COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

WIFI Mousr

Autor: Isabel Ugedo Perez

Director: Jing Jiang

Madrid

Junio de 2019

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 18 de Junio de 2019

ACEPTA

Fdo……………………… [illegible]

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

……………………………………………WIFI Mousr…………………………………………………………

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico ………4……. es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada

de otros documentos está debidamente referenciada.

Fdo.: Isabel Ugedo Perez Fecha: 06/06/2019

[illegible]

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Jing Jiang Fecha: 14/06/2019

Jing Jiang

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

WIFI Mousr

Autor: Isabel Ugedo Perez

Director: Jing Jiang

Madrid

Junio de 2019

WIFI MOUSR

Autor: **Ugedo Pérez, Isabel.**
Director: Jiang, Jing.
Entidad Colaboradora: UIUC - Universidad de Illinois en Urbana-Champaign.

RESUMEN DEL PROYECTO

"WIFI Mousr" es una actualización del juguete automatizado para gatos "Mousr", creado por la start-up Petronics. Este producto consiste en un ratón mecanizado, equipado con distintos sensores de medición de presencia y distancia, dos motores y dos microcontroladores. Su objetivo es proporcionar un juego y servir como distracción para mascotas, al ser capaz de desplazarse por una habitación sin ser atrapado por un gato, así como evitar obstáculos.

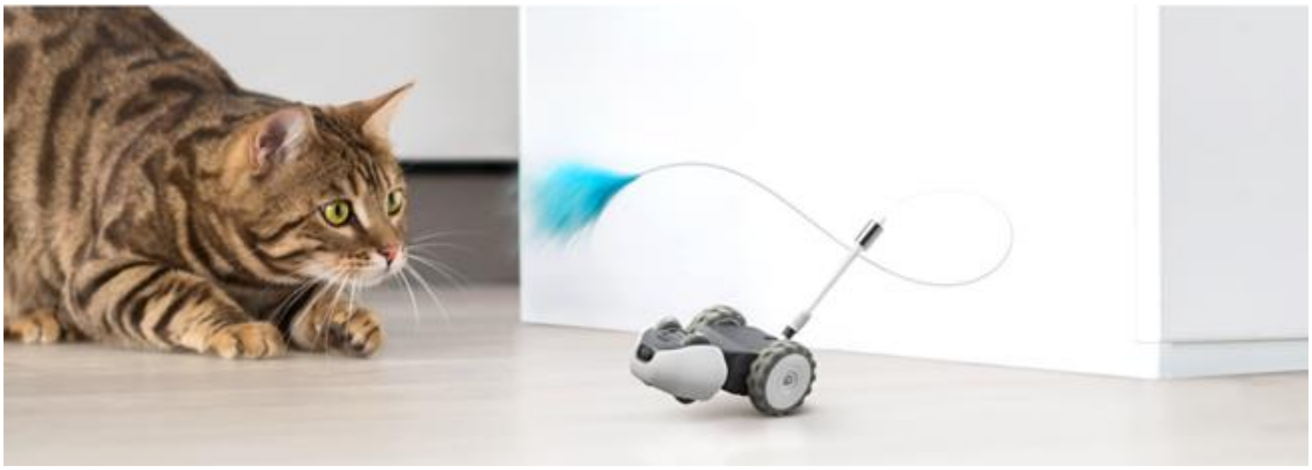


Fig. 1 Mousr by Petronics

Mousr ha ganado popularidad rápidamente, ya que cubre la necesidad de mantener a tu mascota entretenida y activa, sin requerir esfuerzo por parte del usuario. Funciona a través de una aplicación móvil que permite seleccionar sus desplazamientos en tiempo real, o bien activar el modo 'Auto-Play', en el que Mousr se moverá automáticamente sin necesidad de introducir instrucciones.

La versión de Mousr que se encuentra actualmente en el mercado utiliza una conexión Bluetooth para transmitir datos entre microcontrolador y smartphone, lo cual tiene un inconveniente importante: el alcance de la señal está limitado a aproximadamente 10 metros. Esto supone que e n el modo de juego dirigido el usuario tendrá que caminar detrás del dispositivo, lo que puede resultar molesto. En cuanto a la funcionalidad "Auto-Play", esto implica que no se podrá activar remotamente, por lo que el juguete nunca se podrá utilizar cuando el animal esté solo en casa. La posibilidad de mejorar estos aspectos puede aumentar significativamente las ventas de Petronics.

Un breve estudio de mercado revela que la mayoría de juguetes interactivos en venta están diseñados para funcionar durante el día sin necesidad de activación manual. En general, esto se consigue a través de temporizadores que apagan y encienden los dispositivos cada cierto tiempo. Es una solución simple y eficaz, además de suponer un aumento insignificante del precio. Sin embargo, en el caso de Mousr hay otras posibilidades, gracias a la aplicación móvil que incorpora.

La solución que se propone a los inconvenientes mencionados tiene dos elementos. Por un lado, el desarrollo de "WIFI Mousr", un juguete que mantiene la funcionalidad original sustituyendo la conexión Bluetooth por un protocolo BluFi. Será necesario para ello emplear un nuevo microcontrolador que contenga tanto radio Bluetooth como WIFI.

Por otro lado, como respuesta a la necesidad de activación remota de Mousr, se diseñará una nueva funcionalidad: “Scheduled Play”. La idea es permitir al usuario programar horarios de juego a través de la aplicación, de modo que Mousr se encienda o apague en el momento indicado.

El principal beneficio de la conexión BluFi frente al bluetooth es que garantiza al usuario poder controlar el dispositivo desde cualquier parte de la casa, siempre que la señal del rúter sea suficientemente fuerte.

Además de el mayor alcance de la señal, WIFI presenta otras ventajas sobre Bluetooth, especialmente respecto a ancho de banda y velocidad de transmisión de datos. Esto conlleva que es posible enviar archivos más complejos, así como mas cantidad de datos. Esta mejora será aprovechada por Petronics en un futuro para incorporar capacidades de análisis de datos e inteligencia artificial en su producto.

En este proyecto se utilizará WIFI “802.11 g/n” y BT “v4.2”. Ambas señales funcionan en frecuencia 2.4 GHz, pero la velocidad de transferencia teórica de este tipo de WIFI es de hasta 300Mpbs, frente a los 25Mpbs que aporta el Bluetooth. Igual de significativa es la diferencia en el alcance de la señal, 70 metros teóricos en el caso de WIFI y tan solo 10 en el del Bluetooth.

La principal desventaja de estas modificaciones será un aumento en el consumo energético. Esto se debe principalmente a que una conexión WIFI requiere mucha más potencia que su equivalente en Bluetooth. En concreto, y según las cifras del datasheet del micrprocesador, mantener la radio WIFI activa consume 190 mA, frente a el consumo de 100 mA del Bluetooth. Esta diferencia del 50% tendrá un gran efecto sobre el dispositivo. Además, al incorporar el “Scheduled Play” el microcontrolador nunca se podrá apagar completamente, sino que debe quedar en Light Sleep/Deep Sleep mode. Este modo de operación consume apenas 0.8 mA, por lo que resulta despreciable frente al consumo del WIFI.

A alto nivel, los principales objetivos del proyecto son realizar un estudio teórico del rendimiento y confirmar la rentabilidad del WIFI Mousr, demostrar interacción con Mousr a través de conexión WIFI entre microcontrolador y app, y desarrollar un prototipo funcional en PCB.

El protocolo BlueFi implementado consiste en enviar desde la app al microcontrolador las credenciales (SSID y password) del rúter del hogar, utilizando Bluetooth, para que a continuación tanto el micro como el smartphone queden conectados al rúter y obtengan acceso a la red WIFI. Esta conexión permite enviar archivos de diferentes tipos a través de una dirección IP local.

Por otro lado, para obtener un prototipo funcional se diseñarán dos circuitos, de datos y potencia. El circuito de datos se encarga de la transmisión de señales digitales entre los sensores de Mousr y el nuevo microcontrolador, mientras que el circuito de potencia debe proporcionar a cada uno de ellos el voltaje adecuando para su funcionamiento óptimo. Estos sensores son idénticos a los empleados por la versión original de Mousr con una novedad: la incorporación de un display del estado de la batería a través del LED RGB. Los elementos a incorporar en la PCB son IMU (Inertial Measurement Unit), TOF (Time of Flight), IR Receiver (infrarrojo), IR LED, RGB LED y botón.

El IMU se emplea para detectar cuando Mousr esta boca abajo, o en posición vertical atrapado por el gato. El TOF permite evitar obstáculos en el recorrido, y el transceptor IR sirve como detector de gradientes térmicos generados durante la aproximación de un animal. Como se explica anteriormente, el LED RGB se utilizará como display del estado de potencia (ON/OFF) del dispositivo, así como del nivel de betería. El botón se usará para apagar y encender Mousr manualmente, así como para activar el BluFi y consultar el nivel de batería. Este flujo de actividades se muestra en la Figura 2.

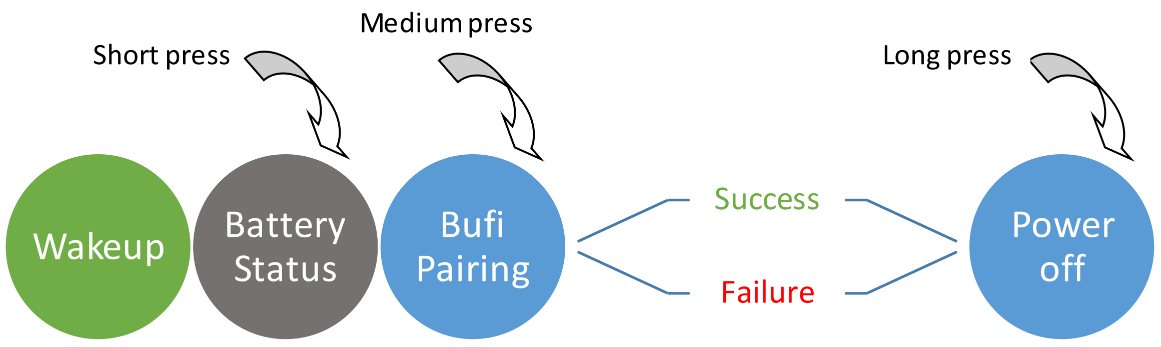


Fig. 2 Flujo de actividad del botón

El cuerpo principal del dispositivo no entra en el alcance de este proyecto. El cuerpo consiste en un segundo microcontrolador conectado a los motores de Mousr y a la batería, al que se podrán enviar instrucciones a través de una conexión UART. Para ello se usará un micro conector de 12 puertos, que transmite tanto corriente procedente de la batería como paquetes de datos. Sin embargo, este micro cuenta con su propio protocolo de reset y es difícil de controlar.

Los sensores se implementarán a través de la PCB diseñada, que incorporará los circuitos de datos y potencia. La alimentación provendrá de la batería situada en el cuerpo de Mousr. La placa incluirá condensadores especificados por el fabricante de cada sensor para reducir el ruido de lectura. Contendrá además las resistencias pull-up del protocolo I2C en el caso del TOF y el IMU. El software empleado en el cadding es EAGLE y el fabricante que se usará para imprimir es PCBWay, que debe dar su visto bueno a la impresión tras realizar una auditoría de las conexiones de la placa.

En este proyecto se utiliza el chip ESP32 para el prototipo WIFI Mousr, debido a su popularidad para aplicaciones WIFI y amplia disponibilidad de documentación. El modelo escogido dentro de" ESP32 series" es el "Wrover Kit", debido a su ampliación del número de pins GPIO, y a la existencia de dos pines con acceso al bus I2C. ESP32 admite diferentes lenguajes, entre ellos Arduino que es muy conveniente debido a que incorpora comandos simplificados del protocolo I2C, que será necesario para la comunicación con los sensores. Arduino además cuenta con funciones especifícas de WIFI y bluetooth, dentro de las librerías diseñadas para integración con ESP32.

En algunas secciones del código se emplean comandos de la librería de FreeRtos para Arduino. FreeRtos es un sistema operativo en tiempo real (RTOS), lo que significa que intercala tareas de forma instantánea dando la impresión de que todas se estén procesando continua y simultáneamente, gracias a su kernel multitarea. Este proceso se muestra en la Figura 3. Su uso es habitual en sistemas integrados, y es compatible con Arduino ya que está implementado en lenguaje C.

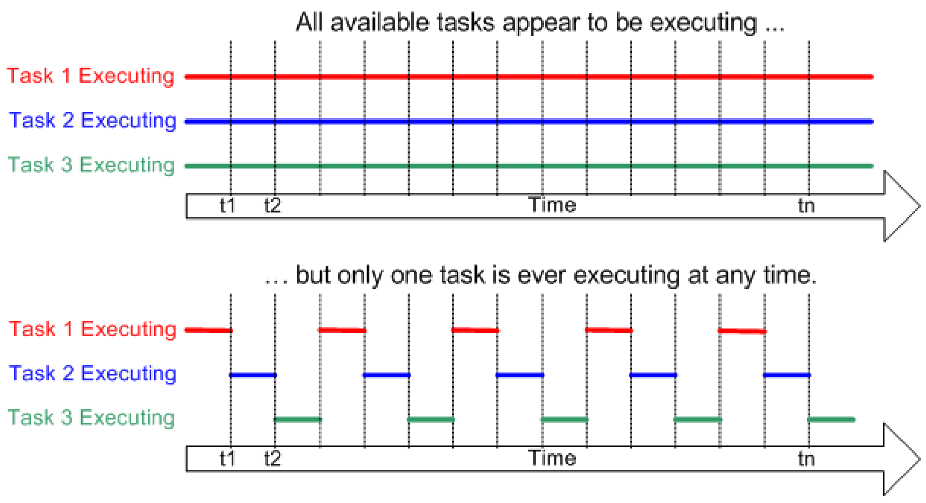


Fig. 3 Comportamiento de RTOS

Finalmente, el código del microprocesador cumple funciones secundarias a la conexión WIFI, entre ellas dirigir el bus I2C, medir el nivel de batería y controlar el display del led RGB. Esta funcionalidad se implementará en un bucle while (true), que llamará a funciones declaradas externamente al mismo.

El protocolo de comunicación bidireccional I2C se basa en dos señales síncronas, SCL (serial clock line) y SDA (serial data line) que se envían entre "master"(micro) y "slave" (sensor). Es una forma estándar de modificar los ajustes y parámetros del sensor alterando determinados valores binarios que este almacena, y de reclamar y transmitir datos. Para su correcto funcionamiento es importante que las vías de SCL y SDA en la PCB tengan longitudes similares, debido a que I2C es muy sensible a retrasos o adelantos en los flancos.

