



# **Automatización y control de bebederos automáticos para especies animales usando LoRa**

2 de septiembre de 2021

## **TRABAJO FIN DE GRADO**

Grado en Ingeniería en Sistemas de  
Telecomunicaciones

**Autora:** Lucía Francoso Fernández

**Tutor:** Juan Pascual García

# Índice

Índice de figuras	9
Índice de tablas	10
Agradecimientos	11
<b>1 Introducción</b>	<b>13</b>
1.1 Contexto y justificación del trabajo . . . . .	13
1.1.1 Caso de aplicación . . . . .	14
1.2 Objetivos del trabajo . . . . .	15
1.3 Enfoque y método seguido . . . . .	15
1.4 Planificación del trabajo . . . . .	15
1.5 Breve sumario de productos obtenidos . . . . .	15
1.6 Breve descripción de los capítulos restantes de la memoria . . . . .	15
<b>2 Estado del arte</b>	<b>16</b>
2.1 Contexto actual . . . . .	16
2.1.1 Ejemplos de proyectos Open Source . . . . .	16
2.1.1.1 Relacionados con alimentación autónoma de animales . . . . .	17
2.1.1.2 Relacionados con monitorización de proyectos por LoRa . . . . .	17
2.1.1.3 Relacionados con alimentación autónoma de dispositivos . . . . .	17
2.1.2 Determinación del tipo de TFG . . . . .	17
2.2 Resumen del capítulo . . . . .	18
<b>3 Diseño del sistema</b>	<b>19</b>
3.1 Funcionalidades a cubrir . . . . .	19
3.2 Búsqueda soluciones . . . . .	19
3.2.1 Alimentación autónoma del dispositivo . . . . .	19
3.2.2 Automatización y monitorización . . . . .	35
3.2.2.1 Microcontrolador . . . . .	35
3.2.2.2 Comunicación radio . . . . .	43
3.2.2.3 Comedero . . . . .	45
3.2.2.4 Bebedero . . . . .	51

3.2.3	Prototipo exterior . . . . .	58
3.3	Elección soluciones . . . . .	59
3.3.1	Alimentación autónoma del dispositivo . . . . .	59
3.3.1.1	Preparación de la alimentación autónoma del dispositivo . . . . .	60
3.3.2	Automatización y monitorización . . . . .	67
3.3.2.1	Entorno Arduino . . . . .	68
3.3.2.2	LoRa . . . . .	74
3.3.2.3	Comedero . . . . .	91
3.3.2.4	Bebedero . . . . .	94
3.3.3	Prototipo exterior . . . . .	98
3.4	Resumen del capítulo . . . . .	102
<b>4</b>	<b>Prototipo y pruebas</b>	<b>103</b>
4.1	Ubicación . . . . .	103
4.2	Prototipo definitivo . . . . .	108
4.2.1	Resumen del funcionamiento del prototipo definitivo . . . . .	108
4.2.1.1	Comedero . . . . .	108
4.2.1.2	Bebedero . . . . .	110
4.2.2	PCB . . . . .	111
4.2.3	Protocolo . . . . .	111
4.2.4	Imágenes del prototipo final . . . . .	115
4.2.5	Configuración del gateway y página web . . . . .	134
4.3	Pruebas . . . . .	137
4.3.1	Pruebas de campo . . . . .	137
4.3.2	Pruebas de interior . . . . .	144
4.3.2.1	LoRa . . . . .	144
4.3.2.2	Bebedero . . . . .	145
4.3.2.3	Comedero . . . . .	145
4.3.2.4	Sistema de alimentación autónoma . . . . .	145
4.4	Comentarios sobre los resultados de las pruebas . . . . .	146
4.5	Presupuesto . . . . .	147
4.6	Resumen del capítulo . . . . .	148
<b>5</b>	<b>Conclusiones y líneas futuras</b>	<b>149</b>

<b>Glosario</b>	<b>150</b>
<b>Bibliografía</b>	<b>154</b>
Enlaces y referencias . . . . .	154
Imágenes . . . . .	156
<b>Anexos</b>	<b>159</b>
Anexo I . . . . .	160
Anexo II . . . . .	161
Anexo III . . . . .	184

# Índice de figuras

1	Logos de la protectora de Patitas Unidas Los Alcázares [21] y de los Objetivos de Desarrollo Sostenible (ODS) [22], respectivamente. . . . .	18
2	Curva de la carga/descarga de una batería [9]. . . . .	22
3	Comparación densidad de energía para diferentes baterías [9]. . . . .	24
4	Curva de descarga de una celda Li-ion y una celda Ni-Cd/Ni-MH (caída de tensión conforme aumentan las horas de uso de la batería) [9]. . . . .	25
5	Capas y componentes que componen un panel solar [21]. . . . .	26
6	Dibujo representativo del efecto fotovoltaico [21]. . . . .	27
7	Curvas J-V e I-V de una célula fotovoltaica . . . . .	28
8	Comportamiento de un panel solar formado por dos células solares idénticas [27]. . . . .	29
9	Curva P-V de una célula solar, indicando el punto MPP [3]. . . . .	31
10	Circuito equivalente de una célula fotovoltaica, en el caso ideal (a) y considerando pérdidas (b) [24]. . . . .	32
11	Curva J-V de una célula fotovoltaica, mostrando el efecto de aumentar o disminuir el valor de las resistencias $R_s$ y $R_p$ o $R_{sh}$ ( <i>shunt</i> ) [8]. . . . .	33
12	Circuito equivalente de una célula fotovoltaica con un modelo de dos diodos, donde $n_1$ y $n_2$ son los factores de idealidad e $I_{d1}$ y $I_{d2}$ las corrientes que atraviesan cada diodo [24]. . . . .	33
13	Variación en la curva I-V ante la radiación solar. . . . .	34
14	Variación en la curva I-V ante la temperatura ambiente . . . . .	35
15	Relación entre temperatura ambiente, radiación solar, con la potencia de salida de una célula PV (15a) y temperatura de ésta (15b) [28]. . . . .	36
16	Variación de la potencia de salida de una célula PV dependiendo de la hora del día, la temperatura y la radiación solar [5]. . . . .	37
17	Comparativa entre LPWAN, en términos de alcance y ancho de banda, con respecto a otras comunicaciones inalámbricas [48]. . . . .	44
18	Partes principales de las que está conformado un servomotor [30]. . . . .	46
19	Partes exteriores de las que está conformado un servomotor, ejemplo del fabricante <i>Parallax</i> [31]. . . . .	47
20	Combinaciones de color posibles para los cables de un servomotor, dependiendo del fabricante [29]. . . . .	49
21	Captura de un sensor ultrasónico HC-SR04 [60]. . . . .	52
22	Esquema de funcionamiento de un sensor ultrasónico HC-SR04 [61]. . . . .	52
23	Captura de sensores de flotación [15]. . . . .	53

24	Esquema de un switch conectado a un microcontrolador con una resistencia <i>pull-down</i> [51]. . . . .	54
25	Imágenes que muestran las partes fundamentales de las que consta un relé [18]. . . . .	55
26	Ejemplo de circuito que emplea relé y un transistor entre relé y pin 9 de un arduino [19] . . . . .	56
27	Esquema general de la alimentación elegida para este proyecto. . . . .	59
28	Esquema general de la alimentación elegida para este proyecto (TP4056 con boost incluido). . . . .	59
29	Detalle del kit de protección para una batería 18650, a falta de tubo termoretráctil y contactos para los polos [24]. . . . .	61
30	Esquema de la protección que se añade a las pilas Li-ion [23]. . . . .	61
31	Detalle de la PCM de protección de la batería 18650 [24]. . . . .	62
32	Fotos de la batería 18650 usada (sin protección, y con protección añadida). . . . .	62
33	Ejemplo de un TP4056 simple [47]. . . . .	63
34	Ejemplo de un TP4056 con boost [26]. . . . .	64
35	Elementos que formarán nuestro conglomerado provisional de alimentación del dispositivo. . . . .	65
36	Vistas del conglomerado de alimentación ya terminado. . . . .	66
37	Vistas del panel solar usado. . . . .	66
38	Conjunto global del sistema de alimentación autónoma creado . . . . .	67
39	Pinout de un Arduino nano. . . . .	70
40	Capturas del transceptor LoRa E32-868T30D [1]. . . . .	76
41	Captura de las antenas usadas [20]. . . . .	76
42	Capturas del transceptor LoRa E22-900T22D [2]. . . . .	77
43	Configuración de los parámetros de la red a simular en Radio Mobile, dentro de <i>Propiedades de las redes</i> . . . . .	78
44	Configuración de la topología de la red a simular en Radio Mobile, dentro de <i>Propiedades de las redes</i> . . . . .	79
45	Configuración de los sistemas de la red a simular en Radio Mobile, dentro de <i>Propiedades de las redes</i> . Ejemplo para transceptor LoRa E22, transmitiendo a 14 dBm. . . . .	80
46	Configuración de los miembros de la red a simular en Radio Mobile, dentro de <i>Propiedades de las redes</i> . Ejemplo para transceptor LoRa E22, transmitiendo a 14 dBm. . . . .	81
47	Enlace radio creado en Radio Mobile para simular la red creada entre la casa de campo y el refugio. . . . .	81
48	Resultado de la simulación del enlace radio creado en Radio Mobile. . . . .	82

49	Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E32 transmitiendo a 30dBm (1W). Enlace descendente: campo - refugio. . . . .	83
50	Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E32 transmitiendo a 30dBm (1W). Enlace ascendente: refugio - campo. . . . .	83
51	Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E32 transmitiendo a 21dBm (125 mW). Enlace descendente: campo - refugio. . . . .	84
52	Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E32 transmitiendo a 21dBm (125 mW). Enlace ascendente: refugio - campo. . . . .	84
53	Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E22 transmitiendo a 20dBm (100 mW). Enlace descendente. . . . .	85
54	Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E22 transmitiendo a 20dBm (100 mW). Enlace ascendente. . . . .	85
55	Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E22 transmitiendo a 10dBm (10 mW). Enlace descendente. . . . .	86
56	Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E22 transmitiendo a 10dBm (10 mW). Enlace ascendente. . . . .	86
57	Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E32 transmitiendo a 21dBm (enlace descendente), con pérdidas de 1 dB debido al alargador SMA. . . . .	87
58	Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E32 transmitiendo a 21dBm (enlace ascendente), con pérdidas de 1 dB debido al alargador SMA. . . . .	88
59	Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E22 transmitiendo a 10dBm (enlace descendente), con pérdidas de 1 dB debido al alargador SMA. . . . .	89
60	Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E22 transmitiendo a 10dBm (enlace ascendente), con pérdidas de 1 dB debido al alargador SMA. . . . .	89
61	Imagen del servomotor elegido para conformar el prototipo inicial de nuestro sistema de alimentación para animales [9] . . . . .	91
62	Capturas de la solución del movimiento de pienso del usuario kitlaan en <i>Thingiverse</i> [34]. . . . .	92
63	Pieza PVC tipo T 87°[11] . . . . .	92
64	Ejemplos de piezas PVC que pueden ser empleadas en el prototipo final. . . . .	93
65	Esquema simplificado de los cables que usa un servomotor MG996R y de su <i>duty cycle</i> [10] . . . . .	93
66	Esquema de las conexiones servo-Arduino realizado en <i>Fritzing</i> . . . . .	94

67	Esquema de las conexiones de sensores y actuadores usados para el bebedero con Arduino realizado en <i>Fritzing</i> . . . . .	95
68	Captura de la bomba de agua usada en el prototipo [16] . . . . .	96
69	Captura del relé usado en el prototipo [17] . . . . .	96
70	Captura de un fragmento de tubo de silicona (ejemplo) [18] . . . . .	99
71	Capturas del diseño e impresión en 3D de la pieza creada para la sujeción del tubo de silicona al bebedero. . . . .	99
72	Foto del bidón usado en el prototipo. . . . .	100
73	Capturas del cuenco y ganchos de sujeción usados para comedero y bebedero [35]. . . . .	100
74	Capturas de una caja estanca (ejemplo) [19]. . . . .	101
75	Fotos de alargadores SMA usados. . . . .	102
76	Foto del refugio de Patitas Unidas Los Alcázares. . . . .	103
77	Foto de las jaulas existentes en refugio de Patitas Unidas. . . . .	104
78	Detalle del exterior de una de las jaulas existentes en refugio de Patitas Unidas. . . . .	104
79	Detalle del interior de una de las jaulas existentes en refugio de Patitas Unidas. . . . .	105
80	Sistema existente en el refugio de Patitas Unidas para abrevar a los animales. . . . .	106
81	Sistema existente en el refugio de Patitas Unidas para alimentar a los animales. . . . .	106
82	Esquema resumen de la funcionalidad de comedero en el prototipo definitivo. . . . .	108
83	Esquema resumen de la funcionalidad de bebedero en el prototipo definitivo. . . . .	110
84	Foto de la PCB usada en el prototipo final. . . . .	111
85	Imagen del interior de la caja, con PCB y componentes. . . . .	115
86	Imagen del interior de la caja con servomotor conectado. . . . .	116
87	Imagen del interior de la caja con servomotor conectado (zoom). . . . .	117
88	Imagen del exterior de la caja (detalle de las abrazaderas). . . . .	118
89	Imagen del exterior de la caja con servomotor conectado. Se muestra, además, parte de los conductos para mover el pienso, y ubicación del servomotor. . . . .	119
90	Imagen del exterior de la caja con servomotor conectado. Zoom de sujeción a pieza PVC. . . . .	120
91	Detalle de la hélice v3 dentro de la pieza PVC tipo T. . . . .	121
92	Aproximación de cómo quedaría la hélice v3 dentro del brazo PVC. . . . .	122
93	Imagen global del comedero. . . . .	123
94	Zoom de las piezas que conforman el sistema de conexión entre reserva y comedero. . . . .	124
95	Detalle de las hélices impresas en 3D para, a través del movimiento del servomotor, hacer llegar el pienso al comedero. . . . .	125

96	Interior de la caja con conexiones a servo, a bomba y a switches. . . . .	126
97	Interior de la caja con conexiones a servo, a bomba y a switches (zoom). . . . .	127
98	Imagen del interior de la reserva de agua (detalle de los switches). . . . .	128
99	Imagen del interior de la reserva de agua (detalle de los switches y bomba). . . . .	129
100	Detalle de la bomba de agua usada en la reserva de agua. . . . .	130
101	Exterior de la reserva de agua (detalle de perforaciones para switches). . . . .	131
102	Exterior de la reserva de agua (detalle de perforaciones para conexiones de la bomba y ventilación). . . . .	132
103	Prototipo conjunto del sistema de alimentación creado, formado por bebedero y comedero. . . . .	133
104	Captura de la página web creada (primera versión), en concreto, de la página principal, donde se muestran los botones, los links y el mensaje de aviso. . . . .	135
105	Captura de la página web creada (primera versión), en concreto, de la página principal, donde se muestra un ejemplo de envío de mensajes y recepción, visible en la consola de la web. . . . .	136
106	Montaje en protoboard para verificar el funcionamiento y alcance del transceptor LoRa E32-868T30D . . . . .	138
107	. . . . .	139
108	Recorrido realizado con el receptor para probar su alcance . . . . .	139
109	Esquema resumen del flujo de la información en ambos sentidos del enlace LoRa, señalando las vías de comunicación entre bloques fundamentales del enlace. . . . .	140
110	Situación de la estación del campo para la prueba de cobertura entre campo y refugio	141
111	Muestras de la consola de la web <i>Open Pet Feeder</i> durante la prueba de alcance en el entorno definitivo (estación del refugio) . . . . .	142
112	Fotos tomadas el día de la prueba de cobertura, con potencia de transmisión 21dBm.	143
113	Imagen resumen del funcionamiento del sistema definitivo . . . . .	144

## Índice de tablas

## Agradecimientos

Primero, me gustaría agradecer a mi tutor, Juan Pascual García, por aceptar este reto junto a mí, y apoyarlo en todo momento. Fue un proyecto diferente, algo arriesgado, y siempre confió en que lo sacaríamos hacia delante. Muchas gracias por tu apoyo a distancia, debido a la Covid-19, que a pesar de todo siempre he obtenido tu ayuda y tu feedback a tiempo, y confianza en todo proceso requerido en este proyecto, pilares fundamentales en cualquier emprendimiento.

Me gustaría, además, agradecer a mi familia, en especial a mis padres, Francisco y María Dolores, ya que es difícil entender que decida estar un año sin apenas créditos matriculados, la importancia de aprender de manera complementaria a la universidad, de buscar (y encontrar) aquello que me gusta dentro de esta carrera que, a pesar de las dificultades, siento que no podría haber elegido otra mejor. Sin ese apoyo, todo habría sido mucho más difícil, y siendo mi situación poco común, agradezco que lo hayan comprendido. A esto se suma su aportación económica, sin ella no habría podido realizar este proyecto, el cual he sentido en todo momento muy personal y muy bueno para mí, en tanto a nivel profesional, como personal como para la protectora que será beneficiaria de este dispositivo. Agradecer a mis hermanas Isabel y Claudia, que a pesar de las diferencias, y a pesar de ser la mayor, tienden a reforzarme en los momentos difíciles, recalando mis virtudes; confiaron en este proyecto, y saben lo bueno que podría conllevar, ya que han venido innumerables veces conmigo al refugio, al igual que mi padre.

También agradecer al resto de mi familia, que en la distancia también me apoyan. En especial a mi tía Bárbara, que siempre me llama para darme ánimos con todo lo que hago y que confía en mí desde siempre. A mis primas Sonia y María, y a mi abuela Guadalupe, las cuales son ejemplos de mujeres admirables en mi familia y a las que quiero muchísimo. Especial mención a mi abuela, Lucía Figueredo, la cual falleció el año pasado y sé que estaría muy orgullosa de ver a su nieta convertirse en ingeniera, y creando este proyecto.

Debo agradecer también a mis suegros, Encarnación y Miguel Ángel, por dedicarme parte de su tiempo siempre que pueden, y en especial, en el desarrollo del prototipo exterior; agradezco enormemente que me hayan prestado sus herramientas, ya que sin ellas el resultado no habría sido el mismo. Muchas gracias por permitirme instalar en vuestra casa el otro extremo de la comunicación, que ha permitido poder concebir este proyecto.

Me gustaría también agradecer a mi amiga Magdalena, mi amiga de toda la vida, la cual me apoya incondicionalmente, y me inspira en su lucha constante por mejorarse así misma y por ganarse la vida en lo que más le gusta, la música. A mi amiga Clara, la cual es una estupenda teleco, que siempre ha confiado en mí más que yo misma, y la cual me inspira por ser una persona fuerte, independiente y que tiene claro lo que quiere. A mi amigo Pablo, porque ni la distancia ni las exigentes carreras que cursamos nos han hecho perder el contacto, ni perder un vínculo que hace años se formó. Es un placer poder contar con estar tres personas.

Especial mención a mi compañero Enrique Fernández Sánchez, el cual ha sido también como mi

tutor, ayudándome muchísimo a comprender la importancia del Open Source, guiándome en el conocimiento sobre microcontroladores y LoRa, y en general, aconsejándome en todo lo que ha podido y más. Gracias a él, la introducción al mundo maker se me ha hecho más sencilla, mucho más consciente y sin excesivos agobios. A esto, añadir que su aportación económica me ha ayudado muchísimo a sacar este proyecto hacia delante. Sin duda, ajeno a este proyecto, aprenderé muchas cosas de él y gracias a él, y no me cabe la menor duda de que es un gran teleco y una gran persona.

Agradecer a Patitas Unidas Los Alcázares por permitirme instalar un dispositivo en sus instalaciones, y por comprender mi ausencia durante algunos meses donde el desarrollo de este proyecto era muy exigente. Especial mención a mi compañera Jen, con la que voy al refugio prácticamente siempre; sus ganas y actitud al tratar con los perros desde luego es una cualidad a admirar, y es una inspiración que recarga fuerzas para seguir luchando por los animales.

Por último, agradecer a todas esas personas que han subido a la nube su proyecto, su documentación y dejan todo ese desarrollo bajo licencia Open Source, permitiendo que gente como yo desarrolle su proyecto basándome en sus librerías, o me den ideas que sin duda impulsen el mío. Estarán debidamente mencionadas, ya que merecen ese crédito.

# 1 Introducción

Tras finalizar los estudios de grado en ingeniería en sistemas de telecomunicaciones, se requiere como último paso para la obtención del título la elaboración del *Trabajo Fin de Grado* (TFG). El TFG como tal tiene unos objetivos claros, los cuales son demostrar que se han adquirido las competencias básicas intrínsecas al grado, y que el alumno es capaz de seguir aprendiendo a partir de los conocimientos ya obtenidos, innovar, desenvolverse ante un problema determinado y, en resumen, saber llevar a cabo una investigación o proyecto que tenga como objetivo la resolución de dicho problema.

En este documento, se recoge el proyecto realizado como TFG, el cual va en línea con la filosofía y objetivos del *Trabajo Fin de Grado* en sí mismo. Se detallará el proceso de creación de un sistema de alimentación para animales automatizado, donde se monitorizan de manera remota el estado de los tanques de reserva de agua y pienso. Así, se pretende exponer el conjunto de elementos hardware y software que serán necesarios para monitorizar, automatizar y acceder a ciertos datos de forma remota empleando, principalmente, Arduino y LoRa.

## 1.1 Contexto y justificación del trabajo

Este proyecto ha sido concebido con el objetivo principal de ayudar a la preservación de la vida animal, ante el aumento de especies en extinción, sobre todo en lo que llevamos de siglo [1]-[4], y ante el hecho de que miles de animales domésticos siguen siendo abandonados al año en España [5]-[8].

Es por ello que se requiere de ayuda activa para paliar estos problemas. Las tareas de carácter solidario, en bastantes casos, no siempre cuentan con suficientes voluntarios; además, siendo un problema tan extendido y avanzado, el número de acciones que hay que llevar a cabo para aliviarlo es alto para un número limitado de voluntarios, los cuales deben incrementar considerablemente el tiempo que pasan realizando este tipo de tareas. Tanto si se trata de un refugio de animales domésticos abandonados, como de reservas naturales donde se intenta repoblar una especie, los animales dependen enteramente del trabajo de los voluntarios. Así pues, es de vital importancia la optimización de las tareas de voluntariado, su automatización e, incluso, control remoto, mediante la creación de herramientas que ayuden a reducir el tiempo que se destina a tareas rutinarias para poder utilizar ese tiempo a otras tareas (de rescate, o de financiación para el mantenimiento de las instalaciones y de los propios animales, por ejemplo).

Con el objetivo en mente de ayudar a la preservación de la fauna (y con ello, de la vida de los ecosistemas terrestres), evitando la extinción de especies y el abandono animal, se ha concebido y desarrollado este proyecto teniendo en cuenta los diferentes casos de uso (emplazamientos donde se podrá instalar el sistema, características del entorno), mejor adaptación a ellos, relación entre buenas prestaciones y bajo consumo, precio total del producto, o facilidad de uso por parte de un usuario medio. Es por ello que desde el principio se propone una serie de actuaciones a realizar, acorde a lo anteriormente mencionado, tales como:

- El diseño del sistema de alimentación será tal que el producto final pueda ser instalado no sólo en hogares, sino en sitios remotos, donde el acceso a recursos tales como la electricidad o Internet son escasos o inexistentes. Así pues, el dispositivo utilizará energía solar y baterías recargables, comunicación de bajo consumo y largo alcance y modo de ahorro de energía.
- El dispositivo final no será pesado ni voluminoso, facilitando así tanto su transporte como su manejo. No se dejará que los dispositivos electrónicos queden al alcance de los animales; de este modo se evitará que los animales puedan sufrir daño (o viceversa), y posibles problemas de humedad presentes en el entorno; se usarán protecciones adecuadas a la calidad de los componentes del dispositivo.
- El dispositivo contará con una pantalla OLED que permitirá ver a la persona que esté físicamente delante de él si funciona correctamente. También se permitirá el acceso a datos a personas interesadas que quieran consultarlos de manera remota.

### 1.1.1 Caso de aplicación

Nuestro caso de aplicación será el entorno donde se desea situar uno de estos dispositivos para validar su funcionamiento, que en este caso se trata del refugio perteneciente a la protectora Patitas Unidas Los Alcázares, situado en el término municipal de Torre Pacheco (Murcia). La elección de esta ubicación se fundamenta en una serie de razones:

- Al ser voluntaria para esta protectora, conozco bien sus necesidades, es decir, qué puede ser de utilidad para la protectora en el refugio y qué soluciones se han probado para determinados problemas que han ido surgiendo; además, conozco bajo qué condiciones climáticas es vulnerable y qué necesidades esporádicas emergen bajo dichas condiciones. Teniendo en cuenta la zona geográfica donde se ubica (Murcia), y los años de experiencia en el refugio, se ha detectado una problemática que se manifiesta durante períodos continuados de lluvias (lluvias torrenciales, DANA, gota fría), la cual consiste en la inundación de las zonas colindantes al refugio, inclusive carreteras de acceso, lo que impide llegar a él (cierre de carreteras, niveles altos de riesgo por precipitación, o el simple hecho de contar con grandes volúmenes de agua en la carretera que impiden la circulación segura por la vía). Es por ello que, mediante el desarrollo de este proyecto, se pretende ofrecer una solución que palie las consecuencias de no poder llegar al refugio bajo esas circunstancias, como es la alimentación de los animales.
- Se trata de un entorno sin electricidad y sin internet. Desarrollar un proyecto y probarlo en este tipo de entorno nos ayudará a la hora de extrapolarlo a otros emplazamientos donde la ausencia de este tipo de recursos supone también una limitación y un aspecto a tener en cuenta para definir y desarrollar el proyecto en sí mismo.
- Se puede establecer un enlace punto a punto, ya que se puede dejar fijo un equipo transmitiendo o recibiendo que, además, se conecte a internet (ya que es un recurso disponible) para cargar datos que reciba del otro extremo a un servidor conocido. Un equipo estará presente

en el refugio y el otro en una casa con internet y electricidad; esto significa que al menos este extremo será más controlable, y será este extremo el que subirá datos a la nube. Nos tendremos que preocupar más del otro extremo, donde no tendremos electricidad ni internet y donde situaremos los sensores que recogerán datos y realizarán la automatización.

## 1.2 Objetivos del trabajo

El objetivo fundamental del presente trabajo es el diseño de un sistema de alimentación para animales que permita la monitorización de los niveles de agua y pienso que se encuentran en depósitos de reserva, los cuales llenan un bebedero y un comedero, respectivamente. Con ello, se pretende automatizar el proceso de alimentación de animales, además del uso de la tecnología LoRa para tener acceso al estado del sistema de forma remota. Así pues, los objetivos concretos son:

- Realizar una aproximación a la tecnología LoRa.
- Diseñar el sistema de alimentación para animales.
- Realizar tanto simulaciones para el enlace LoRa que se creará entre transmisor y receptor, como cálculos teóricos que determinen si el enlace es posible.
- Construir el prototipo y probarlo en entorno controlado y, posteriormente, en entorno real.
- Crear una plataforma de representación de datos.

## 1.3 Enfoque y método seguido

## 1.4 Planificación del trabajo

A continuación, se mostrará una tabla resumen, elaborada durante el proceso de concepción del proyecto (etapa inicial), de manera que, por una parte, se pudiera tener una idea de los recursos necesarios para poder dar comienzo al proyecto (tanto materiales como software); por otra parte, serviría, a grosso modo, para conocer y planificar una serie de pasos desde esa concepción hasta una presentación de un prototipo funcional, resultado del desarrollo del proyecto.

[Insertar tabla con Planificación inicial]

## 1.5 Breve sumario de productos obtenidos

## 1.6 Breve descripción de los capítulos restantes de la memoria

## 2 Estado del arte

En el capítulo anterior ya se planteó la motivación y una aproximación a lo que se pretende realizar en este proyecto, pero en este punto se expone con más detalle la situación actual en el ámbito del control y automatización de sistemas de alimentación para animales (dentro del marco Open Source), así como proyectos relacionados con este ámbito y otros ejemplos que emplean Arduino y/o LoRa.

### 2.1 Contexto actual

Existen muchos proyectos Open Source relacionados con la alimentación automática o semiautomática de animales domésticos, principalmente gatos y perros. Sin embargo, no existen (o al menos, no se han encontrado) proyectos que usen LoRa como tecnología radio, sino que emplean la red local WiFi del hogar donde se sitúe el dispositivo de alimentación; es decir, se trata de entornos con electricidad e internet, y no del entorno en el que trabajará nuestro proyecto.

A pesar de ello, se ha visualizado este tipo de proyectos y tenido en cuenta para el desarrollo del prototipo en tanto a apariencia externa, facilidad de uso, eficiencia de los componentes, o, incluso, opciones para mover el pienso desde la reserva hasta el comedero del animal, o el agua desde la reserva hasta el bebedero. No se realizará comparativa con dispositivos autónomos de alimentación para animales comerciales, ya que se pretende elaborar un proyecto Open Source, que nada tiene que ver con soluciones comerciales propietarias; además, los objetivos de este proyecto (principalmente establecer unas bases para investigar sobre este tipo de dispositivos que ayudan a animales y voluntarios), requerimientos y tecnologías empleadas son totalmente diferentes.

Aunque se hablará con más detalle en la sección 3, en el momento de elaborar la planificación para este proyecto, se tuvo que preconcebir una serie de pasos a seguir. Es en el momento de la concepción del proyecto, mientras se realiza la planificación, cuando se realizó una primera búsqueda de proyectos Open Source, tanto relacionados con alimentación autónoma de animales, como monitorización por LoRa de proyectos y alimentación autónoma de los mismos. No se centró la búsqueda en el primer tópico exclusivamente, ya que, como hemos comentado, no se han encontrado proyectos que engloben alimentación autónoma para animales, monitorización por LoRa y alimentación autónoma del dispositivo.

#### 2.1.1 Ejemplos de proyectos Open Source

A continuación, se comentarán brevemente algunos ejemplos de proyectos Open Source, relacionados con los tópicos que se han enumerado anteriormente y que, en opinión propia, merecen reconocimiento por la gran labor de investigación y divulgación que han realizado con dichos proyectos.

### 2.1.1.1 Relacionados con alimentación autónoma de animales

A través de la plataforma [Thingiverse](#), son varios los usuarios que han compartido sus creaciones relacionadas con alimentación autónoma de animales. Destacaremos al usuario [kitlaan](#), con su dispensador automático de pienso para gatos ([Auger-based Cat Feeder](#)). Con este proyecto, este usuario, entre otros, facilita los archivos de impresión en 3D que usó para construir su dispensador de pienso; documenta su proyecto, especificando las piezas que usa y que no son impresas, así como un vídeo del funcionamiento del dispensador.

### 2.1.1.2 Relacionados con monitorización de proyectos por LoRa

El usuario [xreef](#), Renzo Mischianti, liberó uno de sus proyectos sobre monitorización vía LoRa, llamado [LoRa E32 for Arduino, ESP32 or ESP8266: Specs and Base Use](#). En este proyecto, prueba el transceptor LoRa E32-TTL-100 con dos microcontroladores, el microcontrolador Wemos D1 mini (ESP8266), y el Arduino Uno. Libera las conexiones y los sketches que emplea para probar dichos microcontroladores, así como una [librería](#) de elaboración propia para el E32-TTL-100, sin la cual la comunicación entre microcontrolador en transmisión y microcontrolador en recepción sería imposible (o deberíamos crearla nosotros).

### 2.1.1.3 Relacionados con alimentación autónoma de dispositivos

Para este tópico, se han visualizado muchos proyectos a través de YouTube, y vídeos explicativos de opciones de alimentación autónoma de dispositivos. Se han visualizado asiduamente vídeos de los creadores [GreatScott!](#) y [Andreas Spiess](#), aunque no han sido los únicos. Un proyecto que me gustaría remarcar, ya que fue de los primeros que me impulsaron a realizar este proyecto, es el de [Automating a Greenhouse with LoRa! \(Part 1\)](#), del creador [GreatScott!](#). En este proyecto no solo se usa LoRa para monitorizar un invernadero, sino que se hace uso de paneles solares para alimentar el proyecto.

## 2.1.2 Determinación del tipo de TFG

Por último, es importante comentar que este TFG está catalogado como *TFE Específico de Aprendizaje*; según la normativa sobre TFG específico de aprendizaje: “*aquellos TFE específicos ligados a la consecución de los Objetivos de Desarrollo Sostenible (ODS) a través de la vinculación de manera directa a una solicitud de servicio, a desarrollar por el estudiante en su TFE, realizada por una asociación o institución externa, sin ánimo de lucro*”. Esto es así, ya que:

- Cumple el Objetivo 15 de Desarrollo Sostenible (ODS), el cual trata de “*Gestionar sosteniblemente los bosques, luchar contra la desertificación, detener e invertir la degradación de las tierras, detener la pérdida de biodiversidad*” [41]. Con este proyecto, nos estamos enfocando en ayudar en la detención de la pérdida de biodiversidad, ya que, como se ha comentado anteriormente, se va a crear un dispositivo capaz de alimentar y abreviar animales en lugares

sin electricidad y sin conexión a internet, usando el menor número de dispositivos radio posible (siendo estos de largo alcance y bajo consumo). Instalando estos dispositivos en zonas de cuidado de animales podemos reducir el tiempo que dedican los voluntarios a las diferentes actividades (como hemos mencionado en el apartado 1.1), permitiendo usar ese tiempo en otras actividades igualmente prioritarias pero que difícilmente se pueden realizar en remoto, como el salvamento o rescate de animales, ubicación, repoblación o cuidados veterinarios.

- Existe una solicitud formal de la protectora que va a hacer uso del dispositivo creado con este proyecto, Patitas Unidas Los Alcázares [42] [43]. Dicha solicitud recoge los mínimos que debe tener en cuenta el proyecto, ya expuestos en el apartado 1.2.

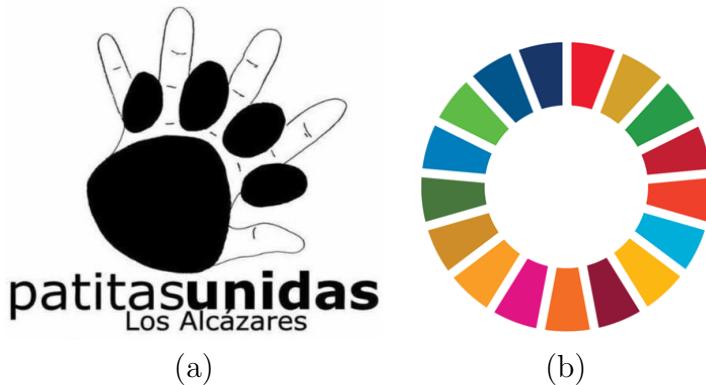


Figura 1: Logos de la protectora de Patitas Unidas Los Alcázares [21] y de los Objetivos de Desarrollo Sostenible (ODS) [22], respectivamente.

## 2.2 Resumen del capítulo

En este capítulo, se ha presentado el contexto actual en el que se presenta este proyecto, cómo se cataloga dentro del ámbito de los Trabajos Fin de Estudios, y qué proyectos similares Open Source existen en la actualidad. Durante la planificación de este proyecto, se realizó una búsqueda de proyectos Open Source, que si bien no realizan la totalidad de funcionalidades que queremos cubrir con este proyecto, de manera individual sí cubrían algunas de las funcionalidades de interés, como son la alimentación autónoma de animales, monitorización de proyectos a través de LoRa y alimentación autónoma de dispositivos. Se han mostrado ejemplos de dichos proyectos.

## 3 Diseño del sistema

A lo largo de este capítulo se pretende mostrar los procesos de concepción, análisis e investigación llevados a cabo para la creación del primer prototipo del sistema de alimentación. Para ello, primeramente, se exponen las funcionalidades que éste debe incorporar; una vez identificadas esas funcionalidades, se presta la realización de una investigación cuyo objetivo sea la búsqueda de opciones que permitan que cada funcionalidad anteriormente expuesta pueda materializarse. Para terminar el presente capítulo, se procederá a exponer las opciones que han sido escogidas, debidamente justificadas, para el diseño del prototipo provisional del sistema de alimentación.

### 3.1 Funcionalidades a cubrir

Teniendo claros los objetivos que se persiguen con este proyecto, es posible definir las funcionalidades que ofrecería el sistema global que se pretende crear, ya que a partir de esta definición podremos empezar a deducir y establecer qué componentes son necesarios para llevar a cabo dichas funcionalidades.

Definimos *tres funcionalidades principales*:

1. **Alimentación autónoma del dispositivo**
2. **Automatización y monitorización.** Esta funcionalidad se compone de cuatro grandes bloques:
  - Bloque microcontrolador
  - Bloque comunicación radio
  - Bloque comedero (sensores, actuadores)
  - Bloque bebedero (sensores, actuadores)
3. **Integridad mediante prototipo exterior.** Se entiende por *Integridad mediante prototipo exterior* todos aquellos elementos que usemos para dar forma al proyecto, evitando el uso de elementos demasiado temporales como pueden ser una protoboard o en general, componentes no definitivos (estos sólo se usarán en pruebas, no en producción). Con esta idea, se buscarán elementos que permitan constituir un primer prototipo usable, sin invertir dinero en exceso, que elimine componentes temporales no funcionales para usar en la ubicación real; es por ello, que se cuenta con la inversión en reservas, conexión de reserva a comedero/bebedero, diseño de una PCB, etc.

### 3.2 Búsqueda de soluciones para cada funcionalidad

#### 3.2.1 Alimentación autónoma del dispositivo

Se entiende por alimentación autónoma la obtención de energía suficiente para el correcto funcionamiento de un dispositivo o un sistema, y que dicha obtención de energía se realice de manera

cíclica e independiente o prácticamente independiente, es decir, no dependa de la acción y supervisión constante de una persona.

Para determinar la forma en la que alimentaremos nuestro proyecto es imprescindible conocer bien las necesidades energéticas que lo limitarán [13]; hay que determinar, por tanto, el consumo estimado de los componentes que conforman el proyecto. También es importante saber el tamaño y/o el destino de nuestro proyecto, para determinar si estará limitado a la hora de su transporte por una fuente de alimentación pesada, o si esta no limita en absoluto; será fundamental, además, conocer la ubicación de nuestro proyecto, es decir, si estará en el exterior o interior, si el ambiente será excesivamente frío, cálido o húmedo.

A continuación, se mencionarán algunas soluciones para alimentación autónoma. Primero, se realizará una introducción teórica a términos y definiciones que nos permitan entender su funcionamiento; después, se presentarán opciones dentro de la categoría, haciendo énfasis en sus ventajas y desventajas, y anotando para qué proyectos es recomendable su uso. Por último, se pueden mostrar datos comparativos entre las opciones dentro de la categoría para una mejor visualización de dichas diferencias.

## Baterías recargables

### *Términos y definiciones*

Las características eléctricas de una batería definen cómo actuará en el circuito global, y las características físicas tendrán un enorme impacto en el tamaño y peso global del producto al que va a alimentar. Es por ello que, a la hora de escoger una batería, hay que tener presente los requerimientos de alimentación del proyecto, ya que ésta debe ser capaz de suministrar suficiente energía para que funcione correctamente, sin que sus características físicas supongan una limitación. Algunos parámetros que hay que comprobar son la tensión nominal, la corriente, la capacidad y el material del que está hecha la batería, pero no son los únicos a tener en cuenta. Se presenta, a continuación, una lista más extensa de los parámetros más importantes de las baterías [9]:

- Una **celda** es un dispositivo electroquímico capaz de suministrar la energía que resulta de una reacción química interna a un circuito eléctrico externo.  
Una batería se compone de una o más celdas, conectadas en paralelo o en serie para obtener la capacidad de corriente/voltaje requerida (las baterías compuestas por celdas conectadas en serie son las más comunes).
- La **resistencia en serie equivalente (ESR)** es la resistencia interna presente en cualquier celda que limita la cantidad de corriente máxima que puede entregar.
- La **capacidad**, medida en amperio-hora (Ah), de una batería (o celda) es su figura de mérito más importante: se define como la cantidad de corriente que una batería puede entregar durante 1 hora antes de que el voltaje de la batería llegue al final de su vida útil.

- La **tasa “c”** es una corriente que es numéricamente igual a la clasificación Ah de la celda. Las corrientes de carga y descarga se expresan típicamente en fracciones o múltiplos de la tasa c.
- El **MPV (voltaje de punto medio)** es el voltaje nominal de la celda y es el voltaje que se mide cuando la batería se ha descargado el 50% de su energía total. La oscilación de voltaje máxima y mínima del valor nominal es una consideración de diseño importante: una curva de descarga ”más plana” significa menos variación de voltaje que el diseño debe tolerar.
- El **voltaje de celda medido al final de su vida útil** se llama **EODV**, que significa *Fin de voltaje de descarga* (algunos fabricantes se refieren a esto como **EOL** o **voltaje de fin de vida útil**). Cuando se carga al máximo, el voltaje real de la celda será más alto que el MPV. Al acercarse al punto EODV (final del voltaje de descarga), el voltaje de la celda será menor que el MPV.
- La **densidad de energía gravimétrica** de una batería es una medida de cuánta energía contiene una batería en comparación con su peso. Normalmente viene expresada en Watt-horas/kilogramo (Wh/kg).
- La **densidad de energía volumétrica** de una batería es una medida de cuánta energía contiene una batería en comparación con su volumen. Normalmente se expresa en Watt-horas/litro (Wh/l)
- Un **cargador de voltaje constante** es un circuito que recarga una batería obteniendo solo la corriente suficiente para forzar el voltaje de la batería a un valor fijo.
- Un **cargador de corriente constante** es un circuito que carga una batería al suministrar una corriente fija a la batería, independientemente del voltaje de la batería.

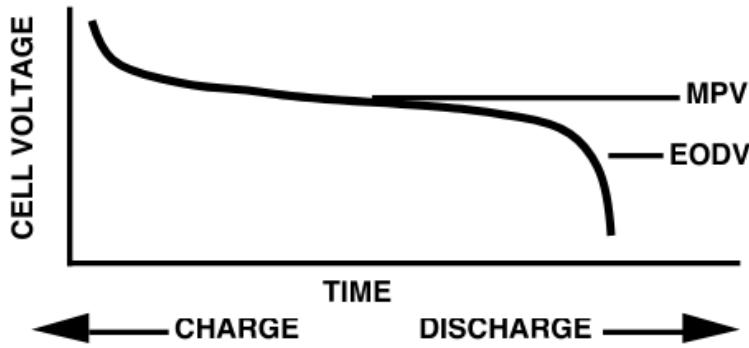


Figura 2: Curva de la carga/descarga de una batería [9].

### *Tipos de baterías recargables*

Las principales baterías recargables [10][11][12] que existen son:

1. *Lithium polymer* (LiPo). Son baterías de gran densidad de energía disponible, es decir, almacenan una gran cantidad de energía en comparación con el tamaño de la pila (se explica en *Términos y definiciones*). Es por esta razón por la que los teléfonos móviles, los portátiles y otros dispositivos ligeros en peso usan baterías LiPo. Sin embargo, esta alta densidad de energía lleva consigo un coste/inconveniente, y es que son caras, y no son sólo las baterías en sí; cuando se trabaja con baterías de litio, hay que realizar una inversión para conseguir una circuitería especial que mantenga segura la batería. Por ejemplo, cuando estas cargando una batería LiPo, hay que asegurarse de que nunca se excedan los 4.2V por celda; las baterías LiPo explotarían si las sobrecargaras. No solo la carga de la batería es delicada, sino que la descarga también lo es; si se descarga una batería LiPo por debajo de 3V por celda, se perderá permanentemente parte de su capacidad, e incluso pueden deformarse/hincharse. Finalmente, hay que estar pendiente constantemente de la temperatura, para evitar que la batería se sobrecaliente. A pesar de todo esto, es común usar baterías LiPo en proyectos ligeros y pequeños, siempre y cuando se tenga un buen circuito de carga, un buen circuito de descarga (que proteja a valores bajos de tensión, haciendo de corte en el mínimo de tensión de la batería) y un buen circuito de monitorización de temperatura.
2. *Lithium-ion* (Li-ion). Son muy similares a las baterías LiPo, pero a diferencia de estas, el electrodo negativo es líquido y el encapsulado y formato de presentación es totalmente diferente (las Li-ion suelen venderse sin protección y en apariencia son similares a unas pilas alcalinas AA). Una batería Li-ion almacena más potencia que las LiPo (mayor capacidad), ofrece mayores corrientes de descarga, es menos costosa de fabricar (y por tanto tienen un precio de venta más asequible), y tiene un tiempo de vida más largo (aunque con el tiempo pierde sus propiedades). Es, al igual que las LiPo, inflamable si se daña, se calienta en exceso o se sobrecarga. El empaquetado más común es el 18650, y en el mercado existen circuitos

de protección varios para evitar la sobrecarga y la sobredescarga para este encapsulado en concreto, por lo que es una opción barata y sencilla para conseguir un prototipado rápido y realizar pruebas del mismo, siempre y cuando el proyecto no exija mucha potencia; si el proyecto exigiera más tensión de la que proporciona la batería, se puede considerar obtener más unidades y realizar una configuración en serie para aumentar la tensión, o en paralelo si lo que se desea es aumentar la capacidad, o una combinación de ambas configuraciones (usando siempre un *Battery Management System* o BMS, para mantener las baterías balanceadas entre sí). Esta opción se puede considerar para baterías LiPo, pero para usuarios principiantes o poco experimentados resultará, probablemente, más complejo.

