



UNIVERSIDAD DE BUENOS AIRES

Facultad de Ingeniería

86.65 Sistemas Embebidos

Memoria del Trabajo Final:

Automatización de tranquera en una manga

Autor:

Sr. Augusto Villacampa Horta

Legajo: 102.602

*Este trabajo fue realizado en la Ciudad Autónoma de Buenos Aires,
entre abril y julio de 2024.*

RESUMEN

En este trabajo se realiza el diseño e implementación de una sistema de control automático para tranqueras en una manga. Este sistema pretende simplificar y agilizar trabajos de aparte en una manga, permitiendo realizar apartes de forma automática en base a algún parámetro de los animales, haciendo uso de una base de datos. A través de lecturas de caravanas electrónicas por RFID y con comunicación con una app de celular, este trabajo presenta una solución práctica para una tarea común, exigente, y a veces peligrosa.

En esta memoria se presenta la motivación del proyecto, los diseños de las distintas partes, y se proponen futuras mejoras y ampliaciones.

Índice General

Registro de versiones	5
Introducción general y motivación	6
1.1 Necesidad detectada y fundamentación de la observación	6
1.2 Solución a implementar	10
1.3 Factores que justifican la implementación y usabilidad	13
1.4 Soluciones similares	16
1.5 Implementación simplificada en este trabajo	16
Introducción específica	18
2.1 Requisitos	18
2.2 Casos de uso	20
2.3 Descripción módulos del sistema	21
2.3.1 Alimentación	21
2.3.2 Microcontrolador	22
2.3.3 Lectura RFID	23
2.3.4 Comunicación Wi-Fi	23
2.3.5 Controles e indicadores	25
2.3.6 Aplicación de celular	26
2.3.7 Conexionado	26
Diseño e implementación	27
3.1 Diseño del Hardware	27
3.1.1 Diseño del hardware de alimentación	28
3.1.2 Diseño del hardware del módulo RFID	29
3.1.3 Diseño del hardware del módulo Wi-Fi/MQTT	30
3.1.4 Diseño del hardware de los controles e indicadores	31
3.1.5 Diseño del hardware del microcontrolador	35
3.2 Diseño del Firmware	36
3.2.1 Firmware de la placa NUCLEO-F429ZI	36
3.2.1.1 Módulo led	39
3.2.1.2 Módulo tranquera	39
3.2.1.3 Módulo scale	39
3.2.1.4 Módulo mode_selector	40
3.2.1.5 Módulo non_blocking_delay	40
3.2.1.6 Módulo MQTT	41
3.2.1.7 Módulos MFRC522 y rfid	42
3.2.1.8 Módulo system	43
3.2.2 Firmware de la placa ESP32 DEVKIT V1	46
3.3 Diseño de la aplicación de celular	49

3.4 Broker MQTT	55
Ensayos y resultados	56
4.1 Desarrollo y pruebas de funcionamiento	56
4.2 Cumplimiento de requisitos	58
Conclusiones	59
5.1 Resultados obtenidos	59
5.2 Próximos pasos	60
Bibliografía	62

Registro de versiones

Revisión	Cambios realizados	Fecha
1.0	Creación del documento	8/7/2024
1.1	Agregado de sección: 1.4 Soluciones similares Agregado de sección: 4.1 Desarrollo y pruebas de funcionamiento	29/7/2024

CAPÍTULO 1

Introducción general y motivación

1.1 Necesidad detectada y fundamentación de la observación

Dentro del mundo agropecuario existe un gran potencial de aplicación tecnológica en distintos sectores, esto hace que en el rubro existan muchas necesidades a la espera de ser satisfechas, a la vez que con el avance de la tecnología aparecen nuevas aplicaciones potenciales en este campo.

Dentro del enorme sector agropecuario, el lado de la agricultura ha ido incorporando muchas soluciones tecnológicas con el paso del tiempo. Sin embargo, en comparación, el lado ganadero ha quedado atrás en este sentido. Es por esta razón que en el sector ganadero pueden encontrarse muchas más necesidades por satisfacer.

El sector ganadero o pecuario de nuestro país es sumamente importante, tanto a nivel nacional como internacional, debido a que se obtienen una gran cantidad de productos consumidos a gran escala, principalmente carne y leche.

A partir de los datos obtenidos del último Censo Nacional Agropecuario (CNA) realizado por el Instituto Nacional de Estadísticas y Censos (INDEC) en 2018 [1] se puede apreciar la importancia de este mercado en nuestro país, teniendo datos sobre el número de cabezas de ganado o la cantidad de Explotaciones Agropecuarias (EAP), entre muchos otros.

El CNA indica que se registraron 40.023.083 cabezas de ganado bovino en 130.929 EAP, representando aproximadamente el 71.9% de las cabezas de ganado de todo el país. El resto se distribuye entre cabezas de ganado ovino, caprino, porcino, equino y otros. También indica que en las EAP trabajaron en forma permanente 418.058 personas.

Por otro lado, en los puestos de trabajo rurales solo una pequeña parte de los trabajadores se encuentran debidamente registrados en el Registro Nacional de Trabajadores Rurales y Empleadores (RENATRE). En 2020, una nota de la página "Bichos de Campo" [2] indicó a partir de estimaciones del INDEC y del RENATRE que el índice de registro de estos trabajadores es cercano al 25 %.

En 2021, el Ingeniero Horacio Giberti realizó un análisis sobre los datos preliminares del CNA 2018 [3] comparando los datos con el anterior CNA, realizado en 2002. En esta

publicación se muestra como la población residente en las EAP (no necesariamente trabajadores) disminuyó, siguiendo la tendencia global, un 42 %.

A partir de esta información se infiere que, si bien el número de trabajadores registrados en el RENATRE puede no ser representativo del número real de trabajadores, un gran porcentaje de los trabajadores no registrados son residentes en las EAP. La reducción del número de residentes impacta entonces directamente en el número de trabajadores rurales.

Este hecho se corroboró al preguntar a productores y trabajadores rurales, quienes dijeron observar de primera mano la falta de personas dispuestas a realizar trabajos rurales.

El CNA también ofrece la distribución de cabezas bovinas y EAP según la orientación productiva. Esta información puede verse resumida en la figura 1.1, extraída directamente del CNA.

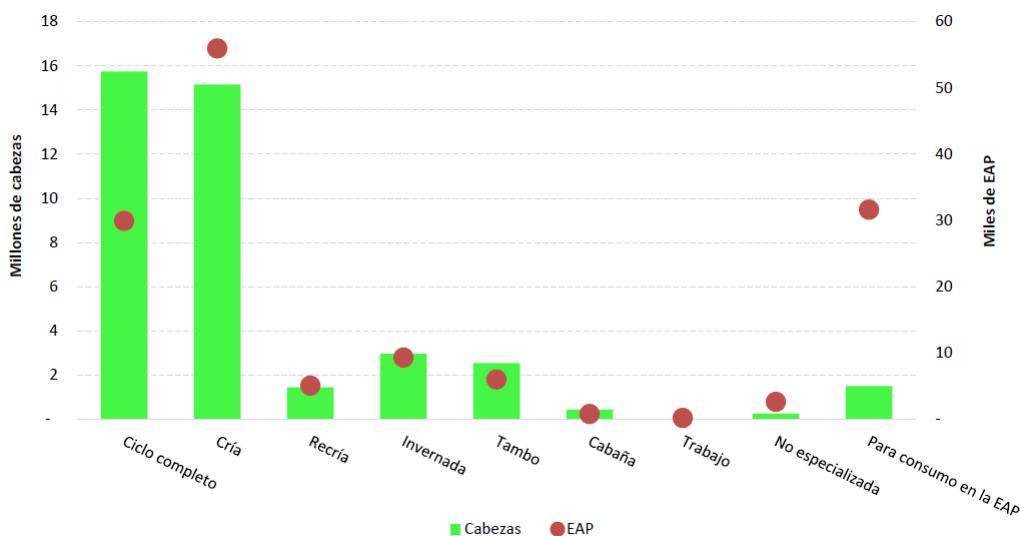


Figura 1.1: EAP y cabezas de ganado bovino según orientación productiva del ganado. Total del país. Fuente: Censo Nacional Agropecuario 2018 [1].

Se observa que la gran mayoría de las cabezas de ganado bovino pertenecen a la orientación productiva de ciclo completo o cría, pero es importante aclarar que el ciclo completo es el que incluye cría, recría e invernada.

Por otro lado se destacan el tambo y la cabaña, que se diferencian del resto de orientaciones productivas por llevarse a cabo en instalaciones que (en general) tienen diferentes tecnologías aplicadas, como podrían ser sistemas de ordeñe y alimentación automáticos o aplicaciones sobre la inseminación.

Aunque en cualquier ámbito es común la aplicaciones de soluciones (tecnológicas o no) para la mejora del rendimiento, en estas últimas orientaciones se avanzó con la tecnología más que en otras por lo antes mencionado, que las actividades suelen llevarse a cabo en instalaciones fijas. La centralización de los trabajos hizo que con el tiempo las instalaciones contaran con conexión a la red eléctrica e incluso a internet, haciendo más fácil el desarrollo y la aplicación de avances tecnológicos que facilitan los trabajos o mejoran el rendimiento.

Este hecho no se da en las otras orientaciones productivas, es decir en el ciclo completo y sus partes, donde las diferentes actividades habituales con el ganado se realizan en varios lugares, por ejemplo en distintos corrales para que los animales se alimenten de las pasturas.

No obstante hay un lugar común a todas estas actividades, y usado recurrentemente por productores, veterinarios y otros trabajos: *la manga*. En la figura 1.2 se puede ver un ejemplo de una manga de 9 metros de largo con cepo.



Figura 1.2: Manga de 9 metros con cepo para ganado bovino. Fuente: agroads.com

También se suele llamar manga al lugar en donde está emplazada la manga propiamente dicha, y por lo tanto también se incluye en la mención a los corrales y otras cosas que pueda tener a su alrededor, y que se utilizan junto con la manga. En la figura 1.3 se muestra el plano de la manga del establecimiento "Don Miguel".

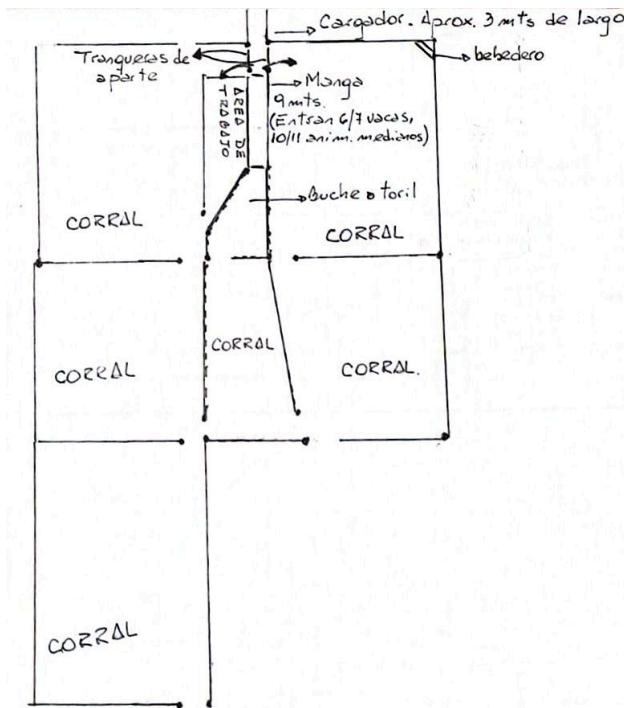


Figura 1.3: Plano de la manga del establecimiento "Don Miguel". Fuente: Proporcionado por el propietario del establecimiento.

Esta manga cuenta con 6 corrales, más el buche o toril. A la salida de la manga propiamente dicha se encuentra el cepo. A continuación el animal tiene 3 caminos posibles, hacia alguno de los corrales laterales o hacia el cargador, que se utiliza para cargar animales a un camión. Los trabajadores guían a cada animal maniobrando las tranqueras de los corrales laterales. Esta manga, al poseer corrales a ambos lados que se interconectan con todos los demás, permite a los trabajadores guiar al ganado cómodamente, pudiendo incluso un animal pasar varias veces por la manga.

Para la realización de las tareas típicas el proceso es el siguiente: primero los trabajadores traen al ganado con el que se va a trabajar. El rodeo de animales puede ser de tamaños diversos, desde unas pocas unidades o decenas de animales, hasta varios cientos, o incluso más. Cuando se trata de rodeos grandes, para la mayoría de las tareas se seleccionan grupos pequeños de animales, se realiza el trabajo, y luego se pasa al siguiente.

Por ejemplo, si se necesitara apartar un rodeo de 300 animales entre machos y hembras se lo haría en grupos de alrededor de 50 animales. Para cada grupo habría al menos una persona encargada de abrir o cerrar tranqueras, mientras que otras personas separarían a cada macho y lo guiarían para que pase por la manga; y separados todos los machos se guiaría a las hembras a través la manga hacia otro corral, para luego seguir con el siguiente grupo. Terminada la tarea los trabajadores llevarían a los animales a sus

respectivos corrales. Un trabajo como este podría durar una hora y media o incluso más de 2 horas sólo en la manga, sin contar el tiempo que se necesite para mover a los animales desde o hacia la manga.

A partir de este ejemplo queda en evidencia el tiempo y esfuerzo que se requiere para este tipo de tareas, además de que se debe contar con un número mínimo de personas para que se pueda realizar el trabajo. El tiempo empleado es, por supuesto, directamente proporcional a la cantidad de animales con los que se trabaje, además de la tarea que se esté realizando. Para trabajos complejos o con grandes rodeos se podría llegar a trabajar toda la jornada.

También es importante aclarar que todas las mangas son distintas. Las mangas propiamente dichas podrían variar en longitud o en el material del que están hechas, pero las mayores diferencias se ven en lo que hay alrededor. A diferencia de la manga que se usó como ejemplo, podría tenerse a la salida del cepo solo una tranquera, que permita el acceso a 2 corrales dependiendo de su posición.

También podría haber más o menos corrales, o que no estén todos interconectados. Estos factores hacen que para los trabajadores pueda ser más o menos cómodo trabajar en determinadas mangas. En algunos establecimientos de cierto tamaño también es posible tener más de una manga, para distintos tipos de trabajos.

Otro factor común a la gran mayoría de mangas es que no cuentan con conexión a la red eléctrica, o a internet. A veces no se tiene tampoco cobertura de telefonía celular. Por este motivo todas las tareas se realizan sin la utilización de maquinarias que ayuden a los trabajadores.

Si a esta dificultad en los trabajos se le agrega la reducción de trabajadores disponibles, se puede observar la necesidad de desarrollar una solución tecnológica que reduzca la cantidad de personas necesarias, el tiempo empleado o facilite las tareas.

1.2 Solución a implementar

A partir de las necesidades detectadas y explicadas se plantea una solución que ayude a los trabajadores y productores en varios aspectos.

Para reducir el trabajo necesario en las distintas tareas de la manga se plantea automatizar las tranqueras (o la tranquera) a la salida del cepo, para que se abran hacia uno u otro lado en función de algún criterio que decida el productor.

