UNIVERSIDAD DE BUENOS AIRES
Facultad de Ingeniería
86.65 Sistemas Embebidos

Memoria del Trabajo Final:

PetFeeder: Sistema de alimentación remota de mascotas.

Autor:

Ulises Montenegro

Legajo: 102.921

Este trabajo fue realizado en las Ciudad Autónoma de Buenos Aires, entre marzo y julio de 2023.

RESUMEN

PetFeeder es un sistema de alimentación remota de mascotas. El sistema da aviso al usuario cuando se vacía el plato de alimento y permite el llenado de una porción configurable mediante un botón o desde un smartphone.

Para el desarrollo del presente trabajo se utilizó una placa NUCLEO-F429ZI. Fue programado en C++ mediante el IDE Keil Studio Cloud y el sistema operativo Mbed. Se aplicaron conceptos adquiridos durante la carrera de Ingeniería Electrónica tales como programación orientada a objetos, máquinas de estados, modularización, manejo de un servomotor mediante una señal PWM, resistencias de pull-up, y cómo llevar a cabo el proyecto realizando pruebas aisladas a los distintos módulos para luego integrarlos.

En la presente memoria se podrá encontrar la motivación del proyecto, comparaciones con productos similares, diseño e implementación tanto del *hardware* como del *firmware*, pruebas funcionales y conclusiones.

ABSTRACT

PetFeeder is a remote pet feeder system. This system notifies the user when the plate is empty and allows its filling with a portion that is a configurable parameter. This can be done with a button or using a smartphone.

For the development of this work, a NUCLEO-F429ZI board was used. It was programmed in C++ using Keil Studio Cloud IDE and Mbed operating system. Knowledge learnt throughout the Electronics Engineering degree such as object oriented programming, finite state machines, modularization, PWM signals, pull-up resistors, and how to tackle the project testing each module separately and then integrating all of them was applied.

In this report the reader can find the project motivation, comparisons with similar products, design and implementation of both hardware and firmware, tests and conclusions.

Agradecimientos

Hago una mención especial a mi amigo Franco Abbenda de Oto, quien fue de gran ayuda en el diseño e impresión 3D así como también en el armado y grabación del video.

Índice General

Registro de versiones

Revisión	Cambios realizados	Fecha
1.0	Creación del documento	05/07/2023

CAPÍTULO 1

Introducción general

1.1 Descripción del trabajo

El principal funcionamiento del *PetFeeder* [1] es llenar el plato de alimento de una mascota de manera remota. De esta manera se reduce la cantidad de veces que se sirve alimento a la mascota, aportando comodidad al usuario. También es útil para tener un control en la cantidad de alimento que consume la mascota.

Teniendo la capacidad de almacenar una cantidad determinada de porciones, una vez que el plato queda vacío, el dispositivo avisa esta situación mediante un LED y por medio del smartphone si este se encuentra conectado via BLE (*Bluetooth Low Energy*) de manera que el usuario pueda dar la orden de volver a llenarlo. Mediante una balanza sensa la cantidad de alimento que hay en el plato, esta información es leída por el microcontrolador y de esta manera se sabe cuando el plato está vacío. Cuando el usuario envía el comando para llenar el plato, el microcontrolador activa un actuador (servomotor) que permite el paso de alimento desde el contenedor hacia el plato. Tanto el peso del plato vacío como la cantidad de alimento de una porción son parámetros modificables que son utilizados por el microcontrolador.

Este proyecto se destaca ya que los existentes en la actualidad están enfocados en alimentar de manera automática según un horario estipulado, mientras que la función de este trabajo es avisar cuando está el plato vacío, siendo el usuario quien habilita el llenado del mismo.

En la Figura 1.1 se presenta el diagrama en bloques del sistema. Se observa que se utilizó la placa NUCLEO-F429ZI [2]. El sistema está formado por dicho módulo, por un sensor de peso y por un servomotor que es el actuador que permite el paso de alimento para la mascota. Además, tiene un módulo que se encarga de la comunicación con el smartphone via BLE, un botón y un LED. El sistema es alimentado por una fuente de 5 V que se enchufa al tomacorriente de 220 V.

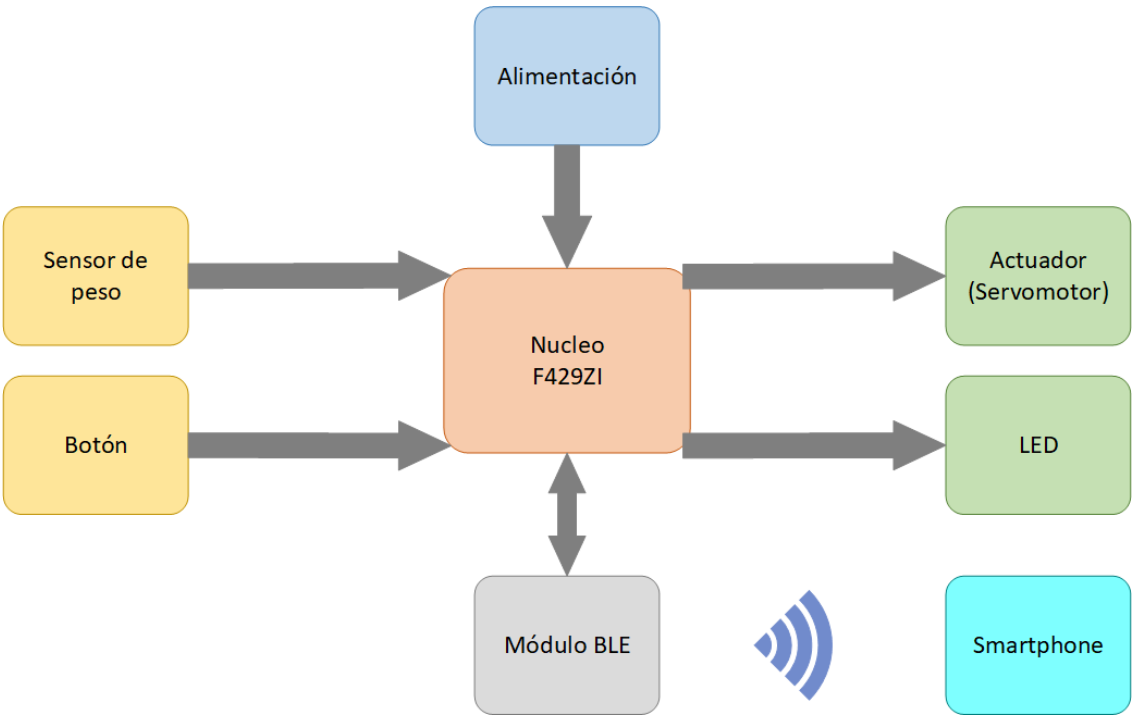


Figura 1.1: Diagrama en bloques del sistema.

1.2 Análisis del estado del arte

Previo al desarrollo del trabajo, se comparó este con otros dos productos similares que se presentan en la Tabla 1.1.

El presente trabajo se diferencia de los otros ya que provee la funcionalidad de dar aviso y que sea el usuario quien de la orden del paso de comida, en vez de programarlo para que lo realice automáticamente.

Tabla 1.1: Resumen de características principales de otros productos similares.

Característica	Producto 1 (Link)	Producto 2 (Link)
Capacidad total	6 L	5.5 L
Alimentación	110 V / 220 V	220 V
Interfaz con el usuario	Smartphone o botonera	Botonera
Conexión	Wi-Fi	No
Funcionamiento	Automático. Programable por horas	Automático. Programable por horas
Precio	US$ 136	US$ 74

CAPÍTULO 2

Introducción específica

2.1 Requisitos

En base a las características mencionadas en la Sección 1.2, se establecieron los requisitos del proyecto presentados en la Tabla 2.1.

Tabla 2.1: Requisitos del proyecto.

Grupo	ID	Descripción
1. Capacidad total	1.1	La capacidad total deberá ser de 5 L como mínimo.
2. Alimentación	2.1	El sistema debe poder alimentarse a una toma de 220 V.
3. Interfaz mediante botones	3.1	El sistema debe proveer una interfaz mediante un botón que permita el llenado del plato.
4. Interfaz mediante smartphone	4.1	El sistema debe proveer una interfaz mediante un smartphone donde el usuario pueda verificar el estado del plato y llenarlo, o configurar una nueva porción y/o un nuevo plato.
5. Modos	5.1	El sistema debe tener un modo de funcionamiento normal.
	5.2	El sistema debe tener un modo de configuración de parámetros (peso del plato y peso de una porción).
6. Conexión BLE	6.1	El sistema debe poder conectarse a la placa mediante BLE.
7. Sensor	7.1	El sistema deberá tener un sensor de peso (como una celda de carga para Arduino) con una capacidad de carga no menor a 1 Kg y una precisión menor al 1%.
8. Actuador	8.1	El sistema deberá tener un servomotor que permita el paso de el alimento al plato.
9. Fecha de finalización	9.1	El proyecto debe estar terminado el día 28 de Junio de 2023.
10. Documentos	10.1	El prototipo deberá estar acompañado de una lista de partes, un diagrama de conexiones, un repositorio del

		código, y una tabla indicando el cumplimiento de los requisitos y usos de caso.
11. Costo	11.1	El prototipo debe costar menos de US$ 136.

Los requisitos correspondientes a la conexión BLE fueron modificados respecto a los originales ya que estos eran mediante Wi-Fi y un bot de Telegram. Esta modificación surgió debido a los tiempos disponibles para la realización del trabajo.