Por último, el 'Scheduled Play' se administra guardando las horas enviadas desde la aplicación en la memoria EEPROM de ESP32, o "electrically erasable programmable read-only memory", con un formato que se explicará en los siguientes documentos. El micrcontrolador las comparará cada cierto cierto y contrastará con la hora real, que debe mantener actualizada indiferentemente de su estado de potencia (ON/OFF). Se aprovechará la conexión a la red WIFI para obtener la hora real a través de un NTP o Network Time Protocol, y será necesario adaptar el formato de la información recibida. En este prototipo no se tendrá en cuenta la fecha, ya que añadirla a la funcionalidad existente es sencillo y no aporta contenido al proyecto, sino que recrea el mismo concepto que la hora. Se utilizará un contador interno para volver a encender ESP32 a la hora establecida, siempre obteniendo el intervalo de tiempo que se debe asignar al contador antes de entrar en light sleep mode.

WIFI MOUSR

Author: **Ugedo Pérez, Isabel.**
Director: Jiang, Jing.
Associated Entity: UIUC - University of Illinois at Urbana-Champaign.

PROJECT SUMMARY

"WIFI Mousr" is an update of the automated toy for cats "Mousr", created by the start-up Petronics. This product consists of a mechanized mouse, equipped with different sensors for presence and distance measurement, two motors and two microcontrollers. Its goal is to serve as a pet distraction by providing an engaging game, being able to move around a room without being caught by a cat while as avoiding obstacles.

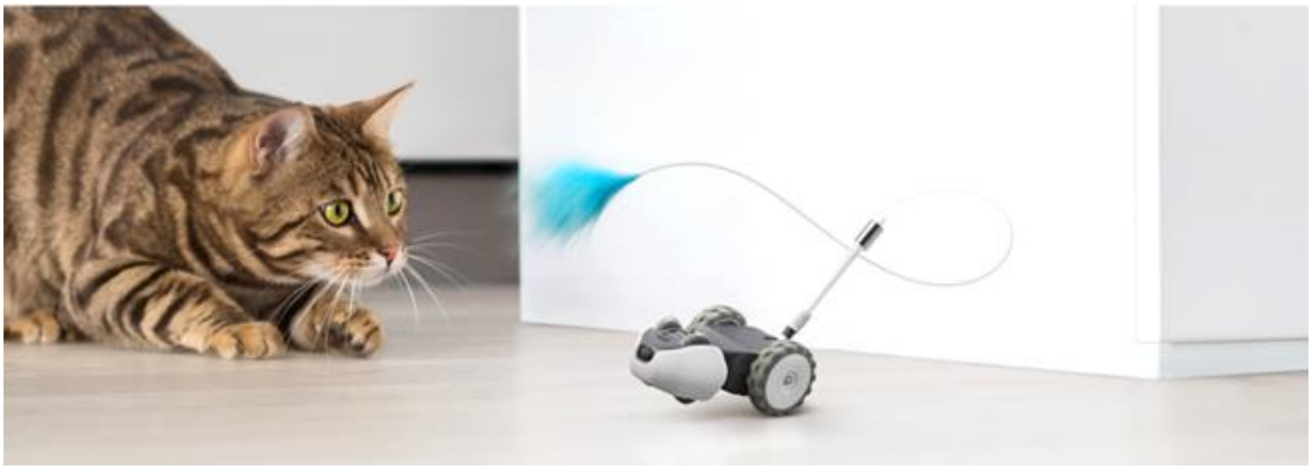


Fig. 1 Mousr by Petronics

Mousr has quickly become popular among cat owners in the US, since it covers the need to keep your pet entertained and active, requiring little to no effort on part of the user. It works through a mobile application that allows you to control the device´s movements remotely in real time, or to activate the 'Auto-Play' mode, in which Mousr will move automatically guided by its sensors.

The version of Mousr that is currently on the market uses a Bluetooth connection to transmit data between microcontroller and smartphone, which has one major drawback: the signal range is limited to approximately 10 meters. This means that in the “live play” mode the user will have to physically follow the device, which might prove annoying and disengage the consumer. As for the "Auto-Play" functionality, this means that it cannot be activated remotely, so the toy can never be used when the animal is alone at home. The possibility of improving these aspects can significantly increase Petronics´ sales.

A brief market study reveals that most interactive toys for sale are designed to work during the day without the need for manual activation. In general, this is achieved through timers that turn the devices off and on every so often. It is a simple and effective solution, as well as inexpensive. However, in the case of Mousr there are other possibilities, thanks to the mobile application that it incorporates.

The solution proposed to the aforementioned drawbacks has two elements. On the one hand, the development of "WIFI Mousr", a toy that maintains the original functionality by replacing the Bluetooth connection with a BluFi protocol. It will be necessary to use a new microcontroller that contains both Bluetooth radio and WIFI. On the other hand, in response to the need for remote activation of Mousr, a new functionality will be designed: "Scheduled Play". The idea is to allow the user to schedule play times through the application, so that Mousr turns on or off at the indicated time.

The main benefit of the BluFi connection to Bluetooth is that it guarantees that the user will be able to control the device from any part of the house, as long as the signal from the router is strong enough. In addition to the greater range of the signal, WIFI presents other advantages over Bluetooth, especially regarding bandwidth and data transmission speed. This means that it is possible to send more complex files, as well as more data. This improvement will be exploited by Petronics in the future to incorporate data analysis and artificial intelligence capabilities into its product.

In this project WIFI "802.11 g / n" and BT "v4.2" will be used. Both signals work in 2.4 GHz frequency, but the theoretical transfer speed of this type of WIFI is up to 300Mpbs, compared to the 25Mpbs provided by Bluetooth. Equally significant is the difference in the scope of the signal, 70 theoretical meters in the case of WIFI and only 10 in the case of Bluetooth.

The main disadvantage of these modifications will be an increase in energy consumption. This is mainly due to the fact that a WIFI connection requires much more power than its Bluetooth equivalent. In particular, and according to the figures of the microprocessor's datasheet, keeping the active WIFI radio consumes 190 mA, compared to the 100 mA consumption of the Bluetooth. This difference of 50% will have a great effect on the device. In addition, by incorporating the "Scheduled Play" the microcontroller can never be completely turned off, but must remain in the Light Sleep / Deep Sleep mode. This mode of operation consumes only 0.8 mA, so it is negligible compared to the consumption of WIFI.

At high level, the main objectives of the project are to perform a theoretical study of the performance and confirm the profitability of WIFI Mousr, demonstrate interaction with Mousr through WIFI connection between microcontroller and app, and develop a functional prototype in PCB.

The BlueFi protocol implemented consists of sending the credentials (SSID and password) of the home router, using Bluetooth, from the app to the microcontroller, so that both the micro and the smartphone are connected to the router and get access to the WIFI network. This connection allows you to send files of different types through a local IP address.

On the other hand, to obtain a functional prototype, two circuits of data and power will be designed. The data circuit is responsible for the transmission of digital signals between the Mousr sensors and the new microcontroller, while the power circuit must provide each of them with the appropriate voltage for optimal operation. These sensors are identical to those used by the original version of Mousr with a novelty: the incorporation of a display of the state of the battery through the RGB LED. The elements to incorporate in the PCB are IMU (Inertial Measurement Unit), TOF (Time of Flight), IR Receiver (infrared), IR LED, RGB LED and button.

The IMU is used to detect when Mousr is upside down, or in an upright position trapped by the cat. The TOF allows to avoid obstacles in the route, and the IR transceiver serves like detector of thermal gradients generated during the approach of an animal. As explained above, the RGB LED will be used as a display of the power status (ON / OFF) of the device, as well as the level of battery. The button will be used to turn the Mousr on and off manually, as well as to activate the BluFi and check the battery level. This flow of activities is shown in Figure 2.

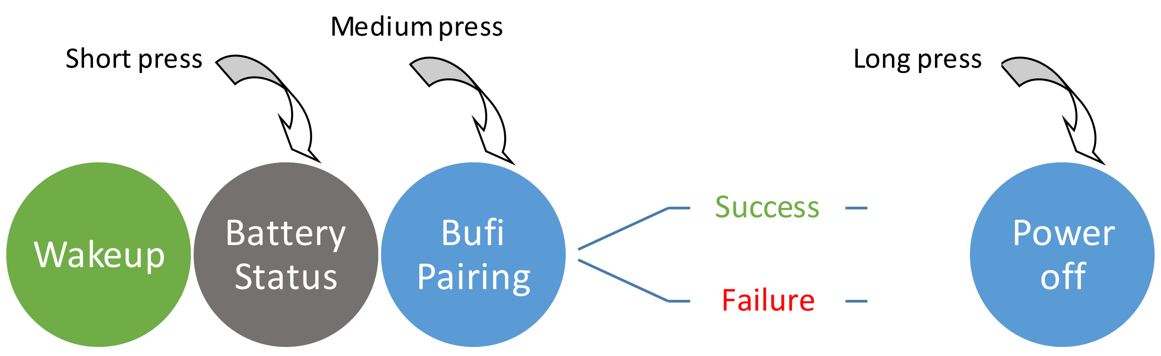


Fig. 2 Flujo de actividad del botón

The main body of the device does not fall within the scope of this project. The body consists of a second microcontroller connected to the Mousr motors and the battery, to which instructions can be sent through a UART connection. For this, a 12-port micro connector will be used, which transmits both current from the battery and data packets. However, this micro has its own reset protocol and is difficult to control.

The sensors will be implemented through the designed PCB, which will incorporate the data and power circuits. The power will come from the battery located in the body of Mousr. The plate will include capacitors specified by the manufacturer of each sensor to reduce reading noise. It will also contain the pull-up resistors of the I2C protocol in the case of the TOF and the IMU. The software used in the cadding is EAGLE and the manufacturer that will be used to print is PCBWay, which must give its approval to the print after performing an audit of the board connections.

In this project the ESP32 chip is used for the WIFI Mousr prototype, due to its popularity for WIFI applications and wide availability of documentation. The model chosen within "ESP32 series" is the "Wrover Kit", due to its extension of the number of GPIO pins, and the existence of two pins with access to the I2C bus. ESP32 supports different languages, including Arduino, which is very convenient because it incorporates simplified I2C protocol commands, which will be necessary for communication with the sensors. Arduino also has specific functions of WIFI and Bluetooth, within the libraries designed for integration with ESP32.

In some sections of the code, commands from the FreeRtos library for Arduino are used. FreeRtos is a real-time operating system (RTOS), which means that it inserts tasks instantaneously giving the impression that all are being processed continuously and simultaneously, thanks to its multi-tasking kernel. This process is shown in Figure 3. Its use is common in integrated systems, and it is compatible with Arduino since it is implemented in C language.

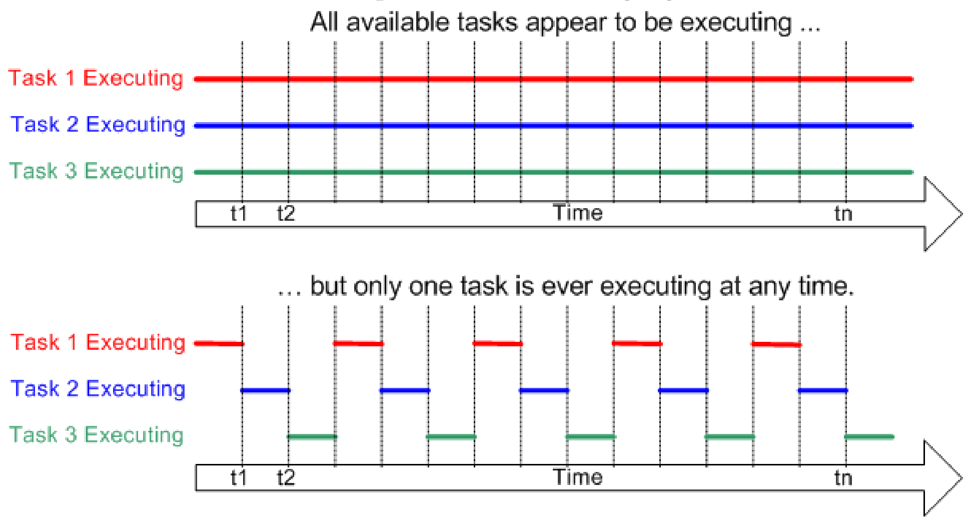


Fig.3 RTOS behavior

Finally, the microprocessor code fulfills secondary functions to the WIFI connection, including directing the I2C bus, measuring the battery level and controlling the RGB LED display. This functionality will be implemented in a while loop (true), which will call functions declared externally to it.

The I2C bidirectional communication protocol is based on two synchronous signals, SCL (serial clock line) and SDA (serial data line) that are sent between 'master' (micro) and 'slave' (sensor). It is a standard way to modify the settings and parameters of the sensor by altering certain binary values that it stores, and to claim and transmit data. For proper operation it is important that the SCL and SDA paths in the PCB have similar lengths, because I2C is very sensitive to delays or advances in the flanks.

Finally, the 'Scheduled Play' is managed by saving the hours sent from the application in the EEPROM memory of ESP32, or “electrically erasable programmable read-only memory”, with a format that will be explained in the following documents. The microcontroller will compare them every certain time and will contrast with the real time, which must be updated regardless of its power status (ON / OFF). The connection to the WIFI network will be used to obtain the real time through an NTP or Network Time Protocol, and it will be necessary to adapt the format of the received information. In this prototype the date will not be taken into account, since adding it to the existing functionality is simple and does not contribute content to the project, but rather recreates the same concept as the time. An internal counter will be used to restart ESP32 at the set time, always obtaining the time interval that must be assigned to the counter before entering light sleep mode.

Table of Contents

Project development

1.Introduction

“WIFI Mousr “is meant to be an upgrade of an existing product, “Mousr” [PETR] by Petronics. This device is a mouse shaped robot invented to entertain cats, able to run around the room avoiding any obstacles such as walls, furniture or the cat itself. Asides from entertaining cats, Mousr provides them with an incentive for exercise, which is essential for the animal’s well being and social behavior. There is a developing market for this type of pet toy, since it contributes to not only the animal’s but the owner´s well being. At this point there are few automatized cat toys available to the public, and Petronics has managed to continuously lead the market by differentiating their products, adding user interaction capabilities.