3. *Lithium Iron Phosphate* (LiFePO<sub>4</sub>). Son similares a las baterías LiPo, con la salvedad de que no son propensas a explosiones, como lo pueden ser las LiPo, si no se manipulan correctamente o las condiciones anteriormente mencionadas no se dan. También requieren una apropiada circuitería para cargarlas y descargarlas con cuidado, y si se comete un error se perderá permanentemente parte de la capacidad de la batería. Además, son un poco más pesadas que las baterías LiPo. Se usan normalmente en robots ligeros, herramientas eléctricas y juguetes por radio control, ya que son baterías bastante buenas entregando grandes cantidades de energía en un formato pequeño y liviano.
4. *Sealed lead-acid* (SLA). Son una opción si se quiere comprar una batería no muy costosa, a la vez de evitar enfrentarnos a circuitos de protección de la batería. Estas baterías son bastante más duraderas que una batería de litio, y también son mejores para hacer frente a sobrecargas accidentales o sobredescargas. Tienen una excelente relación calidad-precio y también son el mejor tipo de batería frente a temperaturas extremas; es por esta razón que las baterías de plomo (o ácido-plomo) son la opción estándar para coches, motos, almacenamiento de energía solar y fuentes de alimentación de emergencia. El inconveniente de usar baterías de plomo es que son grandes y pesadas. No es recomendable, por tanto, usar baterías de plomo en aplicaciones portátiles.
5. Níquel-cadmio (*Nickel-cadmium*, NiCd). Los dispositivos portátiles, como los primeros teléfonos móviles, usaban baterías de níquel-cadmio. Son baterías baratas, que en teoría duran más tiempo que las baterías SLA; sin embargo, sufren el llamado efecto memoria, por lo que hay que regularmente descargarlas por completo, luego recargarlas por completo para mantener una alta capacidad, ya que, de lo contrario, estaríamos perdiendo vida útil de la batería. Así pues, en la actualidad no se suele usar este tipo de baterías, aunque se siguen vendiendo.
6. *Nickel Metal Hydride* (NiMH). Las baterías de níquel-metalhidruro o de níquel hidruro metálico tienen una mayor densidad de energía que las baterías de níquel-cadmio, en tanto a precio no son mucho más costosas y no sufren el efecto memoria. Son más grandes y pesadas que las baterías de litio, pero es bastante seguro trabajar con ellas, por lo que son las baterías más usadas por el consumidor medio en términos de pilas recargables (las encontramos en cualquier supermercado, y cargarlas es barato y simple). Hay que tener en cuenta que las baterías de NiMH, por norma general, tienen un ratio de autodescarga elevado, lo que quiere decir que a pesar de que no se esté haciendo uso de la batería, se autodescargan igualmente en un par de meses, lo que hace que no sean una buena opción para objetos de uso cotidiano (como mandos a distancia). Se recomienda que, si se necesita hacer uso de este tipo de

baterías, se adquieran con bajo ratio de autodescarga (como Panasonic Eneloop), las cuales pueden resultar más baratas a largo plazo que comprar unas pilas alcalinas.

### *Comparativa entre tipos de baterías recargables*

#### **Densidad de energía**

CELL TYPE	NI-MH	NI-CD	LI-ION
<b>GRAVIMETRIC DENSITY (W-HR/KG)</b>	55	50	90
<b>VOLUMETRIC DENSITY (W-HR/L)</b>	180	140	210

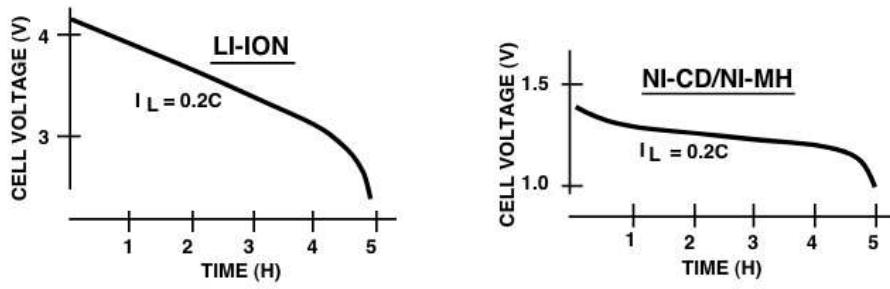
Figura 3: Comparación densidad de energía para diferentes baterías [9].

Al revisar los datos de la Figura 3, la ventaja de Li-Ion en densidad gravimétrica es claramente la más sorprendente, casi duplicando las cifras de Ni-Cd y Ni-MH. Esto significa que los productos que funcionan con celdas de iones de litio se pueden hacer mucho más livianos sin sacrificar el tiempo de ejecución. Alternativamente, si el peso de la batería se mantiene igual, el tiempo de funcionamiento se duplicará si se utilizan baterías de iones de litio. Este hecho explica la razón por la que Li-Ion está reemplazando rápidamente al Ni-MH en los teléfonos celulares y computadoras portátiles de primera línea.

#### **Estabilidad de tensión o tensión de celda**

La tensión proporcionada para alimentar la carga es la más importante: las baterías Ni-Cd y Ni-MH tienen una tensión por celda de 1.25V (sus tensiones de descarga se asumen por norma general idénticas).

La tensión por celda de las baterías Ni-Cd y Ni-MH son sólo una tercera parte de la tensión nominal de 3.6V proporcionada por una celda de Li-ion (Figura 4), lo que significa que se requerirían tres celdas de Ni-Cd/Ni-MH conectadas en serie para igualar la tensión de una única celda de Li-ion. Sin embargo, la Figura 4 muestra también la gran ventaja de las baterías Ni-Cd y Ni-MH: su curva de descarga es extremadamente plana, próxima a la de una batería ideal. Esta importante diferencia entre tipos de baterías significa que las celdas de Ni-Cd y Ni-MH son adecuadas para su uso con reguladores lineales, pero las baterías de Li-ion requieren convertidores reductores (*Buck converters*) para obtener una buena eficiencia de conversión de energía en la fuente de alimentación.



**FIGURE 3. CELL DISCHARGE CURVE**

Figura 4: Curva de descarga de una celda Li-ion y una celda Ni-Cd/Ni-MH (caída de tensión conforme aumentan las horas de uso de la batería) [9].

**Corriente pico**

**Autodescarga**

**Tiempo de carga**

**Coste**

**Fiabilidad**

**Modos de falla relacionados con la edad**

**Temperatura de funcionamiento**

**Panel solar**

Un panel solar, por si solo, no realiza la tarea de alimentación autónoma. Requiere, como mínimo, de una batería donde almacenar la energía que captan, por lo que, necesariamente, habría que escoger entre algunas de las opciones planteadas en el apartado anterior. Además, una batería no irá conectada directamente al panel solar, ya que se requiere un cargador solar que haga de intermediario, por una parte, entre la batería y el panel, y por otra parte, entre la batería y el dispositivo al que alimenta dicha batería.

Aunque esto será explicado con más detalle más adelante, se considera necesario establecer desde un principio los requerimientos mínimos para poder usar un panel solar de cara a la funcionalidad de alimentación autónoma, ya que dependen del apartado que acabamos de presentar sobre baterías recargables, y estos requerimientos mínimos establecerán una base de cara a la explicación

sobre sistemas fotovoltaicos más complejos.

*Funcionamiento y composición de un panel solar*

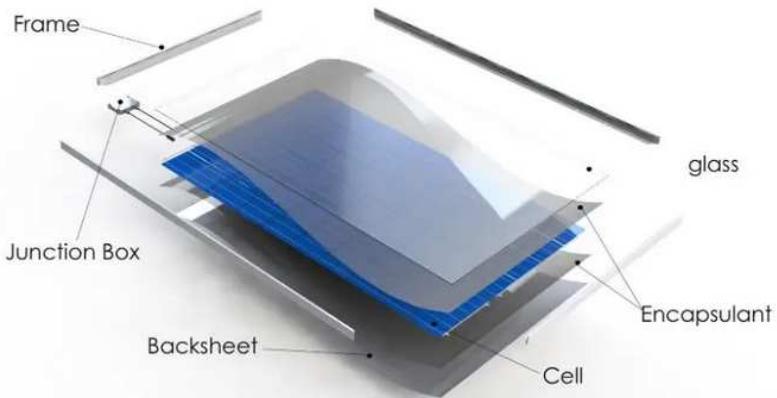


Figura 5: Capas y componentes que componen un panel solar [21].

Un panel solar es una agrupación de muchas células fotovoltaicas (PV) que están cubiertas con vidrio protector y unidas con un marco de metal. Es por eso que el nombre oficial de un panel solar es "módulo fotovoltaico" (*PV module*). Estas células solares fotovoltaicas están hechas de material semiconductor, generalmente silicio, que se corta en tiras muy finas.

Los paneles solares tienen muchas capas. La capa de células fotovoltaicas es donde se produce la electricidad. Otras capas, como las capas de vidrio y encapsulante, están ahí para proteger las células fotovoltaicas para que los módulos puedan producir electricidad de manera adecuada.

Cada célula fotovoltaica tiene una capa negativa y una capa positiva. Para que un panel solar genere electricidad solo necesitamos algo de energía para hacer que esos electrones fluyan de la capa negativa a la capa positiva.

Así pues, en resumen, la energía de los fotones del sol hace que los electrones en el lado negativo de la celda fotovoltaica se muevan (existe electricidad). Esto se llama efecto fotovoltaico.

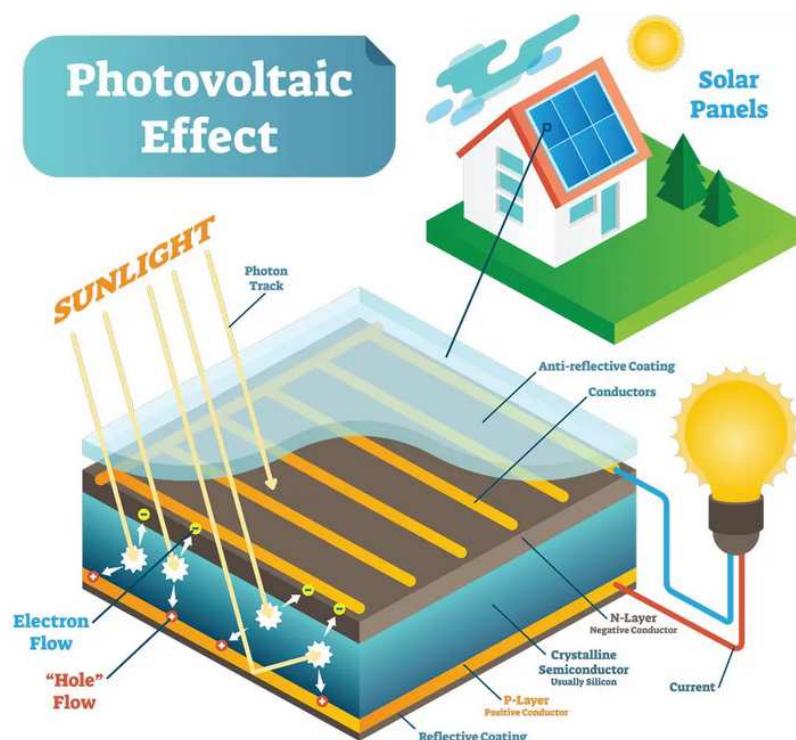


Figura 6: Dibujo representativo del efecto fotovoltaico [21].

Los electrones necesitan un camino a seguir y la electricidad debe estar en una forma útil. Es por ello que la ruta que creemos para los electrones (el circuito) es importante, ya que, cuando salen de la capa negativa de la celda fotovoltaica, queremos que fluyan a través de nuestras cargas (como nuestras luces y electrodomésticos, en el ejemplo del uso doméstico) para que los electrones puedan dar energía a esas cargas a medida que avanzan hacia la capa positiva de la celda fotovoltaica.

Un elemento muy importante en este sentido es el inversor, el cual convierte la energía DC producida por un panel solar, a energía AC, lista para ser usada en el hogar. Sin embargo, para nuestro proyecto no es necesario, ya que nos interesa crear un proyecto que se alimente de manera autónoma, sin uso de la electricidad disponible en el hogar; el panel solar, en este caso, recargaría una pila, y por ello necesitamos la energía DC, y no necesitamos usar un inversor a AC.

### Parámetros

Los principales parámetros que se usan para caracterizar el rendimiento de las células solares son la potencia máxima  $P_{max}$  (*peak power*), la densidad de corriente en cortocircuito  $J_{sc}$ , la tensión en circuito abierto  $V_{oc}$ , y el factor de forma  $FF$ . Estos parámetros se determinan a partir de la curva J-V característica cuando la célula está iluminada, tal como se muestra en la Figura 7a. La eficiencia de conversión  $\eta$  se puede determinar a partir de estos parámetros.

Nótese que se menciona específicamente *parámetros de una célula solar*; un panel solar, como se ha mencionado anteriormente, consta de varias células fotovoltaicas, por lo que los parámetros especificados, en términos globales y tal como se construyen los paneles solares, influyen en las especificaciones finales del panel según el número de células del que conste (ver Figura 8).

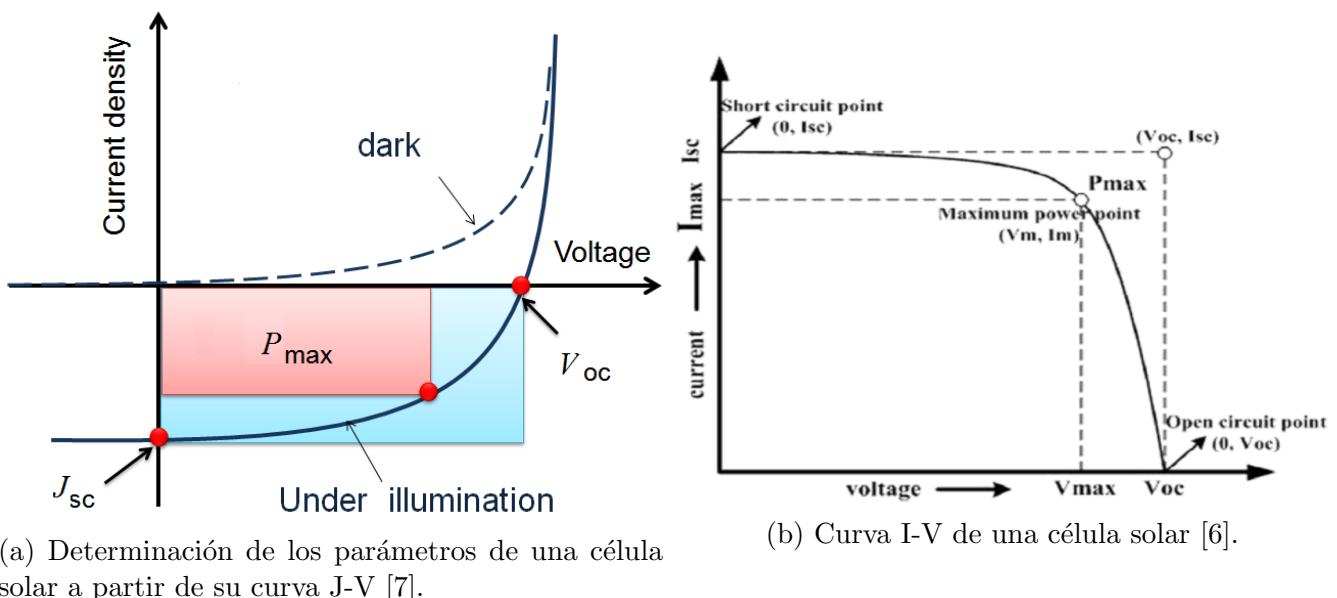


Figura 7: Curvas J-V e I-V de una célula fotovoltaica

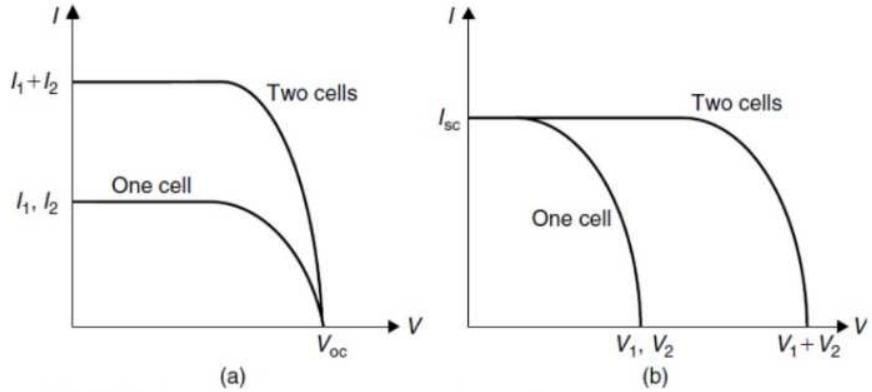


Figura 8: Comportamiento de un panel solar formado por dos células solares idénticas [27].

Para realizar medidas fiables y definir la curva característica J-V, se deben realizar las medidas bajo condiciones estándar de pruebas (*standard test conditions, STC*); estas son, una temperatura constante a  $25^{\circ}C$ , una irradiancia total de  $1000\text{ W/m}^2$  y un espectro solar de AM1.5 (sol directo y con ángulo cenital de  $48,2^{\circ}$ ). Esto lo tendremos en cuenta si queremos comprobar que las especificaciones del producto son correctas, o por el contrario el producto viene tiene defectos. En este proyecto no es prioritario el estudio profundo y comprobación exhaustiva del panel solar, por lo que no se realizarán dichas pruebas; sin embargo, sí realizaremos un estudio teórico de los parámetros que definen a una célula solar, como también lo hemos hecho de los materiales que lo componen y su funcionamiento general.

### **Densidad de corriente en cortocircuito**

La corriente en cortocircuito  $I_{sc}$  es la corriente que fluye a través de un circuito externo cuando los electrodos de una célula solar se cortocircuitan. La corriente en cortocircuito de una célula solar depende de la incidencia del flujo de fotones sobre ella, la cual está determinada por el espectro de la luz incidente. También depende del área de la célula solar. Para eliminar la dependencia del área de la célula solar sobre  $I_{sc}$ , normalmente se usa la densidad de corriente en cortocircuito para describir la máxima corriente entregada por una célula solar. La máxima corriente que una célula solar puede entregar depende mucho de las propiedades ópticas de la célula solar, tales como la absorción en la capa absorbente y la reflexión.

Las células solares de silicio cristalino pueden entregar, bajo un espectro AM1.5, una posible densidad de corriente máxima de  $46\text{ mA/cm}^2$ . En las células solares de c-Si de laboratorio, las medidas de  $J_{sc}$  están por encima de  $42\text{ mA/cm}^2$ , mientras que las células solares comerciales tienen un  $J_{sc}$  que excede los  $35\text{ mA/cm}^2$ .

### **Tensión en circuito abierto, $V_{oc}$**

Es la tensión máxima que una célula solar puede entregar.  $V_{oc}$  corresponde a la tensión de polariza-

zación directa, a la cual la densidad de corriente de oscuridad compensa la densidad de corriente fotovoltaica (*fotocorriente*). Depende de la densidad de corriente de saturación de la célula solar y de la corriente fotogenerada. Mientras ésta última varía poco, la corriente de saturación puede variar en órdenes de magnitud. La corriente de saturación depende de la recombinación en la célula solar, por lo que  $V_{oc}$  es una medida de la cantidad de recombinación de la célula solar.

Las células solares de laboratorio hechas de silicio cristalino dan valores de  $V_{oc}$  de hasta 720 mV bajo condiciones AM1.5 estándar, mientras que las células solares comerciales normalmente dan valores de  $V_{oc}$  por encima de los 600 mV.

### **Factor de forma**

El factor de forma es la relación entre la potencia máxima ( $P_{max} = J_{mpp}V_{mpp}$ ) generada por una célula solar y el producto de  $V_{oc}$  con  $J_{sc}$ .

El subíndice *mpp* hace referencia al *punto de potencia máxima* (*maximum power point, MPP*) de la célula solar, es decir, el punto de la curva característica J-V de la célula solar al cual la célula solar ofrece su máxima potencia de salida. Para optimizar la operación de los sistemas fotovoltaicos, es muy importante que las células solares (o módulos fotovoltaicos) operen sobre el MPP. Esto se puede asegurar con un seguidor de punto de máxima potencia (*maximum power point tracking, MPPT*). Se trata de un algoritmo, el cual se usa en controladores de carga solar para extraer la máxima potencia disponible del módulo fotovoltaico bajo ciertas condiciones; dichas condiciones (se entrará más en detalle más adelante), como son la radiación solar a lo largo del día y año, la temperatura ambiente y la temperatura de la célula, afectan a la potencia máxima de salida de la misma, y es entonces cuando un MPPT actúa, haciendo operar a la célula solar al valor de tensión más eficiente, el MPP. El controlador de carga mantiene la tensión y la corriente a un nivel optimizado, donde el módulo fotovoltaico brinda la mayor cantidad de energía a la carga; es “consciente” del estado de carga de la batería y actúa en consecuencia.

El uso de un seguidor de punto de máxima potencia, sin embargo, encarece el costo total de un proyecto; si bien para aplicaciones que requieran la extracción de la máxima potencia de un panel solar (por limitaciones de espacio, por ejemplo, donde deberíamos explotar las características de los paneles que pudiéramos colocar), en el proyecto que estamos diseñando no es el caso, por lo que no usaremos herramientas de este estilo. Además, aunque pueden encontrarse o construirse de manera aislada (se trata de un conversor DC a DC), también suelen encontrarse incorporados a inversores, los cuales tampoco son necesarios, ya que no es energía solar para uso en hogar. Como anotación, existen otros controladores de carga que no son MPPT, como el PWM o *shunt controller* (regulador de carga tipo paralelo o de desviación).

Por último, anotar que el factor de forma no varía tanto con  $V_{oc}$ , sino más con el material del que están hechas las células solares.

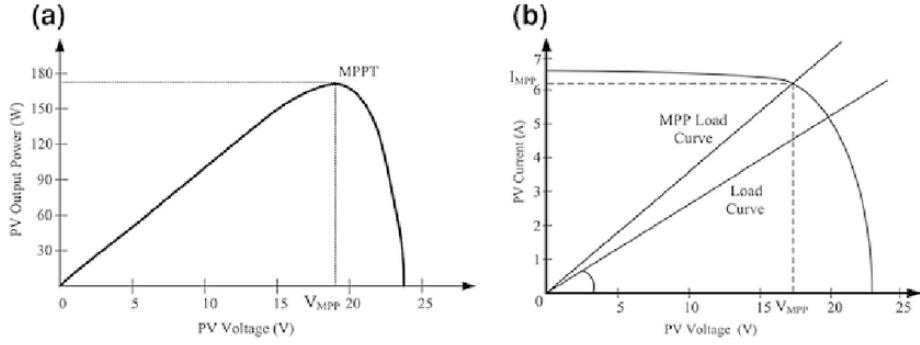


Figura 9: Curva P-V de una célula solar, indicando el punto MPP [3].

### ***Eficiencia de conversión***

La eficiencia de conversión se calcula como la relación entre la potencia generada y la potencia incidente. Como vemos en la siguiente fórmula, y tal como establece su propia definición, depende de todos los parámetros anteriormente descritos:

$$\eta = \frac{P_{max}}{I_{in}} = \frac{J_{mpp}V_{mpp}}{I_{in}} = \frac{J_{sc}V_{oc}FF}{I_{in}} \quad (1)$$

Los parámetros externos típicos de una célula fotovoltaica de silicio cristalino son  $J_{sc} \approx 35mA/cm^2$ ,  $V_{oc}$  hasta 0.65V y FF en el rango entre 0.75 y 0.80; esto implica que la eficiencia de conversión caiga en el rango entre 17 y 18 %.

### ***Circuito equivalente***

Una célula fotovoltaica iluminada puede comportarse como un diodo ideal, y este comportamiento puede describirse a través de un circuito equivalente simple, como el que se muestra en la Figura 10 (a). donde un diodo y una fuente de corriente están conectados en paralelo. El diodo está formado por una unión p-n.

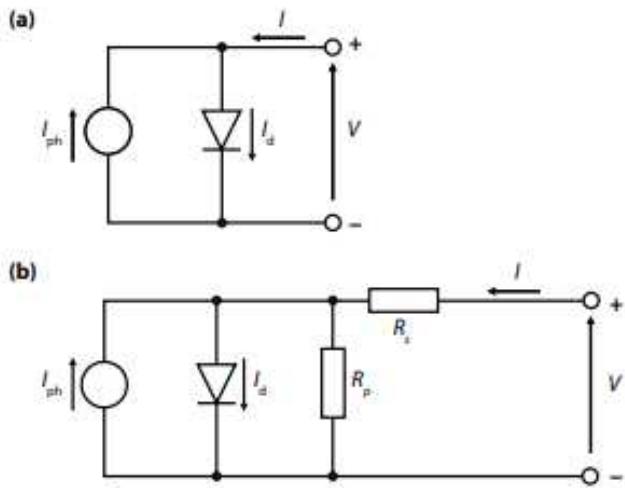


Figura 10: Circuito equivalente de una célula fotovoltaica, en el caso ideal (a) y considerando pérdidas (b) [24].

Sin embargo, en la práctica, el factor de forma está influenciado por una resistencia en serie  $R_s$ , y una resistencia de derivación  $R_p$ . La influencia de estos parámetros en la curva característica J-V de una célula solar puede estudiarse usando el circuito equivalente mostrado en la Figura 10 (b). Además, en células solares reales, el factor de forma también está influenciado por una recombinación adicional que ocurre en la unión p-n. Este diodo no ideal se representa normalmente con un circuito equivalente formado por dos diodos, uno ideal con un factor de idealidad<sup>1</sup> igual a la unidad, y otro diodo no ideal con un factor de idealidad mayor que uno.

*Factores principales que influyen en la potencia de salida de una célula PV*

### **Radiación solar**

---

<sup>1</sup>El factor de idealidad del diodo es una medida de cómo de cerca el diodo sigue la ecuación del diodo ideal.

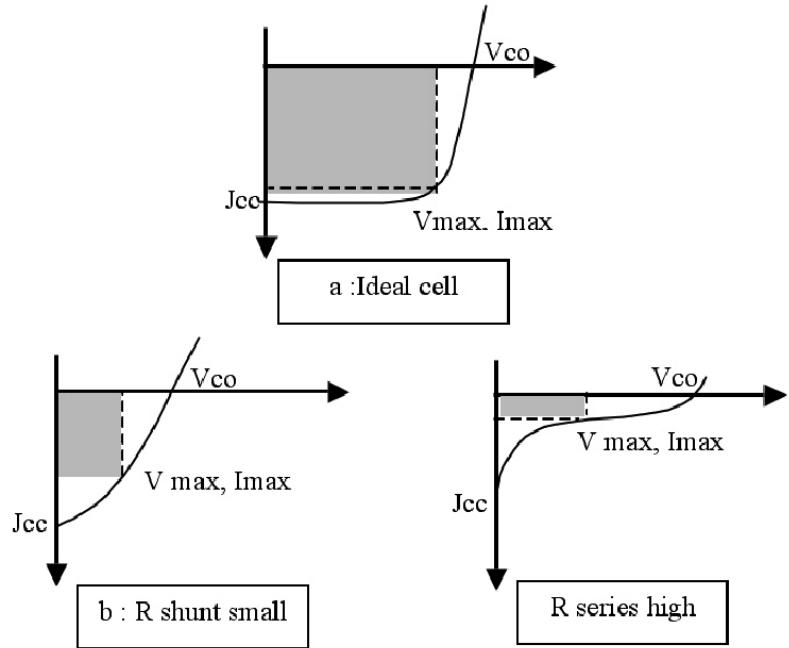


Figura 11: Curva J-V de una célula fotovoltaica, mostrando el efecto de aumentar o disminuir el valor de las resistencias  $R_s$  y  $R_p$  o  $R_{sh}$  (shunt) [8].

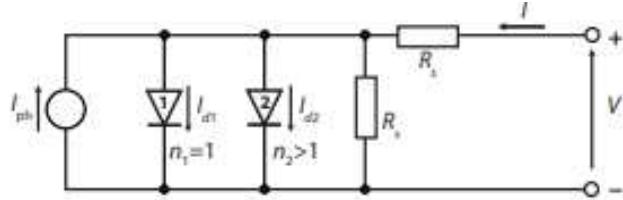
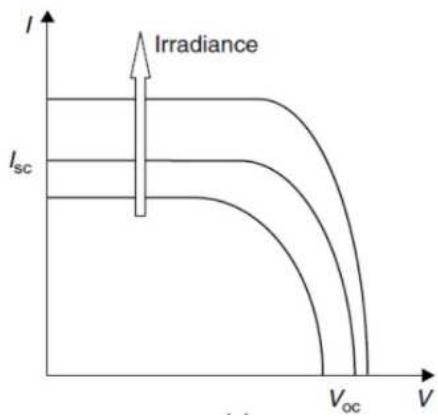


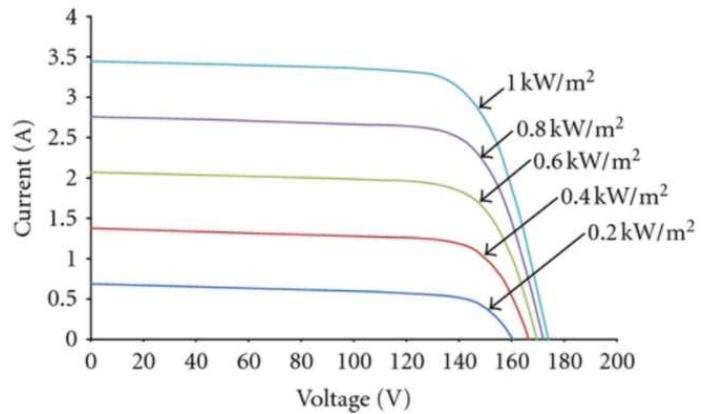
Figura 12: Circuito equivalente de una célula fotovoltaica con un modelo de dos diodos, donde  $n_1$  y  $n_2$  son los factores de idealidad e  $I_{d1}$  y  $I_{d2}$  las corrientes que atraviesan cada diodo [24].

### **Temperatura ambiente y temperatura de la célula**

Nótese que en la Figura 14a se hace referencia a *temperatura de la célula* y en la Figura 14b a *temperatura ambiente*. Si bien son dos magnitudes diferentes, la temperatura ambiente tiene un efecto sobre la temperatura de la célula que puede considerarse lineal. A una temperatura ambiente de  $25^\circ C$ , la temperatura de una célula de silicio cristalino suele situarse en el valor de  $48^\circ C$ .



(a) Efecto general [27].



(b) Efecto según valores concretos de radiación [4].

Figura 13: Variación en la curva I-V ante la radiación solar.

### Opciones: tipos de paneles solares

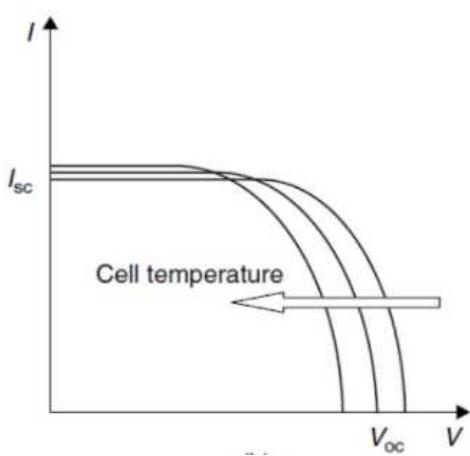
Existen diferentes tipos de células fotovoltaicas disponibles en el mercado, aunque en su mayoría están fabricadas de silicio (Si), el segundo elemento más abundante en la corteza terrestre (después del oxígeno), el cual es el mismo material del cual está hecha la arena. El silicio por sí mismo no es muy puro y debe ser refinado para conseguir una pureza de hasta el 99.99 % antes de que pueda usarse como material semiconductor en muchos tipos distintos de células fotovoltaicas, transistores o circuitos integrados digitales. El uso de silicio en la fabricación de células fotovoltaicas produce la típica célula fotovoltaica de color azul uniforme.

Los dos tipos principales de materiales usados para células fotovoltaicas son el silicio cristalino y los depósitos de película fina, los cuales varían entre unos y otros en términos de eficiencia en absorción de luz, eficiencia en la conversión de energía, tecnología de fabricación y coste de producción. Las células fotovoltaicas (a partir de ahora, células PV, *PhotoVoltaic*) hechas de silicio cristalino son el tipo más común de células PV que se utilizan en la actualidad y también son uno de los primeros dispositivos PV que tuvieron éxito.

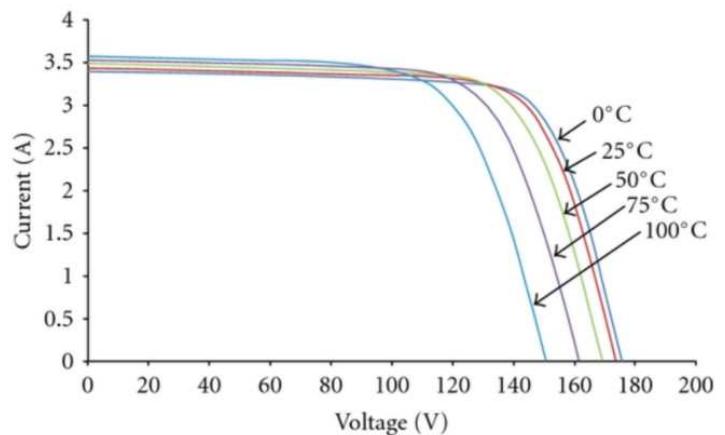
Los tres tipos generales de células PV hechas de silicio son:

- Silicio monocristalino, conocido también como silicio de un único cristal, cristal de silicio único o silicio monocristal (*Mono-crystalline silicon, single-crystal silicon*).
- Silicio policristalino, conocido también como silicio de varios cristales, silicio multicristalino o silicio multicristal (*Poly-crystalline silicon, multi-crystal silicon*).
- Silicio de película fina.

### **Silicio cristalino (c-Si)**



(a) Efecto general [27].



(b) Efecto según valores concretos de temperatura ambiente [4].

Figura 14: Variación en la curva I-V ante la temperatura ambiente

### Célula PV de película fina

#### Otros tipos de células PV

##### 3.2.2 Automatización y monitorización

En este apartado, se mostrará una visión de conjunto acerca de aquellos elementos mediante los cuales consigamos el objetivo de automatizar y monitorizar el sistema de alimentación. Se presentará el resultado de investigar sobre opciones que puedan ayudar a la consecución de los objetivos, tanto en la parte de microcontrolador como en la parte de comunicación radio, y demás actuadores y sensores en las partes de bebedero y comedero. Hay que tener presente que en todo momento se estarán buscando soluciones *Open Source*, ya que se planteó como objetivo y filosofía a seguir en este proyecto.

###### 3.2.2.1 Microcontrolador

*Aclaración:* a lo largo de este apartado y consecutivos se hablará de la palabra microprocesador y microcontrolador. Se usará la palabra microcontrolador para determinar el conjunto global de una placa cuyo corazón sea un microcontrolador, de manera que podamos aligerar un poco la literatura y lectura de este documento. Un microprocesador es la CPU, mientras que un microcontrolador conforma un sistema embebido en un único *chip* formado por memoria, procesador y entradas/-salidas programables. Por ejemplo, llamaremos microcontrolador al arduino nano o uno porque se basa en el microcontrolador ATmega328.

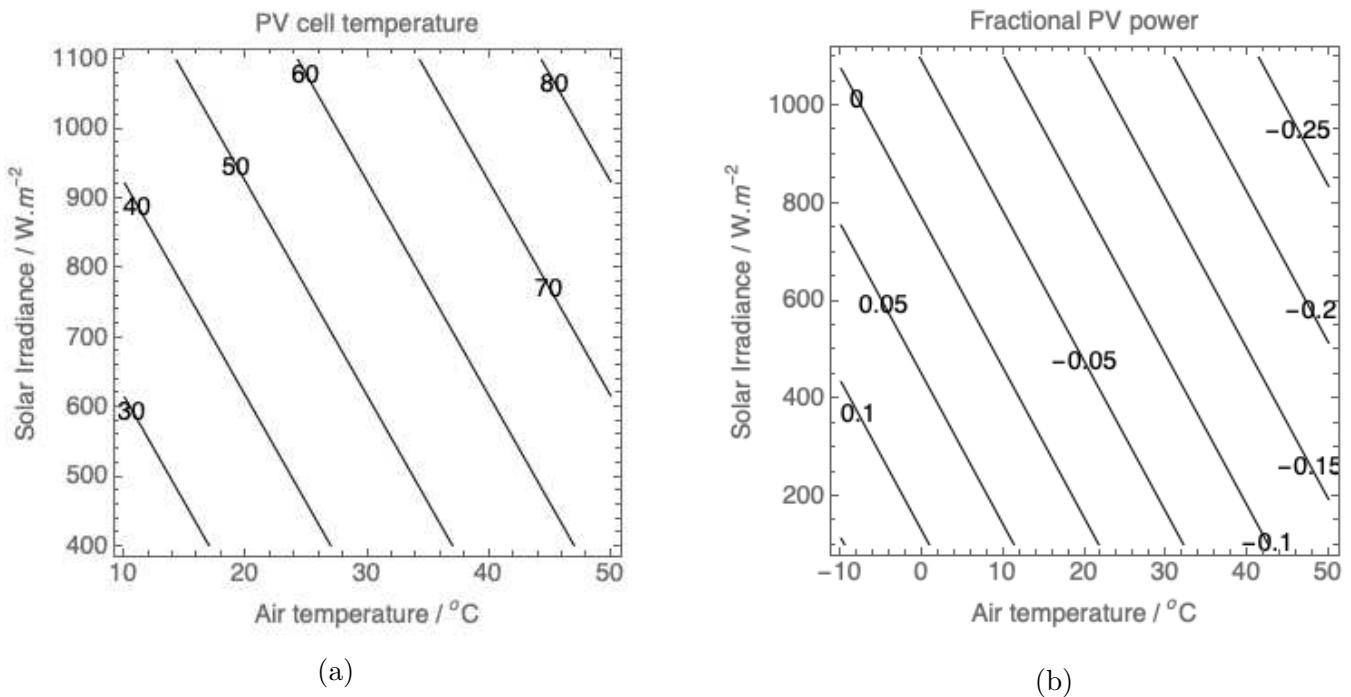


Figura 15: Relación entre temperatura ambiente, radiación solar, con la potencia de salida de una célula PV (15a) y temperatura de ésta (15b) [28].

En tanto a microcontroladores, la opción *Open Source* más conocida seguramente sea Arduino. Sin embargo, no significa que no existan más opciones en tanto a microcontroladores. Es importante recordar que ni Arduino ni las alternativas que se van a presentar son ordenadores, son microcontroladores, esto es, circuitos integrados diseñados para llevar a cabo una tarea única de manera cíclica (una vez que programado para una tarea, ejecutará esa tarea y sólo esa tarea una y otra vez, dependiendo de cómo haya sido programado).

A continuación, se va a presentar una lista con algunas opciones de microcontroladores [29][30][31]. Vale la pena señalar que la mayoría de estas placas tienen variaciones que también vale la pena revisar; la elección de un microcontrolador u otro dependerá de lo que necesite nuestro proyecto (si necesita muchas entradas porque tendremos muchos periféricos, si necesitamos ahorro de energía al máximo, o una buena potencia de procesamiento, etc).

- **Arduino.** Se trata de una plataforma de desarrollo Open Source hardware y software, característica por su facilidad de uso e integración entre la solución hardware (placas Arduino) y la solución software (Arduino IDE). Existe una gama variada de placas Arduino, entre las que podemos destacar la nano o la uno. A modo de ejemplo, se mostrará la información de la uno, placa recomendada por la propia organización Arduino para principiantes.

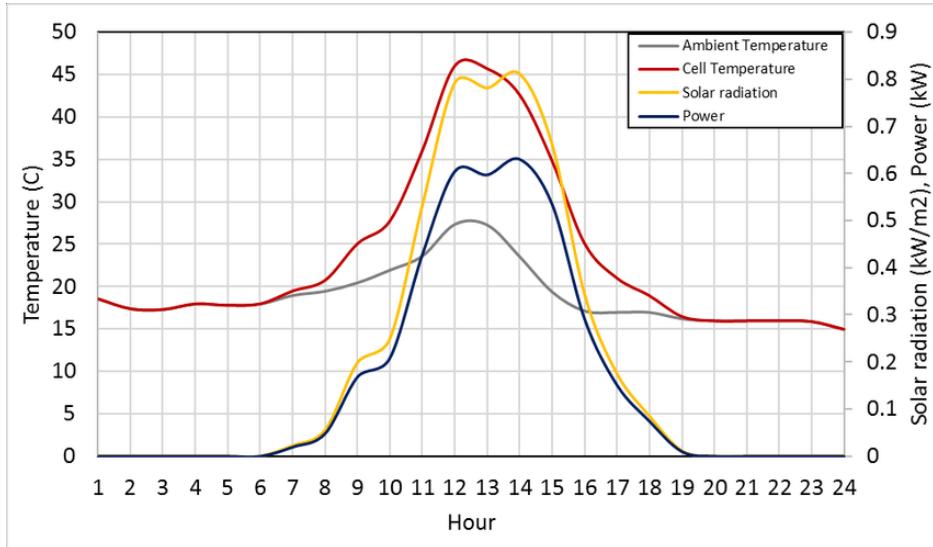


Figura 16: Variación de la potencia de salida de una célula PV dependiendo de la hora del día, la temperatura y la radiación solar [5].

*Especificaciones:* ATMega328P a 16 MHz, 32 kB de memoria flash, 2kB de SRAM y 1kB de EEPROM.

*Características:* 14 pines I/O digitales, 6 entradas analógicas, conexión USB, conector jack.

*Precio:* 20 € en Arduino.

- **NodeMCU.** NodeMCU (*Node MicroController Unit*) es un entorno de desarrollo de software y hardware de código abierto que se basa en un sistema en chip (SoC) muy económico llamado ESP8266. El ESP8266, diseñado y fabricado por Espressif Systems, contiene todos los elementos cruciales de un ordenador: CPU, RAM, redes (wifi) e incluso un sistema operativo y SDK modernos. Cuando se compra por separado, el chip ESP8266 cuesta solo 2\$ por unidad. Esto lo convierte en una buena opción para proyectos de IoT de todo tipo.

*Especificaciones:* SoC basado en ESP8266 a 80 MHz, 64 kB de SRAM, 4 MB de memoria flash.

*Características:* compatibilidad con Arduino IDE, 16 GPIO, USB-TTL en CP2102 incluido onboard, tamaño reducido, puerto micro-USB, WiFi.

*Precio:* aprox. 8 € en Amazon.

- **STM32.** La familia STM32 de microcontroladores de 32 bits basada en el procesador Arm

Cortex-M ofrece productos que combinan un rendimiento muy alto, capacidades en tiempo real, procesamiento de señales digitales, operación de bajo consumo / bajo voltaje y conectividad, mientras se mantiene una integración completa y facilidad de desarrollo.

La gama de microcontroladores STM32, basada en un núcleo estándar de la industria, viene con una amplia variedad de herramientas y software para respaldar el desarrollo de proyectos, lo que hace que esta familia de productos sea ideal tanto para proyectos pequeños como para plataformas de extremo a extremo.

*Especificaciones:* depende del microcontrolador. Ejemplo de STM32F103C876, 72MHz Cortex-M3, SRAM de 20kB, 64/128 kB de memoria flash.

*Características:* software propio pero compatible con Arduino IDE, tamaño similar al arduino nano, dependiendo del microcontrolador tendremos diferentes tipos dentro de la gama STM32 (bajo consumo, altas prestaciones, conectividad, mainstream).

*Precio:* depende del microcontrolador. En este ejemplo, aprox. 10\$ en eBay.

- **The Pinguino 45K50.** Pinguino Project existe para acercar la simplicidad del lenguaje Arduino a los microcontroladores PIC de Microchip pero con un hardware USB incorporado y una placa que se puede construir en casa. El 45K50 es una de las opciones que nace de este proyecto Open Source. Viene como un kit que se debe ensamblar antes de usarlo.

*Especificaciones:* procesador de 12 MIPS de 8 bits que se ejecuta a 48 MHz.

*Características:* 17 entradas / salidas digitales con 5 entradas analógicas compartidas, 2 salidas PWM, UART para comunicación en serie, etc. IDE propio, también Open Source, compatible con el lenguaje Arduino y sus librerías.

*Precio:* aprox. 25\$.

- **Nanode.** Nanode ofrece una amplia gama de microcontroladores. Viene como un kit de desarrollo, lo que significa se debe ensamblar y soldar previamente antes de comenzar con la placa. El microcontrolador Nanode puede servir como alternativa a Arduino Uno, Mega y Yun, y también es Open Source.

*Especificaciones:* ATMega328P a 16MHz, 2kB de SRAM, 1kB de EEPROM, 32kB de memoria flash ISP.

*Características:* mini USB, controlador ethernet ENC8J60, conector ethernet, LEDs de diagnóstico.

*Precio:* aprox. 50\$.

- **Teensy 3.6. o Teensy 2.0.** La familia de placas Teensy fue desarrollada por PJRC durante los últimos años. Inicialmente, Teensy se creó prometiendo un poco más de factor de forma más pequeño. Uno de los Teensy más apreciados es el 3.6.; tiene un procesador ARM de 32 bits, 52 entradas/salidas en total, ADC, lector de tarjetas SD y muchas más funciones. Además, PJRC proporciona un complemento software *Teensyduino* para una compatibilidad casi completa con los programas Arduino.

