BLUETAPP | APP MÓVIL PARA CONTROL DE HERRAMIENTAS POR BLUETOOTH

Rodríguez Pedraos Juan Pablo

rdzpedraos@gmail.com

Universidad Internacional del Trópico Americano

RESUMEN:

Este paper documenta el proceso de creación de una aplicación móvil capaz de enviar información por Bluetooth y extender sus funcionalidades, con el objetivo de controlar un dron casero. Se eligió React Native para el desarrollo debido a su capacidad de construir aplicaciones multiplataforma con un solo código. La comunicación Bluetooth se maneja con el módulo HC06 conectado a un Arduino, configurado mediante comandos AT.

I. INTRODUCCIÓN

Como parte de un objetivo académico, este paper tiene el fin de documentar el proceso llevado a cabo para crear una aplicación móvil que permitiera enviar información por Bluetooth, y documentar cómo extender dichas funcionalidades. El objetivo general del aula de clases es poder crear un dron casero, la idea es poder controlar la dirección y vuelo desde una aplicación móvil, justamente esto último fue lo que se desarrolló.

II. CONTENIDO

A. Qué es React Native.

React Native es una librería diseñada para la construcción de aplicaciones multiplataforma.

Anteriormente, las aplicaciones móviles tenían un reto en particular: debían ser desarrolladas en un lenguaje específico del sistema operativo en el que se deseaba; si se deseaba construir una aplicación que funcionara en un computador, en android y en la suite de apple era necesario desarrollar la misma aplicación en mínimo 3 lenguajes (las 3 plataformas).

A partir de ahí, salieron soluciones que permiten construir “aplicaciones multiplataforma”, la idea es desarrollar un sólo código que al “compilarse” realmente se transforme en el lenguaje del S.O. deseado y permite compilarlo, es decir, crea una capa extra que abstrae las funcionalidades propias de cada S.O.

Dentro de estas soluciones apareció React Native, basado en la librería de React (librería construida con JavaScript y que permite la “reactividad” de aplicaciones de forma muy

intuitiva) creó un conjunto de herramientas para el desarrollo de aplicaciones móviles y de escritorio.

La aplicación pudo ser construida en Kotlin, Swift, Android u otro lenguaje, pero se eligió Javascript usando React Native debido a su reciente demanda, y a que dentro de los objetivos de React (framework frontend más demandado en la actualidad) se encuentra dar soporte de forma nativa (sin react native) a las aplicaciones multiplataforma.

Existen otros beneficios como poder debuggear en tiempo real; usando una herramienta llamada Expo podemos compilar en tiempo real y ver nuestros cambios en el móvil como si estuviéramos desarrollando una aplicación web (casi en tiempo real), nos permite fácilmente switchear nuestras compilaciones entre los distintos S.O, conseguir funcionalidades propias de cada S.O. jugar con el bundle, e inclusive hacer un build y desplegarlo de una vez a la appstore del S.O.

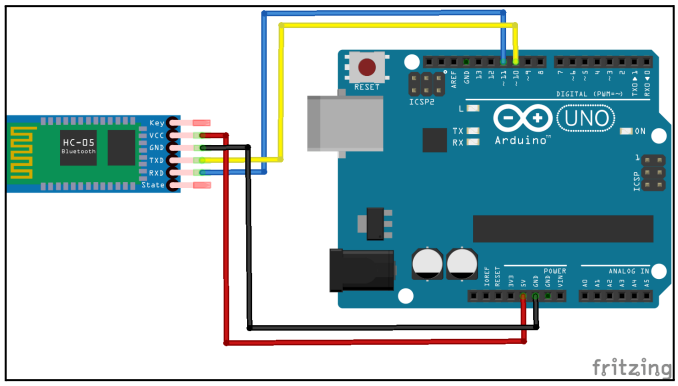
Por todo esto, se decidió trabajar desde este entorno buscando crear una solución que a su vez presentase un reto académico y profesional.