2.2 Casos de uso

En la Tablas 2.2, 2.3, 2.4 y 2.5 se describen 4 casos de uso: llenado del plato a través del smartphone, llenado del plato a través del botón, configuración de una nueva porción, y configuración de un nuevo plato.

Tabla 2.2: Caso de uso #1 - El usuario desea llenar el plato a través del smartphone.

Elemento	Definición
Disparador	El usuario envía un comando a través de una aplicación BLE desde su smartphone.
Precondiciones	El sistema debe estar conectado al smartphone via BLE y el plato debe estar vacío.
Flujo básico	El sistema abre el paso para llenar el plato hasta una porción de alimento.
Flujos alternativos	1.a. El sistema no está conectado al smartphone BLE, por lo tanto no llenará el plato. 1. b. El plato no está vacío, por lo tanto el sistema llenará el plato hasta una porción de alimento.

Tabla 2.3: Caso de uso #2 - El usuario desea llenar el plato a través del botón.

Elemento	Definición
Disparador	El usuario presiona el botón para llenar el plato.
Precondiciones	El plato debe estar vacío.
Flujo básico	El sistema abre el paso para llenar el plato con una porción de alimento.
Flujos alternativos	2. a. El plato no está vacío, por lo tanto el

	sistema llenará el plato hasta una porción de alimento.

Tabla 2.4: Caso de uso #3 - El usuario desea configurar una nueva porción de alimento.

Elemento	Definición
Disparador	El usuario envía un comando a través de una aplicación BLE desde su smartphone para configurar una nueva porción.
Precondiciones	El sistema debe estar conectado con el smartphone via BLE.
Flujo básico	El sistema guarda la información del peso de la cantidad de alimento deseado.
Flujos alternativos	3.a. El sistema no está conectado al smartphone via BLE, por lo tanto no guardará la información del peso de la porción.

Tabla 2.5: Caso de uso #4 - El usuario desea configurar un nuevo plato para su mascota.

Elemento	Definición
Disparador	El usuario envía un comando a través de una aplicación BLE desde su smartphone para configurar un nuevo plato.
Precondiciones	El sistema debe estar conectado al smartphone via BLE.
Flujo básico	El sistema guarda la información del peso del plato.
Flujos alternativos	3.a. El sistema no está conectado al smartphone via BLE, por lo tanto no guardará la información del peso del plato.

2.3 Módulos utilizados

2.3.1 Celda de carga y módulo HX711

Para el sensor de peso se utilizó una celda de carga compatible con Arduino y un módulo amplificador de dicha señal, el HX711 [3]. La celda de carga es un transductor que convierte una fuerza en una señal eléctrica mediante un puente de Wheatstone, dicha señal es luego amplificada y convertida a una señal digital por el módulo HX711 el cual posee un conversor A/D de 24 bits. El HX711 posee 4 pines para conectarse a la placa: VCC, GND, DT y SCK. Es alimentado por 5 V extraídos de la placa, el pin SCK va conectado al Clock de la placa y por el pin DT se transmite la información medida por la celda de carga. Se ilustra dicho módulo en la Figura 2.1.

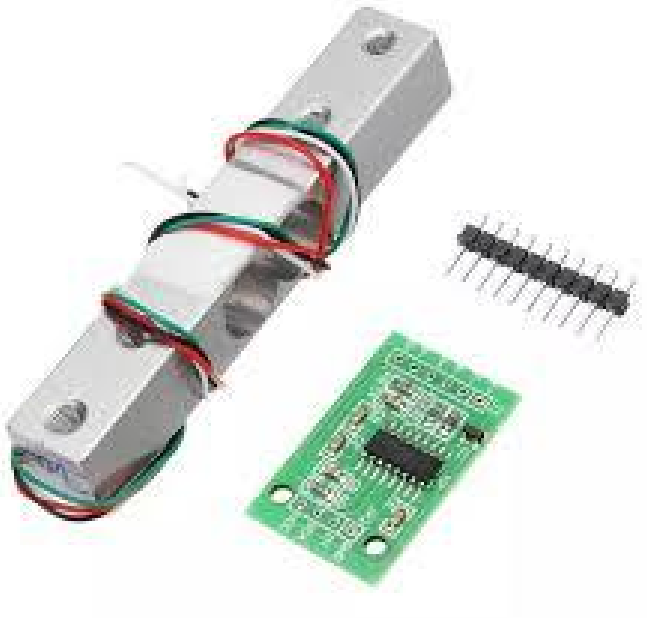


Figura 2.1: Celda de carga y módulo HX711.

2.3.2 Servomotor SG90

Este servomotor SG90 [4], que fue utilizado en el trabajo como actuador, modifica su posición según el *duty cycle* (ciclo de trabajo) de la señal PWM (*Pulse Width Modulation*) que recibe como entrada. Tiene 3 pines de entrada: VCC, GND y PWM. Para este trabajo fue alimentado con 5 V extraídos de la placa. Se presenta dicho módulo en la Figura 2.2.

Figura 2.2: Servomotor SG90.

2.3.3 Módulo BLE HM-10

Para la conexión BLE se utilizó el módulo HM-10 [5], que se encarga de conectarse al smartphone via BLE y al microcontrolador via UART. Posee 6 pines: VCC, GND, TXD, RXD, STATE, EN, de los cuales se utilizaron sólo 4: VCC, que se conectó a 5 V provenientes de la placa, GND, TXD conectado a Rx de la placa, y RXD conectado a Tx de la placa. Se ilustra dicho módulo en la Figura 2.3.

Figura 2.3: Módulo BLE HM-10.

2.4 Bibliotecas utilizadas

2.4.1 Mbed

Para la programación de la placa NUCLEO se utilizó la librería de Mbed [6]. Este es una plataforma y un sistema operativo para dispositivos que se conectan a internet basados en microcontroladores ARM Cortex-M de 32 bits.

2.4.2 arm_book_lib.h

Se utilizó el archivo arm_book_lib.h proveniente del libro del curso [7].

2.4.3 HX711

Para el manejo de la celda de carga y el módulo HX711 se utilizaron los archivos HX711.h y HX711.cpp importados desde el buscador de librerías de Mbed [8].

CAPÍTULO 3

Diseño e implementación

3.1 Hardware del sistema

3.1.1 Diagrama en bloques

En la Figura 3.1 se presenta un diagrama en bloques del hardware del sistema similar al de la Figura 1.1 pero en este caso con los componentes utilizados y los tipos de conexiones con la placa.

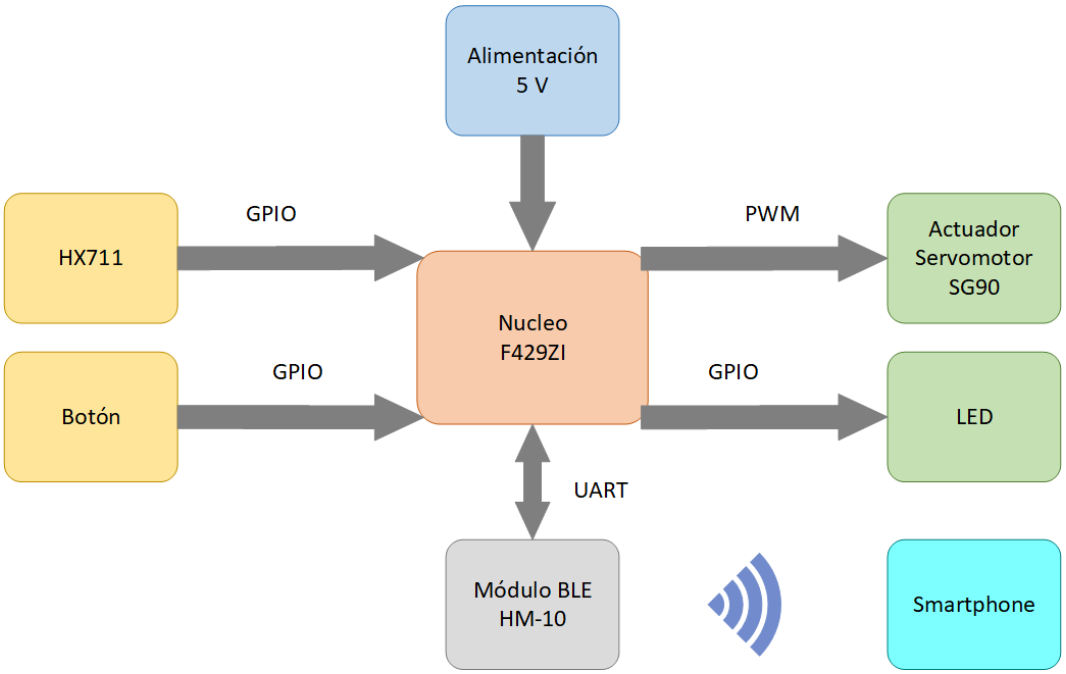


Figura 3.1: Diagrama en bloques con componentes y tipos de conexión.

Se utiliza la placa NUCLEO-F429ZI, la cual se alimenta con una fuente de 5 V conectada al tomacorriente de 220 V. Por medio de GPIO está conectada a dos entradas: módulo HX711 (Sección 2.3.1) y un botón, y una salida: un LED. La otra de sus salidas, del tipo PWM, es el servomotor SG20 (Sección 2.3.2). Finalmente, está conectado via UART al módulo BLE HM-10 (Sección 2.3.3).