*Especificaciones:* 180-MHz Cortex M4F, 256 kB RAM, almacenamiento flash 1MB, 4K EEPROM.

*Características:* Teensyduino para Arduino IDE, lector SD.

*Precio:* aprox. 30 \$ en PJRC.

- **Launchpad MSP430.** Launchpad es una placa de desarrollo fabricada por Texas Instruments (TI). Junto con el hecho de que viene con un MSP430 de TI, también brinda compatibilidad con el software y hardware de TI. Esto viene en forma de paquetes de refuerzo y herramientas de desarrollo en línea. Además, permite la depuración onboard. Existe un IDE similar a Arduino llamado Energia para un enfoque más sencillo.

*Especificaciones:* 24-MHz MSP430, 32 kB Program FRAM, 4 kB RAM.

*Características:* tecnología *EnergyTrace* para bajo consumo, Energia (similar a Arduino IDE), dos botones onboard y LEDs, compatible con el *Booster Pack*.

*Precio:* aprox. 13 \$ en Texas Instruments.

- **Netduino N3 Wi-Fi.** Una alternativa diferente a Arduino de Wilderness Labs, el Netduino N3 utiliza .NET MicroFramework y Netduino.Foundation Framework, lo que significa que está programado en un lenguaje de programación de nivel superior (C#). Esto puede resultar atractivo para aquellos que no quieren lidiar con los niveles más bajos de programación de MCU.

Esta placa, con Wi-Fi incorporado, está diseñada para conexión inalámbrica entre dispositivos. Otra característica notable de Netduino es la disposición de los pines hecha para la compatibilidad del shield Arduino. Dicho todo esto, Netduino puede atraer a aquellos que ya están familiarizados con el marco .NET y les gustaría programar su placa MCU a través de Microsoft Visual Studio y similares.

*Especificaciones:* 168 MHz, 164+ kB RAM, 1408kB de almacenamiento flash.

*Características:* .NET MicroFramework, librerías Netduino.Foundation, Wi-Fi, lector SD, 22 pines de entrada-salida de propósito general (GPIO).

*Precio:* aprox. 50\$.

- **Particle Photon.** Particle Photon es la placa ideal para el proyecto de IoT conectado. El Photon tiene un chip Wi-Fi Cypress integrado y viene con conectividad gratuita con Device Cloud y todas sus funciones. Particle incluso proporciona JavaScript y SDK móviles junto con su IDE web y local.

Gracias a todo el soporte, guías y herramientas proporcionadas por Particle, Photon es una buena alternativa a Arduino. Es ideal para aficionados principiantes o intermedios que buscan la plataforma adecuada para un proyecto de IoT simple.

*Especificaciones:* ARM Cortex M3 de 120 MHz, RAM de 128 kB, almacenamiento flash de 1 MB.

*Características:* Device Cloud, SDK para dispositivos móviles y ParticleJS, 18 GPIO, muchas referencias y guías.

*Precio:* aprox. 19\$.

- **SparkFun Thing Plus.** SparkFun tomó el ESP32 e hizo esta placa. El ESP32 es sin duda una gran opción para una placa IoT. Viene con Wi-Fi, Bluetooth y Bluetooth Low Energy integrados. Incluso agregaron una conexión Li-Po para complementar aún más su naturaleza inalámbrica.

Thing Plus también muestra algunas especificaciones impresionantes y un buen conjunto de sensores. Además, es compatible con Arduino IDE.

*Especificaciones:* 240 MHz, 520 kB SRAM, 16MB de almacenamiento flash.

*Características:* Wi-Fi, Bluetooth, BLE, cargador de batería Li-Po, sensor de efecto Hall, sensor táctil capacitivo, sensor de temperatura.

*Precio:* aprox. 21\$ en SparkFun.

- **Adafruit Feather Huzzah.** El Huzzah proviene de la línea Feather de Adafruit. Está destinado a ser una placa pequeña. Tiene Wi-Fi incorporado, un cargador Li-Po y compatibilidad con Arduino. Además, como ESP8266, se puede utilizar con NodeMCU Lua.

Puede parecer que esta placa parece una versión inferior de Thing Plus, ya que el ESP32 es esencialmente el sucesor del ESP8266. Esta placa, sin embargo, ha existido lo suficiente como para que haya ganado una gran comunidad de usuarios. Esto hace que esta placa sea accesible para los aficionados que deseen la experiencia colectiva de dicha comunidad.

*Especificaciones:* 80 MHz, 50 kB RAM, 4MB de almacenamiento flash.

*Características:* Wi-Fi, cargador Li-Po, compatible con Arduino, NodeMCU Lua, 9 GPIO.

*Precio:* aprox. 17\$ en Adafruit.

- **BeagleBoard PocketBeagle.** El PocketBeagle es el más pequeño de los BeagleBoards. Ejecuta Linux desde el primer momento y se puede programar a través de su navegador web. Esta opción, por tanto, se trataría de un ordenador, como lo es una Raspberry Pi, no un microcontrolador. Ideal para proyectos pequeños que necesitan los recursos de una computadora.

El Sitara AM3358 ARM Cortex-A8 en el PocketBeagle tiene dos PRU (unidades programables en tiempo real) que son esencialmente microcontroladores conectados al procesador principal. Estos son, en palabras de sus preguntas frecuentes, “ideales para una latencia baja predecible, mientras que el procesador ARM es bueno para el rendimiento”.

*Especificaciones:* 1GHz Cortex-A8, 512 MB RAM

*Características:* sistema operativo Linux, conector SD, 2 PRUs, 72 pines de expansión, fácilmente programable a través de un navegador web.

*Precio:* aprox. 35\$.

- **SparkFun RedBoard Artemis.** El RedBoard Artemis es la versión mejorada de SparkFun del Arduino Uno. Viene en el mismo factor de forma y agrega Bluetooth, 24 GPIO, 21 PWM y aumenta significativamente las especificaciones.

SparkFun continúa reconociendo el valor de la plataforma Arduino y, por lo tanto, ha hecho que esta placa sea compatible con el IDE de Arduino. Además, han expuesto el conector JTAG para aquellos usuarios más avanzados que desean depurar correctamente su MCU.

Esta placa relativamente nueva también es capaz de ejecutar modelos TensorFlow, llevándola al IDE de Arduino.

*Especificaciones:* 48 MHz (96 MHz turbo), 384 kB RAM, 1MB de almacenamiento flash.

*Características:* compatibilidad con Arduino IDE, Bluetooth, 24 GPIO.

*Precio:* aprox. 20\$.

- **STM32F3 Discovery.** El STM32F3 es solo una de las diversas placas Discovery fabricadas por ST que vale la pena revisar. Este, sin embargo, viene con un giroscopio y acelerómetro de 3 ejes, un sensor magnético 3D, 10 LED y algunas características más.

Si bien esta alternativa no tiene compatibilidad con Arduino IDE, ST proporciona herramientas de software. Actualmente, su ecosistema STM32Cube incluye un generador de código de inicialización e IDE.

*Especificaciones:* 48 kB RAM, 256kB de almacenamiento flash.

*Características:* Giroscopio/acelerómetro de 3 ejes, sensor magnético 3D, 10 LED, 2 botones, sensor de movimiento.

*Precio:* aprox. 16\$ en ST.

- **Silicon Labs Wonder Gecko.** Esta placa de Silicon Labs se basa en el EFM32. Si bien es una placa más cara, viene con una serie de características muy destacables. Incluyen un micro LCD, un control deslizante táctil, un sensor de luz ambiental y un condensador de 0.03F para proporcionar energía de respaldo, por nombrar algunos.

El Wonder Gecko está más orientado al desarrollo, pero aún tiene una cantidad decente de referencias y guías para ser accesible. Silicon Labs también ofrece su Simplicity Studio. Si el precio no es un problema, esta placa y sus variaciones son buenas alternativas.

*Especificaciones:* 48MHz Cortex M4, 32 kB RAM, 256kB almacenamiento flash.

*Características:* debugger integrado, sistema de monitoreo de energía, varios sensores, LCD, Simplicity Studio.

*Precio:* aprox. 100\$ en Silicon Labs.

- **Raspberry Pi Pico.** Este microcontrolador lleva en el mercado desde el 25 de enero de 2021, apenas unos meses; este hecho, sin embargo, no impide que miles de personas ya lo hayan probado, y es que actualmente existe en internet muchos proyectos y documentación acerca de esta nueva Raspberry Pi, la cual se aleja de su antecesora mini ordenador para acercarse más al mundo de los microcontroladores similares a Arduino. Está construido sobre el propio silicio interno de Raspberry Pi, el RP2040. El RP2040 también forma parte de un ecosistema más amplio. Adafruit, Pimoroni, SparkFun e incluso el equipo de Arduino han

desarrollado placas adaptadas a necesidades más específicas. Uno de los dispositivos de Pimoroni, el PicoSystem, es una “experiencia de creación de juegos portátil” de 58.50 £ (alrededor de 80 \$) que te permite programar una consola incluso más pequeña que la Game Boy Micro.

Para los usuarios avanzados, la organización Raspberry Pi proporciona un C SDK completo, una cadena de herramientas basada en GCC e integración con Visual Studio Code.

*Especificaciones:* ARM Cortex-M0+ de doble núcleo, 264 KB de RAM y 16 MB de memoria flash (2 MB integrados).

*Características:* 30 pines GPIO, controlador USB 1.1 (más el modo de almacenamiento USB), compatibilidad con Arduino IDE.

*Precio:* 4 \$.

- **Opción DIY** (*Do It Yourself*) hacer tú mismo tu propio arduino o microcontrolador). Existen tutoriales varios en internet que nos enseñan a crear nuestro propio microcontrolador haciendo uso de componentes como ICs (*Integrated Circuits*), diodos, resistencias, condensadores, etc. Es una buena opción para aprender cómo funciona un microcontrolador a nivel electrónico, o si queremos personalizar a nuestro gusto para reducir el consumo del sistema global que constituye el microcontrolador, por ejemplo.

### 3.2.2.2 Comunicación radio

En tanto a posibles opciones para llevar a cabo una comunicación radio efectiva, existen muchas, pero la investigación se centrará en aquellas que nos puedan servir para aplicaciones IoT, las cuales se caracterizan por buscar ser autónomas (es decir, priorizar bajo consumo y ahorro de energía), de largo alcance, con transferencia de datos baja/media, seguridad y chequeo de errores, y conectividad entre redes y la nube, entre otros. Se suele hablar de redes LPWAN, para conseguir conectividad IoT, y dentro de esta categoría tenemos varias opciones, entre las que se encuentra LoRa, la cual ya fue preseleccionada para este proyecto por su soporte *Open Source* y equipos asequibles para la creación de redes sencillas, aunque también complejas (Ademas de que conociamos las características del terreno, y demás), y se deseaba hacer un estudio sobre prestaciones de equipos que usarán esta tecnología.

#### Qué es una red LPWAN

Si uno investiga acerca de *Internet of Things*, llega a la conclusión de que existe una cantidad abrumadora de opciones para conectividad IoT: desde Wi-Fi hasta Bluetooth, NB-IoT hasta CAT-M1, y LoRa hasta RPMA. Cada aplicación IoT tiene sus propios requerimientos, lo que quiere decir que

una opción de conectividad IoT que es perfecta para una aplicación puede ser una opción menos adecuada para otra. Por ejemplo, si la aplicación tiene muchos sensores remotos, la vida útil de la batería es una consideración importante, pero si la aplicación necesita enviar muchos datos lo es el ancho de banda, o si involucra datos que son vitales, lo es la latencia. En resumen, es crucial elegir la mejor opción de conectividad IoT que mejor se adapte a las necesidades.

Lo primero que debemos tener claro es que LPWAN (*Low Powered Wide Area Network*) no es un estándar. Es un término amplio que abarca varias implementaciones y protocolos, tanto propietarios como de código abierto, que comparten características comunes, como sugiere el nombre:

*Low power*: operan bajo baterías pequeñas y económicas durante años.

*Wide area*: tienen un alcance de operación típico de más de 2 km en configuraciones urbanas.

Una limitación física (es decir, el precio que hay que pagar) para alcanzar bajo consumo y largo alcance es la cantidad total de datos (pocos datos y mensajes cortos) y las reducidas tasas binarias. La mayoría de tecnologías LPWAN pueden enviar solo menos de 1000 bytes de datos por día o menos de 5000 bits por segundo.

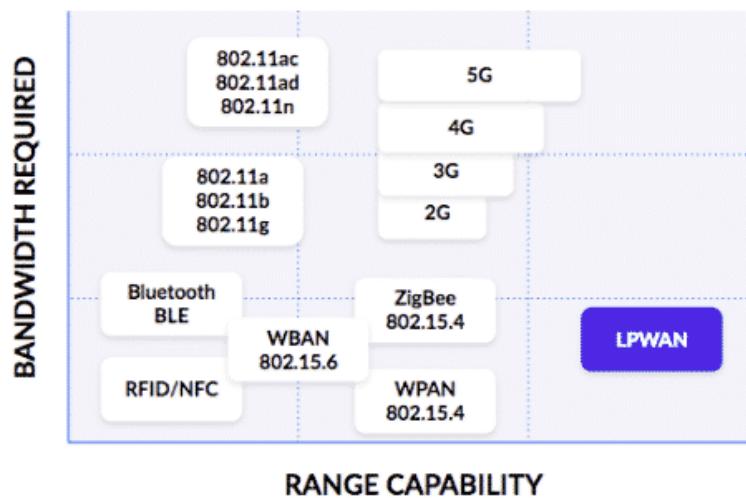


Figura 17: Comparativa entre LPWAN, en términos de alcance y ancho de banda, con respecto a otras comunicaciones inalámbricas [48].

### 3.2.2.3 Comedero

Tanto en este apartado, como en el siguiente sobre búsqueda de soluciones para el bebedero, se mostrarán opciones de sensores y actuadores para llevar a cabo sus funcionalidades asociadas.

Para llevar a cabo la función de comedero, el sistema de alimentación creado debe ser capaz, en resumen, de mover el pienso de la reserva al comedero. Para ello, debemos tener en cuenta:

- La capacidad de la reserva de pienso, y del comedero.
- Si la reserva quedará fija o móvil, y su soporte en todo caso.
- Cómo se gestionará el movimiento del pienso entre la reserva y el comedero; principalmente, las opciones más directas son movimiento basado en la gravedad, o movimiento gestionado por un motor. También será necesario estudiar el camino que realizará el pienso, para establecer conductos por donde pueda fluir hasta el comedero sin gran dificultad (lo que entendemos por *conexionado entre reserva y comedero*).
- Soporte de los conductos que conectan la reserva y comedero.
- Conexionado eléctrico con el microcontrolador para recibir las señales de movimiento, en el caso de usar servo, o sensorización de niveles de pienso en reserva y/o en comedero.

Teniendo presente los puntos recién expuestos, se explicará brevemente en qué consisten aquellos sensores/actuadores que con mayor probabilidad podrán ser empleados.

#### *Servomotor*

Un servomotor es un motor eléctrico al que se le puede configurar tanto la velocidad, como la posición del eje que gira (también llamada *dirección del eje o giro del rotor*). Gracias a esto, con los servomotores podemos crear toda clase movimientos de una forma controlada, por ejemplo en robótica para el control del movimiento del brazo de un robot o en los sistemas de radio control.

Las características principales de un servomotor son el par y la velocidad:

- **El par:** fuerza que es capaz de hacer en su eje. El par también se puede llamar torque. Se suele expresar en Kg/cm, por ejemplo 3Kg/cm. A mayor par, mayor corriente de consumo del servo, pero no suelen consumir mucho, dependiendo del tipo de servomotor.
- **Velocidad:** velocidad angular o de rotación.

Cabe anotar que, normalmente, la tensión de alimentación de los servos en c.c. está entre 4 y 8V (voltios).

## *Partes de un Servomotor*

Un servomotor es un sistema compuesto por:

- **Un motor eléctrico:** es el encargado de generar el movimiento, a través de su eje.
- **Un sistema de regulación:** formado por engranajes, que actúan sobre el motor para regular su velocidad y el par. Mediante estos engranajes, normalmente ruedas dentadas, podemos aumentar la velocidad y el par o disminuirlas.
- **Un sistema de control o sensor:** circuito electrónico que controla el movimiento del motor mediante el envío de pulsos eléctricos. Propio del servomotor e interno.
- **Un potenciómetro:** conectado al eje central del motor que nos permite saber en todo momento el ángulo en el que se encuentra el eje del motor.

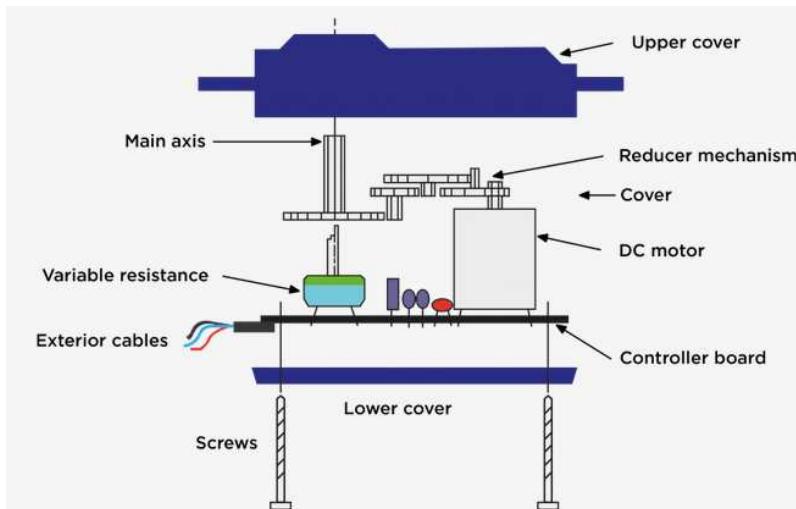


Figura 18: Partes principales de las que está conformado un servomotor [30].

Puede venir todo en una caja formando el servo, pero normalmente el servo no trae incluido el sistema de control. Si pone *encoder* incluido, quiere decir que viene el sistema de control incluido en el servo.

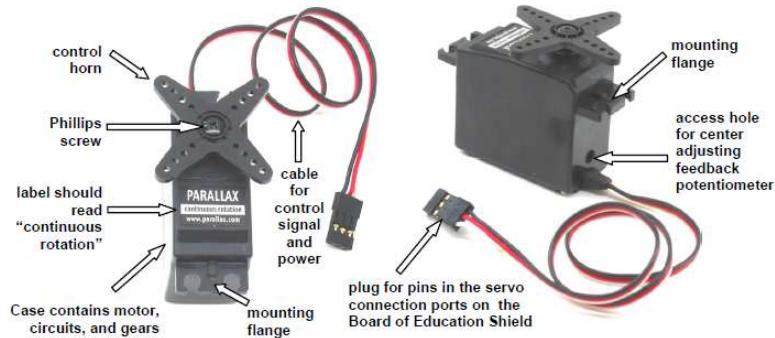


Figura 19: Partes exteriores de las que está conformado un servomotor, ejemplo del fabricante *Parallax* [31].

### Cómo funciona un servomotor

Para posicionar un servomotor tenemos que aplicarle un pulso eléctrico, cuya duración determinará el ángulo de giro del motor. Recibe los pulsos de entrada y ubica al motor en su nueva posición dependiendo de los pulsos recibidos.

Un pulso es sencillamente enviar corriente eléctrica al motor durante un tiempo determinado.

Puedo enviar un corriente durante 0,5ms (un pulso) o durante 1,5ms (otro pulso diferente). Para el pulso de 0,5ms el eje del motor estará en una posición y para un pulso de 1,5ms el eje del motor estará en otra posición. Veamos un ejemplo:

En el caso de la figura, para tiempos de 1ms, el eje del motor estará en la posición izquierda, para pulsos de 1,5ms se colocará en el medio, y para pulsos de 2ms se moverá hasta el extremo derecha, posición final del servomotor.

El rango de actuación del servo de la imagen anterior sería entre 1 y 2ms, que son los rangos más habituales de los servomotores. 1,5ms posición intermedia.

Como ves la duración del pulso posiciona el eje del motor en una posición u otra.

Como ya hemos dicho, normalmente se utiliza un pequeño circuito electrónico para generar estos pulsos de entrada.

Como hemos dicho, el control del servo se hace por medio de pulsos eléctricos.

Imagina un servo cuyos valores posibles de la señal de entrada (pulsos) están entre 1.5ms y 2.5 ms, que posicionan al motor en ambos extremos de giro (0° y 180°, respectivamente).

Fíjate en la imagen. Si mantengo el pulso 1,5ms el motor pasa de la posición inicial a la intermedia (giro de 90°). Si mantengo el pulso 2,5ms el motor gira a la posición final (giro de 180°). Si mantuviera el pulso más tiempo de 2,5ms el motor no giraría más, ya que el potenciómetro del eje detectaría que está en la posición final, y normalmente sonaría un zumbido para indicarnos que está al final del recorrido.

El valor 1.5 ms indicaría la posición central o neutra (90°), mientras que otros valores del pulso lo dejan en posiciones intermedias. El cálculo de la posición para pulsos intermedios es muy sencilla, solo hay que aplicar una regla de tres.

También hay servos que solo tiene dos o tres posiciones. Por ejemplo si mantuviéramos el pulso un tiempo menor de 1,5ms el motor se quedaría en su posición inicial. Para pulsos entre 1,5ms y 2,5ms el motor estaría en la posición intermedia y para pulsos mayores de 2,5 se colocaría en la posición final (solo 3 posiciones).

## Conección de un Servomotor

Un servomotor tiene 3 cables:

- Un cable rojo para alimentación (el positivo). La tensión de alimentación depende del servo.
- Un cable negro para conexión a tierra (el negativo o GND).
- Un cable blanco que es la línea de control por la que se le envía la señal para comunicar el ángulo en el que se debe posicionar según el pulso. Este cable irá al microcontrolador.

No todos los servomotores tienen esos mismos colores de cables (dependerá del fabricante), por lo que se debe ser precavido y buscar en la hoja de especificaciones del fabricante antes de realizar ninguna conexión. Algunas variaciones comunes en los colores de los cables de un servomotor se muestran en la Figura 20.

## Tipos de Servomotores

Hay 4 tipos fundamentales de servomotores:

- **Servomotores de corriente continua (cc):** los más habituales. Funcionan con un pequeño motor de corriente continua. El servomotor se controla por PWM (modulación por ancho de pulso), como ya explicamos.

Servo	Positive (+)	Signal (S)	Negative (-)
Futaba - J	Red	White	Black
JR	Red	Orange	Brown
Hitec	Red	Yellow	Black
Airtronics	Red	Orange	Black
	Red	White	Black
	Red	Black	Black
Airtronics - Z	Red	Blue	Black
Fleet	Red	White	Black
KO	Red	White	Black

Figura 20: Combinaciones de color posibles para los cables de un servomotor, dependiendo del fabricante [29].

- **Servomotores de corriente alterna (ac):** pueden utilizar corrientes más potentes y por lo tanto se usan para mover fuerzas mucho mayores.
- **Servomotores de imanes permanentes o *Brushless*:** se llama *brushless* ya que es un motor de corriente alterna sin escobillas (como las que llevan los de cc). Se utilizan para grandes torques o fuerzas y para altas velocidades. Son los más usados en la industria. Están basados en los motores síncronos.
- **Motor Paso a Paso:** es un motor eléctrico, pero que no gira, sino que avanza un "paso". No giran de manera continua sino por pasos, es decir, giran un número determinado de grados. La característica principal de estos motores es el hecho de poder moverlos un paso a la vez por cada pulso que se le aplique. Su control se basa en polarizar las bobinas que llevan incluidas de manera adecuada para que giren correctamente. Son ideales para la construcción de mecanismos en donde se requieren movimientos muy precisos.

### Precauciones a tener en cuenta

Los servomotores son periféricos que demandan más corriente que otros, como pueden ser sensores de temperatura u otros sensores, incluso actuadores (una pequeña bomba de agua o un relé). Es por ello, principalmente, por lo que debemos tener especial precaución a la hora de hacer uso de él.

Si la idea que tenemos es usarlo como periférico de un microcontrolador, como Arduino, se recomienda no alimentar el servomotor nunca con el pin 5V del microcontrolador. Por norma general, no es ideal alimentar ningún periférico a través de este pin, pero es especialmente poco recomendable hacerlo con un servomotor, ya que consume bastante corriente y haríamos atravesar desde la fuente de alimentación hasta el pin 5V del arduino una corriente superior a la que es aceptable por el arduino (esa corriente atraviesa los circuitos del arduino y los daña).

Otra precaución a tener en cuenta es cuando se produce la subida del programa desde el ordenador al microcontrolador, subida que se puede realizar mediante la conexión USB entre ambos. En el

momento en el que subamos el programa, se empezará a ejecutar y si tenemos conectado el servo en ese momento, dañará el serial del arduino, imposibilitando que lo podamos volver a usar y corremos el riesgo de dañar y dejar inservible también el puerto USB del ordenador al que esté conectado. Esto ocurre porque, como ya hemos mencionado, el servo consume más del amperio que suministra el USB (dependerá del servo, pero es recomendable comprobarlo y tener precaución adicional), y exigirá más corriente de la que admite el serial, por lo que se dañará. Para evitar esto, se recomienda subir el programa en el microcontrolador sin tener ningún periférico conectado, y posterior, alimentar de manera externa a dichos periféricos (sólo conectar al microcontrolador las líneas necesarias para el control o sensorización).

Existen precauciones adicionales, como no alimentar al servo fuera de sus márgenes recomendables, o tener cuidado con lo que se conoce como *stall current* y *free current*. No se entrará mucho en detalle, pero se dará una idea de lo que significan para dejar claro que los servomotores son periféricos más complejos que otros y que hay que ser consciente a la hora de usarlo.

- *Stall current*. Máxima corriente que se genera cuando el motor está aplicando su máximo torque.
- *Free current*. Corriente que se genera cuando el motor está girando libremente a su máxima velocidad, sin otra carga que la propia fricción y fuerzas contraelectromotrices (*back-EMF*).

### 3.2.2.4 Bebedero

De manera análoga a 3.2.2.3, para cubrir esta funcionalidad necesitaremos conseguir la manera de mover agua desde la reserva hasta el bebedero. Para ello, de igual manera, hay que tener en cuenta:

- La capacidad de la reserva de agua, y del bebedero.
- Si la reserva quedará fija o móvil, y su soporte en todo caso.
- Cómo se gestionará el movimiento del agua entre la reserva y el bebedero.
- El soporte de los conductos que conectan la reserva y el bebedero.
- El conexionado eléctrico con el microcontrolador para que éste envíe señales de control o pueda recibir datos resultado de la sensorización.

Como opciones más directas para solucionar esta funcionalidad, están:

- Sensor ultrasónico, situado en la parte superior de la reserva, para realizar la sensorización del nivel de agua de la reserva, y para gestionar el movimiento del agua desde la reserva hasta el bebedero, la combinación de relé y bomba de agua.
- Sensores de nivel de agua (flotación) para realizar la sensorización del nivel de agua de la reserva, y para gestionar el movimiento del agua desde la reserva hasta el bebedero, la combinación de relé y bomba de agua.

A continuación, se introducirán estos elementos de manera individual para comprender su funcionamiento y su manera de actuar dentro de esta funcionalidad en concreto.

#### *Sensor ultrasónico*

Existen varios modelos de sensor ultrasónico compatibles con arduino, pero el que vamos a destacar se caracteriza por ser muy fácil de usar y tener un bajo costo. Se trata del HC-SR04, sensor de distancia de baja precisión basado en ultrasonidos. El aspecto del HC-SR04 es muy característico y se reconoce con facilidad (mirar Figura 21). Además, es un elemento bastante popular en los kits de iniciación para Arduino y necesario para multitud de proyectos. Se identifica fácilmente porque tiene dos «ojos» que realmente son los dispositivos de ultrasonidos que integra este módulo. Uno de ellos es un emisor de ultrasonidos y el otro un receptor. Trabaja a una frecuencia de 40 kHz, por tanto es inaudible para los seres humanos.



Figura 21: Captura de un sensor ultrasónico HC-SR04 [60].

Su funcionamiento se basa en el envío de un pulso de alta frecuencia. Este pulso rebota en los objetos cercanos y es reflejado hacia el sensor, que dispone de un receptor adecuado para esa frecuencia. El sonido se propaga a una velocidad constante en el aire, con lo cual si calculamos el tiempo que demora en emitirse el pulso y la recepción del mismo, causada por el rebote sobre un objeto, podremos calcular la distancia. Dado que la velocidad del sonido es 343 m/s en condiciones de temperatura 20 °C, 50 % de humedad y presión atmosférica medida a nivel del mar, se tiene que:

$$343 \frac{m}{s} \cdot 100 \frac{cm}{m} \cdot \frac{1}{1000000} \cdot \frac{s}{\mu s} = \frac{1}{29,2} \frac{cm}{\mu s} \quad (2)$$

Por tanto, se puede conocer la distancia a un objeto simplemente teniendo en cuenta el valor anterior, de forma que la distancia viene dada por la mitad del tiempo que ha tardado la señal en regresar al sensor. Es decir, si se ha medido 292 microsegundos, entonces la distancia es 5 cm. Para mejor comprensión se muestra esquema en la Figura 22. Si las condiciones de presión temperatura y humedad ambiental son diferentes las medidas no son tan exactas, aunque el error puede despreciarse.

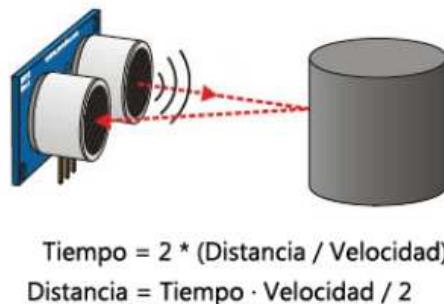


Figura 22: Esquema de funcionamiento de un sensor ultrasónico HC-SR04 [61].

Por último, señalar que este es un sensor activo, ya que debemos alimentarlo para usarlo y también incluye varios circuitos integrados que son los encargados de generar el pulso de sonido y luego recibirlo. Estos circuitos nos facilitan la programación; es por ello que veremos que solo incluye cuatro pines, VCC, Trigger, Echo y GND (mirar Figura 21):

- VCC y GND será la alimentación que obtendremos directamente del arduino.

- Trigger será el pin al cual le enviaremos una señal para que dispare el pulso ultrasónico.
- Por el pin Echo obtendremos otro pulso indicando que la señal ha sido recibida.

### *Bomba de agua*

Se trata de un motor a efectos prácticos, que funciona con corriente continua. Como será un elemento que, con total seguridad, será elegido para estar en el prototipo final, aparecerá en el apartado 3.3.2 con mayor detalle.

### *Sensores de nivel de agua*

Los sensores de nivel de flotación de agua se comportan como switches, y su respuesta (valor que da como salida) son valores lógicos (0 o 1). Como sensor, se usa como periférico a través del cual se pretende obtener ciertos valores para que el microcontrolador pueda actuar en consecuencia (en base a esos valores).



Figura 23: Captura de sensores de flotación [15].

Su funcionamiento es sencillo: si el nivel del agua queda por encima del sensor, el valor lógico asociado que enviará al microcontrolador será de 1. Si por el contrario queda por debajo, mandará un 0. Un sensor de flotación tiene dos cables de alimentación, simétricos, uno para VCC y otro para GND; entre dichos terminales y el microcontrolador se requiere de resistencias de 10k en configuración *pull-down* (mirar Figura 24), ya que al tratarse de switches, al conmutar si no existe esa resistencia da un valor lógico o desconocido o falso 1, consecuencia del ruido del pin que se introduce al estar flotante. El valor de la resistencia *pull-down* dependerá de la situación (leer Glosario para más información).

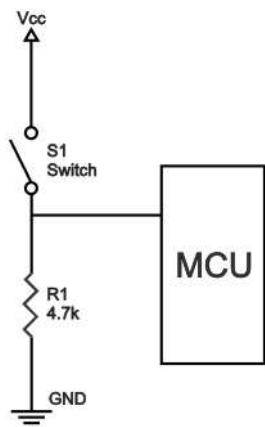


Figura 24: Esquema de un switch conectado a un microcontrolador con una resistencia *pull-down* [51].

### *Relé*

Un relé es un interruptor electromecánico que se conmuta (cambia de estado) mediante una señal eléctrica, y es digital, ya que sólo puede tomar el valor abierto y el cerrado (1 o 0, True o False). Sería algo parecido a un pulsador, pero sustituyendo el gesto mecánico de un dedo pulsando por una señal eléctrica. De esta forma, mediante una señal eléctrica de poca intensidad podemos controlar un circuito de mucha mayor potencia, aislando la parte de control.

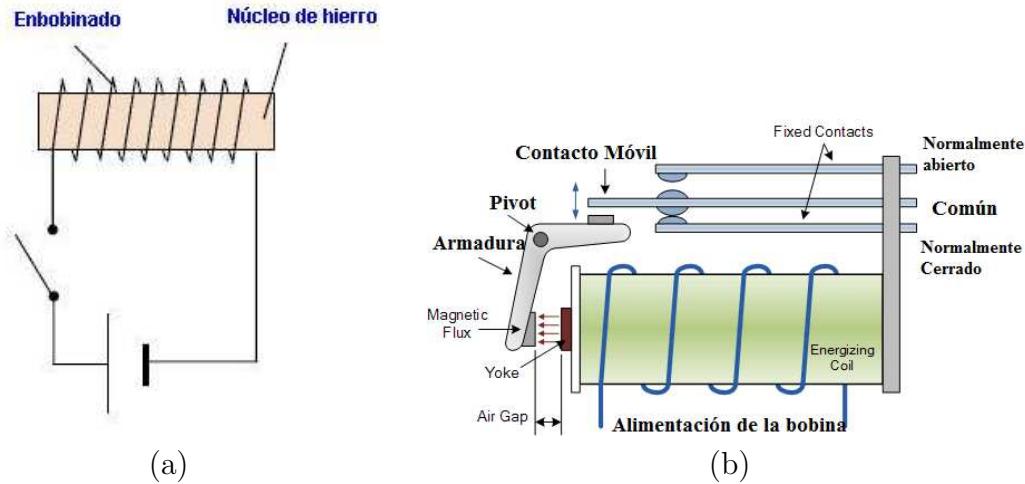


Figura 25: Imágenes que muestran las partes fundamentales de las que consta un relé [18].

El circuito de la Figura 25(a) muestra la base de un relé; consiste sencillamente en un núcleo de hierro (ferromagnético) y un cable conductor eléctrico, arrollado alrededor del núcleo, que se conecta a una batería de corriente continua. Si cerramos el interruptor, en el hierro se induce un campo magnético sostenido, siguiendo las reglas de las leyes de Maxwell. Es decir se convierte en un electroimán. Si el núcleo no está sujeto a las espiras devanadas, se desplazará lateralmente, y si está sujeto el conjunto se comporta como un imán que podemos encender y apagar, lo que aprovechamos, para construir los relés.

Fijándonos en la Figura 25(b), podemos observar que, cuando damos tensión a la bobina, el electroimán atrae la armadura haciaéndola bascular alrededor del pívot, lo que desplaza el contacto móvil hacia arriba conectando el pin *normalmente abierto* con el *común* cerrando el circuito. A este contacto se le llama normalmente abierto (NA), porque el circuito entre el común y el pin NA, esta normalmente abierto (o sin conexión) hasta que se activa el relé. Por el mismo motivo hablamos del normalmente cerrado (NC), ya que el contacto está cerrado cuando el relé no está excitado. Cuando la bobina pierde la alimentación, el efecto imán desaparece y un resorte devuelve la armadura a la posición de reposo. A este tipo de relé, se le llama de armadura.

Se podría llegar a pensar que los relés deberían tener 4 contactos, dos para la alimentación y dos para cerrar el circuito. Aunque existen, esto tendría el problema de que habría que fabricar un modelo NC y otro NA lo que supone un coste de fabricación mayor. Por eso, la mayor parte de los relés son de 5 pines, 2 se usan para comutar el relé y los otros tres son el común y los pines NA y NC (al final un relé es un mecanismo, en el que una parte móvil metálica, cierra un contacto o lo abre).

Seguramente la funcionalidad de un relé recuerde un poco a los transistores, pero tienen finalidades diferentes:

- Hay límites en la corriente que puede aceptar un transistor, sin embargo se pueden diseñar

relés para aguantar cualquier carga.

- El relé aísla completamente el circuito de control del de potencia, algo muy importante para tensiones altas. Utilizaremos relés cuando tengamos cargas de mucha potencia.
- Un transistor conmuta (pasa de corte a saturación o viceversa) millones de veces más rápido que un relé. Utilizaremos un transistor para cargas pequeñas y cuando la velocidad de conmutación sea importante.
- Sin embargo, hay que tener en cuenta que el relé cuenta con mayor desgaste, ya que cada vez que conmuta implica el movimiento de partes móviles mecánicas.

Tanto los transistores como los relés tienen la ventaja de disponer de un circuito de baja (control), y otra circuito de alta (carga), por lo que se aísla eficazmente el circuito de control, normalmente de baja tensión, como nuestro Arduino, del circuito de carga elevada que puede ser un motor de una cierta potencia, y por consecuente, capaz de inhabilitar nuestros circuitos de control. Por eso, cuando se manejan media y alta tensión en la industria, se suelen usar relés para conseguir aislamiento galvánico, por razones de seguridad, y se exige aislar la tierra del circuito de potencia y el negativo o GND del circuito de control, garantizando así el aislamiento galvánico de los dos circuitos. En el mundo Arduino se suelen usar relés de baja potencia, como mucho tensiones alternas alternas de 220V y máximo 5A para pequeños motores.

Aunque hay muchos tipos de relés, por regla general nuestro Arduino no tiene potencia suficiente para activar un relé. Utilizaremos un transistor para amplificar la señal de la base y así poder conmutar el relé. El ejemplo que veremos a continuación incluye un circuito de transistor/relé completo que nos podrá valer para cualquier proyecto que lo requiera.

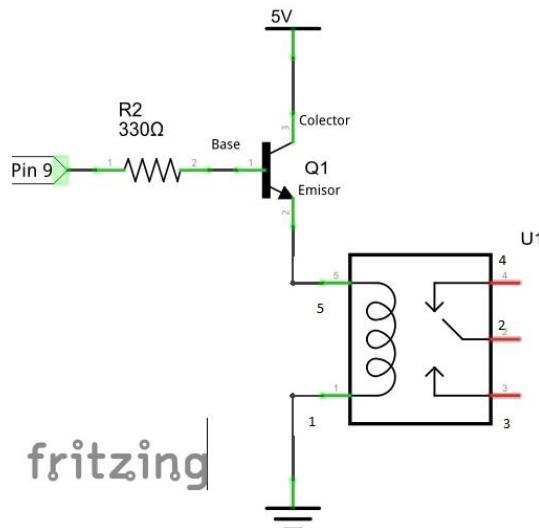


Figura 26: Ejemplo de circuito que emplea relé y un transistor entre relé y pin 9 de un arduino [19]

- Cuando hay un valor HIGH en el pin de control, el transistor pasa a saturación y la corriente entre emisor y colector excita la bobina del relé, haciendo que el contacto cambie de posición (y haciendo además un clic muy característico).
- Si hay un valor LOW en el pin de control el transistor entra en corte e impide el flujo de corriente por lo que la bobina cae y el contacto de salida vuelve a su posición de reposo.
- La salida del relé consta de 3 pines y no de dos como se podría esperar. El motivo es que la conexión entre los pines de salida 2 y 3 es de tipo normalmente abierto (circuito abierto sin excitar) y entre los pines 2 y 4 es normalmente cerrado (circuito cerrado sin excitar la bobina).

#### *Resumen de las opciones para bebedero*

A continuación, se mostrará un resumen de las opciones planteadas al principio de este apartado, tras la visualización del funcionamiento individual de cada componente:

- **Opción A:** *sensor ultrasónico con relé y bomba de agua.* El sensor ultrasónico se situaría en la tapa de la reserva, apuntando al agua. Se programaría de tal manera que, habiendo calibrado el sensor y calculado la distancia máxima a la que se encontraría un nivel crítico de agua (nivel que indicaría que hay que llenar la reserva), si se llega a ese nivel, no se permitiría más recargas de bebedero, y nos avisaría de que hay que llenar la reserva de agua. Mientras tanto, a petición del usuario, el bebedero se rellena gracias a la acción de la bomba, acción permitida por el relé a través de la señal de control del microcontrolador, el cual sabe cuándo debe enviarla (se especifica en programación).
- **Opción B:** *sensor de nivel de flotación con relé y bomba de agua.* Se situarían dos o más sensores de nivel de flotación en uno de los lados de la reserva (en su eje longitudinal), de manera que se podría conocer el estado de la reserva de agua; cuantos más sensores, más preciso será el estado de la reserva. Por ejemplo, con 2 sensores de flotación podríamos tener 4 estados, lleno, por debajo del 70 %, por debajo de 50 % y menos de 25 % (estado crítico en el que habría que llenar la reserva). El movimiento del agua de la reserva al bebedero sería el mismo que en la Opción A; mientras no estemos en el estado crítico, y según haya configurado el usuario (bajo petición y/o cada cierto tiempo), el microcontrolador mandará una señal de control al relé que activará la bomba o la desactivará, de manera que se va llenando el bebedero.

### **3.2.3 Prototipo exterior**

Se trata de una funcionalidad que depende muy estrechamente de las funcionalidades presentadas anteriormente, por lo que para materializar una solución apta habrá que solventar las funcionalidades que le preceden. Se hablará más en detalle en el apartado 3.3.3, una vez se haya expuesto la elección de soluciones para las funcionalidades de alimentación autónoma y automatización y monitorización.

### 3.3 Elección de soluciones para cada funcionalidad

#### 3.3.1 Alimentación autónoma del dispositivo

Para conseguir el objetivo de una alimentación autónoma para el dispositivo de alimentación final que queremos crear, se ha decidido usar la siguiente combinación de componentes, explicados de manera teórica y más ampliamente en el apartado 3.2.1.

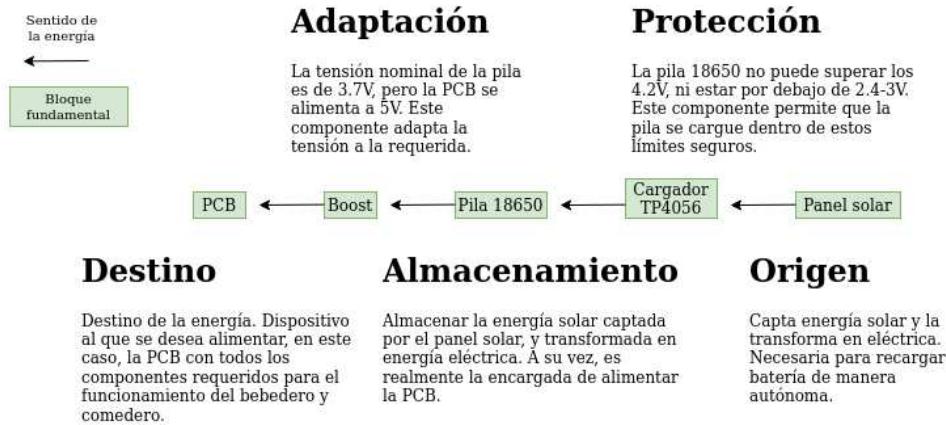


Figura 27: Esquema general de la alimentación elegida para este proyecto.

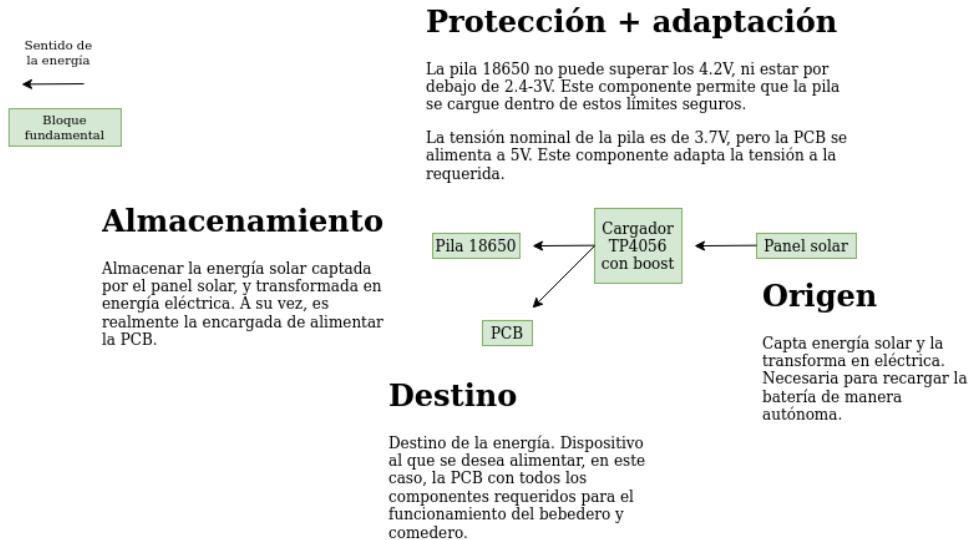


Figura 28: Esquema general de la alimentación elegida para este proyecto (TP4056 con boost incluido).

