

Universidad Nacional de Córdoba
Facultad de Ciencias Exactas, Físicas y Naturales

Tomas Sarquis

Diciembre 2020

Práctica Supervisada

Monitoreo del movimiento de animales vacunos

Supervisor: Gustavo Wolfmann

Índice

1. Introducción	2
2. Entorno de trabajo	3
2.1. Propuesta	3
2.2. Lugar de trabajo	3
2.3. Hardware	3
3. Descripción del trabajo	4
3.1. Investigación	4
3.2. Desarrollo	4
3.2.1. Comunicación serie	5
3.2.2. Configuración del sensor	5
3.2.3. Recepción de mediciones	6
3.2.4. Primeras mediciones obtenidas	7
3.2.5. Almacenamiento	8
3.2.6. Transferencia	8
3.2.7. FreeRTOS	9
3.3. Resultados	9
3.3.1. Mediciones	10
3.3.2. Consumo	10
4. Conclusión	12

1. Introducción

El presente documento detalla la realización de la práctica profesional supervisada llevada a cabo por el estudiante Tomas Sarquis, perteneciente a la carrera de Ingeniería en Computación.

La práctica fue realizada para *Intelydrone*¹, empresa que actualmente se encuentra radicada en la encubadora de empresas de la Universidad Nacional de Córdoba², y que se desempeña en el rubro de *Innovación Tecnológica en Ganadería*. Gustavo Wolfmann, supervisor de la práctica que éste documento detalla, forma parte del equipo fundador de *Intelydrone*.

El trabajo realizado tuvo que ver con uno de los productos que la empresa ofrece: el *Intelytrace*. Este “es un dispositivo de posicionamiento y seguimiento que es colocado en cada animal en forma de collar en su cuello.”³



Figura 1: Vacas con el *Intelytrace* en sus cuellos

El *Intelytrace* tiene varias funcionalidades, sin embargo, la práctica supervisada se centró en el monitoreo del movimiento de los animales.

La práctica profesional tuvo una duración de 200 (doscientas) horas, repartidas en 41 (cuarenta y uno) días.

¹www.intelydrone.com

²www.incubadoraedeempresas.unc.edu.ar

³www.intelydrone.com/root/latest/products.html

2. Entorno de trabajo

2.1. Propuesta

Si bien Intelydrone ya contaba con un sistema que cumplía con la tarea de monitoreo de movimiento, la empresa buscaba rehacer el mismo, ya que el existente no cumplía con ciertos aspectos que se deseaban mejorar. Entre éstos últimos, se encontraban el poco ahorro de energía (cuestión muy importante debido a que el sistema funciona en una placa embebida alimentada por batería) y el hecho de que el programa en la placa estaba desarrollado con librerías de *Arduino*⁴.

Debido a lo anterior, se le propuso al alumno llevar a cabo la tarea anteriormente mencionada cumpliendo los siguientes requisitos:

- El programa debe ejecutarse bajo el sistema operativo de tiempo real *FreeRTOS*⁵.
- El programa debe estar implementado en el lenguaje de programación *C++*.
- El sistema debe ser corrido en la placa *ESP32*⁶ de *Espressif*⁷.
- El sistema debe ahorrar el máximo de energía posible.

2.2. Lugar de trabajo

La práctica profesional se llevó a cabo en un contexto de cuarentena causada por la pandemia mundial del año 2020. Es por esto que el alumno realizó la labor en su domicilio, previamente habiéndose puesto de acuerdo con su supervisor y proporcionándole, éste último, del *hardware* necesario para comenzar a trabajar. Para otras consultas laborales, se recurrió a medios de comunicación virtuales.

2.3. Hardware

El *hardware* utilizado para llevar a cabo el sistema consta de dos partes importantes: el módulo *ESP32-WROOM* (figura 2) y el sensor *MPU6050* (figura 3). Siendo esta última, un módulo que funciona como acelerómetro, termómetro y/o giroscópio.

Además, para llevar a cabo el diseño de la manera más cómoda posible, se utilizó el *ESP32-DevKit*.

