL2303 como puede visualizarse en la 2.5, el cual es un dispositivo electrónico que realiza una conversión de la tecnología TTL, utilizada por el módulo metro láser, a Universal Serial Bus (USB), utilizada por el puerto serie del ordenador, para establecer una correcta comunicación.

En el enlace añadido en el pie de pagina³ puede observarse el *datasheet* del dispositivo explicado en esta sección.

³<https://www.alldatasheet.com/datasheet-pdf/pdf/116844/ETC1/PL2303.html>

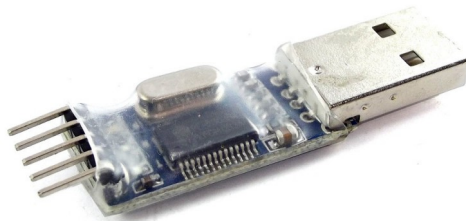


Figura 2.5: Convertidor TTL a USB PL2303 usado en el proyecto. Imagen tomada de: https://naylampmechatronics.com/200-large_default/modulo-pl2303-convertidor-usb-a-ttl-.jpg

HERRAMIENTAS SOFTWARE

En este capítulo habla de las herramientas software utilizadas en el presente proyecto.

3.1 ROS

3.1.1 Introducción

Robot Operating System (ROS) es una colección de frameworks para el desarrollo de software de robots. ROS se desarrolló en 2007 bajo el nombre de *switchyard* por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte al proyecto del Robot con Inteligencia Artificial de Stanford (STAIR). [3]

ROS dispone de una colección de herramientas, librerías y convenciones con el objetivo de simplificar tareas de creación de comportamientos complejos y robustos de robots a través de una amplia variedad de plataformas robóticas. Además provee los servicios estándar de un sistema operativo entre las cuales se incluye abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes.

ROS se divide en dos secciones: la primera es la encargada del sistema operativo y la segunda, llamada *ros-pkg*, es una suite de paquetes aportados por la comunidad de usuarios que proporcionan funcionalidades implementadas entre las cuales se destacan la localización, planificación y percepción de los robots.

Es un software de licencia libre bajo los términos de la licencia BSD, sin embargo las contribuciones de los paquetes en *ros-pkg* están bajo una gran variedad de licencias diferentes.

3.1.2 Descripción y Funcionamiento

A continuación, se explicarán los conceptos más importantes del software ROS que permiten un correcto funcionamiento.

Nodos

Los nodos son los ejecutables encargados de realizar tareas concretas. Éstos están combinados con otros nodos formando un grafo y comunicándose entre ellos mediante *tópicos*, *servicios* y *servicios de parámetros* (<http://wiki.ros.org/Nodes>)

Tipos de mensajes

Un mensaje es una estructura de datos que comprende campos definidos. Éstos contienen la información generada y intercambiada por los nodos.

Tópicos

Los tópicos son los canales por los cuales los nodos se intercambian mensajes. Éstos se comunican anónimamente mediante semántica publicador/subscriptor. En general los nodos no se comunican entre ellos, se comunican publicando a un tópico o suscribiéndose a él, pudiendo tener más de un subscriptor y publicador a estos.

Publicación y suscripción

La publicación/suscripción es un paradigma de comunicación muy flexible, donde la publicación es la inserción la información relevante a un tópico mientras que la suscripción es la extracción de ésta de los tópicos

Servicios

El tipo de comunicación mediante publicación/suscripción no es apropiada cuando se quiere comunicar del tipo solicitud/respuesta, las cuales a menudo son requeridas para en un sistema distribuido. En ese caso se utilizan los servicios, los cuales se definen por un mensaje de solicitud y otro de respuesta.

Además ROS ofrece el valor de un conjunto de herramientas que permite visualizar la comunicación establecida, las medidas proporcionadas por los sensores, modificar los parámetros de los nodos, etc.

Por último, los paquetes de ROS se pueden programar utilizando dos lenguajes de programación diferentes: *c++* y *Python*.

3.2 OnShape

3.2.1 Introducción

Onshape es un programa de Diseño asistido por computadora (en inglés Computer-Aided design) (CAD) en 3D en la nube, con herramientas de modelado en sólidos modernas, utilerías de colaboración y un licenciamiento muy simple. [4]

Esta aplicación fue desarrollada en 2012, por el equipo fundador de SolidWorks. Funciona en un navegador web (Chrome, Safari, Firefox) y en cualquier computadora (Windows, Mac, Linux, Chromebook), no es necesario descargar o instalar software o licencias.

OnShape hace un uso extensivo de servicios de computación a través de una red, que normalmente es Internet, estos servicios se denominan *la computación en la nube* (del inglés *cloud computing*). Este software permite el trabajo en equipo de un mismo diseño debido a los servicios proporcionados por la nube. Además, solo necesita que el ordenador de trabajo esté conectado a la red, pudiendo así prescindir de la instalación de programas ni manejo de licencias. Principalmente está enfocado en productos de maquinaria como componentes electrónicos, dispositivos médicos, impresiones 3D y componentes de máquinas, etc..

Finalmente, cuenta con una función para crear planos a partir de las piezas y ensamblajes creados previamente, pudiendo mostrar las medidas de una forma simple

3.3.2 Funcionamiento

Los modelos renderizados se exportan como un archivo de extensión (.stl) para el proceso de corte. La división convierte un archivo STL en capas, tal y como puede observarse la figura 3.3, de igual grosor en la dirección z en código G, un formato de archivo legible por humanos que especifica la ruta que debe seguir el cabezal de impresión para producir un objeto físico. Este código se exporta mediante un dispositivo de almacenamiento a la impresora 3D, vinculando de esta manera el software con la impresora para que ésta pueda proceder a la impresión del objeto físico [?].

Cura proporciona una serie de funciones para que la impresión se adapte a las necesidades del proyecto. Entre ellas, podemos destacar las opciones de modificación de objetos (movimiento de los objetos, escalarlos o rotar-los), las formas de colocación de las piezas (asimétrica, cara de arriba, abajo, derecha o izquierda), el material que se utilizará en las impresiones así como diferentes opciones de laminación (velocidad de impresión, temperatura de ésta, tipo de adhesión a la cama caliente etc...)

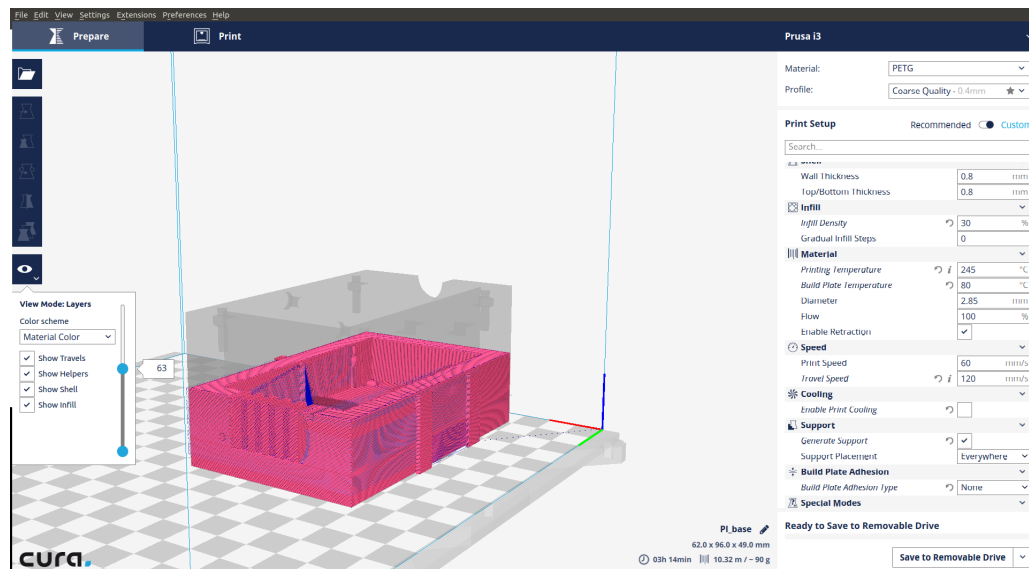


Figura 3.3: Diseño dividido en capas por el software Cura.

3.4 Motive

3.4.1 Introducción

Motive es una plataforma de software diseñada la empresa Natural Point, para controlar sistemas de captura de movimiento para diversas aplicaciones de seguimiento que van desde animación, realidad virtual, ciencias del movimiento y robótica entre otras. Motive no solo permite al usuario calibrar y configurar el sistema, sino que también proporciona interfaces para capturar y procesar datos 3D. Los datos capturados pueden grabarse o transmitirse en vivo a otras canalizaciones. También cuenta con un entorno gráfico donde se puede ver la posición de los marcadores a tiempo real de una forma muy visual tal y como puede observarse en la figura 3.4.

3. HERRAMIENTAS SOFTWARE

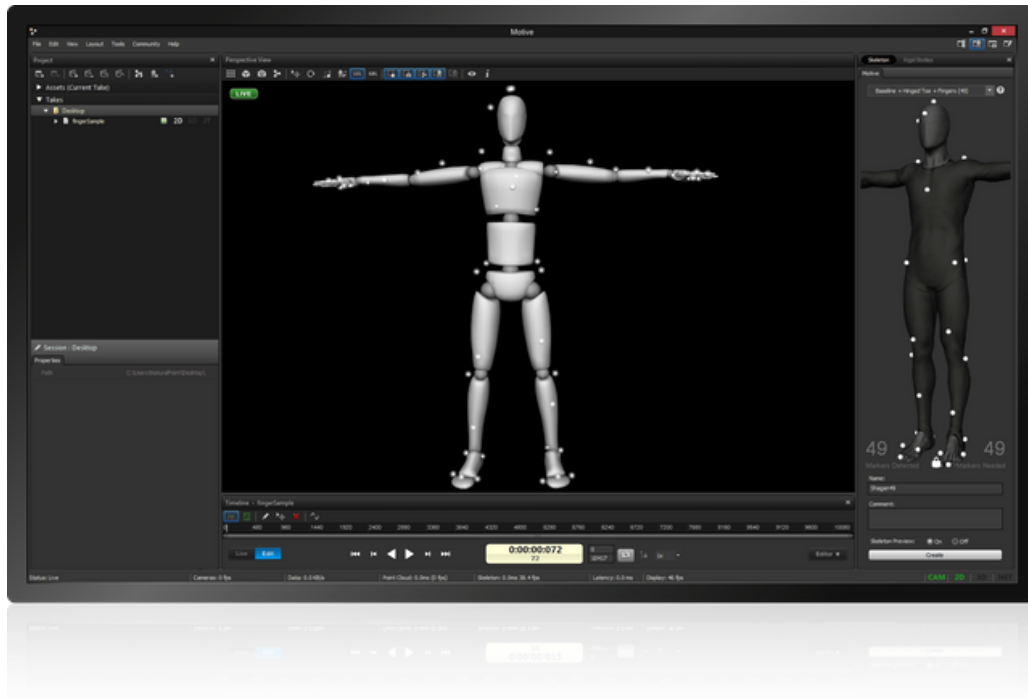


Figura 3.4: Entorno gráfico del sistema de captura de movimiento MOTIVE. Imagen tomada de: <https://steemitimages.com/640x0/https://s3-us-west-2.amazonaws.com/huntimages/production/steemhunt/2018-11-07/1896ed2b-motiveBodyScreenSubjectSkeleton@2x.png>

Cuando se habla de motive es importante mencionar el sistema de captura de movimiento *Optitrack*, el cual está compuesto por un grupo de cámaras que detectan los marcadores que están dentro del campo de visión de éstas. Este tipo de cámaras como se muestra en la figura 3.5 permite tomar capturas con una altísima precisión, pudiendo obtener un error de hasta un milímetro además de contar con una alta velocidad de captura pudiendo llegar a los 240 FPS, lo que permite obtener datos con una alta fiabilidad.

3.4.2 Descripción y funcionamiento

Motive obtiene información en 3D a través de *Reconstrucción*, que es el proceso de compilar múltiples imágenes 2D de marcadores para obtener coordenadas 3D. Utilizando coordenadas 3D de marcadores rastreados aplicando un método de triangulación, Motive puede obtener datos de 6 grados de libertad (posición y orientación 3D) para múltiples cuerpos rígidos y esqueletos, y permite el seguimiento de movimientos complejos en el espacio 3D.



Figura 3.5: Cámara de *Optitrack* utilizada en el proyecto. Imagen tomada de: <https://optitrack.com/public/images/prime-13-perspective-1200w.png>

CAPÍTULO 4

IMPLEMENTACIÓN

En este capítulo se presenta la implementación que se ha realizado para cumplir con los objetivos explicados en la sección 1.2. Para ello, este capítulo se divide en dos bloques: la implementación física del dispositivo de calibración y la implementación del software de éste.

4.1 Vista general de la solución escogida

Con la finalidad de comprender con mayor facilidad la implementación realizada en el presente proyecto, en esta sección se presenta una vista general tanto de la parte hardware como de la parte software de la solución implementada.

En la figura 4.1 se puede observar la vista general del conexionado de los diferentes dispositivos utilizados en el presente proyecto, con la finalidad de aportar una visión más amplia de la implementación y facilitar el entendimiento de esta sección. Cabe destacar que en dicha figura aparecen todos los módulos explicados en el capítulo 2 y se explicará con mayor profundidad el tipo de conexionado de éstos, así como su unión mecánica mediante un comportamiento específicamente diseñado en las secciones 4.2 y 4.3.

Además, como puede observarse en la figura 4.1, el PC que controla todo el sistema está controlado por un mando tipo gamepad del cual se hablará con mayor profundidad en la sección 4.3.2. Como puede observarse en la figura 4.2 y 4.3 éste tiene diferentes funciones asignadas a los botones con la finalidad de cumplir con el objetivo del presente proyecto.

En la figura 4.4 se puede observar un diagrama del software de la solución adoptada, donde cabe destacar que los conceptos introducidos en el interior de los óvalos son los *nodos* creados en el software ROS, los del interior de los rectángulos son los *tópicos* utilizados para la comunicación de los diferentes nodos y por último, el concepto introducido en el dibujo de salida hace referencia a la exportación de los resultados de la calibración en un fichero.