Así pues, se adquirieron los siguientes componentes:

- **Batería recargable Li-ion 18650 sin protección.** Con tensión nominal de 3.7V, para un proyecto en el que las tareas de máximo consumo (rellenar comedero y envío de datos por LoRa) se van a ejecutar, como máximo, dos veces al día (en coste temporal, serán 5 minutos en total), en principio, pueden ser suficientes.
- **Kit protección 18650**, para pilas sin protección. Las pilas que vienen sin protección llevan, como mínimo, una ausencia de protección contra cortocircuitos, es decir, si eléctricamente se conectan sus polos podemos dejar inservible la pila.
- **TP4056 con boost a 5V**. Al tener una tensión nominal de 3.7V, y requerir nuestro proyecto de una alimentación de 5V, necesitaremos algún elemento que eleve esta tensión, que, en nuestro caso, será un boost. No vendrá aislado, ya que lo usaremos en conjunto dentro de una PCB que corresponde a un módulo de carga de baterías li-ion llamado TP4056.
- **Panel solar 6V**. Se ha escogido un panel solar que, en teoría, nos da una tensión mínimamente funcional, de tamaño reducido para un primer prototipo y aproximación a la energía solar. Nos permitirá añadir la opción de recargar la batería de manera autónoma.

El elemento *PCB* de la Figura 28 será explicado más adelante, en la Sección 4. A continuación, se muestran los pasos seguidos para montar el sistema de alimentación autónoma para nuestro dispositivo de alimentación para animales.

### 3.3.1.1 Preparación de la alimentación autónoma del dispositivo

Lo primero fue dejar asegurada la pila. Como se indica, la batería 18650 no tiene protección. Esto significa que cualquier manipulación incorrecta de la misma puede derivar en la rotura de la pila, o incluso en que ésta se inflame. Es por ello que se hizo uso de un kit para protección de baterías 18650, como el que se muestra en la Figura 29.

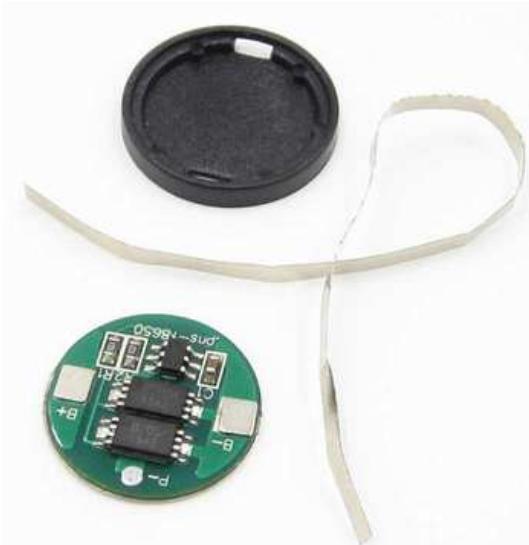


Figura 29: Detalle del kit de protección para una batería 18650, a falta de tubo termoretráctil y contactos para los polos [24].

La protección que se ha añadido con este kit viene determinado por ciertos componentes electrónicos (como el que muestra la Figura 29), pero, además, por el recubrimiento metálico y plástico que se ha añadido a su alrededor. La construcción de la protección alrededor de la batería 18650, a través del kit adquirido, un termoretráctil y dos contactos metálicos para ambos polos, se resumen en la Figura 30.

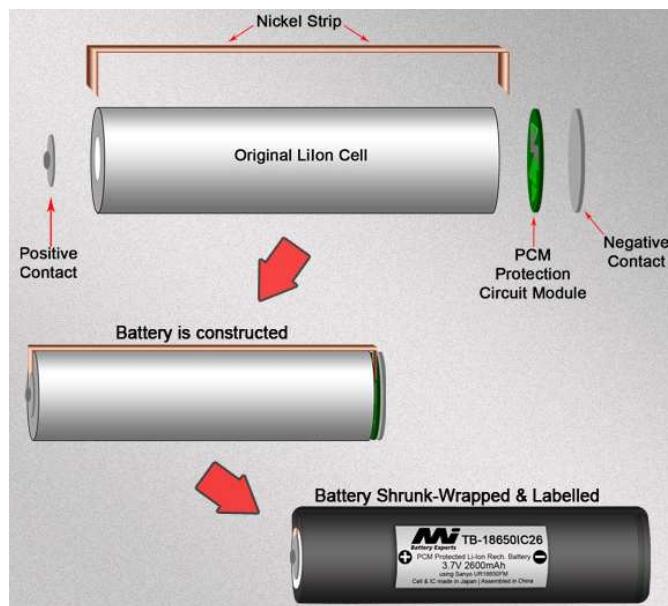


Figura 30: Esquema de la protección que se añade a las pilas Li-ion [23].

Sin entrar en mucho detalle, esta PCB (Figura 31) que incluye el kit (PCM, según la Figura 30) actúa como protección extra para la batería en tanto a cortocircuitos, sobrecarga, sobredescarga y exceso de corriente. La batería incluye interiormente cierta protección, pero siempre es recomendable que si la batería no cuenta con este kit, se le añada (o se adquiera con él ya incluido y no añadirlo nosotros). Se ha señalado en la imagen, con un rectángulo rosa, el conjunto de dos MOSFET (8205A) que actúan como switches en las situaciones anteriormente mencionadas (por ejemplo, si hay exceso de corriente, corta el flujo, como un interruptor); señalado con el rectángulo verde, el circuito integrado DW01A, que permite realizar la lógica necesaria para detectar estos casos y actuar en consecuencia.

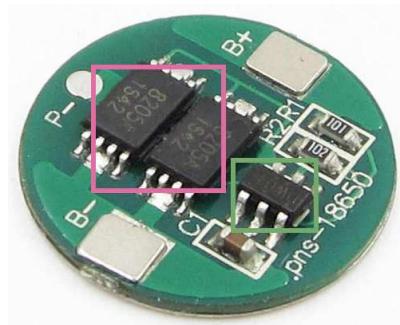


Figura 31: Detalle de la PCM de protección de la batería 18650 [24].

Finalmente, se muestra en la Figura 32 fotos de la pila que se ha usado en este proyecto, antes de ponerle protección y después.

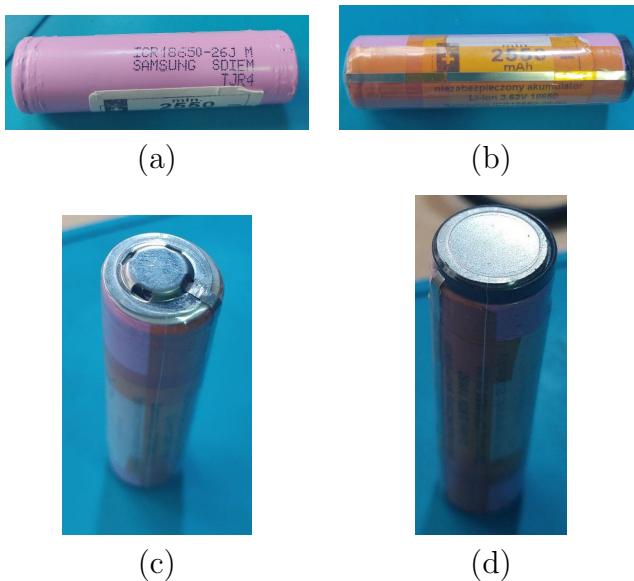


Figura 32: Fotos de la batería 18650 usada (sin protección, y con protección añadida).

Como se indicó en el apartado 3.2.1, dentro de *opciones para baterías recargables*, se necesita de un circuito que ayude a la pila, de tipo Li-ion, a una correcta carga y descarga, precisamente porque conlleva las mismas consecuencias de una mala manipulación, expuestas anteriormente.

## TP4056

Es un circuito integrado que se emplea para gestionar la carga de la batería Li-ion. Realmente, es una PCB donde encontramos el IC TP5046, junto con más componentes, pero de aquí en adelante hablaremos de TP4056 como la PCB global. Su funcionamiento, a grandes rasgos, consiste en transformar la fuente de energía eléctrica que tiene a su entrada para cargar la batería que tiene conectada a su salida; debido a que las baterías requieren que la energía sea cualitativa y cuantitativamente de una manera concreta, este paso es necesario. Además, también controla la temperatura. Las fuentes de energía eléctrica pueden ser diversas, desde un adaptador conectado a la red eléctrica, una fuente de alimentación, un panel solar, un generador, etc. No en todos los casos se necesitan los mismos componentes, depende de cada caso.

Un ejemplo de TP4056 sencillo y fácil de encontrar sería el siguiente:

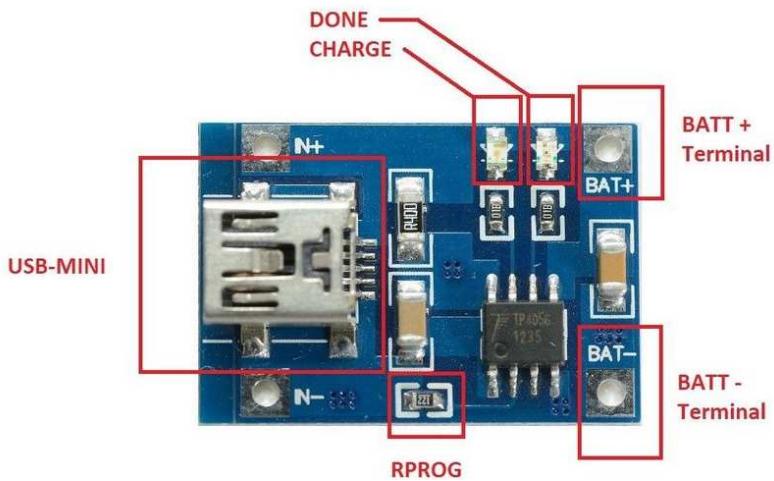


Figura 33: Ejemplo de un TP4056 simple [47].

Del esquema anterior del módulo TP4056, podemos destacar:

- **Puerto mini-USB:** para alimentar tanto la batería como el TP4056 a través de este tipo de cables. Lo usaremos si deseamos cargar la batería de manera externa al proyecto.
- **Bornas IN-, IN+:** están a los lados del puerto. Sirven también para alimentar la batería (y el TP4056). A estas bornas podemos conectar una placa solar, u otra fuente que se necesite. En nuestro caso, será el panel solar de 6V.
- **LEDs de carga y completado:** avisarán cuando la batería esté cargándose o cuando haya terminado dicho proceso.

- **BAT+ y BAT-:** son bornas de salida que irán conectados a la batería que se necesita cargar.

Sin embargo, se requieren más elementos para alimentar al dispositivo, como por ejemplo un conversor de tensión tras la batería (entre batería y dispositivo). En nuestro caso, lo requerimos, ya que la batería usada otorga 3.7V nominales (no existen baterías de 5V) y nosotros necesitamos 5V para alimentar el sistema de alimentación. Necesitamos un elemento que nos aumente la tensión a su salida, para alimentar de manera correcta el sistema creado; usaremos, por tanto, un *boost converter*. Es un dispositivo que puede encontrarse aislado, aunque también podemos encontrarlo en una misma PCB con el TP4056, y será esta última opción la que usemos (Figura 34). Una opción diferente que podríamos haber escogido hubiera sido la de emplear varias baterías 18650 en serie, ya que suman sus tensiones, además del uso de un BMS (como se ha mencionado ya anteriormente en este documento, un *Battery Management System*, que permita a las baterías cargarse y descargarse de manera balanceada), y un *buck converter* en este caso, ya que tendríamos más de 5V (por lo que, con el buck, bajaríamos la tensión de salida). Esta opción se descartó por la simple razón de que el dispositivo no estará funcionando todo el día ni se estima elevado consumo; finalmente, se optó por la opción más sencilla de una única batería, la cual si se observa que es insuficiente, se debería estudiar la opción alternativa (inclusive baterías en paralelo también). Por último, anotar que, si usáramos otras baterías de menor tensión nominal, no necesitaríamos un buck (si la suma de sus tensiones fuera exactamente de 5V), pero estas baterías suelen ser no recargables, y no conseguiríamos el objetivo de autonomía en la alimentación del dispositivo.

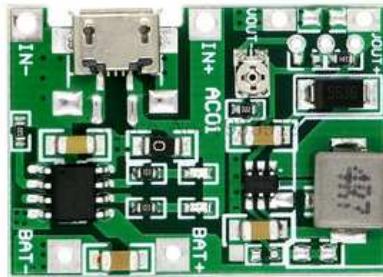
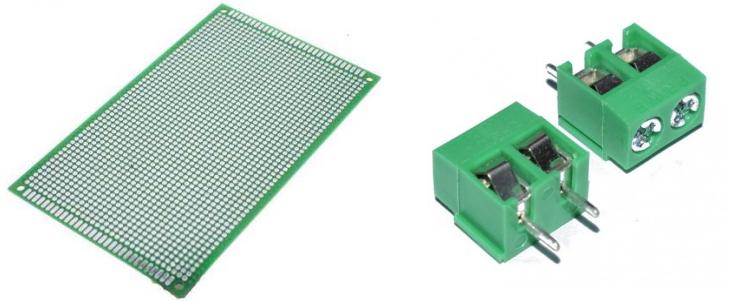


Figura 34: Ejemplo de un TP4056 con boost [26].

Vemos en la Figura 34 que esta PCB cuenta con los elementos principales anteriormente mencionados sobre la Figura 33 (bornas IN+ e IN-, bornas VOUT + y VOUT-, bornas BAT- y BAT+, micro USB y LEDs). Cuenta con 2 IC's principalmente: el TC4056A y el B6289A. El primer IC viene a ser uno con funcionamiento similar al TP4056, y el segundo es el encargado de la función de boost. Cabe señalar que contamos con un potenciómetro con el que podemos variar la corriente con la que se cargará la batería (página 6 del datasheet del TC4056).

Para compactar este conglomerado (pila con TP4056), se han unido soldando las conexiones entre ellos en una placa experimental, como la que se muestra en la Figura 35(a). En concreto, se ha soldado el TP4056 de la Figura 34 con conectores de tipo screw terminal (Figura 35(b)); de esta manera, conectaremos el panel solar a través de sus cables a un screw terminal, el cual unirá mediante soldadura a las bornas IN+ e IN-, y lo mismo sucederá con la pila.



(a) Placa experimental [25] (b) Screw terminal [27]



(c) Holder para una batería 18650 [28]

Figura 35: Elementos que formarán nuestro conglomerado provisional de alimentación del dispositivo.

A continuación, en la Figura 36, se muestra el conglomerado final de los elementos presentados en la Figura 35. Gracias a la presencia de los *screw terminals* podemos conectar y desconectar el conglomerado fácilmente de la fuente de energía de entrada y del dispositivo al que alimentemos, dejándolo de nuevo aislado (tal como aparece en la Figura 36) para probarlo o simplemente moverlo (podemos desconectar la entrada de energía, la cual entra por las bornas IN en este caso, y la salida de energía, la cual sale por las bornas VOUT).

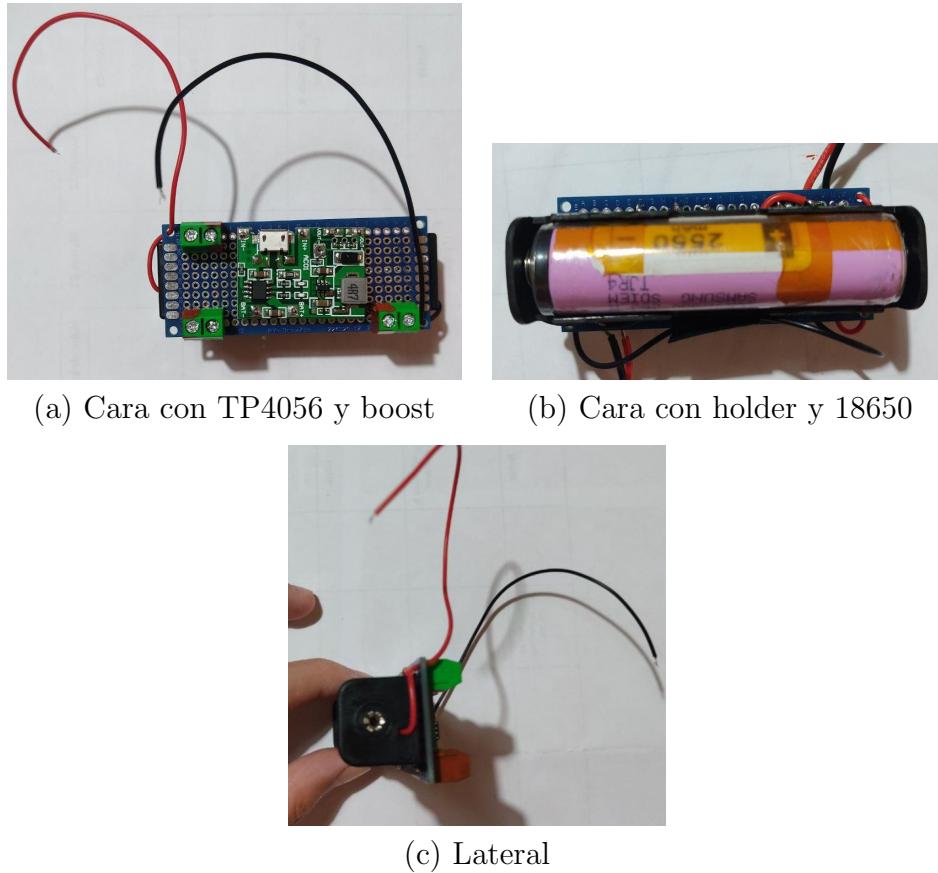


Figura 36: Vistas del conglomerado de alimentación ya terminado.

En la Figura 37 se muestra la pieza impresa en 3D, que actuará como sujeción para el panel solar, adherida mediante silicona al propio panel; además, se han soldado los cables necesarios para conectar el panel solar con el TP4056.

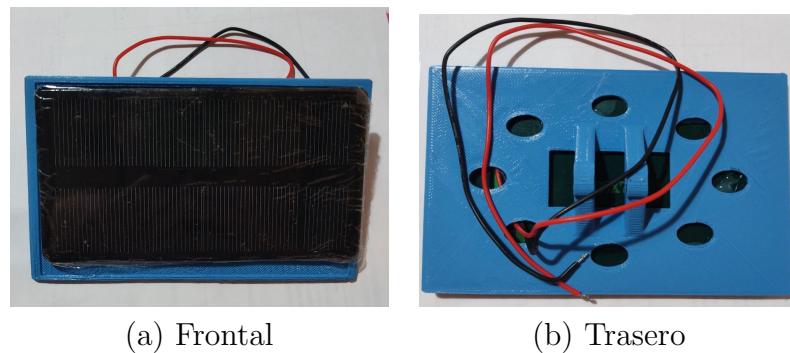


Figura 37: Vistas del panel solar usado.

Por último, en la Figura 38 se muestra el conjunto del sistema de alimentación mediante el cual se alimentará el dispositivo.

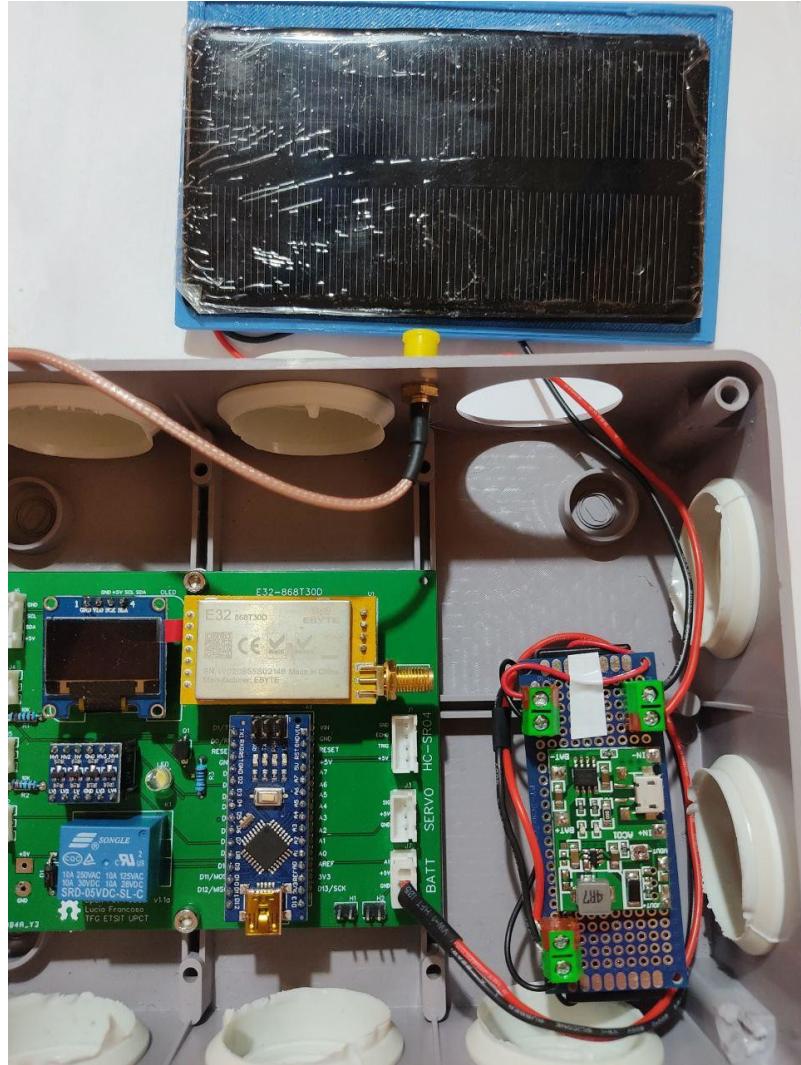


Figura 38: Conjunto global del sistema de alimentación autónoma creado

### 3.3.2 Automatización y monitorización

Para la automatización y monitorización del sistema de alimentación, se ha elegido, en tanto a los bloques de microcontrolador y comunicación radio, las tecnologías Arduino y LoRa, respectivamente. Si bien estas elecciones se habrían realizado a priori por ser opciones *Open Source* muy extendidas y conocidas por la comunidad, no ha supuesto que no se investiguen otras, las cuales se han mencionado en el apartado 3.2.2.

### 3.3.2.1 Entorno Arduino

Arduino es una plataforma de electrónica *open-source* basada en hardware y software fáciles de usar. Nació en el *Ivrea Interaction Design Institute* (Italia) como una herramienta fácil para la creación rápida de prototipos, dirigida a estudiantes sin experiencia en electrónica y programación. Tan pronto como llegó a una comunidad más amplia, la placa Arduino comenzó a cambiar para adaptarse a las nuevas necesidades y desafíos, tanto de usuarios principiantes como expertos.

Las placas Arduino son capaces, por una parte, de ejecutar una serie de instrucciones (determinadas por el usuario) a través de su microcontrolador, de manera que a partir de la lectura de sus entradas analógicas/digitales se puedan generar salidas (analógicas/digitales); por otra parte, son programables a través del software Arduino, lo que se conoce como Arduino IDE.

#### Características

La principal característica de Arduino es la simplicidad para trabajar con microcontroladores. A raíz de esto, ofrece una serie de ventajas para sus usuarios, como son:

- **Es de bajo coste.** Las placas de Arduino son relativamente de bajo coste comparadas con las plataformas de otros microcontroladores. La versión más económica de un módulo Arduino puede ser ensamblada a mano, e incluso los módulos Arduino pre-ensamblados cuestan menos de 50 \$.
- **Multiplataforma.** El software de Arduino, Arduino IDE, es capaz de ejecutarse en Macintosh OSX, Windows y Linux. La mayoría de sistemas de microcontroladores están limitados a Windows.
- **Entorno de programación simple y claro.** El software de Arduino (IDE) es fácil de usar para principiantes, sin dejar de ser lo suficientemente flexible para usuarios más avanzados.
- **Software Open Source y extensible.** El software Arduino está publicado como un conjunto de herramientas open source, disponible para ser ampliada por otros programadores a través de librerías basadas en el lenguaje C++; las personas que quieran comprender los detalles técnicos pueden dar el salto de Arduino al lenguaje de programación AVR C en el que se basa. Del mismo modo, se puede agregar código AVR-C directamente en los programas Arduino si así se desea.
- **Hardware Open Source y extensible.** Los planos de las placas Arduino están publicados bajo licencia *Creative Commons*, así pues, diseñadores de circuitos experimentados pueden hacer su propia versión del módulo, extendiéndola y mejorándola. Incluso usuarios sin experiencia pueden crear una versión en protoboard del módulo para entender cómo funciona y ahorrar dinero.

#### Hardware: placas Arduino

Existen diferentes placas Arduino, cuya elección dependerá del objetivo que persiga nuestro proyecto y de los requerimientos y limitaciones sobre dicho proyecto.

Algunas de las placas básicas de Arduino más usadas son Arduino nano, Arduino uno o Arduino mega. Nos centraremos en el Arduino nano, ya que es el que vamos a usar; aún así, ya se comentaron las especificaciones y características del Arduino uno en el apartado 3.2.2, sección *Microcontroladores*, como ejemplo de placa Arduino de cara a comparar con otras opciones de microcontrolador, por lo que no se volverá a explicar en esta sección.

### *Especificaciones del Arduino nano*

- Microcontrolador: ATmega328
- Arquitectura: AVR
- Tensión de funcionamiento: 5V
- Memoria flash: 32kB de los cuales 2kB son para el gestor de arranque (*bootloader*).
- SRAM: 2kB
- Velocidad del reloj: 16 MHz
- Pines de entrada analógicos: 8
- EEPROM: 1 kB
- Corriente CC por pin de entrada/salida: 40mA (pines I/O)
- Tensión de entrada: 7-12V
- Pines digitales de entrada/salida: 22 (de los cuales 6 son PWM)
- Salida PWM (*Pulse Width Modulation*): 6
- Consumo de energía: 19 mA
- Tamaño de la PCB: 18 x 45 mm
- Peso: 7g

*Anotaciones:* usamos una versión diferente a la de Arduino nano vendida por la organización Arduino. Las diferencias en hardware son ínfimas, y en software (en el Arduino IDE) simplemente debemos indicarlo en *Tools > Processor*. Dichas diferencias son el procesador, que es un ATmega328P y el regulador, el cual es CH340.

- El microcontrolador no es ATmega328 (usado en el nano de la organización Arduino), sino que es ATmega328P. La diferencia reside en que es una versión posterior diseñada para consumir menos potencia que el 328, esencialmente tienen la misma arquitectura y características en tanto a memoria flash, EEPROM y SRAM (la P viene de *picoPower*).
- Este arduino usa un regulador diferente, el CH340G, frente al FT232 que usa el arduino original. Además, el fusible/diodo Schottky que usan para proteger la comunicación serial frente a sobrecorrientes es diferente, siendo el MBR0520 para el caso de los arduinos originales, frente al B5819W que usa el arduino que vamos a usar.

### *Pinout del Arduino nano*

La funcionalidad de cada uno de los pines analógicos y digitales se pueden ver en la Figura 39.

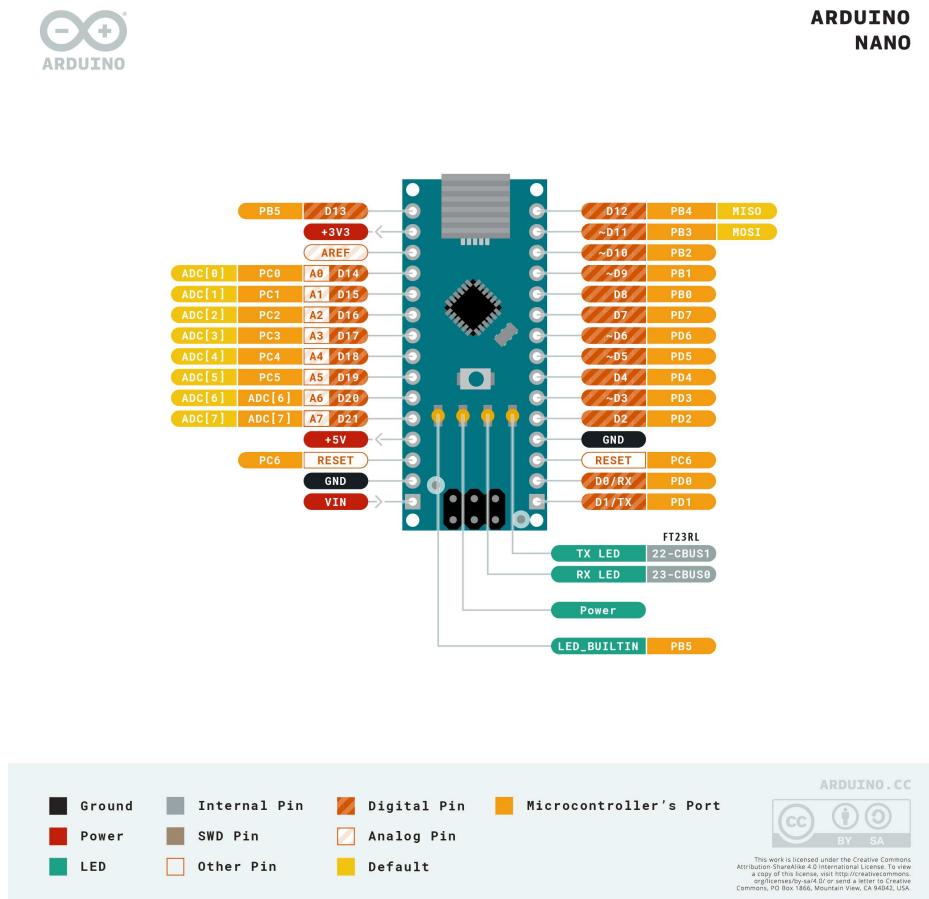


Figura 39: Pinout de un Arduino nano.

Tendremos la misma cantidad de pines digitales configurables como entrada o salida (D2 a D13), donde 6 de ellos tendrán capacidad PWM, igual que arduino uno. Respecto a los pines analógicos tenemos los pines A0 a A5 igual que el modelo uno, pero aparecen dos pines adicionales, A6 y A7, consiguiendo así un total de 8 entradas analógicas.

[Foto pines digitales y analógicos, señalando los que son PWM.]

Los dos primeros pines de la esquina inferior derecha corresponden a D1 para TX y D0 para RX de la comunicación serie por UART(*Universal Asynchronous Receiver-Transmitter*); en Arduino uno estos mismos pines también aparecían pero en orden invertido. Son un reflejo de las líneas de comunicación provistas por el conversor USB-TTL, aunque por lo general usamos la librería *Software serial* para definir otros dos pines cualquiera para dicha función. Ya que estamos hablando de comunicación serie, tenemos los pines asociados a la interfaz SPI (*Serial Peripheral Interface*), iguales que en el modelo uno; el pin digital 10 ejecutando la función de *slave select*, el 11 como MOSI (*Master Output Slave Input*), el 12 como MISO (*Master Input Slave Output*) y el 13 como *serial clock*. Al igual que en el modelo uno, la interfaz de dos cables, I2C (*Inter-Integrated Circuit*), estará presente en A4 para SDA (*System Data*) y en A5 para SCL (*System Clock*).

[Foto pines digitales y analógicos, señalando los especiales para comunicaciones SPI, I2C y UART.]

El resto de pines son: salida de 3.3V, referencia de las entradas analógicas AREF, salida regulada de 5V (usada normalmente para alimentar sensores y dispositivos), y dos pines de GND en ambos lados de la placa; un pin de RESET para realizar dicha función de forma remota si no tenemos acceso al pulsador físico de la placa. Finalmente, Vin se usará para alimentar la placa con una fuente no regulada de entre 7 y 12V (en arduino uno teníamos un conector para alimentación externa de tipo Jack, en arduino nano, como la placa tiene un tamaño reducido, no resultaría práctico colocar un conector tan voluminoso, por eso tenemos un pin destinado para alimentación externa).

[Foto pines digitales y analógicos, señalando los especiales para comunicaciones SPI, I2C y UART, y pines de alimentación.]

El pin de salida 5V refleja los mismos 5V del puerto del ordenador; si ahora deseamos trabajar de manera autónoma, sin conexión al ordenador, podremos alimentar la placa con una fuente externa con valores de tensión de entre 7 y 12 V. Dicha tensión se aplica de forma directa al regulador lineal de tensión, el cual convertirá la tensión de entrada no regulada a 5V regulados con un suministro máximo de corriente de hasta 500mA. Para la salida de 3.3V, es diferente en nano respecto de uno; en nano, los 3.3V son generados por el propio conversor USB-TTL (el circuito integrado lleva incorporado el regulador); el modelo uno dispone de un regulador por separado y destinado exclusivamente para el riel de 3.3V. En uno, dicho regulador puede suministrar hasta 150 mA de corriente máxima, mientras que en nano, como no tenemos un regulador especializado, y forma parte de la funcionalidad del conversor, el suministro de corriente será menor, de 50 mA como máximo.

Cabe mencionar que si conocemos bien la fuente de alimentación, es decir, sabemos que es estable en 5V, podemos aplicar dicha tensión directamente al pin 5V. Retomando una anotación anterior, realizada en el apartado 3.3.1.1, acerca de una opción alternativa de batería, como el uso de varias en serie, aumentando la tensión nominal total, con el uso de un buck que redujera la tensión que se suministra al dispositivo. Esto es si se introduce la alimentación a través del pin 5V, si se introduce a través del Vin (siempre y cuando se llegue a los 7V mínimo con esa tensión nominal total) no es necesario el buck para reducir la tensión y dejar estable esa tensión, ya que entrando por Vin pasa por el regulador de tensión del Arduino.

[Foto parte rear del nano señalando partes implicadas en la alimentación del nano.]

#### *Componentes que forman el Arduino nano*

**Vista superior:** conector mini USB, más pequeño que el que usa arduino uno. El arduino nano tiene el mismo microcontrolador que el arduino uno (ATmega328P con encapsulado de montaje superficial), por lo que los programas/instrucciones que se desarrollen para uno son válidos/equivalentes para el otro.

[Foto vista superior e inferior del Arduino nano.]

**Vista inferior:** tenemos una tirada de pines macho para conectar a protoboard o a cables directamente. Difiere del arduino uno en este sentido, ya que el arduino uno tenía pines hembra en la parte superior. La idea principal para estos pines macho es conectarla a protoboard en un circuito impreso con un zócalo o soldado de forma directa. Cerca del conector USB tenemos un CI, un conversor USB-TTL, que nos permite conectar la placa al ordenador, obtener de él alimentación y comunicación para cargar programas y monitoreo serie. Finalmente, un regulador de tensión de 5V, permite obtener dicha tensión en un pin de salida para alimentar dispositivos externos cuando es alimentada la placa de forma externa mediante un pin especial llamado Vin.

#### *Justificación de elección para el proyecto*

Se ha elegido esta opción por una serie de razones, que a continuación se expondrán:

- Bajo consumo
- Precio reducido
- Facilidad para la incorporación a una PCB
- Tamaño reducido
- Amplia cobertura por la comunidad
- Open Source

Existen placas Arduino que incorporan conectividad IoT (no sólo LoRa, también WiFi, BLE, Bluetooth o Sigfox), como Arduino MKR FOX 1200, Arduino MKR WAN 1300, Arduino MKR WAN 1310, Arduino MKR NB 1500, Arduino Nano 33 BLE Sense, Arduino Uno WiFi REV2, Arduino MKR1000 WiFi, Arduino Nano 33 BLE, Arduino Nano 33 IoT, Arduino Nano 33 BLE, o Arduino MKR WiFi 1010. Al ser soluciones ya integradas y muy concretas, orientadas más a redes LPWAN con conectividad a internet, no se han considerado como solución para este proyecto, pero se mencionan ya que se han estudiado y pueden resultar útiles para futuras investigaciones o proyectos similares.

## Software: Arduino IDE

Como hemos mencionado anteriormente, las placas Arduino se pueden programar a través del software Arduino IDE (existen más opciones, pero esta es una de las más conocidas), empleando un lenguaje similar al lenguaje de programación C. A continuación, se explicarán los dos bloques principales en los que se divide un *sketch* de Arduino.

### *Bloque setup()*

Se realiza la configuración del dispositivo y la asignación de pines (entradas y salidas analógicas y digitales). Las comunicaciones que necesitemos inicializar lo harán aquí (como la serial), también la programación de mensajes de inicio (por ejemplo, mensaje que aparezca cuando damos alimentación al microcontrolador o lo reiniciemos)

### *Bloque loop()*

En esta sección de código, se definen las tareas que realizará el microcontrolador; siempre que se encienda o reinicie, empezará a ejecutar desde el principio de este bloque. Sería algo así como el programa principal, donde se incluyen todas las sentencias necesarias para obtener la funcionalidad deseada. Al tratarse de un microcontrolador, realizará esta tarea en bucle, y así se marca en el código (estas tareas se enmarcan dentro del bloque *loop*).

### *Comandos ajenos a los bloques setup() y loop()*

No todo el código que se incluye en un *sketch* de Arduino pertenece a uno de los dos bloques recién introducidos. Por ejemplo:

- Definición de variables globales, escribiendo de manera explícita el tipo (`char`, `int...`) y el nombre de la variable (el valor de dicha variable se dará más adelante). Mediante el comando `#define` damos valores a constantes antes de que se compile el programa.
- Incluir librerías, mediante el comando `#include`. Existe infinidad de librerías compatibles con Arduino, muchas de código abierto y de fácil acceso, que pueden servir para diversos proyectos.

- Creación de objetos, algunos propios de las librerías que hemos incluido, como un objeto serial o un servomotor; también puede ser una estructura de datos, dependerá de lo que necesitemos.

### 3.3.2.2 LoRa

#### Qué es

El término LoRa proviene de la unión de las palabras *Long* y *Range*, es decir, largo alcance, evidenciando, así, a través de su nombre una de sus principales características. Se trata de la capa física del protocolo *LoRaWAN*; es a través de esta capa que se permite el establecimiento del enlace de comunicación de largo alcance, mientras que LoRaWAN define el protocolo de comunicación y la arquitectura del sistema para la red.

También se trata de una tecnología inalámbrica propiamente dicha que ofrece una transmisión de largo alcance, bajo consumo y segura para aplicaciones IoT y M2M. Se basa en la modulación de espectro ensanchado, la cual presenta características de bajo consumo como la modulación FSK pero puede ser usada para comunicaciones de largo alcance. Puede ser usada para conectar sensores, gateways, máquinas, dispositivos, animales, personas, etc. de manera inalámbrica a la nube.

#### Planes de frecuencia

La tecnología LoRa opera en diferentes frecuencias dependiendo de la región en la que nos encontramos, por ejemplo:

- En Norteamérica, opera en la banda de 915 MHz.
- En Europa, en la banda de 868 MHz.
- En Asia, en la banda de 865 a 867 MHz y en la banda 920 a 923 MHz, aunque es común también la banda 433 MHz.

Dado que los planes de frecuencia para LoRa varían según la región, inclusive país, es recomendable asegurarse de las regulaciones vigentes en cada país para el uso del espacio radioeléctrico. A modo resumen y como orientación inicial a este asunto, se puede consultar esta [lista de planes de frecuencia para cada país](#), elaborada por *The Things Network*.

#### Origen

La tecnología LoRa fue creada por una compañía francesa llamada *Cycleo*, la cual posteriormente fue adquirida por Semtech en 2012. Semtech fue el miembro fundador de *LoRa Alliance*, que

actualmente es el organismo rector de LoRa Technology. LoRa Alliance es una de las alianzas tecnológicas de más rápido crecimiento. Esta asociación sin ánimo de lucro está formada por más de 500 empresas miembro, comprometidas a permitir la implementación a gran escala de redes IoT de amplia cobertura y baja potencia (LPWAN) a través del desarrollo y la promoción del estándar abierto LoRaWAN.

### Especificaciones en términos generales de la tecnología LoRa

- Órgano rector: LoRa Alliance
- Estándar: 801.15.4g
- Frecuencia: banda ISM, 868/916 MHz
- Alcance: hasta 5 km (urbano) y hasta 15 km (rural)
- Datarate: 27 kbps
- Modulación: espectro ensanchado basado en la tecnología de modulación FM
- Seguridad: CRC de 32 bits

### Regulación LoRa en Europa: ERC 70-03

#### ERC 70-03

### Transceptores LoRa elegidos: E32-868T30D

Los módulos LoRa que vamos a usar son los E32-868T30D. En la Figura 40 se adjunta una imagen del dispositivo en cuestión.

La ficha técnica de este dispositivo se puede consultar en el Anexo I. Destacaremos algunos parámetros en los que se ha basado su elección:

- 

Se usarán antenas de 5 dBi con diagrama de radiación tipo dipolo.

### Simulación en Radio Mobile

Una vez elegidos los componentes (transceptores y antenas) y antes de realizar las pruebas de campo, es procedente simular el enlace radio con los parámetros de los componentes escogidos. Para ello, se empleó el programa de simulación *Radio Mobile*. No simularemos únicamente los

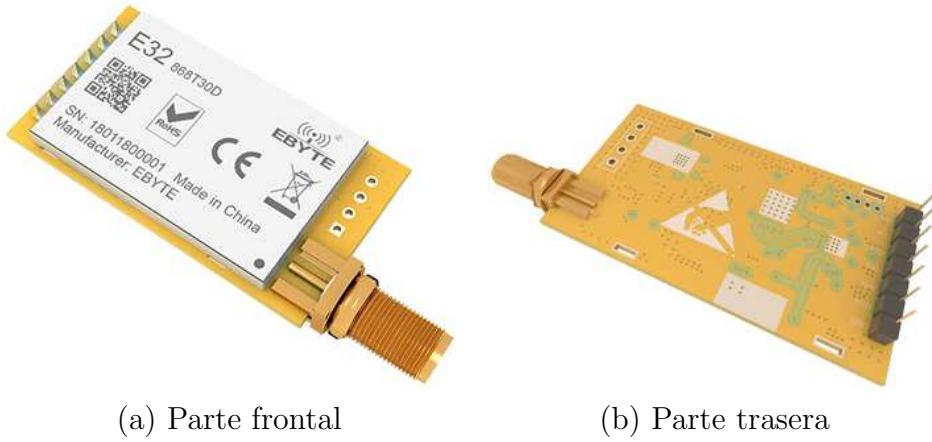


Figura 40: Capturas del transceptor LoRa E32-868T30D [1].



Figura 41: Captura de las antenas usadas [20].

transceptores E32-868T30D, sino que, además, se simulará para el transceptor E22-900T22D, un transceptor que se comentó en el apartado de *Búsqueda de soluciones*, y que además de tener una tecnología muy similar al E32, nos puede proporcionar el RSSI, un parámetro que nos ayuda a valorar la intensidad de la señal en recepción sin necesidad de otras herramientas (las cuales habría que comprar o tomar prestadas, calibrar, etcétera). El estudio en profundidad de este transceptor no entra dentro del alcance programado para este proyecto, sin embargo, se realizan las simulaciones correspondientes para allanar terreno de cara a futuras investigaciones.

Radio Mobile es gratuito y fácil de usar e instalar, por lo que para realizar simulaciones de enlaces no muy complejos es altamente recomendable. En este proyecto, sólo contaremos, en un principio, con dos dispositivos equivalentes en tanto a características radio, separados una distancia de 3.5 kms, aproximadamente, en un entorno con pocas fluctuaciones del terreno, rural y pocos obstáculos entre ambos (existe visión directa, *LoS*, *Line-of-Sight*, entre los extremos). El proceso de configuración seguido para realizar la simulación del escenario planteado es el que sigue:

- En Google Earth, se tomaron las coordenadas de la casa de campo donde se va a encontrar el gateway, y las coordenadas del refugio.

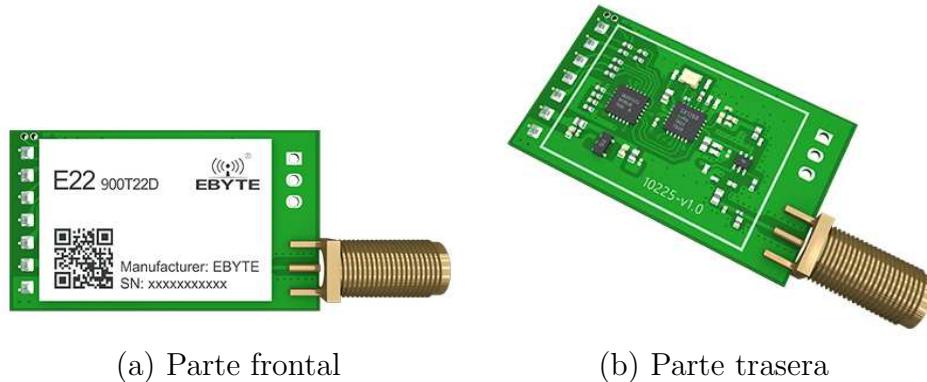


Figura 42: Capturas del transceptor LoRa E22-900T22D [2].

- Ya en Radio Mobile, dentro de *Propiedades del mapa*, introducimos las coordenadas (latitud/ longitud) del campo. Clicamos en *extraer*.
- En *Editar*, seleccionamos *Combinar imágenes* con OpenStreetMaps, y dentro, clicamos en *Multiplicar y mantener imagen*, para poder visualizar el mapa con la capa del terreno sobre él.
- Volvemos a *Propiedades del mapa* y usamos el cursor sobre el mapa para encontrar el punto medio (aproximadamente) para centrar el mapa (para usar coordenadas del cursor, clicamos sobre *Usar posición del cursor*, y vamos centrando y ajustando el alto a valores inferiores para que no nos muestre mucho más mapa del que necesitamos).
- Guardamos el mapa clicando en *Grabar mapa como...* (Dentro de directorio Radio\_Mobile \Mapas, previamente creado) Archivo map (topografía).
- Guardamos la imagen clicando en *Grabar imagen como...* (Dentro de directorio Radio\_Mobile \Imágenes, previamente creado) Archivo imagen, conjunto de mapa e imagen de OpenStreetMap (pero no tiene información topográfica).
- Una vez generados los archivos topográficos e imagen, el siguiente paso es crear la red, clicando en *Nuevas redes* (creación de redes). Primeramente, dejamos los valores por defecto, clicamos en *Usar mapa* y en *ok*.
- Abrimos el mapa ya creado y la imagen ya creada.
- Ahora debemos clicar en *Propiedades de redes: Parámetros*, e introducir los parámetros de nuestra red.