Para la aplicación de esta automatización es necesario poder identificar a cada animal individualmente, y así poder analizar el criterio que se está usando para la selección. Para esta identificación se propone usar caravanas electrónicas.

Estas caravanas electrónicas funcionan con tecnología RFID (Radio Frequency Identification), en particular con etiquetas pasivas en las caravanas. Es decir que cuando un lector interactúa con la caravana, se induce en ésta la corriente necesaria para transmitir la respuesta hacia el lector. Este tipo de etiquetas solo contienen un número en su memoria, normalmente de 16 dígitos, y que se transmite durante la lectura. Cada caravana electrónica posee un número distinto en su memoria, por lo que cada animal podría identificarse sin problemas. Además estas caravanas son reutilizables. En la figura 1.4 se muestra una caravana electrónica.



Figura 1.4: Caravana electrónica producida por Allflex.

Por otro lado, el Servicio Nacional de Sanidad y Calidad Agroalimentaria (SENASA) exige a los productores el uso de una caravana en cada animal, y que es obligatoria al momento de la venta de animales. Esta caravana suele ser de plástico, y posee un código único impreso que contiene información del partido, el establecimiento y el número interno del animal. Las caravanas se utilizan con fines de trazabilidad, y las otorga SENASA a los productores a través de un veterinario.

Sabiendo entonces que cada animal posee un código único, se propone crear un sistema de identificación y gestión de los animales que asocie el código de la caravana exigida por SENASA con el código interno de la caravana electrónica.

En un software pueden cargarse distintos parámetros correspondientes al animal, a los que se accede de forma manual en la base de datos o de forma automática al leer una caravana electrónica. Estos parámetros pueden luego usarse para el manejo de la tranquera automática.

Dentro de todos los datos que contenga el sistema habrá varios que serán comunes a todos los animales, como lo son fecha de nacimiento o ingreso al establecimiento, raza, género, peso, etc. Mientras que otros parámetros dependerán del animal del que se trate, por ejemplo se podría querer guardar la información de las fechas de inseminación para

vacas de cría. Se podrían tener entonces distintos conjuntos de datos del animal dependiendo de la edad, orientación productiva, sexo, o algún otro factor.

Otra posibilidad que tendrá el productor es usar una balanza portátil, que es especial para su uso en mangas, y permitiría actualizar el peso de los animales en el sistema. Tener valores actualizados de peso permitiría clasificar a los animales según este parámetro.

El modelo común de balanza para mangas consiste en dos barras sobre las que se coloca un tablón, en el que se para el animal para ser pesado. Estas barras se conectan a un monitor que muestra la medición. La balanza puede ser colocada para pesar al animal que se encuentra en el cepo o en una jaula especial para pesar, dependiendo de la preferencia del productor. Estas balanzas tienen, por lo general, una salida por puerto serie para extraer las mediciones, que se usará para cargar estos datos automáticamente al sistema. En la figura 1.5 se muestra una balanza para mangas.



Figura 1.5: Balanza para mangas producida por Distribal.

En cuanto a la lectura de las caravanas, existen 2 tipos de lectores o antenas. Por un lado las de tipo bastón, con un alcance corto y manejadas directamente por una persona. Por otro lado están los lectores tipo panel antena, que son fijas pero poseen un mayor alcance. Este último tipo puede usarse en la manga, para sensar automáticamente al animal que se encuentre en el cepo. El lector tipo bastón también puede utilizarse para sensar animales que no estén en el cepo. En el caso de este trabajo, se propone utilizar lectores de tipo panel antena, de modo de sensar únicamente al animal que se encuentre en el cepo. En la figura 1.6 se muestra un lector RFID de tipo panel antena.



Figura 1.6: Lector de caravanas electrónicas de tipo panel antena producido por Tru-Test.

1.3 Factores que justifican la implementación y usabilidad

La implementación de este sistema en un establecimiento trae enormes y variados beneficios para los productores, como la mejora del rendimiento de la producción, la simplificación de trabajos o la disminución de los trabajadores necesarios, mejorando tiempos y costos.

Un factor muy importante que posee el producto es que los beneficios que aporta a los trabajadores no invaden las costumbres y tradiciones que existen en este rubro, además de que la instalación y uso de componentes físicos es reducida, compuesta principalmente por la tranquera con su motor y alimentación, además de las caravanas y los lectores. Esto hace que disminuya el rechazo que podría surgir por parte de los trabajadores o los productores ante la implementación de este sistema. El sector agropecuario es, por tradición, un sector que comprende a mucha gente con costumbres y formas de trabajar muy arraigadas, que suelen tender a hacerse de forma manual y sin la intervención de tecnologías que reemplacen las tareas; y es por este motivo que el sistema pretende ser complementario y no reemplazar por completo a los trabajadores o los métodos usados. Cumplir este requisito amplía la aceptación y el mercado que puede tener el sistema.

Un ejemplo se vería a la hora de realizar un tacto a vacas presuntamente preñadas. A la hora de realizar esta tarea un veterinario realiza el tacto y marca a las vacas que resultan no estar preñadas. Luego otra persona realiza una marca más permanente, como el corte de los pelos de la cola, para poder diferenciar al animal. Hecho esto el productor podría apartar los dos grupos o no, ya que la marca fue aplicada. En cambio si se contara con el sistema propuesto sólo se necesitaría de un operador que cargue los datos al

sistema del estado de preñez, el tiempo estimado desde la preñez, y alguna otra observación que haga el veterinario. Una vez cargados los datos se podría apartar fácilmente por cualquiera de estos parámetros, pero lo importante es señalar que esto no impide seguir realizando las marcas tradicionales, y solo aporta beneficios. Los datos antes mencionados podrían anotarse en una libreta si no se contara con este sistema, sin embargo esta es una tarea que comúnmente no se realiza.

Para los trabajadores, la utilización de este sistema para apartar animales reduciría el excesivo esfuerzo para realizar algunas tareas, además de aportar una mejora en la seguridad de las personas. Algunas de las tareas comunes, sobre todo con rodeos grandes de animales, requieren por un lado subdividir en grupos pequeños con los que trabajar, y por otro un contacto directo con los animales en un ambiente en el que podrían sentirse estresados y tener comportamientos violentos.

Un ejemplo de esto se dio anteriormente cuando se explicó el proceso para apartar un grupo grande de animales entre machos y hembras en la sección 1.1. En cambio al usar este sistema podrían enviarse a todos los animales a través de la manga de una vez, ya que el sistema se encargaría de clasificarlos. De este modo se reduciría en gran medida el tiempo empleado, sobre todo en grupos grandes de animales, y se evitaría que los trabajadores deban recorrer los corrales para apartar manualmente, mejorando la seguridad y reduciendo el esfuerzo necesario.

Este tipo de tareas necesita ahora menor cantidad de personal, lo cual beneficia al productor en cuanto a la cantidad de gente que debe contratar, cubriendo el problema de la falta de trabajadores disponibles.

La implementación del sistema aporta también al administrador del establecimiento (que podría o no ser el dueño productor) una herramienta para gestionar los animales, teniendo acceso a una gran variedad de datos de cada animal que le permiten planificar y realizar tareas de forma más eficiente.

Por ejemplo, podría tenerse un registro con las fechas de parto de cada vaca, y usar luego estos datos para seleccionar los animales que se quiera inseminar. De esta forma se evitaría inseminar vacas que, por haber parido hace poco tiempo, no puedan quedar preñadas nuevamente, ahorrando dinero y esfuerzo. Contar con gran cantidad de datos o estadísticas le daría al administrador herramientas para mejorar la toma de decisiones que podrían aumentar el rendimiento del establecimiento.

Almacenar los datos en la nube permite también que se pueda acceder a ellos en cualquier momento o lugar. Si se diera el caso de que el dueño del establecimiento no fuera también el administrador, acceder a estos datos le permitiría realizar un

seguimiento de los trabajos realizados por sus empleados, llevando a un mejor control de la producción en su establecimiento.

Usar este sistema permite también realizar tareas fuera de la manga, ya que se podría contar con un lector de tipo bastón. Se podría entonces, por ejemplo, realizar una lectura de la caravana de un animal que esté caído y así proporcionarle al veterinario todos sus datos, para que pueda realizar un mejor diagnóstico. Otro uso sería al recorrer corrales con vacas en época de parto, para cargar en el sistema la fecha en la que se encuentre una vaca que haya parido hace horas o unos pocos días, y así llevar un control de partos individual de cada vaca. Estos ejemplos y otros también podrían realizarse sin usar el sensor tipo bastón, ya que sería posible leer el código de la caravana exigida por SENASA y buscar manualmente al animal en el sistema, esto da a los trabajadores cierta libertad para trabajar como les resulte más cómodo.

Otra solución que provee este sistema es a la pérdida de caravanas. Si bien no es lo más común que un animal pierda su caravana, es algo que puede pasar. Contar con este sistema hace que al perder una de las caravanas se pueda identificar al animal usando la otra, y de esta forma reemplazar la faltante. En caso de perder la de SENASA, el código está en el sistema, por lo que podría recuperarse leyendo la caravana electrónica. Y en caso de perder la caravana electrónica se podría buscar al animal ingresando manualmente el código de la caravana de SENASA, y luego colocarle una nueva caravana electrónica. La pérdida de ambas caravanas sería sumamente improbable, pero si se diera el caso se podría identificar al animal sensando a los demás y hallando en el sistema, por descarte, al que perdió las caravanas.

Se debe mencionar que el uso de la caravana electrónica no representaría ningún inconveniente para el productor en cuanto a la cantidad de caravanas que puede llegar a tener el animal a la vez. En el peor de los casos el animal podría llegar a contar con: la caravana tipo botón de SENASA en la oreja derecha, la caravana electrónica, una caravana tipo tarjeta para uso interno, y una caravana insecticida. Distribuyendo correctamente las caravanas no se tendría ningún problema en la habitual realización de las tareas.

Finalmente, otro factor importante que justifica la implementación de este sistema es que en otras orientaciones productivas (especialmente en el tambo) estas prácticas ya se realizan. Por ejemplo en los tambos se suelen utilizar las caravanas electrónicas y un software de gestión para guardar los datos de rendimiento de la leche que da cada vaca, la alimentación, los partos, la inseminación, etc. Como se mencionó anteriormente, en estas orientaciones productivas la tecnología avanzó más rápidamente, pero el hecho de que prácticas similares ya se lleven a cabo demuestra que los beneficios son reales y el uso es viable en la práctica, aumentando la aceptación que podría recibir este sistema en el área en la que se pretende aplicar.

1.4 Soluciones similares

No se encontró en el mercado ningún producto similar, aunque sí existen algunos que resuelven parcialmente algunas de las necesidades planteadas en este trabajo.

Por un lado se puede mencionar a la marca Farmquip [4], que ofrece una gran variedad de productos para el sector ganadero. Dentro de las soluciones ofrecidas, existe un brete automático desarrollado en asociación con Cattler [5] que también propone realizar apartes de forma automática. Sin embargo, en este caso se requiere la instalación de un brete completamente nuevo. En cambio en este trabajo se propone que la solución se instale sobre la manga existente para reducir el costo.

También existen distintos sistemas de gestión de ganado, pero se trata simplemente del software donde se almacenan los datos. Estos sistemas solo sirven para realizar un seguimiento y control del ganado, pero sin una aplicación como la de aparte automático propuesta en este trabajo.

1.5 Implementación simplificada en este trabajo

Para el desarrollo de este trabajo se propone, en base a las secciones anteriores, la implementación de una versión simplificada de la solución antes detallada. En esta versión simplificada se busca realizar el diseño central del sistema, especialmente la lectura mediante RFID, la comunicación con un dispositivo que permita al trabajador operar el sistema, y el control automático a partir de las lecturas. De esta forma, se pretende obtener un prototipo del sistema que cuente con las funcionalidades esenciales y que sirva como prueba de concepto del producto final.

Además de lo antes mencionado, se propone agregar algunos controles básicos, como el control manual de la tranquera, algunos indicadores del funcionamiento, y la posibilidad de simular un aparte bajo algún parámetro. En este sentido, se propone simular una balanza de modo de poder apartar según el peso de los animales. Para el manejo del sistema se propone la implementación de una aplicación de celular sencilla, que cuente además con una base de datos para cargar animales. En la figura 1.7 se muestra un diagrama del sistema propuesto.

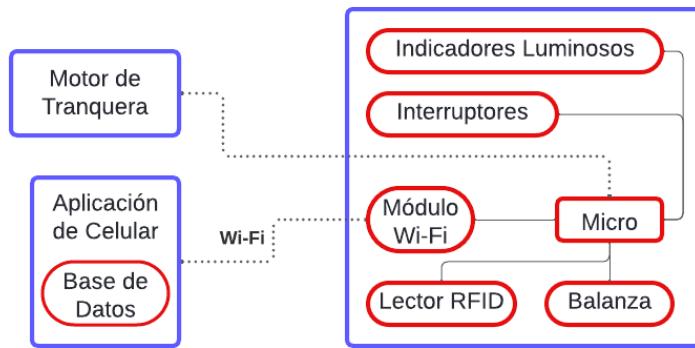


Figura 1.7: Diagrama conceptual del sistema propuesto.

CAPÍTULO 2

Introducción específica

2.1 Requisitos

A fin de obtener un sistema que cumpla con las expectativas detalladas en la sección 1.4 se define una lista de requisitos para el sistema a implementar. Esta lista se presenta en la tabla 2.1.

Tabla 2.1: Requisitos del sistema a implementar

Grupo	ID.	Descripción
Identificación	1.1	El sistema realizará la lectura de las caravanas electrónicas a través de RFID
Comunicación	2.1	El sistema se comunicará mediante Wi-Fi con la aplicación de celular
	2.2	El sistema deberá enviar a la aplicación la información de lectura
	2.3	El sistema deberá comunicarle a la aplicación el estado de la tranquera
	2.4	El sistema deberá comunicarle a la aplicación si se encuentra forzado en modo Solo Lectura
	2.5	El sistema deberá comunicarle a la aplicación las mediciones de peso
Aplicación	3.1	La aplicación tendrá la base de datos de los animales
	3.2	La aplicación deberá permitirle al usuario configurar el modo de uso
	3.3	La aplicación deberá poder mostrar la información del animal que el sistema haya leído
	3.4	La aplicación deberá poder controlar manualmente el estado de la tranquera
	3.5	La aplicación deberá permitirle al usuario configurar el

parámetro para el aparte

- 3.6 La aplicación deberá poder modificar la información del peso de un animal
- 3.7 La aplicación deberá poder crear o eliminar animales de la base de datos
- 3.8 La aplicación deberá poder mostrar la información de cualquier animal de la base de datos
- 3.9 La aplicación deberá mostrarle al usuario el modo actual de funcionamiento

Aparte	4.1	El sistema deberá controlar el estado de la tranquera cuando se esté en modo Aparte, de forma de separar los animales censados bajo el parámetro configurado
Botones	5.1	El sistema contará con botones para abrir o cerrar la tranquera de forma manual
	5.2	El sistema contará con un interruptor para forzar el modo Solo Lectura
	5.3	El sistema contará con un interruptor para encender o apagar la balanza
Indicadores	6.1	El sistema contará con indicadores luminosos para identificar el modo de funcionamiento
	6.2	El sistema contará con un indicador luminoso que se encenderá por 5 segundos cuando se lea una caravana
	6.3	El sistema contará con un indicador luminoso que muestre el estado de la conexión Wi-Fi

2.2 Casos de uso

En las tablas 2.2, 2.3 y 2.4 se presentan tres casos de uso para ejemplificar un uso común del sistema.

