Universitat de les Illes Balears

escola politècnica superior

Treball Final de Grau

GRAU D'ENGINYERIA ELECTRÒNICA INDUSTRIAL I AUTOMÀTICA

Desarrollo de una interfaz de teleoperación de robots móviles para dispositivos Android

DAVID ARÉVALO JIMÉNEZ

Tutors

Javier Antich Alberto Ortiz

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, 8 de septiembre de 2017

Me gustaría agradecer especialmente a mis tutores Javier Antich y Alberto Ortiz el haber confiado en mi y darme la oportunidad de poder trabajar en un proyecto de estas características. Y por supuesto, dar las gracias también a la Universidad y sus profesores por todo lo enseñado a lo largo de estos 4 años haciendo posible el desarrollo de este proyecto.

ÍNDICE GENERAL

Índice general **iii**

Índice de figuras **vii**

Índice de tablas **xi**

Acrónimos **xii**

Resumen **xiii**
Resumen . . . xiii
Summary . . . xiii

1 Introducción **1**
1.1 Motivación . . . 1
1.2 Objetivos . . . 1
1.3 Comunicación cliente-servidor . . . 2
1.4 Fusión de sensores . . . 3
1.5 Estructura del documento . . . 5

2 Especificaciones de la aplicación a desarrollar **7**

3 Herramientas software utilizadas **10**
3.1 Android Studio . . . 10
3.2 NetBeans . . . 10
3.3 Robot Operating System (ROS) . . . 12
3.3.1 Comunicación básica en ROS . . . 12
3.3.2 Turtlesim . . . 12
3.3.3 Modular Open Robots Simulation Engine (MORSE) . . . 14

4 Diseño y creación de la App **15**
4.1 Fusión de sensores y representación gráfica del resultado . . . 15
4.1.1 Punto de partida . . . 15
4.1.2 Renderizado 3D . . . 16
4.1.3 Movimiento del punto central . . . 18
4.2 Diseño de la interfaz de control principal . . . 21
4.2.1 Interfaz gráfica . . . 21
4.2.2 Testeo con la imagen de un dron . . . 21
4.2.3 Elementos extras . . . 24

4.3 Comunicación cliente-servidor . . . 27
4.3.1 Netbeans . . . 27
4.3.2 Implementación del Cliente . . . 27
4.4 Diseño y distribución general de la App . . . 29
4.4.1 Arquitectura . . . 29
4.4.2 Traducción de idiomas . . . 39

5 Simulaciones en ROS 42
5.1 Concepto de package en ROS . . . 42
5.2 Estrategia y objetivos . . . 43
5.3 TurtleSim . . . 43
5.3.1 Comunicación con Topics . . . 44
5.3.2 Control del robot . . . 46
5.3.3 Servidor . . . 47
5.3.4 Demostraciones . . . 49
5.4 MORSE . . . 52
5.4.1 Introducción . . . 52
5.4.2 Robot terrestre ATRV . . . 53
5.4.3 Robot aéreo Quadrotor . . . 58

6 Conclusiones 65
6.1 Aplicación Android . . . 65
6.2 Ubuntu . . . 66
6.3 Futuras mejoras . . . 66

A Código fuente 68
A.1 Código de la App . . . 68
A.1.1 AndroidManifest.xml . . . 68
A.1.2 HomeActivity.java . . . 69
A.1.3 SensorSelectionActivity.xml . . . 75
A.1.4 Rotation.java . . . 84
A.1.5 OrientationVisualisationFragment.java . . . 88
A.1.6 Settings.java . . . 92
A.1.7 CubeRender.java . . . 97
A.1.8 Client.java . . . 106
A.1.9 Circle.java . . . 108
A.1.10 Lines.java . . . 111
A.1.11 Cuadrado.java . . . 112
A.1.12 Joystick.java . . . 117
A.1.13 Compass.java . . . 124
A.1.14 OrientationProvider.java . . . 127
A.1.15 ImprovedOrinetationSensor1Provider.java . . . 130
A.1.16 layout_home.xml . . . 139
A.1.17 layout_help_selection.xml . . . 144
A.1.18 activity_sensor_selection.xml . . . 146
A.1.19 settings.xml . . . 148
A.2 Implementación en TurtleSim . . . 150

A.2.1 robot_controller.cpp 150
A.3 Implementación en MORSE simulator 157
A.3.1 car_controller.cpp 157
A.3.2 drone_controller.cpp 168

Bibliografía **183**

ÍNDICE DE FIGURAS

1.1 Esquema de la estructura Cliente-Sevidor utilizada 2
1.2 Sistema de coordenadas en un dispositivo Android 2
1.3 Arquitectura Cliente-Servidor 3
1.4 Implementación del Servidor en Ctt ROS 3
1.5 Implementación del Cliente en Java 4

2.1 Menu inicio de la App 7
2.2 Modos de manejo: Normal y táctil 8
2.3 Demostración del cambio de modo de control 9

3.1 Interfaz de trabajo de Android Studio 11
3.2 Interfaz de trabajo de NetBeans 11
3.3 Ejemplo de comunicación mediante Topcis en ROS 13
3.4 Interfaz de Turtlesim 13
3.5 Interfaz de MORSE 14
3.6 Robots utilizados en el proyecto: ATRV y Quadrotor 14

4.1 Librerías incluidas en el proyecto Sensor Fusion Demo for Android 16
4.2 Pantalla inicial del proyecto Sensor Fusion Demo for Android 17
4.3 Ejemplo de cómo definir las coordenadas de un triángulo 17
4.4 Implementación del método draw() 18
4.5 Implementación del método onSufaceCreated() 19
4.6 Implementación de la selección de sensores 19
4.7 Esquema de la implementación del movimiento del punto de control (punto amarillo) 20
4.8 Teorema del coseno y cálculo del arco de una circunferencia 20
4.9 Implementación del clicado largo para calibrar el punto) 21
4.10 Interfaz gráfica únicamente con el punto de control 22
4.11 Inicialización de las circunferencias del fondo 22
4.12 SurfaceView con todos los elementos a renderizar 23
4.13 Declaración del ViewPager en el layout 23
4.14 Visualización del modo de testeo de sensores con un dron 24
4.15 Vista con Seekback implementado 25
4.16 Layout donde se implementa la brújula 26
4.17 Uso de la brújula utilizada 26
4.18 Vista del joystick 26
4.19 Implementación del Joystick 27

4.20 Implementación del método doInBackground 28
4.21 Implementación del método onDrawFrame 29
4.22 Diagrama de flujo de la App 30
4.23 Ejemplo del subprograma de acceso al control del robot desde el menú inicial 31
4.24 Vista del menú principal 31
4.25 Implementación del control de acceso mediante un dialogo de password . 32
4.26 Vistas del menú de ajustes (Settings) 33
4.27 Implementación del layout de ajustes mediante una PreferenceScreen . . . 33
4.28 Lectura de los elementos de tipo preference 34
4.29 Interfaz intermedia para la correcta colocación del dispositivo móvil 35
4.30 Condición de orientación del móvil para acceder al control del robot 35
4.31 *Intent* para acceder a la pantalla de control 35
4.34 Uso del adaptador en la ViewPager 36
4.32 Implementación del FragmentPagerAdapter 37
4.35 Selección de la técnica de sensores a aplicar 37
4.33 Ejemplo de dos actividades con una y varias páginas 38
4.36 Implementación del método onCreateView 38
4.37 Un fichero strings.xml por cada idioma soportado 39
4.39 Implementación del método Refresh() 39
4.38 Implementación del cambio de idioma 40
4.40 Selección de idioma 41
4.41 Ejemplo de diferentes vistas de la App en diferentes idiomas 41

5.1 Archivos del package 42
5.2 Declaración de interés de CMakeList.txt y Package.xml 43
5.3 Ejecución de Roscore, TurleSim y el nodo turtlesim_controller 44
5.4 Demostración del control en TurtleSim del modo ABSOLUTE 45
5.5 Gráfico de la comunicación mediante Topics entre ROS y MORSE 45
5.6 Declaración de variables globales y métodos en TurtleSim 45
5.7 Asignación de Topics al publisher y subscriber 46
5.8 Lectura de la posición de la tortuga 46
5.9 Cálculo de la condición de salida del bucle 47
5.10 Cálculo del sentido de giro del robot en TurtleSim 47
5.11 Implementación del Servidor 48
5.12 Aceptación de una conexión y creación de un nuevo thread 48
5.13 Implementación de la lectura de mensajes enviados por el Cliente 49
5.14 Conversión del ángulo del punto amarillo de control al sistemas de coordenadas utilizado por el simulador TurtleSim 50
5.15 Sistemas de coordenadas del cliente (izquierda Normal mode y derecha Tactile mode) y el utilizado por el simulador TurtleSim 50
5.16 Simulaciones con TurtleSim controlado desde el móvil. 51
5.17 Escena para la simulación de un robot terrestre en MORSE 52
5.18 Imagen de la simulación del robot terrestre 53
5.19 Topics utilizados para el robot terrestre 54
5.20 Gráfico de la comunicación mediante Topics entre ROS y MORSE para el robot terrestre 54
5.21 Descripción de la variable tipo PoseStamped 55

5.22 Cambios producidos por la nueva variable tipo PoseStamped 55
5.23 Subscriber y Publisher actualizados 56
5.24 Explicación gráfica del modo Relative 56
5.25 Implementación del modo RELATIVE 57
5.26 Selección del modo de control 57
5.27 Cambios en la condición de salida del bucle de giro 58
5.28 Escena implementada para el control del cuadricóptero 59
5.29 Lista de los Topics utilizados para controlar el vehículo aéreo 59
5.30 Implementación de la escena del Quadrotor 60
5.31 Parámetros del actuador del robot modificados experimentalmente 60
5.33 Comunicación entre ROS, MORSE y el cuadricóptero 61
5.32 Nuevos Topics para el Subscriber y Publisher del dron 61
5.34 Subprograma de control de altura del robot 62
5.35 Utilización del subprograma checkHeight en el modo Absolute 62
5.36 Explicación gráfica del modo Relative 62
5.37 Implementación para orientar el cuadricóptero al sistema de coordenadas del dispositivo 63
5.38 Implementación del nuevo algoritmo de control Position 64

ÍNDICE DE TABLAS

1.1 Tipos de sensores considerados 5

ACRÓNIMOS

IDE Integrated Development Environment

IP Internet Protocol

MORSE Modular Open Robots Simulation Engine

ROS Robot Operating System

TCP Transport Control Protocol

RESUMEN

Resumen

La utilización de robots aéreos tipo cuadricópteros o también conocidos como drones se están haciendo cada vez más presente en el mundo cotidiano siendo accesibles para cualquiera. Además, debido a su enorme potencial son fuente de estudio e investigación en muchas universidades y laboratorios. Planteado este escenario, la UIB propone la creación de una App móvil que permita la teleoperación de sus robots móviles desde dispositivos android, con especial interés en los cuadricópteros. A su vez, se intenta hacer una aplicación de fácil uso y entendimiento mediante el control a través de los sensores de orientación internos del dispositivo.

Al tratarse de un trabajo extenso y complejo, se ha dividido en dos fases de la cual este proyecto abarcará la primera: el desarrollo de la aplicación y control robots móviles tanto aereos como terrestres mediante simulador, mientras que la segunda se centraría en su implementación en robots real. Este proyecto adoptará una estructura de cliente-servidor, donde el cliente será el dispositivo móvil con la App para Android y el servidor, programado en ROS, será el encargado del control del robot simulado en MORSE.

Summary

The use of aerial robots type quadrotor or also known as drones are becoming increasingly present in the everyday world being accessible to anyone. In addition, because of their enormous potential they are a source of study and research in many universities and laboratories. With this context, the UIB proposes the creation of a mobile App that allows the control of its robots with android smartphones, with special interest in its quadrotors. For this reason it is tried to make an application of easy use and control with the internal orientation sensors of the mobile.

As it is an extensive and complex work, it has been divided into two phases of which this project will cover the first, the development of the application and control of mobile robots (aerial and terrestrial) in a simulator, while the second would focus on its implementation in a real robots. This project will have a client-server structure, where the client will be the mobile device with the Android App and the server, programmed in ROS, will be in charge of the control of the simulated quad-core in MORSE.

CAPÍTULO 1

INTRODUCCIÓN

1.1 Motivación

Actualmente existe un gran interés sobre el potencial y aplicaciones que ofrecen los robots móviles. Este interés se incrementa en instituciones tecnológicas como la Universitat de les Illes Balears donde encontramos un laboratorio centrado en estas tecnologías.

Paralelamente, observamos la importancia de las tecnologías móviles y sus Apps, con una gran influencia en las nuevas generaciones y su integración en nuevos campos, situación que aprovecharemos para crear una nueva herramienta con la que mostrar el potencial que ofrecen estas plataformas.

Dado este contexto, se propone implementar un nuevo modo de control de los los robots móviles de la UIB mediante una aplicación móvil con el fin de simplificar la interacción.

1.2 Objetivos

El propósito general de este trabajo es el diseño de una aplicación para Android con la que poder controlar un robot móvil de una manera fácil, intuitiva y segura sin necesidad de un elemento de control auxiliar como un control remoto por radiofrecuencia o un joystick.

Dadas las dimensiones del proyecto, éste trabajo se centrará en el desarrollo y diseño de la aplicación móvil y del correcto funcionamiento en un simulador como MORSE. Para ello se propone una arquitectura cliente-servidor, siendo el dispositivo móvil el cliente y el servidor el ordenador con ROS que se comunicará con el simulador (Fig. 1.1).

En relación a la manejabilidad, se pretende poder controlar el robot con los sensores internos del móvil (Fig. 1.2), es decir, haciendo uso de los sensores de posicionamiento y orientación del propio celular como pueden ser el giróscopo, el acelerómetro o la brú-

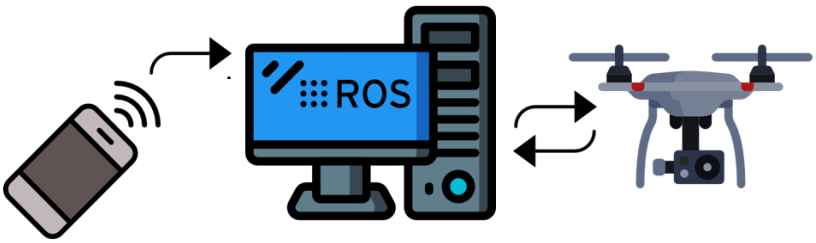

Figura 1.1: Esquema de la estructura Cliente-Sevidor utilizada

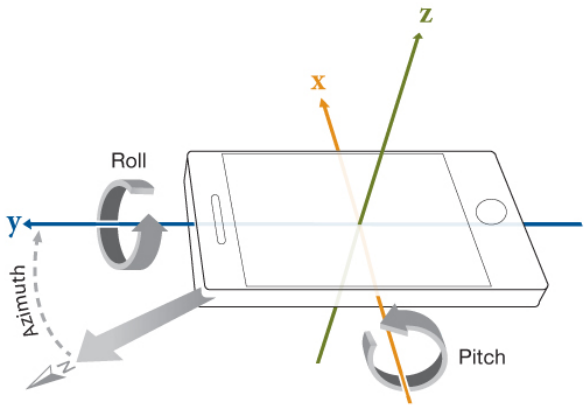

Figura 1.2: Sistema de coordenadas en un dispositivo Android

jula, facilitando la interacción y reduciendo la complejidad de un mando convencional para un amateur.

1.3 Comunicación cliente-servidor

La comunicación entre clientes y servidores se basa en la programación de sockets de flujo (Transport Control Protocol (TCP)). Un socket es la conexión entre dos computadores que les permite comunicarse e intercambiar datos. Para que esto ocurra debe existir un nodo Cliente y uno Servidor [1].

En nuestro caso, el Cliente está programado en Java (Android Studio) y el Servidor en C++ (ROS). Además, cabe destacar que se trata de una conexión multithread, por lo que por cada mensaje intercambiado se crea una nueva conexión, lo que permitiría, si se desease, poder conectar varios clientes (Fig. 1.3).

Para su implementación (Fig. 1.4) hacemos uso de la clase Socket(). En el caso del Servidor, se crea un socket con una dirección Internet Protocol (IP) y un número de puerto, se realiza una operación bind() para comprobar que la dirección esta libre para nuestro servidor y se deja el nodo en modo escucha mediante la instrucción listen(). Una vez el cliente se conecta a la dirección IP y al puerto de nuestro servidor, se le asigna un nuevo socket (con un puerto diferente), para dejar libre la conexión para otro cliente que lo solicite, y comienza el intercambio de información. En nuestro caso, el

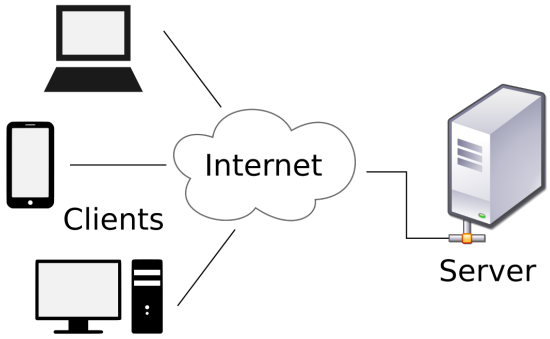

Figura 1.3: Arquitectura Cliente-Servidor

```
**********SERVER*************
[...]
//Create socket
socket_desc = socket(AF_INET , SOCK_STREAM , 0);
if (socket_desc == -1){
    printf("Could not create socket");
}
puts("Socket created");

//Prepare the sockaddr_in structure
server.sin_family = AF_INET;
server.sin_addr.s_addr = inet_addr("192.168.1.50");
server.sin_port = htons( 8080 );

//Bind
if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0){
    //print the error message
    perror("bind failed. Error");
    return 1;
}
puts("bind done");
[...]

//Listen
listen(socket_desc , 3);

//Accept and incoming connection
puts("Waiting for incoming connections...");
while( (client_sock = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&sizeof(struct sockaddr_in))) ){
    puts("Connection accepted");
    [...]
}
[...]
```

Figura 1.4: Implementación del Servidor en Ctt ROS

móvil enviará datos para estimar la dirección y velocidad a la que queremos que vaya el robot, entre otros parámetros. El cliente es más simple: crea un socket a la dirección IP y puerto del servidor y mediante un output.print(data) envía los datos de interés al servidor. La implementación se muestra en la Fig. 1.5.

1.4 Fusión de sensores

Un móvil contiene diversos sensores útiles para estimar la posición y orientación de éste. Por tal de conseguir resultados más precisos y rápidos, se aconseja utilizar

```
/*********************CLIENT*****************************/
public class Client extends AsyncTask<Void, Void, Void> {

    String dstAddress;
    int dstPort;

    public Client(String addr, int port) {
        dstAddress = addr;
        dstPort = port;
    }

    @Override
    protected Void doInBackground(Void... arg0) {

        Socket socket = null;
        OutputStream outputStream;
        try {
            socket = new Socket("IP", "PORT");

            PrintStream output = new PrintStream(socket.getOutputStream());
            output.print(data1,data2,...);
            output.close();

        } catch (UnknownHostException e) {
            [...]
        }
        return null;
    }
    [...]

}
```

Figura 1.5: Implementación del Cliente en Java

más de uno de estos sensores. A esto se le llama fusión de sensores o datos. En la Tabla. 1.1 se muestran los sensores internos utilizados para la fusión. La fusión de datos permite reducir las incertidumbres, suavizar los señales ruidosas, evitar interferencias ambientales que puedan afectar a los sensores y evitar que el sistema deje de funcionar en caso de avería.

En nuestro caso se han implementado 6 técnicas de fusión de sensores diferentes para poder elegir y comparar la efectividad de cada una de ellas:

- *Improved Orientation Sensor 1*. Técnica de fusión que utiliza los datos proporcionados por el giróscopo y el vector de rotación de Android. De manera genérica, su funcionamiento se basa en comprobar la divergencia entre los datos de cada uno de los sensores, en el caso de que superen el margen establecido de divergencia se selecciona el sensor que proporciona los datos menos dispares. Ademas se utilizan filtros que permiten obtener mejores resultados para cada sensor. Si las lecturas de ambos sensores son óptimas, se interpolan los datos.

- *Improved Orientation Sensor 2*. Este modo de fusión es muy similar al anterior, diferenciándose por los valores en los filtros y umbrales utilizados. Éste es más estable pero menos preciso que el anterior.

Sensor	Descripción
TYPE_ACCELEROMETER	Mide la aceleración en m/s^2 que se aplica al dispositivo en los tres ejes del espacio. Incluye la fuerza de la gravedad.
TYPE_GRAVITY	Mide la fuerza de la gravedad ejercida sobre el dispositivo en los ejes XYZ.
TYPE_GYROSCOPE	Mide la velocidad de rotación del dispositivo alrededor de cada uno de los tres ejes físicos.
TYPE_ROTATION_VECTOR	Mide la orientación del dispositivo proporcionando los tres elementos del vector de rotación del dispositivo.
TYPE_MAGNETIC_FIELD	Mide el campo magnético ejercido en cada uno de los tres ejes del espacio del dispositivo.

Tabla 1.1: Tipos de sensores considerados

- *Android Rotation Vector*. Esta nueva técnica de fusión hace uso del acelerometro, el giroscopio y la brújula del dispositivo. A diferencia de las dos fusiones anteriores, éste se basa en los llamados filtros de Kalman.
- *Calibrated Gyroscope*. Este modo se centra en un único sensor, el giróspoco del móvil, al que se le aplica un filtro de Kalman para mejorar sus lecturas respecto a las proporcionadas por defecto.
- *Gravity+Compass*. Utiliza los sensores de campo magnético y gravitatorio para obtener la orientación del dispositivo a partir del método getRotatioMatrix de la clase SensorManager.
- *Accelerometer+Compass*. La obtención de los datos fusionados se obtiene de la misma manera que en el caso anterior, pero para el acelerometro y la brújula.

Estas agrupaciones de sensores han sido obtenidas de la tesis de máster del doctor Alexander Apacha [2], por los buenos resultados experimentales que presenta.

1.5 Estructura del documento

El presente documento se ha dividido en siete capítulos. La progresión de los capítulos está hecha de manera que el lector sea capaz de ir introduciéndose poco a poco en el tema principal e intentar lograr una mejor comprensión del mismo.

- *Capítulo 1: Introducción*, en el que se realiza una introducción al trabajo.
- *Capítulo 2: Especificaciones de la aplicación a desarrollar*, en el que se detallan las funcionalidades de la App.
- *Capítulo 3: Herramientas software utilizadas*, en el que se describen los programas utilizados.
- *Capítulo 4: Diseño y creación de la App*, en el que se explica el desarrollo y funcionamiento de la aplicación móvil.

- *Capítulo 5: Simulaciones en ROS*, en el que se muestra la implementación del programa de control en diferentes simuladores, Turtlesim y MORSE, empezando por robots terrestres y pasando luego a aéreos.
- *Capítulo 6: Conclusiones*, en el que se muestran los resultados obtenidos, los diferentes problemas encontrados y futuras posibles mejoras al proyecto.

CAPÍTULO 2

ESPECIFICACIONES DE LA APLICACIÓN A DESARROLLAR

Los usuarios del sistema serán principalmente no expertos en manejo de robots móviles. Además, tiene que ser una aplicación escalable, de manera que en un futuro se puedan incorporar nuevas funciones. Por estos motivos, se ha buscado que tenga un diseño

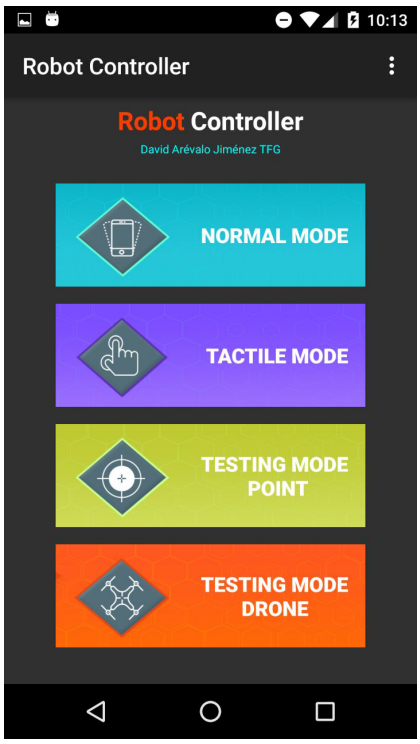

Figura 2.1: Menu inicio de la App

simple y atractivo.

La aplicación presentará un menú inicial (Fig. 2.1) en el que se presentan diferentes opciones de uso, tanto de manejo del robot como de testeo de los sensores utilizados. Estas últimas tienen la utilidad de ayudar al usuario a determinar qué sensores presentan un mejor funcionamiento en su dispositivo móvil, ya que no todos los móviles son iguales.

Se diseñarán dos modos de control que se diferenciarán en el tipo de manejo (Fig. 2.2). El primer modo utilizará los sensores de orientación del móvil, de manera que sea la propia orientación del dispositivo la que determine las acciones que debe realizar el robot, mientras que el segundo modo permitirá hacer uso de un joystick virtual, siendo más parecido al método tradicional.

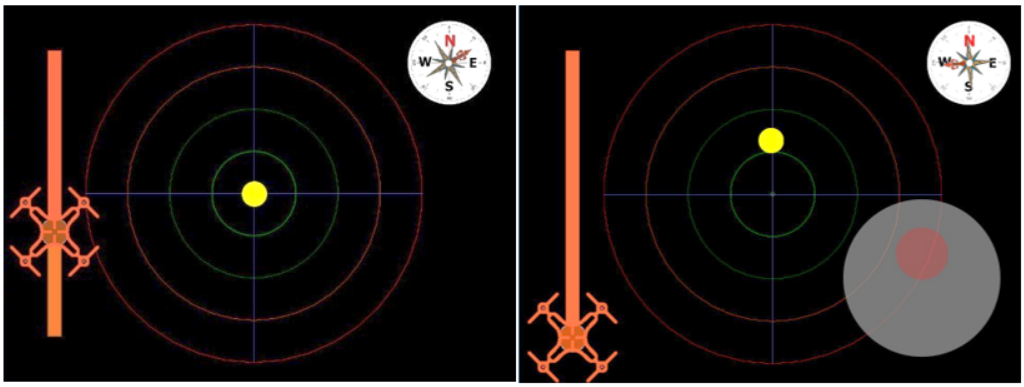

Figura 2.2: Modos de manejo: Normal y táctil

Sin embargo, en ambas opciones se pretende implementar varios modos de control del robot con la intención de facilitar y adaptar el manejo a la habilidad de cada usuario. A continuación, se enumeran y describen brevemente los 3 modos:

- *Absolute mode*. En este modo el punto amarillo representa el extremo del vector velocidad, indicándonos el ángulo la dirección en coordenadas absolutas a la que queremos que se dirija el robot y el módulo la velocidad lineal.
- *Relative mode*. El vector posición del punto amarillo representa los vectores de velocidad lineal y angular, siendo la longitud la primera de ellas y el ángulo respecto a la vertical la angular. Este modo es el que se usa normalmente en mandos de control de cuadricópteros. Sin embargo, se presenta como el modo de manejo más avanzado y complejo de los tres.
- *Position mode*. A diferencia de las dos anteriores, las circunferencias representan el espacio real como si se tratase de una vista aérea del espacio, siendo el robot representado por el punto amarillo y la circunferencia de radio mayor (la de color rojo) los límites de circulación del robot. Este modo, por tanto, se presenta como el modo más simple y seguro gracias a los limites espaciales y la orientación fija del robot.

Los modos se sitúan en el menú de la esquina superior derecha de la pantalla. Para poder cambiar de modo será necesario tener el robot en el suelo o lo que es lo mismo

el seekbar en posición 0 por motivos de seguridad. De lo contrario, aparecerá un *toast* (mensaje flotante) indicando el problema (Fig. 2.3).

En todos los casos, la altura se controlará mediante una seekbar que determine el porcentaje sobre la altura máxima permitida. También se introducirá una brújula por si el usuario la requiriese.

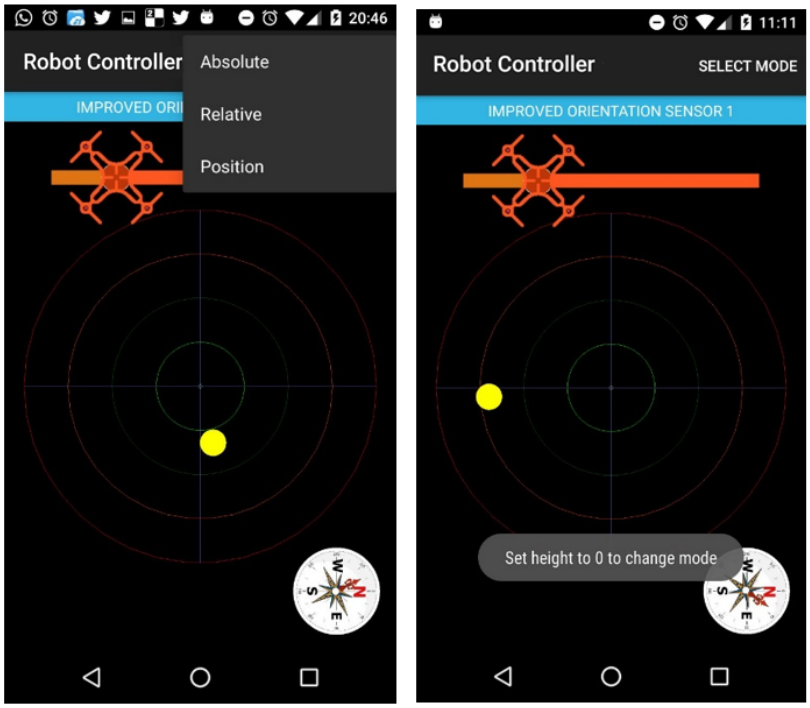

Figura 2.3: Demostración del cambio de modo de control

Por último, se pretende diseñar una sección de ajustes donde poder controlar todos los parámetros de interés como pueden ser velocidades máximas, sensibilidad, tipo de fusión de sensores, altura máxima entre otras. El acceso a este menú se realizará mediante un *password* solo conocido por el usuario administrador, dada la importancia de estos parámetros para un correcto funcionamiento.

CAPÍTULO 3

HERRAMIENTAS SOFTWARE UTILIZADAS

En este capítulo se revisarán brevemente las diferentes herramientas software utilizadas para desarrollar este proyecto: el entorno Android Studio, Netbeans y ROS.

3.1 Android Studio

Android Studio es un entorno de desarrollo integrado (Integrated Development Environment (IDE)) oficial para la plataforma Android, el famoso sistema operativo de Google. Un IDE es un editor de código que facilita la programación de una aplicación software gracias a las herramientas que proporciona: un editor de código, un depurador, herramientas de construcción automática de software, un auto-completado de código muy eficaz, un compilador y un intérprete.

Android Studio se instala de serie junto con un simulador virtual de Android mediante el cual se puede probar la aplicación en desarrollo mientras se realiza. Se puede observar un ejemplo de cómo es el aspecto del entorno de programación en la Fig. 3.1. Este programa ha sido el utilizado a lo largo de este proyecto para la realización de nuestra aplicación de Android.

3.2 NetBeans

NetBeans, al igual que Android Studio, es un entorno de desarrollo integrado libre desarrollado principalmente para el lenguaje de programación Java. En este caso se ha utilizado para programar y entender el funcionamiento de un cliente en lenguaje C y un servidor en lenguaje Java. En la Fig. 3.2 se puede ver la apariencia y diseño del entorno de la aplicación.

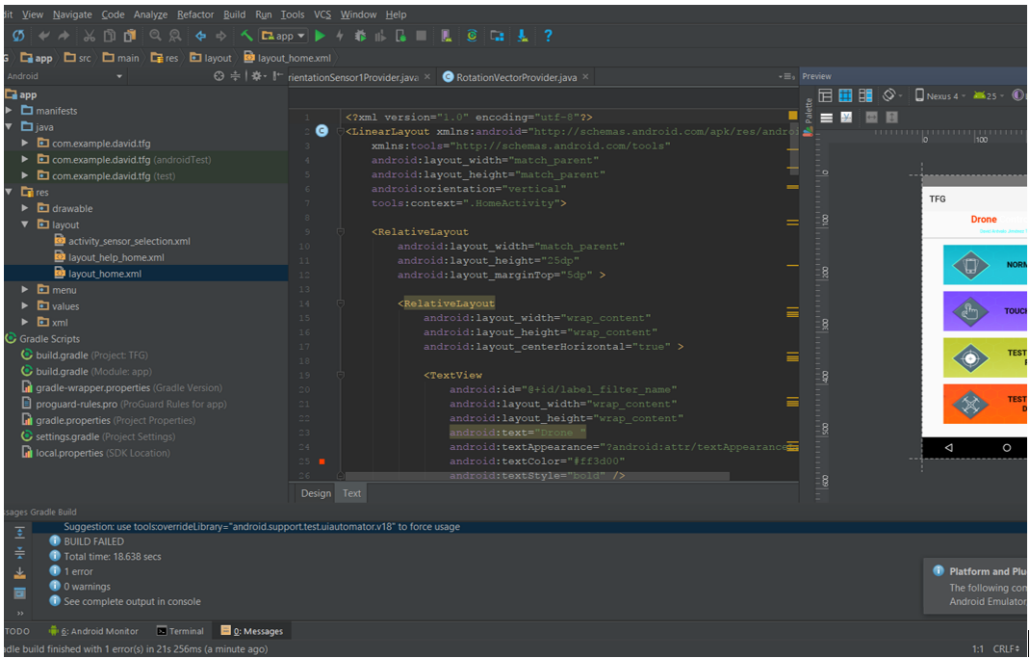

Figura 3.1: Interfaz de trabajo de Android Studio

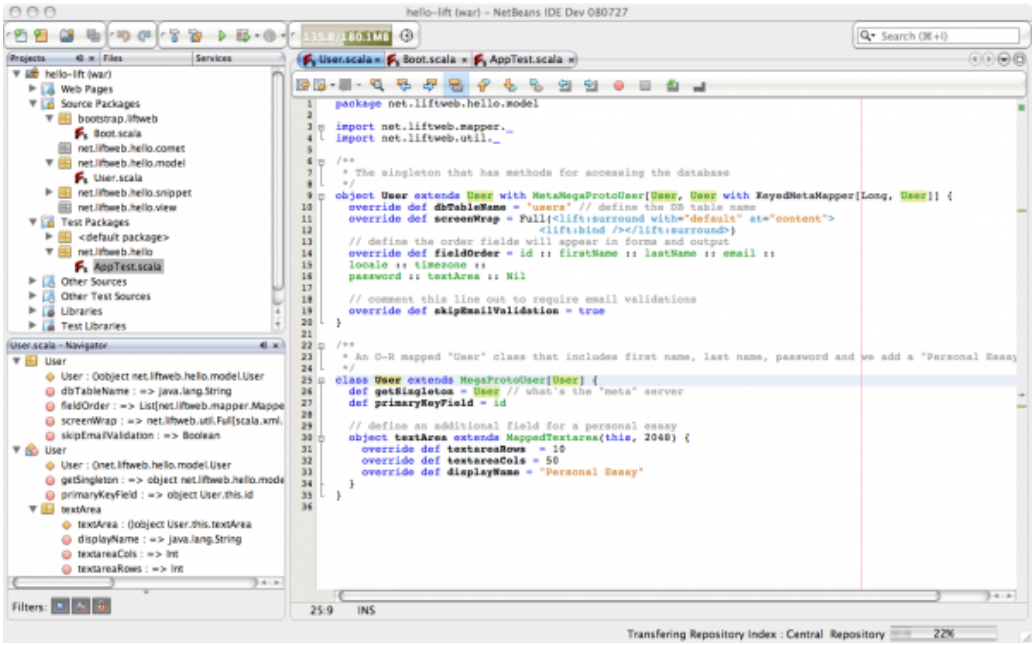

Figura 3.2: Interfaz de trabajo de NetBeans

3.3 ROS

El Sistema Operativo Robótico (ROS) es un entorno para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo. Su uso se restringe a Ubuntu por lo que en un primer momento se decidió utilizar una imagen virtual de Nootrix mediante VirtualBox. Nuestro proyecto se programa en ROS debido a la cantidad de paquetes que se nos facilitan para la comunicación con robots.

3.3.1 Comunicación básica en ROS

Las comunicaciones en ROS involucran los siguientes elementos [3]:

- *Nodos*. Los nodos son programas ejecutables que ejecuta el computador para cumplir con una función. Cada nodo que se ejecuta tiene un nombre único que lo identifica. Para el funcionamiento de esta simulación se utilizan varios nodos que se explicarán más adelante. Un beneficio de utilizar nodos es que hace al conjunto del sistema más robusto ante fallos; es decir, si un nodo falla, el resto de nodos del sistema puede seguir funcionando.

- *Mensajes*. Un mensaje es una estructura de datos que utilizan los nodos para comunicarse entre ellos. Son soportados tanto los tipos primitivos (booleano, entero, etc.) como los tipos compuestos tipo vector.

- *Topic*. Es el mecanismo mediante el cual los nodos intercambian mensajes. Los mensajes se enrutan mediante un publicador y un subscriptor. Un nodo publica un mensaje por un Topic determinado. Si hay un nodo que necesita un tipo de dato en concreto, debe suscribirse al Topic en cuestión para recibirlo. Puede haber varios publicadores y subscriptores simultáneos en cada Topic. El nombre del Topic se puede utilizar para identificar el contenido del mensaje.

- *Nodo Master*. Es el nodo principal de ROS, se encarga de realizar las conexiones entre los nodos que requieran comunicarse.

A modo de ejemplo, el gráfico de la Fig. 3.3 muestra un pequeño sistema compuesto por dos nodos, teleop_turtle y turtlesim, los cuales son representados mediante una elipse. Éstos se comunican a través de un topic llamado turtle1/command_velocity por el cual se envían los mensajes. La flecha indica que teleop_turtle es el nodo que envía mensajes (el nodo publicador) por dicho topic y turtlesim es el nodo que se ha suscrito al topic para recibirlos.

3.3.2 Turtlesim

Como primer simulador utilizaremos el nodo turlesim_node del paquete Turtlesim que nos permitirá desarrollar un primer programa de control de un robot terrestre en un entorno bidimensional. En la Fig. 3.4 se muestra la apariencia del simulador. Como se puede observar, el simulador emplea un robot con apariencia de tortuga para expresar el movimiento dentro del simulador.