Fig 1. Mousr by Petronics

In Mousr, this user interaction is possible through the Auto-Play mode, which enables real-time remote control of the engines through an Android app [PETR]. It is also possible to select an automatic play configuration, available in different difficulty levels to fit each cat´s stamina and motivation.

Mechanically, Mousr is a combination of several movement and acceleration sensors, engines and a moving tail. Its integration is accomplished through two microcontrollers which use UART [RINC] connections to send and receive data. Additionally, the device relies on a Bluetooth connection to a smartphone, used to send simple instructions from the app to the main microcontroller.

The aim of this project is to replace this Bluetooth bond with a WIFI connection, which should be implemented via an IP address by the client’s specification. While this approach would result in a shorter battery life, it has many advantages.

First and foremost, WIFI creates a longest signal range which means the user would not have to worry about maintaining proximity to the moving device, which might prove inconvenient and even challenging. The only requirement would be for the home router signal to be strong enough to reach both the smartphone and the Mousr device. Another advantage would be an increased data transfer rate between the microcontroller and the phone. This is important to Petronics since they plan on adding data analytics to the toy in the future, which makes accessing sensor readings and data in real time essential.

In order to develop the WIFI connection, Mousr functionality will be recreated in a new chip, the ESP32 [ESPR] suggested by Petronics, Figure 2. The reasoning behind this choice is that it is a very popular controller for real time systems, and there is a lot of documentation and code available. It also contains both Bluetooth and WIFI, which is necessary for the connection we aim for, for reasons that will be explained on this document. Moreover, ESP32 can be programmed using Arduino IDE [ARDU], which offers advantages such as simplified memory storage, WIFI and I2C [I2CI] protocol commands.

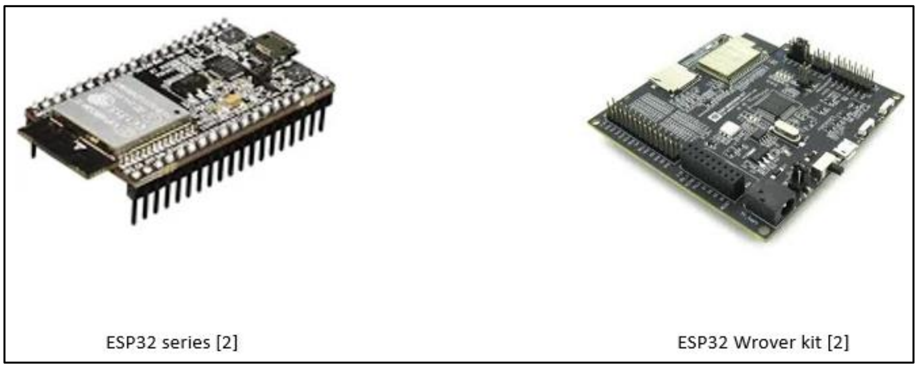


Fig. 2 ESP32 microcontroller

In addition to recreating current functionality, this project will add a Scheduled-play function, as a solution to the long-distance activation issue. It will allow the user to set a schedule through the app, which will be stored in ESP32 and then treated as device power on/power off intervals.

To summarize, there are three elements to this project. Fist, a PCB will be designed to connect all sensors to both microcontrollers, which must be soldered and debugged. Then, WIFI connection and functionality must be coded in Arduino and loaded into the microcontroller. Finally, a new Android app will be developed to incorporate all added capabilities.

2. Motivation

As stated in the introduction, the aim of this project is to add IoT (internet of things) capabilities to an existing product. Although this is done by client´s request, and it is not a personal choice, it is important to analyze the benefits and inconvenient of such a significant update. ESP32 has a dual WIFI and Bluetooth radio, that supports BT v4.2 and WIFI 802.11 b/g/n and 802.11 n.

Bluetooth v4 is not able to support IoT devices due to its reduced speed and bandwidth which is a good argument in favor of implementing Blufi, to make up for this issues. It has a speed of 1Mbps and a bandwidth of 2.1 Mpbs, according to the "ESP32 series" datasheet. Moreover, it has a 10 meter range indoors, and can only transfer as much data as can be stored in 20 bytes (originally 31, but 30% are used for protocol and setup).

WIFI 802.11 b/g/n can support frequencies of 2.4 GHz (b/g) and 5GHz (n), and a bandwidth of up to 54Mpbs. Most routers work in the 2.4 GHz, which has a much broader range, but the newest models are switching to the 5HGz frequency, which is faster as well as being less crammed by devices. Regarding the b/g/n modalities, 'b' is obsolete due to having a very small bandwidth, 'g' is the most common, and has a maximum transfer rate of 54Mbps and 'n' is the latest development and achieves a transfer rate of 300 Mpbs [TURBO].

It must be noted that all this figures are theoretical and may significantly vary in real life, due to many exposure factors. They are summarized in Tables 1 and 2:

Table 1. WIFI Specifications by Standard

	802.11 b	**802.11 g**	**802.11 n**
Frequency (GHz)	2.4	2.4	5
Transfer Rate (Mpbs)	11	54	300
Indoor range (m)	35	38	70

Table 2. BT Specifications

	BT v4.2
Frequency (GHz)	2.4
Transfer Rate (Mpbs)	25
Indoor range (m)	10

When comparing the two sets of specs, it becomes clear that WIFI provides a much greater range and speed, while operating at the same frequency. It is a much better option when dealing with large volumes of data, and given that Petronics intends to incorporate data analytics and AI to its products, enabling WIFI will become essential.

However, there is a significant drawback: a shortened battery life. According to the datasheet, WIFI (in data receiving mode) consumes 190 mA when active, while BT only consumes 100mA. This means the battery will be depleted twice as fast while Mousr is active, which must be taken into consideration. The company will probably need to consider power saving measures, such as buying more efficient sensors or somehow fitting a bigger battery inside the device.

Additionally, the project will aim to improve the project by adding a scheduled play functionality. As explained before, this would solve the issue of remote activation, which is the most significant inconvenient of automitez cat toys. This will allow users to save battery by only activating the toy when the cat will be around to play with it, and to reduduce the need to be aware of the toy´s state.

This is only possible due to the WIFI functionality, which allows ESP32 to download the real date and time stamp from the WIFI network, through an NTP (Network Time Protocol) request.

3. Project objectives

As stated in the Introduction, there are three main components to this project: the PCB design, the ESP32 code and integration and the android app. I will now describe in detail the requirements of each block, as well as how they were accomplished, and to what degree of success.

The PCB has been design to contain two separate circuits: power and data. The power circuit, as to ESP32. Two voltage regulators were needed to answer the different voltage requirements throughout the circuit.

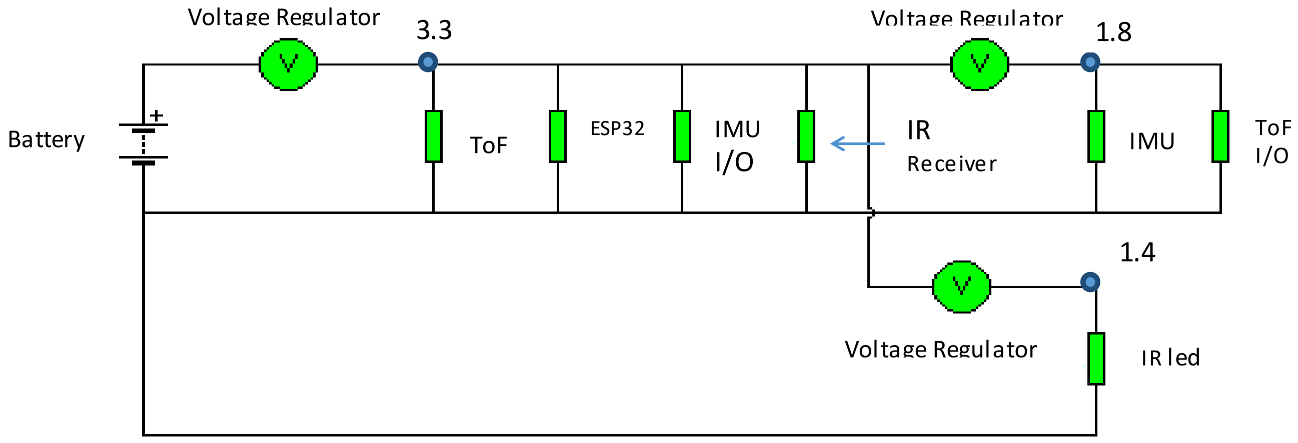


Fig.3 Power Circuit Schematics

As seen in Figure 3, the battery supplies 4.1V, which are then converted to 3.3V and 1.4V, as specified in the individual sensor datasheets.

On the other hand, the data circuit will be used to send digital highs and lows to the microcontroller, and it requires several resistors and capacitors to clear noise. The main challenge within this second circuit will be implementing I2C protocol, which is a serial communication protocol between a master and a slave. It defines two essential digital signals, SDA or Serial Data and SCL or Serial Clock, shown in Figure 4. They must follow very specific patterns, which are also very sensitive to timing. For I2C to Work properly, SDA and SCL PCB vias lengths must be almost identical, to avoid delays. This method is very common in commercialized sensors which process complex data or which have many different working condition settings.

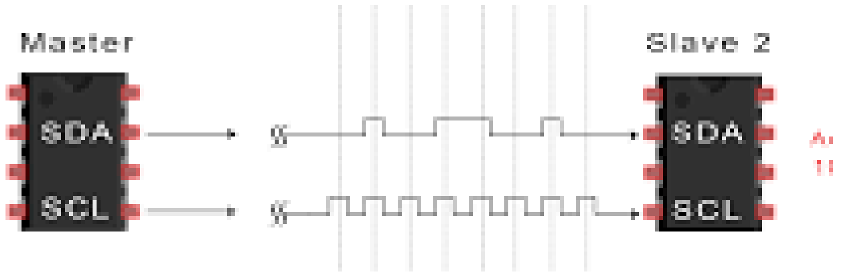


Fig.4 I2C schematic

The sensors to be used are those currently on Mousr, with the addition of an RGB led to display the state of the battery. They are: IR led and receptor, TOF (Time of Flight) and IMU (Inertial Measurement Unit), pushbutton and RGB LED, as well as the three voltage regulators. All this elements are structured as shown in Figure 5, the block diagram. Both IMU and TOF utilize I2C protocol.

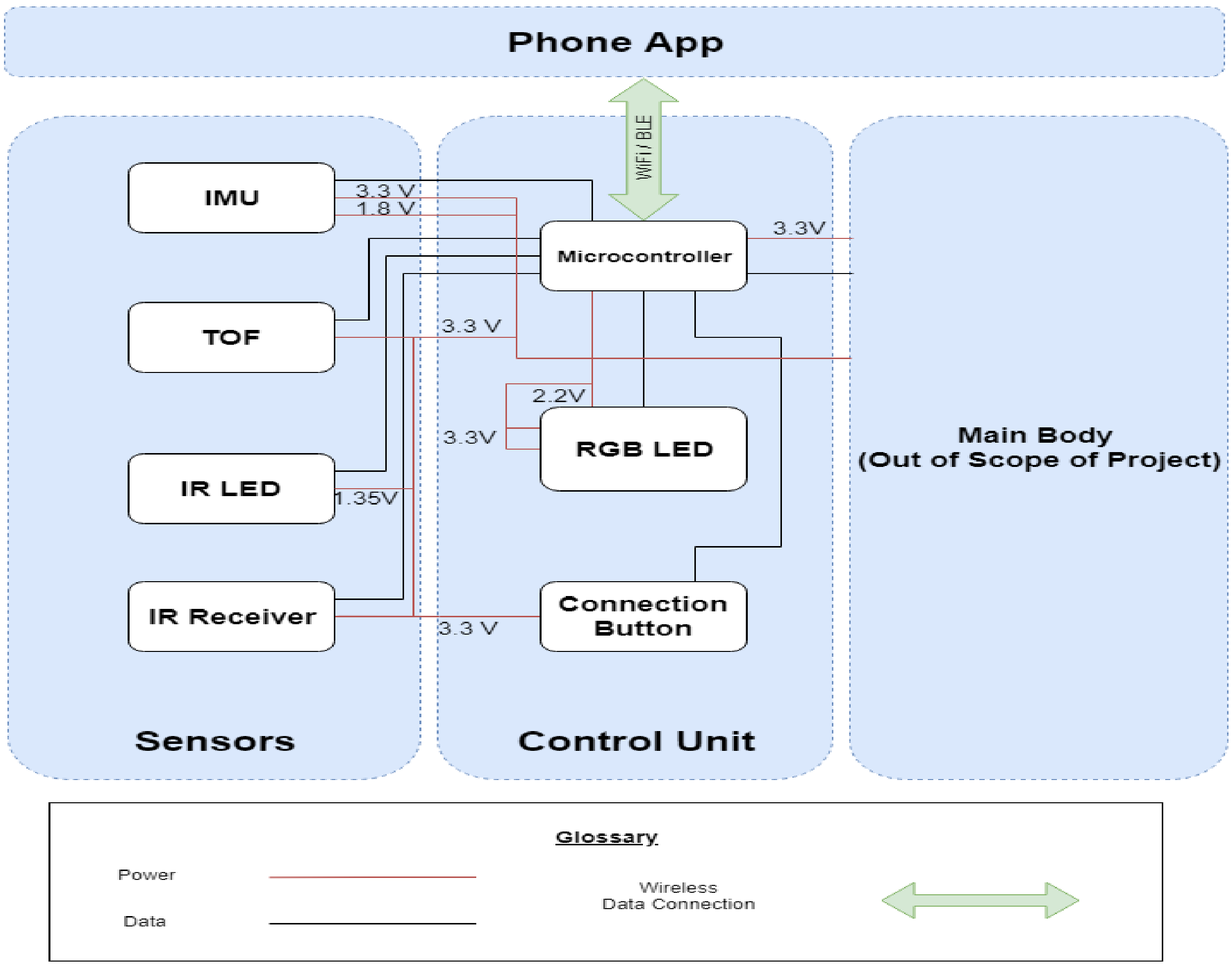


Fig.5 Block Diagram

In Petronics' Mousr, these sensors are placed as seen in Figure 6. However, we will not modify or evaluate this disposition in the scope of this project, since only a prototype will be developed.