Tabla 2.2: El usuario quiere realizar un aparte por peso mayor a 100 kg

Elemento	Definición
Disparador	Se quiere realizar un aparte
Precondiciones	<ul style="list-style-type: none"> • El sistema está encendido • La aplicación está conectada al sistema • El sistema no está forzado a Solo Lectura
Flujo principal	El usuario elige en la aplicación el modo Aparte y selecciona como parámetro el peso y mayor a 100kg. Luego el usuario debe conducir a los animales por la manga para realizar el aparte.
Flujos alternativos	<ul style="list-style-type: none"> A. Se pierde la conexión Wi-Fi. El sistema no puede continuar con el aparte automático. Se indica que se perdió la conexión B. Se fuerza el modo Solo Lectura. El aparte no puede continuar porque el sistema ignora los comandos de la aplicación C. Control manual de la tranquera. Si se decide que un animal no corresponde al grupo que la aplicación determina se puede cambiar el estado de la tranquera manualmente. Luego el aparte puede continuar

Tabla 2.3: El usuario quiere monitorear animales

Elemento	Definición
Disparador	Se quiere realizar un monitoreo sin aparte
Precondiciones	<ul style="list-style-type: none"> • El sistema está encendido • La aplicación está conectada al sistema
Flujo principal	El usuario elige en la aplicación el modo Solo Lectura. Luego el usuario debe conducir a los animales por la manga para realizar el monitoreo.
Flujos alternativos	<ul style="list-style-type: none"> A. Se pierde la conexión Wi-Fi. El sistema no puede continuar con el monitoreo. Se indica que se perdió la conexión B. Se fuerza el modo Solo Lectura. El monitoreo puede

- continuar con normalidad
- C. Monitoreo de peso. Si se enciende la balanza se puede monitorear también la medición del peso.
 - D. El usuario puede controlar manualmente la tranquera para realizar un aparte manual si así lo desea

Tabla 2.4: El usuario quiere realizar un pesaje

Elemento	Definición
Disparador	Se quiere realizar un pesaje
Precondiciones	<ul style="list-style-type: none"> • El sistema está encendido • La aplicación está conectada al sistema
Flujo principal	El usuario elige en la aplicación el modo Solo Lectura. Luego el usuario debe conducir a los animales por la manga para realizar el pesaje de cada uno. Mientras el animal está en la balanza el usuario debe actualizar el valor del peso
Flujos alternativos	<ul style="list-style-type: none"> A. Se pierde la conexión Wi-Fi. El sistema no puede continuar con el pesaje. Se indica que se perdió la conexión B. Se fuerza el modo Solo Lectura. El pesaje puede continuar con normalidad C. Se apaga la balanza. El pesaje no puede continuar D. El usuario puede controlar manualmente la tranquera para realizar un aparte manual si así lo desea

2.3 Descripción módulos del sistema

2.3.1 Alimentación

Para la alimentación general del sistema se cuenta con una fuente transformadora de 220 VAC a 5 VDC con capacidad para entregar hasta 2500 mA, cuyo modelo es FU-5V2500, fabricado por Prontex.

Con esta fuente se alimenta por un lado el microcontrolador y el módulo Wi-Fi, y a su vez una fuente de modelo Mb-102, capaz de entregar a la salida 5 V o 3,3 V seleccionables, y hasta 700 mA. Con esta última se alimenta el módulo lector de RFID, los indicadores luminosos, y se usa también para la conexión de los botones, ya que su diseño es especial para su uso en protoboards. En la figura 2.1 se muestra la fuente Mb-102.



Figura 2.1: Fuente de alimentación Mb-102.

2.3.2 Microcontrolador

Como controlador principal del sistema se utiliza la placa NUCLEO-F429ZI. La elección de esta placa recayó exclusivamente en la disponibilidad, teniendo además como requerimiento la cantidad de memoria, pines y periféricos de la placa. Dado que la placa cuenta con todo lo necesario, no hubo inconvenientes en la utilización de esta placa.

Para la programación se utilizó la plataforma *Keil Studio Cloud* y el lenguaje *C++*. En la figura 2.2 se muestra la placa usada.



Figura 2.2: Placa NUCLEO-F429ZI.

2.3.3 Lectura RFID

Teniendo en cuenta que se pretende hacer un prototipo se buscó un lector RFID de bajo costo, y se eligió el modelo RC522. En la figura 2.3 se muestra el módulo usado. Este módulo cuenta con el circuito integrado MFRC522, que es un lector inalámbrico que trabaja a 13,56 MHz, frecuencia específica en la que se debe leer las caravanas electrónicas.

El módulo RC522 cuenta con todos los componentes externos necesarios para que el circuito integrado MFRC522 funcione correctamente, sobre todo la antena. La distancia de lectura de este módulo es de unos pocos centímetros e incluye dos tarjetas que pueden ser leídas, y que se utilizarán para hacer pruebas simulando las caravanas electrónicas.

Para comunicarse con este módulo se utiliza el protocolo SPI, y para la programación de la placa NUCLEO se utilizó un módulo de software escrito por Martin Olejar y publicado en la página oficial de mbed [6]. Este módulo contiene toda la programación necesaria para la correcta utilización del módulo RC522.



Figura 2.3: Módulo RC522.

2.3.4 Comunicación Wi-Fi

Para esta comunicación se debe destacar que para este prototipo no era estrictamente necesario usar Wi-Fi, y se podría haber reemplazado por ejemplo con una comunicación vía Bluetooth. Sin embargo, para la versión final se tiene la intención de adicionar el uso de bastones lectores RFID, que se comunicarían por Bluetooth directamente con el celular. Por esta razón se decidió utilizar otra vía de comunicación, y se eligió el Wi-Fi. Además, al usar este protocolo se tiene un rango de comunicación aceptable, y se podría acceder en futuras ampliaciones a la comunicación desde la placa con más de un celular, de modo que más de una persona pueda ver las lecturas RFID.

Habiendo decidido utilizar Wi-Fi, se observó que, dada la información que se necesita transmitir, resultaba muy útil y cómodo usar el protocolo MQTT (*Message Queuing Telemetry Transport*). Este protocolo está diseñado para la transferencia de datos en redes con ancho de banda limitado y alta latencia, y es ideal para aplicaciones de Internet de las

Cosas (IoT) debido a su eficiencia y simplicidad. Para utilizarlo es necesario contar con un *Broker* o Servidor, y cada dispositivo o cliente puede publicar mensajes en distintos Tópicos, y suscribirse a otros. De esta manera cada dispositivo envía mensajes con la información que necesita transmitir en el tópico que corresponda, y se suscribe a aquellos tópicos de los que necesite recibir información. En este caso se cuenta con solo dos dispositivos, pero es muy útil para la comunicación de varios clientes ya que permite la publicación y suscripción de cualquiera de estos.

Para esta implementación, dado que la placa NUCLEO no cuenta con un módulo Wi-Fi integrado, es necesario resolverlo externamente. Para ello la intención original era usar el módulo ESP-01s, basado en el chip ESP8266. En la figura 2.4 se muestra este módulo. Las ventajas de este módulo son su bajo costo y tamaño, y su simplicidad, dado que puede usarse con solo una comunicación UART, los pines de alimentación y uno de habilitación. La configuración de este módulo se puede realizar mediante comandos AT, y de esta manera el módulo solo tendría la función de antena y de mediador entre las señales Wi-Fi y el microcontrolador. Sin embargo, durante la implementación no se pudo implementar el uso de MQTT con este módulo, y fue necesario reemplazarlo.



Figura 2.4: Módulo ESP-01s.

Como solución se decidió utilizar un módulo ESP32 DEVKIT V1, basado en el chip ESP-Wroom-32. Este chip es también un microcontrolador de 32 bits, y el módulo cuenta con antena Wi-Fi, lo que lo hace un módulo especialmente pensado para implementaciones que usan Wi-Fi. En la figura 2.5 se muestra el módulo utilizado. Dado que es un microcontrolador, es necesario programarlo de manera independiente a la placa NUCLEO, lo que por un lado puede ser una desventaja por no tener todo el control del programa centralizado en una placa, pero a la vez al contar con dos procesadores realizando tareas distintas puede agilizarse el flujo de programa. Con este cambio, se puede utilizar a este módulo para recibir los mensajes MQTT, preprocesarlos, y enviarlos a la placa mediante UART, y recibir mensajes por esa misma vía y enviarlos por MQTT,

quitando cierto procesamiento a la placa principal. También se tiene como ventaja la posibilidad de programar ciertas señales utilizando los pines del módulo. Para la programación de este módulo se utilizó la plataforma de Arduino, además de dos bibliotecas de código. Una de ellas permite la configuración de la comunicación Wi-Fi, y la otra está diseñada para manejar el protocolo MQTT, y está escrita por Nick O'Leary. La instalación de bibliotecas de código se realiza en este caso en el mismo entorno de desarrollo.



Figura 2.5: Módulo ESP32 DEVKIT V1.

2.3.5 Controles e indicadores

Para los indicadores se decidió utilizar simplemente LEDs, teniendo siempre el cuidado de controlarlos con la placa NUCLEO pero alimentarlos con el módulo Mb-102, de modo de no exigir demasiada corriente a los respectivos pines y puertos de la placa. Para esto se utilizaron transistores y resistencias según conviniera. Para esta implementación se utilizaron dos LEDs para simular el estado de la tranquera, que luego pueden ser reemplazados por un Relé que controle un verdadero actuador que mueva una tranquera.

Para los controles se utilizaron botones sin retención para el control manual de la tranquera, y botones con retención para la selección de modo y el encendido o apagado de la balanza.

Para simular la balanza se utilizó simplemente un potenciómetro para controlar manualmente el peso que se desea asignar a algún animal. Para completar esta implementación se programó en el microcontrolador que un extremo del potenciómetro se interprete como 0 kg, mientras que la otra como 500 kg.

2.3.6 Aplicación de celular

Para el diseño de la aplicación de celular se utilizó la plataforma App Inventor, creada por el MIT (Massachusetts Institute of Technology). Esta plataforma permite la creación de aplicaciones simples, utilizando una programa con bloques, lo que habilita su utilización a usuarios sin experiencia en la creación de aplicaciones o sin conocimientos de los lenguajes de programación necesarios.

A la hora de diseñar la aplicación se tuvo en cuenta desde la comunicación el tipo de datos que se debe intercambiar con el microcontrolador (a través del Broker MQTT y el módulo ESP32), programando la correcta recepción e interpretación de los mensajes, así como el envío de mensajes hacia el microcontrolador. Para el apartado visual y de navegación, se buscó tener un diseño simple que contuviera las funciones necesarias para el uso del sistema y que fuera fácil e intuitiva de utilizar.

2.3.7 Conexionado

Para el montaje del sistema se utilizaron dos protoboard, lo que permitió la interconexión de los distintos módulos de forma cómoda y ordenada, manteniendo la flexibilidad necesaria en esta etapa del desarrollo.

CAPÍTULO 3

Diseño e implementación

El desarrollo del proyecto se guardó en un repositorio de GitHub, incluyendo firmware, hardware, aplicación de celular y otros documentos. A través del siguiente enlace se puede acceder al repositorio:

https://github.com/avillacampafiuba/Proyecto_Tranquera

3.1 Diseño del Hardware

Como se mencionó en el capítulo anterior, para el montaje del hardware del sistema se utilizaron dos protoboards, lo que permitió distribuir las conexiones de forma que se obtuvo un prototipo prolíjo y ordenado. A partir de uno de los laterales de los protoboards se realizan las conexiones hacia el microcontrolador. En la figura 3.1 se puede ver el montaje final del prototipo.

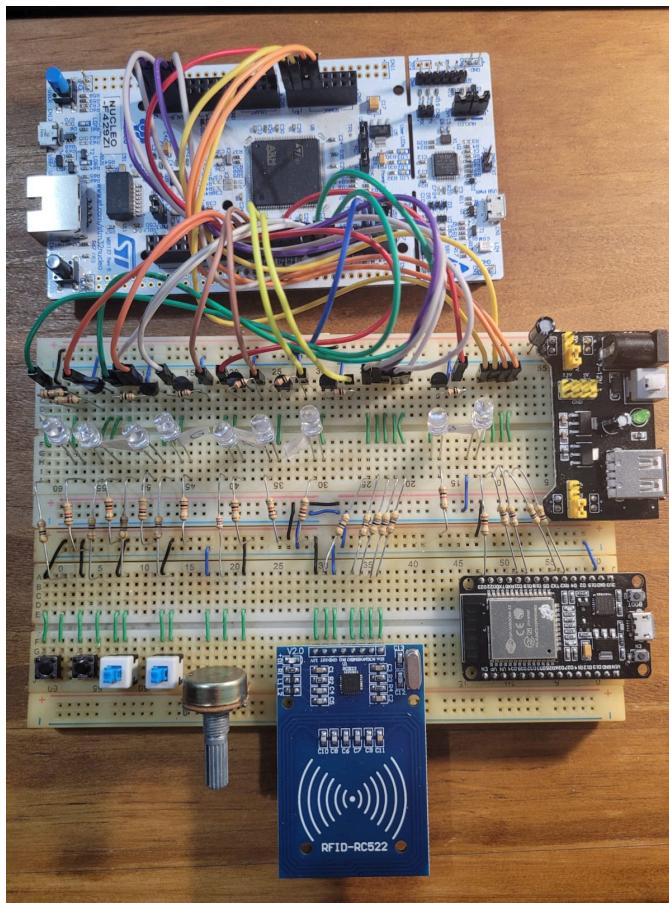


Figura 3.1: Montaje final del prototipo.

Para esta implementación se muestra en las subsecciones 3.1.1 a 3.1.5 los circuitos esquemáticos de cada parte, desarrollados en KiCad. Se debe mencionar la importancia de interconectar las referencias de 0 V de los distintos módulos.

3.1.1 Diseño del hardware de alimentación

En la figura 3.2 se muestra el circuito esquemático del hardware de alimentación. Se destaca la utilización del capacitor C1 para la estabilización de la tensión de salida del módulo Mb-102.