Figura 3.3: Ejemplo de comunicación mediante Topcis en ROS

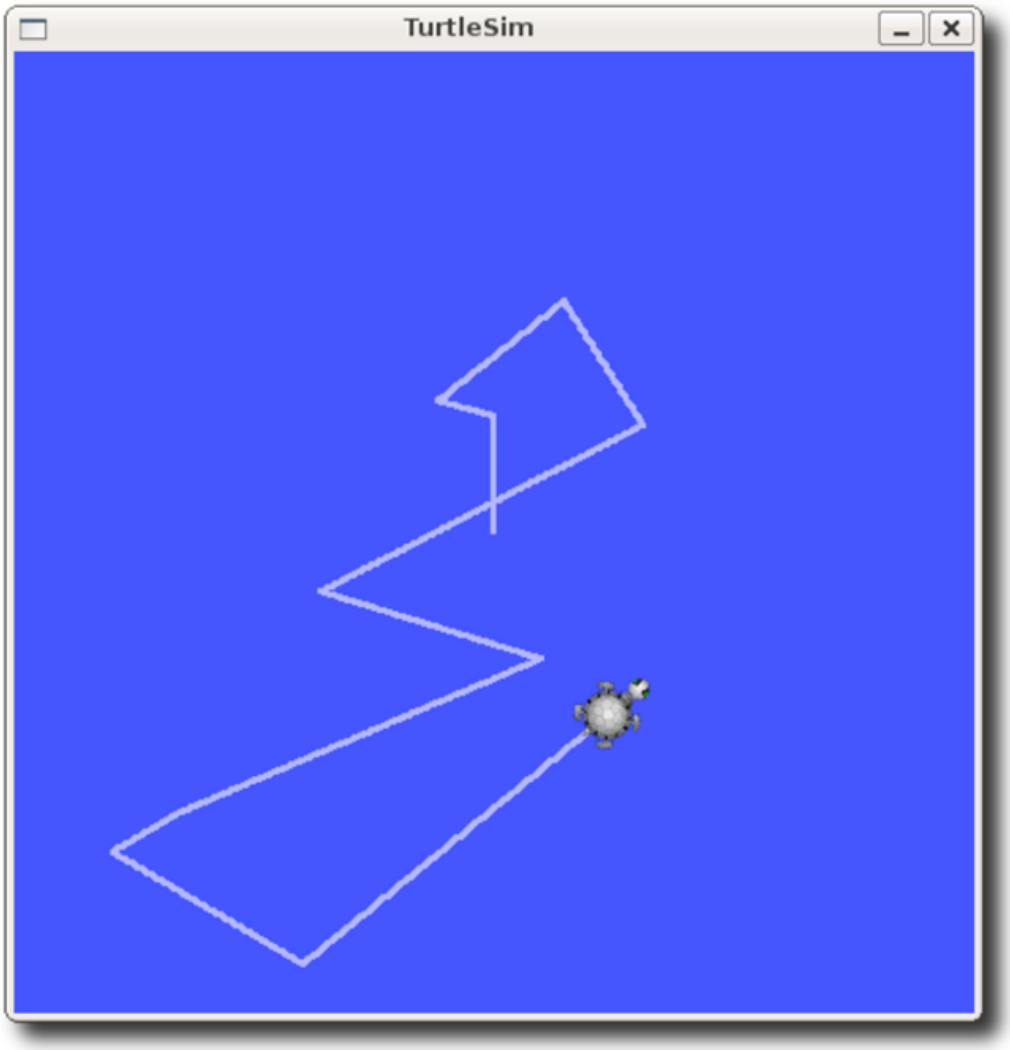

Figura 3.4: Interfaz de Turtlesim

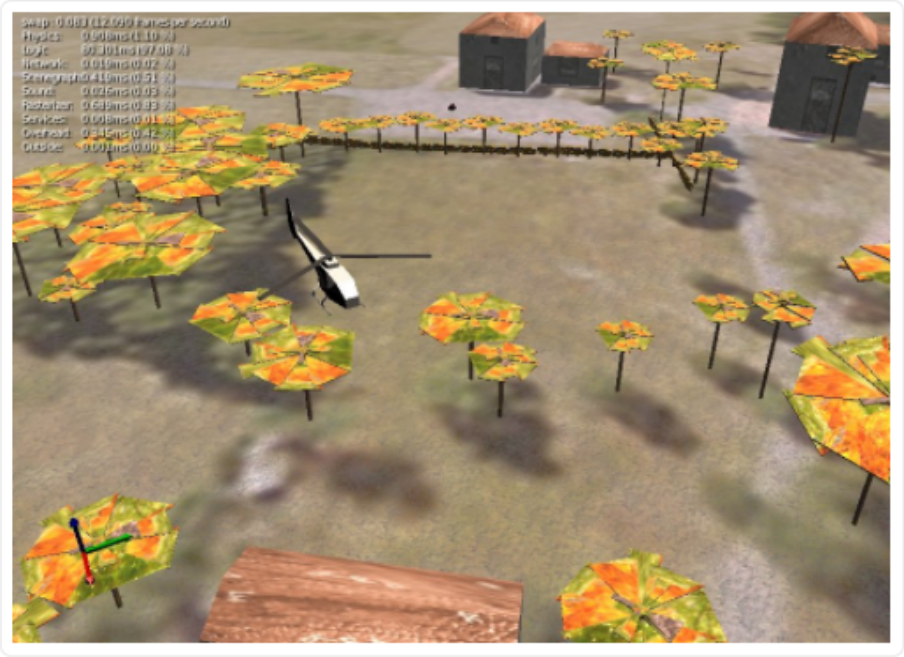

Figura 3.5: Interfaz de MORSE

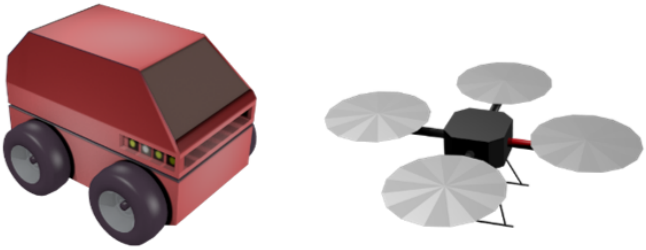

Figura 3.6: Robots utilizados en el proyecto: ATRV y Quadrotor

3.3.3 MORSE

Una vez familiarizados con ROS y la comunicación mediante Topics, pasamos a hacer uso de un simulador tridimensional más completo llamado MORSE. Éste es un simulador basado en scripts en Phyton y Blender que contiene un conjunto de librerías con robots terrestres y aéreos con sensores (cámaras, escáner láser, GPS, odometría,...), actuadores (por velocidad lineal, velocidad angular, posición,...) y escenas. En la Fig. 3.5 se observa un ejemplo de simulación en MORSE.

Este simulador permite la comunicación con ROS mediante Topics pudiendo enviar y recibir información a los actuadores y sensores de nuestro robot. Los robots que se utilizarán son el robot ATRV() y el robot Quadrotor() de la Fig. 3.6.

CAPÍTULO 4

DISEÑO Y CREACIÓN DE LA APP

Tal y como se comentó en el Capítulo 2, el sistema está enfocado a ser intuitivo, sencillo y fácil de usar, por lo que el diseño de la aplicación debe ser simple y directo. En la aplicación cliente es muy importante la interfaz gráfica, ya que ésta va enfocada a los usuarios; en este caso se ha optado por un estilo minimalista sin distracciones ni demasiadas opciones que puedan entorpecer su uso.

Para este capítulo se seguirá el mismo orden que para el desarrollo de la App, lo que permitirá al lector entender mejor el funcionamiento de ésta. A continuación, se enumeran cada una de las secciones.

1. Fusión de sensores y representación gráfica del resultado.
2. Diseño de la interfaz de control principal.
3. Comunicación cliente-servidor.
4. Diseño y distribución general de la App.

4.1 Fusión de sensores y representación gráfica del resultado

4.1.1 Punto de partida

Como ya se ha explicado en el Capítulo 1.4, la fusión de datos sensoriales permite obtener mejores estimaciones, más fiables y mas robustas. Como se pretende poder controlar un robot móvil con los valores proporcionados por los propios sensores del smartphone o tablet, es un punto importante dentro del proyecto. Sin embargo, es un tema complejo que requiere de un estudio adicional.

Dado que existen estudios previos y que este TFG presenta ya suficientes objetivos a lograr, se ha optado por reutilizar el trabajo del doctor Alexander Apacha en su proyecto *Sensor fusion demo for Android* [4] por los buenos resultados obtenidos en relación a la fusión se sensores en dispositivos Android[2]. Este proyecto incluye dos colecciones

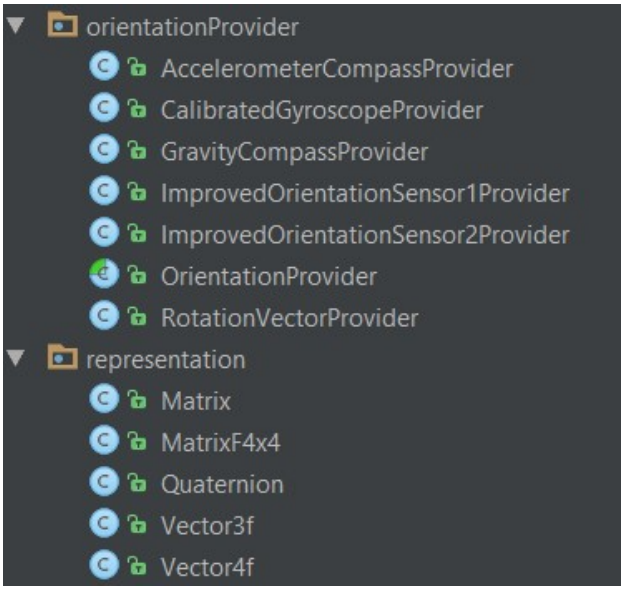

Figura 4.1: Librerías incluidas en el proyecto Sensor Fusion Demo for Android

de librerías con el fusionado de sensores y las herramientas matemáticas necesarias para implementar las operaciones correspondientes (matrices de rotación, vectores, cuaterniones...) tal y como se muestra en la Fig. 4.1. Además incluye un archivo java con el que se puede renderizar un entorno tridimensional mediante la clase GLSurfaceView.Renderer de la librería OpenGL de Android. En la Fig. 4.2 se muestra la apariencia de la aplicación a partir de la cual iniciamos nuestro trabajo.

4.1.2 Renderizado 3D

Para familiarizarnos con el entorno 3D y el uso de las clases OpenGL ha sido necesaria la lectura del capítulo 7: Gráficos en 3D en Android de la Unidad 1 del libro Desarrollo de aplicaciones para Android II [5] [6]. Partiendo del punto de partida explicado en el apartado anterior, se ha adaptado el entorno de renderizado a nuestras necesidades. A diferencia del cubo utilizado inicialmente, se opta por utilizar un punto como referencia, estando éste en el centro de la pantalla cuando el dispositivo se encuentra dispuesto horizontalmente. Para ello utilizamos la función *Circle(float radio, int type, float[] color)* que nos permite crear círculos en un entorno tridimensional de OpenGL. En nuestro caso, se ha decidido que sea un punto, por lo que mediante la variable *type* indicaremos que se trata de un círculo de color sólido (amarillo).

Como cualquier clase que defina una forma geométrica, se deben crear buffers para los vértices (Fig. 4.3), los colores y un floatbuffer para renderizar. Una vez definido, mediante un método *draw()* como el de la Fig. 4.4 se visualiza la figura por pantalla.

CubeRender.java es la clase encargada de llamar a los métodos draw() de cada uno de los elementos. En esta clase se implementa el método GLSurfaceView.Renderer. Primero se inicializan los diferentes objetos a dibujar, en nuestro caso un punto mediante la clase Circle(), y luego se inicializa el entorno de renderizado como se presenta en la Fig. 4.5. Finalmente, se ejecuta el método onDrawFrame() donde se dibuja el punto

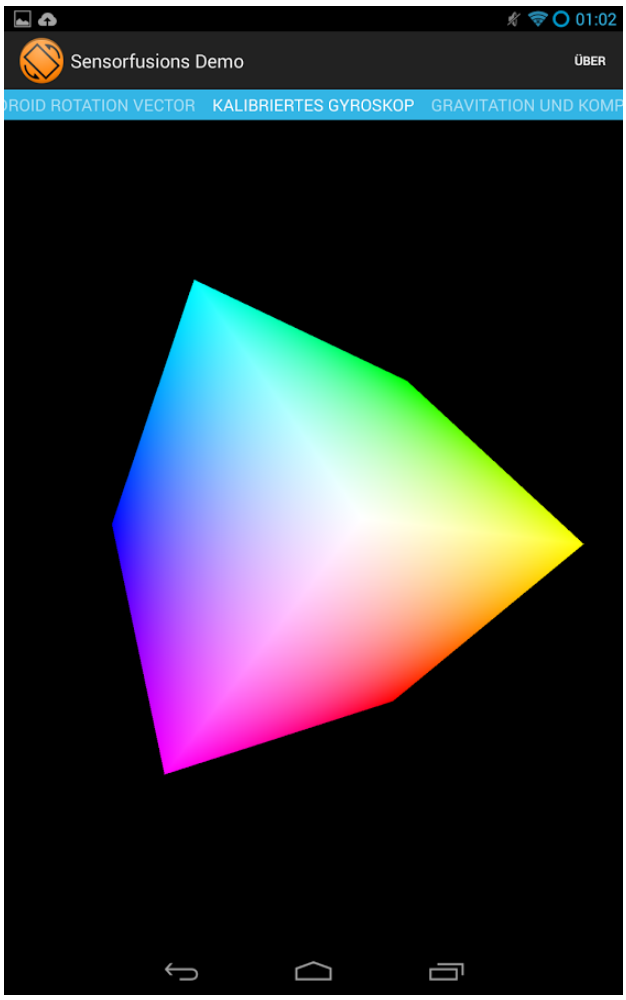

Figura 4.2: Pantalla inicial del proyecto Sensor Fusion Demo for Android

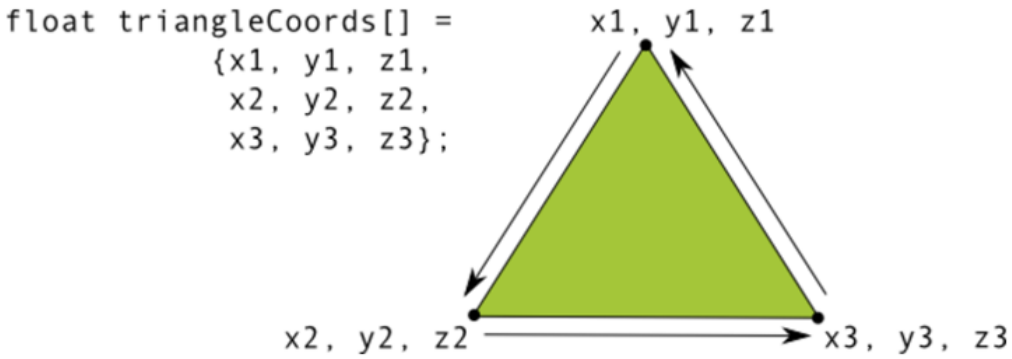

Figura 4.3: Ejemplo de cómo definir las coordenadas de un triángulo

```
public void draw(GL10 gl) {

    gl.glDisable(GL10.GL_TEXTURE_2D);
    gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
    // Establecemos el color del triángulo en modo RGBA
    gl.glColor4f(colors[0], colors[1], colors[2], colors[3]);

    // Dibujamos al revés de las agujas del reloj
    gl.glFrontFace(GL10.GL_CCW);
    // Indicamos el nº de coordenadas (3), el tipo de datos de la
    // matriz (float), la separación en la matriz de los vértices
    // (0) y el buffer con los vértices
    gl.glVertexPointer(2, GL10.GL FLOAT, 0, toiletBuff);
    // Indicamos al motor OpenGL al que le hemos pasado una matriz de
    // vértices
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glDisableClientState(GL10.GL_COLOR_ARRAY);

    // Dibujamos la superficie
    if(types==0){
        // Draw hollow circle
        gl.glDrawArrays(GL10.GL_LINE_LOOP, 1, circumferencePoints);
    } else {
        // Draw circle as filled shape
        gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 1, circumferencePoints);// 1 para no empezar en el centro
    }

    // Desactivamos el buffer de los vértices
    gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
}
```

Figura 4.4: Implementación del método draw()

para cada frame. Además, es en este método donde se implementa el movimiento del punto en función de la orientación del dispositivo.

4.1.3 Movimiento del punto central

Para poder entender cómo se mueve nuestro punto de control, debemos primero explicar brevemente la función que realizarán los ficheros OrientationVisualisationFragment.java y CubeRender.java. El primero realiza la función de runner del renderizado, además de pasarle mediante el método setOrientationProvider la matriz de orientación del dispositivo en cada momento. Mientras que el segundo actúa como interfaz de dibujo donde se detalla el código que indica lo que se debe dibujar.(ver Fig. 4.6).

A partir de estos datos, podemos operar con matrices y vectores para obtener la posición donde queremos que el punto sea dibujado. La idea general que se ha llevado a cabo ha sido la creación de un vector con la posición de nuestro punto con el dispositivo en reposo. Si multiplicamos la matriz con la nueva orientación del dispositivo móvil por el punto obtendremos el punto en el nuevo sistema de coordenadas. Una vez obtenida la nueva posición, mediante la función glTranslatef(x,y,z) trasladamos nuestro punto a la posición calculada. Cabe destacar que el punto se encuentra a una distancia de 2.5 de su centro de referencia. Por tanto, el espacio que puede abarcar es el de una esfera hueca de 2.5 de radio. Esto es importante para poder calcular correctamente la posición del punto, ya que aparentemente desde el dispositivo móvil parece que se tratase de un entorno bidimensional, pero no es así; las distancias X e Y del punto en

```
@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config) {
    // dither is enabled by default, we don't need it
    gl.glDisable(GL10.GL_DITHER);

    mCuadrado.loadGLTexture(gl, this.context);

    // clear screen in black
    gl.glClearColor(0, 0, 0, 1);
    // Configura el buffer de profundidad
    gl.glClearDepthf(1.0f);
    // Modo de renderizado de la profundidad
    gl.glEnable(GL10.GL_DEPTH_TEST);
    // Función de comparación de la profundidad
    gl.glDepthFunc(GL10.GL_LEQUAL);
    // Cómo se calcula la perspectiva
    gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST);

    port=OrientationVisualisationFragment.port;
    ip=OrientationVisualisationFragment.ip;
}
```

Figura 4.5: Implementación del método onSufaceCreated()

```
switch (getArguments().getInt(ARG_SECTION_NUMBER)) {
        case 1:
            currentOrientationProvider = new ImprovedOrientationSensor1Provider((SensorManager) getActivity()
                    .getSystemService(SensorSelectionActivity.SENSOR_SERVICE));

            break;
        case 2:
            currentOrientationProvider = new ImprovedOrientationSensor2Provider((SensorManager) getActivity()
                    .getSystemService(SensorSelectionActivity.SENSOR_SERVICE));
            break;
        case 3:
            currentOrientationProvider = new RotationVectorProvider((SensorManager) getActivity().getSystemService(
                    SensorSelectionActivity.SENSOR_SERVICE));
            break;
        case 4:
            currentOrientationProvider = new CalibratedGyroscopeProvider((SensorManager) getActivity()
                    .getSystemService(SensorSelectionActivity.SENSOR_SERVICE));
            break;
        case 5:
            currentOrientationProvider = new GravityCompassProvider((SensorManager) getActivity().getSystemService(
                    SensorSelectionActivity.SENSOR_SERVICE));
            break;
        case 6:
            currentOrientationProvider = new AccelerometerCompassProvider((SensorManager) getActivity()
                    .getSystemService(SensorSelectionActivity.SENSOR_SERVICE));
            break;
        default:
            break;
    }

    // Create our Preview view and set it as the content of our Activity
    mRenderer = new CubeRenderer(getContext());
    mRenderer.setOrientationProvider(currentOrientationProvider);
```

Figura 4.6: Implementación de la selección de sensores

```
// Get the rotation from the current orientationProvider as quaternion and matrix
orientationProvider.getQuaternion(quaternion);
orientationProvider.getRotationMatrix(matrixRot);

//Initial point coordinates
vectorOrigen.setXYZW(0,0,-2.5f,1);
vectorPointMoved.setXYZW(0,0,-2.5f,1);

//New coordinates
matrixRot.multiplyVector4fByMatrix(vectorPointMoved);

//Obtein module
moduloA=vectorPointMoved.modulo(); //nuevo punto
moduloB=vectorOrigen.modulo();//origen
vectorPointMoved.subtract(vectorOrigen,vectorDiference);
moduloC=vectorDiference.modulo(); //distancia entre ambos puntos

angulo=(float) Math.acos((moduloA*moduloA+moduloB*moduloB-moduloC*moduloC)/(2*moduloA*moduloB)); //teorema del coseno
longitud = vectorOrigen.modulo()*angulo; //arco=longitud

//final values
if(longitud>2){
    longitud=2;
}
longitud = OrientationVisualisationFragment.sensibility*longitud;
angulo = (float) Math.atan2(vectorPointMoved.getY(), vectorPointMoved.getX());

//move point
gl.glTranslatef((float) (longitud*Math.cos(angulo)),(float) (longitud*Math.sin(angulo)), 0.0f); //rota a 2 del centro
```

Figura 4.7: Esquema de la implementación del movimiento del punto de control (punto amarillo)

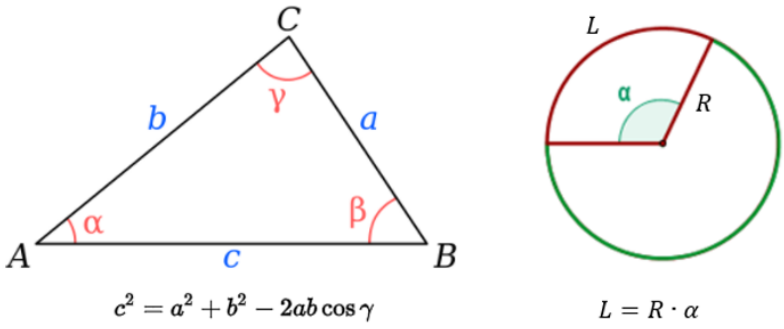

Figura 4.8: Teorema del coseno y cálculo del arco de una circunferencia

nuestra pantalla no serán líneas rectas sino arcos de una esfera de 2.5 de radio. Una vez aclarado, la implementación final se observa en la Fig. 4.7.

Para calcular la longitud del arco debemos obtener el ángulo entre los dos puntos (origen y punto resultante de la nueva orientación) y multiplicar por el radio de la esfera. Para el cálculo de ese ángulo se ha utilizado el teorema del coseno (Fig. 4.8). Observamos que la longitud se limita a un valor de 2 como se explicará más adelante.

Finalmente, comentar que el método onDrawFrame() contempla dos posibles modos de funcionamiento: un modo que sería el explicado anteriormente y otro en el que se ha implementado una función que permite calibrar la posición inicial del punto para un mejor y más cómodo manejo del dispositivo móvil. La gran diferencia entre ambos modos es el uso de un nuevo punto origen para el cálculo del ángulo anterior mediante el teorema del coseno. Este nuevo punto contiene los valores de la posición en la que se encontraba el punto después de mantener pulsada la pantalla

```
mGLSurfaceView.setOnLongClickListener(new OnLongClickListener() {

        @Override
        public boolean onLongClick(View v) {
            if(HomeActivity.i!=3){
                mRenderer.togglescreenTouched();
            }
            return true;
        }
    });
```

Figura 4.9: Implementación del clicado largo para calibrar el punto)

(mediante un método setOnLongClickListener). Por tanto, para activar la calibración se debe mantener la pulsación sobre la pantalla (Fig. 4.9)

4.2 Diseño de la interfaz de control principal

4.2.1 Interfaz gráfica

En el capítulo anterior se han introducido los conceptos necesarios para comprender cómo funciona la librería OpenGL [7] y se ha explicado la incorporación del punto central como elemento de control, obteniendo como resultado la interfaz gráfica de la Fig. 4.10. En este capítulo hablaremos del resto de elementos gráficos que se han introducido en la SurfaceView.

Se ha creado un sistema de referencia basado en cinco circunferencias concéntricas para delimitar el espacio de acción del punto central en 4 zonas que indican el 25 %, el 50 %, el 75 % y el 100 % del módulo máximo abarcable, facilitando así el control al usuario. En este caso, se ha decidido que ese valor sea el del radio de la circunferencia mayor, 2 (Fig. 4.11). Por tanto, el módulo del vector posición del punto amarillo contendrá valores de 0 a 2.

Al igual que para dibujar el punto de control, se hace uso de la clase Circle(), pero indicando mediante la variable type que se trata de un círculo hueco. A diferencia del caso anterior, queremos que las circunferencias permanezcan fijas, por lo que es importante realizar un draw() de ellas antes de trasladar o rotar la vista (Fig. 4.12).

Con la intención de facilitar aún más la comprensión de la interfaz por parte del usuario, se han pintado cada una de las circunferencias con diferentes colores siendo el verde flojo el más concéntrico y rojo el más externo, simbolizando menor y mayor velocidad. Además, se han incluido dos líneas que separan la vista en cuatro cuadrantes haciendo de ejes de coordenadas, elemento que facilita notablemente el manejo final del robot.

4.2.2 Testeo con la imagen de un dron

La interfaz gráfica del apartado anterior es la misma para todas las funciones de la aplicación salvo para una, el Testing Mode Drone. Con los modos de testeo se pretende poder visualizar el funcionamiento de los 6 diferentes modos de fusión de sensores implementados. Para ello, hacemos uso de un elemento grafico llamado ViewPager que

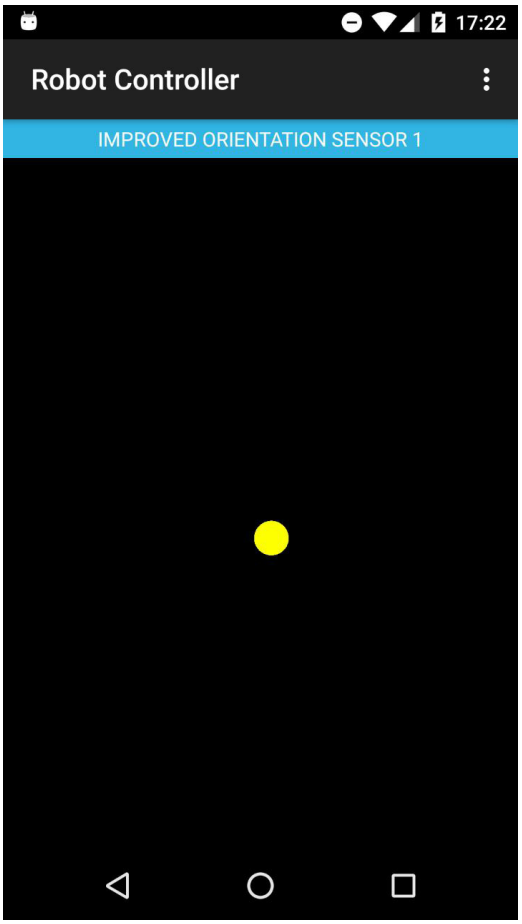

Figura 4.10: Interfaz gráfica únicamente con el punto de control

```
mPoint = new Circle(0.15f,1,amarillo);
mCircle0 = new Circle(0.02f,0,verdeLight);
mCircle1 = new Circle(0.5f,0,verde);
mCircle2 = new Circle(1f,0,verdeDark);
mCircle3 = new Circle(1.5f,0,naranja);
mCircle4 = new Circle(2f,0,rojo);
```

Figura 4.11: Inicialización de las circunferencias del fondo

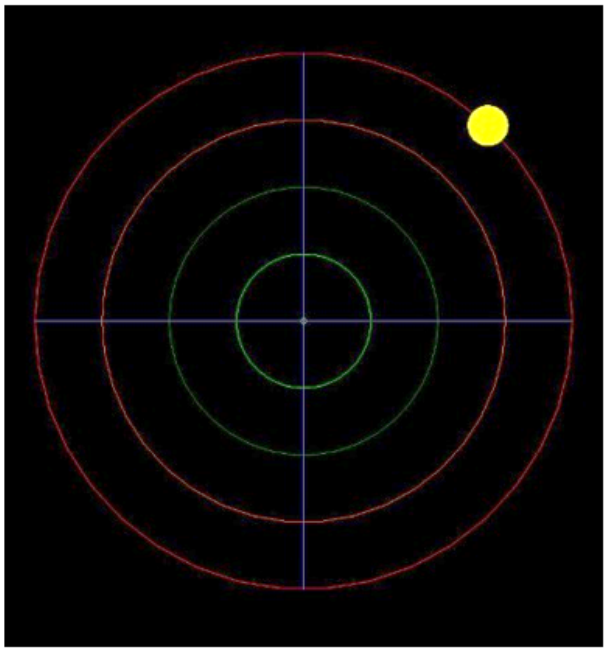

Figura 4.12: SurfaceView con todos los elementos a renderizar

```
<android.support.v4.view.ViewPager
        android:id="@+id/pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".SensorSelectionActivity" >

    <!--
    This title strip will display the currently visible page title, as well as the page
    titles for adjacent pages.
    -->

        <android.support.v4.view.PagerTitleStrip
            android:id="@+id/pager_title_strip"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="top"
            android:background="#33b5e5"
            android:paddingBottom="4dp"
            android:paddingTop="4dp"
            android:textColor="#fff" />
    </android.support.v4.view.ViewPager>
```

Figura 4.13: Declaración del ViewPager en el layout

permite al usuario deslizar pantallas con el dedo, y en nuestro caso, cambiar entre los 6 modos de fusión de sensores previamente mencionados (Fig. 4.13).

Para visualizar mejor el efecto que tiene el uso de un modo de fusión u otro se ha creado una vista adicional a la del punto amarillo, conformada por un elemento de la clase Cuadrado. Además, se ha decidido hacer uso de texturas para transformar el cuadrado en la vista aérea de un cuadricóptero (Fig. 4.14) a partir de un drawable mediante el método loadGLTexture() [8]. Para poder hacer uso de texturas 2D se debe

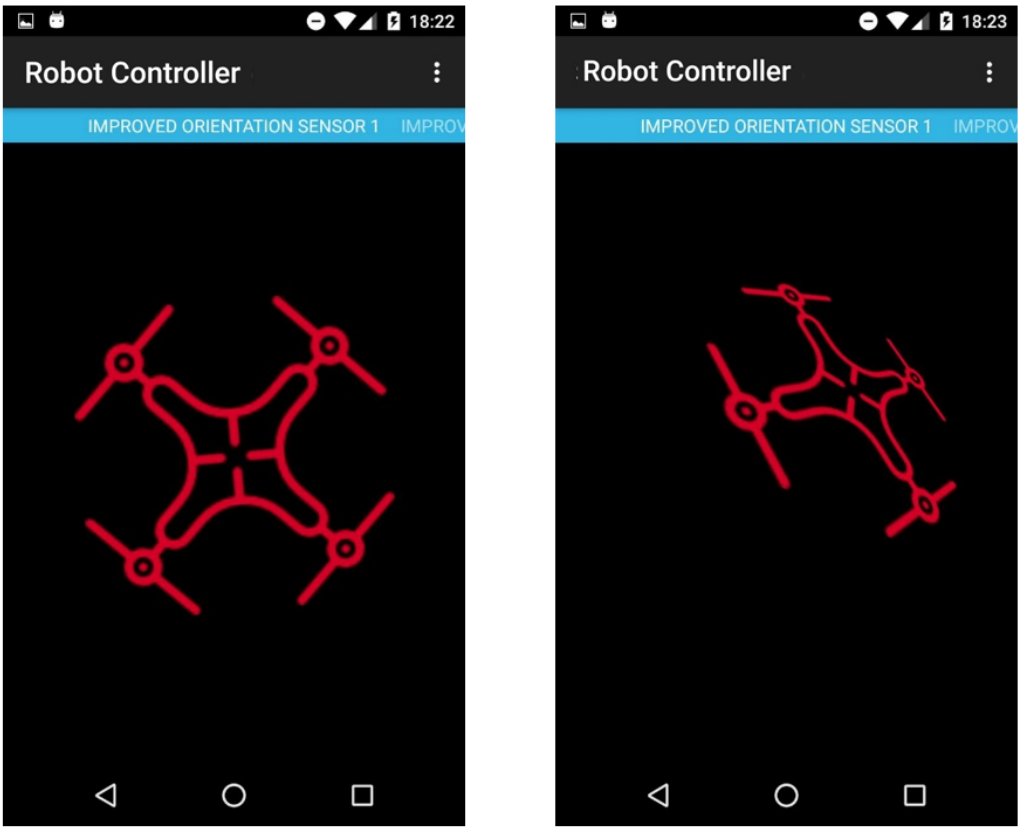

Figura 4.14: Visualización del modo de testeo de sensores con un dron

habilitar el parámetro GL10.GLTEXTURE2D del objeto.

4.2.3 Elementos extras

En este apartado se explican los elementos extras introducidos en la interfaz, en este caso, estos elementos no forman parte de la SuferceView sino que son elementos de la propia actividad (SensorSelectionActivity). Como son elementos extras, pueden ser desactivados desde ajustes, como se explicará más adelante. Estos elementos son:

- *Seekbar*: Para el control de la altura del robot aéreo.
- *Compass*: Brújula para tener referencia del campo magnético terrestre si fuera necesario.
- *Joystick*: Elemento de control que forma parte del modo de control táctil.

Seekbar

Se ha utilizado un CrystalSeekbar [9], seekbar mejorado que permite la utilización de imágenes como elemento de control de éste. De esta manera, se ha substituido el clásico punto por un drawable de un cuadricóptero, dando a entender al usuario de manera gráfica su uso, el control de la altura del robot.

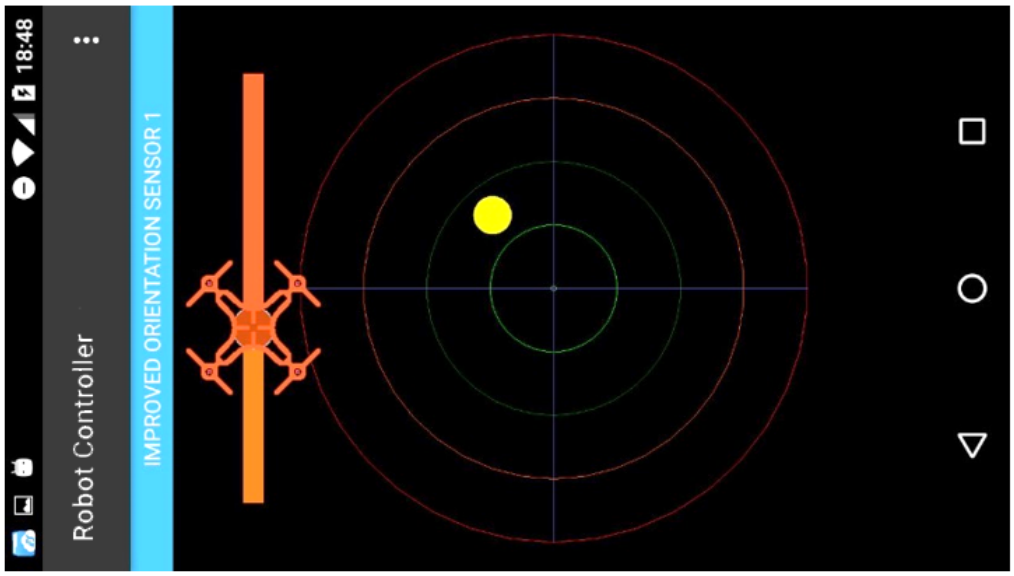

Figura 4.15: Vista con Seekback implementado

Además, este seekbar de la librería CrystalRangeSeekbar ofrece la posibilidad de crear un efecto bicolor como se observa en la Fig. 4.15. Se ha decidido no mostrar el valor de la altura porque entorpece la vista y no añade ningún valor funcional desde el punto de vista del control.

Brújula

Se ha incorporado la clase Compass [10] del proyecto *Compass* del profesor Viacheslav Lutin. Esta clase facilita la creación de una brújula, teniéndonos que preocupar únicamente por el diseño de ésta.

Para ello creamos un RelativeLayout que incluya el cuerpo y las agujas de la brujula (ambas creadas con Photoshop) como se observa en la Fig. 4.16. Una vez creada la vista, tan solo es necesario pasar por parámetro la ImageView correspondiente a las agujas para que funcione correctamente como una brújula (ver Fig. 4.17).

Joystick

El Tactile Mode o modo táctil utiliza un joystick (Fig. 4.18) para el manejo del dron a diferencia del Normal Mode que usaba los sensores de orientación del dispositivo móvil. Por tanto, el joystick se encarga del control del robot en el plano X-Y.

Se ha introducido la clase JoyStick [11] que incorpora todo lo necesario para la creación de un joystick. Para su uso se necesita inicializar un listener, elegir el color y, en caso de que se desee, elegir el diseño (no es nuestro caso)(ver Fig. 4.19).

```
<RelativeLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="right|bottom"
    android:layout_gravity="right|bottom"
    android:layout_marginRight="15dp"
    android:layout_marginBottom="15dp"
    android:id="@+id/rotation2">
    <ImageView
        android:id="@+id/main_image_dial"
        android:layout_width="80dp"
        android:layout_height="80dp"
        android:layout_centerInParent="true"
        android:contentDescription="@string/compass_dial"
        android:src="@drawable/dial3" />

    <ImageView
        android:id="@+id/main_image_hands"
        android:layout_width="53dp"
        android:layout_height="53dp"
        android:layout_centerInParent="true"
        android:contentDescription="@string/compass_hands"
        android:scaleType="fitXY"
        android:src="@drawable/hands2" />
</RelativeLayout>
```

Figura 4.16: Layout donde se implementa la brújula

```
compass = new Compass(this);
compass.arrowView = (ImageView) findViewById(R.id.main_image_hands);
```

Figura 4.17: Uso de la brújula utilizada

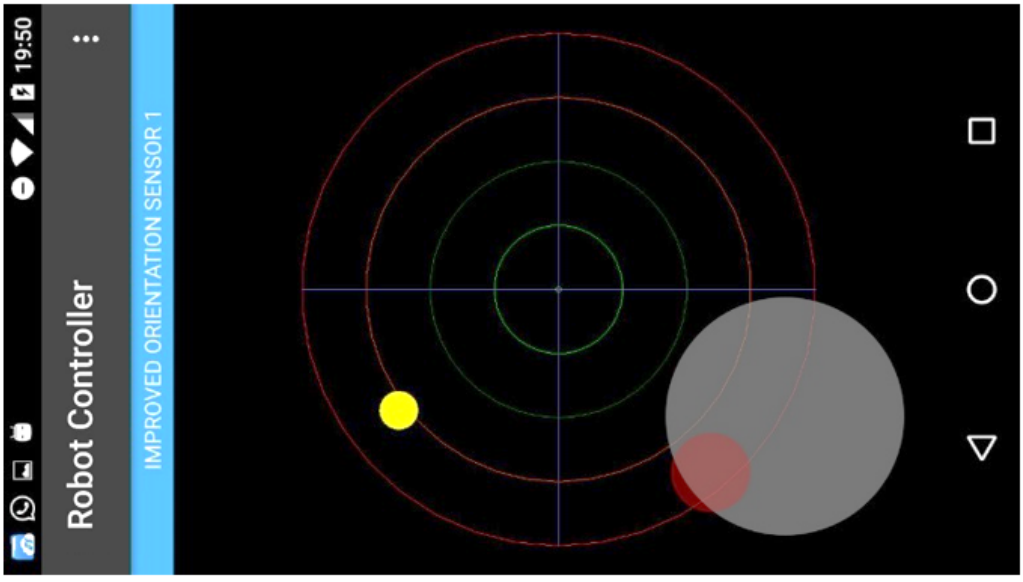

Figura 4.18: Vista del joystick

```
joy1.setListener(this);
joy1.setPadColor(Color.parseColor("#55ffffff"));
joy1.setButtonColor(Color.parseColor("#55ff0000"));
```

Figura 4.19: Implementación del Joystick

4.3 Comunicación cliente-servidor

En este apartado se explicará únicamente la implementación del cliente, la del servidor se detallará en el Capítulo 5.

4.3.1 Netbeans

La aplicación presenta una estructura cliente-servidor donde la App Android es el cliente. Para familiarizarnos con la comunicación entre un cliente y un servidor se ha utilizado el programa Netbeans. Primeramente, se han creado un cliente y un servidor sencillo en C++ para entender el funcionamiento de la clase Socket. Se ha decidido programar en C++ ya que el servidor será utilizado posteriormente en ROS, de esta manera el paso de un programa a otro será más sencillo. También se ha implementado un cliente en Java, aunque finalmente se haya utilizado otro implementado directamente en Android Studio [12]. Para que la aplicación tenga acceso a Internet se han tenido que incorporar los permisos específicos en el fichero AndroidManifest.xml.

4.3.2 Implementación del Cliente

Nuestro cliente se basa en la clase Client. Ésta contiene la implementación de un cliente multitarea en Android a partir de un AsyncTask. En éste se crea un socket usando la IP y el puerto del servidor, mediante el cual el cliente envía información al servidor.

La clase AsyncTask es una de las maneras de ejecutar tareas en segundo plano que ofrece Android, realizando la misma función que la clase Thread. La utilización de clases como las mencionadas evita que acciones como la que se desea implementar bloqueen la aplicación por utilizar el hilo principal o main thread. Las ventajas que ofrece AsyncTask es que es más sencilla y limpia de implementar para lo que se desea realizar; en nuestro caso, sólo será necesario implementar un método doInBackground y onPostExecute.

En el método principal (Fig. 4.20) se crea un Socket con la dirección del servidor y se crea un PrintStream donde se enviarán los datos de interés hacia el servidor. En nuestro caso los datos a enviar son los siguientes:

- *HomeActivity.angulo*: Contiene el valor del ángulo del punto amarillo respecto a su sistema de coordenadas.
- *HomeActivity.longitud*: Distancia a la que se encuentra el punto amarillo del origen de coordenadas.
- *HomeActivity.mode*: Indica el modo de control que se desea utilizar (Absolute, Relative o Position).

```
protected Void doInBackground(Void... arg0) {

    Socket socket = null;

    try {
        socket = new Socket(OrientationVisualisationFragment.ip, OrientationVisualisationFragment.port);

        PrintStream output = new PrintStream(socket.getOutputStream());

        output.print(data1,data2...);

        //finalizar output
        output.flush();
        output.close();

    //posibles errores y finalizar socket
    } catch (UnknownHostException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        response = "UnknownHostException: " + e.toString();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        response = "IOException: " + e.toString();
    } finally {
        if (socket != null) {
            try {
                socket.close(); //cerrar socket
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
    return null;
}
```

Figura 4.20: Implementación del método doInBackground

- *OrientationVisualizationFragment.MaxLinealVel*: Asigna que porcentaje de la velocidad máxima del robot se pretende utilizar como velocidad lineal máxima (comprende valores de 0 a 100).
- *OrientationVisualizationFragment.MaxLinealAngle*: Asigna el valor que se pretende utilizar como velocidad angular máxima (es un porcentaje sobre la máxima del robot).
- *HomeActivity.altura*: Contiene el valor de altura asignado mediante el seekbar. Su valor es un porcentaje que comprende valores de 0 a 100.
- *OrientationVisualizationFragment.alturaMax*: Altura máxima que se permite que alcance el cuadricóptero. Asigna, por tanto, el valor de altura cuando el seekback se encuentra al 100 %.
- *OrientationVisualizationFragment.areaMax*: Indica el radio máximo de la esfera que define el espacio abarcable por el dron en el modo Position.