⁴www.arduino.cc

⁵www.freertos.org

⁶www.espressif.com/en/products/modules/esp32

⁷www.espressif.com



Figura 2: Módulo (izquierda) y *DevKit* (derecha)

3. Descripción del trabajo

3.1. Investigación

Las primeras horas de trabajo fueron dedicadas a la familiarización del *hardware*: se le instaló el sistema operativo y se le dieron tareas simples para probar su correcto funcionamiento.

El paso siguiente fue el de leer la *API Guide* de *Espressif*, con la cual el estudiante pudo conocer las funciones proveídas por los desarrolladores oficiales y conocer el alcance de las mismas. Dichas funciones son usadas constantemente en el programa, y sin ellas la implementación sería mucho más ardua.

Posteriormente, la documentación del sensor también fue estudiada, ya que uno debe conocer los registros internos y sus distintas maneras de funcionamiento.

Se pudo, además, acceder a la documentación de la primera implementación del sistema (recordar que *Intelydrone* ya contaba con el mismo, pero se buscaba mejorar). De esta forma, el estudiante adquirió una idea clara del producto final.

La etapa de investigación, que duró 10 (diez) días, sirvió para tener un mejor entendimiento general de la labor, y así poder encarar de forma óptima la etapa de desarrollo.

Cabe aclarar que el estudiante se encontró en situaciones de investigaciones posteriores, a pesar de haber terminado la etapa.

3.2. Desarrollo

Una vez familiarizado con los dispositivo (*ESP32* y *MPU6050*), lo primero que se hizo fue armar el circuito de pruebas (figura 4) para poder empezar a

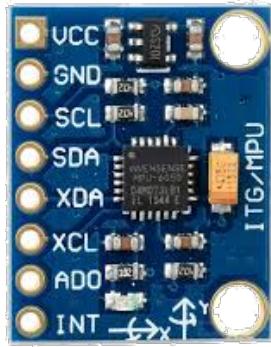


Figura 3: Sensor de movimiento

tomar las primeras medidas de aceleración. Dicho circuito se utilizó para el desarrollo únicamente, ya que más adelante se iba a utilizar la placa integrada del *Intelytrace*.

En las siguientes subsecciones, se detalla el desarrollo de las partes principales del sistema en orden cronológico.

3.2.1. Comunicación serie

Debe existir una constante comunicación entre el micro y el sensor, tanto para la configuración del último, como para la recepción de las mediciones. Para esto, el sensor es compatible con el protocolo de comunicación serie *I²C*⁸, que utiliza el modelo de control *master/slave*⁹.

Para la configuración del *driver* se utilizan las funciones de los fabricantes, mientras que para el uso del *bus*, o sea, envío y recepción de datos, se usa la librería *i2cdevlib*, de *jrowberg*¹⁰.

3.2.2. Configuración del sensor

Cuando el programa bootea (inicia por primera vez), lo primero que se hace es configurar el sensor para poder empezar a utilizarlo. Como se dijo anteriormente, esto se realiza mediante una comunicación serie, en la cual se leen y/o escriben registros del sensor.

Algunas de las configuraciones hechas, que se consideran importantes, son:

- **Activación de FIFO:** Se activa una cola, del tipo *FIFO*, de 1024 bytes para almacenar mediciones.

⁸docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/i2c.html

⁹[en.wikipedia.org/wiki/Master/slave_\(technology\)](https://en.wikipedia.org/wiki/Master/slave_(technology))