B. Configuración módulo HC06 para la comunicación por Bluetooth.

Se ha decidido manejar la comunicación vía Bluetooth ya que es una característica que tienen todos los celulares hoy en día; para lograr una comunicación efectiva a nuestro hardware haremos uso del módulo HC06 conectado a Arduino, este módulo es meramente con propósitos de investigación y prueba, realmente la aplicación es agnóstica al tipo de módulo que se use siempre que este permita ser esclavo, y a su vez no use low energy ya que estaremos enviando constantemente paquetes de información.

A continuación se va a presentar el código y el proceso realizado para darle este servicio a arduino y poder visualizar el envío de paquetes desde un equipo.

El Arduino debe estar conectado correctamente al módulo a través de los pines RX y TX, y que ambos dispositivos compartan un mismo GND para la conexión eléctrica adecuada.



Generalmente los pines RX y TX serán conectados a los pines 1, 0 o 11, 10 de la placa arduino respectivamente. Estos serán justamente los que permitan el flujo de comunicación con el Bluetooth.

Se utilizará un código básico en Arduino que permitirá establecer la comunicación con el módulo HC06 y enviar los comandos AT necesarios para la configuración. A continuación se muestra el código:

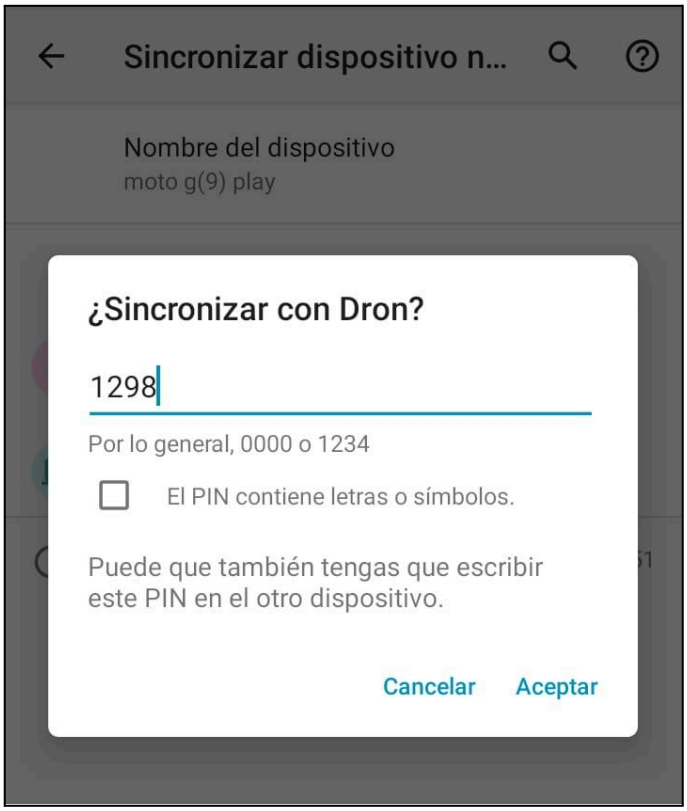
```ino
#include <SoftwareSerial.h>

SoftwareSerial BTSerial(11, 10); // RX (pin 0), TX (pin 1)

void setup() {
    Serial.begin(9600); // Inicializa la comunicación serial con la consola
    BTSerial.begin(9600); // Inicializa la comunicación serial con el HC-05
}
```