Comentar que los datos listados se utilizan como variables globales y no se pasan por parámetro a la clase Client para evitar problemas en la comunicación. Pues como se explicará más adelante, experimentalmente se ha comprobado que, sino, el cliente envía valores incorrectos.

```
public void onDrawFrame(GL10 gl) {
    [...]
    HomeActivity.angulo=angulo;//global variables to avoid client problems
    HomeActivity.longitud=longitud;

    Client myClient = new Client(ip,port);//192.168.1.X 8080
    myClient.execute();
}
```

Figura 4.21: Implementación del método onDrawFrame

Para hacer uso de la clase Client.java ya implementada se necesita crear un cliente y ejecutarlo tal y como se muestra en la Fig. 4.21. Estas líneas de código se encuentran en el método onDrawFrame del fichero CubeRender.java pues nos interesa que para cada frame, es decir, para cada nuevo valor de posición de nuestro punto de control se envíe su valor al servidor.

4.4 Diseño y distribución general de la App

4.4.1 Arquitectura

La arquitectura de la App se basa en el diagrama de flujo de la Fig. 4.22. Para la explicación de la estructura e implementación general de la aplicación se seguirá el mismo orden que el de la figura para facilitar la comprensión.

HomeActivity.java

Al abrir la aplicación se muestra un menú de selección cuya vista pertenece al fichero HomeActivity.java. Esta clase es muy simple, se compone por 4 subprogramas que implementan el método onClickListener de cada uno de los 4 botones, y de un menú en la AppBar que nos permite acceder a los ajustes.

En la Fig. 4.23 se muestra un ejemplo de la implementación de los métodos. Como se puede observar, se asignan valores a las siguientes variables:

- *i*: Identificador del modo seleccionado.
- *joystick*: Indica si el joystick debe estar visible o no.
- *num_fused_sensors*: Determina el número de posibles sensores a mostrar. En el caso del control del robot se asigna uno único; en cambio,en el caso del testeo se asignan los 6 implementados (se seleccionan mediante un swipe horizontal).
- *seekbar*: Activa o desactiva la visibilidad de la barra de control de altura.
- *HorizontalRotation*: Se encarga de indicar si el dispositivo se encuentra en la posición de sujeción óptima para el manejo del robot (volteado 90º a la izquierda).
- *compass*: Habilita la visibilidad de la brújula.

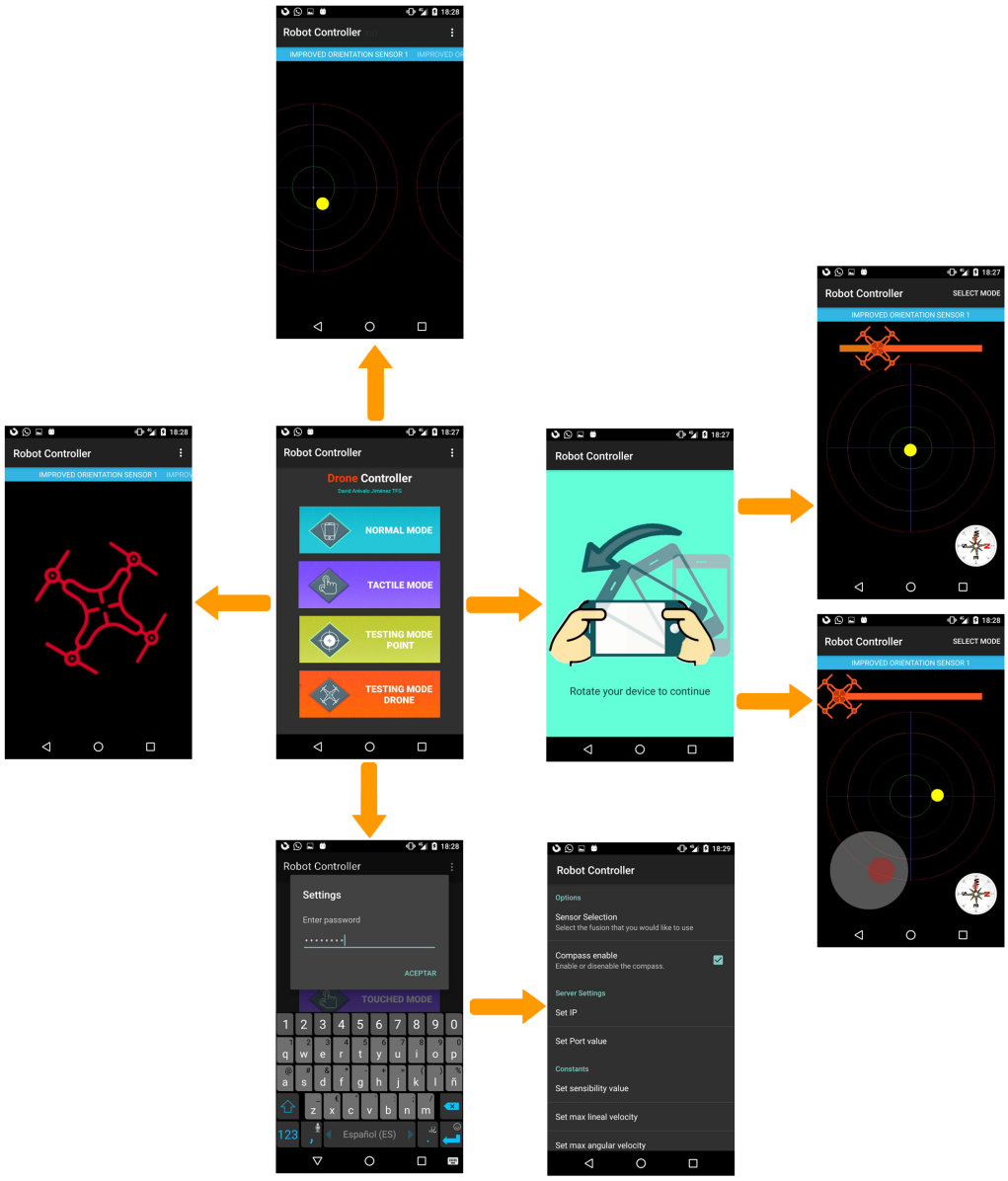

Figura 4.22: Diagrama de flujo de la App

```
private void initButtonNormal()
{
    Button button = (Button) this.findViewById(R.id.button_normal_mode);

    button.setOnClickListener(new View.OnClickListener()
    {
        public void onClick(View v)
        {
            i=1;
            num_fused_sensors=1;
            joystick=false;
            seekbar=true;
            horizontalRotation=false;
            compass=true;
            Intent intent = new Intent(HomeActivity.this,
                    Rotation.class);
            startActivity(intent);
        }
    });
}
```

Figura 4.23: Ejemplo del subprograma de acceso al control del robot desde el menú inicial

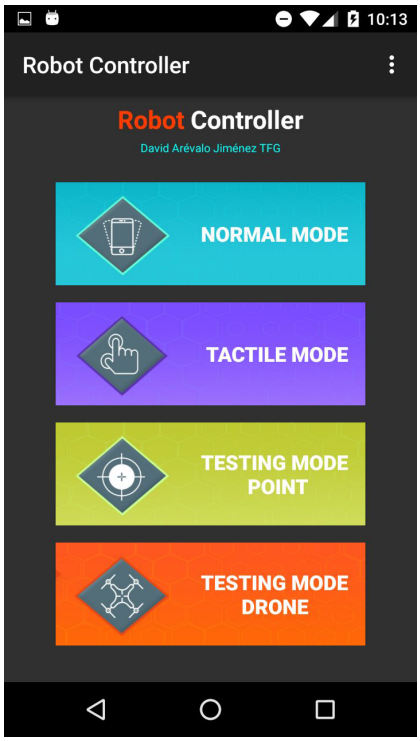

Figura 4.24: Vista del menú principal

```
private void showSettings()
{
    Intent i = new Intent(this, Settings.class);
    startActivity(i);
}
private void showHelpDialog()
{
    new MaterialDialog.Builder(this)
            .title(R.string.title)
            .content(R.string.input_content)
            .inputType(InputType.TYPE_CLASS_TEXT | InputType.TYPE_TEXT_VARIATION_PASSWORD)
            .input(R.string.input_hint, R.string.input_prefill, new MaterialDialog.InputCallback() {
                @Override
                public void onInput(MaterialDialog dialog, CharSequence input) {
                    // Do something
                    if(input.toString().equals(getString(R.string.password))){
                        showSettings();
                    } else {
                        Context context = getApplicationContext();
                        CharSequence text = "Incorrect password";
                        int duration = Toast.LENGTH_SHORT;

                        Toast toast = Toast.makeText(context, text, duration);
                        toast.show();
                    }
                }
            }).show();
}
```

Figura 4.25: Implementación del control de acceso mediante un dialogo de password

Si se selecciona la opción Settings (del menú R.menu.menu_home) correspondiente a los tres puntos de la esquina superior derecha de la Fig. 4.24, aparece un diálogo donde se solicita una contraseña para acceder a los ajustes. Como la aplicación está pensada para su uso por niños y/o gente inexperta, por seguridad se ha decidido implementar un control de acceso mediante el password *professorat*. Su implementación se muestra en la Fig. 4.25.

Settings.java

Si la contraseña es correcta accedemos al menú de ajustes implementado en el archivo Settings.java. Éste es una extensión de un PreferenceActivity [13], por lo que su comportamiento es un poco diferente al de un Activity convencional. En lugar de asignar un layout, se asigna un fichero de tipo xml que implemente una PreferenceScreen permitiendo así crear un menú como el deseado (ver Fig. 4.26) de una manera cómoda y con mejores resultados que con un layout convencional.

Para diseñar esta vista, se han creado tres PreferenceCategory para separar en categorías las diferentes opciones de ajuste (en la Fig. 4.27 se muestra un ejemplo). Los elementos utilizados para esta pantalla han sido:

- *ListPreference*: Crea una lista de elementos seleccionable.
- *CheckBoxPreference*: Permite activar o desactivar una variable mediante un checkbox.
- *EditTextPreference*: Presenta la misma funcionalidad que un edittext convencional (introducir texto).

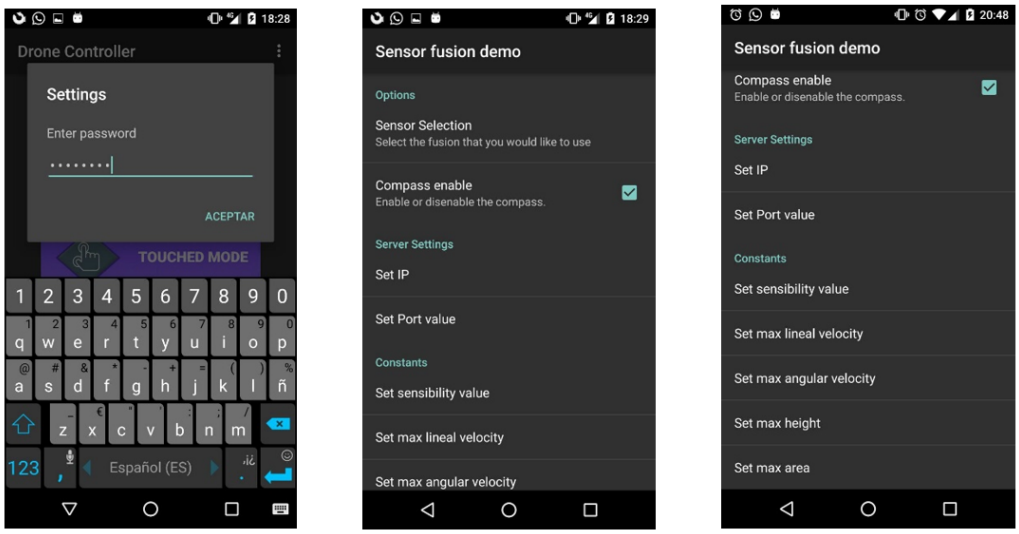

Figura 4.26: Vistas del menú de ajustes (Settings)

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">

    <PreferenceCategory android:title="Options" >
        <ListPreference     android:title="Sensor Selection"
            android:summary="Select the fusion that you would like to use"
            android:key="SensorFusion"
            android:defaultValue="1"
            android:entries="@array/listArray"
            android:entryValues="@array/listValues" />
        <CheckBoxPreference android:title="Compass enable"
            android:defaultValue="true"
            android:summary="Enable or diseable the compass."
            android:key="compass" />-->
    </PreferenceCategory>

    <PreferenceCategory android:title="Server Settings" >
        <EditTextPreference
            android:defaultValue="@string/default_value_ip"
            android:dialogMessage="Put your computer IP"
            android:dialogTitle="Set IP number"
            [...]
```

Figura 4.27: Implementación del layout de ajustes mediante una PreferenceScreen

```
public void onCreate(final Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.settings);

        final ListPreference listPreference = (ListPreference) findPreference("SensorFusion");
        listPreference.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference, Object newValue) {
                String listValue = (String) newValue;
                OrientationVisualisationFragment.SensorSelected=Integer.parseInt(listValue);
                return true;
            }
        });
        final EditTextPreference sensibility = (EditTextPreference) findPreference("sense");
        sensibility.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference, Object newValue) {
                String editValue = (String) newValue;
                OrientationVisualisationFragment.sensibility=Float.parseFloat(editValue+'f');
                return true;
            }
        });
        [...]
```

Figura 4.28: Lectura de los elementos de tipo preference

En cuanto al PreferenceActivity, se deberá implementar un setOnPreferenceChangeListener para cada uno de los elementos de nuestro menú. Dentro del Listener se leerá el nuevo valor y se actualizará en la variable que tenga asignada tal y como se muestra en la Fig. 4.28.

Rotation.java

Desde la pantalla de selección inicial podemos seleccionar 4 modos de los cuales el Normal Mode y Tactile Mode nos envían a una pantalla intermedia vinculada con el fichero Rotation.java. La vista ofrecida por esta clase muestra una imagen en la que se indica al usuario que debe colocar su dispositivo en la orientación de la Fig. 4.29 (voltear hacia la izquierda 90º) para poder acceder al modo de control seleccionado.

Más concretamente, la orientación del dispositivo móvil tiene que coincidir con la de la Fig. 4.30. Este FragmentActivity se ha creado para evitar posibles dudas del usuario sobre cómo manejar el dispositivo, ya que el sistema de coordenadas es fijo y coincide con la orientación indicada .

Para su implementación, tal y como se ha mostrado en la Fig. 4.30, se ha utilizado la clase Connect que consiste en un Listener de un booleano, myBoolean, cuyo valor inicial es *false* y cuando se cumple la condición de orientación se pone a *true*. La utilización de esta clase es útil para compartir variables entre diferentes ficheros java; en nuestro caso, esa compartición es necesaria entre los ficheros CubeRender.java (encargada de comprobar la orientación del móvil) y Rotation.java (que realiza un intent —Fig. 4.31— para cambiar a la actividad de control —SensorSelectionActivity— una vez cumplida la condición).

Cabe destacar la instrucción intent.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY) que permite volver atrás desde la actividad de control al menú de selección sin pasar por la ventana intermedia de comprobación de la orientación.

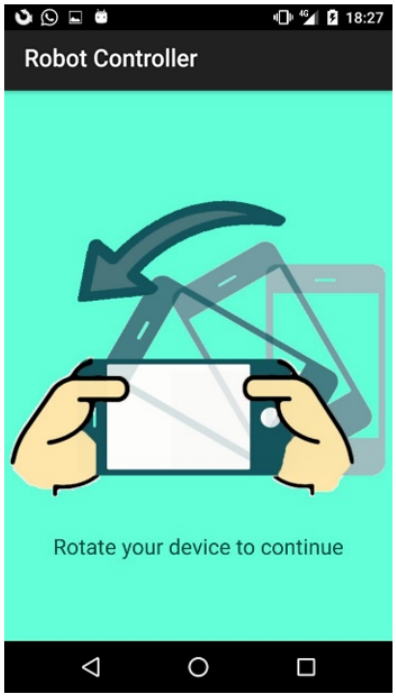

Figura 4.29: Interfaz intermedia para la correcta colocación del dispositivo móvil

```
if(!HomeActivity.horizontalRotation && vectorPointMoved.getX()<-0.5 && vectorPointMoved.getY()<0.5
&& vectorPointMoved.getY()>-0.5 && vectorPointMoved.getZ()<0){
    Connect.setMyBoolean(true);
}
```

Figura 4.30: Condición de orientación del móvil para acceder al control del robot

```
Connect.addMyBooleanListener(new com.example.david.tfg.ConnectionBooleanChangedListener5() {
    @Override
    public void OnMyBooleanChanged() {
        if (Connect.getMyBoolean()) {
            HomeActivity.horizontalRotation=true;
            Intent intent = new Intent(Rotation.this,
                    SensorSelectionActivity.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY);//evitar problemas de interacción entre ventanas
            startActivity(intent);
            finish();
        }
    }
});
```

Figura 4.31: *Intent* para acceder a la pantalla de control

SensorSelectionActivity.java

Una vez cumplida la condición de orientación, nos encontramos en la clase SensorSelectionActivity. La estructura de este archivo es muy similar a la del apartado anterior, utilizando hasta el mismo *layout*, pero siendo ahora más completo. Por este motivo hemos decidido explicarlo con más profundidad.

Esta clase java utiliza un elemento llamado ViewPager que permite crear diferentes páginas dentro de una misma actividad y desplazarse entre ellas mediante *swipes* laterales. En cada Page se desea implementar una de las técnicas de fusión de sensores creados, a diferencia de los modos de control (Normal y Tactile mode) donde sólo se utilizará una única página con una de las técninas de fusión (por defecto ImprovedOrientationSensor1Provider).

Por lo tanto, se debe implementar un adaptador que permita seleccionar una técnica de fusión de sensores diferente para cada una de las páginas o pages. Para ello, se ha implementado la clase SectionPagerAdapter. Esta clase es una extensión de un FragmentPagerAdapter donde se inicializa un Fragment de la clase OrientationVisualisationFragment (se explicará con más detalle en el próximo apartado) con el modo de fusión de sensores seleccionado mediante la variable args del método getItem() de la Fig. 4.32.

Finalmente se ha creado un PagerTitleStrip que permite que cada página tenga el nombre del modo de fusión de sensores utilizado como cabecera (Fig. 4.33) mediante el método getPageTitle.

El uso de la clase SectionPagerAdapter es relativamente simple, primero debemos crear un elemento de esta clase y un ViewPager. Una vez creados se realiza un setAdpter() al ViewPager tal y como se muestra en la Fig. 4.34.

```
SectionsPagerAdapter mSectionsPagerAdapter;
ViewPager mViewPager;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_sensor_selection);

    // Create the adapter that will return a fragment for each of the three
    // primary sections of the app.
    mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());

    // Set up the ViewPager with the sections adapter.
    mViewPager = (ViewPager) findViewById(R.id.pager);
    mViewPager.setAdapter(mSectionsPagerAdapter);

    [...]
}
```

Figura 4.34: Uso del adaptador en la ViewPager

OrientationVisualisationFragment.java

Siendo uno de los archivos más importantes del proyecto, se encarga de la selección de las técnicas de fusión de sensores creadas de la carpeta orientationProvider y de crear la GLSurfaceView con su correspondiente Render (tratado en el Capítulo 1.4). La selección del modo de fusión de sensores se realiza mediante un elemento de la clase

```
public class SectionsPagerAdapter extends FragmentPagerAdapter {

    public SectionsPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public Fragment getItem(int position) {
        // getItem is called to instantiate the fragment for the given page.
        Fragment fragment = new OrientationVisualisationFragment();
        Bundle args = new Bundle();
        if(getCount()==1){
            position=OrientationVisualisationFragment.SensorSelected-1;//nº posibles sensores
        }
        args.putInt(OrientationVisualisationFragment.ARG_SECTION_NUMBER, position + 1);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public int getCount() {
        // Show 6 total pages.
        return HomeActivity.num_fused_sensors;
    }

    @Override
    public CharSequence getPageTitle(int position) {
        Locale l = Locale.getDefault();

        if(getCount()==1){
            position=OrientationVisualisationFragment.SensorSelected-1;
        }
        //Section tittle
        switch (position) {
        case 0:
            return getString(R.string.title_section1).toUpperCase(l);
        case 1:
            return getString(R.string.title_section2).toUpperCase(l);
        case 2:
            [...]
        }
        return null;
    }
}
```

Figura 4.32: Implementación del FragmentPagerAdapter

OrientationProvider que implementa un SensorEventListener. Esta clase extrae el valor de orientación del dispositivo móvil de acuerdo con la técnica de fusión seleccionada en forma de matriz y cuaternión gracias a los métodos que tiene implementados. Por tanto, mediante un *switch case* podremos seleccionar el modo de fusión que se desee utilizar tal y como se observa en la Fig. 4.35.

```
//Sensor Selection
switch (getArguments().getInt(ARG_SECTION_NUMBER)) {
    case 1:
        currentOrientationProvider = new ImprovedOrientationSensor1Provider((SensorManager) getActivity()
                .getSystemService(SensorSelectionActivity.SENSOR_SERVICE));

        break;
    case 2:
        currentOrientationProvider = new ImprovedOrientationSensor2Provider((SensorManager) getActivity()
                .getSystemService(SensorSelectionActivity.SENSOR_SERVICE));
        break;
    case 3:
        [...]
}
```

Figura 4.35: Selección de la técnica de sensores a aplicar

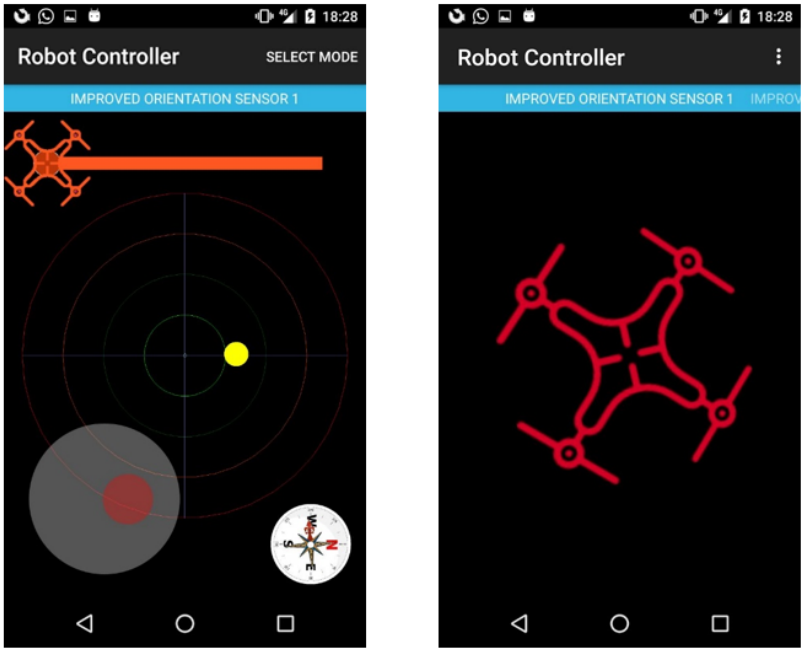

Figura 4.33: Ejemplo de dos actividades con una y varias páginas

Una vez obtenido y guardado el valor de orientación de nuestro dispositivo en la variable currentOrientationProvider, se crea la GLSurfaceView y su Render. Es necesario pasar por parámetro la orientación del dispositivo al Render mediante el método setOrientationProvider() para poder utilizarla en dicha clase. Una vez renderizada la escena, se retorna como vista por el método onCreateView del Fragment, tal y como se muestra en la Fig. 4.36.

```
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    [...]
    // Create our Preview view and set it as the content of our Activity
    mRenderer = new CubeRenderer(getContext());
    mRenderer.setOrientationProvider(currentOrientationProvider);

    mGLSurfaceView = new GLSurfaceView(getActivity());
    mGLSurfaceView.setEGLConfigChooser(8, 8, 8, 8, 16, 0);
    mGLSurfaceView.setRenderer(mRenderer);

    mGLSurfaceView.setOnLongClickListener(new OnLongClickListener() {

        @Override
        public boolean onLongClick(View v) {
            if(HomeActivity.i!=3){
                mRenderer.togglescreenTouched();
            }
            return true;
        }
    });
    return mGLSurfaceView;
}
```

Figura 4.36: Implementación del método onCreateView

4.4.2 Traducción de idiomas

Como último paso en el desarrollo de la App, se ha adaptado la aplicación a tres idiomas diferentes: Catalán, Castellano e Inglés. Inicialmente, la aplicación móvil se creó en Inglés con la intención de hacerla entendible para un mayor número de personas. Además, los pocos textos que presenta y su simplicidad no suponía ningún problema de comprensión para su correcto manejo. Sin embargo, dado que su uso está pensado para la utilización en la Universitat de les Illes Balears, cuyas lenguas predominantes son el Catalán y el Castellano, se ha creído conveniente la adaptación de la aplicación móvil a dichas lenguas.

Implementación

Para la implementación de idiomas en aplicaciones móviles existen varias alternativas, pero la recomendada es la utilización de diferentes ficheros Strings.xml en función del idioma del dispositivo (ver Fig. 4.37). De esta manera se tienen agrupados todos los textos, facilitando la incorporación de nuevos idiomas.

Figura 4.37: Un fichero strings.xml por cada idioma soportado

Se ha creado una lista con las 3 traducciones disponibles en la sección de ajustes. Su implementación se realiza de la misma manera que para el resto de elementos del Preference, con la diferencia que, además de guardar el valor seleccionado, debemos cambiar el idioma de la aplicación para que utilice los strings correspondientes. Para ello, se hace uso de las clases Locale y Configuration que permiten establecerlo tal y como se muestra en la Fig. 4.38.

Se observa la utilización de un método llamado Refresh() (Fig. 4.39), que se encarga de cerrar y volver a abrir el activity principal para que el cambio de idioma sea instantáneo.

```
public static void refresh(){
    Intent intent=new Intent();
    intent.setClass(HomeActivity.home, HomeActivity.home.getClass());
    HomeActivity.home.finish();
    HomeActivity.home.startActivity(intent);
}
```

Figura 4.39: Implementación del método Refresh()

Por último, ha sido necesario la creación de un SharedPreference, es decir, de un archivo donde guardar el idioma seleccionado, para conservar el lenguaje deseado una

```
final ListPreference languagePreference = (ListPreference) findPreference("language");
    languagePreference.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
        @Override
        public boolean onPreferenceChange(Preference preference, Object newValue) {
            String listValue = (String) newValue;
            OrientationVisualisationFragment.language=Integer.parseInt(listValue);
            Locale localizacion;
            switch (OrientationVisualisationFragment.language){
                case 1:
                    localizacion = new Locale("en", "EN");
                    break;
                case 2:
                    localizacion = new Locale("es", "ES");
                    break;
                case 3:
                    localizacion = new Locale("ca", "CA");
                    break;
                default:
                    localizacion = new Locale("en", "EN");
                    break;
            }
            Locale.setDefault(localizacion);
            Configuration config = new Configuration();
            config.locale = localizacion;
            getActivity().getBaseContext().getResources().updateConfiguration
            (config, getActivity().getBaseContext().getResources().getDisplayMetrics());

            final SharedPreferences prefs = getActivity().getSharedPreferences
            ("SettingsDroneControl", Context.MODE_PRIVATE);

            SharedPreferences.Editor editor = prefs.edit();
            editor.putInt("language", OrientationVisualisationFragment.language);
            editor.commit();

            refresh();
            return true;
        }
    });
```

Figura 4.38: Implementación del cambio de idioma

vez cerrada la App. Este archivo está oculto y tiene el nombre de SettingsDroneControl. Al iniciar la aplicación se leerá el valor que contiene; en el caso de que no exista este fichero se utilizará el mismo idioma que el del teléfono móvil (Fig. 4.40); si el idioma es cambiado manualmente desde la sección de ajustes, el nuevo idioma se guardará y se realizará un Refresh. Para que el cambio se haga efectivo es necesario el uso del método *commit()* como se muestra al final de la Fig. 4.38.

En la Fig. 4.38 se muestra un ejemplo de diferentes vistas en cada uno de los tres idiomas disponibles.

```
final SharedPreferences prefs = getSharedPreferences("SettingsDroneControl",Context.MODE_PRIVATE);
    language=prefs.getInt("language",language());
    Locale localizacion;
    switch (language){
        case 1:
            localizacion = new Locale("en", "EN");
            break;
        case 2:
            localizacion = new Locale("es", "ES");
            break;
        case 3:
            localizacion = new Locale("ca", "CA");
            break;
        default:
            localizacion = new Locale("en", "EN");
            break;
    }
    Locale.setDefault(localizacion);
    Configuration config = new Configuration();
    config.locale = localizacion;
    this.getBaseContext().getResources().updateConfiguration(config,
    this.getBaseContext().getResources().getDisplayMetrics());
```

Figura 4.40: Selección de idioma

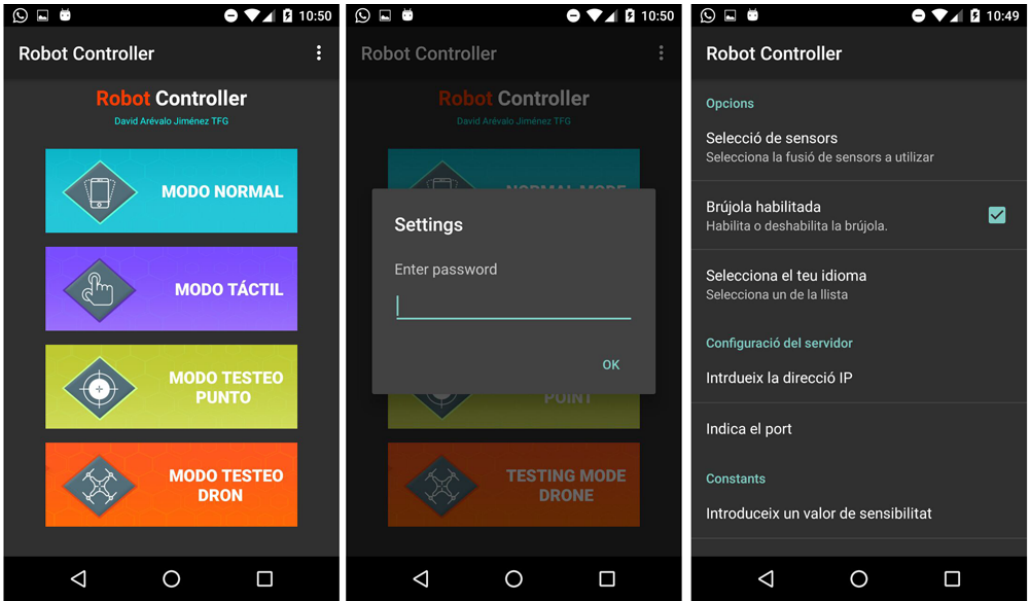

Figura 4.41: Ejemplo de diferentes vistas de la App en diferentes idiomas

CAPÍTULO 5

SIMULACIONES EN ROS

Este capítulo abarca el trabajo realizado con todo lo relacionado con las simulaciones en ROS. Se divide en tres grandes secciones por orden de desarrollo temporal: el simulador TurtleSim y simulaciones con robots terrestres y aéreos en MORSE.

5.1 Concepto de package en ROS

ROS [14] es un meta sistema operativo de código abierto que proporciona los servicios y librerías para el control de robots haciéndolo idóneo para nuestro proyecto. Sin embargo, trabajar con ROS es bastante diferente de lo habitual lo que lo hace difícil de manejar en un principio. Por este motivo ha sido necesario realizar los tutoriales introductorios a ROS [15]. A continuación se describe brevemente el concepto package (paquete) de ROS, dado que es un elemento esencial del desarrolo de software en ROS. Cada uno de los packages creados tiene la siguiente estructura (Fig. 5.1):

- *Carpeta SRC*: Contiene el archivo cpp con el programa.
- *CMakeLists.txt*: Se declaran los paquetes que se usan en el programa y se le crea y asigna un nodo (Fig. 5.2).
- *Package.xml*: Se declaran las herramientas de construcción de los paquetes externos (Fig. 5.2).

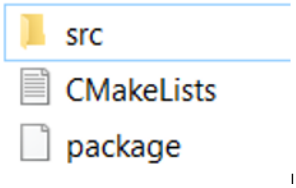

Figura 5.1: Archivos del package

```
find_package(catkin REQUIRED COMPONENTS
    roscpp
    rospy
    std_msgs
    geometry_msgs
    sensor_msgs
    message_generation
)

add_executable(turtlesim_controller_node
               src/robot_controller.cpp)
target_link_libraries(turtlesim_controller_node
                      ${catkin_LIBRARIES})
add_dependencies(turtlesim_controller_node
                 beginner_tutorials_gencpp)
```

```
<buildtool_depend>catkin</buildtool_depend>

  <buildtool_depend>roscpp</buildtool_depend>
  <buildtool_depend>rospy</buildtool_depend>
  <buildtool_depend>std_msgs</buildtool_depend>
  <buildtool_depend>geometry_msgs</buildtool_depend>
  <buildtool_depend>sensor_msgs</buildtool_depend>
  <buildtool_depend>message_runtime</buildtool_depend>


  <run_depend>roscpp</run_depend>
  <run_depend>rospy</run_depend>
  <run_depend>std_msgs</run_depend>
  <run_depend>geometry_msgs</run_depend>
  <run_depend>sensor_msgs</run_depend>
  <run_depend>message_runtime</run_depend>
```

Figura 5.2: Declaración de interés de CMakeList.txt y Package.xml

5.2 Estrategia y objetivos

Dada la complicación que presenta ROS inicialmente, se ha optado por plantear objetivos intermedios de menor a mayor dificultad y complejidad por tal de ir entendiendo el lenguaje y ser capaz de lograr el objetivo final, controlar un robot móvil desde el dispositivo android. A continuación, se listan los pasos a seguir:

- *Turtlesim*: Se ha planteado como primer objetivo ser capaz de entender el funcionamiento de la comunicación en ROS mediante Topics. Para ello haremos uso del Turtlesim, un simulador bidimensional mas simple.
- *MORSE*: Una vez establecida la conexión del dispositivo móvil con ROS se plantea sustituir el simulador Turtlesim por MORSE. De igual manera se plantea como primer objetivo controlar un robot terrestre y, una vez conseguido, implementarlo para un cuadricóptero.
  - *ATRV*: Robot terrestre utilizado.
  - *Quadrotor*: Robot aéreo utilizado para las simulaciones finales.

Para ejecutar la aplicación, se necesita ejecutar el comando Roscore [16], iniciar TurtleSim o MORSE (dependiendo del simulador que se quiera utilizar) y ejecutar el comando Rosrun para el nodo recién creado, turtlesim_controller (ver Fig. 5.3).

5.3 TurtleSim

Para la realización del programa implementado en TurtleSim se han seguido los tutoriales de la documentación de la wiki de ROS [17]. Los Topics utilizados han sido:

- *turtleX/cmd_vel*: Topic que asigna la velocidad lineal y angular de la tortuga. Por tanto, se publicará el valor calculado de velocidades. Es de tipo geometry_msgs/Twist.
- *turtleX/pose*: Topic que devuelve la posición de la tortuga en el entorno de navegación. Es de tipo turtlesim/Pose y contiene la posición x e y, la orientación en el eje Z (theta) y las velocidades actuales. Nuestro programa deberá suscribirse a este Topic.

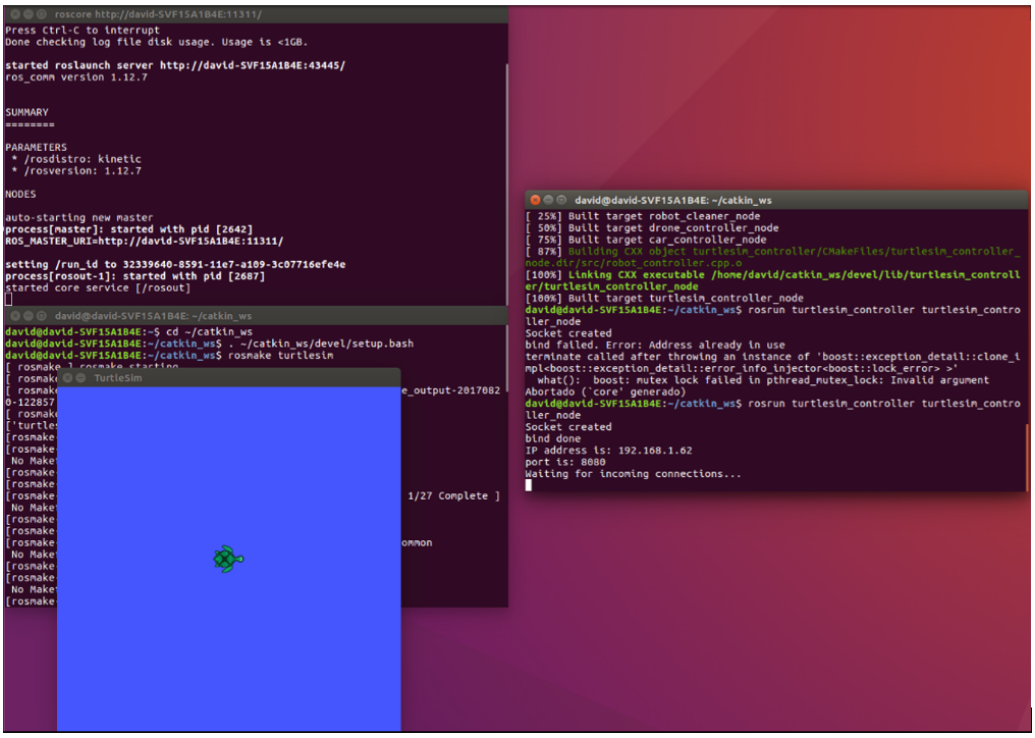

Figura 5.3: Ejecución de Roscore, TurleSim y el nodo turtlesim_controller

Inicialmente, se siguieron escrupulosamente los tutoriales ya mencionados. Sin embargo, los resultados obtenidos no eran los deseados en cuanto al control, pues la posición indicada presentaba demasiado error. Por este motivo se modificó el programa encargado de la rotación del robot introduciendo una lectura de la variable theta de su orientación como comprobación, mejorando los resultados notablemente. El programa que se implementó es el posteriormente denominado Absolute Mode; éste indica la dirección a la que debe moverse la tortuga en coordenadas absolutas como se muestra en la Fig. 5.4.

5.3.1 Comunicación con Topics

Como ya se ha explicado en el Capítulo 1, la comunicación en ROS se basa en suscriptores y publicadores de Topics. En la Fig. 5.5 se muestra el diagrama de comunicación de este programa en TurtleSim.

Se puede comprobar que nuestro nodo turtlesim_controller publica en el Topic cmd_vel y está suscrito al Topic Pose. A su vez, nuestra tortuga turtle1 envía y recibe los valores de TurtleSim. Si analizamos el código, lo primero es declarar las variables globales y los métodos como se muestra en la Fig. 5.6.

Se ha creado un Publisher y un Subscriber conjuntamente con una variable de tipo Pose denominada turtlesim_pose donde se introducirán los valores de pose_subscriber. Una vez creadas estas variables, debemos inicializarlas en el *main*, es decir, asociarlas con los Topics deseados (Fig. 5.7). Para el caso del subscriptor debemos crear un Call-

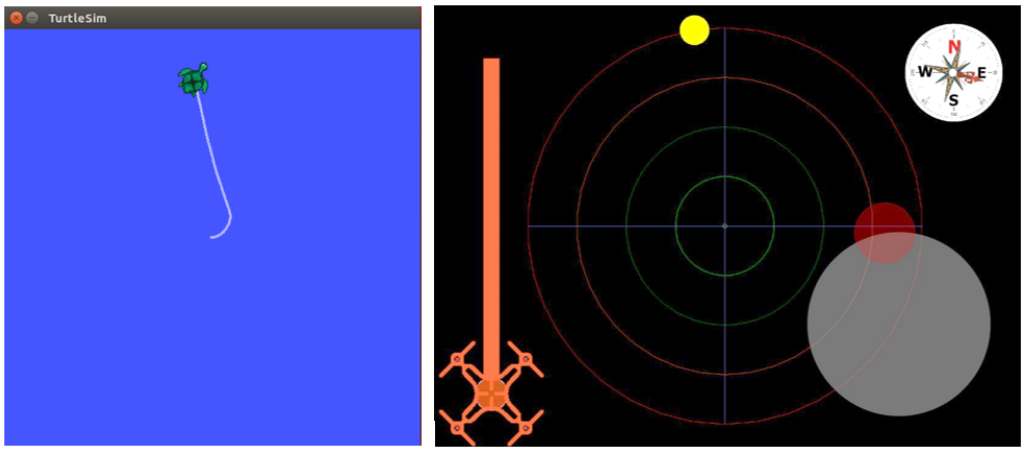

Figura 5.4: Demostración del control en TurtleSim del modo ABSOLUTE

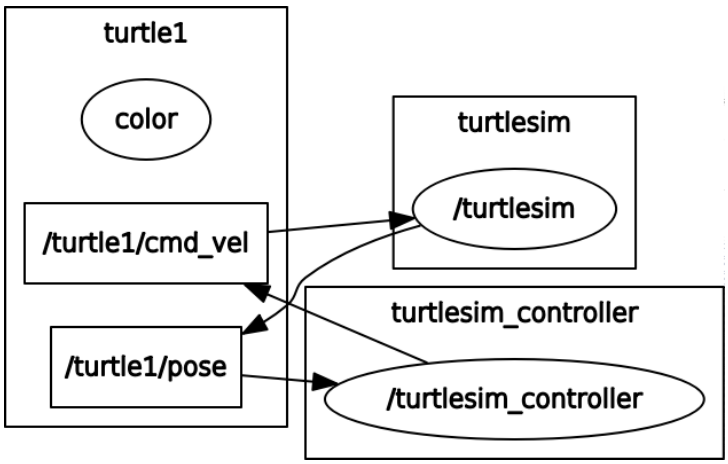

Figura 5.5: Gráfico de la comunicación mediante Topics entre ROS y MORSE