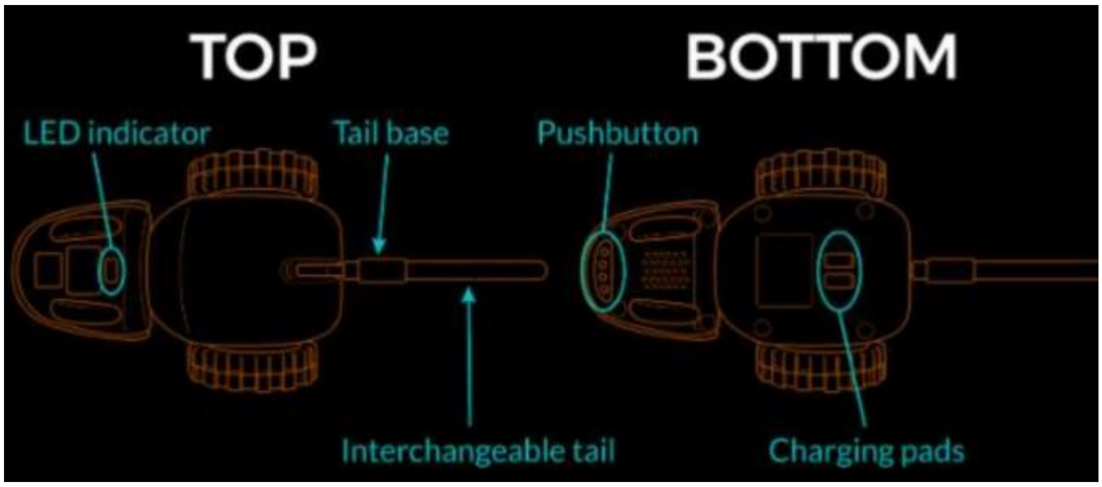


Fig. 6 Mousr by Petronics, component placement

Regarding the sensor´s purpose, the IMU and TOF´s function is possibly the most obvious. The former is meant to indicate whether Mousr is upside down or held in vertical position, in order to stop or invert the engine´s movements and avoid hurting the cat. The TOF will simply detect any inert obstacles and alert the micro of their approximate location. Similarly, the IR transceiver is used to generate a cat presence detector, which can be used to avoid the animal as well as to shut down Mousr when there are no cats nearby. Lastly, the pushbutton will allow the user to manually turn on and off the device and to trigger WIFI pairing, and the RGB led will act as a display of the power and connection states.

The resulting digital PCB is shown in Figures 7 and 8. In addition, Figure 7 shows board dimensions in millimeters.

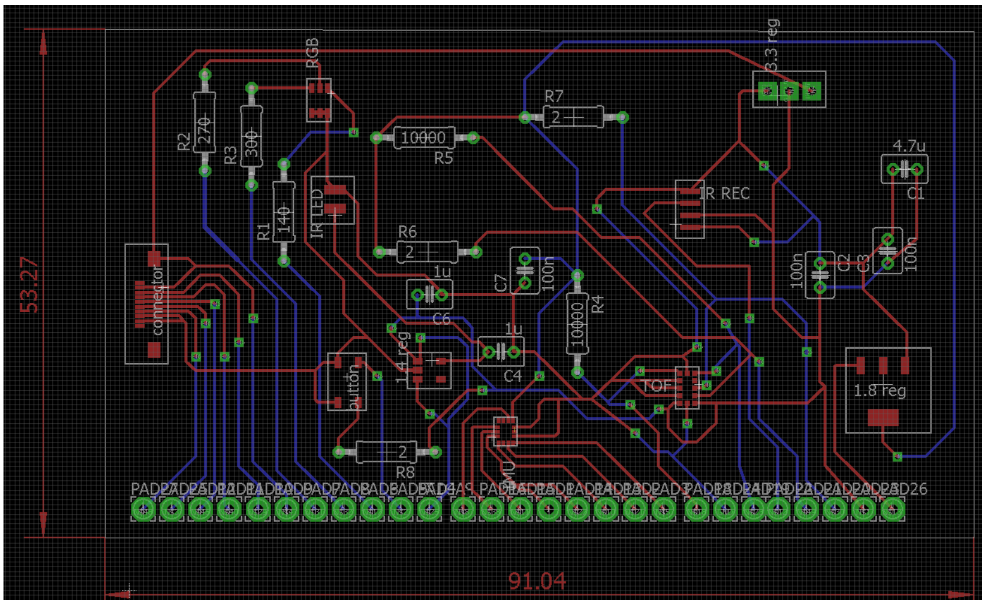


Fig. 7 PCB cad view

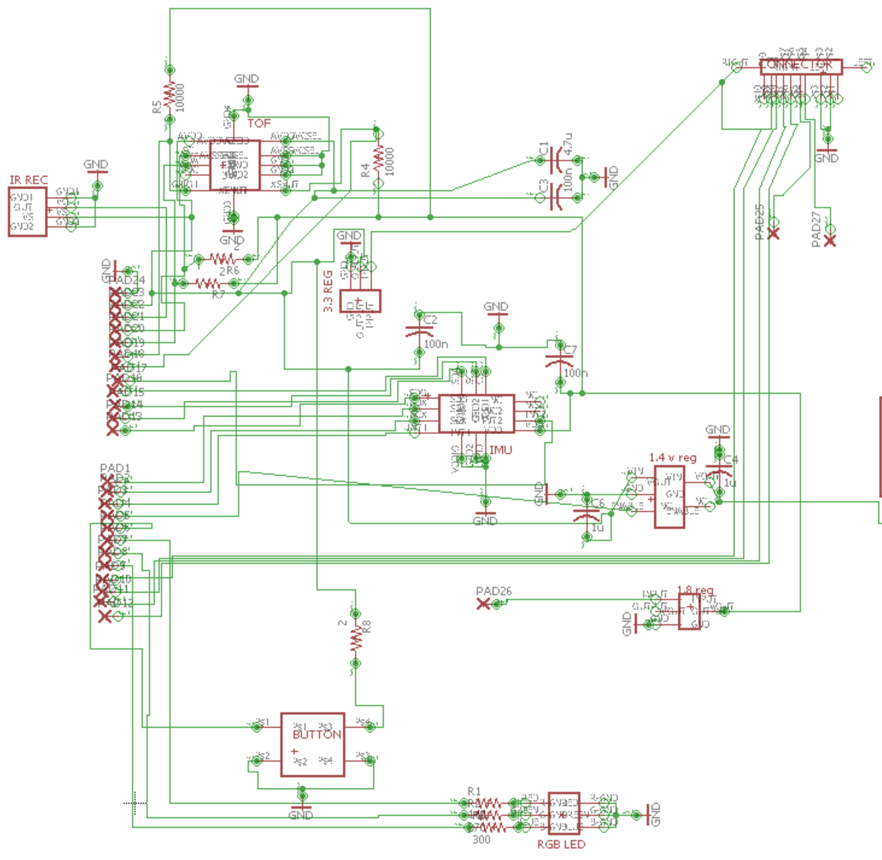


Fig.8 PCB schematic

We will now discuss how each of this sensors has been implemented in more depth.

3.1 Push Button:

To ensure the correct functionality of the pushbutton with the microcontroller, an interrupt is set to detect the falling edge in the GPIO (General Purpose In Out pin). Additionally, it is essential to place a pull-up resistor for the pushbutton, to avoid short cutting the microcontroller. The circuit implemented is shown in Figure 9.

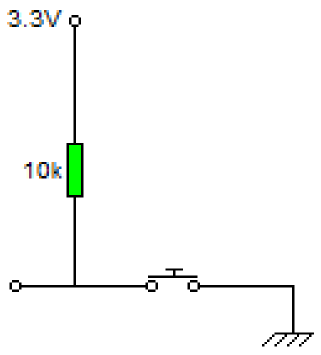


Fig.9 Pushbutton Circuit

As we can see, each time we press the button, the corresponding input pin in ESP32 is set to low voltage, thus activating the interrupt. If it is not pressed, the pin remains high and nothing changes in the microcontroller.

3.2 RGB LED

Regarding the RGB LED, straightforward calculations were computed to obtain the required resistor values for each color port. First, we obtained each diode's internal resistance by looking at the nominal values provided in the corresponding datasheet [6]:

$$R_{int} = \frac{V_{nom}}{I_{typical}} \qquad 1.1$$

Since we knew the LED would be connected to a GPIO from the microcontroller, which provides us a signal of 3.3V, we wrote a simple voltage divider:

$$V_{LED} = \frac{R_{int}}{R_{int}+R_x} \qquad 1.2$$

Resulting values and corresponding parameters are displayed in Table 3:

Table 3. LED Resistances

LED COLOR	Rint (Ω)	VLED (v)	Rx (Ω)
Red	140	2.1	140
Green	270	2.7	270
Blue	300	3	300

3.3 TOF

When choosing the Time of Flight's connections, we follow the schematic recommended in the datasheet by the manufacturer, which is attached to this document, as Figure 10.

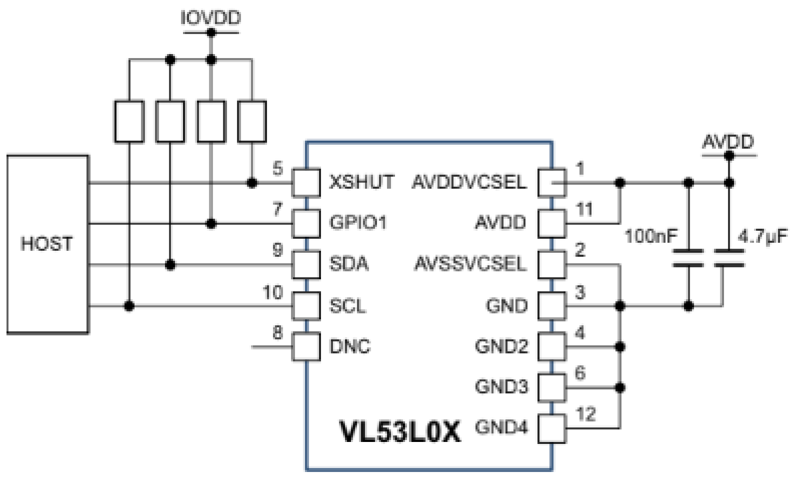


Fig.10 TOF Circuit

The resistors connected to IOVDD signal are I2C pull-up resistors [TEXA], which are meant as a protection against shortcuts. Their required minimum and maximum values are given by the following equations:

$$R_{min} = \frac{V_{cc} - V_{ol}}{I_{ol}} \qquad 2.1$$

$$R_{max} = \frac{t_r}{0.8473 * C_b} \qquad 2.2$$

Where: - Vcc is the sensors DC supply

- Vol is the max voltage that can be interpreted as a digital low

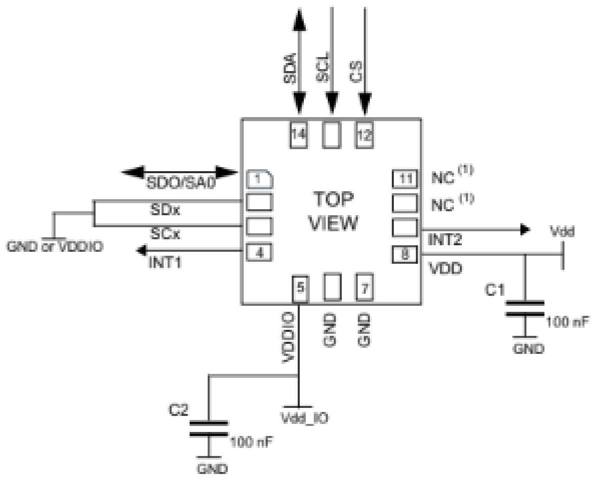
- Iol is the current which corresponds to this voltage.

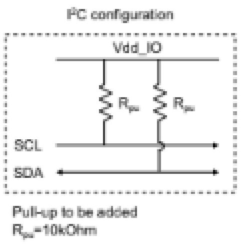
- tr is the maximum rise time

- Cb the capacitive load for each bus line.

A lower resistor value would result in smaller power consumption, but also increased transfer rate due to reduced RC delay. However, since there aren´t any specific power consumption or speed specs, intermediate values will be adopted.

3.4 IMU





For the IMU, the manufacturer provides the values of the pull-up resistances in the datasheet for the I2C protocol, as well as a recommended implementation schematic. Both were adopted in the PCB design. The connections are shown below on Figure 11:

Fig.11 IMU Circuit

3.5 IR LED and IR Receiver

The IR LED is used to provide a reference value for the IR Receiver, which is designed to detect variations in heat intensity, rather than its presence or absence. The only PCB requirement is that the emitter and the receiver have to be separated by a certain distance, of about 10 inches, and there cannot be any object in the space between them. This was implemented as shown in Figure 12:

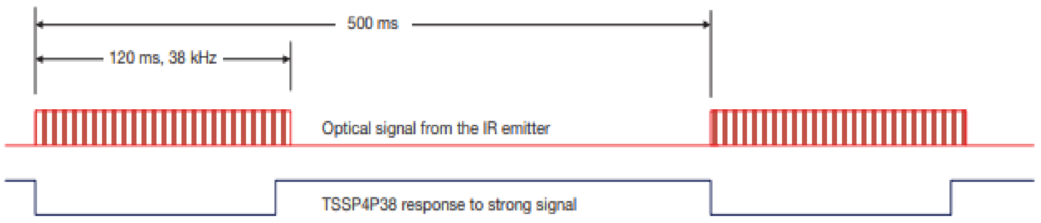

500 ms
120 ms, 38 kHz
Optical signal from the IR emitter
TSSP4P38 response to strong signal

Fig.12 Signal Supply for IR LED and IR Receiver Response

As previously mentioned, the IR circuit is used as a presence detector, using the manufacturer´s article “Vishay’s TSSP-AGC P Sensor Series for Proximity Sensing” as a reference on the topic [VISH]. To summarize, the IR led must generate the light pulse specified in Figure 12 at a 38 kHz frequency, for the receiver to react as expected.

This signal is generated through ESP32, by creating a PWM (Pulse-Width Modulation) wave and switching between two duty cycles to generate the DC or pulse signal. Timing will be respected by adding delays.

The PCB was created using EAGLE [AUTO] cad software. First, each sensor was introduced by hand, specifying the pads and outline dimensions. This process, as shown in Figures 13, 14 and 15, involves designing a "package", a "symbol" and a "device" to join them. The "package" contains physical dimensions, which will appear in the PCB as pads and silk screen markers. The "symbol" consists of component schematic, specifying number of pins and the "device" assigns a schematic pin to each physical pin location. It was then audited and printed through PCBWay [PCBW].

