Grau en Enginyeria en Tecnologies Industrials

Treball Final de Grau

Disseny i implementació d'un sistema de comunicacions WiFi per a una xarxa de vehicles autònoms

MEMÒRIA

Autor: Antoni Riera Seguí

Directors: Arnau Dòria-Cerezo

Víctor Repecho del Corral

Convocatòria: Juny 2016

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum

En aquest treball es dissenya i implementa un sistema de comunicacions WiFi per a una xarxa de vehicles autònoms. Els vehicles es connecten a la mateixa xarxa sense fils, cosa que permet l'intercanvi d'informació entre ells i també ser monitoritzats i controlats des d'un PC.

Els vehicles estan governats per un microcontrolador ARM. S'utilitza el dispositiu ESP8266, comercialitzat per l'empresa xinesa Expressif Systems, per a dotar-los de connectivitat WiFi. S'ha dissenyat un software per a microcontroladors que permet el govern del dispositiu WiFi sense interferir en cap cas amb l'execució d'altres rutines prioritàries per part del microcontrolador. S'ha dissenyat un altre software per a PC que permet la monitorització de l'estat dels vehicles connectats a la xarxa, l'enviament d'ordres, la imposició remota de consignes i la modificació de paràmetres. El sistema de comunicacions no es limita a vehicles autònoms si no que pot ser implementat en qualsevol sistema integrat.

El software és d'implementació fàcil i transparent, i de funcionalitat flexible. S'inclou una guia d'usuari, centrada en la instal·lació i posada a punt del sistema de comunicacions, la seva parametrització i ampliació de funcionalitat.

Sumari

1. Glossari

ARM Família d'arquitectures per a microprocessadors de tipus RISC (Reduced Instruction Set Computer, computador amb conjunt d'instruccions reduïdes), desenvolupat per la companyia homònima. És l'arquitectura més usada en sistemes integrats pel seu cost reduït i baix consum energètic.

API Conjunt de rutines que ofereix una biblioteca per a ser utilitzades per un altre software. Acrònim de Application Programming Interface (Interfície de programació d'aplicacions)

ASCII Estàndard de codificació de caràcters utilitzat per gairebé tots els sistemes informàtics actuals a l'hora de representar textos i comunicar-se amb dispositius que treballen amb text. Utilitza 7 bits per a representar 128 caràcters (imprimibles o de control). Acrònim de American Standard Code for Information Interchange.

Buffer Memòria intermèdia utilitzada per a emmagatzemar dades temporalment quan, en una transmissió, els processos emissor i receptor no treballen de forma síncrona.

CPU Component d'un ordinador encarregat d'interpretar les instruccions que conformen els programes i executar les operacions que especifiquen (aritmètiques, lògiques, entrada/sortida, etc). Abreviat de Central Processing Unit (Unitat Central de Processament).

DMA Característica present en alguns sistemes informàtics que permet establir un canal de comunicació directe entre la memòria principal i un perifèric, de manera que aquest pugui efectuar operacions de lectura i escriptura sense necessitat d'executar instruccions en la CPU. Abreviat de Direct Memory Access (Accés Directe a Memòria)

ESP8266 Dispositiu integrat que incorpora una CPU, memòria RAM, una antena WiFi i un port UART (sèrie). Té la capacitat de connectar-se a xarxes WiFi mitjançant el protocol TCP/IP i efectuar algunes operacions senzilles, incloent establir connexions amb varis clients i enviar i rebre paquets de xarxa. Es controla mitjançant comandes que s'envien pel canal sèrie de comunicació. És un dispositiu de molt baix cost i de mida molt reduïda, desenvolupat per la companyia Expressif.

Firmware Software que s'encarrega del control de dispositius electrònics al més baix nivell.

HAL Capa de software intermèdia que permet programar aplicacions vàlides per a diferents famílies de dispositius. És l'acrònim de Hardware Abstraction Layer (Capa d'Abstracció de Hardware).

IP Protocol de comunicacions digital que permet establir xarxes amb múltiples dispositius i transmetre paquets entre els dispositius connectats a la xarxa. Cada dispositiu s'identifica amb una adreça (adreça IP). El protocol comprèn les normes per a dirigir cada paquet fins a l'adreça IP destinatària. És l'abreviatura d'Internet Protocol (Protocol d'Internet).

Interrupció En l'àmbit de l'electrònica, una interrupció és una senyal emesa per hardware o software a la CPU que indica la necessitat d'interrompre el flux actual d'execució. Si es considera que la interrupció és prioritària a l'execució actual, la CPU pot posar-la en pausa (desant l'estat actual dels registres) i executar codi més important (anomenat rutina de servei)

Microprocessador Unitat Central de Processament (CPU) materialitzada en un únic circuit integrat.

Python Llenguatge de programació orientat a objectes d'alt nivell. És un llenguatge interpretat (no es compila el codi sinó que es processa durant l'execució). És usat àmpliament en l'àmbit científic per la facilitat d'ús, la llegibilitat del codi i les nombroses llibreries disponibles.

Pyqtgraph Llibreria que permet dibuixar distints tipus de gràfics i actualitzar-los ràpidament amb Python.

Router Dispositiu que s'encarrega de redistribuir paquets de xarxa per ta que arribin al seu destinatari.

