



**EDUCACIÓN**  
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO  
NACIONAL DE MÉXICO



# TECNOLÓGICO NACIONAL DE MÉXICO campus LEÓN

## INFORME TÉCNICO DE RESIDENCIA PROFESIONAL

### TÍTULO DEL TRABAJO

DISEÑO DE UN TRAJE DE CAPTURA DE MOVIMIENTO PARA TERAPIA

### QUE PRESENTA:

IVÁN ALEJANDRO MUÑOZ VERA

Vo:Bo

### CARRERA:

INGENIERÍA EN MECATRÓNICA

### CON LA ASESORÍA DE:

M.C. MIGUEL ÁNGEL CASILLAS ARAIZA

León, Guanajuato, Fecha: Julio / 2021



**EDUCACIÓN**  
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO  
NACIONAL DE MÉXICO

Instituto Tecnológico de León  
Departamento de Metal Mecánica

**"2021: Año de la Independencia"**

León, Gto., 12/Abril/2021  
Ficha Única de Asignación  
OFICIO No. MCX-019 -2021  
PERIODO: MARZO - JULIO 2021

**IVÁN ALEJANDRO MUÑOZ VERA**  
**No. de control : 15240556**  
**PRESENTE**

Por este conducto, me permito comunicarle a usted, que ha sido asignado(a) a la empresa:  
Tecnologico nacional de México/Instituto tecnologico de León

Para la realización de su residencia profesional, en el proyecto:

Diseño de un traje de captura de movimiento para terapia

cuyo asesor docente es: M.C. Miguel Ángel Casillas Araiza  
y cuyo asesor externo se tiene al: Dra. María del Rosario Baltazar Flores

Sin más por el momento, quedo de usted.

**Atentamente**

Excelencia en Educación Tecnológica  
Ciencia Tecnología y Libertad

**MC. Bulmaro Aranda Cervantes**  
**Jefe del Departamento**



BAC/merm  
C.c. p. División de Estudios Profesionales  
Depto. De Gestión Tecnológica y Vinculación  
Asesor Académico  
Asesor Externo  
Archivo



DIN EN 9100  
2017-04-30 - 2021-04-30

Av. Tecnológico s/n Fracc. Industrial  
Julián de Obregón C.P 37290  
León, Gto. México Tel. 01 (477) 7105200,  
e-mail: tecleon@leon.tecnm.mx  
tecnm.mx | leon.tecnm.mx



# Agradecimientos y/o Dedicatorias

## **A mi papá:**

Por enseñarme, con su ejemplo, la importancia del trabajo, compromiso y dedicación en todo lo que me proponga. Su enseñanza forjó mi carácter y deseo de superación personal.

## **A mi mamá:**

Por su cariño, comprensión y apoyo en todas las decisiones que he tomado en mi vida. Representa un gran pilar en todo lo que he logrado y lograré el día de mañana.

## **A mis hermanos:**

Por estar ahí en todos los buenos y malos momentos de mi vida. Son con quienes aprendí a ser mejor persona.

## **A muchos de mis profesores:**

Por su dedicación hacia la docencia, por la sabiduría de su enseñanza y por inspirar en mí el deseo de aprender.

## **A mis asesores de residencias profesionales:**

Por permitirme trabajar en este increíble proyecto, por su accesibilidad, experiencia, comprensión, guía y apoyo.

## **Al Mtro. Gustavo Terán Moreno:**

Por su gran trato hacia mi persona. Su forma de ser inspiró buenos cambios en mi persona. Siempre vivirá en mi corazón. En paz descanse.

# Resumen

El presente informe técnico de residencia profesional comienza con el capítulo de **Generalidades del proyecto**, el cual contiene información que vincula este proyecto con la empresa en donde se desarrolla la residencia. De forma tal que en este capítulo se da una breve descripción de la empresa, el planteamiento del problema encontrado, y la solución propuesta con el presente proyecto.

En el **Capítulo 1**, o Capítulo de Marco Teórico, se presentan y explican los conceptos más importantes para que el lector pueda tener una mejor comprensión de todo el contenido de este informe técnico de residencias. Se abordan aspectos matemáticos, mecánicos, electrónicos y computacionales afines al proyecto, también se da cobertura a las herramientas de hardware y software que se emplearon en el proyecto.

En el **Capítulo 2**, o Capítulo de Desarrollo, se mencionan los métodos, esquemas, modelos, algoritmos, hardware, software, archivos y códigos empleados en el presente proyecto. Este capítulo está estructurado en un orden continuo y progresivo en el tiempo para cada sección y subsección del mismo, lo cual coincide con el orden en que los dispositivos y herramientas fueron utilizadas: el capítulo comienza con lo desarrollado en Arduino; después, Matlab; luego, Unreal; y finalmente, Raspberry Pi.

En el **Capítulo 3**, o Capítulo de Resultados, se presentan los resultados parciales y finales de cada uno de los temas, subtemas y subsubtemas del Capítulo 2. Contiene, además, ligas a repositorios web y a plataformas de almacenamiento y reproducción de vídeos por internet. Este capítulo se encuentra estructurado de forma similar al Capítulo 2. En este capítulo se muestran figuras que fueron fruto de poner en práctica todo lo visto en el Capítulo 2.

En el **Capítulo de conclusiones y trabajo a futuro** se mencionan mis aprendizajes técnicos y personales que fueron adquiridos como fruto de la realización de la residencia profesional, se mencionan algunas de las recomendaciones que su servidor considera

necesarias para continuar con el presente proyecto, y por último, se mencionan los problemas que se presentaron y su solución si es que la hubo.

En la **Sección de Competencias Aplicadas** se establece un vínculo entre los 8 puntos del perfil de egreso del estudiante en Ingeniería mecatrónica del TecNM campus León y los objetivos específicos del presente proyecto.

# Contenido General

<b>Agradecimientos y/o Dedicatorias</b>	<b>III</b>
<b>Resumen</b>	<b>IV</b>
<b>Contenido General</b>	<b>VI</b>
<b>Contenido de Figuras</b>	<b>XIII</b>
<b>Contenido de Tablas</b>	<b>XVIII</b>
<b>1 Marco Teórico</b>	<b>1</b>
1.1 Estado del Arte . . . . .	1
1.1.1 Mining Spatial-Temporal Patterns and Structural Sparsity for Human Motion Data Denoising . . . . .	1
1.1.2 Experimental Comparison of Sensor Fusion Algorithms for Attitude Estimation . . . . .	1
1.1.3 Evaluación de algoritmos de fusión de datos para estimación de la orientación de vehículos aéreos no tripulados . . . . .	3
1.1.4 Wireless Motion Capture System for Upper Limb Rehabilitation . . .	4
1.1.5 Validation of a Kinect V2 based rehabilitation game . . . . .	5
1.2 Estado de la Técnica . . . . .	6
1.2.1 A treatment device for hemiplegia . . . . .	6
1.2.2 Systems, apparatus and methods for non-invasive motion tracking to augment patient administered physical rehabilitation . . . . .	7

1.2.3	System and method for identifying and interpreting repetitive motions	8
1.2.4	Rehabilitation and exercise machine	9
1.2.5	Wireless game controller for strength training and physiotherapy	9
1.3	Fundamentos teóricos empleados en el proyecto	10
1.3.1	Sensores MARG	10
	Acelerómetro	10
	Magnetómetro	11
	Giróscopo	13
1.3.2	Herramientas matemáticas para la localización espacial	13
	Representación de la posición	14
	Representación de la orientación	14
	Matrices de transformación homogénea	18
	Composición de matrices homogéneas	19
1.3.3	Conversión directa e inversa entre algunas herramientas matemáticas de localización espacial	20
	Relación entre ángulos de Euler XYZ y matriz de transformación homogénea	20
	Relación entre cuaternión y matriz de transformación homogénea	21
1.3.4	Descenso del gradiente	22
1.3.5	Sistemas de Referencia de Actitud y Rumbo	24
	Filtro Madgwick	24
1.3.6	Comunicaciones seriales	26
	I2C	26
	SPI	26
1.3.7	Modelo OSI	27
	Modelo TCP/IP	27
	WIFI	29
	Internet Protocol (IP)	30

UDP . . . . .	31
Internet . . . . .	31
1.3.8 Microcontroladores y microcomputadores . . . . .	32
Arduino . . . . .	32
Raspberry Pi 3B . . . . .	32
1.3.9 Entornos de desarrollo integrados . . . . .	33
Arduino . . . . .	33
Matlab . . . . .	33
Microsoft Visual Studio . . . . .	33
Unreal Engine . . . . .	34
Qt Creator . . . . .	34
1.3.10 Motion Capture . . . . .	35
<b>2 Desarrollo</b>	<b>36</b>
2.1 Descripción del proyecto . . . . .	36
2.1.1 Propuesta de valor . . . . .	37
2.2 Cronograma de actividades programadas . . . . .	37
2.3 Proceso de investigación inicial . . . . .	38
2.3.1 Investigación de mercado del Motion Capture . . . . .	38
Análisis de la oferta . . . . .	38
2.3.2 Variables físicas útiles que debe medir un sensor para poder obtener la orientación y la posición . . . . .	39
2.3.3 Principales ofertas de sensores MARG en el mundo . . . . .	40
2.3.4 Selección de los 2 sensores más aptos para el proyecto . . . . .	41
Cantidad de ejes/GDL's necesarios para calcular la orientación de un sensor MPU o MARG . . . . .	41
Compra por internet de los 2 sensores MARG más adecuados para testeo. . . . .	41



2.4	Pruebas de comunicación y funcionamiento: Arduino - sensores MARG . . .	44
2.4.1	Sensor MPU9250: consideraciones iniciales . . . . .	44
2.4.2	Conexiones eléctricas entre Arduino y el sensor MPU9250 para em- plear el protocolo I2C . . . . .	45
2.4.3	Primeras pruebas Arduino - MPU9250 . . . . .	46
2.4.4	Pruebas de comportamiento de los sensores MARG a las perturba- ciones de entrada . . . . .	47
	Procedimiento para la prueba Self Test del sensor MPU9250 . . .	48
2.4.5	Código para resetear y setear el sensor MPU9250 cuando se bloquee	48
2.5	Pruebas de comunicación y funcionamiento: Matlab - Arduino - Sensores MARG . . . . .	49
2.5.1	Comunicación USB entre Matlab y Arduino . . . . .	49
2.5.2	Graficación de vectores 3D en Matlab con los datos enviados por Arduino . . . . .	50
	Algoritmos de calibración automática bajo condiciones reales y simuladas en Matlab . . . . .	50
2.5.3	Filtro Madgwick . . . . .	52
	Macros en el código del sensor MPU9250 en Arduino para esco- ger un protocolo de comunicación (I2C o SPI) . . . . .	53
2.5.4	Selección de un modelo de sensor MARG para que este sea em- pleado en todo el traje . . . . .	53
2.6	Pruebas de comunicación y funcionamiento: Unreal - Matlab - Arduino - Mpu9250 . . . . .	56
2.6.1	Consideraciones iniciales para Unreal Engine . . . . .	56
2.6.2	Plugins Unreal Editor . . . . .	58
2.6.3	Lenguajes de programación para Unreal Engine . . . . .	59
2.6.4	Microsoft Visual Studio . . . . .	60
2.6.5	Protocolos de comunicación entre Unreal y Matlab . . . . .	61

2.6.6	Comunicación entre Unreal y Matlab . . . . .	62
2.7	Pruebas de comunicación y funcionamiento: Unreal - Raspberry Pi - Mpu9250	65
2.7.1	Conexión remota entre un ordenador y la Raspberry Pi 3B . . . . .	65
2.7.2	Qt Creator . . . . .	67
2.7.3	Análisis en Github y Gitlab de códigos de terceros . . . . .	68
2.8	Código generado con Qt Creator en la Raspberry Pi 3B . . . . .	70
2.8.1	Qt files . . . . .	71
	Mocap.pro: . . . . .	71
	mainwindow.ui: . . . . .	71
2.8.2	Header files . . . . .	71
	RegistersAndMask.h: . . . . .	71
	Complementos.h: . . . . .	72
	arduino.h: . . . . .	72
	SPI.h: . . . . .	72
	mainwindow.h: . . . . .	73
	matlab.h: . . . . .	73
	mpu9250.h: . . . . .	73
	mocapsuit.h: . . . . .	73
2.8.3	Source files . . . . .	74
	Arduino.cpp: . . . . .	74
	mainwindow.cpp: . . . . .	74
	matlab.cpp: . . . . .	74
	mpu9250.cpp: . . . . .	75
	mocapsuit.cpp: . . . . .	75
	main.cpp: . . . . .	75
2.9	Unreal . . . . .	76
2.9.1	Blueprint: ThirdPersonCharacter . . . . .	77
	Condición de alineación y arranque de la captura de movimiento .	77

Datagramas UDP (obtención de las orientaciones mocap) . . . . .	77
2.9.2 Blueprint: ThirdPersonExampleMap . . . . .	80
2.9.3 Blueprint: Animation . . . . .	80
Event Graph . . . . .	80
Anim Graph . . . . .	81
2.10 Traje de captura de movimiento . . . . .	82
2.10.1 Selección del tipo de cable para conectar los componentes del traje	83
2.10.2 Conexiones eléctricas/electrónicas del traje . . . . .	84
2.10.3 Vinculación entre el traje mocap y un personaje en Unreal . . . . .	86
2.10.4 Alineación del traje mocap (sensores MPU9250 y RPi) con el perso- naje virtual en Unreal . . . . .	86
<b>3 Resultados</b>	<b>93</b>
3.1 Arduino - MPU9250 . . . . .	93
3.1.1 Primera prueba de comunicación y funcionamiento . . . . .	93
3.1.2 Escáner de registros . . . . .	94
3.1.3 Código final . . . . .	96
3.2 Matlab - Arduino - MPU9250 . . . . .	96
3.2.1 Vector 3D: Acelerómetro y magnetómetro . . . . .	96
3.2.2 Calibración automática . . . . .	98
3.2.3 Filtro Madgwick . . . . .	98
3.3 Unreal - Matlab - Arduino - MPU9250 . . . . .	102
3.4 Traje Mocap . . . . .	104
3.5 Resultados finales . . . . .	106
Referencias . . . . .	119
<b>Glosario de Términos</b>	<b>122</b>
<b>Acrónimos</b>	<b>123</b>

<b>Anexos</b>	<b>123</b>
<b>A Hojas de datos: MPU-9250</b>	<b>125</b>
A.1 MPU-9250 Product Specification . . . . .	125
A.2 MPU-9250 Register Map and Descriptions . . . . .	136
A.3 AK8963 datasheet . . . . .	140
A.4 MPU-9250 Accelerometer, Gyroscope and Compass Self-Test Implemen- tation . . . . .	147
A.5 MPU Hardware Offset Registers: Application note . . . . .	148
<b>B Qt Project: Raspberry Pi 3B</b>	<b>149</b>
B.1 main.cpp . . . . .	149
B.2 RegistersAndMask.h . . . . .	150
B.3 arduino.h . . . . .	151
B.4 Complementos.h . . . . .	151
B.5 arduino.cpp . . . . .	152
B.6 mainwindow.cpp . . . . .	154
B.7 mainwindow.h . . . . .	154
B.8 mainwindow.ui . . . . .	155
B.9 matlab.cpp . . . . .	156
B.10 matlab.h . . . . .	163
B.11 Mocap.pro . . . . .	165
B.12 mocapsuit.cpp . . . . .	165
B.13 mocapsuit.h . . . . .	167
B.14 mpu9250.cpp . . . . .	167
B.15 mpu9250.h . . . . .	179
B.16 SPI.h . . . . .	181

# Contenido de Figuras

Figura 1.1	Esquema de funcionamiento del modelo propuesto. Recuperado de: (Feng y cols., 2015).	2
Figura 1.2	Errores en los ángulos vs frecuencia de muestreo. Recuperado de: (Cristóbal y Humberto, 2017).	3
Figura 1.3	Representación de la extremidad superior como un brazo robótico. Recuperado de: (Tsilomitrou, Gkountas, Evangeliou, y Dermatas, 2021).	5
Figura 1.4	Mystic Isle. Recuperado de: (Ma, Proffitt, y Skubic, 2018).	6
Figura 1.5	Vista en perspectiva que muestra un estado en el que un sujeto usa un aparato de tratamiento hemiparalítico. Recuperado de: (Gwana-ro, 2013).	6
Figura 1.6	Es una vista en perspectiva de la configuración de un sistema de se- guimiento de movimiento terapéutico. Recuperado de: (Komatireddy, Hut- chins, y Shah, 2012).	7
Figura 1.7	Es una vista en perspectiva de un sistema de seguimiento de movi- miento. Recuperado de: (Canavan y Hughes, 2020).	8
Figura 1.8	Vista isométrica de una máquina de ejercicios y rehabilitación mejo- rada construida de acuerdo con los principios de esta invención. Recupe- rado de: (Canavan y Hughes, 2020).	9
Figura 1.9	Lucha libre en línea habilitada por esta invención. Recuperado de: (Canavan y Hughes, 2020).	10
Figura 1.10	Acelerómetro capacitivo. Recuperado de: (Ramírez, Jiménez, y Ca- rreño, 2014).	11

Figura 1.11 Representación del efecto Hall. Recuperado de: (Ramírez y cols., 2014).	12
Figura 1.12 Principios de funcionamiento de un giróscopo de estado sólido Recuperado de: (Ramírez y cols., 2014).	13
Figura 1.13 Sistemas de coordenadas. Recuperado de: (Barrientos, Peñin, y Carlos Balaguer, 2007).	14
Figura 1.14 Ejemplo de rotaciones básicas. Recuperado de: (Barrientos y cols., 2007).	15
Figura 1.15 Mismo vector <b>P</b> referido en 2 sistemas coordenados distintos. Recuperado de: (Barrientos y cols., 2007).	16
Figura 1.16 Ángulos de Euler XYZ (Yaw, Pitch y Roll). Recuperado de: (Barrientos y cols., 2007).	17
Figura 1.17 Representación gráfica de la rotación que ocurre al emplear cuaterniones. Recuperado de: (Barrientos y cols., 2007).	18
Figura 1.18 Aplicación de diversas transformaciones para localizar (posición y orientación) un objeto. Recuperado de: (Barrientos y cols., 2007).	20
Figura 1.19 Un ejemplo de búsqueda de gradiente para un punto estacionario. Recuperado de: (Singer, 2016).	24
Figura 1.20 Diagrama del algoritmo Madgwick para sensores MARG. Recuperado de: (Madgwick, 2010).	25
Figura 1.21 Diagrama de conexión SPI entre 1 maestro y 2 esclavos. Recuperado de: (TDK InvenSense, 2016).	27
Figura 1.22 Niveles o capas del modelo OSI. Recuperado de: (Tanenbaum y Wetherall, 2012).	28
Figura 1.23 Modelo TCP/IP	29
Figura 1.24 Encabezado de IPv4. Recuperado de: (Tanenbaum y Wetherall, 2012).	30
Figura 1.25 Encabezado de UDP. Recuperado de: (Tanenbaum y Wetherall, 2012).	31
Figura 1.26 Placas electrónicas empleadas en pruebas y desarrollo.	32

Figura 2.1	Esquema de funcionamiento . . . . .	37
Figura 2.2	Sensores MARG puestos a prueba durante el proyecto. . . . .	43
Figura 2.3	Esquemático de la placa breakout GY-9250, empleada en el proyecto. Recuperado de: (ProtoSupplies, s.f.). . . . .	45
Figura 2.4	Conexiones eléctricas entre Arduino y el sensor MPU9250 para emplear el protocolo I2C . . . . .	46
Figura 2.5	Conexión SPI entre Arduino y el sensor MPU9250 . . . . .	54
Figura 2.6	Esquemático de conexiones entre Arduino y el sensor MPU9250 para el protocolo SPI . . . . .	55
Figura 2.7	Versión del Unreal Editor empleado para el presente proyecto . . .	56
Figura 2.8	Interfaz principal del Unreal Editor 4.26. Se pueden apreciar contenidos, objetos, detalles, etc., empleados en el presente proyecto . . . . .	57
Figura 2.9	Content Browser: Carpetas empleadas para el presente proyecto. Dentro de las carpetas se encuentran las animaciones, materiales, texturas, blueprints, etc., necesarios para el desarrollo del videojuego . . . . .	58
Figura 2.10	Interfaz principal de Microsoft Visual Studio . . . . .	61
Figura 2.12	Lista de componentes que pertenecen al ThirdPersonCharacter . . .	62
Figura 2.13	Panel de configuración del componente UDP . . . . .	63
Figura 2.11	Explorador de archivos del proyecto de Unreal en Microsoft Visual .	64
Figura 2.14	Programa Advanced IP Scanner empleado para buscar direcciones IP	65
Figura 2.15	Programa Vnc Viewer empleado para establecer un escritorio remoto con la Rpi . . . . .	66
Figura 2.16	Programa PuTTY empleado para establecer una sesión/conexión SSH con la Raspberry Pi . . . . .	67
Figura 2.17	Versión del IDE Qt Creator . . . . .	68
Figura 2.18	Mensaje obtenido luego de activar satisfactoriamente la interfaz SPI en la RPi 3B . . . . .	70
Figura 2.19	GUI resultante del proyecto en Qt . . . . .	74

Figura 2.20 Blueprints empleados para las funcionalidades del ThirdPersonCharacter (personaje virtual con bordes anaranjados en la Figura 2.8) . . . . .	78
Figura 2.21 Blueprints necesarios para mandar la trama UDP que comenzará la alineación del traje en la Rpi . . . . .	79
Figura 2.22 Blueprints necesarios para leer las tramas UDP enviadas por la Rpi .	79
Figura 2.23 Blueprints empleados para la interacción de los objetos dentro del ThirdPersonExampleMap . . . . .	81
Figura 2.24 Blueprints necesarios para poder animar al personaje (correr, saltar, etc.) . . . . .	82
Figura 2.25 Blueprints para asignar la orientación al personaje ThirdPersonCharacter . . . . .	82
Figura 2.26 Conexiones SPI entre la Raspberry Pi 3B V1.2 y los sensores del traje	85
Figura 2.27 Ubicación de los 17 sensores necesarios para el desarrollo del proyecto . . . . .	87
Figura 2.28 Esqueleto virtual del personaje en Unreal Engine . . . . .	88
Figura 2.29 Assets agregados para usarse en ThirdPersonCharacter: m_T-Pose y NewT-PoseAsset . . . . .	89
Figura 2.30 Comparación entre la pose que por defecto viene con la plantilla del videojuego en tercera persona, y la pose requerida para el desarrollo del traje mocap . . . . .	90
Figura 3.1 Impresión en el monitor serial de Arduino los valores de 6 de los 9 ejes del sensor MPU9250. . . . .	94
Figura 3.2 Impresión en el monitor serial de todos los registros y los valores contenidos en esos registros para el sensor MPU9250 . . . . .	95
Figura 3.3 Impresión de las mediciones del sensor MPU-9250 a través del monitor serial de Arduino . . . . .	96



Figura 3.4 Gráfica de los vectores acelerómetro y magnetómetro a través de Matlab, al colocar el sensor en 3 orientaciones diferentes . . . . .	97
Figura 3.5 Parámetros de calibración obtenidos por regresión lineal. Impresión a través del comand window de Matlab . . . . .	98
Figura 3.6 Plot generado en Matlab de la orientación estimada por el filtro Madgwick . . . . .	99
Figura 3.7 Plot de diversas orientaciones del sensor MPU9250 . . . . .	100
Figura 3.8 Comparación entre la orientación real del sensor MPU9250 y la orientación estimada por el filtro de Madgwick . . . . .	101
Figura 3.9 Comparación entre la orientación del sensor MPU9250 y la orientación transmitida por Matlab mediante UDP a Unreal. . . . .	103
Figura 3.10 Avances del traje Mocap . . . . .	105
Figura 3.11 Alineación entre el portador del traje mocap y el personaje de Unreal Engine . . . . .	107
Figura 3.12 Captura del movimiento en Unreal cuando el portador del traje se inclina hacia el frente. . . . .	108
Figura 3.13 Captura del movimiento en Unreal cuando el portador del traje se inclina hacia atrás. . . . .	109
Figura 3.14 Captura del movimiento en Unreal cuando el portador del traje se gira hacia la izquierda. . . . .	110
Figura 3.15 Captura del movimiento en Unreal cuando el portador del traje gira hacia la derecha. . . . .	111

# Contenido de Tablas

Tabla 1.1	Error cuadrático medio en los ángulos roll, pitch y yaw. Extraído de: (Cavallo y cols., 2014).	2
Tabla 1.2	Consumo de recursos por algoritmo. Extraído de: (Cavallo y cols., 2014).	2
Tabla 1.3	Ciclos para converger y tiempo de ejecución. Recuperado de: (Cristóbal y Humberto, 2017).	4
Tabla 2.1	Fabricantes y sus productos de captura de movimiento.	38
Tabla 2.3	Comparación de características por cantidad de GDL's presentes. Recuperado de: (Freescale Semiconductor, 2015).	41
Tabla 2.2	Comparación de las especificaciones técnicas de diversos modelos de sensores MARG	42
Tabla 2.4	Hojas de datos empleadas para el uso del sensor MPU9250	47
Tabla 2.5	Registros necesarios para resetear el sensor MPU9250.	48
Tabla 2.6	Registros necesarios para setear el sensor MPU9250.	49
Tabla 2.7	Plugins de Unreal útiles para el proyecto	59
Tabla 2.8	Parámetros para inicializar cada sensor MPU-9250	76

# Generalidades del Proyecto

## Introducción

La terapia física se lleva a cabo con el fin de recuperar las capacidades motrices del paciente. Un traje de captura de movimiento puede ser útil en el diagnóstico y tratamiento de alguna discapacidad motriz, ya que permite capturar y procesar información a muy alta velocidad.

En esta residencia profesional se diseñó un traje de captura de movimiento con sensores MPU-9250, los cuales constan de 9 ejes (3 por el acelerómetro, 3 por el giroscopio y 3 por el magnetómetro). La obtención de la orientación fue posible gracias al filtro de Madgwick y a un algoritmo de calibración automática. Los datos de los sensores son capturados y pre-procesados por una Raspberry Pi 3B V1.2; y posteriormente, enviados como datagramas UDP mediante Wi-Fi hacia un ordenador que se encuentra ejecutando un videojuego o simulación en Unreal Engine.

## **Descripción de la empresa**

El Instituto Tecnológico de León se basa para la realización de sus actividades en la Constitución y lineamientos políticos del Gobierno Federal, principalmente de los planes y programas emanados de la Secretaría de Educación Pública. Así como también en el propósito de contribuir al desarrollo y progreso de la sociedad.

El Instituto Tecnológico de León depende en forma directa del TECNOLÓGICO NACIONAL DE MÉXICO, organismo desconcentrado dependiente de la Secretaría de Educación Pública, bajo este marco institucional, el objetivo general del Tecnológico de León es “lograr la vinculación real y efectiva del Instituto en los aspectos social, político, económico y productivo de la región”.

## **Antecedentes Históricos de la Empresa**

El Instituto Tecnológico de León, inició sus actividades el 18 de septiembre de 1972, en ese entonces era el I.T.R.L. (Instituto Tecnológico Regional de la ciudad de León No. 24). El Tecnológico inició sus labores con una población escolar de 518 alumnos, distribuidos de la siguiente forma: 233 del plan anual y 285 del plan semestral.

Por recomendación de la Asociación Nacional de Universidades e Instituciones de Educación Superior (ANUIES) y por disposición de la Secretaría de Educación Pública (SEP), en el año de 1985 se inició el proceso de segregación del bachillerato, entrando en él todas las carreras a nivel técnico que existían en el tecnológico en sus modalidades de escolarizada y abierta, por lo que, a partir del año 1986, se cerró definitivamente este nivel educativo en nuestro instituto. A partir de este año, también se cambia la denominación a solo Instituto Tecnológico de León.

En el año de 1996, el Instituto fue afiliado como socio fundador de la Institución Guanajuato para la Calidad y, en 1997 fue designado ganador del Premio Guanajuato a la Calidad, en nivel superior del sector educación, en el año 2006 obtuvo el Premio Guanajuato 2000 y el Reconocimiento a la Calidad SEP 2006, en el año 2007 obtuvo la Constancia de Inscripción por parte de CONACYT como Centro de Investigación; en Diciembre del 2008 conquistó el reconocimiento SEP a la Calidad por tener más del 75 por ciento de su matrícula inscritos en programas de calidad.

## **Misión**

Formar profesionales íntegros que promuevan la cultura, los valores humanos y el conocimiento científico, que se orienten a un crecimiento constante y trascendente manteniendo su arraigo con la sociedad mediante una conducta de servicio que permita proporcionar calidad de vida a su comunidad.

## **Visión**

En el año 2025 el Instituto Tecnológico de León comprometido con su proyecto de alta calidad e innovación. Congruente con su naturaleza académica y pertinente con relación a las necesidades del país, se consolida en sus procesos educativos, con el reconocimiento público en la búsqueda de la equidad, el humanismo y compromiso con la construcción de una sociedad mejor.

## **Valores**

- Compromiso
- Calidad
- Equidad
- Humanismo
- Congruencia

## **Ubicación**

Av. Tecnológico S/N – Fracc. Industrial Julián de Obregón

León, Guanajuato – C.P. 37290

Teléfono (477) 710 5200 – Fax (477) 711 2072

e-mail: [tecleon@leon.tecnm.mx](mailto:tecleon@leon.tecnm.mx)

## **Giro**

Educación pública superior

## **Caracterización del área de desarrollo del proyecto**

Las áreas en las que se emplean trajes de captura de movimiento pueden ser muy diversas (p. ej., en ciencias de la salud, ciencias del deporte y en entretenimiento). En ciencias de la salud se emplea para terapia, rehabilitación, biomecánica y ergonomía; en cuanto a los usos para entretenimiento están cinematografía, animación, videojuegos y publicidad.

## **Planteamiento del problema**

Actualmente, existen en el mercado costosos equipos capaces de ofrecer terapias psicomotrices, y en ocasiones consisten en máquinas muy robustas, las cuales son costosas de adquirir y mantener, estas requieren, en muchos casos, personal especializado para operarlas.

En cuanto a los trajes de captura de movimiento, desafortunadamente, los proveedores de estos los terminan vendiendo por precios muy por encima de lo que la mayoría de la gente podría estar dispuesta a pagar para usarlo en diagnóstico y tratamiento del paciente con discapacidad motriz (el precio de un traje mocap, en la mayoría de los casos, está por encima de los \$2000.00 dólares), consiguiendo así que estos sean un producto que solo algunas compañías pueden adquirir.

## **Objetivos**

### **Objetivo General**

Desarrollar un prototipo de traje para la captura del movimiento, de forma responsable y acorde a las normas nacionales e internacionales que apliquen, para que sea empleado en terapia psicomotriz posterior a su retroalimentación técnica y terapéutica.

## **Objetivos Específicos**

- Establecer las actividades y subactividades necesarias para planificar, administrar y evaluar el correcto desarrollo del proyecto.
- Recopilar información científica, técnica y comercial de actualidad para su posterior asimilación, adaptación e innovación hacia el proyecto.
- Diseñar el sistema, proceso y equipo tanto mecánico, eléctrico, electrónico y computacional del traje de captura de movimiento (MoCap) para que sea acorde a las necesidades latentes, tanto tecnológicas y sociales, de las terapias hacia las personas con discapacidad en la actualidad.
- Desarrollar un traje para capturar el movimiento empleando los conocimientos de ing. mecatrónica (mecánica, eléctrica, electrónica y sist. computacionales), el cual será ergonómico, de bajo costo y esté apegado, de forma responsable, a las normas nacionales e internacionales que apliquen.
- Vincular el presente proyecto con personas que tienen alguna(s) discapacidad(es) para ajustar la calidad, eficiencia y rentabilidad, y se siga desarrollando en momentos posteriores a la finalización de este proyecto.

## **Justificación del Proyecto**

El presente proyecto tiene como finalidad sentar las bases para futuras investigaciones y desarrollos tecnológicos, tanto de investigadores, profesores y estudiantes del Instituto Tecnológico de León, el proyecto está diseñado especialmente para fines de la División de Estudios de Posgrado e Investigación del ITL. Esta tecnología puede auxiliar a investigadores para realizar diagnóstico, terapia y rehabilitación física sobre pacientes con alguna(s) capacidad(es) diferente(s).



Al monitorear al paciente mediante un equipo computacional es posible observar patrones que ocurren a altas velocidades (muchas plataformas hardware como la Raspberry Pi trabajan con velocidades de procesamiento superiores a 1GHz). Además, en un futuro el traje se podría emplear para ofrecer terapias interactivas, ya que se emplea Unreal Engine (uno de los motores de videojuegos más avanzados del mundo).

## **Alcances y limitaciones**

### **Alcances**

- Se diseñará un traje mocap funcional con capacidad para transmitir los datos de orientación y posición relativa de forma inalámbrica y en tiempo real hacia una computadora y/o celular. Posiblemente, se podría desarrollar un videojuego que sea controlado por el traje; el videojuego podría ir de la mano de un visor de realidad virtual.
- Se desarrollará un algoritmo de calibración automática para corregir mediante software el offset-bias y el factor de escala de sensibilidad.
- Se desarrollará un algoritmo para alinear el cuerpo humano y los sensores MARG de forma automática.
- El programa/aplicación que se ejecutará en la Rpi, lo hará bajo las restricciones del S.O. Raspberry Pi OS (anteriormente Rasbian).
- La exactitud e inmunidad al ruido de los cálculos para obtener la orientación dependerán en gran medida de las especificaciones técnicas del sensor MPU9250.
- El proyecto será diseñado de acuerdo a las necesidades de los pacientes con algún daño cerebral, de forma que les sirva para su terapia.
- El presente proyecto estará centrado en todos los aspectos funcionales del traje mocap, el aspecto estético no es prioridad.

- El material y equipo que se habrán de emplear para el traje mocap deberá ser económico, pero sin descuidar la calidad.

## **Limitaciones**

- Las prestaciones para procesamiento computacional del proyecto (microprocesador, memoria RAM, GPU, etc.) no podrán ser superiores a las especificaciones técnicas de la Raspberry Pi
- No está contemplado el diseño electrónico y la producción de las placas de circuito impreso para ningún equipo electrónico empleado en el presente proyecto, por lo que serán adquiridos de acuerdo a la oferta de proveedores existentes.
- La tasa de transferencia de escritura de datos del sensor MPU9250 no podrá ser mayor a 1 MHz, y la tasa de lectura será menor a 20 MHz, de acuerdo a las especificaciones del sensor MPU9250.
- En el desarrollo de la interfaz gráfica del videojuego solo se podrán de emplear las funcionalidades permitidas en el entorno de Unreal Engine.

# Capítulo 1

## Marco Teórico

### 1.1. Estado del Arte

#### 1.1.1. Mining Spatial-Temporal Patterns and Structural Sparsity for Human Motion Data Denoising

En este artículo se presenta un filtro para reducir la cantidad de ruido presente y los valores atípicos al momento de capturar el movimiento. Para ello, se propuso un modelo dirigido por datos y basado en datos que utiliza partlets (estructura cinemática que muestra similitudes entre extremidades del cuerpo humano, y que presentan patrones similares) para generar diccionarios mediante *Dictionary learning*. El esquema de funcionamiento se muestra en la Figura 1.1.

De acuerdo con Feng y cols. (2015), el algoritmo presenta un mejor desempeño comparado con otros métodos de filtrado de ruido que forman parte del estado del arte como filtros basados en Kalman, Gaussain y Wavelet.

#### 1.1.2. Experimental Comparison of Sensor Fusion Algorithms for Attitude Estimation

Tal como lo menciona Cavallo y cols. (2014), el artículo está enfocado en el diseño e implementación de un sistema hardware altamente preciso y con algoritmos de fusión rápida de datos para estimar la orientación relativa de un cuerpo rígido respecto a un marco de referencia.

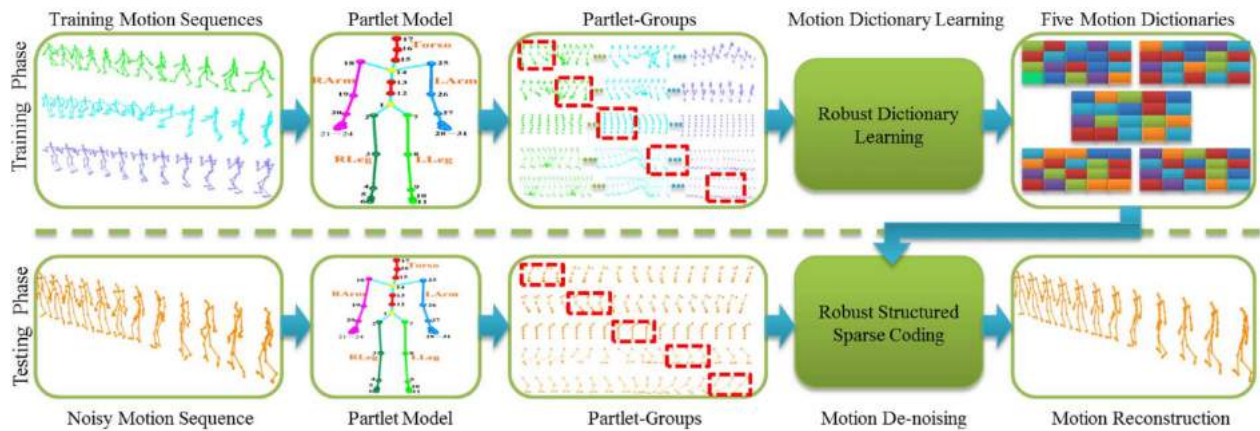


Figura 1.1: Esquema de funcionamiento del modelo propuesto. Recuperado de: (Feng y cols., 2015).

Se emplea un robot KUKA para comparar el error cuadrático medio en los ángulos de roll, pitch y yaw de los diferentes modelos AHRS como Madgwick, Mahony y Kalman. (ver Tabla 1.1). La placa empleada fue una placa de desarrollo basada en ARM-Cortex-M4, la cual se utilizó para calcular el consumo de recursos de los algoritmos (ver Tabla 1.2).

Euler angles [°]	EKF	Madgwick	Mahony
Roll (static)	0.04	0.03	0.02
Pitch (static)	0.01	0.05	0.05
Yaw (static)	0.30	1.92	1.85
Roll (slow)	4.71	4.85	5.07
Pitch (slow)	1.91	2.65	2.89
Yaw (slow)	5.19	5.13	5.67
Roll (fast)	6.55	6.51	6.69
Pitch (fast)	2.83	3.34	2.85
Yaw (fast)	6.71	7.07	6.92

Tabla 1.1: Error cuadrático medio en los ángulos roll, pitch y yaw. Extraído de: (Cavallo y cols., 2014).

Algorithm	Matlab/Simulink [ms]	Embedded System [ms]
EKF	0.1	2.7
Madgwick	0.017	0.15
Mahony	0.014	0.11

Tabla 1.2: Consumo de recursos por algoritmo. Extraído de: (Cavallo y cols., 2014).

### 1.1.3. Evaluación de algoritmos de fusión de datos para estimación de la orientación de vehículos aéreos no tripulados

El objetivo de este trabajo es evaluar y comparar los tres algoritmos de procesamiento de datos más usados en sistemas de referencia de orientación y rumbo (AHRS, por sus siglas en inglés), para vehículos aéreos no tripulados (UAVs por sus siglas en inglés).

De acuerdo con los resultados de Cristóbal y Humberto (2017), el algoritmo que mejor funcionó es el filtro complementario de Robert Mahony (ver Figura 1.2) debido a su mayor velocidad de convergencia. De los tres ángulos de rotación alrededor de los ejes principales xyz, en todas las estimaciones evaluadas, el ángulo alrededor de  $z(\psi)$  fue el que presentó la magnitud del error más grande, lo cual indica, que sigue existiendo cierta deficiencia en aquellas estimaciones que dependen del magnetómetro.

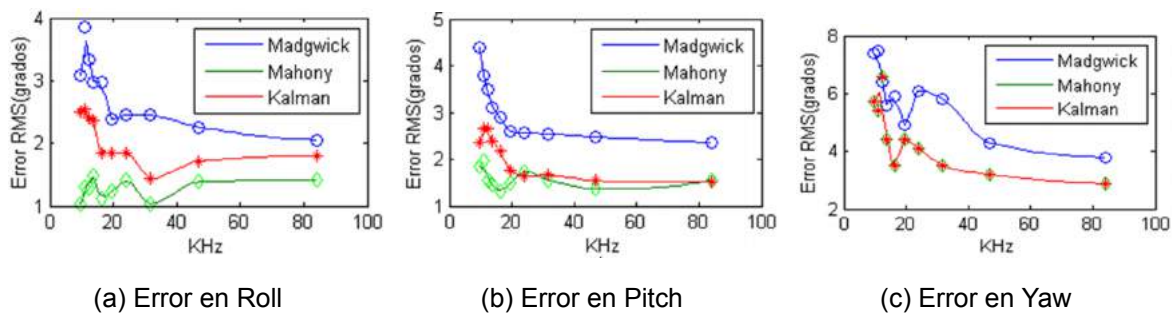


Figura 1.2: Errores en los ángulos vs frecuencia de muestreo. Recuperado de: (Cristóbal y Humberto, 2017).

En la Tabla 1.3, se muestran los resultados del tiempo de ejecución y el número de iteraciones que debe realizar cada algoritmo ante un cambio de  $30^\circ$ ; también muestra la raíz del error cuadrado medio para cada algoritmo, con una velocidad de muestreo de 84Hz.

	<b>Mahony</b>	<b>Madgwick</b>	<b>Kalman</b>
<b>Tiempo de ejecución</b>	2.8E-4	3.754E-4	1.9E-4
Error en roll	1.73	2.06	1.80
Error en pitch	1.56	2.36	1.51
Error en yaw	3.28	3.81	2.9
Ciclos para converger	2	2	4

Tabla 1.3: Ciclos para converger y tiempo de ejecución. Recuperado de: (Cristóbal y Humberto, 2017).

#### **1.1.4. Wireless Motion Capture System for Upper Limb Rehabilitation**

Este trabajo está dedicado a la presentación de un Sistema de Sensor Inalámbrico para la rehabilitación de miembros superiores para que sea utilizado como un sistema complementario para la supervisión del progreso del paciente durante los ejercicios de rehabilitación (véase la Figura 1.3). Un nodo sensor de captura de movimiento rentable compuesto por una unidad de medición inercial (IMU) de 9 grados de libertad (DoF) se monta en los segmentos de la extremidad superior del paciente y envía de forma inalámbrica las señales medidas correspondientes a una estación base. (Tsilomitrou y cols., 2021)

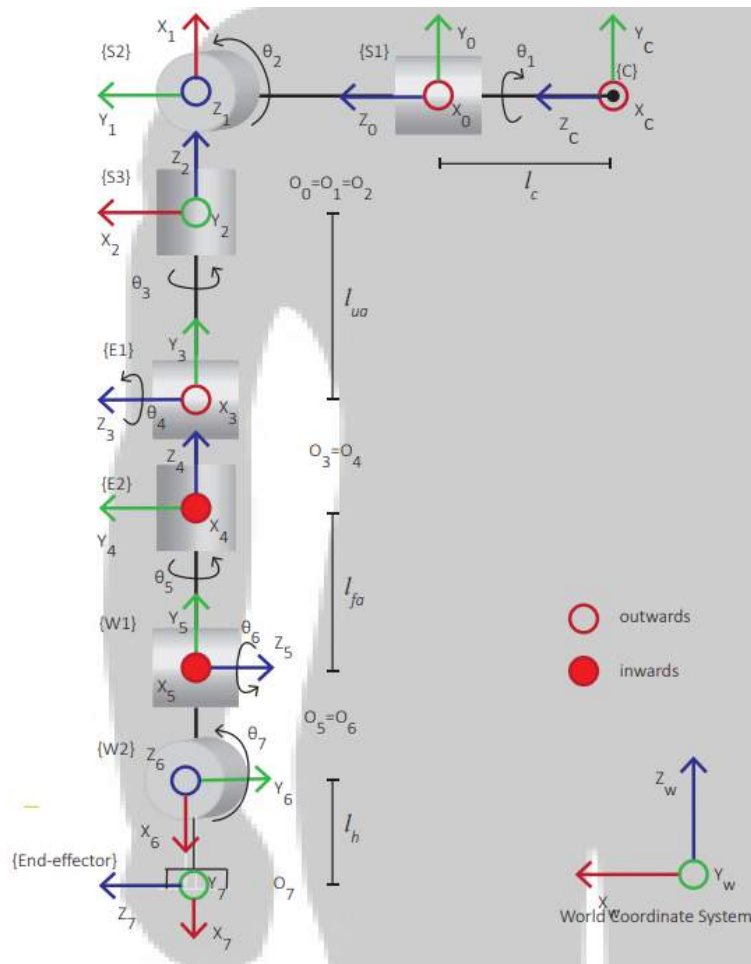
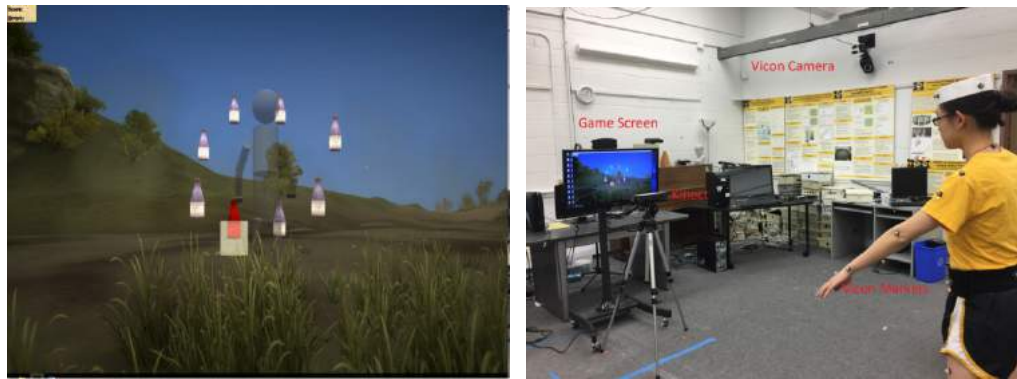


Figura 1.3: Representación de la extremidad superior como un brazo robótico. Recuperado de: (Tsilomitrou y cols., 2021).

### 1.1.5. Validation of a Kinect V2 based rehabilitation game

En este artículo se desarrolló un videojuego de cuerpo entero llamado *Mystic Isle* para ser empleado como rehabilitación, para ello emplearon un Microsoft Kinect (ver Figura 1.4). El objetivo es ayudar a los pacientes con accidentes cerebrovasculares para mejorar sus funciones motoras. Mediante el juego se evalúan los movimientos de los jugadores de forma precisas, según Ma y cols. (2018).



(a) Avatar que recolecta objetivos en un juego de rehabilitación basado en Kinect. (b) Participante con marcadores Vicon en el cuerpo.

Figura 1.4: Mystic Isle. Recuperado de: (Ma y cols., 2018).

## 1.2. Estado de la Técnica

### 1.2.1. A treatment device for hemiplegia

Esta invención se refiere a un aparato de tratamiento hemipléjico para tratar un cuerpo de hemiparesia de un sujeto. Consta de un sistema de medición del movimiento para mover al robot en base a estas. El robot se lleva montado en el cuerpo (ver Figura 1.5) al ser un exoesqueleto para el tratamiento de la hemiplejia (Gwana-ro, 2013).

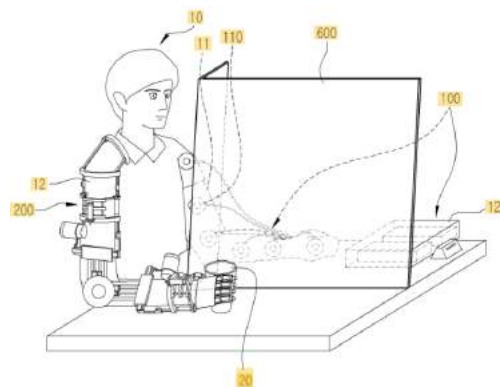


Figura 1.5: Vista en perspectiva que muestra un estado en el que un sujeto usa un aparato de tratamiento hemiparalítico. Recuperado de: (Gwana-ro, 2013).



### 1.2.2. Systems, apparatus and methods for non-invasive motion tracking to augment patient administered physical rehabilitation

En esta patente se proporciona un sistema para el seguimiento de movimiento no invasivo para aumentar la fisioterapia administrada al paciente a través de un aparato de seguimiento de movimiento (ver Figura 1.6), una pantalla y una plataforma informática.

La plataforma informática sirve para proporcionar una interfaz dirigida por menús al paciente, una instrucción para el paciente, una determinación del movimiento o acción del paciente en respuesta a la instrucción, una comparación entre la instrucción al paciente y la determinación del movimiento del paciente o acción, y para proporcionar una pantalla de retroalimentación al paciente. (Komatireddy y cols., 2012)

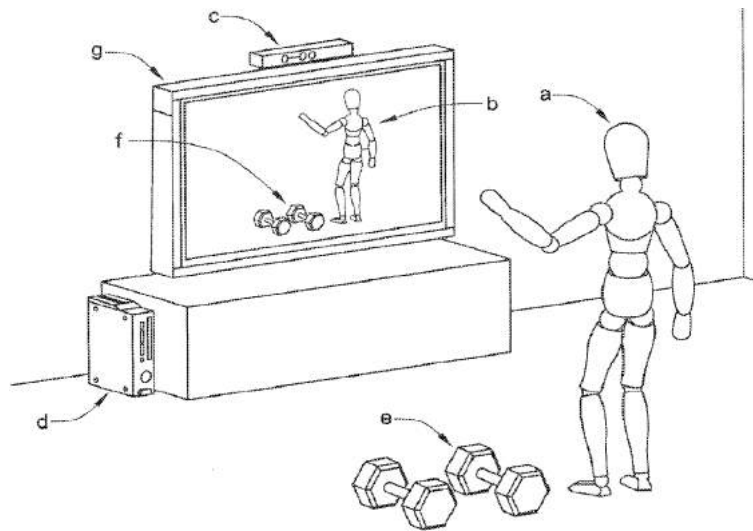


Figura 1.6: Es una vista en perspectiva de la configuración de un sistema de seguimiento de movimiento terapéutico. Recuperado de: (Komatireddy y cols., 2012).

### 1.2.3. System and method for identifying and interpreting repetitive motions

Esta patente consiste en un sistema de seguimiento de movimiento que supervisa los movimientos realizados por un usuario en función de los datos de movimiento recibidos de uno o más sensores (véase la Figura 1.7). Se emplea software para procesar los datos, con los cuales se interpreta el equipo que está utilizando el usuario.

Se le proporciona retroalimentación al usuario, ya sea visual, auditiva o táctil durante y/o después de que el usuario haya realizado un movimiento o un conjunto de movimientos. La aplicación se puede utilizar para controlar una rutina en un entorno que bien podría ser deportivo, fitness, industrial o médico. (Canavan y Hughes, 2020)

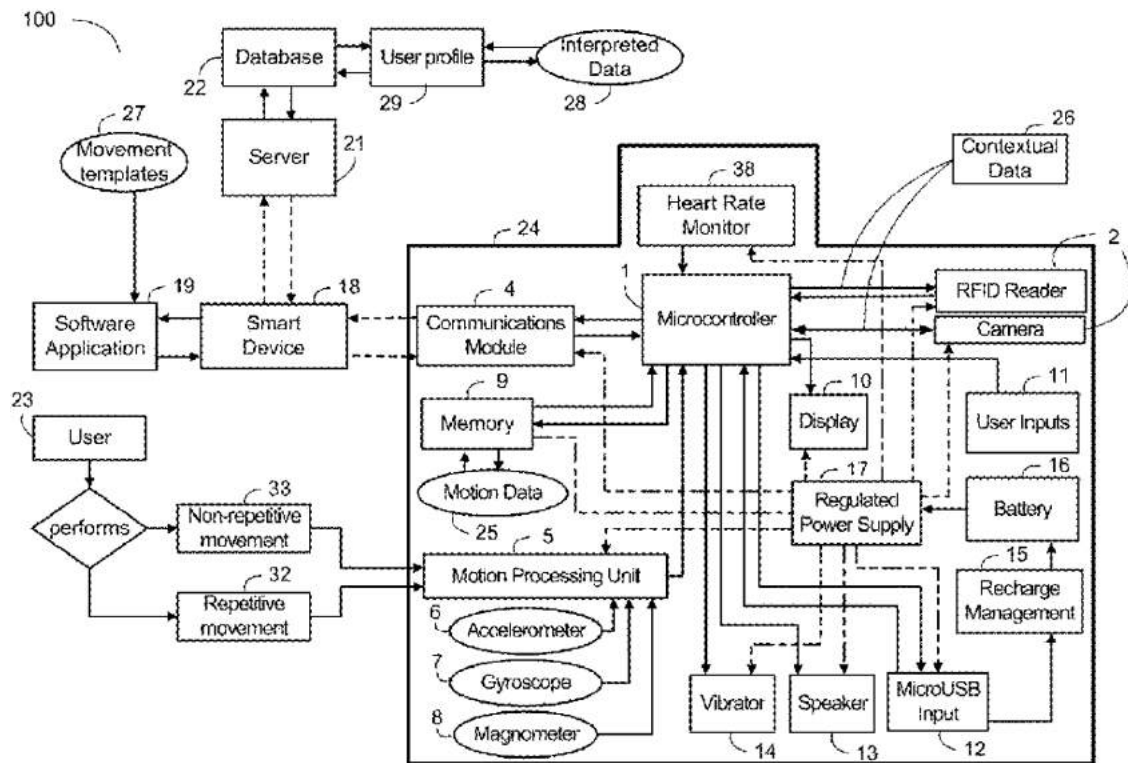


Figura 1.7: Es una vista en perspectiva de un sistema de seguimiento de movimiento.

Recuperado de: (Canavan y Hughes, 2020).

#### 1.2.4. Rehabilitation and exercise machine

Esta patente presenta una máquina de ejercicios y rehabilitación mejorada, mostrada en la Figura 1.8, que permite a una persona con limitaciones físicas, discapacidades o condiciones crónicas utilizar la máquina para rehabilitar sus músculos, mejorar la flexibilidad de las articulaciones y mejorar la aptitud cardiovascular (BurnfieldYu, Shu, Taylor, Buster, y Nelson, 2012).

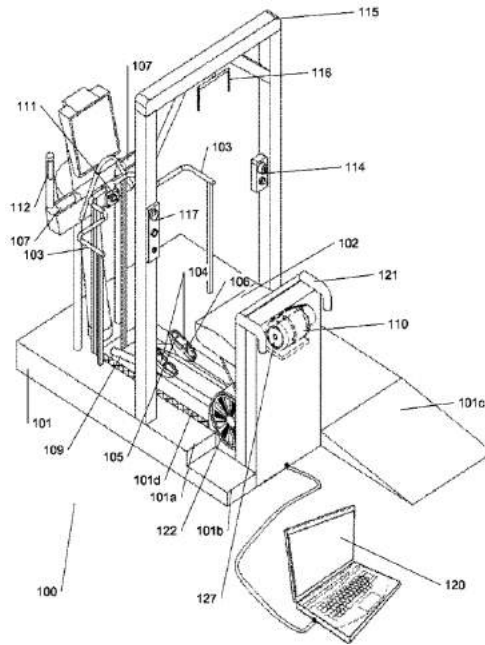


Figura 1.8: Vista isométrica de una máquina de ejercicios y rehabilitación mejorada construida de acuerdo con los principios de esta invención. Recuperado de: (Canavan y Hughes, 2020).

### 1.2.5. Wireless game controller for strength training and physiotherapy

El controlador de juegos inalámbrico proporciona una resistencia variable durante el ejercicio de entrenamiento de fuerza y se combina con un sistema de videojuegos estándar como Nintendo Wii o una computadora personal mediante una conexión Bluetooth o WiFi (véase la Figura 1.9). La resistencia la proporciona un motor/generador.

El sistema de videojuegos se puede configurar para ejecutar software de ejercicio que brinde al usuario una experiencia de juego reforzada, entrenamiento y fisioterapia. El software de ejercicios rastrea la pose del usuario usando marcas fiducias en el controlador y datos del acelerómetro 3D integrado en el controlador. La resistencia se calcula en función de uno o más de los elementos siguientes: tipo de ejercicio, posición, velocidad, perfil de usuario y número de repeticiones. (Zavadsky y Sherstyuk, 2011)

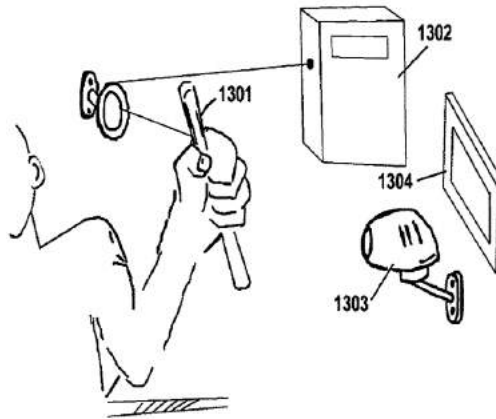


Figura 1.9: Lucha libre en línea habilitada por esta invención. Recuperado de: (Canavan y Hughes, 2020).

## 1.3. Fundamentos teóricos empleados en el proyecto

### 1.3.1. Sensores MARG

#### Acelerómetro

Los acelerómetros capacitivos, como el que se muestra en la Figura 1.10 se encuentran fabricados con tecnología MEMS. Estos tienen un electrodo móvil (una masa móvil que varía la capacitancia) y uno fijo. Cuando el sensor es excitado por una aceleración externa, la masa móvil se aproxima o se aleja, de tal forma que el cambio de capacitancia será la correlación entre la aceleración y el cambio de alguna variable eléctrica en el circuito de lectura. (Ramírez y cols., 2014)

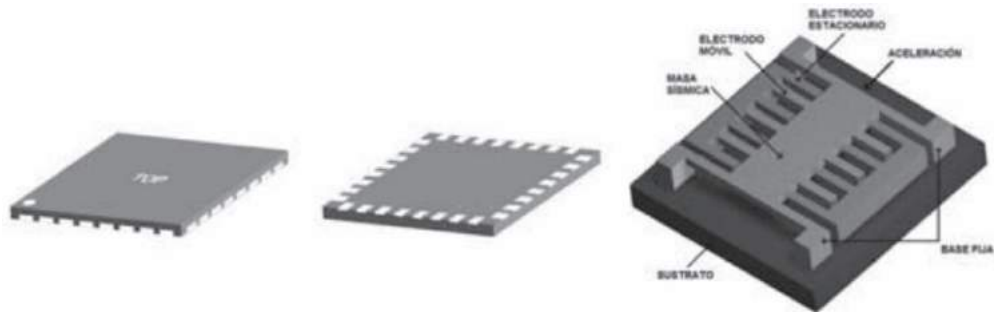


Figura 1.10: Acelerómetro capacitivo. Recuperado de: (Ramírez y cols., 2014).

## Magnetómetro

Un magnetómetro es un dispositivo que se encarga de cuantificar la intensidad y dirección de un campo magnético; en el caso de los sistemas de navegación, los magnetómetros miden la intensidad de campo magnético de la Tierra, si es que a su alrededor no se encuentran una fuente de campo mucho mayor. (Ramírez y cols., 2014)

El principio de transducción de este tipo de sensores es muy variado. Por ejemplo, algunos sensores se basan en el efecto Hall, algunos otros usan la fuerza de Lorentz o el principio piezoresistivo (Ramírez y cols., 2014).

## Efecto Hall

Mediante este efecto es posible lograr la transducción de un campo magnético en un voltaje equivalente. El efecto Hall, descubierto por Edwin F. Hall, establece que si una corriente eléctrica ( $I$ ) fluye a través de un conductor en presencia de un campo magnético ( $B$ ), se ejerce una fuerza transversal (también llamada fuerza de Lorentz) que busca equilibrar el efecto de dicho campo, produciendo un voltaje, llamado voltaje Hall, medible en los extremos del conductor (Ramírez y cols., 2014).

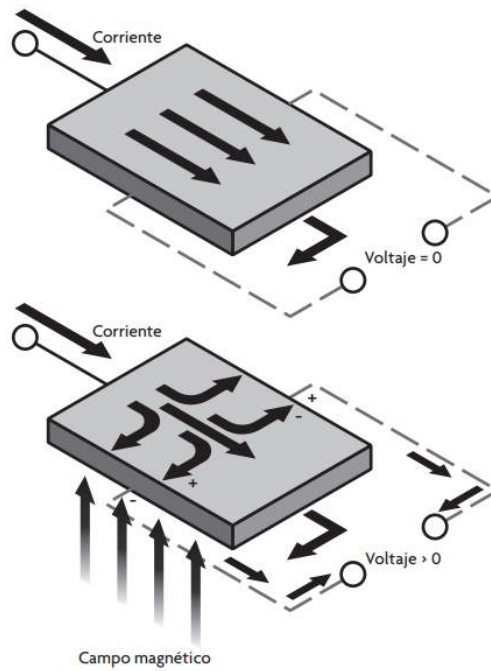


Figura 1.11: Representación del efecto Hall. Recuperado de: (Ramírez y cols., 2014).

De acuerdo con Ramírez y cols. (2014), el voltaje Hall ( $V_H$ ) para una placa conductora simple se puede calcular usando la Ecuación (1.1):

$$V_H = \frac{IB}{ned_p} \quad (1.1)$$

Donde:

- $n$ : densidad de portadores
- $e$ : carga del electrón
- $d_p$ : espesor de la placa conductora
- $I$ : corriente eléctrica
- $B$ : campo magnético

## Gir6scopo

Un giroscopio de estado s6lido (sensor MEMS) est6 formado por un cuerpo que presenta simetr6a en su rotaci6n; en este caso, sup6ngase que la masa ( $m$ ) se desplaza dentro del chip a una velocidad lineal  $\vec{v}$ , cuando al chip se le aplica un momento de fuerza, este girar6 a una velocidad angular (ver Figura 1.12a); esta combinaci6n de movimientos rotacional y lineal genera la llamada fuerza de Coriolis (v6ase la Figura 1.12b), que ser6 perpendicular al eje de movimiento lineal inicial. (Ram6rez y cols., 2014)

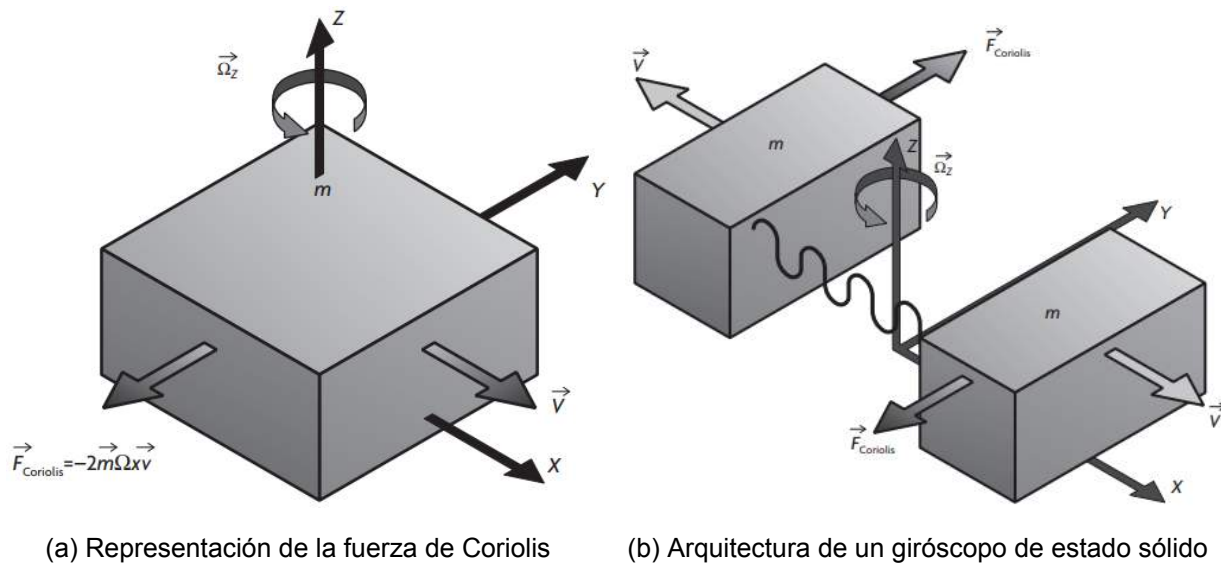


Figura 1.12: Principios de funcionamiento de un giroscopio de estado s6lido Recuperado de:  
(Ram6rez y cols., 2014).

### 1.3.2. Herramientas matem6ticas para la localizaci6n espacial

Localizar un cuerpo r6gido en el espacio precisa de conocer su posici6n y orientaci6n respecto de un sistema de referencia bien definido. Existen herramientas matem6ticas para especificar la orientaci6n y/o la posici6n de un cuerpo r6gido, mismas que se abordar6n a continuaci6n.

## Representación de la posición

En un plano tridimensional, la posición de un cuerpo rígido precisa de tres grados de libertad, por tanto, la posición del cuerpo quedará definida por tres componentes independientes; Para ello, se puede emplear un sistema con coordenadas cartesianas, cilíndricas o esféricas; de ser necesario, se puede convertir un sistema de coordenadas en cualquier otro (véase la Figura 1.13).