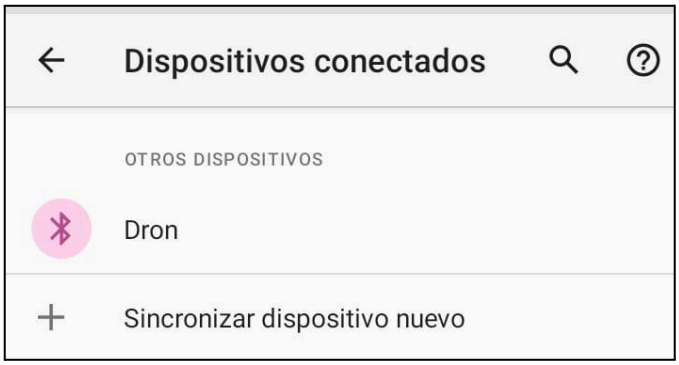
```ino
void loop() {
    // Leer datos desde el HC-05 y enviarlos a la consola serial
    if (BTSerial.available()) {
        char receivedChar = BTSerial.read();
        Serial.write(receivedChar);
    }

    // Leer datos desde la consola serial y enviarlos al HC-05
    if (Serial.available()) {
        char inputChar = Serial.read();
        BTSerial.write(inputChar);
    }
}
```

Este código nos permitirá configurar desde la consola nuestro Bluetooth y también visualizar los paquetes de información recibidos de otros dispositivos. Para configurar el bluetooth, abra la consola serial y siga las instrucciones:

1. Escriba **AT** y presione Enter, debe recibir una respuesta OK, lo que indica que el módulo está listo para recibir comandos AT.
2. A continuación, escriba **AT+NAME={name}** y presione Enter, donde {name} representa el nombre que desea asignar al módulo HC06, como por ejemplo, "Dron".
3. Escriba **AT+PIN={pin}** y presione Enter en la consola serial, donde {pin} puede ser reemplazado por el PIN deseado, como por ejemplo, "1298"

Si todo ha sido realizado con éxito, deberíamos visualizar nuestro Bluetooth desde el celular





C. Instalación del proyecto.

Para crear un proyecto en React Native usando Expo podemos seguir las instrucciones que aparecen en la documentación oficial de expo:

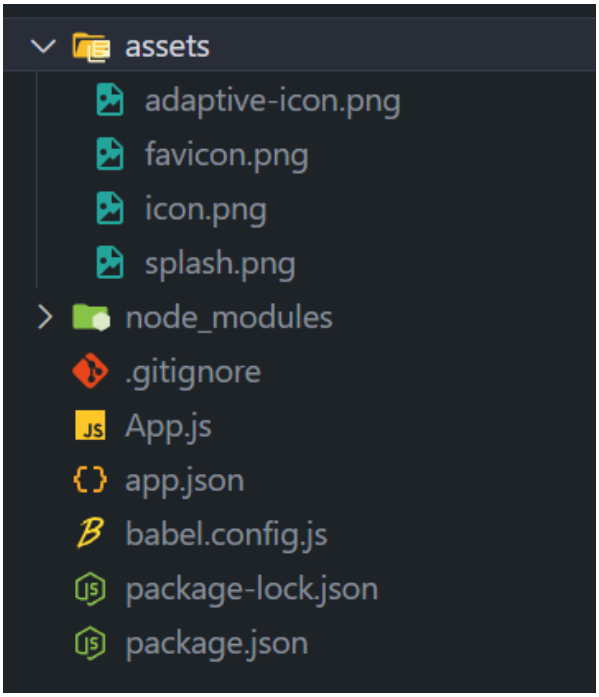
Antes que nada debemos instalar npm en nuestro equipo para que corra todos los comandos que se usan en el entorno de trabajo de javascript, npm es un administrador de paquetes, lo cual quiere decir que nos dará acceso a todas las librerías que se

han desarrollado para JavaScript y que son de código abierto.

Una vez instalado, procederemos a instalar el proyecto de react native con Expo usando el comando “npx create-expo-app {ProyectName} –template blank” (Expo). Esto nos instalará las dependencias de ser necesario y creará un boilerplate (una plantilla muy básica) para empezar a trabajar.

```
λ npx create-expo-app ControlApp --template bla
Need to install the following packages:
create-expo-app@2.3.5
Ok to proceed? (y)
```

Una vez instalado, veremos que se ha creado una carpeta con el nombre que definimos. Al abrir esta carpeta veremos la siguiente estructura:



Nos ha creado los archivos **package** y **package-lock**, encargados de guardar el historial de librerías, dependencias instaladas y las versiones de estas.