Sistema integrat: (Embedded System) Dispositiu informàtic que conté la majoria dels components integrats en una mateixa placa base i que es programa per a executar alguna funció dedicada. Els sistemes integrats normalment s'enfronten a tasques de processament en temps real.

STM32F4 Família de microcontroladors amb processador d'arquitectura ARM Cortex-M produïts per l'empresa ST Microelectronics.

TCP Protocol de comunicacions que permet establir connexions entre dos dispositius situats a la mateixa xarxa. Mitjançant aquestes connexions es poden enviar dades amb validació de recepció. S'executa sobre el protocol IP. Abreviat de Transmission Control Protocol (Protocol de control de la transmissió).

Temps quasi-real Nom alternatiu per a referir-se a la computació a temps real quan es vol remarcar la presència d'un retard de propagació. Un sistema en temps real és aquell que interacciona als els estímuls externs en un període de temps establert i reduït.

UART Dispositiu que s'encarrega de la comunicació sèrie, transmetent i rebent els bits individuals de forma seqüencial. És l'acrònim de Universal Asynchronous Receiver-Transmitter (Transmissor Receptor Asíncron Universal).

UDP Protocol de comunicacions que permet l'enviament de paquets de dades entre dos dispositius situats a una mateixa xarxa, sense necessitat d'establir una connexió entre ells. Els paquets s'envien de forma unidireccional, no hi ha confirmació ni validació de la seva recepció. S'executa sobre el protocol IP. Abreviat de User Datagram Protocol.

WiFi Tecnologia que permet la connexió sense fils de dispositius electrònics. Els dispositius es connecten a una xarxa generada per un dispositiu mestre (punt d'accés). El protocol es basa en diferents estàndards IEEE 802.11.

2. Prefaci

Aquest treball final de grau es desenvolupa conjuntament amb el treball d'un altre estudiant de grau que inclou la construcció d'un vehicle seguidor de línia. Disposar d'aquests vehicles, juntament amb un sistema de comunicacions que permeti l'intercanvi d'informació entre ells, és la primera passa d'un projecte de recerca orientat a l'aplicació de tècniques de control en diferents escenaris d'interacció entre vehicles autònoms).

El sistema de comunicacions s'ha desenvolupat de forma independent al vehicle, i el software produït pot ser implementat en qualsevol altra aplicació, tot i que s'ha pres aquest com a referència a l'hora de definir la funcionalitat. En etapes intermèdies del desenvolupament s'han mesclat els dos projectes i s'ha verificat el correcte funcionament del vehicle amb el software de comunicacions instal·lat.

El vehicle seguidor de línia està format pels següents components (Prats Martinho, 2016):

- Placa de desenvolupament STM32F4 Discovery
- Mòdul WiFi ESP8266 (Expressif Systems)
- 2 rodes amb motors DC bidireccionals i codificadors rotatius
- Controlador del motor L298N
- Sensor ultrasònic HC-SR04

Una fotografia del vehicle totalment muntat es mostra a la Figura 2-1.

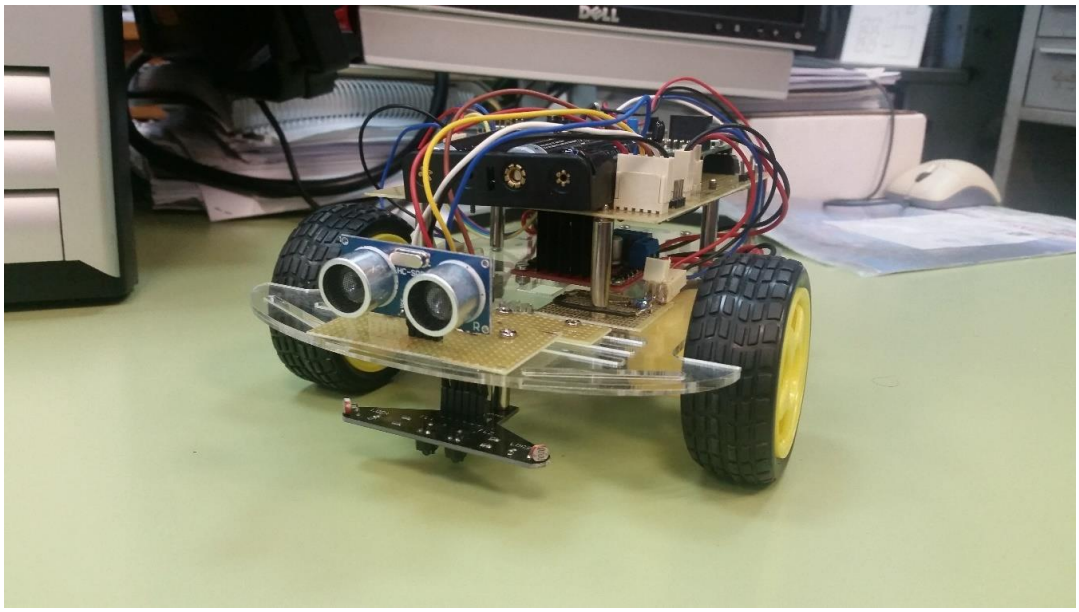

Figura 2-1. Vehicle seguidor de línia usat com a referència. (Prats Martinho, 2016)

3. Introducció

L'objectiu principal d'aquest treball és implementar un sistema de comunicacions WiFi per a vehicles autònoms, prenent com a base el vehicle seguidor de línia presentat al prefaci. Els vehicles estan governats per un microcontrolador ARM.