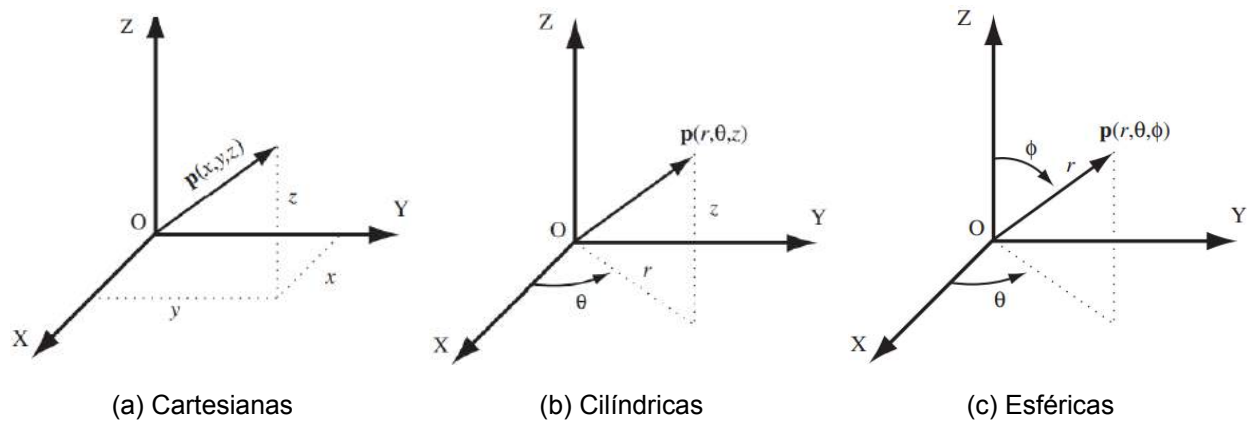


Figura 1.13: Sistemas de coordenadas. Recuperado de: (Barrientos y cols., 2007).

### Sistema de coordenadas cartesianas:

Son un tipo de coordenadas ortogonales usadas en espacios euclídeos para representar la posición. Las coordenadas se determinan respecto al origen como la longitud de cada una de las proyecciones ortogonales de un punto dado sobre cada uno de los ejes (ver Figura 1.13a).

## Representación de la orientación

De forma similar, la orientación en el espacio se ve definida por 3 variables independientes o 3 grados de libertad, tal y como se muestra en la Figura 1.14.



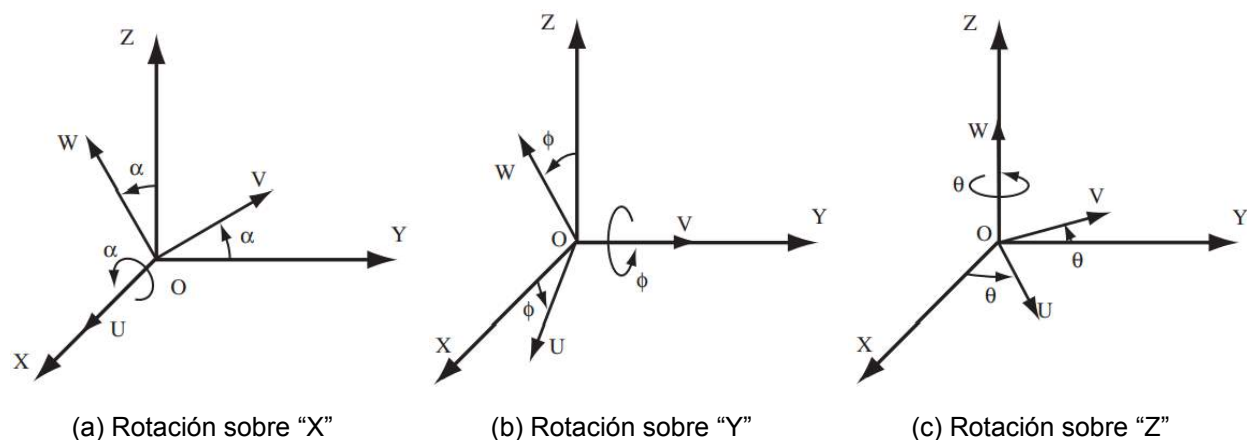


Figura 1.14: Ejemplo de rotaciones básicas. Recuperado de: (Barrientos y cols., 2007).

### Matrices de rotación:

Si los sistemas  $OXYZ$  y  $OUVW$  son coincidentes en el origen, un vector  $\mathbf{P}$  en el espacio tridimensional podrá ser referido en cualquiera de los 2 sistemas, como se muestra a continuación en la Ecuación (1.2a) y (1.2b):

$$\mathbf{p}_{uvw} = \begin{bmatrix} p_u & p_v & p_w \end{bmatrix}^T = p_u \mathbf{i}_u + p_v \mathbf{j}_v + p_w \mathbf{k}_w \quad (1.2a)$$

$$\mathbf{p}_{xyz} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T = p_x \mathbf{i}_x + p_y \mathbf{j}_y + p_z \mathbf{k}_z \quad (1.2b)$$

Donde:

- $OXYZ$  es el sistema de referencia fijo
- $OUVW$  es el sistema solidario al objeto cuya orientación se desea definir
- Los vectores unitarios del sistema  $OXYZ$  serán  $i_x, j_y, k_z$ .
- Los vectores unitarios del sistema  $OUVW$  serán  $i_u, j_v, k_w$ .

Ahora bien, es posible conocer la orientación del sistema  $OUVW$  respecto del sistema  $OXYZ$ , definida por la matriz de rotación  $\mathbf{R}$ , si se conoce un vector  $\mathbf{P}$  que esté referido

en cada uno de estos 2 sistemas (véase la Figura 1.15, y la Ecuación (1.3)), para ello se procede como se muestra a continuación:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \mathbf{R} \begin{bmatrix} p_u \\ p_v \\ p_w \end{bmatrix} \quad (1.3)$$

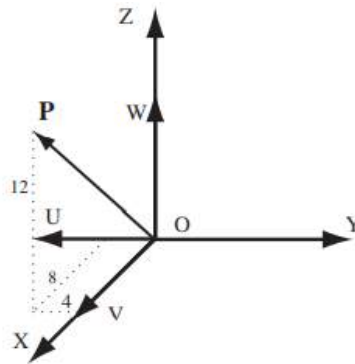


Figura 1.15: Mismo vector **P** referido en 2 sistemas coordenados distintos. Recuperado de: (Barrientos y cols., 2007).

De forma que **R** equivale a lo mostrado en la Ecuación (1.4):

$$\mathbf{R} = \begin{bmatrix} i_x i_u & i_x j_v & i_x k_w \\ j_y i_u & j_y j_v & j_y k_w \\ k_z i_u & k_z j_v & k_z k_w \end{bmatrix} \quad (1.4)$$

### Ángulos de Euler XYZ:

Los ángulos de Euler XYZ son una composición de rotaciones sobre ejes fijos también se le conoce como Yaw, Pitch y Roll (véase la Figura 1.16). La matriz de rotación se obtiene realizando 3 giros en el siguiente orden:

1. Girar el sistema  $OUVW$  un ángulo ( $\phi$ ) con respecto al eje  $OX$ . Es el denominado Yaw o guiñada.
2. Girar el sistema  $OUVW$  un ángulo ( $\theta$ ) con respecto al eje  $OY$ . Es el denominado Pitch o cabeceo.
3. Girar el sistema  $OUVW$  un ángulo ( $\psi$ ) con respecto al eje  $OZ$ . Es el denominado Roll o alabeo.

NOTA: Existen diferentes representaciones de los ángulos de Euler, las más habituales son WUW, WWV y XYZ, pudiendo componer rotaciones sobre ángulos previamente girados, o bien sobre ángulos fijos.

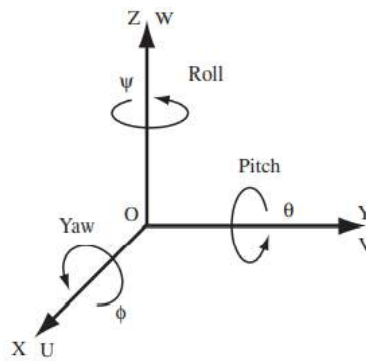


Figura 1.16: Ángulos de Euler XYZ (Yaw, Pitch y Roll). Recuperado de: (Barrientos y cols., 2007).

### Cuaterniones:

Un cuaternión  $Q$  consta de 4 componentes:  $q_0$ ,  $q_1$ ,  $q_2$  y  $q_3$ , que representan las coordenadas del cuaternión en una base  $\mathbf{e}$ ,  $\mathbf{i}$ ,  $\mathbf{j}$ ,  $\mathbf{k}$ . La componente en  $\mathbf{e}$ :  $q_0$ , frecuentemente, se le denomina como la parte escalar del cuaternión; y al resto de componentes, como la parte vectorial. De modo que un cuaternión se puede representar como se muestra en la Ecuación (1.5).

$$Q = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix} = \begin{bmatrix} s & \mathbf{v} \end{bmatrix} \quad (1.5)$$

Para asociar la representación de la orientación con un cuaternión es necesario conocer el ángulo de giro  $\theta$  sobre un vector  $\vec{k}$ , de la siguiente forma (véase la Figura 1.17 y la Ecuación (1.6)):

$$Q = \mathbf{Rot}(\mathbf{k}, \theta) = \left( \cos\left(\frac{\theta}{2}\right), \mathbf{k} \sin\left(\frac{\theta}{2}\right) \right) \quad (1.6)$$

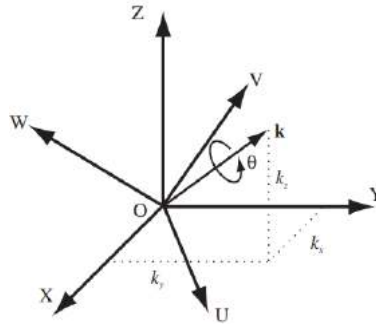


Figura 1.17: Representación gráfica de la rotación que ocurre al emplear cuaterniones.

Recuperado de: (Barrientos y cols., 2007).

### Matrices de transformación homogénea

La matriz de transformación homogénea **T** se define como una matriz de dimensión 4x4 que representa la transformación de un vector de coordenadas homogéneas de un sistema de coordenadas a otro, tal como se muestra en la Ecuación (1.7):

$$\mathbf{T} = \begin{bmatrix} R_{3 \times 3} & P_{3 \times 1} \\ f_{1 \times 3} & w_{1 \times 1} \end{bmatrix} = \begin{bmatrix} \text{Rotación} & \text{Traslación} \\ \text{Perspectiva} & \text{Escalado} \end{bmatrix} \quad (1.7)$$

La matriz **T** de transformación se suele escribir como se muestra en la Ecuación (1.8):

$$\mathbf{T} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{n} & \mathbf{o} & \mathbf{a} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.8)$$

Donde:

- **n**, **o** y **a** es una terna ortonormal que representa la orientación, tal como lo hace una matriz de rotación.
- **p** es un vector que representa la posición

### Composición de matrices homogéneas

Para referir la localización de un objeto con respecto a un sistema OXYZ de referencia, tras diversos giros y traslaciones consecutivas del objeto, es necesario componer diversas matrices de transformación homogéneas.

En la Figura 1.18 se tiene un manipulador cuya base está referida al sistema OXYZ. Para localizar el extremo de la herramienta “H” del manipulador con respecto al sistema de referencia OXYZ, se procede como se muestra en la Ecuación (1.9):

$${}^M\mathbf{T}_H = {}^M\mathbf{T}_R {}^R\mathbf{T}_E {}^E\mathbf{T}_H = {}^M\mathbf{T}_O {}^O\mathbf{T}_H \quad (1.9)$$

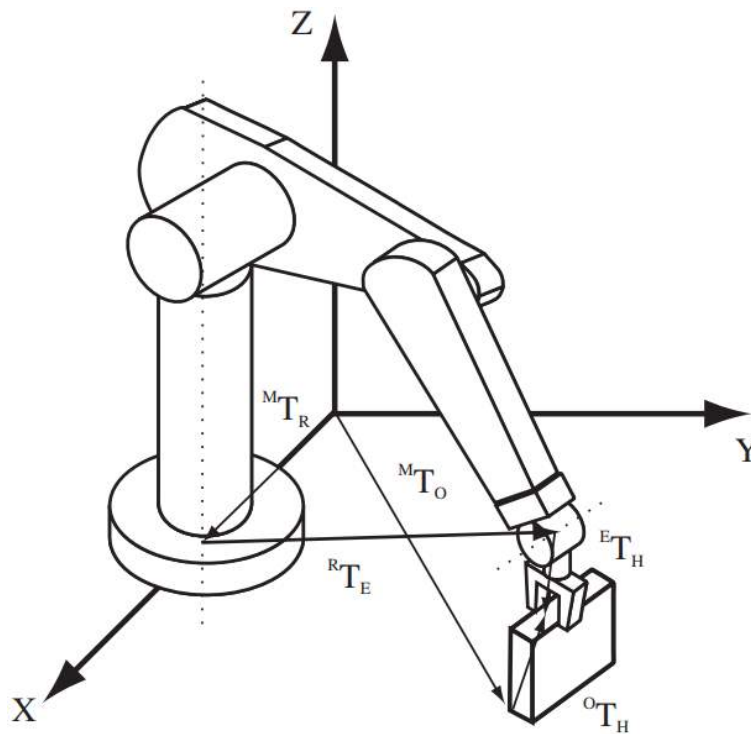


Figura 1.18: Aplicación de diversas transformaciones para localizar (posición y orientación) un objeto. Recuperado de: (Barrientos y cols., 2007).

NOTA: El producto de matrices no es conmutativo. Si se invierte el orden de las transformaciones, el resultado será distinto.

### 1.3.3. Conversión directa e inversa entre algunas herramientas matemáticas de localización espacial

#### Relación entre ángulos de Euler XYZ y matriz de transformación homogénea

##### Relación directa:

Para obtener la matriz de transformación homogénea en función de los ángulos de Euler, basta con efectuar la composición de matrices asociadas a estos ángulos, tal como se muestra en la Ecuación (1.10)

$$\begin{aligned}
\mathbf{T}_{ZYX} &= \mathbf{Rot}_z(\psi) \mathbf{Rot}_y(\theta) \mathbf{Rot}_x(\phi) = \\
&\begin{bmatrix} C\psi & -S\psi & 0 & 0 \\ S\psi & C\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C\theta & 0 & S\theta & 0 \\ 0 & 1 & 0 & 0 \\ -S\theta & 0 & C\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\phi & -S\phi & 0 \\ 0 & S\phi & C\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \\
&\begin{bmatrix} C\psi C\theta & C\psi S\phi S\theta - S\psi C\phi & C\psi C\phi S\theta + S\psi S\phi & 0 \\ S\psi C\theta & S\psi S\phi S\theta + C\psi C\phi & S\psi C\phi S\theta - C\psi S\phi & 0 \\ -S\theta & S\phi C\theta & C\phi C\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.10)
\end{aligned}$$

### Relación inversa:

Para obtener los ángulos de Euler en función de las componentes de una matriz de transformación homogénea, basta con hacer algunas operaciones trigonométricas entre estas componentes (mostradas en la Ecuación (1.10)), y de ahí despejar los ángulos de Euler, tal como se puede apreciar en la Ecuación (1.11a), (1.11b) y (1.11c).

$$\phi = \arctan\left(\frac{\mathbf{T}_{ZYX}(3, 2)}{\mathbf{T}_{ZYX}(3, 3)}\right) \quad (1.11a)$$

$$\theta = \arcsin(-\mathbf{T}_{ZYX}(3, 1)) \quad (1.11b)$$

$$\psi = \arctan\left(\frac{\mathbf{T}_{ZYX}(2, 1)}{\mathbf{T}_{ZYX}(1, 1)}\right) \quad (1.11c)$$

### Relación entre cuaternión y matriz de transformación homogénea

#### Relación directa:

La representación de una matriz de transformación homogénea  $T$  en función de las componentes de un cuaternión  $Q$  viene dada por la Ecuación (1.12):

$$\mathbf{T} = 2 \begin{bmatrix} q_0^2 + q_1^2 - \frac{1}{2} & q_1 q_2 - q_3 q_0 & q_1 q_3 + q_2 q_0 & 0 \\ q_1 q_2 + q_3 q_0 & q_0^2 + q_2^2 - \frac{1}{2} & q_2 q_3 - q_1 q_0 & 0 \\ q_1 q_3 - q_2 q_0 & q_2 q_3 + q_1 q_0 & q_0^2 + q_3^2 - \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{bmatrix} \quad (1.12)$$

#### Relación inversa:

La relación inversa, mostrada en la Ecuación (1.13a), (1.13b), (1.13c) y (1.13d), se puede encontrar igualando la suma y/o resta de algunos elementos de la diagonal principal de la Ecuación (1.8) y (1.12).

$$q_0 = \frac{1}{2} \sqrt{n_x + o_y + a_z + 1} \quad (1.13a)$$

$$q_1 = \operatorname{sgn}(o_z - a_y) \frac{1}{2} \sqrt{n_x - o_y - a_z + 1} \quad (1.13b)$$

$$q_2 = \operatorname{sgn}(a_x - n_z) \frac{1}{2} \sqrt{-n_x + o_y - a_z + 1} \quad (1.13c)$$

$$q_3 = \operatorname{sgn}(n_y - o_x) \frac{1}{2} \sqrt{-n_x - o_y + a_z + 1} \quad (1.13d)$$

#### 1.3.4. Descenso del gradiente

De acuerdo con Phillips (2017), el descenso del gradiente es una familia de técnicas que para una función diferenciable  $f : \mathbb{R}^d \rightarrow \mathbb{R}$ , intenta identificar ya sea:

$$\min_{x \in \mathbb{R}^d} f(x)$$

O bien,

$$x^* = \arg \min_{x \in \mathbb{R}^d} f(x)$$



Esto es efectivo cuando  $f$  es convexa y no tenemos una solución de “forma cerrada” de  $x^*$ . El algoritmo es iterativo, ya que puede que nunca alcance el  $x^*$  completamente óptimo, pero sigue acercándose cada vez más. (Phillips, 2017)

---

**Algorithm 1** Gradient Descent( $f, x_{start}$ )

---

initialize  $x^{(0)} = x_{start} \in \mathbb{R}^d$

**repeat**

  |  $x^{(k+1)} := x^{(k)} - \gamma_k \nabla f(x^{(k)})$

**until** ( $\|\nabla f(x^{(k)})\| \leq \tau$ );

**return**  $x^{(k)}$

---

Básicamente, para un punto de partida  $x^{(0)}$ , el algoritmo se mueve a otro punto en la dirección opuesta al gradiente, esto es, en la dirección que disminuye  $f$ , localmente, más rápido (véase la Figura 1.19).

**Condición de parada.** El parámetro  $\tau$  es la tolerancia del algoritmo. Si asumimos que la función es diferenciable, entonces en el mínimo  $x^*$ , debemos tener que  $\nabla f(x) = (0, 0, \dots, 0)$ . Tan cerca del mínimo, también debería tener una pequeña norma/magnitud. (Phillips, 2017)

El parámetro más crítico del descenso de gradientes es  $\gamma_k$ , la tasa de aprendizaje. En muchos casos, el algoritmo mantendrá  $\gamma_k = \gamma$  fijo para todo  $k$ . Controla qué tan rápido funciona el algoritmo. Pero si es demasiado grande, cuando nos acercamos al mínimo, entonces el algoritmo puede ir demasiado lejos y sobrepasarlo. (Phillips, 2017)

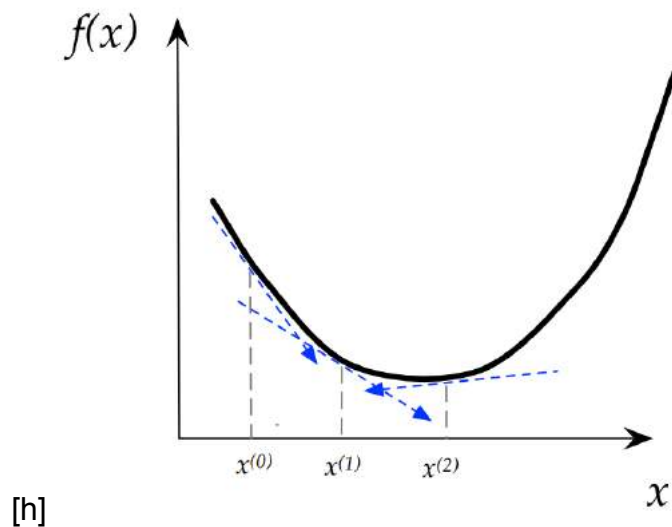


Figura 1.19: Un ejemplo de búsqueda de gradiente para un punto estacionario. Recuperado de:  
(Singer, 2016).

### 1.3.5. Sistemas de Referencia de Actitud y Rumbo

Los Sistemas de Referencia de Actitud y Rumbo (AHRS, por sus siglas en inglés) es un equipo pensado en la navegación inercial. Normalmente, los AHRS están formados por acelerómetro, magnetómetro y giroscopo, algunos pueden incluir también un receptor GPS. Cuando proporcionan información adicional como la altitud, temperatura del aire exterior y la velocidad del avión relativa al aire pasan a denominarse como Air Data, Attitude and Heading Reference Systems (ADAHRS).

#### Filtro Madgwick

Es un filtro de orientación aplicable a sensores IMU y MARG, los cálculos se efectúan mediante cuaterniones y se emplea el algoritmo del descenso del gradiente (véase el Subtema 1.3.4) para calcular la dirección del error de medición del giroscopo. El filtro incluye una compensación de la distorsión magnética y de la deriva de sesgo del giroscopio, tal como se muestra en la Figura 1.20.

Beneficios del filtro: (1) computacionalmente económico, requiriendo 109 (IMU) o 277 (MARG) operaciones aritméticas escalares cada actualización de filtro; (2) efectivo a bajas tasas de muestreo (p.ej., a 10 Hz), y (3) contiene 1 (IMU) o 2 (MARG) parámetros ajustables definidos por características observables del sistema (Madgwick, 2010).

“Los resultados indican que el filtro alcanza niveles de precisión superiores a los del algoritmo basado en Kalman; <0,6° error RMS estático, <0,8° error RMS dinámico” (Madgwick, 2010).

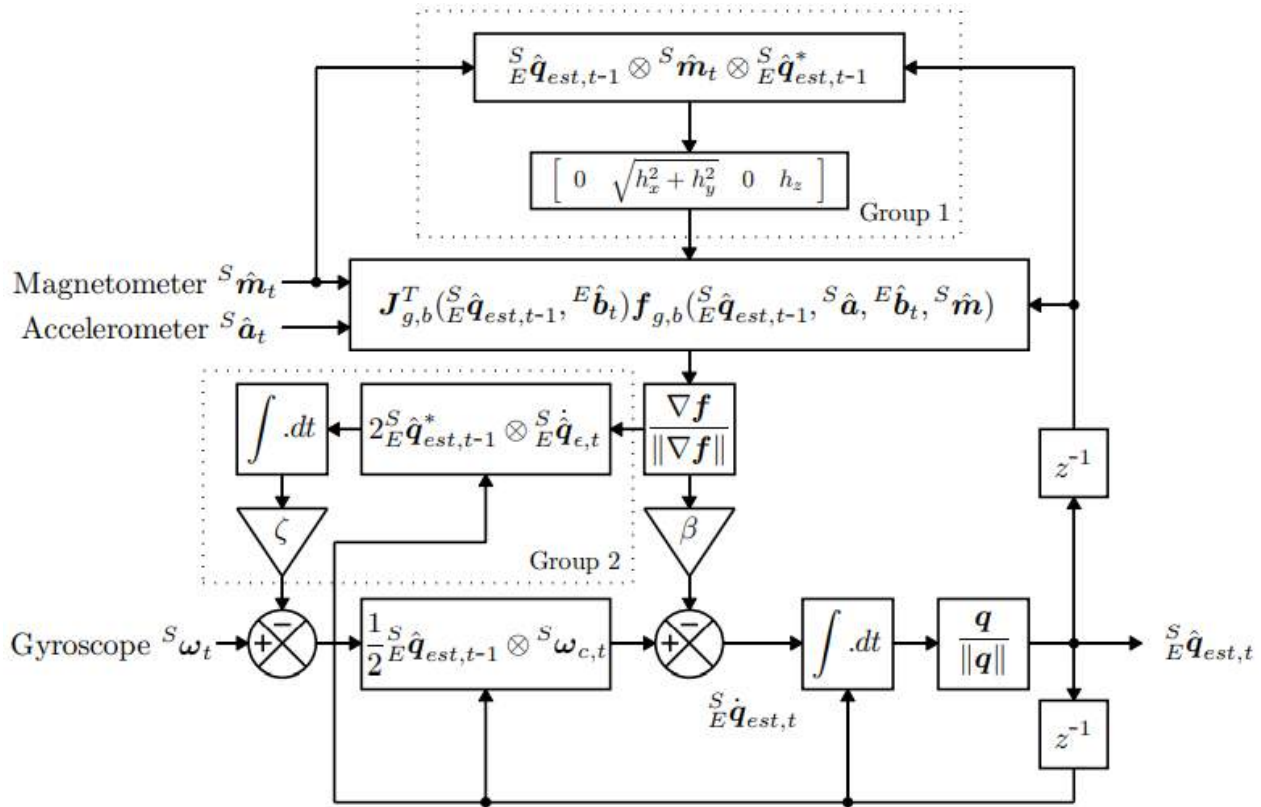


Figura 1.20: Diagrama del algoritmo Madgwick para sensores MARG. Recuperado de: (Madgwick, 2010).

**Velocidad angular en cuaterniones:**

Sea  $q(t)$  una función de cuaternión unitaria, y  $\omega(t)$  la velocidad angular determinada por  $q(t)$ . De acuerdo con Jia (2020), la derivada de  $q(t)$  se muestra en la Ecuación (1.14).

$$\dot{q} = \frac{1}{2}\omega q \quad (1.14)$$

**1.3.6. Comunicaciones seriales****I2C**

I2C es una interfaz de dos cables compuesta por señales de datos en serie (SDA) y reloj en serie (SCL). En general, las líneas son de drenaje abierto y bidireccionales. En una implementación de interfaz I2C generalizada, los dispositivos conectados pueden ser un maestro o un esclavo. El dispositivo maestro coloca la dirección del esclavo en el bus, y el dispositivo esclavo con la dirección coincidente reconoce al maestro. (TDK InvenSense, 2016)

**SPI**

SPI es una interfaz serial síncrona de 4 cables que utiliza 2 líneas de control y 2 líneas de datos. Con respecto al maestro son 3 salidas y una entrada: las salidas son SCLK (reloj en serie), SDO (datos en serie) y CS (Chip Select); y la entrada es SDI. Cada esclavo SPI requiere su propia línea CS del maestro. Como ejemplo, en la Figura 1.21 se muestra un diagrama de conexión SPI.

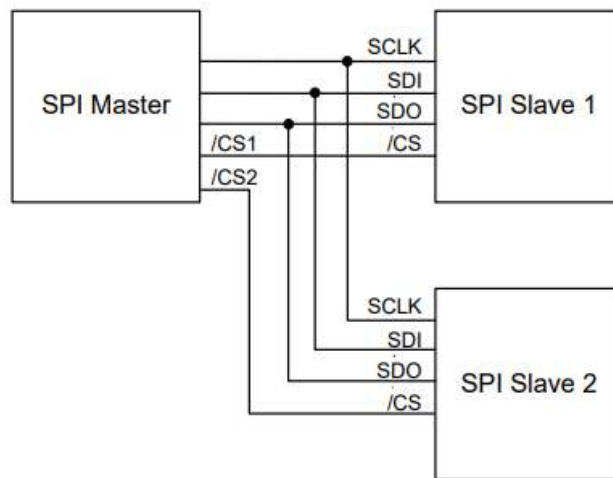


Figura 1.21: Diagrama de conexión SPI entre 1 maestro y 2 esclavos. Recuperado de: (TDK InvenSense, 2016).

### 1.3.7. Modelo OSI

OSI (en inglés, Open Systems Interconnection) es un modelo de referencia para los protocolos de red, está dividido en 7 capas: física, enlace de datos, red, transporte, sesión, presentación y aplicación, donde cada capa tiene sus propias funciones para alcanzar un objetivo global: interconectar sistemas para enviar y recibir información, como se aprecia en la Figura 1.22

Por tanto, una de las finalidades del modelo OSI es sentar una base común para la coordinación en el desarrollo de normas destinadas a la interconexión de sistemas.

### Modelo TCP/IP

El modelo TCP/IP está formado por un conjunto de protocolos que permiten la comunicación entre los ordenadores que forman la red (véase la Figura 1.23a). las siglas TCP/IP hace referencia a 2 protocolos para redes de computadoras distintos: TCP e IP, donde IP es la parte que obtiene la dirección a la que se envían los datos, y TCP se encarga de la entrega fiable de los datos una vez hallada dicha dirección IP.

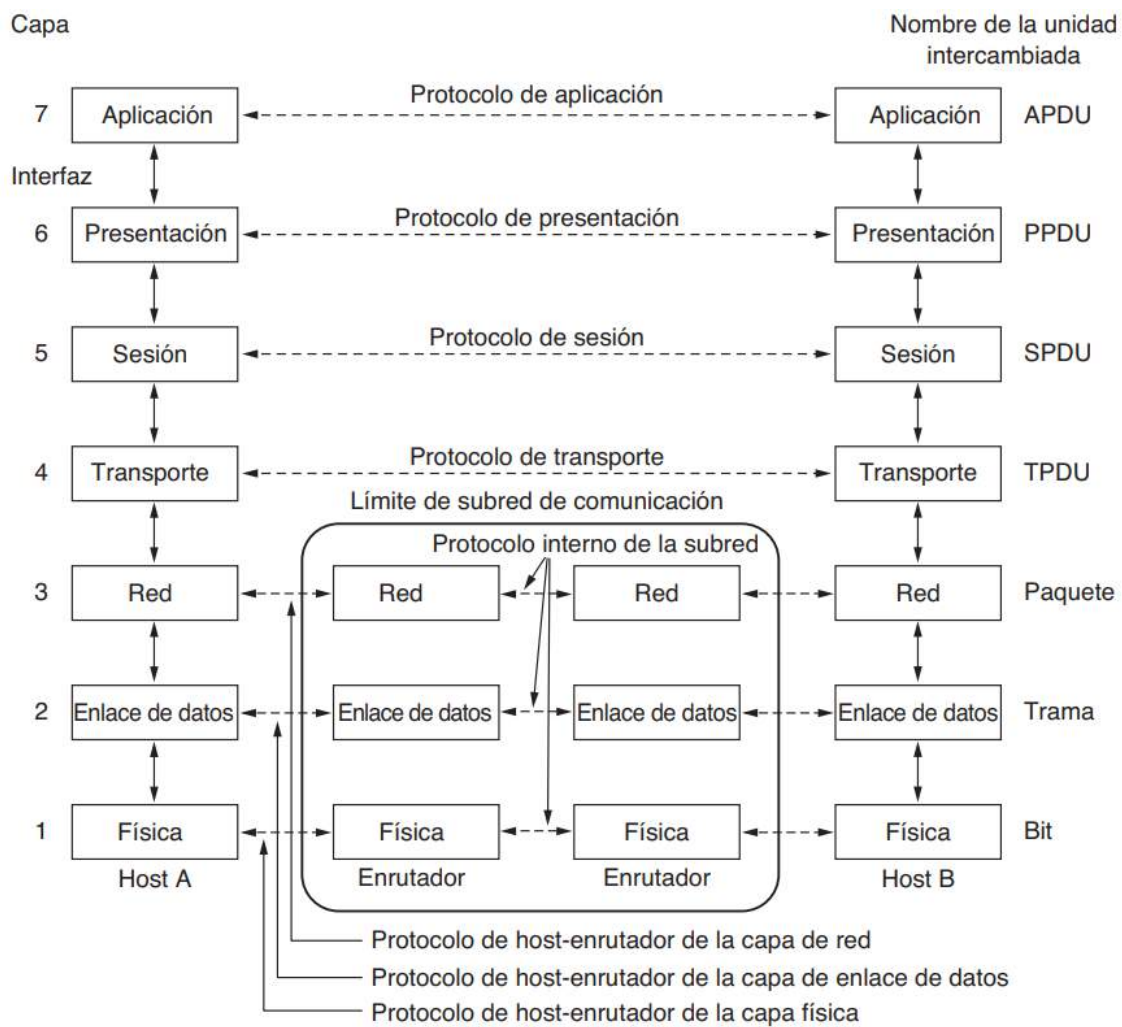
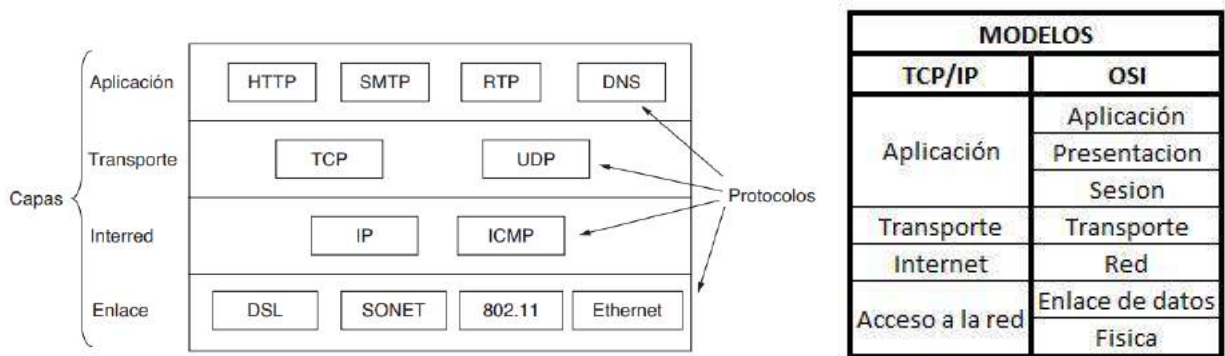


Figura 1.22: Niveles o capas del modelo OSI. Recuperado de: (Tanenbaum y Wetherall, 2012).

El modelo TCP/IP se divide en cuatro capas diferenciadas: Acceso a la red, internet, transporte y aplicación, tal como se muestra en la Figura 1.23b.

Actualmente, la mayoría de los ordenadores que están conectados a alguna red lo hacen utilizando el modelo TCP/IP.



(a) Algunos protocolos del TCP/IP. Recuperado de: (Tanenbaum y Wetherall, 2012).

(b) Comparación entre el modelo TCP/IP y el modelo OSI. Recuperado de: (Rvelandia, 2008).

Figura 1.23: Modelo TCP/IP

## WIFI

WiFi (en inglés, Wireless Fidelity) es una tecnología de red inalámbrica a través de la cual los dispositivos como computadoras, dispositivos móviles y otros equipos (impresoras, videocámaras, etc.) puedan intercambiar información entre sí, ya sea en una red local o mediante internet.

Se basa en el estándar 802.11, el cual define los protocolos de comunicación que permiten la comunicación con routers y puntos de acceso inalámbrico. Los estándares funcionan con diversas frecuencias, proporcionan distintos anchos de banda y admiten distintas cantidades de canales.

El protocolo 802.11, o WiFi como también se le conoce en la jerga computacional, se sitúa abarcando la capa física y la capa de enlace del modelo OSI, o bien la capa de acceso a la red del modelo TCP (véase la Figura 1.23).

## Internet Protocol (IP)

Es un protocolo de comunicación que proporciona la entrega de paquetes a través de la red, no está orientado a la conexión (es decir, cada trama puede ir por diversas rutas, no se establece conexión), no es fiable (es decir, no garantiza la entrega de paquetes) y su función principal es el encaminamiento. IP está situado en la capa de red del modelo OSI (véase la Figura 1.23), y representa el corazón del conjunto de protocolos de Internet.

Existen 2 variantes mundialmente extendidas del protocolo IP: IPv4 e IPv6. IPv4 fue la primera versión oficial del protocolo IP. IPv6 fue diseñada para sustituir a IPv4, tiene direcciones IP de 16 Bytes de longitud (en lugar de los 4 Bytes de IPv4).

### Protocolo IP versión 4:

Un datagrama IPv4 consta de 2 partes: encabezado y el cuerpo (carga útil). El formato de encabezado se muestra en la Figura 1.24. El encabezado consta de una parte de longitud fija de 20 Bytes, y una parte opcional con longitud variable.

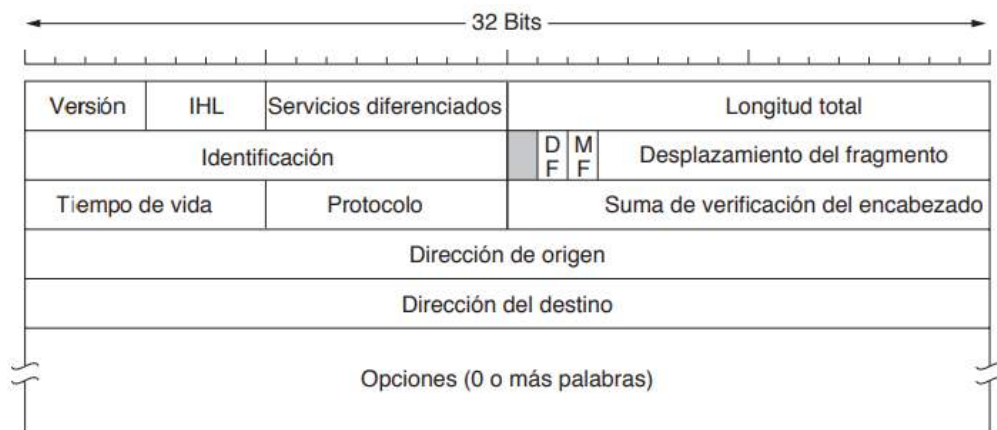


Figura 1.24: Encabezado de IPv4. Recuperado de: (Tanenbaum y Wetherall, 2012).



## UDP

UDP (Protocolo de Datagrama de Usuario, del inglés User Datagram Protocol) es un protocolo no orientado a la conexión, no fiable, no cuenta con control de flujo y de congestión (es posible que cierta información enviada se pierda). UDP está situado en la capa de transporte del modelo OSI (véase la Figura 1.23).

UDP es empleado para la transmisión de voz y video a través de una red, esto porque la entrega de información en tiempo real es muy importante en estas aplicaciones.

### Puertos UDP:

El campo de la cabecera destinada al puerto de origen o de destino tiene una longitud de 16 bits, por lo que rango vas desde 0 a 65535 (véase la Figura 1.25).

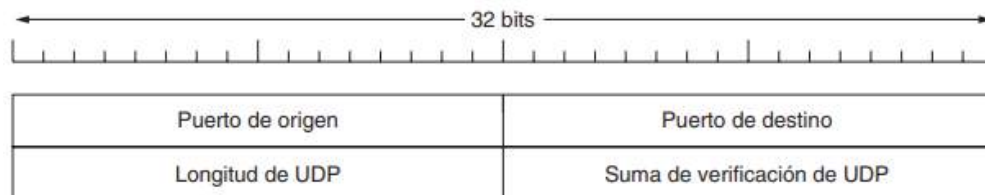


Figura 1.25: Encabezado de UDP. Recuperado de: (Tanenbaum y Wetherall, 2012).

## Internet

Internet es un enorme conjunto descentralizado de redes que utilizan protocolos comunes, a través de internet se ofrecen diversos servicios como lo es el intercambio de archivos, la mensajería instantánea, el correo electrónico, etc.

A la familia de protocolos que forman Internet, en ocasiones, se les denominan conjunto de protocolos TCP/IP como referencia a los 2 protocolos más importantes que la componen.

### 1.3.8. Microcontroladores y microcomputadores

#### Arduino

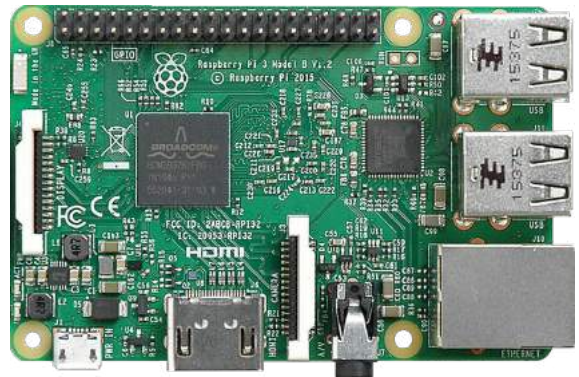
Arduino<sup>1</sup> es una plataforma electrónica de código abierto basada en hardware y software fáciles de usar. Las placas Arduino pueden leer entradas (un botón, luz en un sensor, etc.) y convertirlo en una salida (Arduino, 2018). Arduino Uno R3 es un modelo de las placas electrónicas Arduino (ver Figura 1.26a).

#### Raspberry Pi 3B

La Raspberry Pi es una serie de ordenadores de placa simple (SBC, por sus siglas en inglés) de bajo coste desarrollados en el Reino Unido por la Raspberry Pi Foundation, con el objetivo de poner en manos de las personas de todo el mundo el poder de la informática y la creación digital (Raspberry Pi, 2021). Raspberry Pi 3B V1.2 se muestra en la Figura 1.26b, y es un modelo de las placas electrónicas de la Raspberry Pi Foundation.



(a) Arduino Uno R3. Recuperado de: (Oomlout, 2015).



(b) Raspberry Pi 3B. Recuperado de: (Herbfargus, 2016).

Figura 1.26: Placas electrónicas empleadas en pruebas y desarrollo.

<sup>1</sup>La palabra Arduino puede hacer referencia al hardware (placas electrónicas), al IDE, al nombre de la compañía, etc.; por lo que su uso puede parecer un poco ambiguo.

### **1.3.9. Entornos de desarrollo integrados**

Un entorno de desarrollo integrado (IDE) es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software. Generalmente, un IDE cuenta con las siguientes características: editor de código fuente, automatización de compilaciones locales y depurador (Entorno de desarrollo integrado, 2021).

#### **Arduino**

El lenguaje de programación Arduino está basado en Wiring, y el Software Arduino (IDE) está basado en Processing (Arduino, 2018).

Arduino nació en el Ivrea Interaction Design Institute como una herramienta fácil para la creación rápida de prototipos, dirigida a estudiantes sin experiencia en electrónica y programación (Arduino, 2018).

#### **Matlab**

“MATLAB es una plataforma de programación y cálculo numérico utilizada por millones de ingenieros y científicos para analizar datos, desarrollar algoritmos y crear modelos” (Matlab, s.f.).

Prestaciones de Matlab: análisis de datos, presentar gráficas, desarrollo de algoritmos, creación de apps, utilizarse junto a otros lenguajes de programación (C/C++, Fortran, Java, Python, etc.), conexiones con hardware, cálculo paralelo, cálculo en la nube, y más. (Matlab, s.f.)

#### **Microsoft Visual Studio**

Es un IDE desarrollado para Windows y Mac, permite trabajar con múltiples lenguajes de programación como C++, C#, Visual Basic .NET, Python, JavaScript, etc., pudiendo

ofrecer soluciones para Windows, aplicaciones móviles, juegos, aplicaciones web, bases de datos, etc. (Microsoft, s.f.)

## **Unreal Engine**

Unreal Engine es un conjunto completo de herramientas de desarrollo para cualquiera que trabaje con tecnología en tiempo real, es la herramienta de creación 3D en tiempo real más abierta y avanzada del mundo, con imágenes fotorrealistas y experiencias propias del estado del arte. (Epic Games, s.f.)

Unreal Engine ofrece soluciones a industrias como la de los videojuegos, arquitectura, automoción y transporte, transmisión, eventos en vivo, cine, televisión, simulación, etc., ya que entre sus características está el poder crear mundos virtuales, animaciones, jugabilidad, interactividad, simulaciones, efectos, renderizado, iluminación, producción virtual, etc.; además de tener soporte multimedia y herramientas para desarrollo. (Epic Games, s.f.)

## **Qt Creator**

Qt Creator es un IDE multiplataforma que corre en Windows, Linux y MacOS. Con él se pueden desarrollar aplicaciones para escritorio, móviles o sistemas embebidos; cuenta con herramientas de diseño y desarrollo, y permite trabajar con C++, QML, Python, JavaScript, entre otros. (The Qt Company, s.f.)

Qt creator es parte del SDK para Qt. Qt contiene un conjunto completo de clases de bibliotecas C++ altamente intuitivas y modularizadas, y está cargado con API's para simplificar el desarrollo de su aplicación. Qt produce código altamente legible, fácil de mantener y reutilizable con alto rendimiento en tiempo de ejecución y tamaño reducido, y es multiplataforma. (The Qt Company, s.f.)

### **1.3.10. Motion Capture**

Motion Capture (o captura de movimiento) es la obtención de secuencias del movimiento de un actor o un objeto (en su mayoría actores) a través de un dispositivo específico, y por medio de un ordenador asignársela a un personaje u objeto 3D; de este modo, la figura virtual se mueve de la misma manera que el actor u objeto físico.

El motion capture (mocap) es actualmente una técnica muy utilizada en el desarrollo de contenido digital, principalmente en películas, animaciones y videojuegos, es una técnica que sigue en constante crecimiento.

Para poder capturar el movimiento se puede recurrir a diversas técnicas que aprovechan diferentes leyes de la física en campos de estudio como la acústica, óptica, mecánica y radio-frecuencia.

# Capítulo 2

## Desarrollo

Primero que nada, me gustaría hacer del conocimiento del lector que los procedimientos aquí mostrados los puede llevar a la práctica con los códigos, programas y dispositivos que aquí se mencionen. Por tanto, es conveniente empezar este capítulo con 2 links que llevan a todos los códigos empleados para el proyecto, ambos links tienen el mismo contenido, pero están almacenados en 2 plataformas online diferentes: [GitHub](https://github.com/alejandroivan10/ResidenciaProfesional_IvanAlejandroMunozVera)<sup>1</sup> y [Google Drive](https://drive.google.com/drive/u/0/folders/11ay_yAM95xgmOL4HYDAj85NgqfiTDedR)<sup>2</sup>. A lo largo de este capítulo se hará mención en reiteradas ocasiones a estos dos repositorios.

Aunado a lo anterior, el código final logrado durante esta residencia profesional estará contenido en el Anexo B (y también estará almacenado en los repositorios antes mencionados), mismo que se podrá descargar, estudiar, modificar, compilar y mejorar.

### 2.1. Descripción del proyecto

El presente proyecto consiste en el diseño y desarrollo de un traje de Motion Capture para que sea empleado en futuras terapias. El proceso de diseño está orientado para que pueda ser un traje que pueda ser usado por una persona con discapacidad motriz.

El traje consiste en una red de sensores montados sobre este, serán tantos sensores como partes del cuerpo se quieran monitorear. Los datos de estos sensores son recogidos y procesados por una Raspberry Pi 3B para calcular la orientación de cada sensor y enviarla a través de Wi-Fi a una computadora personal que se encontrará ejecutando el programa de Unreal (véase la Figura 2.1).

---

<sup>1</sup>[https://github.com/alejandroivan10/ResidenciaProfesional\\_IvanAlejandroMunozVera](https://github.com/alejandroivan10/ResidenciaProfesional_IvanAlejandroMunozVera)

<sup>2</sup>[https://drive.google.com/drive/u/0/folders/11ay\\_yAM95xgmOL4HYDAj85NgqfiTDedR](https://drive.google.com/drive/u/0/folders/11ay_yAM95xgmOL4HYDAj85NgqfiTDedR)

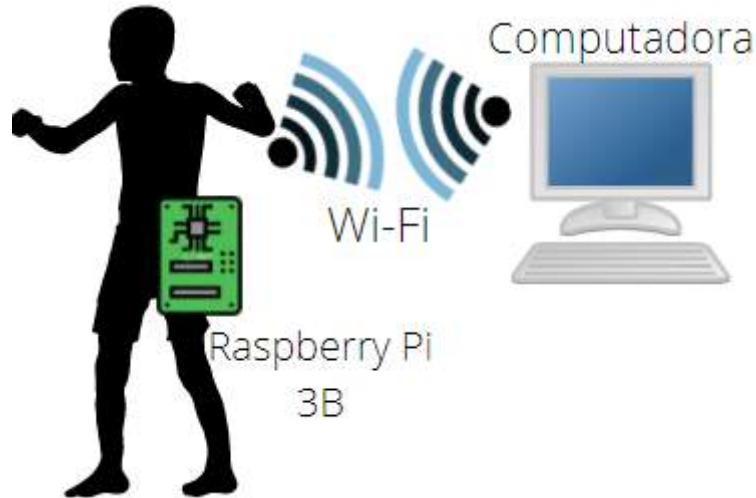


Figura 2.1: Esquema de funcionamiento

### 2.1.1. Propuesta de valor

Parte de la realización de este proyecto es con la finalidad de realizar un producto más barato y asequible; diseñar un traje que pueda ser usado por muchas personas, abarcando un espectro de tallas/medidas mucho más amplio, y así no tener que comprar tantos trajes mocap como tallas/medidas tenga cada persona que lo quiera utilizar.

Además, este traje está pensado para poderse poner y quitar de una forma más cómoda para pacientes con discapacidades motrices, ya que consiste en el chaleco, la electrónica, los cables y las vendas para sujetar los cables. Las ventajas que esto conlleva se mencionarán más adelante.

## 2.2. Cronograma de actividades programadas

Para planificar, administrar y evaluar el correcto desarrollo de la residencia profesional se establecieron las actividades y subactividades que se consideran necesarias, y se acomodaron dentro de un cronograma.

El cronograma consta de aprox. 450 actividades y 120 objetivos específicos, motivo por el cual es inadecuado mostrar una imagen del mismo; por tanto, se deja un [link](#)<sup>3</sup> a la ubicación en Google Drive. Las actividades se encuentran ordenadas de forma continua y progresiva en el tiempo.

## 2.3. Proceso de investigación inicial

### 2.3.1. Investigación de mercado del Motion Capture

#### Análisis de la oferta

Para comprender mejor la oferta de mercado en el área del motion capture es indispensable mencionar empresas, así como los productos que estas empresas tienen actualmente a la venta, y entender su modelo de negocio y su propuesta de valor de mercado.

Algunas de las empresas ligadas a la venta de productos de motion capture son: Xsens, Rokoko (Face capture, SmartGloves, Smartsuit Pro) y Chordata. Estas empresas ofrecen diversos productos para cubrir ciertas necesidades presentes en la industria del motion capture (algunos de sus productos se encuentran cotejados en la Tabla 2.1).

Tabla 2.1: Fabricantes y sus productos de captura de movimiento.

Fabricante	Productos	Tipo	# Sensores	Precio (Euros)
Xsens	MVN Awinda Starter	correas + playera	17	3490
	MVN Awinda	correas + playera	17	6490
	MVN Link	traje	17	11565
Rokoko	Smartsuit Pro	traje	19	2495
	Smartgloves	guantes	7	995
Chordata	Full motion plug &play	traje	15	1199
	Full motion	traje	15	899

<sup>3</sup>[https://drive.google.com/drive/u/0/folders/11ay\\_yAM95xgm0L4HYDAj85NgqfiTDedR](https://drive.google.com/drive/u/0/folders/11ay_yAM95xgm0L4HYDAj85NgqfiTDedR)



En cuanto al uso del mocap en la industria de los videojuegos, actualmente existen diversos dispositivos complementarios para las consolas más importantes: Kinect para Xbox (de Microsoft), Ps Move para Play Station (de Sony) y Wii Mote 3 para el Wii (de Nintendo). Estos controles explotan el uso del motion capture para utilizarlo con fines de entretenimiento.

### **2.3.2. Variables físicas útiles que debe medir un sensor para poder obtener la orientación y la posición**

Al comenzar el proceso de diseño del traje mocap, surgieron cuestiones sobre qué tipo de sensores/dispositivos se tenían que emplear, qué variables de medición podrían ser convenientes, el costo del material y equipo al emplear ciertas tecnologías, etc.

Recopilación de tecnologías o variables que se consideraron que podían ser útiles: GPS (DGNSS, SBAS, RTK y PPP), presión atmosférica, ultrasonido, magnetómetro, giróscopo y acelerómetro. Muchas de estas tecnologías o variables se estudiaron y analizaron para ver la factibilidad de si un traje pudiese ser desarrollado con estas.

De hecho, para poder capturar el movimiento se puede recurrir a diversas técnicas que aprovechan diferentes leyes de la física en campos de estudio como la acústica, óptica, mecánica y radio-frecuencia.

Es posible conocer la orientación y posición de un objeto al emplear bocinas y micrófonos ultrasónicos, o también al emplear marcadores pasivos que son detectados por cámaras de vídeo. Otra forma para capturar el movimiento sería empleando sensores MARG (véase el Subtema 1.3.1 para una descripción de estos sensores), los cuales obtienen la dirección y magnitud de la aceleración, del campo magnético y del eje de giro, pudiendo así estimar la orientación absoluta y la posición relativa de un objeto mediante modelos matemáticos.

Luego del análisis, se concluyó que la mejor tecnología por precio, rendimiento y prestaciones es aquella asociada a los sensores MARG. Enunciaré brevemente los motivos por los cuales algunas otras tecnologías no fueron seleccionadas:

- La familia de tecnologías asociadas al GNSS, o erróneamente conocido como GPS, como lo son el DGNSS, SBAS, RTK y PPP, se descartaron porque, si bien se podrían lograr precisiones cartométricas bajo ciertas condiciones, la señal rebota dentro de lugares cerrados, y esto puede ocasionar errores en la estimación de la posición por 10 metros o más debido a su principio de funcionamiento. Otro factor es el sobre costo que conllevaría un traje con esta tecnología.
- Las tecnologías de triangulación de la posición por ultrasonido para la captura de movimiento pertenecen a un área de la ingeniería poco explorado y, por tanto, riesgoso para una residencia profesional. Además, las bocinas y micrófonos comerciales cubren un espectro del ultrasonido que no sobrepasa los 40KHz, un espectro de frecuencia muy pobre para la transmisión masiva de datos.

### **2.3.3. Principales ofertas de sensores MARG en el mundo**

Algunos de los distribuidores de componentes electrónicos de clase mundial como Digi-Key, SnapEDA y Mouser fueron consultados para buscar a los principales fabricantes de sensores MARG, esto con la finalidad de investigar los productos ofertados por estos fabricantes en sus respectivas páginas web, obtener los datasheets y comparar sus especificaciones técnicas.

Las principales empresas para la fabricación de sensores MARG son: TDK Corporation (antes Invensense), Bosch y STMicroelectronics. Quiero aclarar que no es la lista completa, pero sí son los que ofrecen los productos más destacados, según los criterios empleados por su servidor.

En la Tabla 2.2 se encuentran contenidas las especificaciones técnicas de los sensores MARG que se consideraron como los mejores por su servidor, tanto por sus especificaciones técnicas y por la facilidad de conseguirlos.

#### 2.3.4. Selección de los 2 sensores más aptos para el proyecto

##### Cantidad de ejes/GDL's necesarios para calcular la orientación de un sensor MPU o MARG

Con el fin de seleccionar la mejor combinación (acelerómetro-magnetómetro, acelerómetro-giróscopo, acelerómetro-magnetómetro-giróscopo). Se tabularán las características asociadas a la cantidad de ejes/GDL's<sup>4</sup> empleados (véase la Tabla 2.3).

Tabla 2.3: Comparación de características por cantidad de GDL's presentes. Recuperado de:  
(Freescale Semiconductor, 2015).

Característica	Solo Acel	Solo Mag	Solo Giro	Acel + Giro	Acel + Mag	Acel + Mag + Giro
Roll / Pitch / Tilt en grados	Sí	No	Sí	Sí	Sí	Sí
Yaw en grados	No	Sí	Sí	No	Sí	Sí
Velocidad angular en grados / seg	2 ejes virtuales	Solo Yaw	Sí	Sí	3 ejes virtuales	Sí

##### Compra por internet de los 2 sensores MARG más adecuados para testeo.

Tras investigar en plataformas online de comercio electrónico como Amazon, AliExpress, Ebay, entre otras, los mejores precios para varios modelos de sensores mencionados en la Tabla 2.2 se encontraron en AliExpress. Cabe mencionar que los sensores ya se encuentran montados en una PCB.

<sup>4</sup>Grados de libertad (o GDL) es una expresión que utilizan en inglés para caracterizar a los sensores IMU, MARG, etc. Es la traducción de la palabra de habla inglesa *DOF* (Degrees of Freedom)

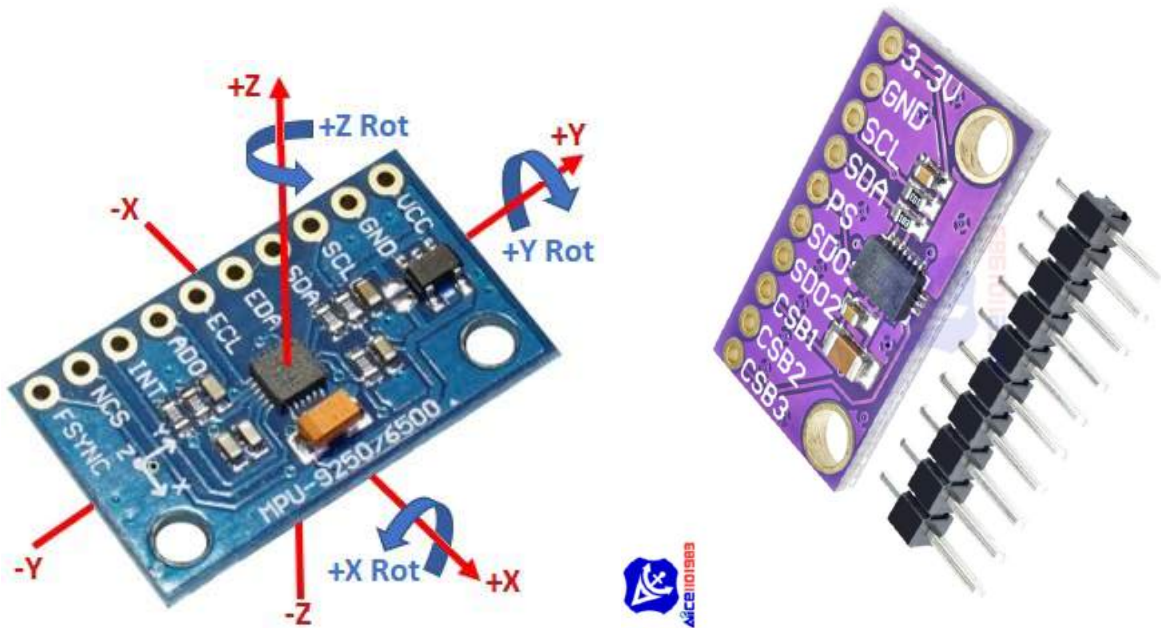
Tabla 2.2: Comparación de las especificaciones técnicas de diversos modelos de sensores

MARG

Sensor	Parámetro	Unidades	MPU9250	ICM20948	BMX055	L3GD20(gyr) & LSM303D(ac-mg)	LSM9DS0	LSM9DS1
ACCELEROMETER	Sensitivity tolerance	%	+3	+0.5	-	-	-	-
	Sensitivity tolerance/ Temperature	‰/°C	+0.026	+0.026	+0.03	+0.01	+1.5% (-40 a 85)	-
	Nonlinearity	%	+0.5	+0.5	+0.5	-	-	-
	Zero-G	mg	+60 0 +80	+25 0 +50	+80	+60	+60	+90
	Zero-G/ Temperature	mg/°C	+1.5	+0.80	+1	+0.5	+0.5	-
	Noise Spectral Density	ug/sqrt(Hz)	300	230	150	150	-	-
	ODR	Hz	4000	4500	2000(bandwidth)	Por definir	-	-
	Sensitivity tolerance	%	+3	+1.5	+1	-	-	-
	Sensitivity tolerance/ Temperature	‰/°C	+4% (-40 a +85)	+3% (-40 a 85)	+0.03	+2% (-40 a 85)	+3% (-40 a 85)	-
	Nonlinearity	%	+0.1	+0.1	+0.05	0.2	-	-
GYROSCOPE	Zero-G	‰/seg	+5	+5	+1	+10 +15 +75	+10 +15 +25	+30
	Zero-G/ Temperature	(‰/seg)/°C	+30°/seg(-40 a 85)	+0.05	+0.015	+0.03 0 +0.04	+0.05	-
	Noise Spectral Density	(‰/seg)/sqrt(Hz)	0.01	0.015	0.014	0.03	-	-
	ODR	Hz	8000	9000	2000	760	-	-
	Sensitivity tolerance	%	-	-	Gain error: +5	-	-	-
	Sensitivity tolerance/ Temperature	‰/°C	-	-	+0.01	+0.05	+3% (-40 a 85)	-
MAGNETOMETER	Nonlinearity	%	-	-	1	-	-	-
	Zero-G	uT	-	-	+40 0 +2	-	-	+100
	Zero-G/ Temperature	uT/°C	-	-	-	-	-	-
	Noise Spectral Density	uT/sqrt(Hz)	-	-	0.3 uT (High Accuracy mode)	0.5 uT RMS	-	-
	ODR	Hz	100	-	20 0 300	-	-	-
	Precio más bajo	-	57	117	77	75	172	125
GENERALES	SPI speed	Mhz	1 (20 ReadOnly)	7	10 (Vddio >1.6)	LSM303D = 20	10	10
	I2C speed	khz	400	400	400	400 (if ODR <228)	400	400

El sensor BMX055 fue, después del sensor MPU9250, el segundo mejor sensor de acuerdo a su relación calidad-precio, y por lo mismo fue un sensor que se compró y probó su funcionamiento para compararlo en la práctica con el MPU9250 (mismo ya se había adquirido tiempo atrás, previo al comienzo del desarrollo de este proyecto).

En Aliexpress se realizó la adquisición del sensor BMX055, el sensor se puede encontrar ya montado (al igual que con el MPU-9250) en una tarjeta de circuito impreso (PCB, por sus siglas en inglés), lo cual facilita la conexión con el sensor. Las tarjetas PCB de los 2 sensores que se pusieron a prueba se muestran en la Figura 2.2



(a) Ejes X, Y y Z para el sensor GY-9250 (MPU-9250). Recuperada de: (ProtoSupplies, s.f.).

(b) PCB del sensor BMX055. Recuperado de: (Diymore Alice1101983, s.f.).

Figura 2.2: Sensores MARG puestos a prueba durante el proyecto.

## 2.4. Pruebas de comunicación y funcionamiento: Arduino

### - sensores MARG

#### 2.4.1. Sensor MPU9250: consideraciones iniciales

La placa Breakout<sup>5</sup> GY-9250 contiene al sensor MPU-9250 en su interior, Esta placa es la que fue utilizada durante todas las pruebas y el desarrollo del proyecto.

Nota: En el presente proyecto al hacer mención del sensor MPU9250 durante el capítulo de Desarrollo o de Resultados, me refiero implícitamente a la placa GY-9250.

Es necesario saber que las mediciones que serán realizadas por el sensor MPU9250 serán acordes al esquema de referencia que se muestra en la Figura 2.2a; esta referencia es solidaria al sensor MPU9250, por tanto, si el sensor se rota  $90^\circ$  en  $Z$ , el sistema de la Figura 2.2a se rota  $90^\circ$  en  $Z$ . Las mediciones del sensor trabajan con un sistema de coordenadas cartesianas, como el de la Figura 1.13a.

Ahora para comprender los valores medidos que arroja el acelerómetro se presentará un ejemplo hipotético: supongamos que la gravedad apunta en la dirección del eje  $-Z$  solidario al sensor MPU9250 (de aquí se puede deducir que el sensor MPU9250 se encuentra orientado verticalmente, tal como en la Figura 2.2a); y por tanto el valor arrojado por el acelerómetro, bajo condiciones ideales (es decir, el acelerómetro está *perfectamente* calibrado), sería equivalente a  $\langle 0, 0, 9.809 \rangle \left[ \frac{m}{s^2} \right]$ . Observar que si la gravedad apunta en  $-Z$  del sensor MPU9250, esto hará que la medición del acelerómetro sea positiva en  $Z$ , o sea que las mediciones del acelerómetro necesitan un cambio de signo.

---

<sup>5</sup>Las placas Breakout cumplen con una función específica y no son de utilidad por si solas; por tanto, es necesario conectarlas, p. ej., a un microcontrolador

En cuanto al magnetómetro, el valor medido del sensor es positivo en algún eje (X, Y o Z) si ese eje coincide con el vector del campo magnético en dirección al norte magnético (se puede deducir entonces que las mediciones del sensor son algo así como una “brújula” que apuntan al polo norte magnético).

## 2.4.2. Conexiones eléctricas entre Arduino y el sensor MPU9250 para emplear el protocolo I2C

Para conocer los niveles lógicos con los que se puede trabajar con el sensor MPU9250, se revisó el esquemático de la placa breakout del sensor GY-9250, dicho esquemático se puede observar en la Figura 2.3.