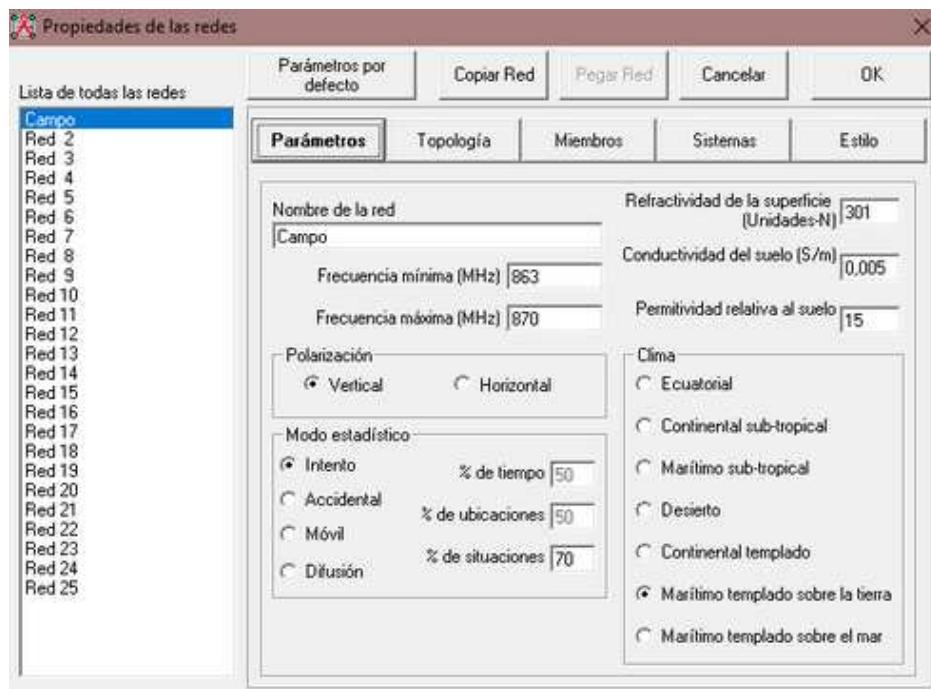


Figura 43: Configuración de los parámetros de la red a simular en Radio Mobile, dentro de *Propiedades de las redes*.

- Se configura la topología de la red según corresponda.

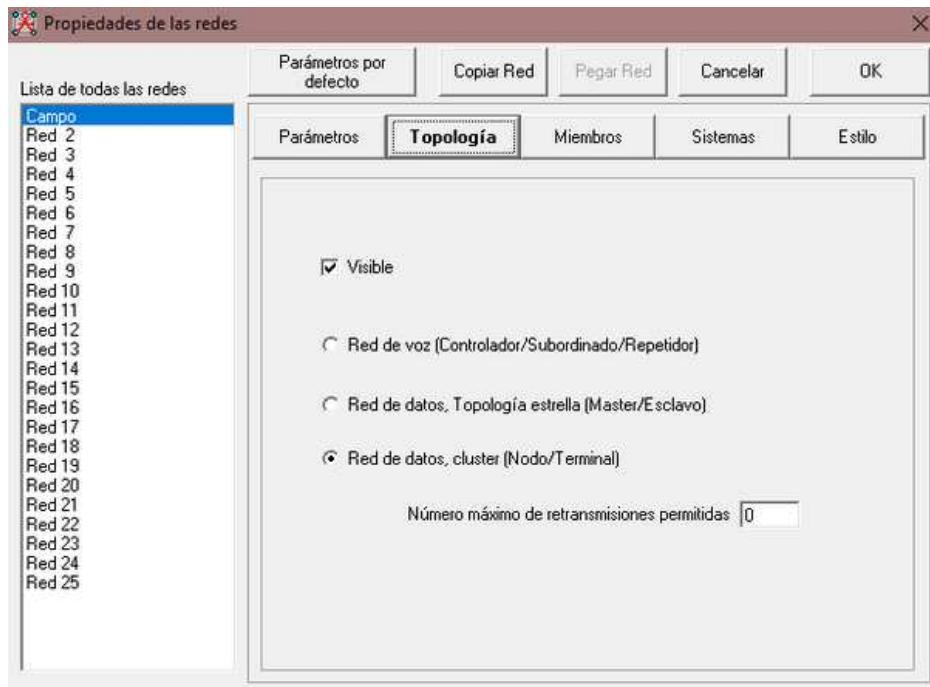


Figura 44: Configuración de la topología de la red a simular en Radio Mobile, dentro de *Propiedades de las redes*.

- El siguiente paso es la creación de los sistemas que conforman esa red, que serán dos, uno el localizado en la casa de campo y otro en el refugio. Habrá más de un sistema campo o refugio, ya que variarán en función de la potencia de transmisión, altura de las antenas o diagrama de radiación, principalmente, parámetros que cambiamos dependiendo de la simulación. Podemos cambiar, también la ganancia de las antenas o las pérdidas, pero como son parámetros que, en nuestro caso, no van a cambiar, no serán variables a considerar. Cuando creamos un sistema, debemos clicar en *Agregar a Radiosys.dat*.

Para el caso de la potencia de transmisión 14dBm (se consigue con E22):

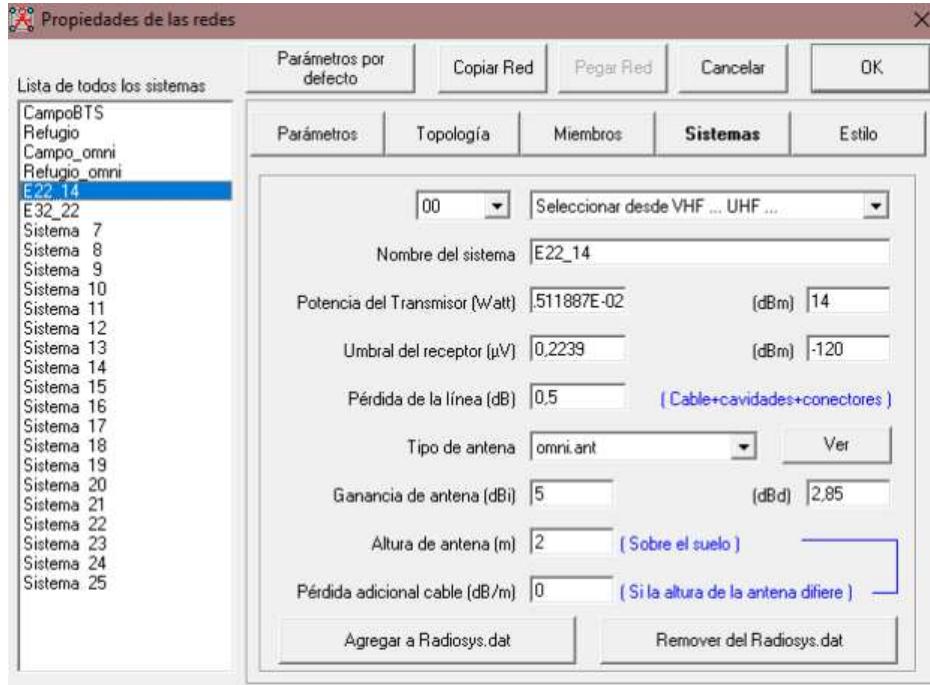


Figura 45: Configuración de los sistemas de la red a simular en Radio Mobile, dentro de *Propiedades de las redes*. Ejemplo para transceptor LoRa E22, transmitiendo a 14 dBm.

- Añadimos los sistemas como unidades clicando en *Propiedades de las unidades* (para añadir marcadores de posición de los sistemas creados). Como sabemos las coordenadas, clicamos en *Ingresar LAT LON o QRA*, y las añadimos ahí.
- Una vez creados los sistemas y asociados a una unidad, hay que asociarles un rol dentro de la red, en *Miembros*, dentro de *Propiedades de las redes*. *Miembros* y *Sistemas* depende de la simulación. Se ha creado una sola red, pero varios sistemas dependiendo de la potencia de transmisión. Como hay que asociar los sistemas a los miembros, miembro se convierte en un valor dinámico según el sistema que haya que simular; creando dos miembros y asignando ese rol al sistema que se deba simular en ese momento, es suficiente. Además, siendo ambos extremos de la comunicación equivalentes, sería suficiente con crear un sistema y asignándole los dos miembros creados posteriormente, en nuestro caso. Dentro de *Miembros*, cambiando Sistema (E32\_22 o Campo\_omni/Refugio\_omni, que sería el caso del E32 a 30 dBm, por ejemplo) se pueden configurar los equipos para las potencias de transmisión probadas.

Siguiendo con el mismo ejemplo que el presentado en *Sistemas* (E22 a 14 dBm):

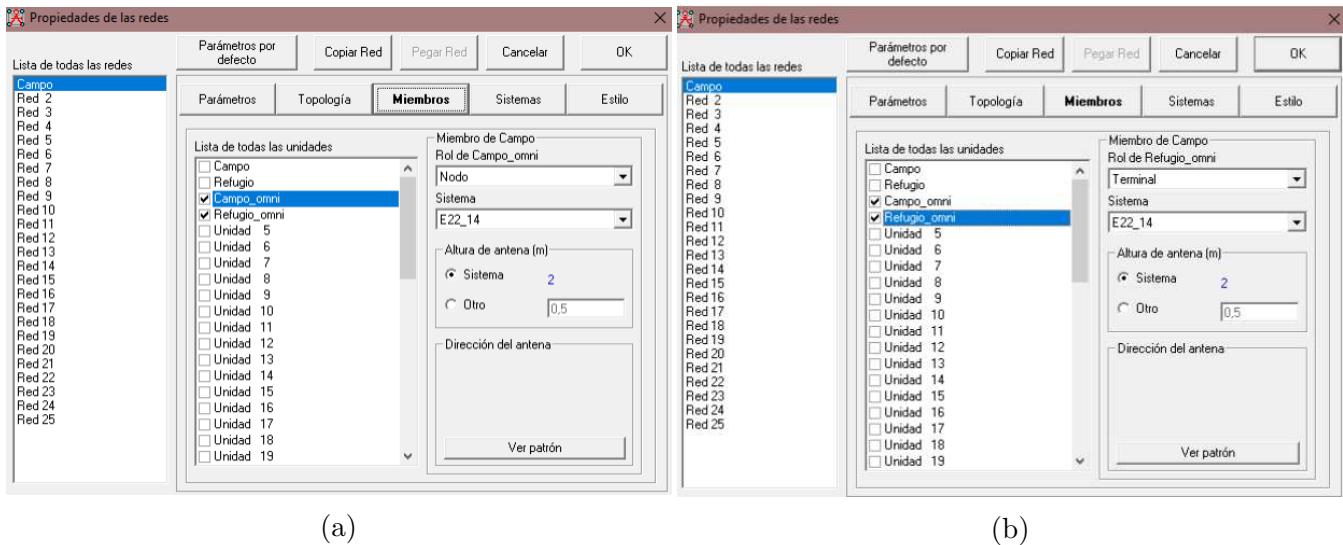


Figura 46: Configuración de los miembros de la red a simular en Radio Mobile, dentro de *Propiedades de las redes*. Ejemplo para transceptor LoRa E22, transmitiendo a 14 dBm.

- Si todo ha sido configurado correctamente, clicando en *Ver - Mostrar redes - Todo*, nos enseñará en el mapa el enlace radio, con las unidades asociadas a sistemas.



Figura 47: Enlace radio creado en Radio Mobile para simular la red creada entre la casa de campo y el refugio.

- Por último, clicamos en *Herramientas - Enlace radio*, para realizar el estudio del enlace, y ver si se crea de manera satisfactoria.

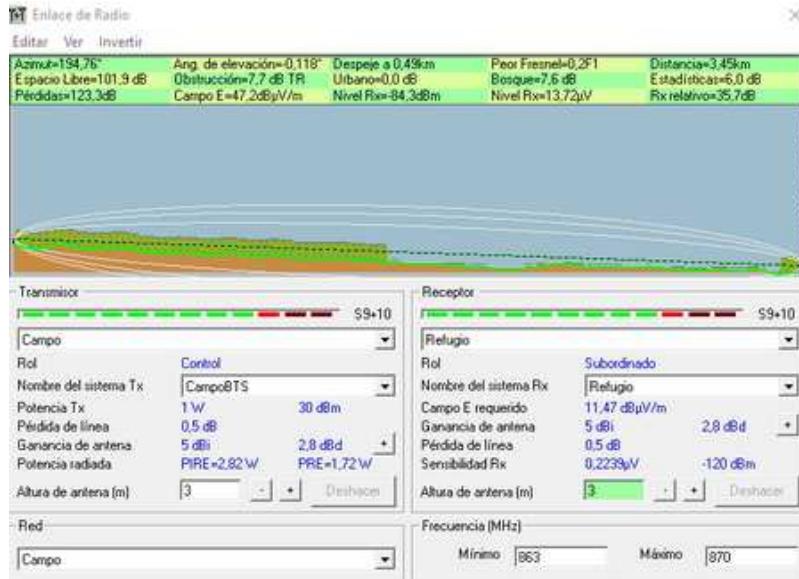


Figura 48: Resultado de la simulación del enlace radio creado en Radio Mobile.

Explicado el procedimiento a seguir para simular un determinado radioenlace, se muestran las simulaciones realizadas para comprobar si se establece dicho radioenlace de manera satisfactoria. Para ello, se han realizado simulaciones variando la potencia de transmisión a los valores máximo y mínimo posibles (según el transceptor), ya que son los valores críticos. La altura de las antenas no será modificada si a la altura de 2 m los resultados son satisfactorios, ya que es la altura mínima a la que se encontrarán las antenas.

## Simulación con E32-868T30D

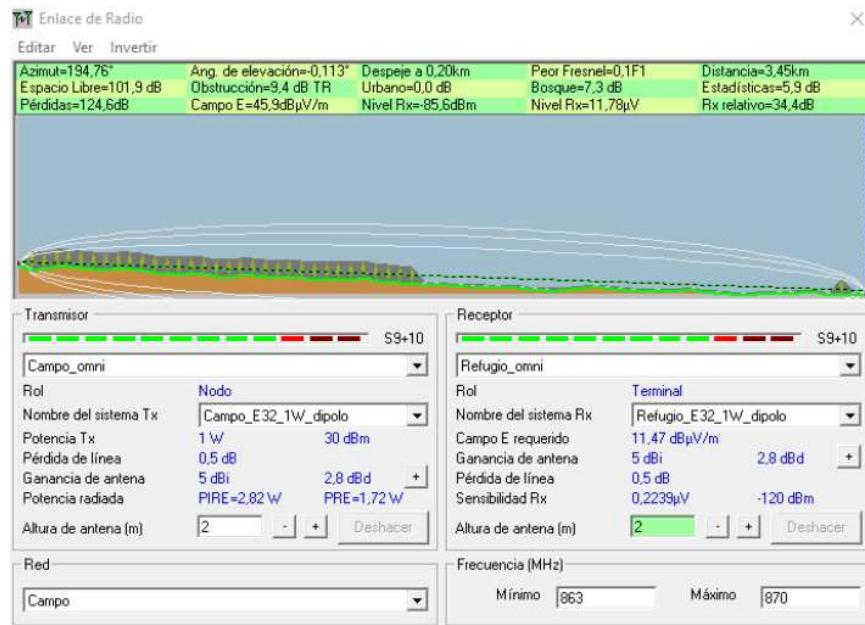


Figura 49: Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E32 transmitiendo a 30dBm (1W). Enlace descendente: campo - refugio.



Figura 50: Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E32 transmitiendo a 30dBm (1W). Enlace ascendente: refugio - campo.

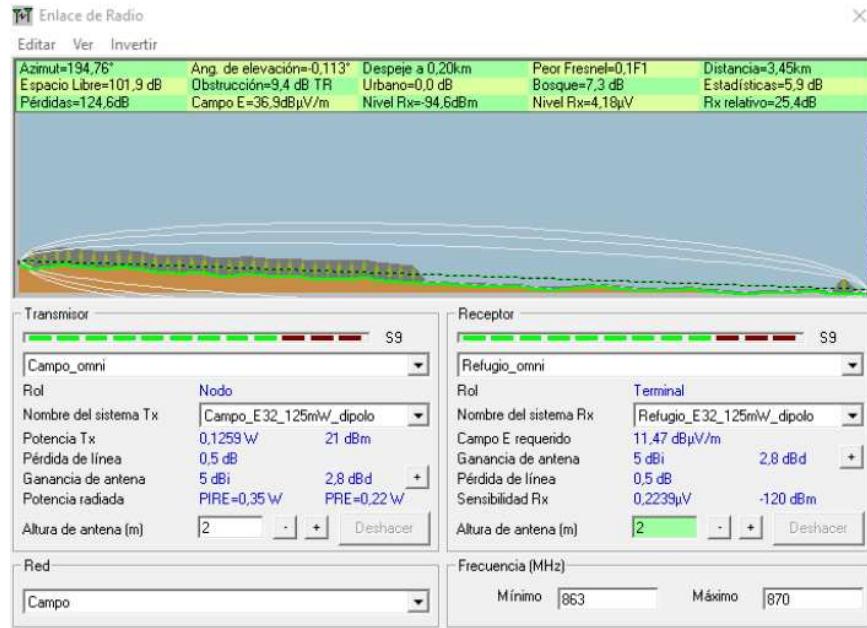


Figura 51: Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E32 transmitiendo a 21dBm (125 mW). Enlace descendente: campo - refugio.

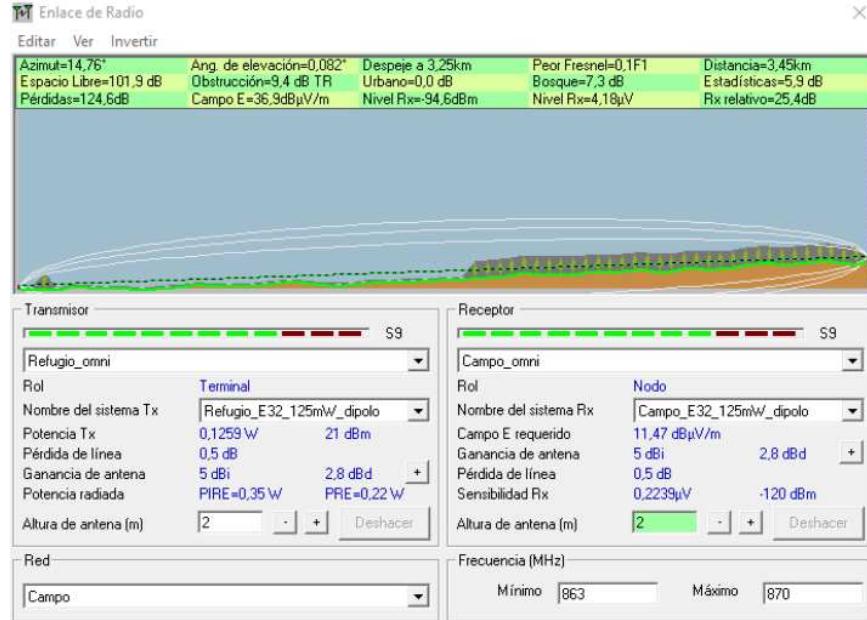


Figura 52: Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E32 transmitiendo a 21dBm (125 mW). Enlace ascendente: refugio - campo.

## Simulación con E22-900T22D

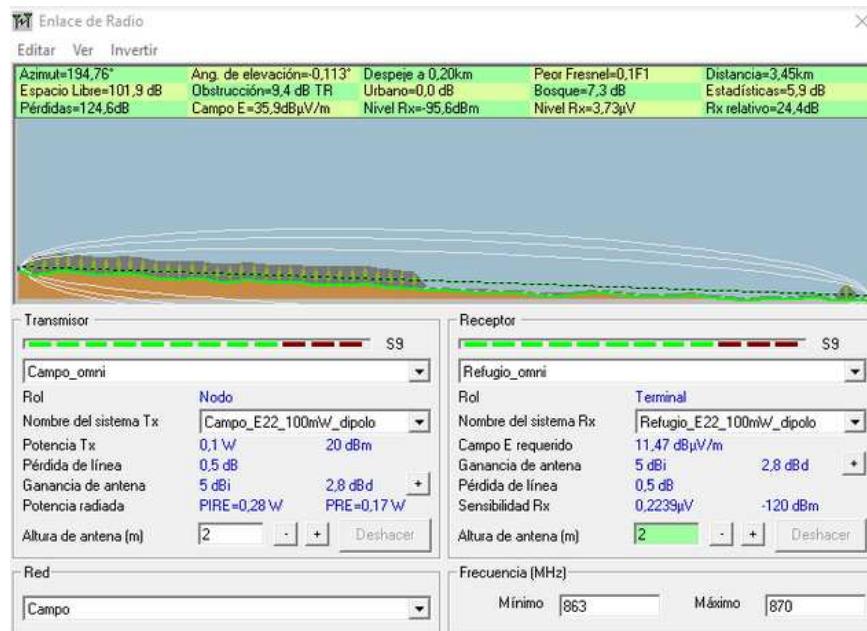


Figura 53: Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E22 transmitiendo a 20dBm (100 mW). Enlace descendente.

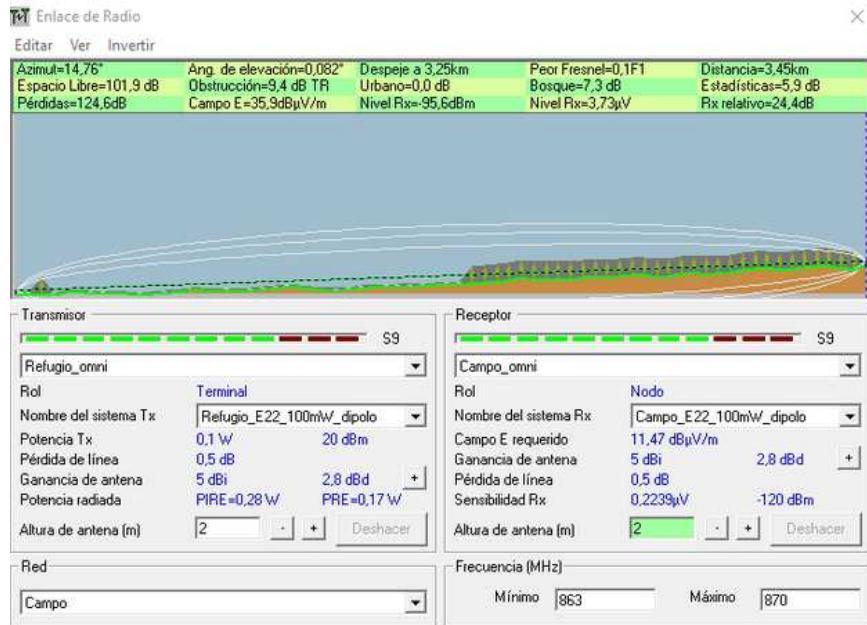


Figura 54: Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E22 transmitiendo a 20dBm (100 mW). Enlace ascendente.

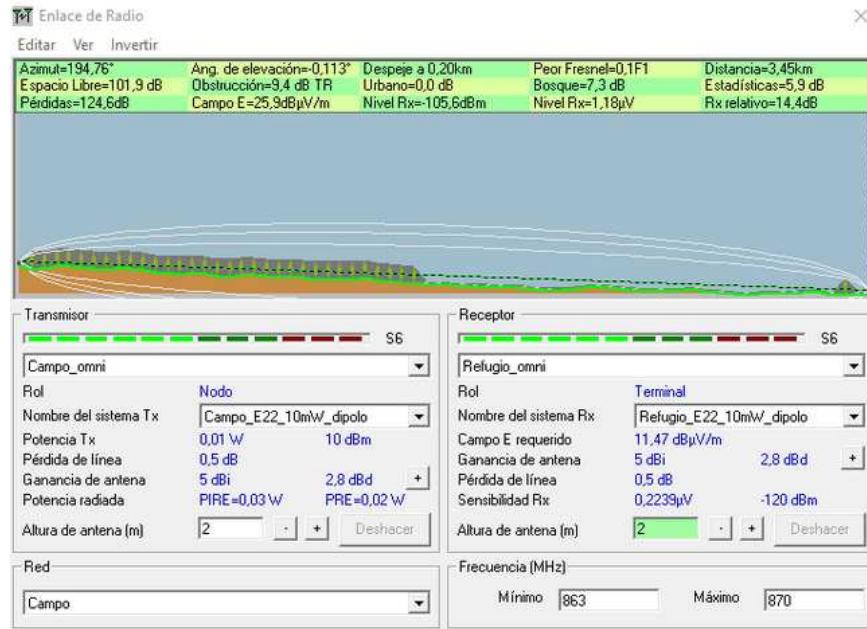


Figura 55: Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E22 transmitiendo a 10dBm (10 mW). Enlace descendente.

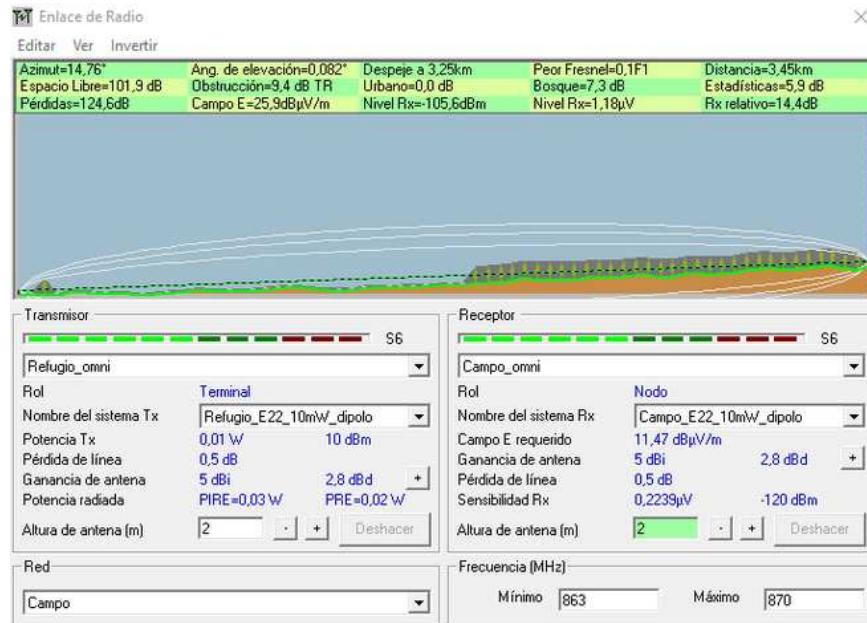


Figura 56: Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E22 transmitiendo a 10dBm (10 mW). Enlace ascendente.

A continuación, se mostrarán las simulaciones realizadas teniendo en cuenta las pérdidas adicionales por el alargador SMA de 1.8 m. Según el datasheet sobre RG316, el tipo de cable que se ha usado como alargador, dicho cable introduce unas pérdidas a 900 MHz de 26.3 dB/100ft. Si el cable que se ha usado es de longitud 3FT (hay otro interno en la caja pero al ser de 15 cms se ha despreciado), se introducen unas pérdidas de 0.7dB, por lo que redondeando al caso peor, introduce 1dB de pérdidas. Se ha introducido este dB de pérdidas en *Propiedades de las redes - Sistemas - Pérdida de la línea*. Se desprecian las pérdidas en los conectores SMA.

### **Simulación con E32-868T30D**

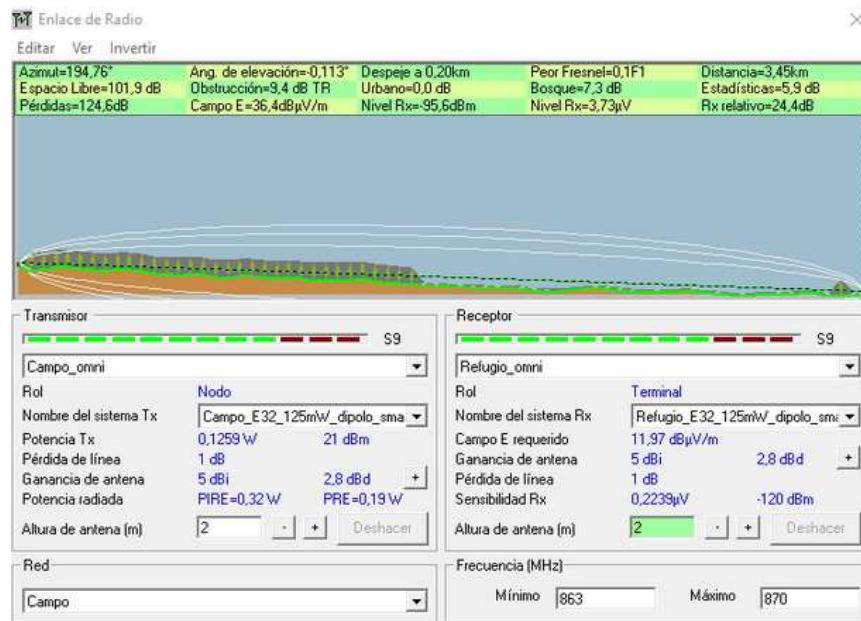


Figura 57: Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E32 transmitiendo a 21dBm (enlace descendente), con pérdidas de 1 dB debido al alargador SMA.

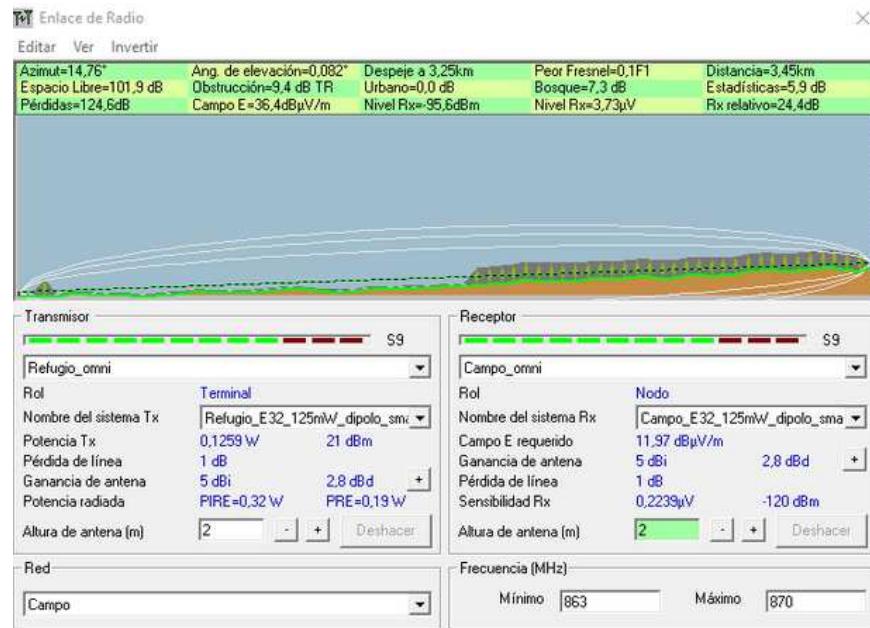


Figura 58: Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E32 transmitiendo a 21dBm (enlace ascendente), con pérdidas de 1 dB debido al alargador SMA.

## Simulación con E22-900T22D

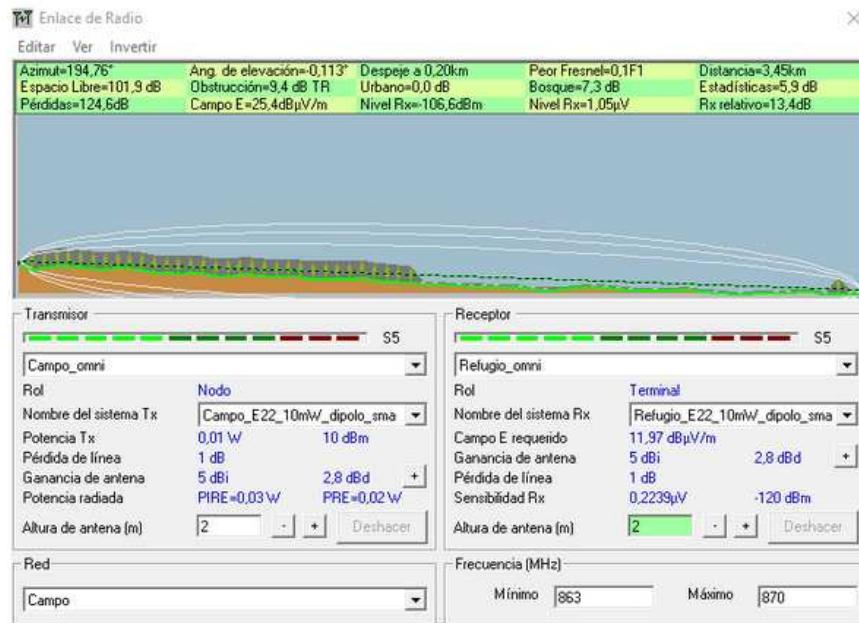


Figura 59: Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E22 transmitiendo a 10dBm (enlace descendente), con pérdidas de 1 dB debido al alargador SMA.

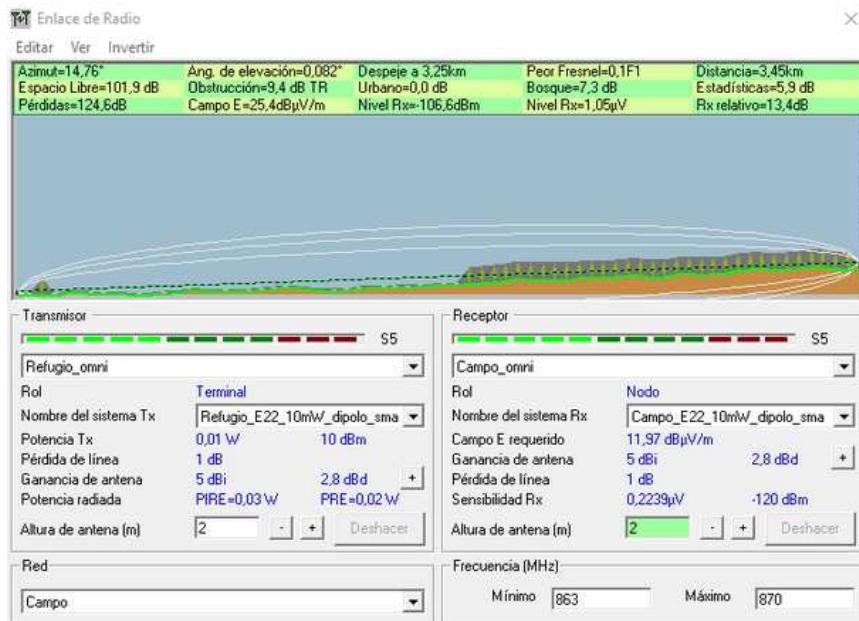


Figura 60: Resultado de la simulación del enlace radio creado en Radio Mobile, para transceptor E22 transmitiendo a 10dBm (enlace ascendente), con pérdidas de 1 dB debido al alargador SMA.

Resumiendo, para una altura de las antenas de 2m, tanto en el caso del transceptor LoRa E32-868T30D como en el caso del transceptor E22-900T22D, conseguimos establecer un enlace LoRa satisfactorio. En todos los casos se observa que el nivel de la señal en el receptor está en verde, y por tanto, se recibe la potencia necesaria. Es más, en todo punto de la línea entre transmisor y receptor hay cobertura, ya que ésta aparece dibujada en color verde.

### 3.3.2.3 Comedero

Se ha elegido como gestor del movimiento del pienso el uso de un motor, en concreto, de un servomotor de 5V, capaz de girar 360º y capaz de mover cargas de hasta 10 kg (para leer información más detallada, ir a Anexo I).



Figura 61: Imagen del servomotor elegido para conformar el prototipo inicial de nuestro sistema de alimentación para animales [9]

Este servo, con la ayuda de unas hélices imprimidas en 3D, moverá el pienso en dirección longitudinal a una pieza T de PVC donde irán insertados. En la Figura 61 se muestra un kit bastante común de un servomotor MG996R, con varias hélices para transferir el movimiento rotatorio del servo; sin embargo, para el planteamiento que se pretende llevar a cabo se requieren unas hélices especiales. Se buscan unas hélices que ayuden al pienso en su recorrido longitudinal a través de la pieza T de PVC. Este planteamiento está inspirado en la creación de [kitlaan](#), subido a la plataforma *Thingiverse*, bajo el nombre [Auger-based Cat Feeder](#).



(a) Partes impresas en 3D (desmontado)



(b) Partes impresas en 3D (montado)

Figura 62: Capturas de la solución del movimiento de pienso del usuario kitlaan en *Thingiverse* [34].

Además, se hará uso de una serie de piezas de PVC para crear la conexión comedero-reserva, desde la salida de pienso en el fondo de la reserva hasta la salida del pienso hacia el comedero. Como mínimo, será necesaria una pieza de PVC tipo T.



Figura 63: Pieza PVC tipo T 87°[11]

Otras piezas PVC que podrán ser empleadas son codos de 45°, prolongaciones de tubo PVC o conectores de unión, como los que se muestran en la Figura 64.



(a) Pieza PVC tipo codo 45°[12]



(b) Tubo PVC para prolongaciones [13]



(c) Pasamuros de PVC [14]

Figura 64: Ejemplos de piezas PVC que pueden ser empleadas en el prototipo final.

Por último, las conexiones con el microcontrolador se realizarán a través de los cables que accionan el servo: uno para señal, otro para alimentación, y otro para tierra, tal como se muestra en la Figura 65. El esquema del conexionado entre servo, Arduino y alimentación en protoboard se muestra en la Figura 66.

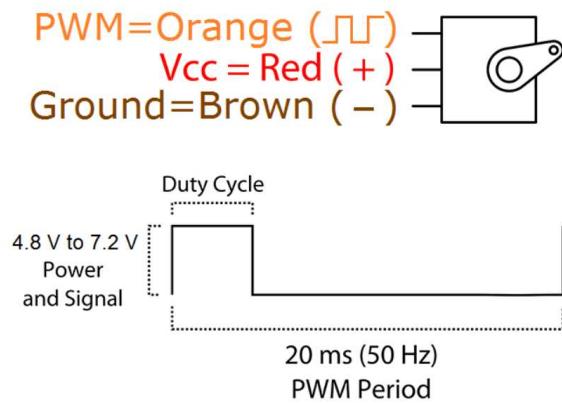
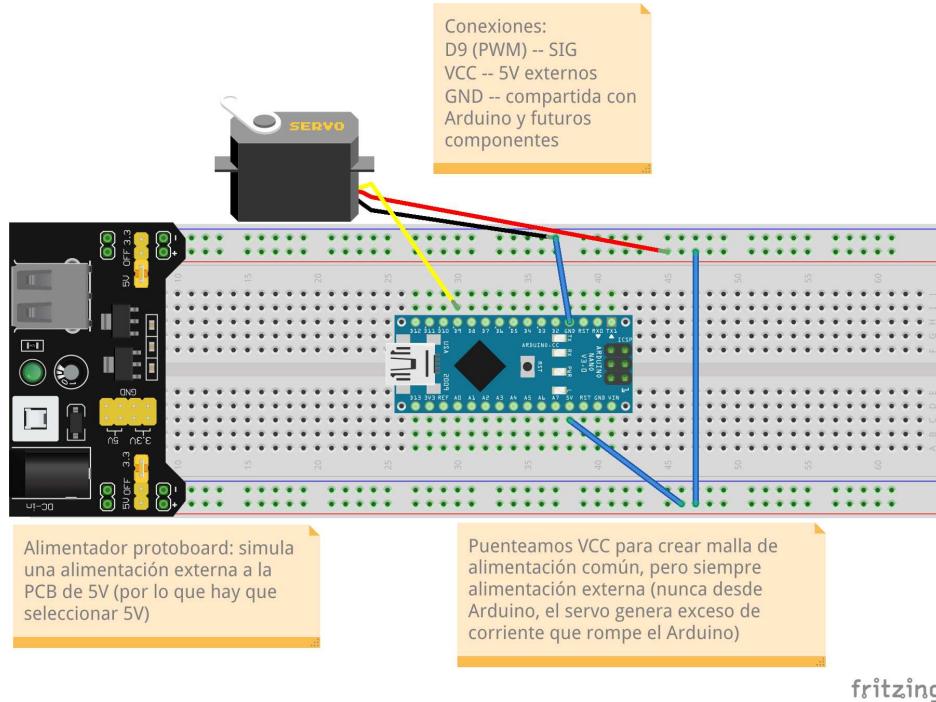


Figura 65: Esquema simplificado de los cables que usa un servomotor MG996R y de su *duty cycle* [10]



fritzing

Figura 66: Esquema de las conexiones servo-Arduino realizado en *Fritzing*.

### 3.3.2.4 Bebedero

Para cubrir esta funcionalidad, se ha elegido la combinación de opciones formada por sensores de nivel de flotación, relé y bomba. El funcionamiento de esta combinación se resume de la siguiente manera:

- Se colocarán dos sensores de nivel en posiciones concretas de la reserva de agua para detectar, conforme se va haciendo uso de la reserva, qué cantidad de agua queda en ella.
- El movimiento del agua desde la reserva al bebedero tendrá lugar gracias a una bomba de agua y un tubo de silicona transparente, por donde circulará el agua propulsada por la bomba.
- Por último, se hará uso de un relé entre arduino y bomba de agua, ya que ésta no tiene una lógica extra que permita su control.

En la Figura 67 mostramos la configuración en protoboard de todos estos componentes.

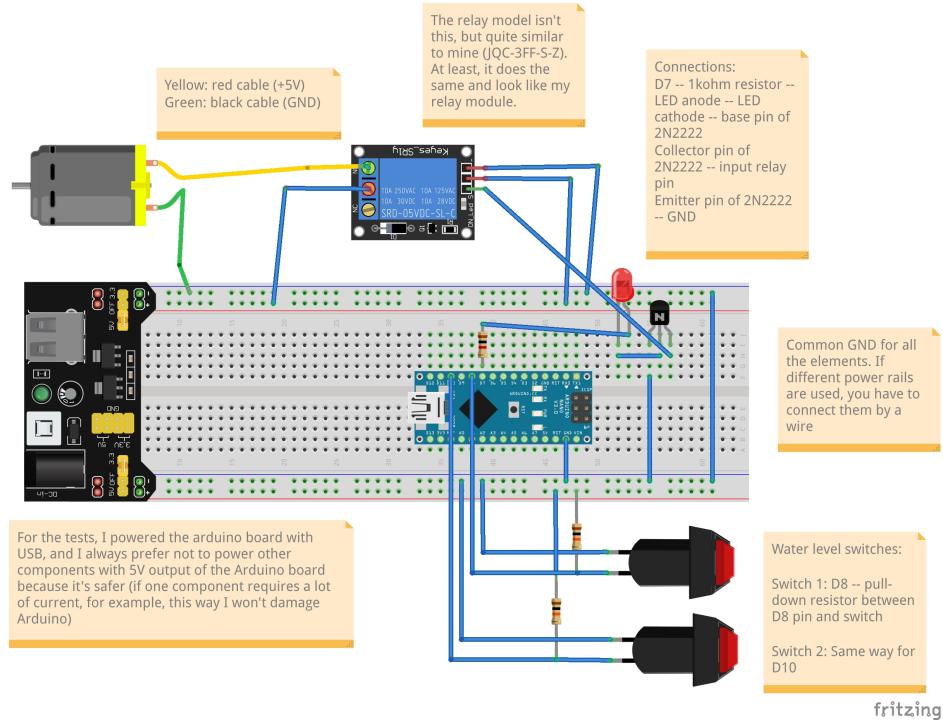


Figura 67: Esquema de las conexiones de sensores y actuadores usados para el bebedero con Arduino realizado en *Fritzing*.

A continuación, se tratarán los principales componentes de forma individual, recordando su funcionamiento.

### Sensores de nivel

Ya explicados con más detalle en el apartado 3.2.2.4. Se situarán en uno de los laterales de la reserva de agua, separados entre sí una distancia suficiente como para monitorizar de manera correcta los diferentes estados en los que puede encontrarse la reserva. Irán conectados al microcontrolador a través de los pines digitales D8 y D10, tal como aparece en la Figura 67. Como se comentó en el apartado 3.2.2.4, necesitaremos hacer uso de una resistencia *pull-down* para evitar estados lógicos no deseados a la entrada de los pines D8 y D10; en este caso, las resistencias empleadas tienen un valor de 10k, y realizarán un puente entre alimentación y el cable del sensor de nivel que vaya al pin digital del arduino.

En tanto a apariencia, se usará el sensor de nivel que ya se mostró en la Figura 23, dentro del apartado 3.2.2.4.

## Bomba de agua

La bomba se comporta como un motor DC, y tiene dos cables únicamente (+5V, GND), por lo que, para poder controlar su funcionamiento, necesitaremos usar un actuador que haga de intermediario entre el microcontrolador y la bomba. Elegimos un relé, el cual está conectado al pin D7 del arduino, según podemos ver en la Figura 67 (explicamos el relé más adelante).