S'utilitza el mòdul WiFi ESP8266 per a dotar de connectivitat WiFi als vehicles. Aquest és un dispositiu de molt baix cost que es governa mitjançant ordres senzilles enviades a través d'un port sèrie. Per tant, pot ser implementat en multitud de dispositius. En aquest cas, el mateix microcontrolador que s'encarrega del control dels vehicles serà l'encarregat de governar el mòdul WiFi i gestionar les comunicacions. Cal que aquesta feina no interfereixi amb la tasca principal del microcontrolador, que en aquest cas és el control del vehicle.

La finalitat d'aquest sistema de comunicacions és permetre l'intercanvi d'informació entre diversos vehicles, o entre els vehicles i un ordinador d'escriptori. L'ordinador és un element opcional del sistema que ha de servir per monitoritzar l'estat de tots els vehicles, transmetre ordres, modificar paràmetres i visualitzar dades en temps real. En general, les característiques s'han d'acabar de definir després d'analitzar les possibilitats del mòdul WiFi ESP8266.

A la Figura 3-1 es representen els elements que formaran part del sistema de comunicacions, sense definir encara de forma específica l'estratègia de connexió.

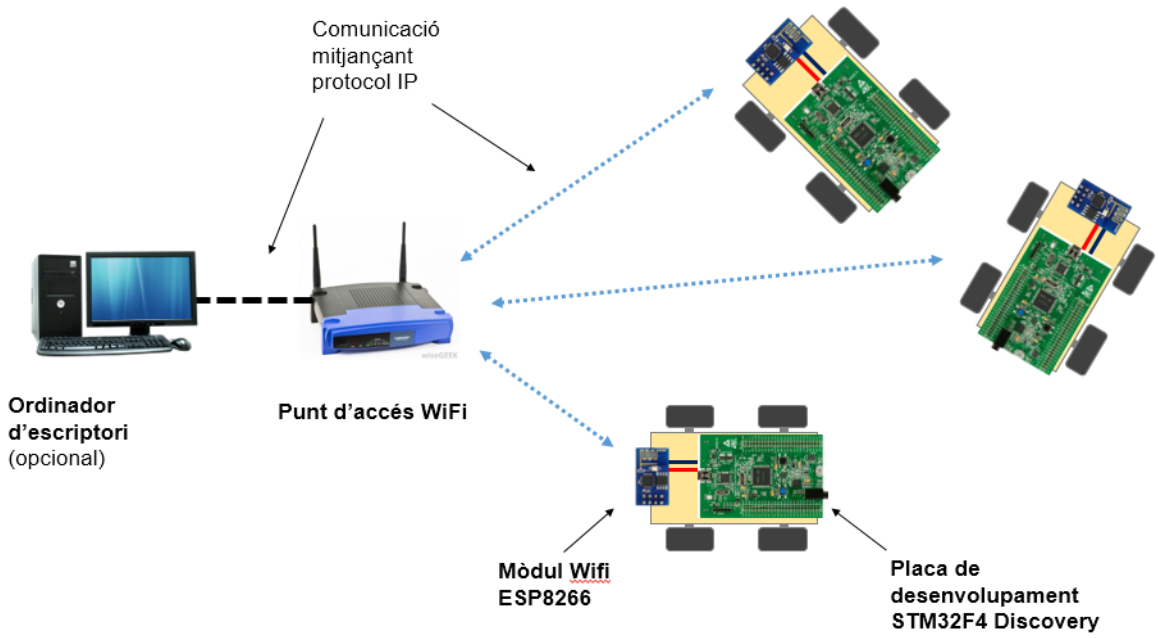

Figura 3-1. Elements que formen part del sistema de comunicacions.

El sistema de comunicacions dissenyat s'ha de poder adaptar de forma transparent a qualsevol tasca prioritària que executin els vehicles. La seva funcionalitat ha de ser fàcil de modificar i/o expandir, i s'han de documentar els procediments necessaris.

3.1. Abast del projecte i objectius específics

Aquest projecte comprèn la implementació d'un sistema de comunicacions per a vehicles autònoms construïts sobre una placa de desenvolupament STM32F4 Discovery, amb un mòdul WiFi ESP8266. La tria d'aquests dispositius està prefixada per requeriments del projecte i no entra dins l'abast d'aquest treball l'anàlisi de les diferents alternatives disponibles en el mercat i la selecció d'una d'elles. Sí que entra dins l'abast fer un anàlisi exhaustiu del dispositiu WiFi i determinar quines característiques pot tenir el sistema de comunicacions a partir de les possibilitats que ofereixi.