¹⁰www.github.com/jrowberg/i2cdevlib

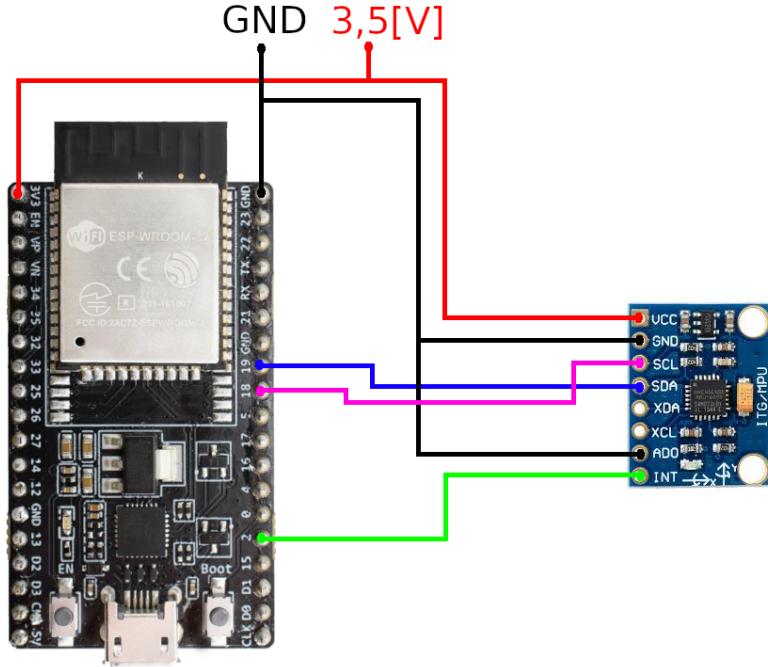


Figura 4: Circuito de pruebas

- **Interrupciones:** Se avisa al micro, mediante una interrupción, que la cola del sensor se ha llenado. Mientras el sensor va llenando la cola, el micro ahorra energía, y cuando la cola se llena, el micro “despierta” y la vacía.
- **Apagado de sensores:** Se apagan el sensor giroscópico y el de temperatura, ya que no son tenidos en cuenta.
- **Calibración:** Se calibra el sensor, de forma que, cuando se encuentra quieto, las mediciones arrojen un valor cercano al cero.
- **Arranque de mediciones:** En su estado inicial el sensor no realiza mediciones, por lo que se lo debe configurar para que empiece a hacerlo.

3.2.3. Recepción de mediciones

A medida que el sensor va tomando mediciones, los valores de las mismas van escribiéndose en la cola *FIFO*. La estructura y el orden de lo anterior se refleja en la tabla 1.

Una medición consiste en 3 valores: uno por cada eje gravitacional (X,Y,Z). Los valores de cada eje tienen un largo de 16 bits, que son guardados en 2

Registro
ACCEL_XOUT_H_0
ACCEL_XOUT_L_0
ACCEL_YOUT_H_0
ACCEL_YOUT_L_0
ACCEL_ZOUT_H_0
ACCEL_ZOUT_L_0
ACCEL_XOUT_H_1
ACCEL_XOUT_L_1
ACCEL_YOUT_H_1
ACCEL_YOUT_L_1
ACCEL_ZOUT_H_1
ACCEL_ZOUT_L_1
...
ACCEL_XOUT_H_N
ACCEL_XOUT_L_N
ACCEL_YOUT_H_N
ACCEL_YOUT_L_N
ACCEL_ZOUT_H_N
ACCEL_ZOUT_L_N

Cuadro 1: Estructura de cola con mediciones registradas, siendo N la cantidad de las mismas

registros de 8 *bits* cada uno, lo que da un total de 6 registros por medición (un total de 48 *bits*). Lo que interesa es saber el módulo del vector gravitacional, que es calculado como:

$$|ACCEL| = \sqrt{ACCEL_OUT_X^2 + ACCEL_OUT_Y^2 + ACCEL_OUT_Z^2}$$

Finalmente, el módulo es mapeado a un valor entero de 8 *bits*, que puede ir desde 0 (mínima aceleración) a 255 (máxima aceleración)¹¹.

3.2.4. Primeras mediciones obtenidas

Los primeros resultados arrojados se pueden observar en la figura 5, los cuales fueron obtenidos midiendo la aceleración humana caminando y trotando a distintas velocidades. La cantidad de mediciones es de 506.