Figura 68: Captura de la bomba de agua usada en el prototipo [16]

Esta bomba, mientras tenga alimentación, estará propulsando agua, la cual entra, en este caso, por un lateral de la bomba, y sale por el orificio superior (mirar Figura 68). A este orificio hay que conectarle un tubo de silicona y guiarlo hasta donde queramos que termine llegando el agua.

## Relé

El relé actúa como un conmutador, que responde a la señal de control de su entrada, actuando en consecuencia según su valor. Al contrario de lo que ocurría en el caso de un sensor, al ser un actuador, la señal de control vendrá del microcontrolador y determinará la actuación del relé. Más adelante veremos cómo, en el apartado 4.2.1; resumiendo, el microcontrolador le dice al relé, mediante esa señal de control, cuándo cambiar de la posición NA a NC (o NC a NA), para que, así, la corriente fluya o no hacia la bomba y ésta se active, o se desactive.



Figura 69: Captura del relé usado en el prototipo [17]

Volviendo a la Figura 67, entre el pin D7 y el relé, vemos que existe una serie de componentes intermedios (un LED, una resistencia de un  $1k\Omega$  y un BJT). Se usan debido a que el relé necesita

ver a su entrada una corriente superior a la que sale del microcontrolador, y el transistor BJT amplifica dicha corriente para su correcto funcionamiento. La resistencia sirve para estabilizar la corriente a la entrada del LED, el cual usamos para ver visualmente cuándo cambia de estado el relé (aunque cuando lo hace, emite un sonido característico).

Como anotación, decir que la configuración elegida para el BJT está basada en el circuito que aparece en la Figura 26; se modificó a un BJT en configuración emisor común (en la Figura 26 se emplea una configuración emisor común, pero situando el relé en la patilla emisor y no colector, y allí no se produce una amplificación), ya que permite tener una ganancia en corriente, lo cual nos interesa; es la configuración más usada para amplificar señales y no sólo eso, sino que es la que produce mayor ganancia de corriente y ganancia en potencia de las tres configuraciones BJT (emisor común, colector común, base común). Hay que tener en cuenta que esta configuración es inversora (desplazamiento de  $180^\circ$ ) y presenta una ganancia de tensión menor [59], de cara a usarla para otras funcionalidades.

### 3.3.3 Prototipo exterior

En este apartado se esbozarán las opciones que se han planteado para el prototipo exterior. Recordemos lo que ya se comentó sobre esta funcionalidad en la subsección 3.1; en este punto, se pretende unificar las demás funcionalidades, teniendo presente que se desea algo ligero, intuitivo y de bajo coste.

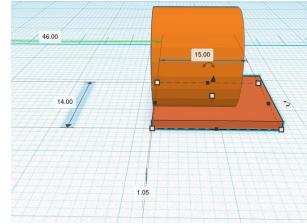
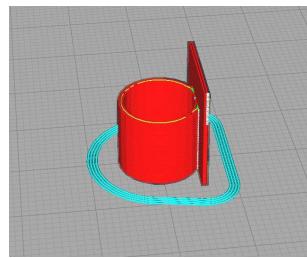
*A qué intentaremos encontrar solución en este punto*

- Conexionado entre reserva de pienso y comedero
- Conexionado entre reserva de agua y bebedero
- Elección de reservas
- Elección de comederos
- Protección de la electrónica
- Localización de las antenas

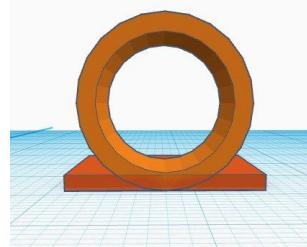
En el apartado 3.3.2.3 ya se explica lo que pueda ser necesario para el conexionado reserva-comedero (piezas de PVC principalmente). Para la funcionalidad de bebedero, en concreto para conectar la reserva de agua con el bebedero, se hará uso de un tubo de silicona transparente (Figura 70, 10mm de diámetro, grosor de 1mm y longitud 1m) y de una pieza impresa en 3D para sujetar el tubo de silicona al bebedero (Figura 71).



Figura 70: Captura de un fragmento de tubo de silicona (ejemplo) [18]



(a) Pieza impresa en 3D (desde arriba) (b) Pieza impresa en 3D (lateral)



(c) Pieza impresa en 3D (frontal)

Figura 71: Capturas del diseño e impresión en 3D de la pieza creada para la sujeción del tubo de silicona al bebedero.

En tanto a las reservas que serán empleadas, se trata de dos bidones de plástico (uno para pienso y otro para agua), de capacidad 5L, suficiente para una prueba de concepto (Figura 72).



Figura 72: Foto del bidón usado en el prototipo.

Los recipientes usados como comedero y bebedero son unos cuencos de acero inoxidable (Figura 73), pensado para mascotas pequeñas, con capacidad aproximada de 200mL. Igual que con el bidón, para una prueba de concepto será suficiente. Trae consigo unos ganchos, también de acero inoxidable, de uso opcional, para colgar los cuencos en una jaula (o similar).



Figura 73: Capturas del cuenco y ganchos de sujeción usados para comedero y bebedero [35].

Para proteger la electrónica, se emplearán cajas estancas (Figura 74), ya que están preparadas para proteger conexiones eléctricas (cables u otros), incluso en el exterior. Además, se abren desatornillando unos tornillos en su parte frontal, y es posible hacer agujeros a medida para pasar cables desde su interior hasta el exterior en diferentes zonas de silicona plástica que tienen en todos sus laterales.



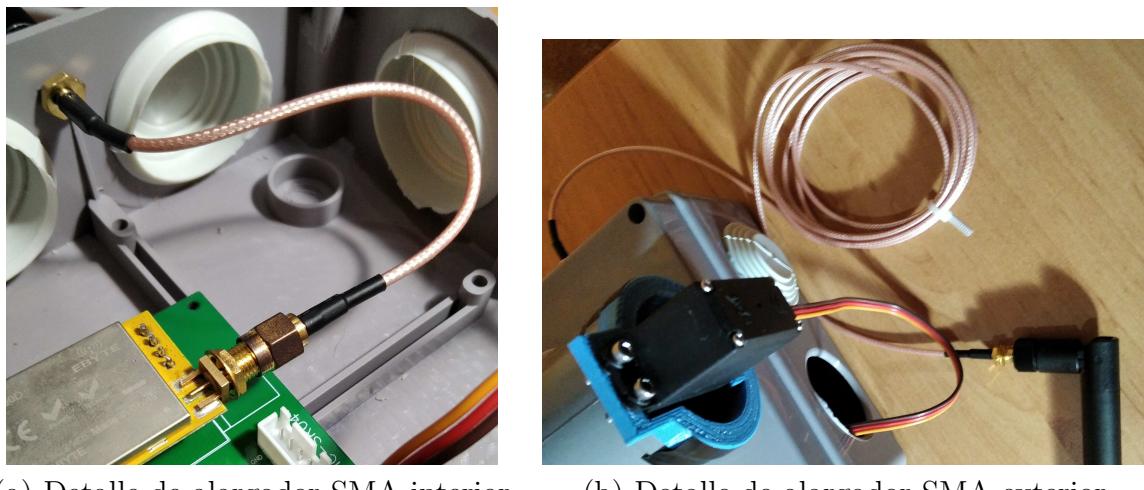
(a) Exterior de una caja estanca    (b) Detalle de interior de una caja estanca



(c) Detalle de ubicación de los tornillos y de zonas perforables para cables.

Figura 74: Capturas de una caja estanca (ejemplo) [19].

Por último, faltaría determinar la ubicación de las antenas. Para mejorar la emisión/recepción de los mensajes, debemos dar una altura mínima a las antenas. Para ello, se propone conectar un alargador SMA desde el transceptor LoRa (el cual está en el interior de la caja estanca) hasta una de las paredes de la caja estanca (Figura 75(a)), la cual habrá que perforar con un taladro para sacar la salida de ese alargador al exterior. De esta manera, podremos conectar otro alargador (de la longitud que se desee) desde el exterior de la caja estanca hasta la altura que dé dicho alargador (Figura 75(b)), y conectar en tal punto la antena (permitiendo, así, mejorar la conexión entre ambos nodos).



(a) Detalle de alargador SMA interior

(b) Detalle de alargador SMA exterior

Figura 75: Fotos de alargadores SMA usados.

### 3.4 Resumen de capítulo

En este capítulo se ha hablado primero, de las funcionalidades que hay que cubrir con este proyecto para poder cumplir con los objetivos marcados. Una vez determinadas las funcionalidades, se realiza y se muestra un estudio acerca de posibles soluciones mediante las cuales podamos llevar a la práctica la funcionalidad en concreto que se esté analizando. Por último, se muestra la selección final de tecnologías (hardware y software) para cada funcionalidad. En resumen, se sacan conclusiones para la creación de un prototipo que pueda ser ejecutado y probado en el entorno objetivo.

## 4 Prototipo y pruebas

### 4.1 Ubicación

La idea fundamental de este proyecto era el diseño del sistema y su posterior verificación mediante la construcción del prototipo y su instalación en una de las jaulas del refugio de Patitas Unidas Los Alcázares. Por tanto, es momento de especificar el emplazamiento del prototipo, de forma que quede más claro el lugar concreto donde se ha instalado.

El refugio se encuentra en el término municipal de Torre Pacheco, entre las pedanías de Santa Rosalía y Los Meroños. Se trata de un entorno rural, rodeado de carreteras medianas y pequeñas, campos de cultivo, zonas ganaderas y algunas casas dispersas. Como se indicó en la introducción del presente documento, se trata de una finca sin electricidad y sin internet. El conocimiento del entorno donde vamos a probar el funcionamiento de nuestro proyecto es fundamental, ya que nos permite definir la totalidad del mismo, al tener que amoldarlo a las condiciones del entorno; además, se desea extrapolar a entornos similares.



Figura 76: Foto del refugio de Patitas Unidas Los Alcázares.



Figura 77: Foto de las jaulas existentes en refugio de Patitas Unidas.



Figura 78: Detalle del exterior de una de las jaulas existentes en refugio de Patitas Unidas.

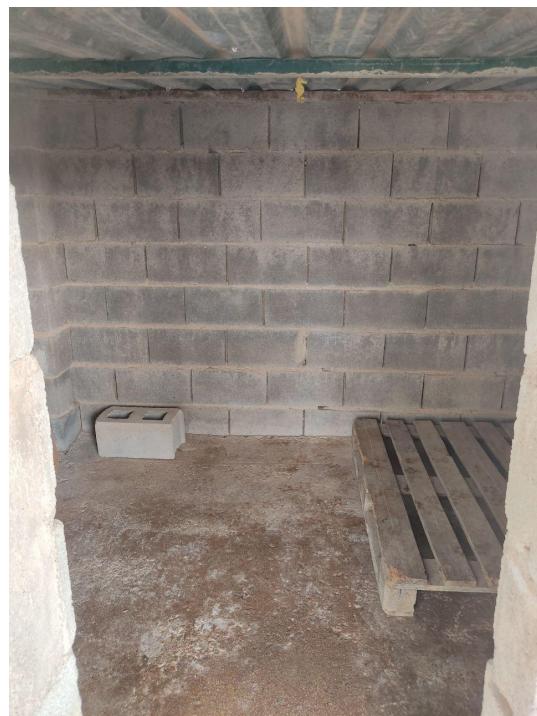


Figura 79: Detalle del interior de una de las jaulas existentes en refugio de Patitas Unidas.

Se adjuntan imágenes donde se contextualiza la manera actual de alimentar y abreviar a los animales alojados en el refugio.



(a) Foto del bebedero estándar



(b) Bidón estándar para llenar los bebederos

Figura 80: Sistema existente en el refugio de Patitas Unidas para abrevar a los animales.



(a) Foto del comedero estándar



(b) Foto medida estándar para llenar el comedero

Figura 81: Sistema existente en el refugio de Patitas Unidas para alimentar a los animales.

Como podemos observar, tanto en la Figura 80 como en la Figura 81, el proceso de abrevar y alimentar a los animales requiere de la presencia física de voluntarios en el refugio para que se lleve a cabo. Para llenar los bebederos, se emplea una reserva común, desde la que se coge agua en cubos y desde esos cubos se llenan los bebederos de piedra (Figura 80(a)), situados dentro de las jaulas (se puede ver en la Figura 78). En tanto al proceso de llenar el comedero, se muestra

en la Figura 81(b) la medida que se usa para rellenarlo, en concreto, se suele usar dos medidas de este bote, que supondría un total aproximado de 150g de pienso. El pienso se almacena también en una reserva común dentro de un almacén, y se recarga el comedero a mano con las medidas recién expuestas.

## 4.2 Prototipo definitivo

### 4.2.1 Resumen del funcionamiento del prototipo definitivo

En este punto se mostrará el funcionamiento del prototipo final del sistema de alimentación para animales creado. Se usarán esquemas de elaboración propia, realizados en *draw.io*, con el objetivo de mostrar las conexiones principales entre los bloques fundamentales del sistema creado, de manera sencilla e intuitiva. Se acompañará de una explicación resumen sobre cómo se ha realizado la automatización y monitorización.

#### 4.2.1.1 Comedero

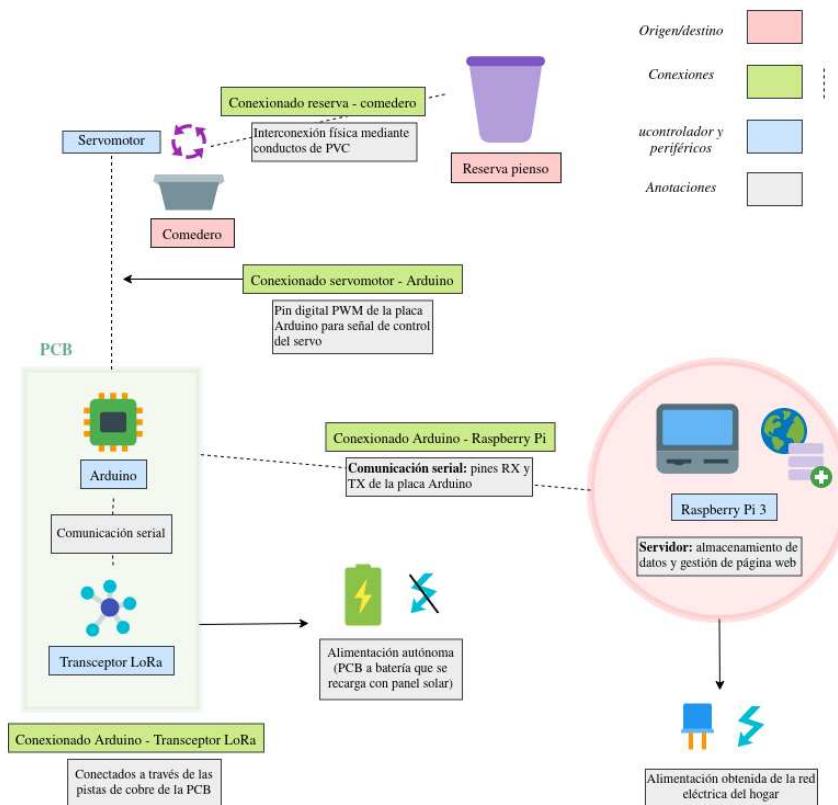


Figura 82: Esquema resumen de la funcionalidad de comedero en el prototipo definitivo.

Nótese en la Figura 82 que se omite la presencia de la segunda PCB que recibe los datos de monitorización del pienso.

A modo resumen, se emitirán unos comentarios previos a la presentación del prototipo, acerca del hardware y software empleados para el comedero. Se hará lo mismo para el bebedero. Así pues:

- Comentarios sobre el hardware.

- **Comentarios sobre el software.**

- Es necesario calibrar en software el arduino que monitorizará el pienso; esto se consigue calculando el número de recargas que ponemos realizar dada la capacidad de la reserva y del comedero. En nuestro caso, tenemos una reserva de 5 kg y un comedero de 200g, por lo que, haciendo cálculos, tenemos para 25 ciclos de relleno. Por seguridad y por desnivel de la reserva (la pieza por donde cae el pienso está a un nivel superior del tope inferior de la reserva, por lo que esos dos centímetros de desnivel no permitirían al pienso restante caer por él), reduciremos esos ciclos al 20 %, por lo que realmente consideraremos que serán 20 ciclos de relleno. En el sketch de Arduino que se adjunta en el Anexo I se muestran las variables que permiten ajustar este cálculo y permiten la calibración si se cambiara la reserva o el comedero.
- Es necesario calibrar en software el arduino que monitorizará el pienso; esto se consigue calculando el número de recargas que ponemos realizar dada la capacidad de la reserva y del comedero. En nuestro caso, tenemos una reserva de 5 kg y un comedero de 200g, por lo que, haciendo cálculos, tenemos para 25 ciclos de relleno. Por seguridad y por desnivel de la reserva (la pieza por donde cae el pienso está a un nivel superior del tope inferior de la reserva, por lo que esos dos centímetros de desnivel no permitirían al pienso restante caer por él), reduciremos esos ciclos al 20 %, por lo que realmente consideraremos que serán 20 ciclos de relleno. En el sketch de Arduino que se adjunta en el Anexo I se muestran las variables que permiten ajustar este cálculo y permiten la calibración si se cambiara la reserva o el comedero. Tendremos un contador que permitirá saber en qué ciclo de relleno estamos, si quedan recargas o necesitamos llenar la reserva. Esta información se transmitirá desde el transceptor LoRa del refugio hasta el transceptor LoRa en el otro extremo, y será gestionado por la Raspberry Pi al que está conectada la PCB de este extremo.
- Ajustar el tiempo de un ciclo de relleno para desactivar el movimiento del servomotor y que se rellene el comedero la cantidad deseada.

#### 4.2.1.2 Bebedero

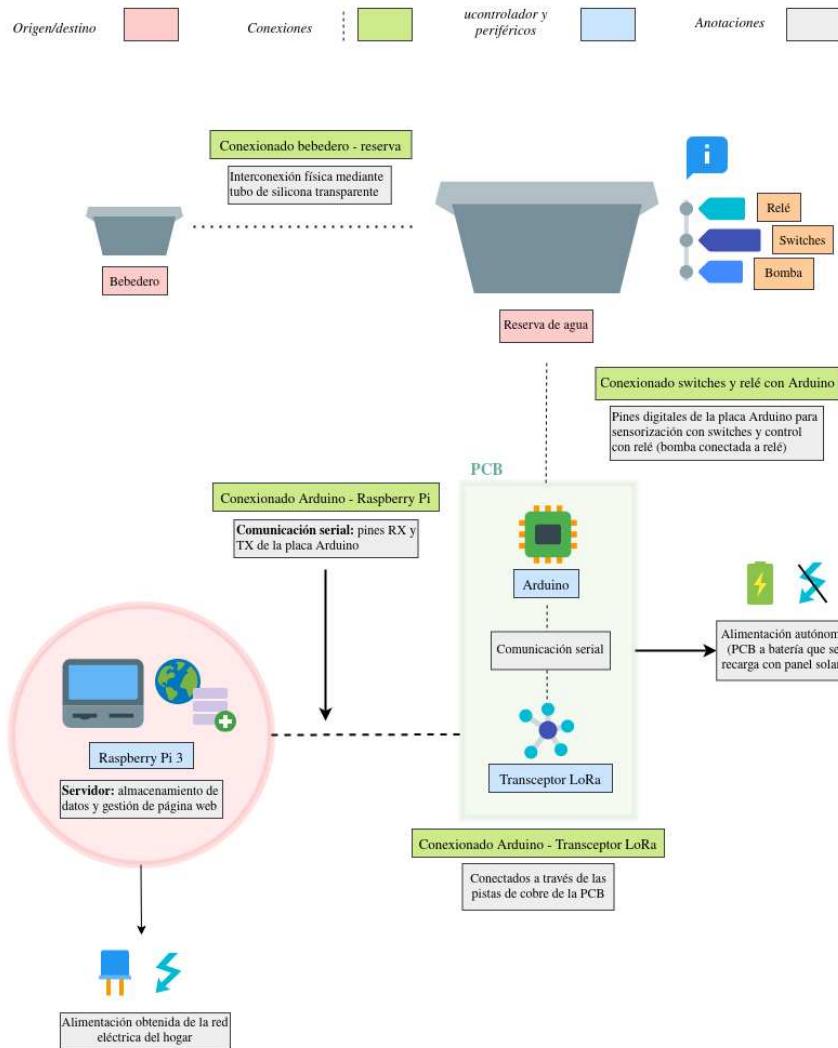


Figura 83: Esquema resumen de la funcionalidad de bebedero en el prototipo definitivo.

Nótese en la Figura 83 que se omite la presencia de la segunda PCB que recibe los niveles de sensorización de agua: se miden niveles de agua con los sensores, y controlamos la acción de la bomba con el relé, conectados al arduino, el cual envía datos mediante transceptor LoRa al transceptor LoRa que tenemos en el otro extremo, el cual no necesitará alimentación autónoma al estar en una casa, y es este el que irá conectado a la Raspberry (que tiene función de servidor y gestor de los datos en recepción).

#### 4.2.2 PCB

Como hemos visto en la sección 3, subsección 3.3, son muchos componentes los que forman el cómputo total del sistema de alimentación para animales. Para facilitar el prototipado y simplificar las conexiones de todos los elementos, se creó una PCB, como la que se muestra a continuación:

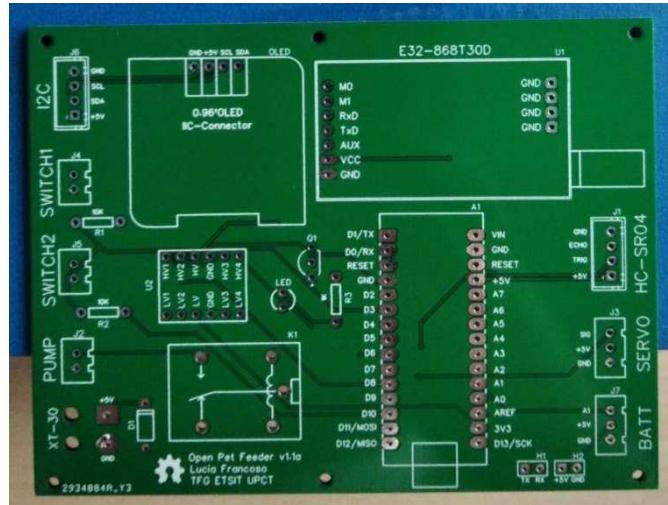


Figura 84: Foto de la PCB usada en el prototipo final.

El fabricante de la PCB es JLCPCB, y se usó su editor EasyEDA. Para mayor detalle del esquemático o de la PCB, consultar el repositorio de GitHub creado (poner con href), donde se incluye un archivo .json que, importándolo a EasyEDA, permitirá su visualización.

#### 4.2.3 Protocolo

Tan importante es simplificar el sistema en un único bloque hardware, como lo es unificarlo en software. Es por ello que se creó un sketch de Arduino llamado *refugio\_protocolo*, donde incorporamos la traducción de mensajes recibidos para el microcontrolador del refugio, de manera que éste sepa cómo actuar en consecuencia. También creamos un sketch para el microcontrolador del extremo al que llamamos gateway, el cual conecta con nuestro servidor, alojado en una Raspberry Pi 3; este sketch es el *gw\_protocolo*, el cual explicaremos más adelante (al igual que *refugio\_protocolo*). En el Anexo II se adjunta el código Arduino creado para tal fin, de manera detallada; a continuación, se describirá a grandes rasgos en qué consiste el protocolo.

Antes de explicar el código creado, cabe decir que no habría sido posible el desarrollo de este protocolo sin el sustento proporcionado por la librería **EBYTE** [49], a través de la cual:

- Se crea una conexión serial por software entre arduino y transceptor LoRa.

- Se configuran los parámetros radio del transceptor. Esta librería soporta varios transceptores.
- Se crea una estructura de datos para formar un mensaje que pueda ser enviado por radio.
- Se envían los datos vía radio.
- Se reciben datos vía radio.

El uso de esta librería nos permite realizar todo lo anterior, y modificar según necesitemos, además de incorporarlo de manera intuitiva a código de elaboración propia, como se ha hecho en los sketches *gw\_protocolo* y *refugio\_protocolo*.

*Anotación:* de aquí en adelante, se hablará de estación del refugio (conjunto formado por microcontrolador, transceptor LoRa y sensores y actuadores para las funcionalidades de comedero y bebedero) y estación del gateway (microcontrolador y transceptor LoRa, con conexión serial a Raspberry Pi).

### ***gw\_protocolo***

Corresponde al código empleado para programar el arduino nano de la estación del gateway, de manera que:

- Pueda comunicarse a través de la comunicación serial ordenador - arduino
- Puedan comunicarse arduino y transceptor LoRa mediante una comunicación serial (librería EBYTE).
- Se crea una estructura de datos para dar forma a los mensajes que se van a recibir y a enviar.
- Bloque `setup()`: se inicializan las comunicaciones seriales entre pc - arduino y entre arduino - LoRa. Se configuran los parámetros del transceptor según nuestros requerimientos, haciendo uso de las funciones de la librería (funciones internas del transceptor). Para mejor entendimiento con el usuario, mostramos una lista de comandos aptos, es decir, comandos programados en el protocolo que al recibirse en el otro extremo generan una respuesta (ya que ésta está programada).
- Bloque `loop()`:
  - *Si hay conexión serial entre pc y arduino:* este código sirve para enviar datos hacia el transceptor para que sean enviados vía radio. Así pues, el arduino leerá lo que le llegue por serial y lo almacenará en una variable, la cual se muestra por consola para debugging. Posteriormente, envío ese dato vía radio mediante una función llamada `sendMessage (String data)`, la cual toma como argumento el dato, y tras una serie de transformaciones, lo envía invocando funciones internas del transceptor LoRa.

- *Si hay conexión serial entre arduino - LoRa*: esta fracción de código sirve para procesar los datos entrantes al transceptor. Se invocan funciones internas del transceptor LoRa, y se comprueba cada cierto tiempo (regulable en el propio código) si ha llegado un nuevo mensaje; si no, lo hace saber por consola.

Configuración GW: 4.2.5. De momento, está programado y funcionando el código recién comentado. El envío de mensajes desde gw hasta refugio están programados de manera que estos sean escritos por consola serial; queda que puedan enviarse mensajes desde una web piloto (la consola serial es más para el testeo en fase de desarrollo, y la web piloto es la primera interfaz que vendría siendo parte de la fase de producción). Además, de momento está programado que cuando recibe mensajes sólo los ”decodifique”, pero no hace nada con ellos, salvo mostrarlos por consola; la idea es que lo suba a la Raspberry Pi, como mínimo, y si todo va bien, que también pueda mostrarse en la web piloto.

### ***refugio\_protocolo***

Corresponde al código empleado para programar el arduino nano de la estación del refugio, de manera que:

- Se crea una estructura de datos que define la forma de los mensajes que se van a recibir y enviar.
- Se crea la comunicación serial entre arduino y LoRa. Posteriormente, se inicializará dicha comunicación, al igual que la serial entre pc y arduino.
- Bloque `setup()`: se inicializarán los sensores y actuadores y los parámetros del transceptor.
- Bloque `loop()`: *si hay conexión serial entre arduino - LoRa*, comprueba si hay mensajes entrantes. Si lo hay, lo decodifica mediante funciones internas del transceptor (librería EBYTE), lo muestra en la consola del serial del pc, y ejecuta la llamada a la función `executeCommand (String data)`. Esta función es la que permite realizar una equivalencia entre datos recibidos y respuesta a ejecutar, de manera que, en función de un determinado mensaje recibido, el microcontrolador sepa actuar en consecuencia. La lista de comandos programados, y por tanto, válidos, se mostrará a continuación. Cabe señalar que los mensajes respuesta a estos comandos tienen la siguiente estructura: `ack;field1;var1`.
  - `status`: si se recibe la orden `status`, se realizará una comprobación de los niveles de la reserva de agua (llamando a la función `checkWaterLevel()`) y la reserva de pienso. Si la reserva de agua está por encima del 25% y quedan más de 5 recargas de pienso, se enviará como mensaje respuesta a `status un msg;status;ok`. Si por el contrario, el nivel de la reserva de agua es igual o está por debajo del 25% o quedan 5 o menos recargas de pienso, entonces se enviará `msg;status;ko`.
  - `ping`
  - `reset_comedero`

- rellenar\_comedero
  - rellenar\_bebedero
  - check\_levels
  - update\_oled
- También se hace uso de la función `sendMessage`, introducida anteriormente, para enviar en este caso los datos requeridos por los comandos recién expuestos (o bien datos requeridos por actualizaciones automáticas, que por el momento no están implementadas).

#### 4.2.4 Imágenes del prototipo final

##### Comedero

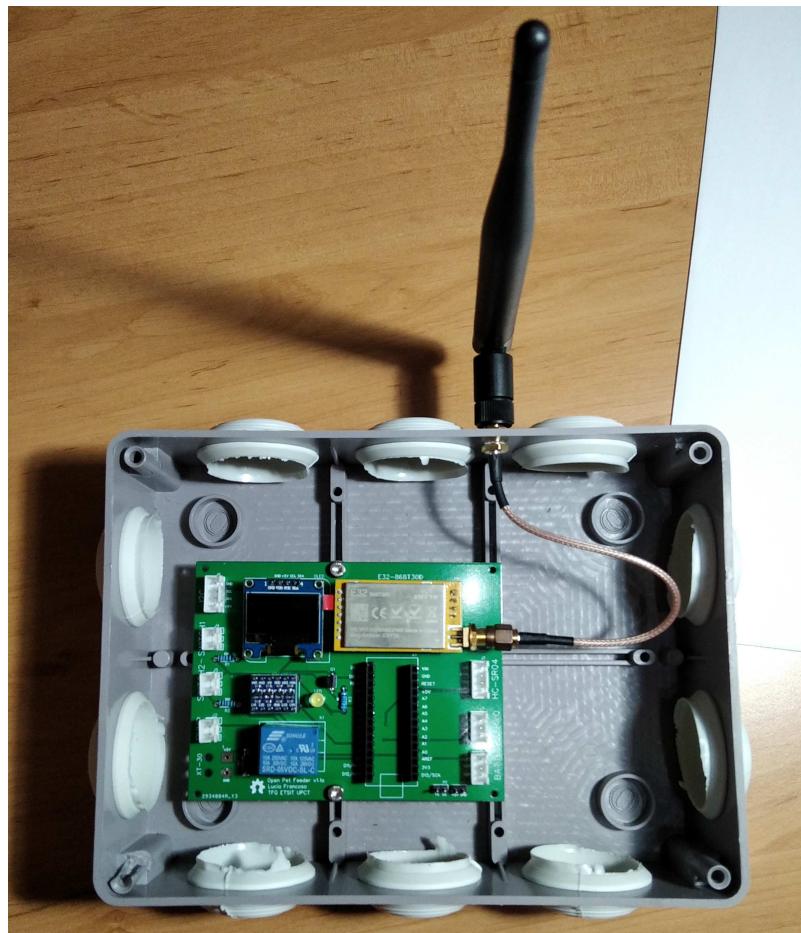


Figura 85: Imagen del interior de la caja, con PCB y componentes.

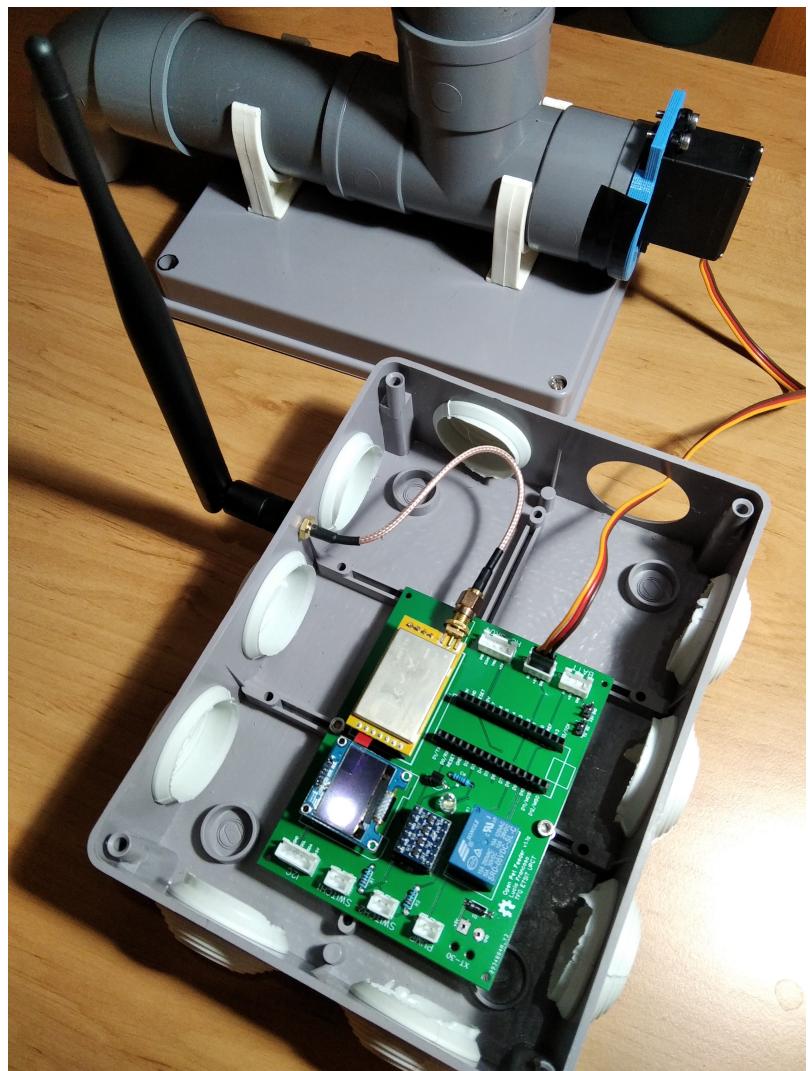


Figura 86: Imagen del interior de la caja con servomotor conectado.

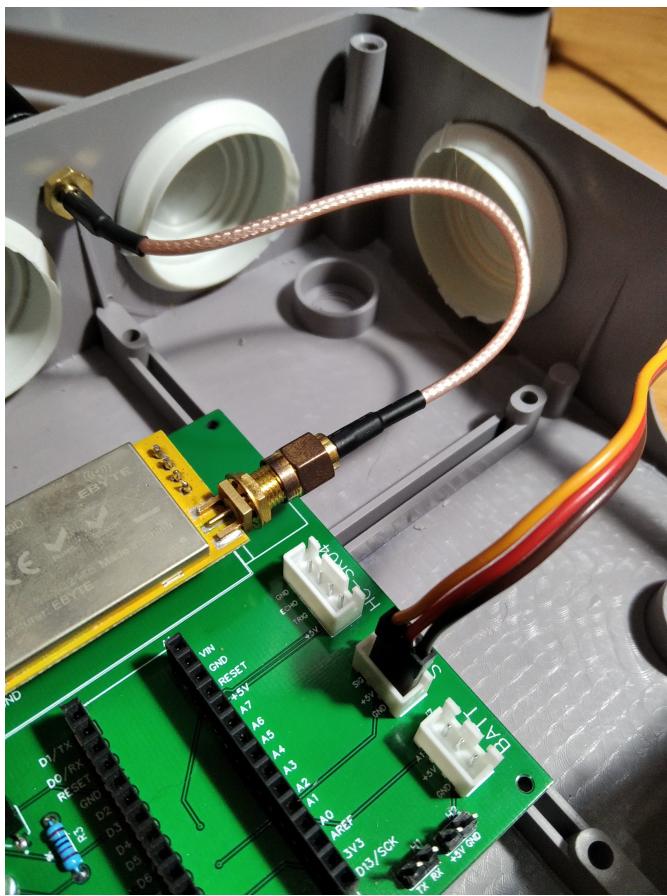


Figura 87: Imagen del interior de la caja con servomotor conectado (zoom).



Figura 88: Imagen del exterior de la caja (detalle de las abrazaderas).

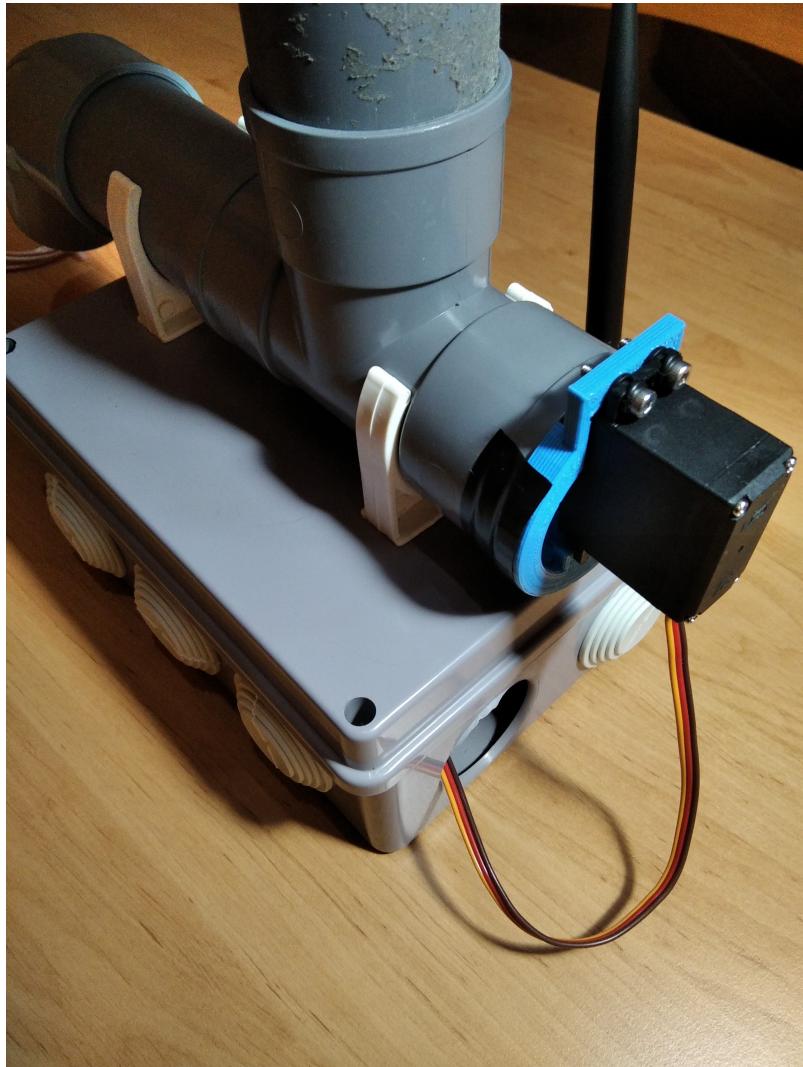


Figura 89: Imagen del exterior de la caja con servomotor conectado. Se muestra, además, parte de los conductos para mover el pienso, y ubicación del servomotor.

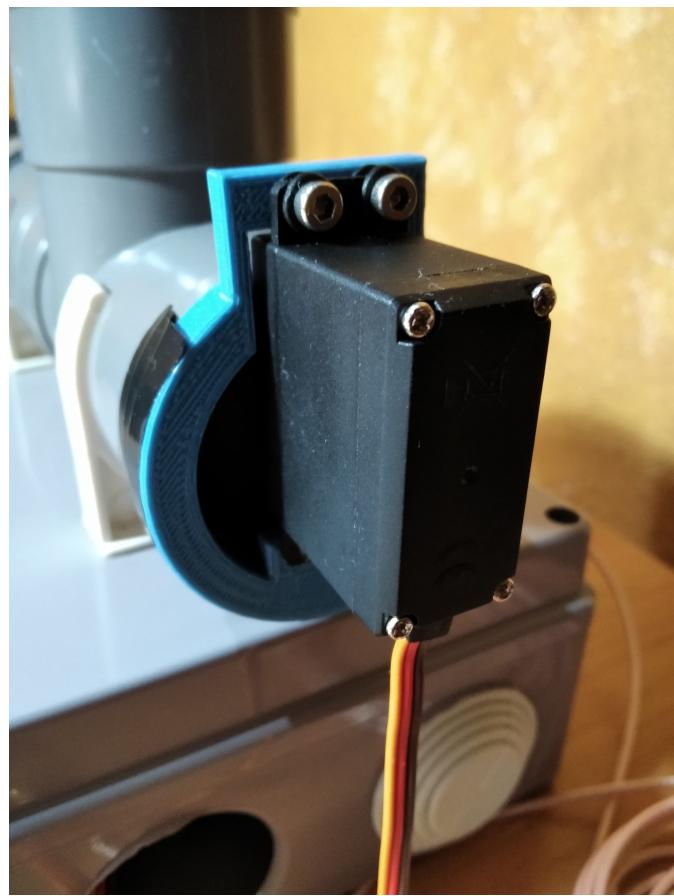


Figura 90: Imagen del exterior de la caja con servomotor conectado. Zoom de sujeción a pieza PVC.



Figura 91: Detalle de la hélice v3 dentro de la pieza PVC tipo T.



Figura 92: Aproximación de cómo quedaría la hélice v3 dentro del brazo PVC.



Figura 93: Imagen global del comedero.



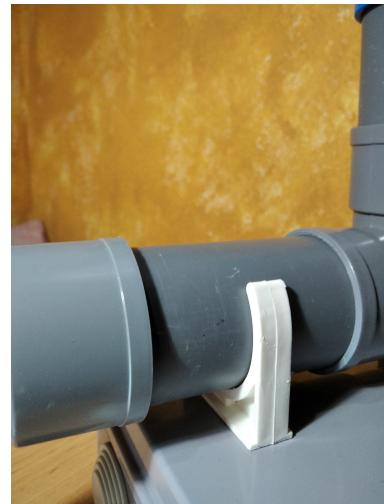
(a) Pieza 1 PVC



(b) Pieza 2 PVC



(c) Pieza 3 PVC

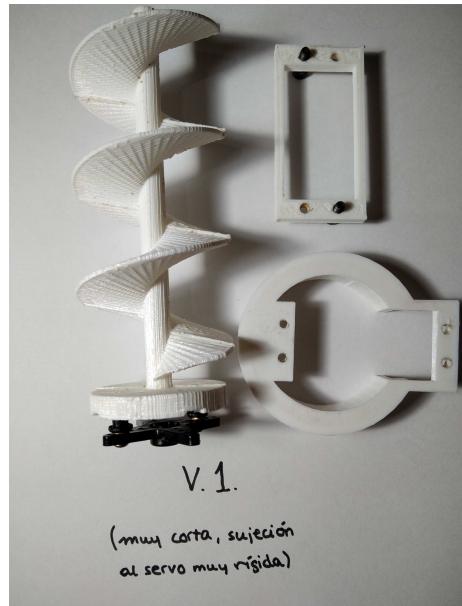


(d) Pieza 4 PVC

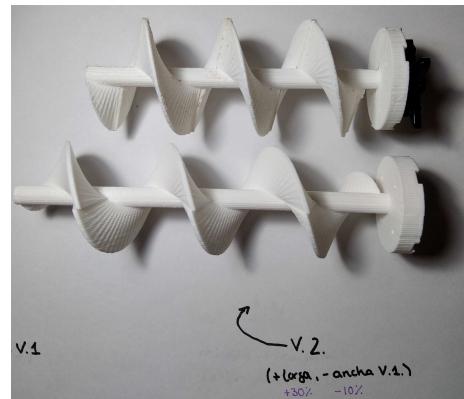


(e) Pieza 5 PVC

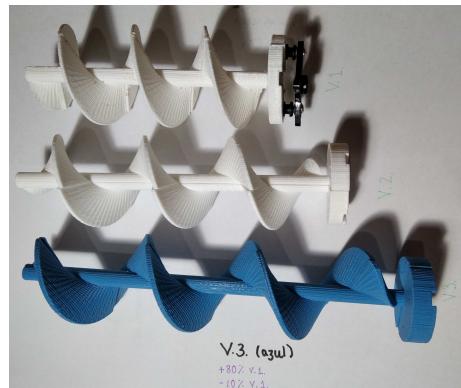
Figura 94: Zoom de las piezas que conforman el sistema de conexionado entre reserva y comedero.



(a) Versión 1 de las hélices



(b) Versión 2 de las hélices (comparada con v1)



(c) Versión 3 de las hélices (comparado con v1 y v2)

Figura 95: Detalle de las hélices impresas en 3D para, a través del movimiento del servomotor, hacer llegar el pienso al comedero.

## Bebedero

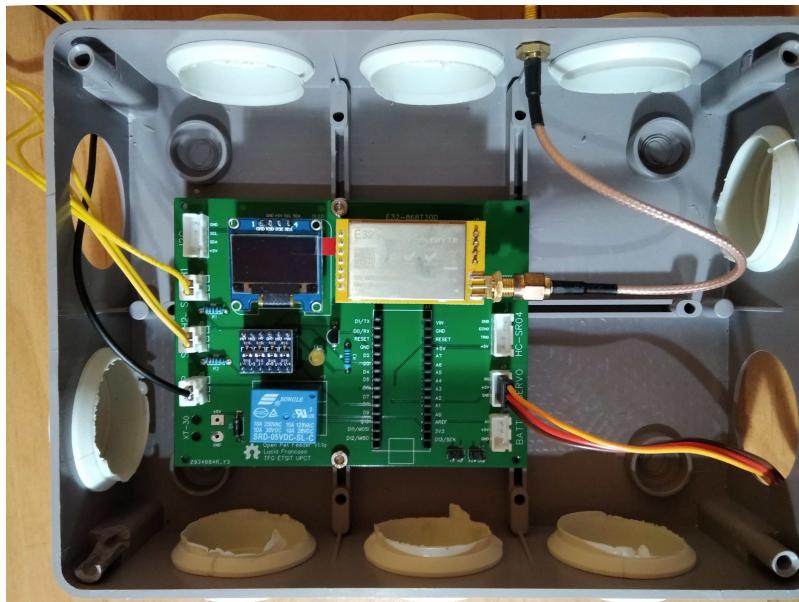


Figura 96: Interior de la caja con conexiones a servo, a bomba y a switches.