3.1.2 Conexiones

En la Figura 3.2 se ilustra el diagrama de conexiones, y en las Tablas 3.1 a 3.3 se resumen las mismas.

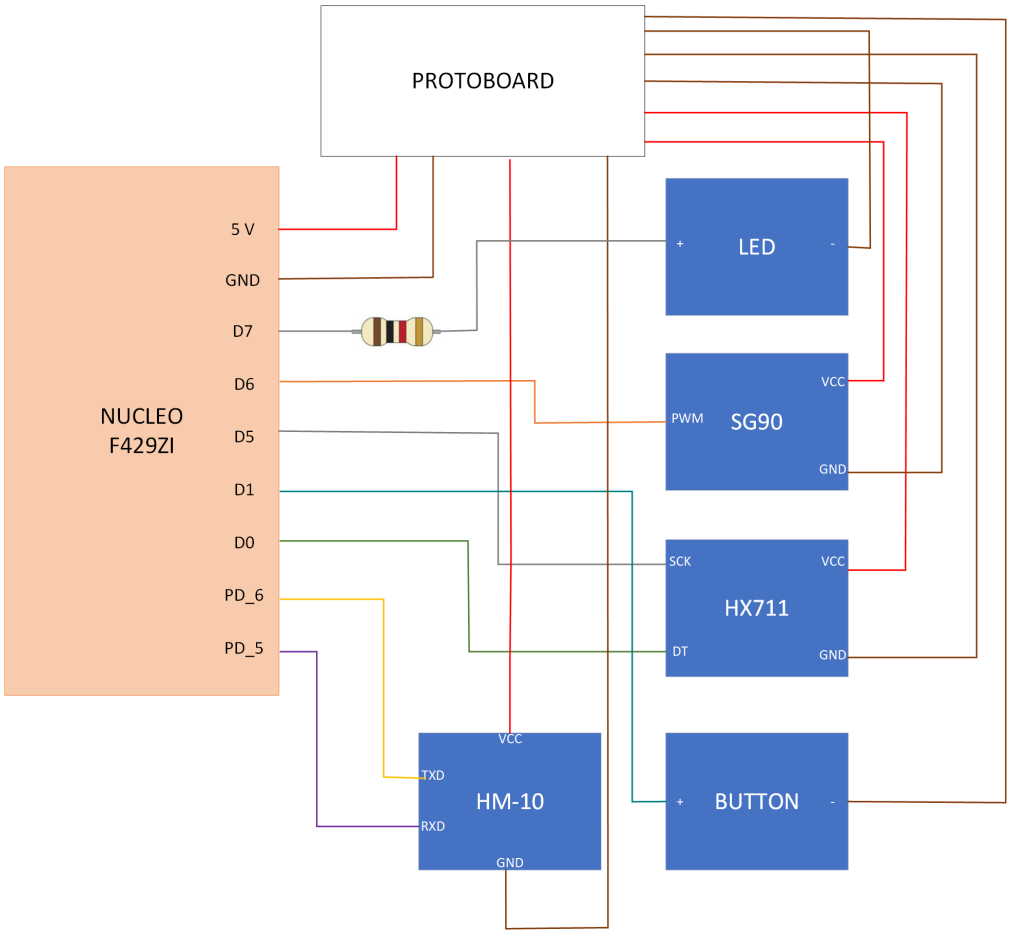


Figura 3.2: Diagrama de conexiones.

Tabla 3.1: Resumen de conexiones entre la placa NUCLEO-F429ZI y el módulo HM-10.

NUCLEO-F429ZI	HM-10
5 V	VCC
GND	GND
PD_6 (UART2_RX)	TXD
PD_5 (UART2_TX)	RXD

Tabla 3.2: Resumen de conexiones entre la placa NUCLEO-F429ZI y el módulo HX711.

NUCLEO-F429ZI	HM-10
5 V	VCC
GND	GND
D5	SCK
D0	DT

Tabla 3.3: Resumen de conexiones entre la placa NUCLEO-F429ZI y el servomotor SG90.

NUCLEO-F429ZI	HM-10
5 V	VCC
GND	GND
D6 (PWM1/1)	PWM

3.1.3 Lista de componentes

Para el correcto funcionamiento del prototipo se diseñó una pieza en 3D la cual sostiene el contenedor de alimento y permite de manera correcta el paso del mismo hacia el plato. En la Figura 3.3 se muestra un render de dicha pieza.

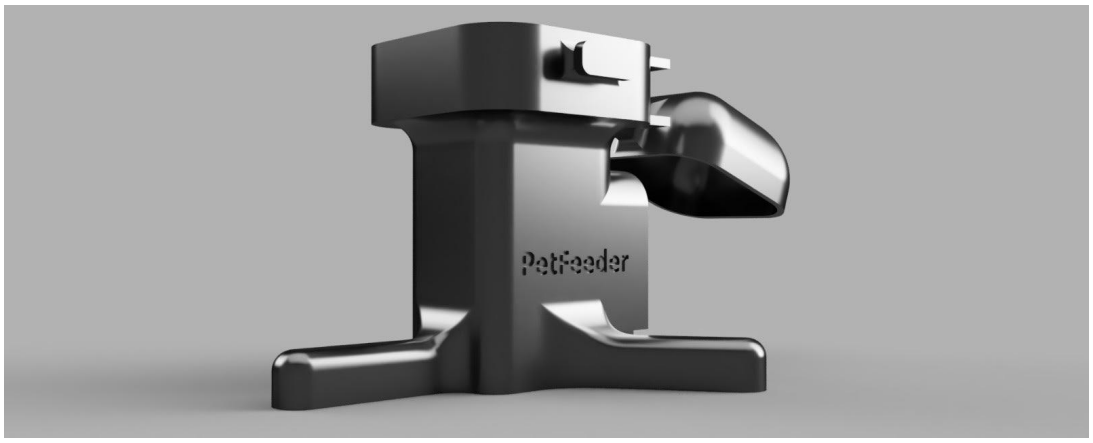


Figura 3.3: Render de la pieza diseñada e impresa en 3D.

En la Tabla 3.4 se presentan los componentes, link de compra y sus respectivos costos en dólares, así como también el costo material total del trabajo. Cabe mencionar que actualmente la placa NUCLEO-F429ZI no se consigue, de todas maneras en la Sección 5.2 se presenta una alternativa. También se aclara que la pieza 3D no tiene link ya que fue de desarrollo personal.

Tabla 3.4: Componentes, links y sus respectivos precios (en US$).

Componente	Link	Cantidad	Precio (US$)
NUCLEO-F429ZI	Link	1	$73,1
Celda de carga +Módulo HX711	Link	1	$ 6,9
Módulo HM-10	Link	1	$4,5
Servomotor SG90	Link	1	$3,4
Botón / Pulsador	Link	1	$1,5
LED	Link	1	$0,2
Resistor 1000 Ω	Link	1	$0,1
Pieza 3D	N/A	1	$9,8
TOTAL		$99,5	

3.1.4 Prototipo del sistema

Se presenta el prototipo armado del sistema en la Figura 3.4.

Figura 3.4: Prototipo del sistema armado.

3.2 Firmware del sistema

3.2.1 Funcionamiento

El *firmware* del sistema fue desarrollado en C++ utilizando el IDE Keil Studio Cloud y el sistema operativo Mbed. Keil Studio Cloud permite conectarse con Git y, entonces, hacer adds, commits, pushs, etc. Además, permite debuggear el código directamente en la placa.

Dicho *firmware* consta de un main.cpp que inicializa el objeto petFeeder y luego en un bloque while(true) lo actualiza. Dicho objeto inicializa a los demás objetos que el sistema

necesita para su funcionamiento. Cuando petFeeder es actualizado, este actualiza los objetos necesarios y corre una máquina de estados representada en la Figura 3.5. Luego, espera un tiempo de 10 ms.

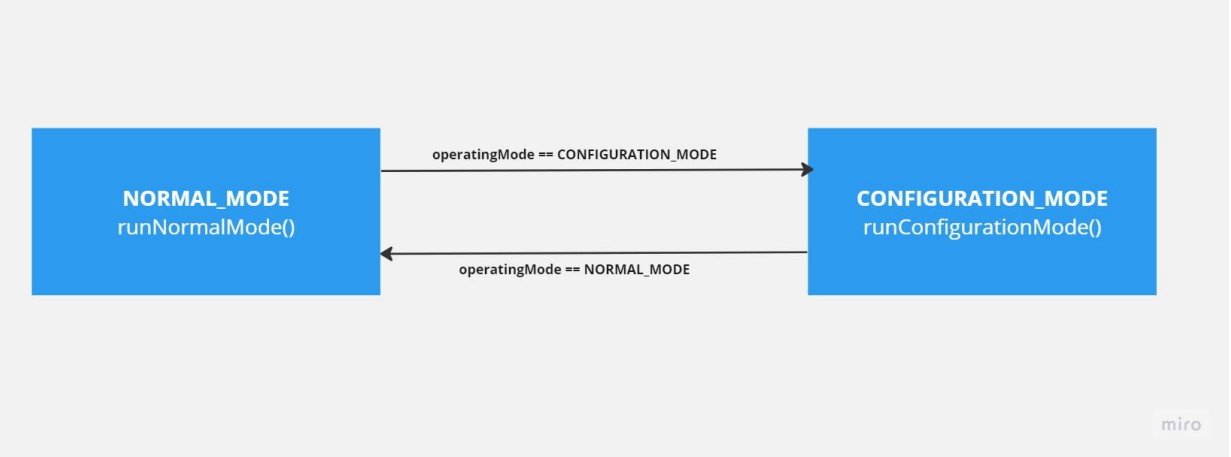


Figura 3.5: Máquina de estados principal.