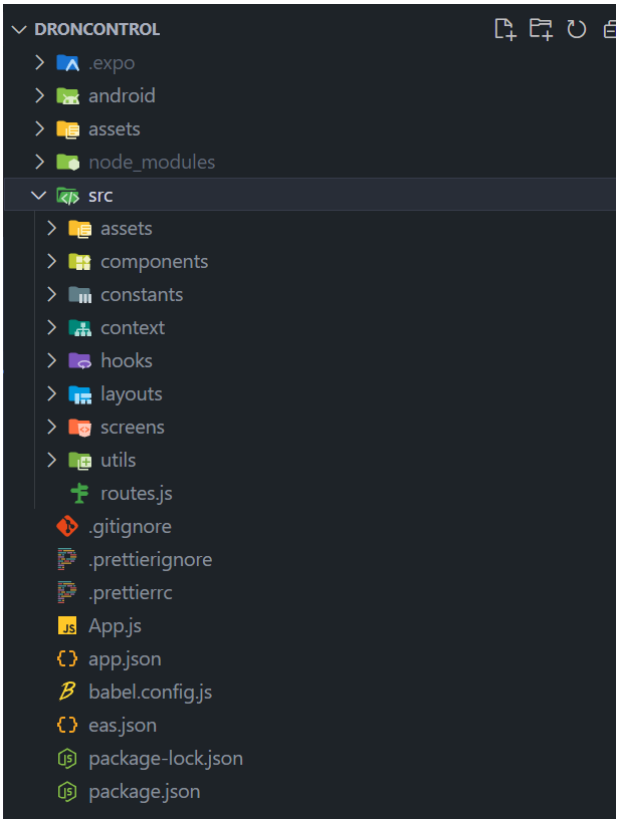
Tenemos **app.json** que contiene atributos clave para la definición de nuestro proyecto, como la versión, el nombre, las imágenes a usar para presentar la app, entre otras cosas.

El archivo **.gitignore** nos permitirá relacionar carpetas y archivos que no queramos que se trackeen en el versionamiento de cambios (Git), para este proyecto no hace falta ahondar en el tema pero puede consultar más acerca de git aquí.

Assets agrupa archivos multimedia que se usarán en la compilación de la aplicación, y **node_modules** contiene todas las dependencias usadas en el proyecto, normalmente esta última carpeta debe sólo existir en el proyecto local y nunca debe hacer parte del repositorio en la nube del proyecto.

Por último, **App.js** será el origen de nuestra aplicación.

Una vez entendido el contexto de la aplicación por defecto, veamos la estructura de carpetas diseñada para la aplicación:



Hemos agregado **prettier** para manejar un estándar de código, nos permitirá formatear y tener siempre una misma estructura de código, mejorando la lectura del código en cada archivo (Prettier).

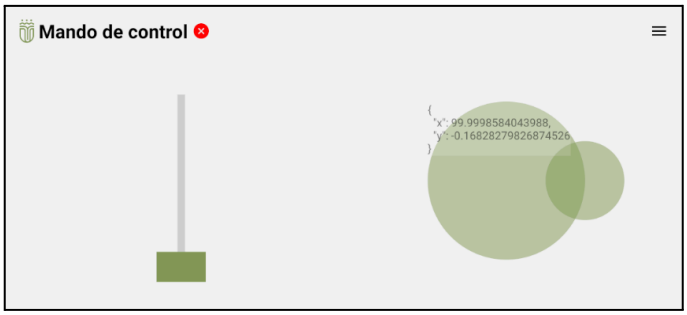
El archivo **eas.json** simplemente nos permitirá definir una estructura para la compilación de nuestro proyecto, por ahora este archivo y la carpeta Expo y Android no importa mucho, ya se explicará más adelante.