ALIMENTACIÓN

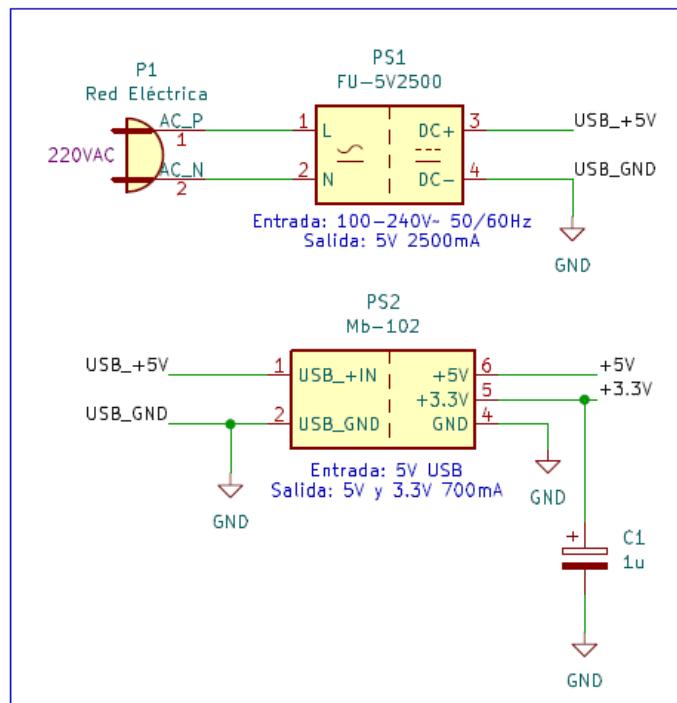


Figura 3.2: Circuito esquemático del hardware de alimentación.

3.1.2 Diseño del hardware del módulo RFID

En la figura 3.3 se muestra el circuito esquemático del hardware del módulo RFID. Se observa la utilización de los pines de MISO, MOSI, SCK y CS para la comunicación SPI, además del conexionado del pin de *Reset* para la correcta configuración del módulo. Se destaca además la utilización de resistores de $47\ \Omega$ en los pines que no son de alimentación. Eso se hace para que ante un eventual error en el conexionado no se produzca un cortocircuito directo, pudiendo evitar así la destrucción de los componentes.



MÓDULO RFID

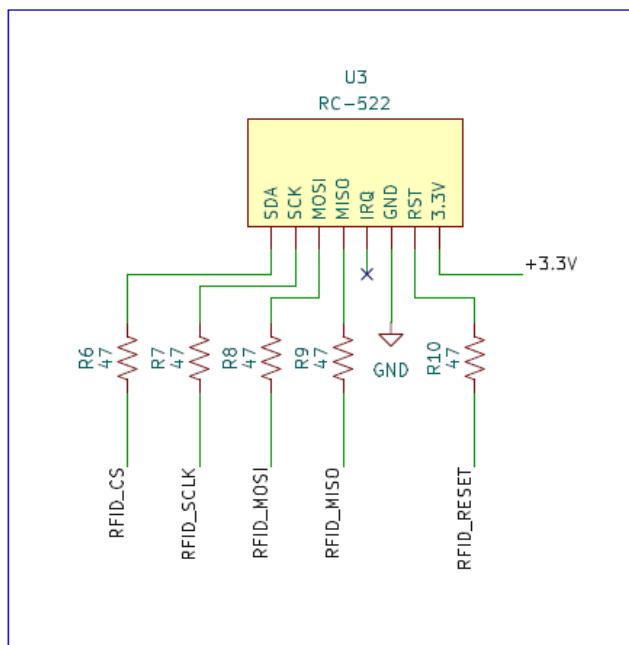


Figura 3.3: Circuito esquemático del hardware del módulo RFID.

3.1.3 Diseño del hardware del módulo Wi-Fi/MQTT

En la figura 3.4 se muestra el circuito esquemático del hardware del módulo Wi-Fi/MQTT. En este caso se representa el chip ESP32-WROOM-32, agregando además los nombres de los pines en el módulo ESP32 DEVKIT V1, a los cuales se realizan las conexiones realmente.

Se observa que además de los pines de recepción y transmisión del protocolo UART se hace uso de los pines D19, D21 y D23. A través de D19 se transmite la información de si se recibió un mensaje nuevo a través de MQTT, mientras que D21 se usa para transmitir si la conexión MQTT está activa o no. Para los pines de UART y para D19 y D21 se utilizan nuevamente resistores de $47\ \Omega$, por la misma razón explicada en la subsección 3.1.2. En el pin D23 se conecta un LED que se usa para hacer una prueba de la conexión MQTT. Este LED puede ser encendido exclusivamente por este módulo (no por la placa NUCLEO), y para su conexión se utiliza un resistor de $1\ k\Omega$ para limitar la corriente.



MÓDULO WIFI / MQTT

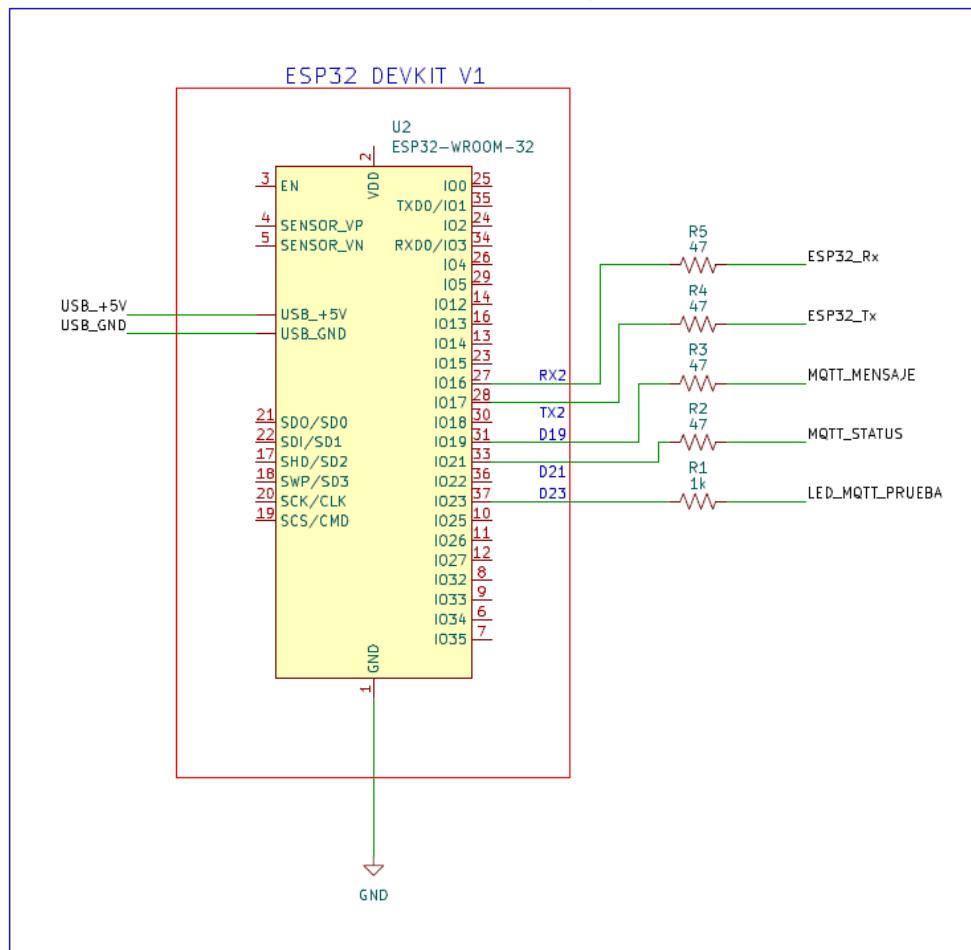


Figura 3.4: Circuito esquemático del hardware del módulo Wi-Fi/MQTT.

3.1.4 Diseño del hardware de los controles e indicadores

En la figura 3.5 se muestra el circuito esquemático del hardware de los controles. Se observa nuevamente el uso de resistores de $47\ \Omega$ para aquellas señales que van directamente al microcontrolador.

Para la conexión de los cuatro botones se observa que, con o sin retención, una posición cierra la conexión hacia 3,3 V, mientras que la otra corta la conexión. Esto se implementa de esta manera ya que se propone utilizar resistores de *Pull-Down* internos en el microcontrolador, que conectarán los respectivos pines a 0 V cuando los botones no fueren la conexión a 3,3 V.

Para el potenciómetro que reemplaza la balanza se observa que se utilizó uno de valor 10 k Ω , que a través de R15 se conecta a un pin analógico de la placa NUCLEO. De esta

manera con el potenciómetro se puede variar la tensión de entrada al pin desde 0 V hasta (casi) 3,3 V.

CONTROLES

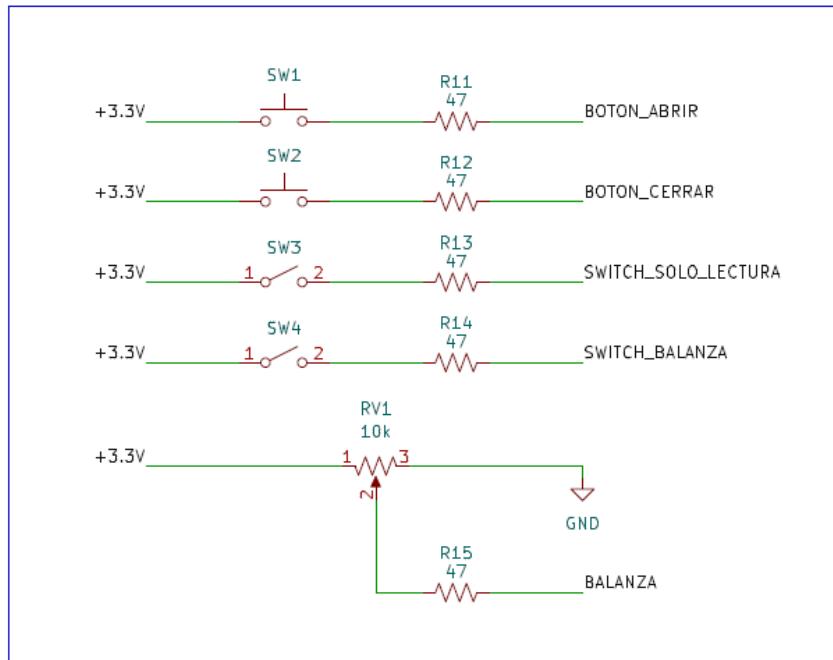


Figura 3.5: Circuito esquemático del hardware de los controles.

Para los indicadores, en las figuras 3.6 y 3.7 se muestra el diagrama esquemático del conexionado.

INDICADORES

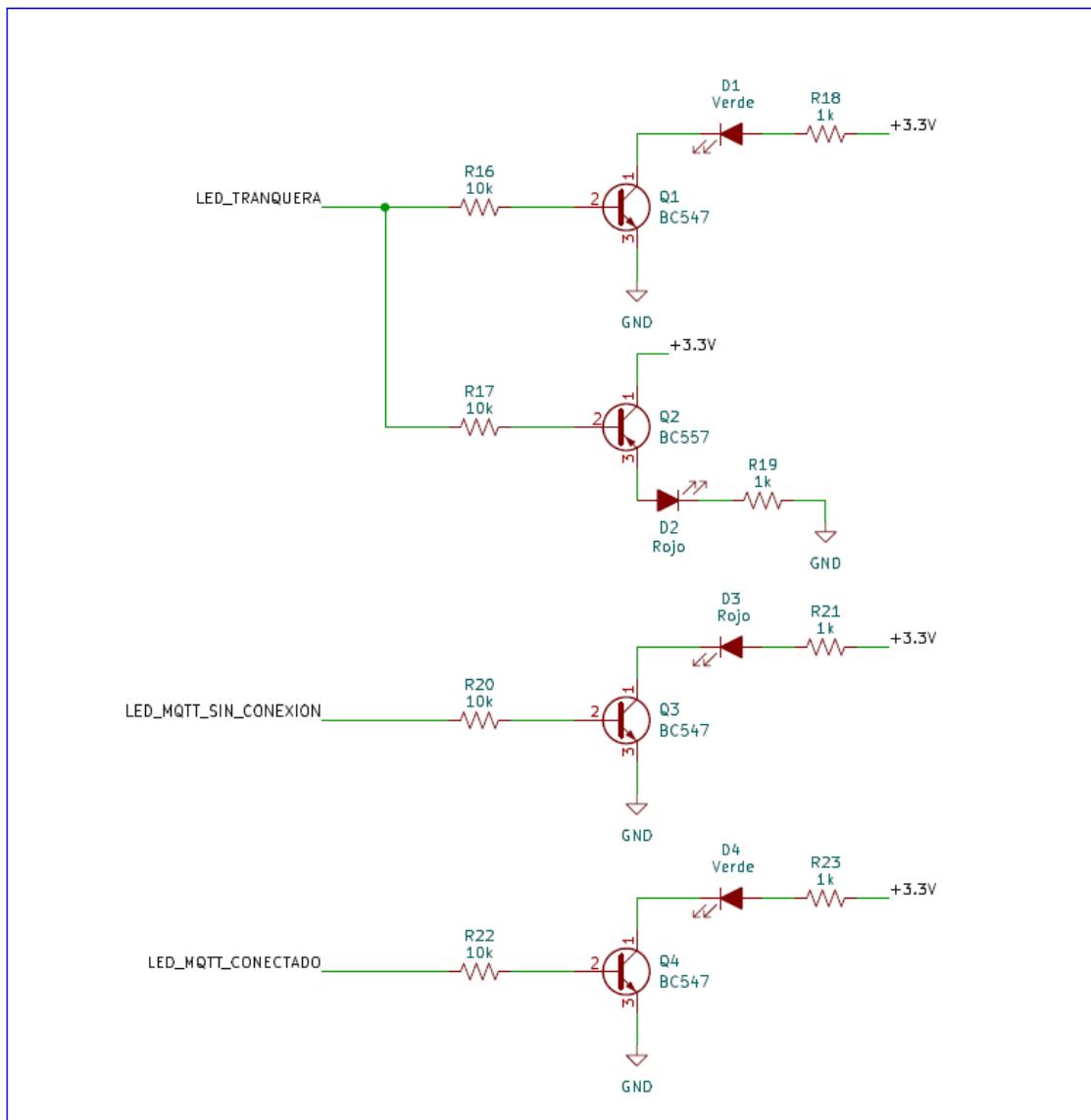


Figura 3.6: Circuito esquemático del hardware de los indicadores. Parte 1.