¹¹En las pruebas realizadas, éste valor no supera las 100 unidades

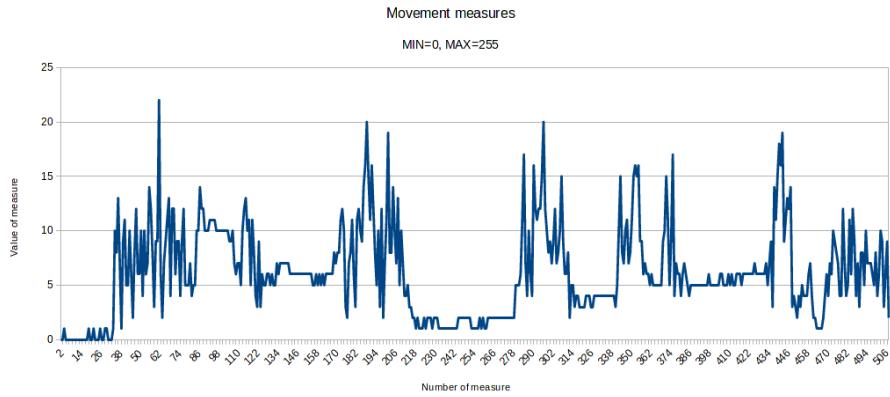


Figura 5: Primeras mediciones obtenidas

3.2.5. Almacenamiento

El microcontrolador cuenta con distintas maneras de almacenar datos.

Primeramente se optó por utilizar la memoria *NVS*¹² (“Almacenamiento no volátil”) por su facilidad, pero se tuvo que pensar en otra manera debido a que este método utiliza la memoria *Flash*, y el uso constante de la misma tiende a degradarla.

Finalmente, con ayuda del supervisor de práctica, se decidió por usar un sistema (figura 6) de memorias de dos niveles: los datos recibidos desde el sensor se van acumulando en la memoria *RTC RAM* del microcontrolador (que no se degrada con el uso), y cuando ésta se llena, se transfieren los datos al *filesystem SPIFFS* (que tiene muchas más capacidad de almacenamiento), dejando vacía la primera. Luego se vuelve a usar la *RTC RAM*, y el proceso se repite hasta que el *filesystem* agota su capacidad.

El sistema *SPIFFS* utiliza la memoria *Flash*, pero como está en un “segundo nivel”, no se escribe tan seguido y por lo tanto, no se degrada tan rápidamente.

Cuando el *filesystem* se llena, se debe vaciar y transferir los datos al servidor.

Se busca que el envío de las mediciones no se realice de manera frecuente, ya que el consumo de energía se vería afectado negativamente. Es por esto que se intentó aprovechar al máximo la fase de almacenamiento.

3.2.6. Transferencia

La idea es que, cuando sea necesario, se pueda enviar todas las mediciones a un servidor de la red local (en un *drone* que circula cerca) de forma segura.

¹²docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/storage/nvs_flash.html

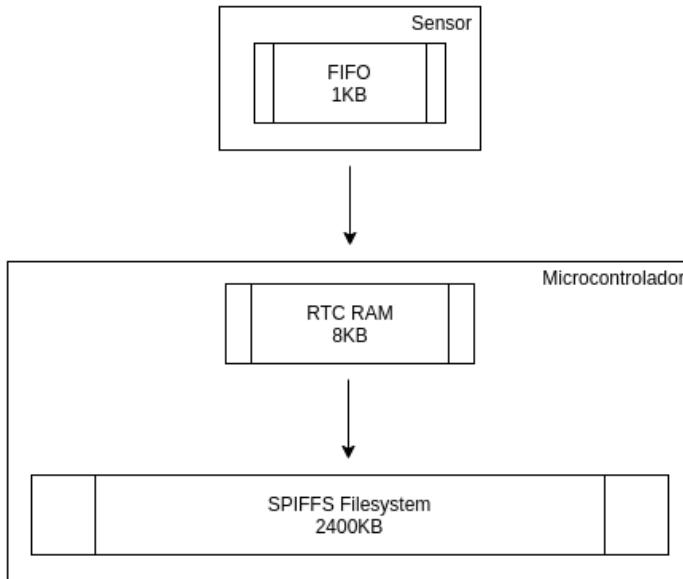


Figura 6: Esquema de memoria en niveles

Para esto, se usó el protocolo *HTTP*, mientras que la tecnología inalámbrica fue *Wi-Fi*¹³.

La transferencia consiste en varios paquetes *HTTP* de tamaño configurable. Para las pruebas se han usado paquetes de 100 datos cada uno.