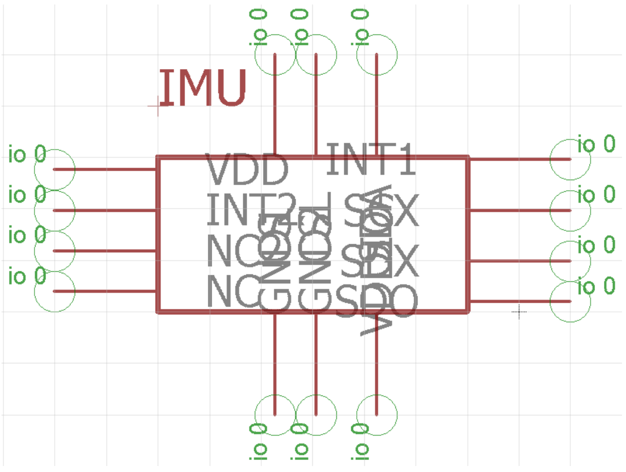


Fig.13 IMU symbol

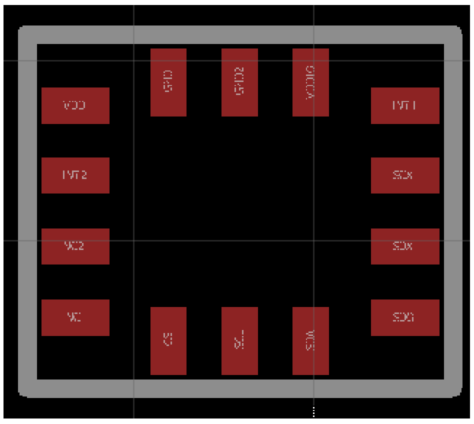
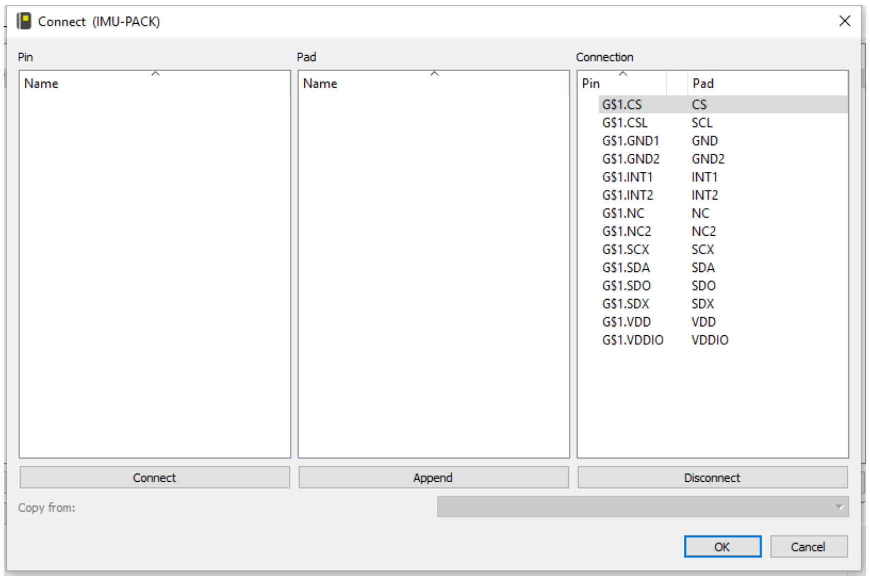


Fig.14 IMU package



Connect (IMU-PACK)

Pin	Pad
G$1.CS	CS
G$1.CSL	SCL
G$1.GND1	GND
G$1.GND2	GND2
G$1.INT1	INT1
G$1.INT2	INT2
G$1.NC	NC
G$1.NC2	NC2
G$1.SCX	SCX
G$1.SDA	SDA
G$1.SDO	SDO
G$1.SDX	SDX
G$1.VDD	VDD
G$1.VDDIO	VDDIO

Fig.15 IMU device

In order to print a board, Gerber files [SEED] must be available to send to the manufacturing company. They are images of each relevant layer of the board, which cannot be modified. In our case, this layers are both copper layers (top and bottom), both silk layers (t-place and b-place) as well as the through hole layer. In EAGLE, Gerber files are extracted using the “CAM processor”, which provides a way to pick the information to display in the Gerber images. “CAM” is a reference to the image file specific to Casio’s [FILE] digital cameras. The processor requires a “CAM” file to use as a reference, which can be made from scratch or downloaded from a provider. For this PCB, Sparkfun Electronic´s free-download CAM file [SPAR] was used.

Soldering work was mostly done with a regular soldering iron, although some of the smaller components required using soldering paste and an oven. Asides from soldering the sensors, each through hole had to be filled with conducting material in order for the circuit to work. The resulting board is shown in Figure 16.

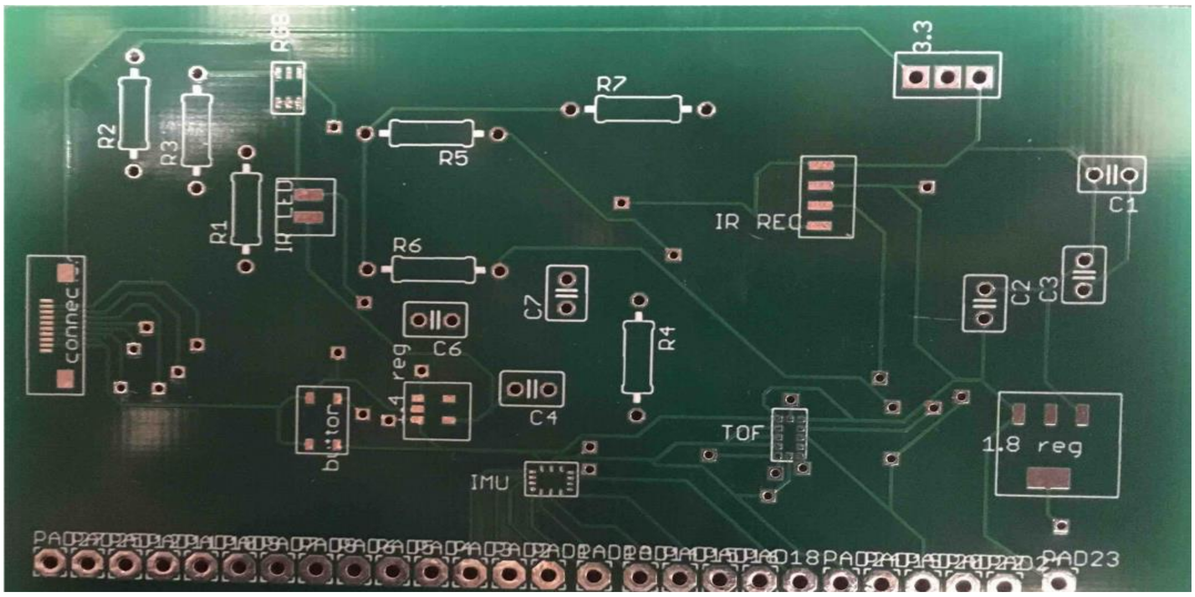


Fig.16 Printed PCB

The second element of this project, as stated at the beginning of this section, is the ESP32 code which is implemented in Arduino. To set up ESP32 in Arduino IDE, it is necessary to download the .json (JavaScript Object Notation) package from the Espressif website, which includes libraries and board settings.

Several Espressif boards will then appear on the Arduino "board manager". The "Wrover kit" must be selected. This variant of the ESP32 series has been chosen for the project due to its greater number of GPIO pins, and to the availability of two I2C bus ports.

The main issue that raised when compiling the code was the size of the program. When completed, took over 100% of ESP32´s processing capacity, which isn't viable. Modifying the partition sizes from regular app to large app with reduced OTA (Over The Air updates), which is easy to do in Arduino, solved the problem.

The first functionality to be added was the interrupt for the pushbutton. The button can be used to turn ESP32 (and therefore Mousr) on and off, as well as to manually activate Blufi [ESPR] pairing. The RGB led was simultaneously configured to act as an indicator of the power and connection state. The implemented logic is explained in Figure 17, where the background color of each state correspond to the assigned led color.

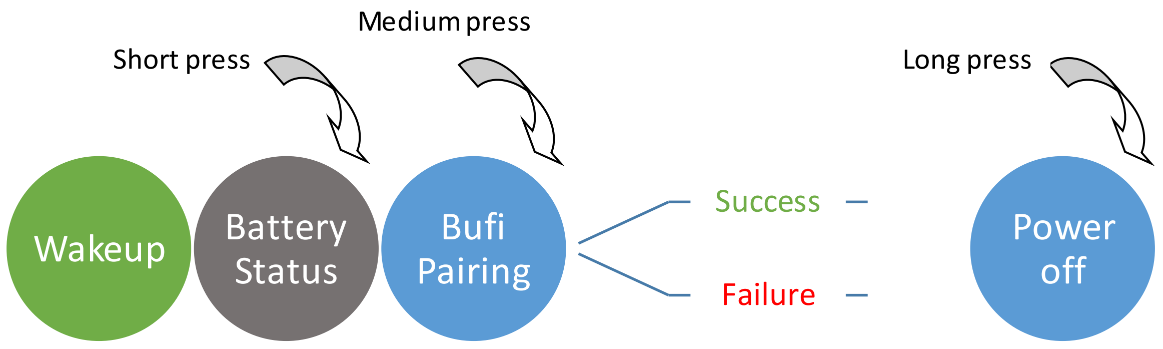


Fig.17 Pushbutton activity flowchart

ach
time the button is pressed, and depending on how long it remains pressed, the state machine will jump to the corresponding state. Corresponding pseudo code will be included shortly. In order to be able to wake up ESP32 with an interrupt, it can never be completely powered off. Therefore, light sleep mode will be used which means, according to the ESP32 data sheet that: "The CPU is paused. The RTC memory and RTC peripherals, as well as the ULP co-processor are running. Any wake-up events (MAC, host, RTC timer, or external interrupts) will wake up the chip". However, the fact that ESP32 will never be powered off or enter deep sleep will lead to a shortened battery life. An estimation is included below, using the data gathered from the "ESP32 series" datasheet and shown in Table 4.

Table 4. Esp32 Power Consumption

Power state	WIFI | receiving (mA)	BT (mA)
Active	190	100
Modem sleep	45	■
Light sleep	0.8	

When WIFI is connected, the chip will alternate between active and modem sleep modes. We will assume the worst case scenario in complexity terms, which means assuming that Mousr is not connected to the WIFI network and needs to go through the Blufi steps. We will allow 5 minutes for the user to find and input WIFI credentials, during which Bluetooth would be active. After this period, WIFI will remain connected while Mousr is powered on. We will also assume that in a 24h period Mousr is active for 4 hours. Since the datasheet currents are measured at 3.3 volts, we use this figure in our calculations.

Then:

$$Iaverage = \frac{1}{24h} * \left[(20h * 0.8) + \left(\frac{5min}{60min} * 100\right) + \left(\frac{235min}{60min} * 190\right)\right] = 32 \text{ mA} \qquad 3.1$$

$$E = P * t = Iaverage * V * t = 32\text{mA}*3.3\text{V}*24\text{h}*3600\text{s/h} = 9123.84 \text{ J} \qquad 3.2$$

To put this value in context, we can compare it to the energy stored in an AA commercial battery cell, which is approximately 12960 J. Our device can therefore be powered for almost 1.5 days with an AA cell, which is about the size of Mousr´s internal battery.

Afterwards, the battery power display was coded. This functionality is meant to inform the user of the battery state through the led color. Three additional states were created, as shown in Figure 18, where the colors correspond once more to the selected led light color.

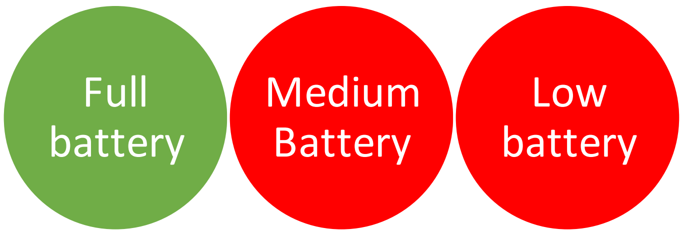


Fig.18 Led battery state display flowchart

This circuit was debugged by connecting the ESP32 GPIO pin to different points of a voltage resistor circuit, which was set in a breadbox. The voltage intervals that define each state are shown in Table 5, which is designed for a maximum battery supply of 4.1 volts.

Table 5. Battery states voltage range

State	Lower Voltage (V)	Higher Voltage (V)
Full	2.20	4.10
Medium	1.10	2.20
Low	0	1.10

Regarding sensor control, the microcontroller must implement I2C protocols for the TOF and IMU and correctly receive and interpret the data. This is accomplished through drivers that set the sensor settings and send the correct SDA and SCL configuration, as seen in Figure 19.

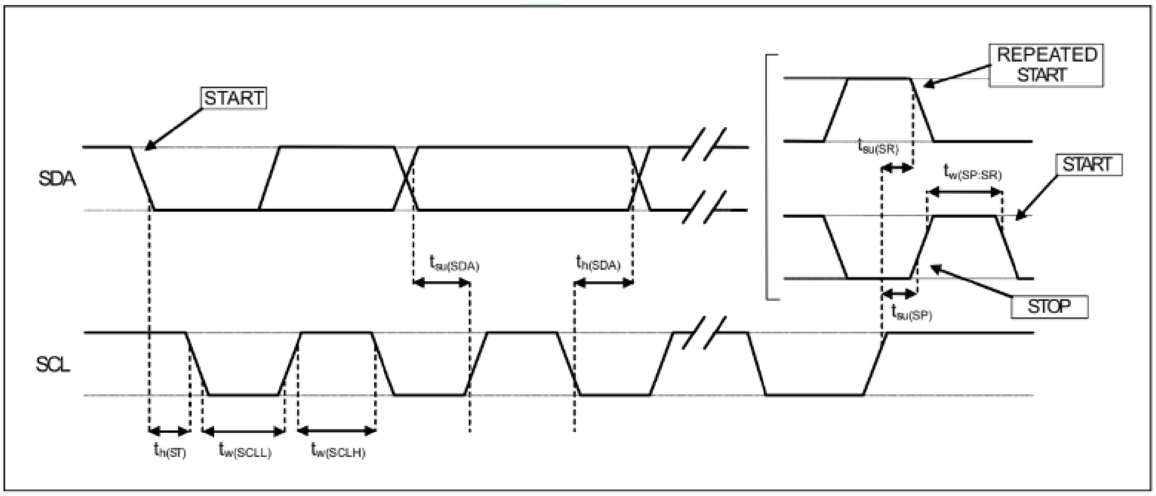


Fig.19 IMU I2C pattern

Once ESP32 has access to the data, it is able to send coherent commands to the engines. Wifi Mousr´s behavior, which is very simple as it is just meant to show proof of concept, is shown in the following chart, Figure 20.