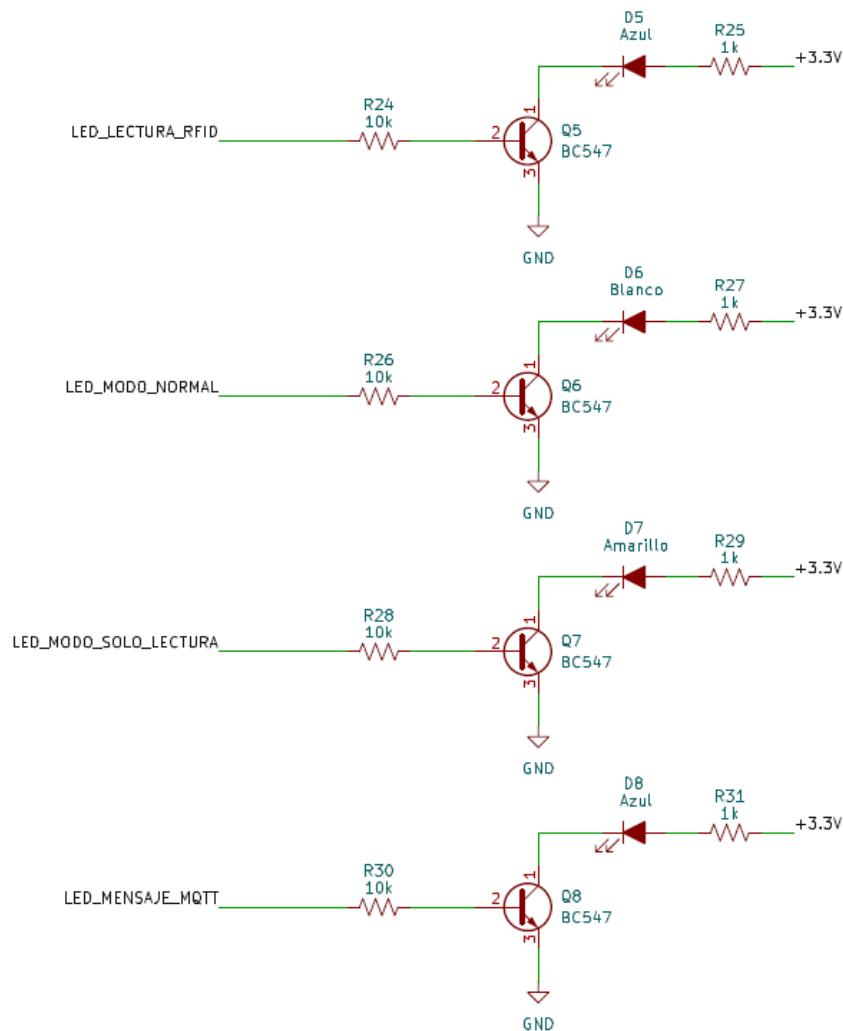


Figura 3.7: Circuito esquemático del hardware de los indicadores. Parte 2.

Se observa que en todos los casos la señal que llega desde el microcontrolador se conecta a través de un resistor de $10\text{ k}\Omega$ a la base del transistor que controla el encendido del LED. A través de este transistor se alimenta el LED usando un resistor de $1\text{ k}\Omega$ como limitador de corriente. La energía se toma entonces del módulo Mb-102, mientras que el microcontrolador solo debe aportar la corriente necesaria para activar el transistor.

Excepto para el caso de la tranquera se utilizaron transistores NPN BC547B. El caso de la tranquera es, como se puede observar en la figura 3.6, ligeramente distinto. Para la indicación del estado de la tranquera se decidió utilizar dos LEDs (abierto y cerrado), pero conectados a un único pin, de modo que cuando el pin se encuentra en estado alto se

enciendo un LED, mientras que si se encuentra en estado bajo se enciende el otro. Para esta conexión se siguió el mismo esquema que para los demás indicadores, pero adicionando un circuito con un transistor PNP BC557B. Así, los transistores alternan el funcionamiento según el estado del pin.

3.1.5 Diseño del hardware del microcontrolador

En las figuras 3.8 y 3.9 se muestra el circuito esquemático del conexionado del microcontrolador NUCLEO-F429ZI. En este caso en estas figuras se muestra únicamente la conexión de las distintas señales a sus respectivos pines de la placa, así como la conexión a la referencia común de 0 V. Se debe mencionar que para esta implementación se buscó utilizar únicamente pines a los que se pudiera acceder a través de alguno de los zócalos, de modo de no necesitar realizar soldaduras. Esta decisión se tomó únicamente por tratarse del desarrollo de un prototipo.

MICROCONTROLADOR
NUCLEO-F429ZI

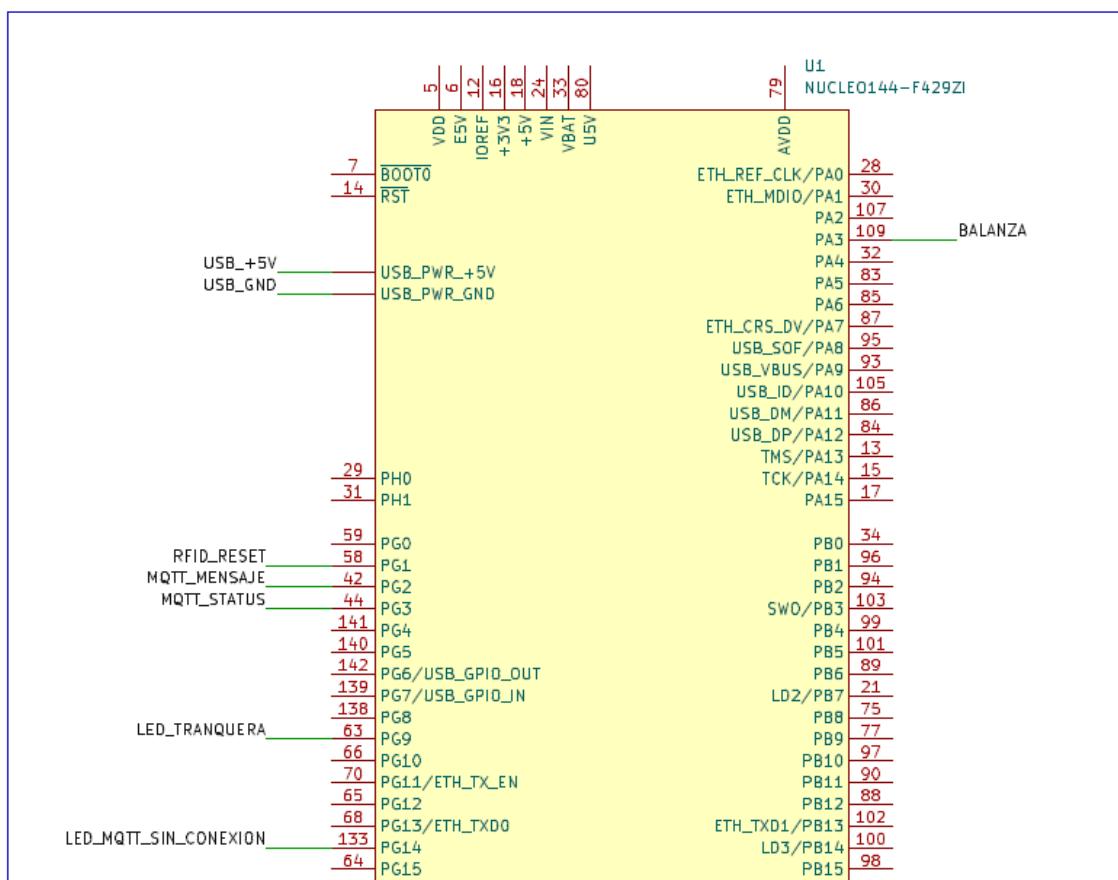


Figura 3.8: Circuito esquemático del conexionado del microcontrolador. Parte 1.

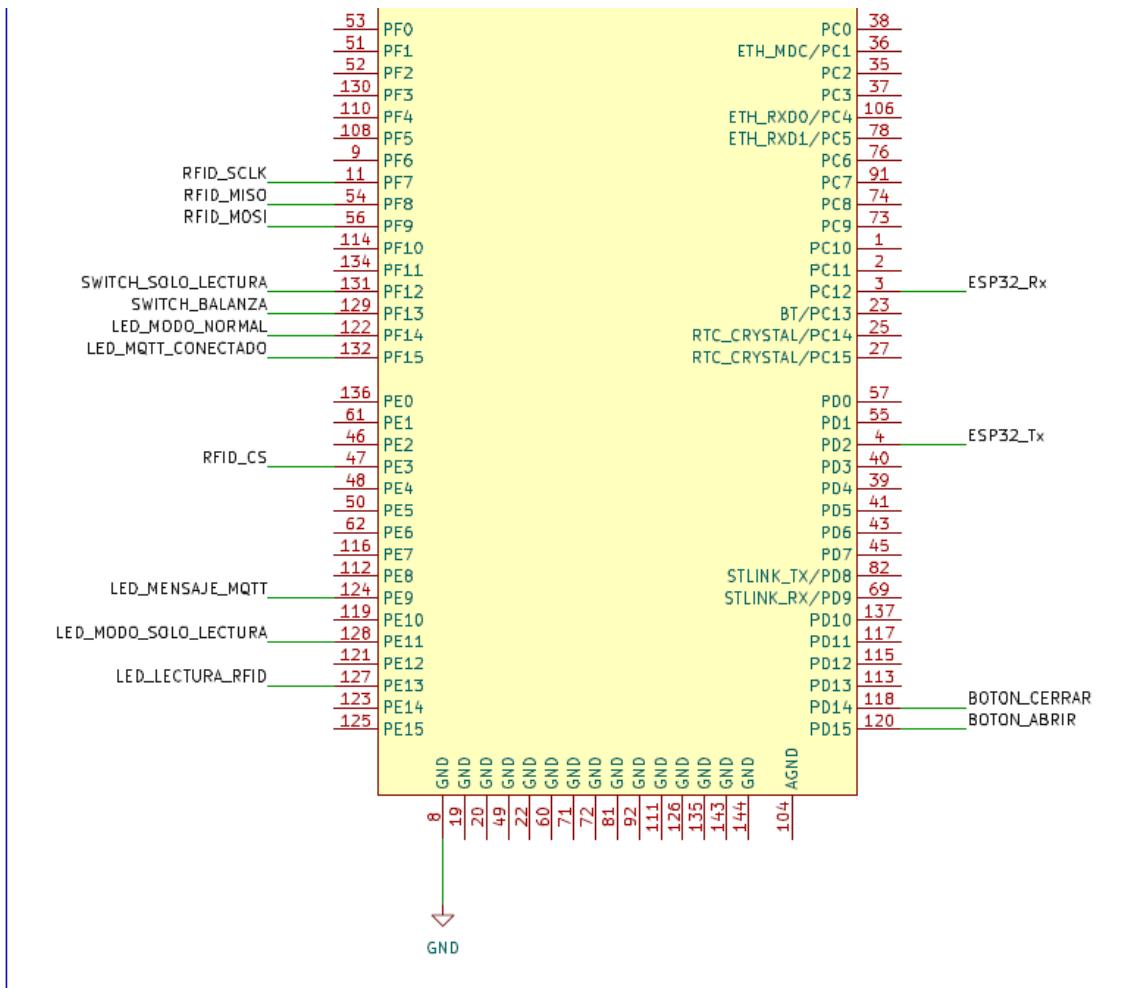


Figura 3.9: Circuito esquemático del conexionado del microcontrolador. Parte 2.

3.2 Diseño del Firmware

3.2.1 Firmware de la placa NUCLEO-F429ZI

Como se mencionó en el capítulo 2, el desarrollo del firmware para esta placa se realizó en la plataforma *Keil Studio Cloud* utilizando el lenguaje *C++*. Para estructurar el código se decidió organizar el programa en módulos, y para ello se tienen por un lado los archivos *main.cpp*, *arm_book_lib.h*, *globalDefines.h* y *mbed_app.json*. Estos archivos contienen, respectivamente, la función *main* del programa, numerosas definiciones y algunas configuraciones. Por otro lado se tiene una carpeta llamada *modules* que a su vez contiene carpetas con los archivos de cada módulo, un archivo *.cpp* y otro *.h*. También se debe mencionar que se utiliza mbed OS 6, el sistema operativo que corre de fondo en el microcontrolador.

A continuación se listan los diferentes archivos del firmware de esta placa:

- *main.cpp*
- *mbed_app.json*
- modules
 - *arm_book_lib.h*
 - *global_defines.h*
 - MFRC522
 - *MFRC522.cpp*
 - *MFRC522.h*
 - MQTT
 - *MQTT.cpp*
 - *MQTT.h*
 - led
 - *led.cpp*
 - *led.h*
 - mode_selector
 - *mode_selector.cpp*
 - *mode_selector.h*
 - non_blocking_delay
 - *non_blocking_delay.cpp*
 - *non_blocking_delay.h*
 - rfid
 - *rfid.cpp*
 - *rfid.h*
 - scale
 - *scale.cpp*
 - *scale.h*
 - system
 - *system.cpp*
 - *system.h*
 - tranquera
 - *trajquera.cpp*
 - *trajquera.h*

Para diseñar el flujo natural del programa se eligió que la función *main* únicamente tenga una llamada inicial a la función *system_init* del módulo *system*, y luego entra en un bucle donde se llama a la función *system_update*, también del módulo *system*. En la figura 3.10 se muestra el código del archivo *main.cpp*.

```

1  //=====[Libraries]=====
2
3  #include "mbed.h"
4  #include "arm_book_lib.h"
5
6  #include "system.h"
7
8  //=====[Main function, the program entry point after power on or reset]=====
9
10 int main()
11 {
12     system_init();
13     while (true) {
14         system_update();
15     }
16 }
```

Figura 3.10: Código del archivo *main.cpp*.

Por otro lado, en el archivo *mbed_app.json* se configura el tamaño de los buffers de transmisión y recepción de la comunicación UART, como se ve en la figura 3.11.

```

1  {
2      "target_overrides": {
3          "*": {
4              "platform.stdio-convert-newlines": 1,
5              "target.printf_lib": "std",
6              "drivers.uart-serial-rxbuf-size": 1024,
7              "drivers.uart-serial-txbuf-size": 256
8          }
9      }
10 }
```

Figura 3.11: Código del archivo *mbed_app.json*.

Y en el archivo *global_defines.h*, como se mencionó anteriormente, se hacen numerosas definiciones para todos los aspectos de la implementación. Primeramente se realiza la definición de los pines de la placa a los que se conectarán cada señal, de modo de realizar cambios únicamente en este archivo si el diseño de hardware necesitará algún cambio. También se hizo uso de la compilación condicional para que todas estas definiciones estén asociadas a una definición anterior, en donde se decide qué placa se usará. Solo fueron definidos los pines para la placa que se usa en este trabajo, pero se tomó esta decisión para simplificar el trabajo si en un futuro se decide utilizar otra placa.

Luego se realizan varias definiciones asociadas a distintos módulos, como las duraciones de ciertos retardos y los tópicos y mensajes que se enviarán por MQTT.

3.2.1.1 Módulo led

En este módulo se define la clase *Led*, que no es más que una abstracción de la clase *DigitalOut*, ya que esta clase solo cuenta en sus métodos con dos constructores, una función para leer y otra para escribir. Con esta clase se controla el estado de un pin digital, pero se usa para controlar aquellos pines en los que se conectan los LEDs indicadores.

3.2.1.2 Módulo tranquera

Este módulo es el encargado del control de la tranquera, y dado que en esta implementación la tranquera se simula con dos LEDs este módulo incluye al módulo led. En este módulo se define la clase *Tranquera*, que cuenta con su pin de salida (un objeto *Led*), dos interrupciones de hardware asociadas a los botones de apertura y cierre manual de la tranquera, y una máquina de estados que almacena la posición de la tranquera (abierta o cerrada). Para crear un objeto de esta clase es necesario entonces definir el pin donde están conectados los LEDs de salida, los pines donde están conectados los botones de control manual, y los modos de estos botones. Como se dijo anteriormente, el modo elegido es *Pull-Down*.