```
//Global variables
ros::Publisher velocity_publisher; //create a publisher
ros::Subscriber pose_subscriber; //suscribe to Pose
turtlesim::Pose turtlesim_pose;

const double PI = 3.14159265359;
double relative_angle; double linear_speed;


//metodos
void rotateABS (double angular_speed, double angle, double linear_speed);
double degrees2radians(double angle_in_degrees);
void poseCallback(const turtlesim::Pose::ConstPtr & pose_message);
```

Figura 5.6: Declaración de variables globales y métodos en TurtleSim

```
//subscribers and publishers
velocity_publisher = n.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 100);
pose_subscriber = n.subscribe("/turtle1/pose", 10, poseCallback);
```

Figura 5.7: Asignación de Topics al publisher y subscriber

```
void poseCallback(const turtlesim::Pose::ConstPtr & pose_message){
    turtlesim_pose.x=pose_message->x;
    turtlesim_pose.y=pose_message->y;
    turtlesim_pose.theta=pose_message->theta;
}
```

Figura 5.8: Lectura de la posición de la tortuga

back que asigne el valor de Pose a la variable turtlesim_pose cada vez que se actualice (Fig. 5.8).

5.3.2 Control del robot

En este apartado se explicará el programa implementado para el control del robot. Para el simulador Turtlesim se ha implementado un único algoritmo de control, el método rotateABS(). Recordar que, en este método, el vector posición del punto de control amarillo representa el vector velocidad lineal en coordenadas absolutas. Primero se crea una variable de tipo geometry_msgs::Twist y con nombre vel_msg que utilizaremos para asignar las velocidades al robot. Inicializamos estas variables a 0 exceptuando la componente X correspondiente a la velocidad lineal, que tomará el valor del módulo del vector posición del punto amarillo enviado por el cliente. A nivel general, dado que el robot presenta ruedas unidireccionales, el programa hace rotar al robot mientras se desplaza hasta que la distancia entre su ángulo theta, proporcionado por el subscriptor Pose, y el ángulo deseado (el enviado por el cliente) es menor que 0; esta diferencia de ángulos se ha llamado end_angle. Para que se cumpla esta condición deberemos identificar, para los diferentes casos posibles, el valor de la variable end_angle:

- Sentido antihorario de 0 a 6.28 rad (clockwise=0)
- Paso del cuarto cuadrante al primero (clockwise=1)
- Sentido horario de 6.28 a 0 rad (clockwise=2)
- Paso del primer cuadrante al cuarto (clockwise=3)

Una vez diferenciados estos cuatro casos mediante la variable clockwise, calculamos el valor correcto de end_angle para que se cumpla la condición de que end_angle<0 cuando el robot se orienta según el ángulo deseado (Fig. 5.9).

El algoritmo además es capaz de detectar el sentido de rotación para que la orientación deseada se alcance lo más rápidamente posible. Este sentido de giro se calcula en el mismo bloque que la variable clockwise antes del bucle de comprobación del ángulo, pues es necesario publicar la velocidad (velocity_publisher.publish(vel_msg)) para que el robot empiece a rotar y desplazarse. Se puede observar en la Fig. 5.10 que el

```
//Calculate correct end_angle
}if (big2small_angle && clockwise==2){ //horario
 end_angle = -(relative_angle-positive_theta);
}else if (clockwise==1 && big2small_angle && relative_angle<positive_theta && !firstTime){ //neg a pos antihorario
 end_angle = -(relative_angle-positive_theta);
}else{
 if(clockwise==3){
     if(positive_theta<degrees2radians(180)){
         end_angle = (relative_angle-positive_theta); //paso de primero al cuarto cuadrante
     }else{
         end_angle = positive_theta-relative_angle;
     }
 }else{
     end_angle = relative_angle-positive_theta;//antihorario
     firstTime=1;
 }
}
```

Figura 5.9: Cálculo de la condición de salida del bucle

```
//Calculate the type of rotation
 if (relative_angle>positive_theta) //camino corto
     if((theta360-relative_angle)<(relative_angle-positive_theta))
         vel_msg.angular.z =-abs(angular_speed), clockwise=3; //de angulo positivo a negativo horario
     else
           vel_msg.angular.z =abs(angular_speed), clockwise=0;//antihorario de 0 2pi no de 2pi a 0
     else if ((relative_angle360-positive_theta)<(positive_theta-relative_angle))
     vel_msg.angular.z =abs(angular_speed), clockwise=1;//de negativo a positivo antihorario
else
     vel_msg.angular.z =-abs(angular_speed), clockwise=2; //horario de 2pi a 0
```

Figura 5.10: Cálculo del sentido de giro del robot en TurtleSim

sentido de giro se traduce en asignar un valor positivo o negativo a la componente Z de la velocidad angular de nuestro robot.

5.3.3 Servidor

En este apartado se trata todo lo relacionado con la implementación del servidor en ROS, entendiendo por servidor el programa que se comunica con la aplicación del smartphone. Dentro de turtlesim_controller se ha creado un subprograma llamado Server() donde se encuentra el código de creación del servidor para liberar y simplificar el main. A continuación, se detalla la estructura del código:

1. Primero creamos el socket perteneciente al servidor con la dirección IP y el puerto donde queremos conectar el server. En nuestro caso, tendrá por defecto la siguiente forma: 192.168.1.X y 8080 (Fig. 5.11).

2. Comprobamos que la dirección IP está disponible mediante la instrucción bind() (Fig. 5.11).

3. Mantenemos el socket en escucha por si algún cliente desea conectarse (Fig. 5.11).

4. Una vez la petición del cliente es recibida, se ejecuta la instrucción accept(). En el caso de que la conexión sea posible se creará un nuevo thread en la misma dirección IP pero diferente puerto por tal de dejar libre el thread principal para nuevos clientes. A esta nueva conexión se le asigna un connection_handler (Fig. 5.12), subprograma que se ocupa de recibir los datos del cliente y almacenarlos.

```
//Create socket
socket_desc = socket(AF_INET , SOCK_STREAM , 0);
if (socket_desc == -1){
    printf("Could not create socket");
}
puts("Socket created");

//Prepare the sockaddr_in structure
server.sin_family = AF_INET;
server.sin_addr.s_addr = inet_addr("192.168.1.46");
server.sin_port = htons( 8080 );

//Bind
if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0){
    //print the error message
    perror("bind failed. Error");
    return 1;
}
puts("bind done");

//Listen
listen(socket_desc , 3);
puts("Waiting for incoming connections...");
c = sizeof(struct sockaddr_in);
```

Figura 5.11: Implementación del Servidor

```
//Accept and incoming connection
while( (client_sock = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c)) ){
    puts("Connection accepted");
    [...]
    pthread_t sniffer_thread;
    new_sock = (int *)malloc(1);
    *new_sock = client_sock;

    if( pthread_create( &sniffer_thread , NULL ,  connection_handler , (void*) new_sock) < 0){
        perror("could not create thread");
        return 1;
    }
    [...]
}
```

Figura 5.12: Aceptación de una conexión y creación de un nuevo thread

En relación al subprograma connection_handler, (Fig. 5.13) cabe destacar que es el encargado de leer, mediante la instrucción recv(), el output del cliente. Los datos se almacenan en el buffer client_message[] y, posteriormente, son tratados para obtener los valores del ángulo y módulo de la posición del punto de control amarillo del cliente. Una vez leídos estos datos, limpiamos los buffers utilizados y ejecutamos nuestro subprograma rotateABS(). Por último liberamos el socket.

Para simplificar la lectura de datos desde el servidor, el cliente envía los valores de ángulo y longitud con un formato fijo de 6 decimales utilizando el carácter "␣"como elemento de separación de datos y así poder enviarlos dentro de un único mensaje. Además, se han adaptado los valores del ángulo al sistema de coordenadas utilizado por el servidor, comprendiendo valores entre 0 y 2pi y volteando 90º el sistema de coordenadas para que así coincidan (ver Fig. 5.14). La implementación de este proceso se explica gráficamente en la Fig. 5.15 donde se muestran los sistemas de coordenadas originales para el Normal Mode y el Tactile Mode, y el sistema deseado y finalmente implementado.

```
void *connection_handler(void *socket_desc){
    [...]
    float angulo=0,longitud=0;

    while( (read_size = recv(sock , client_message , 2000 , 0)) > 0 ){
        //Read angle
        while(client_message[i]!=' '){
            angulo_message[i]=client_message[i];
            i++;
        }
        //string to float
        angulo=angulo_message[0] - '0';
        angulo=angulo+0.1*(angulo_message[2] - '0');
        angulo=angulo+0.01*(angulo_message[3] - '0');
        angulo=angulo+0.001*(angulo_message[4] - '0');
        [...]
        //Read longitud
        while(client_message[i]!=' '){
            longitud_message[e]=client_message[i];
            i++;
            e++;
        }
        i=0;
        e=0;

        //String to float
        longitud=longitud_message[0] - '0';
        longitud=longitud+0.1*(longitud_message[2] - '0');
        longitud=longitud+0.01*(longitud_message[3] - '0');
        [...]

        //clean the buffers
        memset(client_message, 0, strlen(client_message));
        [...]

        relative_angle=(double)angulo;
        linear_speed=(double)longitud*5/2;

        rotateABS (degrees2radians(180));
    }
```

Figura 5.13: Implementación de la lectura de mensajes enviados por el Cliente

5.3.4 Demostraciones

En este último apartado de TurtleSim se mostrarán algunas imágenes de simulaciones realizadas para observar la precisión del control (Fig. 5.16). En todas las imágenes la tortuga ha sido controlada desde la App.

Analizando las imágenes de izquierda a derecha y de arriba a bajo, observamos que la primera imagen muestra una demostración donde el robot realiza un camino lineal recorriendo las cuatro esquinas del entorno. Se concluye que lo ha realizado de manera controlada y sin ningún tipo de zig-zag en los tramos rectos. La segunda imagen muestra un recorrido con un número mayor de curvas en un menor espacio lo que provoca que los resultados no sean tan exactos, dado que ésto incrementa la dificultad de manejo.

En cuanto a las dos imágenes restantes, se han realizado a mayor velocidad e intentando abarcar toda la superficie siguiendo diferentes trayectorias. La primera muestra

```
//Adjust angle to server
if(HomeActivity.i==1 || HomeActivity.i==2){

    //cambiar valores de angulos de (-3.14→0←3.14) a (0→6.28) y girar
    angulo=(float)(angulo+Math.PI*2.5);
    if(angulo>Math.PI*2){
        angulo=(float)(angulo-Math.PI*2);
    }
    Log.v("CoordenadasA", "yaw= " +angulo+'\n');

} else { //joystick
    //girar sistema 90° en sentido horario
    angulo=(float)(angulo+Math.PI*0.5);
    if(angulo>Math.PI*2){
        angulo=(float)(angulo-Math.PI*2);
    }
    Log.v("CoordenadasA", "yaw= " +angulo+'\n');
}
```

Figura 5.14: Conversión del ángulo del punto amarillo de control al sistemas de coordenadas utilizado por el simulador TurtleSim

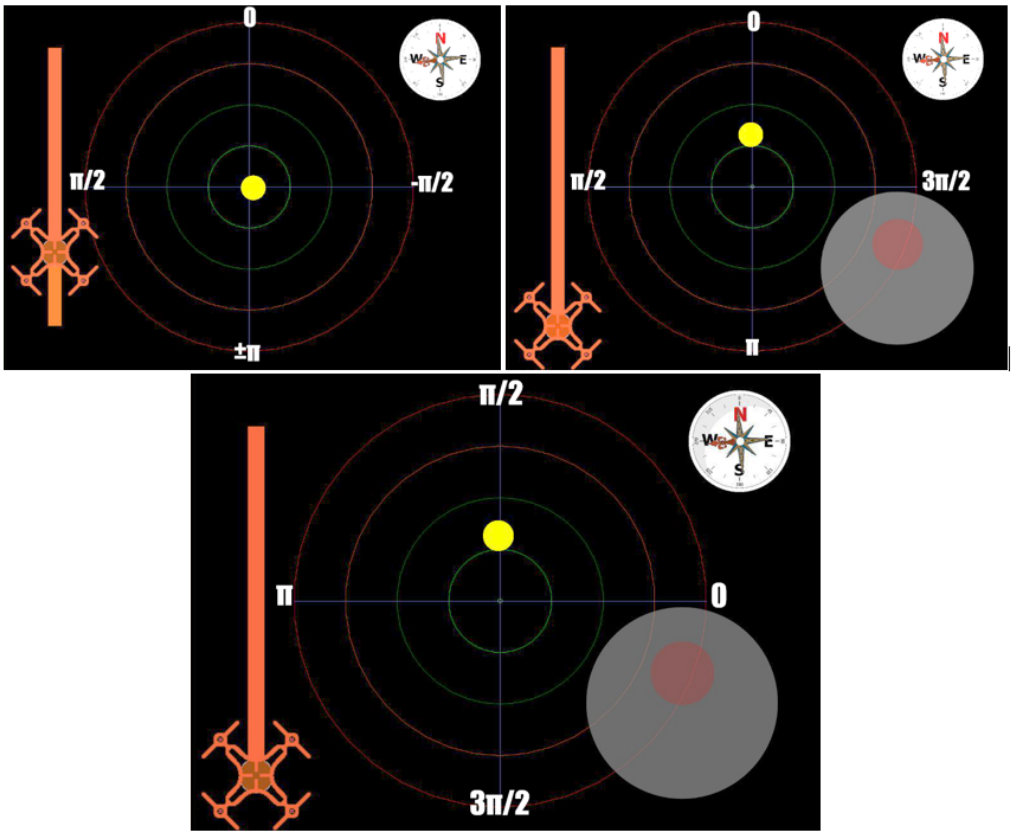

Figura 5.15: Sistemas de coordenadas del cliente (izquierda Normal mode y derecha Tactile mode) y el utilizado por el simulador TurtleSim

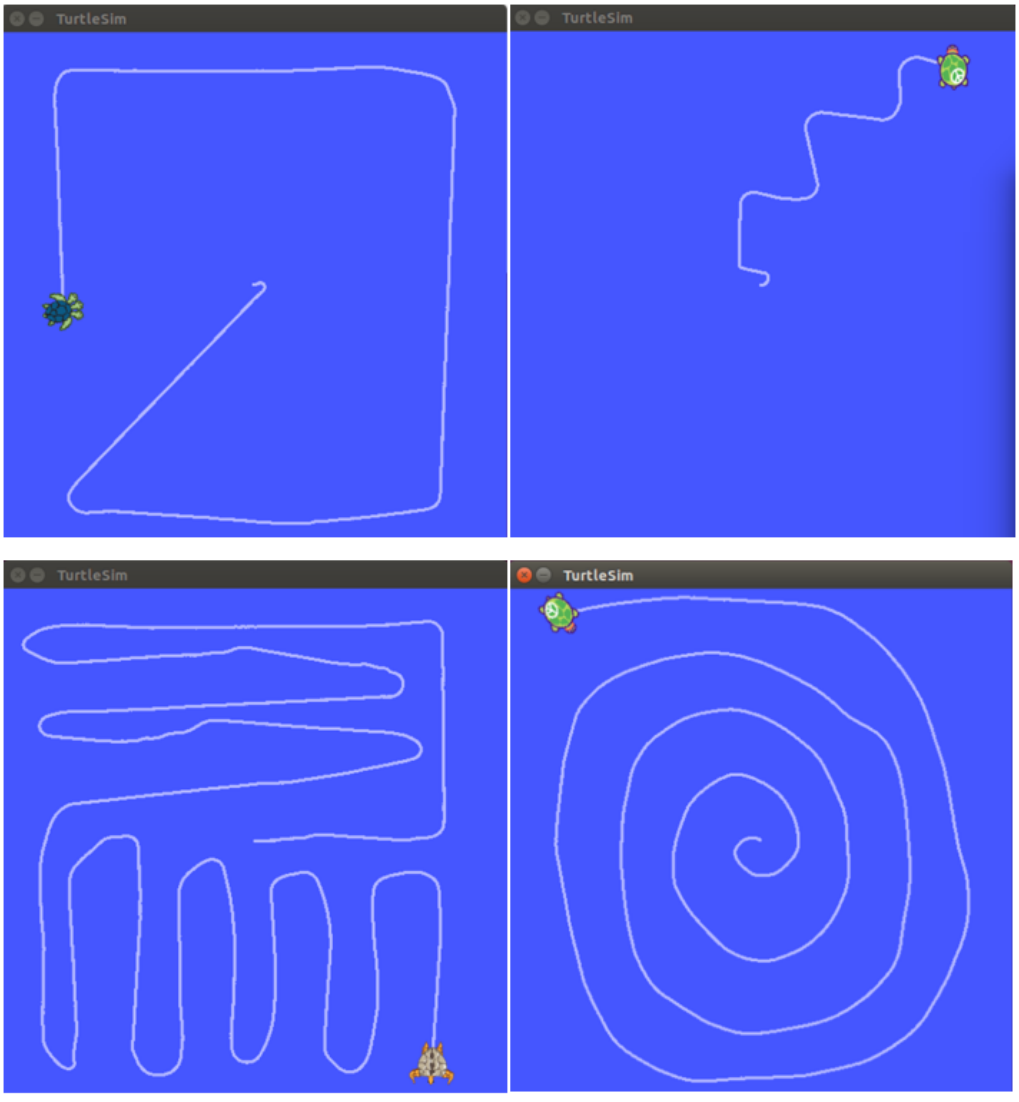

Figura 5.16: Simulaciones con TurtleSim controlado desde el móvil.

```
from morse.builder import *

# Land robot
atrv = ATRV()

pose = Pose()
pose.translate(z = 0.75)
atrv.append(pose)

motion = MotionVW()
atrv.append(motion)

# Add default interface for our robot's components
atrv.add_default_interface('ros')

env = Environment('indoors-1/indoor-1')
```

Figura 5.17: Escena para la simulación de un robot terrestre en MORSE

una recorrido imitando el de un robot de limpieza. Se observa como al ir a mayor velocidad los giros no son tan suaves como en los casos anteriores. Finalmente, se ha realizado una trayectoria con forma de espiral. Este tipo de recorridos son fáciles de realizar con bastante precisión en ambos modos de control. Sin embargo, se aconseja no incrementar demasiado la velocidad lineal pues el espacio disponible en este simulador es muy reducido.

5.4 MORSE

Para conseguir simulaciones más realistas en entornos más complejos y con otros tipos de robots (por ejemplo, drones), se ha optado por utilizar el simulador MORSE. Utilizar este nuevo simulador ofrece muchas ventajas que se han comentado en capítulos anteriores, pero también conlleva una serie de inconvenientes.

Para la ejecución del simulador TurtleSim se utilizó VirtualBox, una máquina virtual que permite emular Ubuntu desde Windows. A partir de una imagen de Nootrix que incluía Ubuntu y ROS fue relativamente sencilla su instalación, Sin embargo, la ejecución del simulador MORSE en una máquina virtual no es viable debido a sus altos requisitos computacionales que hacen que su ejecución sea extremadamente lenta. Para resolver este problema fue necesario instalar Ubunutu y ROS en una nueva partición del disco duro. Ésta es una tarea delicada y compleja, sobre todo cuando es la primera vez que lo haces.

5.4.1 Introducción

Para utilizar el simulador MORSE primero se debe crear lo que se conoce como una escena. Una escena es un archivo Python [18] donde se indica qué tipo de robot vamos a utilizar, qué sensores y actuadores queremos que incorpore y en qué ambiente queremos situarlo. MORSE incluye cada uno de estos elementos, por esto es ideal para la simulación de robots. En la Fig. 5.17 se muestra el código que carga la escena para nuestro robot terrestre.

Otro parámetro importante que se debe asignar en la escena es la comunicación que va a presentar cada uno de los sensores y actuadores. En nuestro caso, como queremos

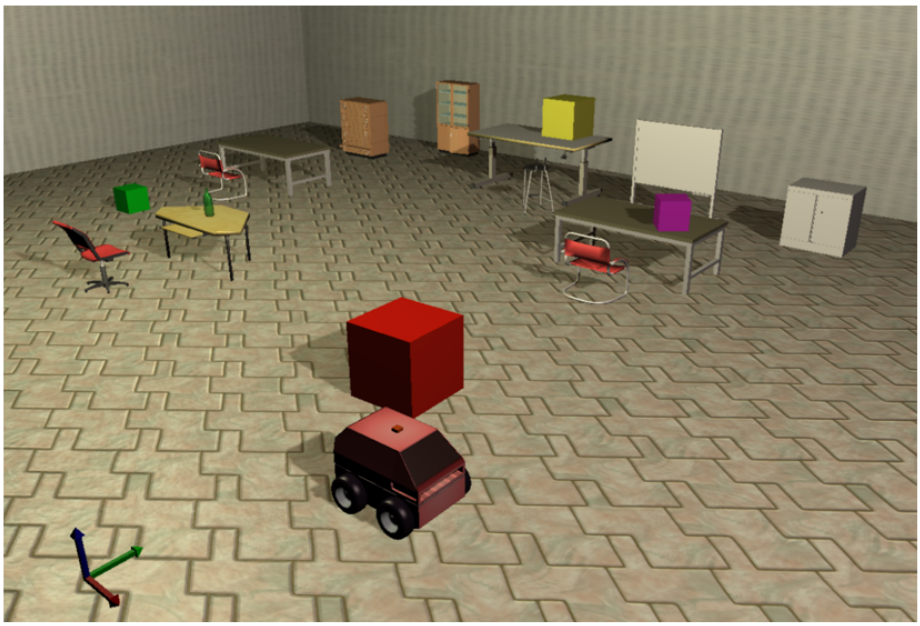

Figura 5.18: Imagen de la simulación del robot terrestre

que se comunique mediante ROS, utilizamos *atrv.add_default_interface('ros')*, de la misma manera que en el caso de TurtleSim. Por ejemplo, en la imagen anterior (robot ATRV) se crearan los Topics atrv/pose y atrv/motion. Una vez ejecutemos el archivo mediante la instrucción *morse run <archivo>*, se crearán los Topics correspondientes a los elementos creados.

5.4.2 Robot terrestre ATRV

Como primera simulación se utiliza el robot terrestre ATRV() dejando para más adelante el caso de robots aéreos. De esta manera trabajamos sobre el plano X-Y, dejando el movimiento en Z para cuando se haya practicado y dominado el nuevo simulador.

Escena

La escena implementada es la de la Fig. 5.18 que presenta un único robot ATRV con un sensor de posición (Pose) y un actuador, MotionVW, que le permite moverse tras recibir comandos de velocidad lineal (v) y angular (w). Estos dos serán los Topics asociados al robot de la figura. Observamos que MotionVW es de tipo Twist al igual que lo era cmd_vel en turtlesim. Sin embargo, el sensor de posición pasa a ser de tipo geometry_msgs/PoseStamped, lo que supondrá un cambio respecto al archivo cpp asociado al TurtleSim (Fig. 5.19).

car_controller.cpp

El código fuente del controlador es similar al usado para interaccionar con TurtleSim. Las principales diferencias son:

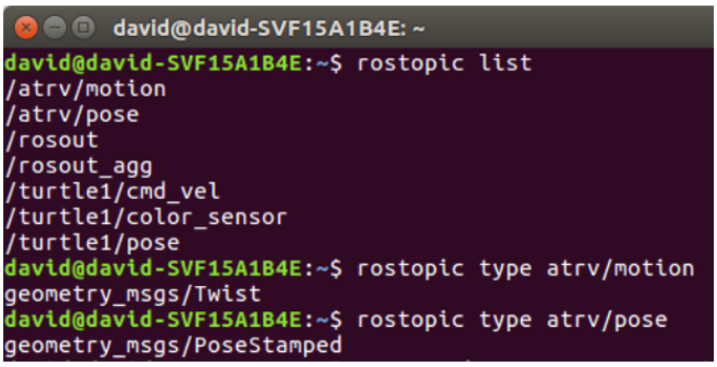

Figura 5.19: Topics utilizados para el robot terrestre

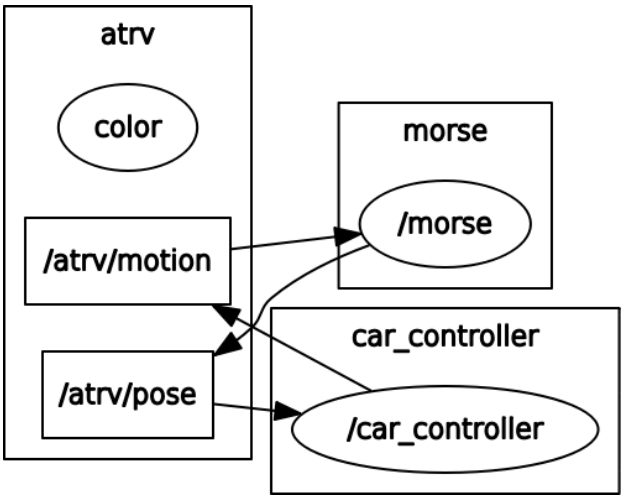

Figura 5.20: Gráfico de la comunicación mediante Topics entre ROS y MORSE para el robot terrestre

- Nuevos Topics y cambio de tipo de variable para el posicionamiento: de turtlesim/Pose a geometry_msgs/PoseStamped.
- Nuevo algoritmo de control implementado en el nuevo método llamado rotateREL().
- Nuevas variables ajustables desde el cliente: mode, velMax y angleMax.

Como ya se adelantaba en el apartado anterior, el nodo que se suscribe al Topic /atrv/pose pasa a enviar la orientación mediante una variable de tipo PoseStamped. En la Fig. 5.20 se muestra el gráfico de las conexiones mediante los Topics existentes.

La Fig. 5.21 muestra los componentes de un mensaje de este tipo. Observamos que está compuesto por un header y una variable de tipo geometry_msgs/Pose, lo que no supondría ningún problema si tuviera la variable theta que presentaba Turtlesim para saber la orientación del robot, pero no es el caso.

Para solucionarlo, se ha aplicado la transformación matemática [19] que permite obtener de un cuaternion de orientación el valor de la variable theta. El cálculo se

```
david@david-SVF15A1B4E:~$ rosmsg show geometry_msgs/PoseStamped
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

Figura 5.21: Descripción de la variable tipo PoseStamped

```
//Global variables
[...]
geometry_msgs::PoseStamped car_pose;
[...]
//métodos
void poseCallback(const geometry_msgs::PoseStamped::ConstPtr & pose_message);
[...]
void poseCallback(const geometry_msgs::PoseStamped::ConstPtr & pose_message){
    car_pose.pose.orientation.x=pose_message->pose.orientation.x;
    car_pose.pose.orientation.y=pose_message->pose.orientation.y;
    car_pose.pose.orientation.z=pose_message->pose.orientation.z;
    car_pose.pose.orientation.w=pose_message->pose.orientation.w;

    double t3= 2.0 * (car_pose.pose.orientation.w*car_pose.pose.orientation.z
                      +car_pose.pose.orientation.x*car_pose.pose.orientation.y);
    double t4= 1.0-2.0 *((car_pose.pose.orientation.y*car_pose.pose.orientation.y)
                         +car_pose.pose.orientation.z*car_pose.pose.orientation.z);

    car_pose.pose.orientation.w=std::atan2(t3, t4); //theta
}
```

Figura 5.22: Cambios producidos por la nueva variable tipo PoseStamped

realiza en la función Callback y se guarda en la componente W de dicho cuaternion como se observa en la Fig. 5.22. Otro cambio respecto a la implementación en Turtlesim son los Topics, siendo ahora los de la Fig. 5.23.

Con la intención de mejorar la aplicación y facilitar el manejo de ésta por parte del usuario se ha creado un nuevo modo de control respecto al ya implementado. La funcionalidad es la ya introducida en capítulos anteriores del modo Relative que simula el modo de control tradicional de un cuadricóptero. En este modo, el módulo del vector de posición del punto amarillo representa la velocidad lineal al igual que en el modo Absolute. Sin embargo, el ángulo respecto a la vertical ahora representa la velocidad angular a la que va a girar el robot, siendo 90º el valor máximo de la componente w. Además, los dos primeros cuadrantes representan velocidades lineales positivas y el tercero y cuarto negativas. Es decir, en este modo el robot puede ir marcha atrás. En la Fig. 5.24 se muestra el funcionamiento gráficamente siendo las anotaciones blancas las relacionadas con la velocidad angular y las turquesas con la velocidad lineal.

La implementación (Fig. 2.1) es más sencilla que la del modo Absolute pues no necesita comparar con la posición actual del robot. El procedimiento es el siguiente:

```
//Global variables
ros::Publisher velocity_publisher; //create a publisher
ros::Subscriber pose_subscriber; //subscribe to Pose

int main(int argc, char **argv){
    [...]
    //subscribers and publishers
    velocity_publisher = n.advertise<geometry_msgs::Twist>("/atrv/motion", 100);
    pose_subscriber = n.subscribe("/atrv/pose", 10, poseCallback);
    [...]
}
```

Figura 5.23: Subscriber y Publisher actualizados

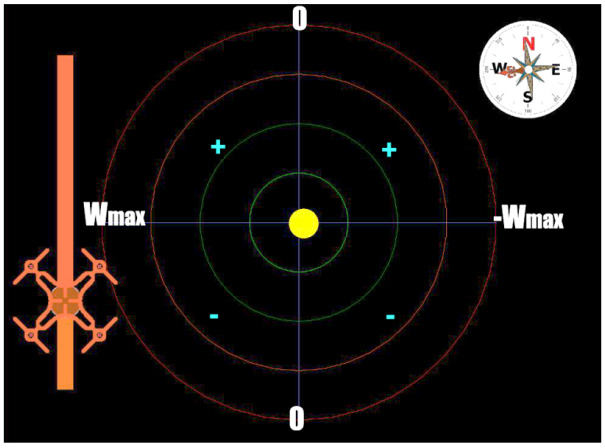

Figura 5.24: Explicación gráfica del modo Relative

1. Calcular el valor de w asignado al punto.
2. Asignar el sentido de giro.
3. Asignar la velocidad lineal con el signo correspondiente.
4. Publicar el resultado.

En relación al servidor se han implementado nuevas lecturas que permiten elegir entre los dos modos de control implementados y leer la velocidad máxima lineal y angular (siendo un % sobre la máxima del robot) que el cliente asigna desde la aplicación. La implementación es muy parecida a la anterior por lo que si se desea se podrá leer en los anexos. Tan solo destacar la implementación de la selección de modo mediante un switch case como el de la Fig. 5.26. Se puede observar en la Fig. 5.25 que la velocidad lineal tiene un valor máximo de 5m/s y la angular de 200º/s. Estos valores se han asignado de manera experimental observando que un valor máximo dificulta el manejo del robot. Sim embargo, se recomienda, en caso de hacer uso de un robot real, adaptarlos a las velocidades máximas de éstos. 5.26.

Respecto al modo Absolute prácticamente no se ha variado nada. Lo único que destaca es la condición para la variable end_angle que ha variado de 0 a 0.17 como margen de error dada las velocidades que puede llegar a alcanzar el robot. También se

```
void rotateREL (double relative_angle, double linear_speed,double maxRotVel){
    double angular_speed;
    //set a 0 velocity in the x-y-z axis
    vel_msg.linear.x =linear_speed;
    vel_msg.linear.y =0;
    vel_msg.linear.z =0;
    vel_msg.angular.x = 0;
    vel_msg.angular.y = 0;
    vel_msg.angular.z = 0;
    ros::spinOnce(); //iniciarlizar valores

    //Cálculo de la w correspondiente al punto
    if(relative_angle<=degrees2radians(90) || (relative_angle<=degrees2radians(270)
        && relative_angle>=degrees2radians(180))){
        if(relative_angle<=degrees2radians(90)){
            angular_speed=maxRotVel-(relative_angle*maxRotVel/degrees2radians(90));
        }else{
            angular_speed=maxRotVel-((relative_angle-degrees2radians(180))*maxRotVel/degrees2radians(90));
        }
    }
    if(relative_angle>=degrees2radians(270) || (relative_angle<=degrees2radians(180)
        && relative_angle>=degrees2radians(90))){
        if(relative_angle>=degrees2radians(270)){
            angular_speed=(relative_angle-degrees2radians(270))*maxRotVel/degrees2radians(90);
        }else{
            angular_speed=(relative_angle+degrees2radians(-270+180))*maxRotVel/degrees2radians(90);
        }
    }
    //asignar sentido de giro
    if(relative_angle>=degrees2radians(90) && relative_angle<=degrees2radians(270))
        vel_msg.angular.z =abs(angular_speed);
    else
        vel_msg.angular.z =-abs(angular_speed);

    if(linear_speed==0){
        vel_msg.angular.z = 0;
    }
    //linear speed
    if(relative_angle>degrees2radians(180)){
        vel_msg.linear.x = -linear_speed;
    }
    //publish
    velocity_publisher.publish(vel_msg);
}
```

Figura 5.25: Implementación del modo RELATIVE

```
angulos=(double)angulo;
velocidades=(double)longitud*5/2*velMax/100;
switch (mode){
    case 0:
        rotateABS (degrees2radians(200)*angularMax/100);
        break;
    case 1:
        rotateREL (angulos, velocidades,degrees2radians(200)*angularMax/100);
        break;
    default:
        break;
}
```

Figura 5.26: Selección del modo de control

```
}while(end_angle>0.17 && velocidades>0);
vel_msg.angular.z =0;
velocity_publisher.publish(vel_msg);
```

Figura 5.27: Cambios en la condición de salida del bucle de giro

ha añadido la condición de que el robot tenga un valor de velocidad lineal superior a 0 para evitar que rote cuando se desea pararlo (Fig. 5.27) .

Demostraciones

En este apartado se mostrará el comportamiento y control de un robot terrestre simulado en MORSE. Para ello se han realizado dos vídeos, uno para cada modo de control (Absolute y Relative). El vídeo está editado de manera que se puede visualizar la pantalla del smartphone superpuesta a la del simulador. Para que se puedan apreciar todos los elementos, se recomienda visualizar los vídeos a 720p y pantalla completa.

En el modo Absolute, se ha utilizado el control táctil (joystick) como se observa en el siguiente vídeo: https://youtu.be/PkqePkhnqdg. Se puede ver como el robot responde perfectamente a las ordenes enviadas por el dispositivo móvil. Ademas, el control para el usuario es sencillo lo que permite acceder por caminos estrechos con el robot sin dificultad.

Para el modo Relative se ha utilizado el control normal (usando los sensores internos de orientación del dispositivo móvil) como se observa en el siguiente vídeo: https://youtu.be/q1i0ctlcIUQ. Este control requiere de una mayor visión espacial del usuario lo que lo hace mas complejo de controlar que el modo anterior. Sin embargo, el control sobre el robot es mayor, incluyendo marcha atrás, lo que nos permite acceder por recorridos difíciles con mayor facilidad una vez entendido su funcionamiento.

5.4.3 Robot aéreo Quadrotor

En esta sección se pretende adaptar la interfaz para usar robots aéreos. Para ello utilizaremos el robot aéreo disponible en MORSE llamado Quadrotor (Fig. 5.28). Para comprender el funcionamiento de los nuevos actuadores y sensores se ha realizado el tutorial flying_cat_and_mouse [20].

Una vez llegados a este punto se ha comprendido el funcionamiento del simulador MORSE, su comunicación con ROS y la creación de escenas hasta el punto de ser capaces de realizar el control de un robot aéreo con un dispositivo móvil. La utilización de un robot como el Quadrotor() presenta el mismo tipo de variables en cuanto a Topics (Fig. 5.29). Sin embargo, el comportamiento de los actuadores es un poco diferente lo que ha supuesto tener que realizar algunas modificaciones en parámetros de la escena y del programa principal.

Escena

Por un lado, la simulación se ha realizado seleccionando un actuador RotorcraftVelocity [21], cuyo input son los valores de velocidad lineal y angular, un sensor de posición tipo Pose y una cámara tipo SemanticCamera que nos permite visualizar lo que ve el robot,

Figura 5.28: Escena implementada para el control del cuadricóptero

```
david@david-SVF15A1B4E:~$ rostopic list
/atrv/motion
/atrv/pose
/cat/camera
/cat/catPose
/cat/velocity
/rosout
/rosout_agg
/tf
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
david@david-SVF15A1B4E:~$ rostopic type cat/catPose
geometry_msgs/PoseStamped
david@david-SVF15A1B4E:~$ rostopic type cat/velocity
geometry_msgs/Twist
```

Figura 5.29: Lista de los Topics utilizados para controlar el vehículo aéreo

```
from morse.builder import *
from math import pi

""" Cat (Quadrotor) """
cat = Quadrotor()
cat.translate(z=3.0)

velocity = RotorcraftVelocity() #RotorcraftVelocity
velocity.properties(HorizontalPgain=3,YawPgain=80,YawDgain=20,VerticalPgain=15)
cat.append(velocity)
catPose = Pose()
cat.append(catPose)

camera = SemanticCamera()
camera.translate(x=0.3, z=-0.05)
camera.rotate(x=+0.2)
cat.append(camera)
camera.properties(Vertical_Flip=True)

cat.add_default_interface('ros')

env = Environment('outdoors')
env.set_camera_location([-3.0, -9, 8.0]) #0,0,22
env.set_camera_rotation([1, 0, 0])
env.select_display_camera(camera)
```

Figura 5.30: Implementación de la escena del Quadrotor

proporcionando una vista auxiliar al usuario. En cuanto a los entornos seleccionados se han realizado pruebas con los escenarios *outdoors* y *indoors-1*/*indoor-1* (Fig. 5.30).

A diferencia del resto de robots utilizados, ha sido necesario adaptar parámetros del actuador experimentalmente para lograr el resultado de manejo deseado (Fig. 5.31).

```
velocity = RotorcraftVelocity() #RotorcraftVelocity
velocity.properties(HorizontalPgain=3,YawPgain=80,YawDgain=20,VerticalPgain=15)
cat.append(velocity)
```

Figura 5.31: Parámetros del actuador del robot modificados experimentalmente

Los cuatro parámetros modificados hacen referencia a ganancias, lo que se traduce a la capacidad del robot para mantener su posición cuando se le indica y no seguir avanzando. Por ejemplo, el parámetro HorizontalPgain determina la capacidad de que el robot se pare o siga con una cierta inercia en el plano horizontal después de que se le haya indicado que debe pararse. Lo mismo sucede para el resto de parámetros, para la rotación y la velocidad en el eje Z. Con los valores propuestos se consiguen resultados realistas con gran precisión.

drone_controller.cpp

Como ya se ha comentado, en líneas generales la implementación es muy parecida al caso anterior (car_controller.cpp), habiendo una serie de cambios que merece la pena comentar:

- Se utilizan nuevos Topics.