Por último, vemos que se ha creado una carpeta llamada **src** y contiene dentro de ella los folders **assets, components, constants, context, hooks, layouts, screens, utils**, y el archivo **routes.js**; todo esto se conoce normalmente como "estructura de carpetas" que va ligado a veces con la arquitectura que se usa, la idea de este documento no es comentar como funciona React, como se usa y por qué se definió estas carpetas; sólo tener presente que la idea es tener cada tipo de funcionalidad separada, si desea investigar más, siéntase cómodo de revisar el repositorio y consultar más sobre cada tema (ejemplo, si quiere saber más sobre la carpeta context, investigue react context, y así).

D. Desarrollo de aplicación.

Uno de los desafíos encontrados fue la incorporación de un módulo preexistente para el manejo de controles, específicamente un joystick, el cual no pudo ser integrado de manera satisfactoria. Ante esta situación, se tomó la decisión de desarrollar el componente de joystick desde cero.

El componente de joystick diseñado para la aplicación móvil presenta una funcionalidad clave al generar un vector con valores de posición en los ejes **X y Y**. Este vector puede tener una longitud que varía de 0 a un radio determinado, lo que permite al usuario controlar la dirección y la intensidad con la que debe moverse el dron de forma intuitiva.



Un aspecto importante a destacar es la implementación de la normalización de este vector. La normalización se lleva a cabo para asignar valores de potencia en un rango de 0 a 1 a cada uno de los cuatro motores del dron, permitiendo así un control preciso y equilibrado de los movimientos del dron en todas las direcciones.

Para ilustrar el funcionamiento de la normalización del vector en tres situaciones específicas, consideremos los siguientes ejemplos

I. X = 0, Y = 0 (dron sin movimiento)

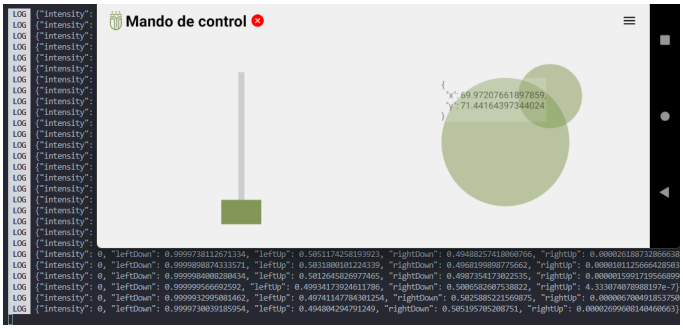
El estado por defecto pondrá en cada motor un valor medio que será ".5" (valor que es la mitad entre el mínimo 0 y el máximo 1).

II. X = radio, Y = 0 (a la derecha)

En esta situación, el usuario está indicando que desea que el dron se mueva hacia la derecha en su máxima capacidad. Esto se traduce en una potencia mayor en los motores leftDown y leftUp en comparación con los motores rightUp y rightDown. La normalización del vector asignará valores de potencia más altos a los motores izquierdos para lograr este movimiento deseado.

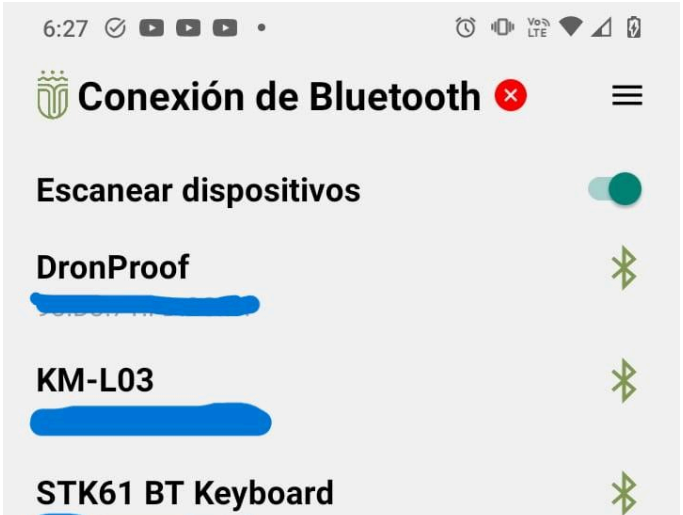
III. X = Y = radio/sqrt(2) (45 grados del primer cuadrante)