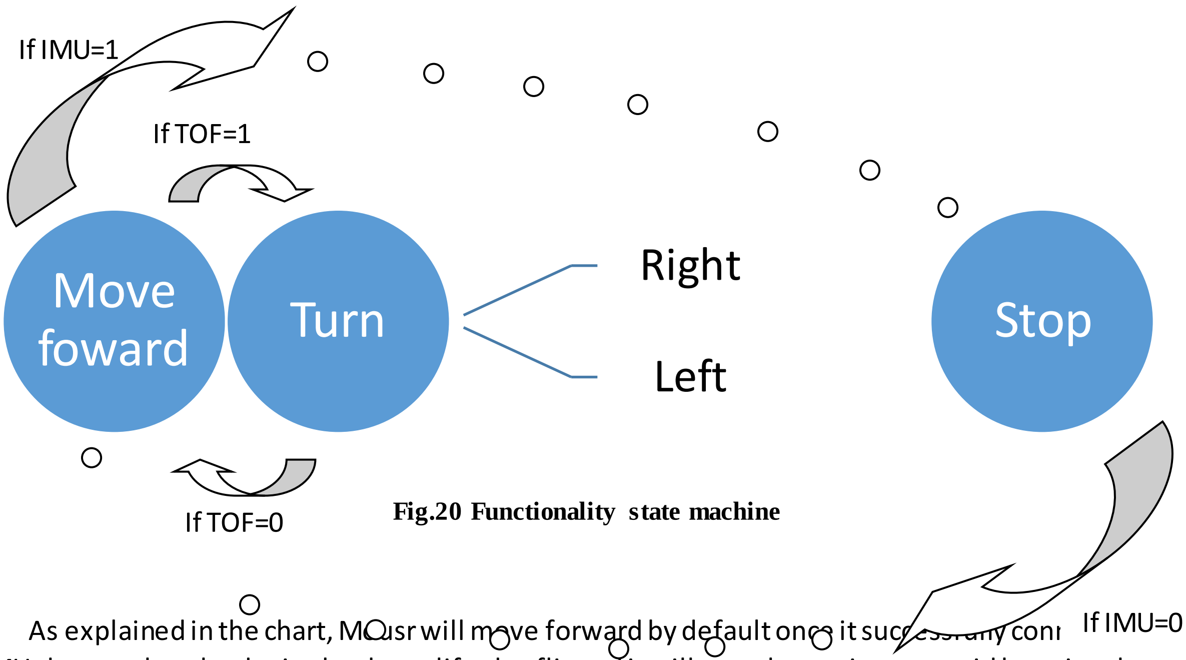
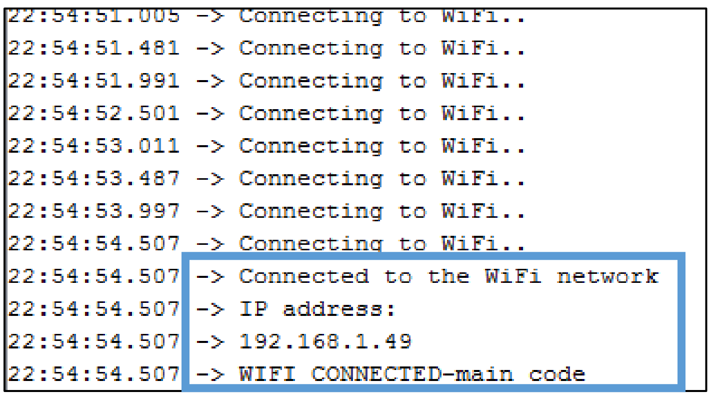


Fig.20 Functionality state machine

As explained in the chart, Mousr will move forward by default once it successfully conn the IMU detects that the device has been lifted or flipped it will stop the engine, to avoid harming the cat. If instead the TOF detects an obstacle, Mousr will turn. The direction of the turn will alternate between left and right. Additionally, the chip must generate a PWM signal for the IR circuit, which was already explained on this document.

Finally, ESP32 must implement the Blufi protocol, as we explained in the introduction. The implementation will be based on existing GitHub [GITH] repositories and Espressif's [ESPR] related publications. All copyrights and intellectual property rights were checked before reusing any code. Blufi is a method which allows users to connect the microcontroller to a router, removing the need for the WIFI credentials (SSID and password) to be typed into the chip´s code. It uses both BLE (Bluetooth Low Energy) [ESPR] and WIFI radios. Once the chip is successfully connected to the router it is assigned an IP, which may be used to exchange data. The Arduino monitor is programmed to show updates throughout this process, as seen in Figure 21:

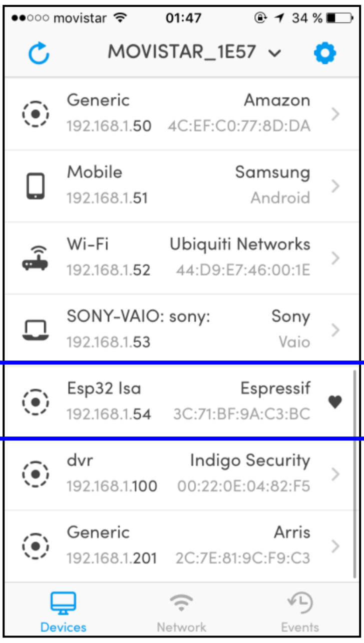


```
22:54:51.005 -> Connecting to WiFi..
22:54:51.481 -> Connecting to WiFi..
22:54:51.991 -> Connecting to WiFi..
22:54:52.501 -> Connecting to WiFi..
22:54:53.011 -> Connecting to WiFi..
22:54:53.487 -> Connecting to WiFi..
22:54:53.997 -> Connecting to WiFi..
22:54:54.507 -> Connecting to WiFi..
22:54:54.507 -> Connected to the WiFi network
22:54:54.507 -> IP address:
22:54:54.507 -> 192.168.1.49
22:54:54.507 -> WIFI CONNECTED-main code
```

Fig. 21 Arduino monitor during Blufi protocol

The protocol begins by activating BLE in ESP32, and “advertising” the device, which is basically broadcasting its availability to nearby Bluetooth receivers. Then, the user must select the device, and proceed to set the WIFI network and its respective password through an Android app, which will be explained in more detail later on. The app will send this text strings to ESP32 via BLE, and the micro will then proceed to connect to the WIFI network. Once this process is finalized, the IP can be retrieved with a simple Arduino command and sent back to the app via Bluetooth. Communication is then established.

The most straightforward method to check whether ESP32 is bonded to the router is to use an app like “Fing” [FING]. It shows a list of all the devices which are connected to the router, in real-time. In Figure 22 we can spot a Espressif device within this list, which proves the connection.


●●○○○ movistar 01:47 34 %

MOVISTAR_1E57

Generic 192.168.1.50	Amazon 4C:EF:C0:77:8D:DA
Mobile 192.168.1.51	Samsung Android
Wi-Fi 192.168.1.52	Ubiquiti Networks 44:D9:E7:46:00:1E
SONY-VAIO: sony: 192.168.1.53	Sony Vaio
Esp32 Isa 192.168.1.54	Espressif 3C:71:BF:9A:C3:BC
dvr 192.168.1.100	Indigo Security 00:22:0E:04:82:F5
Generic 192.168.1.201	Arris 2C:7E:81:9C:F9:C3

Fig.22 Fing app connected devices list

At this point, any WIFI device (laptop, smartphone) that is connected to the same network can access the URL with a web browser, which has the format http://IP/info, where IP is a set of numbers and Info is text. As an alternative, https:// may be used for an encrypted and safe transfer. Regardless, ESP32 will detect that a client connected to the IP and receive any text information they input. ESP32 will also be notified when the client severs the connection, and it will proceed to free the server for its next use. The Arduino monitor updates are shown in Figure 23.

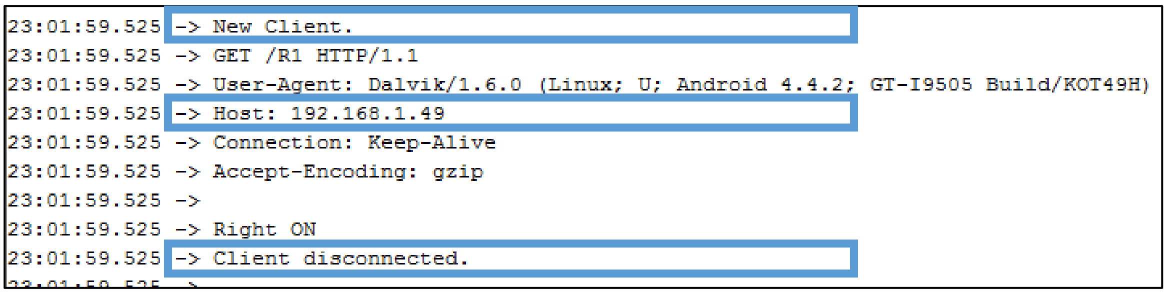

```
23:01:59.525 -> New Client.
23:01:59.525 -> GET /R1 HTTP/1.1
23:01:59.525 -> User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.4.2; GT-I9505 Build/KOT49H)
23:01:59.525 -> Host: 192.168.1.49
23:01:59.525 -> Connection: Keep-Alive
23:01:59.525 -> Accept-Encoding: gzip
23:01:59.525 ->
23:01:59.525 -> Right ON
23:01:59.525 -> Client disconnected.
23:01:59.525 ->
```

Fig. 23 Arduino monitor during client connection to server

All these functionalities are implemented inside a while (true) loop, which is the main section of the code and runs continuously. It relies on functions defined outside as well as the Arduino setup () function, which runs once and sets all the server parameters. Pseudo code for this while (true) loop is included on the next page.

```
if ESP32 is connected to WIFI network:
        if a client connects to the IP:
                read and sort incoming data
                led set to red and no blink
else:
led set to red and blink

if (button is pressed): //GPIO 13 high to low
        get current time
        wait until -> button is released
        get new time

        if (new time-old time) is between 10^5 and 10^6:
        read and display battery status mode
        led set according to battery state, no blink

        else if (new time-old time) is between 10^6 and 2.5*10^6:
        pairing mode (BLUFI)
        led set to blue and blink

        else if (new time-old time) > 2.5 * 10^6:
        enable light sleep wakeup by button
        enter light sleep mode
```

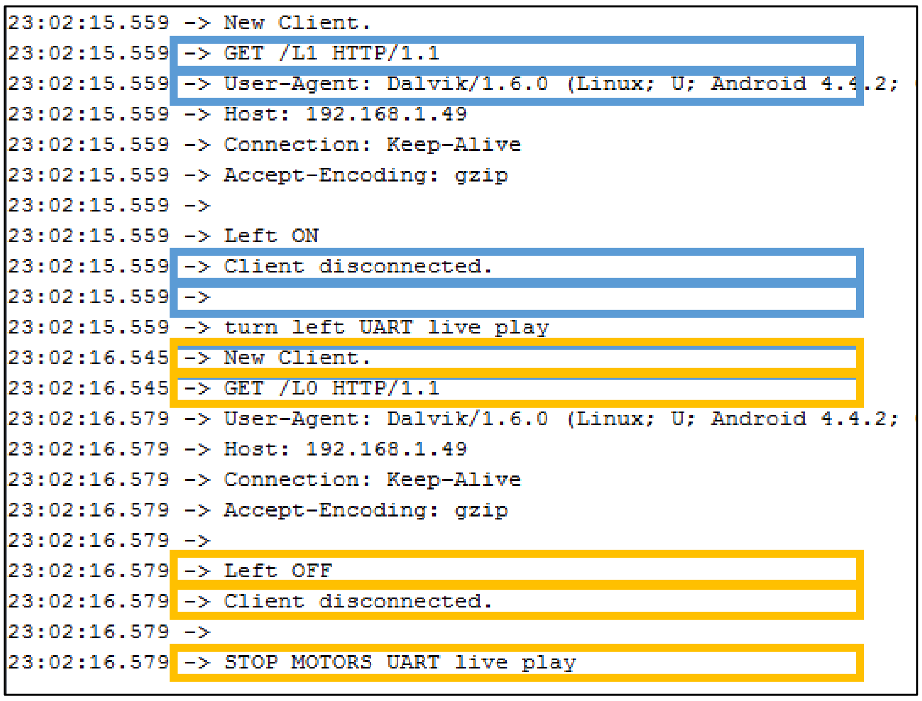
Therefore, a long button press will turn off the device at any time, while a short one will set Mousr in battery status display and a medium one will activate pairing mode.

In this project, an Android app will be used to send all data. This data can be classified into two structures: engine instructions and play times. Engine instructions, which are used to prompt Mousr's real time movements, are laid out in Table 6:

Table 6. Engine instructions variable structure

Trigger	Action (as displayed in Arduino monitor)	Text sent to ESP32
Push Left Button	Left ON	L1
Release Left Button	Left OFF	L0
Push Right Button	Right ON	R1
Release Right Button	Right OFF	R0
Push Forwards Button	Forwards ON	F1
Release Forwards Button	Forwards OFF	F0
Push Backwards Button	Backwards ON	B1
Release Backwards Button	Backwards OFF	B0

Figure 24 contains an example of handling this data type:



```
23:02:15.559 -> New Client.
23:02:15.559 -> GET /L1 HTTP/1.1
23:02:15.559 -> User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.4.2;
23:02:15.559 -> Host: 192.168.1.49
23:02:15.559 -> Connection: Keep-Alive
23:02:15.559 -> Accept-Encoding: gzip
23:02:15.559 ->
23:02:15.559 -> Left ON
23:02:15.559 -> Client disconnected.
23:02:15.559 ->
23:02:15.559 -> turn left UART live play
23:02:16.545 -> New Client.
23:02:16.545 -> GET /L0 HTTP/1.1
23:02:16.579 -> User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.4.2;
23:02:16.579 -> Host: 192.168.1.49
23:02:16.579 -> Connection: Keep-Alive
23:02:16.579 -> Accept-Encoding: gzip
23:02:16.579 ->
23:02:16.579 -> Left OFF
23:02:16.579 -> Client disconnected.
23:02:16.579 ->
23:02:16.579 -> STOP MOTORS UART live play
```

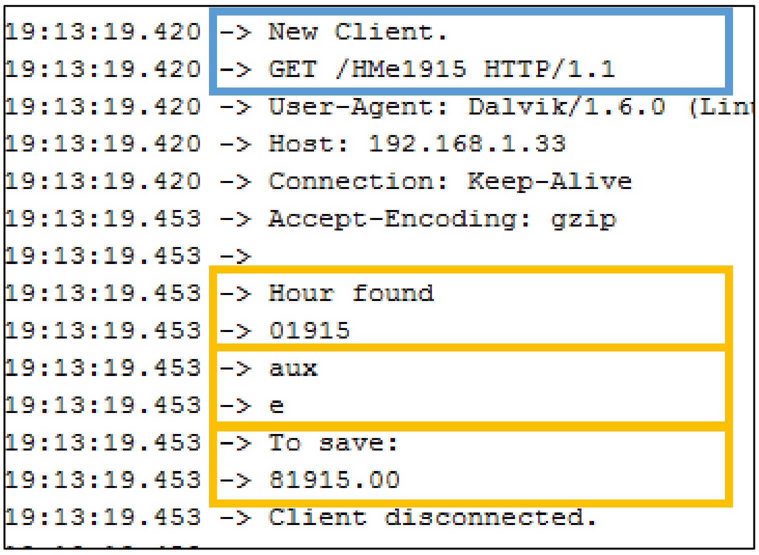
Fig. 24 Receiving motor commands

The remaining data class are times, which are used to set the scheduled-play. They just be received and stored by ESP32, and used as indicators of when to enter and leave light sleep mode. EEPROM (Electrically Erasable Programmable Read-Only Memory) memory will be used for storage, since it can be erased and reprogrammed and it saves data even when ESP32 is powered off. We must differentiate begin-play and end-play times, which is done by adding prefixes to the numbers as shown in Table 7.