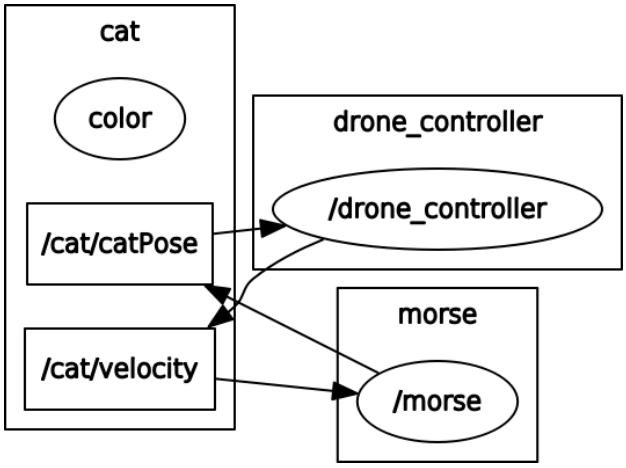

Figura 5.33: Comunicación entre ROS, MORSE y el cuadricóptero

- Se tienen que gestionar nuevas variables de control en el servidor.
- Se ha creado un nuevo modo de control llamado Position implementado dentro del método moveGoal.

El cambio más obvio es el cambio por los nuevos Topics para comunicarse con el quadrotor, tal como se muestra en la Fig. 5.32.

```
velocity_publisher = n.advertise<geometry_msgs::Twist>("/cat/velocity", 100);
pose_subscriber = n.subscribe("/cat/catPose", 10, poseCallback);
```

Figura 5.32: Nuevos Topics para el Subscriber y Publisher del dron

La comunicación entre los diferentes nodos se representa gráficamente en la Fig. 5.33.

Una vez la comunicación ha sido establecida observamos que el control del dron es el deseado. Tan solo nos falta crear la variable asociada a la altura entre otras que se incluirán en la lectura de datos del servidor, tales como la altura máxima deseada o el área máxima (todas ellas en forma de porcentaje sobre el valor máximo alcanzable por el robot). El área máxima se usará para un nuevo modo de control (Position) que se explicará más adelante.

Para determinar la altura se utilizará el método checkHeight() de la Fig. 5.34 donde se comprueba que la diferencia entre la altura del robot y la deseada sea menor a 0.2m. Esta función se ejecutará antes de invocar el método publish(), tanto dentro del bucle de comprobación del ángulo como fuera, ya que se debe comprobar independientemente de la orientación en el plano XY del robot (Fig. 5.35).

Finalmente se ha implementado un nuevo modo de control llamado Position Mode. Este nuevo modo es el más simple de los tres y permite posicionar el cuadricóptero en la posición del punto amarillo suponiendo la circunferencia máxima como el área abarcable por el robot. Es decir, el punto amarillo simboliza el dron visto desde arriba. Para visualizarlo mejor se ha creado una escena con una vista aérea donde se ha superpuesto la pantalla del móvil (Fig. 5.36).

```
void checkHeight(){
    if(abs(alturaMax*altura/100-drone_pose.pose.position.z)<0.2){
        vel_msg.linear.z=0;
    } else {
        if(alturaMax*altura/100-drone_pose.pose.position.z<0){
            vel_msg.linear.z=-1.5;
        } else {
            vel_msg.linear.z=1.5;
        }
    }
}
```

Figura 5.34: Subprograma de control de altura del robot

```
void rotateABS (double angular_speed){
    [...]
    ros::Rate loop_rate(80);
    do{
        [...]

        checkHeight();

        velocity_publisher.publish(vel_msg);
       }while(end_angle>0.17 && velocidades>0);
        vel_msg.angular.z =0;

        checkHeight();

        velocity_publisher.publish(vel_msg);
}
```

Figura 5.35: Utilización del subprograma checkHeight en el modo Absolute

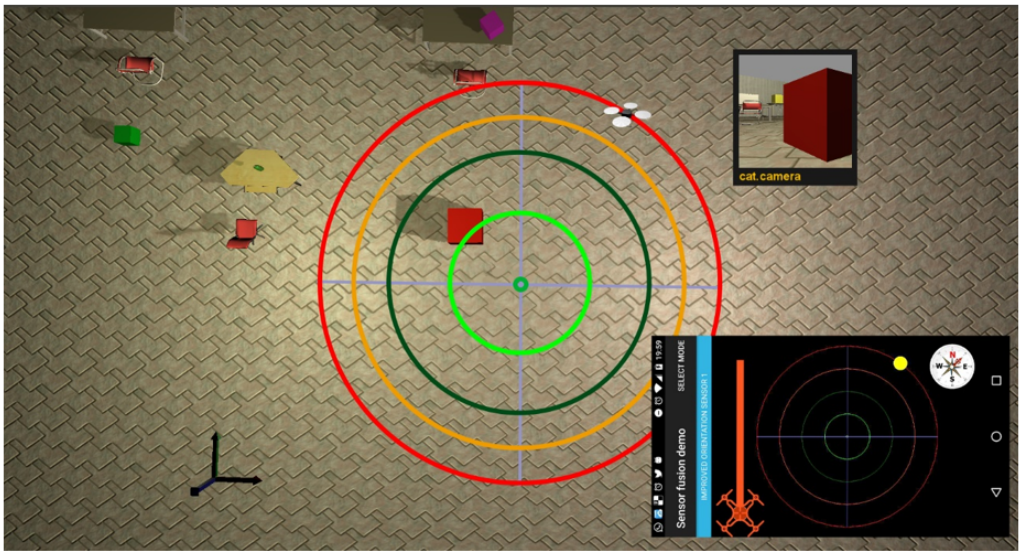

Figura 5.36: Explicación gráfica del modo Relative

```
//Orientar el drona a la posicion inicial
while(drone_pose.pose.orientation.w>0.5 || drone_pose.pose.orientation.w<-0.5){
    rotating=true;
    if(drone_pose.pose.orientation.w>2  || drone_pose.pose.orientation.w<-0.5){
        vel_msg.angular.z = 2;
    }else{
        vel_msg.angular.z = 0.4;
    }
    if(drone_pose.pose.orientation.w<-0.5){
        vel_msg.angular.z=-vel_msg.angular.z;
    }
    velocity_publisher.publish(vel_msg);
    //cout<<drone_pose.pose.orientation.w<<endl;
}
```

Figura 5.37: Implementación para orientar el cuadricóptero al sistema de coordenadas del dispositivo

Este nuevo modo está implementado en el método moveGoal(). Lo primero que se debe hacer es orientar el robot para que su sistema de coordenadas coincida con el del móvil (Fig. 5.37). Una vez orientado, la mantendrá mientras este modo esté activo.

Una vez orientado el robot se comprueba que su posición X-Y coincida con las del punto amarillo del cliente. Para que el cuadricóptero alcanzase la posición de una manera controlada ha sido necesario implementar distancias de frenado, es decir, que a menos de 0.7m del punto objetivo el robot disminuye su velocidad para evitar pasarse (Fig. 5.38).

Demostraciones

En este apartado se mostrará el comportamiento y control de un robot aéreo tipo cuadricóptero simulado en MORSE. Para ello se han realizado tres vídeos, uno para cada modo de control (Absolute, Relative y Position). El vídeo está editado de manera que se puede visualizar la pantalla del smartphone superpuesta a la del simulador. Para que se puedan apreciar todos los elementos, se recomienda visualizar los vídeos a 720p y pantalla completa.

El modo Absolute se ha realizado con el control normal mediante sensores. Se puede observar la buena respuesta del dron en las tres dimensiones lo que permite, incluso, hacer aterrizar el robot encima de objetos. La simulación mostrada en el vídeo https://youtu.be/diPwbt3J2KU se ha realizado en un ambiente exterior a diferencia del resto.

El vídeo correspondiente al modo Relative (https://youtu.be/3xtwIk0d_3Q) se ha realizado con el control táctil. Como ya se ha mencionado anteriormente, el modo Relative permite un mayor control por parte del usuario a pesar de incrementar la dificultad de control. Por este motivo, se ha realizado un vídeo en el que se pueda apreciar la respuesta del robot y su control haciéndolo aterrizar a diferentes alturas y en varios objetos.

Para el modo Position, se ha variado la cámara de la escena proporcionando una vista aérea de la sala. En el vídeo https://youtu.be/7YJi6dl5OeY se observa que el control implementado es el táctil y que el dron se posiciona en la posición deseada controlando su velocidad (frenando) al acercarse al punto objetivo.

```
void moveGoal(double angle, double longitud){
    [...]
    x=longitud*areaMax/2*cos(angle);
    y=longitud*areaMax/2*sin(angle);

    distX=x-drone_pose.pose.position.x;
    distY=y-drone_pose.pose.position.y;

    //Comprobar posicion X
    if(abs(distX) <0.2){
        vel_msg.linear.x =0;
    } else {
        if(abs(distX) <0.7){
            velocidad=0.4;
        } else {
            velocidad=2.1;
        }
        if(abs(distX)>=abs(distY)){
            vel_msg.linear.x =velocidad*distX/abs(distX);
        }else{
            vel_msg.linear.x =velocidad*distX/abs(distY);
        }
    }
    //Comprobar posicion Y
    if(abs(distY) <0.2){
        vel_msg.linear.y =0;
    } else {
        if(abs(distY) <0.7){
            velocidad=0.4;
        } else {
            velocidad=2.1;
        }
        if(abs(distX)>=abs(distY)){
            vel_msg.linear.y =velocidad*distY/abs(distX);
        }else{
            vel_msg.linear.y =velocidad*distY/abs(distY);
        }
    }

    checkHeight();
    velocity_publisher.publish(vel_msg);
}
```

Figura 5.38: Implementación del nuevo algoritmo de control Position

CAPÍTULO 6

CONCLUSIONES

A lo largo del presente trabajo de final de grado se ha desarrollado una aplicación Android de alto nivel de complejidad que incorpora desde elementos tridimensionales a comunicación mediante sockets como parte de un cliente con conexión a un servidor implementado en ROS, que a su vez se comunica con los simuadores TurtleSim y MORSE. Con esta arquitectura se ha logrado controlar diferentes robots móviles mediante un dispositivo móvil a través de sus sensores internos de orientación o mediante un joystick aplicando tres posibles algoritmos de control sobre el robot.

Cada uno de estos algoritmos tiene un grado de complejidad diferente en cuanto a manejabilidad facilitando el control del robot. El más sencillo, *Position Mode*, implementa el control indicando la posición en la que se desea que se posicione el robot dentro de un espacio cerrado de vuelo. El siguiente algoritmo de control en términos de complejidad sería el *Absolute Mode*, que permite indicar hacia que dirección se quiere que se dirija el robot en coordenadas absolutas, por lo que será un modo menos restrictivo que el anterior. Por último, se presenta el control tradicional mediante la indicación de la velocidad lineal y angular relativa que se quiere que tome el robot. A pesar de ser el más complejo también es el modo más completo pues permite un control casi total del robot (*Relative Mode*).

6.1 Aplicación Android

Se pretendía desarrollar una aplicación intuitiva y sencilla orientada a usuarios inexpertos del manejo de drones y a base de probar con diferentes usuarios se ha conseguido. Se ha testeado con personas con las características indicadas que han ayudado a ir perfilando aspectos como la necesidad de modos de control más simples, pantallas intermedias con imágenes de cómo orientar el dispositivo o un control de acceso a los ajustes. Sin embargo, la aplicación móvil está diseñada para ser utilizada bajo la vigilancia de un experto que supervise y explique los tres modos de manejo.

Los principales problemas que se encontraron en la creación de la App fueron

la comprensión y el uso de elementos 3D teniendo que trabajar con cuaterniones y matrices combinado con la complejidad de la arquitectura utilizada para poder manipular cómodamente la fusión de sensores.

6.2 Ubuntu

En este apartado se citarán los diferentes problemas que han surgido en todo lo relacionado con Ubuntu [22] incluyendo ROS y MORSE. A pesar de la dificultad que requería la creación de una App como la diseñada, la utilización de un sistema operativo nuevo y la puesta a punto para poder utilizar programas como ROS y MORSE ha llevado mucho más tiempo del esperado. Uno de los principales problemas fue la necesidad de particionar el disco duro que significó tener que reinstalar Windows debido a problemas internos del PC.

Por otro lado, el entendimiento y manejo de las librerias de ROS ha supuesto una gran inversión de tiempo realizando tutoriales y pruebas hasta llegar al punto de ser capaz de hacer nuestro propio programa.

6.3 Futuras mejoras

A pesar de los buenos resultados obtenidos, se plantean nuevos objetivos que se podrían implementar para hacer de ésta una aplicación de control de robots aún más completa. Se plantean diversas ideas que se citarán y explicarán a continuación:

- *La implementación en robots reales.* Éste sería el objetivo más importante de los que se citarán, pues es el paso final del proyecto, formando parte de una segunda fase. Éste permitirá poder usar la App para el fin que ha sido planteado.

- *Incorporación de un algoritmo de evitación de obstáculos.* Con la intención de incrementar la seguridad, se plantea la opción de implementar algún algoritmo de evitación de obstáculos.

- *Mapeado de obstáculos y visualización en el cliente.* Una funcionalidad práctica seria incorporar un algoritmo de mapeado que permitiese visualizar desde la pantalla del móvil los obstáculos cercanos al dron, permitiendo así su control sin necesidad de ver el entorno real. Para realizarlo el robot necesitaría incorporar sensores de detección de presencia y enviar los datos recogidos del servidor al cliente.

- *Cámara.* Visualización de la cámara del robot en tiempo real desde la App como opción añadida podría ser útil y necesaria una vez realizada la implementación en robots reales. En nuestro caso las imágenes capturadas por la cámara del robot han podido ser visualizadas desde la pantalla del simulador MORSE, por lo que no ha sido necesaria esta mejora.

- *Incorporación de un modo experto.* Tal y como se ha diseñado la App, el control no permite hacer acrobacias o maniobras avanzadas. Por este motivo, la implementación de un modo de nivel avanzado podría ser útil para realizar exhibiciones por

parte del profesorado más experto. Las características de este modo de control dependerían de las posibilidades del robot utilizado.

APÉNDICE A

CÓDIGO FUENTE

A.1 Código de la App

A.1.1 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/
    android"
        package="com.example.david.tfg" >

        <uses-permission android:name="android.permission.INTERNET
            "/>
        <application
                android:allowBackup="true"
                android:supportsRtl="true"
                android:icon="@drawable/ic_tfg"
                android:label="@string/app_name">
                <activity
                        android:name="com.example.david.tfg.
                            HomeActivity"
                        android:configChanges="orientation|
                            keyboardHidden"
                        android:label="@string/app_name"
                        android:screenOrientation="portrait" >
                        <intent-filter>
                                <action android:name="android.
                                    intent.action.MAIN" />
                                <category android:name="android.
                                    intent.category.LAUNCHER" />
                        </intent-filter>
```

```
                </activity>
                <activity
                        android:name="com.example.david.tfg.
                            SensorSelectionActivity"
                        android:label="@string/
                            title_activity_sensor_selection"
                        android:screenOrientation="nosensor" >
                </activity>
                <activity
                        android:name="com.example.david.tfg.
                            Rotation"
                        android:label="@string/
                            title_activity_sensor_selection"
                        android:screenOrientation="nosensor" >
                </activity>
                <activity
                        android:name="com.example.david.tfg.
                            Settings"
                        android:label="@string/
                            title_activity_sensor_selection"
                        android:screenOrientation="nosensor" >
                </activity>
        </application>

</manifest>
```

A.1.2 HomeActivity.java

```
package com.example.david.tfg;

import android.app.Activity;
import android.app.Dialog;
import android.content.Context;
import android.content.Intent;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.os.Handler;
import android.support.v4.app.DialogFragment;

import android.support.v4.app.FragmentManager;
import android.text.InputType;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.Window;
```

```
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.afollestad.materialdialogs.MaterialDialog;

import com.example.david.tfg.R;

/**
* A class that provides a navigation menu
*
*/
public class HomeActivity extends Activity
{
public static int i=0;
public static Boolean joystick=false;
public static Boolean seekbar=false;
public static int num_fused_sensors=6;
static public Boolean horizontalRotation=false;
static public Boolean compass;
static public int mode=0;
static public int altura=3;
public static float angulo;
public static float longitud;
public static Activity home;
public static int language;


@Override
protected void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);
        home=this;
        final SharedPreferences prefs = getSharedPreferences("
            SettingsDroneControl",Context.MODE_PRIVATE);

        language=prefs.getInt("language",language());
        Locale localizacion;
        switch (language){
                case 1:
                        localizacion = new Locale("en", "EN");
                        break;
                case 2:
                        localizacion = new Locale("es", "ES");
                        break;
                case 3:
                        localizacion = new Locale("ca", "CA");
```

```
                        break;
                default:
                        localizacion = new Locale("en", "EN");
                        break;
        }
        Locale.setDefault(localizacion);
        Configuration config = new Configuration();
        config.locale = localizacion;
        this.getBaseContext().getResources().updateConfiguration(
            config, this.getBaseContext().getResources().
            getDisplayMetrics());

        setContentView(R.layout.layout_home);

        initButtonNormal();
        initButtonTouch();
        initButtonPoint();
        initButtonDrone();

}

@Override
public boolean onCreateOptionsMenu(Menu menu)
{
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu_home, menu);
        return true;
}

/**
* Event Handling for Individual menu item selected Identify single
    menu
* item by it's id
* */
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
        switch (item.getItemId())
        {
        case R.id.action_settings:
                showHelpDialog();
                return true;

        default:
                return super.onOptionsItemSelected(item);
        }
}
```

```
@Override
public void onPause()
{
        super.onPause();
}

@Override
public void onResume()
{
        super.onResume();
}
@Override
public void onBackPressed() {
        super.onBackPressed();
        finish();
}

private void initButtonNormal()
{
        Button button = (Button) this.findViewById(R.id.
            button_normal_mode);

        button.setOnClickListener(new View.OnClickListener()
        {
                public void onClick(View v)
                {
                        i=1;
                        num_fused_sensors=1;
                        joystick=false;
                        seekbar=true;
                        horizontalRotation=false;
                        compass=true;
                        Intent intent = new Intent(HomeActivity.
                            this,
                        Rotation.class);

                        startActivity(intent);
                }
        });
}


private void initButtonTouch()
{
        Button button = (Button) this.findViewById(R.id.
            button_touching_mode);
```

```
        button.setOnClickListener(new View.OnClickListener()
        {
                public void onClick(View v)
                {
                        i=3;
                        joystick=true;
                        num_fused_sensors=1;
                        seekbar=true;
                        horizontalRotation=false;
                        compass=true;
                        Intent intent = new Intent(HomeActivity.
                            this,
                        Rotation.class);
                        startActivity(intent);
                }
        });
}

private void initButtonPoint()
{
        Button button = (Button) this.findViewById(R.id.
            button_point_mode);

        button.setOnClickListener(new View.OnClickListener()
        {
                public void onClick(View v)
                {
                        i=2;
                        num_fused_sensors=6;
                        joystick=false;
                        seekbar=false;
                        horizontalRotation=true;
                        compass=false;
                        Intent intent = new Intent(HomeActivity.
                            this,
                        SensorSelectionActivity.class);

                        startActivity(intent);
                }
        });
}

private void initButtonDrone()
{
        Button button = (Button) this.findViewById(R.id.
            button_drone_mode);
```

```
        button.setOnClickListener(new View.OnClickListener()
                {
                public void onClick(View v)
                {
                        i=0;
                        num_fused_sensors=6;
                        joystick=false;
                        seekbar=false;
                        horizontalRotation=true;
                        compass=false;
                        Intent intent = new Intent(HomeActivity.
                            this,
                        SensorSelectionActivity.class);

                        startActivity(intent);
                }
        });
}
private void showSettings()
{
        Intent i = new Intent(this, Settings.class);
        startActivity(i);
}
private void showHelpDialog()
{
        new MaterialDialog.Builder(this)
                .title(R.string.title)
                .content(R.string.input_content)
                .inputType(InputType.TYPE_CLASS_TEXT | InputType.
                    TYPE_TEXT_VARIATION_PASSWORD)
                .input(R.string.input_hint, R.string.input_prefill
                    , new MaterialDialog.InputCallback() {
                @Override
                public void onInput(MaterialDialog dialog,
                    CharSequence input) {
                        // Do something
                        if(input.toString().equals(getString(R.
                            string.password))){
                                showSettings();
                        } else {
                                Context context =
                                    getApplicationContext();
                                CharSequence text = "Incorrect␣
                                    password";
                                int duration = Toast.LENGTH_SHORT;
```

```
                                Toast toast = Toast.makeText(
                                    context, text, duration);
                                toast.show();
                            }
                        }
                }).show();
}
        int language (){ //primera vez, pref vacio
                switch (Locale.getDefault().getDisplayLanguage()){
                        case "en":
                                return 1;
                        case "es":
                                return 2;
                        case "ca":
                                return 3;
                        default:
                                return 1;
                }
        }
}
```

A.1.3 SensorSelectionActivity.xml

```
package com.example.david.tfg;

import java.util.Locale;

import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.res.Configuration;
import android.graphics.Color;
import android.graphics.drawable.Drawable;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import android.support.v4.view.ViewPager;
import android.text.InputType;
import android.util.Log;
import android.view.Display;
import android.view.Gravity;
import android.view.KeyEvent;
import android.view.Menu;
```

```
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.Surface;
import android.view.View;
import android.view.WindowManager;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.RelativeLayout;
import android.widget.Toast;

import com.afollestad.materialdialogs.MaterialDialog;
import com.crystal.crystalrangeseekbar.interfaces.
    OnSeekbarChangeListener;
import com.erz.joysticklibrary.*;
import com.erz.joysticklibrary.JoyStick;

/**
* The main activity where the user can select which sensor-fusion
    he wants to try out
*
* @author David Arévalo Jiménez
*
*/
public class SensorSelectionActivity extends FragmentActivity
    implements com.erz.joysticklibrary.JoyStick.JoyStickListener{

/**
* The {@link android.support.v4.view.PagerAdapter} that will
    provide
* fragments for each of the sections. We use a {@link android.
    support.v4.app.FragmentPagerAdapter} derivative,
* which
* will keep every loaded fragment in memory. If this becomes too
    memory
* intensive, it may be best to switch to a {@link android.support.
    v4.app.FragmentStatePagerAdapter}.
*/
SectionsPagerAdapter mSectionsPagerAdapter;

/**
* The {@link ViewPager} that will host the section contents.
*/
ViewPager mViewPager;
static public double angleTouch=0;
static public double powerTouch=0;

private Compass compass;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sensor_selection);

        // Create the adapter that will return a fragment for each
             of the three
        // primary sections of the app.
        mSectionsPagerAdapter = new SectionsPagerAdapter(
            getSupportFragmentManager());

        // Set up the ViewPager with the sections adapter.
        mViewPager = (ViewPager) findViewById(R.id.pager);
        mViewPager.setAdapter(mSectionsPagerAdapter);

        compass = new Compass(this);
        compass.arrowView = (ImageView) findViewById(R.id.
            main_image_hands);

        LinearLayout rotation = (LinearLayout) findViewById(R.id.
            rotation); //
        rotation.setVisibility(View.GONE);
        RelativeLayout rotation2 = (RelativeLayout) findViewById(R
            .id.rotation2);

        com.erz.joysticklibrary.JoyStick joy1 = (com.erz.
            joysticklibrary.JoyStick) findViewById(R.id.joy1);
        com.crystal.crystalrangeseekbar.widgets.CrystalSeekbar
            UpDown = (com.crystal.crystalrangeseekbar.widgets.
            CrystalSeekbar) findViewById(R.id.UpDown);

        UpDown.setOnSeekbarChangeListener(new
            OnSeekbarChangeListener() {
                @Override
                public void valueChanged(Number minValue) {
                        HomeActivity.altura=Integer.parseInt(
                            String.valueOf(minValue));
                        Log.v("altura", "altura=" + HomeActivity.
                            altura +'\n');
                }
        });

        joy1.setListener(this);
        joy1.setPadColor(Color.parseColor("#55ffffff"));
        joy1.setButtonColor(Color.parseColor("#55ff0000"));
```

```
        Connect.addMyBooleanListener(new com.example.david.tfg.
            ConnectionBooleanChangedListener5() {
                @Override
                public void OnMyBooleanChanged() {
                        if (Connect.getMyBoolean()) {
                                LinearLayout rotation = (
                                    LinearLayout) findViewById(R.id
                                    .rotation);
                                rotation.setVisibility(View.GONE);
                        }
                }
        });
        if(HomeActivity.compass){
                rotation2.setVisibility(View.VISIBLE);
        } else {
                rotation2.setVisibility(View.GONE);
        }
                if(HomeActivity.seekbar){
                UpDown.setVisibility(View.VISIBLE);
        } else {
                UpDown.setVisibility(View.GONE);
        }
        if(HomeActivity.joystick){
                joy1.setVisibility(View.VISIBLE);
                //joy2.setVisibility(View.VISIBLE);
        } else {
                joy1.setVisibility(View.GONE);
                //joy2.setVisibility(View.GONE);
        }

        // Check if device has a hardware gyroscope
        SensorChecker checker = new HardwareChecker((SensorManager
            ) getSystemService(SENSOR_SERVICE));
        if(!checker.IsGyroscopeAvailable()) {
                // If a gyroscope is unavailable, display a
                    warning.
                displayHardwareMissingWarning();
        }
}
@Override
public void onBackPressed() {
        super.onBackPressed();
        finish();
}
private void displayHardwareMissingWarning() {
        AlertDialog ad = new AlertDialog.Builder(this).create();
```

```
        ad.setCancelable(false); // This blocks the 'BACK' button
        ad.setTitle(getResources().getString(R.string.
            gyroscope_missing));
        ad.setMessage(getResources().getString(R.string.
            gyroscope_missing_message));
        ad.setButton(DialogInterface.BUTTON_NEUTRAL, getResources
            ().getString(R.string.OK), new DialogInterface.
            OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int
                    which) {
                        dialog.dismiss();
                }
        });
        ad.show();
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if
            it is present.
        if(HomeActivity.i==1 || HomeActivity.i==3){
                getMenuInflater().inflate(R.menu.sensor_selection,
                    menu);
        } else {
                getMenuInflater().inflate(R.menu.menu_home, menu);
        }
        return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
        if(HomeActivity.i==1 || HomeActivity.i==3){
                switch (item.getItemId())
                {
                case R.id.absolute:
                        if(HomeActivity.altura==0){
                                HomeActivity.mode=0;
                        } else {
                                Toast toast = Toast.makeText(
                                    getApplicationContext(), "Set␣
                                    height␣to␣0␣to␣change␣mode",
                                    Toast.LENGTH_SHORT);
                                toast.show();
                        }
                        break;
                case R.id.relative:
```

```
                        if(HomeActivity.altura==0){
                    HomeActivity.mode=1;
                    } else {
                        Toast toast = Toast.makeText(
                            getApplicationContext(), "Set␣
                            height␣to␣0␣to␣change␣mode",
                            Toast.LENGTH_SHORT);
                        toast.show();
                    }
                    break;
                case R.id.position:
                        if(HomeActivity.altura==0){
                        HomeActivity.mode=2;
                    } else {
                        Toast toast = Toast.makeText(
                            getApplicationContext(), "Set␣
                            height␣to␣0␣to␣change␣mode",
                            Toast.LENGTH_SHORT);
                        toast.show();
                    }
                    break;
                default:
                    break;
                }
        } else {
        switch (item.getItemId())
        {
                // Start the vector activity
                case R.id.action_settings:
                    showHelpDialog();
                    return true;

                default:
                    return super.onOptionsItemSelected(item);
                }
        }
        return false;
}

@Override
public void onMove(JoyStick joyStick, double angle, double power,
    int direction) {
        switch (joyStick.getId()) {
        case R.id.joy1:
                //gameView.move(angle, power);
                angleTouch=angle;
                powerTouch=power;
```

```
                    break;
            }
}

@Override
            public void onTap() {

}

@Override
            public void onDoubleTap() {

}
@Override
            protected void onStart() {
            super.onStart();
            //Log.d(TAG, "start compass");
            compass.start();
}

@Override
            protected void onPause() {
            super.onPause();
            compass.stop();
}

@Override
            protected void onResume() {
            super.onResume();
            compass.start();
}

@Override
protected void onStop() {
            super.onStop();
            //Log.d(TAG, "stop compass");
            compass.stop();
}

/**
* A {@link FragmentPagerAdapter} that returns a fragment
    corresponding to
* one of the sections/tabs/pages.
*/
public class SectionsPagerAdapter extends FragmentPagerAdapter {

/**
```

```
* Initialises a new sectionPagerAdapter
*
* @param fm the fragment Manager
*/
public SectionsPagerAdapter(FragmentManager fm) {
        super(fm);
}

@Override
public Fragment getItem(int position) {
        // getItem is called to instantiate the fragment for the
            given page.
        // Return a DummySectionFragment (defined as a static
            inner class
        // below) with the page number as its lone argument.
        Fragment fragment = new OrientationVisualisationFragment()
            ;
        Bundle args = new Bundle();
        if(getCount()==1){
                position=OrientationVisualisationFragment.
                    SensorSelected-1;//cambiar los sensores ajustes
        }
        args.putInt(OrientationVisualisationFragment.
            ARG_SECTION_NUMBER, position + 1);
        fragment.setArguments(args);
        return fragment;
}

@Override
public int getCount() {
        // Show 6 total pages.
        return HomeActivity.num_fused_sensors;
}

@Override
public CharSequence getPageTitle(int position) {
        Locale l = Locale.getDefault();

        if(getCount()==1){
        position=OrientationVisualisationFragment.SensorSelected
            -1; //cambiar los sensores ajustes
        }
        switch (position) {
                case 0:
                        return getString(R.string.title_section1).
                            toUpperCase(l);
                case 1:
```

```
                    return getString(R.string.title_section2).
                        toUpperCase(l);
                case 2:
                    return getString(R.string.title_section3).
                        toUpperCase(l);
                case 3:
                    return getString(R.string.title_section4).
                        toUpperCase(l);
                case 4:
                    return getString(R.string.title_section5).
                        toUpperCase(l);
                case 5:
                    return getString(R.string.title_section6).
                        toUpperCase(l);
                }
                return null;
        }
}
private void showSettings()
{
        Intent i = new Intent(this, Settings.class);
        startActivity(i);
}
private void showHelpDialog()
{
        new MaterialDialog.Builder(this)
        .title(R.string.title)
        .content(R.string.input_content)
        .inputType(InputType.TYPE_CLASS_TEXT | InputType.
            TYPE_TEXT_VARIATION_PASSWORD)
        .input(R.string.input_hint, R.string.input_prefill, new
            MaterialDialog.InputCallback() {
        @Override
                public void onInput(MaterialDialog dialog,
                    CharSequence input) {
                    // Do something
                    if(input.toString().equals(getString(R.
                        string.password))){
                            showSettings();
                    } else {
                            Context context =
                                getApplicationContext();
                            CharSequence text = "Incorrect␣
                                password";
                            int duration = Toast.LENGTH_SHORT;

                            Toast toast = Toast.makeText(
```

```
                                        context, text, duration);
                                    toast.show();
                                }
                            }
                        }).show();
    }
}
```

A.1.4 Rotation.java

```
package com.example.david.tfg;

import java.util.Locale;

import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.res.Configuration;
import android.graphics.Color;
import android.graphics.drawable.Drawable;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import android.support.v4.view.ViewPager;
import android.util.Log;
import android.view.Display;
import android.view.Gravity;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.Surface;
import android.view.View;
import android.view.WindowManager;
import android.widget.LinearLayout;
import android.widget.RelativeLayout;

import com.erz.joysticklibrary.*;
import com.erz.joysticklibrary.JoyStick;

import com.example.david.tfg.R;

/**
* The main activity where the user can select which sensor-fusion
```

```
    he wants to try out
*
* @author Alexander Pacha
*
*/
public class Rotation extends FragmentActivity{

/**
* The {@link android.support.v4.view.PagerAdapter} that will
    provide
* fragments for each of the sections. We use a {@link android.
    support.v4.app.FragmentPagerAdapter} derivative,
* which
* will keep every loaded fragment in memory. If this becomes too
    memory
* intensive, it may be best to switch to a {@link android.support.
    v4.app.FragmentStatePagerAdapter}.
*/
SectionsPagerAdapter mSectionsPagerAdapter;

/**
* The {@link ViewPager} that will host the section contents.
*/
ViewPager mViewPager;
static public double angleTouch=0;
static public double powerTouch=0;


@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sensor_selection);

        // Create the adapter that will return a fragment for each
            of the three
        // primary sections of the app.
        mSectionsPagerAdapter = new SectionsPagerAdapter(
            getSupportFragmentManager());

        // Set up the ViewPager with the sections adapter.
        mViewPager = (ViewPager) findViewById(R.id.pager);
        mViewPager.setAdapter(mSectionsPagerAdapter);

        LinearLayout rotation = (LinearLayout) findViewById(R.id.
            rotation);
        rotation.setVisibility(View.VISIBLE);
```

```
        Connect.addMyBooleanListener(new com.example.david.tfg.
            ConnectionBooleanChangedListener5() {
                @Override
                public void OnMyBooleanChanged() {
                        if (Connect.getMyBoolean()) {
                                HomeActivity.horizontalRotation=
                                    true;
                                Intent intent = new Intent(
                                    Rotation.this,
                                SensorSelectionActivity.class);
                                intent.addFlags(Intent.
                                    FLAG_ACTIVITY_NO_HISTORY); //
                                    evitar problemas de interacción
                                     entre ventanas
                                startActivity(intent);
                                finish();
                        }
                }
        });

        // Check if device has a hardware gyroscope
        SensorChecker checker = new HardwareChecker((SensorManager
            ) getSystemService(SENSOR_SERVICE));
        if(!checker.IsGyroscopeAvailable()) {
                // If a gyroscope is unavailable, display a
                    warning.
                displayHardwareMissingWarning();
        }
}

private void displayHardwareMissingWarning() {
        AlertDialog ad = new AlertDialog.Builder(this).create();
        ad.setCancelable(false); // This blocks the 'BACK' button
        ad.setTitle(getResources().getString(R.string.
            gyroscope_missing));
        ad.setMessage(getResources().getString(R.string.
            gyroscope_missing_message));
        ad.setButton(DialogInterface.BUTTON_NEUTRAL, getResources
            ().getString(R.string.OK), new DialogInterface.
            OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int
                    which) {
                        dialog.dismiss();
                }
        });
        ad.show();
```

```
}
@Override
public void onBackPressed() {
        super.onBackPressed();
        this.finish();
}

/**
* A {@link FragmentPagerAdapter} that returns a fragment
    corresponding to
* one of the sections/tabs/pages.
*/
public class SectionsPagerAdapter extends FragmentPagerAdapter {

        /**
        * Initialises a new sectionPagerAdapter
        *
        * @param fm the fragment Manager
        */
        public SectionsPagerAdapter(FragmentManager fm) {
                super(fm);
        }

        @Override
        public Fragment getItem(int position) {
                // getItem is called to instantiate the fragment
                    for the given page.
                // Return a DummySectionFragment (defined as a
                    static inner class
                // below) with the page number as its lone
                    argument.
                Fragment fragment = new
                    OrientationVisualisationFragment();
                Bundle args = new Bundle();
                args.putInt("section_number", position+1); //
                    OrientationVisualisationFragment.
                    ARG_SECTION_NUMBER);
                fragment.setArguments(args);
                return fragment;
        }

        @Override
        public int getCount() {
        // Show 6 total pages.
                return 1;
        }
```

```
        @Override
        public CharSequence getPageTitle(int position) {
                Locale l = Locale.getDefault();
                return null;
        }
}

}
```

A.1.5 OrientationVisualisationFragment.java

```
package com.example.david.tfg;

import com.example.david.tfg.orientationProvider.
    AccelerometerCompassProvider;
import com.example.david.tfg.orientationProvider.
    CalibratedGyroscopeProvider;
import com.example.david.tfg.orientationProvider.
    GravityCompassProvider;
import com.example.david.tfg.orientationProvider.
    ImprovedOrientationSensor1Provider;
import com.example.david.tfg.orientationProvider.
    ImprovedOrientationSensor2Provider;
import com.example.david.tfg.orientationProvider.
    OrientationProvider;
import com.example.david.tfg.orientationProvider.
    RotationVectorProvider;
import com.example.david.tfg.representation.MatrixF4x4;
import com.example.david.tfg.representation.Quaternion;
import com.example.david.tfg.representation.Vector4f;

import android.content.Context;
import android.content.SharedPreferences;
import android.content.res.Resources;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.drawable.Drawable;
import android.hardware.SensorManager;
import android.opengl.GLSurfaceView;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.support.v4.app.Fragment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.View.OnLongClickListener;
```

```
import com.example.david.tfg.R;

/**
* A fragment that contains the same visualisation for different
    orientation providers
*/
public class OrientationVisualisationFragment extends Fragment {
/**
* The surface that will be drawn upon
*/
private GLSurfaceView mGLSurfaceView;
/**
* The class that renders the cube
*/
private CubeRenderer mRenderer;
/**
* The current orientation provider that delivers device
    orientation.
*/
private OrientationProvider currentOrientationProvider;

/**
* The fragment argument representing the section number for this
* fragment.
*/
public static String ARG_SECTION_NUMBER = "section_number";
public static Bitmap b ;
public static Vector4f vectorNewOrigen = new Vector4f();

static public int SensorSelected=1;
static public float sensibility=0.75f;
static public String ip;
static public int port;
static public int MaxLinealVel,MaxLinealAngle,alturaMax,areaMax;

@Override
public void onResume() {
        // Ideally a game should implement onResume() and onPause
            ()
        // to take appropriate action when the activity looses
            focus
        super.onResume();
        currentOrientationProvider.start();
        mGLSurfaceView.onResume();
}
```

```
@Override
public void onPause() {
        // Ideally a game should implement onResume() and onPause
           ()
        // to take appropriate action when the activity looses
           focus
        super.onPause();
        currentOrientationProvider.stop();
        mGLSurfaceView.onPause();
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState) {
        // Initialise the orientationProvider
        vectorNewOrigen.setXYZW(0,0,-2.5f,1);

        //Menu Settings
        PreferenceManager.setDefaultValues(getActivity(), R.xml.
           settings, false);
        SharedPreferences sharedPreferences = PreferenceManager.
           getDefaultSharedPreferences(getActivity());
        HomeActivity.compass = sharedPreferences.getBoolean("
           compass", true);

        SharedPreferences sharedPreferencesIP = PreferenceManager.
           getDefaultSharedPreferences(getActivity());
        ip = sharedPreferencesIP.getString("ip", "10.0.2.15");

        SharedPreferences sharedPreferencesPort =
           PreferenceManager.getDefaultSharedPreferences(
           getActivity());
        port = Integer.parseInt(sharedPreferencesPort.getString("
           port", "8080"));

        SharedPreferences sharedPreferencesMaxLinealVel =
           PreferenceManager.getDefaultSharedPreferences(
           getActivity());
        MaxLinealVel = Integer.parseInt(
           sharedPreferencesMaxLinealVel.getString("maxVel", "50")
           );

        SharedPreferences sharedPreferencesMaxLinealAngle =
           PreferenceManager.getDefaultSharedPreferences(
           getActivity());
        MaxLinealAngle = Integer.parseInt(
           sharedPreferencesMaxLinealAngle.getString("maxAngle", "
```

```
        50"));

    SharedPreferences sharedPreferencesMaxAltura =
        PreferenceManager.getDefaultSharedPreferences(
        getActivity());
    alturaMax = Integer.parseInt(sharedPreferencesMaxAltura.
        getString("maxAltura", "20"));

    SharedPreferences sharedPreferencesMaxArea =
        PreferenceManager.getDefaultSharedPreferences(
        getActivity());
    areaMax = Integer.parseInt(sharedPreferencesMaxArea.
        getString("maxArea", "7"));

    //Sensor Selection
    switch (getArguments().getInt(ARG_SECTION_NUMBER)) {
            case 1:
                    currentOrientationProvider = new
                        ImprovedOrientationSensor1Provider((
                        SensorManager) getActivity()
                    .getSystemService(SensorSelectionActivity.
                        SENSOR_SERVICE));

                    break;
            case 2:
                    currentOrientationProvider = new
                        ImprovedOrientationSensor2Provider((
                        SensorManager) getActivity()
                    .getSystemService(SensorSelectionActivity.
                        SENSOR_SERVICE));
                    break;
            case 3:
                    currentOrientationProvider = new
                        RotationVectorProvider((SensorManager)
                        getActivity().getSystemService(
                    SensorSelectionActivity.SENSOR_SERVICE));
                    break;
            case 4:
                    currentOrientationProvider = new
                        CalibratedGyroscopeProvider((
                        SensorManager) getActivity()
                    .getSystemService(SensorSelectionActivity.
                        SENSOR_SERVICE));
                    break;
            case 5:
                    currentOrientationProvider = new
                        GravityCompassProvider((SensorManager)
```