Si el vector está posicionado en un ángulo de 45 grados del primer cuadrante, esto indica un movimiento diagonal hacia la esquina superior derecha. En este caso, se aplicará una normalización del vector para asignar potencia de manera equilibrada a los motores correspondientes. Los motores leftUp y rightDown recibirán una potencia ligeramente mayor, mientras que el motor leftDown tendra casi el 100% de su potencia y rightUp será el que menos tenga.

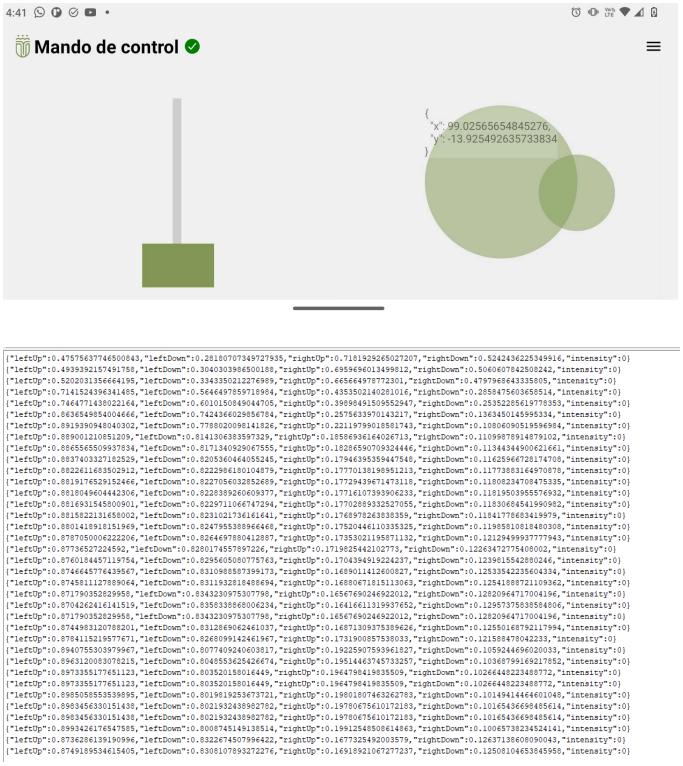


Así mismo, se implementó un botón de tipo rango, que permite variar entre 0 y 1 la intensidad con la que deben ir los motores.

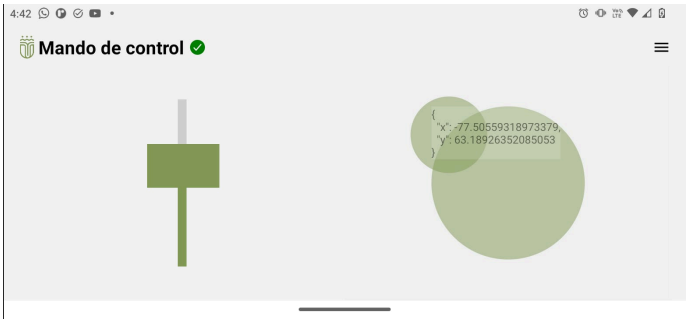
Por último, se implementó una interfaz que permite seleccionar el bluetooth al cual queremos conectarnos. Haciendo uso de la librería react-native-bluetooth-classic mostramos los dispositivos que están vinculados al móvil, en este caso nos conectamos al dispositivo que configuramos previamente.



Una vez realizada la conexión, veremos un chulito en verde en la barra superior de la aplicación, esto nos indicará que la conexión ha sido exitosa; al abrir la sección de control de dron, cada que alguno de los botones sea pulsado, se enviará un objeto json con la posición de los motores y la intensidad al dispositivo vinculado.