Els objectius específics a assolir són:

- Estudiar les possibilitats que ofereix el mòdul WiFi ESP8266 i el seu rendiment. Determinar en quins escenaris pot resultar útil i quines aplicacions se li poden donar.
- Determinar, segons les possibilitats del mòdul WiFi, les característiques del sistema de comunicacions i la funcionalitat que pot oferir.
- Dissenyar un software per a sistemes integrats (prenent com a referent la placa de desenvolupament STM32F4 Discovery) que permeti governar el mòdul WiFi ESP8266 i gestionar les comunicacions sense interferir de cap manera amb el programa principal, que es considera totalment prioritari (lectura de sensors, control, etc). La integració s'ha de poder fer de forma transparent, sense haver d'adaptar ni el software principal ni el de comunicacions.
- Definir un codi per a l'intercanvi d'informació entre els dispositius, segons les característiques establertes.
- Dissenyar una aplicació per a PC que permeti monitoritzar l'estat de tots els vehicles, transmetre ordres, modificar paràmetres i visualitzar dades en temps real.
- Dissenyar un sistema de comunicacions que pugui ser modificable i expansible, i fàcil d'adaptar a diferents aplicacions i entorns. Documentar els procediments en forma de manual d'usuari.

4. Descripció i anàlisi dels components. Limitacions.

El vehicle seguidor de línia està construït per un microcontrolador STM32F4 (en una placa de desenvolupament STM32F4 Discovery) muntat sobre un xassís, juntament amb diversos sensors (òptics i de proximitat) i actuadors. Aquest s'encarrega de processar les dades rebudes pels sensors, de calcular l'acció de control establerta i de generar la senyal PWM necessària per a controlar els motors de les rodes. La seva funcionalitat serà ampliada en un futur.

S'utilitzarà un mòdul WiFi ESP8266 per a poder connectar els vehicles a una xarxa sense fils i permetre l'intercanvi d'informació entre diversos vehicles o entre un vehicle i un PC. Com es veurà, l'ús d'aquest dispositiu implica utilitzar el protocol de comunicacions IP, cosa que presenta diversos avantatges i desavantatges.

Resulta important recordar que la tria d'aquests dispositius està prefixada des de l'inici del projecte. L'anàlisi de les diferents alternatives disponibles en el mercat i la selecció d'una d'elles queda fora de l'abast d'aquest treball i és per això que no s'ha inclòs en aquesta memòria. El microcontrolador STM32F4 és la base del vehicle seguidor de línia al qual es vol implementar un sistema de comunicacions. El dispositiu WiFi ESP8266, tot i ser un model d'aparició molt recent, s'ha triat pel seu baix cost i per la seva gran acollida i potencial. Addicionalment, un dels objectius específics d'aquest projecte és estudiar les possibilitats i el rendiment d'aquest dispositiu.

La major part de la feina necessària per a portar a terme aquest projecte és programar el microcontrolador per tal que es comuniqui amb el mòdul WiFi sense interferir de cap manera amb el programa principal (que s'encarrega del control i d'altres tasques). Tot i que en aquesta memòria es detallaran els procediments per al microcontrolador STM32F4, el software creat pot ser adaptat fàcilment a qualsevol altre dispositiu per al qual s'hi pugui compilar codi C. A l'Annex A.2 (guia d'instal·lació) s'indiquen els requisits que ha de complir el microcontrolador i els passos que s'han de seguir.

4.1. Descripció i anàlisi del dispositiu WiFi ESP8266

El mòdul ESP8266 és un dispositiu integrat produït per la companyia xinesa Expressif Systems. Es presenta en diferents configuracions, les quals incorporen totes una CPU, memòria RAM, una antena WiFi i un port UART (sèrie) en unes dimensions molt reduïdes. Té la capacitat de connectar-se a xarxes WiFi mitjançant el protocol IP i efectuar algunes

operacions senzilles, incloent establir connexions amb varis clients i enviar i rebre paquets de xarxa. La connexió es fa a xarxes sense fils que segueixen l'estàndard IEEE802.11b, g i n. Es controla mitjançant comandes que s'envien pel canal sèrie de comunicació.

El dispositiu es va posar a la venda a finals de l'any 2014. Tot i que inicialment el software (firmware) que el governa presentava un rendiment molt dolent, el dispositiu en sí es va fer molt famós a causa del seu baix preu (a partir d'uns 4€). Fins a mitjans de 2015 la documentació oficial només estava disponible en xinès i no era gaire exhaustiva. (Brian Benchoff, 2015). Avui (2016) hi ha molta documentació disponible en anglès, oficial i no oficial. No obstant això gairebé tota aquesta documentació està enfocada a la implementació de solucions sota la plataforma Arduino, la qual incorpora amb el seu SDK oficial una llibreria per a comunicar-se amb mòduls ESP8266.

Es comercialitzen múltiples variants d'aquest dispositiu, però a priori totes són compatibles entre sí. Aquest projecte s'ha portat a terme amb la versió ESP-1, que és la més usada per a aplicacions de caire general. A la Figura 4-1 es presenta una fotografia amb l'aspecte físic d'aquesta variant.