```
                        getActivity().getSystemService(
                    SensorSelectionActivity.SENSOR_SERVICE));
                    break;
            case 6:
                    currentOrientationProvider = new
                        AccelerometerCompassProvider((
                        SensorManager) getActivity()
                    .getSystemService(SensorSelectionActivity.
                        SENSOR_SERVICE));
                    break;
            default:
                    break;
        }

        // Create our Preview view and set it as the content of
            our Activity
        mRenderer = new CubeRenderer(getContext());
        mRenderer.setOrientationProvider(
            currentOrientationProvider);

        mGLSurfaceView = new GLSurfaceView(getActivity());
        mGLSurfaceView.setEGLConfigChooser(8, 8, 8, 8, 16, 0);
        mGLSurfaceView.setRenderer(mRenderer);

        mGLSurfaceView.setOnLongClickListener(new
            OnLongClickListener() {

        @Override
        public boolean onLongClick(View v) {
                if(HomeActivity.i!=3){
                mRenderer.togglescreenTouched();
                }
                    return true;
                }
        });
        return mGLSurfaceView;
}
}
```

A.1.6 Settings.java

```
package com.example.david.tfg;

/**
* Created by David on 03/07/2017.
*/
```

```
import android.os.Bundle;
import android.preference.CheckBoxPreference;
import android.preference.EditTextPreference;
import android.preference.ListPreference;
import android.preference.Preference;
import android.preference.PreferenceActivity;
import android.preference.PreferenceFragment;
import android.util.Log;

import com.example.david.tfg.R;

public class Settings extends PreferenceActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getFragmentManager().beginTransaction().replace(android.R.
            id.content, new SettingsFragment()).commit();

}

public static class SettingsFragment extends PreferenceFragment
{
        @Override
        public void onCreate(final Bundle savedInstanceState)
        {
                super.onCreate(savedInstanceState);
                addPreferencesFromResource(R.xml.settings);

                final ListPreference listPreference = (
                    ListPreference) findPreference("SensorFusion");
                listPreference.setOnPreferenceChangeListener(new
                    Preference.OnPreferenceChangeListener() {
                @Override
                        public boolean onPreferenceChange(
                            Preference preference, Object newValue)
                             {
                                String listValue = (String)
                                    newValue;
                                OrientationVisualisationFragment.
                                    SensorSelected=Integer.parseInt
                                    (listValue);
                        return true;
                }
                });
                final ListPreference languagePreference = (
                    ListPreference) findPreference("language");
```

```
languagePreference.setOnPreferenceChangeListener(
    new Preference.OnPreferenceChangeListener() {
        @Override
        public boolean onPreferenceChange(
            Preference preference, Object newValue)
             {
                String listValue = (String)
                    newValue;
                OrientationVisualisationFragment.
                    language=Integer.parseInt(
                    listValue);
                Locale localizacion;
                switch (
                    OrientationVisualisationFragment
                    .language){
                        case 1:
                                localizacion = new
                                    Locale("en", "
                                    EN");
                                break;
                        case 2:
                                localizacion = new
                                    Locale("es", "
                                    ES");
                                break;
                        case 3:
                                localizacion = new
                                    Locale("ca", "
                                    CA");
                                break;
                        default:
                                localizacion = new
                                    Locale("en", "
                                    EN");
                                break;
                        }
                Locale.setDefault(localizacion);
                Configuration config = new
                    Configuration();
                config.locale = localizacion;
                getActivity().getBaseContext().
                    getResources().
                    updateConfiguration(config,
                    getActivity().getBaseContext().
                    getResources().
                    getDisplayMetrics());
                final SharedPreferences prefs =
```

```
                        getActivity().
                        getSharedPreferences("
                        SettingsDroneControl", Context.
                        MODE_PRIVATE);

                    SharedPreferences.Editor editor =
                        prefs.edit();
                    editor.putInt("language",
                        OrientationVisualisationFragment
                        .language);
                    editor.commit();

                    refresh();
                    return true;
                }
            });
            final EditTextPreference sensibility = (
                EditTextPreference) findPreference("sense");
            sensibility.setOnPreferenceChangeListener(new
                Preference.OnPreferenceChangeListener() {
            @Override
                public boolean onPreferenceChange(
                    Preference preference, Object newValue)
                    {
                        String editValue = (String)
                            newValue;
                        OrientationVisualisationFragment.
                            sensibility=Float.parseFloat(
                            editValue+'f');
                return true;
            }
            });
            final EditTextPreference IPaddress = (
                EditTextPreference) findPreference("ip");
            IPaddress.setOnPreferenceChangeListener(new
                Preference.OnPreferenceChangeListener() {
            @Override
                public boolean onPreferenceChange(
                    Preference preference, Object newValue)
                    {
                        String editValue = (String)
                            newValue;
                        OrientationVisualisationFragment.
                            ip=editValue;
                        return true;
                }
            });
```

```
                final EditTextPreference Port = (
                    EditTextPreference) findPreference("port");
                Port.setOnPreferenceChangeListener(new Preference.
                    OnPreferenceChangeListener() {
                @Override
                        public boolean onPreferenceChange(
                            Preference preference, Object newValue)
                             {
                                String editValue = (String)
                                    newValue;
                                OrientationVisualisationFragment.
                                    port=Integer.parseInt(editValue
                                    );
                                return true;
                        }
                });
                final EditTextPreference MaxVel = (
                    EditTextPreference) findPreference("maxVel");
                MaxVel.setOnPreferenceChangeListener(new
                    Preference.OnPreferenceChangeListener() {
                @Override
                        public boolean onPreferenceChange(
                            Preference preference, Object newValue)
                             {
                                String editValue = (String)
                                    newValue;
                                OrientationVisualisationFragment.
                                    MaxLinealVel=Integer.parseInt(
                                    editValue);
                                return true;
                        }
                });
                final EditTextPreference MaxHeight = (
                    EditTextPreference) findPreference("maxAltura")
                    ;
                MaxHeight.setOnPreferenceChangeListener(new
                    Preference.OnPreferenceChangeListener() {
                @Override
                        public boolean onPreferenceChange(
                            Preference preference, Object newValue)
                             {
                                String editValue = (String)
                                    newValue;
                                OrientationVisualisationFragment.
                                    alturaMax=Integer.parseInt(
                                    editValue);
                                return true;
```

```
                }
            });
            final EditTextPreference MaxArea = (
                EditTextPreference) findPreference("maxArea");
            MaxArea.setOnPreferenceChangeListener(new
                Preference.OnPreferenceChangeListener() {
            @Override
                public boolean onPreferenceChange(
                    Preference preference, Object newValue)
                     {
                        String editValue = (String)
                            newValue;
                        OrientationVisualisationFragment.
                            areaMax=Integer.parseInt(
                            editValue);
                        return true;
                }
            });
            final CheckBoxPreference checkboxpreference = (
                CheckBoxPreference) findPreference("compass");
            checkboxpreference.setOnPreferenceChangeListener(
                new Preference.OnPreferenceChangeListener() {
            @Override
                public boolean onPreferenceChange(
                    Preference preference, Object newValue)
                     {
                        HomeActivity.compass = Boolean.
                            valueOf(newValue.toString());
                        return true;
                }
            });
        }
}

        public static void refresh(){
                Intent intent=new Intent();
                intent.setClass(HomeActivity.home, HomeActivity.
                    home.getClass());
                HomeActivity.home.finish();
                HomeActivity.home.startActivity(intent);
        }
}
```

A.1.7 CubeRender.java

```
package com.example.david.tfg;
```

```
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

import com.example.david.tfg.orientationProvider.
    OrientationProvider;
import com.example.david.tfg.representation.MatrixF4x4;
import com.example.david.tfg.representation.Quaternion;
import com.example.david.tfg.representation.Vector3f;
import com.example.david.tfg.representation.Vector4f;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;
import android.opengl.GLUtils;
import android.os.Handler;
import android.util.Log;

/**
* Class that implements the rendering of a Point with the current
    rotation of the device that is provided by a
* OrientationProvider
*
* @author David Arévalo Jiménez
*
*/
public class CubeRenderer implements GLSurfaceView.Renderer {
/**
* The colour-cube that is drawn repeatedly
*/
private Circle mPoint;
private Circle mCircle0;
private Circle mCircle1;
private Circle mCircle2;
private Circle mCircle3;
private Circle mCircle4;
private Cuadrado mCuadrado;
private Lines mLines;

/**
* The current provider of the device orientation.
*/
private OrientationProvider orientationProvider = null;
private Quaternion quaternion = new Quaternion();
private MatrixF4x4 matrixRot = new MatrixF4x4();
private Vector4f vectorOrigen = new Vector4f();
```

```
private Vector4f vectorPointMoved = new Vector4f();
private Vector4f vectorDiference = new Vector4f();
private float angulo;
private float angulo2D;
private float moduloA;
private float moduloB;
private float moduloC;
private float longitud;
float x;
float y;
float z;
float w;
private int port;
private String ip;


private float amarillo[]= new float[]{1.0f, 1.0f, 0.0f, 1.0f}; //
    AMARILLO;
private float verdeLight[]= new float[]{0.565f, 0.933f, 0.565f,
    1.0f}; //VERDE CLARO
private float verde[]= new float[]{0.0f, 1.0f, 0.0f, 1.0f}; //VERDE
private float verdeDark[]= new float[]{0.000f, 0.392f, 0.000f, 1.0
    f}; //VERDE OSCURO
private float tomate[]= new float[]{1.0f, 0.388f, 0.278f, 1.0f};
    //TOMATE
private float naranja[]= new float[]{1.0f, 0.271f, 0.0f, 1.0f}; //
    NARANJA
private float rojo[]= new float[]{1.0f, 0.0f, 0.0f, 1.0f}; //ROJO

private Context context;


/**
* Initialises a new CubeRenderer
*/
public CubeRenderer(Context context) {
        this.context = context;
        //colors= new float[]{1.0f, 1.0f, 0.0f, 1.0f}; //AMARILLO
        mPoint = new Circle(0.15f,1,amarillo);
        //colors= new float[]{0.0f, 1.0f, 0.0f, 1.0f};//VERDE LIMA
        mCircle0 = new Circle(0.02f,0,verdeLight);
        //colors= new float[]{0.196f, 0.804f, 0.196f, 1.0f};//
            VERDE OSCURO
        mCircle1 = new Circle(0.5f,0,verde);
        //colors= new float[]{1.0f, 0.388f, 0.278f, 1.0f}; //
            TOMATE
        mCircle2 = new Circle(1f,0,verdeDark);
```

```
        //colors= new float[]{1.0f, 0.271f, 0.0f, 1.0f}; //NARANJA
        mCircle3 = new Circle(1.5f,0,naranja);
        //colors= new float[]{1.0f, 0.0f, 0.0f, 1.0f};//ROJO
        mCircle4 = new Circle(2f,0,rojo);

        mLines = new Lines();

        mCuadrado = new Cuadrado();

}

/**
* Sets the orientationProvider of this renderer. Use this method
    to change which sensor fusion should be currently
* used for rendering the cube. Simply exchange it with another
    orientationProvider and the cube will be rendered
* with another approach.
*
* @param orientationProvider The new orientation provider that
    delivers the current orientation of the device
*/
public void setOrientationProvider(OrientationProvider
    orientationProvider) {
        this.orientationProvider = orientationProvider;
}


@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        // dither is enabled by default, we don't need it
        gl.glDisable(GL10.GL_DITHER);

        mCuadrado.loadGLTexture(gl, this.context);

        // clear screen in black
        gl.glClearColor(0, 0, 0, 1);
        // Configura el buffer de profundidad
        gl.glClearDepthf(1.0f);
        // Modo de renderizado de la profundidad
        gl.glEnable(GL10.GL_DEPTH_TEST);
        // Función de comparación de la profundidad
        gl.glDepthFunc(GL10.GL_LEQUAL);
        // Cómo se calcula la perspectiva
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.
            GL_NICEST);

        port=OrientationVisualisationFragment.port;
```

```
        ip=OrientationVisualisationFragment.ip;
}


/**
* Perform the actual rendering of the cube for each frame
*
* @param gl The surface on which the cube should be rendered
*/
public void onDrawFrame(GL10 gl) {
        // clear screen
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.
            GL_DEPTH_BUFFER_BIT);

        // set-up modelview matrix
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        gl.glTranslatef(0.0f,0.0f, -3.0f);//coloca el cubo a una
            distancia 3 de la pantalla
        if (HomeActivity.i!=0){
                mCircle0.draw(gl);
                mCircle1.draw(gl);
                mCircle2.draw(gl);
                mCircle3.draw(gl);
                mCircle4.draw(gl);
                mLines.draw(gl);
        }
if (orientationProvider != null) {
// Get the rotation from the current orientationProvider as
    quaternion
orientationProvider.getQuaternion(quaternion);
orientationProvider.getRotationMatrix(matrixRot);

vectorOrigen.setXYZW(0,0,-2.5f,1);
vectorPointMoved.setXYZW(0,0,-2.5f,1);

matrixRot.multiplyVector4fByMatrix(vectorPointMoved);

if (!screenTouched) {
x=vectorPointMoved.getX();
y=vectorPointMoved.getY();
z=vectorPointMoved.getZ();
w=vectorPointMoved.getW();

moduloA=vectorPointMoved.modulo(); //nuevo punto
moduloB=vectorOrigen.modulo();//origen
vectorPointMoved.subtract(vectorOrigen,vectorDiference);
```

```
moduloC=vectorDiference.modulo(); //distancia entre ambos puntos

if(!HomeActivity.horizontalRotation && vectorPointMoved.getX()
    <-0.5 && vectorPointMoved.getY()<0.5 &&
vectorPointMoved.getY()>-0.5 && vectorPointMoved.getZ()<0){
        Connect.setMyBoolean(true);
}

Log.v("CoordenadasXYZ", "x=" + vectorPointMoved.getX() +"␣y=" +
    vectorPointMoved.getY()+ "z=" + vectorPointMoved.getZ() + "w="
    + vectorPointMoved.getW() +'\n');
if(HomeActivity.horizontalRotation){
        if (HomeActivity.i==0) {
                gl.glRotatef((float) (-2.0f * Math.acos(quaternion
                    .getW()) * 180.0f / Math.PI), quaternion.getX()
                    , quaternion.getY(), -quaternion.getZ());
                } if (HomeActivity.i==1 || HomeActivity.i==2 ||
                                        HomeActivity.i==3) {
                        if(HomeActivity.i==1 || HomeActivity.i==2)
                            {
                                angulo=(float) Math.acos((moduloA*
                                    moduloA+moduloB*moduloB-moduloC
                                    *moduloC)/(2*moduloA*moduloB));
                                     //teorema del coseno
                                longitud = vectorOrigen.modulo()*
                                    angulo; //arco
                                longitud =
                                    OrientationVisualisationFragment
                                    .sensibility*longitud;
                                angulo = (float) Math.atan2(
                                    vectorPointMoved.getY(),
                                    vectorPointMoved.getX());
                        } else {
                                angulo=(float)(-
                                    SensorSelectionActivity.
                                    angleTouch+Math.PI);
                                longitud=(float)
                                    SensorSelectionActivity.
                                    powerTouch*2/100;
                        }
                        Log.v("Touch","angle:␣"+angulo+"␣"+"power:
                            ␣"+longitud);

                        if(longitud>2){
                                longitud=2;
                        }
                        gl.glTranslatef((float) (longitud*Math.cos
```

```
                        (angulo)),(float) (longitud*Math.sin(
                        angulo)), 0.0f); //rota a 2 del centro
                }
                //Adjust angle to server
                if(HomeActivity.i==1 || HomeActivity.i==2){
                        //cambiar valores de angulos de
                            (-3.14?0?3.14) a (0?6.28) y girar
                        angulo=(float)(angulo+Math.PI*2.5);
                        if(angulo>Math.PI*2){
                                angulo=(float)(angulo-Math.PI*2);
                        }
                        Log.v("CoordenadasA", "yaw=␣" +angulo+'\n'
                            );

                        } else { //joystick
                                //girar sistema 90º en sentido
                                    horario
                                angulo=(float)(angulo+Math.PI*0.5)
                                    ;
                                if(angulo>Math.PI*2){
                                        angulo=(float)(angulo-Math
                                            .PI*2);
                                }
                                Log.v("CoordenadasA", "yaw=␣" +
                                    angulo+'\n');
                        }
                        Log.v("messages", "angulo=␣" +String.
                            format("%.6f", angulo/*180/Math.PI*/) +
                            "long=␣" + String.format("%.6f",
                            longitud)+'\n'); //angulos de 0 a 6.28
                        if(longitud==0){
                        angulo=0;
                }
        }
} else if (HomeActivity.horizontalRotation) {
        OrientationVisualisationFragment.vectorNewOrigen.setXYZW(x
            ,y,z,w);

        moduloA=vectorPointMoved.modulo(); //nuevo punto
        moduloB=OrientationVisualisationFragment.vectorNewOrigen.
            modulo(); //origen
        vectorPointMoved.subtract(OrientationVisualisationFragment
            .vectorNewOrigen,vectorDiference);
        moduloC=vectorDiference.modulo(); //distancia entre ambos
            puntos

        Log.v("Coordenadas", "x=" + vectorOrigen.getX() +"␣y=" +
```

```
            vectorOrigen.getY() + "z=" + vectorOrigen.getZ() + "w="
             + vectorOrigen.getW() +'\n');

        if (HomeActivity.i==0) {
                gl.glRotatef((float) (-2.0f * Math.acos(quaternion
                    .getW()) * 180.0f / Math.PI), quaternion.getX()
                    , quaternion.getY(), -quaternion.getZ());
        } if (HomeActivity.i==1 || HomeActivity.i==2) {
                angulo=(float) Math.acos((moduloA*moduloA+moduloB*
                    moduloB-moduloC*moduloC)/(2*moduloA*moduloB));
                    //teorema del coseno
                longitud = vectorOrigen.modulo()*angulo; //arco
                longitud = OrientationVisualisationFragment.
                    sensibility*longitud;

                angulo2D = (float) Math.atan2((vectorDiference.
                    getY()), (vectorDiference.getX()));
                if(angulo2D>=Math.PI){
                        vectorPointMoved.setZ(-vectorPointMoved.
                            getZ());
                }
                if(longitud>2){
                        longitud=2;
                        gl.glTranslatef((float) (longitud*Math.cos
                            (angulo2D)),(float) (longitud*Math.sin(
                            angulo2D)), 0.0f); //rota a 1 del
                            centro
                } else {
                        gl.glTranslatef((float) (longitud*Math.cos
                            (angulo2D)),(float) (longitud*Math.sin(
                            angulo2D)), 0.0f);
                }
        }
        if(HomeActivity.i==1 || HomeActivity.i==2){

                //cambiar valores de angulos de (-3.14?0?3.14) a
                    (0?6.28)
                angulo2D=(float)(angulo2D+Math.PI*2.5);

                if(angulo2D>Math.PI*2){
                        angulo2D=(float)(angulo2D-Math.PI*2);
                }
        } else {
                //girar sistema 90º en sentido horario
                angulo2D=(float)(angulo2D+Math.PI*0.5);
                if(angulo2D>Math.PI*2){
                        angulo2D=(float)(angulo2D-Math.PI*2);
```

```
                }
        }
        Log.v("messages2D", "angulo2D=␣" +String.format("%.6f",
            angulo2D/*180/Math.PI*/) +"long=␣" + String.format("%.6
            f", longitud)+'\n'); //angulo2Ds de 0 a 6.28
        if(longitud==0){
                angulo2D=0;
        }
        angulo=angulo2D;
}
}


        // draw our object
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);

        if (HomeActivity.i==0){
                mCuadrado.draw(gl);
        } else {
                mPoint.draw(gl);
        }
        HomeActivity.angulo=angulo;//global variables to avoid
            client problems
        HomeActivity.longitud=longitud;

        if(HomeActivity.i==1 || HomeActivity.i==3){
                Client myClient = new Client(ip,port);//192.168.1.
                    X 8080
                myClient.execute();
        }
}

/**
* Update view-port with the new surface
*
* @param gl the surface
* @param width new width
* @param height new height
*/
public void onSurfaceChanged(GL10 gl, int width, int height) {
                if(height == 0) {
                                        //Prevent A Divide By Zero By
                height = 1;
                                //Making Height Equal One
        }

        gl.glViewport(0, 0, width, height);       //Reset The
```

```
            Current Viewport
        gl.glMatrixMode(GL10.GL_PROJECTION);      //Select The
            Projection Matrix
        gl.glLoadIdentity();                                    //
            Reset The Projection Matrix

        //Calculate The Aspect Ratio Of The Window
        GLU.gluPerspective(gl, 90.0f, (float)width / (float)height
            , 0.1f, 100.0f);

        gl.glMatrixMode(GL10.GL_MODELVIEW);       //Select The
            Modelview Matrix
        gl.glLoadIdentity();                                    //
            Reset The Modelview Matrix
}

/**
* Flag indicating whether you want to view inside out, or outside
    in
*/
private boolean screenTouched = false;

/**
* Toggles whether the cube will be shown inside-out or outside in.
*/
public void togglescreenTouched() {
        this.screenTouched = !screenTouched;
}

}
```

A.1.8 Client.java

```
package com.example.david.tfg;

import java.io.BufferedReader;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.Socket;
import java.net.UnknownHostException;
import android.os.AsyncTask;
```

```
import android.util.Log;
import android.widget.TextView;

public class Client extends AsyncTask<Void, Void, Void> {

        String dstAddress;
        int dstPort;
        String response = "";

        public Client(String addr, int port) {
                dstAddress = addr;
                dstPort = port;
        }

        @Override
        protected Void doInBackground(Void... arg0) {

                Socket socket = null;

                try {
                        socket = new Socket(
                            OrientationVisualisationFragment.ip,
                            OrientationVisualisationFragment.port);

                        PrintStream output = new PrintStream(
                            socket.getOutputStream());

                        output.print(String.format("%.6f",
                            HomeActivity.angulo)+"␣"+String.format(
                            "%.6f", HomeActivity.longitud)+"␣"+
                        HomeActivity.mode+"␣"+
                            OrientationVisualisationFragment.
                            MaxLinealVel+"␣"+
                        OrientationVisualisationFragment.
                            MaxLinealAngle+"␣"+HomeActivity.altura+
                            "␣"+
                        OrientationVisualisationFragment.alturaMax
                            +"␣"+OrientationVisualisationFragment.
                            areaMax+"␣");

                        output.flush(); //limpia el mensaje
                        output.close();


                } catch (UnknownHostException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
```

```
                        response = "UnknownHostException: " + e.
                            toString();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                        response = "IOException: " + e.toString();
                } finally {
                        if (socket != null) {
                                try {
                                        socket.close();
                                } catch (IOException e) {
                                        // TODO Auto-generated
                                            catch block
                                        e.printStackTrace();
                                }
                        }
                }
                return null;
        }

        @Override
                protected void onPostExecute(Void result) {
                Log.v("Server","Respuesta: "+response +'\n');
                super.onPostExecute(result);
        }
}
```

A.1.9 Circle.java

```
package com.example.david.tfg;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;

public class Circle {

        // Circle variables
        int circlePoints = 40;
        float radius = 1.5f;
        float center_x = 0.0f;
        float center_y = 0.0f;
        int types;
        float colors[];

        // Outer vertices of the circle i.e. excluding the
```

```
        center_x, center_y
    int circumferencePoints = circlePoints-1;

    // Circle vertices and buffer variables
    int vertices = 0;
    float circleVertices[] = new float[circlePoints*2];
    private FloatBuffer toiletBuff; // 4 bytes per float

    // Color values
    private float rgbaValues[] = {
            1f,     0f,  0f,     0f,
            0f,  1f,  0f,  0f,
            0f,     0f,  1f,     0f,
    };

    private FloatBuffer colorBuff;

    public Circle(float radio, int type, float[] color)
    {
            radius=radio;
            types=type;
            colors=color;
            // The initial buffer values
            circleVertices[vertices++] = center_x;
            circleVertices[vertices++] = center_y;

            // Set circle vertices values
            for (int i = 0; i < circumferencePoints; i++)
            {
                    float percent = (i / (float) (
                        circumferencePoints - 1));
                    float radians = (float) (percent * 2 *
                        Math.PI);

                    // Vertex position
                    float outer_x = (float) (center_x + radius
                         * Math.cos(radians));
                    float outer_y = (float) (center_y + radius
                         * Math.sin(radians));

                    circleVertices[vertices++] = outer_x;
                    circleVertices[vertices++] = outer_y;
            }

            // Float buffer short has four bytes
            ByteBuffer toiletByteBuff = ByteBuffer
            .allocateDirect(circleVertices.length * 4);
```

```
            // Garbage collector won't throw this away
            toiletByteBuff.order(ByteOrder.nativeOrder());
            toiletBuff = toiletByteBuff.asFloatBuffer();
            toiletBuff.put(circleVertices);
            toiletBuff.position(0);
    }

    // Draw methods
    public void draw(GL10 gl) {
            gl.glDisable(GL10.GL_TEXTURE_2D);
            gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
            // Establecemos el color del triángulo en modo
               RGBA
            gl.glColor4f(colors[0], colors[1], colors[2],
               colors[3]);

            // Dibujamos al revés que las agujas del reloj
            gl.glFrontFace(GL10.GL_CCW);
            // Indicamos el nº de coordenadas (3), el tipo de
               datos de la
            // matriz (float), la separación en la matriz de
               los vértices
            // (0) y el buffer con los vértices
            gl.glVertexPointer(2, GL10.GL_FLOAT, 0, toiletBuff
               );
            // Indicamos al motor OpenGL que le hemos pasado
               una matriz de
            // vértices
            gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
            gl.glDisableClientState(GL10.GL_COLOR_ARRAY);

            // Dibujamos la superficie
            if(types==0){
                    // Draw hollow circle
                    gl.glDrawArrays(GL10.GL_LINE_LOOP, 1,
                       circumferencePoints);
            } else {
                    // Draw circle as filled shape
                    gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 1,
                       circumferencePoints);// 1 para no
                       empezar en el centro
            }

            // Desactivamos el buffer de los vértices
            gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
            gl.glDisableClientState(GL10.GL_COLOR_ARRAY); //
```

```
                    super importante
        }
}
```

A.1.10 Lines.java

```
package com.example.david.tfg;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;

import javax.microedition.khronos.opengles.GL10;

import static javax.microedition.khronos.opengles.GL10.
    GL_COLOR_BUFFER_BIT;
import static javax.microedition.khronos.opengles.GL10.
    GL_DEPTH_BUFFER_BIT;

public class Lines {
        // Indica si deseamos colorear el triángulo
        private boolean conColor;
        // Buffer de tipo float que usamos para pasar
        // a la librería OpenGL los vértices del triángulo
        private FloatBuffer bufferVertices;
        // Vértices del triángulo
        private float vertices[] =
        {
                0.0f, 2, 0.0f, // Arriba
                0.0f, -2, 0.0f, // aBAJO
                -2, 0.0f, 0.0f, // Abajo izquierda
                2, 0.0f, 0.0f, // Abajo derecha
        };
        // Constructor del triángulo
        public Lines() {
                // Definimos el buffer con los vértices del
                    polígono.
                // Un número float tiene 4 bytes de longitud, así
                    que
                // multiplicaremos x 4 el número de vértices.
                ByteBuffer byteBuf=ByteBuffer.allocateDirect(
                    vertices.length*4);
                // Establecemos el orden de los bytes en el buffer
                     con el valor
                // nativo (es algo así como indicar cómo se leen
                    los bytes de
                // izq a dcha o al revés).
```

```
                byteBuf.order(ByteOrder.nativeOrder());
                // Asignamos el nuevo buffer al buffer de esta
                    clase
                bufferVertices = byteBuf.asFloatBuffer();
                // Introducimos los vértices en el buffer
                bufferVertices.put(vertices);
                // Movemos la posición del buffer al inicio
                bufferVertices.position(0);
                // Guardamos si es necesario colorear el polígono
        }
        // Método que invoca el Renderer cuando debe dibujar el
            triángulo
        public void draw(GL10 gl) {
                // Dibujamos al revés que las agujas del reloj

                gl.glColor4f(0.5f, 0.5f, 1.0f, 1.0f);
                gl.glFrontFace(GL10.GL_CCW);
                // Indicamos el nº de coordenadas (3), el tipo de
                    datos de la
                // matriz (float), la separación en la matriz de
                    los vértices
                // (0) y el buffer con los vértices
                gl.glVertexPointer(3, GL10.GL_FLOAT, 0,
                    bufferVertices);
                // Indicamos al motor OpenGL que le hemos pasado
                    una matriz de
                // vértices
                gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);

                // Dibujamos la superficie mediante la matriz en
                    el modo
                // triángulo
                gl.glDrawArrays(GL10.GL_LINES, 0, vertices.length
                    /3 );
                // Desactivamos el buffer de los vértices
                gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);

                gl.glDisableClientState(GL10.GL_COLOR_ARRAY); //
                    super importante
        }
} // end clase
```

A.1.11 Cuadrado.java

```
package com.example.david.tfg;

import java.io.IOException;
```

```
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;

import javax.microedition.khronos.opengles.GL10;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.opengl.GLUtils;

import com.example.david.tfg.R;

/**
* This class is an object representation of
* a Cube containing the vertex information,
* texture coordinates, the vertex indices
* and drawing functionality, which is called
* by the renderer.
*
* @author INsanityDesign
*/
public class Cuadrado {

        /** The buffer holding the vertices */
        private FloatBuffer vertexBuffer;
        /** The buffer holding the texture coordinates */
        private FloatBuffer textureBuffer;
        /** The buffer holding the indices */
        private ByteBuffer indexBuffer;

        /** Our texture pointer */
        private int[] textures = new int[1];

        /**
        * The initial vertex definition
        *
        * Note that each face is defined, even
        * if indices are available, because
        * of the texturing we want to achieve
        */
        private float vertices[] = {
                //Vertices according to faces
                -1.0f, -1.0f, 1.0f, //Vertex 0
                1.0f, -1.0f, 1.0f,  //v1
                -1.0f, 1.0f, 1.0f,  //v2
```

```
            1.0f, 1.0f, 1.0f,   //v3
    };

    /** The initial texture coordinates (u, v) */
    private float texture[] = {
            //Mapping coordinates for the vertices
            0.0f, 0.0f,
            0.0f, 1.0f,
            1.0f, 0.0f,
            1.0f, 1.0f,
    };

    /** The initial indices definition */
    private byte indices[] = {
            //Faces definition
            0,1,3, 0,3,2,
    };

    /**
    * The Cuadrado constructor.
    *
    * Initiate the buffers.
    */
    public Cuadrado() {
            //
            ByteBuffer byteBuf = ByteBuffer.allocateDirect(
                vertices.length * 4);
            byteBuf.order(ByteOrder.nativeOrder());
            vertexBuffer = byteBuf.asFloatBuffer();
            vertexBuffer.put(vertices);
            vertexBuffer.position(0);

            //
            byteBuf = ByteBuffer.allocateDirect(texture.length
                * 4);
            byteBuf.order(ByteOrder.nativeOrder());
            textureBuffer = byteBuf.asFloatBuffer();
            textureBuffer.put(texture);
            textureBuffer.position(0);

            //
            indexBuffer = ByteBuffer.allocateDirect(indices.
                length);
            indexBuffer.put(indices);
            indexBuffer.position(0);
    }
```

```
        /**
        * The object own drawing function.
        * Called from the renderer to redraw this instance
        * with possible changes in values.
        *
        * @param gl - The GL Context
        */
        public void draw(GL10 gl) {

                gl.glDisableClientState(GL10.GL_COLOR_ARRAY); //
                    super importante

                // Activamos la texturas en 2D (superficie)
                gl.glEnable(GL10.GL_TEXTURE_2D);
                //Bind our only previously generated texture in
                    this case
                gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[0]);

                //Point to our buffers
                gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
                gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY
                    );

                //Set the face rotation
                gl.glFrontFace(GL10.GL_CCW);

                //Enable the vertex and texture state
                gl.glVertexPointer(3, GL10.GL_FLOAT, 0,
                    vertexBuffer);
                gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0,
                    textureBuffer);

                //Draw the vertices as triangles, based on the
                    Index Buffer information
                gl.glDrawElements(GL10.GL_TRIANGLES, indices.
                    length, GL10.GL_UNSIGNED_BYTE, indexBuffer);

                //Disable the client state before leaving
                gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
                gl.glDisableClientState(GL10.
                    GL_TEXTURE_COORD_ARRAY);
                gl.glDisableClientState(GL10.GL_TEXTURE_2D);
        }

        /**
        * Load the textures
        *
```

```
        * @param gl - The GL Context
        * @param context - The Activity context
        */
        public void loadGLTexture(GL10 gl, Context context) {
                //Get the texture from the Android resource
                    directory
                InputStream is = context.getResources().
                    openRawResource(R.drawable.drone);
                Bitmap bitmap = null;
                try {
                        //BitmapFactory is an Android graphics
                            utility for images
                        bitmap = BitmapFactory.decodeStream(is);

                } finally {
                        //Always clear and close
                        try {
                        is.close();
                        is = null;
                        } catch (IOException e) {
                        }
                }

                //Generate one texture pointer...
                gl.glGenTextures(1, textures, 0);
                //...and bind it to our array
                gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[0]);

                //Create Nearest Filtered Texture
                gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.
                    GL_TEXTURE_MIN_FILTER, GL10.GL_NEAREST);
                gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.
                    GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);

                //Different possible texture parameters, e.g. GL10
                    .GL_CLAMP_TO_EDGE
                gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.
                    GL_TEXTURE_WRAP_S, GL10.GL_REPEAT);
                gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.
                    GL_TEXTURE_WRAP_T, GL10.GL_REPEAT);

                //Use the Android GLUtils to specify a two-
                    dimensional texture image from our bitmap
                GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap,
                    0);

                //Clean up
```

```
                bitmap.recycle();
        }
}
```

A.1.12 Joystick.java

```
package com.example.david.tfg;

import android.content.Context;
import android.content.res.TypedArray;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.RectF;
import android.util.AttributeSet;
import android.view.GestureDetector;
import android.view.MotionEvent;
import android.view.View;

import com.example.david.tfg.R;

/**
* Created by edgarramirez on 10/30/15.
*/
public class JoyStick extends View implements GestureDetector.
    OnGestureListener, GestureDetector.OnDoubleTapListener {

        public static final int DIRECTION_CENTER = -1;
        public static final int DIRECTION_LEFT = 0;
        public static final int DIRECTION_LEFT_UP = 1;
        public static final int DIRECTION_UP = 2;
        public static final int DIRECTION_UP_RIGHT = 3;
        public static final int DIRECTION_RIGHT = 4;
        public static final int DIRECTION_RIGHT_DOWN = 5;
        public static final int DIRECTION_DOWN = 6;
        public static final int DIRECTION_DOWN_LEFT = 7;

        public static final int TYPE_8_AXIS = 11;
        public static final int TYPE_4_AXIS = 22;
        public static final int TYPE_2_AXIS_LEFT_RIGHT = 33;
        public static final int TYPE_2_AXIS_UP_DOWN = 44;

        private JoyStickListener listener;
        private Paint paint;
        private RectF temp;
```

```
        private GestureDetector gestureDetector;
        private int direction = DIRECTION_CENTER;
        private int type = TYPE_8_AXIS;
        private float centerX;
        private float centerY;
        private float posX;
        private float posY;
        private float radius;
        private float buttonRadius;
        private double power = 0;
        private double angle = 0;

        //Background Color
        private int padColor;

        //Stick Color
        private int buttonColor;

        //Keeps joystick in last position
        private boolean stayPut;

        //Button Size percentage of the minimum(width, height)
        private int percentage = 25;

        //Background Bitmap
        private Bitmap padBGBitmap = null;

        //Button Bitmap
        private Bitmap buttonBitmap = null;

        public interface JoyStickListener {
                void onMove(JoyStick joyStick, double angle,
                    double power, int direction);

                void onTap();

                void onDoubleTap();
        }

        public JoyStick(Context context) {
                this(context, null);
        }

        public JoyStick(Context context, AttributeSet attrs) {
                super(context, attrs);
                paint = new Paint();
                paint.setStyle(Paint.Style.FILL);
```

```
                paint.setAntiAlias(true);
                paint.setFilterBitmap(true);

                temp = new RectF();

                gestureDetector = new GestureDetector(context,
                    this);
                gestureDetector.setIsLongpressEnabled(false);
                gestureDetector.setOnDoubleTapListener(this);

                padColor = Color.WHITE;
                buttonColor = Color.RED;

                if (attrs != null) {
                        TypedArray typedArray = context.
                            obtainStyledAttributes(attrs, R.
                            styleable.JoyStick);
                        if (typedArray != null) {
                                padColor = typedArray.getColor(R.
                                    styleable.JoyStick_padColor,
                                    Color.WHITE);
                                buttonColor = typedArray.getColor(
                                    R.styleable.
                                    JoyStick_buttonColor, Color.RED
                                    );
                                stayPut = typedArray.getBoolean(R.
                                    styleable.JoyStick_stayPut,
                                    false);
                                percentage = typedArray.getInt(R.
                                    styleable.JoyStick_percentage,
                                    25);
                                if (percentage > 50) percentage =
                                    50;
                                if (percentage < 25) percentage =
                                    25;

                                int padResId = typedArray.
                                    getResourceId(R.styleable.
                                    JoyStick_backgroundDrawable,
                                    -1);
                                int buttonResId = typedArray.
                                    getResourceId(R.styleable.
                                    JoyStick_buttonDrawable, -1);

                                if (padResId > 0) {
                                        padBGBitmap =
                                            BitmapFactory.
```

```
                                        decodeResource(
                                        getResources(),
                                        padResId);
                                }
                                if (buttonResId > 0) {
                                        buttonBitmap =
                                        BitmapFactory.
                                        decodeResource(
                                        getResources(),
                                        buttonResId);
                                }
                                typedArray.recycle();
                        }
                }
        }

        @Override
        protected void onMeasure(int widthMeasureSpec, int
            heightMeasureSpec) {
                super.onMeasure(widthMeasureSpec,
                    heightMeasureSpec);
                float width = MeasureSpec.getSize(widthMeasureSpec
                    );
                float height = MeasureSpec.getSize(
                    heightMeasureSpec);
                centerX = width / 2;
                centerY = height / 2;
                float min = Math.min(width, height);
                posX = centerX;
                posY = centerY;
                buttonRadius = (min / 2f * (percentage / 100f));
                radius = (min / 2f * ((100f - percentage) / 100f))
                    ;
        }

        @Override
        public void onDraw(Canvas canvas) {
                if (canvas == null) return;
                if (padBGBitmap == null) {
                        paint.setColor(padColor);
                        canvas.drawCircle(centerX, centerY, radius
                            , paint);
                } else {
                        temp.set(centerX - radius, centerY -
                            radius, centerX + radius, centerY +
                            radius);
                        canvas.drawBitmap(padBGBitmap, null, temp,
```

```
                        paint);
                }
                if (buttonBitmap == null) {
                        paint.setColor(buttonColor);
                        canvas.drawCircle(posX, posY, buttonRadius
                            , paint);
                } else {
                        temp.set(posX - buttonRadius, posY -
                            buttonRadius, posX + buttonRadius, posY
                             + buttonRadius);
                        canvas.drawBitmap(buttonBitmap, null, temp
                            , paint);
                }
        }

        @Override
        public boolean onTouchEvent(MotionEvent event) {
                gestureDetector.onTouchEvent(event);

                switch (event.getAction()) {
                        case MotionEvent.ACTION_DOWN:
                        case MotionEvent.ACTION_MOVE:
                                posX = event.getX();
                                posY = event.getY();

                                if (type == TYPE_2_AXIS_LEFT_RIGHT
                                    ) {
                                posY = centerY;
                                } else if (type ==
                                    TYPE_2_AXIS_UP_DOWN) {
                                posX = centerX;
                                } else if (type == TYPE_4_AXIS) {
                                if (Math.abs(posX - centerX) >
                                    Math.abs(posY - centerY)) posY
                                    = centerY;
                                else posX = centerX;
                                }

                                float abs = (float) Math.sqrt((
                                    posX - centerX) * (posX -
                                    centerX)
                                + (posY - centerY) * (posY -
                                    centerY));
                                if (abs > radius) {
                                posX = ((posX - centerX) * radius
                                    / abs + centerX);
                                posY = ((posY - centerY) * radius
```

```
                                / abs + centerY);
                            }

                            angle = Math.atan2(centerY - posY,
                                 centerX - posX);