De esta manera, cuando se presionan los botones de control manual de la tranquera el estado de esta (en la máquina de estados y en los LEDS) se actualiza automáticamente a través de las respectivas funciones de *Callback* de las interrupciones.

3.2.1.3 Módulo scale

Con este módulo se controla el potenciómetro que reemplaza la balanza. Aquí se declara la clase *Scale* y una máquina de estados que guarda el estado de la balanza (apagada o encendida). La clase cuenta con una objeto *AnalogIn* para controlar el pin de entrada del potenciómetro, un objeto *DigitalIn* para leer el estado del botón que controla el encendido de la balanza, y dos objetos *InterruptIn* para detectar los cambios de estados del botón antes mencionado. Así, cada vez que se produce un cambio en el botón se actualiza la máquina de estados automáticamente a través de las respectivas funciones de *Callback* de las interrupciones.

Cuando se realiza una lectura de la balanza, si esta se encuentra apagada se devuelve “-”, mientras que si está encendida se devuelve la medición con dos cifras decimales. Esta función de lectura puede verse en la figura 3.12.

```

41  bool Scale::read(char * buffer)
42  {
43      // Verificar si la balanza está apagada
44      if (state == APAGADO) {
45          strcpy(buffer, "-");
46          return false; // La balanza está apagada, devuelve false
47      }
48
49      // La sobrecarga de float() en Scale llamará a AI.read() automáticamente
50      // Leer el valor analógico
51      float value = AI.read() * 500.0f;
52
53      // Formatear el valor con dos dígitos decimales en buffer
54      sprintf(buffer, 16, "%.2f", value);
55
56      return true; // La balanza está encendida, devuelve true
57  }

```

Figura 3.12: Código de la definición del método *read* de la clase *Scale*, en el archivo *scale.cpp*.

3.2.1.4 Módulo mode_selector

En este módulo se resuelve la definición del modo de funcionamiento del sistema, que puede ser normal (*Idle*) o forzado a *Solo Lectura*. Para esto el módulo cuenta con una máquina de estados para el modo de funcionamiento (los dos antes mencionados) y define la clase *ModeSelector*. Esta clase cuenta con dos objetos de la clase *Led* para manejar los LEDs que indican el modo actual, un objeto *DigitalIn* para leer el estado del botón de modo, y un objeto *InterruptIn* para que a través de funciones de *Callback* se actualice automáticamente el modo ante algún flanco en el pin del botón.

3.2.1.5 Módulo non_blocking_delay

Este módulo se utiliza para resolver todos los retardos dentro del programa. Se define la clase *nonBlockingDelay*, que se puede construir indicando sólo la duración del retardo necesario, o se puede además definir una función de *Callback* que se llamará cuando el retardo finalice.

Esta clase cuenta también con cuatro sobrecargas del método *Start*, que puede iniciar un retardo como se definió inicialmente, o puede cambiar, la duración, la función del *Callback* o ambas. Además se tienen métodos para cambiar la duración del retardo o la función de *Callback* sin iniciar el contador y otro para leer si el contador finalizó.

Para esto se tiene dos objetos de la clase *Ticker*, a uno de los cuales se le asocia una función de *Callback* que apaga el contador cuando el retardo finaliza, y otro al cual se le asocia la función de *Callback* que define el usuario y que quiere que sea llamada cuando el

retardo finalice. Si el usuario no define función de *Callback* entonces no se ejecuta ninguna.

También es importante destacar que al usar esta clase se generan retardos no bloqueantes para preservar el flujo del programa.

3.2.1.6 Módulo MQTT

En este módulo se maneja toda la comunicación del protocolo MQTT, que en realidad es una comunicación por UART con el módulo de hardware ESP32. Primero se define una máquina de estados para saber el estado de la conexión MQTT, que puede estar conectada o desconectada. También se define una estructura *PendingMessage_t* para almacenar los mensajes recibidos y enviados y los tópicos a los que pertenecen. Finalmente se define la clase MQTT.

Esta clase cuenta con un objeto de la clase *BufferedSerial* para manejar la comunicación con la placa ESP32, un objeto *DigitalIn* para leer el estado de la conexión MQTT, y tres objetos Led para manejar los LEDs que indican el estado de la conexión MQTT y un tercer LED que se enciende por cierto tiempo cuando se recibe un mensaje MQTT. Para manejar el tiempo de encendido de este último LED se utiliza un retardo no bloqueante con la clase *nonBlockingDelay*.

Entre los métodos públicos con los que cuenta esta clase están *subscribe* y *unsubscribe* para suscribirse y desuscribirse a determinados tópicos respectivamente, *publish* para publicar un mensaje en algún tópico, *receive* para procesar un mensaje recibido y *processPendings* para procesar envíos pendientes de mensajes.

En esta implementación se cuenta con vectores para almacenar los tópicos a los que se está suscrito, las suscripciones pendientes, las desuscripciones pendientes y los mensajes pendientes. Los tópicos a los que se está suscrito se agregan al respectivo vector únicamente cuando se recibió una respuesta por parte de la placa ESP32 de que el mensaje de suscripción fue enviado, y lo mismo aplica para las desuscripciones. Y para eso cuenta con los métodos públicos *confirmSubscription* y *confirmUnsubscription*.

Además de la comunicación UART existe conexión con la placa ESP32 a través de otros dos pines, en los cuales se recibe un pulso cuando cambia el estado de la conexión MQTT o cuando se recibe un mensaje. De esta forma se pueden definir interrupciones que realicen acciones cuando se reciban estos pulsos. En el caso de un cambio de estado de la conexión, la interrupción se encarga de cambiar la máquina de estados y el estado de los LEDs. Además, si se detecta una desconexión todos los tópicos a los que se está suscrito pasan a estar pendientes, para volver a enviar los mensajes de suscripción cuando se restablezca la conexión. En el caso de la recepción de un mensaje por UART, que equivale a la recepción de un mensaje por MQTT excepto en las confirmaciones de suscripción o

desuscripción, la interrupción se encarga de encender el respectivo LED y comenzar el retardo que lo apagará cuando termine. En el archivo *global_defines.h* se definió que este retardo sea de 1 segundo.

3.2.1.7 Módulos MFRC522 y rfid

Como se mencionó anteriormente, el módulo MFRC522 corresponde a una biblioteca de código creada por un tercero, y que define la clase *MFRC522*. Por este motivo se creó el módulo *rfid*, que define la clase *RFID* y funciona como capa de abstracción. Está diseñado para hacer uso del módulo MFRC522 de forma interna. Para esto se cuenta con un objeto de la clase *MFRC522* y en el constructor de *RFID* se llama al método *PCD_Init* de la clase *MFRC522*, que inicializa todos los parámetros necesarios. Luego el método *read* de *RFID* llama a las funciones necesarias de la clase *MFRC522* para realizar la lectura.

En el módulo *rfid* se define también una máquina de estados que determina si el módulo está leyendo o esperando para leer. También se define un retardo, que se utiliza para que, cuando se realiza una lectura, este módulo se bloquee durante cierto tiempo y no pueda realizar otra lectura. De esta forma se evita realizar la misma lectura múltiples veces en un corto tiempo. En el archivo *global_defines.h* se definió que este retardo sea de 3 segundos. Esta clase cuenta también con un objeto de la clase *Led*, para controlar un LED que se enciende cuando se realiza una lectura, y solo se apaga cuando el retardo de lectura terminó y es posible realizar una nueva lectura.

En la figura 3.13 se puede observar el método *read* de la clase *RFID*, donde se realiza una lectura, se guarda el contenido y se cambia el estado a *LEYENDO*, que a su vez enciende el LED correspondiente y comienza el contador para una nueva lectura.

```

33     char * RFID::read()
34 {
35     if(this->RFID_status == ESPERANDO){
36         // Limpiar el buffer del ID
37         memset(id, 0, sizeof(id));
38
39         // Buscar nuevas tarjetas
40         if (reader.PICC_IsNewCardPresent()) {
41             // Seleccionar una de las tarjetas
42             if (reader.PICC_ReadCardSerial()) {
43                 RFID::set_status(LEYENDO);
44                 // Convertir el UID a una cadena de caracteres hexadecimales
45                 for (uint8_t i = 0; i < this->reader.uid.size; i++) {
46                     sprintf(&this->id[i * 2], "%02X", this->reader.uid.uidByte[i]);
47                 }
48                 this->id[this->reader.uid.size * 2] = '\0'; // Agregar el carácter nulo al final
49                 return this->id;
50             }
51         }
52     }
53     return nullptr; // Devolver nullptr si no se leyó ninguna tarjeta
54 }
```

Figura 3.13: Código de la definición del método *read* de la clase *RFID*, en el archivo *rfid.cpp*.

3.2.1.8 Módulo system

Finalmente, en este módulo se controla el flujo principal del programa. En este módulo se definen dos funciones públicas, *system_init* y *system_update*, que se llaman desde el archivo *main.cpp*, y tres funciones privadas que son *parse_message_received*, *send_tranquera_position* y *send_system_mode*, y que se utilizan en este módulo.

También se crean los objetos necesarios de las clases definidas en los distintos módulos. En este caso se crean objetos de las clases *Tranquera*, *Scale*, *RFID*, *MQTT*, *ModeSelector* y *nonBlockingDelay*. En la figura 3.14 se pueden ver los objetos creados, así como los parámetros necesarios en cada caso, y que fueron definidos en el archivo *globalDefines.h*. Por último se crean algunas variables necesarias para el funcionamiento del código.

```

26 //=====Declaration and initialization of public global objects=====
27
28 // ---Módulos---
29 static Tranquera tranquera(PIN_TRANQUERA, PIN_BOTON_ABRIR, MODE_PIN_BOTON_ABRIR,
30 |   PIN_BOTON_CERRAR, MODE_PIN_BOTON_CERRAR);
31 static Scale scale(PIN_BALANZA, PIN_SWITCH_BALANZA, MODE_PIN_SWITCH_BALANZA);
32 static RFID rfid_reader(PIN_RFID_MOSI, PIN_RFID_MISO, PIN_RFID_SCLK, PIN_RFID_CS,
33 |   PIN_RFID_RESET, PIN_LED_LECTURA);
34 static MQTT mqtt(PIN_MQTT_TX, PIN_MQTT_RX, MQTT_BAUDRATE, PIN_MQTT_STATUS,
35 |   PIN_LED_WIFI_CONECTADO, PIN_LED_WIFI_SIN_CONEXION, PIN_LED_MENSAJE_MQTT);
36 static ModeSelector system_mode(PIN_SWITCH_SOLO_LECTURA, MODE_PIN_SWITCH_SOLO_LECTURA,
37 |   PIN_LED_MODE_NORMAL, PIN_LED_MODE_SOLO_LECTURA);
38 // ---
39
40 // ---Delays---
41 static nonBlockingDelay RFID_read_delay(RFID_READ_TRY_DELAY_TIME_MS);
42 // ---
```

Figura 3.14: Código del archivo *system.cpp* en donde se crean varios objetos.

La función *system_init*, que puede verse en la figura 3.15, únicamente comienza un contador asociado a la lectura RFID y envía los comandos de suscripción a dos tópicos.

```

58 void system_init()
59 {
60     RFID_read_delay.Start();
61
62     mqtt.subscribe(TOPIC_TRANQUERA);
63     mqtt.subscribe(TOPIC_SYSTEM_MODE);
64 }
```

Figura 3.15: Código de la definición de la función *system_init* del archivo *system.cpp*.

El retardo *RFID_read_relay* se utiliza para hacer un intento de lectura cada cierto tiempo y es independiente del retardo interno de la clase *RFID*. Este retardo se definió en 250 ms y se implementó porque no es una acción que deba realizarse de forma inmediata, y de esta forma se da prioridad a otras partes del código. Se asume que al realizar una lectura no se sostendrá la tarjeta (o caravana) en el área de lectura por menos de ese tiempo.

A través de los tópicos a los que se suscribe el sistema se reciben los comando de apertura o cierre de la tranquera o un pedido del estado actual de esta en el primer caso, o un pedido del estado del sistema en el segundo caso, es decir si el sistema está funcionando normalmente o se encuentra forzado a sólo lectura.

Por otro lado, la función *system_update* se muestra en la figura 3.16.

```

66 void system_update()
67 {
68     if(RFID_read_delay.isReady()){
69         RFID_read_delay.Start();
70
71         lectura_rfid = rfid_reader.read();
72         if(lectura_rfid != nullptr){
73             scale.read(lectura_scale);
74             strcpy(msg, lectura_rfid);
75             strcat(msg, ":" );
76             strcat(msg, lectura_scale);
77             mqtt.publish(TOPIC_RFID_LECTURA, msg);
78         }
79         if(tranquera.position_changed){
80             send_tranquera_position();
81             tranquera.position_changed = false;
82         }
83         if(system_mode.mode_changed){
84             send_system_mode();
85             system_mode.mode_changed = false;
86         }
87     }
88     mqtt.processPending();
89
90     if(mqtt.receive(topic_received, message_received)){
91         parse_message_received(topic_received, message_received);
92     }
93 }
```

Figura 3.16: Código de la definición de la función *system_update* del archivo *system.cpp*.

En esta función primero se lee si el retardo definido anteriormente ya finalizó. En caso afirmativo, primero comienza un nuevo retardo y luego hace una lectura RFID. Si se encuentra una tarjeta que leer entonces se hace una lectura de la balanza y luego se envía por MQTT el ID de la tarjeta leída junto con la lectura de la balanza. Luego, si hubo un cambio en el estado de la tranquera o el modo de funcionamiento entonces se llama a las respectivas funciones que envían por MQTT el nuevo estado si corresponde.

Luego, y fuera del retardo *RFID_read_delay*, se llama al método *processPending*s del objeto mqtt para enviar los mensajes pendientes. Y finalmente, si hay algún mensaje recibido desde MQTT lo procesa.

En la figura 3.17 se muestra la definición de la función *parse_message_received*, que procesa los mensajes recibidos.

```

97 void parse_message_received(const char * topic, const char * message)
98 {
99     if (strcmp(topic, "subscribed") == 0) {
100         mqtt.confirmSubscription(message);
101     }
102     else if (strcmp(topic, "unsubscribed") == 0) {
103         mqtt.confirmUnsubscription(message);
104     }
105
106     else if (strcmp(topic, TOPIC_TRANQUERA) == 0) {
107         if(strcmp(message, MESSAGE_TRANQUERA_ABRIR) == 0 && system_mode == IDLE){
108             tranquera = ABIERTO;
109         }
110         else if(strcmp(message, MESSAGE_TRANQUERA_CERRAR) == 0 && system_mode == IDLE){
111             tranquera = CERRADO;
112         }
113         else if(strcmp(message, MESSAGE_TRANQUERA_PEDIR_ESTADO) == 0){
114             send_tranquera_position();
115         }
116     }
117
118     else if (strcmp(topic, TOPIC_SYSTEM_MODE) == 0) {
119         if(strcmp(message, MESSAGE_SYSTEM_MODE) == 0){
120             send_system_mode();
121         }
122     }
123 }
```

Figura 3.17: Código de la definición de la función *parse_message_received* del archivo *system.cpp*.