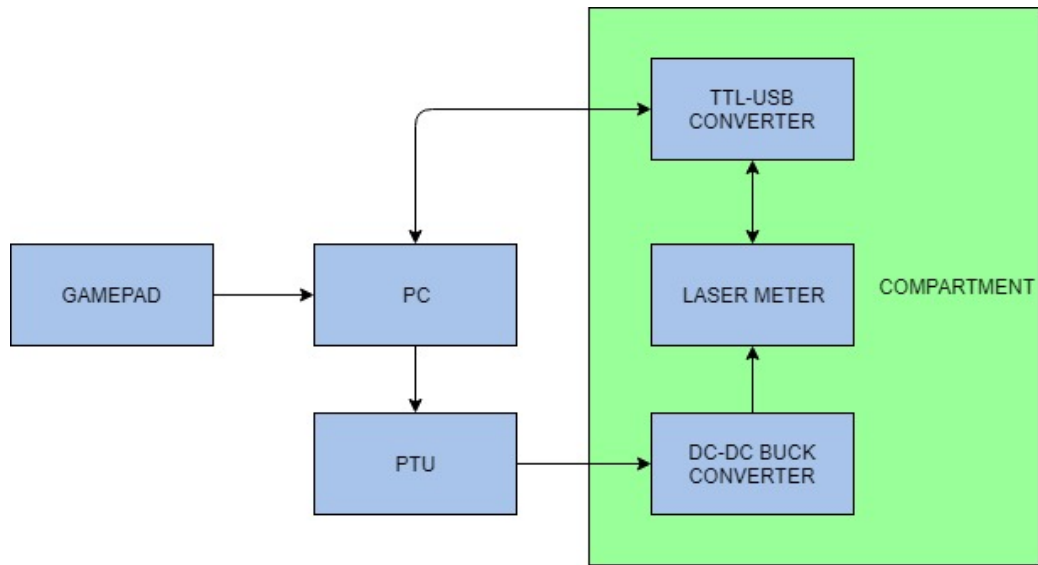


Figura 4.1: Vista general del conexionado de la solución escogida.



Figura 4.2: Funciones del mando de control.

El nodo *System Controller* es el nodo principal del programa implementado. Es el encargado de controlar a los diferentes dispositivos hardware comunicándose por los diferentes tópicos y también es el encargado de exportar un fichero de salida de texto mostrando todos los resultados de la sesión. Además, se encarga de realizar los cálculos de la posición del objeto medido en 3D a partir de los datos aportados por los nodos *Arbotix* y *Laser Drivers*.

El nodo *Laser Driver* se ha implementado con la función de establecer la comunicación con el módulo medidor de distancias. Éste se comunica mediante el *módulo conversor TTL-USB* al PC, del cual recibe una serie de consignas asignadas por el fabricante y cuenta con diferentes funciones que se explicaran en la sección 4.3.1. Para acceder a éstas, el nodo realiza una operación de escritura de sus respectivas consignas por el puerto serie del PC y se mantiene a la espera hasta recibir la señal de confirma-



Figura 4.3: Funciones secundarias del mando de control.

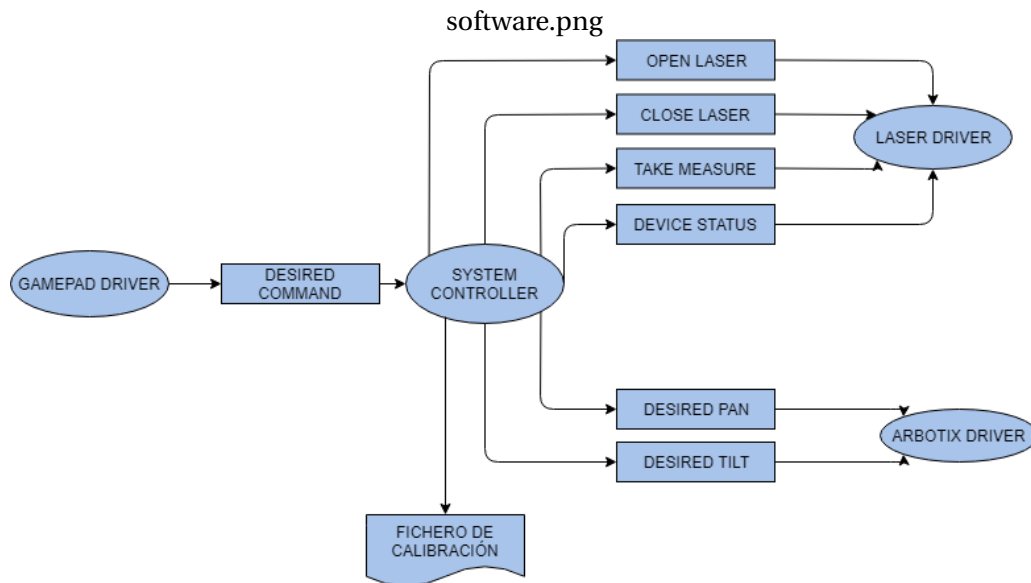


Figura 4.4: Vista general del software la solución escogida.

ción que realiza el módulo medidor de distancias realizando una operación de lectura del mismo puerto.

El nodo *Gamepad Driver* es un nodo predeterminado de ROS, llamado *Joy*, encargado de realizar la función de *driver* del mando tipo gamepad, el cual enviará una señal al nodo *System Controller* cada vez que se pulsen los botones.

El nodo *Arbotix Driver* viene por defecto con la compra del producto de la PTU y es el encargado de controlar la posición de ésta aportando una serie de funciones ya implementadas por defecto como el movimiento de los servomotores a una orientación deseada, o conocer el estado de los servomotores.

A continuación se explica como debe realizarse una sesión de calibración correctamente usando el dispositivo implementado.

4. IMPLEMENTACIÓN

1. Primero de todo, se debe situar el dispositivo en cualquier lugar, donde las balizas a medir estén situadas dentro del rango de trabajo del dispositivo, sin olvidarse de las limitaciones de la PTU, explicadas en la sección 2.2.
2. A continuación se debe orientar el dispositivo teniendo en cuenta que el sistema de coordenadas está asignado a la orientación de éste, como se puede observar en la figura 4.5.
3. Posteriormente, se debe escoger y guardar la posición de referencia para localizar las futuras mediciones de las balizas. Esta posición será el origen de coordenadas de todas las mediciones posteriores.
4. Una vez asignada ésta, se debe proceder a la medición de la posición de las balizas recomendable en sentido antihorario.
5. Finalmente, se exportará el fichero con los resultados de la sesión realizada, incluyendo las coordenadas 3 de todas las balizas UWB, referenciadas respecto al origen de coordenadas fijado.

4.2 Implementación física del dispositivo de calibración

En esta sección se explican las tareas necesarias para diseñar e imprimir el dispositivo hardware.

4.2.1 Diseño e impresión 3D del compartimento

Primero de todo, se desea que el dispositivo de posicionamiento a diseñar tenga la capacidad de desacople de los diferentes componentes hardware con facilidad y rapidez. Para ello se ha diseñado e impreso un compartimento capaz de conectar los diferentes módulos hardware utilizados en el proyecto con la pletina de la PTU, formando así un sólido rígido unido mecánicamente por 4 tornillos.

El compartimento se ha diseñado con forma de prisma recto tal y como puede observarse en la figura 4.6 debido a la forma de los módulos de su interior así como a la intención de reducir su volumen el máximo posible. El diseño de éste permite formar un sólido rígido con el elemento terminal de la PTU y el módulo láser. Gracias a ello, el dispositivo permite al usuario controlar la orientación del láser, pudiendo así tomar muestras con diferente orientación.

Con la finalidad de facilitar el montaje y desmontaje de los módulos utilizados para su uso en otras tareas, el diseño del compartimento se ha dividido en dos partes. La parte inferior es la que almacena los diferentes dispositivos hardware mientras que la superior cumple con la función de tapa del compartimento.

A continuación se explica detalladamente la realización del diseño de las dos partes del compartimento las cuales se han desarrollado utilizando el software OnShape, explicado en la sección 3.2, así como la impresión 3D de estos.

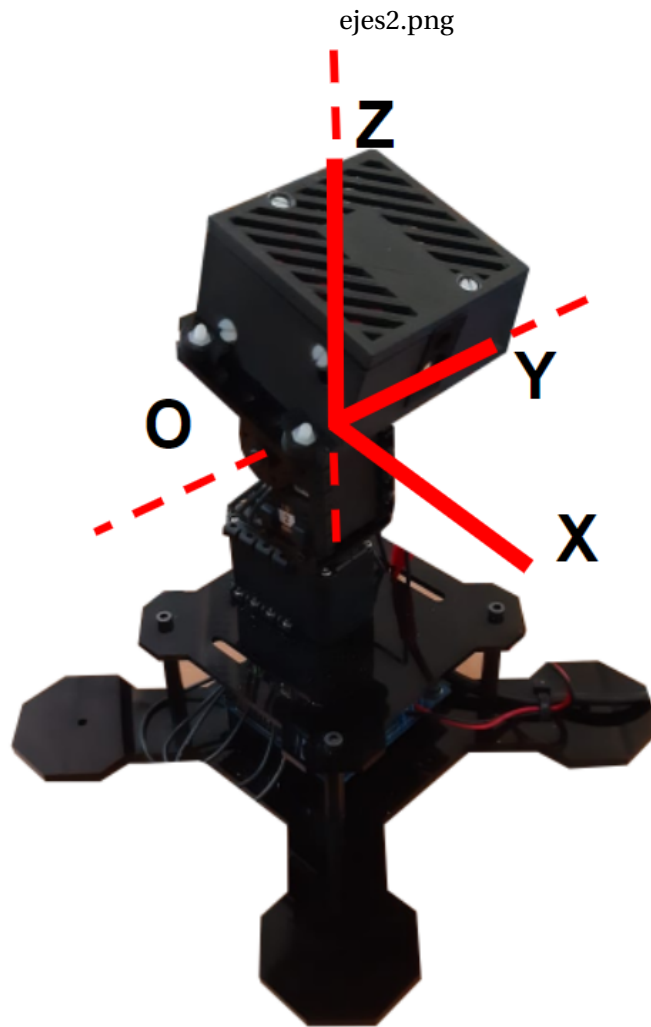


Figura 4.5: Sistema de coordenadas del dispositivo.

Boceto de la parte inferior del compartimento.

Previamente a la realización del diseño del compartimento, se han realizado diferentes bocetos de módulos utilizados dentro de éste en el software *OnShape* con la finalidad de obtener una visión más ajustada del espacio requerido.

El diseño de la parte inferior del compartimento se ha realizado de tal manera que disponga del espacio suficiente para poder introducir los diferentes módulos. Éstos se han colocado para que en su conjunto ocupasen el mínimo espacio posible teniendo como referencia las medidas de la pletina de la PTU. También se ha reducido el máximo la altura para disminuir el momento angular provocado por la altura del centro de masas de éste respecto al eje de rotación del servomotor de la PTU con rotación *Tilt*. De esta manera, se consigue la capacidad de orientarlo libremente sin que éste supere el par máximo de éste proporcionados en el apéndice tal y como se ha explicado en la sección 2.2.

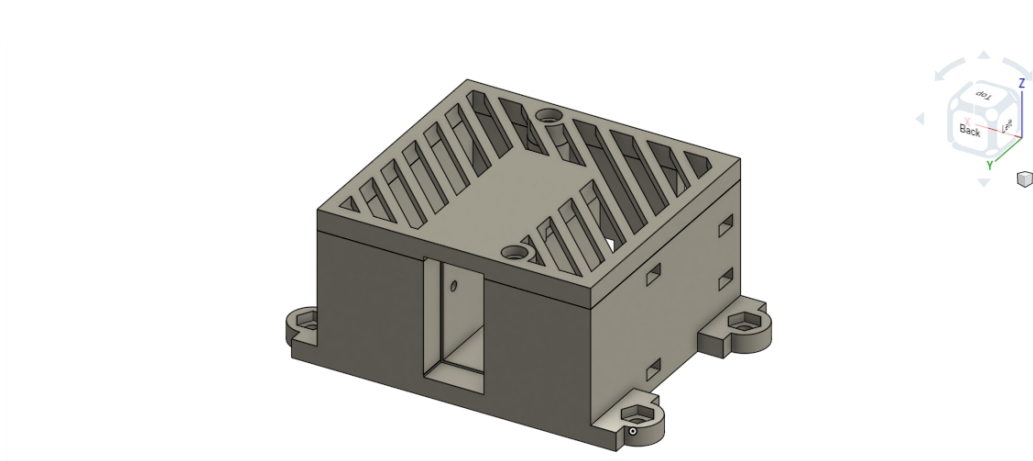


Figura 4.6: Diseño del compartimento realizado con OnShape.

También se han diseñado en la parte trasera del compartimento diferentes orificios para dejar pasar todo el cableado de una forma ordenada. Concretamente en el caso del *cable USB*, debido a la rigidez de éste se ha optado por alargar el orificio hasta la parte superior del compartimento, siendo la tapa la encargada de cerrar el orificio. De esta manera conseguimos simplificar el montaje del cableado.

Además se ha reducido todo lo posible el tamaño del compartimento con el objetivo de reducir su peso. El grosor de las paredes, se ha ajustado a 3 cm. Como se ha mencionado anteriormente, se ha utilizado la pletina de la PTU como referencia de manera que la parte inferior del compartimento mide finalmente $59 \times 56 \times 29$.

Para reducir el máximo el espacio, se decidió colocar el módulo *conversor TTL a USB* y el *conversor BUCK* en posición vertical debido a la anchura que ofrecía la pletina y a los laterales del compartimento para facilitar su conexión mecánica a éste además de dejar el espacio central al módulo medidor de distancias, el cual se ha posicionado perfectamente en el centro del compartimento, tal y como puede observarse en la figura 4.7. Posicionando éste en el centro se consigue reducir considerablemente el error producido en el calculo de la transformada realizada para la correcta obtención de las medidas. Esta transformación se explicará con detalle en el en la sección 4.3.1.

También se han tenido en cuenta otros factores en el diseño como la fijación de los diferentes componentes según las opciones que éstos ofrecían. En el caso del *módulo convertidor BUCK*, disponía de dos agujeros que se han utilizado para conectar mecánicamente a la pared del compartimento mediante el uso de unos tornillos. A diferencia del anterior, el diseño del *módulo conversor de TTL a USB* no disponía de ninguna opción para su conexión mecánica con el compartimento y se optó por utilizar dos bridas que fijan el módulo impidiendo su movimiento a partir de unos límites. Por último, el módulo medidor de distancias tampoco ofrecía un diseño fácil para su conexión mecánica con el compartimento. Gracias a la alta precisión de la que se disponía para realizar la impresión, se decidió incorporar unas paredes en el compartimento que impiden el movimiento del módulo láser.

Finalmente en la figura 4.8 se puede observar el resultado del compartimento implementado.

En el apéndice B.1 puede observarse el plano del diseño realizado para la parte

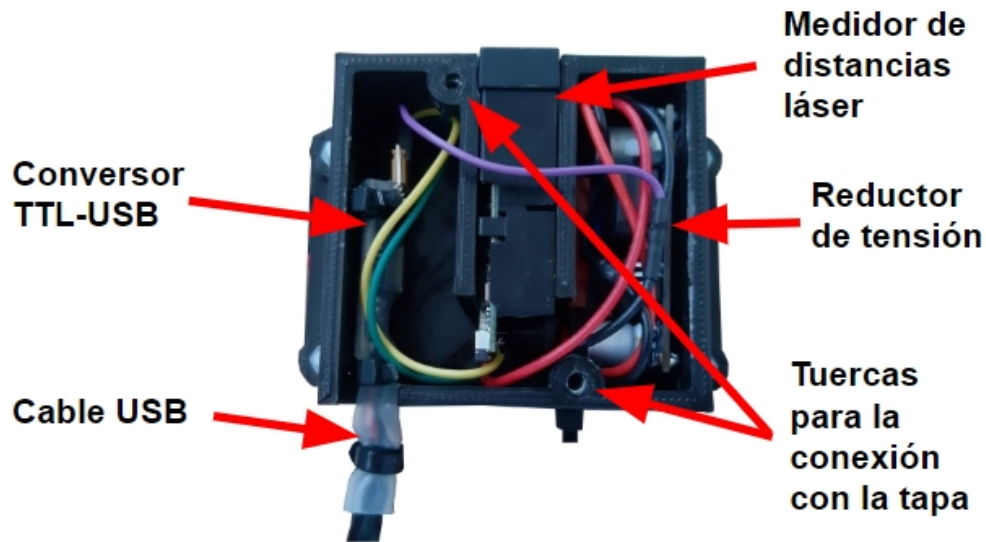


Figura 4.7: Vista superior del compartimento.