En Código 3.1 se muestra el código correspondiente al funcionamiento mencionado.

```
67  void petFeederUpdate()
68  {
69      weightSensorUpdate();
70      fillPlateButtonUpdate();
71      bleComUpdate();
72
73      switch(operatingMode){
74          case NORMAL_MODE:
75              runNormalMode();
76          break;
77          case CONFIGURATION_MODE:
78              runConfigurationMode();
79          break;
80      }
81      delay(UPDATE_TIME_INCREMENT_MS);
82  }
```

Código 3.1: Actualización del objeto petFeeder.

El sistema se inicializa en modo normal (*NORMAL_MODE*), el cual corre una máquina de estados que se muestra en la Figura 3.6.

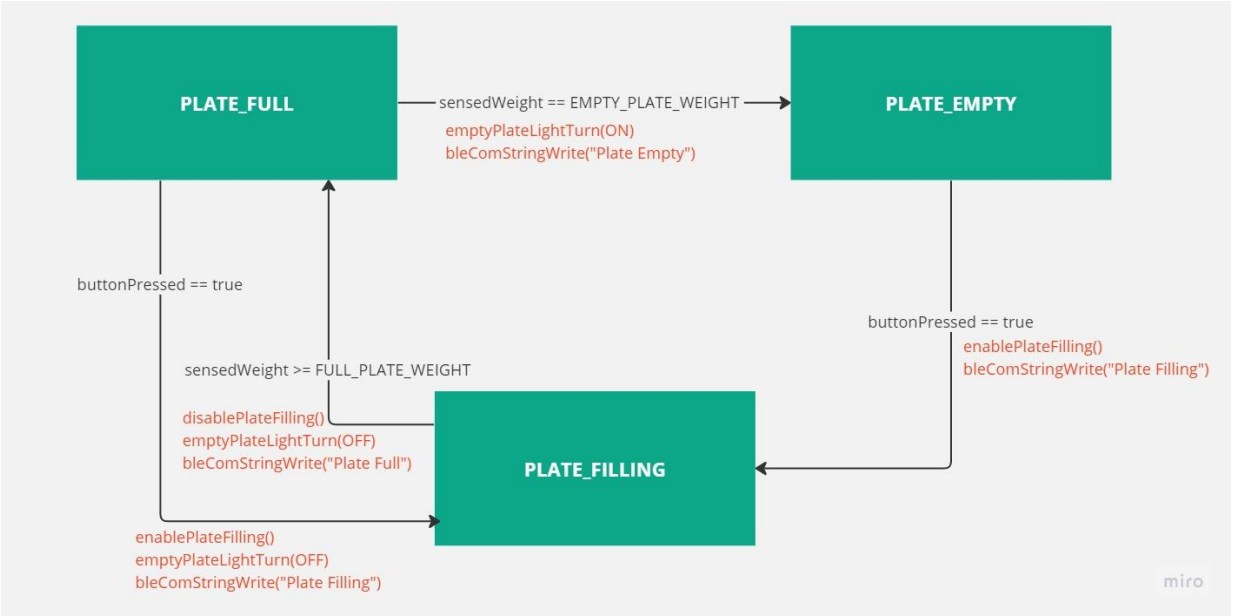


Figura 3.6: Máquina de estados del modo normal.

El sistema se inicializa en el estado de plato lleno (*PLATE_FULL*). Si el valor sensado es menor o igual al parámetro *emptyPlateWeight* (es una variable que representa el peso del plato vacío y se inicializa en un valor por default llamado *EMPTY_PLATE_WEIGHT*), entonces enciende el LED, da el aviso al smartphone via BLE y pasa al estado de plato vacío (*PLATE_EMPTY*); si esto no ocurre y detecta que el botón fue presionado (tanto real como desde el smartphone), pasa al estado de plato llenándose (*PLATE_FILLING*), permite el llenado del plato, notifica al usuario via BLE, y apaga el LED en caso de que esté prendido.

Encontrándose el sistema en estado de plato vacío, si detecta que el botón fue presionado (o si se da el comando desde el smartphone) permite el llenado del plato, notifica al usuario, y pasa al estado de plato llenándose.

Cuando el sistema está en estado de plato llenándose, verifica si el peso sensado es mayor o igual que *fullPlateWeight* (es una variable que representa el peso del plato lleno y se inicializa en un valor por default llamado *FULL_PLATE_WEIGHT*) y, en caso de serlo, cierra el paso de alimento al plato, apaga el LED, da aviso al usuario y pasa al estado de plato lleno.

Cuando, al actualizarse petFeeder, detecta que se envió una 'C', entra en modo configuración, configura el parámetro deseado ya sea el peso del plato o el de una porción, y vuelve al modo normal. En la Figura 3.7 se presenta un diagrama de flujo de este funcionamiento.

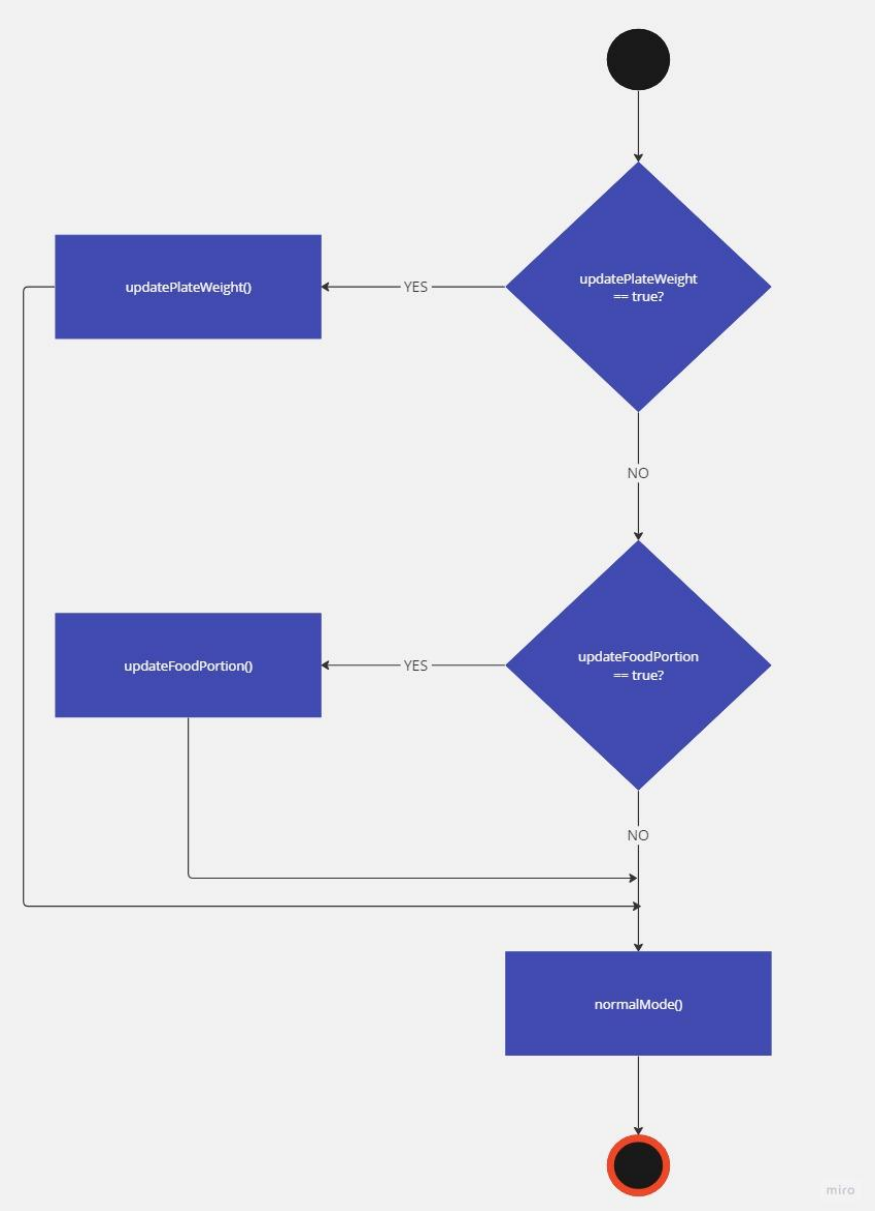


Figura 3.7: Diagrama de flujo del funcionamiento en modo configuración.

3.2.2 Modularización

En la Figura 3.8 se ilustra el diagrama de módulos del *firmware*, mientras que en la Tabla 3.5 se da una breve descripción de cada uno de ellos. En la sección 3.2.3 se describen de manera más detallada.