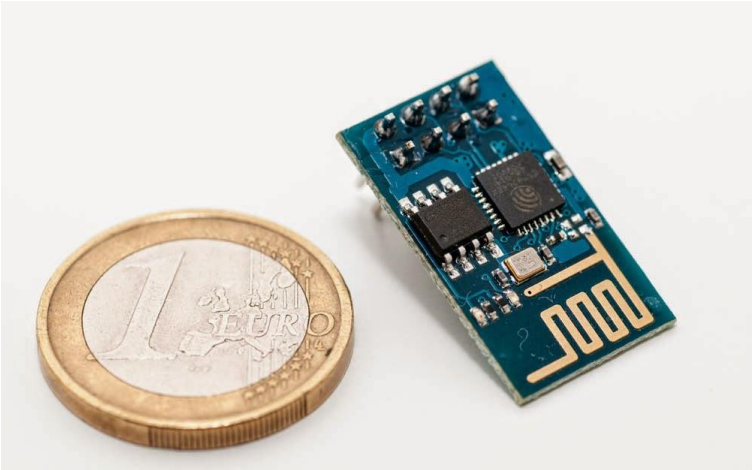

Figura 4-1. Dispositiu WiFi ESP8266, variant ESP-1. (Tacks, V., 2014)

4.1.1. Descripció tècnica

El mòdul ESP8266 incorpora una CPU, memòria RAM, una antena WiFi i un port sèrie. A la Taula 4-1 es presenten breument les especificacions mínimes d'aquest mòdul. Per defecte inclou un firmware que permet detectar i connectar-se a xarxes WiFi, establir connexions TCP i UDP amb altres dispositius de la xarxa (Internet inclòs) i enviar i rebre paquets d'aquest tipus. El control es fa mitjançant comandes que s'envien pel port sèrie. Aquest tipus de comunicació és usat àmpliament en l'àmbit de les telecomunicacions, tot i que no està estandarditzat.

S'utilitza una variant del *Conjunt d'Ordres Hayes*, també anomenades *comandes AT (introduït a l'apartat 4.2.2)*.

Especificacions mínimes ESP8266	
CPU	CPU RISC de 32 bits
Memòria RAM	64KB per a instruccions 96KB per a dades
Memòria Flash	512 KB o 1024 KiB. *(Amb 512KB no es poden instal·lar els firmwares moderns)*
Connectivitat Wifi	Controlador propi • Compatible amb els estàndards IEEE 802.11 b/g/n. • Compatible amb els estàndards de xifratge WEP, WPA, WPA2.
Entrada i Sortida	16 pins GPIO Controlador UART ADC de 10 bits

Taula 4-1. Especificacions mínimes que han de complir totes les variants de mòduls WiFi ESP8266. (Expressif Systems, 2015)

El dispositiu treballa amb una tensió de 3,3V. Tensions superiors als 4,5V el deixen inutilitzat instantàniament, per tant cal extremar les precaucions durant el desenvolupament. Les connexions no estan indicades al PCB, a la Figura 4-2 es mostra la llegenda de cada pin. S'alimenta mitjançant els pins Vcc i GND, i la comunicació amb el microcontrolador es fa a través dels pins Tx, Rx i GND mitjançant UART (sèrie).

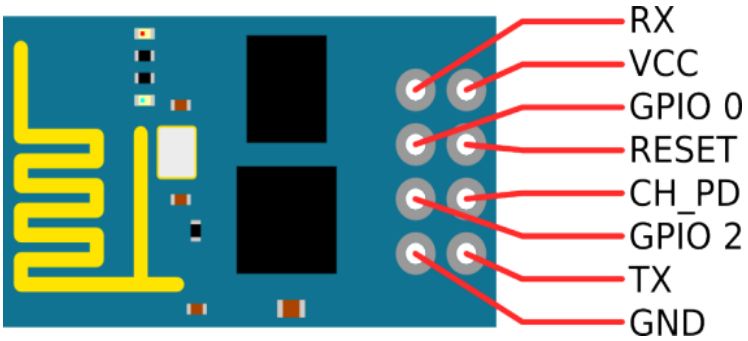

Figura 4-2. Pins de connexió del mòdul WiFi ESP8266, configuració ESP-1. Cal alimentar-lo amb 3,3V. Els pins Tx i Rx (i GND) corresponen a la comunicació UART. (Expressif Systems, 2015)

Les especificacions deixen oberta la possibilitat de programar aplicacions senzilles directament en el mòdul WiFi. Existeixen diversos SDK no oficials que permeten incorporar-

hi rutines personalitzades mantenint la connectivitat WiFi. Són projectes de codi obert desenvolupats i mantinguts per una comunitat de voluntaris. Els dos projectes més actius són *NodeMCU* i *ESP8266 Lua Loader*. Malauradament, els dos són projectes relativament recents i encara els falta molt per a considerar-se projectes acabats, en paraules des propis desenvolupadors dels respectius projectes. A les etapes inicials del desenvolupament d'aquest projecte es van fer proves amb l'SDK *ESP8266 Lua Loader* i es va determinar que no era adequat usar-lo per la seva poca robustesa i freqüents errors. Per això s'ha decidit estudiar els firmwares oficials publicat per Expressif Systems.

Utilitzar firmwares oficials implica tractar el mòdul ESP8266 com una *caixa negra*. Cal adaptar-se a les possibilitats que ofereix, intentar preveure el seu comportament de la forma més afinada possible i adaptar-se a les seves limitacions. En aquest cas, la restricció més significativa és que tota comunicació entre dispositius s'ha de fer obligatòriament mitjançant el protocol IP i l'estàndard de xarxa sense fils IEEE 802.11. No hi ha cap forma d'enviar ni rebre informació crua, i cal tenir en compte la càrrega addicional que això comporta. En concret, buscar xarxes, establir i tancar connexions comporta obligatòriament uns segons de processament exclusiu.