Figura 97: Interior de la caja con conexiones a servo, a bomba y a switches (zoom).



Figura 98: Imagen del interior de la reserva de agua (detalle de los switches).



Figura 99: Imagen del interior de la reserva de agua (detalle de los switches y bomba).



Figura 100: Detalle de la bomba de agua usada en la reserva de agua.



Figura 101: Exterior de la reserva de agua (detalle de perforaciones para switches).



Figura 102: Exterior de la reserva de agua (detalle de perforaciones para conexiones de la bomba y ventilación).

## Sistema de alimentación conjunto (prototipo)



Figura 103: Prototipo conjunto del sistema de alimentación creado, formado por bebedero y comedero.

#### 4.2.5 Configuración del gateway y página web

El dispositivo usado como gateway es una Raspberry Pi 3, una opción *Open Source*, económica, sencilla y bien documentada. Al ser un dispositivo con recursos limitados (es una de las opciones más básicas para servidor), instalaremos *Raspbian* (actualmente, *Raspberry Pi OS*), una distribución de linux ligera y adaptada a Raspberry Pi [36].

Este dispositivo será nuestro servidor, y estará conectado por serial con el arduino nano en del extremo de la comunicación que entendemos como estación del campo (extremo de control). Así pues, será donde se aloje la página web creada para el control remoto del bebedero y comedero.

Para crear la página web, se han empleado Python como lenguaje de programación, y los lenguajes de *maquetado* (lenguajes de *marcado*) HTML + CSS + JavaScript. Se ha hecho uso, además de dos frameworks, uno para la parte front-end de la aplicación web, y otro para la parte back-end:

- **Back-end.** Está programado con el lenguaje de programación Python, haciendo uso del framework conocido como *Flask*, el cual permite un importante grado de personalización, idóneo para emprender proyectos web sencillos (aunque el framework ofrece la posibilidad de que escale al gusto de cada uno)[38]. A través de una librería especial de Flask, ofrecida por el usuario [RedFash](#), con el nombre [\*flask-serial\*](#), establecemos la comunicación serial entre la página web y el arduino nano del GW, para poder enviar órdenes desde la página web y que el arduino nano los reciba y pueda transmitir vía LoRa hacia la estación del refugio [37].
- **Front-end.** Haciendo uso de los lenguajes de marcado mencionados, se define el estilo de la web (su apariencia) y se emplea el framework *Bootstrap*; se trata de un framework que combina CSS y JavaScript para estilizar los elementos de una página HTML, proporcionando interactividad y dinamismo en la página web, donde se mostrarán los elementos de la misma sin necesidad de volver a cargar la página para ver cómo varían en apariencia o valor. Su característica más significativa es que permite la construcción de páginas web *responsive*, es decir, páginas web cuya presentación se adaptará al dispositivo donde se visualicen [39][40].

La apariencia de la página web creada es tal como muestra la Figura 104:

# Open Pet Feeder

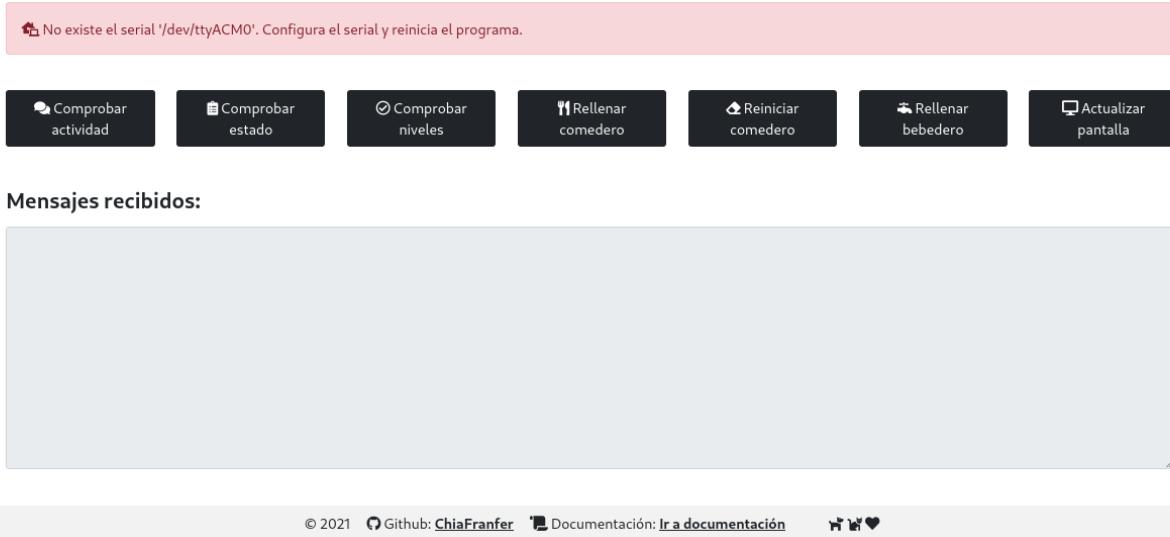


Figura 104: Captura de la página web creada (primera versión), en concreto, de la página principal, donde se muestran los botones, los links y el mensaje de aviso.

Los enlaces que aparecen en el pie de la página, tal como muestra la Figura 104, redirigen, respectivamente, a mi cuenta de GitHub (donde se podrá visualizar el repositorio creado para este proyecto y otros, con todo el código y documentación necesaria para recrear este proyecto), y la segunda página de esta web, donde hay un apartado con documentación relativa al uso de la misma (se trata de una guía de usuario a efectos prácticos).

El aviso que aparece en rojo existe para avisar al usuario de que no se detecta el dispositivo en el puerto especificado para realizar la comunicación serial. Si todo está conectado correctamente, aparecerá un aviso en verde ("Conectado a GW"), el cual nos informará sobre la disponibilidad de dicho puerto, comunicando que estamos conectados al gateway y que, por tanto, se puede realizar la comunicación serial (Figura 105).

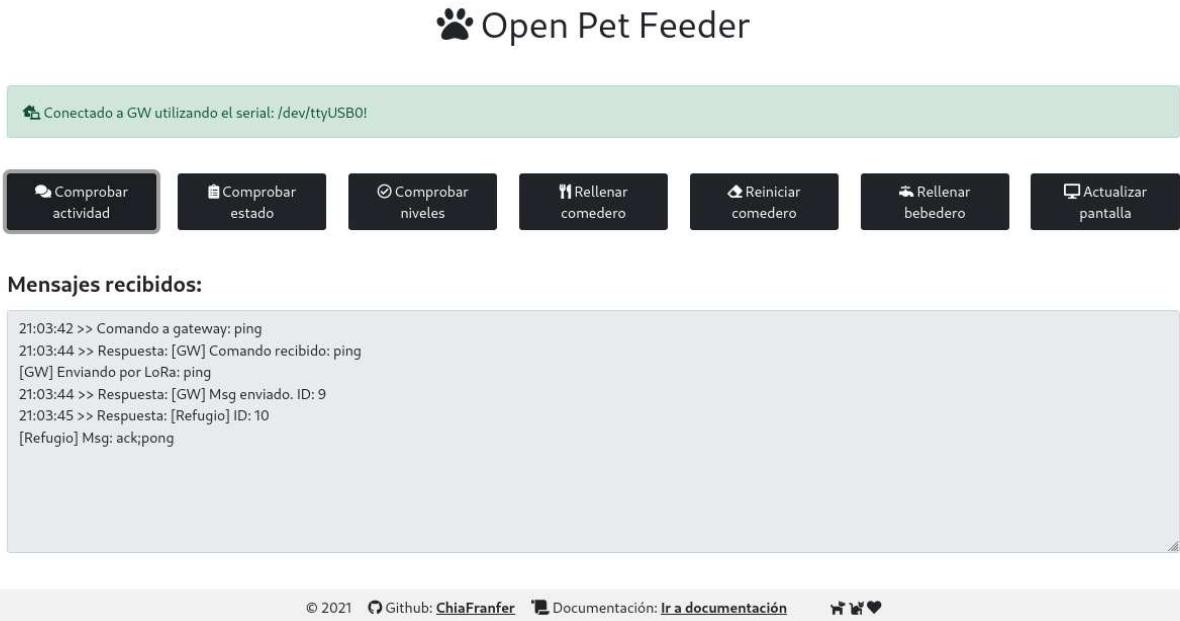


Figura 105: Captura de la página web creada (primera versión), en concreto, de la página principal, donde se muestra un ejemplo de envío de mensajes y recepción, visible en la consola de la web.

Llegados a este punto, concluimos que el prototipo definitivo es capaz de crear un enlace radio vía LoRa y que desde una web de creación propia podemos enviar comandos hacia el refugio y recibir respuesta de manera satisfactoria. Para entender los diferentes comandos enviados desde la web y, en general, cómo usarla, se ha creado un documento PDF con documentación sobre la web, alojado en el repositorio donde se incluye todo lo relacionado con el proyecto: [GitHub: documentación sobre la web](#). Tras estos resultados, resta realizar pruebas de campo que validen la creación de un enlace radio LoRa en entornos realistas.

## 4.3 Pruebas

Para este proyecto, se han realizado numerosas pruebas, pero podríamos diferenciarlas en dos grandes grupos; las pruebas de campo, para comprobar la cobertura del enlace punto a punto, y las pruebas de interior, en las que se comprueba que todo lo que se ha programado y preparado funciona como se espera.

### 4.3.1 Pruebas de campo

Por el momento, se han realizado dos, una en un entorno conocido, llegando hasta los 3 kms de alcance del enlace radio, y otra en el entorno real, también conocido.

#### Prueba 1: entorno conocido, no definitivo

Esta prueba se realizó con el prototipo provisional, en protoboard (Figura 106), durante el mes de abril de 2021. Primero se realizó una prueba en interior para investigar cómo funcionaba este transceptor y verificar la comunicación entre ambos nodos (se explicará más en detalle en *Pruebas de interior*). Una vez completado ese paso, se investigó sobre la librería [Adafruit\\_SSD1306](#) [62], la cual permite trabajar con OLEDs monocromáticas SSD1306 de resolución 128x64 y 128x32. Se incorporó las funcionalidades necesarias de dicha librería para poder mostrar información en la OLED; para esta primera prueba la OLED nos fue muy útil, ya que pudimos comprobar si los datos que llegaban eran legibles y coherentes sin tener que hacer uso de una consola en el ordenador para leer por serial lo que se recibe por radio, opción que resulta más incómoda mientras se realizan pruebas de cobertura a pie con el prototipo en la mano.

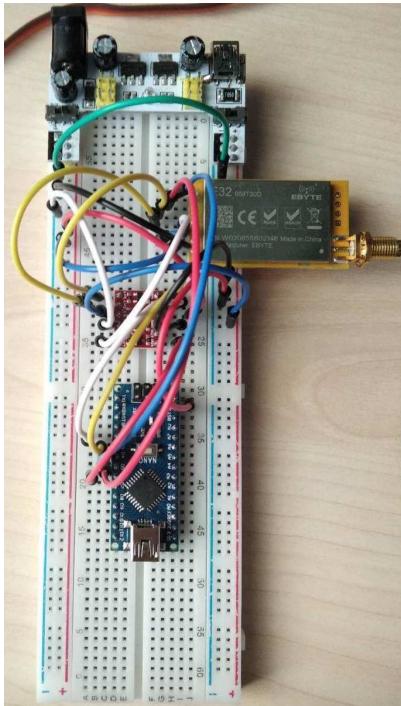


Figura 106: Montaje en protoboard para verificar el funcionamiento y alcance del transceptor LoRa E32-868T30D

Dado que en esta prueba sólo se quiere comprobar el alcance del enlace LoRa (y la calidad, si se envían datos y éstos son legibles), se creó un *sketch* de prueba sencillo y provisional; una vez se probara y se pudiera pasar a la prueba en el entorno definitivo, se usaría un *sketch* que tuviera en cuenta todo lo que se quiere hacer con este prototipo (lo que hemos explicado sobre el *Protocolo*). Se programó el envío cada 5 segundos de la medida de tensión de un pin analógico (concretamente, el A0) del transmisor. También se enviaba, junto con la medida de tensión (mirar OLED de la Figura 107, es el dato que aparece tras *Volts*:) el número de bits de los que consta el mensaje (en la OLED, valor tras *Bits*:) y el número del mensaje que se está enviando (*Count*:); este último empieza desde 0 y se va autoincrementando conforme se envía un mensaje (si se reinicia o apaga el microcontrolador del transmisor, la cuenta empieza desde 0 de nuevo). En el Anexo II se muestra el código empleado para esta prueba (*ReceiveEBYTE\_OLED.ino* para el receptor que iba a desplazarse, y *SendEBYTE.ino* para el extremo que iba a quedar fijo transmitiendo datos).

Se dejó el nodo transmisor transmitiendo mensajes con estos valores, mientras permanecía conectado a la corriente, en la segunda planta de un hogar (unos 5m, en el exterior). Por otro lado, el receptor iba conectado a una OLED donde mostraba los valores de *Volts*, *Bits* y *Count* del mensaje que recibía; la alimentación la recibía a través de una batería portátil.

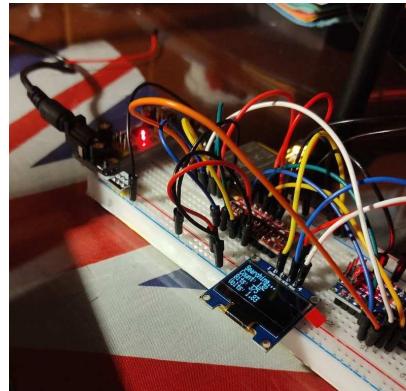


Figura 107

En la Figura 108 se muestra el recorrido que se realizó con el receptor, un recorrido de longitud total 2.7 kms.



Figura 108: Recorrido realizado con el receptor para probar su alcance

En tanto a resultados, la señal llegó en todo momento. Se perdió en torno a un 15 % de los mensajes en zonas donde la vegetación es más densa y, sobretodo, a partir del kilómetro 2.5, si nos situábamos detrás de domicilios, cortando la línea de visión directa (bajo esta situación, no llegaba ningún mensaje, pero reestablecíamos la conexión al desbloquear la línea de visión). Si que es cierto que quizás a esa distancia y estando a poco más de un metro y medio sobre el suelo no se pueda hablar de *línea de visión directa*, pero mirando la Figura 108, donde se muestra que el recorrido fue, en esencia, una línea recta, he decidido denominarlo así.

## Prueba 2: entorno conocido, definitivo

Estas pruebas se realizaron durante el mes de agosto de 2021, en las inmediaciones del refugio de Patitas Unidas. Previo a realizar estas pruebas, se tuvieron que dar por concluidos los detalles referidos al prototipo definitivo, por lo que se demoraron unos meses con respecto a las pruebas en el entorno no definitivo. En la Figura 109 podemos observar la composición del enlace LoRa (tanto dispositivos como conexión entre ellos) que se va a testear con esta prueba.

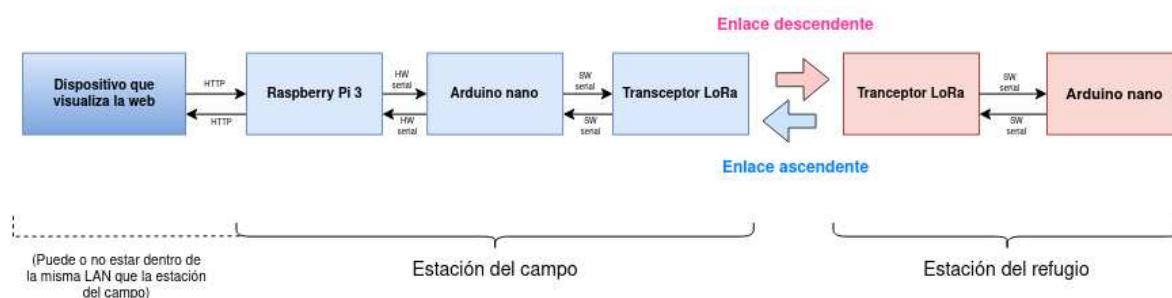


Figura 109: Esquema resumen del flujo de la información en ambos sentidos del enlace LoRa, señalando las vías de comunicación entre bloques fundamentales del enlace.

Para poder comenzar esta prueba, fue imprescindible dejar preparada la estación del campo. Para ello, en una ventana de la segunda planta de la casa se instaló la antena que transmitiría y recibiría los mensajes de esta estación. Para evitar interferencias, se usó un prolongador que separara la antena de la fachada de la casa y de tuberías de cobre que había cerca; esto fue posible gracias a un alargador SMA de longitud 3FT (aproximadamente 1.8m), ya que nos daba cierto margen para sacar la antena al exterior, desde la caja estanca de electrónica hasta el extremo del prolongador empleado (mirar Figura 110).



(a)



(b)



(c)



(d)

Figura 110: Situación de la estación del campo para la prueba de cobertura entre campo y refugio

Como el funcionamiento del conjunto electrónico contenido en la caja estanca ya estaba probado, y el de la Raspberry Pi (ya se explicó en el apartado 4.2.5 que se instaló una distribución especialmente ligera de Linux, la Raspbian y se dejó lista para usar), sólo quedaba para poder probar cobertura del enlace la subida de la página web a la Raspberry Pi, para que se ejecutara desde ahí (actuando como servidor) y se comunicara vía serial con el arduino de esta estación. Recordemos que esto lo hacemos porque el arduino nano no tiene la capacidad de comunicarse con la red LAN, por eso lo que reciba el transceptor LoRa, el arduino está programado para que lo envíe por serial a la Raspberry y esta lo muestra en la consola de la web.

Así pues, ya una vez estuvimos en las inmediaciones del refugio de Patitas Unidas, con la estación del refugio alimentada con una batería portátil, cargamos la página web en un teléfono móvil para mandarle órdenes a dicha estación (desde la estación del campo, ya que es desde donde se ejecuta la web); en la Figura 111 se muestran capturas donde podemos ver, en la consola, los mensajes enviados desde la estación del campo y los mensajes de respuesta, enviados desde la estación del refugio (enviando los mensajes con una potencia de 30dBm). Con estos resultados, podemos afirmar que se establece el enlace radio.

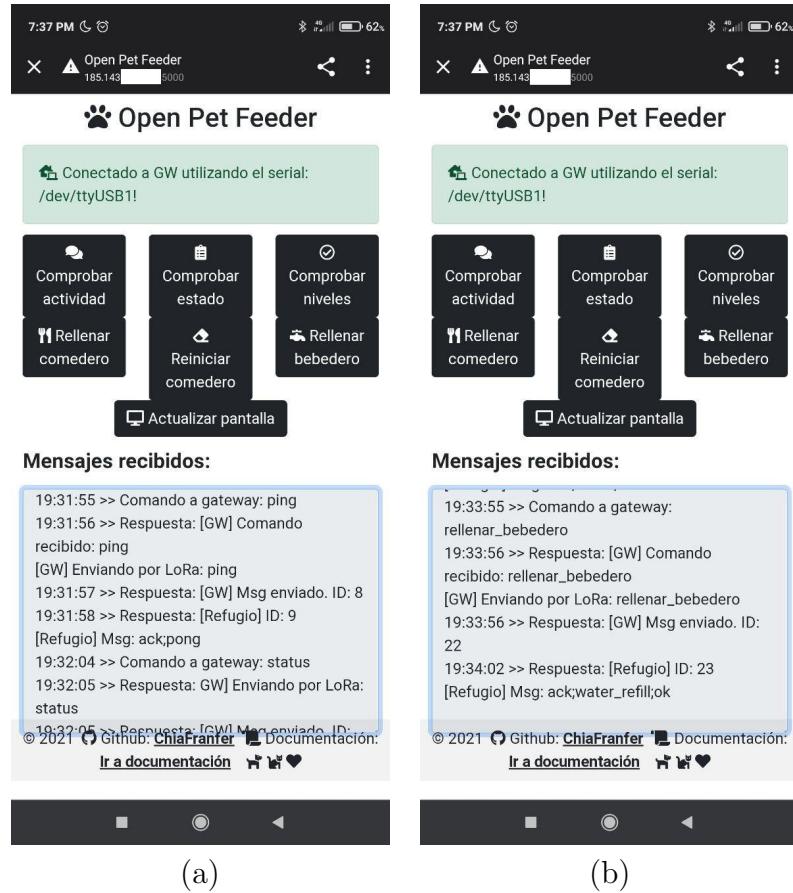


Figura 111: Muestras de la consola de la web *Open Pet Feeder* durante la prueba de alcance en el entorno definitivo (estación del refugio)

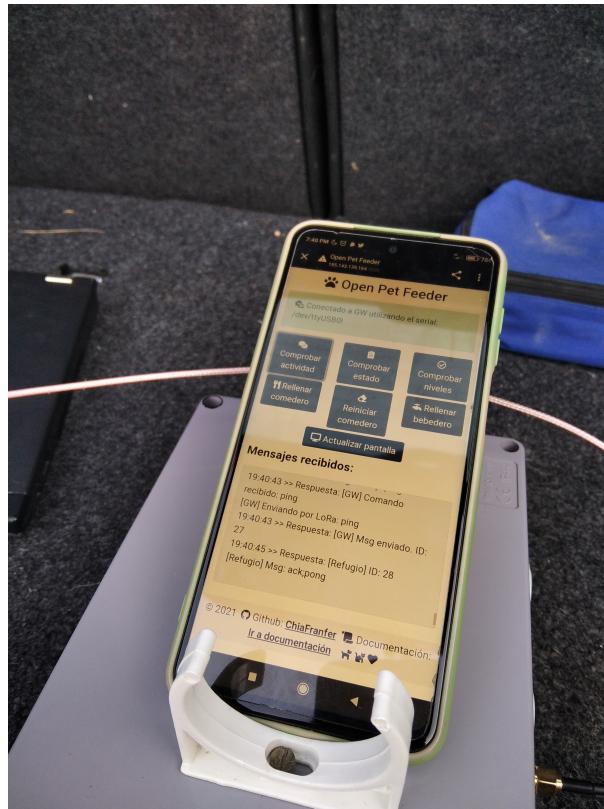
Por último, también en agosto de 2021, unos días después, se probó cobertura, pero esta vez transmitiendo con una potencia de 21dBm. Se pudo establecer el enlace, una vez más, con facilidad y los mensajes se recibían a los pocos segundos de haber enviado el comando desde la web. En la Figura 112 se muestran imágenes de cómo se realizó esta prueba y la anterior (fotografías tomadas el día de la prueba con 21dBm), y una imagen donde se ve que, efectivamente, llega la señal tanto en enlace descendente como ascendente (descendente porque los comandos han llegado a la estación del refugio desde la estación del campo, y ascendente porque la estación del refugio responde correctamente a esos comandos y lo sabemos porque todo queda reflejado en la consola de la web).



(a)



(b)



(c)

Figura 112: Fotos tomadas el día de la prueba de cobertura, con potencia de transmisión 21dBm.

Anotar que el montaje mostrado en la Figura 112 fue el mismo para ambas pruebas (tanto para probar el funcionamiento con potencia de transmisión 30dBm como 21dBm).

Por último, a modo resumen, se muestra en la Figura 113 el funcionamiento que se ha comprobado en estas pruebas de campo.

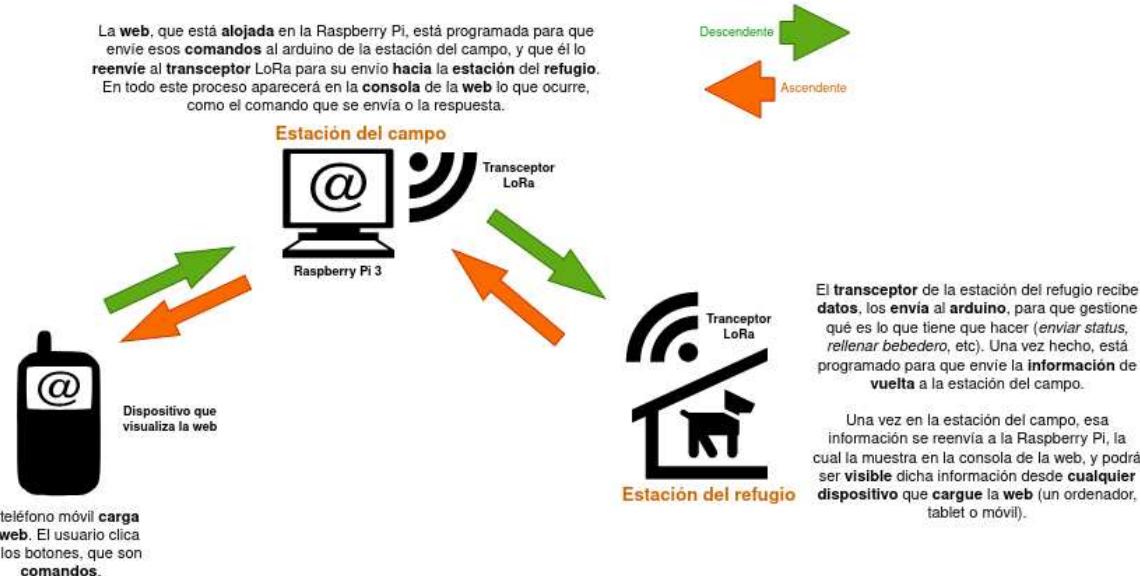


Figura 113: Imagen resumen del funcionamiento del sistema definitivo

#### 4.3.2 Pruebas de interior

Estas pruebas han sido las más numerosas, y pueden desglosarse en:

- Testeo de los componentes de manera individual. Tras la recepción de los productos, y antes de usarlos en ningún prototipo, hay que probarlos de manera individual. Incluye tanto la programación sencilla de programas para el testeo del hardware, como la creación de circuitos sencillos en protoboard para ejecutar esos programas y verificar el estado del componente que se está testeando.
- Testeo de los componentes en protoboard formando el prototipo provisional.
- Testeo de los componentes en la PCB, formando el prototipo final. Tenemos en cuenta, además, el formato elegido para el exterior del diseño (reservas, cableados y conexionado físico), probamos su estabilidad y comodidad, y descartamos opciones.

##### 4.3.2.1 LoRa

Para comprobar su funcionamiento, se usó el montaje mostrado en la Figura 106.

#### 4.3.2.2 Bebedero

**Bomba de agua:** se comprobó simplemente conectando su cable USB a la alimentación con un cargador, introduciéndola en un tarro con agua y un tubo transparente conectada a la salida de agua de la bomba, llevando el agua a un tarro vacío. Para uso posterior de la bomba de agua, se cortó el USB y se dejó los cables por separado de la bomba.

**Relé:**

#### 4.3.2.3 Comedero

**Servomotor:**

#### 4.3.2.4 Sistema de alimentación autónoma

**Batería 18650:**

**Panel solar y TP4056:**

## 4.4 Comentarios sobre los resultados de las pruebas

Aunque se hablará más adelante en *Conclusiones y líneas futuras* con más detalle tanto de deducciones que extraen tras las pruebas, como las que se extraen de los demás apartados de este proyecto, en este punto podemos dejar por escrito ciertos comentarios relacionados con las pruebas que acabamos de comentar:

- Se ha conseguido crear un enlace radio LoRa a 868MHz satisfactorio, donde, tanto a la distancia entre nodos de 2.7kms como a 3.45kms, hemos conseguido mantener una comunicación eficaz y sin problemas.
- La cobertura empeora conforme aumenta la presencia de edificaciones, y la densidad de éstas, sobretodo si la altura de las antenas es bastante inferior a la altura de dichas edificaciones.
- El límite inferior de potencia a la que transmite este transceptor es de 21dBm, pero el límite legal para emitir en la banda de LoRa, en Europa, es de 14dBm. Como ya se ha comentado anteriormente, habría que probar con otro transceptor muy similar, el E22-900T22D, ya que éste puede transmitir a menor potencia y, a demás, dar valores de RSSI, algo que no hemos podido medir en este proyecto por limitaciones del transceptor en este sentido.
- Destacar la importancia de realizar simulaciones antes de la realización de pruebas de campo, ya que éstas requieren de tiempo, tanto de planificación previa de la jornada como de presenciarse in situ en el lugar de pruebas. Si una simulación nos dice que es posible que un enlace no se establezca, nos está indicando que quizás no merezca la pena probarlo bajo esas condiciones en el entorno real, y lo ideal sería probar otras combinaciones.
- Las órdenes (o comandos) pueden ser enviados usando la frecuencia 868MHz sin problema desde la página web creada, desde dispositivos diferentes, y fuera de la red LAN donde está situado el servidor que aloja esa web. Esta característica lo hace especialmente funcional, ya que no supone la presencialidad en ninguna de las dos estaciones para que funcione.
- El sistema de alimentación autónoma no se probó en el entorno definitivo, ya que en tanto a tiempo y dedicación se escapaba del alcance de este trabajo. Si bien es uno de los objetivos de este proyecto, se ha conseguido desarrollar un prototipo compatible con alimentación autónoma, y un primer prototipo de alimentación autónoma, prototipo que supone un proyecto aparte por su complejidad. Sí se probó en interior, y se sacó en conclusión precisamente que era un tema que tenía que investigarse más en profundidad de manera aislada, ya que aunque funcione, hay que estudiar cómo se comporta a lo largo del tiempo y tener vigilado el sistema, ya que un imprevisto podría perjudicar seriamente la integridad del proyecto.

## 4.5 Presupuesto

En el Anexo III se presenta el presupuesto del prototipo construido, sin incluir tiempo de diseño del sistema, prototipado, construcción, instalación del prototipo y pruebas. Como puede verse, el presupuesto material, excluyendo el relacionado con lo requerido por herramientas tales como impresora 3D (filamento) o soldador (estaño, flux), asciende a 248.9€, que, para ser un pequeño proyecto académico, presenta un coste considerable, incluso habiendo recurrido a productos *low cost*.

## **4.6 Resumen del capítulo**

En este capítulo, partiendo de la especificación de la ubicación donde se realizan las pruebas, se ha mostrado el prototipo del sistema diseñado junto con las pruebas realizadas y algunos comentarios sobre los resultados obtenidos. Por un lado, se ha construido un primer prototipo que emplea 868MHz en el enlace LoRa punto a punto y se ha comprobado mediante pruebas que cumple con lo esperado. Por otro lado, se ha realizado una pequeña descripción del prototipo definitivo que se ha construido, de forma que tanto el sistema de bebedero como el de comedero funcionen, así como nuestro gateway personalizado, manteniendo funcional el enlace punto a punto. Pese a los condicionantes, se ha podido comprobar que el sistema funciona y el prototipo construido, aunque es mejorable, logra los objetivos planteados inicialmente. Para acabar, además del presupuesto material, se han expuesto algunas conclusiones que se han obtenido tras la realización de las pruebas, comentando aspectos relacionados con [...].

## 5 Conclusiones y líneas futuras

**Alimentación:** estudiaría las baterías LiPo, incluso LiFePO<sub>4</sub>, que aunque son más caras, no limitan tanto en el tema de la protección. También investigaría el uso de varias baterías Li-ion en paralelo para aumentar la capacidad. Investigaría más profundamente en general el tema de la alimentación autónoma, ya que es un mundo muy extenso que debe tratarse de manera externa a la parte radio o a la parte de microcontrolador; podemos realizar proyectos donde abarquemos todo, pero para llegar a conclusiones donde se lleguen a prototipos iniciales, que en posteriores investigaciones se expandan y mejoren sus funcionalidades de manera individual, sin perder de vista el proyecto global. Diseñaría una PCB sólo para alimentación, ya que los requerimientos de alimentación para cada proyecto son diferentes y para un proyecto de dimensión media-grande es mucho más práctico diseñar un sistema personalizado de alimentación de dispositivo, algo que conseguimos mediante una PCB, compactando los sistemas que, sin PCB, se traducen en numerosas PCBs ya fabricadas que quizás no se adapten del todo a lo que necesitamos. Podría parecer que la PCB de un TP4056 es suficiente, pero en la práctica vemos cómo carga y consumo tiene que ser en momentos no simultáneos, lo cual puede suponer un problema si se sincroniza con el funcionamiento del dispositivo al que está alimentando. Es un tema complejo que hay que investigar aparte, como se ha mencionado.

**Microcontrolador:** el arduino nano para principiantes, con previas nociones de electrónica y con previsiones de realizar una PCB, siempre y cuando el proyecto no sea muy extenso, puede ser empleado. Sin embargo, yo he experimentado que en espacio de memoria se me quedaba corto, sobretodo al usar librerías como OLED, las cuales debía excluir si quería cargar en el arduino mi código restante; teniendo en cuenta que debía importar librerías obligatoriamente para usar por ejemplo la OLED, el servomotor o el transceptor LoRa, este es un punto muy negativo, ya que es un código que no puede ser reducido. Tuve que buscar trucos para compactar el uso de dicha memoria debido a estas limitaciones. Probaría con otras opciones mencionadas en la sección 3.2.2, donde hablamos de opciones de microcontroladores, como la Adafruit, NodeMCU o Teensy.

**LoRa:** el transceptor LoRa usado en este prototipo (E32-868T30D) no nos proporciona el valor de RSSI. Sin embargo, ya se ha presentado a lo largo del documento otra opción, entre otras que existen, que sí nos proporcionan este valor, como el E22-900T22D, más compacto, con una tecnología igualmente funcional y al mismo precio. Su potencia máxima de transmisión es menor, pero se pueden adquirir otros transceptores E22 que pueden transmitir a 30dBm (E22-900T30D). No obstante, es un valor de potencia que no está permitido en la regulación LoRa europea (ya la hemos presentado en este documento). Otro módulo que resulta interesante es el E220-900T22D, por ser ligero, consumir considerablemente menos que los transceptores anteriormente mencionados y tener las mismas prestaciones en tanto a alcance y potencia de transmisión.

Comentar que los transceptores que se han investigado para este TFG tienen diferentes modos, pero no se han empleado. Es otra investigación que queda abierta para posteriores proyectos.

## Glosario

**SRAM (*Static Random Access Memory*)**. Para placas Arduino basadas en arquitectura AVR, la SRAM es donde se almacenan datos temporales y datos en tiempo de ejecución (el sketch crea y manipula variables cuando se ejecuta). Es una memoria volátil, que se borra cada vez que el microcontrolador deja de tener alimentación [54][55][56].

**EEPROM (*Electrically Erasable Programmable Read-Only Memory*)**. EEPROM en Arduino se usa generalmente para almacenar una pequeña cantidad de datos como estados de dispositivos de entrada o salida para que se puedan retener incluso si el Arduino deja de tener alimentación (ergo, es una memoria no volátil, que usamos para almacenar información a largo plazo) [54][55][56].

**Flash memory**. Es la memoria de programa, donde Arduino almacena el sketch. La memoria flash también se llama Flash ROM. En Arduino, la memoria flash almacena el código de la aplicación que se ejecutará. En segundo plano, esto compilará el sketch y producirá un archivo binario y almacenará el archivo binario en la memoria flash de Arduino. Al reiniciar, el Arduino leerá las instrucciones almacenadas en la memoria flash y realizará la operación necesaria. Una parte de esta memoria flash es realmente utilizada por el Bootloader, que es responsable de almacenar el archivo binario en la memoria flash a través de la interfaz en serie.

En realidad, la memoria flash también es un tipo de EEPROM. La principal diferencia es que EEPROM se puede borrar a nivel de byte, mientras que la memoria flash se puede borrar a nivel de bloque [54][55][56].

**Bootloader**. Al encender o reiniciar, un bootloader es una sección de la memoria del programa que se ejecuta antes de que se ejecute el código principal. Se puede utilizar para configurar el microcontrolador o proporcionar una capacidad limitada para actualizar el código del programa principal. No disponer de un bootloader implica realizar estas funciones a través de un programador externo.

En el caso del Uno, el bootloader de Arduino queda en espera a la vez que observa los pines UART. Si el bootloader no recibe una secuencia particular de bytes a través del puerto serie, entonces el procesador salta a la sección del programa “usuario” para cargar lo que ya esté en la memoria del programa. Este salto carga nuestro programa (o sketch) [57][58].

**PWM (*Pulse Width Modulation*)**. La modulación de ancho de pulso está formada por una señal de onda cuadrada que no siempre tiene la misma relación entre el tiempo que está en alto y el tiempo que está en bajo. El tiempo que la señal se encuentra en el nivel alto (5V) lo denominamos Ton, mientras que el tiempo que está en nivel bajo lo denominamos Toff; la suma de ambos es el periodo de la señal (T). La variación de ancho de pulso consiste en variar Ton y Toff; al cambiar el valor de un PWM, en realidad se están modificando estos tiempos.

Una de las características más importantes de una señal PWM es su ciclo de trabajo o *Duty Cycle*, es decir, la relación entre el tiempo de encendido y el periodo o tiempo total del PWM. Cuanto mayor sea el duty cycle, más tiempo estará la señal de tensión en alto, sin variar el periodo. Por

consecuencia, como el periodo no varía y la suma de Ton y Toff, si el tiempo de encendido aumenta, el tiempo de apagado disminuye. Este es el motivo por el cual se llama modulación de ancho de pulso, porque literalmente se varía el ancho del pulso de nivel alto.

Lo cierto es que al variar el duty cycle de una señal PWM, lo que estamos haciendo es variar su tensión media; cuando una señal media de tensión atraviesa ciertos componentes electrónicos, puede hacer que su comportamiento cambie. Por ejemplo, los LED, los motores de corriente continua o ventiladores, incluso altavoces y zumbadores. Existen muchas alternativas a la hora de generar una señal PWM. Una de las más clásicas es usar un microcontrolador (en Arduino, la señal de PWM la generamos mediante el uso del método `analogWrite()`). Otra posibilidad es usar el 555 [66].

**UART (Universal Asynchronous Receiver Transmitter).** No es un protocolo de comunicación como SPI e I2C, sino un circuito físico en un microcontrolador o un IC independiente. El propósito principal de un UART es transmitir y recibir datos en serie. Una de las ventajas de UART es que solo usa dos cables para transmitir datos entre dispositivos.

**SPI (Serial Peripheral Interface).**

**MOSI (Master Out Slave In).**

**SDA (Serial Data).**

**SCL (Serial Clock).**

**I2C (Inter-Integrated Circuit).**

**Torque** (en un motor), o par. En mecánica newtoniana, se denomina momento de una fuerza o torque a una magnitud vectorial, obtenida como producto vectorial del vector de posición del punto de aplicación de la fuerza por el vector fuerza, en ese orden. Para un motor, esto es, la fuerza que puede aplicar a través del movimiento rotacional de su eje para levantar o mover una carga; es importante el contexto mecánico previo que hemos establecido para entender la importancia de la dirección en la que apliquemos el movimiento del servo y la fuerza en sí [53].

**Fuerza contraelectromotriz (back EMF).** Fuerza electromagnética que aparece en un circuito inductivo en una dirección tal que se opone a cualquier cambio de corriente en el circuito. De hecho, la conversión de energía en un motor DC es causada por esta fuerza: la dirección de la corriente en la armadura conductiva que tienen los motores es contraria a la dirección del efecto contraelectromotriz, por lo que debemos realizar un trabajo eléctrico para contrarrestar ese efecto (usar la corriente en contra del *back EMF*), el cual se traduce en realizar un trabajo mecánico, el cual es la función principal de un motor DC. Además, el *back EMF* hace que los motores se autorregulen; por ejemplo, si no existe carga, se hace menos torque, se induce menor corriente y por tanto la fuerza contraelectromotriz aumenta (sería equivalente a la tensión suministrada, ya que su efecto

se mide en voltios), pero si aumentamos la carga, la velocidad disminuye, decrece el *back EMF*, aumenta la corriente y la magnitud del torque. Existen fórmulas que relacionan estas variables [52].

**Resistencia *pull-down*.** Las resistencias *pull-down*, y *pull-up*, son resistencias que se usan en circuitos lógicos para asegurar un nivel lógico bien definido en un pin bajo cualquier circunstancia. Como recordatorio, los circuitos lógicos digitales tienen tres estados lógicos: alto, bajo y flotante (alta impedancia). El estado de alta impedancia ocurre cuando un pin no llega a estar en valor alto o bajo, y se queda “flotando”. Una buena manera de ilustrar esta situación es un pin de entrada desconectado de un microcontrolador; no se encuentra en un estado lógico alto o bajo, y el MCU podría interpretar de manera impredecible el valor de entrada como alto o bajo. También ocurre si en un pin del MCU tenemos un switch; si no hay una resistencia *pull-down* o *pull-up*, el pin estará flotando cuando el switch se abra, y dando valores lógicos conocidos sólo cuanto éste se cierre. Son resistencias de valor fijo, que dependiendo de la aplicación del circuito digital que estemos diseñando tendrán un valor u otro (y de la impedancia de entrada del pin del MCU al que queramos conectarnos, por ejemplo). La diferencia entre una resistencia *pull-down* y una *pull-up* es que la primera se coloca entre tierra y el pin en cuestión, haciendo que, cuando el switch se abra, el pin tenga valor lógico bajo, mientras que la segunda se coloca entre VCC y el pin, haciendo que, cuando el switch se abra, el pin tenga valor lógico de alto [51].

**Lenguaje de marcado.** Es un modo de codificar (redactar) un documento donde, junto con el texto, se incorporan etiquetas (marcas o anotaciones) con información adicional relativa a la estructura del texto o su formato de presentación. Permiten hacer explícita la estructura de un documento, su contenido semántico o cualquier otra información lingüística o extralingüística que se quiera hacer patente [50]. Ejemplos de lenguaje de marcado son HTML o LaTeX, este último usado para la creación del presente documento.

**Framework.** Un framework, que se podría traducir aproximadamente como marco de trabajo, es el esquema o estructura que se establece y que se aprovecha para desarrollar y organizar un software determinado; podría resumirse como el entorno pensado para hacer más sencilla la programación de cualquier aplicación o herramienta actual. Sirve para poder escribir código o desarrollar una aplicación de manera más sencilla, ya que permite una mejor organización y control de todo el código elaborado, así como una posible reutilización en el futuro (entre algunas ventajas, la automatización de procesos para tareas repetitivas como consulta en base de datos o realizar llamadas a internet). Debido a esto, garantiza una mayor productividad que los métodos más convencionales y una minimización del coste al agilizar las horas de trabajo volcadas en el desarrollo. Por otra parte, su acción es algo que afecta también a los errores, minimizándolos considerablemente [63].

**Front-end.** El Front end es la parte de una web que conecta e interactúa con los usuarios que la visitan. Es la parte visible, la que muestra el diseño, los contenidos y la que permite a los visitantes navegar por las diferentes páginas. Es una de las dos mitades en las que se divide la estructura de cualquier página web; la otra mitad es el Back end. Ambas se reúnen en cualquier sitio web que visite una persona y son las que, trabajando, hacen que funcione en todo momento tal y como lo hace. Esta parte es la que engloba y muestra todo el trabajo de diseño web y, por lo general, reúne en su interior hasta 3 lenguajes de programación diferentes: HTML, CSS y JavaScript. Cada uno

orientado a determinados fines en concreto, se suman para conseguir el resultado final que aparece por la pantalla de cada usuario que entra en una web, sea cual sea [64].

**Back-end.** El backend es la parte del desarrollo web que se encarga de que toda la lógica de una página web funcione. Se trata del conjunto de acciones que pasan en una web pero que no vemos como, por ejemplo, la comunicación con el servidor. Algunas de las funciones que se gestionan en la parte del back-end son: desarrollo de funciones que simplifiquen el proceso de desarrollo, acciones de lógica, conexión con bases de datos o uso de librerías del servidor web (por ejemplo para implementar temas de caché o para comprimir las imágenes de la web). Además, tiene que velar por la seguridad de los sitios web que gestiona y optimizar al máximo los recursos para que las páginas sean ligeras. Algunos lenguajes de programación back end son Python, PHP, Java o .Net 65.

# Bibliografía

## Enlaces y referencias

1. National Geographic: animales en peligro de extinción. Fecha última consulta: 15/07/21.
2. BBC: animales en peligro de extinción. Fecha última consulta: 15/07/21.
3. WWF: qué significa *animales en peligro de extinción*. Fecha última consulta: 15/07/21.
4. Fundación AQUAE: causas de la pérdida de biodiversidad. Fecha última consulta: 15/07/21.
5. Fundación Affinity: estudio *Él nunca lo haría (2020)*, sobre el abandono y adopción. Fecha última consulta: 15/07/21.
6. 20minutos: abandono animal tras el confinamiento de 2020. Fecha última consulta: 15/07/21.
7. La Razón: comentarios sobre el estudio de la Fundación Affinity acerca de tasa de abandono animal durante el año 2019. Fecha última consulta: 15/07/21.
8. RTVE: abandono animal tras el confinamiento. Fecha última consulta: 15/07/21.
9. TI: *characteristics of rechargeable batteries*. Consultado el 15/07/21.
10. Hackaday: battery basics. Choosing a battery for your project. Consultado el 16/08/21.
11. YouTube: How to choose a battery. A battery chemistry tutorial. Consultado el 16/08/21.
12. Blog: qué diferencias hay entre una Li-ion y una Li-Po. Consultado el 16/08/21.
13. Adafruit Learning System: How to pick the right battery for your project. Consultado el 16/08/21.
14. Prometec: las baterías. Consultado el 16/08/21.
15. Prometec: cuánto consume Arduino. Consultado el 16/08/21.
16. Prometec: Arduino y el modo *sleep*. Consultado el 16/08/21.
17. Prometec: relés. Consultado el 16/08/21.
18. Prometec: relés (más info). Consultado el 16/08/21.
19. Prometec: arduino y los relés. Consultado el 16/08/21.
20. Fly power: the rechargeable batteries's main parameters. Consultado el 16/08/21.
21. Solar Reviews: How do solar panels work?. Consultado el 16/08/21.
22. Energía solar: regulador de carga. Consultado el 16/08/21.
23. CED greentech: how do MPPT charge controllers work?. Consultado el 16/08/21.