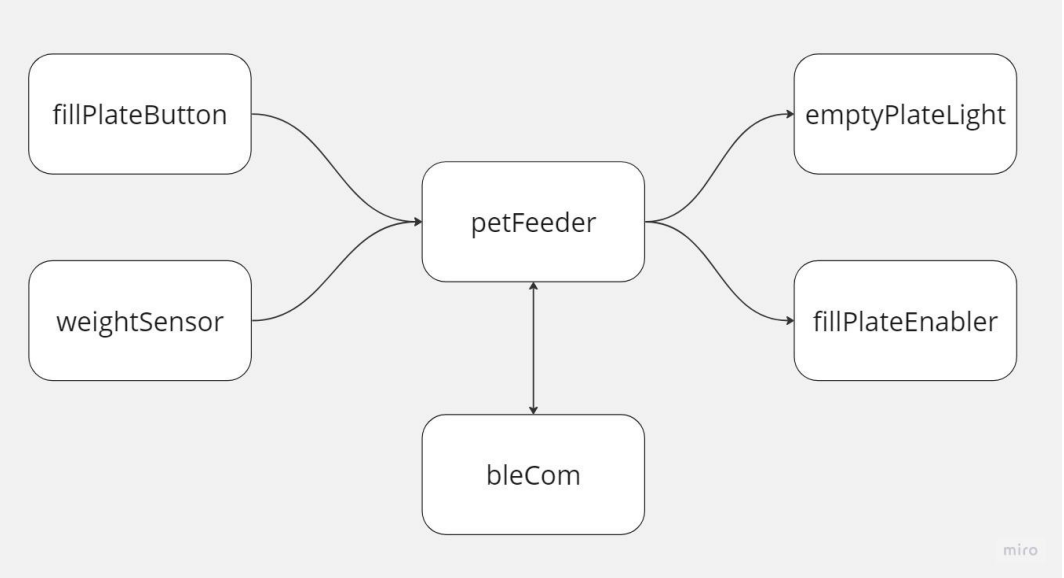


Figura 3.8: Diagrama de módulos del *firmware*.

Tabla 3.5: Descripción de los módulos del *firmware*.

Módulo	Descripción
petFeeder	Es el módulo principal del sistema. Se encarga tanto de inicializar los módulos y variables necesarias como de actualizar los módulos necesarios para correr las máquinas de estados mencionadas.
fillPlateButton	*Driver* del botón de llenado del plato. Representa tanto el botón físico como el comando desde el smartphone.
weightSensor	*Driver* del sensor de peso.
emptyPlateLight	*Driver* del LED.
fillPlateEnabler	*Driver* del servomotor. Habilita y deshabilita el llenado del plato.
bleCom	*Driver* del módulo BLE. Permite la conexión con el smartphone via BLE.

Luego de mencionar los módulos, se muestra cómo quedaron organizados los archivos en la Figura 3.9. Se destaca el concepto de modularización visto en el curso y durante la carrera.

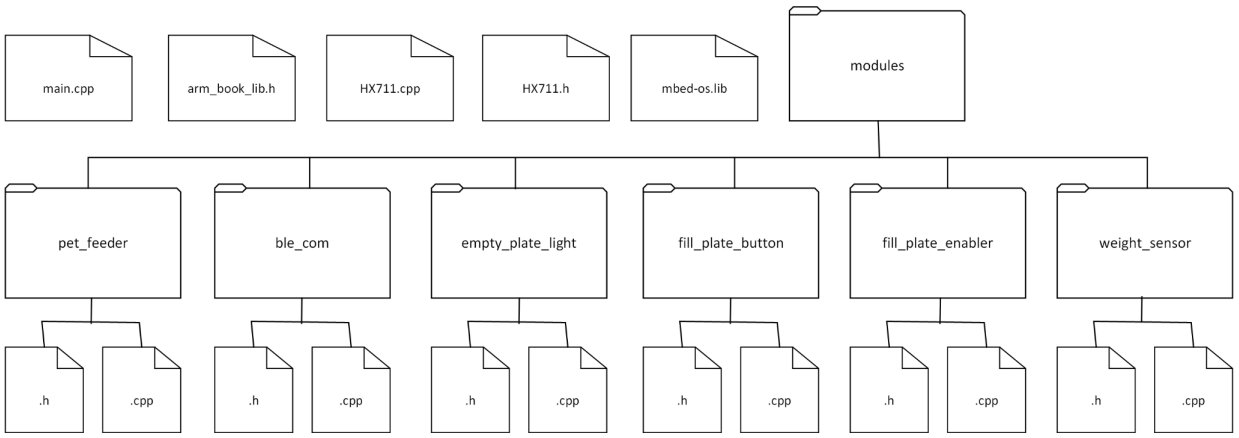


Figura 3.9: Organización de archivos y carpetas.

3.2.3 Descripción de los módulos

En las Tablas 3.6 a 3.17 se muestran las variables, objetos y métodos públicos de los módulos del programa, así como también qué otros módulos utilizan dichos métodos. Se hizo hincapié en los roles y responsabilidades de cada módulo de manera de no romper encapsulamiento.

Tabla 3.6: Variables y objetos del módulo petFeeder.

Nombre	Tipo	Descripción
operatingMode_t	typedef enum (tipo enumerativo)	Puede tomar los valores NORMAL_MODE o CONFIGURATION_MODE.
plateState_t	typdef enum (tipo enumerativo)	Puede tomar los valores PLATE_FULL, PLATE_FILLING o PLATE_EMPTY.
operatingMode	operatingMode_t (variable)	Se utiliza para manejar la máquina de estados principal del programa.
plateState	plateState_t (variable)	Se utiliza para manejar la máquina de estados del modo normal.

emptyPlateWeight	long (variable)	Se utiliza para almacenar el peso del plato vacío configurado.
fullPlateWeight	long (variable)	Se utiliza para almacenar el peso del plato llenado con una porción (plato vacío + porción)
foodPortionWeight	long (variable)	Se utiliza para almacenar el peso de una porción de alimento.
updatePlateWeightSignal	bool (variable)	Se utiliza para saber si el usuario pidió configurar un nuevo plato.
updateFoodPortionSignal	bool (variable)	Se utiliza para saber si el usuario pidiò configurar una nueva porción.

Tabla 3.7: Métodos públicos del módulo petFeeder.

Nombre	Descripción	Módulos que lo usan
petFeederInit()	Inicializa los módulos y variables necesarias para el correcto funcionamiento del programa.	main.cpp
petFeederUpdate()	Actualiza los módulos que necesita. Corre la máquina de estados principal.	main.cpp
updatePlateWeight()	Pone en true la variable updatePlateWeightSignal.	bleCom
updateFoodPortion()	Pone en true la variable updateFoodPortionSignal.	bleCom
configurationMode()	Pone la máquina de estados principal en modo configuración y avisa al usuario que se entró en dicho modo mediante el smartphone.	bleCom

Tabla 3.8: Variables y objetos del módulo bleCom.

Nombre	Tipo	Descripción
uartBle	UnbufferedSerial (Objeto)	Se utiliza para comunicar la placa con el módulo BLE mediante UART.

Tabla 3.9: Métodos públicos del módulo bleCom.

Nombre	Descripción	Módulos que lo usan
bleComUpdate()	Llama a los correspondientes métodos según el caracter que recibe (ver Figura 3.10).	petFeeder
bleComStringWrite()	Escribe en el smartphone una cadena de caracteres pasada como parámetro.	petFeeder

```
28  void bleComUpdate()
29  {
30      char receivedChar = bleComCharRead();
31      if( receivedChar != '\0' ) {
32          switch (receivedChar) {
33              case 'C':
34                  configurationMode();
35              break;
36              case 'P':
37                  updatePlateWeight();
38              break;
39              case 'F':
40                  updateFoodPortion();
41              break;
42              case 'B':
43                  pressButton();
44              break;
45          }
46      }
```

Figura 3.10: Método bleComUpdate().

Como se puede observar en las líneas 33 y 34, si se recibe una 'C' entra en modo configuración. En las líneas 36 y 37 se observa que, al recibir una 'P', actualiza el peso del plato. Al recibir una 'F' actualiza el peso de la porción de alimento (líneas 39 y 40). Y finalmente, al recibir una 'B' acciona el botón para el llenado del plato (líneas 42 y 43).

Tabla 3.10: Variables y objetos del módulo emptyPlateLight.

Nombre	Tipo	Descripción
emptyPlateLight	DigitalOut (Objeto)	Se utiliza para manejar el LED.

Tabla 3.11: Métodos públicos del módulo emptyPlateLight.

Nombre	Descripción	Módulos que lo usan
emptyPlateLightInit()	Inicializa el LED apagado.	petFeeder
emptyPlateLightTurn()	Prende o apaga el LED según el parámetro que se le pasa.	petFeeder

Tabla 3.12: Variables y objetos del módulo fillPlateButton.

Nombre	Tipo	Descripción
fillPlateButton	DigitalIn (Objeto)	Se utiliza para manejar el botón ya sea físico o su comando equivalente desde el smartphone.
fillPlateButtonPressed	bool (variable)	Se utiliza para saber si el botón fue apretado (ya sea físico o desde el smartphone)

Tabla 3.13: Métodos públicos del módulo fillPlateButton.

Nombre	Descripción	Módulos que lo usan
fillPlateButtonInit()	Inicializa el botón físico como pull-up. Inicializa la variable fillPlateButtonPressed en false.	petFeeder
fillPlateButtonUpdate()	Actualiza el valor de la variable fillPlateButtonPressed según se apretó o no el botón físico.	petFeeder
isButtonPressed()	Se utiliza para saber si el botón fué apretado (ya sea físico o desde el smartphone). Devuelve el estado de fillPlateButtonPressed.	petFeeder

pressButton()	Pone en true la variable fillPlateButtonPressed.	bleCom

Tabla 3.14: Variables y objetos del módulo fillPlateEnabler.