4.1.2. Comunicació mitjançant comandes AT

Els firmwares oficials d'Expressif Systems es comuniquen mitjançant comandes Hayes (habitualment anomenades comandes AT). Aquest llenguatge és utilitzat àmpliament en l'àmbit de les comunicacions (sobre tot en mòdems i telèfons GSM), tot i que no està estandarditzat i cada fabricant en fa la seva pròpia implementació. Va ser concebut originàriament per a transmetre pel mateix cable les ordres de control del mòdem o telèfon, i la informació transmesa (que en aquells moments era analògica).

Aquest conjunt d'ordres consisteix en una sèrie de comandes simples, codificades en ASCII, que en aquest cas s'enviaran mitjançant un canal de comunicació sèrie. Cada comanda efectua una acció, com per exemple, escanejar les xarxes disponibles, connectar-se a una xarxa, obrir una connexió TCP, preparar l'enviament d'un paquet, etc.

El dispositiu ESP8266 processa les comandes seqüencialment. En acabar d'efectuar l'acció necessària, respon a l'emissor amb el resultat. No admet noves comandes fins que no ha acabat de processar l'ordre actual.

Malauradament el temps de processament, la sintaxi, i fins i tot la funcionalitat d'algunes ordres clau pot variar entre firmwares. Cal considerar aquest fet i fer una tria definitiva de firmware abans de començar a implementar qualsevol programa que faci ús d'aquest dispositiu.

Per a utilitzar el mòdul WiFi és imprescindible disposar, com a mínim, del text *"ESP8266 AT Instruction Set - Version 1.4"* (Expressif Systems, 2015), on 1.4 és la versió de firmware triada. Aquest document descriu la sintaxi i la funcionalitat de les ordres que admet el dispositiu. Resulta difícil trobar versions dels documents de referència en llengua anglesa perquè la majoria només són mantinguts en xinès pel fabricant. A més, la versió de firmware usada es considera obsoleta (la tria s'explica a l'apartat 4.1.3). Per aquest motiu s'ha inclòs la versió anglesa d'aquest document amb el fitxer ZIP publicat conjuntament amb aquesta memòria (no s'inclou aquí la sintaxi ni la descripció de les diferents ordres).

4.1.3. Tria de firmware

El dispositiu ESP8266 és presentat pel fabricant com un dispositiu ràpid, fiable i de baix consum energètic. En concret, el fabricant anuncia que el temps de transmissió d'un paquet és de 2 ms. No obstant això, el rendiment del dispositiu en la configuració de fàbrica és preocupant: en condicions reals s'ha determinat que el temps de transmissió d'un paquet és de 100 ms com a mínim (amb molta variabilitat) i la separació mínima entre paquets de 400 ms, xifres inacceptables per als objectius que es volen assolir en aquest projecte. La configuració de fàbrica tampoc aprova en robustesa, ja que el dispositiu es reinicia sol o es bloqueja per instants de manera totalment imprevisible. Com s'ha indicat anteriorment, el funcionament d'aquest dispositiu varia molt àmpliament segons el software que executa. Per sort, s'ha trobat un firmware oficial que soluciona totes aquestes mancances i és adequat per a implementar el sistema de comunicacions.

El firmware de fàbrica està datat a novembre de 2014, i no sembla estar catalogat. Hi ha nombroses versions del firmware oficial. Degut a que es disposa d'un Hardware amb només 512KB de memòria flash, la versió més recent que es pot instal·lar és la 1.4.1, publicada l'octubre de 2015. Aquesta versió no té errors coneguts. S'ha decidit, per tant, començar les proves amb aquesta versió. Després de diversos experiments i d'un ús prolongat, s'ha determinat que el seu rendiment, robustesa i estabilitat eren adequats, per tant es decideix usar-la definitivament i no provar versions inferiors.

A l'apartat 4.1.4 (verificació de les especificacions) es detallen alguns d'aquests experiments.

4.1.4. Verificació de les especificacions

Hi ha determinats valors que s'han de determinar experimentalment i que podrien limitar les especificacions finals del sistema de comunicacions, o directament fer-lo inviable. Es detalla

breument en aquest mateix apartat la metodologia seguida en els experiments i es discuteixen els resultats.

4.1.4.1. Velocitat i taxa d'enviament de paquets

Per a la visualització i hipotètic tractament de la informació transmesa pel vehicle en temps quasi-real (amb un retard raonable), és important que el temps de transmissió entre dos clients de la xarxa sigui elevat, però és encara més important que la taxa de paquets sigui elevada i que aquests estiguin equidistribuïts temporalment. Per a l'ESP8266 assolir aquest segon objectiu ha demostrat ser molt més difícil que assolir el primer.

S'ha efectuat un experiment per a determinar la taxa òptima de transmissió de paquets. Els resultats es mostren a la Taula 4-2 i la metodologia es detalla al final d'aquest apartat. Els resultats han estat impecables fins a una taxa de transmissió de 25 paquets per segon (40ms entre paquets). El temps de transmissió dels paquets es manté acotat entre 2-5 ms i la pèrdua de paquets és inexistent. El problema principal que apareix a partir d'aquesta xifra és que els paquets deixen d'enviar-se instantàniament i comencen a ser enviats per blocs. En el cas d'enviar un valor variable en el temps, aquest fenomen fa que un percentatge de paquets siguin inútils. També es perd un petit percentatge de paquets, cosa que podria ser conseqüència del fet anterior. El valor es considera suficient per a la visualització en temps quasi-real de variables dependents del temps. Una taxa de refresc de 25Hz pot resultar útil per a determinades tasques de diagnòstic, però és totalment insuficient per a rutines que necessitin actuar instantàniament sobre aquests resultats ("tancar el llaç").