                            power = (100 * Math.sqrt((posX -
                                centerX)
                            * (posX - centerX) + (posY -
                                centerY)
                            * (posY - centerY)) / radius);

                            direction = calculateDirection(
                                Math.toDegrees(angle));

                            invalidate();
                            break;
                            case MotionEvent.ACTION_UP:
                            case MotionEvent.ACTION_CANCEL:
                            if (!stayPut) {
                            posX = centerX;
                            posY = centerY;
                            direction = DIRECTION_CENTER;
                            angle = 0;
                            power = 0;
                            invalidate();
                            }
                            break;
                }

                if (listener != null) {
                        listener.onMove(this, angle, power,
                            direction);
                }
                return true;
        }

        @Override
        public boolean onDown(MotionEvent motionEvent) {
                return true;
        }

        @Override
        public void onShowPress(MotionEvent motionEvent) {}

        @Override
        public boolean onSingleTapUp(MotionEvent motionEvent) {
```

```
            return false;
        }

        @Override
        public boolean onScroll(MotionEvent motionEvent,
            MotionEvent motionEvent1, float v, float v1) {
            return false;
        }

        @Override
        public void onLongPress(MotionEvent motionEvent) {}

        @Override
        public boolean onFling(MotionEvent motionEvent,
            MotionEvent motionEvent1, float v, float v1) {
            return false;
        }

        @Override
        public boolean onSingleTapConfirmed(MotionEvent
            motionEvent) {
        if (listener != null) listener.onTap();
            return false;
        }

        @Override
        public boolean onDoubleTap(MotionEvent motionEvent) {
        if (listener != null) listener.onDoubleTap();
            return false;
        }

        @Override
        public boolean onDoubleTapEvent(MotionEvent motionEvent) {
            return false;
        }

        private static int calculateDirection(double degrees) {
            if ((degrees >= 0 && degrees < 22.5) || (degrees <
                0 && degrees > -22.5)) {
                    return DIRECTION_LEFT;
            } else if (degrees >= 22.5 && degrees < 67.5) {
                    return DIRECTION_LEFT_UP;
            } else if (degrees >= 67.5 && degrees < 112.5) {
                    return DIRECTION_UP;
            } else if (degrees >= 112.5 && degrees < 157.5) {
                    return DIRECTION_UP_RIGHT;
            } else if ((degrees >= 157.5 && degrees <= 180) ||
```

```
                    (degrees >= -180 && degrees < -157.5)) {
                        return DIRECTION_RIGHT;
                } else if (degrees >= -157.5 && degrees < -112.5)
                    {
                        return DIRECTION_RIGHT_DOWN;
                } else if (degrees >= -112.5 && degrees < -67.5) {
                        return DIRECTION_DOWN;
                } else if (degrees >= -67.5 && degrees < -22.5) {
                        return DIRECTION_DOWN_LEFT;
                } else {
                        return DIRECTION_CENTER;
                }
        }
}
```

A.1.13 Compass.java

```
package com.example.david.tfg;

import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.util.Log;
import android.view.animation.Animation;
import android.view.animation.RotateAnimation;
import android.widget.ImageView;

public class Compass implements SensorEventListener {
        private static final String TAG = "Compass";

        private SensorManager sensorManager;
        private Sensor gsensor;
        private Sensor msensor;
        private float[] mGravity = new float[3];
        private float[] mGeomagnetic = new float[3];
        private float azimuth = 0f;
        private float currectAzimuth = 0;

        // compass arrow to rotate
        public ImageView arrowView = null;

        public Compass(Context context) {
                sensorManager = (SensorManager) context
                .getSystemService(Context.SENSOR_SERVICE);
                gsensor = sensorManager.getDefaultSensor(Sensor.
```

```
                TYPE_ACCELEROMETER);
        msensor = sensorManager.getDefaultSensor(Sensor.
            TYPE_MAGNETIC_FIELD);
    }

    public void start() {
        sensorManager.registerListener(this, gsensor,
        SensorManager.SENSOR_DELAY_GAME);
        sensorManager.registerListener(this, msensor,
        SensorManager.SENSOR_DELAY_GAME);
    }

    public void stop() {
        sensorManager.unregisterListener(this);
    }

    private void adjustArrow() {
        if (arrowView == null) {
            Log.i(TAG, "arrow␣view␣is␣not␣set");
            return;
        }

        Log.i(TAG, "will␣set␣rotation␣from␣" +
            currectAzimuth + "␣to␣"
        + azimuth);

        Animation an = new RotateAnimation(-currectAzimuth
            , -azimuth,
        Animation.RELATIVE_TO_SELF, 0.5f, Animation.
            RELATIVE_TO_SELF,
        0.5f);
        currectAzimuth = azimuth;

        an.setDuration(400);
        an.setRepeatCount(0);
        an.setFillAfter(true);

        arrowView.startAnimation(an);
    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        final float alpha = 0.97f;

        synchronized (this) {
            if (event.sensor.getType() == Sensor.
                TYPE_ACCELEROMETER) {
```

```
                mGravity[0] = alpha * mGravity[0]
                    + (1 - alpha)
                * event.values[0];
                mGravity[1] = alpha * mGravity[1]
                    + (1 - alpha)
                * event.values[1];
                mGravity[2] = alpha * mGravity[2]
                    + (1 - alpha)
                * event.values[2];

                // mGravity = event.values;

                // Log.e(TAG, Float.toString(
                    mGravity[0]));
            }

            if (event.sensor.getType() == Sensor.
                TYPE_MAGNETIC_FIELD) {
                // mGeomagnetic = event.values;

                mGeomagnetic[0] = alpha *
                    mGeomagnetic[0] + (1 - alpha)
                * event.values[0];
                mGeomagnetic[1] = alpha *
                    mGeomagnetic[1] + (1 - alpha)
                * event.values[1];
                mGeomagnetic[2] = alpha *
                    mGeomagnetic[2] + (1 - alpha)
                * event.values[2];
                // Log.e(TAG, Float.toString(event
                    .values[0]));

            }

            float R[] = new float[9];
            float I[] = new float[9];
            boolean success = SensorManager.
                getRotationMatrix(R, I, mGravity,
            mGeomagnetic);
            if (success) {
                float orientation[] = new float
                    [3];
                SensorManager.getOrientation(R,
                    orientation);
                // Log.d(TAG, "azimuth (rad): " +
                    azimuth);
```

```
                                azimuth = (float) Math.toDegrees(
                                    orientation[0]); // orientation
                                azimuth = (azimuth + 360) % 360;
                                // Log.d(TAG, "azimuth (deg): " +
                                    azimuth);
                                adjustArrow();
                            }
                        }
                    }

                    @Override
                    public void onAccuracyChanged(Sensor sensor, int accuracy)
                        {
                    }
}
```

A.1.14 OrientationProvider.java

```
/**
*
*/
package com.example.david.tfg.orientationProvider;

import java.util.ArrayList;
import java.util.List;

import com.example.david.tfg.representation.MatrixF4x4;
import com.example.david.tfg.representation.Quaternion;

import android.hardware.Sensor;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;

/**
* Classes implementing this interface provide an orientation of
    the device
* either by directly accessing hardware, using Android sensor
    fusion or fusing
* sensors itself.
*
* The orientation can be provided as rotation matrix or quaternion
    .
*
* @author Alexander Pacha
*
*/
public abstract class OrientationProvider implements
```

```
SensorEventListener {
    /**
    * Sync-token for syncing read/write to sensor-data from
        sensor manager and
    * fusion algorithm
    */
    protected final Object synchronizationToken = new Object()
        ;

    /**
    * The list of sensors used by this provider
    */
    protected List<Sensor> sensorList = new ArrayList<Sensor
        >();

    /**
    * The matrix that holds the current rotation
    */
    protected final MatrixF4x4
        currentOrientationRotationMatrix;

    /**
    * The quaternion that holds the current rotation
    */
    protected final Quaternion currentOrientationQuaternion;

    /**
    * The sensor manager for accessing android sensors
    */
    protected SensorManager sensorManager;

    /**
    * Initialises a new OrientationProvider
    *
    * @param sensorManager
    *            The android sensor manager
    */
    public OrientationProvider(SensorManager sensorManager) {
    this.sensorManager = sensorManager;

    // Initialise with identity
    currentOrientationRotationMatrix = new MatrixF4x4();

    // Initialise with identity
    currentOrientationQuaternion = new Quaternion();
    }
```

```
        /**
        * Starts the sensor fusion (e.g. when resuming the
            activity)
        */
        public void start() {
                // enable our sensor when the activity is resumed,
                    ask for
                // 10 ms updates.
                for (Sensor sensor : sensorList) {
                        // enable our sensors when the activity is
                            resumed, ask for
                        // 20 ms updates (Sensor_delay_game)
                        sensorManager.registerListener(this,
                            sensor,
                        SensorManager.SENSOR_DELAY_GAME);
                }
        }

        /**
        * Stops the sensor fusion (e.g. when pausing/suspending
            the activity)
        */
        public void stop() {
                // make sure to turn our sensors off when the
                    activity is paused
                for (Sensor sensor : sensorList) {
                        sensorManager.unregisterListener(this,
                            sensor);
                }
        }

        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy)
            {
                // Not doing anything
        }

        /**
        * Get the current rotation of the device in the rotation
            matrix format (4x4 matrix)
        */
        public void getRotationMatrix(MatrixF4x4 matrix) {
                synchronized (synchronizationToken) {
                        matrix.set(
                            currentOrientationRotationMatrix);
                }
        }
```

```
        /**
        * Get the current rotation of the device in the quaternion
            format (vector4f)
        */
        public void getQuaternion(Quaternion quaternion) {
                synchronized (synchronizationToken) {
                        quaternion.set(
                            currentOrientationQuaternion);
                }
        }

        /**
        * Get the current rotation of the device in the Euler
           angles
        */
                public void getEulerAngles(float angles[]) {
                synchronized (synchronizationToken) {
                        SensorManager.getOrientation(
                            currentOrientationRotationMatrix.matrix
                            , angles);
                }
        }
}
```

A.1.15 ImprovedOrinetationSensor1Provider.java

```
package com.example.david.tfg.orientationProvider;

import com.example.david.tfg.Client;
import com.example.david.tfg.representation.Quaternion;

import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorManager;
import android.os.Handler;
import android.util.Log;
import com.example.david.tfg.Client;

/**
* The orientation provider that delivers the absolute orientation
   from the {@link Sensor#TYPE_GYROSCOPE
* Gyroscope} and {@link Sensor#TYPE_ROTATION_VECTOR Android
   Rotation Vector sensor}.
*
* It mainly relies on the gyroscope, but corrects with the Android
    Rotation Vector which also provides an absolute
```

```
* estimation of current orientation. The correction is a static
    weight.
*
* @author Alexander Pacha
*
*/
public class ImprovedOrientationSensor1Provider extends
    OrientationProvider {

        /**
        * Constant specifying the factor between a Nano-second and
            a second
        */
        private static final float NS2S = 1.0f / 1000000000.0f;

        /**
        * The quaternion that stores the difference that is
            obtained by the gyroscope.
        * Basically it contains a rotational difference encoded
            into a quaternion.
        *
        * To obtain the absolute orientation one must add this
            into an initial position by
        * multiplying it with another quaternion
        */
        private final Quaternion deltaQuaternion = new Quaternion
            ();

        /**
        * The Quaternions that contain the current rotation (Angle
            and axis in Quaternion format) of the Gyroscope
        */
        private Quaternion quaternionGyroscope = new Quaternion();

        /**
        * The quaternion that contains the absolute orientation as
            obtained by the rotationVector sensor.
        */
        private Quaternion quaternionRotationVector = new
            Quaternion();

        /**
        * The time-stamp being used to record the time when the
            last gyroscope event occurred.
        */
        private long timestamp;
```

```
/**
* This is a filter-threshold for discarding Gyroscope
    measurements that are below a certain level and
* potentially are only noise and not real motion. Values
    from the gyroscope are usually between 0 (stop) and
* 10 (rapid rotation), so 0.1 seems to be a reasonable
    threshold to filter noise (usually smaller than 0.1)
    and
* real motion (usually > 0.1). Note that there is a chance
     of missing real motion, if the use is turning the
* device really slowly, so this value has to find a
    balance between accepting noise (threshold = 0) and
    missing
* slow user-action (threshold > 0.5). 0.1 seems to work
    fine for most applications.
*
*/
private static final double EPSILON = 0.1f;

/**
* Value giving the total velocity of the gyroscope (will
    be high, when the device is moving fast and low when
* the device is standing still). This is usually a value
    between 0 and 10 for normal motion. Heavy shaking can
* increase it to about 25. Keep in mind, that these values
     are time-depended, so changing the sampling rate of
* the sensor will affect this value!
*/
private double gyroscopeRotationVelocity = 0;

/**
* Flag indicating, whether the orientations were
    initialised from the rotation vector or not. If false,
    the
* gyroscope can not be used (since it's only meaningful to
     calculate differences from an initial state). If
* true,
* the gyroscope can be used normally.
*/
private boolean positionInitialised = false;

/**
* Counter that sums the number of consecutive frames,
    where the rotationVector and the gyroscope were
* significantly different (and the dot-product was smaller
     than 0.7). This event can either happen when the
* angles of the rotation vector explode (e.g. during fast
```

```
        tilting) or when the device was shaken heavily and
    * the gyroscope is now completely off.
    */
    private int panicCounter;

    /**
    * This weight determines directly how much the rotation
        sensor will be used to correct (in
    * Sensor-fusion-scenario 1 - SensorSelection.
        GyroscopeAndRotationVector). Must be a value between 0
        and 1.
    * 0 means that the system entirely relies on the gyroscope
        , whereas 1 means that the system relies entirely on
    * the rotationVector.
    */
    private static final float DIRECT_INTERPOLATION_WEIGHT =
        0.005f;

    /**
    * The threshold that indicates an outlier of the rotation
        vector. If the dot-product between the two vectors
    * (gyroscope orientation and rotationVector orientation)
        falls below this threshold (ideally it should be 1,
    * if they are exactly the same) the system falls back to
        the gyroscope values only and just ignores the
    * rotation vector.
    *
    * This value should be quite high (> 0.7) to filter even
        the slightest discrepancies that causes jumps when
    * tiling the device. Possible values are between 0 and 1,
        where a value close to 1 means that even a very small
    * difference between the two sensors will be treated as
        outlier, whereas a value close to zero means that the
    * almost any discrepancy between the two sensors is
        tolerated.
    */
    private static final float OUTLIER_THRESHOLD = 0.85f;

    /**
    * The threshold that indicates a massive discrepancy
        between the rotation vector and the gyroscope
        orientation.
    * If the dot-product between the two vectors
    * (gyroscope orientation and rotationVector orientation)
        falls below this threshold (ideally it should be 1, if
    * they are exactly the same), the system will start
        increasing the panic counter (that probably indicates a
```

```
     * gyroscope failure).
     *
     * This value should be lower than OUTLIER_THRESHOLD (0.5 -
         0.7) to only start increasing the panic counter,
     * when there is a
     * huge discrepancy between the two fused sensors.
     */
    private static final float OUTLIER_PANIC_THRESHOLD = 0.65f
        ;

    /**
     * The threshold that indicates that a chaos state has been
         established rather than just a temporary peak in the
     * rotation vector (caused by exploding angled during fast
        tilting).
     *
     * If the chaosCounter is bigger than this threshold, the
        current position will be reset to whatever the
     * rotation vector indicates.
     */
    private static final int PANIC_THRESHOLD = 60;

    /**
     * Some temporary variables to save allocations
     */
    final private float[] temporaryQuaternion = new float[4];
    final private Quaternion correctedQuaternion = new
        Quaternion();
    final private Quaternion interpolatedQuaternion = new
        Quaternion();

    /**
     * Initialises a new ImprovedOrientationSensor1Provider
     *
     * @param sensorManager The android sensor manager
     */
    public ImprovedOrientationSensor1Provider(SensorManager
        sensorManager) {
            super(sensorManager);

            //Add the gyroscope and rotation Vector
            sensorList.add(sensorManager.getDefaultSensor(
                Sensor.TYPE_GYROSCOPE));
            sensorList.add(sensorManager.getDefaultSensor(
                Sensor.TYPE_ROTATION_VECTOR));
    }
```

```
@Override
public void onSensorChanged(SensorEvent event) {

        if (event.sensor.getType() == Sensor.TYPE_ROTATION_VECTOR)
            {
                // Process rotation vector (just safe it)
                // Calculate angle. Starting with API_18, Android
                    will provide this value as event.values[3], but
                     if not, we have to calculate it manually.
                SensorManager.getQuaternionFromVector(
                    temporaryQuaternion, event.values);

                // Store in quaternion
                quaternionRotationVector.setXYZW(
                    temporaryQuaternion[1], temporaryQuaternion[2],
                     temporaryQuaternion[3], -temporaryQuaternion
                    [0]);
                if (!positionInitialised) {
                        // Override
                        quaternionGyroscope.set(
                            quaternionRotationVector);
                        positionInitialised = true;
                }

        } else if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE
            ) {
                // Process Gyroscope and perform fusion

                // This timestep's delta rotation to be multiplied
                     by the current rotation
                // after computing it from the gyro sample data.
                if (timestamp != 0) {
                        final float dT = (event.timestamp -
                            timestamp) * NS2S;
                        // Axis of the rotation sample, not
                            normalized yet.
                        float axisX = event.values[0];
                        float axisY = event.values[1];
                        float axisZ = event.values[2];

                        // Calculate the angular speed of the
                            sample
                        gyroscopeRotationVelocity = Math.sqrt(
                            axisX * axisX + axisY * axisY + axisZ *
                             axisZ);

                        // Normalize the rotation vector if it's
```

```
                    big enough to get the axis
                if (gyroscopeRotationVelocity > EPSILON) {
                        axisX /= gyroscopeRotationVelocity
                            ;
                        axisY /= gyroscopeRotationVelocity
                            ;
                        axisZ /= gyroscopeRotationVelocity
                            ;
                }

                // Integrate around this axis with the
                    angular speed by the timestep
                // in order to get a delta rotation from
                    this sample over the timestep
                // We will convert this axis-angle
                    representation of the delta rotation
                // into a quaternion before turning it
                    into the rotation matrix.
                double thetaOverTwo =
                    gyroscopeRotationVelocity * dT / 2.0f;
                double sinThetaOverTwo = Math.sin(
                    thetaOverTwo);
                double cosThetaOverTwo = Math.cos(
                    thetaOverTwo);
                deltaQuaternion.setX((float) (
                    sinThetaOverTwo * axisX));
                deltaQuaternion.setY((float) (
                    sinThetaOverTwo * axisY));
                deltaQuaternion.setZ((float) (
                    sinThetaOverTwo * axisZ));
                deltaQuaternion.setW(-(float)
                    cosThetaOverTwo);

                // Move current gyro orientation
                deltaQuaternion.multiplyByQuat(
                    quaternionGyroscope,
                    quaternionGyroscope);

                // Calculate dot-product to calculate
                    whether the two orientation sensors
                    have diverged
                // (if the dot-product is closer to 0 than
                     to 1), because it should be close to 1
                     if both are the same.
                float dotProd = quaternionGyroscope.
                    dotProduct(quaternionRotationVector);
```

```
                    // If they have diverged, rely on
                       gyroscope only (this happens on some
                       devices when the rotation vector "jumps
                       ").
                    if (Math.abs(dotProd) < OUTLIER_THRESHOLD)
                         {
                            // Increase panic counter
                            if (Math.abs(dotProd) <
                               OUTLIER_PANIC_THRESHOLD) {
                                  panicCounter++;
                            }

                            // Directly use Gyro
                            setOrientationQuaternionAndMatrix(
                               quaternionGyroscope);

                    } else {
                            // Both are nearly saying the same
                               . Perform normal fusion.

                            // Interpolate with a fixed weight
                                between the two absolute
                               quaternions obtained from gyro
                               and rotation vector sensors
                            // The weight should be quite low,
                                so the rotation vector
                               corrects the gyro only slowly,
                               and the output keeps responsive
                               .
                            quaternionGyroscope.slerp(
                               quaternionRotationVector,
                               interpolatedQuaternion,
                               DIRECT_INTERPOLATION_WEIGHT);

                            // Use the interpolated value
                               between gyro and rotationVector
                            setOrientationQuaternionAndMatrix(
                               interpolatedQuaternion);
                            // Override current gyroscope-
                               orientation
                            quaternionGyroscope.copyVec4(
                               interpolatedQuaternion);

                            // Reset the panic counter because
                                both sensors are saying the
                               same again
                            panicCounter = 0;
```

```
                        }

                        if (panicCounter > PANIC_THRESHOLD) {
                                Log.d("Rotation Vector",
                                "Panic counter is bigger than
                                    threshold; this indicates a
                                    Gyroscope failure. Panic reset
                                    is imminent.");

                                if (gyroscopeRotationVelocity < 3)
                                    {
                                        Log.d("Rotation Vector",
                                        "Performing Panic-reset.
                                            Resetting orientation
                                            to rotation-vector
                                            value.");

                                        // Manually set position
                                            to whatever rotation
                                            vector says.
                                        setOrientationQuaternionAndMatrix
                                            (
                                            quaternionRotationVector
                                            );
                                        // Override current
                                            gyroscope-orientation
                                            with corrected value
                                        quaternionGyroscope.
                                            copyVec4(
                                            quaternionRotationVector
                                            );

                                        panicCounter = 0;
                                } else {
                                        Log.d("Rotation Vector",
                                        String.format(
                                        "Panic reset delayed due
                                            to ongoing motion (user
                                            is still shaking the
                                            device). Gyroscope
                                            Velocity: %.2f > 3",
                                        gyroscopeRotationVelocity)
                                            );
                                }
                        }
                }
        timestamp = event.timestamp;
```

```
        }
}


        /**
        * Sets the output quaternion and matrix with the provided
            quaternion and synchronises the setting
        *
        * @param quaternion The Quaternion to set (the result of
            the sensor fusion)
        */
        private void setOrientationQuaternionAndMatrix(Quaternion
            quaternion) {
                correctedQuaternion.set(quaternion);
                // We inverted w in the deltaQuaternion, because
                    currentOrientationQuaternion required it.
                // Before converting it back to matrix
                    representation, we need to revert this process
                correctedQuaternion.w(-correctedQuaternion.w());

                synchronized (synchronizationToken) {
                        // Use gyro only
                        currentOrientationQuaternion.copyVec4(
                            quaternion);

                        // Set the rotation matrix as well to have
                             both representations
                        SensorManager.getRotationMatrixFromVector(
                            currentOrientationRotationMatrix.matrix
                            , correctedQuaternion.array());
                }
        }
}
```

A.1.16 layout_home.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        tools:context=".HomeActivity">

        <RelativeLayout
                android:layout_width="match_parent"
                android:layout_height="25dp"
```

```
            android:layout_marginTop="5dp" >

            <RelativeLayout
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_centerHorizontal="true" >

                    <TextView
                            android:id="@+id/label_filter_name
                                "
                            android:layout_width="wrap_content
                                "
                            android:layout_height="
                                wrap_content"
                            android:text="Drone␣"
                            android:textAppearance="?
                                android:attr/
                                textAppearanceLarge"
                            android:textColor="#ff3d00"
                            android:textStyle="bold" />

                    <TextView
                            android:id="@+id/
                                label_filter_description"
                            android:layout_width="wrap_content
                                "
                            android:layout_height="
                                wrap_content"
                            android:layout_toRightOf="@+id/
                                label_filter_name"
                            android:text="Control"
                            android:textAppearance="?
                                android:attr/
                                textAppearanceLarge"
                            android:textColor="#fff"
                            android:textStyle="bold" />
            </RelativeLayout>
    </RelativeLayout>

    <RelativeLayout
            android:id="@+id/layout_statistics"
            android:layout_width="match_parent"
            android:layout_height="30dp"
            android:gravity="center">
                    <TextView
                            android:id="@+id/label_units"
                            android:layout_width="wrap_content
```

```
                                    "
                                android:layout_height="
                                    wrap_content"
                                android:text="David␣Arévalo␣
                                    Jiménez␣TFG"
                                android:textColor="#00ffff"
                                android:textSize="10dp" />
        </RelativeLayout>

        <ScrollView
                android:id="@+id/ScrollView01"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent">

                <FrameLayout
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content"
                        android:layout_marginTop="15dp"
                        android:orientation="vertical"
                        android:paddingBottom="25dp">

                        <LinearLayout
                                android:id="@+id/menu_bar"
                                android:layout_width="fill_parent"
                                android:layout_height="
                                    wrap_content"
                                android:gravity="center_horizontal
                                    "
                                android:orientation="vertical">

                                <LinearLayout
                                        android:layout_width="
                                            fill_parent"
                                        android:layout_height="
                                            wrap_content"
                                        android:gravity="center"
                                        android:orientation="
                                            horizontal"
                                        android:weightSum="1">

                                        <Button
                                        android:id="@+id/
                                            button_normal_mode"
                                        android:layout_width="0dp"
                                        android:layout_height="90
                                            dp"
                                        android:layout_weight="
```

```
                                    0.75"
                                android:background="
                                    @drawable/button_normal
                                    "
                                android:gravity="right|
                                    center_vertical"
                                android:paddingRight="15dp
                                    "
                                android:scaleType="
                                    centerInside"
                                android:text="@string/
                                    normal"
                                android:textSize="18dp"
                                android:textStyle="bold" /
                                    >
                        </LinearLayout>

                        <LinearLayout
                                android:layout_width="
                                    fill_parent"
                                android:layout_height="
                                    wrap_content"
                                android:layout_marginTop="
                                    15dp"
                                android:gravity="center"
                                android:orientation="
                                    horizontal"
                                android:weightSum="1">

                                <Button
                                android:id="@+id/
                                    button_touching_mode"
                                android:layout_width="0dp"
                                android:layout_height="90
                                    dp"
                                android:layout_weight="
                                    0.75"
                                android:background="
                                    @drawable/button_touch"
                                android:gravity="right|
                                    center_vertical"
                                android:paddingRight="15dp
                                    "
                                android:scaleType="
                                    centerInside"
                                android:text="@string/
                                    touched"
```

```
                android:textSize="18dp"
                android:textStyle="bold" />
        </LinearLayout>

        <LinearLayout
                android:layout_width="
                    fill_parent"
                android:layout_height="90
                    dp"
                android:layout_marginTop="
                    15dp"
                android:gravity="center"
                android:orientation="
                    horizontal"
                android:weightSum="1">

                <Button
                android:id="@+id/
                    button_point_mode"
                android:layout_width="0dp"
                android:layout_height="90
                    dp"
                android:layout_weight="
                    0.75"
                android:background="
                    @drawable/button_point"
                android:gravity="right|
                    center_vertical"
                android:paddingRight="15dp
                    "
                android:scaleType="
                    centerInside"
                android:text="@string/test
                    "
                android:textSize="18dp"
                android:textStyle="bold" />
        </LinearLayout>

        <LinearLayout
                android:layout_width="
                    fill_parent"
                android:layout_height="
                    wrap_content"
                android:layout_marginTop="
                    15dp"
```

```
                                                            android:gravity="center"
                                                            android:orientation="
                                                                horizontal"
                                                            android:weightSum="1">

                                                            <Button
                                                            android:id="@+id/
                                                                button_drone_mode"
                                                            android:layout_width="0dp"
                                                            android:layout_height="90
                                                                dp"
                                                            android:layout_weight="
                                                                0.75"
                                                            android:background="
                                                                @drawable/button_drone"
                                                            android:gravity="right|
                                                                center_vertical"
                                                            android:paddingRight="15dp
                                                                "
                                                            android:scaleType="
                                                                centerInside"
                                                            android:text="@string/
                                                                drone"          "
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣android:textSize="18dp"
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣android:textStyle="bold"␣
    />
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</LinearLayout>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</LinearLayout>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</FrameLayout>
␣␣␣␣␣␣␣␣</ScrollView>
␣␣␣␣␣␣␣␣<View
␣␣␣␣␣␣␣␣android:layout_width="fill_parent"
␣␣␣␣␣␣␣␣android:layout_height="1dp"
␣␣␣␣␣␣␣␣android:background="@android:color/darker_gray"␣/>
</LinearLayout>
```

A.1.17 layout_help_selection.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <ScrollView
                android:id="@+id/ScrollView01"
```

```
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:fillViewport="true" >

                <LinearLayout
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:orientation="vertical" >

                        <RelativeLayout
                                android:layout_width="match_parent
                                    "
                                android:layout_height="
                                    wrap_content"
                                android:layout_marginTop="20dp" >

                        <RelativeLayout
                                android:layout_width="wrap_content
                                    "
                                android:layout_height="
                                    wrap_content"
                                android:layout_centerHorizontal="
                                    true" >

                                <TextView
                                        android:id="@+id/
                                            label_sensor_name"
                                        android:layout_width="
                                            wrap_content"
                                        android:layout_height="
                                            wrap_content"
                                        android:text="@string/
                                            app_name"
                                        android:textAppearance="?
                                            android:attr/
                                            textAppearanceMedium"
                                        android:textColor="#ff3d00
                                            "
                                        android:textStyle="bold" /
                                            >

                                <TextView
                                        android:id="@+id/
                                            label_app_description"
                                        android:layout_width="
                                            wrap_content"
                                        android:layout_height="
```

```
                                                wrap_content"
                                            android:layout_toRightOf="
                                                @+id/label_sensor_name"
                                            android:text="help"
                                            android:textAppearance="?
                                                android:attr/
                                                textAppearanceMedium"
                                            android:textColor="#fff"
                                            android:textStyle="bold" /
                                                >
                            </RelativeLayout>
                            </RelativeLayout>
                    </LinearLayout>
            </ScrollView>
</LinearLayout>
```

A.1.18 activity_sensor_selection.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:id="@+id/pagers"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.v4.view.ViewPager
                android:id="@+id/pager"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                tools:context=".SensorSelectionActivity" >

                <!--
                This title strip will display the currently
                    visible page title, as well as the page
                titles for adjacent pages.
                -->

                <android.support.v4.view.PagerTitleStrip
                        android:id="@+id/pager_title_strip"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:layout_gravity="top"
                        android:background="#33b5e5"
                        android:paddingBottom="4dp"
                        android:paddingTop="4dp"
                        android:textColor="#fff" />
```

```
        </android.support.v4.view.ViewPager>

        <com.erz.joysticklibrary.JoyStick
                android:id="@+id/joy1"
                android:layout_width="200dp"
                android:layout_height="200dp"
                android:layout_gravity="bottom"/>

        <com.crystal.crystalrangeseekbar.widgets.CrystalSeekbar
                android:id="@+id/UpDown"
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                app:min_value="0"
                app:max_value="100"
                app:steps="1"
                app:bar_color="#ff5722"
                app:bar_highlight_color="#E07416"
                app:left_thumb_image="@drawable/thumb"
                app:left_thumb_image_pressed="@drawable/
                    thumb_pressed"
                app:data_type="_integer"
                android:layout_marginTop="35dp"/>
        <RelativeLayout
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:gravity="right|bottom"
                android:layout_gravity="right|bottom"
                android:layout_marginRight="15dp"
                android:layout_marginBottom="15dp"
                android:id="@+id/rotation2">
                <ImageView
                        android:id="@+id/main_image_dial"
                        android:layout_width="80dp"
                        android:layout_height="80dp"
                        android:layout_centerInParent="true"
                        android:contentDescription="@string/
                            compass_dial"
                        android:src="@drawable/dial3" />

                <ImageView
                        android:id="@+id/main_image_hands"
                        android:layout_width="53dp"
                        android:layout_height="53dp"
                        android:layout_centerInParent="true"
                        android:contentDescription="@string/
                            compass_hands"
                        android:scaleType="fitXY"
```

```
                android:src="@drawable/hands2" />
        </RelativeLayout>
        <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:gravity="center"
                android:layout_gravity="center"
                android:background="#64FFDA"
                android:orientation="vertical"
                android:id="@+id/rotation">

                <ImageView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:src="@drawable/rotatedevice"
                        android:layout_gravity="center"/>
                <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text="@string/rotate"
                        android:textColor="#004D40"
                        android:textSize="20dp"
                        android:layout_marginTop="15dp"/>
        </LinearLayout>
</FrameLayout>
```

A.1.19 settings.xml

```
<?xml version="1.0" encoding="utf-8"?>
        <PreferenceScreen xmlns:android="http://schemas.android.
            com/apk/res/android">

        <PreferenceCategory android:title="@string/options" >
                <ListPreference      android:title="@string/sensors
                    "
                        android:summary="@string/selection"
                        android:key="SensorFusion"
                        android:defaultValue="1"
                        android:entries="@array/listArray"
                        android:entryValues="@array/listValues" />
                <CheckBoxPreference android:title="@string/compass
                    "
                        android:defaultValue="true"
                        android:summary="@string/disable"
                        android:key="compass" />-->
                <ListPreference      android:title="@string/
                    language"
```

```
                android:summary="@string/list"
                android:key="language"
                android:defaultValue="1"
                android:entries="@array/listArrayLanguages
                    "
                android:entryValues="@array/
                    listValuesLanguages" />
    </PreferenceCategory>

    <PreferenceCategory android:title="@string/server" >
        <EditTextPreference
                android:defaultValue="@string/
                    default_value_ip"
                android:dialogMessage="@string/ip"
                android:dialogTitle="@string/number"
                android:inputType="textNoSuggestions"
                android:key="ip"
                android:title="@string/setip" />
        <EditTextPreference
                android:defaultValue="@string/
                    default_value_port"
                android:dialogMessage="@string/computer"
                android:dialogTitle="@string/port"
                android:inputType="number"
                android:key="port"
                android:title="@string/setport" />
    </PreferenceCategory>
    <PreferenceCategory android:title="@string/constants" >
        <EditTextPreference
                android:defaultValue="@string/
                    default_value_sense"
                android:dialogMessage="@string/dm1"
                android:dialogTitle="@string/dt1"
                android:inputType="numberDecimal"
                android:key="sense"
                android:title="@string/t1" />
        <EditTextPreference
                android:defaultValue="@string/
                    default_value_maxvel"
                android:dialogMessage="@string/dm2"
                android:dialogTitle="@string/dt2"
                android:inputType="number"
                android:key="maxVel"
                android:title="@string/t2" />
        <EditTextPreference
                android:defaultValue="@string/
                    default_value_maxangle"
```

```
                android:dialogMessage="@string/dm3"
                android:dialogTitle="@string/dt3"
                android:inputType="number"
                android:key="maxAngle"
                android:title="@string/t3" />
        <EditTextPreference
                android:defaultValue="@string/
                    default_value_maxheight"
                android:dialogMessage="@string/dm4"
                android:dialogTitle="@string/dt4"
                android:inputType="number"
                android:key="maxAltura"
                android:title="@string/t4" />
        <EditTextPreference
                android:defaultValue="@string/
                    default_value_maxarea"
                android:dialogMessage="@string/dm5"
                android:dialogTitle="@string/dt5"
                android:inputType="number"
                android:key="maxArea"
                android:title="@string/t5" />
    </PreferenceCategory>
</PreferenceScreen>
```

A.2 Implementación en TurtleSim

A.2.1 robot_controller.cpp

```
/*
* Author: David Arévalo Jiménez
* Year: 2017
* Treball de fi de grau EEIA
* University: Universitat de les Illes Balears (UIB)
*
*/



#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "turtlesim/Pose.h"
#include <sstream>

#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h> //inet_addr
#include <unistd.h>    //write
```

```
#include <pthread.h> //for threading , link with lpthread

using namespace std;


//Global variables
ros::Publisher velocity_publisher; //create a publisher
ros::Subscriber pose_subscriber; //suscribe to Pose
turtlesim::Pose turtlesim_pose;

const double PI = 3.14159265359;
double relative_angle; double linear_speed;


//metodos
void rotateABS (double angular_speed, double angle, double
    linear_speed);
double degrees2radians(double angle_in_degrees);
void poseCallback(const turtlesim::Pose::ConstPtr & pose_message);
int Server();
void *connection_handler(void *);


int main(int argc, char **argv)
{
        // Initiate new ROS node named "talker"
        ros::init(argc, argv, "turtlesim_controller");
        ros::NodeHandle n;

        //subscribers and publishers
        velocity_publisher = n.advertise<geometry_msgs::Twist>("/
            turtle1/cmd_vel", 100);
        pose_subscriber = n.subscribe("/turtle1/pose", 10,
            poseCallback);

        double angular_speed, angle;

        //**********SERVER*************
        int socket_desc , client_sock , c , *new_sock;
        struct sockaddr_in server , client;
        char client_message[2000];
        char buf[100];

        Server();

        return 0;
```

```
}

void rotateABS (double angular_speed){

        geometry_msgs::Twist vel_msg;

        //set a 0 linear velocity in the x-y-axis
        vel_msg.linear.x =linear_speed;
        vel_msg.linear.y =0;
        vel_msg.linear.z =0;
        //set a 0 angular velocity in the x-y-axis
        vel_msg.angular.x = 0;
        vel_msg.angular.y = 0;

        ros::spinOnce(); //actualizar valores

        int clockwise=0;
        double positive_theta=turtlesim_pose.theta;

        //Theta values 0 to 6.28
        if (turtlesim_pose.theta<0)
        positive_theta=degrees2radians(360)+turtlesim_pose.theta;

        double relative_angle360=relative_angle+2*PI;
        double theta360=positive_theta+2*PI;

        //Calculate the type of rotation
        if (relative_angle>positive_theta) //camino corto
        if((theta360-relative_angle)<(relative_angle-
            positive_theta))
        vel_msg.angular.z =-abs(angular_speed), clockwise=3; //de
            angulo positivo a negativo horario
        else
        vel_msg.angular.z =abs(angular_speed), clockwise=0;//
            antihorario de 0 2pi no de 2pi a 0
        else if ((relative_angle360-positive_theta)<(
            positive_theta-relative_angle))
        vel_msg.angular.z =abs(angular_speed), clockwise=1;//de
            negativo a positivo antihorario
        else
        vel_msg.angular.z =-abs(angular_speed), clockwise=2; //
            horario de 2pi a 0

        double end_angle = relative_angle-positive_theta; //
            diference between goal and current angle
        bool big2small_angle = 0;
```

```
bool firstTime=0;

if (relative_angle-positive_theta>0)
big2small_angle=0;
else
big2small_angle=1;

ros::Rate loop_rate(100);

do{
        velocity_publisher.publish(vel_msg);

        //theta 0 to 6.28
        if (turtlesim_pose.theta<0){
                positive_theta=degrees2radians(360)+
                    turtlesim_pose.theta;
        }else{
                positive_theta=turtlesim_pose.theta;

        //Calculate correct end_angle
        }if (big2small_angle && clockwise==2){ //horario
                end_angle = -(relative_angle-
                    positive_theta);
        }else if (clockwise==1 && big2small_angle &&
            relative_angle<positive_theta && !firstTime){
            //neg a pos       antihorario
                end_angle = -(relative_angle-
                    positive_theta);
        }else{
                if(clockwise==3){
                        if(positive_theta<degrees2radians
                            (180)){
                                end_angle = (
                                    relative_angle-
                                    positive_theta); //paso
                                     de primero al cuarto
                                    cuadrante
                        }else{
                                end_angle = positive_theta
                                    -relative_angle;
                        }
                }else{
                        end_angle = relative_angle-
                            positive_theta;//antihorario
                        firstTime=1;
                }
        }
```

```
                //reload Pose
                ros::spinOnce();
                loop_rate.sleep();
        }while(end_angle>0);
        //set angular velocity to 0
        vel_msg.angular.z =0;
        velocity_publisher.publish(vel_msg);
}

double degrees2radians(double angle_in_degrees){
        return angle_in_degrees *PI /180.0;
}

void poseCallback(const turtlesim::Pose::ConstPtr & pose_message){
        turtlesim_pose.x=pose_message->x;
        turtlesim_pose.y=pose_message->y;
        turtlesim_pose.theta=pose_message->theta;
}


/*
* This will handle connection for each client
* */
void *connection_handler(void *socket_desc){
        //Get the socket descriptor
        int sock = *(int*)socket_desc;
        int read_size;
        char *message , client_message[2000],angulo_message[20],
            longitud_message[20];
        int numbytes;
        int i=0,e=0;
        float angulo=0,longitud=0;
        memset(client_message, 0, strlen(client_message));

        while( (read_size = recv(sock , client_message , 2000 , 0)
            ) > 0 ){
                //Read angle
                while(client_message[i]!='␣'){
                        angulo_message[i]=client_message[i];
                        i++;
                }
                //string to float
                angulo=angulo_message[0] - '0';
                angulo=angulo+0.1*(angulo_message[2] - '0');
                angulo=angulo+0.01*(angulo_message[3] - '0');
                angulo=angulo+0.001*(angulo_message[4] - '0');
                angulo=angulo+0.0001*(angulo_message[5] - '0');
```

```
            angulo=angulo+0.00001*(angulo_message[6] - '0');
            angulo=angulo+0.000001*(angulo_message[7] - '0');


            i++;
            e=0;
            //Read longitud
            while(client_message[i]!=' '){
            longitud_message[e]=client_message[i];
            i++;
            e++;
        }
        i=0;
        e=0;

        //String to float
        longitud=longitud_message[0] - '0';
        longitud=longitud+0.1*(longitud_message[2] - '0');
        longitud=longitud+0.01*(longitud_message[3] - '0');
        longitud=longitud+0.001*(longitud_message[4] - '0');
        longitud=longitud+0.0001*(longitud_message[5] - '0');
        longitud=longitud+0.00001*(longitud_message[6] - '0');
        longitud=longitud+0.000001*(longitud_message[7] - '0');

        //printf("%s\n",angulo_message);
        //printf("%f\n",angulo);
        //printf("%f\n",longitud);

        //clean the buffers
        memset(client_message, 0, strlen(client_message));
        memset(angulo_message, 0, strlen(angulo_message));
        memset(longitud_message, 0, strlen(longitud_message));
        relative_angle=(double)angulo;
        linear_speed=(double)longitud*5/2;

        rotateABS (degrees2radians(180));
        }

        if(read_size == 0){
        puts("Client disconnected");
        fflush(stdout);
        }
        else if(read_size == -1){
        perror("recv failed");
        }
        //Free the socket pointer
        free(socket_desc);
```

```
        close(sock);
        pthread_exit(NULL);
        return 0;
}

int Server(){
        //Create socket
        socket_desc = socket(AF_INET , SOCK_STREAM , 0);
        if (socket_desc == -1){
        printf("Could not create socket");
        }
        puts("Socket created");

        //Prepare the sockaddr_in structure
        server.sin_family = AF_INET;
        server.sin_addr.s_addr = inet_addr("192.168.1.46");
        server.sin_port = htons( 8080 );
        int count = 0;

        //Bind
        if( bind(socket_desc,(struct sockaddr *)&server , sizeof(
            server)) < 0){
                //print the error message
                perror("bind failed. Error");
                return 1;
        }
        puts("bind done");

        printf("IP address is: %s\n", inet_ntoa(server.sin_addr));
        printf("port is: %d\n", (int) ntohs(server.sin_port));

        //Listen
        listen(socket_desc , 3);
        puts("Waiting for incoming connections...");
        c = sizeof(struct sockaddr_in);

        //Accept and incoming connection
        while( (client_sock = accept(socket_desc, (struct sockaddr
             *)&client, (socklen_t*)&c)) ){
                puts("Connection accepted");
                count++;
                printf("Thread #%i \n",count);
                printf("Client IP address: %s\n", inet_ntoa(client
                    .sin_addr));
                printf("port is: %d\n", (int) ntohs(client.
                    sin_port));
```

```
                pthread_t sniffer_thread;
                new_sock = (int *)malloc(1);
                *new_sock = client_sock;

                if( pthread_create( &sniffer_thread , NULL ,
                    connection_handler , (void*) new_sock) < 0){
                perror("could not create thread");
                return 1;
                }
                //Now join the thread , so that we dont terminate
                    before the thread
                //pthread_join( sniffer_thread , NULL); // was
                    commented before
                puts("Handler assigned");
        }

        if (client_sock < 0){
                perror("accept failed");
                return 1;
        }
        return 0;
}
```