Nombre	Tipo	Descripción
servomotor	PwmOut (Objeto)	Se utiliza para manejar el servomotor habilitando o deshabilitando el paso de alimento.
dutyCycle	float (variable)	Se utiliza para establecer la posición del servomotor.

Tabla 3.15: Métodos públicos del módulo fillPlateEnabler.

Nombre	Descripción	Módulos que lo usan
fillPlateEnablerInit()	Establece el período de la señal PWM. Inicializa el servomotor de manera que el paso de alimento esté deshabilitado.	petFeeder
enablePlateFilling()	Habilita el paso de alimento al plato.	petFeeder
disablePlateFilling()	Deshabilita el paso de alimento al plato.	petFeeder

Tabla 3.16: Variables y objetos del módulo weightSensor.

Nombre	Tipo	Descripción
balance	HX711 (Objeto)	Se utiliza para manejar el sensor de peso.
currentWeight	long (variable)	Se utiliza para almacenar el valor actual sensado por el sensor de peso.

tareValue	long (variable)	Se utiliza al principio para no tener en cuenta el peso propio de la balanza en el momento de realizar una medición.

Tabla 3.17: Métodos públicos del módulo weightSensor.

Nombre	Descripción	Módulos que lo usan
weightSensorInit()	Inicializa currentWeight en cero y tareValue en el valor sensado en ese momento.	petFeeder
weightSensorUpdate()	Actualiza currentWeight tomando una muestra del peso sensado en el momento y restándole tareValue.	petFeeder
sensedWeight()	Devuelve currentWeight (el valor de sensado en la última actualización).	petFeeder

CAPÍTULO 4

Ensayos y resultados

4.1 Pruebas funcionales

4.1.1 Simulación del comportamiento general

En primer lugar se armó un banco de pruebas con componentes bien conocidos para corroborar de manera rápida el correcto funcionamiento del programa. En primer lugar se verificó que funcionen bien las primeras conexiones (LED y botón) como se muestra en la Figura 4.1.

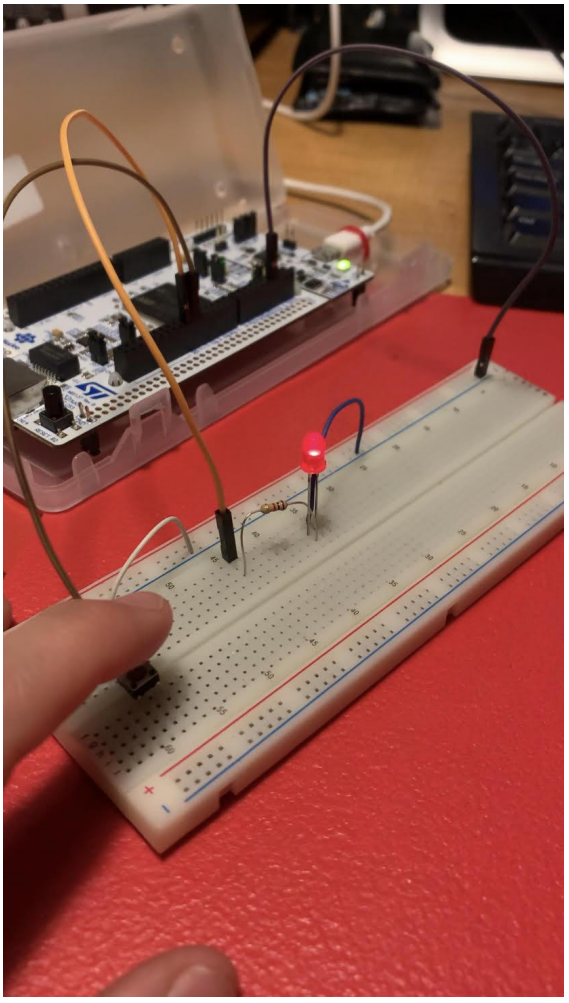


Figura 4.1: Banco de pruebas para botón y LED.

Luego se simuló la balanza con un potenciómetro como se muestra en la Figura 4.2 y el Código 4.1.

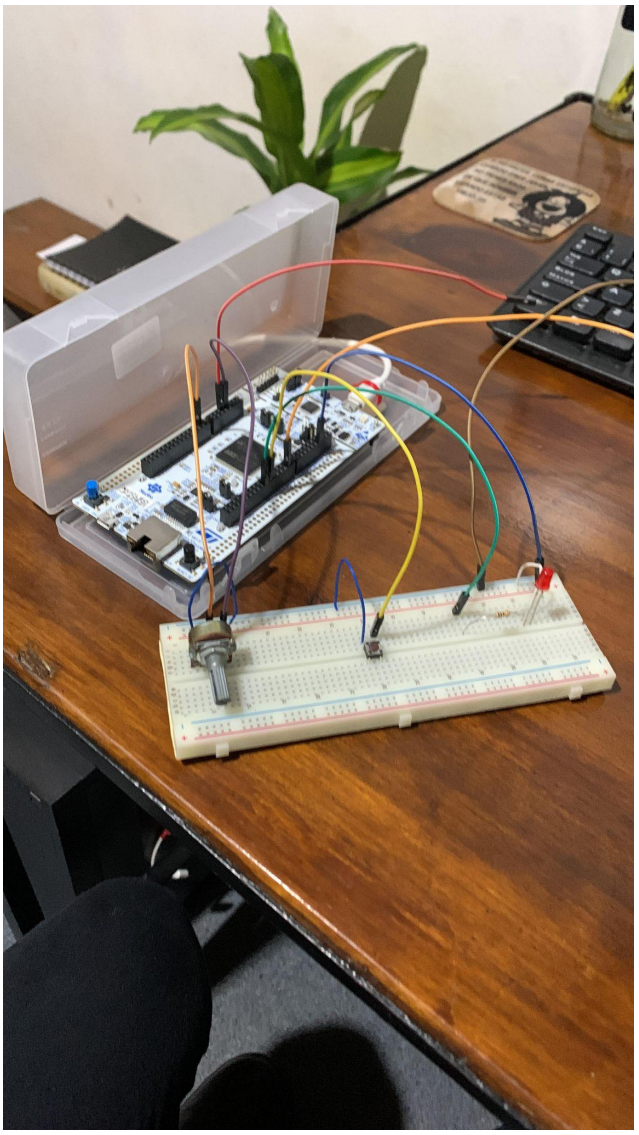


Figura 4.2: Banco de pruebas para simular la balanza con un potenciómetro.

```
65  void weightSensorUpdate()
66  {
67
68      currentWeight = potentiometer.read();
69      printf("Potentiometer Value: %.f", currentWeight);
70      if(currentWeight >= EMPTY_PLATE_WEIGHT){
71          emptyPlateLightTurn(ON);
72      }
73      else{
74          emptyPlateLightTurn(OFF);
75      }
76  }
```

Código 4.1: Prueba del potenciómetro simulando la balanza.

4.1.2 Módulo BLE HM-10

El módulo BLE se probó con el mismo banco de pruebas utilizado en la sección 4.1.1, pero esta vez escribiendo mensajes via BLE como se puede observar en el Código 4.2.

```
31          case NORMAL_MODE:
32              static void runNormalMode(){
33          case PLATE_FULL:
34              if(sensedWeight() <= emptyPlateWeight){
35                  emptyPlateLightTurn(ON);
36                  bleComStringWrite("PEM\r\n");
37                  plateState = PLATE_EMPTY;
38              }else if(isButtonPressed() == true){
39                  enablePlateFilling();
40                  emptyPlateLightTurn(OFF);
41                  bleComStringWrite("PFI\r\n");
42                  plateState = PLATE_FILLING;
43              }
44          break;
```

Código 4.2: Prueba del funcionamiento del módulo BLE HM-10.

En este caso, si el valor sensado es menor o igual que emptyPlateWeight, enciende el LED y envía el mensaje “PEM” al smartphone via BLE como se muestra en la línea 36. En caso de que el botón haya sido apretado, apaga el LED y envía el mensaje “PFI” al

smartphone via BLE (línea 41). Los resultados de esta prueba se pueden ver en la Figura 4.3.

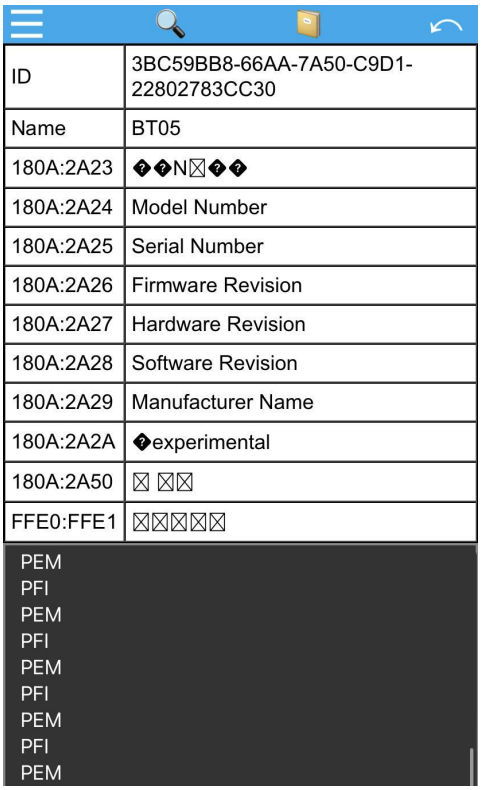


Figura 4.3: Captura de pantalla del smartphone para probar el funcionamiento de la comunicación via BLE.