[1] Consola de arduino, donde está llegando la información



[2] Consola de arduino, donde está llegando la información

E. Compilación y despliegue

Es probable que desee descargar el proyecto para hacer ajustes y necesite poder instanciar la

herramienta de expo o generar el apk. Por ello, tenga en cuenta lo siguiente:

I. *Compilación en desarrollo, ver cambios en tiempo real.*

Configure una máquina virtual (puede ser las que provee android studio) o habilite las habilidades de desarrollador y la depuración por usb en su móvil, conéctelo al equipo y realice los siguientes comandos estando en la raíz del proyecto:

"*npx expo prebuild –clean":* En caso de modificar logos, el nombre de la app, o algo delicado como el módulo de bluetooth, deberá ejecutar el comando para volver a crear una instancia de la aplicación

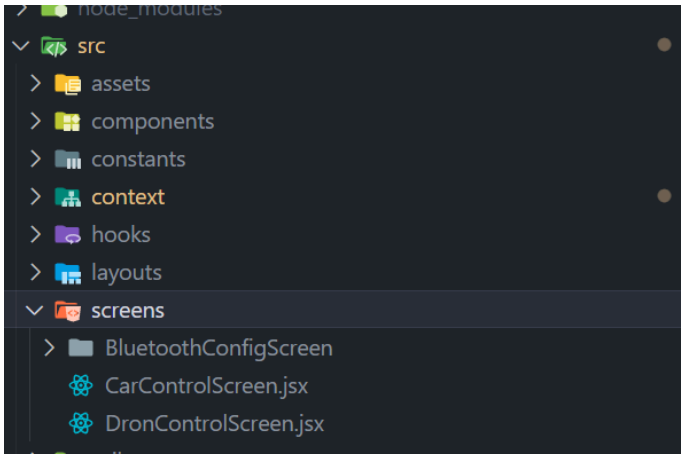
"*npx expo run:android":* Si simplemente está modificando la información embebida en la carpeta src o app.js, siéntase libre de simplemente ejecutar el comando y seguir las instrucciones que indique expo

"npx eas build --profile development --platform android": En caso de querer generar el apk, será necesario configurar un usuario y seguir las instrucciones expuestas aqui. Una vez realizado, debería solo ejecutar el comando cuando desee crear un apk.

F. Consideraciones.

El aplicativo es open source, y nada me haría más feliz que recibir sus pull request para mejorar el producto! Es por esto que quiero dejar unas consideraciones para que sea usable al interior de otras instituciones y explicar como implementar otros controles.

I. *¿Cómo puedo crear otras vistas?*



Dentro de src/screens deberemos crear nuestras vistas en React, cada screen en principio debería usar el layout principal "MainLayout"

```
export default function CarControlScreen(props) {

    return (
        <MainLayout title="Mando de carro" {...props}>
            <View style={[styles.container, { backgroundColor: COLORS.SECONDARY }]}>
                Hola!!
            </View>
        </MainLayout>
    );
}
```

Este componente MainLayout, se encuentra dentro de src/layouts y permite definir el título de la pantalla así como la dirección que debería usar (si la pantalla debería voltearse o no). Frente a "props" simplemente mantener dicho atributo para que el MainLayout siga funcionando.

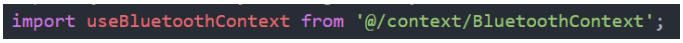
Si desea que esta ventana sea indexada en el menú lateral, por favor ingrese a **routes.js** y defina dicho componente dentro del arreglo "routes", puede copiar la estructura y seguir la idea, donde "name" es el nombre del componente (una llave única), le pasamos el componente (importado

previamente), y options son aquellos valores que toma la librería @react-navigation/drawer para generar las rutas.