Table 7. Play-time variable structure

Type	App to ESP32 format	ESP32 storage format
Begin-play	bHHMM	9HHMM
End-play	eHHMM	8HHMM

As seen in Table 7, times are in format HHMM or two-digit hour followed by two-digit minute; e.g. 1422 represents 2:22pm. Since times are sent to ESP32 as text strings, it is possible to simply add a lowercase "b" or "e" to indicate begin or end. However, when storing the values in EEPROM [RAND] memory, only float numbers may be used. The code must convert the string to text and add +9000 or +8000 to achieve the format specified. Numbers 8 and 9 are chosen randomly. This also serves the function to maintain the HHMM format, since a string containing the number "0809", for example, would be modified to "809" when converted to float. An example is attached in Figure 25.



```
19:13:19.420 -> New Client.
19:13:19.420 -> GET /HMe1915 HTTP/1.1
19:13:19.420 -> User-Agent: Dalvik/1.6.0 (Lin
19:13:19.420 -> Host: 192.168.1.33
19:13:19.420 -> Connection: Keep-Alive
19:13:19.453 -> Accept-Encoding: gzip
19:13:19.453 ->
19:13:19.453 -> Hour found
19:13:19.453 -> 01915
19:13:19.453 -> aux
19:13:19.453 -> e
19:13:19.453 -> To save:
19:13:19.453 -> 81915.00
19:13:19.453 -> Client disconnected.
```

Fig. 25 Monitor while sending time.

The Arduino code sorts the incoming data, through a switch loop. It looks for the letters R/L/F/B at the beginning of the text to detect engine instructions, and for the lowercase letters e and b to detect times. Storage is accessed through the Arduino commands EEPROM.get() [ARDU] and EEprom.put() [ARDU], with addresses generated by a simple for loop. The address range depends on the byte size of the float numbers inputted, and the maximum capacity is 512 bytes.

Finally, the engine instructions are sent to the engines through an UART connection, while the times are stored in the permanent memory and constantly checked against the real time. An UART (Universal Asynchronous Receiver-Transmitter) connection is a hardware device for serial data transfer through a wire, commonly used in ports. All communications consist on packages of 8 bits which begin with a digital low and end with a digital high. This pattern is shown in Figure 26. This ensures that the bits can be assembled into bytes in the sequence they were meant to, and retain their meaning. Transmission speed and format may be configured. The speed or transfer rate can be selected in the Arduino code in "baud", which is a unit that measures the number of signal changes (high/low) in one second. Some examples of standard rates are 9600, 57600 and 115200.

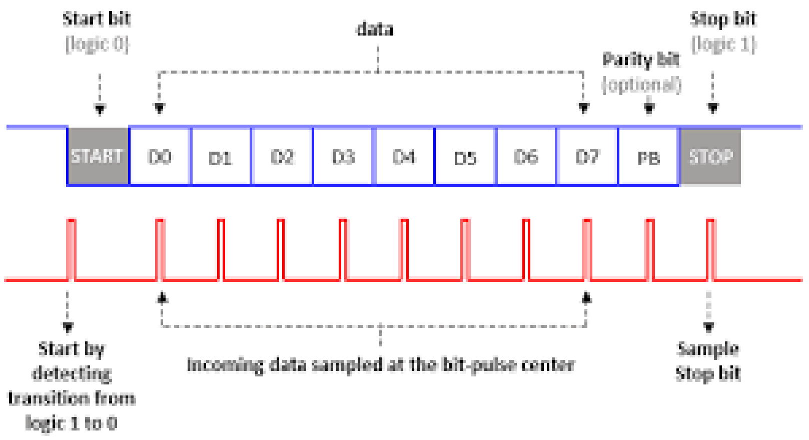


Fig. 26 UART protocol (solitontech.com)

The transmission is asynchronous, meaning there is no clock signal transferring between the two UART ports. Instead, the receiving UART will set its internal clock according to the incoming bits pulse. Some UARTs have a FIFO (first in first out), a buffer which can store a large number of bytes. This gives the system more time to react to the incoming data, and ensure no bits are lost.

In order to implement the UART communication with the engines, it was necessary to establish a connection between both engines. This was achieved via a 12 port connector, which included the following signals (Figure 27):

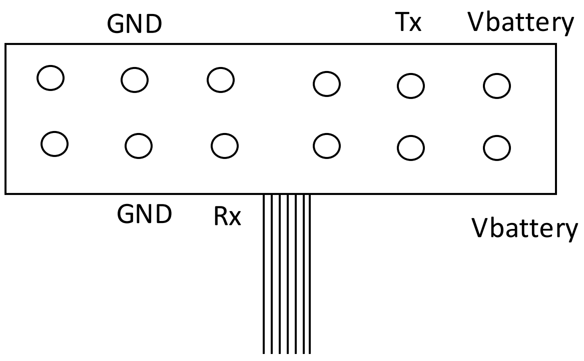


Fig. 27 Body microcontroller 12 port connector

Pins Tx and Rx are the primary and secondary UART signals, where Tx stands for transfer and Ty for receive. Since we want to input binary data into this secondary microcontroller, we will use Rx, by connecting it to the GPIO assigned the function of Tx in ESP32. GND is the electric ground and Vbattery the 4.1 voltage supply which may be used to power ESP32 and the sensors and other PCB components. The remaining pins are used to boot the body microcontroller before using it, through a protocol designed by Petronics that must remain confidential. The format used to send engine instructions is also confidential, but it must specify which engine the command applies to (right/left/tail motor) as well as movement direction and force.

The main issue with the implementation was that the ESP32 code was continuously sending the same instruction through the UART, therefore saturating the body microcontroller. This was fixed by sending each movement command only once, when a change was required.

Regarding the stored times, it is clear we must obtain real time in order to turn ESP32 on and off at the appropriate times. This is done using the existing WIFI connections, through an NTP or Network Time Protocol. The retrieved time and date comes in the format: "YYYY-MM-DDTHH:MM:SSZ" which stands for "*year-month-dayThour:minute:secondZ*" (eg: "2018-09-23T07:23:15Z"). We must convert this format to our time structure from Table 7: 9HHMM or 8HHMM, as well as from string to float.

First, we extract the time from the NTP string. The date won´t be used in this prototype, but would be a simple addition to the concept. We then slice the string in such a way that only the hour and minute digits remain, getting rid of the colon (“:”). An example of this is illustrated in Figure 28.

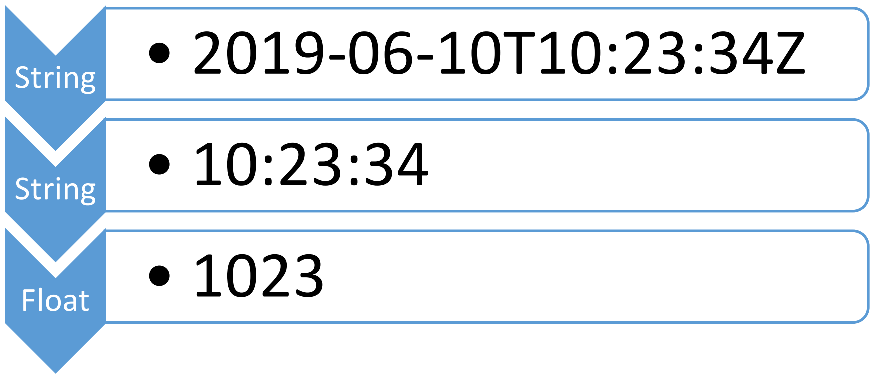


Figure 28: Time data structure modification example

Once this is handled, a loop will check the current time against all the times stored in EEPROM. If there is a match, and it is with an “end-play” time, ESP32 will enter light sleep mode. Before shutting down, however, an internal timer must be set for ESP32 to eventually turn back on. We must refer to the next stored value in ESP32, which will necessarily be a “begin-play” time. We then calculate the difference between current time and “begin-play” time in microseconds, and set a timer accordingly. This accomplishes the scheduled-play functionality. For further clarity, pseudo code for this process is attached in Figure 29:

```
While (true): ///main loop

        Connect to wifi
        NTP: get real time from network ->time 2
        time2-> format to HHMM  structure

        For (memory address in range (0, max)):
                Retrieve time stored in address -> time1

                if (time1==time2) and (time1 is an end play):
                        get time from address+1 -> StartTime
                        StartTime -> convert to microseconds
                        time2 -> convert to microseconds
                        inteval -> StartTime – time2
                        set timer to wake up in “interval”
                        enter light sleep mode
```

Figure 29: Scheduled play pseudo code

The last component of the project is the Android app, developed using the "App Inventor" [APPI] software for android apks, by MIT [MIT_]. It is a block coding platform with many IOT (Internet of Things) features. To download it to an Android Smartphone, App Inventor generates a QR code, which leads to a download server when scanned. When opened, this app will initialize to a main screen, shown below in Figure 30:

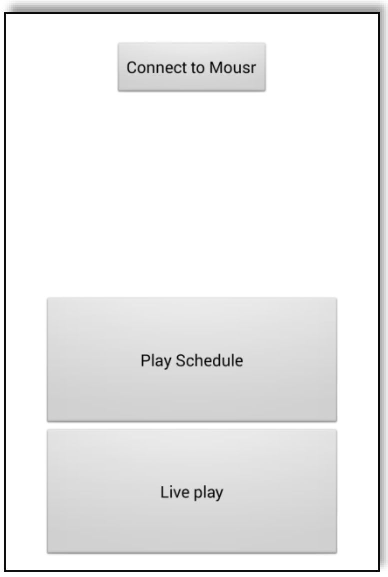
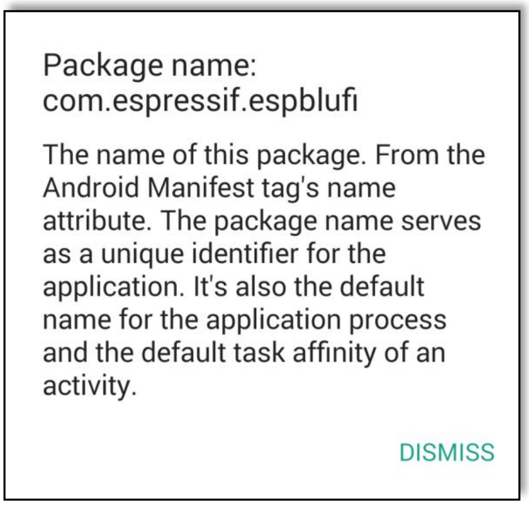


Fig. 30 App Main Screen

From there the user must select the option, "Connect to Mousr", which will cause the app to display the Espressif EspBlufi main screen. This is accomplished through an App Inventor functionality called "Activity Starter" [APPI], which is able to download from the Play Store and open any apk. Input parameters were "Activity Package", which is used to find the app either on the phone or the Play Store, and "Activity Class", which is unique to each of the screens of the app. These two parameters combined allow the programmer to select the app to download or open and the screen it opens to. They were retrieved from EspBlufi via an "Apk inspector" [PLAY] app. The resulting text strings for the "EspBlufi" app are shown in Figures 31 and 32.

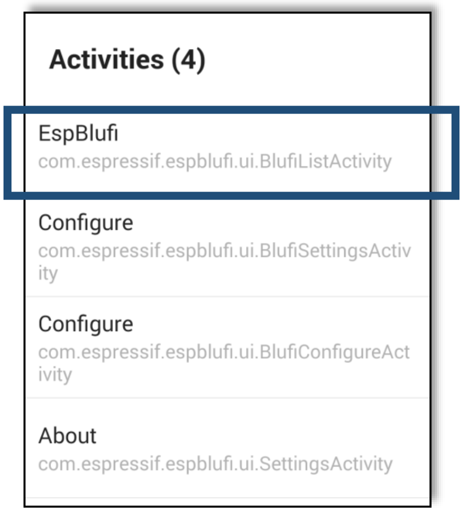


Package name:
com.espressif.espblufi

The name of this package. From the Android Manifest tag's name attribute. The package name serves as a unique identifier for the application. It's also the default name for the application process and the default task affinity of an activity.