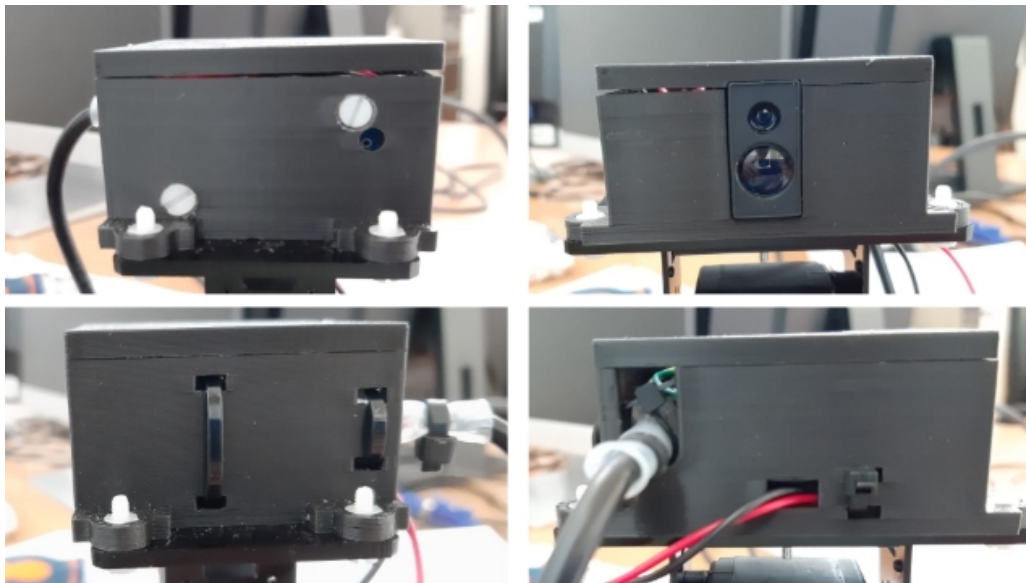


Figura 4.8: Vista completa del compartimento implementado.

inferior del compartimento.

Boceto de la parte superior del compartimento.

En el caso de la tapa se ha realizado el diseño teniendo en cuenta la protección superior de los diferentes módulos utilizados en el proyecto así como la ventilación, el peso y la sujeción superior del módulo medidor de distancias impidiendo su movimiento.

Concretamente, el diseño de la tapa del compartimento se ha diseñado en forma

4. IMPLEMENTACIÓN

de rejilla de ventilación aportando mayor densidad en la zona central con la finalidad de soportar la fuerza ejercida por el módulo medidor de distancias y evitar así el sobrecalentamiento de los diferentes dispositivos utilizados en este proyecto. Además, cuenta con unos conductos para dejar pasar los cables que conectan el *módulo BUCK* con el *módulo conversor de TTL a USB*.

El diseño de la tapa combinado con el de la parte inferior, permite la conexión mecánica entre éstos mediante dos tornillos, lo que supone una gran ventaja en el tiempo de montaje de los diversos dispositivos introducidos en el interior del compartimento.

En el apéndice B.2 puede observarse el plano del diseño realizado para la parte superior del compartimento.

Impresión del diseño.

Una vez finalizado el diseño de las dos partes del compartimento se han exportado los dos bocetos con extensión .STL, la cual permite su interpretación por el software Cura, explicado en la sección 3.3. Se ha seleccionado una densidad de las paredes del 20%, como se puede observar en la figura 4.9, suficiente para aguantar el peso de los diferentes dispositivos sin deformarse y sin ejercer una fuerza superior a la máxima por los servomotores, explicado en el apartado 4.2.1. Del siguiente modo, se ha podido ajustar el peso del compartimento a 58 gramos. Además, también se ha realizado la impresión utilizando el polímero PLA de 1.75 mm lo que aporta a la estructura la rigidez y dureza necesarias para no sufrir alteraciones en la repetitividad debido a la fuerza centrípeta ejercida por el movimiento de rotación de la PTU.

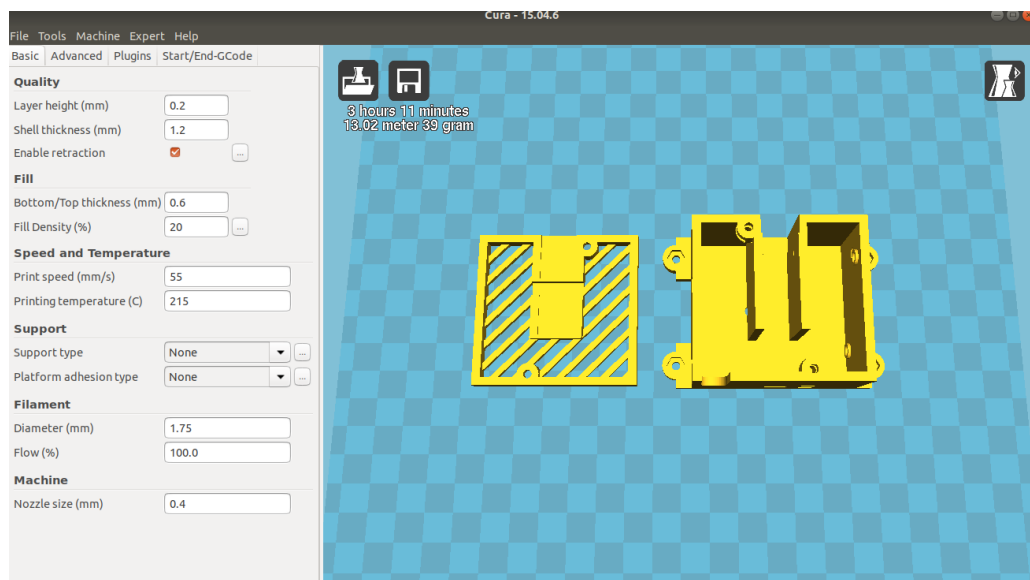


Figura 4.9: Vista superior del compartimento con CURA.

Por último, se ha realizado un acabado sobre el diseño antes de imprimirlo que mejora su visibilidad como producto a nivel estético, añadiendo chaflán en la posición de los tornillos para que estos no sobresalgan, añadiendo orificios de un tamaño ajustado para permitir el cableado que sobresale del compartimento, tanto de alimentación

como de comunicación, además de permitir el montaje de los diferentes dispositivos con facilidad.

En el apéndice B.3 puede observarse el plano del diseño realizado del compartimento que se ha enviado a imprimir.

4.2.2 Conexionado de los diferentes módulos

Primero de todo, se explica el proceso de conexionado de los diferentes dispositivos, explicados en el capítulo 2, dentro del compartimento diseñado.

Primero de todo, el dispositivo de posicionamiento diseñado se controla utilizando un mando de tipo *gamepad*. Se comunica mediante *bluetooth* al ordenador encargado de lanzar el programa que permite controlar la PTU. Ésta es la encargada de alimentar el *módulo láser* pero debido a que el rango de tensión de entrada es muy inferior a la subministrada, se ha decidido utilizar un *módulo reductor*. Posteriormente a la reducción del voltaje, el distanciómetro requiere de un *convertor de tecnología TTL-USB* para establecer la comunicación con el ordenador. El convertor va conectado mediante un cable USB que se soldó previamente como se explica más adelante en la presente sección.

Como puede observarse en la figura 4.1 los módulos utilizados en el presente proyecto se han colocado en el compartimento diseñado, quedando conectada de ésta manera la PTU con el compartimento mediante los cables de alimentación de ésta y el cable de comunicación USB.

A continuación, se explican los ajustes y calibraciones necesarias en los diferentes dispositivos para conseguir el correcto funcionamiento del sistema.

Inicialmente se proporcionó un módulo USB que contenía el conector macho soldado a éste tal y como puede observarse en la figura 4.10, sin embargo para comunicar a éste con la CPU de manera que se aislaran los tres módulos utilizados en este TFG, se desoldó el conector y se conectaron los pines mostrados en la figura 4.10 a un cable USB de un metro de longitud, permitiendo así la capacidad de poder conectar el dispositivo sin la necesidad de extraer el módulo del compartimento, formando una unidad dentro del compartimento y sin la necesidad de permanecer en contacto con el ordenador.

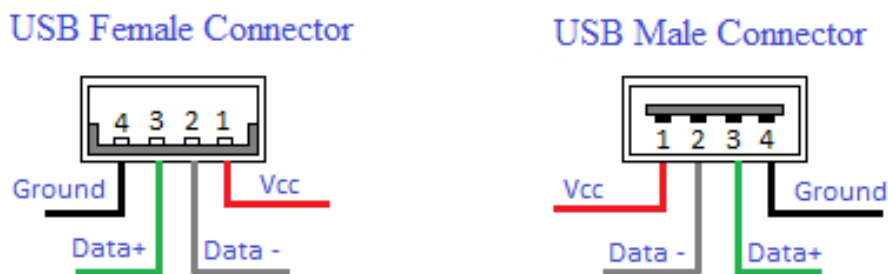


Figura 4.10: PINOUT de un cable USB. Imagen tomada de: <https://qph.fs.quoracdn.net/main-qimg-e0856d1289eb3384932797e94e0b816a>

El rango de tensión de entrada del distanciómetro comprende desde: 2,9 a 3,3 V. Por lo tanto, se ajustó la tensión de salida del *DC-DC Buck* a 3,1 voltios ajustando un potenciómetro del módulo para poder alimentar al módulo medidor de distancias dentro de su rango de funcionamiento.

4.3 Implementación del software de calibración

En esta sección se explicarán todas las funciones implementadas de tipo software. Para ello se ha utilizado el *middleware* ROS explicado en la sección 3.1.

4.3.1 Interfaz con el módulo láser

La interfaz con el módulo láser cumple la función de *driver* y es la encargada de comunicar los diferentes dispositivos con el módulo medidor de distancias para realizar las funciones de las que éste dispone, explicadas en el apartado 3. Previamente a la programación de los drivers y al montaje del módulo en el compartimento, se ha testado su funcionamiento utilizando el software *cutecom* para el sistema operativo *Ubuntu*, como terminal para establecer la comunicación por el puerto serie con el módulo. Posteriormente, se ha procedido a su montaje en el compartimento dando como resultado una tarea fácil y rápida debido al diseño del compartimento.

A continuación, se ha implementado el nodo de ROS encargado de establecer la comunicación con el módulo ofreciendo diferentes servicios que realizan las funciones de éste. Para el diseño de este nodo, se ha decidido utilizar los servicios frente a la publicación/subscripción debido a que los servicios se ajustan mejor a este tipo de tareas aperiódicas.

El nodo ofrece cuatro servicios diferentes:

- Encender dispositivo,
- Apagar dispositivo,
- Conocer la tensión de entrada y la temperatura del dispositivo.
- Medir la distancia en modo lento y

Los tres primeros están vacíos, es decir que no tienen ningún parámetro, tanto de entrada como de salida, siendo el servicio de medir la distancia el único en devolver un mensaje de tipo *float*.

El proceso que realizan los drivers consiste en el envío por el puerto serie de los comandos correspondientes a los servicios solicitados y a la lectura de la confirmación de estos. Tal y como se explica en la sección 4.3.2, el nodo *System Controller* está diseñado para que todos los servicios sean llamados utilizando diferentes botones a excepción de las funciones *encender* y *apagar el dispositivo*, donde se utiliza únicamente un botón con la finalidad de ahorrar el número de botones en uso.

Entrando en mayor profundidad, el módulo láser aporta la distancia desde el extremo del láser hasta un chip que contiene en el interior de su Printed Circuit Board (PCB). No obstante, lo que realmente se desea es conocer la distancia respecto al eje de rotación del servomotor superior de la PTU debido a que se ha decidido tomar ese punto

como el origen. Para conseguir referenciar la posición medida, se deben de realizar una serie de transformadas. Para ello se ha empleado una herramienta matemática de localización en el espacio, llamada la matriz de transformación homogénea, la cual se utiliza en caso de querer representar la posición y orientación de un sistema después de ser trasladado y rotado. Esta herramienta se puede extrapolar en dicha situación y el proceso seguido es el siguiente:

Se dispone de una PTU con capacidad de rotar sobre dos ejes: Y y Z (estas rotaciones se denominan *Pitch* y *Roll* respectivamente y son representados por las letras griegas θ y ϕ). Se desea calcular la distancia respecto al origen de coordenadas situado en el eje de rotación del servomotor superior de la PTU, el causante de orientación en *Tilt*, tal y como se muestra en la figura 4.5. La distancia denominada L es la que mide desde el chip interno del módulo láser a dicho origen. Teniendo en cuenta que nos encontramos en un espacio en 3D se utilizará Lx, Ly y Lz y denominamos D a la distancia medida por el dispositivo.

Primero de todo, se definen las siguientes matrices de transformación según los datos expuestos anteriormente donde a partir de este punto se abreviarán las operaciones trigonométricas de seno y coseno por una S y una C respectivamente. Las matrices T_P y T_T son la matrices homogéneas que hacen referencia a la rotación respecto al eje Z (*pan*) y respecto al eje Y (*Tilt*).

$$T_T = \begin{pmatrix} C(\theta) & 0 & S(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -S(\theta) & 0 & C(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.1)$$

$$T_P = \begin{pmatrix} C(\phi) & -S(\phi) & 0 & 0 \\ S(\phi) & C(\phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.2)$$

Para realizar la traslación desde el origen de coordenadas del dispositivo hasta el chip del interior del módulo medidor de distancias se ha definido la siguiente matriz de transformación homogénea de traslación, siendo L la distancia en tres dimensiones. Esta distancia se ha calculado con precisión midiendo las piezas.

$$T_L = \begin{pmatrix} 1 & 0 & 0 & Lx \\ 0 & 1 & 0 & Ly \\ 0 & 0 & 1 & Lz \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.3)$$

Por otro lado, el punto proyectado por el haz láser puede expresarse respecto al módulo medidor como:

$$P_L = \begin{pmatrix} D \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad (4.4)$$

Debido a que el eje de rotación del servomotor inferior coincide con el eje de rotación del servomotor superior, se ha realizado la multiplicación de sus respectivas matrices homogéneas debido a que las dos rotaciones se realizan desde el mismo origen de coordenadas situado en el eje de rotación del servomotor superior.

$$T_{PT} = \begin{pmatrix} C(\phi)C(\theta) & -S(\phi) & C(\phi)S(\theta) & 0 \\ S(\phi)C(\theta) & C(\phi) & S(\phi)S(\theta) & 0 \\ -S(\theta) & 0 & C(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.5)$$

Finalmente, las coordenadas del punto proyectado por el módulo láser expresados respecto el origen de coordenadas del dispositivo son:

$$P_o = T_{PT} T_L P_L \quad (4.6)$$

$$P_o = \begin{pmatrix} DC(\phi)C(\theta) + LxC(\phi)C(\theta) - LyS(\phi) + LzC(\phi)S(\theta) \\ DS(\phi)C(\theta) + LxS(\phi)C(\theta) + LyC(\phi) + LzS(\phi)S(\theta) \\ -DS(\theta) - LxS(\theta) + LzC(\theta) \\ 1 \end{pmatrix} \quad (4.7)$$

Donde:

$$X = (DC(\phi)C(\theta) + LxC(\phi)C(\theta) - LyS(\phi) + LzC(\phi)S(\theta)) \quad (4.8)$$

$$Y = DS(\phi)C(\theta) + LxS(\phi)C(\theta) + LyC(\phi) + LzS(\phi)S(\theta) \quad (4.9)$$

$$Z = -DS(\theta) - LxS(\theta) + LzC(\theta) \quad (4.10)$$

Es importante recordar que el dispositivo implementado calcula la posición de las balizas respecto a la primera posición asignada. Por lo tanto, conocidas las coordenadas de la baliza, utilizando el procedimiento explicado anteriormente, se resta dicha posición respecto a la tomada como "origen del mundo" que se ha seleccionado en la primera medida.