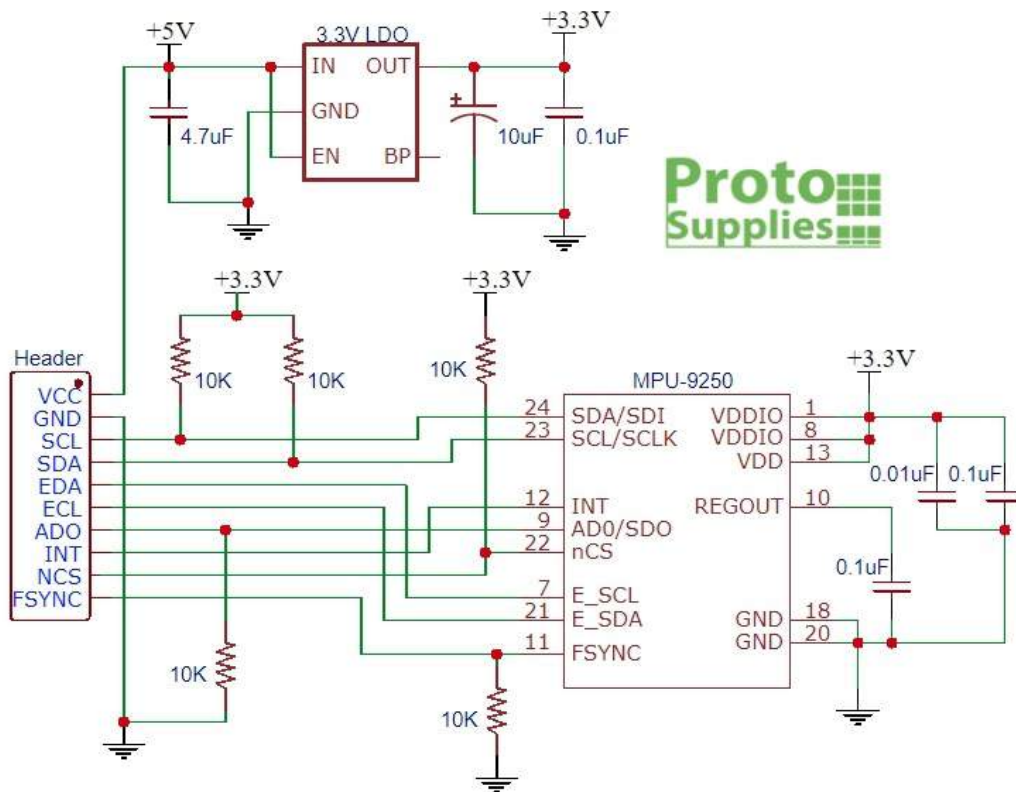


Figura 2.3: Esquemático de la placa breakout GY-9250, empleada en el proyecto. Recuperado de: (ProtoSupplies, s.f.).

La conexión empleada para leer los valores del sensor MPU9250 mediante el protocolo I2C (el protocolo I2C se describe en el Subtema 1.3.6) se muestra en la Figura 2.4, el cual está apegado a las conexiones recomendadas en la página 20 de la hoja de datos del Anexo A.1 (revisar la subsección de *Información de uso*). En la Figura 2.4a se muestra una representación del cableado eléctrico; mientras que en la Figura 2.4b se muestra el esquemático de dicha conexión.

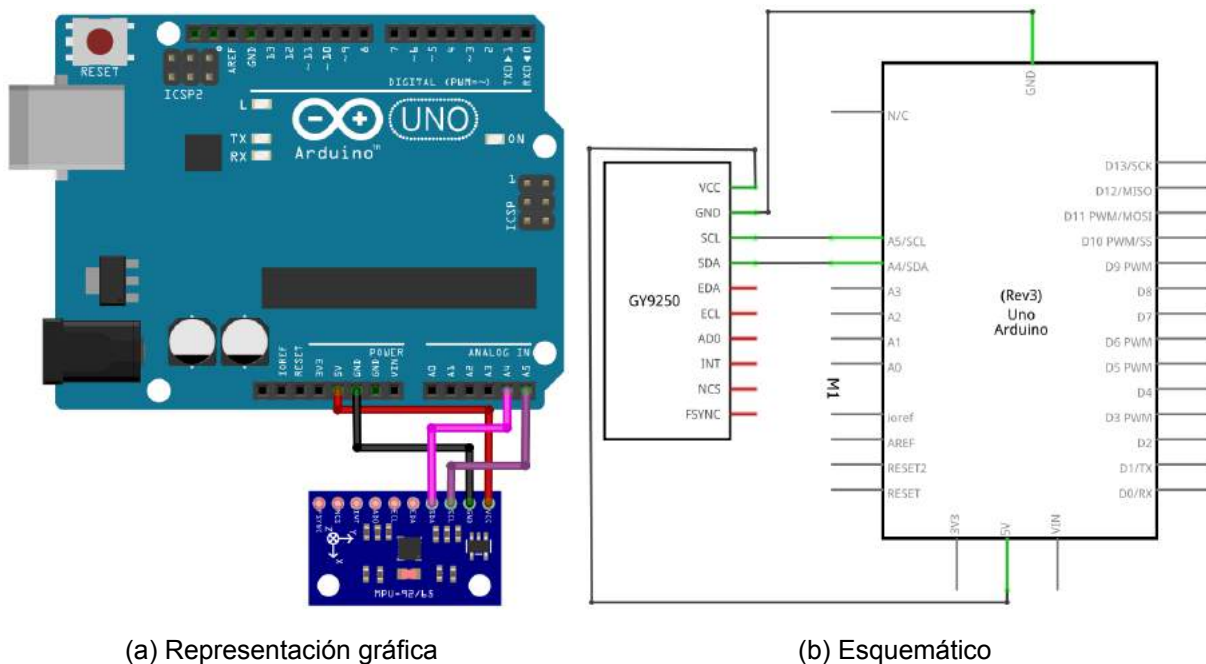


Figura 2.4: Conexiones eléctricas entre Arduino y el sensor MPU9250 para emplear el protocolo I2C

### 2.4.3. Primeras pruebas Arduino - MPU9250

Para poder trabajar con el sensor MPU9250 es necesario conocer de antemano toda la información sobre su funcionamiento, una pequeña porción de la información consultada para el proyecto se encuentra en el Anexo A, este anexo contiene extractos de los documentos originales, mismos que se describen brevemente en la Tabla 2.4.



Tabla 2.4: Hojas de datos empleadas para el uso del sensor MPU9250

Sección del anexo	Título de sección	No. páginas del documento original	Contenido
A.1	MPU-9250 Product Specification	42	Características eléctricas de 2 sensores (Ac-Gy), diagramas de conexión, funcionamiento, interfaz digital, etc.
A.2	MPU-9250 Register Map and Descriptions	55	Descripción de cada uno de los registros de los 3 sensores (Ac-Gy-Mg)
A.3	AK8963 datasheet	38	Características generales del magnetómetro, interfaz digital, diagramas, etc.
A.4	MPU-9250 Accelerometer, Gyroscope and Compass Self-Test Implementation	14	Procedimiento para validar si el sensor funciona debidamente
A.5	MPU Hardware Offset Registers: Application note	8	Procedimiento para calcular y modificar los registros de Offset internos

Al principio, fue necesario buscar código de terceros para analizar qué configuraciones le hacían al sensor antes de empezar a leer los registros de las mediciones del acelerómetro, giróscopo y magnetómetro. No obstante, el primer código en Arduino se elaboró pensando solo en probar la comunicación I2C con el sensor, el código está disponible en GitHub, en el siguiente archivo: **ARDUINO CODE / MPU9250 / PrimerPrueba / PrimerPrueba.ino**. Como resultado de ejecutar este código se obtiene lo que se muestra en la Figura 3.1

#### 2.4.4. Pruebas de comportamiento de los sensores MARG a las perturbaciones de entrada

Para saber si el sensor se encontraba en buen estado, se pueden hacer sencillas pruebas a cada uno de los ejes X, Y y Z, para lo cual se deben observar las marcas que vienen impresas en la PCB del sensor (ver Figura 2.2, o bien la página 38 del Anexo A.1: subsección de *Ensamble* para ver los ejes del sensor).

## Procedimiento para la prueba Self Test del sensor MPU9250

Para conocer si un sensor se encuentra en buen estado o si está defectuoso, se tiene que realizar un procedimiento conocido como Self-Test, mismo que se puede encontrar en el Anexo A.4. Se considera que el sensor pasa la prueba de Self-Test cuando los valores arrojados al ejecutar el código **ARDUINO CODE / MPU9250 / Self-Test / Self-Test.ino** (mismo que solo se encuentra en [Github](#)<sup>6</sup> por cuestiones de espacio) son inferiores a los valores que se encuentran en el Anexo A.4

### 2.4.5. Código para resetear y setear el sensor MPU9250 cuando se bloquee

En ocasiones, el dispositivo maestro SPI o I2C, dígame Arduino o Raspberry Pi, identifica ciertos patrones irregulares en las mediciones proporcionadas por el sensor MPU-9250, para corregir esas irregularidades es necesario mandar a reset al acelerómetro, el giróscopo y el magnetómetro. La ubicación y el valor que se debe asignar a los registros para resetear al sensor MPU-9250 se mencionan en la Tabla 2.5 (esto se puede corroborar accediendo a la hoja de datos original del Anexo A.2).

Tabla 2.5: Registros necesarios para resetear el sensor MPU9250.

Sensor (dirección I2C)	Registros	Ubicación	Valor	Motivo
MPU9250_ADDR (0x68)	ACC_GYRO_RESET	0x6B	0x80	Reset Accel-Gyro
MAG_ADDRESS (0x0C)	MAG_CNTL_2	0x0B	0x01	Soft Reset Magnetometer

Una vez que ha transcurrido cierto tiempo, luego de mandar a reset al sensor MARG, es necesario volver a introducir en ciertos registros algunos valores de configuración. La ubicación y los valores que se deben asignar a estos registros se cotejaron en la Tabla 2.6.

---

<sup>6</sup>[https://github.com/alejandroivan10/ResidenciaProfesional\\_IvanAlejandroMunozVera](https://github.com/alejandroivan10/ResidenciaProfesional_IvanAlejandroMunozVera)

Tabla 2.6: Registros necesarios para setear el sensor MPU9250.

Sensor (dirección I2C)	Registros	Ubicación	Valor	Motivo
MPU9250_ADDR (0x68)	28	28	ACC_FSCALE_2_G	Acc_FScale = $\pm 2g(19.62m/s^2)$
MPU9250_ADDR (0x68)	ACC_CONFIG_1+1	0x1D	0x06	Acc_DLPF = 5Hz
MPU9250_ADDR (0x68)	27	27	GYRO_FSCALE_250_DPS	Gyro_FScale = $\pm 250grados/seg$
MPU9250_ADDR (0x68)	ACC_GYRO_CONFIG	0x1A	0x02	Gyro_DLPF = 92 Hz
MPU9250_ADDR (0x68)	MPU_INT_PIN_CFG	0x37	0x02	Registro INT_PIN_CFG
MAG_ADDRESS (0x0C)	MAG_CNTL_1	0x0A	MAG_MODE_2   MAG_FSCALE_16_bit	16-bit Mag mode 2 (ODR=100Hz)

Otros registros necesarios para trabajar con el sensor MPU9250 son aquellos que tienen una relación directa con las mediciones, por lo que estos deben ser leídos para conocer el estado actual del sensor.

## 2.5. Pruebas de comunicación y funcionamiento: Matlab - Arduino - Sensores MARG

### 2.5.1. Comunicación USB entre Matlab y Arduino

Se probó la comunicación USB entre Arduino y Matlab (ambos IDE's se describen en el Subtema 1.3.9) para asegurarse que existía una buena transferencia de información. La información enviada por Arduino mediante USB consistía en 10 valores (3 del Acelerómetro, 3 del magnetómetro, 3 del giroscopio y 1 uno más que contenía el tiempo transcurrido desde el arranque del programa en Arduino), los datos tienen el siguiente formato (ver Figura 3.3):

$Ac=[\ %.3f\ \%.3f\ \%.3f]\ Gy=[\ %.3f\ \%.3f\ \%.3f]\ Mg=[\ \%.3f\ \%.3f\ \%.3f]\ \%.3f$

Donde:  $\%.3f$  se debe remplazar por un valor de punto flotante que es truncado justo en el 3er decimal. Tanto el acelerómetro, giróscopo y magnetómetro arrojan 3 valores.

De lo anterior, el primer valor es el eje en X, el segundo en Y y el tercero en Z, el último valor es igual al valor retornado de la ejecución de la función *unsigned long micros()* en Arduino.

Es importante mencionar que el código que fue empleado en Arduino para probar todos los códigos en Matlab fue: **ARDUINO CODE / FINAL\_GENERAL / FINAL\_GENERAL.ino** (mismo que se encuentra en [GitHub](#))

Una vez leída la trama de datos en Matlab, hay que descomponerla, extraer los datos y asignarlos a la posición X, Y o Z según el sensor que les corresponde (Acel-Giro-Mag).

### 2.5.2. Graficación de vectores 3D en Matlab con los datos enviados por Arduino

Con la finalidad de analizar patrones de comportamiento del acelerómetro y del magnetómetro, se graficaron los 3 ejes (X, Y y Z) de ambos sensores en Matlab. Este programa se encuentra en el archivo: **MATLAB / Pruebas / MPU9250\_Acc\_Gyr\_Mag\_3D.m**, mismo que se puede consultar en el [link](#) de Github.

El resultado de ejecutar este código se puede observar en la Figura 3.4, donde el vector rojo es la gráfica de las mediciones del acelerómetro y los 2 vectores amarillos son fruto de las mediciones del magnetómetro (un vector es el valor medido tal cual del sensor, y el otro vector equivale a la dirección estimada del norte magnético).

#### Algoritmos de calibración automática bajo condiciones reales y simuladas en Matlab

Se empleó Matlab para probar el funcionamiento de diversos algoritmos esenciales para este proyecto, como lo son la calibración automática y el filtro de Madgwick. Para realizar la calibración automática fue necesario emplear un set de datos capturados por el sensor y otro creado aleatoriamente. En cuanto al set aleatorio, se tuvo en consideración la Ecuación (2.1):

$$\left(\frac{x - h_x}{a}\right)^2 + \left(\frac{y - h_y}{b}\right)^2 + \left(\frac{z - h_z}{c}\right)^2 = 1 \quad (2.1)$$

La Ecuación (2.1) corresponde a un elipsoide en  $\mathbb{R}^3$ , y los valores por calibrar son  $h_x$ ,  $h_y$ ,  $h_z$ ,  $a$ ,  $b$  y  $c$ .

Ambos sets de datos se usaron para probar 2 algoritmos de calibración automática, en ambos se busca resolver un sistema de ecuaciones, uno de ellos es lineal (el cual es el más óptimo, y es el que se escogió para el proyecto); y el otro es un algoritmo no lineal, el método Newton-Raphson, el cual se ejecutaba de forma iterativa, y tenía un tamaño de paso para actualizar los valores por calibrar.

Ambos algoritmos se pueden encontrar en el link de GitHub antes mencionado. Y la rutas son: **MATLAB / Pruebas / Calibracion\_NewtonRaphson (Unused)**, el cual contiene 3 códigos independientes entre si, los cuales implementan el método de Newton-Raphson; y el archivo **MATLAB / ArduinoUnrealUDP / Madgwick / Calibración\_RegresionLineal / RegresiLinealMultip.m**, que contiene 2 partes que implementan la regresión lineal: un código sin comentar, el cual es una función que se debe invocar por algún otro código; y un código oculto en los comentarios, el cual se puede ejecutar por cuenta propia, y sirvió para testear que el algoritmo se encontraba correctamente programado.

El algoritmo que ha sido desarrollado bajo el método de Newton-Raphson solo se probó, y si bien dio resultados satisfactorios, se optó por emplear aquel algoritmo que funciona bajo ecuaciones propias de una regresión lineal. Por tanto, a partir de ahora, cuando se mencione *algoritmo de calibración automática* se hace alusión a este último método por regresión, salvo se mencione lo contrario.

El algoritmo de la calibración automática consta de 4 fases:

1. Validación de los datos: Consiste en analizar si las mediciones han sido capturadas correctamente y, de ser así, discriminar aquella información redundante.
2. Captura de los datos: Consiste en almacenar en memoria las mediciones ya validadas de los 3 ejes (X, Y y Z) del acelerómetro, giroscopio y magnetómetro (9 datos

en total). El algoritmo espera iteración tras iteración de la función `void loop()` hasta captura “n” filas (n es igual a “20” para fines de este proyecto), y cada fila contiene los 9 datos antes mencionados.

3. Obtención de los parámetros de corrección (calibración automática): Consiste en implementar una regresión lineal múltiple al set de datos almacenados y, después, validar los nuevos parámetros obtenidos de la regresión (esto quiere decir que los nuevos parámetros se evalúan, y si el resultado está por debajo de la tolerancia máxima, se consideran como aceptables).
4. Actualizar los parámetros de calibración: Los valores se actualizan mediante un filtro complementario (es decir,  $x[i+1] = a * x[i] + b * x_{\text{Regresión\_Lineal}}$ . Donde:  $a + b = 1$ , i es igual al número de iteración).

El resultado de invocar la función *RegresiLinealMultip(raw)* en el código **MATLAB / ArduinoUnrealUDP / Madgwick / Madgwick\_PLUS\_Calibration.m** se muestra en la Figura 3.5.

### 2.5.3. Filtro Madgwick

Para la obtención de la orientación se empleó el código del archivo **MATLAB / ArduinoUnrealUDP / Madgwick / Madgwick.m**, el cual contiene pequeñas correcciones al filtro de Madgwick (ver el Subtema 1.3.5), pero la esencia se mantiene. Este filtro es la columna vertebral de todo el proyecto, ya que su buen desempeño y poco consumo de recursos han favorecido al desarrollo del proyecto.

Las gráficas obtenidas por Matlab, luego de implementar el filtro de Madgwick, se pueden observar en la Figura 3.6 y 3.7. Estas gráficas son fruto del uso de la función de Matlab *plot3()*, la cual permite imprimir vectores desde alguna coordenada inicial a una final, además, permite ajustar el color y el grosor con que el vector se graficará.

Para visualizar tanto la estimación de la orientación dada por la ejecución del código *Madgwick.m*, y también visualizar la orientación real del sensor, se optó por tomar fotografías, en las cuales se sobrepuso el sensor MPU9250 al monitor del ordenador. Los resultados se muestran en la Figura 3.8

### **Macros en el código del sensor MPU9250 en Arduino para escoger un protocolo de comunicación (I2C o SPI)**

Con la finalidad de escribir poco código para probar el protocolo de comunicación SPI (para ver una breve descripción del protocolo SPI ver el Subtema 1.3.6) entre Arduino y los sensores MPU9250 y BMX055, se agregaron macros a los códigos principales de ambos sensores (disponibles en GitHub), de forma que en estos códigos principales solo basta con agregar una macro y quitar otra para activar el protocolo SPI, o bien el protocolo I2C. Estas macros son `#define SPI_PROTOCOL` (activa el protocolo SPI) y `#define I2C_PROTOCOL` (activa el protocolo I2C). Solo debe declararse una de estas dos macros al mismo tiempo en el código.

Las conexiones para conectar el sensor MPU9250 y Arduino mediante el protocolo SPI son las que se muestran en la Figura 2.5 y 2.6, estas conexiones están basadas en el diagrama mostrado en la página 38 de la hoja de datos del Anexo A.1. Aquellas conexiones para el sensor BMX055 se omiten por razones de espacio, y por no ser el sensor empleado para las etapas posteriores del proyecto, pero se pueden encontrar por internet o en la hoja de datos de este.

#### **2.5.4. Selección de un modelo de sensor MARG para que este sea empleado en todo el traje**

Se realizaron pruebas de velocidad de comunicación mediante Arduino para conocer, de forma práctica, la frecuencia máxima de la señal del reloj CSK o CLK capaz de ser tolerada por los sensores BMX055 y MPU9250. La velocidad de transmisión SPI o I2C es uno

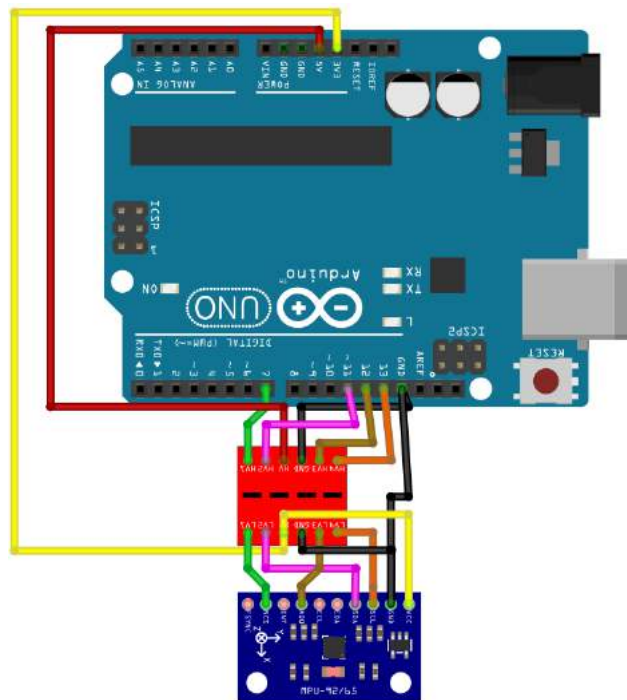


Figura 2.5: Conexión SPI entre Arduino y el sensor MPU9250

de los factores para seleccionar el sensor, otros factores fueron, la cantidad de líneas eléctricas necesarias para la comunicación con los sensores, el precio en el mercado, y las especificaciones técnicas que se muestran en la Tabla 2.2.

El sensor escogido fue el MPU9250, ya que solo requiere de un cable adicional por cada nuevo sensor en la red (en comparación con los 3 del BMX055). Además, la lectura y escritura de datos en el MPU9250 fue mucho más rápida que en el BMX055 (en el sensor BMX055 no pude hacer una lectura múltiple, cuando en el datasheet se supone que si es posible efectuarla).

Al principio, Aliexpress fue la tienda online seleccionada para comprar los 18 sensores MPU9250, ya que se podían encontrar precios alrededor de los \$70 MXN (sin considerar el envío). Desafortunadamente, los sensores aquí comprados estaban dañados y, debido al vencimiento de la garantía, ya no se pudieron devolver.



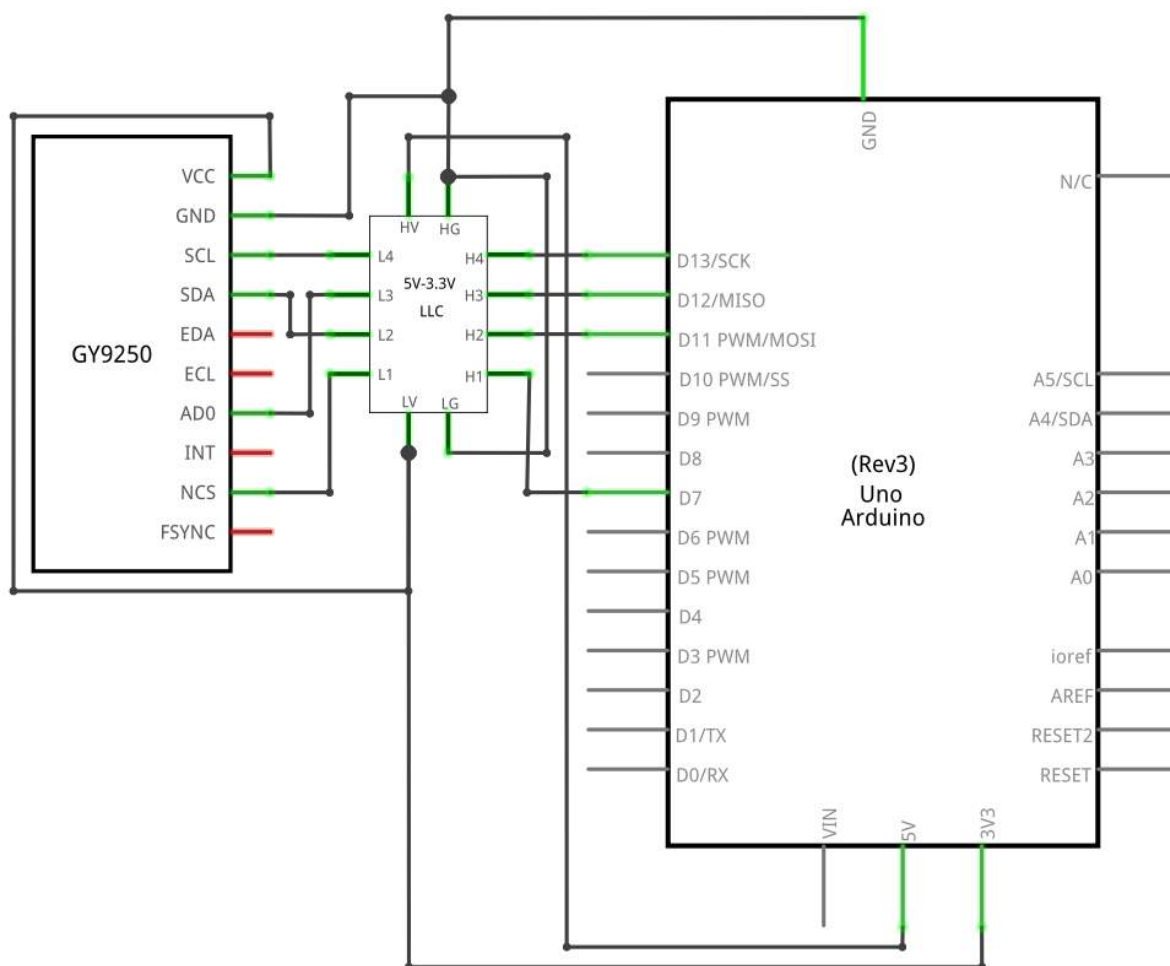


Figura 2.6: Esquemático de conexiones entre Arduino y el sensor MPU9250 para el protocolo SPI

Posteriormente, se recurrió a comprar en Mercado Libre un total de 19 sensores a 4 proveedores, de los cuales en 8 sensores se demostró, mediante vídeos enviados a 2 de los proveedores (los vídeos están disponibles en [YouTube](https://www.youtube.com/playlist?list=PL_8Avt9po9eW3ZPi79-Xft01zYMVcPYU5)<sup>7</sup>), que estos no funcionaban como se mencionaba en su página web. Esto obligo a estos 2 proveedores que me enviaron estos sensores a tener que “devolver” el dinero de la compra.

<sup>7</sup>[https://www.youtube.com/playlist?list=PL\\_8Avt9po9eW3ZPi79-Xft01zYMVcPYU5](https://www.youtube.com/playlist?list=PL_8Avt9po9eW3ZPi79-Xft01zYMVcPYU5)

## 2.6. Pruebas de comunicación y funcionamiento: Unreal - Matlab - Arduino - Mpu9250

### 2.6.1. Consideraciones iniciales para Unreal Engine

Con la intención de probar la comunicación entre todo el traje y un entorno virtual, se escogió la versión 4.26.1 de Unreal para el desarrollo del proyecto (ver Figura 2.7).

Ahora, para poder crear un videojuego en Unreal Editor, primero hay que crear un proyecto (como en todo programa), en mi caso empecé por utilizar la plantilla de 3ra persona (esta incluye por default un personaje y un entorno con el cual interactuar), y por escoger Blueprint como lenguaje de programación. Para más información de Unreal se puede leer el Subtema 1.3.9.

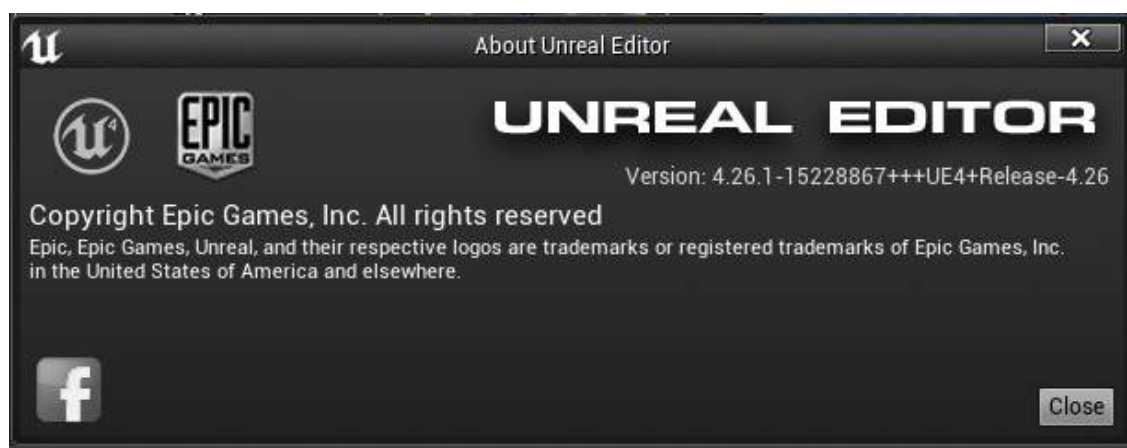


Figura 2.7: Versión del Unreal Editor empleado para el presente proyecto

La interfaz principal de Unreal Engine es la que se muestra en la Figura 2.8, esta consta, a grandes rasgos, de los siguientes paneles: Viewport (ubicada al centro superior, es donde se aprecia visualmente el videojuego), el Content Browser (inferior izquierda, funciona como explorador de archivos del proyecto), World Outliner (superior derecha, muestra a todos los actores que están en la escena) y el Details Panel (inferior derecha, es donde

se muestran todos los atributos y demás del actor que se encuentre actualmente seleccionado del World Outliner).



Figura 2.8: Interfaz principal del Unreal Editor 4.26. Se pueden apreciar contenidos, objetos, detalles, etc., empleados en el presente proyecto

Echando un vistazo al *Content Browser* (véase la Figura 2.9), podremos encontrar las carpetas que más se van a mencionar en el presente proyecto: *Content / ThirdPersonBP / Blueprints*, *Content / SpaceShips*, *Content / Animations* y *Content / Character / Mesh*.

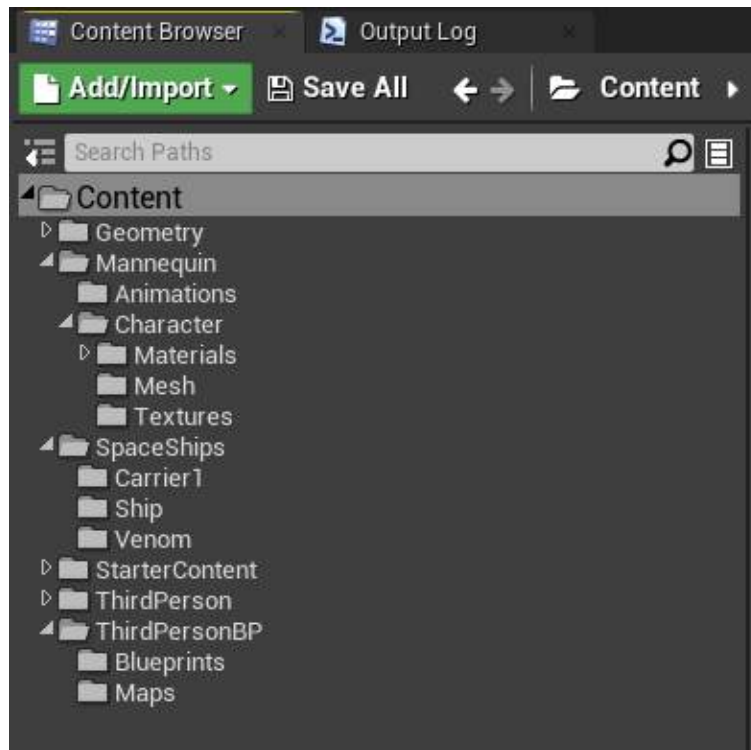


Figura 2.9: Content Browser: Carpetas empleadas para el presente proyecto. Dentro de las carpetas se encuentran las animaciones, materiales, texturas, blueprints, etc., necesarios para el desarrollo del videojuego

### 2.6.2. Plugins Unreal Editor

Los 3 plugins útiles para el proyecto: UE4Duino, socketio-client-ue4 y udp-ue4 son proyectos Open Source, y se puede consultar su código fuente por medio de GitHub. En la Tabla 2.7 se puede encontrar la información necesaria para ahondar más en su funcionamiento.

Tabla 2.7: Plugins de Unreal útiles para el proyecto

Plugin	Dueño del repositorio	link	Descripción
udp-ue4	getnamo (Jan Kaniewski)	<a href="https://github.com/getnamo/udp-ue4">https://github.com/getnamo/udp-ue4</a>	Wrapper conveniente de un componente UDP para un actor en UE4
socketio-client-ue4	getnamo (Jan Kaniewski)	<a href="https://github.com/getnamo/socketio-client-ue4">https://github.com/getnamo/socketio-client-ue4</a>	Biblioteca de comunicación bidireccional en tiempo real de alto rendimiento para UE4
UE4Duino	RVillani (Rodrigo Villani)	<a href="https://github.com/RVillani/UE4Duino">https://github.com/RVillani/UE4Duino</a>	Plugin para UE4 para la comunicación con Arduino por un puerto COM en Windows

Con el Plugin [udp-ue4](#) se puede añadir un componente UDP a un personaje. Este componente UDP es la interfaz de comunicación entre la RPi y el videojuego en Unreal, de forma que Unreal podrá recibir y enviar datagramas UDP desde y hacia las direcciones y puertos especificados.

El plugin [socketio-client-ue4](#) es, entre otras cosas, una librería con funciones pensadas para efectuar comunicación bidireccional. Un ejemplo es la función que puede convertir un arreglo de *Bytes* en *String*.

### 2.6.3. Lenguajes de programación para Unreal Engine

Para programar un videojuego en Unreal se pueden emplear 2 lenguajes de programación: C++ y Blueprints. C++ al ser un lenguaje más cercano al código máquina que los Blueprints, es, por tanto, un lenguaje con el que se pueden compilar códigos que aprovechan los recursos de forma más eficiente en tiempo de ejecución.

Para programar en C++ es necesario instalar Microsoft Visual Studio (este IDE se describe en el Subtema 1.3.9) y un conjunto de paquetes para Unreal Engine que podemos seleccionar al momento de ejecutar el instalador de Microsoft. Una ventaja muy importan-

te de hacer videojuegos con C++ es que, al ser el motor de Unreal un proyecto de código abierto (Open Source), podemos agregar, modificar o eliminar algunas funcionalidades para que el videojuego tenga el comportamiento deseado.

El otro lenguaje para programar Unreal son los Blueprints, los cuales son una alternativa para aquellos que se inician en el desarrollo de los videojuegos. Los blueprints son un lenguaje de programación visual en el cual se interconectan los nodos de bloques funcionales (bloques que también se les conoce como Blueprints porque son la base de este lenguaje).

Cabe mencionar que en el presente proyecto se han empleado ambos lenguajes de programación. De hecho, el proyecto comenzó a desarrollarse mediante Blueprints; pero debido a la necesidad de tener que compilar ciertos plugins para Unreal Editor, se tuvo que descargar todo el código fuente de Unreal Engine e incorporarle el código C++ de los plugins. Por tanto, el presente proyecto es híbrido, y se puede continuar su desarrollo con cualquiera de los 2 lenguajes de programación, ambos conviven dentro del proyecto (los cambios hechos en un lenguaje se pueden sincronizar en el otro).

Es necesario configurar el proyecto para que se pueda programar con C++ y Blueprints al mismo tiempo, también se deben configurar los plugins *Socketio-client-ue4* y *udp-ue4*.

#### **2.6.4. Microsoft Visual Studio**

Microsoft Visual Studio fue importante para el proyecto, ya que para añadirle funcionalidades UDP a un personaje se necesita emplear el código fuente tanto de los plugins como el de Unreal Engine. En la interfaz de Visual Studio se pueden ver los códigos de los plugins y el de Unreal, mismos que se pueden editar para añadir más funcionalidades (véase la Figura 2.10).

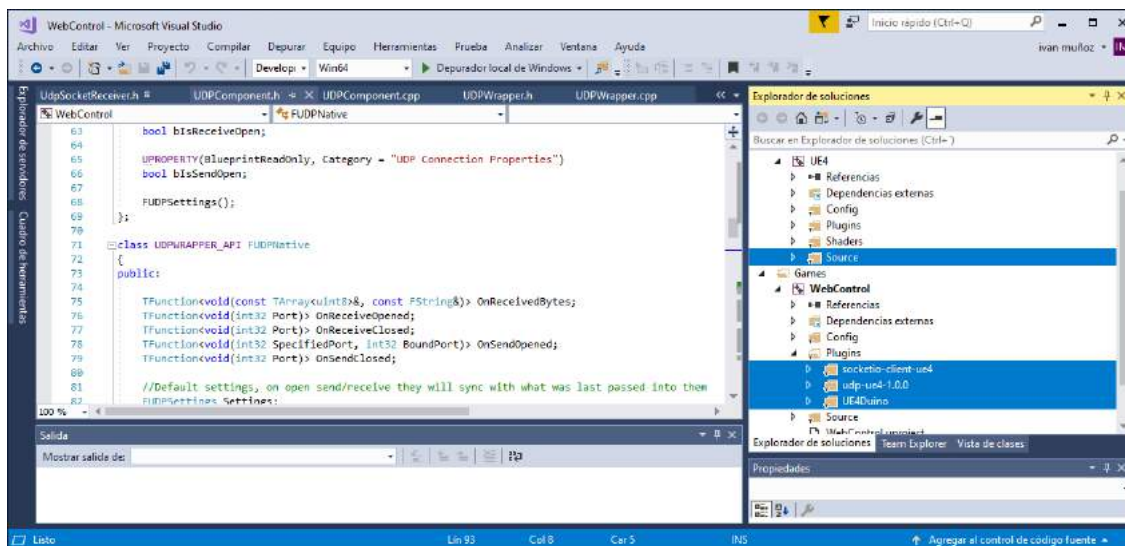


Figura 2.10: Interfaz principal de Microsoft Visual Studio

La estructura de los archivos para desarrollar el videojuego en C++ se muestra en la Figura 2.11. En la parte superior de la figura se observan las carpetas del código fuente de Unreal (carpeta *Engine / UE4 / Source*); y en la parte inferior, las carpetas del código específico para cada proyecto, en este caso contiene una carpeta *Games / WebControl / Plugins* (dentro de ella ya están los plugins instalados correctamente para el proyecto) y, más abajo, contiene una carpeta con la ruta *Games / WebControl / Source* donde se puede acceder al código fuente para modificar y construir el videojuego empleando C++.

### 2.6.5. Protocolos de comunicación entre Unreal y Matlab

Como ya se mencionó, el traje de mocap debe ser inalámbrico y, por tanto, se debe usar un protocolo para transmitir una gran cantidad de datos a largas distancias, que esté estandarizado, y que sea accesible y robusto. Claro está que el mejor protocolo para este fin pertenece a la familia IEEE802.11, mejor conocida como Wi-Fi (ver Subtema 1.3.7). El Wi-Fi solo puede cubrir los requerimientos del proyecto que pertenecen a la capa de enlace del modelo TCP/IP (para una descripción del modelo TCP/IP ver el Subtema 1.3.7), tal como se muestra en la Figura 1.23a.

Ahora falta escoger el protocolo de transporte y el de red, y para el de aplicación no se requiere alguno en específico, este podría ser personalizado por su servidor para los fines que convengan en este proyecto. Para el de transporte se escogió el protocolo UDP (ver el Subtema 1.3.7), ya que al no ser orientado a la conexión permite realizar streaming de datos; y para el protocolo de red se empleó el protocolo IP (ver Subtema 1.3.7)

### 2.6.6. Comunicación entre Unreal y Matlab

Para emplear la comunicación UDP en Matlab es muy sencillo, un ejemplo de trabajo mínimo (MWE, por sus siglas en inglés) se puede apreciar en el código contenido en el archivo **MATLAB / Pruebas / UDP\_test.m**, el cual es capaz de leer datagramas UDP recibidos y enviar los datagramas escritos en Matlab.

Para la comunicación UDP en Unreal se requiere de un componente UDP. Para ello, estando dentro de la ventana de un *Actor*, *Pawn* o *Character*, se hace clic en + *Add Component* (ver Figura 2.12), se busca un componente UDP en la lista desplegable y se inserta. Al hacer clic sobre este componente, el panel de detalles (ubicado a la derecha de la interfaz) mostrará lo que se aprecia en la Figura 2.13.

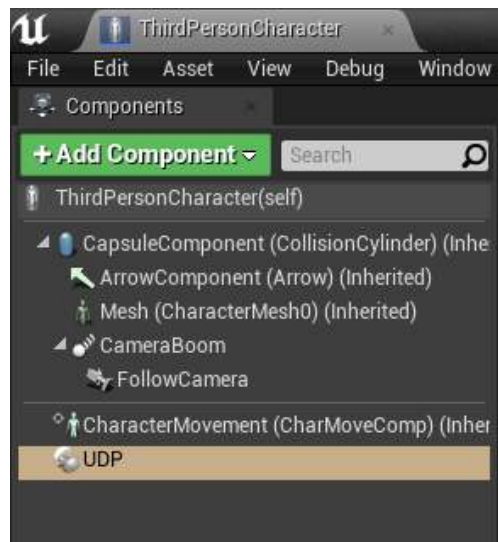
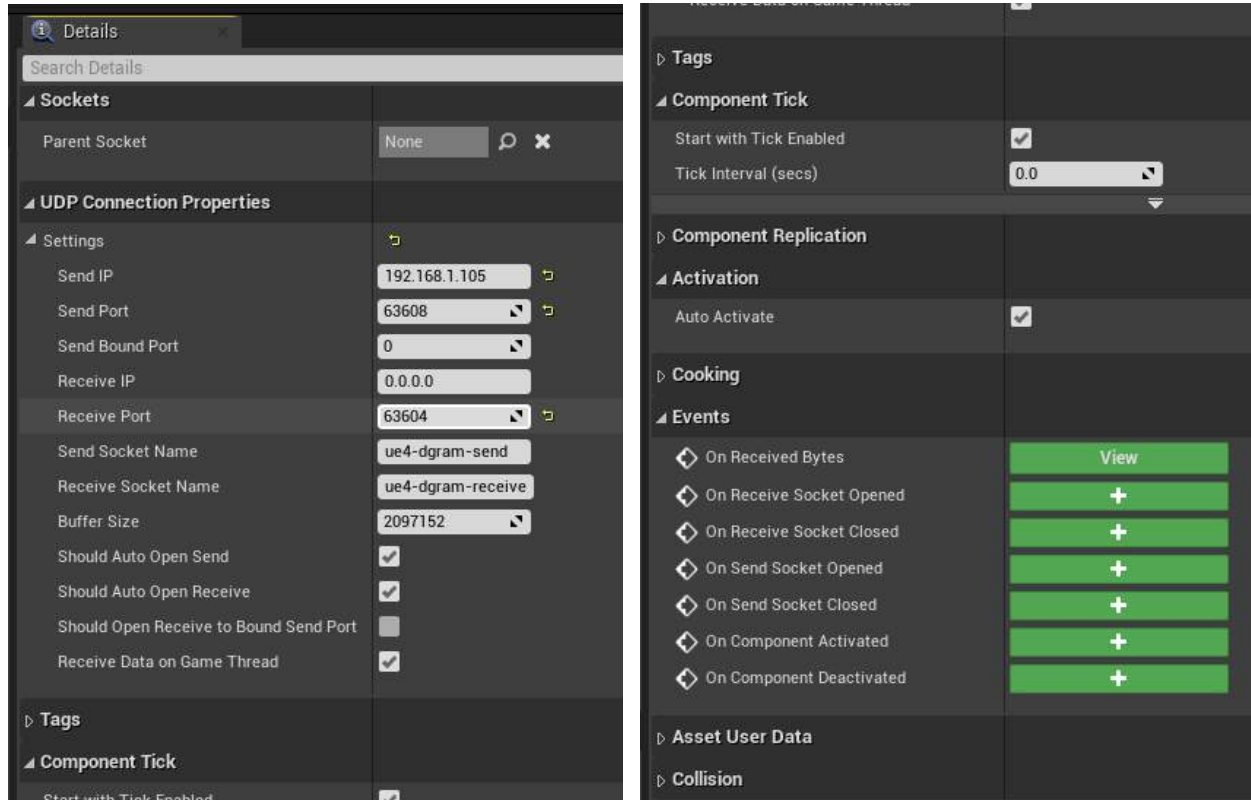


Figura 2.12: Lista de componentes que pertenecen al ThirdPersonCharacter



En el panel de la Figura 2.13a se introduce el valor de la dirección IP y del puerto (necesarios para los datagramas de la Figura 1.24 y 1.25), tanto de la RPi como de la computadora que ejecuta el videojuego, de forma que, cuando el videojuego reciba un paquete UDP se ejecutará el evento *OnReceivedBytes* que se muestra en la Figura 2.13b.



(a) Parte #1: Propiedades de conexión UDP

(b) Parte #2: Eventos del componente UDP

Figura 2.13: Panel de configuración del componente UDP

Una vez llegado a este punto, la comunicación UDP ya está configurada, y ya se pueden utilizar las “funcionalidades UDP” dentro de Unreal.

Luego de poner en práctica lo mencionado hasta este punto del proyecto, además, de ciertos temas que se mencionarán a continuación, se presenta en el Capítulo 3 la Figura 3.9 y, también, 2 vídeos en YouTube del funcionamiento de un sensor MPU9250, un Arduino y 2 computadoras (una ejecutando Matlab; y la otra, un videojuego en Unreal).

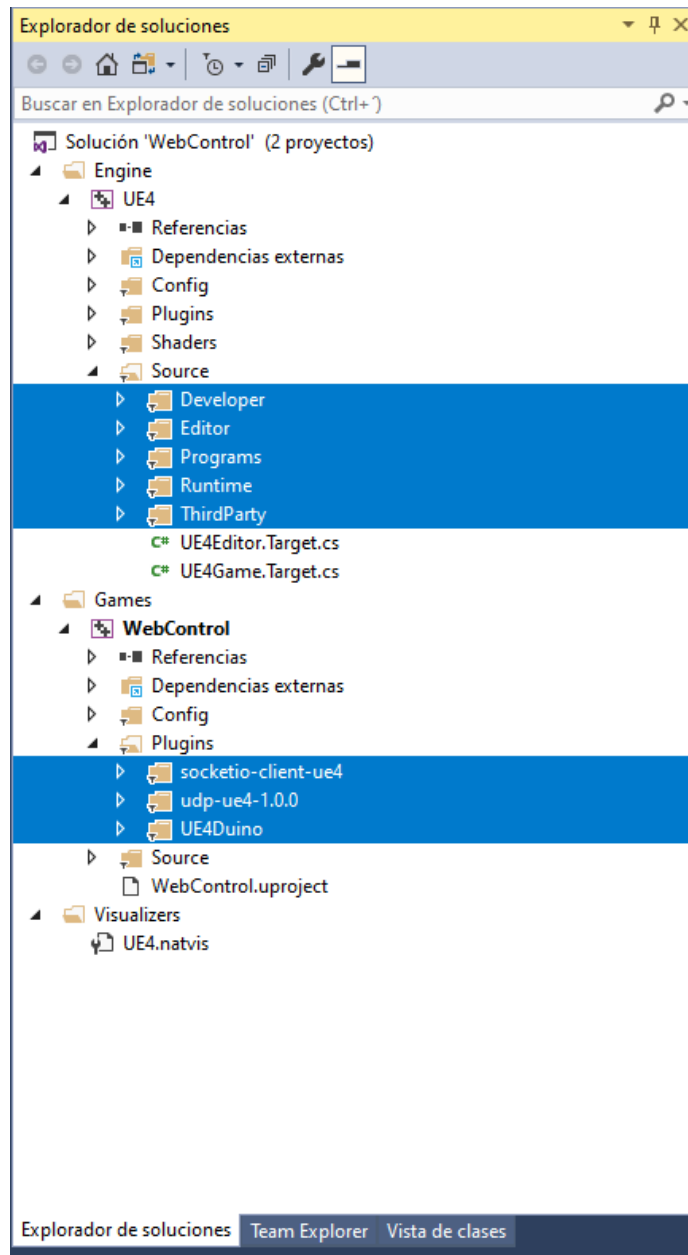


Figura 2.11: Explorador de archivos del proyecto de Unreal en Microsoft Visual

## 2.7. Pruebas de comunicación y funcionamiento: Unreal - Raspberry Pi - Mpu9250

### 2.7.1. Conexión remota entre un ordenador y la Raspberry Pi 3B

Lo primero que se debe conocer para establecer una conexión/sesión con la RPi es la dirección IP de la misma. Y para esto, hay que conectar la Raspberry Pi a una red.

Para poder establecer una conexión a una red de forma automática al encender la Raspberry Pi 3B, se debe conocer el SSID y la contraseña de un router inalámbrico. Posteriormente, se debe sustraer la tarjeta SD de la RPi y configurar la información anterior en los archivos del sistema (dicho procedimiento no se detalla en este informe). Para más información de la RPi se puede consultar el Subtema 1.3.8.

Advanced IP Scanner es un programa que nos permite escanear que dispositivos están conectados dentro de una misma red (ver Figura 2.14).

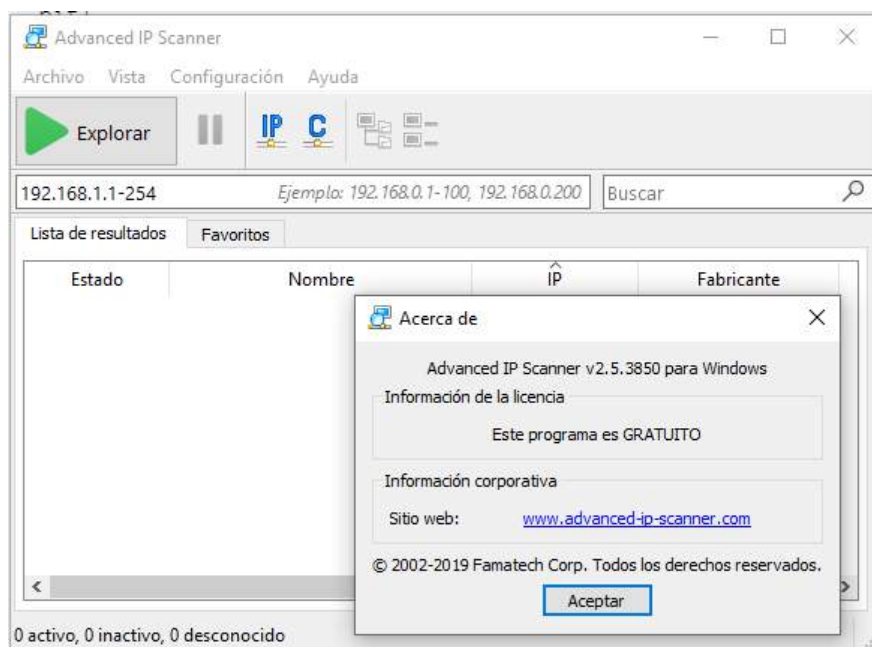


Figura 2.14: Programa Advanced IP Scanner empleado para buscar direcciones IP

Se empleó Vnc viewer del lado del cliente (computadora de escritorio con Windows) y Vnc Server del lado del servidor (Raspberry Pi 3B) para establecer un escritorio remoto, y poder controlar completamente a la Raspberry Pi de forma “remota” (ver Figura 2.15). Para utilizar este programa es necesario conocer 3 datos del servidor: su dirección IP, un usuario (por defecto: “pi”) y contraseña (por defecto: “raspberry”).

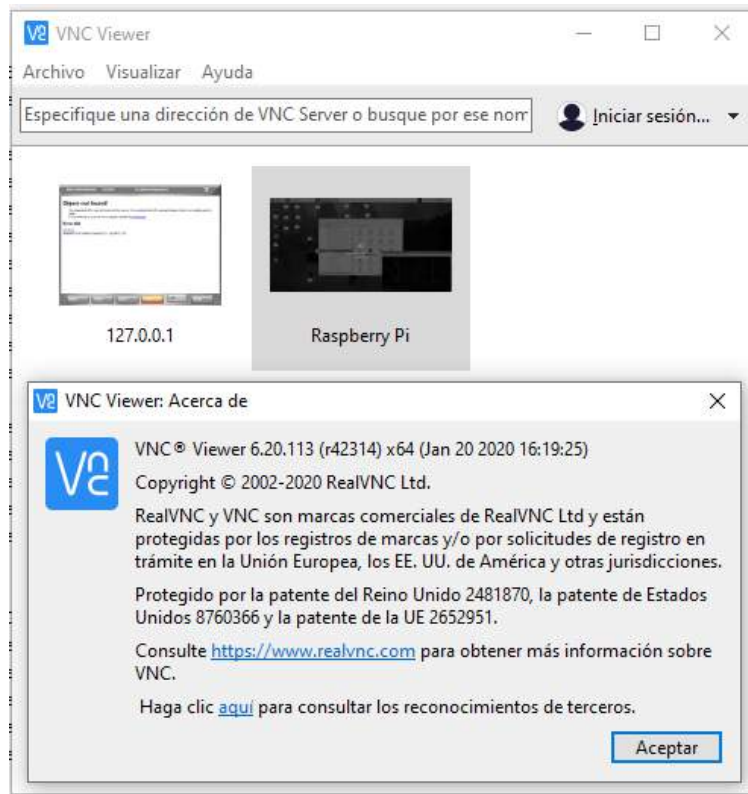


Figura 2.15: Programa Vnc Viewer empleado para establecer un escritorio remoto con la Rpi

De igual forma, también se empleó PuTTY para establecer una conexión SSH<sup>8</sup>. Para trabajar con SSH es necesario conocer los mismos 3 datos que con Vnc Viewer (ver Figura 2.16), y saber trabajar con la consola de comandos del sistema operativo del servidor al que estamos accediendo “remotamente”.

<sup>8</sup>Secure Shell (SSH, por sus siglas en inglés) es el nombre de un protocolo cuya función es establecer un acceso remoto a un servidor mediante una comunicación cifrada. Por defecto, el puerto TCP asignado para SSH es el 22

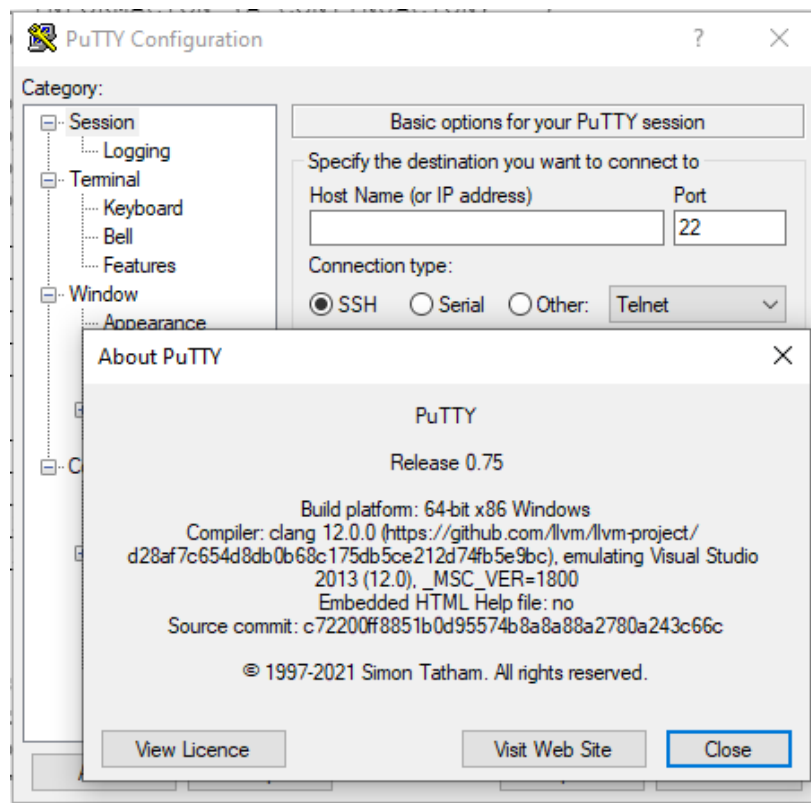


Figura 2.16: Programa PuTTY empleado para establecer una sesión/conexión SSH con la Raspberry Pi

### 2.7.2. Qt Creator

Para programar la Raspberry Pi 3B mediante C++ se escogió al IDE Qt Creator (ver Figura 2.17), el cual se describe brevemente en el Subtema 1.3.9. Para usarlo es necesario instalarlo a través de la terminal de comandos (el procedimiento no se mencionará en el presente proyecto), dicho IDE cuenta con una variada oferta de librerías (framework) como lo son <QString>, <QDebug>, <QUdpSocket>, <QApplication>, <QMainWindow>, <QVector>, <QObject> y <QTimer>, mismas que fueron empleadas en alguna sección del código (ver Anexo B).

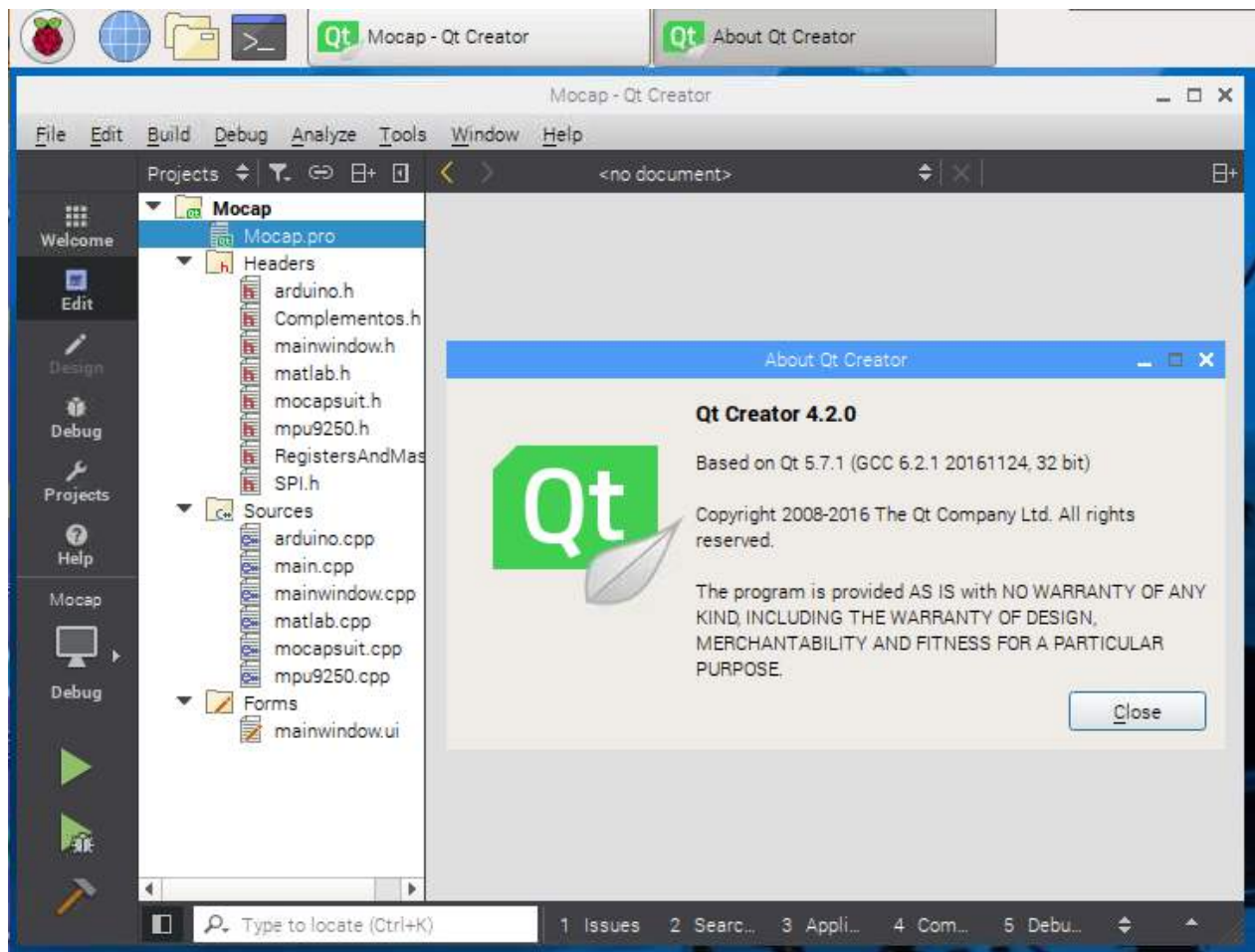


Figura 2.17: Versión del IDE Qt Creator

### 2.7.3. Análisis en Github y Gitlab de códigos de terceros

Para poder comunicar la Raspberry Pi y el sensor MPU9250 vía I2C o SPI se buscó código *open source* de terceros con el cual poder utilizar SPI o I2C en la Rpi (variables, funciones y clases), posteriormente, se analizó su funcionamiento para ver que funciones del código pueden ser útiles en el presente proyecto.

Para poder emplear las capacidades de comunicación SPI y otras funcionalidades “Wiring”<sup>9</sup> en la Raspberry Pi 3B, se emplearon las bibliotecas de Wiring Pi, las cuales se

<sup>9</sup>El software de Arduino está basado en la estructura del lenguaje de programación Wiring. Wiring está

pueden descargar e instalar libremente. Una forma de hacerlo es a través de la consola de comandos de Raspberry Pi OS, también se pueden encontrar forks<sup>10</sup> en Github de bibliotecas “Wiring Pi” que son desarrolladas y mantenidas por terceros ajenos al autor de Wiring Pi (Gordon Henderson).

WiringPi es una librería escrita en C, se usa para controlar los pines GPIO para los SoC BCM2835, BCM2836 y BCM2837 de Raspberry Pi. Además, para controlar los pines GPIO en C existen otras librerías como pigpio, bcm2835, sysfs, etc.

Las funciones que se emplearon de la librería “wiringPi.h” son:

- void pinMode(int pin, int mode)
- void digitalWrite(int pin, int value)
- int wiringPiSPISetup(int channel, int speed)
- int wiringPiSetup(void)
- void delay(unsigned int)
- void delayMicroseconds(unsigned int)
- unsigned int millis(void)
- unsigned int micros (void)

Las funciones que se emplearon de la librería “wiringPiSPI.h” son:

- int wiringPiSPIDataRW(int channel, unsigned char \*data, int len)
- int wiringPiSPISetupMode(int channel, int speed, int mode)

Es importante también activar la interfaz SPI de la RPi. Para esto, debemos ejecutar el comando “sudo raspi-config” en la consola de comandos de Raspberry Pi OS; luego, con

pensado para que gente sin muchos conocimientos de hardware pudiera trabajar con él fácilmente

<sup>10</sup>Fork o bifurcación es una nueva rama creada a partir de un código ya existente, con el fin de dar inicio a un proyecto con base en otro

las flechas del teclado desplazarse hasta “5 Interfacing Options - Configure connections to peripherals”, y teclear Intro (Enter); en la nueva interfaz hay que desplazarse hasta “P4 SPI - Enable/Disable automatic loading of SPI kernel module”, y dar Enter; por último, nos desplazamos hasta “Yes (Sí)”, y damos Enter. Ya con esto se mostrará un mensaje como el que se muestra en la Figura 2.18

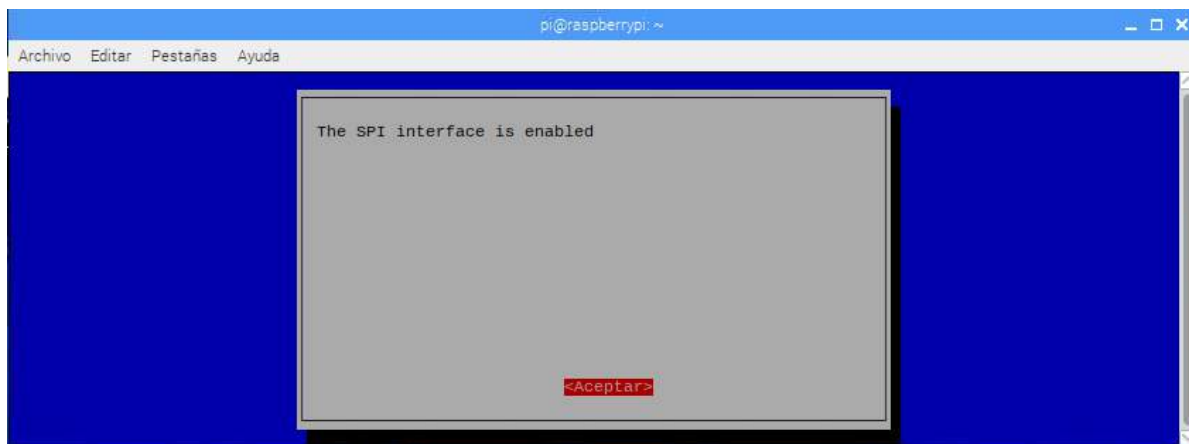


Figura 2.18: Mensaje obtenido luego de activar satisfactoriamente la interfaz SPI en la RPi 3B

## 2.8. Código generado con Qt Creator en la Raspberry Pi 3B

El código C++ escrito y desarrollado durante el presente proyecto consta de más de 2600 líneas de código. Parte de las mismas se desarrollaron para Arduino, algunas otras se transcribieron de lo que se tenía en Matlab, y otras más se escribieron específicamente para la Raspberry Pi.