ESP8266 firmware 1.4.1: Rendiment enviament paquets UDP segons taxa de transmissió				
Taxa de transmissió (Paquets/s)	**Separació entre paquets (ms)**	**Paquets perduts /1000 (arrodonit)**	**Temps de transmissió (ms)**	**Paquets equidistribuïts temporalment (error < 5%)**
5	200	0	2 – 5	Sí
10	100	0	2 – 5	Sí
15	66,67	0	2 – 5	Sí
20	50	0	2 – 5	Sí
25	**40**	**0**	**2 – 5**	**Sí**
27	37,04	9	2 – 35	No
30	33,33	31	2 – 35	No

Taula 4-2. Rendiment enfront la taxa d'enviament de paquets UDP en un ESP8266 amb firmware 1.4.1, mesurada en paquets perduts, temps de transmissió i si els paquets es troben equidistribuïts temporalment. S'ha destacat el valor òptim. La metodologia es descriu en aquest apartat.

La metodologia seguida en la realització de l'experiment ha estat la següent:

- S'ha establert una comunicació sèrie amb el dispositiu ESP8266 a 112500 bauds a través del port sèrie d'un ordinador d'escriptori.
- S'ha connectat el dispositiu a una xarxa WiFi generada pel mateix ordinador. S'ha situat el dispositiu a un metre de l'emissor (atenuació -40 dBm)
- S'han enviat 1000 paquets UDP de 30 bytes des de l'ESP8266. L'experiment s'ha repetit per a diferents taxes de transmissió (s'ha programat un script per tal que l'ordinador enviï les ordres corresponents pel port sèrie).
- El dispositiu ESP8266 no realitzava cap altra tasca durant l'experiment.
- S'han capturat els paquets UDP enviats mitjançant un programari servidor.
- S'ha determinat el nombre de paquets que no han arribat, el temps que triguen en ser rebuts pel servidor (comparant les marques de temps) i si els paquets arriben equidistribuïts temporalment.
- S'ha repetit l'experiment cinc cops per a les taxes de transmissió de 25 i 27 paquets per segon.
- S'ha validat que els resultats obtinguts per a 25 paquets per segon es poden reproduir connectant l'ESP8266 al microcontrolador, i a una distància de 4 metres amb obstacles. (no s'ha mesurat l'atenuació)

4.1.4.2. Velocitat i taxa de recepció de paquets

Amb un raonament idèntic a l'exposat a l'apartat 4.1.4.1, s'ha verificat quina és la taxa màxima de paquets que pot rebre i processar correctament el dispositiu. Aquest valor és vital en el cas de voler imposar remotament una consigna variable en el temps.

Els resultats de l'experiment es mostren a la Taula 4-3 i la metodologia es detalla al final d'aquest apartat. La taxa de recepció de paquets que ha ofert resultats raonables. Novament es presenta el mateix fenomen present en la taxa d'enviament: a partir d'una certa taxa de recepció els paquets deixen d'arribar equidistribuïts temporalment. Es determina que el valor òptim de recepció és de 50 paquets per segon. Resulta superior al d'enviament, probablement a causa del menor temps de processament associat.

ESP8266 firmware 1.4.1: Rendiment recepció paquets UDP segons taxa de transmissió				
Taxa de transmissió (Paquets/s)	**Separació entre paquets (ms)**	**Paquets perduts /1000 (arrodonit)**	**Temps de transmissió (ms)**	**Paquets equidistribuïts temporalment (error < 5%)**
10	100	0	2 – 5	Sí
20	50	0	2 – 5	Sí
30	33,33	0	2 – 5	Si
40	25	0	2 - 5	Si
50	**20**	**0**	**2 - 5**	**Si**
55	18,18	47	2 - 50	No
60	16,67	115	-	No

Taula 4-3. Rendiment enfront la taxa de recepció de paquets UDP en un ESP8266 amb firmware 1.4.1, mesurada en paquets perduts, temps de transmissió i si els paquets es troben equidistribuïts temporalment. S'ha destacat el valor òptim. La metodologia es descriu en aquest apartat.

La metodologia seguida en la realització de l'experiment ha estat la següent:

- S'ha establert una comunicació sèrie amb el dispositiu ESP8266 a 112500 bauds a través del port sèrie d'un ordinador d'escriptori.
- S'ha connectat el dispositiu a una xarxa WiFi generada pel mateix ordinador. S'ha situat el dispositiu a un metre de l'emissor (atenuació -40 dBm)
- S'han enviat 1000 paquets UDP de 30 bytes des de l'ordinador cap a l'ESP8266. L'experiment s'ha repetit per a diferents taxes de transmissió.
- El dispositiu ESP8266 no realitzava cap altra tasca durant l'experiment.
- S'han capturat els paquets UDP rebuts per l'ESP8266 establint una connexió sèrie i guardant tota la sortida en un fitxer.
- S'ha determinat el nombre de paquets que no han arribat, el temps que triguen en ser rebuts pel servidor (comparant les marques de temps) i si els paquets arriben equidistribuïts temporalment.
- S'ha validat que els resultats obtinguts per a 50 paquets per segon es poden reproduir connectant l'ESP8266 al microcontrolador, i a una distància de 4 metres amb obstacles. (no s'ha mesurat l'atenuació)

4.1.4.3. Enviament i recepció simultanis

S'han aconseguit reproduir els mateixos resultats amb el dispositiu emetent i rebent simultàniament a una taxa de 25 aquets per segon. La metodologia ha estat idèntica a la dels apartats anteriors.