En el apéndice A.2 puede observarse el código de programación usado en la implementación del software encargado de realizar dichas funciones del nodo *LASER DRIVER*.

4.3.2 Software de control del dispositivo

Como se ha explicado brevemente en la sección 4.1 se ha creado un nodo, llamado *System Controller* encargado de coordinar y controlar las diferentes conexiones con los dispositivos de los cuales se encuentran: un mando tipo *gamepad*, la PTU y el módulo medidor de distancias.

Primero de todo, se ha decidido que el dispositivo se controlaría desde un mando tipo *gamepad* al cual se le han asignado diferentes funciones para cada botón de éste tal y como se ha explicado en la sección 4.1. Además, para entender los siguientes puntos de la memoria es importante mencionar que el software diseñado para el dispositivo medidor de distancias en 3D almacena para cada medida: el identificador asignado a cada posición, la distancia medida por el módulo láser, las coordenadas 3D del punto respecto al origen del dispositivo y por último la orientación de los dos servomotores por separado.

A continuación se explican las funciones indicadas en la figura 4.2:

Medir la distancia

Ésta es la función principal del dispositivo para cumplir con los objetivos que se han asignado para el presente proyecto. Para ello, el software implementado permite, apretando un botón, realizar una medición sin necesidad de guardar la información de ésta. Ésta función tiene la finalidad de utilizarse en diferentes tareas como la comprobación del correcto funcionamiento del dispositivo, la coherencia de la medida antes de ser guardada o donde no se requiera guardar dicha información. Por otro lado, cuenta con otros botones que disponen de funciones para guardar y eliminar la información de la muestra tomada que se almacena en la memoria del programa.

Consulta del estado del sistema

Cuenta con una función que muestra por pantalla el estado de los diferentes dispositivos hardware utilizados en el sistema que disponen de dicha capacidad de autoevaluación, permitiendo así conocer la temperatura o tensión de entrada del módulo láser y de la PTU, aportando para ésta última el par mecánico soportado por los servomotores. También cuenta con una función que permite mostrar por pantalla la información de las posiciones guardadas hasta el momento. Esta función se ha implementado debido a la conocida sensibilidad de los dispositivos hardware en entornos húmedos como es el caso del interior de un buque.

Control de velocidad

La PTU dispone de dos velocidades de movimiento para los dos servomotores, siendo el joystick izquierdo el encargado de controlar la orientación con una mayor velocidad y las flechas las responsables de tener un control más preciso disminuyendo notablemente su velocidad. Además cuenta con un tercer nivel de velocidad que permite el movimiento de los servomotores paso a paso cuyo ángulo se puede revisar en la sección 2.2 y también en el apéndice C.2 del presente proyecto. Es importante mencionar que dichas velocidades están asignadas por unos parámetros predeterminados que pueden ser modificados en cualquier momento para ajustar el funcionamiento del sistema en diferentes entornos.

Mover directamente a posiciones memorizadas

El dispositivo cuenta con una función que permite, una vez memorizadas una o más posiciones, mover el dispositivo directamente a cada una de ellas o en su defecto a la

posición de referencia. El objetivo asignado al que se moverá el dispositivo depende del botón del mando pulsado. En figura 4.2 se puede observar que el botón R1 dirige hacia el objetivo anterior y el L1 hacia el posterior. Una vez el dispositivo apunta sobre una posición guardada previamente se puede repetir la medida volviendo a pulsar el botón de *Tomar medida (guardar en memoria)*.

Mostrar el manual del mando

Esta función es absolutamente necesaria y se utiliza en el caso de que el operario en algún momento pueda tener alguna duda sobre el control del dispositivo y requiera de una ayuda inmediata. Ésta muestra una lista por pantalla de las funciones asignadas a los botones.

Encender/Apagar haz láser

Esta función permite al operario poder manejar si desea proyectar el láser o no. Realmente éstas son dos funciones asignadas al mismo botón. Se decidió implementar de esta manera debido que se ha intentado optimizar al máximo el uso de los botones y a que son dos funciones totalmente compatibles. La primera vez que se pulsa el botón se proyectará el haz láser, la siguiente vez se apagará.

Orientar a la posición inicial

Esta función se utiliza solo una vez después de colocar correctamente la orientación del dispositivo, ésta sirve para iniciar el recorrido de una sesión de calibración en el sentido antihorario, por lo tanto orienta el dispositivo todo lo posible hacia la derecha.

Por último, es importante explicar que la orientación del sistema de coordenadas lo decide la orientación de la PTU tal y como puede observarse en la figura 4.5, donde se puede apreciar que se ha escogido una referencia física para poder conocer la orientación sin tener que encender el sistema. Es importante mencionar que una vez asignado el punto de origen de las futuras mediciones ya no se puede mover el dispositivo de posición ni de orientación, en cuyo caso se deberá repetir el proceso de medición desde el principio debido al que el sistema precisará de la repetición del proceso de calibración.

En el apéndice A.1 puede observarse el código de programación usado en la implementación del software encargado de realizar dichas funciones del nodo *SYSTEM CONTROLLER*.

4.3.3 Otras funciones

A continuación, se explicarán otras funciones que solo se pueden acceder a ellas mediante la combinación de más de un botón tal y como puede observarse en la figura 4.3. Para activar estos controles secundarios se debe mantener el botón L2. Éstas son las siguientes:

Escritura del fichero de calibración

La siguiente función consiste en exportar la recopilación de información de todas las posiciones memorizadas a lo largo de una sesión de calibración aportando la infor-

mación explicada en la introducción de esta sección. Esta función no muestra dicha información por pantalla sino que la exporta en un fichero de texto, pudiendo así utilizar dicha información para otras funciones, como por ejemplo el uso de un sistema de posicionamiento UWB.

Orientación de calibración del dispositivo

Esta función se recomienda utilizarla al acabar la sesión de trabajo del dispositivo. De la misma manera que existe un botón para orientar el dispositivo a la orientación de inicio de una sesión de calibración, también se ha implementado una función que permite orientarla con Pan y Tilt nulos. Ésta se utiliza por un motivo estético que permite observar a primera vista el dispositivo en la orientación asignada como origen.

Método para realizar la evaluación

Esta función solo se ha utilizado en el proceso de evaluación y no se recomienda utilizarla en una sesión de trabajo. Ésta consiste en la realización de 100 mediciones con un tiempo de espera entre muestras de 2 segundos, guardando todos los datos relacionados con los experimentos y sus respectivos resultados y permite exportar diferentes ficheros utilizando el número de marcador utilizado en los experimentos.

EXPERIMENTOS

En este capítulo se explican los diferentes experimentos realizados con el objetivo de evaluar *la precisión y la repetitividad* del dispositivo de posicionamiento diseñado y comprobar que cumple con los objetivos puestos en el presente proyecto. Primero de todo, se define la metodología que se ha utilizado para realizar los experimentos. A continuación se explican los experimentos realizados utilizando el software *MOTIVE*. Finalmente, se explica el experimento de un caso real de calibración utilizando unas balizas de tecnología UWB.

5.1 Metodología

Previamente al análisis de los experimentos, es importante mencionar que todos los ensayos realizados para evaluar el diseño del dispositivo se han realizado en el laboratorio de robótica aérea debido a que este cuenta con la tecnología necesaria para poder comprobar los resultados con una alta precisión y debido a que dispone del espacio de trabajo suficiente.

Para evaluar los experimentos relacionados con las simulaciones de calibración de balizas UWB, se ha utilizado el sistema de captura de movimiento *Optitrack* (con el software *MOTIVE*) para obtener los valores de "Ground Truth". Esto se debe a que las posiciones de los objetos a medir están definidas con tan alta precisión por dicho software, como se explica en la sección 3.4, que se pueden considerar que los datos aportados por éste son exactos. Estos objetos a medir son unos marcadores recubiertos de un material especial que permite a las cámaras de *Optitrack* estimar su posición.

A continuación, se ha procedido a la medición de los marcadores utilizando el dispositivo de posicionamiento 3D. No obstante, es importante mencionar que debido a que el dispositivo utilizado para medir distancias es el módulo medidor de distancias láser, éste no puede apuntar directamente sobre éstos dispositivos de *optitrack* y se decidió buscar una alternativa sin dejar de utilizar dicho software. La solución escogida fue apuntar 5 cm debajo de cada marcador para evaluar el dispositivo. Se ha calculado

5. EXPERIMENTOS

la variación estándar del error cometido en cada marcador para evaluar la repetitividad, y la media del error para evaluar la precisión.

5.2 Evaluación de la repetitividad y precisión del dispositivo 1

En este experimento se han evaluado la precisión y la repetitividad del dispositivo de posicionamiento colocado en una única posición siguiendo la metodología explicada en la sección 5.1.

Para realizar dicha evaluación se han utilizado un total de 12 marcadores posicionados de forma estratégica en la zona de trabajo de las cámaras de *Optitrack* para poder evaluar el dispositivo correctamente. En la figura 5.1 puede observarse como están distribuidos los marcadores utilizados en este experimento. La posición de éstas se ha registrado en una lista como puede observarse en la tabla 5.1. Para cada marcador, se han tomado 100 medidas sin mover el dispositivo de sitio ni de orientación y con un tiempo de espera de 2 segundos entre cada medición. Como se ha indicado en el capítulo anterior, para realizar el experimento con mayor rapidez se programó un método que realizaba el experimento de forma automática que además finalizaba escribiendo un fichero con los datos tomados y los resultados del experimento.

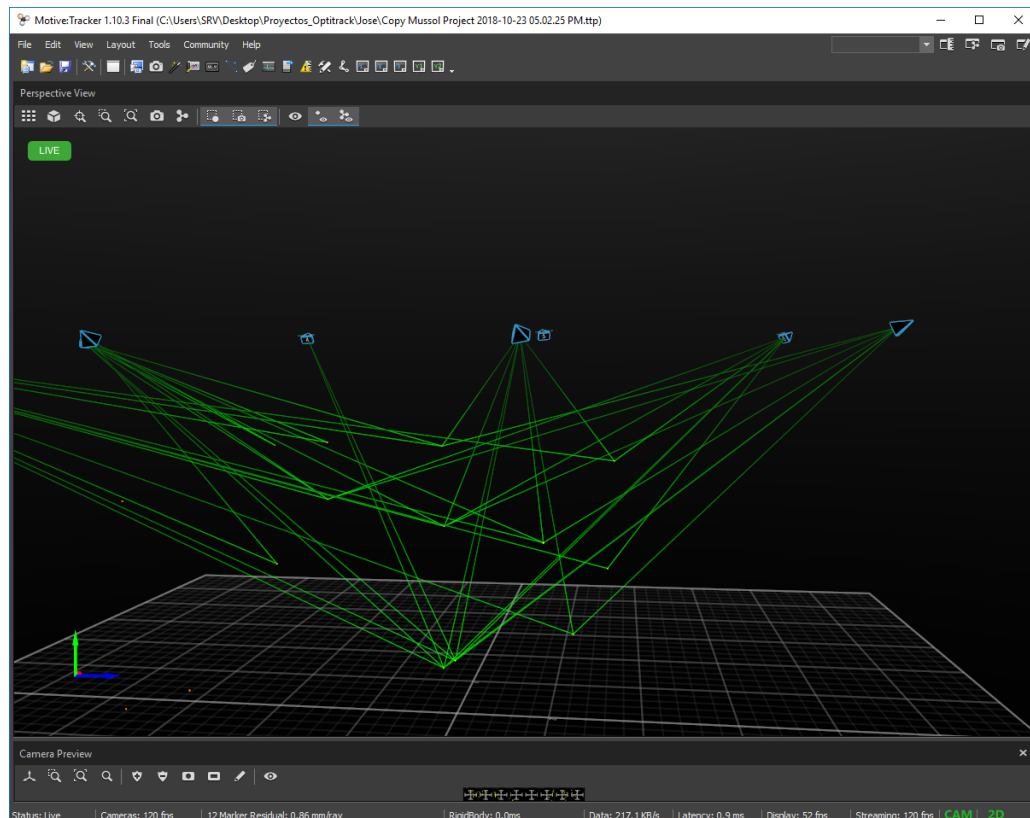


Figura 5.1: Vista de las cámaras apuntando a los marcadores utilizando *Optitrack*

Una vez realizado el experimento, se procedió a analizar los datos extraídos mediante un software de tipo hoja de cálculo con el cual se pudo calcular la repetitividad y

5.2. Evaluación de la repetitividad y precisión del dispositivo 1

Posición real			
ID marcador	X	Y	Z
1	0,23	-2,57	0,76
2	0,46	-2,32	1,78
3	2,09	-1,59	2,57
4	2,09	-1,58	1,17
5	0,95	0,21	0,25
6	0,80	0,32	0,21
7	2,09	0,60	2,74
8	2,08	0,58	1,67
9	2,07	2,19	2,80
10	2,07	2,18	2,03
11	2,07	2,90	2,74
12	2,07	2,90	1,12

Tabla 5.1: Posiciones de los marcadores proporcionados por *MOTIVE*

ID marcador	Desviación típica del error			Media del error		
	X	Y	Z	X	Y	Z
1	0,0006	0,0000	0,0001	0,02	0,02	-0,03
2	0,0030	0,0001	0,0010	0,02	0,00	-0,04
3	0,0005	0,0001	0,0002	0,03	0,04	-0,07
4	0,0004	0,0001	0,0000	0,02	0,00	0,01
5	0,0005	0,0005	0,0001	-0,04	-0,01	-0,04
6	0,0003	0,0003	0,0001	0,04	-0,03	-0,03
7	0,0008	0,0006	0,0003	0,04	-0,03	-0,03
8	0,0008	0,0005	0,0001	0,03	-0,04	0,01
9	0,0004	0,0004	0,0002	0,04	-0,02	-0,04
10	0,0004	0,0004	0,0001	0,04	-0,04	0,04
11	0,0008	0,0010	0,0003	0,04	-0,01	-0,04
12	0,0025	0,0032	0,0002	0,03	-0,03	0,05

Tabla 5.2: Resultados repetitividad y precisión por cada marcador.

precisión global de la totalidad de los datos recopilados de las 1200 muestras realizadas a partir de las 100 muestras de los 12 marcadores.