El código desarrollado durante el presente proyecto consta de archivos “.h”, “.cpp”, “.ui” y “.pro” (estos últimos 2 son propios del framework Qt Creator).



### 2.8.1. Qt files

#### **Mocap.pro:**

Este archivo contiene la información relativa de qué y cómo se debe compilar el proyecto, por ejemplo: `QMAKE_CXXFLAGS += -std=c++17` le indica al compilador qué versión o estándar de C++ se está empleando en el proyecto.

De igual forma, `LIBS += -L/usr/local/lib -lwiringPi` le especifica qué librerías tienen que ser ligadas al proyecto, le indica su dirección absoluta en memoria precedida por `-L`, y el nombre de la librería (`wiringPi`) precedido por `-l`. Ligar esta carpeta es muy importante para poder trabajar con todas las funciones de acceso a los pines GPIO (p. ej., incluye la comunicación SPI).

El archivo se encuentra en el Anexo B.11. Para más información de los archivos “.pro” de QT favor de visitar: <https://doc.qt.io/qt-5/qmake-project-files.html>

#### **mainwindow.ui:**

Este archivo contiene la información referente a los detalles gráficos de la Interfaz Gráfica de Usuario (GUI, por sus siglas en inglés) empleada en el proyecto. Por ejemplo, las coordenadas, márgenes y nombres de los elementos que componen la GUI (pushbuttons, radio buttons, check boxes, etc.). Este archivo se encuentra en el Anexo B.8.

### 2.8.2. Header files

Los archivos de cabecera (o header files) se usan, normalmente, para contener un conjunto de declaraciones de clases, métodos, funciones, variables, etc.

#### **RegistersAndMask.h:**

Este archivo contiene una serie de macros con las direcciones de los registros utilizados, y de los valores que se deben asignar a estas direcciones para activar ciertas funcio-

nes del sensor MPU9250 (ver el Anexo B.2). También, contiene las direcciones I2C del acelerómetro-giróscopo (“MPU9250\_ADDR” = 0x68) y del magnetómetro (“MAG\_ADDRESS” = 0x0C).

### **Complementos.h:**

Este archivo consta de 2 partes: la primera contiene una serie de macros para habilitar o deshabilitar ciertas declaraciones de funciones y clases dependiendo de si se desea emplear la comunicación I2C o SPI, y en segundo lugar, este archivo contiene las clases *Mean* e *Index* (se puede constatar lo anterior en el Anexo B.4).

La clase *Index* envuelve métodos y miembros para funcionar como un contador que indica la posición de un número que está dentro de un arreglo de flotantes.

La clase *Mean* se desarrolló con la finalidad de promediar las “n” últimas mediciones, y así filtrar ruidos indeseados provenientes de la medición del sensor. Al ser un arreglo, tras insertar una nueva medición, esta clase elimina la medición “n+1” más antigua.

### **arduino.h:**

Contiene la declaración de la clase `_Serial`, la cual permite que algunas funciones de la clase *Serial* de Arduino puedan ser usadas en Qt Creator; también pasa algo similar con la función *String()* de Arduino. En ambos casos solo se sobrecargaron las funciones que fueron utilizadas en el proyecto. Este archivo se encuentra en el Anexo B.3.

### **SPI.h:**

Contiene la declaración de la clase `_SPI`, la cual busca volver “compatibles” las funciones empleadas de la librería *SPI.h* de Arduino, pero ahora dentro de Qt Creator. Este archivo se encuentra en el Anexo B.16.

**mainwindow.h:**

Este archivo ubicado en el Anexo B.7 contiene la definición de la clase *MainWindow*, la cual no es esencial para este proyecto. Esta clase es parte de una plantilla de Qt Creator para aquellos proyectos en los que se requiere de una interfaz gráfica (en este proyecto, la interfaz gráfica solo se utilizó para depurar todos los códigos del Tema 2.8). El funcionamiento de los objetos instanciados de esta clase se mencionará en el subsubtema “mainwindow.cpp”.

**matlab.h:**

Esta biblioteca junto con el código de matlab.cpp está autocontenida; básicamente, solo la función *delay(int)* que pertenece a la biblioteca de *wiringPi.h*, se invoca desde el exterior, y lo hace únicamente dentro de la función *void pause(uint8\_t seconds)*. La cabecera “matlab.h” solo contiene las declaraciones de las clases Vector, Matriz y MatrizCuadrada, y de las funciones y métodos que operan con estas clases. Este archivo se encuentra en el Anexo B.10.

**mpu9250.h:**

Este archivo se encuentra en el Anexo B.15, y está compuesto por las declaraciones de clases, métodos, funciones, miembros y variables que son necesarios para poder trabajar con un sensor individual del modelo MPU9250. Gracias a la clase *Mpu9250* se pueden instanciar tantos sensores como sean necesarios (p. ej., ver el Anexo B.1).

**mocapsuit.h:**

Este archivo contiene las declaraciones de clases y métodos que coordinarán todos los sensores MPU-9250. La clase *MocapSuit* está pensada para tener el control total sobre todos los sensores (centralizarlos), de forma que desde aquí se manden a llamar a los métodos y se actualicen los atributos de la clase *Mpu9250*. El archivo se encuentra en el Anexo B.13.

### 2.8.3. Source files

Los archivos de código fuente consisten, en buena medida, en las definiciones de las declaraciones de los archivos de cabecera (Subtema 2.8.2).

#### **Arduino.cpp:**

Como se puede ver en el Anexo B.5, este archivo solo define las funciones heredadas de Arduino como *Serial.print()*, *Serial.println()* y *String()*, porque son las únicas funciones de Arduino que se emplearon en su plataforma y que no requieren *#include <>*.

#### **mainwindow.cpp:**

Este archivo se puede consultar en el Anexo B.6. Su utilidad consiste en funcionar como “Back-End” (analogía alusiva al desarrollo web) de la interfaz gráfica (GUI) de la Figura 2.19. La función *on\_pushButton\_clicked()* es la única que no es propia de la plantilla, y sirve para enviar un datagrama UDP que contiene el texto introducido en el recuadro, el datagrama se envía cada que se presiona el botón.

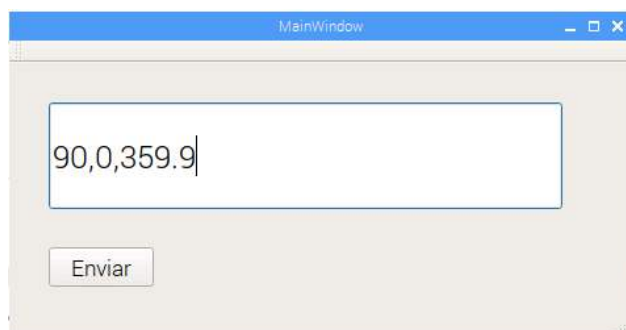


Figura 2.19: GUI resultante del proyecto en Qt

#### **matlab.cpp:**

Con este archivo se pueden realizar sumas, restas, multiplicaciones y divisiones del tipo elemento a elemento entre variables, vectores y matrices. Además, tiene las instrucciones para calcular la inversa de una *MatrizCuadrada*, contiene el algoritmo de regresión lineal

para la calibración automática, y contiene las instrucciones para hacer conversiones entre ángulos de Euler, matrices de rotación y cuaterniones. Este archivo está pensado para poder transcribir hacia C++ (Qt Creator) todas las funciones “nativas” utilizadas de Matlab. El archivo se encuentra en el Anexo B.9.

#### **mpu9250.cpp:**

En este archivo tienen lugar, prácticamente todas, las escrituras y lecturas en registros del sensor MPU9250, incluidas las mediciones del acelerómetro, giróscopo y magnetómetro. Incluye las llamadas para resetear y setear el sensor, alberga el filtro de Madgwick y el algoritmo de calibración automática (véase el Anexo B.14).

Este archivo consiste en la conjunción del archivo **ARDUINO CODE / FINAL\_GENERAL / FINAL\_GENERAL.ino** con el archivo **MATLAB / ArduinoUnrealUDP / Madgwick / Madgwick\_PLUS\_Calibration.m**, ambos se pueden encontrar en [GitHub](#)<sup>11</sup> y en [Google Drive](#)<sup>12</sup>.

#### **mocapsuit.cpp:**

El archivo contiene a las definiciones de la clase *MocapSuit*, la cual, además de lo ya mencionado, se encarga de alinear a todos los sensores tan pronto reciba la instrucción del videojuego; de esta clase salen, prácticamente todos, los paquetes UDP; incluso, con esta clase los métodos y atributos de cada sensor individual (instancias de la clase *Mpu9250*) pueden ser ejecutados o modificados, respectivamente. Este archivo se encuentra en el Anexo B.12.

#### **main.cpp:**

Si bien este archivo contiene a la función principal del todo el programa, o *void main()*, la cantidad de código que se puede encontrar vertido en él es ínfima; no obstante, contiene

---

<sup>11</sup>[https://github.com/alejandroivan10/ResidenciaProfesional\\_IvanAlejandroMunozVera](https://github.com/alejandroivan10/ResidenciaProfesional_IvanAlejandroMunozVera)

<sup>12</sup>[https://drive.google.com/drive/folders/11ay\\_yAM95xgmOL4HYDAj85NgqfiTDedR?usp=sharing](https://drive.google.com/drive/folders/11ay_yAM95xgmOL4HYDAj85NgqfiTDedR?usp=sharing)

las inicializaciones de todos los sensores y del traje MainSuit. Este archivo se encuentra en el Anexo B.1

Cada sensor Mpu9250 se inicializa con 6 parámetros, mismos que se describen en la Tabla 2.8 (NOTA: floatList es equivalente a `std::initializer_list<float>`).

Tabla 2.8: Parámetros para inicializar cada sensor MPU-9250

# de parámetro	Tipo	Motivo o utilidad del parámetro
1	uint	Indica en qué Pin físico GPIO se encuentra el Chip Select de este sensor
2	MocapSuit&	Indica a qué traje Mocap se encuentra este sensor conectado (solo existe el traje MainSuit)
3	floatList	Contiene 6 flotantes para el acelerómetro: los 3 primeros son la sensibilidad en X, Y y Z; los últimos 3 son el Offset en X, Y y Z
4	floatList	Este parámetro es una lista con 3 flotantes para el giróscopo: el Offset en X, Y y Z
5	floatList	Contiene 6 flotantes para el magnetómetro: los 3 primeros son la sensibilidad en X, Y y Z; los últimos 3 son el Offset en X, Y y Z
6	floatList	Son los ángulos de Euler iniciales extraídos del ThirdPersonCharacter en Unreal Editor

## 2.9. Unreal

Debido a que los blueprints son bloques, no se puede agarrar el código de forma sencilla para ponerlo en los mismos repositorios de GitHub; y subir todo el código C++ del videojuego tampoco parece ser una buena idea porque la carpeta del proyecto pesa alrededor de 12GB si se incluye el código fuente de Unreal. Solo mencionar que la estructura de las carpetas y archivos se puede encontrar en la Figura 2.9.

La intención de su servidor está en subir los archivos necesarios para que terceras personas puedan acceder a ellos y, con esto, replicar todo el proyecto. En caso de conseguir los archivos faltantes de Unreal, estos estarán en el mismo enlace de GitHub, o bien en los repositorios del usuario que aparece tras abrir dicho enlace.

### **2.9.1. Blueprint: ThirdPersonCharacter**

El programa en blueprints para recibir los datagramas UDP, descomponer los datagramas en arreglos de flotantes y, posteriormente, convertirlos en variables de tipo *rotator*, se muestra en la Figura 2.8. Para acceder al programa se debe ir a la carpeta *Contents / ThirdPersonBP / Blueprints*, y hacer doble clic en archivo *ThirdPersonCharacter*.

El resultado de abrir el archivo se muestra en la Figura 2.20, en la cual se pueden observar bloques de blueprints que están al interior de cajas de comentarios color gris. Algunos de estos bloques están pensados para soportar diversos tipos de controladores (dígase un teclado, un mouse, un control de videojuego, un gamepad VR, etc.); los 2 bloques que se encuentran en la parte inferior y que contienen bordes anaranjados están pensados para poder leer paquetes UDP, almacenar los valores leídos, esperar a que llegue la señal de comienzo, etc.

#### **Condición de alineación y arranque de la captura de movimiento**

Los blueprints que se muestran en la Figura 2.21 sirven para emitir un mensaje UDP con dirección a la Raspberry Pi. El mensaje enviado es tal que la Raspberry Pi lo va a comparar y con ello decidir si el mensaje que acaba de recibir es el de arranque del programa en Qt Creator, o no.

#### **Datagramas UDP (obtención de las orientaciones mocap)**

Los blueprints en la parte inferior derecha de la Figura 2.20 están pensados para tener un comportamiento variable, de forma que se ejecutarán tantos bucles como cantidad

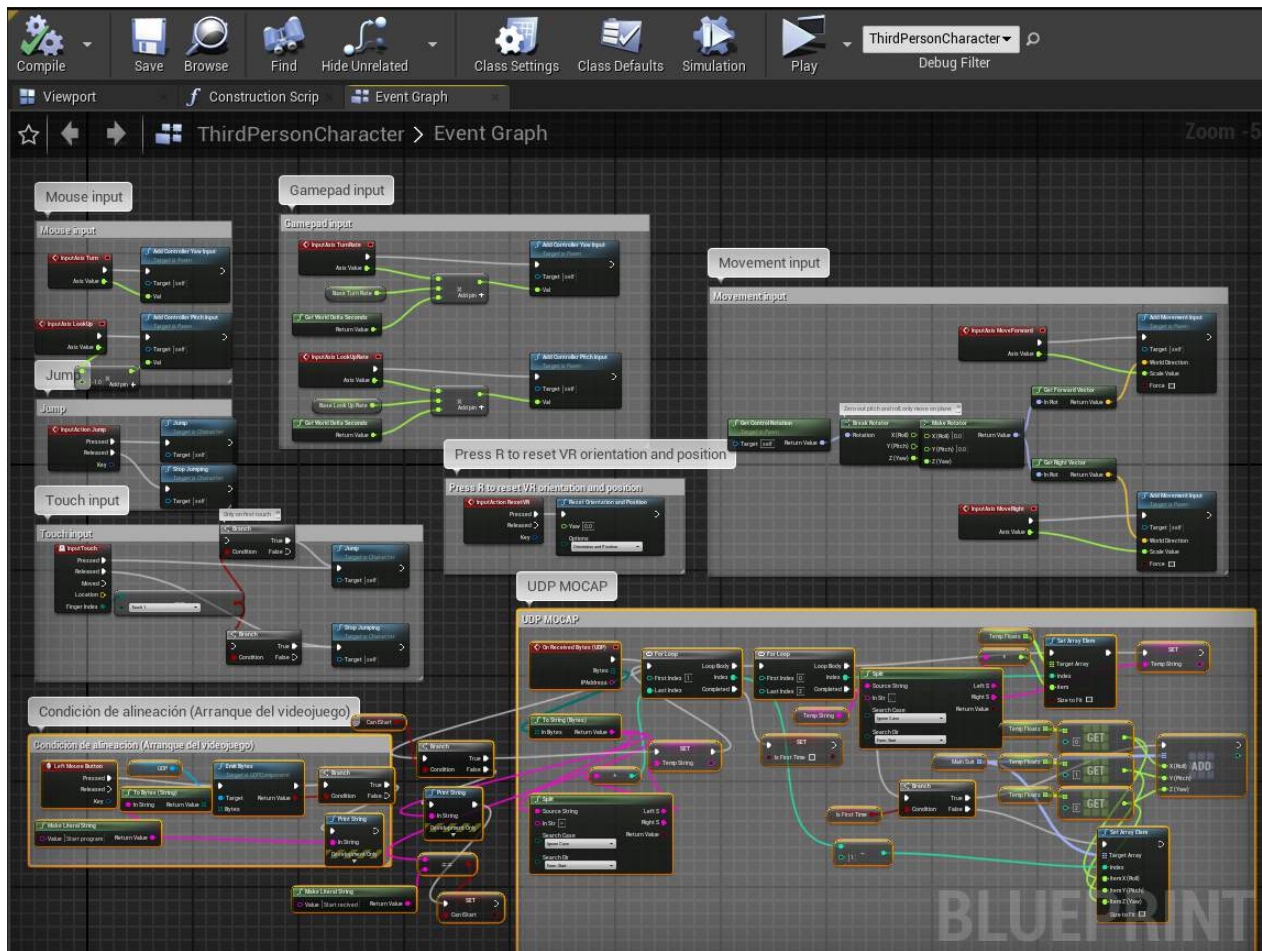


Figura 2.20: Blueprints empleados para las funcionalidades del ThirdPersonCharacter (personaje virtual con bordes anaranjados en la Figura 2.8)

de ángulos de Euler recibidos (véase la Figura 2.22); por tanto, las Raspberry Pi podría transmitir los ángulos de Euler de un sensor MPU9250, o bien de 20 sensores (un traje mocap), o bien de tantos como las capacidades computacionales lo permitan.

Lo anterior es posible porque al comienzo del paquete UDP, la Raspberry Pi introduce la cantidad de sensores MPU9250 que esta tiene conectados y, posteriormente, adjunta los 3 ángulos de Euler respectivos de cada sensor.



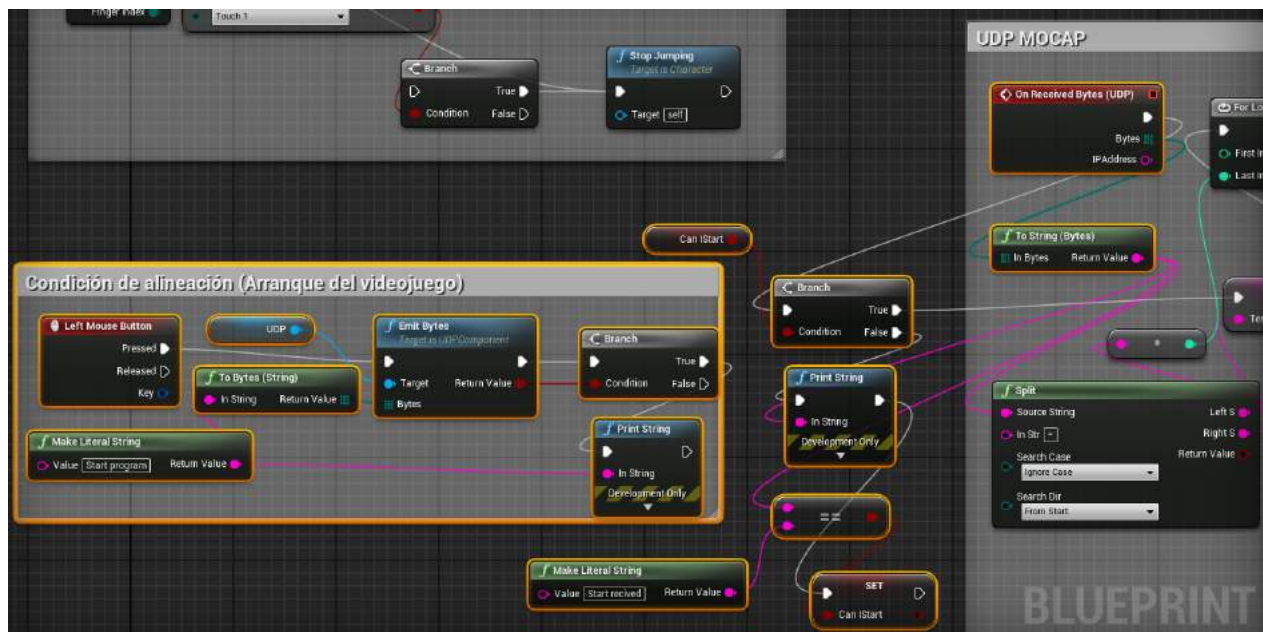


Figura 2.21: Blueprints necesarios para mandar la trama UDP que comenzará la alineación del traje en la Rpi

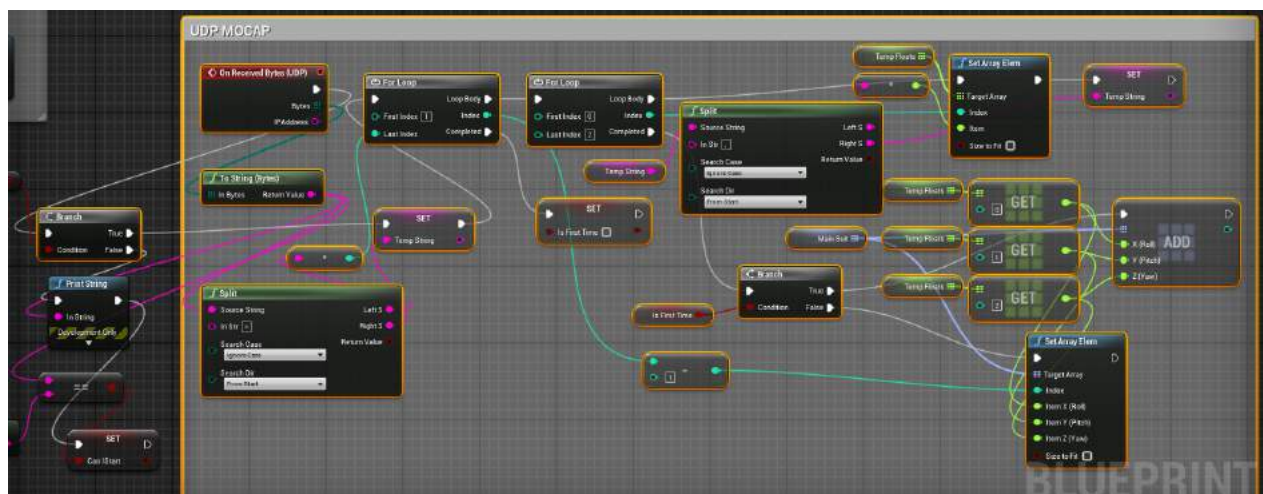


Figura 2.22: Blueprints necesarios para leer las tramas UDP enviadas por la Rpi

### **2.9.2. Blueprint: ThirdPersonExampleMap**

Dado que Unreal fue diseñado para desarrollar videojuegos en él, es necesario utilizar los blueprints del nivel o mapa para poder intercambiar información entre los diversos actores que se encuentren vinculados a ese nivel o mapa. Para programar en los blueprints del nivel procedemos como sigue: una vez en el menú principal (ver Figura 2.8), encontraremos en la parte superior del Viewport un bloque llamado blueprints, el cual contiene una pestaña a la derecha, hacemos clic sobre ella, y se desplegarán varias opciones; de ahí seleccionamos abrir blueprint del nivel (open level blueprint, en inglés), y se abrirá un script con todos los blueprints (ver Figura 2.23) empleados para la interacción entre los actores del mapa.

Para fines de este proyecto, se toma del arreglo MainSuit una orientación para asignarla a las naves. MainSuit pertenece al objeto del tipo ThirdPersonCharacter, y contiene un arreglo de tamaño variable con las orientaciones (ángulos de Euler) de todos los sensores conectados a la Raspberry Pi, este arreglo es modificado/actualizado cada que se recibe un paquete UDP en Unreal, como ya se mencionó anteriormente.

### **2.9.3. Blueprint: Animation**

La animación es una de las partes más importantes de este proyecto, aquí es donde se asignan las orientaciones de los sensores MPU9250 a ciertas partes del cuerpo del personaje en Unreal.

#### **Event Graph**

Entre lo más relevante que ocurre en este programa está el hacer una copia local de la variable MainSuit del ThirPersonCharacter, esta copia local es necesaria para asignársela a la animación del personaje. El programa blueprint Event Graph se muestra en la Figura 2.24.

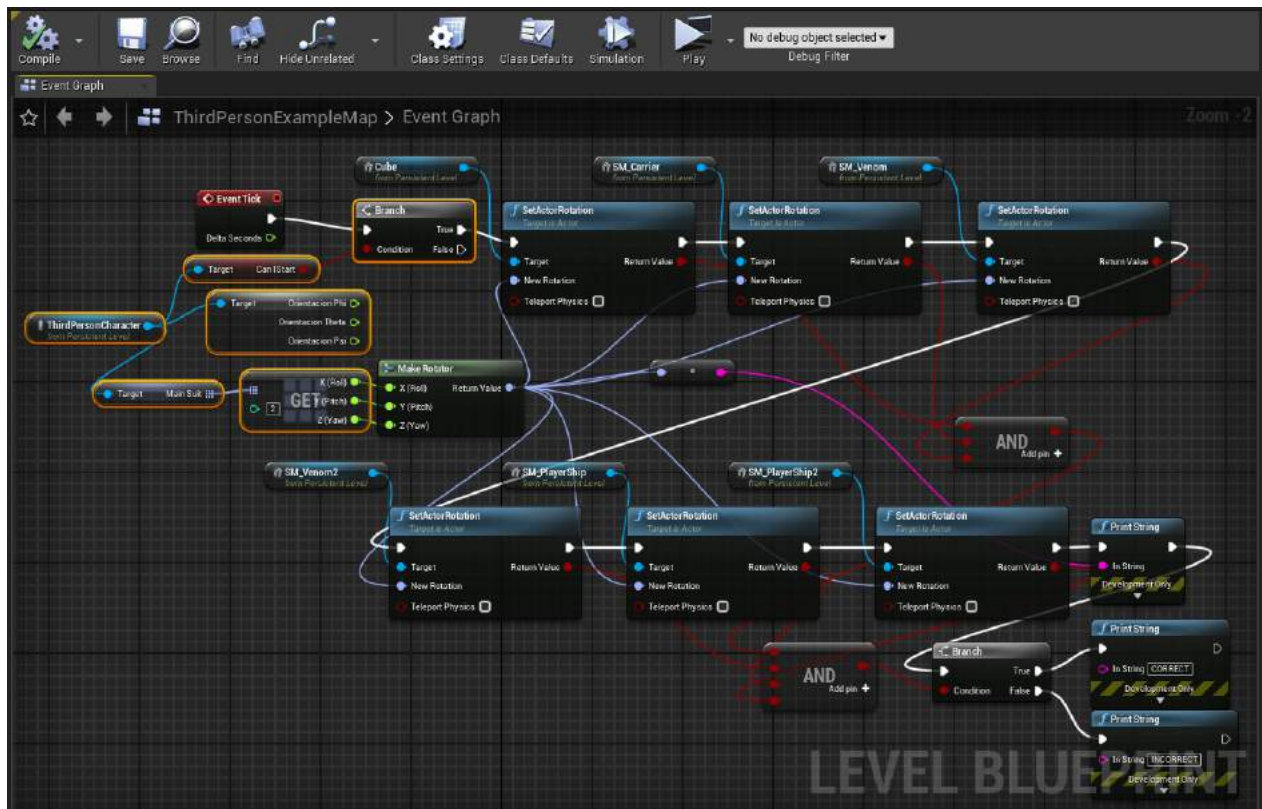


Figura 2.23: Blueprints empleados para la interacción de los objetos dentro del ThirdPersonExampleMap

## Anim Graph

El último programa blueprint es mostrado en la Figura 2.25, sirve para definir qué orientación del arreglo *MainSuit* se desea asignar a cierta parte del cuerpo del personaje ThirdPersonCharacter. Antes de asignar el valor del sensor MPU9250 al personaje virtual, se tiene que realizar una transformación de orientación como las mencionadas en el Subtema 1.3.2.

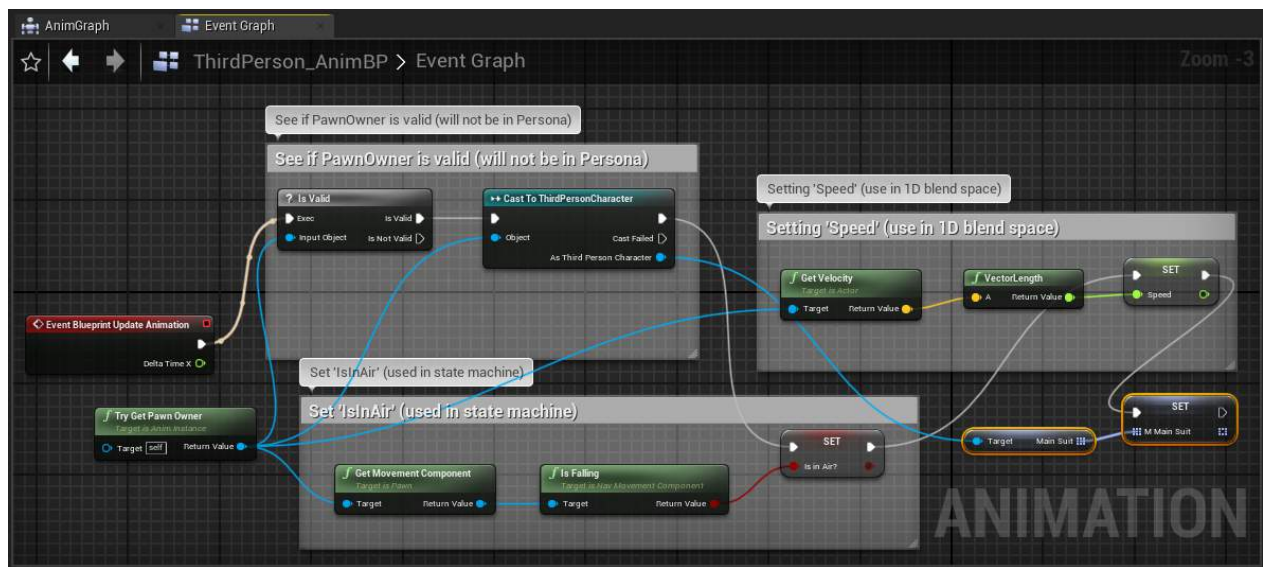


Figura 2.24: Blueprints necesarios para poder animar al personaje (correr, saltar, etc.)



Figura 2.25: Blueprints para asignar la orientación al personaje ThirdPersonCharacter

## 2.10. Traje de captura de movimiento

Al principio, parecía conveniente seguir la misma ruta que ya habían trazado la mayoría de las grandes empresas de trajes mocap: desarrollar un traje de cuerpo completo. Pero esto conlleva que el traje se debería elaborar por tallas (chica, mediana, grande, etc.), lo cual no es la intención en este proyecto.



Sería deseable que el traje pudiera cubrir un rango más amplio de medidas o tallas, razón por la cual se optó por emplear un chaleco ajustable y unas cuantas vendas. Se optó por el chaleco porque, al no tener mangas y al ser ajustable, con él se puede ampliar el rango de tallas soportadas (grandes o pequeños, delgados o con sobrepeso), y facilitar el vestir y desvestir.

Se optó por emplear vendas porque son económicas, pueden lograr un traje multitalla (se envuelven sin importar las dimensiones de la persona) y, además, ayudan a fijar la posición de los sensores (evitando la desalineación respecto al portador).

Si bien el chaleco se compró en talla mediana, se realizaron ciertas costuras en lo ancho y alto, quedando un chaleco con aspecto de talla chica, pero con la ventaja de tener algunas dimensiones de la talla original. Este chaleco tiene un largo mínimo para la cintura de 80cm y un máximo de 110cm, pero sería recomendable conseguir cintas más largas para extender el rango.

Para mencionar como dato: hay a la venta trajes comerciales en donde se emplean muñequeras, tobilleras, coderas y rodilleras, algo similar a las vendas que se contemplan en el presente proyecto.

### **2.10.1. Selección del tipo de cable para conectar los componentes del traje**

En este proyecto es muy importante escoger un tipo de cable adecuado, ya que se requiere que sea flexible, plano y de bajo calibre: flexible porque se encuentra sujeto a los movimientos bruscos que pueda realizar una persona; plano porque así todos los conductores, al estar dispuestos en un arreglo horizontal, sufren casi los mismos esfuerzos mecánicos de flexión; de bajo calibre porque solo se requiere de baja potencia y, además, así no incomodan al portador del traje.

Se investigaron por internet diferentes tipos de cables flexibles y/o planos disponibles en el mercado a través de plataformas como DigiKey, Mouser, Mercado libre. Se comenzó investigando en las primeras 2 dada la gran diversidad de su inventario.

La desventaja principal de plataformas como DigiKey y Mouser es que para pedidos cuyo precio sea menor a cierta cantidad se cobra una tarifa de envío. En DigiKey se cobra un cargo de envío de 30 USD para pedidos inferiores a 100 USD, y un envío gratuito para pedidos de más de 100 USD. En Mouser no hay costo de envío para pedidos por encima de los 40 USD para la mayoría de los productos.

Dado que el presente proyecto es un prototipo, se optó por comprar modelos de cables ampliamente usados en todo el mundo, como el HDMI o Ethernet. Se empleó la plataforma de Mercado Libre para realizar el pedido, en particular, de un modelo de cable Ethernet plano, flexible, con 10m de longitud, Cat6 (+100 Mb/s), y supuestamente calibre 28 AWG (cuando en realidad eran calibre 23 AWG).

### **2.10.2. Conexiones eléctricas/electrónicas del traje**

Como ya se ha mencionado, con el protocolo SPI los sensores comparten 3 líneas de datos (MISO, MOSI y SCLK). De hecho, es posible conectar, en la teoría, muchos sensores SPI a estas mismas líneas sin ningún menoscabo por la calidad del intercambio de los datos. En la Figura 2.26 se muestran las conexiones empleadas para el prototipo del traje mocap con 3 sensores MPU-9250.

Para el diseño final será necesario colocar en paralelo las líneas MISO, MOSI y SCLK en los 17 sensores, y por cada sensor se deberá agregar una línea más de *chip select*. Es recomendable utilizar un segundo puerto SPI en caso de contar con él (la RPi 3B cuenta con tres puertos SPI, de los cuales solo dos están disponibles mediante los pines GPIO), en el caso del prototipo solo se trabajó con 1 puerto SPI.

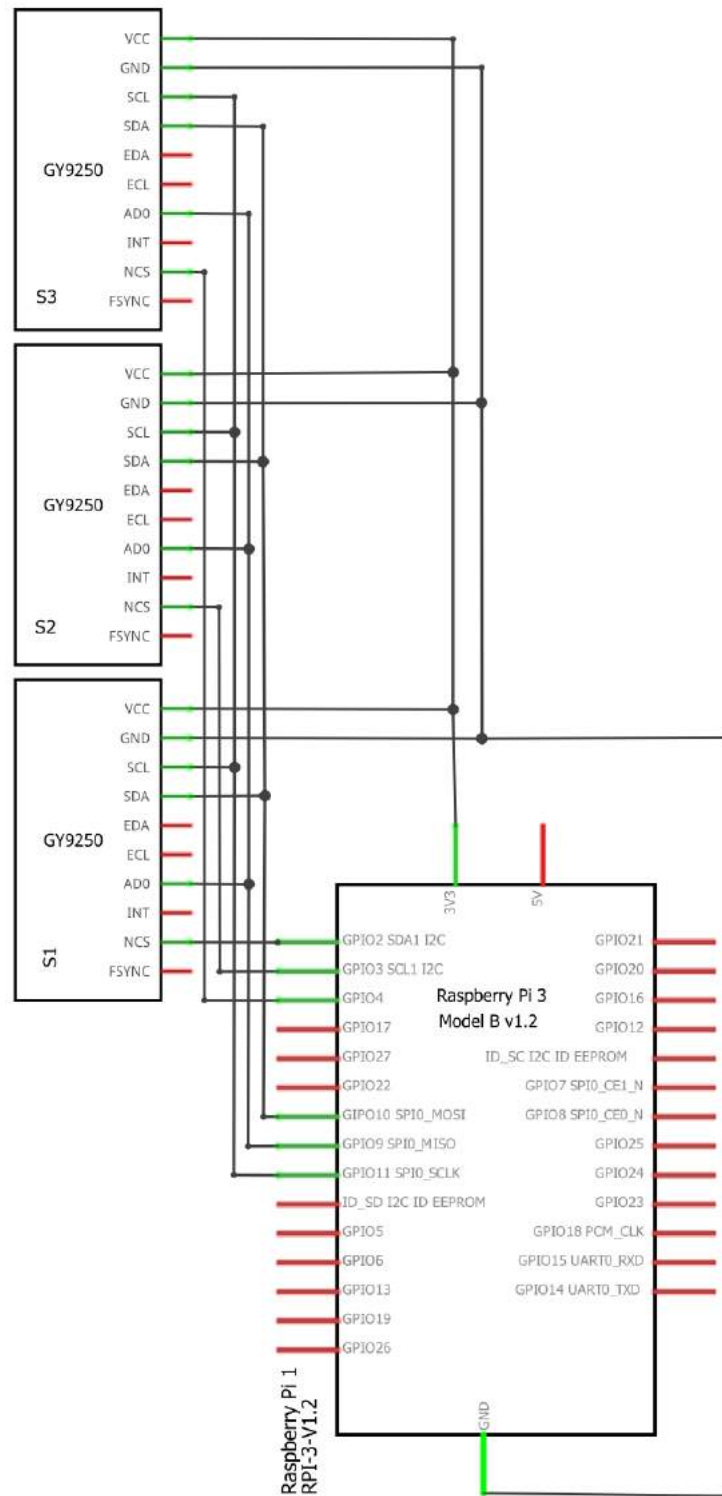


Figura 2.26: Conexiones SPI entre la Raspberry Pi 3B V1.2 y los sensores del traje

Los puntos en donde será necesario colocar los sensores deberán de estar relativamente cercas de los mostrados en la Figura 2.27. El prototipo del traje consta de 3 sensores, pero lo ideal será elaborar un traje que contenga los 17 sensores mostrados. Se consideraron solo 17 sensores porque con estos basta para poder reproducir de forma aceptable el movimiento de una persona.

Las ubicaciones de los sensores de los brazos y las piernas son poco críticas, siempre y cuando se encuentren entre las articulaciones que les corresponden. Aunque en un futuro posterior a la residencia, para cuando el algoritmo de alineación automática esté terminado, sí será importante saber que tan cercas o lejos se encuentra un sensor respecto a cierta articulación.

### **2.10.3. Vinculación entre el traje mocap y un personaje en Unreal**

En la Figura 2.28 se muestra el esqueleto del personaje virtual ThirdPersonCharacter<sup>13</sup>, el cual contiene todos los “huesos” que requieren para el presente proyecto, y muchos otros más que no se van a utilizar.

Para vincular qué sensor del traje mocap debe controlar tal o cuál “hueso” en el personaje de Unreal, lo que se tiene que hacer es asignar/vincular ciertas orientaciones recibidas por UDP (que fueron enviadas por la RPi) con ciertos “huesos” o partes del cuerpo del personaje en Unreal (esto se realizó en el blueprint de la Figura 2.25).

### **2.10.4. Alineación del traje mocap (sensores MPU9250 y RPi) con el personaje virtual en Unreal**

En una situación deseable, aunque muy improbable, el sensor MPU9250 y una parte del cuerpo del personaje de Unreal (brazo, pierna, cabeza o cualquier otro) estarán alineados

---

<sup>13</sup>Hay que considerar que ciertos personajes podrían tener su propia estructura de esqueleto (p. ej., los animales).





(a) Frontal del traje mocap



(b) Reverso del traje mocap

Figura 2.27: Ubicación de los 17 sensores necesarios para el desarrollo del proyecto

y, por tanto, la rotación del sensor tendrá el mismo efecto de rotación en el personaje. Dado que la situación anterior nunca se va a presentar del todo, es necesario desarrollar código para alinear los sensores.

De hecho, con quitarte y volverte a poner el traje mocap ya es suficiente para tener que volver a alinear los sensores. Nótese que aun cuando uno de estos sensores está, p. ej., estimando la orientación de la muñeca, este se puede encontrar por la parte de afuera (tocando los nudillos, p. ej.), o bien por la parte de adentro (haciendo contacto con la palma de la mano).



Figura 2.28: Esqueleto virtual del personaje en Unreal Engine

Para alinear un sensor solo se requiere de 2 entradas: la orientación inicial del sensor  $S_0$  y la del “hueso o eslabón” que se quiere controlar del personaje en Unreal  $U_0$ . La única condición es que el portador del traje mocap y el personaje de Unreal deben tener la misma postura y orientación. En cuanto a la postura, esto significa que ambos (traje y personaje) deben colocarse, por ejemplo, en posición “T” (o *T-Pose*, en inglés), tal como se muestra en la Figura 3.11.

Para posicionar al personaje *ThirdPersonCharacter* en posición “T” se le debe asignar el *asset* de nombre *NewT-PoseAsset* que se muestra en la Figura 2.29. En la Figura 2.30a se muestra el antes, mientras que en la Figura 2.30b se muestra el después de dicha asignación.

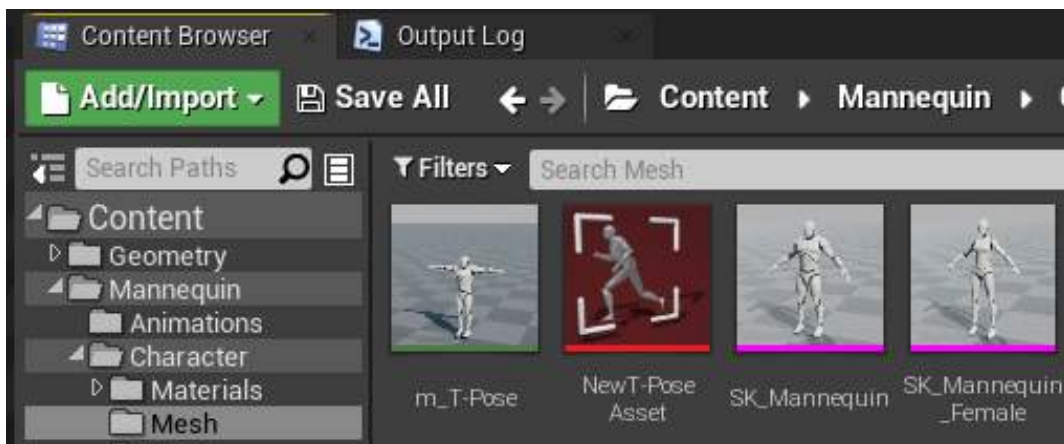
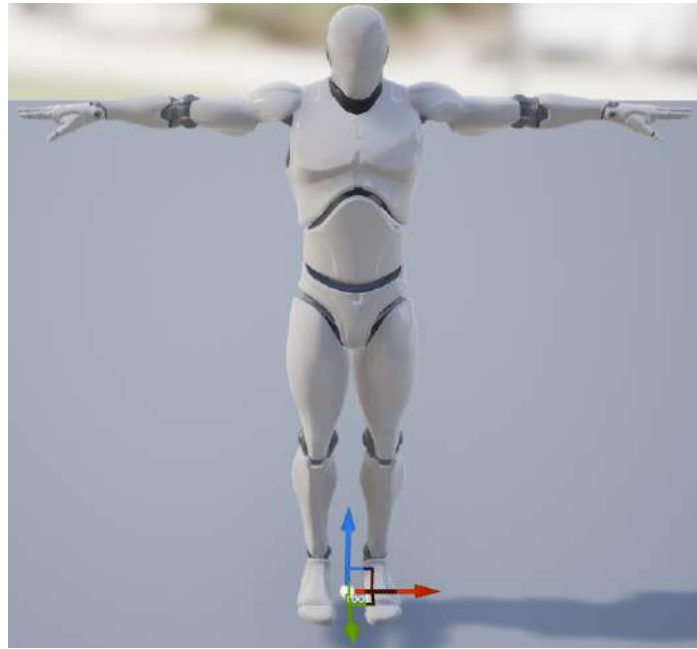


Figura 2.29: Assets agregados para usarse en *ThirdPersonCharacter*: *m\_T-Pose* y *NewT-PoseAsset*

Se empleó la Ecuación (2.2) para encontrar la matriz de ajuste  $Adj$  de cada sensor. Con estas matrices se podrá alinear al personaje con el traje (siempre y cuando se cumpla lo anteriormente expuesto sobre el tema).



(a) Pose por defecto: A-Pose



(b) Pose para alinear el traje: T-Pose

Figura 2.30: Comparación entre la pose que por defecto viene con la plantilla del videojuego en tercera persona, y la pose requerida para el desarrollo del traje mocap

Siendo  $S_0$  la orientación inicial del sensor; y  $U_0$ , la del “hueso” que se quiere controlar en Unreal; tenemos que<sup>14</sup>:

$$\begin{aligned} S_0 * Adj &= U_0 \\ \therefore Adj &= S_0^{-1} * U_0 \end{aligned} \tag{2.2}$$

Donde:

- $S_0$  es una matriz de rotación de  $3 \times 3$ , y se le asigna un valor cuando el portador del traje presiona un botón en el gamepad bluetooth. Su valor es constante en el tiempo mientras los sensores no se desalineen, en cuyo caso se debe volver a presionar el botón cuando el portador esté en posición “T”.

<sup>14</sup>Nota: La teoría sobre la que se sustentan las siguientes ecuaciones se puede encontrar en el Subtema 1.3.2

- $Adj$  es una matriz de rotación de  $3 \times 3$  que sirve de “ajuste”. Su valor depende de  $S_0$ , y equivale a la rotación necesaria para alinear al traje mocap con el personaje en Unreal.
- $U_0$  es una matriz de rotación de  $3 \times 3$ , y es constante en el tiempo porque se obtiene de la orientación de algún “hueso” del personaje en pose-T (al ser virtual no se puede desalinear).

Por tanto,  $Adj$  es una matriz de rotación relativa entre ambas orientaciones iniciales:  $S_0$  y  $U_0$ . Nótese que  $Adj$  se obtuvo en condiciones “estáticas” o “iniciales”, ya que el portador del traje no se tenía que mover de la posición “T” mientras el traje mocap se alineaba.

En condiciones “dinámicas”, es decir, cuando el portador está en movimiento, se puede reescribir la Ecuación (2.2) de tal forma que se obtiene la Ecuación (2.3). El motivo de dichas modificaciones se tratará más adelante.

$$R_S * Adj = R_U * U_0 \quad (2.3)$$

*Donde:*

- $R_S$  es una matriz de rotación de  $3 \times 3$ , es el valor de la orientación dado por el filtro Madgwick según las mediciones del sensor. Es variable en el tiempo.
- $Adj$  es la matriz de ajuste (mencionada anteriormente).
- $R_U$  es una matriz de rotación de  $3 \times 3$ , equivale al valor de la orientación que se debe aplicar al personaje en Unreal para apreciar el mocap. Es variable en el tiempo.
- $U_0$  es la matriz de orientación inicial del personaje en Unreal (mencionada anteriormente).

Tanto  $S_0$  y  $R_S$  equivalen al valor arrojado por el filtro de Madgwick, es decir, equivalen a la orientación del sensor MPU9250. El valor de  $S_0$  es constante (como ya se mencionó),

mientras que el valor de  $R_S$  depende de los movimientos del portador. De hecho, cuando el portador se posiciona en “T”,  $S_0$  y  $R_S$  tienen aproximadamente el mismo valor.

Ahora bien, para entender la Ecuación (2.3) se empleará un ejemplo de su uso en la práctica: supongamos que el portador del traje y el personaje en Unreal acaban de ser alineados y, por ende, ambos se encuentran en posición “T”, posteriormente, el portador decide bajar los brazos (o bien llevarlos a cualquier otra posición), lo cual implica una rotación  $R_u$ , misma que también se debe de reflejar en el personaje porque ambos están alineados; lo cual quiere decir que, en términos matemáticos, ambos lados de la Ecuación (2.2) deben ser multiplicados por un factor  $R_u$ , tal como se muestra en la Ecuación (2.4).

$$\begin{aligned} R_U * (S_0 * Adj) &= R_U * (U_0) \\ \therefore (R_U * S_0) * Adj &= R_U * U_0 \end{aligned} \tag{2.4}$$

Como se puede observar en la Ecuación (2.4),  $R_S$  es fruto de la multiplicación  $(R_U * S_0)$ , lo cual quiere decir que cualquier rotación que le apliquemos al sensor (posterior a su orientación inicial  $S_0$ ) hará que este arroje un nuevo valor de orientación  $R_S$ .

Como se pueden imaginar, la incógnita en la Ecuación (2.3) es  $R_U$ , ya que las otras 3 matrices ya las conocemos. Además,  $R_U$  es la matriz de rotación que se le debe aplicar al personaje en Unreal; para ello, es importante que el personaje esté inicialmente en posición “T” porque  $R_U$  es igual a la rotación que le hace falta al personaje en posición “T” para estar alineado con el traje.

Para calcular  $R_U$  se sustituye  $Adj$  de la Ecuación (2.2) en la Ecuación (2.3), obteniendo la Ecuación (2.5), como se muestra a continuación:

$$\begin{aligned} R_S * (S_0^{-1} * U_0) &= R_U * U_0 \\ \therefore R_S * S_0^{-1} &= R_U \end{aligned} \tag{2.5}$$

# Capítulo 3

## Resultados

Primero que nada, me gustaría hacer del conocimiento del lector que los procedimientos aquí mostrados se pueden llevar a la práctica con los códigos, programas y dispositivos que aquí se mencionan. Es, por tanto, conveniente empezar este capítulo con 2 links a todos los códigos empleados para el proyecto, ambos links contienen el mismo contenido, pero están almacenados en 2 plataformas online diferentes: [GitHub](#)<sup>1</sup> y [Google Drive](#)<sup>2</sup>. A lo largo de este capítulo se hará mención en reiteradas ocasiones a estos dos repositorios.

Aunado a lo anterior, el código final logrado durante esta residencia profesional está contenido en el Anexo B (y también está almacenado en los repositorios antes mencionados), mismo que se podrá descargar, estudiar, modificar, ejecutar y mejorar.

### 3.1. Arduino - MPU9250

Arduino fue fundamental para el prototipado del proyecto, ya que me permitió hacer pruebas de funcionamiento, de velocidad de comunicación, de configuración de los sensores MPU9250 y BMX055, depurar errores cuando algo salía mal con la Raspberry Pi, entre otras cosas.

#### 3.1.1. Primera prueba de comunicación y funcionamiento

El resultado de la ejecución del código **ARDUINO CODE / MPU9250 / PrimerPrueba / PrimerPrueba.ino**, mismo que se menciona en la Sección 2.4.3, se encuentra mostrado

---

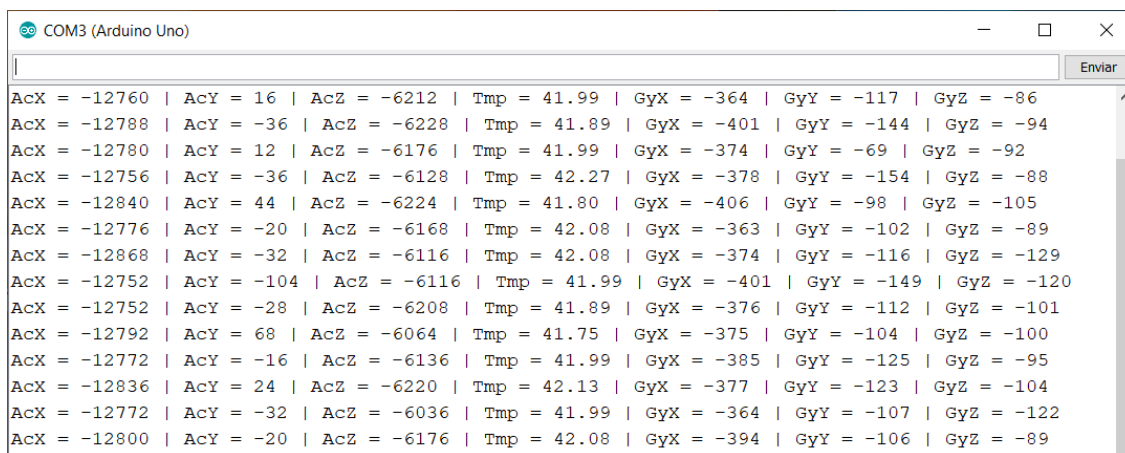
<sup>1</sup>[https://github.com/alejandroivan10/ResidenciaProfesional\\_IvanAlejandroMunozVera](https://github.com/alejandroivan10/ResidenciaProfesional_IvanAlejandroMunozVera)

<sup>2</sup>[https://drive.google.com/drive/u/0/folders/11ay\\_yAM95xgmOL4HYDAj85NgqfiTDedR](https://drive.google.com/drive/u/0/folders/11ay_yAM95xgmOL4HYDAj85NgqfiTDedR)



en la Figura 3.1. En ella se muestran la impresión de la temperatura y de 6 ejes del sensor MPU9250.

Las primeras 3 columnas son de las mediciones del acelerómetro (en X, Y y Z, respectivamente); luego, la siguiente columna es para la temperatura; y las últimas 3 columnas son para el giróscopo (en X, Y y Z, respectivamente). Los valores son leídos directamente de los registros, por lo que los valores impresos en pantalla no contienen las unidades de medición correspondientes a las variables medidas.

The image shows a screenshot of the Arduino IDE's Serial Monitor window. The title bar reads 'COM3 (Arduino Uno)'. The window contains a list of 15 lines of sensor data, each line representing a single reading. The data is formatted as a string with fields separated by vertical bars. The fields are: AcX (acceleration X), AcY (acceleration Y), AcZ (acceleration Z), Tmp (temperature), GyX (gyro X), GyY (gyro Y), and GyZ (gyro Z). The values for AcX range from -12760 to -12800. AcY ranges from -36 to 68. AcZ ranges from -6212 to -6036. Tmp ranges from 41.80 to 42.27. GyX ranges from -406 to -375. GyY ranges from -149 to -104. GyZ ranges from -120 to -89. There is a scroll bar on the right side of the text area, and an 'Enviar' button at the top right.

```
COM3 (Arduino Uno)
AcX = -12760 | AcY = 16 | AcZ = -6212 | Tmp = 41.99 | GyX = -364 | GyY = -117 | GyZ = -86
AcX = -12788 | AcY = -36 | AcZ = -6228 | Tmp = 41.89 | GyX = -401 | GyY = -144 | GyZ = -94
AcX = -12780 | AcY = 12 | AcZ = -6176 | Tmp = 41.99 | GyX = -374 | GyY = -69 | GyZ = -92
AcX = -12756 | AcY = -36 | AcZ = -6128 | Tmp = 42.27 | GyX = -378 | GyY = -154 | GyZ = -88
AcX = -12840 | AcY = 44 | AcZ = -6224 | Tmp = 41.80 | GyX = -406 | GyY = -98 | GyZ = -105
AcX = -12776 | AcY = -20 | AcZ = -6168 | Tmp = 42.08 | GyX = -363 | GyY = -102 | GyZ = -89
AcX = -12868 | AcY = -32 | AcZ = -6116 | Tmp = 42.08 | GyX = -374 | GyY = -116 | GyZ = -129
AcX = -12752 | AcY = -104 | AcZ = -6116 | Tmp = 41.99 | GyX = -401 | GyY = -149 | GyZ = -120
AcX = -12752 | AcY = -28 | AcZ = -6208 | Tmp = 41.89 | GyX = -376 | GyY = -112 | GyZ = -101
AcX = -12792 | AcY = 68 | AcZ = -6064 | Tmp = 41.75 | GyX = -375 | GyY = -104 | GyZ = -100
AcX = -12772 | AcY = -16 | AcZ = -6136 | Tmp = 41.99 | GyX = -385 | GyY = -125 | GyZ = -95
AcX = -12836 | AcY = 24 | AcZ = -6220 | Tmp = 42.13 | GyX = -377 | GyY = -123 | GyZ = -104
AcX = -12772 | AcY = -32 | AcZ = -6036 | Tmp = 41.99 | GyX = -364 | GyY = -107 | GyZ = -122
AcX = -12800 | AcY = -20 | AcZ = -6176 | Tmp = 42.08 | GyX = -394 | GyY = -106 | GyZ = -89
```

Figura 3.1: Impresión en el monitor serial de Arduino los valores de 6 de los 9 ejes del sensor MPU9250.

### 3.1.2. Escáner de registros

la Figura 3.2 muestra impreso en pantalla el resultado de ejecutar el código ubicado en el archivo **ARDUINO CODE / MPU9250 / ScannerRegistros / ScannerRegistros.ino**. Los valores impresos son los de cada registro del sensor MPU9250 al momento de ser leídos con Arduino.



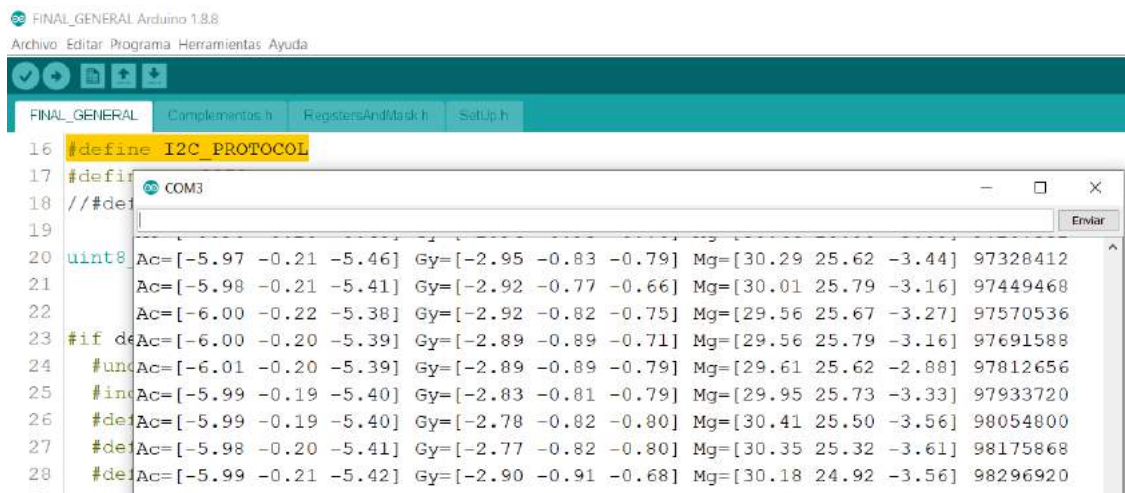
COM3 (Arduino Uno)	COM3 (Arduino Uno)	COM3 (Arduino Uno)	COM3 (Arduino Uno)
<div> <div></div> <div>COM3 (Arduino Uno)</div> </div> <div> <div></div> <div>Registro #0 = 202</div> <div>Registro #1 = 211</div> <div>Registro #2 = 225</div> <div>Registro #3 = 239</div> <div>Registro #4 = 163</div> <div>Registro #5 = 243</div> <div>Registro #6 = 24</div> <div>Registro #7 = 76</div> <div>Registro #8 = 1</div> <div>Registro #9 = 40</div> <div>Registro #10 = 252</div> <div>Registro #11 = 1</div> <div>Registro #12 = 0</div> <div>Registro #13 = 102</div> <div>Registro #14 = 94</div> <div>Registro #15 = 125</div> <div>Registro #16 = 217</div> <div>Registro #17 = 203</div> <div>Registro #18 = 168</div> <div>Registro #19 = 0</div> <div>Registro #20 = 0</div> <div>Registro #21 = 0</div> <div>Registro #22 = 0</div> <div>Registro #23 = 0</div> <div>Registro #24 = 0</div> <div>Registro #25 = 0</div> <div>Registro #26 = 0</div> <div>Registro #27 = 0</div> <div>Registro #28 = 0</div> <div>Registro #29 = 0</div> <div>Registro #30 = 0</div> <div>Registro #31 = 0</div> <div>Registro #32 = 0</div> <div>Registro #33 = 0</div> <div>Registro #34 = 0</div> <div>Registro #35 = 0</div> <div>Registro #36 = 0</div> <div> <input checked="" type="checkbox"/> Autoscroll <div> <input type="checkbox"/> <div>Mostrar marca temporal</div> </div> </div> </div>	<div> <div></div> <div>COM3 (Arduino Uno)</div> </div> <div> <div></div> <div>Registro #35 = 0</div> <div>Registro #36 = 0</div> <div>Registro #37 = 0</div> <div>Registro #38 = 0</div> <div>Registro #39 = 0</div> <div>Registro #40 = 0</div> <div>Registro #41 = 0</div> <div>Registro #42 = 0</div> <div>Registro #43 = 0</div> <div>Registro #44 = 0</div> <div>Registro #45 = 0</div> <div>Registro #46 = 0</div> <div>Registro #47 = 0</div> <div>Registro #48 = 0</div> <div>Registro #49 = 0</div> <div>Registro #50 = 0</div> <div>Registro #51 = 0</div> <div>Registro #52 = 0</div> <div>Registro #53 = 0</div> <div>Registro #54 = 0</div> <div>Registro #55 = 2</div> <div>Registro #56 = 0</div> <div>Registro #57 = 0</div> <div>Registro #58 = 5</div> <div>Registro #59 = 206</div> <div>Registro #60 = 172</div> <div>Registro #61 = 0</div> <div>Registro #62 = 8</div> <div>Registro #63 = 230</div> <div>Registro #64 = 204</div> <div>Registro #65 = 7</div> <div>Registro #66 = 64</div> <div>Registro #67 = 254</div> <div>Registro #68 = 130</div> <div>Registro #69 = 255</div> <div>Registro #70 = 163</div> <div>Registro #71 = 255</div> </div>	<div> <div></div> <div>COM3 (Arduino Uno)</div> </div> <div> <div></div> <div>Registro #70 = 163</div> <div>Registro #71 = 255</div> <div>Registro #72 = 180</div> <div>Registro #73 = 0</div> <div>Registro #74 = 0</div> <div>Registro #75 = 0</div> <div>Registro #76 = 0</div> <div>Registro #77 = 0</div> <div>Registro #78 = 0</div> <div>Registro #79 = 0</div> <div>Registro #80 = 0</div> <div>Registro #81 = 0</div> <div>Registro #82 = 0</div> <div>Registro #83 = 0</div> <div>Registro #84 = 0</div> <div>Registro #85 = 0</div> <div>Registro #86 = 0</div> <div>Registro #87 = 0</div> <div>Registro #88 = 0</div> <div>Registro #89 = 0</div> <div>Registro #90 = 0</div> <div>Registro #91 = 0</div> <div>Registro #92 = 0</div> <div>Registro #93 = 0</div> <div>Registro #94 = 0</div> <div>Registro #95 = 0</div> <div>Registro #96 = 0</div> <div>Registro #97 = 0</div> <div>Registro #98 = 1</div> <div>Registro #99 = 0</div> <div>Registro #100 = 0</div> <div>Registro #101 = 0</div> <div>Registro #102 = 0</div> <div>Registro #103 = 0</div> <div>Registro #104 = 0</div> <div>Registro #105 = 0</div> <div>Registro #106 = 0</div> <div>Registro #107 = 1</div> <div>Registro #108 = 0</div> <div>Registro #109 = 0</div> <div>Registro #110 = 0</div> <div>Registro #111 = 176</div> <div>Registro #112 = 0</div> <div>Registro #113 = 0</div> <div>Registro #114 = 0</div> <div>Registro #115 = 0</div> <div>Registro #116 = 0</div> <div>Registro #117 = 115</div> <div>Registro #118 = 0</div> <div>Registro #119 = 21</div> <div>Registro #120 = 14</div> <div>Registro #121 = 0</div> <div>Registro #122 = 234</div> <div>Registro #123 = 184</div> <div>Registro #124 = 0</div> <div>Registro #125 = 33</div> <div>Registro #126 = 88</div> </div>	<div> <div></div> <div>Registro #91 = 0</div> <div>Registro #92 = 0</div> <div>Registro #93 = 0</div> <div>Registro #94 = 0</div> <div>Registro #95 = 0</div> <div>Registro #96 = 0</div> <div>Registro #97 = 0</div> <div>Registro #98 = 1</div> <div>Registro #99 = 0</div> <div>Registro #100 = 0</div> <div>Registro #101 = 0</div> <div>Registro #102 = 0</div> <div>Registro #103 = 0</div> <div>Registro #104 = 0</div> <div>Registro #105 = 0</div> <div>Registro #106 = 0</div> <div>Registro #107 = 1</div> <div>Registro #108 = 0</div> <div>Registro #109 = 0</div> <div>Registro #110 = 0</div> <div>Registro #111 = 176</div> <div>Registro #112 = 0</div> <div>Registro #113 = 0</div> <div>Registro #114 = 0</div> <div>Registro #115 = 0</div> <div>Registro #116 = 0</div> <div>Registro #117 = 115</div> <div>Registro #118 = 0</div> <div>Registro #119 = 21</div> <div>Registro #120 = 14</div> <div>Registro #121 = 0</div> <div>Registro #122 = 234</div> <div>Registro #123 = 184</div> <div>Registro #124 = 0</div> <div>Registro #125 = 33</div> <div>Registro #126 = 88</div> </div>

(a) Registros del #0 al #36    (b) Registros del #35 al #71    (c) Registros del #70 al #106    (d) Registros del #91 al #126

Figura 3.2: Impresión en el monitor serial de todos los registros y los valores contenidos en esos registros para el sensor MPU9250

### 3.1.3. Código final

En la Figura 3.3 se muestra el resultado de la ejecución del código ubicado en el archivo **ARDUINO CODE / FINAL\_GENERAL / FINAL\_GENERAL.ino**. Este es el último código escrito con Arduino para el proyecto, mismo que está asociado a todas las pruebas siguientes efectuadas entre Arduino y Matlab. En esta Figura se puede observar que los datos se encuentran en un formato que encapsula los valores X, Y y Z del acelerómetro, giróscopo y del magnetómetro. El valor de la última columna corresponde al retornado por la función *unsigned long millis()* de Arduino.