A.3 Implementación en MORSE simulator

A.3.1 car_controller.cpp

```
/*
* Author: David Arévalo Jiménez
* Year: 2017
* Treball de fi de grau EEIA
* University: Universitat de les Illes Balears (UIB)
*
*/


#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "geometry_msgs/PoseStamped.h"
#include <sstream>
#include <stdio.h>

#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h> //inet_addr
#include <unistd.h>    //write
#include <pthread.h> //for threading , link with lpthread
```

```
#include <cmath>

using namespace std;


//Global variables
ros::Publisher velocity_publisher; //create a publisher
ros::Subscriber pose_subscriber; //subscribe to Pose
geometry_msgs::PoseStamped car_pose;
geometry_msgs::Twist vel_msg;
geometry_msgs::Twist vel_msg_stop;

double angulos;
double velocidades;

const double PI = 3.14159265359;

int mode=0,velMax=50,angularMax=50;


//metodos
void rotateREL (double relative_angle, double linear_speed,double
    maxRotVel);
void rotateABS (double angular_speed);
double degrees2radians(double angle_in_degrees);
void poseCallback(const geometry_msgs::PoseStamped::ConstPtr &
    pose_message);
int Server();

void *connection_handler(void *);


int main(int argc, char **argv)
{
        // Initiate new ROS node named "talker"
        ros::init(argc, argv, "car_controller");
        ros::NodeHandle n;
        //subscribers and publishers
        velocity_publisher = n.advertise<geometry_msgs::Twist>("/
            atrv/motion", 100);
        pose_subscriber = n.subscribe("/atrv/pose", 10,
            poseCallback);
        double speed, angular_speed,angle;

        Server();

        return 0;
```

```
}

void rotateREL (double relative_angle, double linear_speed,double
    maxRotVel){
        double angular_speed;
        //set a 0 linear velocity in the y-z axis
        vel_msg.linear.x =linear_speed;
        vel_msg.linear.y =0;
        vel_msg.linear.z =0;
        //set a 0 angular velocity in the x-y-z axis
        vel_msg.angular.x = 0;
        vel_msg.angular.y = 0;
        vel_msg.angular.z = 0;
        ros::spinOnce(); //iniciarlizar valores

        //Cálculo de la w correspondiente al punto
        if(relative_angle<=degrees2radians(90) || (relative_angle
            <=degrees2radians(270) && relative_angle>=
            degrees2radians(180))){
                if(relative_angle<=degrees2radians(90)){
                        angular_speed=maxRotVel-(relative_angle*
                            maxRotVel/degrees2radians(90));
                }else{
                        angular_speed=maxRotVel-((relative_angle-
                            degrees2radians(180))*maxRotVel/
                            degrees2radians(90));
                }
        }
        if(relative_angle>=degrees2radians(270) || (relative_angle
            <=degrees2radians(180) && relative_angle>=
            degrees2radians(90))){
                if(relative_angle>=degrees2radians(270)){
                        angular_speed=(relative_angle-
                            degrees2radians(270))*maxRotVel/
                            degrees2radians(90);
                }else{
                        angular_speed=(relative_angle+
                            degrees2radians(-270+180))*maxRotVel/
                            degrees2radians(90);
                }
        }
        //asignar sentido de giro
                if(relative_angle>=degrees2radians(90) &&
                    relative_angle<=degrees2radians(270))
                vel_msg.angular.z =abs(angular_speed);
        else
```

```
                vel_msg.angular.z =-abs(angular_speed);

        if(linear_speed==0){
                vel_msg.angular.z = 0;
        }
        //linear speed
        if(relative_angle>degrees2radians(180)){
                vel_msg.linear.x = -linear_speed;
        }
        //publish
        velocity_publisher.publish(vel_msg);
}

void rotateABS (double angular_speed){

        //set a 0 linear velocity in the x-y-axis
        vel_msg.linear.x =velocidades;
        vel_msg.linear.y =0;
        vel_msg.linear.z =0;
        //set a 0 angular velocity in the x-y-axis
        vel_msg.angular.x = 0;
        vel_msg.angular.y = 0;
        vel_msg.angular.z = 0; //por si acaso
        ros::spinOnce(); //si no da 0 w

        int clockwise=0;
        double positive_w=car_pose.pose.orientation.w;

        if (car_pose.pose.orientation.w<0)
                positive_w=degrees2radians(360)+car_pose.pose.
                    orientation.w;

        double angulos360=angulos+2*PI;
        double w360=positive_w+2*PI;

        if (angulos>positive_w) //falta camino corto
                if((w360-angulos)<(angulos-positive_w))
                        vel_msg.angular.z =-abs(angular_speed),
                            clockwise=3;//de angulo positivo a
                            negativo horario
                else
                        vel_msg.angular.z =abs(angular_speed),
                            clockwise=0;//antihorario de 0 2pi no
                            de 2pi a 0
                else if ((angulos360-positive_w)<(positive_w-
                    angulos))
                        vel_msg.angular.z =abs(angular_speed),
```

```
                        clockwise=1;//de negativo a positivo
                            antihorario
                else
                        vel_msg.angular.z =-abs(angular_speed),
                            clockwise=2;//horario de 2pi a 0

        double end_angle = angulos-positive_w;//diference between
            goal and current angle
        bool big2small_angle = 0;
        bool firstTime=0;

        if (angulos-positive_w>0)
                        big2small_angle=0;
                else
                        big2small_angle=1;

        ros::Rate loop_rate(80);

        do{
                if(velocidades==0){
                        vel_msg.angular.z =0,
                        velocity_publisher.publish(vel_msg);
                }else{

                        if (car_pose.pose.orientation.w<0)
                                positive_w=degrees2radians(360)+
                                    car_pose.pose.orientation.w;
                        else
                                positive_w=car_pose.pose.
                                    orientation.w;

                        if (big2small_angle && clockwise==2) //
                            horario
                                end_angle = -(angulos-positive_w);
                        else if (clockwise==1 && big2small_angle
                            && angulos<positive_w && !firstTime) //
                            current_angle>angulos
                                end_angle = -(angulos-positive_w);
                                //cout<<"neg to pos"<<endl;
                        else
                        if(clockwise==3)
                                if(positive_w<degrees2radians(180)
                                    )
                                        end_angle = (angulos-
                                            positive_w);
                                        //cout<<"horarioPos"<<endl
                                            ;
```

```
                                    else
                                            end_angle = positive_w-
                                                angulos;
                                            //cout<<"horarioNeg"<<endl
                                                ;
                                else
                                        end_angle = angulos-positive_w,
                                        firstTime=1;

                                ros::spinOnce();
                                loop_rate.sleep();
                                if(end_angle>0.17){
                                velocity_publisher.publish(vel_msg);
                                }
                        }
                }while(end_angle>0.17 && velocidades>0);
                vel_msg.angular.z =0;
                velocity_publisher.publish(vel_msg);

}

double degrees2radians(double angle_in_degrees){
        return angle_in_degrees *PI /180.0;
}

void poseCallback(const geometry_msgs::PoseStamped::ConstPtr &
    pose_message){
        car_pose.pose.orientation.x=pose_message->pose.orientation
            .x;
        car_pose.pose.orientation.y=pose_message->pose.orientation
            .y;
        car_pose.pose.orientation.z=pose_message->pose.orientation
            .z;
        car_pose.pose.orientation.w=pose_message->pose.orientation
            .w;

        double t3= 2.0 * (car_pose.pose.orientation.w*car_pose.
            pose.orientation.z+car_pose.pose.orientation.x*car_pose
            .pose.orientation.y);
        double t4= 1.0-2.0 *((car_pose.pose.orientation.y*car_pose
            .pose.orientation.y)+car_pose.pose.orientation.z*
            car_pose.pose.orientation.z);

        car_pose.pose.orientation.w=std::atan2(t3, t4);

        //printf("%f\n",car_pose.pose.orientation.w);
}
```

```
int Server(){
        //**********SERVER************
        int socket_desc , client_sock , c , *new_sock;
        struct sockaddr_in server , client;
        char client_message[2000];

        char buf[100];



        //Create socket
        socket_desc = socket(AF_INET , SOCK_STREAM , 0);
        if (socket_desc == -1){
        printf("Could not create socket");
        }
        puts("Socket created");

        //Prepare the sockaddr_in structure
        server.sin_family = AF_INET;
        server.sin_addr.s_addr = inet_addr("192.168.1.64");
        server.sin_port = htons( 8080 );
        int count = 0;

        //Bind
                if( bind(socket_desc,(struct sockaddr *)&server ,
                    sizeof(server)) < 0){
                //print the error message
                perror("bind failed. Error");
                return 1;
        }
        puts("bind done");

        printf("IP address is: %s\n", inet_ntoa(server.sin_addr));
        printf("port is: %d\n", (int) ntohs(server.sin_port));

        //Listen
        listen(socket_desc , 3);

        //Accept and incoming connection
        puts("Waiting for incoming connections...");
        c = sizeof(struct sockaddr_in);


        while( (client_sock = accept(socket_desc, (struct sockaddr
            *)&client, (socklen_t*)&c)) ){
                puts("Connection accepted");
```

```
                count++;
                printf("Thread #%i \n",count);
                printf("Client IP address: %s\n", inet_ntoa(client
                    .sin_addr));
                printf("port is: %d\n", (int) ntohs(client.
                    sin_port));

                pthread_t sniffer_thread;
                new_sock = (int *)malloc(1);
                *new_sock = client_sock;

                if( pthread_create( &sniffer_thread , NULL ,
                    connection_handler , (void*) new_sock) < 0){
                        perror("could not create thread");
                        return 1;
                }


                //Now join the thread , so that we dont terminate
                    before the thread
                //pthread_join( sniffer_thread , NULL); // was
                    commented before
                puts("Handler assigned");
        }

        if (client_sock < 0){
                perror("accept failed");
                return 1;
        }
        return 0;
}

/*
* This will handle connection for each client
* */
void *connection_handler(void *socket_desc)
{
        //Get the socket descriptor
        int sock = *(int*)socket_desc;
        int read_size;
        char *message , client_message[2000],angulo_message[20],
            longitud_message[20],mode_message[1],velMax_message[5],
            angularMax_message[5];
        int numbytes;
        int i=0,e=0;
        float angulo=0,longitud=0;
        memset(client_message, 0, strlen(client_message));
```

```
while( (read_size = recv(sock , client_message , 2000 , 0)
    ) > 0 )
{
        //Send the message back to client
        //write(sock , client_message , strlen(
            client_message)); //no necesario ahora

//angulo
while(client_message[i]!='␣'){
        angulo_message[i]=client_message[i];
        i++;
}
angulo=angulo_message[0] - '0';
angulo=angulo+0.1*(angulo_message[2] - '0');
angulo=angulo+0.01*(angulo_message[3] - '0');
angulo=angulo+0.001*(angulo_message[4] - '0');
angulo=angulo+0.0001*(angulo_message[5] - '0');
angulo=angulo+0.00001*(angulo_message[6] - '0');
angulo=angulo+0.000001*(angulo_message[7] - '0');

//longitud
i++;
e=0;
while(client_message[i]!='␣'){
        longitud_message[e]=client_message[i];
        i++;
        e++;
}
i++;
mode_message[0]=client_message[i];

e=0;

i++;
i++;

while(client_message[i]!='␣'){
        velMax_message[e]=client_message[i];
        i++;
        e++;
}

longitud=longitud_message[0] - '0';
longitud=longitud+0.1*(longitud_message[2] - '0');
longitud=longitud+0.01*(longitud_message[3] - '0');
longitud=longitud+0.001*(longitud_message[4] - '0');
```

```
        longitud=longitud+0.0001*(longitud_message[5] - '0');
        longitud=longitud+0.00001*(longitud_message[6] - '0');
        longitud=longitud+0.000001*(longitud_message[7] - '0');

        //mode
        mode=mode_message[0] - '0';

        //max velocity
        if(e>3){
                e=3;
        }
        if (e==3){
                velMax=100*(velMax_message[0] - '0');
                velMax=velMax+10*(velMax_message[1] - '0');
                velMax=velMax+1*(velMax_message[2] - '0');
        } else if (e==2){
                velMax=10*(velMax_message[0] - '0');
                velMax=velMax+(velMax_message[1] - '0');
        } else {
                velMax=(velMax_message[0] - '0');
        }

        //max angle
        i++;
        e=0;

        while(client_message[i]!='␣'){
                angularMax_message[e]=client_message[i];
                i++;
                e++;
        }
        if(e>3){
                e=3;
        }
        if (e==3){
                angularMax=100*(angularMax_message[0] - '0');
                angularMax=angularMax+10*(angularMax_message[1] -
                    '0');
                angularMax=angularMax+1*(angularMax_message[2] - '
                    0');
        } else if (e==2){
                angularMax=10*(angularMax_message[0] - '0');
                angularMax=angularMax+(angularMax_message[1] - '0'
                    );
        } else {
                angularMax=(angularMax_message[0] - '0');
        }
```

```
        i=0;
        e=0;

        //printf("%s\n",angulo_message);
        //printf("%f\n",angulo);
        //printf("%f\n",longitud);
        //printf("%i\n",mode);
        //printf("%i\n",velMax);
        memset(client_message, 0, strlen(client_message));
        memset(angulo_message, 0, strlen(angulo_message));
        memset(longitud_message, 0, strlen(longitud_message));
        memset(mode_message, 0, strlen(mode_message));
        memset(velMax_message, 0, strlen(velMax_message));

        angulos=(double)angulo;
        velocidades=(double)longitud*5/2*velMax/100;
        switch (mode){
                case 0:
                        rotateABS (degrees2radians(200)*angularMax
                            /100);
                        break;
                case 1:
                        rotateREL (angulos, velocidades,
                            degrees2radians(200)*angularMax/100);
                        break;
                default:
                        break;
                }
        }

        if(read_size == 0)
        {

                //memset(client_message, 0, sizeof client_message)
                    ;
                puts("Client␣disconnected");
                fflush(stdout);
        }
        else if(read_size == -1)
        {
                perror("recv␣failed");
        }


        //Free the socket pointer
        free(socket_desc);
```

```
        close(sock);
        pthread_exit(NULL);
        return 0;
}
```

A.3.2 drone_controller.cpp

```
/*
* Author: David Arévalo Jiménez
* Year: 2017
* Treball de fi de grau EEIA
* University: Universitat de les Illes Balears (UIB)
*
*/



#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "geometry_msgs/PoseStamped.h"
#include "sensor_msgs/LaserScan.h"
#include <sstream>
#include <stdio.h>
#include <math.h>

#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h> //inet_addr
#include <unistd.h>    //write
#include <pthread.h> //for threading , link with lpthread
#include <cmath>

using namespace std;


//Global variables
ros::Publisher velocity_publisher; //create a publisher
ros::Subscriber pose_subscriber; //suscribe to Pose

geometry_msgs::PoseStamped drone_pose;
geometry_msgs::PoseStamped cat_msg;
geometry_msgs::Twist vel_msg;
geometry_msgs::Twist vel_msg_stop;

double angulos, velocidades,altura=3;

const double PI = 3.14159265359;
```

```
int mode=0,velMax=50,angularMax=50,alturaMax=20,areaMax=7;
bool rotating=false;

//metodos
void moveGoal(double angle, double longitud);
void rotateREL (double relative_angle, double linear_speed,double
    maxRotVel);
void rotateABS (double angular_speed);
double degrees2radians(double angle_in_degrees);
double getDistance(double x1, double y1, double x2, double y2);
void poseCallback(const geometry_msgs::PoseStamped::ConstPtr &
    pose_message);
int Server();
void checkHeight();
void *connection_handler(void *);


int main(int argc, char **argv)
{
        // Initiate new ROS node named "talker"
        ros::init(argc, argv, "drone_controller");
        ros::NodeHandle n;
        //subscribers and publishers
        velocity_publisher = n.advertise<geometry_msgs::Twist>("/
            cat/velocity", 100);
        pose_subscriber = n.subscribe("/cat/catPose", 10,
            poseCallback);
        double angular_speed;

        Server();

        return 0;

}
void moveGoal(double angle, double longitud){
        double distX,distY,x,y,velocidad;
        ros::spinOnce(); //obtener valores

        if(angle<0){
                angle=2*PI+angle;
        }
        x=longitud*areaMax/2*cos(angle);
        y=longitud*areaMax/2*sin(angle);

        distX=x-drone_pose.pose.position.x;
        distY=y-drone_pose.pose.position.y;
```

```
vel_msg.linear.x = 0;
vel_msg.linear.y = 0;
vel_msg.linear.z = 0;
vel_msg.angular.x = 0;
vel_msg.angular.y = 0;
vel_msg.angular.z = 0;

//Orientar el drona a la posicion inicial
        while(drone_pose.pose.orientation.w>0.5 ||
            drone_pose.pose.orientation.w<-0.5){
        rotating=true;
                if(drone_pose.pose.orientation.w>2  ||
                    drone_pose.pose.orientation.w<-0.5){
                vel_msg.angular.z = 2;
        }else{
                vel_msg.angular.z = 0.4;
        }
        if(drone_pose.pose.orientation.w<-0.5){
                vel_msg.angular.z=-vel_msg.angular.z;
        }
        velocity_publisher.publish(vel_msg);
        //cout<<drone_pose.pose.orientation.w<<endl;
}
if(rotating){
        vel_msg.angular.z = 0;
        velocity_publisher.publish(vel_msg);
        rotating=false;
}

//Comprobar posicion X
if(abs(distX) <0.2){
        vel_msg.linear.x =0;
} else {
        if(abs(distX) <0.7){
                velocidad=0.4;
        } else {
                velocidad=2.1;
        }
        if(abs(distX)>=abs(distY)){
                vel_msg.linear.x =velocidad*distX/abs(
                    distX);
        }else{
                vel_msg.linear.x =velocidad*distX/abs(
                    distY);
        }
}
```

```
        //Comprobar posicion Y
        if(abs(distY) <0.2){
                vel_msg.linear.y =0;
        } else {
                if(abs(distY) <0.7){
                velocidad=0.4;
                } else {
                velocidad=2.1;
                }
                if(abs(distX)>=abs(distY)){
                vel_msg.linear.y =velocidad*distY/abs(distX);
                }else{
                vel_msg.linear.y =velocidad*distY/abs(distY);
                }
        }
        checkHeight();
        velocity_publisher.publish(vel_msg);
}

void rotateREL (double relative_angle, double linear_speed,double
    maxRotVel){
        double angular_speed;
        //set a 0 linear velocity in the x-y-axis
        vel_msg.linear.x =linear_speed;
        vel_msg.linear.y =0;
        vel_msg.linear.z =0;
        //set a 0 angular velocity in the x-y-axis
        vel_msg.angular.x = 0;
        vel_msg.angular.y = 0;
        vel_msg.angular.z = 0; //por si acaso

        ros::spinOnce(); //actualiza valores

        if(relative_angle<=degrees2radians(90) || (relative_angle
            <=degrees2radians(270) && relative_angle>=
            degrees2radians(180))){
                if(relative_angle<=degrees2radians(90)){
                        angular_speed=maxRotVel-(relative_angle*
                            maxRotVel/degrees2radians(90));
                }else{
                        angular_speed=maxRotVel-((relative_angle-
                            degrees2radians(180))*maxRotVel/
                            degrees2radians(90));
                }
        }
        if(relative_angle>=degrees2radians(270) ||       (
            relative_angle<=degrees2radians(180) && relative_angle
```

```
            >=degrees2radians(90))){
                if(relative_angle>=degrees2radians(270)){
                        angular_speed=(relative_angle-
                            degrees2radians(270))*maxRotVel/
                            degrees2radians(90);
                }else{
                        angular_speed=(relative_angle+
                            degrees2radians(-270+180))*maxRotVel/
                            degrees2radians(90);}
        }
        if(relative_angle>=degrees2radians(90) &&
            relative_angle<=degrees2radians(270))
                vel_msg.angular.z =-abs(angular_speed);
        else
                vel_msg.angular.z =abs(angular_speed);

        if(linear_speed==0){
                vel_msg.angular.z = 0;
        }
        if(relative_angle>degrees2radians(180)){
                vel_msg.linear.x = -linear_speed;
        }

        if(abs(alturaMax*altura/100-drone_pose.pose.position.z)
            <0.2){
                vel_msg.linear.z=0;
        } else {
                if(alturaMax*altura/100-drone_pose.pose.position.z
                    <0){
                        vel_msg.linear.z=-1.5;
                } else {
                        vel_msg.linear.z=1.5;
                }
        }
        velocity_publisher.publish(vel_msg);
}

void rotateABS (double angular_speed){
        //set a 0 linear velocity in the x-y-axis
        vel_msg.linear.x =velocidades;
        vel_msg.linear.y =0;
        vel_msg.linear.z =0;
        //set a 0 angular velocity in the x-y-axis
        vel_msg.angular.x = 0;
        vel_msg.angular.y = 0;
        vel_msg.angular.z = 0; //por si acaso
```

```
        ros::spinOnce(); //si no da 0 w

        int clockwise=0;
        double positive_w=drone_pose.pose.orientation.w;

        if (drone_pose.pose.orientation.w<0)
                positive_w=degrees2radians(360)+drone_pose.pose.
                    orientation.w;

        double angulos360=angulos+2*PI;
        double w360=positive_w+2*PI;

        if (angulos>positive_w) //camino corto
                if((w360-angulos)<(angulos-positive_w))
                        vel_msg.angular.z =abs(angular_speed),
                            clockwise=3;//,cout<<"camino corto; "<<
                            w360<<endl; //camino corto
                else
                        vel_msg.angular.z =-abs(angular_speed),
                                    clockwise=0/*,cout<<angulos<<",
                                     "<<positive_w<<endl*/;
                else if ((angulos360-positive_w)<(positive_w-
                    angulos))
                        vel_msg.angular.z =-abs(angular_speed),
                            clockwise=1;
                else
                        vel_msg.angular.z =abs(angular_speed),
                            clockwise=2;

        double end_angle = angulos-positive_w;
        bool big2small_angle = 0;
        bool firstTime=0;

        if (angulos-positive_w>0)
                big2small_angle=0;
        else
                big2small_angle=1;

        ros::Rate loop_rate(80);

        do{
                if(velocidades==0){
                        vel_msg.angular.z =0;
                }else{
                        if (drone_pose.pose.orientation.w<0)
                                positive_w=degrees2radians(360)+
                                    drone_pose.pose.orientation.w;
```

```
                        else
                                positive_w=drone_pose.pose.
                                    orientation.w;

                        if (big2small_angle && clockwise==2) //
                            horario
                                end_angle = -(angulos-positive_w);
                        else if (clockwise==1 && big2small_angle
                            &&      angulos<positive_w && !firstTime
                            ) //neg a pos antihorario
                                end_angle = -(angulos-positive_w);
                        //cout<<"neg to pos"<<endl;
                        else
                                if(clockwise==3)
                                        if(positive_w<
                                            degrees2radians(180))
                                                end_angle = (
                                                    angulos-
                                                    positive_w); //
                                                    paso del
                                                    primero al
                                                    cuarto
                                                    cuadrante
                                        else
                                                end_angle =
                                                    positive_w-
                                                    angulos;
                                else
                                        end_angle = angulos-
                                            positive_w, //
                                            antihorario
                                        firstTime=1;
                }
                checkHeight();
                velocity_publisher.publish(vel_msg);

                ros::spinOnce();
                loop_rate.sleep();

        }while(end_angle>0.17 && velocidades>0);
        vel_msg.angular.z =0;
        checkHeight();
        velocity_publisher.publish(vel_msg);
}

double degrees2radians(double angle_in_degrees){
        return angle_in_degrees *PI /180.0;
```

```
}
double getDistance(double x1, double y1, double x2, double y2){
        return sqrt(pow((x1-x2),2)+pow((y1-y2),2));
}

void poseCallback(const geometry_msgs::PoseStamped::ConstPtr &
    pose_message){
        drone_pose.pose.position.x=pose_message->pose.position.x;
        drone_pose.pose.position.y=pose_message->pose.position.y;
        drone_pose.pose.position.z=pose_message->pose.position.z;

        drone_pose.pose.orientation.x=pose_message->pose.
            orientation.x;
        drone_pose.pose.orientation.y=pose_message->pose.
            orientation.y;
        drone_pose.pose.orientation.z=pose_message->pose.
            orientation.z;
        drone_pose.pose.orientation.w=pose_message->pose.
            orientation.w;

        double t3= 2.0 * (drone_pose.pose.orientation.w*drone_pose
            .pose.orientation.z+drone_pose.pose.orientation.x*
            drone_pose.pose.orientation.y);
        double t4= 1.0-2.0 *((drone_pose.pose.orientation.y*
            drone_pose.pose.orientation.y)+drone_pose.pose.
            orientation.z*drone_pose.pose.orientation.z);

        drone_pose.pose.orientation.w=std::atan2(t3, t4);

        //printf("%f\n",drone_pose.pose.orientation.w);
}

void checkHeight(){
        if(abs(alturaMax*altura/100-drone_pose.pose.position.z)
            <0.2){
                vel_msg.linear.z=0;
        } else {
                if(alturaMax*altura/100-drone_pose.pose.position.z
                    <0){
                        vel_msg.linear.z=-1.5;
                } else {
                        vel_msg.linear.z=1.5;
                }
        }
}

/*
```

```
* This will handle connection for each client
* */
void *connection_handler(void *socket_desc)
{
        //Get the socket descriptor
        int sock = *(int*)socket_desc;
        int read_size;
        char *message , client_message[2000],angulo_message[20],
            longitud_message[20],mode_message[1],velMax_message[5],
            angularMax_message[5],altura_message[5],
            alturaMax_message[5],areaMax_message[5];
        int numbytes;
        int i=0,e=0;
        float angulo=0,longitud=0;
        memset(client_message, 0, strlen(client_message));

        while( (read_size = recv(sock , client_message , 2000 , 0)
            ) > 0 )
        {
        //Angle
        while(client_message[i]!='␣'){
                angulo_message[i]=client_message[i];
                i++;
        }
        angulo=angulo_message[0] - '0';
        angulo=angulo+0.1*(angulo_message[2] - '0');
        angulo=angulo+0.01*(angulo_message[3] - '0');
        angulo=angulo+0.001*(angulo_message[4] - '0');
        angulo=angulo+0.0001*(angulo_message[5] - '0');
        angulo=angulo+0.00001*(angulo_message[6] - '0');
        angulo=angulo+0.000001*(angulo_message[7] - '0');

        //Longitud
        i++;
        e=0;
        while(client_message[i]!='␣'){
                longitud_message[e]=client_message[i];
                i++;
                e++;
        }
        i++;

        //Mode
        mode_message[0]=client_message[i];

        e=0;
```

```
        //Max linear velocity
        i++;
        i++;

        while(client_message[i]!='␣'){
                velMax_message[e]=client_message[i];
                i++;
                e++;
        }

        //String to float 6f
        longitud=longitud_message[0] - '0';
        longitud=longitud+0.1*(longitud_message[2] - '0');
        longitud=longitud+0.01*(longitud_message[3] - '0');
        longitud=longitud+0.001*(longitud_message[4] - '0');
        longitud=longitud+0.0001*(longitud_message[5] - '0');
        longitud=longitud+0.00001*(longitud_message[6] - '0');
        longitud=longitud+0.000001*(longitud_message[7] - '0');

        mode=mode_message[0] - '0';

        if(e>3){
                e=3;
        }
        if (e==3){
                velMax=100*(velMax_message[0] - '0');
                velMax=velMax+10*(velMax_message[1] - '0');
                velMax=velMax+1*(velMax_message[2] - '0');
        } else if (e==2){
                velMax=10*(velMax_message[0] - '0');
                velMax=velMax+(velMax_message[1] - '0');
        } else {
                velMax=(velMax_message[0] - '0');
        }

        //Max angle
        i++;
        e=0;

        while(client_message[i]!='␣'){
                angularMax_message[e]=client_message[i];
                i++;
                e++;
        }
        if(e>3){
                e=3;
        }
```

```
        if (e==3){
                angularMax=100*(angularMax_message[0] - '0');
                angularMax=angularMax+10*(angularMax_message[1] -
                    '0');
                angularMax=angularMax+1*(angularMax_message[2] - '
                    0');
        } else if (e==2){
                angularMax=10*(angularMax_message[0] - '0');
                angularMax=angularMax+(angularMax_message[1] - '0'
                    );
        } else {
                angularMax=(angularMax_message[0] - '0');
        }

        //Height
        i++;
        e=0;

        while(client_message[i]!='␣'){
                altura_message[e]=client_message[i];
                i++;
                e++;
        }
        if (e==3){
                altura=100;
        } else if (e==2){
                altura=10*(altura_message[0] - '0');
                altura=altura+(altura_message[1] - '0');
        } else {
                altura=(altura_message[0] - '0');
        }

        //Max height
        i++;
        e=0;

        while(client_message[i]!='␣'){
                alturaMax_message[e]=client_message[i];
                i++;
                e++;
        }
        if(e>3){
                e=3;
        }
        if (e==3){
                alturaMax=100*(alturaMax_message[0] - '0');
                alturaMax=alturaMax+10*(alturaMax_message[1] - '0'
```

```
            );
        alturaMax=alturaMax+1*(alturaMax_message[2] - '0')
            ;
    } else if (e==2){
        alturaMax=10*(alturaMax_message[0] - '0');
        alturaMax=alturaMax+(alturaMax_message[1] - '0');
    } else {
        alturaMax=(alturaMax_message[0] - '0');
    }

    //Max area for mode 3
    i++;
    e=0;

    while(client_message[i]!='␣'){
        areaMax_message[e]=client_message[i];
        i++;
        e++;
    }
    if(e>3){
        e=3;
    }
    if (e==3){
        areaMax=100*(areaMax_message[0] - '0');
        areaMax=areaMax+10*(areaMax_message[1] - '0');
        areaMax=areaMax+1*(areaMax_message[2] - '0');
    } else if (e==2){
        areaMax=10*(areaMax_message[0] - '0');
        areaMax=areaMax+(areaMax_message[1] - '0');
    } else {
        areaMax=(areaMax_message[0] - '0');
    }
    i=0;
    e=0;

    memset(client_message, 0, strlen(client_message));
    memset(angulo_message, 0, strlen(angulo_message));
    memset(longitud_message, 0, strlen(longitud_message));
    memset(mode_message, 0, strlen(mode_message));
    memset(velMax_message, 0, strlen(velMax_message));
    memset(angularMax_message, 0, strlen(angularMax_message));
    memset(altura_message, 0, strlen(altura_message));
    memset(alturaMax_message, 0, strlen(alturaMax_message));
    memset(areaMax_message, 0, strlen(areaMax_message));

    //global variables to avoid problems
    angulos=(double)angulo;
```

```
        velocidades=(double)longitud*5/2*velMax/100;


        switch (mode){
                case 0:
                        rotateABS (degrees2radians(360)*angularMax
                            /100);
                        break;
                case 1:
                        rotateREL (angulos, (double)velocidades,
                            degrees2radians(200)*angularMax/100);
                        break;
                case 2:
                        moveGoal(angulos, (double)longitud);
                        break;
                default:
                        break;
                }
        }

        if(read_size == 0)
        {

                //memset(client_message, 0, sizeof client_message)
                    ;
                puts("Client disconnected");
                fflush(stdout);
        }
        else if(read_size == -1)
        {
                perror("recv failed");
        }


        //Free the socket pointer
        free(socket_desc);
        close(sock);
        pthread_exit(NULL);
        return 0;
}
int Server(){
        //**********SERVER*************
        int socket_desc , client_sock , c , *new_sock;
        struct sockaddr_in server , client;
        char client_message[2000];

        char buf[100];
```

```
//Create socket
socket_desc = socket(AF_INET , SOCK_STREAM , 0);
if (socket_desc == -1){
        printf("Could not create socket");
}
puts("Socket created");

//Prepare the sockaddr_in structure
server.sin_family = AF_INET;
server.sin_addr.s_addr = inet_addr("192.168.1.46");
server.sin_port = htons( 8080 );
int count = 0;

//Bind
if( bind(socket_desc,(struct sockaddr *)&server , sizeof(
    server)) < 0){
        //print the error message
        perror("bind failed. Error");
        return 1;
}
puts("bind done");

printf("IP address is: %s\n", inet_ntoa(server.sin_addr));
printf("port is: %d\n", (int) ntohs(server.sin_port));

//Listen
listen(socket_desc , 3);
puts("Waiting for incoming connections...");
c = sizeof(struct sockaddr_in);

//Accept and incoming connection
while( (client_sock = accept(socket_desc, (struct sockaddr
     *)&client, (socklen_t*)&c)) ){
        puts("Connection accepted");
        count++;
        printf("Thread #%i \n",count);
        printf("Client IP address: %s\n", inet_ntoa(client
            .sin_addr));
        printf("port is: %d\n", (int) ntohs(client.
            sin_port));

        pthread_t sniffer_thread;
        new_sock = (int *)malloc(1);
        *new_sock = client_sock;

        if( pthread_create( &sniffer_thread , NULL ,
```

```
                    connection_handler , (void*) new_sock) < 0){
                        perror("could not create thread");
                        return 1;
                }
                //Now join the thread , so that we dont terminate
                    before the thread
                //pthread_join( sniffer_thread , NULL); // was
                    commented before
                puts("Handler assigned");
        }

        if (client_sock < 0){
                perror("accept failed");
                return 1;
        }
        return 0;
}
```

BIBLIOGRAFÍA

[1] BracaMan, "Programación Básica de Sockets en Unix para Novatos," 2016, Último acceso: 06/09/2017. [Online]. Available: http://es.tldp.org/Tutoriales/PROG-SOCKETS/prog-sockets.html 1.3

[2] A. Apacha, "Sensor fusion for robust outdoor Augmented Reality tracking on mobile devices," Master's thesis, Augsburg University, 2013, Último acceso: 05/09/2017. [Online]. Available: https://alexanderpacha.files.wordpress.com/2017/05/masterthesis-pacha.pdf 1.4, 4.1.1

[3] "ROS wiki: Understanding ROS Topics," 2017, Último acceso: 01/09/2017. [Online]. Available: http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics 3.3.1

[4] A. Apacha, "Sensor fusion demo for Android," 2017, Último acceso: 30/08/2017. [Online]. Available: https://github.com/apacha/sensor-fusion-demo 4.1.1

[5] D. Robledo Fernández, *Desarrollo de Aplicaciones para Android II.* Aula Mentor, 2015. 4.1.2

[6] W.-M. Lee, *Beginning Android 4 Application Development.* WROX PRESS, 2012. 4.1.2

[7] P.-E. Bergman, "OpenGL ES Tutorial for Android," 2010, Último acceso: 06/09/2017. [Online]. Available: https://blog.jayway.com/2010/01/14/opengl-es-tutorial-for-android-part-iv-adding-colors/ 4.2.1

[8] C. Aliotta, "NEHE ANDROID PORTS," 2016, Último acceso: 07/09/2017. [Online]. Available: https://insanitydesign.com/wp/projects/nehe-android-ports/ 4.2.2

[9] S. O. Ali, "Crystal Range Seekbar," 2017, Último acceso: 01/09/2017. [Online]. Available: https://github.com/syedowaisali/crystal-range-seekbar 4.2.3

[10] S. Crayons, "Compass," 2017, Último acceso: 01/09/2017. [Online]. Available: https://github.com/iutinvg/compass 4.2.3

[11] ERZ, "JoyStick," 2017, Último acceso: 28/08/2017. [Online]. Available: https://github.com/erz05/JoyStick 4.2.3

[12] AndroidSRC, "Android Client-Server Using Sockets - Client," 2017, Último acceso: 28/08/2017. [Online]. Available: http://androidsrc.net/android-client-server-using-sockets-client-implementation/ 4.3.1

[13] SGOLIVER, "Preferencias en Android II: PreferenceActivity," 2011, Último acceso: 01/09/2017. [Online]. Available: http://www.sgoliver.net/blog/preferencias-en-android-ii-preferenceactivity/ 4.4.1

[14] A. M. y Enrique Fernández, *Learning ROS for Robotics Programming*. Packt, 2013. 5.1

[15] "ROS wiki: ROS Tutorials," 2017, Último acceso: 29/08/2017. [Online]. Available: http://wiki.ros.org/ROS/Tutorials 5.1

[16] D. Haskins, *C-Programming in Linux*. BookBoon, 2009. 5.2

[17] R. wiki, "turtlesim," 2016, Último acceso: 05/09/2017. [Online]. Available: http://wiki.ros.org/turtlesim 5.3

[18] M. Pilgrim, *Dive Into Python 3*. Apress, 2004. 5.4.1

[19] Wikipedia, "Conversion between quaternions and Euler angles," 2017, Último acceso: 04/09/2017. [Online]. Available: https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles 5.4.2

[20] Openrobots, "Flying cat and mouse game tutorial," 2017, Último acceso: 31/08/2017. [Online]. Available: https://www.openrobots.org/morse/doc/latest/user/advanced_tutorials/flying_cat_and_mouse.html 5.4.3

[21] ——, "Rotorcraft Velocity motion controller," 2016, Último acceso: 07/09/2017. [Online]. Available: https://www.openrobots.org/morse/doc/stable/user/actuators/rotorcraft_velocity.html 5.4.3

[22] W. William Von Hagen, *Ubuntu Linux Bible*. Wiley Publishing, Inc., 2009. 6.2