DISMISS

Fig. 30 EspBlufi app "Activity Package"



Activities (4)

EspBlufi
com.espressif.espblufi.ui.BlufiListActivity

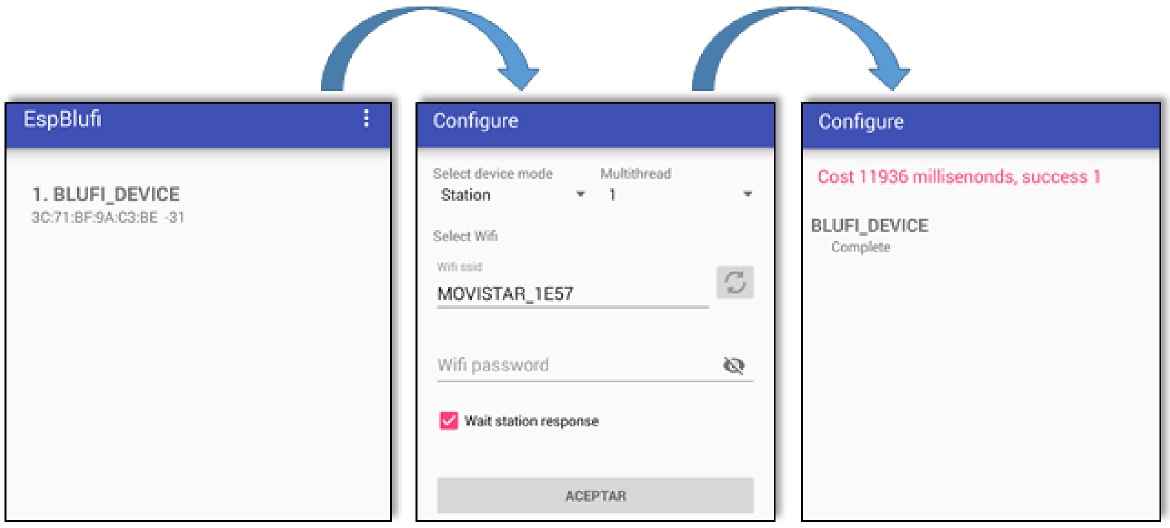
Configure
com.espressif.espblufi.ui.BlufiSettingsActivity

Configure
com.espressif.espblufi.ui.BlufiConfigureActivity

About
com.espressif.espblufi.ui.SettingsActivity

Fig. 31 EspBlufi app "Activity Class"

Once the user completes the Blufi connectivity screen flow, summarized in Figure 33, the two remaining buttons in the main screen are enabled. The user may therefore choose between accessing the auto play or scheduled play functionalities. If auto play is selected, the screen shown in Figure 34 will display on screen. It fountains four buttons which send data to ESP32 when they're either pressed or released, to begin or end play. Data will be sent via a HHTTP GET request, which is frequently used to make a request from a server. The best option to post in a server would be a HTTP POST request, which is safer due to the fact that the text is sent in a separate parameter, rather than appended in the URL like is the case with GET requests. However, for simplicity we will settle for the GET method, especially since we are sending very simple strings. In case WIFI Mousr ever needed to send large volumes of data, the app would have to be adapted to operate with POST requests.


EspBlufi

1. BLUFI_DEVICE
3C:71:BF:9A:C3:BE -31

Configure

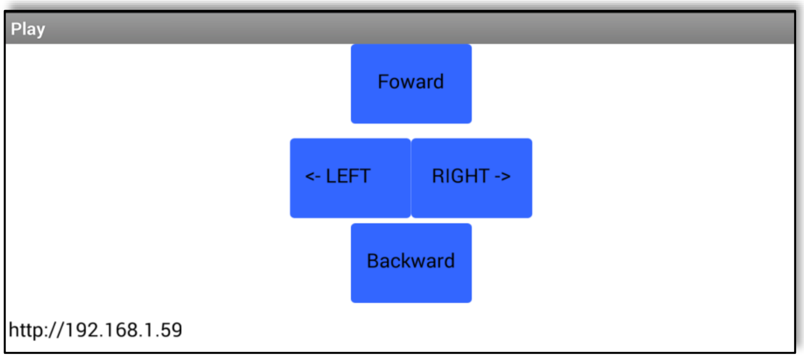
Select device mode: Station
Multithread: 1
Select Wifi
Wifi ssid: MOVISTAR_1E57
Wifi password
Wait station response
ACEPTAR

Configure

Cost 11936 millisenonds, success 1

BLUFI_DEVICE
Complete

Fig. 33 EspBlufi app flow



http://192.168.1.59

Fig. 34 App: Live Play Screen (controls)

In addition, the screen contains a text box with the local IP to which Mousr has been assigned, to confirm App reception.

Finally, the scheduled-play screen consists on a list of times, which are retrieved from the app's memory whenever the screen is accessed. It can be seen in Figure 35. To input a new time, the user should select the option "Add time", which will open the screen shown in Figure 36. Once the time is selected, it will be sent to ESP32 in the format specified earlier on this section.

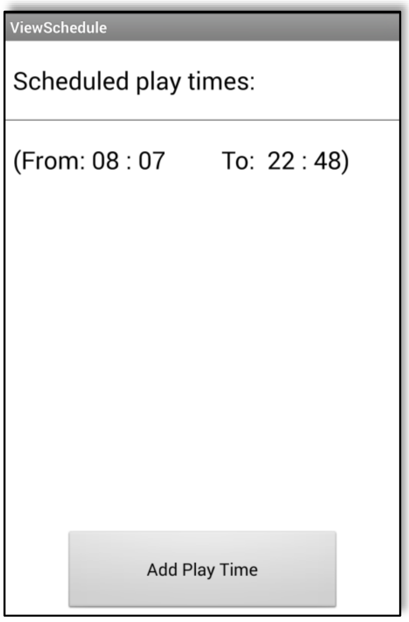


Fig. 35 Scheduled Times screen

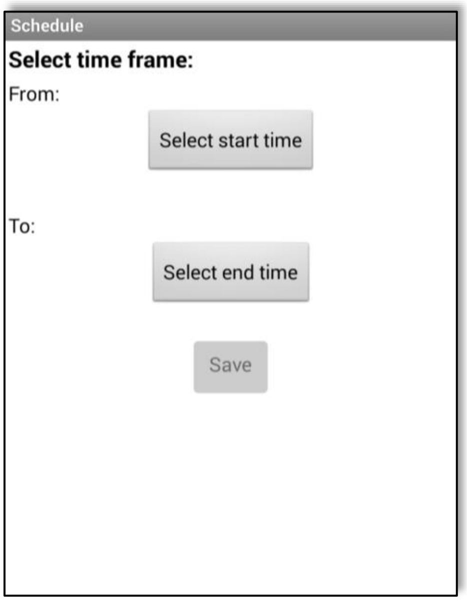


Fig. 36 Add time screen

<u>Isabel Ugedo, 12/06/2019</u>

Budget

Table 8. Bill of Materials

Part	Part Specification /Manufacturer	Supplier	Unit Cost (prototype)	Unit Cost (manufacturing)
ESP32Wrover Kit V4.1	*GC-ESP-WROVER-KIT/Espressif*	*GridConnect*	$40.00	$5.00
IMU	*LSM6DSLTR/STMicroelectronics*	*Mouser*	$3.98	$2.50
TOF	*VL53L0CXV0DH/1/STMicroelectronics*	*Mouser*	$4.5	$2.83
RGB LED	*CLY6D-FKC CK1N1D1BB7D3D3m/Cree*	*DigiKey*	$0.318	$0.29
IR Receiver	*TSSP77038TT/Vishay Semiconductors*	*Mouser*	$1.79	$0.82
IR LED	*VSMB3940X01-GS08/Vishay Semiconductors*	*DigiKey*	$0.73	$0.34
Pushbutton	*PTS810 SJG 250 SMTR LFS/C&K*	*DigiKey*	$0.31	$0.24
FCC Connector	*687112183722/Wurth Electronics*	*Mouser*	$1.83	$1.22
Voltage regulator 3.3V	*LM1117T-3.3/NOPB/Texas Instruments*	*Mouser*	$1.54	$0.89
Voltage regulator 1.8V	*LM1117MPX-1.8/NOPB/Texas Instruments*	*DigiKey*	$2.78	$0.68
Voltage regulator 1.4V	*NCP699SN14T1G/ON Semiconductors*	*Mouser*	$0.47	$0.14
PCB	-	*PCBWay*	$5.00	$0.50
Resistor	*Standard*	*UIUC*	$0.50	$0.01
Capacitor	*Standard*	*UIUC*	$0.70	$0.03
Female/Male Wire	*Standard*	*UIUC*	$0.05	

Table 9. Partial Cumulative A

PCB	Parts	Units	Added Cost (prototype)	Added Cost (manufacturing)
	Board	1	$5.00	$0.50
	IMU	1	$3.98	$2.50
	TOF	1	$4.5	$2.83
	RGB LED	1	$0.318	$0.29
	IR Receiver	1	$1.79	$0.82
	IR LED	1	$0.73	$0.34
	Pushbutton	1	$0.31	$0.24
	FCC Connector	1	$1.83	$1.22
	Voltage regulator 3.3V	1	$1.54	$0.89
	Voltage regulator 1.8V	1	$2.78	$0.68
	Voltage regulator 1.4V	1	$0.47	$0.14
	Resistor	8	$4	$0.08
	Capacitor	6	$4.2	$0.18
	Soldering materials (tin, flux, oven paste)	-	$26	$20
Total			$57.45	$30.71

Table 10. Partial Cumulative B

ESP32	Parts	Units	Added Cost (prototype)	Added Cost (manufacturing)
	ESP32Wrover Kit V4.1	1	$40.00	$5.00
	Female/Male Wires	27	$1.35	$1.35
	USB to Mini-B cable	1	$1.00	$0.50
Total			$42.35	$6.85

Table 11. General Budget

Element	Units	Added Cost (prototype)	Added Cost (manufacturing)
ESP32	1	$42.35	$6.85
PCB	1	$57.45	$30.71
Labor	$3500		
Shipping costs	$42.97		
Total		$3642.77	$3580.53

We obtained an approximation of cost of labor using the following equation:

$$Total\ Cost = \frac{(Avg\ Salary\ of\ BS\ in\ EE) + (Avg\ Salart\ of\ BS\ in\ CE)}{2} * \frac{1\ Year}{Labor\ Weeks * 45\ Hours} * (Total\ Time\ [h])$$

The average salary data was obtained from the ECE website of UIUC. Total time was estimated to be 110 hours per worker. The resulting value of the equation 4.1 is shown below.

$$Total\ Cost = \frac{\$67000 + \$84250}{2} * \frac{1\ Year}{52 * 45\ Hours} * (110)$$

$$Total\ Cost = \$3555.02$$

References

[TURBO] Turbofuture, "WIFI Questions: What does A/B/G/N Mean?". https://turbofuture.com/computers/Things-you-mustwant-to-know-about-WIFI. Last access: 17/06/2019 a las 10:35.

[PETR] Petronics, "Mousr". https://petronics.io/ . Last access: 09/06/2019 a las 17:20.

[PETR] Petronics/open source , "EspBlufi". https://play.google.com/store/apps/details?id=com.espressif.espblufi&hl=es. Last access: 03/06/2019 a las 18:50.

[RINC] Rincón Ingenieril, "Como funciona el Puerto serie y la UART". https://www.rinconingenieril.es/funciona-puerto-serie-la-uart/. Last access: 10/06/2019 a las 17:15.

[ESPR] Espressif , "Features and Specifications, The internet of things with ESP32. https://www.espressif.com/en/products/hardware/esp32/overview. Last access: 04/06/2019 a las 12:25.

[ARDU] Arduino, "Arduino Software". https://www.arduino.cc/en/Main/Software. Last access: 16/06/2019 a las 12:00.

[I2CI] I2C Info, "I2C Bus, Interface and Protocol". https://i2c.info/. Last access: 11/05/2019 a las 10:40.

[TEXA] Texas Instruments , "I2C Bus Pullup Resistor Calculation". http://www.ti.com/lit/an/slva689/slva689.pdf. Last access: 11/05/2019 a las 11:45.

[VISH] Vishay Semiconductors , "Vishay's TSSP-AGC P Sensor Series for Proximity Sensing". https://pdfs.semanticscholar.org/b33d/0a973e715ca6b67689cb69ef4e8b62826f7b.pdf. Last access: 21/05/2019 a las 16:15.

[AUTO] Autodesk , "EAGLE: PCB design made easy", Autodesk. https://www.autodesk.com/products/eagle/overview. Last access: 02/06/2019 a las 17:50.

0[PCBW] "PCBway". https://www.pcbway.es/?gclid=EAIaIQobChMI-f-dxKzV4gIViSnTCh12SQyVEAAYASAAEgJjbPD_BwE. Last access: 02/06/2019 a las 12:05.

[SEED] Seedstudio , "What are Gerber files?". http://support.seeedstudio.com/knowledgebase/articles/493833-what-is-gerber-file. Last access: 27/05/2019 a las 16:25.

[FILE] File Info, "CAMFile Extension". https://fileinfo.com/extension/cam. Last access: 07/05/2019 a las 12:05.

[SPAR] Sparkfun Electronics , "Better PCBs in EAGLE". https://www.sparkfun.com/tutorials/115. Last access: 03/05/2019 a las 12:05.

[ESPR] Espressif , "Blufi". https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/blufi.html. Last access: 23/05/2019 a las 13:00.

[GITH] Github , "Repositories". https://github.com/. Last access: 18/06/2019 a las 11:30.

[ESPR] Espressif , "Wi-Fi and Bluetooth chipsets and solutions". https://www.espressif.com/. Last access: 16/05/2019 a las 14:15.

[ESPR] Espressif , "ESP32 Bluetooth Networking", Espressif. https://www.espressif.com/sites/default/files/documentation/esp32_bluetooth_networking_user_guide_en.pdf. Last access: 15/05/2019 a las 10:00.

[FING] Fing , "Fing app", Fing. https://www.fing.com/. Last access: 8/06/2019 a las 10:35.

[RAND] Random Nerd Tutorials , "ESP32 Flash Memory – Store Permanent Data (Write and Read)". https://randomnerdtutorials.com/esp32-flash-memory/. Last access: 4/06/2019 a las 11:00.

[ARDU] Arduino ¸"EEPROM.get Tutorial". https://www.arduino.cc/en/Tutorial/EEPROMGet. Last access: 4/06/2019 a las 12:30.

[ARDU] Arduino , "EEPROM.put Tutorial". https://www.arduino.cc/en/Tutorial/EEPROMPut. Last access: 4/06/2019 a las 16:00.

[APPI] App Inventor¸ "Anyone Can Build Apps That Impact the World". https://appinventor.mit.edu/explore/. Last access: 25/05/2019 a las 16:30.

[MIT_] MIT, http://www.mit.edu/. Last access: 27/05/2019 a las 11:05.

[APPI] App Inventor ¸"Using the Activity Starter (App Inventor 2)". http://appinventor.mit.edu/explore/ai2/activity-starter.html. Last access: 11/06/2019 a las 10:25.

[PLAY] Play Store , "Apk Inspector". https://play.google.com/store/apps/details?id=net.jevinstudios.apkinspector&hl=es. Last access: 27/05/2019 a las 14:10.