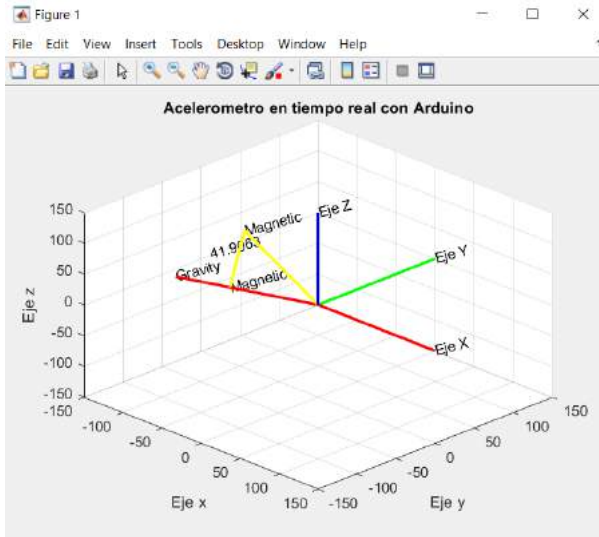
```
16 #define I2C_PROTOCOL
17 #define COM3
18 //...
19
20 uint8_t Ac=[-5.97 -0.21 -5.46] Gy=[-2.95 -0.83 -0.79] Mg=[30.29 25.62 -3.44] 97328412
21 Ac=[-5.98 -0.21 -5.41] Gy=[-2.92 -0.77 -0.66] Mg=[30.01 25.79 -3.16] 97449468
22 Ac=[-6.00 -0.22 -5.38] Gy=[-2.92 -0.82 -0.75] Mg=[29.56 25.67 -3.27] 97570536
23 #if def Ac=[-6.00 -0.20 -5.39] Gy=[-2.89 -0.89 -0.71] Mg=[29.56 25.79 -3.16] 97691588
24 #undef Ac=[-6.01 -0.20 -5.39] Gy=[-2.89 -0.89 -0.79] Mg=[29.61 25.62 -2.88] 97812656
25 #include Ac=[-5.99 -0.19 -5.40] Gy=[-2.83 -0.81 -0.79] Mg=[29.95 25.73 -3.33] 97933720
26 #define Ac=[-5.99 -0.19 -5.40] Gy=[-2.78 -0.82 -0.80] Mg=[30.41 25.50 -3.56] 98054800
27 #define Ac=[-5.98 -0.20 -5.41] Gy=[-2.77 -0.82 -0.80] Mg=[30.35 25.32 -3.61] 98175868
28 #define Ac=[-5.99 -0.21 -5.42] Gy=[-2.90 -0.91 -0.68] Mg=[30.18 24.92 -3.56] 98296920
```

Figura 3.3: Impresión de las mediciones del sensor MPU-9250 a través del monitor serial de Arduino

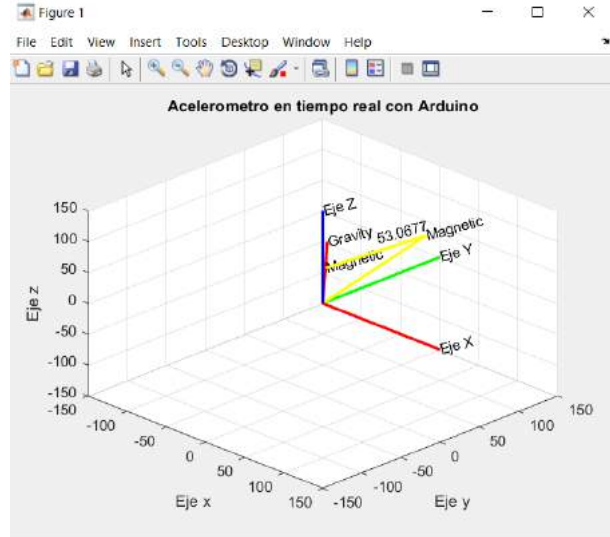
## 3.2. Matlab - Arduino - MPU9250

### 3.2.1. Vector 3D: Acelerómetro y magnetómetro

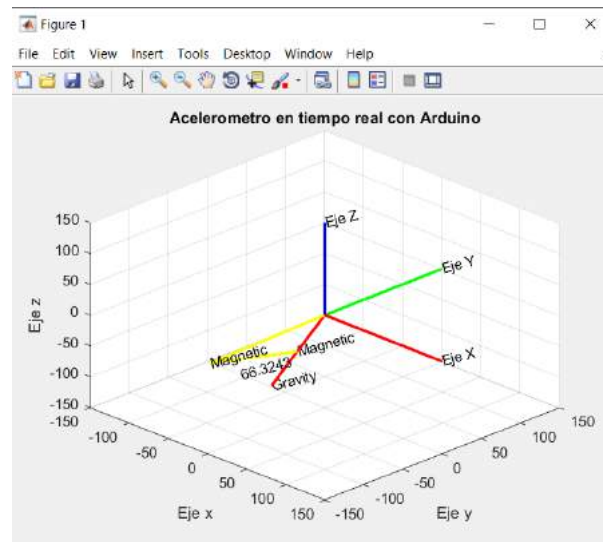
En la Figura 3.4 se muestra el resultado de la ejecución del código ubicado en el archivo **MATLAB / Pruebas / MPU9250\_Acc\_Gyr\_Mag\_3D.m** del repositorio de Github o Google Drive, este código se menciona en la Sección 2.5.2.



(a) Vector 1



(b) Vector 2



(c) Vector 3

Figura 3.4: Gráfica de los vectores acelerómetro y magnetómetro a través de Matlab, al colocar el sensor en 3 orientaciones diferentes

### 3.2.2. Calibración automática

En la Figura 3.5 se muestra la impresión en el Command Window de Matlab cuando se manda a llamar la función ubicada en el archivo **MATLAB / ArduinoUnrealUDP / Madgwick / Calibración\_RegresionLineal / RegresiLinealMultip.m**, ya sea por ejecutar el código **MATLAB / ArduinoUnrealUDP / FINAL\_UDPArduinoUnreal.m**, o bien por ejecutar el código **MATLAB / ArduinoUnrealUDP / Madgwick / Madgwick\_PLUS\_Calibration.m**.

```
Command Window
i=1028 Ac=[-0.078 -0.156 0.985] Mg=[0.571 -0.058 -0.819] Gy=[-0.069 0.215 -0.037]
i=1029 Ac=[-0.141 -0.171 0.975] Mg=[0.596 -0.063 -0.801] Gy=[-0.015 0.273 -0.000]
AcTol_Err = 1.009

Ac_Adj =

    1.0000    1.0758    0.9639    0.0699    0.3698    1.9022

Ac_Adj = 10.487  9.748  10.879  0.070  0.370  1.902
MgTol_Err = 1.027

Mg_Adj =

    1.0000    0.9057    0.8393   -7.2711    8.5834    0.1528

Mg_Adj = 48.592  53.649  57.898  -7.271  8.583  0.153
20 VALOR(ES) CAPTURADO(S)
i=1030 Ac=[-0.179 -0.136 0.974] Mg=[0.564 -0.030 -0.825] Gy=[0.047 0.115 -0.028]
i=1031 Ac=[-0.177 -0.116 0.977] Mg=[0.570 -0.031 -0.821] Gy=[0.130 -0.078 -0.006]
```

Figura 3.5: Parámetros de calibración obtenidos por regresión lineal. Impresión a través del comand window de Matlab

### 3.2.3. Filtro Madgwick

En la Figura 3.6 se muestra el resultado más importante que se obtiene luego de ejecutar el código ubicado en el archivo **MATLAB / ArduinoUnrealUDP / Madgwick / Madgwick.m**, el cual obtiene la orientación mediante cuaterniones, y después realiza una conversión a matrices de rotación.

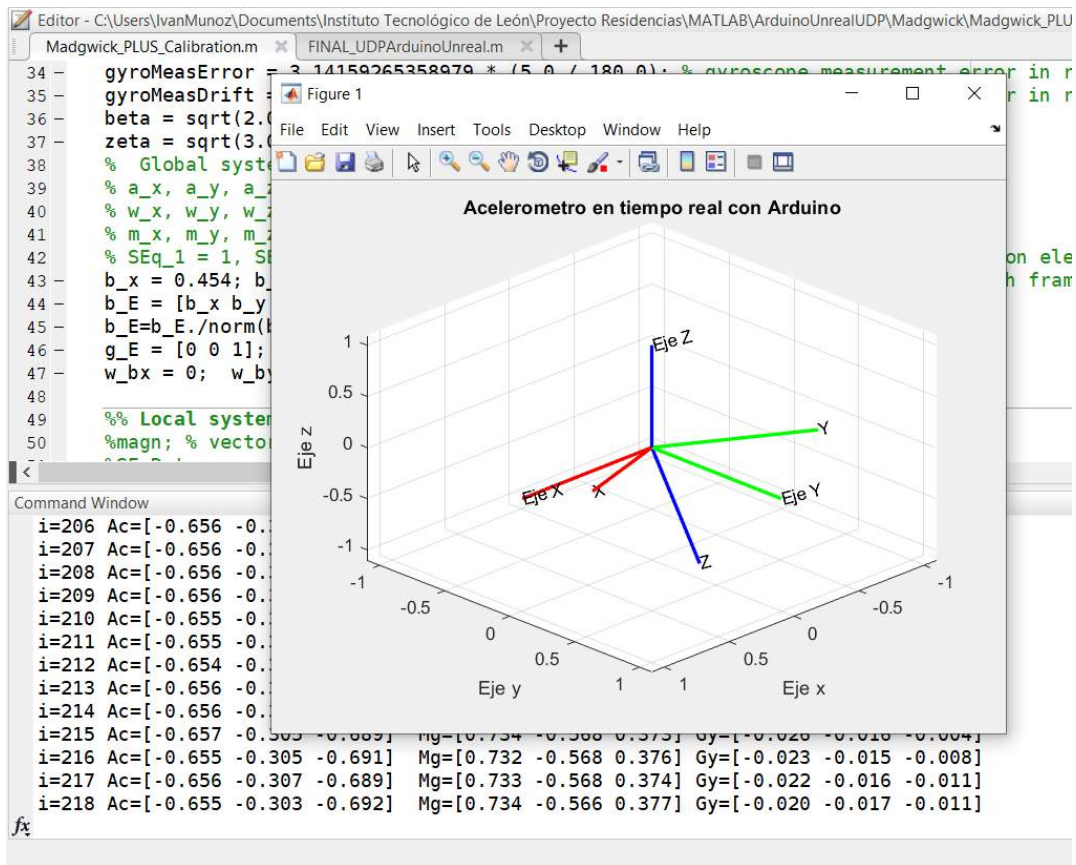


Figura 3.6: Plot generado en Matlab de la orientación estimada por el filtro Madgwick

En la Figura 3.7 se muestra la proyección isométrica del plot 3D generado por Matlab conforme a la orientación estimada por el filtro de Madgwick; si bien el resultado es fruto del mismo código que da forma a la Figura 3.6, la intención de mostrar 4 gráficas en diferentes momentos y orientaciones es para visualizar que existen 3 vectores que se quedan fijos (Eje X, Eje Y y Eje Z), a la vez que hay 3 vectores que cambian de ubicación (X, Y y Z).

Los vectores fijos hacen alusión al sistema de referencia, es bien sabido que, en algún momento dado, el vector gravedad y el vector campo magnético apuntarán en la misma dirección sin importar la rotación del instrumento que los mida. Por lo tanto, en una misma posición del planeta tierra, estos 2 vectores son “fijos”, y sirven de referencia para poder

calcular la orientación relativa respecto a algún sensor MARG (y, a su vez, calcular la orientación del objeto solidario al sensor MARG).

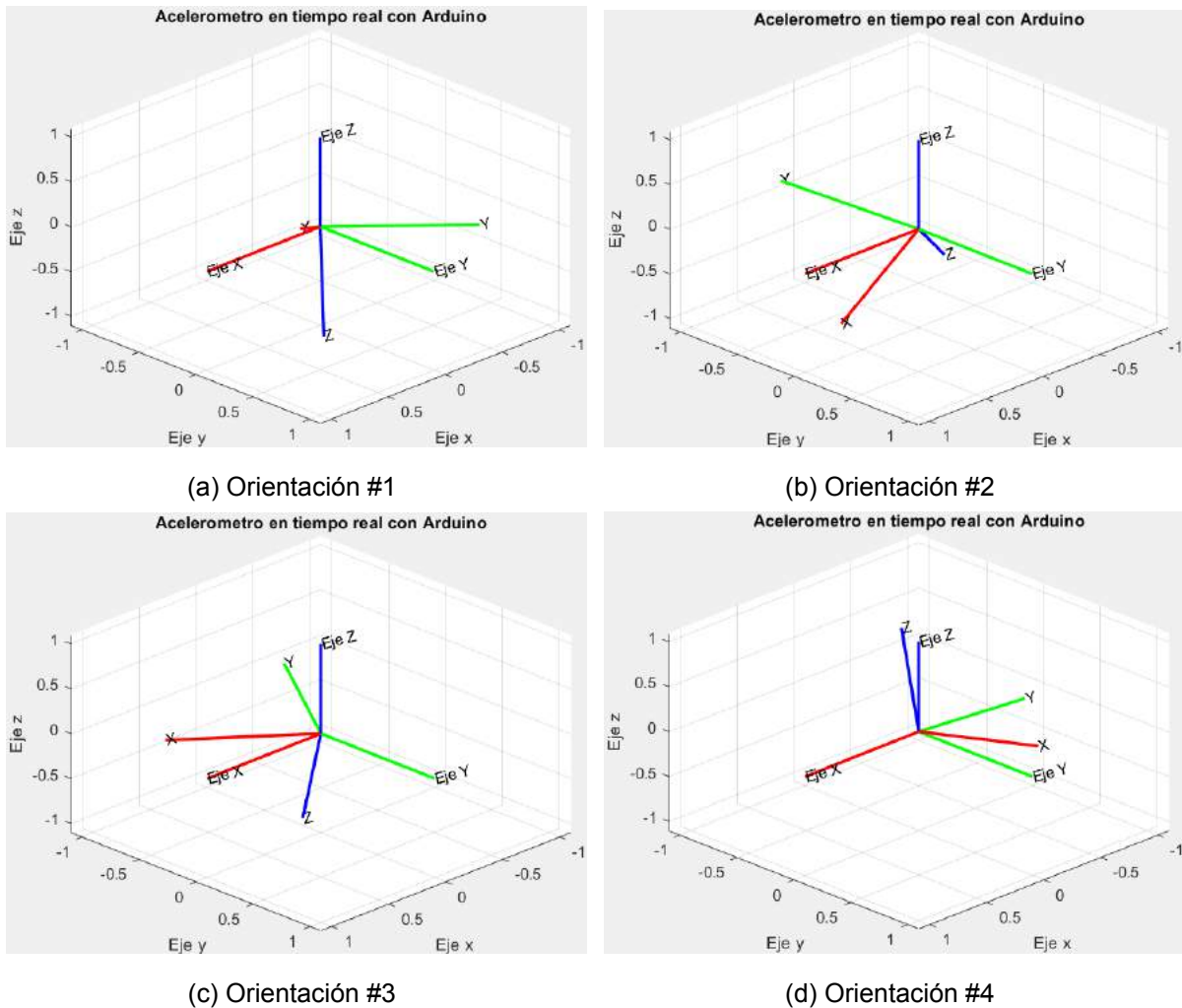
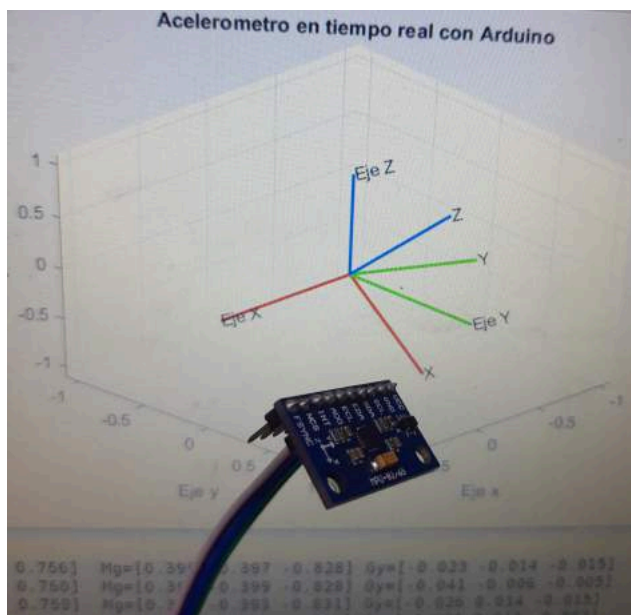


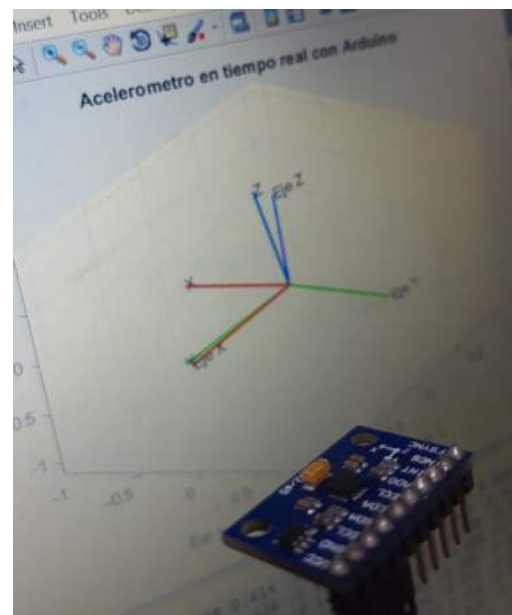
Figura 3.7: Plot de diversas orientaciones del sensor MPU9250

Ahora bien, en la Figura 3.8 se muestra la orientación estimada por el filtro de Madgwick, y la orientación real del sensor. Recordar que las mediciones del sensor están referenciadas en la PCB, tal como se menciona en el Subtema 2.4.1. En estas figuras se puede observar que con el filtro de Madgwick se puede estimar la orientación con buena precisión. El código empleado para la captura de estas figuras es **MATLAB / ArduinoUnrealUDP / Madgwick / Madgwick.m**





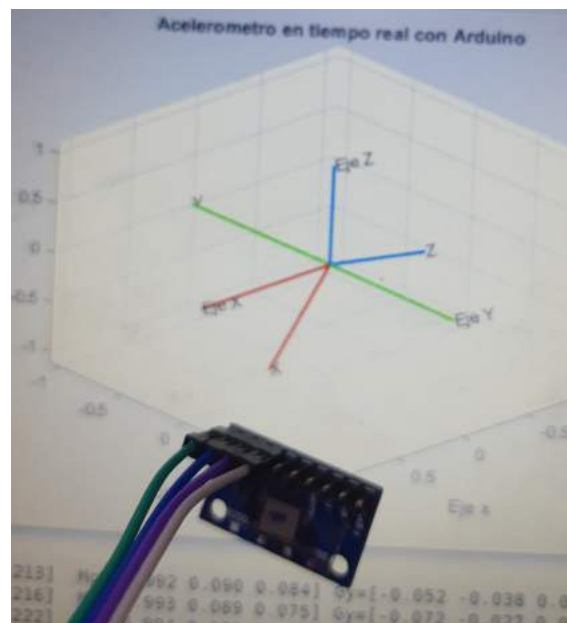
(a) Orientación #1



(b) Orientación #2



(c) Orientación #3



(d) Orientación #4

Figura 3.8: Comparación entre la orientación real del sensor MPU9250 y la orientación estimada por el filtro de Madgwick

De igual forma, se puede consultar un vídeo en YouTube que se encuentra dentro de la playlist del siguiente [link](#)<sup>3</sup>. En este vídeo se podrán observar en funcionamiento a las partes involucradas (Matlab, Arduino y el sensor MPU9250).

### 3.3. Unreal - Matlab - Arduino - MPU9250

En la Figura 3.9 se puede observar el resultado de ejecutar los códigos ubicados en los archivos: **MATLAB / ArduinoUnrealUDP / FINAL\_UDPArduinoUnreal.m**, para el caso de Matlab; y para Arduino se empleó el archivo **ARDUINO CODE / FINAL\_GENERAL / FINAL\_GENERAL.ino**.

El código de Unreal necesario para lograr el funcionamiento mostrado en la Figura 3.9 no se incluye en el presente documento ni en los repositorios de GitHub, pero es muy posible que se pueda lograr el mismo funcionamiento con unas cuantas modificaciones al código presentado en la Sección 2.9. Dichas modificaciones consistirían en quitar y reacomodar bloques, ya no es necesario agregar ningún bloque más.

De nueva cuenta, en el mismo [link](#)<sup>4</sup> de la playlist ya mencionada de YouTube se pueden encontrar dos vídeos que fueron grabados al tiempo de ejecutar los códigos que logran lo mostrado en la Figura 3.9.

---

<sup>3</sup>[https://www.youtube.com/playlist?list=PL\\_8Avt9po9eWGSbH3mykkuQFoTwtPQDQI](https://www.youtube.com/playlist?list=PL_8Avt9po9eWGSbH3mykkuQFoTwtPQDQI)

<sup>4</sup>Ver nota 3





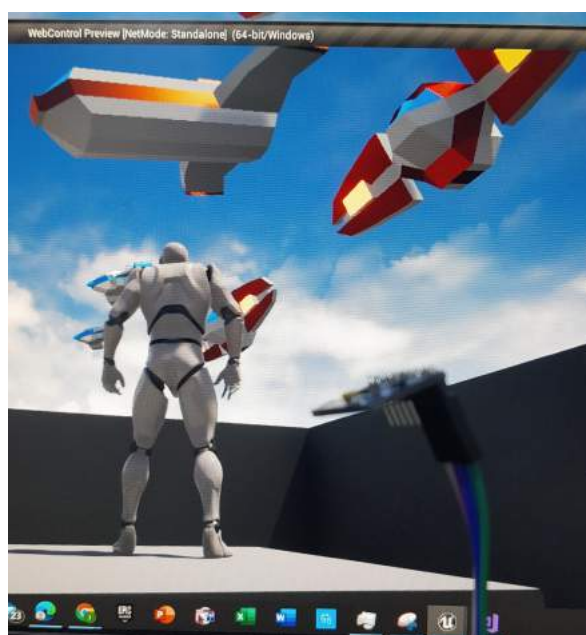
(a) Orientación #1



(b) Orientación #2



(c) Orientación #3



(d) Orientación #4

Figura 3.9: Comparación entre la orientación del sensor MPU9250 y la orientación transmitida por Matlab mediante UDP a Unreal.

### 3.4. Traje Mocap

El chaleco ajustable (véase la Figura 3.10), para fines de elaborar un prototipo, consiste en un chaleco de brigadista ajustable porque tiene las siguientes ventajas (además, de las ya mencionadas anteriormente):

- 4 cierres con hebilla buckle con cinta ajustable (2 por cada lado)
- 2 compartimentos inferiores (en donde se puede introducir la Raspberry Pi 3B)
- 1 cierre principal con cremallera

La desventaja principal de usar un chaleco es que los cables de los brazos y piernas se quedan colgando, por lo cual se puede considerar laborioso el tener que guardarlo y volverlo a emplear, además de que se pueden enredar los cables.

El prototipo del traje final se puede encontrar en la Figura 3.10, donde se puede observar la ubicación recomendada de las vendas, o bien apreciar cómo los cables conectan los sensores de las extremidades tras rodear a estas en forma de espiral. Esto último permite que la longitud del cable no esté sujeta a cierta talla/dimensión específica de los brazos, además, el traje será más fácil de poner y quitar en una persona con discapacidad.



(a) Lateral izquierdo  
del traje mocap



(b) Lateral derecho  
del traje mocap



(c) Raspberry Pi 3B + traje mocap

Figura 3.10: Avances del traje Mocap

### 3.5. Resultados finales

Tal como ya se ha ido mencionando a lo largo del documento, existe una playlist en YouTube sobre algunas pruebas básicas entre Unreal, sensores MPU9250, Arduino, Matlab y Raspberry Pi. El [link](#)<sup>5</sup> se encuentra en el pie de página para la versión impresa del documento, en el link encontrarán un vídeo del cual se tomaron capturas de pantalla que se incluyeron en el presente informe de residencias.

Los códigos que se emplearon para obtener lo que se muestra en las imágenes de esta sección son los que se muestran en la Sección 2.9, y los que se describen en la Sección 2.8 (el código del proyecto Qt en Raspberry se puede encontrar en la carpeta **QT** de [GitHub](#)<sup>6</sup>).

Primero que nada, mencionar que el chaleco prototipo solo contiene 3 sensores, mismos que se asignan en el siguiente orden: 1) Esternón, 2) Clavícula izquierda y 3) clavícula derecha.

En la Figura 3.11 se puede apreciar el comienzo de ambos programas (esto ocurre hasta que se presiona el botón de arranque en un control Bluetooth conectado al ordenador), tanto el de la RPi como el de Unreal, en la imagen se muestra como ambos programas están siendo visualizados por el mismo monitor, ya que existe una conexión inalámbrica vía WiFi entre la Raspberry Pi y el ordenador.

---

<sup>5</sup>[https://www.youtube.com/playlist?list=PL\\_8Avt9po9eWGSbH3mykkuQFoTwtPQDQI](https://www.youtube.com/playlist?list=PL_8Avt9po9eWGSbH3mykkuQFoTwtPQDQI)

<sup>6</sup>[https://github.com/alejandroivan10/ResidenciaProfesional\\_IvanAlejandroMunozVera](https://github.com/alejandroivan10/ResidenciaProfesional_IvanAlejandroMunozVera)

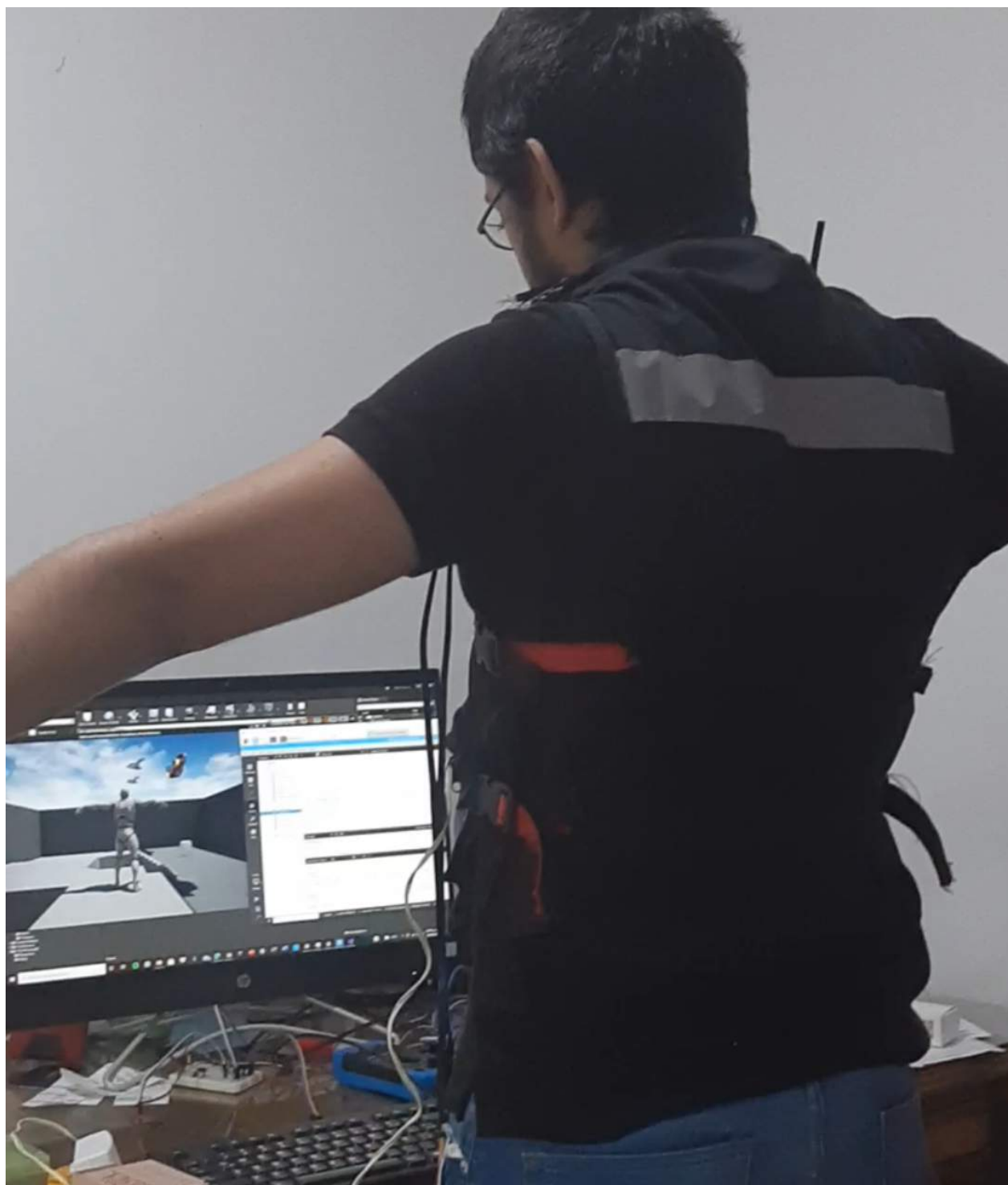


Figura 3.11: Alineación entre el portador del traje mocap y el personaje de Unreal Engine



Una vez que ya se terminaron de alinear los sensores en la Raspberry Pi, esta última comienza a mandar los datagramas UDP a Unreal con la información de las orientaciones de los sensores. Tras recibir los datagramas, Unreal los asigna a las partes que corresponden del cuerpo (observen los blueprints de la Sección 2.9.3), y luego los presenta como parte de la animación del personaje.

En la Figura 3.12 se muestra la captura de movimiento resultante de cuando el portador del traje está parado de frente a la computadora, aunque inclinado hacia adelante. Nótese que solo las orientaciones del esternón y las clavículas del portador son los valores que se están asignando en el videojuego de Unreal.

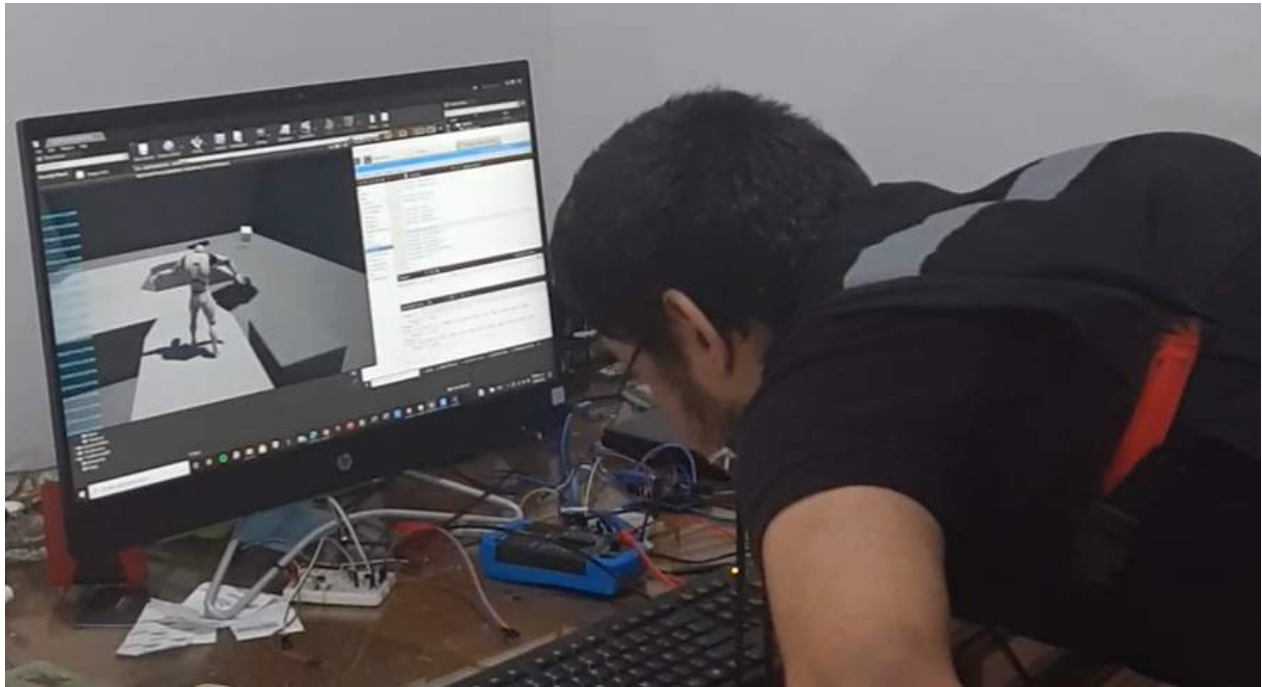


Figura 3.12: Captura del movimiento en Unreal cuando el portador del traje se inclina hacia el frente.

En la Figura 3.13 se muestra la captura de movimiento de cuando el portador está parado de frente a la computadora, aunque inclinado hacia atrás. Al no haber sensores en los brazos para el prototipo, estos pueden ser usados con total libertad.

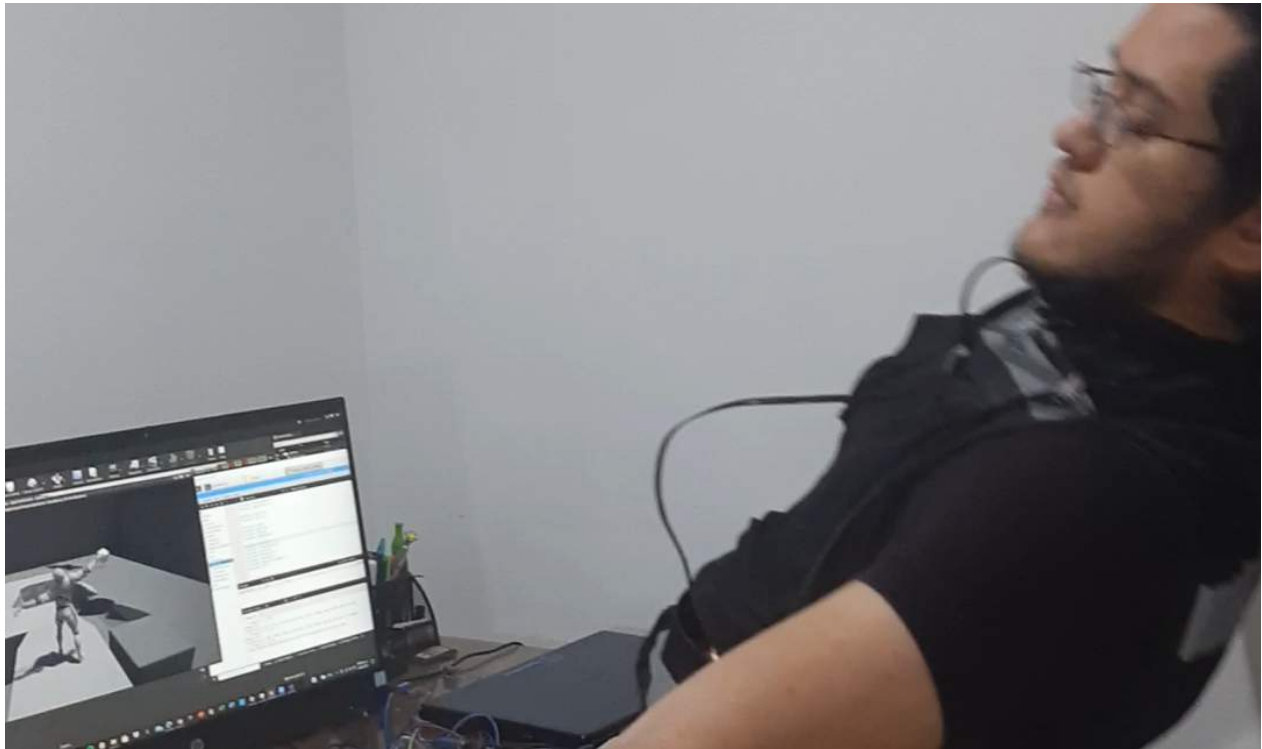


Figura 3.13: Captura del movimiento en Unreal cuando el portador del traje se inclina hacia atrás.

De igual forma, en la Figura 3.14 se muestra la captura de movimiento cuando el portador gira el torso 90° en el sentido contrario de las manecillas del reloj (o bien, un giro de 90° hacia la izquierda).

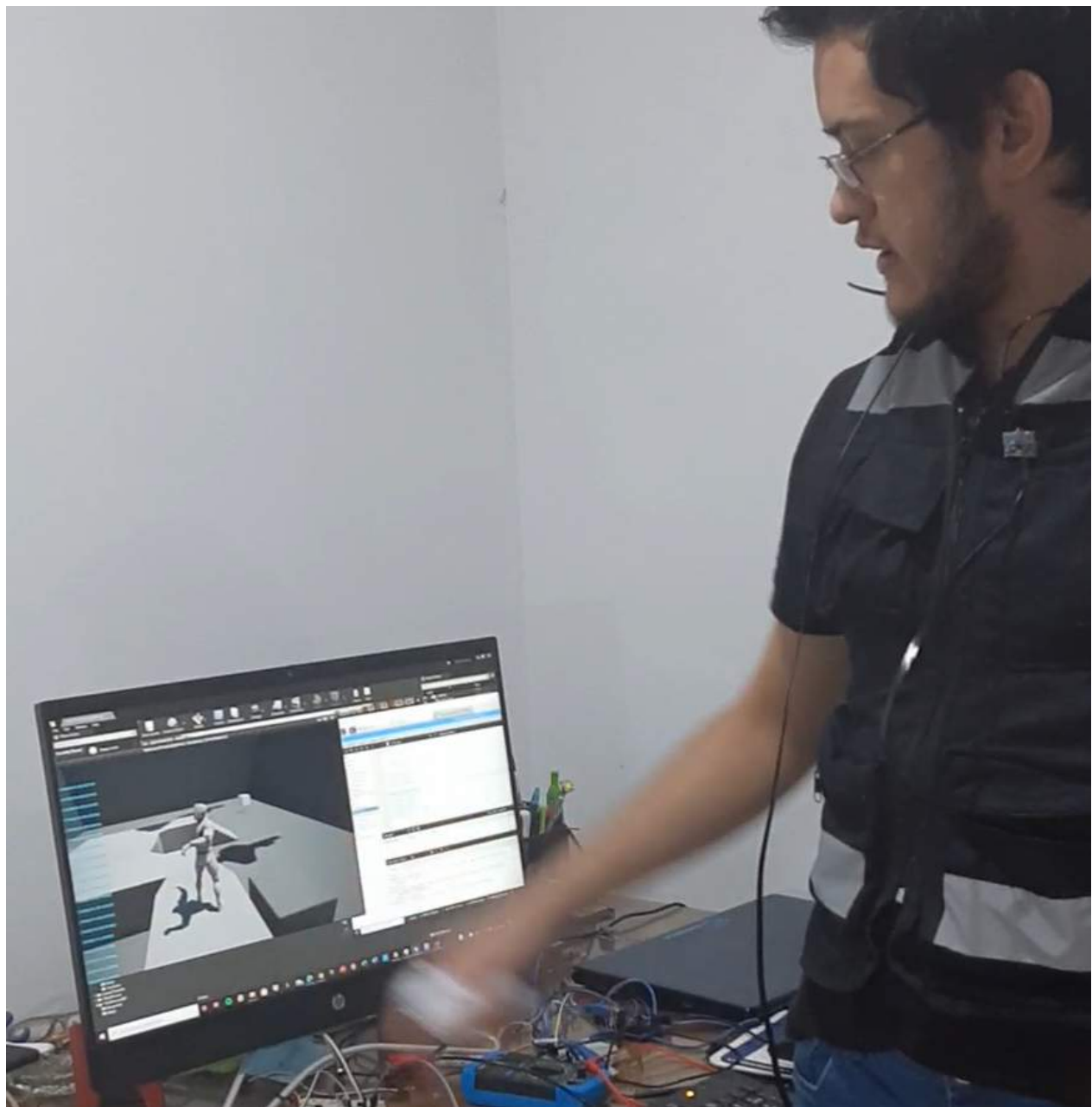


Figura 3.14: Captura del movimiento en Unreal cuando el portador del traje se gira hacia la izquierda.

Y, por último, en la Figura 3.15 se presenta la orientación capturada por Unreal Engine para cuando el portador gira 90 grados en el sentido de las manecillas del reloj (o bien, un giro de 90° hacia la derecha).



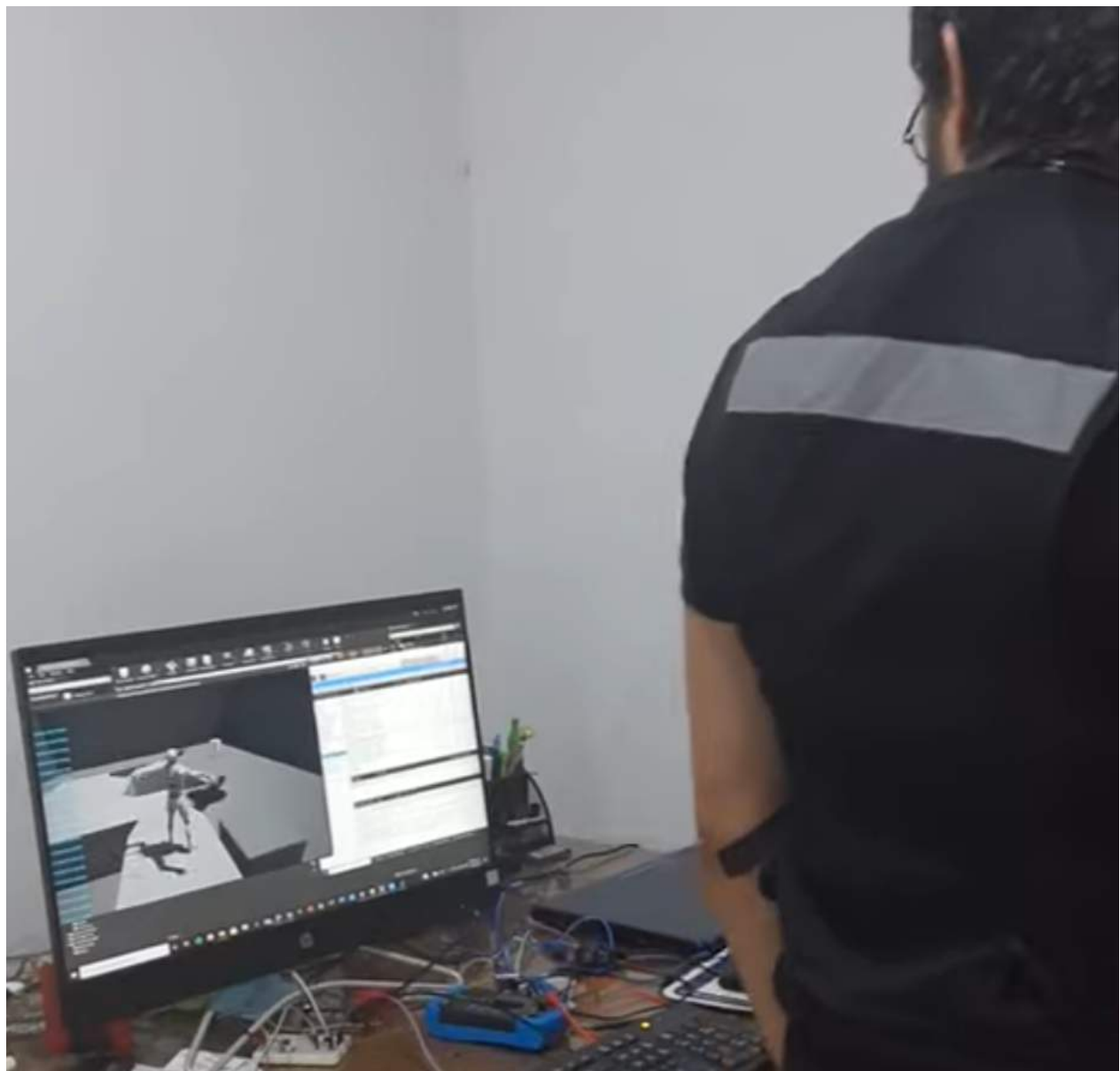


Figura 3.15: Captura del movimiento en Unreal cuando el portador del traje gira hacia la derecha.

# Conclusiones y Trabajo a Futuro

## Conclusiones

Tanto en lo personal como en lo técnico, me llevo muchísimos aprendizajes luego de realizar este proyecto. Cabe mencionar que nunca había aprovechado tantos recursos de la Raspberry Pi (CPU, memoria RAM, puerto SPI, pines GPIO, periféricos de red WiFi, etc.) como lo he hecho en esto proyecto.

A unos días de entregar este informe, tengo claro que un proyecto de diseño y desarrollo de un traje de captura de movimiento, para el fin que convenga (p. ej., terapia), debería ser realizado por más de una persona, o bien en un tiempo mayor a 6 meses (es lo que me ha tomado a mí).

En el tramo final de este proyecto fue evidente que el desarrollo (y no el diseño) del traje fue lo que más contratiempos generó, algunos por errores o desconocimiento del autor (p. ej., desliz con un apuntador de memoria, desconocimiento sobre cómo trabajar con rValues, etc.).

Por último, ajeno al proyecto, quiero mencionar que haber trabajado con  $\text{\LaTeX}$  es algo hermoso, pero muy tortuoso al principio (vaya que me costó algo de tiempo acostumbrarme). Además, haber usado Latex me será muy útil en el futuro.

Hubo muchos aprendizajes que se obtuvieron durante la marcha. Sin duda, lo aquí aprendido me será de utilidad para futuros proyectos, trabajar con un proyecto tan amplio me hizo aprender cosas que con proyectos más cortos sería difícil (por ejemplo, planificar mejor los proyectos de gran envergadura).

## Trabajo a futuro

Me gustaría hacer algunas recomendaciones para aquel que quiera continuar con este proyecto, empezando por una de dos cosas, o bien conseguir un equipo de cómputo con mejores características (+ de 4 núcleos de procesamiento, o + de 1.2 GHz de velocidad), o bien optimizar el código (entre las cuales está el emplear las capacidades de la GPU, las cuales no se aprovecharon).

Recomiendo que el diseño y desarrollo del traje lo hagan en base a otro sensor MARG, y es que encontrar sensores MPU9250 completamente funcionales hoy en día es muy complicado, el sensor MPU9250 ya se encuentra al final de su ciclo de vida (EOL, por sus siglas en inglés). Este sensor no lo pude encontrar en las tiendas más populares de electrónica de todo León, Gto., y de 6 proveedores a los que se les realizó un pedido online, solo 2 me mandaron algunos buenos sensores (incluso, había sensores de estos 2 proveedores que tenían un comportamiento muy irregular).

Otra cosa, sería bueno que los sensores MARG fueran plug & play, ya que cuando se conectan muchos sensores en la red, algunos de estos empiezan a mostrar comportamientos irregulares que no se presentaban cuando se trabajan estos sensores en solitario (uno por uno). Recomendando tener a la mano un osciloscopio para poder encontrar las causas de los fallos, seguro y este instrumento me hubiese sido útil tenerlo.

## Problemas que se presentaron con el traje mocap:

En este proyecto se presentaron muchos problemas, algunos ya se resolvieron como el de los Rvalues y Lvalues. Este problema se presentó en las funciones que devolvían un objeto instanciado dentro de una clase (p. ej., de la clase Vector o Matriz). Haberlo resuelto fue muy útil para poder hacer, en una misma línea de código, múltiples operaciones como sumas, restas, multiplicaciones, etc., con objetos del tipo Vector y Matriz (esto ahorro

mucho tiempo y espacio al programar código, además, así el código luce más elegante y comprensible).

En cuanto al problema de escritura en direcciones inadecuadas de memoria, este se debió a un error humano, ya que en ocasiones para probar si funciona el código, suelo modificar temporalmente ciertas partes del mismo; el problema ocurre cuando esa sección de código no se vuelve a dejar como estaba antes de realizar las pruebas; por tanto, en la función `MagReadByte()` se modificó la cantidad de valores a leer, haciendo que los guardara/escribiera en direcciones de memoria que no deben de estar permitidas.

Algunos otros considerables contratiempos se debieron por causas externas. Por ejemplo, ciertos proveedores me vendieron sensores de modelos que no contienen magnetómetro o, incluso, sensores defectuosos; otro contratiempo fue el calibre del cable que vendía cierto proveedor online, se anunciaba como calibre 23 AWG, pero al recibir el producto se podía observar impreso en el aislante que era calibre 28 AWG (calibre con el cual fue muy difícil trabajar, por lo quebradizo que era).

Al día de hoy se han comprado 37 sensores Mpu9250, de los cuales en 8 ya se procedió con la devolución, otros 18 se tenían que devolver, pero se venció el plazo dado que se entregaron en diciembre y se probó su funcionamiento hasta mayo-junio

Para llegar a la conclusión de que 8 sensores se tenían que devolver, se analizó y, posteriormente, ejecutó el código `hideakitai` que se encuentra en Github. Además, anteriormente ya se habían realizado diferentes pruebas por SPI e I2C para demostrar que los sensores que me entregaron no eran del modelo que se había pedido por internet, sino que eran del modelo MPU6500 (el cual no cuenta con Magnetómetro). Lo anterior se menciona en los vídeos que se enviaron a los proveedores, en el vídeo se justifica el motivo de su devolución, dejó un [link](#) para poder acceder a ellos en YouTube <sup>7</sup>.

---

<sup>7</sup>[https://www.youtube.com/playlist?list=PL\\_8Avt9po9eW3ZPi79-Xft01zYMcPYU5](https://www.youtube.com/playlist?list=PL_8Avt9po9eW3ZPi79-Xft01zYMcPYU5)

Otro reto interesante fue soldar los cables calibre AWG28 porque se rompen muy fácilmente, de aquí la necesidad de usar abundante silicona para evitar la flexión de los cables, sobre todo, en los empalmes eléctricos donde el cobre pierde su protección plástica aislante y, en consecuencia, esto se presta a que se vayan desprendiendo los hilos del cable uno por uno.

# Competencias Aplicadas

El perfil de egreso del estudiante en ingeniería mecatrónica del TecNM campus León consiste en 8 puntos, los cuales a continuación se relacionan con, al menos, un objetivo específico.

En cuanto al punto #1 del perfil de egreso: *“Ejerce su profesión legal y responsablemente para cumplir con las normas nacionales e internacionales que apliquen”*, este se cumplió dada su semejanza al objetivo específico #4, el cual consiste en desarrollar un traje para capturar el movimiento empleando conocimientos afines a la ingeniería en mecatrónica y apegado a las normas que apliquen. El traje elaborado en el presente proyecto contiene equipo electrónico que se encuentra apegado a normas o certificaciones como FCC, UL listed, IC, EC, IEEE 802.11 b/g/n, ROHS, entre otras.

En cuanto al punto #2 del perfil de egreso: *“Analiza, sintetiza, diseña, simula, construye e innova productos, procesos, equipos y sistemas mecatrónicos, para impactar positivamente en su entorno con una actitud investigadora, de acuerdo a las necesidades tecnológicas, sociales actuales y emergentes”*, el objetivo específico #3 y #4 son los que mejor impactan este punto del perfil de egreso, ya que los diseños y desarrollos logrados en este proyecto podrán tener un impacto favorable para su uso en terapia y rehabilitación, ya que este proyecto podrá brindar información muy valiosa para estudiar las mejorías del paciente, diagnosticar alguna enfermedad, o bien estimular el tratamiento mediante experiencias virtuales que sean placenteras para el paciente. Los diseños, análisis, pruebas en equipos, procesos y sistemas se hicieron pensando para su uso terapéutico.

En cuanto al punto #3 del perfil de egreso: *“Instala, opera, optimiza, controla y mantiene sistemas mecatrónicos integrando tecnologías mecánicas, eléctricas, electrónicas y herramientas computacionales”*, el objetivo específico #4 es acorde a lo ya mencionado porque

en este proyecto se instaló, operó, optimizó y controló software y hardware, se emplearon los conocimientos afines a mecatrónica para las consideraciones mecánicas (flexibilidad, resistencia, etc.), eléctricas/electrónicas (ruido, filtros, registros, comunicaciones, etc.) y computacionales (apuntadores de memoria, POO, macros, etc.)

En cuanto al punto #4 del perfil de egreso: *“Planifica, evalúa, genera, administra y transfiere proyectos industriales y de carácter social para el desarrollo tecnológico del país”*, este punto va muy asociado al objetivo específico #1 porque en ambos se menciona la importancia de la planeación, administración y evaluación durante el proyecto; como se había mencionado, durante el proyecto se elaboró un cronograma de 450 actividades y 120 objetivos específicos, mismo que sirvió de base para administrar y evaluar el desarrollo del mismo. Este proyecto podrá ayudar al avance tecnológico mexicano, ya que presenta soluciones innovadoras para el sector salud, ofreciendo herramientas hardware y software de actualidad.

En cuanto al punto #5 del perfil de egreso: *“Participa, coordina y/o dirige grupos multidisciplinarios a través del trabajo en equipo para asegurar la calidad, eficiencia, productividad y rentabilidad en la implementación de proyectos mecatrónicos con sentido de responsabilidad de su entorno social y cultural para un desarrollo sustentable”*, este punto se relaciona al objetivo específico #5 porque vincular el presente proyecto en el sector salud conlleva, implícitamente, la colaboración con profesionistas de otras áreas del conocimiento para que el proyecto tenga un impacto mayor, y para asegurar calidad, eficiencia y rentabilidad. Este proyecto por ser de carácter terapéutico, se desarrolló con responsabilidad hacia el entorno social y cultural. Cabe mencionar que este proyecto se llevó a cabo con la asesoría de la Dra. Rosario, la cual ya tiene unos cuantos años trabajando en proyectos en favor de personas con alguna discapacidad o trastorno psicomotriz. Además, se colaboró con profesionales de otras áreas para conocer sus ideas y propuestas en favor de las personas con discapacidad.

En cuanto al punto #6 del perfil de egreso: *“Posee capacidades de liderazgo, comunicación, interrelaciones personales para transmitir ideas, facilitar conocimientos y trabajar con responsabilidad colectiva para la solución de problemas y desarrollo de proyectos con un sentido crítico y autocrítico”*, de igual forma este punto se relaciona con el objetivo específico #5 porque vincular este proyectos con personas que tengan alguna discapacidad no es una tarea que se pueda hacer solo, hay que tener habilidades interpersonales, comunicación y liderazgo, ya que sumando los conocimientos y las experiencias de otras personas es como se puede enriquecer este proyecto. Durante la residencia se tuvieron aprox. 20 reuniones con personas con conocimientos de diferentes áreas, esto fue muy útil para conocer sus ideas y exponer las mías, de esta forma pude enriquecer el presente proyecto.

En cuanto al punto #7 del perfil de egreso: *“Desarrolla proyectos con un espíritu innovador, emprendedor y comprometido con su actualización profesional continua y autónoma, para estar a la vanguardia en los cambios científicos y tecnológicos que se dan en el ejercicio de su profesión”*, se relaciona a los objetivos específicos #2 y #4, ya que para el diseño y desarrollo del mismo traje de captura de movimiento fue necesario trabajar con lo último en tecnología, misma que fuera accesible y con prestaciones y rendimientos tan altos como los de un celular o computadoras de hace unos pocos años, razón por la cual se empleó una placa Raspberry Pi 3B.

En cuanto al punto #8 del perfil de egreso: *“Interpreta información técnica de las áreas que componen la Ingeniería Mecatrónica para la transferencia, adaptación, asimilación e innovación de tecnologías de vanguardia”*, este punto se relaciona con el objetivo específico #2 y #3 porque cuando se busca producir tecnología de vanguardia, esto implica que se debe recopilar información de actualidad para asimilarla y adaptarla, o bien innovar en favor del desarrollo del proyecto. Cabe mencionar que para ese proyecto se han leído y analizado un aproximado de 200 páginas tan solo por las hojas de datos, aunado a todos los artículos científicos y patentes revisados.



## Referencias

- Arduino. (2018). *What is arduino?* Recuperado el 25 de junio del 2021, de <https://www.arduino.cc/en/Guide/Introduction>
- Barrientos, A., Peñin, L. F., y Carlos Balaguer, R. A. (2007). *Fundamentos de robótica*. McGraw-Hill/interamericana de España. (ISBN: 978-84-481-5636-7)
- BurnfieldYu, J. M., Shu, Taylor, A. P., Buster, T. W., y Nelson, C. A. (2012, mayo 12). *Rehabilitation and exercise machine*. Google Patents. (US Patent 8,177,688 B2)
- Canavan, C., y Hughes, G. (2020, octubre 14). *System and method for identifying and interpreting repetitive motions*. Google Patents. (US Patent 10,799,760 B2)
- Cavallo, A., Cirillo, A., Cirillo, P., De Maria, G., Falco, P., Natale, C., y Pirozzi, S. (2014). Experimental comparison of sensor fusion algorithms for attitude estimation. *IFAC Proceedings Volumes*, 47(3), 7585-7591. Recuperado de <https://www.sciencedirect.com/science/article/pii/S1474667016428089> (19th IFAC World Congress) doi: <https://doi.org/10.3182/20140824-6-ZA-1003.01173>
- Cristóbal, C. ., y Humberto, R. (2017). Evaluation of data fusion algorithms for attitude estimation of unmanned aerial vehicles. *I+D Tecnológico*, 13(2), 90-99.
- Diymore Alice1101983. (s.f.). *Mpu-9250 3-axis accelerometer, gyroscope and magnetometer sensor module* [Imagen]. Aliexpress. Recuperado el 6 de julio del 2021, de <https://es.aliexpress.com/item/32668512794.html?spm=a2g0s.9042311.0.0.4d9963c0otkHKc>
- Entorno de desarrollo integrado. (2021). *Wikipedia, la enciclopedia libre*. Recuperado el 27 de junio del 2021, de [https://es.wikipedia.org/w/index.php?title=Entorno\\_de\\_desarrollo\\_integrado&oldid=136625168](https://es.wikipedia.org/w/index.php?title=Entorno_de_desarrollo_integrado&oldid=136625168)
- Epic Games. (s.f.). *Unreal engine*. Recuperado el 6 de julio del 2021, de <https://www.unrealengine.com/en-US/unreal>
- Feng, Y., Ji, M., Xiao, J., Yang, X., Zhang, J. J., Zhuang, Y., y Li, X. (2015). Mining spatial-temporal patterns and structural sparsity for human motion data denoising.

- IEEE Transactions on Cybernetics*, 45(12), 2693-2706. doi: 10.1109/TCYB.2014.2381659
- Freescale Semiconductor. (2015). *Freescale sensor fusion library for kinetis mcus*.
- Gwana-ro. (2013, abril 15). *A treatment device for hemiplegia*. Google Patents. (KR Patent 101511427B1)
- Herbfargus. (2016, 13 de marzo). *File:raspberry pi 3 model b.png* [Imagen]. Wikimedia Commons, the free media repository. Recuperado el 20 de junio del 2021, de [https://commons.wikimedia.org/w/index.php?title=File:Raspberry\\_Pi\\_3\\_Model\\_B.png&oldid=444138768](https://commons.wikimedia.org/w/index.php?title=File:Raspberry_Pi_3_Model_B.png&oldid=444138768)
- Jia, Y.-B. (2020). *Quaternions*. Recuperado el 13 de mayo del 2021, de <https://web.cs.iastate.edu/~cs577/handouts/quaternion.pdf>
- Komatireddy, R., Hutchins, S., y Shah, M. (2012, agosto 12). *Systems, apparatus and methods for non-invasive motion tracking to augment patient administered physical rehabilitation*. Google Patents. (CA Patent 2844651C)
- Ma, M., Proffitt, R., y Skubic, M. (2018, 08). Validation of a kinect v2 based rehabilitation game. *PLOS ONE*, 13(8), 1-15. Recuperado de <https://doi.org/10.1371/journal.pone.0202338> doi: 10.1371/journal.pone.0202338
- Madgwick, S. (2010). An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Report x-io and University of Bristol (UK)*, 25, 113–118.
- Matlab. (s.f.). *Matlab*. Recuperado el 5 de julio del 2021, de <https://la.mathworks.com/products/matlab.html>
- Microsoft. (s.f.). *Visual studio 2019*. Recuperado el 5 de julio del 2021, de <https://visualstudio.microsoft.com/es/>
- Oomlout. (2015, 01 de febrero). *Ardu-uno-03-front* [Imagen]. Flickr. Recuperado el 30 de junio del 2021, de <https://www.flickr.com/photos/snazzyguy/16421989212/>
- Phillips, J. M. (2017). *An introduction to data analysis through a geometric lens*. Recuperado el 7 de julio del 2021, de <https://www.cs.utah.edu/~jeffp/teaching/IDABook/IDA-GL.pdf>

- ProtoSupplies. (s.f.). *Diymore 9-eje 9dof bmx055 imu precisión integrada sensor de actitud de la cii i2c módulo de interfaz para arduino* [Imagen]. Recuperado el 6 de julio del 2021, de <https://protosupplies.com/product/mpu-9250-3-axis-accel-gryo-mag-sensor-module/>
- Ramírez, L. G. C., Jiménez, G. S. A., y Carreño, J. M. (2014). *Sensores y actuadores. aplicaciones con arduino*. Grupo Editorial Patria. (ISBN: 978-607-438-936-4)
- Raspberry Pi. (2021). *Wikipedia, la enciclopedia libre*. Recuperado el 25 de junio del 2021, de [https://es.wikipedia.org/w/index.php?title=Raspberry\\_Pi&oldid=136097758](https://es.wikipedia.org/w/index.php?title=Raspberry_Pi&oldid=136097758)
- Rvelandia. (2008, 26 de mayo). *File:tcp-ip osi comparison table.jpg*. Wikimedia Commons, the free media repository. Recuperado el 02 de junio del 2021, de [https://commons.wikimedia.org/w/index.php?title=File:TCP-IP\\_OSI\\_comparison\\_table.jpg&oldid=446994336](https://commons.wikimedia.org/w/index.php?title=File:TCP-IP_OSI_comparison_table.jpg&oldid=446994336)
- Singer, Y. (2016, febrero). *Am 221: Advanced optimization. lecture 9*. Recuperado el 7 de julio del 2021, de [https://people.seas.harvard.edu/~yaron/AM221-S16/lecture\\_notes/AM221\\_lecture9.pdf](https://people.seas.harvard.edu/~yaron/AM221-S16/lecture_notes/AM221_lecture9.pdf)
- Tanenbaum, A. S., y Wetherall, D. J. (2012). *Redes de computadoras*. PEARSON EDUCACIÓN. (ISBN: 978-607-32-0818-5)
- TDK InvenSense. (2016). *Mpu-9250 product specification revision 1.1*. Recuperado el 02 de junio del 2021, de <https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf>
- The Qt Company. (s.f.). Qt. Recuperado el 6 de julio del 2021, de <https://www.qt.io/product>
- Tsilomitrou, O., Gkountas, K., Evangeliou, N., y Dermatas, E. (2021). Wireless motion capture system for upper limb rehabilitation. *Applied System Innovation*, 4(1), 14.
- Zavadsky, V., y Sherstyuk, M. (2011, junio 11). Wireless game controller for strength training and physiotherapy. (US Patent 7,967,728 B2)

# Glosario de Términos

Acknowledgment	Es una señal que se transmite entre procesos de comunicación o dispositivos para indicar el reconocimiento o la recepción de un mensaje.
Framework	Es un esquema o estructura desarrollado previamente que se puede aprovechar para desarrollar un proyecto.
Blueprint	Es un sistema completo de secuencias de comandos para juegos hechos con Unreal Editor, utiliza una interfaz basada en nodos.
Plugins	Son pequeños programas complementarios que amplían las funciones de aplicaciones web y programas de escritorio
Grados de libertad	Son el número mínimo de velocidades generalizadas independientes necesarias para definir el estado cinemático de un mecanismo
Placa de Circuito impreso (PCB)	Es una superficie constituida por caminos, pistas o buses de material conductor laminadas sobre una base no conductora.