Se puede ver que lo primero que hace esta función es verificar si se recibieron mensajes de confirmación de suscripción o desuscripción, y realiza las confirmaciones correspondientes según el caso. Luego verifica si el mensaje recibido corresponde al tópico asociado a la tranquera. Si este es el caso y el mensaje es un comando de apertura o cierre de la tranquera lo ejecuta siempre y cuando no se encuentre forzado el modo de

solo lectura, y en el caso de que el comando sea un pedido del estado de la tranquera entonces lo transmite. Se debe aclarar que ante un cambio del estado de la tranquera automáticamente se envía el aviso del nuevo estado a través de MQTT. Finalmente, la función verifica si el tópico es el asociado al modo de funcionamiento, cuyo único posible mensaje es un pedido del estado, el cual envía si se recibe el mensaje correspondiente. Nuevamente, cuando se realiza un cambio en el modo de funcionamiento se envía el aviso del nuevo modo automáticamente.

3.2.2 Firmware de la placa ESP32 DEVKIT V1

El desarrollo de este firmware, como ya se mencionó, se realizó con el entorno de desarrollo de Arduino, que permite crear un código en lenguaje Arduino y programar con él la placa ESP32 directamente, definiendo como placa objetivo la ESP32. En este caso todo el firmware se implementó en un único archivo.

Al comienzo del programa, luego de incluir las bibliotecas de código necesarias, lo primero que hace es definir objetos y variables necesarias. Esto incluye los objetos que manejan la conexión Wi-Fi y MQTT, y variables como la red Wi-Fi y la contraseña, la IP del servidor MQTT y el puerto usado, o los pines que se usarán. Esto se muestra en la figura 3.18.

```

1 #include <WiFi.h>
2 #include <PubSubClient.h>
3
4 WiFiClient esp32Client;
5 PubSubClient mqttClient(esp32Client);
6
7 const char* ssid      = "";
8 const char* password = "";
9
10 char *server = "";
11 int port = 1883;
12
13 int ledpin = 23;           // Pin para el LED
14 int mqttStatusPin = 21;   // Pin para indicar estado MQTT
15 int messagePin = 19;       // Pin para indicar que se recibió un mensaje y se reenvió por UART

```

Figura 3.18: Código inicial del firmware de la placa ESP32.

Luego se definen tres funciones que son *wifiInit*, *callback* y *reconnect*. La primera se encarga de realizar la conexión con la red Wi-Fi y esperar a estar conectado. La segunda es la función que se llama cuando se recibe un mensaje MQTT, y se encarga de procesar el mensaje. La función se muestra en la figura 3.19.

```

29 void callback(char* topic, byte* payload, unsigned int length) {
30     char payload_string[length + 1];
31     memcpy(payload_string, payload, length);
32     payload_string[length] = '\0';
33
34     if (strcmp(topic, "ProyectoTranquera/TestMQTT") == 0) {
35         int resultI = atoi(payload_string);
36         var = resultI;
37     }
38
39     // Reenviar el mensaje por UART
40     Serial2.print(topic);
41     Serial2.print(":");
42     Serial2.print(payload_string);
43     Serial2.print("\n");
44
45     digitalWrite(messagePin, HIGH);
46     digitalWrite(messagePin, LOW);    // Hago un pulso cuando envié un mensaje
47 }
```

Figura 3.19: Código de la definición de la función *callback* del firmware de la placa ESP32.

En esta función se observa que luego de guardar el mensaje recibido verifica si el tópico es *ProyectoTranquera/TestMQTT*, ya que este es el tópico que se utiliza para controlar el LED de prueba de conexión MQTT, conectado al pin 23 como se observa en la figura 3.18. En caso de recibir un mensaje en ese tópico ajusta el valor del pin según el mensaje, que debería ser 0 o 1, indicando el estado deseado del LED. Luego envía cualquier mensaje recibido hacia la placa NUCLEO con un formato específico, que es el tópico y el mensaje separados por el símbolo ':', y terminando el mensaje con '\n'. Finalmente realiza un pulso en el pin correspondiente para activar en la placa NUCLEO la interrupción de recepción de un mensaje.

Luego en la función *reconnect* se entra en un ciclo mientras no se logre reconectarse, en el que se mantiene el pin de estado de la conexión MQTT en estado bajo y se intenta conectar nuevamente. Si se logra reconnectar entonces se vuelve el pin de estar de la conexión MQTT a estado alto y se resubscribe al tópico *ProyectoTranquera/TestMQTT*. Estos intentos de reconexión se realizan cada 5 segundos.

Luego en la función *setup*, que se muestra en la figura 3.20, se realizan todas las configuraciones iniciales del código, ya que es la primera función en ejecutarse.

```
62 void setup() {  
63     pinMode(ledpin, OUTPUT);  
64     pinMode(mqttStatusPin, OUTPUT); // Configurar el pin de estado MQTT como salida  
65     pinMode(messagePin, OUTPUT); // Configurar el pin de mensaje como salida  
66     Serial2.begin(115200, SERIAL_8N1, RXD2, TXD2); // Configuración UART2 a 115200 baudios  
67     delay(10);  
68     wifiInit();  
69     mqttClient.setServer(server, port);  
70     mqttClient.setCallback(callback);  
71     digitalWrite(mqttStatusPin, LOW); // El Pin empieza en LOW  
72     digitalWrite(messagePin, LOW); // El Pin empieza en LOW  
73 }
```

Figura 3.20: Código de la definición de la función *setup* del firmware de la placa ESP32.

Finalmente, la función *loop* contiene el código que se repetirá en un ciclo infinito, que se muestra en la figura 3.21. En esta función primero se verifica el estado de la conexión MQTT, y se ejecuta la función *reconnect* si se perdió la conexión. Luego actualiza el estado del LED de prueba según lo guardado en la variable *var* que se actualiza automáticamente al recibir un mensaje de cambio en ese pin. Finalmente lee el puerto UART a la espera de recibir un mensaje desde la placa. Si hay un mensaje para leer entonces lo procesa para saber si es un mensaje de suscripción o desuscripción, en cuyo caso se suscribe o desuscribe según corresponda y envía la confirmación de vuelta a la placa NUCLEO, o si es otro mensaje. En este último caso simplemente envía el mensaje al tópico que corresponda.

```

75 void loop() {
76     if (!mqttClient.connected()) {
77         reconnect();
78     } else {
79         digitalWrite(mqttStatusPin, HIGH); // Mantener en HIGH si está conectado
80     }
81     mqttClient.loop();

82
83     if (var == 0) {
84         digitalWrite(ledpin, LOW);
85     } else if (var == 1) {
86         digitalWrite(ledpin, HIGH);
87     }
88     if (Serial2.available()) {
89         String uartMessage = Serial2.readStringUntil('\n'); // Leer mensaje UART
90         int separatorIndex = uartMessage.indexOf(':');
91         if (separatorIndex != -1) {
92             String command = uartMessage.substring(0, separatorIndex);
93             String argument = uartMessage.substring(separatorIndex + 1);
94             if (command == "subscribe") {
95                 mqttClient.subscribe(argument.c_str());
96                 Serial2.print("subscribed:");
97                 Serial2.print(argument);
98                 Serial2.print("\n");
99                 digitalWrite(messagePin, HIGH);
100                digitalWrite(messagePin, LOW); // Hago un pulso cuando envié un mensaje
101            } else if (command == "unsubscribe") {
102                mqttClient.unsubscribe(argument.c_str());
103                Serial2.print("unsubscribed:");
104                Serial2.print(argument);
105                Serial2.print("\n");
106                digitalWrite(messagePin, HIGH);
107                digitalWrite(messagePin, LOW); // Hago un pulso cuando envié un mensaje
108            } else {
109                // Publicar mensaje en MQTT
110                mqttClient.publish(command.c_str(), argument.c_str());
111            }
112        }
113    }
114}

```

Figura 3.21: Código de la definición de la función *setup* del firmware de la placa ESP32.

3.3 Diseño de la aplicación de celular

A través de la plataforma MIT App Inventor, el diseño de la aplicación se divide en la parte gráfica y la programación, la cual se realiza con bloques que representan las distintas acciones que pueden realizar los objetos que se agregaron a la parte gráfica. De esta manera, se permite programar, por ejemplo, cómo actuar ante el accionamiento de

algún botón. Para la comunicación MQTT fue necesaria la instalación de una extensión programada por Ullis Roboter Seite en su sitio web [7].

En la aplicación creada lo primero que se debe hacer es conectarse al Broker MQTT a través de la IP, y en el momento es que se establece la conexión la aplicación y la placa NUCLEO comienzan a intercambiar información como el estado de la tranquera o el modo de funcionamiento del sistema. En la parte superior siempre se encuentra la sección de la conexión MQTT, permitiendo conectarse y desconectarse. Justo debajo se encuentra la sección de control manual de la tranquera, que permite abrir o cerrar la tranquera siempre que no se encuentre forzado el modo de sólo lectura, y en donde además se muestra el estado actual de la tranquera. En la parte inferior se encuentra la barra de navegación, que permite cambiar entre las pestañas de *Lectura*, *Animales* y *Apartar*. En las figuras 3.22 a 3.28 se muestran las distintas pestañas de la aplicación.

En la pestaña de *Lectura* se muestran los datos de la última lectura hecha por el lector RFID, que incluyen el ID y el peso directamente de la balanza (si está encendida), y los datos del último peso guardado, sexo y raza si el animal leido ya se encontrara en la base de datos. Permite además actualizar el peso guardado si se recibió una nueva medición de peso.

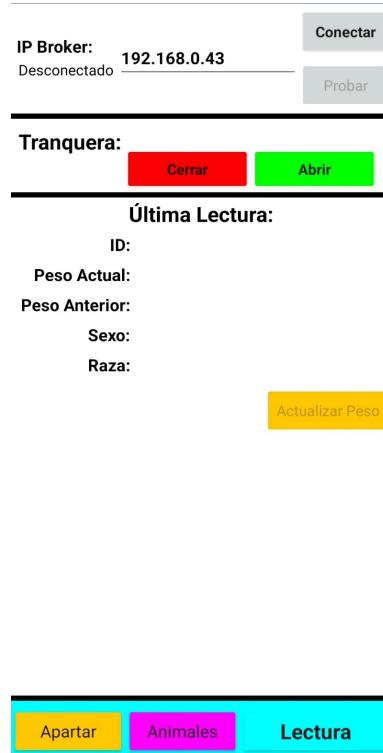


Figura 3.22: Pestaña *Lectura* de la aplicación de celular.

En la pestaña de *Animales* se puede ver una lista de los animales de la base de datos, y desde esta pestaña se puede acceder a la de *Nuevo Animal* para agregar un animal a la base de datos, la de *Editar* para realizar cambios en algún campo de un animal de la base de datos, y a la de *Ver* para ver los detalles de un animal de la base de datos. También permite eliminar animales de la base de datos. En las pestañas *Nuevo Animal* y *Editar* se permite también agregar razas.

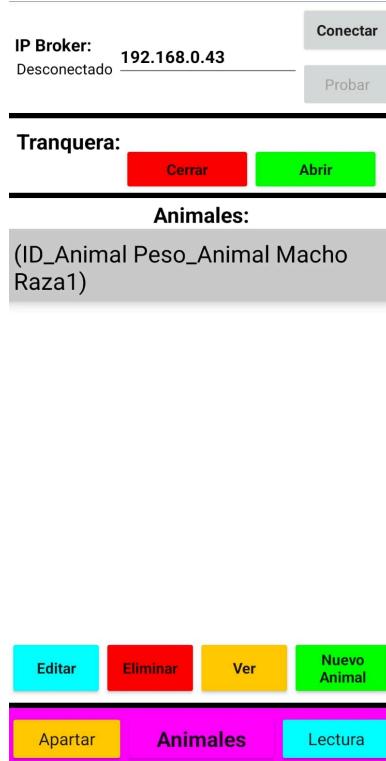


Figura 3.23: Pestaña *Animales* de la aplicación de celular.

**Memoria del Trabajo Final del
Seminario de Sistemas Embebidos
Sr. Augusto Villacampa Horta**



The screenshot shows the 'Nuevo Animal' (New Animal) tab of a mobile application. At the top, there is a header with 'IP Broker: 192.168.0.43' and buttons for 'Conectar' (Connect) and 'Probar' (Test). Below this is a section labeled 'Tranquera:' (Cage) with 'Cerrar' (Close) and 'Abrir' (Open) buttons. The main area is titled 'Nuevo Animal:' (New Animal). It contains fields for 'ID:' (ID), 'Peso:' (Weight), 'Sexo:' (Sex), and 'Raza:' (Breed). There is also a button 'Agregar Raza' (Add Breed) and a 'Limpiar todo' (Clear all) button. At the bottom, there are buttons for 'Agregar Nueva Raza' (Add New Breed), 'Usar Última Lectura' (Use Last Reading), 'Cancelar' (Cancel), and 'Aceptar' (Accept). A navigation bar at the very bottom includes 'Apartar' (Move aside), 'Animales' (Animals) in a pink box, and 'Lectura'.

Figura 3.24: Pestaña *Nuevo Animal* de la aplicación de celular.



The screenshot shows the 'Editar Animal' (Edit Animal) tab of the mobile application. It has a similar header and layout to Figure 3.24. The main area is titled 'Editar Animal:' (Edit Animal) and contains fields for 'ID:' (ID), 'Peso:' (Weight), 'Sexo:' (Sex), and 'Raza:' (Breed). It includes a 'Agregar Raza' (Add Breed) button and a 'Limpiar todo' (Clear all) button. At the bottom, there are buttons for 'Agregar Nueva Raza' (Add New Breed), 'Cancelar' (Cancel), and 'Aceptar' (Accept). A navigation bar at the very bottom includes 'Apartar' (Move aside), 'Animales' (Animals) in a pink box, and 'Lectura'.

Figura 3.25: Pestaña *Editar* de la aplicación de celular.

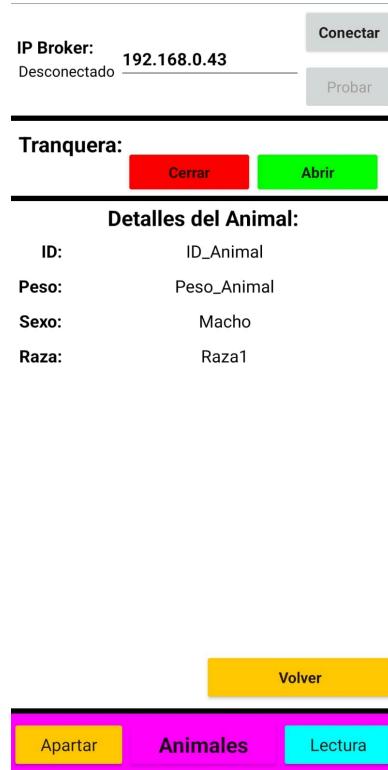


Figura 3.26: Pestaña *Ver* de la aplicación de celular.