Desviación típica del error			Media del error		
X	Y	Z	X	Y	Z
0,02	0,02	0,04	0,02	-0,01	-0,01

Tabla 5.3: Resultados repetitividad y precisión global

Como se puede observar en la tabla 5.2 y la tabla 5.3 la repetitividad del error del dispositivo es muy baja, del orden de milímetros en el peor de los casos, lo que supone que el componente que influye mayormente en la repetitividad del sistema, el módulo medidor de distancias, funciona correctamente para los casos simulados. Por

otro lado, los datos recopilados de la precisión del dispositivo presentan valores más elevados llegando a 9 cm de error. Estos valores son atribuidos a la resolución de los servomotores.

5.3 Evaluación de la repetitividad y precisión del dispositivo 2

Para realizar este experimento se ha decidido calcular la repetitividad y precisión del dispositivo realizando el mismo proceso que en el experimento anterior, realizando el mismo número de mediciones por cada marcador (100 muestras) y utilizando exactamente el mismo número de objetivos (12 marcadores) colocadas en la misma posición, la cual se muestra en la figura ???. No obstante, a diferencia del experimento anterior la posición del dispositivo de medición 3D se modificó antes de realizar las muestras, mientras que el punto tomado como origen de coordenadas (situado en el centro del laboratorio) es el mismo. De esta manera se puede analizar si el posicionamiento del dispositivo puede alterar los resultados de la medición. Para esta primera parte, se utilizó el mismo método que en el experimento anterior debido a las grandes similitudes entre estos y debido a que no se modificó en absoluto la posición de los marcadores.

De la misma manera que en el experimento anterior, se ha calculado el resultado global de todas las muestras obtenidas en éste utilizando la misma metodología para su posterior comparación con las del primer experimento.

De los resultados extraídos mostrados en la tabla 5.4 y la tabla 5.5 se puede concluir que de la misma manera que en el anterior experimento, la repetitividad del error es muy alta para la distancia limitada por el módulo medidor de distancias láser, por lo que supone un dato muy interesante para futuros proyectos relacionados a este. Sin embargo, en este experimento se vuelven a encontrar los mismo problemas que en el experimento anterior, la precisión presenta un valor más alto. Por último, este experimento revela que el cambio de posición del dispositivo de medición no afecta considerablemente a los resultados extraídos.

ID marcador	Desviación típica del error			Media del error		
	X	Y	Z	X	Y	Z
1	0,0003	0,0005	0,0000	0,02	0,0019	-0,07
2	0,0001	0,0001	0,0000	0,06	0,01	-0,05
3	0,0003	0,0002	0,0002	0,04	0,09	-0,08
4	0,0024	0,0012	0,0002	0,02	0,01	-0,05
5	0,0007	0,0001	0,0002	-0,04	0,00	-0,06
6	0,0003	0,0001	0,0001	-0,04	-0,01	-0,05
7	0,0004	0,0001	0,0002	0,00	0,03	-0,04
8	0,0007	0,0001	0,0002	-0,01	0,05	-0,04
9	0,0007	0,0005	0,0004	0,00	-0,01	-0,04
10	0,0004	0,0003	0,0001	0,01	-0,02	-0,09
11	0,0005	0,0004	0,0002	0,02	0,01	-0,11
12	0,0021	0,0019	0,0001	-0,04	0,00	-0,04

Tabla 5.4: Resultados repetitividad y precisión por cada marcador para el 2º experimento

5.4. Evaluación de la precisión de los servomotores

Desviación típica del error			Media del error		
X	Y	Z	X	Y	Z
0,03	0,03	0,02	0,00	0,01	-0,06

Tabla 5.5: Resultados repetitividad y precisión global para el 2º experimento

5.4 Evaluación de la precisión de los servomotores

El objetivo de este experimento es evaluar la precisión de los servomotores. Para realizar este experimento se han tomado 10 medidas de cada marcador modificando la orientación del dispositivo para que éste vuelva al punto origen escogido entre cada medida.

Se ha decidido no modificar la posición de los marcadores debido a que ésta no era relevante para la correcta evaluación de este experimento. Por lo tanto las posiciones de éstas quedan definidas como se muestra en la tabla 5.1.

Para evaluar la precisión de los servomotores nos centramos en los valores de la desviación típica del error, ya que el movimiento de estos dispositivos entre cada muestra puede afectar a este indicador de dispersión de la medida.

Como puede observarse en la tabla 5.6, los resultados del experimento muestran una desviación típica de hasta 2 cm. Estos resultados aportan la información de que el movimiento de la PTU entre medidas, no afecta en gran medida a la precisión debido a que no se puede apreciar ningún cambio notable respecto a los resultados de los experimentos anteriores.

ID marcador	Desviación típica del error		
	X	Y	Z
1	0,0017	0,016	0,012
2	0,003	0,019	0,013
3	0,008	0,008	0,022
4	0,002	0,011	0,006
5	0,015	0,013	0,017
6	0,005	0,005	0,006
7	0,006	0,008	0,006
8	0,009	0,012	0,009
9	0,014	0,0014	0,004
10	0,017	0,014	0,010
11	0,016	0,010	0,011
12	0,007	0,005	0,018

Tabla 5.6: Resultados repetitividad por cada marcador para el 3º experimento

Por último, en este experimento también se ha realizado un análisis global de los resultados extraídos, tal y como se ha hecho para los experimentos explicados anteriormente. Como puede observarse en la tabla 5.7, el resultado de éste en cuanto a la desviación estándar del error de las muestras, éste presenta un error medio de hasta siete centímetros como máximo.

Desviación típica del error		
X	Y	Z
0,030	0,070	0,045

Tabla 5.7: Resultados repetitividad global para el 3º experimento

Por ello, podemos concluir que la precisión de los servomotores es suficiente para realizar una sesión de calibración de UWB.

5.5 Simulación de un caso real de una sesión de calibración de balizas UWB

En este experimento se ha reproducido un caso real de calibración de balizas UWB en el laboratorio de robótica aérea.

Hasta ahora se había apuntado con el dispositivo de calibración a los marcadores de *Optitrack*, no obstante para este experimento se han utilizado las balizas de tecnología UWB. Se han posicionado un total de 8 balizas repartidas a lo largo y ancho del laboratorio y se ha estimado su posición manualmente mediante un medidor láser. Los valores obtenidos se muestran en la columna "Manual" de la tabla 5.8.

Este experimento consta de dos partes. En la primera parte se han realizado 5 sesiones de calibración utilizando 8 balizas sin mover la posición del dispositivo de calibración después de cada sesión. En la segunda parte se han realizado las diferentes sesiones modificando la posición del dispositivo después de finalizar cada sesión. Cabe destacar que hemos tomado siempre el mismo punto del laboratorio como origen de coordenadas global (es decir, que la 1ª lectura la tomamos del mismo punto en cada experimento).

Como puede observarse en la tabla 5.8 los resultados proporcionados por el dispositivo implementado se asemejan notablemente entre ellos y a las muestras estimadas manualmente.

Finalmente, como puede observarse en la tabla 5.9, los resultados correspondientes a las diferentes sesiones realizadas desde diferentes puntos son muy similares y por lo tanto se puede confirmar que la posición de la PTU antes de la sesión de calibración del dispositivo no afecta a los resultados de ésta.

5.5. Simulación de un caso real de una sesión de calibración de balizas UWB

ID baliza		Manual	Repetición 1	Repetición 2	Repetición 3
1	X	2,08	2,08	2,11	2,06
	Y	0,22	0,27	0,18	0,22
	Z	4,65	4,65	4,53	4,64
2	X	2,13	2,15	2,16	2,21
	Y	-3,56	-3,56	-3,57	-3,55
	Z	2,18	2,12	2,35	2,12
3	X	-2,85	-2,74	-2,69	-2,76
	Y	-3,98	-4,04	-4,07	-4,07
	Z	2,50	2,54	2,65	2,53
4	X	-1,78	-1,80	-1,78	-1,83
	Y	1,12	1,14	1,11	1,12
	Z	0,10	0,08	0,08	0,07
5	X	-4,20	-4,17	-4,17	-4,24
	Y	1,33	1,17	1,18	1,19
	Z	2,08	2,75	2,75	2,74
6	X	0,84	0,70	0,72	0,70
	Y	5,71	5,66	5,62	5,61
	Z	1,63	1,59	1,67	1,63
7	X	2,08	2,10	2,15	2,09
	Y	2,96	2,98	2,93	2,90
	Z	4,65	4,65	4,60	4,59
8	X	2,04	1,91	1,94	1,94
	Y	0,97	0,98	1,00	1,00
	Z	0,10	0,03	0,06	0,06

Tabla 5.8: Resultados de una sesión de calibración: sin modificar la posición de la PTU.

5. EXPERIMENTOS

ID baliza		Manual	Repetición 1	Repetición 2	Repetición 3
1	X	2,08	2,08	2,09	2,15
	Y	0,22	0,27	0,17	0,26
	Z	4,65	4,65	4,51	4,67
2	X	2,13	2,15	2,14	2,15
	Y	-3,56	-3,56	-3,54	-3,55
	Z	2,18	2,12	2,18	2,06
3	X	-2,85	-2,74	-2,75	-2,66
	Y	-3,98	-4,04	-3,98	-4,04
	Z	2,50	2,54	2,49	2,54
4	X	-1,78	-1,80	-1,74	-1,76
	Y	1,12	1,14	1,15	1,13
	Z	0,10	0,08	0,13	0,13
5	X	-4,20	-4,17	-4,12	-4,07
	Y	1,33	1,17	1,28	1,18
	Z	2,08	2,75	2,83	2,88
6	X	0,84	0,70	0,84	0,76
	Y	5,71	5,66	5,62	5,67
	Z	1,63	1,59	1,55	1,59
7	X	2,08	2,10	2,17	2,14
	Y	2,96	2,98	2,93	3,00
	Z	4,65	4,65	4,57	4,55
8	X	2,04	1,91	1,95	1,95
	Y	0,97	0,98	0,98	0,97
	Z	0,10	0,03	-0,02	0,01

Tabla 5.9: Resultados de una sesión de calibración: modificando la posición de la PTU.

CONCLUSIONES Y TRABAJO FUTURO

En este capítulo se explicarán las conclusiones extraídas de la realización del presente proyecto además del posible trabajo futuro con el que se podrían obtener mejores prestaciones.

6.1 Conclusiones

Como se ha explicado en el apartado 1 del presente proyecto, la metodología utilizada para calcular la posición de las balizas utilizadas de un sistema UWB supone una tarea larga, tediosa, durante la cual se pueden producir errores con cierta facilidad.

Por ese motivo y además para realizar diferentes funciones que faciliten el proceso de medición y de extracción de la información obtenida, en este proyecto se ha propuesto el diseño de un dispositivo medidor de posiciones 3D. La solución presentada aporta la comodidad al operador de controlar el dispositivo con un simple mando tipo *gamepad*, que permite guardar las coordenadas de los diferentes puntos (las coordenadas de las balizas UWB), además de la reducción en el error y el tiempo invertido.

En primer lugar, se han conectado electrónicamente los diferentes módulos que hacen posible el correcto funcionamiento del sistema y se ha diseñado un compartimento con una impresora 3D con la finalidad de almacenar la totalidad de los distintos dispositivos electrónicos y poder conectarlos a la PTU de forma que resulte fácil el desmontaje de éste.

En segundo lugar, se ha programado el software para ser controlado por un mando tipo *gamepad*, el cual tiene asignadas en los botones las siguientes funciones: el movimiento del dispositivo, acceder al driver diseñado para utilizar las diferentes funciones del modulo medidor de distancias, la extracción de un fichero con los datos recopilados, así como diferentes funciones que permiten una mayor capacidad de manejo, la cual resulta muy útil en tareas que requieren mucho tiempo.

Posteriormente, se ha evaluado el dispositivo de medición dentro del laboratorio de robótica aérea midiendo la precisión y repetitividad usando un sistema de captura

de movimiento.

Además se han realizado diferentes experimentos realizando un proceso real de calibración de un sistema de posicionamiento UWB.

Los resultados de los experimentos efectuados muestran que se han cumplido los objetivos con éxito y que por lo tanto el dispositivo se puede utilizar como un dispositivo de medición, teniendo en cuenta que el error máximo aportado es menor de un 5 % en la posición.

Debemos tener en cuenta que estos valores están directamente relacionados con las características de los elementos *hardware* usados en el presente proyecto (precisión y resolución de los servomotores y del medidor de distancias láser).

Por lo tanto, el uso de otros componentes, en concreto, de mejores prestaciones, se espera que mejorasen también las prestaciones del dispositivo implementado. Algunas posibles mejoras se mencionan en la siguiente sección.

Finalmente, cabe destacar que el dispositivo implementado puede tener otros usos en procesos que requieran la estimación de coordenadas 3D de puntos fijos en el entorno. Además, éste permite la capacidad de ser manipulada por una persona minusválida. Por lo tanto, permite a personas con dicha discapacidad a realizar estas sesiones de calibración o otros trabajos que requieran del uso de este dispositivo.

6.2 Trabajo futuro

En este apartado se presentan algunos de los trabajos futuros que podrían contribuir a mejorar las prestaciones del dispositivo implementado.

- **Módulo medidor de distancias:** El módulo que se utiliza actualmente tiene una distancia máxima de medición de 50 m que puede utilizarse correctamente en un entorno de longitud máxima de 100 metros. Si se desea abarcar entornos de mayor tamaño se requerirá de la adquisición de un módulo láser de mejores prestaciones.
- **Los servomotores de la PTU:** Los servomotores de la PTU adquirida para este proyecto tienen una resolución de 0,29° pudiendo perder información de hasta un máximo de 25 cm en un entorno con una longitud de 50 metros. Si se desea reducir dicho error o aumentar el tamaño del entorno se requiere del uso de servomotores de mayor resolución.
- **Reconocimiento del entorno:** El software implementado en el presente proyecto cumple con los objetivos mencionados en la sección 1.2. No obstante, introduciendo algunas modificaciones, el dispositivo podría realizar otras tareas como la medición de áreas o volúmenes. Esta funcionalidad permitiría conocer el tamaño de un pared o una casa sin necesidad de realizar la operación manualmente.
- **Mapeado del entorno:** Realizando ciertas modificaciones en el software, también se puede conseguir realizar un proceso de barrido automatizado para realizar un mapa del entorno en un entorno gráfico.