# Acrónimos

Mocap	Motion Capture
MEMS	Micro-Electromechanical Systems
MARG	Magnetic, Angular Rate and Gravity
IMU	Inertial Measurement Unit
AHRS	Attitude and Heading Reference Systems
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IDE	Integrated Development Environment
API	Application Programming Interface
PCB	Printed Circuit Board
RPi	Raspberry Pi
DOF	Degrees of Freedom
MWE	Minimal Working Example
SSID	Service Set Identifier
SSH	Secure Shell
GUI	Graphical User Interface

# Anexos

# Apéndice A

## Hojas de datos: MPU-9250

### A.1. MPU-9250 Product Specification

 <i>Sensing Everything</i>	<b>InvenSense Inc.</b> 1745 Technology Drive, San Jose, CA 95110 U.S.A. Tel: +1 (408) 988-7339 Fax: +1 (408) 988-8104 Website: <a href="http://www.invensense.com">www.invensense.com</a>	Document Number: PS-MPU-9250A-01 Revision: 1.1 Release Date: 06/20/2016
--	--	---

### **MPU-9250 Product Specification Revision 1.1**

## Descripción del producto

	<b>MPU-9250 Product Specification</b>	Document Number: PS-MPU-9250A-01 Revision: 1.1 Release Date: 06/20/2016
---	---------------------------------------	---

### 1.2 Purpose and Scope

This document provides a description, specifications, and design related information on the MPU-9250 MotionTracking device. The device is housed in a small 3x3x1mm QFN package.

Specifications are subject to change without notice. Final specifications will be updated based upon characterization of production silicon. For references to register map and descriptions of individual registers, please refer to the MPU-9250 Register Map and Register Descriptions document.

### 1.3 Product Overview

MPU-9250 is a multi-chip module (MCM) consisting of two dies integrated into a single QFN package. One die houses the 3-Axis gyroscope and the 3-Axis accelerometer. The other die houses the AK8963 3-Axis magnetometer from Asahi Kasei Microdevices Corporation. Hence, the MPU-9250 is a 9-axis MotionTracking device that combines a 3-axis gyroscope, 3-axis accelerometer, 3-axis magnetometer and a Digital Motion Processor™ (DMP) all in a small 3x3x1mm package available as a pin-compatible upgrade from the MPU-6515. With its dedicated I<sup>2</sup>C sensor bus, the MPU-9250 directly provides complete 9-axis MotionFusion™ output. The MPU-9250 MotionTracking device, with its 9-axis integration, on-chip MotionFusion™, and run-time calibration firmware, enables manufacturers to eliminate the costly and complex selection, qualification, and system level integration of discrete devices, guaranteeing optimal motion performance for consumers. MPU-9250 is also designed to interface with multiple non-inertial digital sensors, such as pressure sensors, on its auxiliary I<sup>2</sup>C port.

MPU-9250 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs, three 16-bit ADCs for digitizing the accelerometer outputs, and three 16-bit ADCs for digitizing the magnetometer outputs. For precision tracking of both fast and slow motions, the parts feature a user-programmable gyroscope full-scale range of  $\pm 250$ ,  $\pm 500$ ,  $\pm 1000$ , and  $\pm 2000^\circ/\text{sec}$  (dps), a user-programmable accelerometer full-scale range of  $\pm 2g$ ,  $\pm 4g$ ,  $\pm 8g$ , and  $\pm 16g$ , and a magnetometer full-scale range of  $\pm 4800\mu\text{T}$ .

Other industry-leading features include programmable digital filters, a precision clock with 1% drift from  $-40^\circ\text{C}$  to  $85^\circ\text{C}$ , an embedded temperature sensor, and programmable interrupts. The device features I<sup>2</sup>C and SPI serial interfaces, a VDD operating range of 2.4V to 3.6V, and a separate digital IO supply, VDDIO from 1.71V to VDD.

Communication with all registers of the device is performed using either I<sup>2</sup>C at 400kHz or SPI at 1MHz. For applications requiring faster communications, the sensor and interrupt registers may be read using SPI at 20MHz.

By leveraging its patented and volume-proven CMOS-MEMS fabrication platform, which integrates MEMS wafers with companion CMOS electronics through wafer-level bonding, InvenSense has driven the package size down to a footprint and thickness of 3x3x1mm, to provide a very small yet high performance low cost package. The device provides high robustness by supporting 10,000g shock reliability.

### 1.4 Applications

- Location based services, points of interest, and dead reckoning
- Handset and portable gaming
- Motion-based game controllers
- 3D remote controls for Internet connected DTVs and set top boxes, 3D mice
- Wearable sensors for health, fitness and sports



# Descripción del producto

	<b>MPU-9250 Product Specification</b>	Document Number: PS-MPU-9250A-01 Revision: 1.1 Release Date: 06/20/2016
---	---------------------------------------	---

## 2 Features

### 2.1 Gyroscope Features

The triple-axis MEMS gyroscope in the MPU-9250 includes a wide range of features:

- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyroscopes) with a user-programmable full-scale range of  $\pm 250$ ,  $\pm 500$ ,  $\pm 1000$ , and  $\pm 2000^\circ/\text{sec}$  and integrated 16-bit ADCs
- Digitally-programmable low-pass filter
- Gyroscope operating current: 3.2mA
- Sleep mode current: 8 $\mu$ A
- Factory calibrated sensitivity scale factor
- Self-test

### 2.2 Accelerometer Features

The triple-axis MEMS accelerometer in MPU-9250 includes a wide range of features:

- Digital-output triple-axis accelerometer with a programmable full scale range of  $\pm 2g$ ,  $\pm 4g$ ,  $\pm 8g$  and  $\pm 16g$  and integrated 16-bit ADCs
- Accelerometer normal operating current: 450 $\mu$ A
- Low power accelerometer mode current: 8.4 $\mu$ A at 0.98Hz, 19.8 $\mu$ A at 31.25Hz
- Sleep mode current: 8 $\mu$ A
- User-programmable interrupts
- Wake-on-motion interrupt for low power operation of applications processor
- Self-test

### 2.3 Magnetometer Features

The triple-axis MEMS magnetometer in MPU-9250 includes a wide range of features:

- 3-axis silicon monolithic Hall-effect magnetic sensor with magnetic concentrator
- Wide dynamic measurement range and high resolution with lower current consumption.
- Output data resolution of 14 bit (0.6 $\mu$ T/LSB)
- Full scale measurement range is  $\pm 4800\mu$ T
- Magnetometer normal operating current: 280 $\mu$ A at 8Hz repetition rate
- Self-test function with internal magnetic source to confirm magnetic sensor operation on end products

### 2.4 Additional Features

The MPU-9250 includes the following additional features:

- Auxiliary master I<sup>2</sup>C bus for reading data from external sensors (e.g. pressure sensor)
- 3.5mA operating current when all 9 motion sensing axes and the DMP are enabled
- VDD supply voltage range of 2.4 – 3.6V
- VDDIO reference voltage for auxiliary I<sup>2</sup>C devices
- Smallest and thinnest QFN package for portable devices: 3x3x1mm
- Minimal cross-axis sensitivity between the accelerometer, gyroscope and magnetometer axes
- 512 byte FIFO buffer enables the applications processor to read the data in bursts
- Digital-output temperature sensor
- User-programmable digital filters for gyroscope, accelerometer, and temp sensor
- 10,000 g shock tolerant
- 400kHz Fast Mode I<sup>2</sup>C for communicating with all registers
- 1MHz SPI serial interface for communicating with all registers

## Descripción del producto

	<b>MPU-9250 Product Specification</b>	Document Number: PS-MPU-9250A-01 Revision: 1.1 Release Date: 06/20/2016
---	---------------------------------------	---

- 20MHz SPI serial interface for reading sensor and interrupt registers
- MEMS structure hermetically sealed and bonded at wafer level
- RoHS and Green compliant

### 2.5 MotionProcessing

- Internal Digital Motion Processing™ (DMP™) engine supports advanced MotionProcessing and low power functions such as gesture recognition using programmable interrupts
- Low-power pedometer functionality allows the host processor to sleep while the DMP maintains the step count.

## Características eléctricas


	<b>MPU-9250 Product Specification</b>	Document Number: PS-MPU-9250A-01 Revision: 1.1 Release Date: 06/20/2016
---	---------------------------------------	---

- 20MHz SPI serial interface for reading sensor and interrupt registers
- MEMS structure hermetically sealed and bonded at wafer level
- RoHS and Green compliant

### 2.5 MotionProcessing

- Internal Digital Motion Processing™ (DMP™) engine supports advanced MotionProcessing and low power functions such as gesture recognition using programmable interrupts
- Low-power pedometer functionality allows the host processor to sleep while the DMP maintains the step count.

## Características eléctricas

	<b>MPU-9250 Product Specification</b>	Document Number: PS-MPU-9250A-01 Revision: 1.1 Release Date: 06/20/2016
---	---------------------------------------	---

### 3 Electrical Characteristics

#### 3.1 Gyroscope Specifications

Typical Operating Circuit of section 4.2, VDD = 2.5V, VDDIO = 2.5V, T<sub>A</sub>=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
Full-Scale Range	FS_SEL=0		±250		°/s
	FS_SEL=1		±500		°/s
	FS_SEL=2		±1000		°/s
	FS_SEL=3		±2000		°/s
Gyroscope ADC Word Length			16		bits
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)
	FS_SEL=1		65.5		LSB/(°/s)
	FS_SEL=2		32.8		LSB/(°/s)
	FS_SEL=3		16.4		LSB/(°/s)
Sensitivity Scale Factor Tolerance	25°C		±3		%
Sensitivity Scale Factor Variation Over Temperature	-40°C to +85°C		±4		%
Nonlinearity	Best fit straight line; 25°C		±0.1		%
Cross-Axis Sensitivity			±2		%
Initial ZRO Tolerance	25°C		±5		°/s
ZRO Variation Over Temperature	-40°C to +85°C		±30		°/s
Total RMS Noise	DLPFCFG=2 (92 Hz)		0.1		°/s-rms
Rate Noise Spectral Density			0.01		°/s/√Hz
Gyroscope Mechanical Frequencies		25	27	29	KHz
Low Pass Filter Response	Programmable Range	5		250	Hz
Gyroscope Startup Time	From Sleep mode		35		ms
Output Data Rate	Programmable, Normal mode	4		8000	Hz

**Table 1 Gyroscope Specifications**

## Características eléctricas

	<b>MPU-9250 Product Specification</b>	Document Number: PS-MPU-9250A-01 Revision: 1.1 Release Date: 06/20/2016
---	---------------------------------------	---

### 3.2 Accelerometer Specifications

Typical Operating Circuit of section 4.2, VDD = 2.5V, VDDIO = 2.5V, T<sub>A</sub>=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
Full-Scale Range	AFS_SEL=0		±2		g
	AFS_SEL=1		±4		g
	AFS_SEL=2		±8		g
	AFS_SEL=3		±16		g
ADC Word Length	Output in two's complement format		16		bits
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/g
	AFS_SEL=1		8,192		LSB/g
	AFS_SEL=2		4,096		LSB/g
	AFS_SEL=3		2,048		LSB/g
Initial Tolerance	Component-Level		±3		%
Sensitivity Change vs. Temperature	-40°C to +85°C AFS_SEL=0 Component-level		±0.026		%/°C
Nonlinearity	Best Fit Straight Line		±0.5		%
Cross-Axis Sensitivity			±2		%
Zero-G Initial Calibration Tolerance	Component-level, X,Y		±60		mg
	Component-level, Z		±80		mg
Zero-G Level Change vs. Temperature	-40°C to +85°C		±1.5		mg/°C
Noise Power Spectral Density	Low noise mode		300		µg/√Hz
Total RMS Noise	DLPFCFG=2 (94Hz)			8	mg-rms
Low Pass Filter Response	Programmable Range	5		260	Hz
Intelligence Function Increment			4		mg/LSB
Accelerometer Startup Time	From Sleep mode		20		ms
	From Cold Start, 1ms V <sub>DD</sub> ramp		30		ms
Output Data Rate	Low power (duty-cycled)	0.24		500	Hz
	Duty-cycled, over temp		±15		%
	Low noise (active)	4		4000	Hz

**Table 2 Accelerometer Specifications**

## Características eléctricas

	<b>MPU-9250 Product Specification</b>	Document Number: PS-MPU-9250A-01 Revision: 1.1 Release Date: 06/20/2016
---	---------------------------------------	---

### 3.3 Magnetometer Specifications

Typical Operating Circuit of section [4.2](#), VDD = 2.5V, VDDIO = 2.5V, T<sub>A</sub>=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
<b>MAGNETOMETER SENSITIVITY</b>					
Full-Scale Range			±4800		μT
ADC Word Length			14		bits
Sensitivity Scale Factor			0.6		μT / LSB
<b>ZERO-FIELD OUTPUT</b>					
Initial Calibration Tolerance			±500		LSB

Información de uso

	<b>MPU-9250 Product Specification</b>	Document Number: PS-MPU-9250A-01 Revision: 1.1 Release Date: 06/20/2016
---	---------------------------------------	---

4.2 Typical Operating Circuit

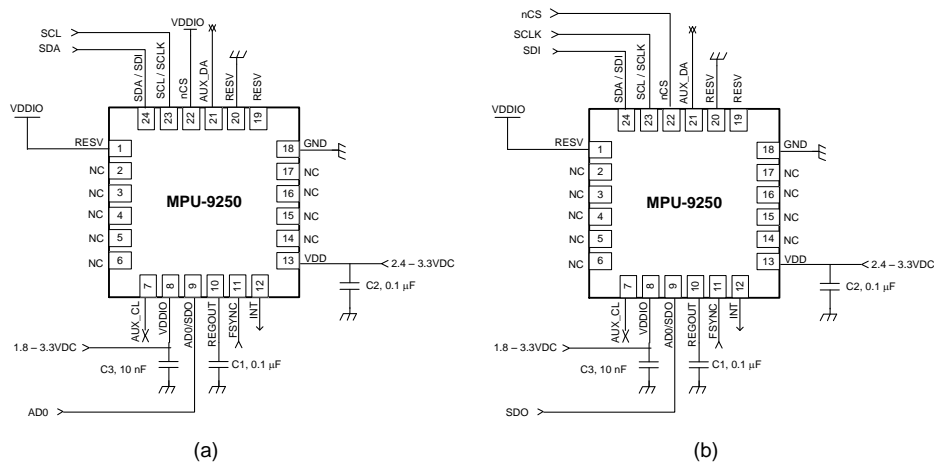


Figure 2 MPU-9250 QFN Application Schematic: (a) I2C operation, (b) SPI operation

Note that the INT pin should be connected to a GPIO pin on the system processor that is capable of waking the system processor from suspend mode.

4.3 Bill of Materials for External Components

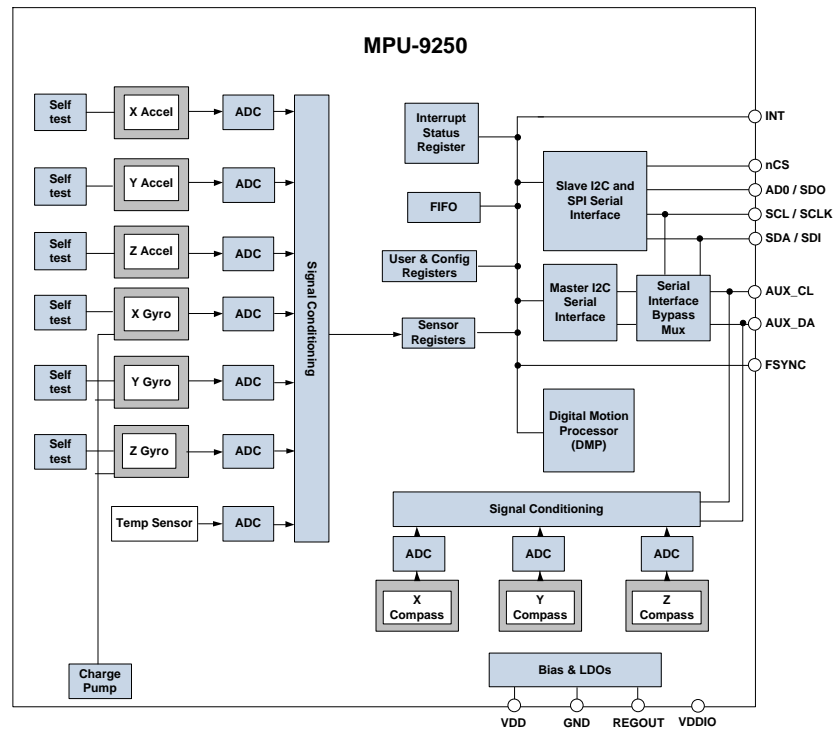
Component	Label	Specification	Quantity
Regulator Filter Capacitor	C1	Ceramic, X7R, 0.1µF ±10%, 2V	1
VDD Bypass Capacitor	C2	Ceramic, X7R, 0.1µF ±10%, 4V	1
VDDIO Bypass Capacitor	C3	Ceramic, X7R, 10nF ±10%, 4V	1

Table 10 Bill of Materials

## Información de uso

	<b>MPU-9250 Product Specification</b>	Document Number: PS-MPU-9250A-01 Revision: 1.1 Release Date: 06/20/2016
---	---------------------------------------	---

### 4.4 Block Diagram





# Ensamble

	<b>MPU-9250 Product Specification</b>	Document Number: PS-MPU-9250A-01 Revision: 1.1 Release Date: 06/20/2016
---	---------------------------------------	---

## 9 Assembly

This section provides general guidelines for assembling InvenSense Micro Electro-Mechanical Systems (MEMS) devices packaged in quad flat no-lead package (QFN) surface mount integrated circuits.

### 9.1 Orientation of Axes

The diagram below shows the orientation of the axes of sensitivity and the polarity of rotation. Note the pin 1 identifier (•) in the figure.

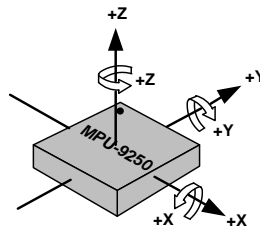


Figure 4. Orientation of Axes of Sensitivity and Polarity of Rotation for Accelerometer and Gyroscope

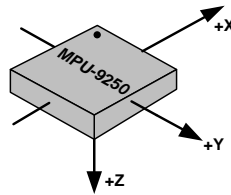


Figure 5. Orientation of Axes of Sensitivity for Compass

### 9.2 Package Dimensions

24 Lead QFN (3x3x1) mm NiPdAu Lead-frame finish

A.2. MPU-9250 Register Map and Descriptions

	MPU-9250 Register Map and Descriptions	Document Number: RM-MPU-9250A-00 Revision: 1.6 Release Date: 01/07/2015
---	--	---

**MPU-9250**  
**Register Map and Descriptions**  
**Revision 1.6**

# Registros para el acelerómetro y el giróscopo

	<b>MPU-9250 Register Map and Descriptions</b>	Document Number: RM-MPU-9250A-00 Revision: 1.6 Release Date: 01/07/2015
---	---	---

## 3 Register Map for Gyroscope and Accelerometer

The following table lists the register map for the gyroscope and accelerometer in the MPU-9250 MotionTracking device.

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00	0	SELF_TEST_X_GYRO	R/W	xg_st_data [7:0]							
01	1	SELF_TEST_Y_GYRO	R/W	yg_st_data [7:0]							
02	2	SELF_TEST_Z_GYRO	R/W	zg_st_data [7:0]							
0D	13	SELF_TEST_X_ACCEL	R/W	XA_ST_DATA [7:0]							
0E	14	SELF_TEST_Y_ACCEL	R/W	YA_ST_DATA [7:0]							
0F	15	SELF_TEST_Z_ACCEL	R/W	ZA_ST_DATA [7:0]							
13	19	XG_OFFSET_H	R/W	X_OFFSETS_USR [15:8]							
14	20	XG_OFFSET_L	R/W	X_OFFSETS_USR [7:0]							
15	21	YG_OFFSET_H	R/W	Y_OFFSETS_USR [15:8]							
16	22	YG_OFFSET_L	R/W	Y_OFFSETS_USR [7:0]							
17	23	ZG_OFFSET_H	R/W	Z_OFFSETS_USR [15:8]							
18	24	ZG_OFFSET_L	R/W	Z_OFFSETS_USR [7:0]							
19	25	SMP_LRT_DIV	R/W	SMP_LRT_DIV[7:0]							
1A	26	CONFIG	R/W	-	FIFO_MODE	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		
1B	27	GYRO_CONFIG	R/W	XGYRO_Ct_en	YGYRO_Ct_en	ZGYRO_Ct_en	GYRO_FS_SEL [1:0]		-	FCHOICE_B[1:0]	
1C	28	ACCEL_CONFIG	R/W	ax_st_en	ay_st_en	az_st_en	ACCEL_FS_SEL[1:0]		-		
1D	29	ACCEL_CONFIG 2	R/W	-				ACCEL_FCHOICE_B		A_DLPF_CFG	
1E	30	LP_ACCEL_ODR	R/W	-				Lposc_clksel [3:0]			
1F	31	WOM_THR	R/W	WOM_Threshold [7:0]							
23	35	FIFO_EN	R/W	TEMP_FIFO_EN	GYRO_XO_UT	GYRO_YO_UT	GYRO_ZO_UT	ACCEL	SLV2	SLV1	SLV0
24	36	I2C_MST_CTRL	R/W	MULT_MST_EN	WAIT_FOR_ES	SLV_3_FIFO_EN	I2C_MST_P_NSR	I2C_MST_CLK[3:0]			
25	37	I2C_SLV0_ADDR	R/W	I2C_SLV0_RNW	I2C_ID_0 [6:0]						
26	38	I2C_SLV0_REG	R/W	I2C_SLV0_REG[7:0]							
27	39	I2C_SLV0_CTRL	R/W	I2C_SLV0_EN	I2C_SLV0_BYTE_SW	I2C_SLV0_REG_DIS	I2C_SLV0_GRP	I2C_SLV0_LEN[3:0]			
28	40	I2C_SLV1_ADDR	R/W	I2C_SLV1_RNW	I2C_ID_1 [6:0]						
29	41	I2C_SLV1_REG	R/W	I2C_SLV1_REG[7:0]							
2A	42	I2C_SLV1_CTRL	R/W	I2C_SLV1_EN	I2C_SLV1_BYTE_SW	I2C_SLV1_REG_DIS	I2C_SLV1_GRP	I2C_SLV1_LEN[3:0]			
2B	43	I2C_SLV2_ADDR	R/W	I2C_SLV2_RNW	I2C_ID_2 [6:0]						
2C	44	I2C_SLV2_REG	R/W	I2C_SLV2_REG[7:0]							
2D	45	I2C_SLV2_CTRL	R/W	I2C_SLV2_EN	I2C_SLV2_BYTE_SW	I2C_SLV2_REG_DIS	I2C_SLV2_GRP	I2C_SLV2_LEN[3:0]			
2E	46	I2C_SLV3_ADDR	R/W	I2C_SLV3_RNW	I2C_ID_3 [6:0]						
2F	47	I2C_SLV3_REG	R/W	I2C_SLV3_REG[7:0]							
30	48	I2C_SLV3_CTRL	R/W	I2C_SLV3_EN	I2C_SLV3_BYTE_SW	I2C_SLV3_REG_DIS	I2C_SLV3_GRP	I2C_SLV3_LEN [3:0]			
31	49	I2C_SLV4_ADDR	R/W	I2C_SLV4_RNW	I2C_ID_4 [6:0]						
32	50	I2C_SLV4_REG	R/W	I2C_SLV4_REG[7:0]							
33	51	I2C_SLV4_DO	R/W	I2C_SLV4_DO[7:0]							
34	52	I2C_SLV4_CTRL	R/W	I2C_SLV4_EN	SLV4_DON_E_INT_EN	I2C_SLV4_REG_DIS	I2C_MST_DLY[4:0]				

# Registros para el acelerómetro y el giróscopo

<b>InvenSense</b>	<b>MPU-9250 Register Map and Descriptions</b>	Document Number: RM-MPU-9250A-00 Revision: 1.6 Release Date: 01/07/2015
-------------------	---	---

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
35	53	I2C_SLV4_DI	R	I2C_SLV4_DI[7:0]							
36	54	I2C_MST_STATUS	R	PASS_THROUGH	I2C_SLV4_DONE	I2C_LOST_ARB	I2C_SLV4_NACK	I2C_SLV3_NACK	I2C_SLV2_NACK	I2C_SLV1_NACK	I2C_SLV0_NACK
37	55	INT_PIN_CFG	R/W	ACTL	OPEN	LATCH_INT_EN	INT_ANYRD_2CLEAR	ACTL_FSYNC	FSYNC_INT_MODE_EN	BYPASS_EN	-
38	56	INT_ENABLE	R/W	-	WOM_EN	-	FIFO_OFLOW_EN	FSYNC_INT_EN	-	-	RAW_RDY_EN
3A	58	INT_STATUS	R	-	WOM_INT	-	FIFO_OFLOW_INT	FSYNC_INT	-	-	RAW_DATA_RDY_INT
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT_H[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT_L[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT_H[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT_L[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT_H[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT_L[7:0]							
41	65	TEMP_OUT_H	R	TEMP_OUT_H[15:8]							
42	66	TEMP_OUT_L	R	TEMP_OUT_L[7:0]							
43	67	GYRO_XOUT_H	R	GYRO_XOUT_H[15:8]							
44	68	GYRO_XOUT_L	R	GYRO_XOUT_L[7:0]							
45	69	GYRO_YOUT_H	R	GYRO_YOUT_H[15:8]							
46	70	GYRO_YOUT_L	R	GYRO_YOUT_L[7:0]							
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT_H[15:8]							
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT_L[7:0]							
49	73	EXT_SENS_DATA_00	R	EXT_SENS_DATA_00[7:0]							
4A	74	EXT_SENS_DATA_01	R	EXT_SENS_DATA_01[7:0]							
4B	75	EXT_SENS_DATA_02	R	EXT_SENS_DATA_02[7:0]							
4C	76	EXT_SENS_DATA_03	R	EXT_SENS_DATA_03[7:0]							
4D	77	EXT_SENS_DATA_04	R	EXT_SENS_DATA_04[7:0]							
4E	78	EXT_SENS_DATA_05	R	EXT_SENS_DATA_05[7:0]							
4F	79	EXT_SENS_DATA_06	R	EXT_SENS_DATA_06[7:0]							
50	80	EXT_SENS_DATA_07	R	EXT_SENS_DATA_07[7:0]							
51	81	EXT_SENS_DATA_08	R	EXT_SENS_DATA_08[7:0]							
52	82	EXT_SENS_DATA_09	R	EXT_SENS_DATA_09[7:0]							
53	83	EXT_SENS_DATA_10	R	EXT_SENS_DATA_10[7:0]							
54	84	EXT_SENS_DATA_11	R	EXT_SENS_DATA_11[7:0]							
55	85	EXT_SENS_DATA_12	R	EXT_SENS_DATA_12[7:0]							
56	86	EXT_SENS_DATA_13	R	EXT_SENS_DATA_13[7:0]							
57	87	EXT_SENS_DATA_14	R	EXT_SENS_DATA_14[7:0]							
58	88	EXT_SENS_DATA_15	R	EXT_SENS_DATA_15[7:0]							
59	89	EXT_SENS_DATA_16	R	EXT_SENS_DATA_16[7:0]							
5A	90	EXT_SENS_DATA_17	R	EXT_SENS_DATA_17[7:0]							
5B	91	EXT_SENS_DATA_18	R	EXT_SENS_DATA_18[7:0]							
5C	92	EXT_SENS_DATA_19	R	EXT_SENS_DATA_19[7:0]							
5D	93	EXT_SENS_DATA_20	R	EXT_SENS_DATA_20[7:0]							
5E	94	EXT_SENS_DATA_21	R	EXT_SENS_DATA_21[7:0]							
5F	95	EXT_SENS_DATA_22	R	EXT_SENS_DATA_22[7:0]							
60	96	EXT_SENS_DATA_23	R	EXT_SENS_DATA_23[7:0]							
63	99	I2C_SLV0_DO	R/W	I2C_SLV0_DO[7:0]							
64	100	I2C_SLV1_DO	R/W	I2C_SLV1_DO[7:0]							
65	101	I2C_SLV2_DO	R/W	I2C_SLV2_DO[7:0]							

## Registros para el acelerómetro y el giróscopo

	<b>MPU-9250 Register Map and Descriptions</b>	Document Number: RM-MPU-9250A-00 Revision: 1.6 Release Date: 01/07/2015
---	---	---

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
66	102	I2C_SLV3_DO	R/W	I2C_SLV3_DO[7:0]							
67	103	I2C_MST_DELAY_CTRL	R/W	DELAY_ES_SHADOW	-	-	I2C_SLV4_DLY_EN	I2C_SLV3_DLY_EN	I2C_SLV2_DLY_EN	I2C_SLV1_DLY_EN	I2C_SLV0_DLY_EN
68	104	SIGNAL_PATH_RESET	R/W	-	-	-	-	-	GYRO_RST	ACCEL_RST	TEMP_RST
69	105	MOT_DETECT_CTRL	R/W	ACCEL_INT_EL_EN	ACCEL_INT_EL_MODE	-		-		-	
6A	106	USER_CTRL	R/W	-	FIFO_EN	I2C_MST_EN	I2C_IF_DIS	-	FIFO_RST	I2C_MST_RST	SIG_COND_RST
6B	107	PWR_MGMT_1	R/W	H_RESET	SLEEP	CYCLE	GYRO_STANDBY	PD_PTAT	CLKSEL[2:0]		
6C	108	PWR_MGMT_2	R/W	-		DIS_XA	DIS_YA	DIS_ZA	DIS_XG	DIS_YG	DIS_ZG
72	114	FIFO_COUNTH	R/W	-			FIFO_CNT[12:8]				
73	115	FIFO_COUNTL	R/W	FIFO_CNT[7:0]							
74	116	FIFO_R_W	R/W	D[7:0]							
75	117	WHO_AM_I	R	WHOAM[7:0]							
77	119	XA_OFFSET_H	R/W	XA_OFFS [14:7]							
78	120	XA_OFFSET_L	R/W	XA_OFFS [6:0]							-
7A	122	YA_OFFSET_H	R/W	YA_OFFS [14:7]							
7B	123	YA_OFFSET_L	R/W	YA_OFFS [6:0]							-
7D	125	ZA_OFFSET_H	R/W	ZA_OFFS [14:7]							
7E	126	ZA_OFFSET_L	R/W	ZA_OFFS [6:0]							-

**Table 1 MPU-9250 mode register map for Gyroscope and Accelerometer**

Note: Register Names ending in \_H and \_L contain the high and low bytes, respectively, of an internal register value.

In the detailed register tables that follow, register names are in capital letters, while register values are in capital letters and italicized. For example, the ACCEL\_XOUT\_H register (Register 59) contains the 8 most significant bits, *ACCEL\_XOUT[15:8]*, of the 16-bit X-Axis accelerometer measurement, *ACCEL\_XOUT*.

The reset value is 0x00 for all registers other than the registers below.

- Register 107 (0x01) Power Management 1
- Register 117 (0x71) WHO\_AM\_I

## A.3. AK8963 datasheet

Asahi**KASEI**

[AK8963]



AK8963

**3-axis Electronic Compass**

### 1. Features

A 3-axis electronic compass IC with high sensitive Hall sensor technology.  
Best adapted to pedestrian city navigation use for cell phone and other portable appliance.

Functions:

- 3-axis magnetometer device suitable for compass application
- Built-in A to D Converter for magnetometer data out
- 14-/16-bit selectable data out for each 3 axis magnetic components
  - Sensitivity: 0.6  $\mu$ T/LSB typ. (14-bit)
  - 0.15 $\mu$ T/LSB typ. (16-bit)
- Serial interface
  - I<sup>2</sup>C bus interface.
  - Standard mode and Fast mode compliant with Philips I<sup>2</sup>C specification Ver.2.1
  - 4-wire SPI
- Operation modes:
  - Power-down, Single measurement, Continuous measurement, External trigger measurement, Self test and Fuse ROM access.
- DRDY function for measurement data ready
- Magnetic sensor overflow monitor function
- Built-in oscillator for internal clock source
- Power on Reset circuit
- Self test function with built-in internal magnetic source

Operating temperatures:

- -30°C to +85°C

Operating supply voltage:

- Analog power supply +2.4V to +3.6V
- Digital Interface supply +1.65V to analog power supply voltage.

Current consumption:

- Power-down: 3  $\mu$ A typ.
- Measurement:
  - Average power consumption at 8 Hz repetition rate: 280 $\mu$ A typ.

Package:

- |         |                      |  |
|---------|----------------------|--|
| AK8963C | 14-pin WL-CSP (BGA): | 1.6 mm $\times$ 1.6 mm $\times$ 0.5 mm (typ.)  |
| AK8963N | 16-pin QFN package:  | 3.0 mm $\times$ 3.0 mm $\times$ 0.75 mm (typ.) |

# Descripción

## 2. Overview

AK8963 is 3-axis electronic compass IC with high sensitive Hall sensor technology. Small package of AK8963 incorporates magnetic sensors for detecting terrestrial magnetism in the X-axis, Y-axis, and Z-axis, a sensor driving circuit, signal amplifier chain, and an arithmetic circuit for processing the signal from each sensor. Self test function is also incorporated. From its compact foot print and thin package feature, it is suitable for map heading up purpose in GPS-equipped cell phone to realize pedestrian navigation function.

AK8963 has the following features:

- (1) Silicon monolithic Hall-effect magnetic sensor with magnetic concentrator realizes 3-axis magnetometer on a silicon chip. Analog circuit, digital logic, power block and interface block are also integrated on a chip.
- (2) Wide dynamic measurement range and high resolution with lower current consumption.

Output data resolution:	14-bit (0.6 $\mu$ T/LSB)
	16-bit (0.15 $\mu$ T/LSB)
Measurement range:	$\pm 4900 \mu$ T
Average current at 8Hz repetition rate:	280 $\mu$ A typ.
- (3) Digital serial interface
  - I<sup>2</sup>C bus interface to control AK8963 functions and to read out the measured data by external CPU. A dedicated power supply for I<sup>2</sup>C bus interface can work in low-voltage apply as low as 1.65V.
  - 4-wire SPI is also supported. A dedicated power supply for SPI can work in low-voltage apply as low as 1.65V.
- (4) DRDY pin and register inform to system that measurement is end and set of data in registers are ready to be read.
- (5) Device is worked by on-chip oscillator so no external clock source is necessary.
- (6) Self test function with internal magnetic source to confirm magnetic sensor operation on end products.

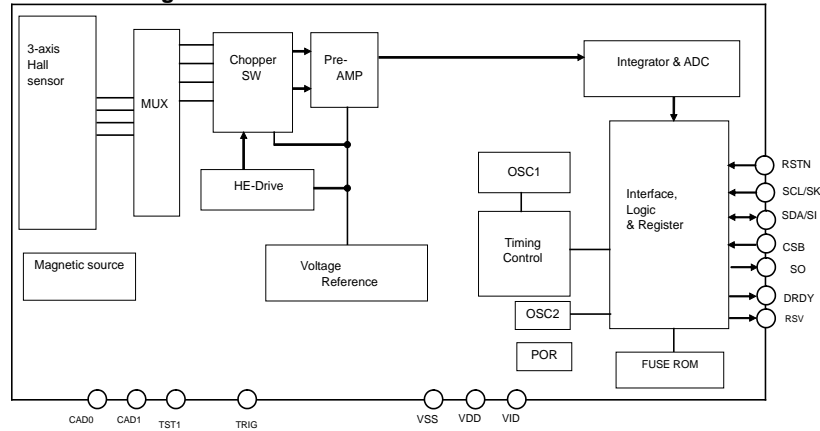
# Configuración del circuito

AsahiKASEI

[AK8963]

## 4. Circuit Configuration

### 4.1. Block Diagram



### 4.2. Block Function

Block	Function
3-axis Hall sensor	Monolithic Hall elements.
MUX	Multiplexer for selecting Hall elements.
Chopper SW	Performs chopping.
HE-Drive	Magnetic sensor drive circuit for constant-current driving of sensor
Pre-AMP	Fixed-gain differential amplifier used to amplify the magnetic sensor signal.
Integrator & ADC	Integrates and amplifies pre-AMP output and performs analog-to-digital conversion.
OSC1	Generates an operating clock for sensor measurement. 12MHz(typ.)
OSC2	Generates an operating clock for sequencer. 128kHz(typ.)
POR	Power On Reset circuit. Generates reset signal on rising edge of VDD.
Interface Logic & Register	Exchanges data with an external CPU. DRDY pin indicates sensor measurement end and data is ready to be read. I <sup>2</sup> C bus interface using two pins, namely, SCL and SDA. Standard mode and Fast mode are supported. The low-voltage specification can be supported by applying 1.65V to the VID pin. 4-wire SPI is also supported by SK, SI, SO and CSB pins. 4-wire SPI works in VID pin voltage down to 1.65V, too.
Timing Control	Generates a timing signal required for internal operation from a clock generated by the OSC1.
Magnetic Source	Generates magnetic field for self test of magnetic sensor.
FUSE ROM	Fuse for adjustment



# Configuración del circuito

AsahiKASEI

[AK8963]

## 4.3. Pin Function

QFN Pin No.	WLCSP Pin No.	Pin name	I/O	Power supply system	Type	Function
1	A1	DRDY	O	VID	CMOS	Data Ready output pin. "H" active. Informs measurement ended and data is ready to be read.
2	A2	CSB	I	VID	CMOS	Chip select pin for 4-wire SPI. "L" active. Connect to VID when selecting I <sup>2</sup> C bus interface.
3	A3	SCL	I	VID	CMOS	When the I <sup>2</sup> C bus interface is selected (CSB pin is connected to VID) SCL: Control data clock input pin Input: Schmidt trigger
		SK				When the 4-wire SPI is selected SK: Serial clock input pin
5	A4	SDA	I/O	VID	CMOS	When the I <sup>2</sup> C bus interface is selected (CSB pin is connected to VID) SDA: Control data input/output pin Input: Schmidt trigger, Output: Open drain
		SI	I			When the 4-wire SPI is selected SI: Serial data input pin
15	B1	VDD	-	-	Power	Analog Power supply pin.
4	B3	RSV	O	VID	CMOS	Reserved. Keep this pin electrically non-connected.
6	B4	SO	O	VID	CMOS	When the I <sup>2</sup> C bus interface is selected (CSB pin is connected to VID) Hi-Z output. Keep this pin electrically non-connected.
						When the 4-wire SPI is selected Serial data output pin
13	C1	VSS	-	-	Power	Ground pin.
14	C2	TST1	I	VDD	CMOS	Test pin. Pulled down by 100kΩ internal resistor. Keep this pin electrically non-connected or connect to VSS.
7	C3	TRG	I	VID	CMOS	External trigger pulse input pin. Enabled only in External trigger mode. Pulled down by 100kΩ internal resistor. When External trigger mode is not in use, keep this pin electrically non-connected or connect to VSS.
8	C4	VID	-	-	Power	Digital interface positive power supply pin.
12	D1	CAD0	I	VDD	CMOS	When the I <sup>2</sup> C bus interface is selected (CSB pin is connected to VID) CAD0: Slave address 0 input pin Connect to VSS or VDD.
						When the 4-wire serial interface is selected Connect to VSS.
11	D2	CAD1	I	VDD	CMOS	When the I <sup>2</sup> C bus interface is selected (CSB pin is connected to VID) CAD1: Slave address 1 input pin Connect to VSS or VDD.
						When the 4-wire serial interface is selected Connect to VSS.
10	D4	RSTN	I	VID	CMOS	Reset pin. Resets registers by setting to "L". Connect to VID when not in use.

# Características generales

AsahiKASEI

[AK8963]

## 5. Overall Characteristics

### 5.1. Absolute Maximum Ratings

V<sub>SS</sub>=0V

Parameter	Symbol	Min.	Max.	Unit
Power supply voltage (V <sub>DD</sub> , V <sub>ID</sub> )	V+	-0.3	+4.3	V
Input voltage	V <sub>IN</sub>	-0.3	(V+)+0.3	V
Input current	I <sub>IN</sub>	-	±10	mA
Storage temperature	T <sub>ST</sub>	-40	+125	°C

(Note 1) If the device is used in conditions exceeding these values, the device may be destroyed. Normal operations are not guaranteed in such exceeding conditions.

### 5.2. Recommended Operating Conditions

V<sub>SS</sub>=0V

Parameter	Remark	Symbol	Min.	Typ.	Max.	Unit
Operating temperature		T <sub>a</sub>	-30		+85	°C
Power supply voltage	V <sub>DD</sub> pin voltage	V <sub>DD</sub>	2.4	3.0	3.6	V
	V <sub>ID</sub> pin voltage	V <sub>ID</sub>	1.65		V <sub>DD</sub>	V

### 5.3. Electrical Characteristics

The following conditions apply unless otherwise noted:

V<sub>DD</sub>=2.4V to 3.6V, V<sub>ID</sub>=1.65V to V<sub>DD</sub>, Temperature range=-30°C to 85°C

#### 5.3.1. DC Characteristics

Parameter	Symbol	Pin	Condition	Min.	Typ.	Max.	Unit
High level input voltage 1	VIH1	CSB RSTN		70% V <sub>ID</sub>			V
Low level input voltage 1	VIL1	TRG				30% V <sub>ID</sub>	V
High level input voltage 2	VIH2	SK/SCL		70% V <sub>ID</sub>		V <sub>ID</sub> +0.5	V
Low level input voltage 2	VIL2	SI/SDA		-0.5		30% V <sub>ID</sub>	V
High level input voltage 3	VIH3	CAD0		70% V <sub>DD</sub>			V
Low level input voltage 3	VIL3	CAD1				30% V <sub>DD</sub>	V
Input current 1	IIN1	SK/SCL SI/SDA CSB RSTN	V <sub>IN</sub> =V <sub>SS</sub> or V <sub>ID</sub>	-10		+10	μA
Input current 2	IIN2	CAD0 CAD1	V <sub>IN</sub> =V <sub>SS</sub> or V <sub>DD</sub>	-10		+10	μA
Input current 3	IIN3	TRG	V <sub>IN</sub> =V <sub>ID</sub>			100	μA
Input current 4	IIN4	TST1	V <sub>IN</sub> =V <sub>DD</sub>			100	μA
Hysteresis input voltage (Note 2)	VHS	SCL SDA	V <sub>ID</sub> ≥2V	5% V <sub>ID</sub>			V
			V <sub>ID</sub> <2V	10% V <sub>ID</sub>			V
High level output voltage 1	VOH1	SO	I <sub>OH</sub> ≥-100μA	80% V <sub>ID</sub>			V
Low level output voltage 1	VOL1	DRDY	I <sub>OL</sub> ≤+100μA			20% V <sub>ID</sub>	V
Low level output voltage 2 (Note 3)(Note 4)	VOL2	SDA	I <sub>OL</sub> ≤3mA V <sub>ID</sub> ≥2V			0.4	V
			I <sub>OL</sub> ≤3mA V <sub>ID</sub> <2V			20% V <sub>ID</sub>	V
Current consumption (Note 5)	IDD1	VDD VID	Power-down mode V <sub>DD</sub> =V <sub>ID</sub> =3.0V		3	10	μA
	IDD2		When magnetic sensor is driven		5	10	mA
	IDD3		Self-test mode		9	15	mA
	IDD4		(Note 6)		0.1	5	μA

(Note 2) Schmitt trigger input (reference value for design)

(Note 3) Maximum load capacitance: 400pF (capacitive load of each bus line applied to the I<sup>2</sup>C bus interface)

(Note 4) Output is open-drain. Connect a pull-up resistor externally.

(Note 5) Without any resistance load

(Note 6) (case1) V<sub>DD</sub>=ON, V<sub>ID</sub>=ON, RSTN pin = "L". (case2) V<sub>DD</sub>=ON, V<sub>ID</sub>=OFF(0V), RSTN pin = "L".

# Registros

AsahiKASEI

[AK8963]

## 8. Registers

### 8.1. Description of Registers

AK8963 has registers of 20 addresses as indicated in Table 8.1. Every address consists of 8 bits data. Data is transferred to or received from the external CPU via the serial interface described previously.

Table 8.1 Register Table

Name	Address	READ/ WRITE	Description	Bit width	Explanation
WIA	00H	READ	Device ID	8	
INFO	01H	READ	Information	8	
ST1	02H	READ	Status 1	8	Data status
HXL	03H	READ	Measurement data	8	X-axis data
HXH	04H			8	
HYL	05H			8	Y-axis data
HYH	06H			8	
HZL	07H			8	Z-axis data
HZH	08H			8	
ST2	09H	READ	Status 2	8	Data status
CNTL1	0AH	READ/ WRITE	Control 1	8	Function Control
CNTL2	0BH		Control 2	8	
ASTC	0CH	READ/ WRITE	Self-test	8	
TS1	0DH	READ/ WRITE	Test 1	8	DO NOT ACCESS
TS2	0EH	READ/ WRITE	Test 2	8	DO NOT ACCESS
I2CDIS	0FH	READ/ WRITE	I <sup>2</sup> C disable	8	
ASAX	10H	READ	X-axis sensitivity adjustment value	8	Fuse ROM
ASAY	11H	READ	Y-axis sensitivity adjustment value	8	Fuse ROM
ASAZ	12H	READ	Z-axis sensitivity adjustment value	8	Fuse ROM
RSV	13H	READ	Reserved	8	DO NOT ACCESS

Addresses 00H-0CH and 10H-12H are compliant with automatic increment function of serial interface respectively. Values of addresses 10H-12H can be read only in Fuse ROM access mode. In other modes, read data is not correct.

# Registros

Asahi**KASEI**

[AK8963]

## 8.2. Register Map

Table 8.2 Register Map


Addr	Register Name	D7	D6	D5	D4	D3	D2	D1	D0
<b>Read-only Register</b>									
00H	WIA	0	1	0	0	1	0	0	0
01H	INFO	INFO7	INFO6	INFO5	INFO4	INFO3	INFO2	INFO1	INFO0
02H	ST1	-	0	0	-	0	0	DOR	DRDY
03H	HXL	HX7	HX6	HX5	HX4	HX3	HX2	HX1	HX0
04H	HXH	HX15	HX14	HX13	HX12	HX11	HX10	HX9	HX8
05H	HYL	HY7	HY6	HY5	HY4	HY3	HY2	HY1	HY0
06H	HYH	HY15	HY14	HY13	HY12	HY11	HY10	HY9	HY8
07H	HZL	HZ7	HZ6	HZ5	HZ4	HZ3	HZ2	HZ1	HZ0
08H	HZH	HZ15	HZ14	HZ13	HZ12	HZ11	HZ10	HZ9	HZ8
09H	ST2	0	0	0	BITM	HOFL	0	0	0
<b>Write/read Register</b>									
0AH	CNTL1	0	0	0	BIT	MODE3	MODE2	MODE1	MODE0
0BH	CNTL2	0	0	0	0	0	0	0	SRST
0CH	ASTC	-	SELF	-	-	-	-	-	-
0DH	TS1	-	-	-	-	-	-	-	-
0EH	TS2	-	-	-	-	-	-	-	-
0FH	I2CDIS	I2CDIS7	I2CDIS6	I2CDIS5	I2CDIS4	I2CDIS3	I2CDIS2	I2CDIS1	I2CDIS0
<b>Read-only Register</b>									
10H	ASAX	COEFX7	COEFX6	COEFX5	COEFX4	COEFX3	COEFX2	COEFX1	COEFX0
11H	ASAY	COEFY7	COEFY6	COEFY5	COEFY4	COEFY3	COEFY2	COEFY1	COEFY0
12H	ASAZ	COEFZ7	COEFZ6	COEFZ5	COEFZ4	COEFZ3	COEFZ2	COEFZ1	COEFZ0
13H	RSV	-	-	-	-	-	-	-	-

When VDD is turned ON, POR function works and all registers of AK8963 are initialized regardless of VID status. To write data to or to read data from register, VID must be ON.

TS1 and TS2 are test registers for shipment test. Do not use these registers.


RSV is reserved register. Do not use this register.

## A.4. MPU-9250 Accelerometer, Gyroscope and Compass Self-Test Implementation

	<b>MPU-9250 Accelerometer, Gyroscope and Compass Self-Test Implementation</b>	Document Number: AN-MPU-9250A-03 Revision: 1.0 Release Date: 5/30/2013
---	---	--

### MPU-9250 Accelerometer, Gyroscope and Compass Self-Test Implementation

## A.5. MPU Hardware Offset Registers: Application note

	<b>InvenSense Inc.</b> 1745 Technology Drive, San Jose, CA, 95110 U.S.A. Tel: +1 (408) 501-2200 Fax: +1 (408) 988-7339 Website: <a href="http://www.invensense.com">www.invensense.com</a>	Document Number: AN-XX-XXXX-XX Revision: 1.0 Release Date: 02/21/2014
---	---	---

### MPU Hardware Offset Registers Application Note

A printed copy of this document is  
**NOT UNDER REVISION CONTROL**  
unless it is dated and stamped in red ink as,  
"REVISION CONTROLLED COPY."

This information furnished by InvenSense is believed to be accurate and reliable. However, no responsibility is assumed by InvenSense for its use, or for any infringements of patents or other rights of third parties that may result from its use. Specifications are subject to change without notice. InvenSense reserves the right to make changes to this product, including its circuits and software, in order to improve its design and/or performance, without prior notice. InvenSense makes no warranties, neither expressed nor implied, regarding the information and specifications contained in this document. InvenSense assumes no responsibility for any claims or damages arising from information contained in this document, or from the use of products and services detailed therein. This includes, but is not limited to, claims or damages based on the infringement of patents, copyrights, mask work and/or other intellectual property rights.

Certain intellectual property owned by InvenSense and described in this document is patent protected. No license is granted by implication or otherwise under any patent or patent rights of InvenSense. This publication supersedes and replaces all information previously supplied. Trademarks that are registered trademarks are the property of their respective companies. InvenSense sensors should not be used or sold in the development, storage, production or utilization of any conventional or mass-destructive weapons or for any other weapons or life threatening applications, as well as in any other life critical applications such as medical equipment, transportation, aerospace and nuclear instruments, undersea equipment, power plant equipment, disaster prevention and crime prevention equipment.

Copyright ©2011 InvenSense Corporation.

# Apéndice B

## Qt Project: Raspberry Pi 3B

### B.1. main.cpp

```
1 #include "mainwindow.h"
2 #include <QApplication>
3
4 #include <typeinfo>
5 #include "stdio.h"
6
7 #include <cmath>
8 #include <QString>
9 #include <QDebug>
10 #include <QUdpSocket>
11
12 #include "arduino.h"
13 #include "Complementos.h"
14 #include "matlab.h"
15 #include "mocapsuit.h"
16 #include "mpu9250.h"
17 #include "RegistersAndMask.h"
18 #include "SPI.h"
19
20 #include "mpu9250.h"
21
22 #include "wiringPi.h" //Borrar libreria
23
24
25 /***** CONEXIONES *****/
26 #include "ARDUINO"
27 //GND - GND
28 //VCC - VCC
29 //SDA - Pin A4
30 //SCL - Pin A5
31 //SDA(MOSI) - Pin D11
32 //ADO(MISO) - Pin D12
33 //SCL(SCK) - Pin D13
34 //NCS(CS) - Pin D7 (arbitrario)
35
36
37
38
39 QUdpSocket* mUdpSocket;
40
41 int main(int argc, char *argv[])
42 {
43     QApplication a(argc, argv);
44     MainWindow w;
45
46     mUdpSocket = new QUdpSocket();
47     MocapSuit MainSuit;
48     // Mpu9250 Esternon(2, MainSuit,
49     // {9.819, 9.584, 10.071, 0.3, 0.131,
50     // -0.317}, {-1.6558, 2.0798, -0.3839},
51     // {39.6496, 43.5980, 40.8667, -9.53,
52     // 11.987, 47.4591}, {-90, 80.52, -90});
53     // // Sensor #2
54
55     // Mpu9250 ClaviculaIzq(3, MainSuit,
56     // {10.132, 9.990, 10.441, 0.736, 0.4,
57     // -1.93}, {3.5523, 0.6303, 2.2918},
58     // {45.687, 44.505, 46.507, 22.771,
59     // 0.5005, 12.820}, {-3.94, -1.82,
60     // -26.13}); // Sensor #3 {-1.39, 0, 0}
61     // Mpu9250 BrazoIzq(21, MainSuit,
62     // {9.624, 9.603, 9.934, -6.046, -3.138,
63     // -6.3}, {0.0, 0.0, 0.0},
64     // {40.687, 40.505, 40.507, -11.0,
65     // 12.0, 15.0}, {-1.39, 0.0, 0.0}); //
66     // Sensor #5
67     // Mpu9250 AntBrazoIzq(22, MainSuit,
68     // {9.624, 9.603, 9.934, 0.0, 0.0, 0.0},
69     // {0.0, 0.0, 0.0},
70     // {40.687, 40.505, 40.507, 0.0, 0.0,
71     // 0.0}, {-7.83, 0.0, 0.0}); // Sensor
72     // #10 (Ant: MuñecaDer)
73     // Mpu9250 BrazoDer(23, MainSuit, {9.624,
74     // 9.603, 9.934, 0.15, -0.2, -0.6},
75     // {0.0, 0.0, 0.0},
76     // {40.687, 40.505, 40.507, 4, 30.0,
77     // 17.5}, {178.61, 0.0, 0.0}); //
78     // Sensor #11
79     // Mpu9250 AntBrazoDer(24, MainSuit,
80     // {9.624, 9.603, 9.934, 0.0, -0.1, 0.0},
81     // {0.0, 0.0, 0.0},
82     // {40.687, 40.505, 40.507, -22.5,
83     // 23.5, 25.5}, {172.161, 0.0, 0.0}); //
84     // Sensor #12
85     // Mpu9250 sinDef4(4, MainSuit, {9.624,
86     // 9.603, 9.934, 0.046, -0.35, -0.55},
87     // {1.2490, -1.9366, -1.2892},
88     // {40.687, 40.505, 40.507, -6.5, 3.5,
89     // 31.5}, {178.61, 0.0, 0.0}); // Sensor
90     // #4 (Ant: ClaviculaIzq)
91     // Mpu9250 sinDef6(6, MainSuit, {9.624,
92     // 9.603, 9.934, 0.146, -0.138, 0.580},
93     // {0.0, 0.0, 0.0},
94     // {40.687, 40.505, 40.507, 10.5, 10.0,
95     // 8.0}, {-83.22, 5.67, 3.34}); //
96     // Sensor #6 (Ant: AntBrazoIzq)
97     // Mpu9250 sinDef7(7, MainSuit, {9.624,
98     // 9.603, 9.934, 0.046, -0.138, -0.480},
99     // {0.0, 0.0, 0.0},
```

```

66 // {40.687, 40.505, 40.507, 0.0, 0.0,
    // 0.0}, {178.61, 0.0, 0.0}); /// Sensor
    // #7 (Ant: MuñecaIzq)
67 // Mpu9250 SinDef(21, MainSuit, {9.624,
    // 9.603, 9.934, 0.0, 0.7, -0.5}, {0.0,
    // 0.0, 0.0},
68 // {40.687, 40.505, 40.507, 0.00, 0.0,
    // 0.0}, {-7.83, 0.0, 0.0}); /// Sensor
    // #8 (Ant: BrazoDer)
69 // Mpu9250 sinDef9(7, MainSuit, {9.624,
    // 9.603, 9.934, 0.0, 0.0, 0.0}, {0.0,
    // 0.0, 0.0},
70 // {40.687, 40.505, 40.507, 0.0, 0.0,
    // 0.0}, {-83.22, 5.67, 3.34}); ///
    // Sensor #9 (Ant: AntBrazoDer, luego
    // Cabeza)
71
72
73 wiringPiSetup();
74 QVector<uint8_t> Unused = {2,3,4,
    // 5,6,7, 21,22,23,24};
75 for(int i = 0; i < Unused.size();
    // i++){
76     pinMode(Unused[i], OUTPUT);
77     digitalWrite(Unused[i], HIGH);
78 }
79 if
    (SPI.setClockDivider(SPI_CLOCK_DIV16)
    // == -1){
80     fprintf(stderr, "FATAL ERROR:
        // SPI.setUp Failed \n");
        // exit(0);
81 }
82
83 // Esternon.setup();
84 // ClaviculaIzq.setup();
85 // BrazoIzq.setup();
86 // AntBrazoIzq.setup();
87 // BrazoDer.setup();
88 // AntBrazoDer.setup();
89
90 // ClaviculaDer.setup();
91 // MunecaIzq.setup();
92 // MunecaDer.setup();
93 // Back.setup();
94 // sinDef9.setup();
95
96 // sinDef.setup();
97
98 MainSuit.setup();
99 delay(5000);
100 MainSuit.loop();
101
102 w.show();
103 a.exec();
104
105 delete mUdpSocket;
106 return 0;
107 }

```

## B.2. RegistersAndMask.h

```

1 #ifndef REGISTERSANDMASK_H
2 #define REGISTERSANDMASK_H
3
4 #define SPI_PROTOCOL
5 #define MPU9250
6
7 /* DIRECCIONES I2C (A CONTINUACIÓN) */
8 #define MPU9250_ADDR 0x68
9 #define MAG_ADDRESS 0x0C
10
11
12 /* MASCARAS E INFORMACION (A
    // CONTINUACIÓN) */
13 #define GYRO_FSCALE_MASK 0b11100111 // Register 27-0x1B [4:3
    // bit]
14 #define GYRO_FSCALE_250_DPS 0x00
15 #define GYRO_FSCALE_500_DPS 0x08
16 #define GYRO_FSCALE_1000_DPS 0x10
17 #define GYRO_FSCALE_2000_DPS 0x18
18 #define ACC_FSCALE_MASK 0b11100111
    // Register 28-0x1C [4:3 bit]
19 #define ACC_FSCALE_2_G 0x00
20 #define ACC_FSCALE_4_G 0x08
21 #define ACC_FSCALE_8_G 0x10
22 #define ACC_FSCALE_16_G 0x18
23
24 #define MAG_FSCALE_MASK 0b11101111
    // Register 0xA [4 bit]
25 #define MAG_FSCALE_14_bit 0x00
26 #define MAG_FSCALE_16_bit 0x10
27
28 #define MAG_MODE_MASK 0b11110000
    // Register 0xA [3:0 bit]
29 #define MAG_MODE_2 0x06
30
31 /* REGISTROS IMPORTANTES (A COTINUACIÓN)
    // */
32 #define WHO_AM_I 0x75 // si es
    // igual a 0x71, es MPU9250; si es 0x73,
    // es MPU9255; si es 0x70, es MPU6500
33 #define GYRO_SELF_TEST 0x00 //Leer 3
    // registros: X Y Z
34 #define ACC_SELF_TEST 0x0D //Leer 3
    // registros: X Y Z
35 #define GYRO_OFFSET 0x13 //Primer
    // byte LOW, 2nd byte HIGH
36 #define ACC_OFFSET 0x77 //Primer
    // byte LOW, 2nd byte HIGH.
37 #define ACC_GYRO_DATA 0x3B //Leer 14
    // registros (big endian):
    // accel(1-6) Temp(7-8)
    // gyro(8-14) Primer byte
    // HIGH, 2nd byte LOW
38
39 #define ACC_GYRO_CONFIG 0x1A
    // Configuraciones generales
40 #define ACC_CONFIG_1 0x1C
    // Accel-Config Register #1
41 #define ACC_GYRO_RESET 0x6B //Reset to
    // "0" all Registers

```



```

42 #define MPU_INT_PIN_CFG 0x37
43 #define MPU_USER_CTRL 0x6A
44 #define MPU_I2C_MST_CTRL 0x24
45 #define MAG_DATA 0x03 //Leer 7
   ↳ registros(Two complement big endian):
46   ↳ //Mag(1-6) Status(7)
   ↳ Primer byte LOW, 2nd byte
   ↳ HIGH
47 #define MAG_CNTL_1 0x0A
   ↳ //Registro #1 de configuración del
   ↳ sensor
48 #define MAG_CNTL_2 0x0B
49 #define MAG_ASTC 0x0C //
   ↳ SELF-TEST Register
50 #define MAG_STATUS_1 0x02 //Status
   ↳ register #1 (Data ready - Data
   ↳ Overrun)
51 #define MAG_ASAX 0x10
   ↳ //Sensitivity adjudgment values. Leer 3
   ↳ registros
52
53 /* REGISTROS PARA "SPI" EN MPU9250*/
54 #define MPU_I2C_SLV0_ADDR 0x25
55 #define MPU_I2C_SLV0_REG 0x26
56 #define MPU_I2C_SLV0_CTRL 0x27

```

```

57 #define MPU_I2C_SLV0_DO 0x63
58 #define MPU_EXT_SENS_DATA_00 0x49
59
60 /* CONTANTES GLOBALES */
61 #define ACC_SENS 2048 // 32768 LSB /
   ↳ 16 G
62 #define GYRO_SENS 33 // 32768 LSB /
   ↳ 1000 deg/seg
63
64
65 /* ----- PROGRAM
   ↳ ----- */
66 #define MAG_MODE_MASK 0b11110000
   ↳ //Register 0xA [3:0 bit]
67 #define MAG_MODE_1 0x02 //Nuevo
   ↳ en Rpi (ODR = 8 Hz)
68 #define MAG_MODE_2 0x06 //(ODR
   ↳ = 100 Hz)
69
70 #define MPU_FIFO_EN 0x23 //
71 #define MPU_USR_CTRL 0x6A //
72
73
74 #endif

```

## B.3. arduino.h

```

1 #ifndef ARDUINOFUNCTIONS_H
2 #define ARDUINOFUNCTIONS_H
3
4 class QString;
5
6 class _Serial{
7 public:
8     void print(QString cadena);
9     void print(float val);
10    void println(QString cadena);
11    void println(float val);
12    void println(void);

```

```

13
14 private:
15
16 };
17
18 extern _Serial Serial;
19
20 QString String (float value);
21 QString String (int value);
22
23
24 #endif // ARDUINOFUNCTIONS_H

```

## B.4. Complementos.h

```

1 #ifndef COMPLEMENTOS_H
2 #define COMPLEMENTOS_H
3
4 #include <stdint>
5
6 #define SPI_PROTOCOL
7 #define MPU9250
8
9 /*
10  * INDISPENSABLES:
11  * + Definir un protocolo de
   comunicación: #define "SPI_PROTOCOL" o
   ↳ "I2C_PROTOCOL"
12  * + Definir un sensor: #define "MPU9250"
   ↳ o "BMX055"

```

```

13 */
14
15
16 #define READ_FLAG 0b10000000
17 #define WRITE_FLAG 0b01111111
18
19 #if defined(I2C_PROTOCOL)
20 //Funcion auxiliar lectura
21 void I2Cread(uint8_t Address, uint8_t
   Register, uint8_t nBytes, uint8_t*
   ↳ Data);
22 // Funcion auxiliar de escritura
23 void I2CwriteByte(uint8_t Address, uint8_t
   Register, uint8_t Data);
24

```

```

25 void I2CwriteNBytes(uint8_t Address,
    uint8_t Register, uint8_t nBytes,
    ↪ uint8_t* Data);
26
27 #elif defined(SPI_PROTOCOL)
28 void SPIread(uint8_t CS, uint8_t Reg,
    ↪ uint8_t nBytes, uint8_t* Data);
29 void SPIwriteByte(uint8_t CS, uint8_t Reg,
    ↪ uint8_t Data);
30 void SPIwriteNBytes(uint8_t CS, uint8_t
    ↪ Reg, uint8_t nBytes, uint8_t* Data);
31
32 #ifdef MPU9250
33 void SPIreadMg(uint8_t Addr, uint8_t CS,
    uint8_t Reg, uint8_t nBytes, uint8_t*
    ↪ Data);
34 void SPIwriteByteMg(uint8_t Addr, uint8_t
    ↪ CS, uint8_t Reg, uint8_t Data);
35 #endif
36 #endif
37
38
39
40 //////////////////////////////////////////
41
42 class Index{
43 public:
44     Index(uint8_t _NumOfIndices){
45         NumOfIndices = _NumOfIndices;
46         idx = 0;
47     }
48     uint8_t getIndex(){return idx;}
49     Index operator++(int a){
50         if(idx >= NumOfIndices - 1){
51             Index i_temp(NumOfIndices);
52             idx = 0;
53             i_temp.idx = 0;

```

```

54         return i_temp;
55     }
56     else{
57         Index i_temp(NumOfIndices);
58         idx++;
59         i_temp.idx = idx;
60         return i_temp;
61     }
62 }
63 private:
64     uint8_t idx;
65     uint8_t NumOfIndices;
66 };
67
68 class Mean{
69 public:
70     Mean(uint8_t _NumOfValues);
71     ~Mean(){
72         delete[] val[0];
73         delete[] val[1];
74         delete[] val[2];
75         delete val;
76         delete idx;
77     }
78     void setNewValues(float NewX, float
    ↪ NewY, float NewZ);
79     float getMeanX(){return mean[0];}
80     float getMeanY(){return mean[1];}
81     float getMeanZ(){return mean[2];}
82 private:
83     Index *idx;
84     float** val;
85     float mean[3];
86     uint8_t NumOfValues;
87 };
88
89 #endif

```

## B.5. arduino.cpp

```

1  #include <QDebug>
2  #include <QString>
3
4  #include "wiringPi.h"
5  #include "wiringPiSPI.h"
6  #include "arduino.h"
7  #include "Complementos.h"
8  #include "RegistersAndMask.h"
9  #include "SPI.h"
10
11 SPI SPI;
12 _Serial Serial;
13
14 void _Serial::print(QString cadena){
15     //QDebug() << cadena;
16     fprintf(stderr,
    ↪ cadena.toLocal8Bit().data());
17 }
18
19 void _Serial::print(float val){
20     fprintf(stderr, "%.3f", val);
21 }
22
23 void _Serial::println(QString cadena){
24     qDebug() << cadena;