4.1.3 Servomotor SG90

Este módulo se testeó de manera hardcodeada con valores ya conocidos de previo uso del mismo como se observa en el Código 4.3.

```
29  void fillPlateEnablerInit()
30  {
31      dutyCycle = 0.027;
32      servomotor.write(dutyCycle);
33      servomotor.period_ms(20);
34  }
35
36  void enablePlateFilling(){
37      dutyCycle = 0.09;
38      servomotor.write(dutyCycle);
39  }
40
41  void disablePlateFilling(){
42      dutyCycle = 0.027;
43      servomotor.write(dutyCycle);
44  }
```

Código 4.3: Prueba del servomotor con valores de *dutyCycle* conocidos.

Si se observa el datasheet del módulo, éste dice que el período de la señal PWM se debe setear en 20 ms, y el *dutyCycle* entre 0.05 % y 0.1 % (1 ms de tiempo en ON y 2 ms de tiempo en ON, respectivamente) para obtener los límites del giro del servo (-90 º y +90 º). Pero, por conocimientos de uso previo del mismo, ya se sabía que el *dutyCycle* debe variar entre 0,027 % (líneas 31 y 42) y 0,09 % (línea 37) aproximadamente para que el servo gire entre 0 º y 90 º. Luego de probar el correcto funcionamiento, dicho hardcodeo se reemplazó por sus correspondientes defines (*ENABLE_PLATE_FILLING_DUTY_CYCLE* y *DISABLE_PLATE_FILLING_DUTY_CYCLE*).

4.1.4 Celda de carga y módulo HX711

Para probar la celda de carga con su respectivo módulo HX711 fue necesario realizar un armado especial para que funcione de manera correcta. En la Figura 4.4 se presenta el primer intento del armado, el cual no fue lo suficientemente confiable ya que fallaba en repetidas ocasiones. Para resolver este problema se optó por otro armado con superficies más grandes y estables como se presenta en la Figura 4.5.

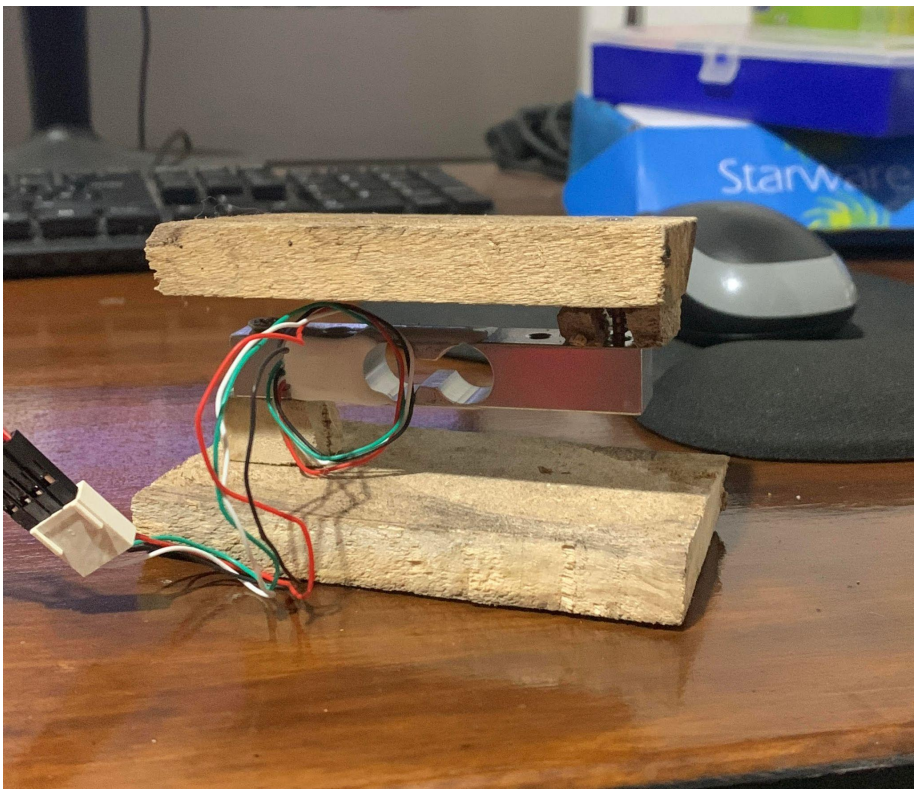


Figura 4.4: Primer armado para el sensor de peso.

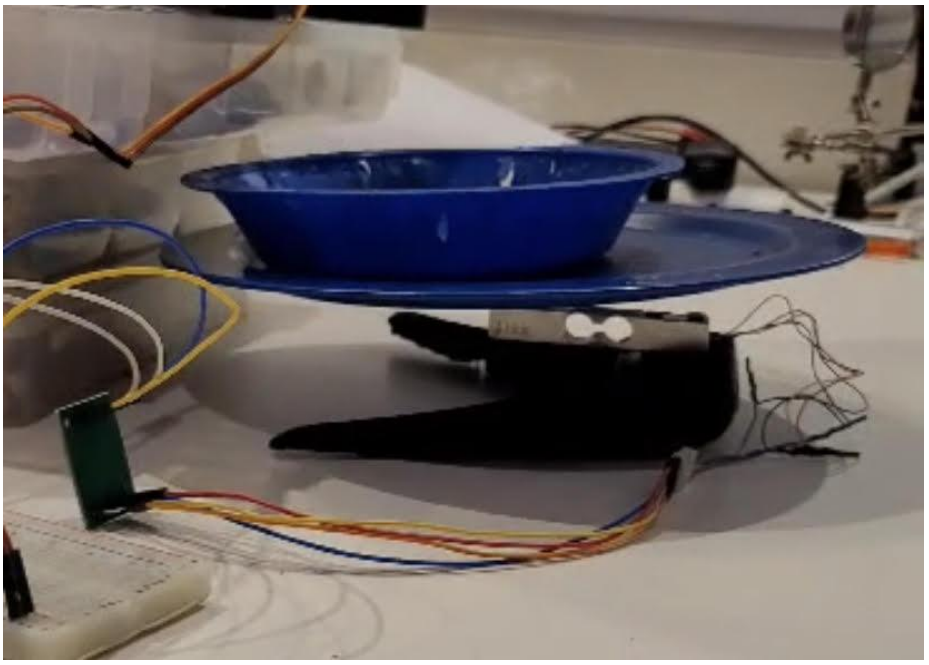


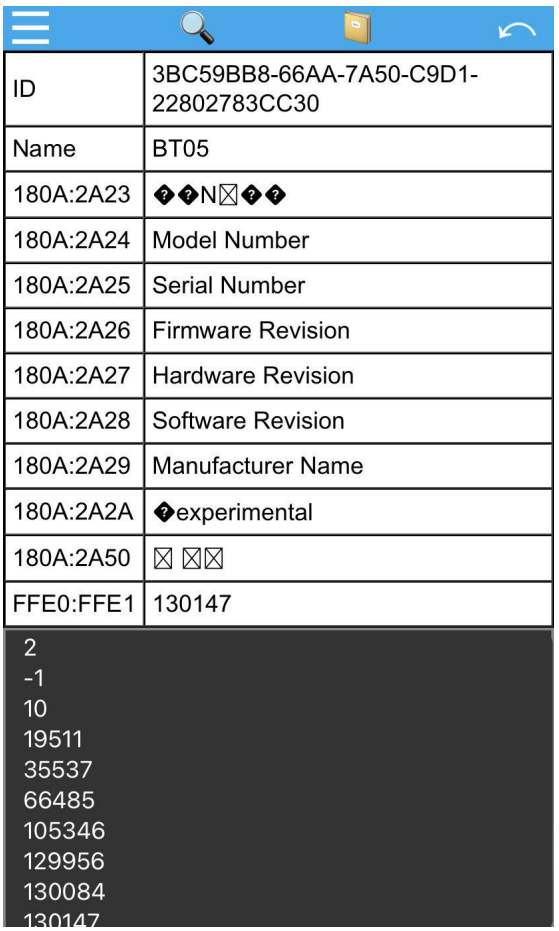
Figura 4.5: Segundo armado para el sensor de peso.

Para testear el funcionamiento se utilizó el fragmento de código mostrado en el Código 4.4.

```
50  int main()
51  {
52      char buffer[100];
53      HX711 balanza(D0, D5);
54      bleComStringWrite("Empezando");
55      long valor=0;
56      long valorTara=0;
57      float peso=0.0;
58
59      valorTara = balanza.averageValue(10);
60      while (true) {
61          delay(200);
62          valor = balanza.averageValue(10);
63          peso = (valor-valorTara);
64          sprintf(buffer, "Peso: %d", (int)peso);
65          bleComStringWrite(buffer);
66      }
67  }
```

Código 4.4: Prueba del sensor de peso.

Como puede observarse en el código, se crea un objeto balanza del tipo HX711 (línea 53) y se miden sus valores imprimiendolos en la pantalla del smartphone vía BLE (líneas 62 a 65). De esta manera se caracterizó el sensor de peso. Los valores obtenidos se presentan en la Figura 4.6.