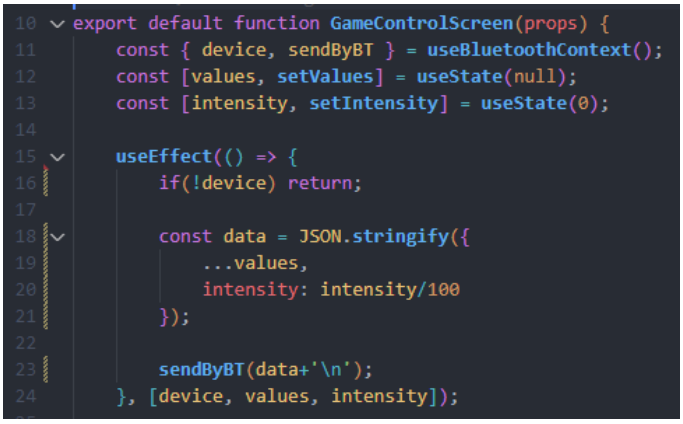
```
const routes = [
    {
        name: "CarControlScreen",
        component: CarControlScreen,
        options: {
            title: "Manubrio para coche",
            drawerIcon: ({ color }) => <ICONS name="car-sport-sharp" size={24} color={color} />,
        }
    },
    {
        name: 'BluetoothConfigScreen',
        component: BluetoothConfigScreen,
        options: {
            title: "Configuración bluetooth",
            drawerIcon: ({ color }) => <ICONS name="bluetooth-sharp" size={24} color={color} />,
        }
    }
]
```

II. ¿Cómo puedo usar el Bluetooth configurado?

Gracias a los contextos de React esto es muy sencillo! Simplemente importe la librería en su componente Screen, y traiga el método sendByBT, este método se encargará de enviar lo que usted le pase (debe ser una cadena de texto) si ya se ha vinculado un dispositivo por Bluetooth. Existen otros atributos que puede extraer como "device" que es nulo cuando no se ha vinculado a un dispositivo.



```
import useBluetoothContext from '@/context/BluetoothContext';
```



```
export default function GameControlScreen(props) {
    const { device, sendByBT } = useBluetoothContext();
    const [values, setValues] = useState(null);
    const [intensity, setIntensity] = useState(0);

    useEffect(() => {
        if(!device) return;

        const data = JSON.stringify({
            ...values,
            intensity: intensity/100
        });

        sendByBT(data+'\n');
    }, [device, values, intensity]);
```

III. ¿Cómo puedo cambiar los logos y colores de la aplicación?

Dentro de la carpeta "src/constants" podemos editar el archivo **colors.js, sizes.js, images.js** para definir los colores, tamaños y logos de la aplicación.

III. CONCLUSIONES

El desarrollo de la aplicación móvil utilizando React Native y el módulo HC06 para comunicación Bluetooth fue exitoso en permitir el control de un dron casero. La elección de React Native facilitó la creación de una solución multiplataforma, y la configuración del módulo Bluetooth fue sencilla y efectiva. El desarrollo de un componente de joystick personalizado y la normalización del vector de control permitieron un manejo preciso del dron.

El proyecto no solo cumplió con los objetivos académicos, sino que también presentó desafíos que fomentaron el crecimiento profesional. La estructura modular y el código open source ofrecen la posibilidad de futuras mejoras y adaptaciones. Este trabajo demuestra la viabilidad y ventajas de usar tecnologías modernas para proyectos de robótica y control remoto, brindando una base sólida para futuros desarrollos y colaboraciones.

Por último, si desea hacer seguimiento del proyecto, aportar o hacer uso del material construido, visite el repositorio de github.

IV. BIBLIOGRAFÍA

Expo. “Create your first app.” *Expo Docs*, https://docs.expo.dev/tutorial/create-your-first-app/. Accessed 2 6 2024.

Git. “Gitignore.” *Git everything is local*, https://git-scm.com/docs/gitignore. Accessed 2 June 2024.

Prettier. “What is Prettier?” *Prettier*, https://prettier.io/docs/en/. Accessed 2 June 2024.