```

```

25 }
26
27 void _Serial::println(float val){
28     fprintf(stderr, "%.3f \n", val);
29 }
30
31 void _Serial::println(void){
32     fprintf(stderr, "\n");
33 }
34
35 QString String (float value){
36     return QString::number(value, 'f', 3);
37 }
38
39 QString String (int value){
40     return QString::number(value);
41 }
42
43
44
45
46
47
48
49
50
51
52
53

```

```

54
55
56 //////////////// SPI.h //////////////////////////
57 int _SPI::setClockDivider(uint8_t
    ↪ Divider){ //Cuidado: Unable to
    ↪ open SPI: demasiados archivos abiertos
58     int success = wiringPiSPISetupMode(0,
    ↪ 16000000/Divider, 3);
59     return success;
60 }
61
62 uint8_t _SPI::transfer(uint8_t Data){
63     wiringPiSPIDataRW(0, &Data, 1);
64     return Data;
65 }
66
67
68
69
70
71
72
73
74
75
76
77
78 //////////////// Complementos.h
79 ↪ ////////////////
80 #define READ_FLAG 0b10000000
81 #define WRITE_FLAG 0b01111111
82
83 #if defined(I2C_PROTOCOL)
84 //Funcion auxiliar lectura
85 void I2Cread(uint8_t Address, uint8_t
    ↪ Register, uint8_t nBytes, uint8_t*
    ↪ Data)
86 {
87     Wire.beginTransaction(Address);
88     Wire.write(Register);
89     Wire.endTransmission();
90
91     Wire.requestFrom(Address, nBytes);
92     uint8_t index = 0;
93     while (Wire.available())
94         Data[index++] = Wire.read();
95 }
96
97 // Funcion auxiliar de escritura
98 void I2CwriteByte(uint8_t Address, uint8_t
    ↪ Register, uint8_t Data)
99 {
100     Wire.beginTransaction(Address);
101     Wire.write(Register);
102     Wire.write(Data);
103     Wire.endTransmission();
104 }
105
106 void I2CwriteNBytes(uint8_t Address,
    ↪ uint8_t Register, uint8_t nBytes,
    ↪ uint8_t* Data)
107 {
108     Wire.beginTransaction(Address);
109     Wire.write(Register);
110     for(int i=0; i<nBytes; i++){
111         Wire.write(Data[i]);
112     }
113     Wire.endTransmission(true);
114 }
115
116 #elif defined(SPI_PROTOCOL)
117 uint us_delay = 1000;
118
119 void SPIread(uint8_t CS, uint8_t Reg,
    ↪ uint8_t nBytes, uint8_t* Data){
120     digitalWrite(CS, LOW);
121     delayMicroseconds(us_delay);
122     SPI.transfer(Reg | 0b10000000); //Primer
    ↪ bit en "1" para Lectura
123     uint8_t index = 0;
124     while (index < nBytes){
125         Data[index] = 0;
126         Data[index] = SPI.transfer(0x00);
127         ++index;
128     }
129     digitalWrite(CS, HIGH);
130     delayMicroseconds(us_delay);
131 }
132
133 void SPIwriteByte(uint8_t CS, uint8_t Reg,
    ↪ uint8_t Data){
134     digitalWrite(CS, LOW);
135     delayMicroseconds(us_delay);
136     SPI.transfer(Reg & 0b01111111);
137     SPI.transfer(Data);
138     digitalWrite(CS, HIGH);
139     delayMicroseconds(us_delay);
140 }
141
142 void SPIwriteNBytes(uint8_t CS, uint8_t
    ↪ Reg, uint8_t nBytes, uint8_t* Data){
143     digitalWrite(CS, LOW);
144     delayMicroseconds(us_delay);
145     SPI.transfer(Reg & 0b01111111); //Primer
    ↪ bit en "0" para Escritura
146     uint8_t index = 0;
147     while (index < nBytes){
148         SPI.transfer(Data[index++]);
149     }
150     digitalWrite(CS, HIGH);
151     delayMicroseconds(us_delay);
152 }
153
154 #ifdef MPU9250
155 void SPIreadMg(uint8_t Addr, uint8_t CS,
    ↪ uint8_t Reg, uint8_t nBytes, uint8_t*
    ↪ Data){
156     SPIwriteByte(CS, MPU_I2C_SLV0_ADDR, Addr
    ↪ | READ_FLAG); // Prepare the I2C
    ↪ slave address and set for read.
157     SPIwriteByte(CS, MPU_I2C_SLV0_REG, Reg);
    ↪ // I2C slave 0 register address from
    ↪ where "data" will be collected
158     if (nBytes > 7) {return;}
159     SPIwriteByte(CS, MPU_I2C_SLV0_CTRL, 0x80
    ↪ | nBytes);
160     delay(2);
161     SPIread(CS, MPU_EXT_SENS_DATA_00,
    ↪ nBytes, Data);
162 }
163
164 void SPIwriteByteMg(uint8_t Addr, uint8_t
    ↪ CS, uint8_t Reg, uint8_t Data){
165     SPIwriteByte(CS, MPU_I2C_SLV0_ADDR, Addr
    ↪ & WRITE_FLAG); // Prepare the I2C
    ↪ slave address and set for read.
166     SPIwriteByte(CS, MPU_I2C_SLV0_REG, Reg);
    ↪ // I2C slave 0 register address from
    ↪ where "data" will be collected
167     SPIwriteByte(CS, MPU_I2C_SLV0_DO, Data);
168     SPIwriteByte(CS, MPU_I2C_SLV0_CTRL,
    ↪ 0x81);

```

```

169 }
170 #endif
171 #endif
172
173
174
175 //*****
176 Mean::Mean(uint8_t NumOfValues){
177     NumOfValues = _NumOfValues;
178     val = new float*[3];
179     val[0] = new float[NumOfValues];
180     val[1] = new float[NumOfValues];
181     val[2] = new float[NumOfValues];
182     idx = new Index(NumOfValues);
183     mean[0] = 0.0; mean[1] = 0.0; mean[2] =
        ↳ 0.0;
184     for(int i=0;i<NumOfValues;i++){
185         val[0][i] = 0.0f;
186         val[1][i] = 0.0f;
187         val[2][i] = 0.0f;

```

```

188     }
189 }
190 void Mean::setNewValues(float NewX, float
        ↳ NewY, float NewZ){
191     uint8_t ix = idx->getIndex();
192     float NOV = float(NumOfValues);
193     mean[0] = (mean[0]*NOV - val[0][ix] +
        ↳ NewX)/NOV;
194     mean[1] = (mean[1]*NOV - val[1][ix] +
        ↳ NewY)/NOV;
195     mean[2] = (mean[2]*NOV - val[2][ix] +
        ↳ NewZ)/NOV;
196     val[0][ix] = NewX;
197     val[1][ix] = NewY;
198     val[2][ix] = NewZ;
199     (*idx)++;
200 }

```

## B.6. mainwindow.cpp

```

1 #include "mainwindow.h"
2 #include "ui_mainwindow.h"
3
4 #include <QUdpSocket>
5
6 extern QUdpSocket* mUdpSocket;
7
8 MainWindow::MainWindow(QWidget *parent) :
9     QMainWindow(parent),
10     ui(new Ui::MainWindow)
11 {
12     ui->setupUi(this);
13 }
14
15 MainWindow::~MainWindow()
16 {

```

```

17     delete ui;
18 }
19
20 void MainWindow::on_pushButton_clicked()
21 {
22     QString sAUX = ui->lineEdit->text();
23     QByteArray prueba =
        ↳ QByteArray(sAUX.toUtf8());
24     mUdpSocket->writeDatagram(prueba,
        ↳ QHostAddress("192.168.1.126"),
        ↳ 63604);
25 }

```

## B.7. mainwindow.h

```

1 #ifndef MAINWINDOW_H
2 #define MAINWINDOW_H
3
4 #include <QMainWindow>
5
6 namespace Ui {
7 class MainWindow;
8 }
9
10 class MainWindow : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:

```

```

15     explicit MainWindow(QWidget *parent =
        ↳ 0);
16     ~MainWindow();
17
18 private slots:
19     void on_pushButton_clicked();
20
21 private:
22     Ui::MainWindow *ui;
23 };
24
25 #endif // MAINWINDOW_H
26

```

## B.8. mainwindow.ui

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ui version="4.0">
3   <class>MainWindow</class>
4   <widget class="QMainWindow"
5     ↪ name="MainWindow">
6     <property name="geometry">
7       <rect>
8         <x>0</x>
9         <y>0</y>
10        <width>474</width>
11        <height>225</height>
12      </rect>
13    </property>
14    <property name="windowTitle">
15      <string>MainWindow</string>
16    </property>
17    <widget class="QWidget"
18      ↪ name="centralWidget">
19      <widget class="QLineEdit"
20        ↪ name="lineEdit">
21        <property name="geometry">
22          <rect>
23            <x>30</x>
24            <y>30</y>
25            <width>391</width>
26            <height>81</height>
27          </rect>
28        </property>
29        <property name="font">
30          <font>
31            <pointsize>18</pointsize>
32          </font>
33        </property>
34      </widget>
35      <widget class="QWidget"
36        ↪ name="layoutWidget">
37        <property name="geometry">
38          <rect>
39            <x>30</x>
40            <y>140</y>
41            <width>411</width>
42            <height>30</height>
43          </rect>
44        </property>
45        <layout class="QHBoxLayout"
46          ↪ name="horizontalLayout">
47          <item>
48            <widget class="QPushButton"
49              ↪ name="pushButton">
50              <property name="font">
51                <font>
52                  <pointsize>13</pointsize>
53                </font>
54              </property>
55              <property name="text">
56                <string>Enviar</string>
57              </property>
58            </widget>
59          </item>
60          <item>
61            <spacer name="horizontalSpacer">
62              <property name="orientation">
63                <enum>Qt::Horizontal</enum>
64              </property>
65              <property name="sizeHint">
66                ↪ stdset="0">
67                <size>
68                  <width>40</width>
69                  <height>20</height>
70                </size>
71              </property>
72            </spacer>
73          </item>
74        </layout>
75      </widget>
76    </widget>
77    <widget class="QMenuBar" name="menuBar">
78      <property name="geometry">
79        <rect>
80          <x>0</x>
81          <y>0</y>
82          <width>474</width>
83          <height>19</height>
84        </rect>
85      </property>
86    </widget>
87    <widget class="QToolBar"
88      ↪ name="mainToolBar">
89      <attribute name="toolBarArea">
90        <enum>TopToolBarArea</enum>
91      </attribute>
92      <attribute name="toolBarBreak">
93        <bool>false</bool>
94      </attribute>
95    </widget>
96    <widget class="QStatusBar"
97      ↪ name="statusBar"/>
98  </widget>
99  <layoutdefault spacing="6" margin="11"/>
100 </resources/>
101 </connections/>
102 </ui>
```

## B.9. matlab.cpp

```

1  #include <initializer_list>
2  #include <stdint>
3  #include <stdlib>
4  #include <QDebug>
5  #include <stdio.h>
6  #include <math.h>
7
8  #include "matlab.h"
9  #include "wiringPi.h"
10 #include <QDebug>
11
12 Vector Vector::operator()(uint8_t p1,
13 ↪ uint8_t p2) const{
14     uint8_t dim = p2+1-p1;
15     if(p2 < dim && dim >= _dim && p2 >
16 ↪ p1){
17         Vector v(_dim);
18         for(int i=0;i<dim;i++){
19             v[i] = vtr[p1 + i];
20         }
21         return v;
22     }else{
23         fprintf(stderr, "Error
24 ↪ Vector::Operator()(p1,p2): Las
25 ↪ dimensiones son incorrectas
26 ↪ \n");
27         exit(0);
28     }
29 }
30
31 void Vector::operator=(Vector& v2){
32     if(dim == v2.getDim()){
33         for(int i=0;i<dim;i++){
34             vtr[i] = v2[i];
35         }
36     }else{
37         fprintf(stderr, "Error
38 ↪ Vector::Operator=(&): Las
39 ↪ dimensiones son diferentes
40 ↪ \n");
41         exit(0);
42     }
43 }
44
45 void Vector::operator=(Vector&& v2){
46     if(dim == v2.getDim()){
47         for(int i=0;i<dim;i++){
48             vtr[i] = v2[i];
49         }
50     }else{
51         fprintf(stderr, "Error
52 ↪ Vector::Operator=(&&): Las
53 ↪ dimensiones son diferentes
54 ↪ \n");
55         exit(0);
56     }
57 }
58
59 Vector operator*(const Vector& _v, float
60 ↪ cte){
61     uint8_t dim = _v.getDim();
62     Vector v(_v);
63     for(int i=0;i<dim;i++){
64         v(i)=v(i)*cte;
65     }
66     return v;
67 }
68
69 Vector operator*(const Vector& v1, const
70 ↪ Vector& v2){
71     if(v1.getDim() == v2.getDim()){
72         uint8_t dim = v1.getDim();
73         Vector Res(_dim);
74         for(int i=0;i<dim;i++){
75             Res[i] = v1[i] * v2[i];
76         }
77         return Res;
78     }else{
79         fprintf(stderr, "Error friend
80 ↪ Vector operator*(Vector,
81 ↪ Vector) \n");
82         exit(0);
83     }
84 }
85
86 Vector operator/(const Vector& _v, float
87 ↪ cte){
88     Vector v(_v);
89     uint8_t dim = v.getDim();
90     for(int i=0;i<dim;i++){
91         v(i)=v(i)/cte;
92     }
93     return v;
94 }
95
96 Vector operator/(float cte, const Vector&
97 ↪ v){
98     Vector v(_v);
99     uint8_t dim = v.getDim();
100    for(int i=0;i<dim;i++){
101        v(i)=cte/v(i);
102    }
103    return v;
104 }
105
106 Vector operator/(const Vector& v1, const
107 ↪ Vector& v2){
108     if(v1.getDim() == v2.getDim()){
109         uint8_t dim = v1.getDim();
110         Vector Res(_dim);
111         for(int i=0;i<dim;i++){
112             Res[i] = v1[i] / v2[i];
113         }
114         return Res;
115     }else{
116         fprintf(stderr, "Error
117 ↪ Operator/(Vector, Vector)
118 ↪ \n");
119         exit(0);
120     }
121 }
122
123 Vector operator+(const Vector& v1, const
124 ↪ Vector& v2){
125     if(v1.getDim() == v2.getDim()){
126         uint8_t dim = v1.getDim();
127         Vector Res(_dim);
128         for(int i=0;i<dim;i++){
129             Res[i] = v1[i] + v2[i];
130         }
131         return Res;
132     }else{
133

```



```

109     fprintf(stderr, "Error friend
110     ↪ Vector operator+() \n");
111     exit(0);
112 }
113 Vector operator-(const Vector& v1, const
114 ↪ Vector& v2){
115     if(v1.getDim() == v2.getDim()){
116         uint8_t _dim = v1.getDim();
117         Vector Res(_dim);
118         for(int i=0;i<_dim;i++){
119             Res[i] = v1[i] - v2[i];
120         }
121         return Res;
122     }else{
123         fprintf(stderr, "Error friend
124         ↪ Vector operator+() \n");
125         exit(0);
126 }
127 Vector operator-(float cte, const Vector&
128 ↪ _v){
129     uint8_t _dim = _v.getDim();
130     Vector v(_v);
131     for(int i=0;i<_dim;i++) v(i) = cte -
132     ↪ v(i);
133     return v;
134 }
135 Vector operator<(const Vector& _v, float
136 ↪ cte){
137     uint8_t _dim = _v.getDim();
138     Vector v(_v);
139     for(int i=0;i<_dim;i++){
140         if(_v[i] < cte) v(i) = 1.0f;
141         else v(i) = 0.0f;
142     }
143     return v;
144 }
145 Vector operator<(float cte, const Vector&
146 ↪ _v){
147     uint8_t _dim = _v.getDim();
148     Vector v(_v);
149     for(int i=0;i<_dim;i++){
150         if(cte < _v[i]) v(i) = 1.0f;
151         else v(i) = 0.0f;
152     }
153     return v;
154 }
155 Vector operator&&(const Vector& v1, const
156 ↪ Vector& v2){
157     if(v1.getDim() == v2.getDim()){
158         uint8_t _dim = v1.getDim();
159         Vector v(_v);
160         for(int i=0;i<_dim;i++){
161             if(v1[i] == 1.0f && v2[i] ==
162             ↪ 1.0f) v(i) = 1.0f;
163             else
164             ↪ v(i) = 0.0f;
165         }
166         return v;
167     }else{
168         fprintf(stderr, "Error
169         ↪ operator&(Vector, Vector)
170         ↪ \n");
171         exit(0);
172     }
173 }
174 Vector operator!(const Vector& _v){
175     uint8_t _dim = _v.getDim();
176     Vector v(_v);
177     for(int i=0;i<_dim;i++){
178         if(v[i] == 0.0) v(i) = 1.0f;
179         else v(i) = 0.0f;
180     }
181     return v;
182 }
183 float norm(const Vector &v){
184     float sum = 0;
185     for(int i=0;i<v.getDim();i++) sum +=
186     ↪ pow(v(i),2);
187     return sqrt(sum);
188 }
189 float dot(const Vector &v1, const Vector
190 ↪ &v2){
191     float sum = 0;
192     uint8_t _dim = v1.getDim();
193     if(v1.getDim() == v2.getDim()){
194         for(int i=0;i<_dim;i++) sum +=
195         ↪ v1(i)*v2(i);
196         return sum;
197     }else{
198         fprintf(stderr, "Dot() Error:
199         ↪ Vecotres de diferente
200         ↪ dimensión \n");
201         return 0;
202     }
203 }
204 Vector quatconj(const Vector &_v){
205     if(_v.getDim() == 4){
206         Vector Res(_v);
207         Res[1] = - Res[1];
208         Res[2] = - Res[2];
209         Res[3] = - Res[3];
210         return Res;
211     }else{
212         fprintf(stderr, "quatconj() Error:
213         ↪ Vector de dimensión diferente
214         ↪ a 4 \n");
215         exit(0);
216     }
217 }
218 Vector append(const Vector &v1, const
219 ↪ Vector &v2){
220     uint8_t _dim1 = v1.getDim();
221     uint8_t _dim2 = v2.getDim();
222     Vector Res(_dim1+_dim2);
223     for(int i=0;i<_dim1;i++){
224         Res[i] = v1[i];
225     }
226     for(int i=0;i<_dim2;i++){
227         Res[_dim1+i] = v2[i];
228     }
229     return Res;
230 }
231 Vector zeros(uint8_t n){
232     Vector v(n);
233     for(int i=0;i<n;i++) v(i) = 0;
234     return v;
235 }
236 Vector ones(uint8_t dim){
237     Vector temp(dim);
238     for(int i=0;i<dim;i++){
239         temp[i] = 1;
240     }
241     return temp;
242 }

```

```

229
230
231
232
233
234
235
236 Matriz::Matriz(uint8_t _dimV, uint8_t
↳ dimH){
237     dimV = _dimV;
238     dimH = _dimH;
239     mtx = new Vector[_dimV];
240     for(int i=0; i<dimV; i++){
241         mtx[i].resize(_dimH);
242     }
243 }
244 Matriz::Matriz(const Matriz &m){
245     dimV = m.getDimV();
246     dimH = m.getDimH();
247     mtx = new Vector[dimV];
248     for(int i=0; i<dimV; i++){
249         mtx[i].resize(dimH);
250     }
251     for(int i=0; i<dimV; i++){
252         for(int j=0; j<dimH; j++){
253             mtx[i][j] = m[i][j];
254         }
255     }
256 }
257 Matriz::Matriz(const Matriz &&m){
258     dimV = m.getDimV();
259     dimH = m.getDimH();
260     mtx = new Vector[dimV];
261     for(int i=0; i<dimV; i++){
262         mtx[i].resize(dimH);
263     }
264     for(int i=0; i<dimV; i++){
265         for(int j=0; j<dimH; j++){
266             mtx[i][j] = m[i][j];
267         }
268     }
269 }
270
271
272 void Matriz::operator=(Matriz& m){
273     uint8_t _dimV = m.getDimV();
274     uint8_t _dimH = m.getDimH();
275     if(dimV == _dimV && dimH == _dimH){
276         for(int i=0; i<dimV; i++){
277             for(int j=0; j<dimH; j++){
278                 mtx[i][j] = m[i][j];
279             }
280         }
281     }else{
282         fprintf(stderr, "Error
↳ operator=(MatrizCuadrada &)");
283         exit(0);
284     }
285 }
286
287
288 Vector& Matriz::operator[](uint16_t _idx)
↳ const{ // //////////// REVISAR QUE
ESTO FUNCIONE: CUIDADO
289     if(_idx < dimV){
290         return mtx[_idx];
291     }
292     else{
293         fprintf(stderr, "Error Matriz
Operator[]: Indice mayor que
↳ la dimension \n");
294         exit(0);

```

```

295     }
296 }
297 Vector& Matriz::operator()(uint8_t _idx)
↳ const{
298     if(_idx < dimV){
299         return mtx[_idx];
300     }
301     else{
302         fprintf(stderr, "Error Matriz
Operator(): Indice mayor que
↳ la dimension \n");
303         exit(0);
304     }
305 }
306 void Matriz::operator=(Matriz&& m){
307     uint8_t _dimV = m.getDimV();
308     uint8_t _dimH = m.getDimH();
309     if(dimV == _dimV && dimH == _dimH){
310         for(int i=0; i<dimV; i++){
311             for(int j=0; j<dimH; j++){
312                 mtx[i][j] = m[i][j];
313             }
314         }
315     }else{
316         fprintf(stderr, "Error
operator=(MatrizCuadrada
↳ &&)");
317         exit(0);
318     }
319 }
320
321
322
323
324
325
326
327
328
329
330 void MatrizCuadrada::operator=(const
↳ MatrizCuadrada& m){
331     if(dimV == m.getDim()){
332         for(int i=0; i<dimV; i++){
333             for(int j=0; j<dimV; j++){
334                 mtx[i][j] = m[i][j];
335             }
336         }
337     }else{
338         fprintf(stderr, "Error
↳ operator=(MatrizCuadrada &)");
339         exit(0);
340     }
341 }
342 void MatrizCuadrada::operator=(const
↳ MatrizCuadrada&& m){
343     if(dimV == m.getDim()){
344         for(int i=0; i<dimV; i++){
345             for(int j=0; j<dimV; j++){
346                 mtx[i][j] = m[i][j];
347             }
348         }
349     }else{
350         fprintf(stderr, "Error
operator=(MatrizCuadrada
↳ &&)");
351         exit(0);
352     }
353 }
354
355

```



```

356 MatrizCuadrada Inversa(const
↳ MatrizCuadrada& _mtx){
357     uint8_t dim = _mtx.getDim();
358     float Mat[dim][2*dim];
359     Vector* const temp = _mtx.getMatriz();
360     //float Inv[dim][dim];
361
362     for(int i=0;i<dim;i++){
363         for(int j=0;j<dim;j++){ //
364             // Agregar matriz a la cuál
365             // queremos obtener su INVERSA
366             Mat[i][j] = temp[i][j];
367         }
368         for(int j=dim;j<2*dim;j++){ //
369             // Agregar matriz identidad
370             if(i==(j-dim)) Mat[i][j] = 1;
371             else Mat[i][j] = 0;
372         }
373     }
374     float val_Piv;
375     for(int piv=0;piv<(dim-1);piv++){ //
376         // piv = (#de Pivote - 1)
377         /* Si el pivote es Igual a "0",
378         Entonces: Intercambiamos filas */
379         if( std::abs(Mat[piv][piv]) >
380             ↳ 0.0001 ){
381             val_Piv = 1 / Mat[piv][piv];
382         }else{
383             for(int i=piv+1;i<dim;i++){
384                 if(Mat[i][piv] > 0.0001){
385                     for(int
386                         ↳ j=0;j<2*dim;j++){
387                         float aux =
388                             ↳ Mat[piv][j];
389                         Mat[piv][j] =
390                             ↳ Mat[i][j];
391                         Mat[i][j] = aux;
392                     }
393                     break;
394                 }else{
395                     if(i == dim-1){
396                         fprintf(stderr,
397                             "Error
398                             ↳ Inversa(): La
399                             ↳ matriz no es
400                             ↳ invertible
401                             ↳ \n");
402                         MatrizCuadrada
403                             ↳ error =
404                             ↳ zerosMC(dim);
405                         // La matriz
406                         ↳ NO tiene
407                         ↳ inversa
408                         return error;
409                     }
410                 }
411             }
412         }
413     }
414     /* Problema del pivote igual a "0"
415     ↳ resuelto hasta esta linea */
416     for(int j=piv;j<2*dim;j++){
417         //Multiplicar "fila pivote"
418         ↳ para que el pivote sea = 1
419         Mat[piv][j] *= val_Piv;
420     }
421     float val;
422     for(int i=piv+1;i<dim;i++){ //
423         ↳ Reucción Gauss
424
425         val = Mat[i][piv];
426         for(int j=piv;j<2*dim;j++){
427             Mat[i][j] = Mat[i][j] -
428                 ↳ val*Mat[piv][j];
429         }
430     }
431 }
432
433 MatrizCuadrada Inv(dim);
434 // float** Inv = new float* [dim];
435 // for(int i=0;i<dim;i++) Inv[i] = new
436 // float[dim];
437
438 /* Replantear
439 for(int i=dim-1;i>=0;i--){
440     for(int k=0;k<dim;k++){
441         Inv[i][k] = Mat[i][k+dim];
442         for(int j=dim-1;j>i;j--){
443             Inv[i][k] = Inv[i][k] -
444                 ↳ Mat[j][k]*Inv[j][i];
445         }
446         Inv[i][k] = Inv[i][k] /
447             ↳ Mat[i][i];
448     }
449 }
450 Fin Replantear */
451
452 for(int j=0;j<dim;j++){
453     for(int i=dim-1;i>=0;i--){
454         Inv[i][j] = Mat[i][j+dim];
455         for(int k=dim-1;k>i;k--){
456             Inv[i][j] = Inv[i][j] -
457                 ↳ Mat[i][k]*Inv[k][j];
458         }
459         Inv[i][j] = Inv[i][j] /
460             ↳ Mat[i][i];
461     }
462 }
463
464 return Inv;
465 }
466
467 MatrizCuadrada zerosMC(uint8_t _dim){
468     MatrizCuadrada temp(_dim);
469     for(int i=0;i<_dim;i++){
470         for(int j=0;j<_dim;j++){
471             temp[i][j] = 0.0f;
472         }
473     }
474     return temp;
475 }
476
477 Matriz ones(uint8_t dim1, uint8_t dim2){
478     Matriz temp(dim1, dim2);
479     for(int i=0;i<dim1;i++){
480         for(int j=0;j<dim2;j++){
481             temp[i][j] = 1;
482         }
483     }
484     return temp;
485 }
486
487 //static MatrizCuadrada Q2RM(3);
488 MatrizCuadrada Quat2RotMat(const Vector
489     ↳ &q){
490     if(q.getDim() == 4){
491         MatrizCuadrada R(3);
492         R[0][0] = float(2*pow(q[0],2) +
493             ↳ 2*pow(q[1],2) - 1);

```

```

462 R[0][1] = float(2*(q[1]*q[2] -
    ↪ q[0]*q[3]));
463 R[0][2] = float(2*(q[1]*q[3] +
    ↪ q[0]*q[2]));
464 R[1][0] = float(2*(q[1]*q[2] +
    ↪ q[0]*q[3]));
465 R[1][1] = float(2*pow(q[0],2) +
    ↪ 2*pow(q[2],2) - 1);
466 R[1][2] = float(2*(q[2]*q[3] -
    ↪ q[0]*q[1]));
467 R[2][0] = float(2*(q[1]*q[3] -
    ↪ q[0]*q[2]));
468 R[2][1] = float(2*(q[2]*q[3] +
    ↪ q[0]*q[1]));
469 R[2][2] = float(2*pow(q[0],2) +
    ↪ 2*pow(q[3],2) - 1);
470 //Q2RM = R;
471 return R;
472 }else{
473     fprintf(stderr, "Quat2Rotmat Error
    ↪ Fatal: Vector no es de
    ↪ dimensión 4 \n");
    exit(0);
474 }
475 }
476 }
477 Matriz operator-(const Matriz& _m, const
    ↪ Vector& _v){
478     if(_m.getDimH() == _v.getDim()){
479         uint8_t _dimV = _m.getDimV();
480         uint8_t _dimH = _m.getDimH();
481         Matriz Res(_dimV, _dimH);
482         for(int i=0; i<_dimV; i++){
483             for(int j=0; j<_dimH; j++){
484                 Res[i][j] = _m[i][j] -
    ↪ _v[j];
485             }
486         }
487         return Res;
488     }else{
489         fprintf(stderr, "Error
    ↪ operator-(Matriz, Vector)
    ↪ \n");
    exit(0);
490     }
491 }
492 }
493 Matriz operator/(const Matriz& _m, const
    ↪ Vector& _v){
494     if(_m.getDimH() == _v.getDim()){
495         uint8_t _dimV = _m.getDimV();
496         uint8_t _dimH = _m.getDimH();
497         Matriz Res(_dimV, _dimH);
498         for(int i=0; i<_dimV; i++){
499             for(int j=0; j<_dimH; j++){
500                 Res[i][j] = _m[i][j] /
    ↪ _v[j];
501             }
502         }
503         return Res;
504     }else{
505         fprintf(stderr, "Error
    ↪ operator/(Matriz, Vector)
    ↪ \n");
    exit(0);
506     }
507 }
508 }
509 Matriz pow(const Matriz& _m){

```

```

510     uint8_t _dimV = _m.getDimV();
511     uint8_t _dimH = _m.getDimH();
512     Matriz Res(_dimV, _dimH);
513     for(int i=0; i<_dimV; i++){
514         for(int j=0; j<_dimH; j++){
515             Res[i][j] = pow(Res[i][j], 2);
516         }
517     }
518     return Res;
519 }
520 }
521 Vector operator*(const Matriz& _m, const
    ↪ Vector& _v){
522     if(_m.getDimH() == _v.getDim()){
523         uint8_t _dimH = _m.getDimH();
524         uint8_t _dimV = _m.getDimV();
525         Vector Res(_dimV);
526         for(int i=0; i<_dimV; i++){
527             Res[i] = 0.0f;
528             for(int j=0; j<_dimH; j++){
529                 Res[i] += _m[i][j] *
    ↪ _v[j];
530             }
531         }
532         return Res;
533     }else{
534         fprintf(stderr, "Error
    ↪ operator*(Matriz, Vector):
    ↪ Dimensiones diferentes \n");
    exit(0);
535     }
536 }
537 }
538 Vector operator*(const Vector& _v, const
    ↪ Matriz& _m){
539     if(_m.getDimV() == _v.getDim()){
540         uint8_t _dimH = _m.getDimH();
541         uint8_t _dimV = _m.getDimV();
542         Vector Res(_dimH);
543         for(uint8_t i=0; i<_dimH; i++){
544             Res[i] = 0.0f;
545             for(uint8_t j=0; j<_dimV; j++){
546                 Res[i] += _v[j] *
    ↪ _m[j][i];
547             }
548         }
549         return Res;
550     }else{
551         fprintf(stderr, "Error
    ↪ operator*(Matriz, Vector):
    ↪ Dimensiones diferentes \n");
    exit(0);
552     }
553 }
554 }
555 MatrizCuadrada operator*(const
    ↪ MatrizCuadrada &_m1, const
    ↪ MatrizCuadrada &_m2){
556     if(_m1.getDimH() == _m2.getDimV()){
557         uint8_t _dimV = _m1.getDimV();
558         uint8_t _dimH = _m2.getDimH();
559         uint8_t _dimHV = _m1.getDimH();
560         MatrizCuadrada Res(_dimV);
561         for(uint8_t i=0; i<_dimV; i++){
562             for(uint8_t j=0; j<_dimH; j++){
563                 Res[i][j] = 0.0f;
564                 for(uint8_t
    ↪ k=0; k<_dimHV; k++){
565                     Res[i][j] += _m1[i][k]
    ↪ * _m2[k][j];

```

```

566     }
567 }
568 }
569 return Res;
570 }else{
571     fprintf(stderr, "Error
operator*(Matriz, Matriz):
Dimensiones diferentes \n");
exit(0);
}
}
572
573 }
574 }
575
576 Vector quatmultiply(const Vector& q1,
577     ↪ const Vector& q2){
578     if(q1.getDim() == 4 && q2.getDim() ==
579     ↪ 4){
580         Vector Res(4);
581         Res[0] = q1[0]*q2[0] - (q1[1]*q2[1]
582         ↪ + q1[2]*q2[2] + q1[3]*q2[3]);
583         Res[1] = q1[0]*q2[1] + q1[1]*q2[0]
584         ↪ + q1[2]*q2[3] - q1[3]*q2[2];
585         Res[2] = q1[0]*q2[2] + q1[2]*q2[0]
586         ↪ + q1[3]*q2[1] - q1[1]*q2[3];
587         Res[3] = q1[0]*q2[3] + q1[3]*q2[0]
588         ↪ + q1[1]*q2[2] - q1[2]*q2[1];
589         return Res;
590     }else{
591         fprintf(stderr, "Error
quatmultiply: Dimensiones
diferentes a 4 \n");
592         exit(0);
593     }
594 }
595
596 static Matriz SysEqf(const Matriz& _m){
597     uint8_t _dimV = _m.getDimV();
598     uint8_t _dimH = 6;
599     Matriz Res(_dimV, _dimH);
600     for(int i=0; i<_dimV; i++){
601         Res[i][0] = pow(_m[i][0], 2);
602         Res[i][1] = -2*_m[i][0];
603         Res[i][2] = pow(_m[i][1], 2);
604         Res[i][3] = -2*_m[i][1];
605         Res[i][4] = pow(_m[i][2], 2);
606         Res[i][5] = -2*_m[i][2];
607     }
608     return Res; // [x.^2 -2*x y.^2 -2*y
609     ↪ z.^2 -2*z];
610 }
611
612 MatrizCuadrada mult_Mt2M(const Matriz& m1,
613     ↪ const Matriz& m2){
614     if(m1.getDimV() == m2.getDimV() &&
615     ↪ m1.getDimH() == m2.getDimH()){
616         uint8_t _dimH = m1.getDimH();
617         uint8_t _dimV = m1.getDimV();
618         MatrizCuadrada Res(_dimH);
619         for(int i=0; i<_dimH; i++){
620             for(int j=0; j<_dimH; j++){
621                 Res[i][j] = 0.0f;
622                 for(int k=0; k<_dimV; k++){
623                     Res[i][j] +=
624                     ↪ m1[k][i]*m2[k][j];
625                 }
626             }
627         }
628         return Res;
629     }else{

```

```

620         fprintf(stderr, "Error mult_vvToM
621         ↪ \n");
622         exit(0);
623     }
624 }
625
626 Vector RegresiLinealMultip(const Matriz&
627     ↪ _m){ // [Amplitud Bias]
628     uint8_t nVal = _m.getDimV();
629     // Regresión lineal multivariable: "Ax
630     ↪ = Y" o bien, "SysEq*XRes =
631     ↪ SysRes"
632     Vector SysRes = ones(nVal); //
633     ↪ Resultado del sistema de
634     ↪ ecuaciones
635     Matriz SysEq = SysEqf(_m); // Sistema
636     ↪ de ecuaciones
637     // Res = mldivide(SysEq'*SysEq,
638     ↪ SysEq'*SysRes);
639     MatrizCuadrada Cuad = mult_Mt2M(SysEq,
640     ↪ SysEq);
641     Cuad = Inversa(Cuad);
642     Vector XRes(6);
643     if(Cuad[0][0] != 0.0f){
644         XRes = SysRes*SysEq;
645         XRes = Cuad*XRes;
646     }else{
647         Vector ampli_bias = {-1.0f, -1.0f,
648         ↪ -1.0f, -1.0f, -1.0f, -1.0f};
649         return ampli_bias;
650     }
651     // [re,co] = rref([SysEq SysRes]);
652     XRes = re(1:6,7);
653     // SysEq = [x.^2 -2*x y.^2 -2*y z.^2
654     ↪ -2*z];
655     if (XRes(0)>0 && XRes(2)>0 &&
656     ↪ XRes(4)>0){
657         // Obtención de parametros
658         ↪ elípticos
659         float lambda =
660         ↪ pow(XRes(1),2)/XRes(0) +
661         ↪ pow(XRes(3),2)/XRes(2) +
662         ↪ pow(XRes(5),2)/XRes(4);
663         float p = lambda/(1+lambda);
664         ↪ float q = 1-p; // q = 1 -
665         ↪ p*org;
666         XRes = XRes*q;
667         Vector tam = { sqrt(1/XRes(0)),
668         ↪ sqrt(1/XRes(2)),
669         ↪ sqrt(1/XRes(4)) };
670         float hx = XRes(1)/XRes(0); float
671         ↪ hy = XRes(3)/XRes(2); float
672         ↪ hz = XRes(5)/XRes(4);
673         Vector ampli_bias = {tam(0),
674         ↪ tam(1), tam(2), hx, hy, hz};
675         return ampli_bias;
676     }else{
677         Vector ampli_bias = {-1.0f, -1.0f,
678         ↪ -1.0f, -1.0f, -1.0f, -1.0f};
679         return ampli_bias;
680     }
681 }
682
683 Vector RotMat2Euler(const MatrizCuadrada&
684     ↪ R){
685     if(R.getDim() == 3){

```

```

663 // R(Phi).*R(Theta).*R(Psi)
664 // phi = -atan2d( R(2,3), R(3,3)
665 // );
666 // theta = asind(-R(3,1) );
667 // psi = atan2d( R(2,1), R(1,1)
668 // );
669 // Rz(Psi).*Ry(Theta).*Rx(Phi)
670 float phi = atan2( R(2,1),
671 // R(2,2) ) * (180.0f/PI);
672 float theta = asin(-R(2,0) ) *
673 // (180.0f/PI);
674 float psi = atan2( R(1,0),
675 // R(0,0) ) * (180.0f/PI);
676 Vector euler = {phi, theta, psi};
677 return euler;
678 }else{
679 fprintf(stderr, "Error:
680 RotMat2Euler() \n");
681 exit(0);
682 }
683 }
684 }
685 MatrizCuadrada Euler2RotMat(const Vector&
686 // V){
687 if(V.getDim() == 3){
688 MatrizCuadrada R(3);
689 float conv = 3.1425926/180;
690
691 float SPhi = sin(V[0]*conv);
692 float CPhi = cos(V[0]*conv);
693 float STheta = sin(V[1]*conv);
694 float CTheta = cos(V[1]*conv);
695 float SPsi = sin(V[2]*conv);
696 float CPsi = cos(V[2]*conv);
697
698 /// Rz(Psi).*Ry(Theta).*Rx(Phi)
699 R[0][0] = CPsi*CTheta;
700 R[0][1] = CPsi*STheta*SPhi -
701 // SPsi*CPhi;
702 R[0][2] = CPsi*STheta*CPhi +
703 // SPsi*SPhi;
704 R[1][0] = SPsi*CTheta;
705 R[1][1] = SPsi*STheta*SPhi +
706 // CPsi*CPhi;
707 R[1][2] = SPsi*STheta*CPhi -
708 // CPsi*SPhi;
709 R[2][0] = -STheta;
710 R[2][1] = CTheta*SPhi;
711 R[2][2] = CTheta*CPhi;
712 return R;
713 }else{
714 fprintf(stderr, "Error:
715 Euler2RotMat() \n");
716 exit(0);
717 }
718 }
719 Vector sqrt(const Vector& _v){
720 uint8_t dim = _v.getDim();
721 Vector v(_v);
722 for(int i=0; i<dim; i++){ v(i) =
723 // sqrt(v(i));
724 return v;
725 }
726 }
727 Vector sum(const Matriz& _m, uint8_t
728 // _idx){
729 if(_idx == 1){

```

```

716 uint8_t _dimV = _m.getDimV();
717 uint8_t _dimH = _m.getDimH();
718 Vector Res = zeros(_dimH);
719 for(int i=0; i<_dimH; i++){
720 for(int j=0; j<_dimV; j++){
721 Res[i] += _m[j][i];
722 }
723 return Res;
724 }else if(_idx == 2){
725 uint8_t _dimV = _m.getDimV();
726 uint8_t _dimH = _m.getDimH();
727 Vector Res = zeros(_dimV);
728 for(int i=0; i<_dimV; i++){
729 for(int j=0; j<_dimH; j++){
730 Res[i] += _m[i][j];
731 }
732 return Res;
733 }else{
734 fprintf(stderr, "Error sum(Matriz,
735 // uint8_t) \n");
736 exit(0);
737 }
738 }
739 float sum(const Vector& _v){
740 uint8_t _dim = _v.getDim();
741 float Res = 0.0f;
742 for(int i=0; i<_dim; i++){
743 Res += _v[i];
744 }
745 return Res;
746 }
747 static float t1 = 0.0f;
748 void tic(void){
749 t1 = float_t(micros()) / 1000000;
750 }
751 float toc(void){
752 float t2 = float_t(micros()) /
753 // 1000000;
754 return t2 - t1;
755 }
756 void pause(uint8_t seconds){
757 delay(seconds*1000);
758 }
759 void imprime(const Vector &v){
760 uint8_t dim = v.getDim();
761 for(int i=0; i<dim; i++){
762 fprintf(stderr, "%.4f ", v[i]);
763 }
764 fprintf(stderr, "\n\n");
765 }
766 void imprime(const Matriz &m){
767 uint8_t dimV = m.getDimV();
768 uint8_t dimH = m.getDimH();
769 for(int i=0; i<dimV; i++){
770 for(int j=0; j<dimH; j++){
771 fprintf(stderr, "%.4f ",
772 // m(i,j));
773 }
774 fprintf(stderr, "\n");
775 }
776 fprintf(stderr, "\n");
777 }

```

## B.10. matlab.h

```

1  #ifndef MATLAB_H
2  #define MATLAB_H
3
4  #include <cmath>
5
6  #define PI 3.1415926535f
7
8  class Vector{
9  public:
10     Vector(void){
11         dim = 0;
12         vtr = nullptr;
13     }
14     Vector(const uint8_t _dim) : vtr(new
15         ↪ float[_dim]){
16         dim = _dim;
17     }
18     Vector(uint8_t _dim, float* _pointer) :
19         ↪ vtr(_pointer){
20         dim = _dim;
21     }
22     Vector(const Vector& v) : vtr(new
23         ↪ float[v.getDim()]){
24         dim = v.getDim();
25         for(int i=0;i<dim;i++){
26             vtr[i] = v[i];
27         }
28     }
29     // Vector(const Vector& v) : vtr(new
30     ↪ float[v.getDim()]){
31     //     dim = v.getDim();
32     //     for(int i=0;i<dim;i++){
33     //         vtr[i] = v[i];
34     //     }
35     }
36     Vector(std::initializer_list<float> l){
37         dim = l.size();
38         vtr = new float[dim];
39         for(int i=0;i<dim;i++) vtr[i] =
40             ↪ l.begin()[i];
41     }
42     // Vector(int dim, ...) : vtr(new
43     ↪ float[_dim]){
44     //     dim = _dim;
45     //     va_list param;
46     //     va_start(n, param);
47     //     for(int i=0;i<dim;i++){
48     //         vtr(i) = float(va_arg(param,
49     ↪ double));
50     //     }
51     }
52     ~Vector(){
53         delete[] vtr;
54         vtr = nullptr;
55     }
56     uint8_t getDim(void) const{
57         return dim;
58     }
59     void resize(uint8_t _dim);
60     float& operator()(uint8_t pointer)
61     ↪ const{
62         return vtr[pointer];
63     }
64     float& operator[] (uint8_t pointer)
65     ↪ const{
66
67         return vtr[pointer];
68     }
69     Vector operator()(uint8_t p1, uint8_t
70     ↪ p2) const;
71     void operator=(Vector& v2);
72     void operator=(Vector&& v2);
73     float* vtr = nullptr;
74 protected:
75     uint8_t dim = 0;
76 };
77
78 inline void Vector::resize(uint8_t _dim){
79     if(dim > 0 && vtr != nullptr) delete[]
80     ↪ vtr;
81     dim = _dim;
82     vtr = new float[_dim];
83 }
84
85 Vector operator*(const Vector& _v, float
86 ↪ cte);
87 Vector operator*(const Vector& v1, const
88 ↪ Vector& v2);
89 Vector operator*(float cte, const Vector&
90 ↪ _v);
91 Vector operator/(const Vector& _v, float
92 ↪ cte);
93 Vector operator/(float cte, const Vector&
94 ↪ _v);
95 Vector operator/(const Vector& v1, const
96 ↪ Vector& v2);
97 Vector operator+(const Vector& v1, const
98 ↪ Vector& v2);
99 Vector operator-(const Vector& v1, const
100 ↪ Vector& v2);
101 Vector operator-(float cte, const Vector&
102 ↪ _v);
103 Vector operator<(const Vector& _v, float
104 ↪ cte);
105 Vector operator<(float cte, const Vector&
106 ↪ _v);
107 Vector operator&&(const Vector& v1, const
108 ↪ Vector& v2);
109 Vector operator!(const Vector& _v);
110 float norm(const Vector& v);
111 float dot(const Vector& v1, const Vector&
112 ↪ v2);
113 Vector quatconj(const Vector& _v);
114 Vector append(const Vector& v1, const
115 ↪ Vector& v2);
116 Vector zeros(uint8_t n);
117 Vector ones(uint8_t dim);
118
119 class Matriz{
120 public:
121     Matriz(uint8_t _dimV, uint8_t _dimH);
122     Matriz(const Matriz& _m);
123     Matriz(const Matriz&& _m);
124     ~Matriz(){
125         for(int i=0;i<dimV;i++){
126             mtx[i].~Vector();

```



```

107     }
108     delete[] mtx;
109     mtx = nullptr;
110 }
111 Vector* const getMatriz() const{
112     ↪ //CUIDADO
113     return mtx;
114 }
115 float& operator()(uint16_t p1, uint8_t
116 p2) const{ // ////////////
117     ↪ REVISAR QUE ESTO FUNCIONE:
118     ↪ CUIDADO
119     return mtx[p1][p2];
120 }
121 Vector& operator[](uint16_t _idx)
122     ↪ const;
123 Vector& operator()(uint8_t _idx)
124     ↪ const;
125 uint8_t getDimV(void) const { return
126     ↪ dimV;}
127 uint8_t getDimH(void) const { return
128     ↪ dimH;}
129
130 void operator=(Matriz & m);
131 void operator=(Matriz && m);
132 Vector* mtx = nullptr;
133 protected:
134     uint8_t dimV = 0;
135     uint8_t dimH = 0;
136 };
137
138 class MatrizCuadrada : public Matriz{
139 public:
140     MatrizCuadrada(uint8_t _dim) :
141         ↪ Matriz(_dim, _dim){
142     }
143     MatrizCuadrada(const MatrizCuadrada&
144         ↪ _m) : Matriz(_m){
145     }
146     MatrizCuadrada(const MatrizCuadrada&&
147         ↪ _m) : Matriz(_m){
148     }
149
150     MatrizCuadrada(std::initializer_list<float>
151         ↪ l) : Matriz(int(sqrt(l.size())),
152         ↪ int(sqrt(l.size()))){
153         ↪ uint8_t dim = dimV;
154         ↪ if(dim*dim != l.size()){
155             ↪ fprintf(stderr, "Constructor
156                 ↪ MatrizCuadrada Error
157                 ↪ Fatal: Initializer list no
158                 ↪ genera una matriz cuadrada
159                 ↪ \n");
160             ↪ delete this;
161             ↪ exit(5);
162         }
163         ↪ else{
164             ↪ for(int i=0;i<dim;i++){
165                 ↪ for(int j=0;j<dim;j++){
166                     ↪ mtx[i][j] =
167                     ↪     ↪ l.begin()[dim*i+j];
168                 }
169             }
170         }
171     }
172 }
173
174 ~MatrizCuadrada(){
175 }
176 uint8_t getDim() const{
177     return dimV;
178 }
179 void operator=(const MatrizCuadrada &
180     ↪ m);
181 void operator=(const MatrizCuadrada &&
182     ↪ m);
183
184 MatrizCuadrada t(void) const{
185     MatrizCuadrada trans =
186         ↪ MatrizCuadrada(dimV);
187     for(int i=0;i<dimV;i++){
188         for(int j=0;j<dimV;j++){
189             trans[i][j] = mtx[j][i];
190         }
191     }
192     return trans;
193 }
194 };
195
196 MatrizCuadrada Inversa(const
197     ↪ MatrizCuadrada& mtx);
198 MatrizCuadrada zerosMC(uint8_t _dim);
199 Matriz ones(uint8_t dim1, uint8_t dim2);
200 MatrizCuadrada Quat2RotMat(const Vector&
201     ↪ q);
202 Matriz operator-(const Matriz& _m, const
203     ↪ Vector& _v);
204 Matriz operator/(const Matriz& _m, const
205     ↪ Vector& _v);
206 Matriz pow(const Matriz& _m);
207
208 Vector operator*(const Matriz& _m, const
209     ↪ Vector& _v);
210 Vector operator*(const Vector& _v, const
211     ↪ Matriz& _m);
212 MatrizCuadrada operator*(const
213     ↪ MatrizCuadrada& _m1, const
214     ↪ MatrizCuadrada& _m2);
215 Vector quatmultiply(const Vector& q1,
216     ↪ const Vector& q2);
217 Vector RegresiLinealMultip(const Matriz&
218     ↪ _m);
219 Vector RotMat2Euler(const MatrizCuadrada&
220     ↪ R);
221 MatrizCuadrada Euler2RotMat(const Vector&
222     ↪ V);
223 Vector sqrt(const Vector& _v);
224 Vector sum(const Matriz& _m, uint8_t
225     ↪ _idx);
226 float sum(const Vector& _v);
227
228 void tic(void);
229 float toc(void);
230 void pause(uint8_t seconds);
231
232 void imprime(const Vector &v);
233 void imprime(const Matriz &m);
234
235 #endif // MATLAB_H

```

## B.11. Mocap.pro

```
1 #-----
2 #
3 # Project created by QtCreator
4 #   2021-05-11T15:45:04
5 #-----
6
7 QT      += core gui network
8
9
10
11 lessThan(QT_VERSION, 5.7) {
12     message("Cannot use Qt
13         ↳  ${QT_VERSION}")
14     error("Use Qt 5.7 or newer")
15 }
16 greaterThan(QT_MAJOR_VERSION, 4): QT +=
17     ↳ widgets
18 QMAKE_CXXFLAGS += -std=c++17
19
20 TARGET = Mocap
21 TEMPLATE = app
22
23 # The following define makes your compiler
24 #   ↳ emit warnings if you use
25 #   ↳ any feature of Qt which as been marked
26 #   ↳ as deprecated (the exact warnings
27 #   ↳ depend on your compiler). Please consult
28 #   ↳ the documentation of the
29 #   ↳ deprecated API in order to know how to
30 #   ↳ port your code away from it.
31 DEFINES += QT_DEPRECATED_WARNINGS
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
```

```
29 # You can also make your code fail to
30 #   ↳ compile if you use deprecated APIs.
31 # In order to do so, uncomment the
32 #   ↳ following line.
33 # You can also select to disable
34 #   ↳ deprecated APIs only up to a certain
35 #   ↳ version of Qt.
36 #DEFINES +=
37     QT_DISABLE_DEPRECATED_BEFORE=0x060000
38     ↳ # disables all the APIs deprecated
39     ↳ before Qt 6.0.0
40
41 SOURCES += main.cpp \
42     mainwindow.cpp \
43     matlab.cpp \
44     mpu9250.cpp \
45     arduino.cpp \
46     mocapsuit.cpp
47
48 HEADERS += mainwindow.h \
49     Complementos.h \
50     RegistersAndMask.h \
51     SPI.h \
52     matlab.h \
53     mpu9250.h \
54     arduino.h \
55     mocapsuit.h
56
57 FORMS += mainwindow.ui
58
59 LIBS += -L/usr/local/lib -lwiringPi
```

## B.12. mocapsuit.cpp

```
1 #include <QApplication>
2 #include <QUdpSocket>
3 #include "arduino.h"
4 #include "matlab.h"
5 #include "mocapsuit.h"
6 #include "wiringPi.h"
7 #include "RegistersAndMask.h"
8
9 extern QUdpSocket* mUdpSocket;
10
11 MocapSuit::MocapSuit(QObject* parent):
12     QObject(parent)
13 {
14     mTimer = new QTimer(this);
15     connect(mTimer, SIGNAL(timeout()),
16         ↳ this, SLOT(onTimeout()));
17     connect(mUdpSocket,
18         ↳ SIGNAL(readyRead()), this,
19         ↳ SLOT(onRecieveUDP()));
20     mUdpSocket->bind(63608,
21         ↳ QUdpSocket::ShareAddress);
22 }
23
```

```
20 void MocapSuit::all_CS_toHigh(){
21     Mpu9250* mpu;
22     QVector<Mpu9250*>::iterator j;
23     for(j=m_Sensores.begin();
24         ↳ j!=m_Sensores.end(); j++){
25         mpu = *j;
26         digitalWrite(mpu->CS, HIGH);
27     }
28 }
29 void MocapSuit::addSensor(Mpu9250* _mpu)
30 {
31     m_Sensores.append(_mpu);
32 }
33
34 void MocapSuit::setup()
35 {
36     all_CS_toHigh();
37     while(true){
38         fprintf(stderr, "WHO_AM_I: Si es igual
39             ↳ a 0x71(113), es MPU9250; si es
40             ↳ 0x73, es MPU9255; si es 0x70, es
41             ↳ MPU6500 \n");
```

```

40 Mpu9250* mpu;
41 QVector<Mpu9250*>::iterator i;
42 delay(10);
43 for(i=m_Sensores.begin();
44     ↪ i!=m_Sensores.end(); i++){
45     mpu = *i;
46     uint8_t data = 0;
47     MARGread(MPU9250_ADDR, mpu->CS,
48             ↪ WHO_AM_I, 1, &data);
49     fprintf(stderr, "WHO_AM_I sensor
50             ↪ %#d = %d \n", mpu->id, data);
51 }
52 // delay(1000);
53 // }
54 while(wait4Start){
55     ↪ QApplication::processEvents(QEventLoop::AllEvents);
56     fprintf(stderr, "wait4Start is
57             ↪ true...\n");
58     delay(2000);
59 }
60 align_Tpose(); //wait4Start = false;
61 ↪ REVISAR
62 void MocapSuit::align_Tpose()
63 {
64     Mpu9250* mpu;
65     QVector<Mpu9250*>::iterator i;
66     for(i=m_Sensores.begin();
67         ↪ i!=m_Sensores.end(); i++){
68         mpu = *i;
69         mpu->isAligned = false;
70     }
71     bool isSuitAligned = false;
72     while(!isSuitAligned){
73         for(i=m_Sensores.begin();
74             ↪ i!=m_Sensores.end(); i++){
75             mpu = *i;
76             if(mpu->isAligned == false){
77                 fprintf(stderr, "Sensor
78                     ↪ %#d: \n", mpu->id);
79                 mpu->loop(Mpu9250::Align);
80             }
81             // else{
82             //     fprintf(stderr, "Sensor
83             //         ↪ %#d: \n", mpu->id);
84             //     mpu->loop(Mpu9250::NoImprimir);
85             // }
86             all_CS_toHigh();
87             fprintf(stderr, "\n\n");
88             isSuitAligned = true;
89             for(i=m_Sensores.begin();
90                 ↪ i!=m_Sensores.end(); i++){
91                 mpu = *i;
92                 isSuitAligned = isSuitAligned
93                 ↪ && mpu->isAligned;
94             }
95             delay(10);
96 }
97 // Once wait4Start is false, then:
98 QString c2 = "Start recived";
99 QByteArray dataSnd(c2.toUtf8());

```

```

97 mUdpSocket->writeDatagram(dataSnd,
98     ↪ QHostAddress("192.168.1.126"),
99     ↪ 63604);
100 fprintf(stderr, "UPD recibido:
101     ↪ Comienza alineación, comienza el
102     ↪ programa");
103 }
104 void MocapSuit::loop()
105 {
106     mTimer->start(500);
107     while(true){
108         QVector<Mpu9250*>::iterator i;
109         Mpu9250* mpu;
110         for(i=m_Sensores.begin();
111             ↪ i!=m_Sensores.end(); i++){
112             mpu = *i;
113             fprintf(stderr, "Sensor %#d:
114                 ↪ \n", mpu->id);
115             mpu->loop();
116             all_CS_toHigh();
117         }
118         ↪ QApplication::processEvents(QEventLoop::AllEvents);
119         fprintf(stderr, "\n\n");
120         delay(10);
121     }
122 }
123 void MocapSuit::onTimeout()
124 {
125     Vector SEuler(3);
126     QString datSEND =
127     ↪ QString::number(m_Sensores.size())+"=";
128     QVector<Mpu9250*>::iterator i;
129     for(i=m_Sensores.begin();
130         ↪ i!=m_Sensores.end(); i++){
131         Mpu9250* mpu = *i;
132         SEuler =
133         ↪ RotMat2Euler(mpu->SErotM.t() *
134             ↪ mpu->Align_Ajd);
135         SEuler = (SEuler < 0.0f)*360.0f
136         ↪ + SEuler;
137         SEuler(1) = 360 - SEuler(1);
138         SEuler(2) = 360 - SEuler(2);
139         datSEND = datSEND +
140         ↪ String(SEuler(0))+" "+String(SEuler(1))+" ";
141     }
142     datSEND = datSEND+" ";
143     QByteArray prueba =
144     ↪ QByteArray(datSEND.toUtf8());
145     mUdpSocket->writeDatagram(prueba,
146         ↪ QHostAddress("192.168.1.126"),
147         ↪ 63604);
148     fprintf(stderr, "%s\n",
149         ↪ prueba.data());
150 }
151 void MocapSuit::onRecieveUDP(){
152     fprintf(stderr, "UDP package
153     ↪ Recieved...\n");
154     ↪ while(mUdpSocket->hasPendingDatagrams()){
155         QByteArray datagramaUDP;
156         ↪ datagramaUDP.resize(mUdpSocket->pendingDatagramSize());

```



```

145         mUdpSocket->readDatagram(datagramaUDP.data(),
        ↳ datagramaUDP.size());
146     QString c1 = "Start program";
147     if(QString(datagramaUDP.data()) ==
        ↳ c1){
        148         wait4Start = false;
        149     }
        150 }
        151 }

```

## B.13. mocapsuit.h

```

1  #ifndef MOCAPSUIT_H
2  #define MOCAPSUIT_H
3
4  #include "mpu9250.h"
5  #include <QVector>
6  #include <QObject>
7  #include <QTimer>
8
9  class MocapSuit : public QObject
10 {
11     Q_OBJECT
12 public:
13     explicit MocapSuit(QObject* parent =
        ↳ NULL);
14
15     void all_CS_toHigh(void);
16     void addSensor(Mpu9250* _mpu);
17
18     void setup();
19     void align_Tpose();
20     void loop();
21 public slots:
22     void onTimeout();
23     void onRecieveUDP();
24 private:
25     QVector<Mpu9250*> m_Sensores;
26     QTimer* mTimer = NULL;
27     bool wait4Start = true;
28 };
29
30 #endif // MOCAPSUIT_H

```

## B.14. mpu9250.cpp

```

1  // #include <QString>
2  // #include "matlab.h"
3  // #include "Complementos.h"
4  #include <QDebug>
5  #include <cmath>
6  #include <QUdpSocket>
7  #include <typeinfo>
8  #include <initializer_list>
9  #include "stdio.h"
10 #include "wiringPi.h"
11 #include "wiringPiSPI.h"
12
13 #include "mpu9250.h"
14 #include "SPI.h"
15 #include "arduino.h"
16 #include "RegistersAndMask.h"
17 // #include "Setup.h"
18 #include "mocapsuit.h"
19
20
21
22 /* VARIABLES GLOBALES */
23 uint8_t Acc_Gyro[14]; //Accel-Gyro DATA
24 uint8_t Mag[7]; //Magnetometer DATA
25
26 QString sAUX;
27 const float MAG_FS = 0.15;
28 const float ACC_FS = 0.59875e-3; //
        ↳ (9.81 * 2) / 2^15
29 const float GYR_FS = 7.6294e-3; // 250
        ↳ / 2^15
30 uint Mpu9250::countOfSens = 0;
31
32 Mpu9250::Mpu9250(uint _CS, MocapSuit&
        ↳ _Suit, floatList _Ac_Adj,
33         floatList _Gy_Adj,
34         floatList _Mg_Adj,
35         floatList _UE_init) :
36     CS(_CS), Ac_Adj(_Ac_Adj),
37     Gy_Adj(_Gy_Adj), Mg_Adj(_Mg_Adj),
38     UE_init(_UE_init)
39 {
40     Ac_Adj_Sens = Ac_Adj(0)/Ac_Adj(0,2);
41     ↳ //Acel_Calib
42     Ac_Adj_Offs = Ac_Adj(3,5);
43     Mg_Adj_Sens = Mg_Adj(0)/Mg_Adj(0,2);
44     ↳ //Mag_Calib
45     Mg_Adj_Offs = Mg_Adj(3,5);
46     ++countOfSens;
47     id = countOfSens;
48     _Suit.addSensor(this);
49 }
50
51 Mpu9250::~Mpu9250(){
52     pinMode(CS, INPUT);
53 }
54
55 void Mpu9250::setup(void){
56     pinMode(CS, OUTPUT);

```

```

54 digitalWrite(CS, HIGH);
55 delayMicroseconds(10);
56 delay(10);
57
58 #ifndef SPI_PROTOCOL
59 uint8_t Dato[2] = {29,44};
60 uint8_t Data[2];
61
62 while(true){
63 SPIwriteBytes(CS, 99, 2, Dato);
64     //Gyro_DLPF => 5 Hz
65     for(uint8_t Reg=1; Reg<=125;
66         Reg++){
67         Data[0]=66;
68         SPIread(CS, Reg, 1, Data);
69         sAUX = "Registro
70             #"+String(Reg)+" =
71             "+String(Data[0]);
72         Serial.println(sAUX);
73     }
74     Serial.println();
75     delay(10000);
76 }
77 #endif
78 #ifndef SPI_PROTOCOL_ARDUINO
79 #ifndef MPU9250
80 //Indispensable para usar "SPI" sobre
81 // el Magnetometro MPU9250
82 MARGwriteByte(MPU9250_ADDR, CS,
83     MPU_INT_PIN_CFG, 0x12);
84 MARGwriteByte(MPU9250_ADDR, CS,
85     MPU_USER_CTRL, 0x30); // I2C
86     Master mode and set I2C_IF_DIS to
87     disable slave mode I2C bus
88 MARGwriteByte(MPU9250_ADDR, CS,
89     MPU_I2C_MST_CTRL, 0x0D); // I2C
90     configuration multi-master IIC
91     400KHz
92
93 MAGwriteByte(MAG_ADDRESS, CS,
94     MAG_CNTL_2, 0x01); /*
95     MARGwriteByte(MPU9250_ADDR, CS,
96     MPU_I2C_SLV0_ADDR, MAG_ADDRESS); //
97     Set the I2C slave address of AK8963 and
98     set for write.
99     MARGwriteByte(MPU9250_ADDR, CS,
100     MPU_I2C_SLV0_REG, MAG_CNTL_2); // I2C
101     slave 0 register address from where to
102     begin data transfer
103     MARGwriteByte(MPU9250_ADDR, CS,
104     MPU_I2C_SLV0_DO, 0x01); // Reset
105     AK8963
106     MARGwriteByte(MPU9250_ADDR, CS,
107     MPU_I2C_SLV0_CTRL, 0x81); // Enable
108     I2C and set 1 byte */
109
110 MAGwriteByte(MAG_ADDRESS, CS,
111     MAG_CNTL_1, 0x12); /*
112     MARGwriteByte(MPU9250_ADDR, CS,
113     MPU_I2C_SLV0_REG, MAG_CNTL_1); // I2C
114     slave 0 register address from where to
115     begin data transfer
116     MARGwriteByte(MPU9250_ADDR, CS,
117     MPU_I2C_SLV0_DO, 0x12); // Register
118     value to 8Hz continuous measurement in
119     16bit

```