CÓDIGO SOFTWARE

A.1 Código del nodo implementado *System Controller*

A.1.1 Fichero `phantomtower.yaml`

```
port: /dev/ttyUSB0
read_rate: 15
write_rate: 25
joints: {
  arm_shoulder_pan_joint: {id: 1, neutral: 0, max_angle: 310, min_angle: 0,
    max_speed: 90},
  arm_shoulder_tilt_joint: {id: 2, max_angle: 110, min_angle: -110, max_speed:
    90},
}
controllers: {
  arm_controller: {type: follow_controller, joints: [arm_shoulder_pan_joint,
    arm_shoulder_tilt_joint], action_name: arm_controller/
    follow_joint_trajectory, onboard: False }
}
```

A.1.2 Fichero `threeD.launch`

```
<launch>

  <!-- arbotix node -->
  <node name="arbotix" pkg="arbotix_python" type="arbotix_driver" output="
    screen">
    <rosparam file="/home/jose/tfg_ws/src/ptu_laser_meter/launch/
      phantom_tower.yaml" command="load" />
  </node>

  <!-- tower_teleop node -->
```

A. C DIGO SOFTWARE

```
<node name="tower_teleop" pkg="ptu_laser_meter" type="tower_teleop.py" output="screen">

    <param name="servo_step" value="0.1" />
    <param name="factor_slow_tilt" value="1" />
    <param name="factor_slow_pan" value="1" />
    <param name="factor_fast_tilt" value="9" />
    <param name="factor_fast_pan" value="9" />
    <param name="frequency" value="50" />
    <param name="P_pan_ini" value="0" />
    <param name="P_tilt_ini" value="19.924" />
    <param name="max_num_anchors" value="100" />

</node>

<!-- joy node -->
<node respawn="true" pkg="joy" type="joy_node" name="joy" output="screen">
    >
    <param name="dev" type="string" value="/dev/input/js0" />
    <param name="deadzone" value="0.1" />

    <remap from="/joy" to="/gamepad" />
</node>

<!-- laser meter node -->
<node respawn="true" pkg="laser_drivers" type="laserMeter_node" name="laser_meter" output="screen">

    <remap from="~open_laser" to="/open_laser" />
    <remap from="~close_laser" to="/close_laser" />
    <remap from="~read_laser" to="/read_laser" />
    <remap from="~laser_conditions" to="/laser_conditions" />

</node>

</launch>
```

A.1.3 Fichero tower_teleop.py

```
#!/usr/bin/env python
# license removed for brevity
#####
#####
#
#   THIS PROGRAM LAUNCH A ROS NODE THAT CONTROLS ALL OF THE PROJECT DEVICES.
#
#   * LIBRARIES
#   * CLASS
#       * INITIALIZATIONS
#       * INTERRUPTIONS CALLBACK
#       * LASER SERVICES
#       * PROGRAM
#           * INITIALIZATION
#           * PRIMARY PROGRAM
#               * MOVEMENT
#               * PRIMARY BUTTONS
```

```

#          * SECONDARY BUTTONS
#          * ROBUSTNESS PROGRAM
#          * FUNCTIONS
#          * MAIN PROGRAMM
#
#####
#####

#####
# LIBRARIES
#####
import rospy
import time
import math
import statistics
from statistics import stdev
from std_msgs.msg import String
from std_msgs.msg import Float64
from sensor_msgs.msg import Joy
from srv_drivers.srv import *

#####
# CLASS
#####
class TowerController(object):

    def __init__(self, buttons, axes, factor_fast_pan, factor_fast_tilt,
                 factor_slow_pan, factor_slow_tilt, servo_step, P_pan_ini, P_tilt_ini,
                 P_pan, P_tilt, button_pressed, P_pan_servo, P_tilt_servo):

#####
# INITIALIZATION
#####
        self.buttons = buttons    # Buttons gamepad
        self.axes = axes          # Axes gamepad
        self.factor_fast_pan = factor_fast_pan    # Speed gamepad
        self.factor_fast_tilt = factor_fast_tilt
        self.factor_slow_pan = factor_slow_pan
        self.factor_slow_tilt = factor_slow_tilt
        self.servo_step = servo_step
        self.P_pan_ini = P_pan_ini    # Slow gamepad
        self.P_tilt_ini = P_tilt_ini
        self.P_pan = P_pan
        self.P_tilt = P_tilt
        self.button_pressed = button_pressed
        self.P_pan_servo = P_pan_servo
        self.P_tilt_servo = P_tilt_servo

# Callback of the gamepad
def callback(self, joy_msg):
    self.buttons = joy_msg.buttons
    self.axes = joy_msg.axes

def pan_callback(self, pan):
    self.P_pan_servo = pan.data * 180.0/math.pi

def tilt_callback(self, tilt):

```

```
        self.P_tilt_servo = tilt.data * 180.0/math.pi

# gamepad node interruption
def joy_listener(self):
    rospy.Subscriber('/gamepad', Joy, self.callback)

def arbotix_listener(self):
    rospy.Subscriber('/arm_shoulder_pan_joint/command', Float64, self.
        pan_callback)
    rospy.Subscriber('/arm_shoulder_tilt_joint/command', Float64, self.
        tilt_callback)

#####
# LASER SERVICES
#####
# Open laser service
def open_laser(self):
    rospy.wait_for_service('/open_laser')

    try:
        opn_laser = rospy.ServiceProxy('/open_laser', OpenLaser)
        opn_laser()
    except rospy.ServiceException, e:
        print "Service_call_failed:_%s"%e

# Close laser service
def close_laser(self):
    rospy.wait_for_service('/close_laser')

    try:
        cls_laser = rospy.ServiceProxy('/close_laser', CloseLaser)
        cls_laser()
    except rospy.ServiceException, e:
        print "Service_call_failed:_%s"%e

# Read laser distance service
def read_laser(self):
    rospy.wait_for_service('/read_laser')

    try:
        rd_laser = rospy.ServiceProxy('/read_laser', ReadLaser)
        laser_distance = rd_laser()
        return laser_distance.dist
    except rospy.ServiceException, e:
        print "Service_call_failed:_%s"%e

# show the laser state information service
def laser_conditions(self):
    rospy.wait_for_service('/laser_conditions')

    try:
        cond_laser = rospy.ServiceProxy('/laser_conditions', LaserConditions)
        cond_laser()
    except rospy.ServiceException, e:
        print "Service_call_failed:_%s"%e

#####
```

```

# INITIALIZATION
#####

# Shows manual and move the servos to the calibration process start position
def setup(self):
    self.setGamepadManual()
    self.movePanToPos(self.P_pan_ini)
    self.moveTiltToPos(self.P_tilt_ini)

#####
# PRIMARY PROGRAM
#####

def loop(self):
    global closed_laser
    global id_anchor
    global anchors_list
    global num_anchors_assigned
    global num_ball

    if self.button_pressed == False:

#####
# MOVEMENT
#####

        # Allows arrows to move the device slowly
        if self.axes[4] != 0 and self.axes[5] == 0:
            if self.buttons[6] == 1: # activate secondary functions
                self.button_pressed = True
                pan_angle_added = self.axes[4] * self.servo_step
                tilt_angle_added = -self.axes[5] * self.servo_step
                self.P_pan = self.P_pan + pan_angle_added
                self.P_tilt = self.P_tilt + tilt_angle_added

            elif self.axes[5] != 0 and self.axes[4] == 0:
                if self.buttons[6] == 1: # activate secondary functions
                    self.button_pressed = True
                    pan_angle_added = self.axes[4] * self.factor_slow_pan * self.servo_step
                    tilt_angle_added = -self.axes[5] * self.factor_slow_tilt * self.servo_step
                    self.P_pan = self.P_pan + pan_angle_added
                    self.P_tilt = self.P_tilt + tilt_angle_added

        # Left joystick to move fastly
        elif (self.axes[0] != 0 or self.axes[1] != 0):
            pan_angle_added = self.axes[0] * self.factor_fast_pan * self.servo_step
            tilt_angle_added = -self.axes[1] * self.factor_fast_tilt * self.servo_step
            self.P_pan = self.P_pan + pan_angle_added
            self.P_tilt = self.P_tilt + tilt_angle_added

        # PTU position limitations
        if self.P_pan < 0.000:

```

```
        self.P_pan = 0.000

    elif self.P_pan > 299.739:
        self.P_pan = 299.739

    elif self.P_tilt < -50:
        self.P_tilt = -50

    elif self.P_tilt > 50.000:
        self.P_tilt = 50.000
    self.movePanToPos(self.P_pan)
    self.moveTiltToPos(self.P_tilt)

#####
# PRIMARY BUTTONS
#####

# Button Triangle to save the position or to write a text file
if self.buttons[0] == 1:
    self.button_pressed = True
    # Write a text file with the anchors position information
    if self.buttons[6] == 1: # activate secondary functions
        if (num_anchors_assigned > 0):
            self.writing_text(anchors_list, num_anchors_assigned)
            self.write_text_distance_A2L(anchors_list,
                                         num_anchors_assigned)
        # Give a warning that no anchors has been assigned
        else:
            if (num_anchors_assigned == -1):
                print("Please, take minimum one anchor measure and its reference")
            else:
                print("Please, take minimum one anchor measure")
        archivo = open("postions_servos.txt", "w")
        archivo.write("sample, " + "distance, " + "X, " + "Y, " + "Z, " + "PAN, " + "TILT, " + "\n")
    # Take measure and save it to the database
    else:
        closed_laser = True
        measure = self.take_measure()
        # Save a new anchor in the database if the ID is not registered
        if (id_anchor == num_anchors_assigned) and (
            num_anchors_assigned < max_num_anchors):
            id_anchor = id_anchor + 1
            num_anchors_assigned = num_anchors_assigned + 1
            anchors_list = self.add_anchors_list(anchors_list,
                                                  id_anchor, measure)
        # Change the anchor information of the new measure
        elif (id_anchor > 0):
            self.change_anchors_list(anchors_list, id_anchor, measure,
                                     num_anchors_assigned)
        self.print_anchors_position(anchors_list, 0,
                                    num_anchors_assigned)

# Button circle to open and close the laser
elif self.buttons[1] == 1:
```



```

self.button_pressed = True
if closed_laser == True:
    closed_laser = False
    self.open_laser()
else:
    closed_laser = True
    self.close_laser()

# Button cross to read laser distance
elif self.buttons[2] == 1:
    self.button_pressed = True
    # Activate the assessment method to make it easier and faster
    if (self.buttons[6] == 1): # activate secondary functions
        if (id_anchor == 0):
            closed_laser = True
            average = [0,0,0]
            sampleX = []
            sampleY = []
            sampleZ = []
            # Take "max_num_anchors" measures for each marker
            for i in range (0, (max_num_anchors)):
                measure = self.take_measure()
                id_anchor = id_anchor + 1
                num_anchors_assigned = num_anchors_assigned + 1
                anchors_list = self.add_anchors_list(anchors_list,
                    id_anchor, measure)
                time.sleep(2)
            # Write a .txt file with the samples data
            archivo = open("Sample_n_%" % num_ball + ".txt", "w")
            archivo.write("sample," + "distance," + "X," + "Y," + "Z,"
                + "PAN," + "TILT," + "\n")
            for i in range (0,max_num_anchors+1):
                text = "%" % anchors_list[i*7] + ","
                for a in range (1,7):
                    text = text + "%.5f" % anchors_list[(i*7+a)] + ","
                archivo.write(text + "\n")
                if (i>0):
                    sampleX.append(anchors_list[(i*7+2)])
                    sampleY.append(anchors_list[(i*7+3)])
                    sampleZ.append(anchors_list[(i*7+4)])
                    average[0] = average[0] + anchors_list[(i*7+2)]
                    average[1] = average[1] + anchors_list[(i*7+3)]
                    average[2] = average[2] + anchors_list[(i*7+4)]
            archivo.close()
            # Write a sample results .txt file
            print ("Results_text_file_is_written")
            self.write_samples_result(average, max_num_anchors,
                sampleX, sampleY, sampleZ, num_ball)
            num_ball = num_ball + 1
            id_anchor = 0
            num_anchors_assigned = 0
            for i in range(0,(7*max_num_anchors)):
                anchors_list.pop()
        else:
            print ("Need_to_take_the_samples_reference_before")
    # Take measure without saving it into the database
else:

```

```
        closed_laser = True
        measure = self.take_measure()
        anchors_list = self.add_anchors_list(anchors_list, id_anchor,
                                              measure)
        self.print_anchors_position(anchors_list,
                                    num_anchors_assigned+1, num_anchors_assigned+1)
        for i in range(0,7):
            anchors_list.pop()

# Button square to delete the last database position
elif self.buttons[3] == 1 and num_anchors_assigned >= 0:
    self.button_pressed = True
    for i in range(0,7):
        anchors_list.pop()
    if id_anchor == num_anchors_assigned:
        id_anchor = id_anchor -1
    num_anchors_assigned = num_anchors_assigned -1
    self.print_anchors_position(anchors_list, 0, num_anchors_assigned
                                )

#####
# SECONDARY BUTTONS
#####

# Button START to show the gamepad manual
elif self.buttons[9] == 1:
    self.button_pressed = True
    self.setGamepadManual()

# Button SELECT to show the position of the servos
elif self.buttons[8] == 1:
    self.button_pressed = True
    P_servo = "\n_THE_SERVO'S_PAN_POSITION_IS_%.2f\n" % self.
        P_pan_servo
    P_servo = P_servo + "THE_SERVO'S_TILT_POSITION_IS_%.2f\n" % self.
        P_tilt_servo
    print(P_servo)
    self.laser_conditions()
    if (num_anchors_assigned>=0):
        self.print_anchors_position(anchors_list, 0,
                                    num_anchors_assigned)
    else:
        print("NO_ANCHORS_AND_REFERENCE_DETECTED\n")
    print(num_anchors_assigned)

# Button R2 to move the servos to their start position
elif self.buttons[7] == 1:
    self.button_pressed = True
    # Move to the Hardware calibration position
    if self.buttons[6] == 1: # activate secondary functions
        self.P_pan = 149.21
    # Move to the calibration season initial position
    else:
        self.P_pan = P_pan_ini
    self.P_tilt = P_tilt_ini
    self.movePanToPos(self.P_pan)
    self.moveTiltToPos(self.P_tilt)
```

```

# Button L1 to go to the next anchor
elif self.buttons[4] == 1:
    self.button_pressed = True
    # Give a warning that no anchors has been assigned
    if num_anchors_assigned <= 0:
        print ("NO_ANCHOR_ASSIGNED,_INVALID_PROCESS")
    else:
        if id_anchor == num_anchors_assigned:
            id_anchor = 0
        else:
            id_anchor = id_anchor +1
        self.P_pan = anchors_list[id_anchor*7 +5]
        self.P_tilt = anchors_list[id_anchor*7 +6]
        self.movePanToPos( self.P_pan)
        self.moveTiltToPos( self.P_tilt)

# Button R1 to go to the next anchor
elif self.buttons[5] == 1:
    self.button_pressed = True
    # Give a warning that no anchors has been assigned
    if num_anchors_assigned <= 0:
        print ("NO_ANCHOR_ASSIGNED,_INVALID_PROCESS")
    else:
        if id_anchor == 0:
            id_anchor = num_anchors_assigned
        else:
            id_anchor = id_anchor -1

        self.P_pan = anchors_list[id_anchor*7 +5]
        self.P_tilt = anchors_list[id_anchor*7 +6]
        self.movePanToPos( self.P_pan)
        self.moveTiltToPos( self.P_tilt)

#####
# PROTECTION PROGRAM
#####

# Allows to press the same or another button again
else:
    buttons_sum = 0
    axes_sum = 0

    for i in range (0,12):
        buttons_sum = buttons_sum + self.buttons[i]

    for i in range (0,6):
        if self.axes[i] != 0:
            axes_sum = 1

    if (buttons_sum == 0) or (self.buttons[6] == 1 and axes_sum == 0 and
        buttons_sum == 1):
        self.button_pressed = False
if (self.P_pan < self.P_pan_servo + 0.1 and self.P_pan > self.P_pan_servo
    - 0.1 and self.P_tilt < self.P_tilt_servo + 0.1 and self.P_tilt >
    self.P_tilt_servo - 0.1):