3.2.7. FreeRTOS

El sistema operativo que orquesta el sistema es *FreeRTOS*, el cuál estuvo diseñado por 3 tareas:

- **Tarea principal:** Encargada de crear las demás tareas y los recursos compartidos.
- **Tarea de acelerómetro:** Su misión es la de recepción y almacenamiento de las mediciones.
- **Tarea de transferencia:** Realiza las tareas de envío de datos al servidor.

3.3. Resultados

Los resultados a continuación detallados han sido recavados de la ejecución del sistema en el *Intelytrace* y no en la placa de pruebas.

¹³El sistema debía usar *LoRa* pero el estudiante no contó con el *hardware* necesario



Figura 7: *Drone* funcionando como servidor

3.3.1. Mediciones

Los resultados arrojados se pueden observar en la figura 8, los cuales fueron obtenidos midiendo la aceleración humana caminando y trotando a distintas velocidades. La cantidad de mediciones es de 1488.

3.3.2. Consumo

Uno de los requisitos principales era que el consumo de la placa sea lo menor posible y para esto se hizo incapié en que el microcontrolador se encuentre en modo *sleep* la mayor parte del tiempo.¹⁴

A pesar de esto, los resultados obtenidos en cuanto al amperaje no fueron los esperados, como se puede ver en la tabla 2.

Consumo esperado	Consumo obtenido
10 [uA]	4.3 [mA]

Cuadro 2: Consumo esperado y obtenido (*deep-sleep mode*)

La causa de esta diferencia en el consumo han sido atribuidas al *hardware* y no al *software* implementado, ya que se han llevado a cabo pruebas con programas

¹⁴El estudio del consumo considera únicamente las etapas de *sleep* ya que en ésta etapa transcurre más del 90 % del programa

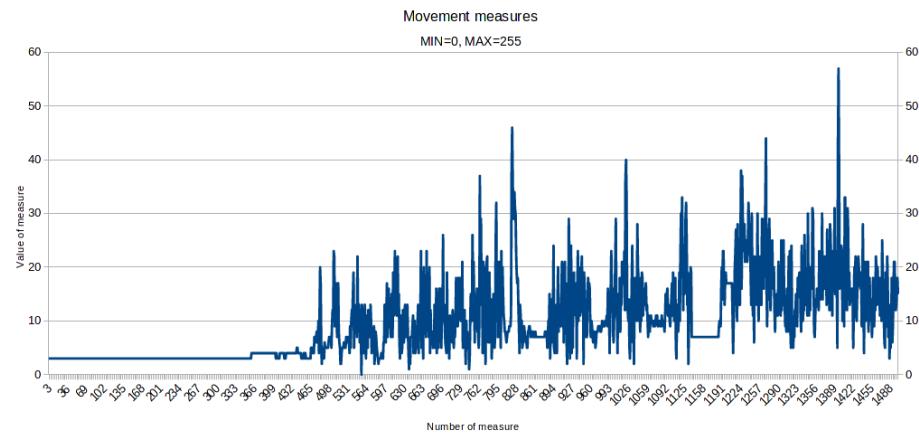


Figura 8: Resultados obtenidos

oficiales¹⁵ y los resultados han sido los mismos.

¹⁵www.github.com/espressif/esp-idf/tree/master/examples/system/deep_sleep

4. Conclusión

La práctica supervisada ha sido una actividad realmente enriquecedora tanto para el alumno como para el supervisor y su empresa. El estudiante tuvo una experiencia real de la implementación completa de un *software*, pasando por el análisis de los requerimientos, el diseño, las posibilidades de desarrollo, las investigaciones posteriores y finalmente el desarrollo y los resultados.

El sistema implementado en estos meses es susceptible a (y debería, en lo posible) ser mejorado, ya que ciertas cuestiones del programa lo ameritan. Sin embargo, teniendo en cuenta el tiempo empleado para el desarrollo y la poca experiencia del desarrollador, el resultado es muy favorable.

Los conocimientos obtenidos en la universidad y las ayudas recibidas por parte del supervisor y de la comunidad informática, fueron de necesidad para el estudiante.