```

92 MARGwriteByte(MPU9250_ADDR, CS,
93     MPU_I2C_SLV0_CTRL, 0x81); //Enable
94     I2C and set 1 byte */
95
96 while(true){
97     MARGread(MAG_ADDRESS, CS,
98     MAG_DATA, 7, Mag);
99     /*for(int Reg=0; Reg<=6; Reg++){
100         MARGread(MPU9250_ADDR, CS,
101         Reg, 1, &Data);
102         sAUX = "Registro
103         #"+String(Reg)+" =
104         "+String(Mag[Reg]);
105         Serial.println(sAUX);
106     }*/
107     uint8_t Data[2];
108     for(uint8_t Reg=1; Reg<=125;
109         Reg++){
110         Data[0]=66;
111         SPIread(CS, Reg, 1, Data);
112         sAUX = "Registro
113         #"+String(Reg)+" = "+String(Data[0]);
114         Serial.println(sAUX);
115     }
116     Serial.println();
117     delay(10000);
118 }
119 #endif
120 #endif
121
122 AcGy_Reset();
123 AcGy_setUp();
124 Mag_Reset();
125 Mag_setUp();
126 delay(1);
127 // Configurar magnetometro
128 MARGwriteByte(MPU9250_ADDR, CS,
129     0x37, 0x02); //Registro
130     INT_PIN_CFG
131
132 {
133     /* Procedimiento para obtener el
134     offset Acel-Giro */
135     MARGwriteByte(MPU9250_ADDR, CS,
136     28, ACC_FSCALE_16_G);
137     MARGwriteByte(MPU9250_ADDR, CS,
138     27, GYRO_FSCALE_1000_DPS);
139     MARGwriteByte(MPU9250_ADDR, CS,
140     ACC_GYRO_CONFIG, 0x06);
141     //Gyro_DLPF => 92 Hz
142     MARGwriteByte(MPU9250_ADDR, CS,
143     ACC_CONFIG_1+1, 0x06); //Acc_DLPF
144     => 92 Hz
145     int32_t AX_OS=0, AY_OS=0,
146     AZ_OS=0, GX_OS=0, GY_OS=0,
147     GZ_OS=0;
148     bool Offs_Finish = false;
149     Off_Error = false;
150     uint16_t i_Offs = 0;
151     uint8_t ErrorAX, ErrorAY,
152     ErrorGX, ErrorGY, ErrorGZ;
153     //while(!Off_Error){

```

```

139 // Offs_Finish = false;
140 // while(!Offs_Finish){
141 //     if(flag){
142 //         flag = 0;
143 //         MARGread(MPU9250_ADDR, CS,
144 //             ACC_GYRO_DATA, 14, Acc_Gyro);
145 //         // Convertir
146 //         registros acelerometro
147 //         int16_t ax = (Acc_Gyro[0]
148 //             << 8 | Acc_Gyro[1]);
149 //         int16_t ay = (Acc_Gyro[2]
150 //             << 8 | Acc_Gyro[3]);
151 //         int16_t az = (Acc_Gyro[4]
152 //             << 8 | Acc_Gyro[5]);
153 //         int16_t gx = (Acc_Gyro[8]
154 //             << 8 | Acc_Gyro[9]);
155 //         int16_t gy = (Acc_Gyro[10]
156 //             << 8 | Acc_Gyro[11]);
157 //         int16_t gz = (Acc_Gyro[12]
158 //             << 8 | Acc_Gyro[13]);
159 //         i_Offs++;
160 //         if(i_Offs <= 50){
161 //             AX_OS += ax;    AY_OS +=
162 //             ay;    AZ_OS += az;    GY_OS +=
163 //             gx;    GZ_OS += gz;
164 //         }else{
165 //             int16_t AXOFF = AX_OS /
166 //             200;
167 //             int16_t AYOFF = AY_OS /
168 //             200;
169 //             int16_t AZOFF = AZ_OS /
170 //             200;
171 //             int16_t GXOFF = GX_OS /
172 //             200;
173 //             int16_t GYOFF = GY_OS /
174 //             200;
175 //             int16_t GZOFF = GZ_OS /
176 //             200;
177 //             sAUX = "ACC:
178 //             "+String(AXOFF)+"
179 //             "+String(AYOFF)+" "+String(AZOFF);
180 //             Serial.println(sAUX);
181 //             sAUX = "GYR:
182 //             "+String(GXOFF)+"
183 //             "+String(GYOFF)+" "+String(GZOFF);
184 //             Serial.println(sAUX);
185 //             sAUX = "ACC_GRAV:
186 //             "+String(float(AXOFF)/ACC_SENS)+"
187 //             "+String(float(AYOFF)/ACC_SENS)+"
188 //             "+String(float(AZOFF)/ACC_SENS);
189 //             Serial.println(sAUX);
190 //             sAUX = "GYR_DPS:
191 //             "+String(float(GXOFF)/GYRO_SENS)+"
192 //             "+String(float(GYOFF)/GYRO_SENS)+"
193 //             "+String(float(GZOFF)/GYRO_SENS);
194 //             Serial.println(sAUX);
195 //             if(AZOFF > 0) {AZOFF -=
196 //             ACC_SENS;}
197 //             else {AZOFF += ACC_SENS;}
198 //             Acc_Gyro[0] = (-GXOFF >>
199 //             8) & 0xFF;
200 //             Acc_Gyro[1] = (-GXOFF
201 //             & 0xFF;
202 //             Acc_Gyro[2] = (-GYOFF >>
203 //             8) & 0xFF;
204 //             Acc_Gyro[3] = (-GYOFF
205 //             & 0xFF;
206 //             Acc_Gyro[4] = (-GZOFF >>
207 //             8) & 0xFF;
208 //             Acc_Gyro[5] = (-GZOFF
209 //             & 0xFF;
210 //             //MARGwriteNBytes(MPU9250_ADDR,
211 //             CS, GYRO_OFFSET, 6, Acc_Gyro);
212 //             MARGread(MPU9250_ADDR,
213 //             CS, ACC_OFFSET, 2, Acc_Gyro);
214 //             Acc_Bias[0] =
215 //             (Acc_Gyro[0] << 8 | Acc_Gyro[1]);
216 //             MARGread(MPU9250_ADDR,
217 //             CS, ACC_OFFSET+3, 2, Acc_Gyro);
218 //             Acc_Bias[1] =
219 //             (Acc_Gyro[0] << 8 | Acc_Gyro[1]);
220 //             MARGread(MPU9250_ADDR,
221 //             CS, ACC_OFFSET+6, 2, Acc_Gyro);
222 //             Acc_Bias[2] =
223 //             (Acc_Gyro[0] << 8 | Acc_Gyro[1]);
224 //             sAUX = "AC_BIAS:
225 //             "+String(Acc_Bias[0])+"
226 //             "+String(Acc_Bias[1])+"
227 //             "+String(Acc_Bias[2]);
228 //             Serial.println(sAUX);
229 //             Acc_Bias[0] -= AXOFF;
230 //             Acc_Bias[1] -= AYOFF;
231 //             Acc_Bias[2] -= AZOFF;
232 //             Offs_Finish = true;
233 //             delay(1000);
234 //         }
235 //     }
236 // }
237 // } // Gyr_Bias = {20, 6, 7}.
238 // Error = {-2.20,-0.35,-0.14}.
239 // Gyr_Bias = {106, 20, 8}. Error =
240 // {0,0,0}
241 }
242 Gyr_Bias[0] = 103; Gyr_Bias[1] = 18;
243 Gyr_Bias[2] = 23; // OFFSET
244 // Giroscopio
245 Acc_Gyro[0] = (Gyr_Bias[0] >> 8) &
246 0xFF;
247 Acc_Gyro[1] = (Gyr_Bias[0]) &
248 0xFF;
249 Acc_Gyro[2] = (Gyr_Bias[1] >> 8) &
250 0xFF;
251 Acc_Gyro[3] = (Gyr_Bias[1]) &
252 0xFF;
253 Acc_Gyro[4] = (Gyr_Bias[2] >> 8) &
254 0xFF;
255 Acc_Gyro[5] = (Gyr_Bias[2]) &
256 0xFF;
257 MARGwriteNBytes(MPU9250_ADDR, CS,
258 GYRO_OFFSET, 6, Acc_Gyro);
259 Acc_Bias[0] = 5406; Acc_Bias[1] =
260 -5410; Acc_Bias[2] = 8058;
261 //OFFSET Acelerometro
262 Acc_Gyro[0] = (Acc_Bias[0] >> 8) &
263 0xFF;
264 Acc_Gyro[1] = (Acc_Bias[0]) &
265 0xFF;

```

```

209 MARGwriteNBytes(MPU9250_ADDR, CS,
    ↳ ACC_OFFSET, 2, Acc_Gyro);
210 Acc_Gyro[0] = (Acc_Bias[1] >> 8) &
    ↳ 0xFF;
211 Acc_Gyro[1] = (Acc_Bias[1]) &
    ↳ 0xFF;
212 MARGwriteNBytes(MPU9250_ADDR, CS,
    ↳ ACC_OFFSET + 3, 2, Acc_Gyro);
213 Acc_Gyro[0] = (Acc_Bias[2] >> 8) &
    ↳ 0xFF;
214 Acc_Gyro[1] = (Acc_Bias[2]) &
    ↳ 0xFF;
215 MARGwriteNBytes(MPU9250_ADDR, CS,
    ↳ ACC_OFFSET + 6, 2, Acc_Gyro);
216
217
218 // /* Procedimiento para obtener el
    ↳ ajuste de SENSIBILIDAD
    ↳ Magnetometro */
219 // MARGwriteByte(MAG_ADDRESS, CS,
    ↳ MAG_CNTL_1, 0x00); //Set -> Power
    ↳ Down Mode
220 // delay(10);
221 // MARGwriteByte(MAG_ADDRESS, CS,
    ↳ MAG_CNTL_1, 0x0F); //Set -> Fuse
    ↳ ROM access Mode
222 // delay(10);
223 // MARGread(MAG_ADDRESS, CS,
    ↳ MAG_ASAX, 3, ASA); //Get
    ↳ sensivity
224 // sAUX = "Sensitivity:
    ↳ "+String(ASA[0])+"
    ↳ "+String(ASA[1])+"
    ↳ "+String(ASA[2]);
225 // Serial.println(sAUX);
226 // MARGwriteByte(MAG_ADDRESS, CS,
    ↳ MAG_CNTL_1, 0x00); //Set -> Power
    ↳ Down Mode
227 const uint8_t ASA[3] = {180, 181,
    ↳ 170}; //Magnetometer sensivity
    ↳ adjdument
228 Mag_Adj[0] = ((ASA[0] - 128) / 256.0f
    ↳ + 1);
229 Mag_Adj[1] = ((ASA[1] - 128) / 256.0f
    ↳ + 1);
230 Mag_Adj[2] = ((ASA[2] - 128) / 256.0f
    ↳ + 1);
231
232 {
233 // /* Procedimiento para
    ↳ realizar SELF-TEST Acel-Giro
    ↳ (Ver abajo) */
234 // MARGwriteByte(MPU9250_ADDR,
    ↳ CS, ACC_GYRO_CONFIG, 0x06);
    ↳ //Gyro_DLPF => 92 Hz
235 // MARGwriteByte(MPU9250_ADDR,
    ↳ CS, ACC_CONFIG_1+1, 0x06);
    ↳ //Acc_DLPF => 92 Hz
236 // MARGwriteByte(MPU9250_ADDR,
    ↳ CS, 28, ACC_FSCALE_2 G);
237 // MARGwriteByte(MPU9250_ADDR,
    ↳ CS, 27, GYRO_FSCALE_250 DPS);
238 // int32_t AX_OS=0, AY_OS=0,
    ↳ AZ_OS=0, GX_OS=0, GY_OS=0,
    ↳ GZ_OS=0;
239
    ↳ // int32_t AX_ST_OS=0,
    ↳ AY_ST_OS=0, AZ_ST_OS=0,
    ↳ GX_ST_OS=0, GY_ST_OS=0,
    ↳ GZ_ST_OS=0;
    ↳ uint16_t i_ST = 0;
    ↳ bool ST_Finish = false;
    ↳ ST_active = false;
    ↳ //Norm_active = false;
    ↳ while(!ST_Finish){
    ↳ if(flag == 1){
    ↳ flag = 0;
    ↳ MARGread(MPU9250_ADDR,
    ↳ CS, ACC_GYRO_DATA, 14,
    ↳ Acc_Gyro);
    ↳ // Convertir
    ↳ registros acelerometro
    ↳ int16_t ax = (Acc_Gyro[0]
    ↳ << 8 | Acc_Gyro[1]);
    ↳ int16_t ay = (Acc_Gyro[2]
    ↳ << 8 | Acc_Gyro[3]);
    ↳ int16_t az = Acc_Gyro[4]
    ↳ << 8 | Acc_Gyro[5];
    ↳ int16_t gx = (Acc_Gyro[8]
    ↳ << 8 | Acc_Gyro[9]);
    ↳ int16_t gy =
    ↳ (Acc_Gyro[10] << 8 |
    ↳ Acc_Gyro[11]);
    ↳ int16_t gz = Acc_Gyro[12]
    ↳ << 8 | Acc_Gyro[13];
    ↳ i_ST++;
    ↳ if(i_ST <= 200){
    ↳ AX_OS += ax; AY_OS
    ↳ += ay; AZ_OS += az;
    ↳ GX_OS += gx; GY_OS
    ↳ += gy; GZ_OS += gz;
    ↳ }else if(i_ST <= 400){
    ↳ if(!ST_active){
    ↳ MARGwriteByte(MPU9250_ADDR,
    ↳ CS, 0x1B, 0b11100000); //Set
    ↳ Gyro SELF-TEST en X, Y y Z
    ↳ //
    ↳ MARGwriteByte(MPU9250_ADDR,
    ↳ CS, 0x1C, 0b11100000); //Set
    ↳ Acc SELF-TEST en X, Y y Z
    ↳ ST_active = true;
    ↳ delay(30);
    ↳ }
    ↳ AX_ST_OS += ax;
    ↳ AY_ST_OS += az; AZ_ST_OS
    ↳ += ay;
    ↳ GX_ST_OS += gx;
    ↳ GY_ST_OS += gz; GZ_ST_OS
    ↳ += gy;
    ↳ }else{
    ↳ MARGwriteByte(MPU9250_ADDR,
    ↳ CS, 0x1B, 0b00000000); //Clear
    ↳ Gyro SELF-TEST en X, Y y Z
    ↳ //
    ↳ MARGwriteByte(MPU9250_ADDR,
    ↳ CS, 0x1C, 0b00000000); //Clear
    ↳ Acc SELF-TEST en X, Y y Z
    ↳ delay(30);
    ↳ }
    ↳ }
    ↳ }
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269

```

```

270 //      int32_t AXST =
271 //      (AX_ST_OS - AX_OS) / 200;
272 //      int32_t AYST =
273 //      (AY_ST_OS - AY_OS) / 200;
274 //      int32_t AZST =
275 //      (AZ_ST_OS - AZ_OS) / 200;
276 //      int32_t GXST =
277 //      (GX_ST_OS - GX_OS) / 200;
278 //      int32_t GYST =
279 //      (GY_ST_OS - GY_OS) / 200;
280 //      int32_t GZST =
281 //      (GZ_ST_OS - GZ_OS) / 200;
282 //      sAUX =
283 //      String(AX_ST_OS)+"
284 //      "+String(AY_ST_OS)+"
285 //      "+String(AZ_ST_OS);
286 //      Serial.println(sAUX);
287 //      sAUX = String(AX_OS)+"
288 //      "+String(AY_OS)+"
289 //      "+String(AZ_OS);
290 //      Serial.println(sAUX);
291 //      sAUX = String(AXST)+"
292 //      "+String(AYST)+"
293 //      "+String(AZST);
294 //      Serial.println(sAUX);
295 //      sAUX =
296 //      String(GX_ST_OS)+"
297 //      "+String(GY_ST_OS)+"
298 //      "+String(GZ_ST_OS);
299 //      Serial.println(sAUX);
300 //      sAUX = String(GX_OS)+"
301 //      "+String(GY_OS)+"
302 //      "+String(GZ_OS);
303 //      Serial.println(sAUX);
304 //      sAUX =
305 //      String(GXST)+"
306 //      "+String(GYST)+"
307 //      "+String(GZST);
308 //      Serial.println(sAUX);
309 //      sAUX = String(GXST)+"
310 //      "+String(GYST)+"
311 //      "+String(GZST);
312 //      Serial.println(sAUX);
313 //      sAUX = String(GXST)+"
314 //      "+String(GYST)+"
315 //      "+String(GZST);
316 //      Serial.println(sAUX);
317 //      ST_Finish = true;
318 //      uint8_t G_ST[3],
319 //      A_ST[3];
320 //      MARGread(MPU9250_ADDR,
321 //      CS, ACC_SELF_TEST, 3, A_ST);
322 //      MARGread(MPU9250_ADDR,
323 //      CS, GYRO_SELF_TEST, 3, G_ST);
324 //      int16_t AXST_OTP =
325 //      2620*pow(1.01, (A_ST[0]-1));
326 //      int16_t AYST_OTP =
327 //      2620*pow(1.01, (A_ST[1]-1));
328 //      int16_t AZST_OTP =
329 //      2620*pow(1.01, (A_ST[2]-1));
330 //      int16_t GXST_OTP =
331 //      2620*pow(1.01, (G_ST[0]-1));
332 //      int16_t GYST_OTP =
333 //      2620*pow(1.01, (G_ST[1]-1));
334 //      int16_t GZST_OTP =
335 //      2620*pow(1.01, (G_ST[2]-1));
336 //      sAUX =
337 //      String(AXST_OTP)+"
338 //      "+String(AYST_OTP)+"
339 //      "+String(AZST_OTP);
340 //      Serial.println(sAUX);

```

```

301 //      sAUX =
302 //      String(GXST_OTP)+"
303 //      "+String(GYST_OTP)+"
304 //      "+String(GZST_OTP);
305 //      Serial.println(sAUX);
306 //      sAUX =
307 //      String(float(AXST)/AXST_OTP)+"
308 //      "+String(float(AYST)/AYST_OTP)+"
309 //      "+String(float(AZST)/AZST_OTP);
310 //      Serial.println(sAUX);
311 //      sAUX =
312 //      String(float(GXST)/GXST_OTP)+"
313 //      "+String(float(GYST)/GYST_OTP)+"
314 //      "+String(float(GZST)/GZST_OTP);
315 //      Serial.println(sAUX);
316 //      delay(10000);
317 //      }
318 //      }
319 //      }
320 //      delay(10);
321 //      /* Procedimiento para
322 //      realizar SELF-TEST
323 //      Magnetometro (Ver abajo) */
324 //      MAGwriteByte(MAG_ADDRESS, CS,
325 //      MAG_CNTL_1, 0x10); //Set ->
326 //      Power Down Mode
327 //      MAGwriteByte(MAG_ADDRESS, CS,
328 //      MAG_ASTC, 0x40); //Magnetic
329 //      Fiel for SelfTest starts
330 //      delay(10);
331 //      MAGwriteByte(MAG_ADDRESS, CS,
332 //      MAG_CNTL_1, 0x18); //Set ->
333 //      SELF-TEST Mode
334 //      do{
335 //      MAGread(MAG_ADDRESS, CS,
336 //      MAG_STATUS_1, 1, &ST1);
337 //      } while (! (ST1 & 0x01));
338 //      MAGread(MAG_ADDRESS, CS,
339 //      MAG_DATA, 7, Mag);
340 //      MAGwriteByte(MAG_ADDRESS, CS,
341 //      MAG_ASTC, 0x00);
342 //      MAGwriteByte(MAG_ADDRESS, CS,
343 //      MAG_CNTL_1, 0x10); //Set ->
344 //      Power Down Mode
345 //      delay(50);
346 //      int16_t mx = (Mag[1] << 8 /
347 //      Mag[0]);
348 //      int16_t my = (Mag[3] << 8 /
349 //      Mag[2]);
350 //      int16_t mz = (Mag[5] << 8 /
351 //      Mag[4]);
352 //      sAUX = "NO SCALE SELF-TEST:
353 //      "+String(mx)+" "+String(my)+"
354 //      "+String(mz);
355 //      Serial.println(sAUX);
356 //      MX = Mag_Adj[0]*(float)mx;
357 //      MY = Mag_Adj[1]*(float)my;
358 //      MZ = Mag_Adj[2]*(float)mz;
359 //      sAUX = "SELF-TEST:
360 //      "+String(MX)+" "+String(MY)+"
361 //      "+String(MZ);
362 //      Serial.println(sAUX);
363 }

```



```

337 Acc_Bias[0] = -95; Acc_Bias[1] = -415;
    Acc_Bias[2] = -50; // MAG_FS =
    ↪ 0.6; -80, -400, -167
338 Gyr_Bias[0] = 0; Gyr_Bias[1] = 0;
    ↪ Gyr_Bias[2] = 0; // ACC_FS =
    ↪ 0.59875e-3
339 Mag_Bias[0] = -8; Mag_Bias[1] = -87;
    ↪ Mag_Bias[2] = -14; // GYR_FS =
    ↪ 7.6294e-3; -26, -83, 10
340 Acc_Adj[0] = 1; Acc_Adj[1] = 1;
    ↪ Acc_Adj[2] = 1;
341 Gyr_Adj[0] = 1; Gyr_Adj[1] = 1;
    ↪ Gyr_Adj[2] = 1;
342 Mag_Adj[0] *= 0.97; Mag_Adj[1] *=
    0.94; Mag_Adj[2] *= 0.97; //
    ↪ Mag_Adj : ASA-->{1.20313, 1.20703,
    ↪ 1.1641}
343
344 // uint8_t Data;
345 // for(uint8_t Reg=0; Reg<=19;
346 // Reg++){
347 //     MAGread(MAG_ADDRESS, CS, Reg,
    ↪ 1, &Data);
348 //     sAUX = "Registro
    ↪ #"+String(Reg)+" =
    ↪ "+String(Data);
349 //     Serial.println(sAUX);
350 // }
351 }
352
353
354
355
356
357
358 void Mpu9250::loop(loopFlag loopType){
359     t_ant = t_act;
360     t_act = micros();
361     t_delta = float(t_act - t_ant) / 1e6;
362     /* LECTURA DE VALORES SENSORES*/
363     //SPI.setClockDivider(SPI_CLOCK_DIV2);
364     MARGread(MPU9250_ADDR, CS,
    ↪ ACC_GYRO_DATA, 14, Acc_Gyro);
365
    ↪ //SPI.setClockDivider(SPI_CLOCK_DIV16);
366     // Convertir registros acelerometro
367     int16_t ax = (Acc_Gyro[0] << 8 |
    ↪ Acc_Gyro[1]);
368     int16_t ay = (Acc_Gyro[2] << 8 |
    ↪ Acc_Gyro[3]);
369     int16_t az = (Acc_Gyro[4] << 8 |
    ↪ Acc_Gyro[5]);
370     AX = float(ax + Acc_Bias[0]) *
    ↪ Acc_Adj[0] * ACC_FS;
371     AY = float(ay + Acc_Bias[1]) *
    ↪ Acc_Adj[1] * ACC_FS;
372     AZ = float(az + Acc_Bias[2]) *
    ↪ Acc_Adj[2] * ACC_FS;
373     Ac_mean.setNewValues(AX, AY, AZ);
374     AX = Ac_mean.getMeanX(); AY =
    ↪ Ac_mean.getMeanY(); AZ =
    ↪ Ac_mean.getMeanZ();
375     AR_ant = AR; AR = sqrt(pow(AX, 2) +
    ↪ pow(AY, 2) + pow(AZ, 2));
376     delta_AR = AR - AR_ant;
377     // Convertir registros giroscopio
378     int16_t gx = (Acc_Gyro[8] << 8 |
    ↪ Acc_Gyro[9]);
379     int16_t gy = (Acc_Gyro[10] << 8 |
    ↪ Acc_Gyro[11]);
380     int16_t gz = (Acc_Gyro[12] << 8 |
    ↪ Acc_Gyro[13]);
381     GX = float(gx + Gyr_Bias[0]) *
    ↪ Gyr_Adj[0] * GYR_FS;
382     GY = float(gy + Gyr_Bias[1]) *
    ↪ Gyr_Adj[1] * GYR_FS;
383     GZ = float(gz + Gyr_Bias[2]) *
    ↪ Gyr_Adj[2] * GYR_FS;
384     Gy_mean.setNewValues(GX, GY, GZ);
385     GX = Gy_mean.getMeanX(); GY =
    ↪ Gy_mean.getMeanY(); GZ =
    ↪ Gy_mean.getMeanZ();
386     GR_ant = GR; GR = sqrt(pow(GX, 2) +
    ↪ pow(GY, 2) + pow(GZ, 2));
    delta_GR = GR - GR_ant;
387
388     // --- Lectura del magnetometro ---
389     uint32_t t_MgStat = millis();
390     do {
391         MAGread(MAG_ADDRESS, CS,
    ↪ MAG_STATUS_1, 1, &ST1);
392         if(millis() - t_MgStat >
    ↪ 1000){
393             Mag_Reset();
394             Mag_setup();
395             t_MgStat = millis();
396             ST1 = 1;
397         }
398     } while (!(ST1 & 0x01));
399     MAGread(MAG_ADDRESS, CS, MAG_DATA, 7,
    ↪ Mag);
400     // Convertir registros magnetometro
401     int16_t mx = (Mag[3] << 8 | Mag[2]);
402     int16_t my = (Mag[1] << 8 | Mag[0]);
403     int16_t mz = (Mag[5] << 8 | Mag[4]);
404     MX = float(mx + Mag_Bias[1]) *
    ↪ Mag_Adj[1] * MAG_FS;
405     MY = float(my + Mag_Bias[0]) *
    ↪ Mag_Adj[0] * MAG_FS;
406     MZ = float(-mz + Mag_Bias[2]) *
    ↪ Mag_Adj[2] * MAG_FS;
407     Ma_mean.setNewValues(MX, MY, MZ); //
    ↪ MX -= 15.0f
408     MX = Ma_mean.getMeanX(); MY =
    ↪ Ma_mean.getMeanY(); MZ =
    ↪ Ma_mean.getMeanZ();
409     // OFFSET: X = -38 a 41 uT Y = -24 a
    ↪ 55 uT Z = -44 a 39 uT
410     MR_ant = MR; MR = sqrt(pow(MX, 2) +
    ↪ pow(MY, 2) + pow(MZ, 2));
411     delta_MR = MR - MR_ant;
412
413     if ((ax_ant == ax) || (ay_ant == ay)
    ↪ || (az_ant == az)) {
414         ++aCSSV;
415         qDebug() << " aCSSV: " << aCSSV;
416         if (aCSSV >= 4) {
417             AcGy_Reset();
418             delay(1);
419             int set = AcGy_setup();
420             if(set < 0){
421

```

```

423         fprintf(stderr, "
                                .....Error %d:
                                ↳ AcGy_Setup().....
                                ↳ \n", set);
424     }
425     aCSSV = 0;
426 }
427 } else {
428     aCSSV = 0;
429 }
430
431 if ((mx_ant == mx) || (my_ant == my)
    ↳ || (mz_ant == mz)) {
432     ++mCSSV;
433     qDebug() << "    mCSSV: " << mCSSV;
434     if (mCSSV == 4) {
435         Mag_Reset();
436         delay(1);
437         Mag_setUp();
438     } else if (mCSSV >= 7) {
439         Mag_Reset();
440         AcGy_Reset();
441         delay(1);
442         int set = AcGy_setUp();
443         if (set < 0) {
444             fprintf(stderr, "
                                    .....Error %d:
                                    ↳ AcGy_Setup().....
                                    ↳ \n", set);
445         }
446         set = Mag_setUp();
447         if (set < 0) {
448             fprintf(stderr, "
                                    .....Error %d:
                                    ↳ Mag_Setup().....
                                    ↳ \n", set);
449         }
450         delay(1);
451         mCSSV = 0;
452     }
453 } else {
454     mCSSV = 0;
455 }
456 ax_ant = ax; ay_ant = ay; az_ant =
    ↳ az;
457 mx_ant = mx; my_ant = my; mz_ant =
    ↳ mz;
458
459 /* IMPRESIÓN DE VALORES POR PANTALLA
    ↳ */
460 /*
461 // Acelerometro
462 QString sAUX;
463 sAUX = "Ac=[" + String(AX) + " " +
    ↳ String(AY) + " " + String(AZ) + "]" ";
464 QByteArray dataSnd = sAUX.toUtf8();
465 Serial.print(sAUX);
466 // sAUX = " Magnitud:
    ↳ "+String(AR)+" Diff:
    ↳ "+String(100*delta_AR/t_delta);
467 // Serial.println(sAUX);
468 // Giroscopio
469
470 sAUX = "Gy=[" + String(GX) + " " +
    ↳ String(GY) + " " + String(GZ) + "]" ";
471 dataSnd.append(sAUX.toUtf8());
472 Serial.print(sAUX);

```

```

473 // sAUX = " Magnitud:
      "+String(GR)+" Diff:
      "+String(100*delta_GR/t_delta);
474 // Serial.println(sAUX);
475 // Magnetometro
476 sAUX = "Mg=[" + String(MX) + " " +
      String(MY) + " " + String(MZ) + "]" ";
477 dataSnd.append(sAUX.toUtf8());
478 Serial.print(sAUX);
479 // sAUX = " Magnitud:
      "+String(MR)+" Diff:
      "+String(100*delta_MR/t_delta);
480 // Serial.println(sAUX);
481
482 //sAUX = "Tiempo 4: "+String(micros()
      - t_p); Serial.println(sAUX);
483 Serial.print(t_act);
484 Serial.println();
485 */
486
487 // uint8_t Data;
488 // for(uint8_t Reg=0; Reg<=19;
      Reg++){
489 // MAGread(MAG_ADDRESS, CS, Reg, 1,
      &Data);
490 // sAUX = "Registro
      #"+String(Reg)+" =
      "+String(Data);
491 // Serial.println(sAUX);
492 // }
493
494
495
496
497
498
499
500
501
502
503
504
505
506 /* COMIENZA MATLAB
      */
507
508 // Ejecutar bucle infinito
509 /// COMIENZA WHILE(TRUE)
510 time_sensor =
      static_cast<float>(micros()) /
      1000000.0f;
511 t1 = t2; t2 = time_sensor; deltat
      = t2 - t1;
512 Vector Ac = {AX, AY, AZ}; Vector Gy =
      {GX, GY, GZ};
513 Vector Mg = {MX, MY, MZ};
514 Vector Ac_Raw = Ac, Mg_Raw = Mg,
      Gy_Raw = Gy;
515 Ac = (Ac - Ac_Adj_Offs) * Ac_Adj_Sens;
516 Mg = (Mg - Mg_Adj_Offs) * Mg_Adj_Sens;
517 Gy = Gy - Gy_Adj;
518
519 // bool isMagAnormal = (20 > norm(Mg)
      || norm(Mg) > 150) && (delta_MR > 15);
520 // bool isAcAnormal = (8.6 > norm(Mg)
      || norm(Mg) > 11) && (delta_MR > 2.5);
521 // if(isMagAnormal || isAcAnormal){
522 // fprintf(stderr, " Reset-SetUp
      por comportamiento anormal: \n");
523 // Mag_Reset();
524 // AcGy_Reset();

```

```

525 //      delay(1);
526 //      if(AcGy_setup() < 0){
527 //          fprintf(stderr, "
↳ .....Error: AcGy_Setup().....
↳ \n");
528 //      }
529 //      if(Mag_setup() < 0){
530 //          fprintf(stderr, "
↳ .....Error: Mag_Setup().....
↳ \n");
531 //      }
532 //      delay(5);
533 //
534 //  }
535
536
537 // OBTENCIÓN DE LA ORIENTACIÓN POR
↳ MADGWICK
538 Vector halfSEq(4), twoSEq(4);
539 halfSEq(0) = 0.5f * SEq(0);
540 halfSEq(1) = 0.5f * SEq(1);
541 halfSEq(2) = 0.5f * SEq(2);
542 halfSEq(3) = 0.5f * SEq(3);
543 twoSEq(0) = 2.0f * SEq(0);
544 twoSEq(1) = 2.0f * SEq(1);
545 twoSEq(2) = 2.0f * SEq(2);
546 twoSEq(3) = 2.0f * SEq(3);
547 // float twob_x = 2.0f * b_x;
548 // float twob_z = 2.0f * b_z;
549 Vector twob_xSEq(4), twob_zSEq(4);
550 twob_xSEq(0) = 2.0f * b_x * SEq(0);
551 twob_xSEq(1) = 2.0f * b_x * SEq(1);
552 twob_xSEq(2) = 2.0f * b_x * SEq(2);
553 twob_xSEq(3) = 2.0f * b_x * SEq(3);
554 twob_zSEq(0) = 2.0f * b_z * SEq(0);
555 twob_zSEq(1) = 2.0f * b_z * SEq(1);
556 twob_zSEq(2) = 2.0f * b_z * SEq(2);
557 twob_zSEq(3) = 2.0f * b_z * SEq(3);
558 //SEq_1SEq_2;
559 float SEq_1SEq_3 = SEq(0) * SEq(2);
560 //SEq_1SEq_4;
561 //SEq_2SEq_3;
562 float SEq_2SEq_4 = SEq(1) * SEq(3);
563 //SEq_3SEq_4;
564 Vector two_m(3);
565 two_m(0) = 2.0f * Mg(0);
566 two_m(1) = 2.0f * Mg(1);
567 two_m(2) = 2.0f * Mg(2);
568
569 // magnalise the accelerometer
↳ measurement
570 float magn;
571 magn = norm(Ac);
572 Vector Ac_one = Ac/magn;
573 // magnalise the magnetometer
↳ measurement
574 magn = norm(Mg);
575 Vector Mg_one = Mg/magn;
576 Gy = Gy*(PI/180.0f); //Conversion
↳ entre (Deg/seg) a (Rad/seg)
577 // fprintf(stderr, "i=%d Ac=[%.3f %.3f
↳ %.3f] Gy=[%.3f %.3f %.3f] Mg=[%.3f
↳ %.3f %.3f]\n",
↳ i+1,AX,AY,AZ,GX,GY,GZ,MX,MY,MZ);
578 // fprintf(stderr, "i=%d Ac=[%.3f %.3f
↳ %.3f] Gy=[%.3f %.3f %.3f]
↳ Mg=[%.3f %.3f %.3f]\n",
↳ i+1,AX,AY,AZ,GX,GY,GZ,MX,MY,MZ);
579

```

```

580
↳ i+1,Ac(0),Ac(1),Ac(2),Gy(0),Gy(1),Gy(2),Mg(0),Mg(1),Mg(2);
581 // fprintf(stderr, "i=%d Ac=[%.3f %.3f
↳ %.3f] Gy=[%.3f %.3f %.3f] Mg=[%.3f
↳ %.3f %.3f]\n",
↳ i+1,Ac(0),Ac(1),Ac(2),Gy(0),Gy(1),Gy(2),Mg(0),Mg(1),Mg(2);
582 //
↳ fprintf(stderr, "Tiempo = %f \n",
↳ t2);
583
584 // compute the objective function and
↳ Jacobian
585 Vector f1(3), f2(3);
586 f1 = SErotM * g_E - Ac_one;
587 f2 = SErotM * b_E - Mg_one;
588 f = append(f1, f2);
589 //f(1:3,1) = [0 0 0];
590 // compute the gradient (matrix
↳ multiplication)
591 Matriz J(6,4);
592 J(2,0) = 0; J(2,3) = 0;
593 J(1,3) = twoSEq(2); J(0,0) =
↳ -J(1,3); // J_11 negated in matrix
↳ multiplication
594 J(0,1) = twoSEq(3); J(1,2) = J(0,1);
595 J(1,1) = twoSEq(0); J(0,2) = -J(1,1);
596 // J_12 negated in matrix
↳ multiplication
597 J(0,3) = twoSEq(1); J(1,0) = J(0,3);
598 J(2,1) = -2.0f * twoSEq(1); // negated
↳ in matrix multiplication
599 J(2,2) = -2.0f * twoSEq(2); // negated
↳ in matrix multiplication
600 J(3,0) = -twob_zSEq(2); // negated in
↳ matrix multiplication
601 J(3,1) = twob_zSEq(3);
602 J(3,2) = -2.0f * twob_xSEq(2) -
↳ twob_zSEq(0); // negated in matrix
↳ multiplication
603 J(3,3) = -2.0f * twob_xSEq(3) +
↳ twob_zSEq(1); // negated in matrix
↳ multiplication
604 J(4,0) = -twob_xSEq(3) + twob_zSEq(1);
↳ // negated in matrix
↳ multiplication
605 J(4,1) = twob_xSEq(2) + twob_zSEq(0);
606 J(4,2) = twob_xSEq(1) + twob_zSEq(3);
607 J(4,3) = -twob_xSEq(0) + twob_zSEq(2);
↳ // negated in matrix
↳ multiplication
608 J(5,0) = twob_xSEq(2);
609 J(5,1) = twob_xSEq(3) - 2.0f *
↳ twob_zSEq(1);
610 J(5,2) = twob_xSEq(0) - 2.0f *
↳ twob_zSEq(2);
611 J(5,3) = twob_xSEq(1);
612 SEqHatDot = f*J;
613 // magnalise the gradient to estimate
↳ direction of the gyroscope error
614 SEqHatDot = SEqHatDot /
↳ norm(SEqHatDot);
615 // compute angular estimated direction
↳ of the gyroscope error
616

```



```

617 Vector err = 2.0f *
    quatmultiply(quatconj(SEq),
    ↪ SEqHatDot);
618 Vector w_err(3); w_err(0) = err(1);
    w_err(1) = err(2); w_err(2) =
    ↪ err(3);
619 // compute and remove the gyroscope
    ↪ baíses
620 w_bx = w_bx + w_err(0) * deltat *
    ↪ zeta;
621 w_by = w_by + w_err(1) * deltat *
    ↪ zeta;
622 w_bz = w_bz + w_err(2) * deltat *
    ↪ zeta;
623 w_bx = 0.0f; w_by = 0.0f; w_bz =
    ↪ 0.0f;
624 Gy(0) = Gy(0) - w_bx;
625 Gy(1) = Gy(1) - w_by;
626 Gy(2) = Gy(2) - w_bz;
627 // compute the quaternion rate
    ↪ measured by gyroscopes
628 Vector Gy0(4); Gy0(0) = 0.0f; Gy0(1)
    = Gy(0); Gy0(2) = Gy(1); Gy0(3)
    ↪ = Gy(2);
629 Vector SEqDot_omega = 0.5f *
    ↪ quatmultiply(SEq, Gy0);
630 //SEqDot_omega = [0 0 0 0];
631
632 // compute then integrate the
    ↪ estimated quaternion rate
633 SEq(0) = SEq(0) + (SEqDot_omega(0) -
    ↪ (beta * SEqHatDot(0))) * deltat;
634 SEq(1) = SEq(1) + (SEqDot_omega(1) -
    ↪ (beta * SEqHatDot(1))) * deltat;
635 SEq(2) = SEq(2) + (SEqDot_omega(2) -
    ↪ (beta * SEqHatDot(2))) * deltat;
636 SEq(3) = SEq(3) + (SEqDot_omega(3) -
    ↪ (beta * SEqHatDot(3))) * deltat;
637 // magnalise quaternion
638 magn = norm(SEq);
639 SEq(0) = SEq(0) / magn;
640 SEq(1) = SEq(1) / magn;
641 SEq(2) = SEq(2) / magn;
642 SEq(3) = SEq(3) / magn;
643 // compute flux in the earth frame
644 float SEq_1SEq_2 = SEq(0) * SEq(1); //
    ↪ recompute auxiliary variables
645 SEq_1SEq_3 = SEq(0) * SEq(2);
646 float SEq_1SEq_4 = SEq(0) * SEq(3);
647 float SEq_3SEq_4 = SEq(2) * SEq(3);
648 float SEq_2SEq_3 = SEq(1) * SEq(2);
649 SEq_2SEq_4 = SEq(1) * SEq(3);
650
651 SErotM = Quat2RotMat(SEq);
652 Vector h_S = (SErotM * two_m);
653 SErotM = SErotM.t();
654
655 // magnalise the flux vector to have
    ↪ only components in the x and z
656 // b_x = sqrt((h_S(1) * h_S(1)) +
    ↪ (h_S(2) * h_S(2)));
657 // b_z = h_S(3);
658 //fprintf('----bx=%.3f bz=%.3f
    ↪ w_bx=%.3f w_by=%.3f w_bz=%.3f \n',
    ↪ b_x, b_z, w_bx, w_by, w_bz);

```

```

661 // CALIBRACIÓN ELÍPTICA AUTOMÁTICA
662 float btw_angle =
663     dot(Ac, Ac_capt(i_capt)) / (norm(Ac) * norm(Ac_capt(i_capt)));
    ↪ // Angle Between measurements
664 bool if_btw_angAc = btw_angle <
    ↪ 0.9659; // btw_angle < que
    ↪ 15°
665 btw_angle =
    dot(Mg, Mg_capt(i_capt)) / (norm(Mg) * norm(Mg_capt(i_capt)));
    ↪ // Angle Between measurements
666 bool if_btw_angMg = btw_angle <
    ↪ 0.9659; // btw_angle < que
    ↪ 15°
667 d_Ac(i % nCalib) = norm(Ac - Ac_ant)
    ↪ / norm(Ac_ant);
668 d_Mg(i % nCalib) = norm(Mg - Mg_ant)
    ↪ / norm(Mg_ant);
669 bool No_Move_Ac_Mg =
    (sum(d_Ac) / nCalib < 0.05) &&
    ↪ (sum(d_Mg) / nCalib < 0.05);
670 //fprintf(stderr, 'Mg_ant= [%3f %3f %3f] \n',
    ↪ Mg, Mg_ant);
671 //fprintf('Ac_ant= [%3f %3f %3f]
    ↪ Ac= [%3f %3f %3f] \n', Ac,
    ↪ Ac_ant);
672 //fprintf('btw_angle= %.3f d_Ac= %.3f
    ↪ d_Mg= %.3f \n', btw_angle,
    ↪ sum(d_Ac) / nCalib,
    ↪ sum(d_Mg) / nCalib);
673 //NoMove_Gy = sum(Gy.^2) < 4; // Gy <
    ↪ 4 [°/seg]
674
675 // Proceso de validación y posible
    ↪ captura de datos
676 Ac_ant = Ac;
677 Gy_ant = Gy;
678 Mg_ant = Mg;
679 //bol = if_btw_ang & No_Move_Ac_Mg
680 if (No_Move_Ac_Mg){ //if true,
    ↪ entonces captura datos
681     if (if_btw_angMg && if_btw_angAc
        ↪ && (aCSSV == 0 && mCSSV ==
        ↪ 0)){
682         i_capt = j % nCapt;
683         Ac_capt(i_capt) = Ac;
684         Gy_capt(i_capt) = Gy;
685         Mg_capt(i_capt) = Mg;
686         if (i_capt == nCapt){
687             Vector Ac_Adj_new =
                ↪ RegresilLinealMultip(Ac_capt);
                ↪ Vector Mg_Adj_new =
                ↪ RegresilLinealMultip(Mg_capt);
                ↪ Vector Gy_Adj_new =
                ↪ sum(Gy_capt, 1);
                ↪ Gy_Adj_new =
                ↪ Gy_Adj_new / nCapt;
690         if (Ac_Adj_new(0) !=
            ↪ -1.0f){ // Si es
            ↪ diferente de -1
                ↪ Vector Ac_TamH_new =
                ↪ Ac_Adj_new;

```

```

693 // Proceso de aprobación/Rechazo de los nuevos parámetros de ajuste
694 Vector Ac_AdjError = 1.0f - sum( pow( (Ac_capt-Ac_Adj_new(3,5))/ Ac_Adj_new(0,2) ), 2 ); //
695 Vector Ac_AdjRel = sqrt(1.0f/(1.0f-Ac_AdjError)); // Rel = (1/(1-Error))~0.5
696 Vector if_Ac_inv = (0 < Ac_AdjRel) && (Ac_AdjRel < 1);
697 Ac_AdjRel = (!if_Ac_inv)*Ac_AdjRel + if_Ac_inv/Ac_AdjRel;
698 float Ac_Adj_TotalErr = sum(Ac_AdjRel) / Ac_capt.getDimV();
699 fprintf(stderr, "AcTol_Err = %.3f\n", Ac_Adj_TotalErr);
700 if (Ac_Adj_TotalErr < Ac_Tol){
701     Ac_Tol = 0.7*(Ac_Tol) + 0.3*(1.08); // = Ac_Tol - 0.4*(Ac_Tol-1.08)
702 // If TRUE, entonces son parámetros validos, y los actualizamos
703 Vector Ac_Adj_new_Sens = Ac_Adj_new(0)/Ac_Adj_new(0,2);
704 Vector Ac_Adj_new_Offs = Ac_Adj(3,5) + Ac_Adj_new(3,5)/Ac_Adj(0,2);
705 Ac_Adj_new_Sens = Ac_Adj(0,2)*Ac_Adj_new(0,2);
706 // mod(k,10)+1;
707 // Ac_Adj_new(4:6) = Ac_Adj(4:6) + Ac_Adj_new(4:6).*Ac_Adj(1:3);
708 // Ac_Adj_new(1:3) = Ac_Adj(1:3).*Ac_Adj_new(1:3);
709 // Mg_Adj_new(4:6) = Mg_Adj(4:6) + Mg_Adj_new(4:6).*Mg_Adj(1:3);
710 // Mg_Adj_new(1:3) = Mg_Adj(1:3).*Mg_Adj_new(1:3);
711 Ac_Adj_Store(i_adj,:) = Ac_Adj; //Ac_Adjust_Storage
712 Mg_Adj_Store(i_adj,:) = Mg_Adj; // Se mezclan los viejos y los nuevos parámetros con 60% y 40% c/u
713 Ac_Adj_new = append(Ac_Adj_new_Sens, Ac_Adj_new_Offs);
714 Ac_Adj = 0.6*Ac_Adj + 0.4*Ac_Adj_new;
715 Vector VPrint = Ac_TamH_new(0)/Ac_Adj(0,2);
716 fprintf(stderr, "Ac_Adj =");
717 fprintf(stderr, " %.3f\n", VPrint[0]);
718 fprintf(stderr, " %.3f\n", VPrint[1]);
719 fprintf(stderr, " %.3f\n", VPrint[2]);
720 fprintf(stderr, " %.3f\n", Ac_Adj[3]);
721 fprintf(stderr, " %.3f\n", Ac_Adj[4]);
722 fprintf(stderr, " %.3f\n", Ac_Adj[5]);
723 pause(5);
724 }else{
725     fprintf(stderr, "ACC: Test de TOLERANCIA de ajuste REPROBADO\n");
726     Ac_Tol = Ac_Tol * 1.08;
727     pause(2);
728 }
729 }else{
730     fprintf(stderr, "ACCEL: AMPL Y OFFS NEGATIVOS\n");
731     pause(2);
732     return;
733 }
734 if (Mg_Adj_new(0) != -1.0f){ // Si es diferente de -1
735     Vector Mg_TamH_new = Mg_Adj_new;

```

```

734 // Proceso de
       aprobación/Rechazo
       de los nuevos
       parámetros de
       ajuste
735 Vector Mg_AdjError =
       1.0f - sum( pow(
       (Mg_capt-Mg_Adj_new(3,5))
       / Mg_Adj_new(0,2)
       ), 2);
736 Vector Mg_AdjRel =
       sqrt(1.0f/(1.0f-Mg_AdjError));
737 Vector if_Mg_inv =
       (0 <
       Mg_AdjRel) &&
       (Mg_AdjRel <
       1);
738 Mg_AdjRel =
       (!if_Mg_inv)*Mg_AdjRel
       + if_Mg_inv/Mg_AdjRel;
739 float Mg_Adj_TotalErr
       = sum(Mg_AdjRel) /
       Mg_capt.getDimV();
740 fprintf(stderr,
       "MgTol_Err =
       %.3f\n\n",Mg_Adj_TotalErr);
741 if (Mg_Adj_TotalErr <
       Mg_Tol){
742     Mg_Tol =
       0.7*(Mg_Tol) +
       0.3*(1.08);
       // = Mg_Tol -
       0.4*(Mg_Tol-1.08)
743 // If TRUE,
       enonces son
       parámetros
       validos, y los
       actualizamos
744 Vector
       Mg_Adj_new_Sens
       =
       Mg_Adj_new(0)/Mg_Adj_new(0,2);
745 Vector
       Mg_Adj_new_Offs
       = Mg_Adj(3,5)
       +
       Mg_Adj_new(3,5)/Mg_Adj(0,2);
746 Mg_Adj_new_Sens =
       Mg_Adj(0,2)*Mg_Adj_new(0,2);
747 // Se mezclan los
       viejos y los
       nuevos
       parametros con
       60% y 40% c/u
749 Mg_Adj_new =
       append(Mg_Adj_new_Sens,
       Mg_Adj_new_Offs);
750 Mg_Adj =
       0.6*Mg_Adj +
       0.4*Mg_Adj_new;
751 Vector VPrint =
       Mg_TamH_new(0)/Mg_Adj(0,2);
       }

752
       fprintf(stderr,
       "Mg_Adj =");
       fprintf(stderr,
       " %.3f
       %.3f %.3f",
       VPrint[0],
       VPrint[1],
       VPrint[2]);

       fprintf(stderr,
       " %.3f
       %.3f
       %.3f \n",
       Mg_Adj[3],
       Mg_Adj[4],
       Mg_Adj[5]);
       pause(5);
       }else{
       fprintf(stderr,
       "MAG: Test de
       TOLERANCIA de
       ajuste
       REPROBADO\n");
       Mg_Tol = Mg_Tol *
       1.08;
       pause(2);
       }
       }else{
       fprintf(stderr,
       "MAGNET: AMPL Y
       OFFS
       NEGATIVOS\n");
       pause(2);
       return;
       }
       }
       j = j + 1;
       fprintf(stderr, "%d VALOR(ES)
       CAPTURADO(S) \n",j);
       }
       }

       // Algoritmo de alineación en pose-T}
       float Align_Err = norm(f);
       fprintf(stderr, " Align_Err = %.3f
       \n", Align_Err);
       if(!isAligned){
       if(loopType == Align){
       float Align_Tol = 0.200;
       if(aCSSV == 0 && mCSSV == 0 &&
       Align_Err < Align_Tol){
       fprintf(stderr, "Sensor
       alineado\n");
       delay(5000);
       MatrizCuadrada
       Mpu_init_inv = SErotM;
       // Mpu_init^-1 =
       SErotM (Previamente
       invertida con "t()")
       Align_Ajd = Mpu_init_inv;
       //Euler2RotMat(UE_init)
       *
       isAligned = true;
       }
       }
       }

```

```

789     }
790
791     /// COMUNICACIÓN UDP MATLAB - UNREAL
792     /// Vector SEuler =
793         ↪ RotMat2Euler(SEuler.t());
794         ↪ SEuler = (SEuler <
795         ↪ 0.0f)*360.0f + SEuler;
796         ↪ SEuler(1) = 360 - SEuler(1);
797         ↪ SEuler(2) = 360 - SEuler(2);
798
799     /// QString datSEND =
800         ↪ String(SEuler(0))+","+String(SEuler(1))+","+String(SEuler(2));
801         ↪ QByteArray prueba =
802         ↪ QByteArray(datSEND.toAscii());
803         ↪ fprintf(stderr, "%s\n",
804         ↪ prueba.data());
805
806     i = i+1;
807 }
808
809 void Mpu9250::AcGy_Reset(void){
810     MARGwriteByte(MPU9250_ADDR, CS,
811     ↪ ACC_GYRO_RESET, 0x80); ///Reset
812     ↪ Accel-Gyro Registers
813 }
814
815 int Mpu9250::AcGy_setUp(void){
816     ///Indispensable para usar "SPI" sobre
817         ↪ el Magnetómetro MPU9250
818     MARGwriteByte(MPU9250_ADDR, CS,
819     ↪ MPU_USER_CTRL, 0x30); ///I2C
820     ↪ Master mode and set I2C_IF_DIS to
821     ↪ disable slave mode I2C bus
822     MARGwriteByte(MPU9250_ADDR, CS,
823     ↪ MPU_INT_PIN_CFG, 0x12);
824     MARGwriteByte(MPU9250_ADDR, CS,
825     ↪ MPU_I2C_MST_CTRL, 0x0D); ///I2C
826     ↪ configuration multi-master IIC
827     ↪ 400KHz
828
829     ///Configurar acelerometro
830     MARGwriteByte(MPU9250_ADDR, CS, 28,
831     ↪ ACC_FSCALE_2_G);
832     MARGwriteByte(MPU9250_ADDR, CS,
833     ↪ ACC_CONFIG_1+1, 0x06); ///Acc_DLPF
834     ↪ => 5 Hz
835     ///Configurar giroscopio
836     MARGwriteByte(MPU9250_ADDR, CS, 27,
837     ↪ GYRO_FSCALE_250_DPS);
838
839     MARGwriteByte(MPU9250_ADDR, CS,
840     ↪ ACC_GYRO_CONFIG, 0x02);
841     ↪ ///Gyro_DLPF Bandwidth ==> 92 Hz
842
843     uint8_t Data;
844     MARGread(MPU9250_ADDR, CS,
845     ↪ MPU_USER_CTRL, 1, &Data);
846     if(Data != 0x30) return -1;
847     MARGread(MPU9250_ADDR, CS,
848     ↪ MPU_INT_PIN_CFG, 1, &Data);
849     if(Data != 0x12) return -2;
850     MARGread(MPU9250_ADDR, CS,
851     ↪ MPU_I2C_MST_CTRL, 1, &Data);
852     if(Data != 0x0D) return -3;
853
854     Serial.println("
855     ↪ .....AcGy_SetUP()
856     ↪ Executed.....");
857     return 0;
858 }
859
860 void Mpu9250::Mag_Reset(void){
861     MARGwriteByte(MAG_ADDRESS, CS,
862     ↪ MAG_CNTL_1+1, 0x01); ///Soft
863     ↪ Reset Magnetometer Registers
864     ///Configurar magnetometro
865     ///MARGwriteByte(MPU9250_ADDR, CS,
866         ↪ 0x37, 0x02); //Registro
867     ↪ INT_PIN_CFG
868 }
869
870 int Mpu9250::Mag_setUp(void){
871     ///Una vez sea posible establecer
872         ↪ comunicación con el Magnetómetro,
873     ↪ entonces:
874     MARGwriteByte(MAG_ADDRESS, CS,
875     ↪ MAG_CNTL_1, MAG_MODE_1 |
876     ↪ MAG_FSCALE_16_bit);
877
878     uint8_t Data;
879     MARGread(MAG_ADDRESS, CS, MAG_CNTL_1,
880     ↪ 1, &Data);
881     if(Data != MAG_MODE_1 |
882     ↪ MAG_FSCALE_16_bit) return -1;
883
884     Serial.println("
885     ↪ .....Mag_SetUP()
886     ↪ Executed.....");
887     return 0;
888 }

```

## B.15. mpu9250.h

```

1  #ifndef MPU9250_H
2  #define MPU9250_H
3
4  #include <QString>
5  #include "matlab.h"
6  #include "Complementos.h"
7
8
9
10 #if defined(I2C_PROTOCOL)
11     #undef SPI_PROTOCOL
12     #include <Wire.h>
13     #define
14         MARGread(Addr, CS, Reg, nBytes, Data)
15         ↪ I2Cread(Addr, Reg, nBytes, Data)
16     #define MARGwriteByte(Addr, CS, Reg, Data)
17         ↪ I2CwriteByte(Addr, Reg, Data)
18     #define
19         MARGwriteNBytes(Addr, CS, Reg, nBytes, Data)
20         ↪ I2CwriteNBytes(Addr, Reg, nBytes, Data)
21     #define MAGread(Addr, CS, Reg, nBytes, Data)
22         ↪ I2Cread(Addr, Reg, nBytes, Data)
23     #define MAGwriteByte(Addr, CS, Reg, Data)
24         ↪ I2CwriteByte(Addr, Reg, Data)
25     #define
26         MAGwriteNBytes(Addr, CS, Reg, nBytes, Data)
27         ↪ I2CwriteNBytes(Addr, Reg, nBytes, Data)
28 #elif defined(SPI_PROTOCOL)
29     #undef I2C_PROTOCOL
30     #include <SPI.h>
31     #define
32         MARGread(Addr, CS, Reg, nBytes, Data)
33         ↪ SPIread(CS, Reg, nBytes, Data)
34     #define MARGwriteByte(Addr, CS, Reg, Data)
35         ↪ SPIwriteByte(CS, Reg, Data)
36     #define
37         MARGwriteNBytes(Addr, CS, Reg, nBytes, Data)
38         ↪ SPIwriteNBytes(CS, Reg, nBytes, Data)
39     #define MAGread(Addr, CS, Reg, nBytes, Data)
40         ↪ SPIreadMg(Addr, CS, Reg, nBytes, Data)
41     #define MAGwriteByte(Addr, CS, Reg, Data)
42         ↪ SPIwriteByteMg(Addr, CS, Reg, Data)
43     #define
44         MAGwriteNBytes(Addr, CS, Reg, nBytes, Data)
45         ↪ SPIwriteNBytesMg(Addr, CS, Reg, nBytes, Data)
46 #else
47     #error "No Protocol has been selected"
48 #endif
49
50 /* VARIABLES GLOBALES */
51 extern uint8_t Acc_Gyro[14]; //Accel-Gyro
52     ↪ DATA
53 extern uint8_t Mag[7];
54     ↪ //Magnetometer DATA
55
56
57 /****** FINAL CÓDIGO *****/
58     ↪ ARDUINO *****
59
60 extern QString sAUX;
61 extern const float MAG_FS;
62
63 extern const float ACC_FS; // (9.81 * 2)
64     ↪ / 2^15
65 extern const float GYR_FS; // 250 /
66     ↪ 2^15
67
68 typedef std::initializer_list<float>
69     ↪ floatList;
70 class MocapSuit;
71
72 class Mpu9250{
73 public:
74     Mpu9250(uint _CS, MocapSuit& _Suit,
75         ↪ floatList _Ac_Adj,
76         ↪ floatList _Gy_Adj, floatList
77         ↪ _Mg_Adj, floatList
78         ↪ _UE_init);
79     ~Mpu9250();
80     enum loopFlag{
81         Normal,
82         Align,
83         NoImprimir
84     };
85     void setup(void);
86     void loop(loopFlag loopType = Normal);
87     void AcGy_Reset(void);
88     int AcGy_setUp(void);
89     void Mag_Reset(void);
90     int Mag_setUp(void);
91     friend class MocapSuit;
92 private:
93     const uint CS; // Chip Select
94     const Vector UE_init; // Ángulos de
95         ↪ Euler en la pose-T de Unreal
96     MatrizCuadrada Align_Ajd = {1,0,0,
97         ↪ 0,1,0, 0,0,1};
98     bool isAligned = false;
99     uint id = 0;
100     static uint countOfSens;
101     float AX, AY, AZ, GX, GY, GZ, MX, MY,
102         ↪ MZ; //9 GDL's DATA
103     float AR, GR, MR, AR_ant, GR_ant,
104         ↪ MR_ant, delta_AR, delta_GR,
105         ↪ delta_MR;
106     int16_t Acc_Bias[3], Gyr_Bias[3],
107         ↪ Mag_Bias[3]; //Sumar Offset
108     float Acc_Adj[3], Gyr_Adj[3],
109         ↪ Mag_Adj[3]; //Multiplicar Offset
110     uint8_t ST1; //Magnetometer
111         ↪ "Status" value
112     //Complementos.h
113     //
114     uint8_t Nmuestras = 3;
115     Mean Ac_mean = Mean(Nmuestras);
116     Mean Gy_mean = Mean(Nmuestras);
117     Mean Ma_mean = Mean(Nmuestras);

```



```

97  //////////// DESPUÉS DEL VOID SETUP, PERO
98  ↪ ANTES DEL VOID LOOP: ////////////
99  int16_t ax_ant, ay_ant, az_ant;
100  ↪ uint8_t aCSSV = 0; //Count
101  ↪ Stagnant Sensor Values
102  int16_t mx_ant, my_ant, mz_ant;
103  ↪ uint8_t mCSSV = 0; //Count
104  ↪ Stagnant Sensor Values
105  uint32_t t_act, t_ant;
106  float t_delta;

107  //////////// COMIENZAN VARIABLES MATLAB:
108  ↪ ////////////
109  // // System constants
110  float deltat = 0.001f; // sampling
111  ↪ period in seconds (shown as 1 ms)
112  float gyroMeasError =
113  ↪ 3.14159265358979f * (5.0f /
114  ↪ 180.0f); // gyroscope measurement
115  ↪ error in rad/s (shown as 5 deg/s)
116  float gyroMeasDrift =
117  ↪ 3.14159265358979f * (0.2f /
118  ↪ 180.0f); // gyroscope measurement
119  ↪ error in rad/s/s (shown as 0.2f
120  ↪ deg/s/s)
121  float beta = sqrt(5.0f / 4.0f) *
122  ↪ gyroMeasError; // compute beta
123  ↪ beta_init = 2)
124  float zeta = sqrt(3.0f / 4.0f) *
125  ↪ gyroMeasDrift; // compute zeta

126  // Global system variables
127  float b_x = 0.454f, b_y = -0.189f, b_z
128  ↪ = -0.871f; // reference direction
129  ↪ of flux in earth frame
130  Vector b = {b_x, b_y, b_z};
131  const Vector b_E = b/norm(b);
132  const Vector g_E = {0.0f, 0.0f, 1.0f};
133  float w_bx = 0.0f, w_by = 0.0f, w_bz
134  ↪ = 0.0f; // estimate gyroscope
135  ↪ biases error

136  // // Local system variables
137  //SEqDot_omega = zeros(1,4); //
138  ↪ quaternion rate from gyroscopes
139  ↪ elements
140  Vector f = zeros(6); // objective
141  ↪ function elements
142  float J_11or24, J_12or23, J_13or22,
143  ↪ J_14or21, J_32, J_33; // objective
144  ↪ function Jacobian elements
145  float J_41, J_42, J_43, J_44, J_51,
146  ↪ J_52, J_53, J_54, J_61, J_62,
147  ↪ J_63, J_64;
148  Vector SEqHatDot = zeros(4); //
149  ↪ estimated direction of the
150  ↪ gyroscope error

151  //w_err(1); w_err(2); w_err(3); //
152  ↪ estimated direction of the
153  ↪ gyroscope error (angular)
154  //h_S(1), h_S(2), h_S(3); // computed
155  ↪ flux in the earth frame
156  // auxiliary variables to avoid
157  ↪ repeated calculations
158  Vector SEq = {1.0f, 0.0f, 0.0f, 0.0f};
159  MatrizCuadrada SErotM =
160  ↪ Quat2RotMat(SEq).t();

161  // inicializar CALIBRACIÓN
162  uint8_t i_capt=0; // Index de captura
163  uint8_t nCapt = 20; // #Num de Valores
164  ↪ por capturar
165  float Ac_Tol = 1.08f; // Tolerancia de
166  ↪ Error Total Accel
167  float Mg_Tol = 1.08f; // Tolerancia de
168  ↪ Error Total Magnet
169  Matriz Ac_capt = ones(nCapt,3); // =
170  ↪ ones(nCapt,3); //It can't be
171  ↪ zeros(), because make "inf" values
172  Matriz Mg_capt = ones(nCapt,3); //Same
173  ↪ comment of "Ac_capt"
174  Matriz Gy_capt = ones(nCapt,3);
175  Vector Ac_ant = {0.1f, 0.1f, 0.1f};
176  ↪ //Same comment of "Ac_capt"
177  Vector Gy_ant = {0.1f, 0.1f, 0.1f};
178  Vector Mg_ant = {0.1f, 0.1f, 0.1f};
179  ↪ //Same comment of "Ac_capt"

180  uint8_t nCalib = 2;
181  Vector d_Ac = ones(nCalib); //It can't
182  ↪ be zeros()
183  Vector d_Mg = ones(nCalib);
184  Vector Ac_Adj; // = {9.5669,
185  ↪ 8.8451, 10.6146, 0.0165,
186  ↪ 0.7612, 1.6092};
187  Vector Ac_Adj_Sens = ones(3); // =
188  ↪ Ac_Adj(0)/Ac_Adj(0,2);
189  ↪ //Acel Calib
190  Vector Ac_Adj_Offs = zeros(3); // =
191  ↪ Ac_Adj(3,5);
192  Vector Gy_Adj; // = {-1.3005,
193  ↪ 0.0240, -0.6723}; // Gyro Calib
194  Vector Mg_Adj; // = {39.6496,
195  ↪ 43.5980, 40.8667, -5.3843,
196  ↪ 10.1385, -12.7517};
197  Vector Mg_Adj_Sens = ones(3); // =
198  ↪ Mg_Adj(0)/Mg_Adj(0,2); //Mag Calib
199  Vector Mg_Adj_Offs = zeros(3); // =
200  ↪ Mg_Adj(3,5);

201  uint32_t i=0, j=0;
202  float t1 = 0.0f, t2 = 0.0f,
203  ↪ time_sensor = 0.0f;

204  };
205  #endif // MPU9250_H

```

## B.16. SPI.h

```
1 #ifndef SPI_H
2 #define SPI_H
3
4 #define SPI_CLOCK_DIV2 2
5 #define SPI_CLOCK_DIV4 4
6 #define SPI_CLOCK_DIV8 8
7 #define SPI_CLOCK_DIV16 16
8 #define SPI_CLOCK_DIV32 32
9 #define SPI_CLOCK_DIV64 64
10
11 class _SPI{
12 public:
```

```
13     static int setClockDivider(uint8_t
14         Divider); ///Agregar la seleccion
15         ⇔ de canal por parametro
16     static uint8_t transfer(uint8_t Data);
17 private:
18 };
19
20 extern _SPI SPI;
21
22 #endif // SPI_H
```