Finalmente en la pestaña de *Aparte* se permite configurar un aparte automático a partir de las lecturas del módulo RFID. En esta implementación se pueden realizar apartes según el peso o según el sexo. En el primer caso, se puede configurar si se desea que la tranquera se abra cuando el peso es superior o inferior a cierto valor, mientras que al apartar por sexo se decide si la tranquera debe abrirse cuando el animal es macho o hembra. En ambos casos se define qué debe hacerse en caso de realizar una lectura de un animal y que el parámetro de aparte no esté disponible. Este sería el caso si, por ejemplo, se realizara un aparte por sexo y se leyera una tarjeta o caravana de un animal que no esté en la base de datos, y por lo tanto no se sepa su sexo. En el caso de apartar por peso, la aplicación indica si se está usando la medición actual (si la balanza está encendida) o si se está usando el último peso guardado.



Figura 3.27: Pestaña *Aparte por Peso* de la aplicación de celular.

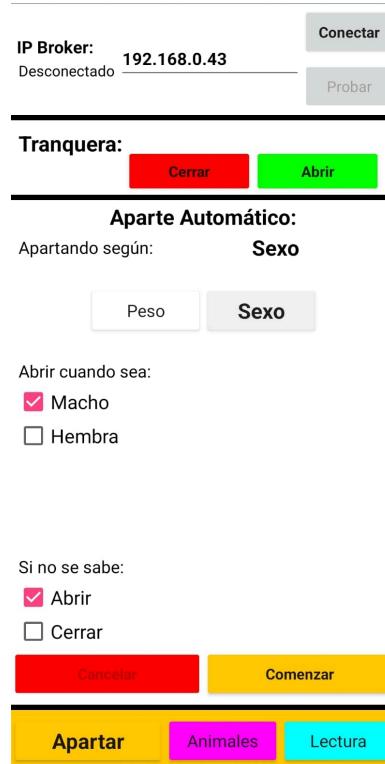


Figura 3.28: Pestaña *Aparte por Sexo* de la aplicación de celular.

3.4 Broker MQTT

Para que exista comunicación entre la placa NUCLEO y la aplicación de celular es necesario que ambos dispositivos (clientes) estén conectados a un mismo Broker MQTT. Para esta implementación se decidió instalar *Mosquitto* en una computadora, de modo que aloje un servidor local de MQTT. Trabajando de este modo, la computadora, la placa ESP32 y el celular deben estar conectados a una misma red Wi-Fi, y los clientes se deben conectar al servidor MQTT de la computadora a través de la IP local asignada a la computadora. Para la configuración del servidor, se decidió habilitar que cualquier dispositivo pueda enviar mensajes y suscribirse a cualquier tópico, y se configuró el servidor en el puerto 1883.

CAPÍTULO 4

Ensayos y resultados

4.1 Desarrollo y pruebas de funcionamiento

Durante el desarrollo de este trabajo la forma de trabajo consistió en la programación por separado de los distintos módulos de software, acompañados de pruebas simples de hardware para validar esa primera versión del código. Esto se cumplió sobre todo para los módulos que definen las clases *Led*, *Tranquera*, *Scale*, *RFID*, *MQTT* y *ModeSelector*, así como para el código de la placa ESP32. Terminados estos módulos se procedió a la prueba conjunta de los distintos módulos, agregando partes, aumentando la complejidad, corrigiendo errores y modificando los módulos. A su vez se avanzó con el módulo de la clase *nonBlockingDelay* y el módulo *system*, para así llegar a la versión final del firmware en donde todo funciona correctamente.

Los primeros módulos en desarrollarse fueron los de las clases *Led*, *Tranquera* y *Scale*, ya que se trata de clases sencillas. Con estas se pudo verificar el correcto funcionamiento de la placa y de la comunicación con el entorno de desarrollo para cargar el programa. Para la prueba de estos módulos se usó el hardware mínimo necesario, como LEDs, pulsadores, o el potenciómetro para la balanza. En el caso de la clase *Scale* se usó la aplicación *CoolTerm* como monitor serial, para mostrar las lecturas del valor del potenciómetro.

La clase *nonBlockingDelay* fue cambiando durante todo el desarrollo del trabajo hasta llegar a la versión final, y las pruebas se realizaron con el módulo *system*, que es el que contiene el flujo principal del programa. Este módulo se modificó para probar las distintas funcionalidades de la clase y verificar que no hubiera errores.

Para el código relacionado al lector RFID se usó el módulo MFRC522, que tiene todo el código interno necesario para la comunicación con el lector. Se desarrolló la clase *RFID* como interfase, en la que se agregó un status, un delay, entre otros. Con estos módulos desarrollados se conectó el hardware del lector y se realizaron distintas pruebas de lectura, mostrando nuevamente los valores leídos usando *CoolTerm*. Ya que este módulo utiliza un delay no bloqueante, se pudo verificar que era posible realizar un bloqueo del lector durante un tiempo luego de una lectura.

Para las pruebas de funcionamiento del protocolo MQTT lo primero que se hizo fue asegurar el funcionamiento del servidor MQTT. Para esto, ya que se usó *mosquitto*, en la PC en la que se levantó el servidor se crearon también dos clientes, uno para enviar mensajes y otro suscrito a algún tópico y a la espera de mensajes. Con esto se pudo verificar que el servidor podría recibir y enviar mensajes con normalidad. Luego se usó una aplicación de celular llamada *MyMQTT* para enviar y recibir mensajes desde un dispositivo distinto.

Cuando se logró la comunicación con el celular se avanzó con el desarrollo del código de la placa ESP32, para poder establecer conexión entre esta y el celular. Para esta prueba se usó un LED de prueba conectado a la ESP32, configurando a la placa para encender o apagar este LED ante determinados mensajes recibidos por MQTT. De esta forma se pudo controlar este LED desde el celular, usando la PC como servidor MQTT.

Pudiendo ahora enviar mensajes desde el celular hasta la placa ESP32, se avanzó con el código de esta última para que pudiera reenviar los mensajes por UART hacia la placa NUCLEO, y se desarrolló la clase *MQTT* para poder recibir y enviar mensajes, realizar suscripciones, manejar los LEDs necesarios y leer el status de la conexión. Para las pruebas en este punto se buscó también controlar algunos LEDs pero esta vez conectados a la placa NUCLEO, y también reiniciar el servidor para verificar que la reconexión se realizara correctamente.

Para finalizar las pruebas de MQTT, luego de desarrollar la aplicación propia, esta reemplazó a la aplicación *MyMQTT* en la comunicación MQTT, pero esta vez con la aplicación configurada para reaccionar automáticamente ante ciertos mensajes. Las pruebas consistieron en controlar desde la nueva aplicación las distintas partes del dispositivo, y se incorporaron también los módulos previamente probados. Por lo tanto fue posible recibir en la aplicación las lecturas desde el lector RFID o de la balanza, y controlar manualmente desde la aplicación el estado de la tranquera.

Habiendo desarrollado los principales módulos y teniendo el trabajo casi completo lo siguiente fue avanzar con el módulo system, para escribir el flujo normal del programa. Durante esta etapa se desarrolló el último módulo: mode_selector. Este módulo controla si el sistema reacciona a comandos desde la aplicación de celular o los ignora, y para probarlo simplemente se verificó que al activar el modo de Solo Lectura el estado de la tranquera no cambiaba ante comandos desde el celular.

Con el desarrollo finalizado se pudieron realizar pruebas exhaustivas a cada módulo y funcionalidad programados, poniendo énfasis en el cumplimiento de los requisitos planteados.

En el repositorio en el que se encuentra disponible el trabajo se puede encontrar un video de demostración del funcionamiento de este prototipo.

4.2 Cumplimiento de requisitos

Luego de la finalización del trabajo se puede concluir que todos los requisitos propuestos en la sección 2.1 fueron cumplidos, habiendo además agregado funcionalidades extras, como podría ser el LED de prueba de conexión MQTT o el que indica la recepción de un nuevo mensaje.

Por otro lado se considera que los casos de uso propuestos en la sección 2.2 se corresponden con las funcionalidades conseguidas en el prototipo final.

CAPÍTULO 5

Conclusiones

5.1 Resultados obtenidos

Habiendo finalizado el desarrollo de este proyecto y habiendo cumplido todos los requerimientos planteados se considera que el prototipo final es un éxito. Se logró implementar todas las funcionalidades deseadas al tiempo que se cuidaron detalles como la consideración de posibles errores a partir de malos usos del usuario y creando módulos de código preparador para adaptarse a una implementación distinta, por ejemplo si se decidiera cambiar de microcontrolador.

Se considera también que la aplicación de celular es lo suficientemente sencilla e intuitiva para que cualquier usuario pueda utilizarla, lo cual se considera un requisito indispensable para la versión final del producto que pueda salir al mercado. Y aunque se tiene en cuenta que en esta versión lo más importante es la funcionalidad, cuidar estos aspectos permite desarrollar el resto del trabajo de forma que se simplifique en un futuro el retrabajo necesario para las siguientes versiones.

Por otro lado, este trabajo incluyó trabajar con diversos temas, como podría ser el desarrollo de aplicaciones móviles, o el uso del protocolo MQTT. Por lo tanto se concluye que al terminar este trabajo se cuenta con un nuevo y amplio abanico de conocimientos de estos ámbitos. Se agrega también a este aspecto el desarrollo de un firmware por primera vez en la plataforma *Keil Studio Cloud* y usando una placa desarrollada por *STMicroelectronics*, lo que permitió ampliar enormemente los conocimientos sobre el desarrollo de proyectos de Sistemas Embebidos.

Particularmente sobre el tema del protocolo MQTT, teniendo en cuenta los problemas sufridos y la necesidad del cambio de módulo, se considera un gran hito haber logrado que funcione correctamente. Se considera muy valioso también por lo útil que resulta este protocolo en este tipo de desarrollos.

También se considera un gran logro la implementación de la clase *nonBlockingDelay*, ya que se partió de un módulo de simples funciones, y se pudo adaptar para transformarlo en una clase, y luego para que fuera más intuitivo de usar, y finalmente para incluir interrupciones al finalizar los contadores y la posibilidad de agregar funciones de *Callback* externar, lo que hace a este módulo muy útil para todo tipo de implementaciones.

Finalmente, lo que se considera más valioso de todo es haber logrado plasmar la idea inicial de forma realmente fiel. Sobre todo se destaca esto ya que la idea original surge de la sugerencia por parte de un veterinario que trabaja frecuentemente en mangas y consideró que un sistema como este podría ser muy útil. Conseguir un producto que satisfaga una necesidad explícitamente planteada por un potencial usuario final dota al proyecto de un enorme potencial para transformarse en un producto exitoso en el mercado.

5.2 Próximos pasos

A partir de lo desarrollado en este trabajo y teniendo en cuenta los logros conseguidos, se considera que este prototipo tiene mucho margen de mejora en varios aspectos. En el aspecto del firmware y hardware existen muchas cosas que podrían agregarse o cambiarse. Por ejemplo, podría querer agregarse un display para poder mostrar determinados aspectos del sistema sin la necesidad de un celular. También se podría proponer cambiar la placa NUCLEO por otro modelo que incluya lo necesario para el manejo de conexiones Wi-Fi, y de esta forma dejaría de ser necesaria la placa ESP32, lo que podría ahorrar dinero y espacio, al tiempo que disminuiría la tasa de fallas por contar con menos componentes y conexiones. Sería deseable también el desarrollo de una placa de circuitos impresa para una versión más avanzada del proyecto.

Desde el lado de la aplicación de celular las posibilidades son aún mayores, ya que lo esencial es únicamente la comunicación con el microcontrolador de ciertos datos como el ID de lectura y el peso medido en la balanza, y los comandos de control de la tranquera. De forma independiente al firmware y hardware se podría desarrollar mucho más la aplicación, de modo que permita agregar muchos otros parámetros a la base de datos, habilitando así la opción de realizar apartes bajo muchos otros criterios, o la visualización de ciertas estadísticas si el usuario tiene la constancia de actualizar frecuentemente los datos. Podrían también agregarse funcionalidades adicionales como un modo de vacunación, en donde el usuario configure que está vacunando y que a cada animal que pase por el lector se le actualicen los datos de forma de reflejar que recibió determinada vacuna.

Como próximos pasos se considera que sería esencial mostrar este prototipo a posibles usuarios, de modo que cumpla su propósito de prueba de concepto y permita tener opiniones que aporten a realizar cambios beneficiosos. A continuación se debería implementar todos los cambios que se consideren necesarios y adaptar el prototipo de forma de reemplazar el lector RFID actual por uno que se pueda usar en una manga, y el potenciómetro por una balanza real. También se debería desarrollar una placa de circuitos impresa según la versión del hardware a la que se llegue en esta etapa. Con esto se podría

tener una siguiente versión del prototipo que permita hacer pruebas en un ambiente real, y de esta forma corregir errores de implementación y de concepto, acercándose a una versión del producto que sea del agrado de los usuarios. Paralelamente se deberían desarrollar las mejoras en la aplicación móvil. Por otro lado, se debe realizar el desarrollo completo del control de una tranquera real, de forma de reemplazar los LEDs que se usan actualmente por un relé que controle un motor real y realice las aperturas y cierres de la tranquera. Este aspecto del desarrollo conlleva sus propias dificultades y obstáculos. También se podría realizar un desarrollo sobre la alimentación de todo el sistema, teniendo en cuenta que es posible que el lugar en el que se implemente no cuente con acceso a la red eléctrica. Esto podría incluir el desarrollo de un sistema de paneles solares y baterías, y los cálculos necesarios para alimentar sobre todo al motor que controla la tranquera.

Bibliografía

[1] Censo Nacional Agropecuario. 2018, realizado por INDEC. Resultados Definitivos.

[Online]. Available:

indec.gob.ar/ftp/cuadros/economia/cna2018_resultados_definitivos.pdf

[2] ¿Cuántos trabajadores registrados existen en el sector agropecuario? Por Matías Longoni. 2020. Bichos de Campo. [Online]. Available:

<https://bichosdecampo.com/cuantos-trabajadores-registrados-existen-en-el-sector-agropecuario/>

[3] La Argentina agropecuaria vista desde las provincias: Un análisis de los resultados preliminares del CNA 2018. [Online]. Available:

https://facaweb.uncoma.edu.ar/wp-content/uploads/2021/08/6_giberti.pdf

[4] Farmquip [Online]. Available: <https://www.farmquip.com.ar/index/>

[5] Brete y aparte automático, Farmquip + Cattler [Online]. Available:

<https://www.youtube.com/watch?v=1L2riEQJHxE>

[6] Módulo de software para el uso del chip MFRC522, por Martin Olejar. [Online].

Available: <https://os.mbed.com/users/AtomX/code/MFRC522/>

[7] Extensión para MIT App Inventor para comunicación MQTT, programada por Ullis Roboter Seite [Online]. Available:

<https://ullisroboterseite.de/android-AI2-PahoMQTT-en.html>