```

```

        self.P_pan = self.P_pan_servo
        self.P_tilt = self.P_tilt_servo
    self.setparams()

#####
# FUNCTIONS
#####
# Shows the gamepad manual on the screen
def setGamepadManual( self ):
    print ( "\n\n" )
    localtime = time.asctime( time.localtime(time.time()) )
    print ( "_____Date_ " + localtime )
    print ( " \n_____GAMEPAD_INFORMATION\n" )
    print ( " _____MOVEMENT: \n" )
    print ( " _____LEFT_JOYSTICK: _MOVE_FASTER" )
    print ( " _____L2+_ARROWS: _____MOVE_STEP_BY_STEP" )
    print ( " _____ARROWS: _____MOVE_SLOWER" )
    print ( " \n_____PRIMARY_BUTTONS: \n" )
    print ( " _____TRIANGLE: _____SAVE_ANCHOR_POSITION" )
    print ( " _____CIRCLE: _____ON/OFF_LASER" )
    print ( " _____CROSS: _____TAKE_MEASURE" )
    print ( " _____SQUARE: _____DELETE_LAST_ANCHOR_POSITION" )
    print ( " \n_____SECONDARY_BUTTONS: \n" )
    print ( " _____START: _____GAMEPAD_INFORMATION" )
    print ( " _____SELEC: _____STATUS" )
    print ( " _____R2: _____MOVE_TO_CALIBRATION_SEASON_INITIAL_
        POSITION" )
    print ( " _____R1: _____MOVE_TO_THE_POSITION_OF_THE_PREVIOUS_
        ANCHOR" )
    print ( " _____L1: _____MOVE_TO_THE_POSITION_OF_THE_NEXT_ANCHOR"
        )
    print ( " \n_____COMBINATION_BUTTONS: \n" )
    print ( " _____L2+_TRIANGLE: _WRITE_IN_A_txt_FILE_THE_ANCHORS_POSITION
        " )
    print ( " _____L2+_R2: _____MOVE_TO_THE_HARDWARE_CALIBRATION_
        POSITION" )
# Move the pan servo position
def movePanToPos( self , pose ):
    pub_pan.publish( pose *math.pi/180 )
# Move the tilt servo position
def moveTiltToPos( self , pose ):
    pub_tilt.publish( pose*math.pi/180 )
# set the parameters
def setparams( self ):
    self.factor_fast_pan = rospy.get_param( '~factor_fast_pan' )
    self.factor_fast_tilt = rospy.get_param( '~factor_fast_tilt' )
    self.factor_slow_pan = rospy.get_param( '~factor_slow_pan' )
    self.factor_slow_tilt = rospy.get_param( '~factor_slow_tilt' )
    self.servo_step = rospy.get_param( '~servo_step' )
    self.frequency = rospy.get_param( '~frequency' )
    self.P_pan_ini = rospy.get_param( '~P_pan_ini' )
    self.P_tilt_ini = rospy.get_param( '~P_tilt_ini' )
    max_num_anchors = rospy.get_param( '~max_num_anchors' )

# perform the calculations to get the measure in 3D
def take_measure( self ):

```

```

distance = self.read_laser() - 0.006 # this last term is the calibration
    calculation, did it with another laser meter.
# Distance between the laser module to the upper servo position
Lx = 0.003
Ly = 0.000
Lz = 0.043
# Trigonometric operations using Pan & Tilt angle
CP = math.cos((self.P_pan_servo-149.21)*math.pi/180)
SP = math.sin((self.P_pan_servo-149.21)*math.pi/180)
CT = math.cos(-self.P_tilt_servo*math.pi/180)
ST = math.sin(-self.P_tilt_servo*math.pi/180)
# Results
X = distance*CP*CT + Lx*CP*CT - Ly*SP + Lz*CP*ST
Y = distance*SP*CT + Lx*SP*CT + Ly*CP + Lz*SP*ST
Z = distance*ST - Lx*ST + Lz*CT
measure=[distance, X, Y, Z]
return measure

# Add the new position to the database
def add_anchors_list(self, anchors_list, id_anchor, measure):
    if id_anchor > 0:
        measure[1] = measure[1] - anchors_list [2]
        measure[2] = measure[2] - anchors_list [3]
        measure[3] = measure[3] - anchors_list [4]
    anchors_list.append(id_anchor)
    anchors_list.append(measure[0])
    anchors_list.append(measure[1])
    anchors_list.append(measure[2])
    anchors_list.append(measure[3])
    anchors_list.append(self.P_pan_servo)
    anchors_list.append(self.P_tilt_servo)
    return anchors_list

# Change a previously assigned position to the database
def change_anchors_list(self, anchors_list, id_anchor, measure,
    num_anchors_assigned):
    if id_anchor == 0:
        for i in range(1,num_anchors_assigned + 1):
            anchors_list[i*7 + 2] = anchors_list[i*7 + 2] + anchors_list[2] -
                measure[1]
            anchors_list[i*7 + 3] = anchors_list[i*7 + 3] + anchors_list[3] -
                measure[2]
            anchors_list[i*7 + 4] = anchors_list[i*7 + 4] + anchors_list[4] -
                measure[3]
        anchors_list[id_anchor*7 + 1] = measure[0]
        anchors_list[id_anchor*7 + 2] = measure[1] - anchors_list [2]
        anchors_list[id_anchor*7 + 3] = measure[2] - anchors_list [3]
        anchors_list[id_anchor*7 + 4] = measure[3] - anchors_list [4]
        anchors_list[id_anchor*7 + 5] = self.P_pan_servo
        anchors_list[id_anchor*7 + 6] = self.P_tilt_servo
    return anchors_list

# Shows the database on the screen
def print_anchors_position(self, anchors_list, initvalue,
    num_anchors_assigned):
    text = ""
    for i in range (initvalue, num_anchors_assigned + 1):
        text = text + "ID=" + str(anchors_list[i*7])
        text = text + "D=" + str(anchors_list[(i*7+1)] + "m"

```

```

        text = text + "_X=%.5f" %anchors_list[(i*7+2)] + "m"
        text = text + "_Y=%.5f" %anchors_list[(i*7+3)] + "m"
        text = text + "_Z=%.5f" %anchors_list[(i*7+4)] + "m"
        text = text + "_PAN=%.2f" %anchors_list[(i*7+5)] + "o"
        text = text + "_TILT=%.2f" %anchors_list[(i*7+6)] + "o"
        text= text + "\n"
    print (text)
#write the database on a .txt file
def writting_text(self, anchors_list, num_anchors_assigned):
    print("escribo_el_fichero_Position_of_the_anchors_(from_reference).txt")
    archivo = open("Position_of_the_anchors_(from_reference).txt","w")
    archivo.write("anchor_ID," + "distance," + "X," + "Y," + "Z," + "PAN," + "
        TILT," + "\n")
    for i in range (0,num_anchors_assigned +1):
        text = "%d" %anchors_list[i*7] + ","
        for a in range (1,7):
            text = text + "%.5f," %anchors_list[(i*7+a)]
        archivo.write(text + "\n")
    archivo.close()
#write the measures of the database without reference (the "world" position)
    comparison on a .txt file
def write_text_distance_A2L(self, anchors_list, num_anchors_assigned):
    print("escribo_el_fichero_Position_of_the_anchors_(without_reference).txt")
    archivo = open("Position_of_the_anchors_(without_reference).txt","w")
    archivo.write("anchor_ID," + "distance," + "X," + "Y," + "Z," + "PAN," + "
        TILT," + "\n")
    for i in range (0,num_anchors_assigned +1):
        text = "%d" %anchors_list[i*7] + ","
        text = text + "%.5f" %anchors_list[(i*7+1)] + ","
        if (i==0):
            text = text + "%.5f" %anchors_list[(i*7+2)] + ","
            text = text + "%.5f" %anchors_list[(i*7+3)] + ","
            text = text + "%.5f" %anchors_list[(i*7+4)] + ","
        else:
            text = text + "%.5f" %(anchors_list[(i*7+2)] + anchors_list[2]) +
                ","
            text = text + "%.5f" %(anchors_list[(i*7+3)] + anchors_list[3]) +
                ","
            text = text + "%.5f" %(anchors_list[(i*7+4)] + anchors_list[4]) +
                ","
        text = text + "%.2f" %anchors_list[(i*7+5)] + ","
        text = text + "%.2f" %anchors_list[(i*7+6)]
        archivo.write(text + "\n")
    archivo.close()
#write the assessment results on a .txt file
def write_samples_result(self, average, max_num_anchors, sampleX, sampleY,
    sampleZ, num_ball):
    archivo = open("Results_n_%d" %num_ball + ".txt","w")
    archivo.write("num_ball," + "sigma_X," + "sigma_Y," + "sigma_Z," + "
        averageX," + "averageY," + "averageZ," + "\n")
    average[0] = average[0]/max_num_anchors
    average[1] = average[1]/max_num_anchors
    average[2] = average[2]/max_num_anchors

    text = "%d" % num_ball
    text = text + ",%s" %(statistics.stdev(sampleX))

```

```

        text = text + ",%s" % (statistics.stdev(sampleY))
        text = text + ",%s" % (statistics.stdev(sampleZ))
        text = text + ",%.5f" % average[0]
        text = text + ",%.5f" % average[1]
        text = text + ",%.5f" % average[2]
        archivo.write(text)
        archivo.close()

if __name__ == '__main__':

    rospy.init_node('System_Controller', anonymous=True)

    # Reading parameters
    factor_fast_pan = rospy.get_param('~factor_fast_pan')
    factor_fast_tilt = rospy.get_param('~factor_fast_tilt')
    factor_slow_pan = rospy.get_param('~factor_slow_pan')
    factor_slow_tilt = rospy.get_param('~factor_slow_tilt')
    servo_step = rospy.get_param('~servo_step')
    frequency = rospy.get_param('~frequency')
    P_pan_ini = rospy.get_param('~P_pan_ini')
    P_tilt_ini = rospy.get_param('~P_tilt_ini')
    max_num_anchors = rospy.get_param('~max_num_anchors')

    #default values
    P_pan = P_pan_ini
    P_tilt = P_tilt_ini
    P_pan_servo = P_pan_ini
    P_tilt_servo = P_tilt_ini
    buttons = [0,0,0,0,0,0,0,0,0,0,0,0]
    axes = [0,0,0,0,0,0,0]
    button_pressed = False
    closed_laser = True
    id_anchor = -1
    anchors_list = []
    num_anchors_assigned = -1
    num_ball = 1

    # Creating publishers
    pub_pan = rospy.Publisher('/arm_shoulders_pan_joint/command', Float64,
                              queue_size=10)
    pub_tilt = rospy.Publisher('/arm_shoulders_tilt_joint/command', Float64,
                               queue_size=10)

    rate = rospy.Rate(frequency) # 10hz

    # Starting to communicate with PTU
    tc = TowerController(buttons, axes, factor_fast_pan, factor_fast_tilt,
                        factor_slow_pan, factor_slow_tilt, servo_step, P_pan_ini, P_tilt_ini,
                        P_pan, P_tilt, button_pressed, P_pan_servo, P_tilt_servo)
    time.sleep(5)
    tc.setup()
    # Interruptions
    try:
        tc.joy_listener()
        tc.arbotix_listener()
    except rospy.ROSInterruptException:

```

```
pass

while not rospy.is_shutdown():
    tc.loop()
    rate.sleep()
```

A.2 Código del nodo implementado *Laser Drivers*

A.2.1 Archivo encargado de crear el nodo *Laser Drivers* de extensión .cpp

```
#include "laser_meter.h"

int main(int argc, char** argv) {

    ros::init (argc, argv, "laser_meter_node");
    ros::NodeHandle n("~");

    srv_drivers::LaserMeter lm(n);
    lm.configure();
    ros::spin();
    return 0;
}
```

A.2.2 Archivo .h

```
#ifndef LASER_METER_H_
#define LASER_METER_H_

#include <ros/ros.h>
// #include <ros/console.h>
#include <sensor_msgs/Range.h>
#include <std_msgs/Float64.h>

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include <boost/thread/mutex.hpp>

#include <boost/filesystem.hpp>
#include <boost/thread.hpp>
#include <boost/asio.hpp>

#include "srv_drivers/OpenLaser.h"
#include "srv_drivers/CloseLaser.h"
#include "srv_drivers/ReadLaser.h"
#include "srv_drivers/LaserConditions.h"

using namespace boost::asio;
using namespace boost::posix_time;
using namespace std;
```



```

namespace srv_drivers {

    class LaserMeter {

    public:

        LaserMeter(const ros::NodeHandle nh);
        virtual ~LaserMeter();
        void configure();

    private:

        ros::NodeHandle nodeHandle;

        //Params
        std::string pPort;
        double min_range, max_range;
        std::string frame_id;
        bool receive_inches;

        //Variables
        boost::asio::serial_port *port; // The serial port this
            instance is connected to

        // Subscribers and publishers
        //ros::Publisher range_publ_;
        ros::ServiceServer serviceOpen, serviceClose, serviceRead
            , serviceConditions;

        //Timers

        // Frames

        //Mutex
        bool open_laser(srv_drivers::OpenLaser::Request &req,
            srv_drivers::OpenLaser::Response &res);
        bool close_laser(srv_drivers::CloseLaser::Request &req,
            srv_drivers::CloseLaser::Response &res);
        bool read_laser(srv_drivers::ReadLaser::Request &req,
            srv_drivers::ReadLaser::Response &res);
        bool laser_conditions(srv_drivers::LaserConditions::
            Request &req, srv_drivers::LaserConditions::Response
            &res);
        bool connect(const std::string& portst);

    };

}

#endif /* LASER_METER_H_*/