ID	3BC59BB8-66AA-7A50-C9D1-22802783CC30
Name	BT05
180A:2A23	��N☒��
180A:2A24	Model Number
180A:2A25	Serial Number
180A:2A26	Firmware Revision
180A:2A27	Hardware Revision
180A:2A28	Software Revision
180A:2A29	Manufacturer Name
180A:2A2A	experimental
180A:2A50	☒ ☒☒
FFE0:FFE1	130147

Figura 4.6: Captura de pantalla del smartphone para caracterizar el sensor de peso.

Se optó por no escalar los valores y trabajar con los mismos. A raíz de esto se definieron los valores por default del peso del plato vacío y del plato lleno (plato vacío + porción de alimento) como se puede observar en las líneas 9 y 10 del Código 4.5.

En la línea 11 se puede observar un define llamado *FOOD_FALLING_DELAY_WEIGHT_CORRECTION* que se debe a una corrección en el peso del plato lleno generada por el tiempo que demora en caer la comida hasta el plato. Ese delay se refleja en la medición del peso y si no se tiene en cuenta, llena el plato por encima del peso requerido. Dicha corrección fue obtenida experimentalmente.

```
9   #define DEFAULT_EMPTY_PLATE_WEIGHT  34590 //
10  #define DEFAULT_FULL_PLATE_WEIGHT  130320 // (Food + plate).
11  #define FOOD_FALLING_DELAY_WEIGHT_CORRECTION 50000
```

Código 4.5: Valores por default y corrección del peso por delay.

4.2 Pruebas de integración

Se realizaron pruebas de integración de los componentes del sistema. Los resultados se pueden observar en el video de presentación del trabajo [9]. Luego de dicha integración se dejó el dispositivo funcionando durante un día para comprobar que funcione correctamente y lo hizo de manera satisfactoria.

4.3 Cumplimiento de requisitos

Los requisitos expresados en la Sección 2.1 fueron cumplidos. En la Tabla 4.1 se presentan dichos requisitos y si se cumplieron. Además, todos los casos de usos presentados en la Sección 2.2 también fueron cumplidos.

Tabla 4.1: Cumplimiento de los requisitos.

Grupo	ID	Descripción	¿Se cumplió?
1. Capacidad total	1.1	La capacidad total deberá ser de 5 L como mínimo.	Sí
2. Alimentación	2.1	El sistema debe poder alimentarse a una toma de 220 V.	Sí
3. Interfaz mediante botones	3.1	El sistema debe proveer una interfaz mediante un botón que permita el llenado del plato.	Sí
4. Interfaz mediante smartphone	4.1	El sistema debe proveer una interfaz mediante un smartphone donde el usuario pueda verificar el estado del plato y llenarlo, o configurar una nueva porción y/o un nuevo plato.	Sí
5. Modos	5.1	El sistema debe tener un modo de funcionamiento normal.	Sí
	5.2	El sistema debe tener un modo de configuración de parámetros (peso del plato y peso de una porción).	Sí
6. Conexión BLE	6.1	El sistema debe poder conectarse a la placa mediante BLE.	Sí
7. Sensor	7.1	El sistema deberá tener un sensor de peso (como una celda de carga para Arduino) con una capacidad de carga no menor a 1 Kg y una precisión menor al 1%.	Sí

8. Actuador	8.1	El sistema deberá tener un servomotor que permita el paso de el alimento al plato.	Sí
9. Fecha de finalización	9.1	El proyecto debe estar terminado el día 28 de Junio de 2023.	Sí
10. Documentos	10.1	El prototipo deberá estar acompañado de una lista de partes, un diagrama de conexiones, un repositorio del código, y una tabla indicando el cumplimiento de los requisitos y usos de caso.	Sí
11. Costo	11.1	El prototipo debe costar menos de US$ 136.	Sí

4.4 Comparación con otros sistemas similares

Luego de haber presentado el trabajo en su totalidad, con sus dificultades y soluciones, se procede a compararlo con los sistemas similares presentados en la Sección 1.2. En la Tabla 4.2 se presenta dicha comparación.

Tabla 4.2: Comparación con otros sistemas similares.

Característica	Producto 1 (Link)	Producto 2 (Link)	PetFeeder
Capacidad total	6 L	5.5 L	5 L
Alimentación	110 V / 220 V	220 V	220 V
Interfaz con el usuario	Smartphone o botonera	Botonera	Smartphone o botonera
Conexión	Wi-Fi	No	BLE
Funcionamiento	Automático. Programable por horas	Automático. Programable por horas	Remoto. El usuario da permiso para el llenado del plato
Precio	US$ 136	US$ 74	US$ 99,5

El PetFeeder se diferencia por su funcionamiento, aportando una utilidad diferente a la de los demás productos. De esta manera el usuario es el que sigue teniendo el control del

llenado del plato pero el sistema le simplifica la tarea y la cantidad de veces que tiene que proveer alimento a la mascota.

4.5 Documentación del desarrollo realizado

En la Tabla 4.3 se resumen los elementos más relevantes del trabajo y sus respectivas referencias.

Tabla 4.3: Documentación relevante del desarrollo.

Elemento	Referencia
Descripción del trabajo	Sección 1.1
Casos de uso	Tablas 2.2 a 2.5
Diagrama en bloques con componentes y tipos de conexión	Figura 3.1
Componentes, links y sus respectivos precios (US$)	Tabla 3.4
Máquina de estados principal	Figura 3.5
Máquina de estados del modo normal	Figura 3.6
Diagrama de flujo del funcionamiento en modo configuración	Figura 3.7
Diagrama de módulos del *firmware*	Figura 3.8
Pruebas de integración	Sección 4.2
Cumplimiento de los requisitos	Tabla 4.1
Comparación con otros sistemas similares	Tabla 4.2
Resultados obtenidos	Sección 5.1
Próximos pasos	Sección 5.2

CAPÍTULO 5

Conclusiones

5.1 Resultados obtenidos

Con el desarrollo del prototipo funcional del sistema se logró:

- Sensar el peso de un plato vacío.
- Sensar el peso de una porción de alimento para una mascota.
- Configurar ambos parámetros (peso del plato vacío y peso de una porción de alimento).
- Habilitar y deshabilitar el paso de alimento desde un contenedor de alimento hacia el plato hasta una cantidad determinada por el usuario.
- Permitir conectividad BLE para notificar al usuario del estado del plato y que el mismo pueda configurar los parámetros mencionados anteriormente, así como también habilitar el llenado del plato via smartphone.
- Permitir al usuario habilitar el llenado del plato mediante un botón.

5.2 Próximos pasos

Teniendo en cuenta que este desarrollo es sólamente un primer prototipo, varios aspectos se pueden mejorar.

En principio, ya que la placa NUCLEO-F429ZI no se consigue en Argentina actualmente, ésta podría ser reemplazada por una NUCLEO-F401 que, además, es más económica. Sin embargo, pensando en producir el producto en escalas mayores, se deberían buscar otras alternativas que disminuyan aún más el costo del producto.

Otro paso sería diseñar un PCB especialmente para el sistema donde queden todos los componentes soldados. También se puede modificar el diseño 3D de manera que contemple el espacio que ocupa la placa y quede dentro del mismo, así como también agregar la parte del contenedor impresa en 3D. Es probable que, si se produce en escalas mayores, sea conveniente un inyectado de plástico en vez de una impresión 3D.

Se pueden agregar muchas funcionalidades al trabajo ya que su escalabilidad es muy grande. Algunos ejemplos son:

- Conexión Wi-Fi.

- App para smartphone.
- Cámara.
- Sensor de movimiento.
- Luz.
- Llenado automático programable.
- Comandos por voz.
- Display.
- Notificaciones al usuario sobre la cantidad de alimento consumida por el animal en un período de tiempo determinado. Análisis de dichos datos.

También podría agregarse (o por separado) un dispositivo similar pero para el agua que consume la mascota.

Bibliografía

[1] Ulises Montenegro (2023). Repositorio del *PetFeeder.* [En línea]. Disponible en: https://github.com/UlisesMontenegro/PetFeeder

[2] Arm. Mbed. Placa NUCLEO-F429ZI. [En linea]. Disponible en: https://os.mbed.com/platforms/ST-Nucleo-F429ZI/

[3] Naylamp Mechatronics. Tutorial transmisor de celda de carga HX711, balanza digital. [En línea]. Disponible en: https://naylampmechatronics.com/blog/25_tutorial-trasmisor-de-celda-de-carga-hx711-balanza-digital.html

[4] Servomotor SG90 datasheet. [En línea]. Disponible en: http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf

[5] Alldatasheet.com. HM-10 datasheet. [En línea]. Disponible en: https://html.alldatasheet.com/html-pdf/1179058/ETC1/HM-10/109/1/HM-10.html

[6] Arm. Página web de Mbed. [En línea]. Disponible en: https://os.mbed.com/

[7] Ariel Lutenberg. Pablo Gomez. Eric Pernia. A Beginner's Guide to Designing Embedded System Applications on Arm Cortex-M Microcontrollers. [En línea]. Disponible en: https://www.arm.com/resources/education/books/designing-embedded-systems

[8] Adrien Corvasier. Eric Bobillier. HX711 - Library which can be used with the HX711 component. [En línea]. Disponible en: https://os.mbed.com/users/Volt72/code/HX711/

[9] Ulises Montenegro (2023). PetFeeder - Sistema de alimentación remota de mascotas. [En línea]. Disponible en: https://www.youtube.com/watch?v=UbiIuWud2UA