24. TU Delft OCW: solar cell parameters and equivalent circuit. Consultado el 16/08/21.
25. Alternative Energy Tutorials: photovoltaic types. Consultado el 16/08/21.
26. Renewable energy hub UK: what Types of Solar Cells Are There. Consultado el 16/08/21.
27. SlidePlayer: Photovoltaic systems (Julian Melvyn Chambers). Consultado el 16/08/21.
28. QVP Research Group: How does air temperature affect photovoltaic solar panel output?. Consultado el 16/08/21.
29. All3DP: 10 Best Arduino Alternatives. Consultado el 16/08/21.
30. Make of use: 6 Best Arduino Alternative Microcontrollers. Consultado el 16/08/21.
31. ITIGIC: The Best Alternatives for Arduino Microcontrollers. Consultado el 16/08/21.
32. Engadget: Raspberry Pi Pico is a \$4 Arduino alternative. Consultado el 16/08/21.
33. Raspberry Pi: Meet Raspberry Silicon: Raspberry Pi Pico now on sale at \$4. Consultado el 16/08/21.
34. Thingiverse: Auger-based Cat Feeder. Consultado el 16/08/21.
35. AliExpress: cuenco acero inoxidable (bebadero, comedero). Consultado el 16/08/21.
36. Raspberry Pi: software, *Raspberry Pi OS (Raspbian)*. Consultado el 16/08/21.
37. GitHub: flask-serial. Consultado el 16/08/21.
38. Flask: documentación. Consultado el 16/08/21.
39. Bootstrap: documentación. Consultado el 16/08/21.
40. Bootstrap: información. Consultado el 16/08/21.
41. Objetivo 15 de Desarrollo Sostenible (ODS). Consultado el 16/08/21.
42. Facebook: Patitas Unidas Los Alcázares. Consultado el 16/08/21.
43. Web de Patitas Unidas Los Alcázares. Consultado el 16/08/21.
44. Create Arduino: *LoRa E32 for Arduino, ESP32 or ESP8266: Specs and Base Use*. Consultado el 16/08/21.
45. GitHub: LoRa\_E32\_Series\_Library. Consultado el 16/08/21.
46. Create Arduino: xreef user. Consultado el 16/08/21.
47. HWLibre. TP4056: el módulo para cargar baterías. Consultado el 16/08/21.
48. IoT For All: LPWAN - The Benefits of LPWAN Technology vs. Other IoT Connectivity Options. Consultado el 16/08/21.

49. GitHub: librería EBYTE, de Kris Kasprzak. Consultado el 16/08/21.
50. TicArte: Qué son los lenguajes de marcas. Consultado el 19/08/21.
51. EEPower: Pull-up and Pull-down Resistors. Consultado el 19/08/21.
52. Circuit Globe: Back EMF in DC Motor. Consultado el 19/08/21.
53. Wikipedia: Momento de fuerza. Consultado el 19/08/21.
54. Arduino: memory foundations. Consultado el 19/08/21.
55. SeeedStudio: Managing Arduino Memory: Flash, SRAM, EEPROM!. Consultado el 19/08/21.
56. ElectronicsHub: Different Types of Memory on Arduino | SRAM, EEPROM, Flash. Consultado el 19/08/21.
57. BaldEngineer: Arduino Bootloader, What is it?. Consultado el 19/08/21.
58. Arduino: Bootloader Development. Consultado el 19/08/21.
59. Electronics tutorials: BJT explanation. Consultado el 19/08/21.
60. HardwareLibre: HC-SR04. Consultado el 19/08/21.
61. Diseño de un sistema de monitorización remota de un depósito de agua mediante LoRa (TFM). Rubén Adrián de la Cámara. Consultado el 19/08/21.
62. GitHub: Adafruit\_SSD1306. Consultado el 27/08/21.
63. NeoAttack: Framework
64. NeoAttack: Front end
65. Rafa Arjonilla: BackEnd
66. tRincón Ingenieril: Qué es PWM y para qué sirve

## Imágenes

1. EBYTE: E32-868T30D. Consultado el 15/07/21.
2. EBYTE: E22-900T22D. Consultado el 22/07/21.
3. ResearchGate: Use, Operation and Maintenance of Renewable Energy Systems. Control Methods Applied in Renewable Energy Systems Consultado el 16/08/21.
4. ResearchGate: Two-Stage Fault Diagnosis Method Based on the Extension Theory for PV Power Systems Consultado el 16/08/21.

5. ResearchGate: The PV cell temperature effect on the energy production and module efficiency Consultado el 16/08/21.
6. ResearchGate: Mathematical modeling of Photovoltaic module and evaluate the effect of various parameters on its performance Consultado el 16/08/21.
7. ResearchGate: Functional graphene: synthesis, characterization and application in optoelectronics Consultado el 16/08/21.
8. SciELO: Organic Photovoltaic Cells: History, principle and techniques Consultado el 16/08/21.
9. TodoMicro: servomotor MG996R Consultado el 16/08/21.
10. Forum Pycom: servomotor MG996R Consultado el 16/08/21.
11. Nikoi: pieza PVC T 87° Consultado el 16/08/21.
12. Poolaria: pieza PVC codo 45° Consultado el 16/08/21.
13. Amazon: tubo PVC Consultado el 16/08/21.
14. Mano a mano: pasamuros de PVC Consultado el 16/08/21.
15. DHgate: sensor de nivel de agua Consultado el 16/08/21.
16. Electrostore: bomba de agua Consultado el 16/08/21.
17. Geek factory: relé Consultado el 16/08/21.
18. Yellow pimento: tubo de silicona Consultado el 16/08/21.
19. Cablematic: caja estanca Consultado el 16/08/21.
20. AliExpress: antenas 868MHz 5dBi Consultado el 16/08/21.
21. Patitas Unidas (Facebook): logo Consultado el 16/08/21.
22. ONU: logo ODS Consultado el 16/08/21.
23. Master Instruments: protected lithium ion cells Consultado el 16/08/21.
24. AliExpress: dual MOS 18650 protection Consultado el 16/08/21.
25. AliExpress: placa experimental 9x15cms Consultado el 16/08/21.
26. AliRadar: TP4056 con boost Consultado el 16/08/21.
27. rareComponents: screw terminals Consultado el 16/08/21.
28. Solarbotics: holder para 18650 Consultado el 16/08/21.
29. Zaragoza Maker Space: Curso Arduino y robótica: Servomotores – Control en bucle. Consultado el 02/08/21.

30. Ebotics: PROJECT Nº 3: Control the servomotor. Consultado el 02/08/21.

31. Parallax: Connect Servo Motors and Batteries. Consultado el 02/08/21.

## **Anexos**

Se presentan tres anexos. El Anexo I mostrará detalles técnicos de los componentes usados. El Anexo II, el código de programación creado para este proyecto. Por último, en el Anexo III se mostrará el presupuesto detallado del mismo.

## **Anexo I**

[Fichas técnicas de los componentes usados]

## Anexo II

### Código Arduino usado para prueba de componentes

#### *Sensores de nivel de agua y relé: level\_switch\_test.ino*

```
1 //Sensor de nivel tipo boyas
2 //Salida a rele, para bomba de agua
3
4 #define nivel 9 //mido el nivel de agua en el pin D9
5 #define RELAY_PIN 7 //conecto arduino con rele a traves de pin D7
6
7 void setup()
8 {
9     Serial.begin(9600);
10    pinMode(nivel, INPUT);
11    pinMode(RELAY_PIN, OUTPUT);
12 }
13
14 void loop()
15 {
16     int SensorNivel = digitalRead(nivel); //leemos lo que marca el nivel
17
18     Serial.println(SensorNivel); //comprobamos la lectura
19
20     if (SensorNivel == 1){
21         digitalWrite(RELAY_PIN, LOW); //si el agua supera el nivel, cortamos el paso
22         con rele a 0
23     } else {
24         digitalWrite(RELAY_PIN, HIGH);
25     }
26
27     delay(1000);
28 }
```

#### *Bebedero: sketch\_waterPumpRelay.ino*

```
1 #include <LowPower.h>
2 #include "LowPower.h"
3
4
5 #define SLEEP_CYCLES 4          // tiempo = 4S * SLEEP_CYCLES[0:5] = 16 seg
6
7 //defines for bebedero
8
9 #define RELAY_PIN 7
10 #define REFILL_TIME 5000      // 5000 ms == 5s para rellenar bebedero (default)
11
12 void llenarBebedero() {
13
14     Serial.println("Rellenando bebedero...");
```

```

15
16  digitalWrite(RELAY_PIN, HIGH);
17  delay(REFILL_TIME);
18  digitalWrite(RELAY_PIN, LOW);
19
20  delay(500);
21
22  Serial.println("Bebedero lleno");
23 }
24
25 void setup() {
26
27  Serial.begin(115200);
28
29  delay(1000);
30
31 // setup bebedero
32 pinMode(RELAY_PIN, OUTPUT);
33
34
35  Serial.println("Ready...");
36  delay(1000);
37 }
38
39 void loop() {
40
41 // tengo q decirle aqui que lo haga solo cuando el arduino este despierto, y se
42 // supone q esta despierto el tiempo justo
43 // para q se cojan datos de sensores y q se envie por lora
44 Serial.println("Go to sleep...");
45  delay(200);
46
47  for (int i = 0; i < SLEEP_CYCLES; i++) {
48    LowPower.powerDown(SLEEP_4S, ADC_OFF, BOD_OFF); //se puede subir a SLEEP_8S
49  }
50  Serial.println("Awaken");
51
52  rellenarBebedero();
53
54  delay(500);
55
56 }
57

```

### **Servomotor: sketch\_servoTest.ino**

```

1 #include <Servo.h>
2 #include "LowPower.h"
3
4 Servo servol;
5
6 #define PIN_SERVO 9           //conectado a D9 del arduino nano, que es pin digital y

```

```

    PWM
7 #define NUM_CICLO 4
8 #define SPEED_SERVO 1500      //default: 1500. Lower value, it goes faster
9
10#define SLEEP_CYCLES 4        // tiempo = 4S * SLEEP_CYCLES[0:5) = 16 seg
11
12void rellenarComedero(){
13    Serial.println("Rellenando comedero...");
14    for(int i=0; i<=NUM_CICLO; i++){
15        //Serial.println(i);
16        servol.write(0);
17        delay(SPEED_SERVO);
18        servol.write(105);
19        delay(SPEED_SERVO);
20    }
21
22    // Posicion estatica
23    servol.write(90);
24    delay(SPEED_SERVO);
25    Serial.println("OK");
26}
27
28void setup() {
29
30    Serial.begin(9600);
31    servol.attach(PIN_SERVO);
32
33    delay(1000);
34    //rellenarComedero();
35
36    Serial.println("Ready...");
37    delay(1000);
38}
39
40void loop() {
41
42    // tengo q decirle aqui que lo haga solo cuando el arduino este despertado, y se
43    // supone q esta despertado el tiempo justo
44    // para q se cojan datos de sensores y q se envie por lora, formacion de paquetes
45    Serial.println("Go to sleep...");
46    delay(200);
47
48    for (int i = 0; i < SLEEP_CYCLES; i++) {
49        LowPower.powerDown(SLEEP_4S, ADC_OFF, BOD_OFF);      //se puede subir a SLEEP_8S
50    }
51    Serial.println("Awaken");
52
53    rellenarComedero();
54
55    delay(500);
56
57}
58
```

## OLED: OLED\_display.ino

```
1 #include <Wire.h>
2 #include <Adafruit_GFX.h>
3 #include <Adafruit_SSD1306.h>
4
5 #define SCREEN_WIDTH 128 // OLED display width, in pixels
6 #define SCREEN_HEIGHT 32 // OLED display height, in pixels
7
8 // Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
9 // The pins for I2C are defined by the Wire-library.
10 // On an arduino UNO:          A4(SDA), A5(SCL)
11 #define OLED_RESET      -1 // Reset pin # (or -1 if sharing Arduino reset pin)
12 #define SCREEN_ADDRESS 0x3C // See datasheet for Address; 0x3C for 128x32
13
14 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
15
16 void setup() {
17     Serial.begin(115200);
18
19     // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
20     if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
21         Serial.println(F("SSD1306 allocation failed"));
22         for(;;); // Don't proceed, loop forever
23     }
24
25     // Show initial display buffer contents on the screen --
26     // the library initializes this with an Adafruit splash screen.
27     display.display();
28     delay(2000); // Pause for 2 seconds
29
30     // Clear the buffer
31     display.clearDisplay();
32
33     // Example
34     display.setTextSize(1);
35     display.setTextColor(WHITE);
36     display.setCursor(0, 0);
37
38     display.println("Hola mundo! :D");
39     display.println("msg recibido");
40     display.println("3");
41     display.println("4");
42     display.display();
43     delay(100);
44
45     // Scrolling mode
46     //display.startscrollright(0x00, 0x0F);
47     //delay(2000);
48     //display.stopscroll();
49     //delay(1000);
50     //display.startscrollleft(0x00, 0x0F);
51     //delay(2000);
```

```

52 //display.stopscroll();
53 //delay(1000);
54 }
55
56 void loop() {
57 // put your main code here, to run repeatedly:
58
59 }
60

```

**Comunicación LoRa:** nodo de control, *SendEBYTE.ino*

```

1 #include <SoftwareSerial.h>
2 #include "EBYTE.h"
3 // #include <avr/io.h>
4
5 #define PIN_RX 2
6 #define PIN_TX 3
7 #define PIN_M0 4
8 #define PIN_M1 5
9 #define PIN_AX 6
10
11 // replace with your own
12 struct DATA {
13     unsigned long Count;
14     int Bits;
15     float Volts;
16     //float Amps;
17 };
18
19 int Chan;
20 DATA MyData;
21
22 // you will need to define the pins to create the serial port
23 SoftwareSerial ESerial(PIN_RX, PIN_TX);
24
25
26 // create the transceiver object, passing in the serial and pins
27 EBYTE Transceiver(&ESerial, PIN_M0, PIN_M1, PIN_AX);
28
29 void setup() {
30
31     Serial.begin(115200);
32
33     // start the transceiver serial port--data sheet says to keep on 9600
34     ESerial.begin(9600);
35
36     Serial.println("Starting Sender");
37
38     // this init will set the pinModes for you
39     Transceiver.init();
40
41     // all these calls are optional but shown to give examples of what you can do

```

```

42 //Serial.println(Transceiver.GetAirDataRate());
43 //Serial.println(Transceiver.GetChannel());
44
45 Transceiver.SetMode(MODE_NORMAL);
46 //Transceiver.SetMode(MODE_WAKEUP);
47
48 //Transceiver.SetUARTBaudRate(UDR_9600);
49 //Transceiver.SetAirDataRate(ADR_2400);
50 //Transceiver.SetAddressH(2);
51 //Transceiver.SetAddressL(0);
52 //Chan = 5;
53 //Transceiver.SetChannel(Chan);
54 //Transceiver.SetTransmitPower(OPT_TP21);
55 //Transceiver.SetPullupMode(0);
56 //Transceiver.SetFECMode(1);
57
58 // save the parameters to the unit,
59 //Transceiver.SaveParameters(PERMANENT);
60 Transceiver.SaveParameters(Temporary);
61
62
63 // you can print all parameters and is good for debugging
64 // if your units will not communicate, print the parameters
65 // for both sender and receiver and make sure air rates, channel
66 // and address is the same
67 Transceiver.PrintParameters();
68
69 MyData.Count = 0;
70 }
71
72 void loop() {
73
74 // measure some data and save to the structure
75 MyData.Count++;
76 //MyData.Count = 10;
77 MyData.Bits = analogRead(A0);
78 MyData.Volts = MyData.Bits * ( 5.0 / 1024.0 );
79 //MyData.Volts = 2000.0;
80
81 // i highly suggest you send data using structures and not
82 // a parsed data--i've always had a hard time getting reliable data using
83 // a parsing method
84 Transceiver.SendStruct(&MyData, sizeof(MyData));
85 //Transceiver.SendByte(8);
86
87 // let the user know something was sent
88 Serial.println("Sending: Count: " + (String) MyData.Count + " Volts: " + MyData.
89   Volts + " Bits: " + MyData.Bits );
90 delay(4000);
91
92 }
93

```

**Comunicación LoRa:** nodo sensorización, *ReceiveEBYTE-OLED.ino*

```
1 // LoRa E32
2 #include <SoftwareSerial.h>
3 #include "EBYTE.h"
4 #include <avr/io.h>
5
6 // OLED Display
7 #include <Wire.h>
8 #include <Adafruit_GFX.h>
9 #include <Adafruit_SSD1306.h>
10
11 // LoRa E32
12 #define PIN_RX 2
13 #define PIN_TX 3
14 #define PIN_M0 4
15 #define PIN_M1 5
16 #define PIN_AX 6
17
18 // OLED Display
19 // Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
20 // The pins for I2C are defined by the Wire-library.
21 // On an arduino UNO:      A4(SDA), A5(SCL)
22 #define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)
23 #define SCREEN_ADDRESS 0x3C // See datasheet for Address; 0x3C for 128x32
24 #define SCREEN_WIDTH 128 // OLED display width, in pixels
25 #define SCREEN_HEIGHT 32 // OLED display height, in pixels
26
27
28 struct DATA {
29     unsigned long Count;
30     int Bits;
31     float Volts;
32 };
33
34 // replace with your own
35 int Chan;
36 DATA MyData;
37 unsigned long Last;
38
39 // connect to any digital pin to connect to the serial port
40 // don't use pin 01 and 1 as they are reserved for USB communications
41 SoftwareSerial ESerial(PIN_RX, PIN_TX);
42
43 // create the transceiver object, passing in the serial and pins
44 EBYTE Transceiver(&ESerial, PIN_M0, PIN_M1, PIN_AX);
45
46 uint8_t read;
47
48 // OLED Display
49 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
50
51 void setup() {
```

```

52 Serial.begin(115200);
53
54 delay(500);
55
56 // OLED Display
57 // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
58 if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
59   Serial.println(F("SSD1306 allocation failed"));
60   for(;;); // Don't proceed, loop forever
61 }
62
63 Serial.println("OLED Ready");
64
65 // Show initial display buffer contents on the screen --
66 // the library initializes this with an Adafruit splash screen.
67 display.display();
68 delay(200); // Pause for 2 milliseconds
69
70 // Clear the buffer
71 display.clearDisplay();
72
73 // LoRa Init
74 ESerial.begin(9600);
75 Serial.println("Starting Reader");
76
77 // this init will set the pinModes for you
78 Transceiver.init();
79
80 // all these calls are optional but shown to give examples of what you can do
81
82 // Serial.println(Transceiver.GetAirDataRate());
83 // Serial.println(Transceiver.GetChannel());
84 Transceiver.SetUARTBaudRate(UDR_9600);
85 Transceiver.SetMode(MODE_NORMAL);
86 Transceiver.SetAirDataRate(ADR_2400);
87 Transceiver.SetAddressH(1);
88 Transceiver.SetAddressL(0);
89 Chan = 5;
90 Transceiver.SetChannel(Chan);
91 Transceiver.SetTransmitPower(OPT_TP21);
92 Transceiver.SetPullupMode(0);
93 Transceiver.SetFECMode(1);
94
95 // save the parameters to the unit,
96 Transceiver.SaveParameters(TEMPORARY);
97
98 // you can print all parameters and is good for debugging
99 // if your units will not communicate, print the parameters
100 // for both sender and receiver and make sure air rates, channel
101 // and address is the same
102 Transceiver.PrintParameters();
103
104 // OLED Searching... text
105 display.setTextSize(1);

```

```

106 display.setTextColor(WHITE);
107 display.setCursor(0, 0);
108
109 display.println("Searching...");
110 display.display();
111 delay(200);
112 }
113
114 void loop() {
115
116 // if the transceiver serial is available, proces incoming data
117 // you can also use Transceiver.available()
118
119
120 if (ESerial.available()) {
121
122 // i highly suggest you send data using structures and not
123 // a parsed data--i've always had a hard time getting reliable data using
124 // a parsing method
125
126 Transceiver.GetStruct(&MyData, sizeof(MyData));
127 //read = Transceiver.GetByte();
128
129 //Serial.println(read);
130
131 // Update OLED Info
132 updateDisplay(&MyData);
133
134 // dump out what was just received
135 Serial.print("Count: "); Serial.println(MyData.Count);
136 Serial.print("Bits: "); Serial.println(MyData.Bits);
137 Serial.print("Volts: "); Serial.println(MyData.Volts);
138 Serial.println("****");
139 // if you got data, update the checker
140 Last = millis();
141
142 }
143 else {
144 // if the time checker is over some prescribed amount
145 // let the user know there is no incoming data
146 if ((millis() - Last) > 1000) {
147 Serial.println("Searching: ");
148 Last = millis();
149 }
150
151 }
152 }
153
154 void updateDisplay(struct DATA *msg) {
155 // Clear the buffer
156 display.clearDisplay();
157
158 display.setTextSize(1);
159 display.setTextColor(WHITE);

```

```

160 display.setCursor(0, 0);
161
162 display.println("Searching...");  

163 // Updating new values  

164 display.println("Count: " + (String) msg->Count);  

165 display.println("Bits: " + (String) msg->Bits);  

166 display.println("Volts: " + (String) msg->Volts);  

167 display.display();  

168 delay(200);  

169 }  

170

```

## Código Arduino usado para prototipo definitivo

**Protocolo**, para terminal de estación del refugio: *refugio\_protocolo.ino*

```

1 #include <SoftwareSerial.h>
2 #include "EByte.h"
3 #include <avr/io.h>
4 #include <Servo.h>
5
6 // v1.1a PCB
7 #define PUMP 7
8 #define LEVEL_SW1 8 // nivel superior
9 #define LEVEL_SW2 10 // nivel inferior
10 #define SERVO 9
11 #define PIN_RX 2
12 #define PIN_TX 3
13 #define PIN_M0 4
14 #define PIN_M1 5
15 #define PIN_AX 6
16
17 #define LORA_CHANNEL 5
18
19 #define REFILL_TIME 5000 // rellenarBebedero: 5000 ms == 5s para rellenar bebedero
19 (default)
20
21 #define SPEED_SERVO 1500 // default: 1500. Lower value, it goes faster
22
23 #define NUM_CICLO 4 // rellenarComedero, mantener movimiento giratorio en un
23 ciclo de relleno
24
25 #define TOTAL_REFILL 20 // rellenarComedero, total de recargas de pienso desde
25 reserva hasta llegar al 80% de vaciado (20% no lo contamos por seguridad)
26
27 #define BAUDRATE 115200
28
29 int num_refill = 0; // contador de veces que se ha rellenado el comedero
29 desde la reserva, si es igual a TOTAL_REFILL entonces dejamos de rellenar
30 Servo servol;
31
32 SoftwareSerial ESerial(PIN_RX, PIN_TX); // "falso" serial para comunicarnos con el
32 E32 (creado en SW)

```

```

33 EBYTE Transceiver(&ESerial, PIN_M0, PIN_M1, PIN_AX);
34
35 struct ebyte_struct {
36     unsigned long count;
37     char msg[64];
38 };
39 ebyte_struct ebyte_msg;
40
41 unsigned long Last;
42
43 void setup() {
44     Serial.begin(BAUDRATE);
45     Serial.println("Serial initialized");
46     delay(300);
47
48     // start the transceiver serial port--i have yet to get a different
49     // baud rate to work--data sheet says to keep on 9600
50     ESerial.begin(9600);
51
52     // init ebyte
53     Serial.println("Init transceiver");
54     // this init will set the pinModes for you
55     Transceiver.init();
56
57     // all these calls are optional but shown to give examples of what you can do
58
59     //Serial.println(Transceiver.GetAirDataRate());
60     //Serial.println(Transceiver.GetChannel());
61
62     Transceiver.SetMode(MODE_NORMAL);
63     //Transceiver.SetMode(MODE_WAKEUP);
64
65     Transceiver.SetUARTBaudRate(UDR_9600);
66     Transceiver.SetAirDataRate(ADR_2400);
67     Transceiver.SetAddressH(1);
68     Transceiver.SetAddressL(0);
69
70     Transceiver.SetChannel(LORA_CHANNEL);
71     Transceiver.SetTransmitPower(OPT_TP21); //change transceiver tx-rx power
72     Transceiver.SetPullupMode(0);
73     Transceiver.SetFECMode(1);
74
75     // save the parameters to the unit,
76     //Transceiver.SaveParameters(PERMANENT);
77     Transceiver.SaveParameters(TEMPORARY);
78
79     // you can print all parameters and is good for debugging
80     // if your units will not communicate, print the parameters
81     // for both sender and receiver and make sure air rates, channel
82     // and address is the same
83     Transceiver.PrintParameters();
84
85     ebyte_msg.count = 0;
86

```

```

87 // inicializar sensores
88 Serial.println("");
89 pinMode(LEVEL_SW1, INPUT);
90 pinMode(LEVEL_SW2, INPUT);
91 pinMode(PUMP, OUTPUT);
92
93 servol.attach(SERVO);
94
95 Serial.println("Ready...");
96 delay(500);
97 }
98
99 void loop() {
100
101 // if the transceiver serial is available, proces incoming data
102 // you can also use Transceiver.available()
103
104 if (ESerial.available()) {
105
106 // i highly suggest you send data using structures and not
107 // a parsed data--i've always had a hard time getting reliable data using
108 // a parsing method
109
110 Transceiver.GetStruct(&ebyte_msg, sizeof(ebyte_msg));
111
112 // dump out what was just received
113 Serial.print("ID: "); Serial.println(ebyte_msg.count);
114 Serial.print("Msg: "); Serial.println(ebyte_msg.msg);
115 //Serial.println("***");
116 // if you got data, update the checker
117 Last = millis();
118
119 //ejecutar comandos segun llegue orden por radio
120
121 executeCommand(ebyte_msg.msg);
122
123 }
124 else {
125 // if the time checker is over some prescribed amount
126 // let the user know there is no incoming data
127 if ((millis() - Last) > 1000) {
128 //Serial.println("Searching: ");
129 Last = millis();
130 }
131
132 }
133 }
134
135 void executeCommand(String data) {
136 String static msg;
137
138 if (data.equals("status")) {
139
140

```

```

141     int waterLevel = checkWaterLevel();
142
143     int feederLevel = TOTAL_REFILL - num_refill;
144
145     String status;
146
147     if (waterLevel > 25 && feederLevel > 5) {
148         status = "ok";
149     } else if (waterLevel <= 25 || feederLevel <= 5) {
150         status = "ko";
151     } else {
152         status = "unknown";
153     }
154
155     msg = "ack;status;" + status;
156
157     sendMessage(msg);
158 }
159
160 if (data.equals("ping")) {
161     msg = "ack;pong";
162
163     sendMessage(msg);
164 }
165
166 if (data.equals("reset_comedero")) {
167     num_refill = 0;
168
169     msg = "ack;reset_comedero";
170
171     sendMessage(msg);
172 }
173
174 if (data.equals("rellenar_comedero")) {
175     String feeder = rellenarComedero();
176
177     msg = "ack;feeder_refill;" + feeder + ";num_refill;" + num_refill;
178
179     sendMessage(msg);
180 }
181
182 if (data.equals("rellenar_bebedero")) {
183     String refill = rellenarBebedero();
184
185     msg = "ack;water_refill;" + refill;
186
187     sendMessage(msg);
188 }
189
190 if (data.equals("check_levels")) {
191     int waterLevel = checkWaterLevel();
192
193     msg = "ack;water;" + String(waterLevel);

```

```

195     sendMessage(msg);
196 }
197
198 if (data.equals("update_oled")) {
199     Serial.println("to do");
200 }
201
202 }
203 //Datos recibidos por serial, por eso printeo desde Serial.
204 void sendMessage(String data) {
205     Serial.println("Sending data: " + data);
206
207     // Send to radio
208     data.toCharArray(ebyte_msg.msg, data.length() + 1);
209     ebyte_msg.count++;
210
211     Transceiver.SendStruct(&ebyte_msg, sizeof(ebyte_msg));
212     Serial.println("[Refugio] Enviado. ID: " + String(ebyte_msg.count));
213     //Serial.println("****");
214     delay(1000);
215 }
216
217 int checkWaterLevel() {
218     boolean sw1 = digitalRead(LEVEL_SW1);
219     boolean sw2 = digitalRead(LEVEL_SW2);
220
221     if (sw1) {
222         return 100; // 100 %
223     } else if (sw2) {
224         return 50; // 50 %
225     } else {
226         return 25; // < 25 %
227     }
228 }
229
230
231 String rellenarBebedero() {
232     int waterLevel = checkWaterLevel();
233
234     if (waterLevel > 25) {
235         //accionar rele para activar bomba
236         Serial.println("Rellenando bebedero...");
237
238         digitalWrite(PUMP, HIGH);
239         delay(REFILL_TIME);
240         digitalWrite(PUMP, LOW);
241
242         //delay(500);
243
244         Serial.println("Bebedero lleno");
245
246         return "ok";
247     } else {
248         return "ko";

```

```

249 }
250 }
251 }
252 }
253 String rellenarComedero() {
254
255     if (num_refill >= 0 && num_refill <= TOTAL_REFILL) {
256
257         Serial.println("Rellenando comedero...");
258         Serial.println("Contador de llenado: " + String(num_refill));
259         Serial.println("Ciclos restantes de llenado: " + String((TOTAL_REFILL-
260             num_refill)));
261
262         for (int i = 0; i <= NUM_CICLO; i++) { //ajustar en defines NUM_CICLO si
263             //queremos un ciclo de llenado mas largo o mas corto
264             //Serial.println(i);
265             servol.write(0);
266             delay(SPEED_SERVO);
267             servol.write(105);
268             delay(SPEED_SERVO);
269
270             // Posicion estatica
271             servol.write(90);
272             delay(SPEED_SERVO); // ajustar en define
273
274             num_refill++;
275             Serial.println("Ciclo " + String(num_refill) + "/20 de llenado finalizado.");
276
277             return("ok");
278         } else{
279             return("ko");
280         }
281     }
282 }
```

**Protocolo**, para terminal de estación del campo: *gw-protocolo.ino*

```
1 #include <SoftwareSerial.h>
2 #include "EByte.h"
3 #include <avr/io.h>
4
5 #define PIN_RX 2
6 #define PIN_TX 3
7 #define PIN_M0 4
8 #define PIN_M1 5
9 #define PIN_AX 6
10
11
12 #define LORA_CHANNEL 5
13
14 struct ebyte_struct {
15     unsigned long count;
16     char msg[64];
17 };
18 ebyte_struct ebyte_msg;
19
20 unsigned long Last;
21
22 SoftwareSerial ESerial(PIN_RX, PIN_TX);
23 EBYTE Transceiver(&ESerial, PIN_M0, PIN_M1, PIN_AX);
24
25 void setup() {
26     Serial.begin(115200);
27
28     //different baudrate for nano-E32 serial comm (recommend value of 9600)
29     ESerial.begin(9600);
30     Serial.println("Starting GW");
31
32     // this init will set the pinModes for you
33     Transceiver.init();
34
35     // all these calls are optional
36
37     // Serial.println(Transceiver.GetAirDataRate());
38     // Serial.println(Transceiver.GetChannel());
39
40     Transceiver.SetMode(MODE_NORMAL);
41
42     Transceiver.SetUARTBaudRate(UDR_9600);
43     Transceiver.SetAirDataRate(ADR_2400);
44     Transceiver.SetAddressH(1);
45     Transceiver.SetAddressL(0);
46
47     Transceiver.SetChannel(LORA_CHANNEL);
48     Transceiver.SetTransmitPower(OPT_TP21); //change transceiver tx-rx power
49     Transceiver.SetPullupMode(0);
50     Transceiver.SetFECMode(1);
51
```

```

52 // save the parameters to the unit,
53 Transceiver.SaveParameters(TEMPORARY);
54
55 // you can print all parameters and is good for debugging
56 // if your units will not communicate, print the parameters
57 // for both sender and receiver and make sure air rates, channel
58 // and address is the same
59 Transceiver.PrintParameters();
60
61 ebyte_msg.count = 0;
62
63 //Reduced SRAM cause arduino nano -- optimize sram on string shown on Serial.
64 //println using "F" function
65 //Example of F(): https://techexplorations.com/guides/arduino/programming/f-macro/
66 //https://learn.adafruit.com/memories-of-an-arduino/optimizing-sram
67
68 Serial.println(F("Resumen de comandos aptos: status, ping, llenar_bebedero,"));
69 Serial.println(F("rellenar_comedero, reset_comedero, check_levels y update_oled"));
70
71 Serial.println(F("Formato de los mensajes de respuesta: ack;field1;var1"));
72
73 }
74
75 void loop() {
76
77 // arduino serial: write on serial interface and send msg via LoRa to refugio
78 // station
79
80 if (Serial.available() > 0) {
81     String static dat_rec;
82
83     dat_rec = Serial.readString();
84     dat_rec.trim(); //quita espacio en blanco
85
86     Serial.println("[GW] Comando recibido: " + dat_rec);
87
88     sendMessage(dat_rec); //dat_rec will be sent via radio/LoRa
89 }
90
91 // if the transceiver serial is available, proces incoming data
92 // you can also use Transceiver.available()
93
94 if (ESerial.available()) {
95
96     // i highly suggest you send data using structures and not
97     // a parsed data--i've always had a hard time getting reliable data using
98     // a parsing method
99
100    Transceiver.GetStruct(&ebyte_msg, sizeof(ebyte_msg));
101
102    // dump out what was just received

```

```

102     Serial.print("[Refugio] ID: ");
103     Serial.println(ebyte_msg.count);
104     //Serial.print("msg: ");
105     Serial.print("[Refugio] Msg: ");
106     Serial.println(ebyte_msg.msg);
107     //Serial.println("****");
108     // if you got data, update the checker
109     Last = millis();
110
111 }
112 else {
113     // if the time checker is over some prescribed amount
114     // let the user know there is no incoming data
115     if ((millis() - Last) > 1000) {
116         //Serial.println("Searching: ");
117         Last = millis();
118     }
119 }
120 }
121 }
122
123 //Data received by pc-nano serial comm -- will be transmitted by our gw LoRa
124 void sendMessage(String data) {
125     Serial.println("[GW] Enviando por LoRa: " + data);
126
127     // Send to radio
128     data.toCharArray(ebyte_msg.msg, data.length()+1);
129     ebyte_msg.count++;
130
131     Transceiver.SendStruct(&ebyte_msg, sizeof(ebyte_msg));
132     Serial.println("[GW] Msg enviado. ID: " + String(ebyte_msg.count));
133     //Serial.println("****");
134     delay(1000);
135 }
136

```

### *Aplicación web: app.py*

```

1 import engineio
2 from flask import Flask, render_template
3 from flask_socketio import SocketIO
4 from flask_serial import Serial
5 from datetime import datetime
6
7 import json
8 import socketio
9 import os
10
11 app = Flask(__name__)
12
13 ## Configuration Serial GW
14 serial_port = '/dev/ttyUSB0'                      # Modificar aqui el serial del

```

```

    Arduino, lo dice el IDE
15 app.config['SERIAL_PORT'] = serial_port
16 app.config['SERIAL_TIMEOUT'] = 0.1
17 app.config['SERIAL_BAUDRATE'] = 115200
18 app.config['SERIAL_BYTESIZE'] = 8
19 app.config['SERIAL_PARITY'] = 'N'
20 app.config['SERIAL_STOPBITS'] = 1
21
22 socketio = SocketIO(app, async_mode="gevent", ping_timeout=5, logger=False,
23     engineio_logger=False)
24 ser = Serial(app)
25
26 # Endpoints
27 #
28
29 # Endpoint home page
30 @app.route("/")
31 def index():
32     # Comprobamos que el serial existe
33     if os.path.exists(serial_port):
34         serial_exists = True
35     else:
36         serial_exists = False
37
38     return render_template("index.html", serial_port=serial_port, serial_exists=
39         serial_exists)
40
41 # Endpoint documentation page
42 @app.route("/doc")
43 def doc():
44     return render_template("doc.html")
45
46 # SocketIO
47 #
48
49 # Handler SocketIO [messages of web]
50 @socketio.on('send')
51 def handle_send(json_str):
52     data = json.loads(json_str) ["message"]
53     print("Boton pulsado: {}".format(data))
54
55     # Enviamos por serial
56     try:
57         ser.on_send(data)
58     except Exception as ex:
59         print("Error on serial: {}".format(ex))
60
61     # Enviamos en el socket 'receive_message' [hacer en recepcion serial]
62     now = datetime.now()
63     msg = "{} >> Comando a gateway: {}".format(now.strftime("%H: %M: %S"), data)
64     socketio.emit("receive_message", data={"message":str(msg)})
65

```

```

66 #
67 #     Flask-Serial
68 #
69
70 # Handler incoming serial messages
71 @ser.on_message()
72 def handle_message(msg):
73     try:
74         print("receive message: {}".format(msg))
75         now = datetime.now()
76         msg = "{} >> Respuesta: {}".format(now.strftime("%H:%M:%S"), msg.decode("utf-8")
77                                         ).strip('\n'))
78         print("Send socket msg: {}".format(msg))
79         socketio.emit("receive_message", data={"message": str(msg)})
80     except Exception as ex:
81         print(ex)
82
83 # Log thread serial
84 @ser.on_log()
85 def handle_logging(level, info):
86     print(level, info)
87
88 if __name__ == '__main__':
89     socketio.run(app, debug=False, log_output=True)

```

### *Página principal de la web: index.html*

```

1 <!doctype html>
2 <html lang="es">
3     <head>
4         <!-- Required meta tags -->
5         <meta charset="utf-8">
6         <meta name="viewport" content="width=device-width, initial-scale=1">
7         <link rel="icon" type="image/png" href="static/favicon.ico">
8
9         <!-- Bootstrap CSS -->
10        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpooCOnLASjC" crossorigin="anonymous">
11        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css" integrity="sha512-iBBXm8fW90+nuLcSKlbmrPcLa0OT92x01BIsZ+ywDWZCvqsWgccV3gFoRBv0z+8dLJgyAHIhR35VZc2oM/gI1w==" crossorigin="anonymous" referrerPolicy="no-referrer" />
12        <script src="https://cdn.socket.io/3.1.3/socket.io.min.js" integrity="sha384-cPwlPLvBTa3sKAgddT6krw0cJat7egBga3DJepJyrLl4Q9/5WLra3rrnMcyTyOnh" crossorigin="anonymous"></script>
13        <script src="https://code.jquery.com/jquery-3.6.0.min.js" integrity="sha256-/xUj+3OJU5yExlq6GSYGSKh7tPXikynS7ogEvDej/m4=" crossorigin="anonymous"></script>
14
15        <title>Open Pet Feeder</title>
16    </head>
17    <body>

```

```

18     <h1 style="text-align: center; margin-top: 2%;"><a href="/" style="text-decoration: none; color: inherit;"><i class="fas fa-paw"></i> Open Pet Feeder</a></h1>
19
20
21     <div class="container" style="margin-top: 3%;>
22     { % if serial_exists == True %
23         <div class="alert alert-success" role="alert" style="margin-bottom: 3%;">
24             <i class="fas fa-laptop-house"></i> Conectado a GW utilizando el serial: {{serial_port}}!
25         </div>
26     { % else %
27         <div class="alert alert-danger" role="alert" style="margin-bottom: 3%;">
28             <i class="fas fa-laptop-house"></i> No existe el serial '{{serial_port}}'.
29             Configura el serial y reinicia el programa.
30         </div>
31     { % endif %
32
33         <div class="row" style="text-align: center;">
34             <div class="col">
35                 <button type="button" class="btn btn-dark mt-1" id="send_ping" value="ping"><i class="fas fa-comments"></i> Comprobar actividad</button>
36             </div>
37             <div class="col">
38                 <button type="button" class="btn btn-dark mt-1" id="send_status" value="status"><i class="fas fa-clipboard-list"></i> Comprobar estado</button>
39             </div>
40             <div class="col">
41                 <button type="button" class="btn btn-dark mt-1" id="send_check_levels" value="check_levels"> <i class="far fa-check-circle"></i> Comprobar niveles</button>
42             </div>
43             <div class="col">
44                 <button type="button" class="btn btn-dark mt-1" id="send_comedero" value="rellenar_comedero"><i class="fas fa-utensils"></i> Rellenar comedero</button>
45             </div>
46             <div class="col">
47                 <button type="button" class="btn btn-dark mt-1" id="send_reset_comedero" value="reset_comedero"><i class="fas fa-eraser"></i> Reiniciar comedero</button>
48             </div>
49             <div class="col">
50                 <button type="button" class="btn btn-dark mt-1" id="send_bebedero" value="rellenar_bebedero"><i class="fas fa-faucet"></i> Rellenar bebedero</button>
51             </div>
52
53             <div class="col">
54                 <button type="button" class="btn btn-dark mt-1" id="send_oled" value="update_oled"><i class="fas fa-desktop"></i> Actualizar pantalla</button>
55             </div>
56         </div>
57
58     <div class="container" style="margin-top: 3%;">
59         <div class="console">

```

```

60      <label class="form-label" for="receive"> <h4> <b>Mensajes recibidos:</b> </
61      h4> </label>
62      <textarea readonly class="form-control" name="receive" id="receive" rows="8
63      " style="width: 100%; height: auto;"></textarea>
64      </div>
65  </div>
66
67  <!-- Footer -->
68  <div class="footer fixed-bottom mt-4 py-4">
69      <div class="text-center p-2" style="background-color: rgba(0, 0, 0, 0.05);
70      margin-bottom: -1%;">
71          C 2021 <i class="fab fa-github" style="margin-left: 1%;"></i> Github:
72          <a class="text-reset fw-bold" href="https://github.com/ChiaFranfer">
73              ChiaFranfer</a>
74          <i class="fas fa-scroll" style="margin-left: 1%;"></i> Documentacion: <a
75          class="text-reset fw-bold" href="{{url_for('doc')}}"> Ir a documentacion</a>
76          <i class="fas fa-dog" style="margin-left: 3%;"></i> <i class="fas fa-cat"><
77          /i> <i class="fas fa-heart"></i>
78      </div>
79  </div>
80
81  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.
82  bundle.min.js" integrity="sha384-MrcW6ZMFYlzcLA8Nl+
83  NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM" crossorigin="anonymous"></script
84  >
85  <script type="text/javascript" charset="utf-8">
86      $(document).ready(function() {
87          var socket = io.connect('http://' + document.domain + ':' + location.port);
88
89          $('#send_status').on('click', function() {
90              var message = $(this)[0].value;
91              console.log('Enviando: ' + message);
92              var data = '{' + '"message": "' + message + '"';
93              socket.emit('send', data=data);
94          });
95
96          $('#send_ping').on('click', function() {
97              var message = $(this)[0].value;
98              console.log('Enviando: ' + message);
99              var data = '{' + '"message": "' + message + '"';
100             socket.emit('send', data=data);
101         });
102
103         $('#send_reset_comedero').on('click', function() {
104             var message = $(this)[0].value;
105             console.log('Enviando: ' + message);
106             var data = '{' + '"message": "' + message + '"';
107             socket.emit('send', data=data);
108         });
109
110         $('#send_comedero').on('click', function() {
111             var message = $(this)[0].value;
112             console.log('Enviando: ' + message);
113             var data = '{' + '"message": "' + message + '"';
114             socket.emit('send', data=data);
115         });
116     });
117 
```

```

105     socket.emit('send', data=data);
106   });
107
108   $('#send_bebedero').on('click', function() {
109     var message = $(this)[0].value;
110     console.log('Enviando: ' + message);
111     var data = '{' + '"message": "' + message + '"';
112     socket.emit('send', data=data);
113   });
114
115   $('#send_check_levels').on('click', function() {
116     var message = $(this)[0].value;
117     console.log('Enviando: ' + message);
118     var data = '{' + '"message": "' + message + '"';
119     socket.emit('send', data=data);
120   });
121
122   $('#send_oled').on('click', function() {
123     var message = $(this)[0].value;
124     console.log('Enviando: ' + message);
125     var data = '{' + '"message": "' + message + '"';
126     socket.emit('send', data=data);
127   });
128
129   socket.on('receive_message', function(data) {
130     console.log(data);
131     var text = data['message'];
132     var $textarea = $('#receive');
133     $textarea.val($textarea.val() + text + '\n');
134     $textarea.scrollTop($textarea[0].scrollHeight);
135   })
136 });
137
138 </script>
139 </body>
140 </html>
141

```

## **Anexo III**

[Insertar tabla excel del presupuesto].