```

A.2.3 Archivo .cpp

```
#include "laser_meter.h"
```

```
namespace srv_drivers {

    LaserMeter::LaserMeter(const ros::NodeHandle nh):
        nodeHandle(nh)
    {

    }

    LaserMeter::~LaserMeter() {
    }

    void LaserMeter::configure() {

        // Parameters

        nodeHandle.param("min_range", min_range, 0.3);
        ROS_INFO("Min_range:_%2.2f", min_range);

        nodeHandle.param("max_range", max_range, 10.0);
        ROS_INFO("Max_range:_%2.2f", max_range);

        nodeHandle.param<std::string>("frame_id", frame_id, "laser_meter"
        );
        ROS_INFO("Frame_id:_%s", frame_id.c_str());

        nodeHandle.param<std::string>("port", pPort, "/dev/ttyUSB1");
        ROS_INFO("Using_port:_%s", pPort.c_str());

        // Variables

        // Advertising Topics
        serviceOpen = nodeHandle.advertiseService("open_laser", &
            LaserMeter::open_laser, this);
        serviceClose = nodeHandle.advertiseService("close_laser", &
            LaserMeter::close_laser, this);
        serviceRead = nodeHandle.advertiseService("read_laser", &
            LaserMeter::read_laser, this);
        serviceConditions = nodeHandle.advertiseService("laser_conditions
            ", &LaserMeter::laser_conditions, this);

        // Subscribing Topics

        // Timers

        connect(pPort);

    }

    bool LaserMeter::connect(const std::string& portst){

        io_service io;
        const char *PORT = portst.c_str();
        serial_port_base::baud_rate baud_option(19200);

        // how big is each "packet" of data (default is 8 bits)
```

```

serial_port_base::character_size char_size( 8 );
// what flow control is used (default is none)
serial_port_base::flow_control flow_ctrl( serial_port_base::
    flow_control::none );
// what parity is used (default is none)
serial_port_base::parity parity( serial_port_base::parity::none )
;
// how many stop bits are used (default is one)
serial_port_base::stop_bits stop( serial_port_base::stop_bits::
    one );

try{

    port = new serial_port(io);
    port->open(PORT);
    port->set_option(baud_option); // set the baud rate
    port->set_option(char_size);
    port->set_option(flow_ctrl);
    port->set_option(parity);
    port->set_option(stop);
    cout << "(Laser_meter)_port_" << PORT << "_opened\n";

} catch (boost::system::system_error &e){

    boost::system::error_code ec = e.code();
    cerr << "(laser_meter)_cannot_open_port_" << PORT << "__"
        error_code:_" << ec.category().name() << std::endl;
    return false;

} catch(std::exception e){

    cerr << "(lasermeter)_cannot_open_port_" << PORT << "__"
        error_code:_" << e.what() << endl;
    return false;

}

return true;
}

bool LaserMeter::open_laser(srv_drivers::OpenLaser::Request &req,
    srv_drivers::OpenLaser::Response &res)
{
    unsigned char laser_command='O';
    unsigned char bu[1];

    boost::asio::write(*(port), boost::asio::buffer(&laser_command,
        1)); // write the "O" command for the serial port to open
        laser
    printf ("_n_LASER_ON\n");

    while((bu[0]) != '\n'){
        boost::asio::read(*(port), boost::asio::buffer(bu, 1));
        // read the response of the device from the serial
        port
        printf ("%c" , bu[0]);
    }
}

```

```
        return true;
    }

    bool LaserMeter::close_laser(srv_drivers::CloseLaser::Request &req,
                                srv_drivers::CloseLaser::Response &res)
    {
        unsigned char laser_command='C';
        unsigned char bu[1];

        boost::asio::write(*(port), boost::asio::buffer(&laser_command,
            1)); // write the "C" command for the serial port to close
            laser
        printf ("\n_LASER_OFF\n");

        while((bu[0]) != '\n'){
            boost::asio::read(*(port), boost::asio::buffer(bu, 1));
            // read the response of the device from the serial
            port
            printf ("%c" , bu[0]);
        }
        return true;
    }

    bool LaserMeter::read_laser(srv_drivers::ReadLaser::Request &req,
                                srv_drivers::ReadLaser::Response &res)
    {
        unsigned char laser_command='D';
        unsigned char bu[1];
        unsigned char buffer[6];
        int bufferCnt = 0;

        boost::asio::write(*(port), boost::asio::buffer(&laser_command,
            1)); // write the "D" command for the serial port to take
            measure
        printf ("\n_LASER_READ\n");

        while((bu[0]) != '\n'){
            boost::asio::read(*(port), boost::asio::buffer(bu, 1));
            // read the response of the device from the serial
            port

            if (bu[0] == ':'){
                bufferCnt = 0;

                while(bu[0] != 'm'){
                    boost::asio::read(*(port), boost::asio::
                        buffer(bu, 1)); // read character by
                        character all the measure taken
                    buffer[bufferCnt] = bu[0];
                    bufferCnt++;
                }

                if (buffer[0] == '_'){
                    buffer[0] = '0';
                }
                res.dist = ((int)buffer[0]-48)*10+((int)buffer
                    [1]-48)+((int)buffer[3]-48)*0.1+((int)buffer
```

A.2. Código del nodo implementado *Laser Drivers*

```
        [4]-48)*0.01+((int)buffer[5]-48)*0.001; //
        Makes the conversation of the read to take
        the distance in meters
    }
}
return true;
}
bool LaserMeter::laser_conditions(srv_drivers::LaserConditions::Request &
req, srv_drivers::LaserConditions::Response &res)
{
    unsigned char laser_command='S';
    unsigned char bu[1];

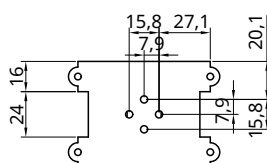
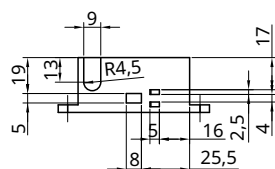
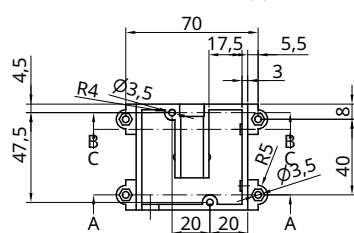
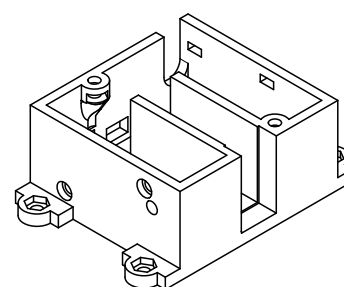
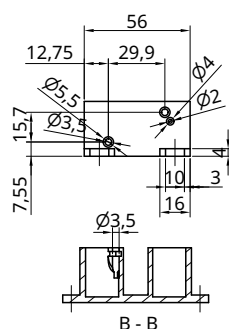
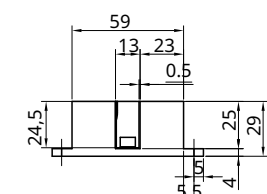
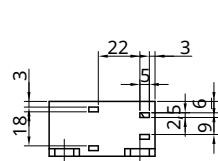
    boost::asio::write(*(port), boost::asio::buffer(&laser_command,
        1)); // write the "S" command for the serial port to know the
        device state.
    printf ("\n_LASER_CONDITIONS\n");



    while((bu[0]) != '\n'){
        boost::asio::read(*(port), boost::asio::buffer(bu, 1));
        printf ("%c" , bu[0]);
    }
    return true;
}
}
```



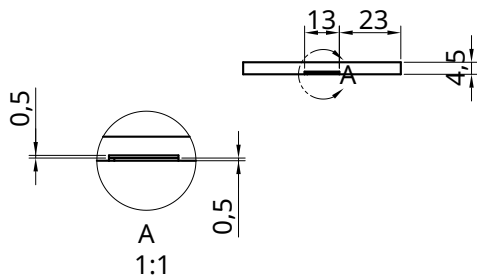
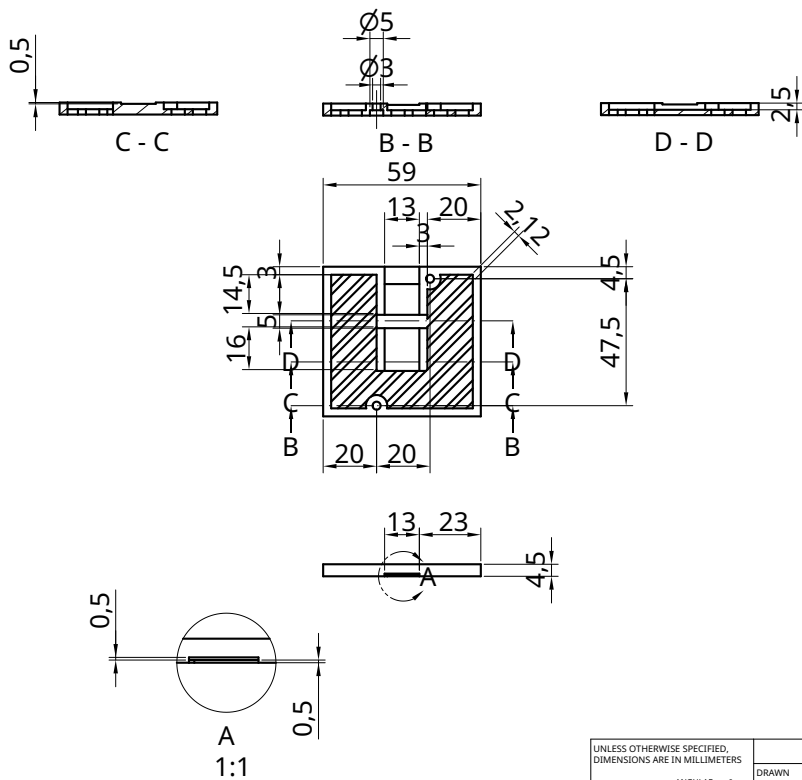

PLANOS DEL COMPARTIMENTO IMPLEMENTADO



B.1 Plano de la parte superior



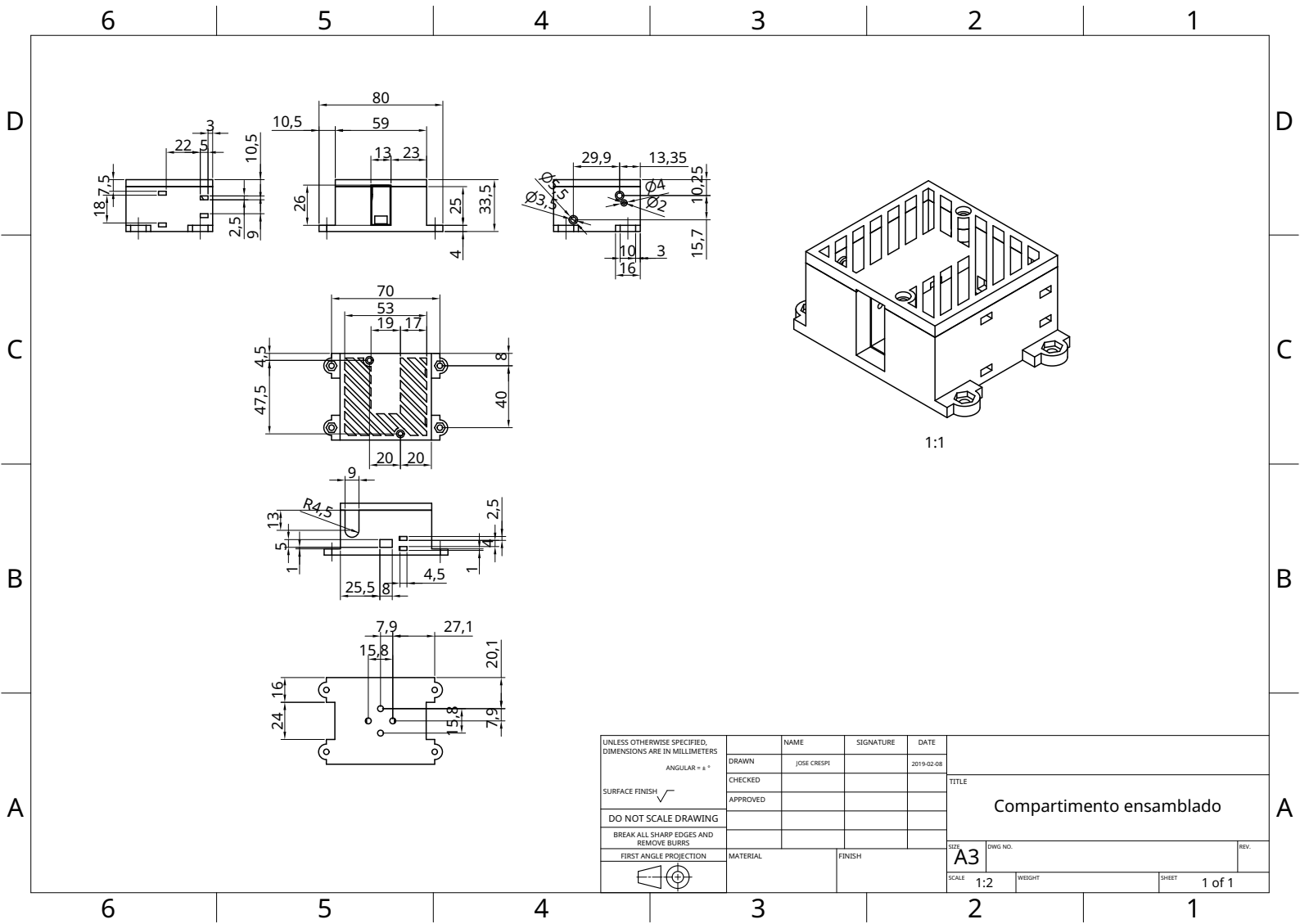
UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN MILLIMETERS ANGULAR = ± ° SURFACE FINISH  DO NOT SCALE DRAWING BREAK ALL SHARP EDGES AND REMOVE BURRS FIRST ANGLE PROJECTION 	NAME	SIGNATURE	DATE	TITLE <div>Compartimento</div>				
	DRAWN	JOSE CRESPI	2019-09-10					
	CHECKED							
	APPROVED							
MATERIAL	FINISH		SIZE		A3	DWG NO.	REV.	
		SCALE		1:2	WEIGHT		SHEET	1 of 2

B.2 Plano de la parte inferior



UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN MILLIMETERS ANGULAR = ± ° SURFACE FINISH  DO NOT SCALE DRAWING BREAK ALL SHARP EDGES AND REMOVE BURRS FIRST ANGLE PROJECTION 	DRAWN	JOSE CRESPI		2019-09-10	TITLE Tapa compartimento		
	CHECKED						
	APPROVED						
					SIZE A4	DWG NO.	REV.
	MATERIAL		FINISH		SCALE 1:2	WEIGHT	SHEET 2 of 2

B.3 Plano del compartimento ensamblado



BIBLIOGRAFÍA

- [1] G. C. O.-d.-G. A. F.-M. J. M. F. González J., Blanco J.L. and M. J.L., “Localización de vehículos combinando tecnología uwb y gps en entornos interiores y exteriores,” *Departamento de Ingeniería de Sistemas y Automática*.
- [2] J. y. A. González Jiménez y Ollero Baturone, *Estimación de la Posición de un Robot Móvil*. Dpto. de Ingeniería de Sistemas y Automática. Universidad de Málaga y Dpto. Ingeniería de Sistemas y Automática. Universidad de Sevilla, 2015.
- [3] D. O. Delgado, “¿qué es ros?” 2017. [Online]. Available: <https://openwebinars.net/blog/que-es-ros/>
- [4] —, “Conoce onshape el nuevo programa de diseño mecánico 3d en la nube,” 2015. [Online]. Available: <http://www.3dcadportal.com/conoce-onshape-el-nuevo-programa-de-diseno-mecanico-3d-en-la-nube.html>