4.1.4.4. Altres tasques

Pensant en el dispositiu com una “caixa negra”, és important conèixer el temps que trigarà en executar diverses tasques. S’ha mesurat el temps de detecció de xarxes WiFi, el temps de connexió a una xarxa WiFi i el temps d’establiment d’una connexió TCP. (Com s’ha indicat, el dispositiu no pot realitzar cap altra tasca durant la seva execució, ni tan sols enviar o rebre paquets).

S’han determinat aquests temps d’execució experimentalment, de forma aproximada. Els resultats es presenten a la Taula 4-4. Aquests presenten una gran variabilitat, l’origen de la qual no s’ha aconseguit establir. A banda d’això, els resultats són no només imprevisibles sinó molt elevats. En concret, la detecció i connexió a un punt d’accés WiFi pot demorar-se fins als 8 segons. Per tant, queda descartada qualsevol utilitat que no impliqui estar connectat ininterrompudament a un punt d’accés fixe.

ESP8266 firmware 1.4.1: Temps d’execució de diverses tasques	
Tasca	**Temps d’execució (ms)**
Detecció de xarxes WiFi properes	1000 - 3000
Connexió a un punt d’accés WiFi	2000 - 5000
Establiment d’una connexió TCP	500 - 2000

Taula 4-4. Temps d'execució de diverses tasques que es pot esperar amb un dispositiu ESP8266 amb firmware 1.4.1, mesurat experimentalment (aproximat). No s’ha pogut determinar l’orígen de la variabilitat.

4.1.4.5. Consum energètic

El fabricant anuncia aquest dispositiu com a un dispositiu de “baix consum”. En el document de referència publicat pel mateix fabricant (ESP8266 Datasheet, Expressif Systems, Juny 2015.) s’indiquen els valors de consum energètic mitjans mostrats a la Taula 4-5. No obstant això, també s’indica que es poden produir sobrepics de fins a 1A.

Experimentalment s’han validat aquests valors per a una connexió 802.11g. El major sobrepic observat, sota condicions d’alta activitat, ha estat de 600mA.

Parameter	Typical	Unit
Tx 802.11b, CCK 11Mbps, P_{OUT}=+17dBm	170	mA
Tx 802.11g, OFDM 54Mbps, P_{OUT} =+15dBm	140	mA
Tx 802.11n, MCS7, P_{OUT} =+13dBm	120	mA
Rx 802.11b, 1024 bytes packet length , -80dBm	50	mA
Rx 802.11g, 1024 bytes packet length, -70dBm	56	mA
Rx 802.11n, 1024 bytes packet length, -65dBm	56	mA
Modem-Sleep	15	mA
Light-Sleep	0.5	mA
Power save mode DTIM 1	1.2	mA
Power save mode DTIM 3	0.9	mA
Deep-Sleep	10	uA
Power OFF	0.5	uA

Taula 4-5. Valors mitjans del consum energètic de l'ESP8266 (alimentat a 3,3V). (Expressif Systems, Juny 2015).

4.1.5. Conclusions

Analitzar les possibilitats del mòdul WiFi ESP8266 forma part de l'abast d'aquest treball. Després de fer-ho, s'ha determinat que aquest dispositiu pot ser adequat, o no, segons l'escenari en el qual es vulgui utilitzar.

Nombrosos punts desaconsellen el seu ús per a la seva implementació en un producte com a eina per a la transmissió d'informació crítica, principalment la lentitud i el funcionament imprevisible en alguns casos, la relativament baixa taxa de transmissió de paquets màxima, i fet que el suport que proporciona el fabricant no està enfocat cap a aquest aspecte. No és acceptable que la sintaxi i la funcionalitat de les ordres canviï contínuament amb actualitzacions de firmware consecutives, especialment quan aquestes actualitzacions es fan imprescindibles per a corregir bugs presents en versions anteriors. Aquest producte esta plantejat més aviat com una eina de molt baix cost enfocada a prototipus i projectes casolans. Respecte el consum energètic, cal valorar si hi ha alternatives millors.

Per altra banda, el seu cost molt reduït i funcionament suficient el fan adequat per a l'ús que se li vol donar en aquest treball: transmissió de dades de diagnòstic durant la fase de desenvolupament d'un projecte, com podria ser un vehicle seguidor de línia.

El màxim de 16 connexions simultànies permetria crear, teòricament, una xarxa formada per 16 vehicles (o 15 vehicles i un PC), on totes les parelles de vehicles tenen una connexió establerta. Malauradament, la limitació de 25 paquets per segon fa que no sigui raonable mantenir 15 fluxos continuats d'informació amb un únic mòdul WiFi. En aquest cas, els vehicles només podrien enviar-se informació puntual (com a màxim un o dos paquets per segon) entre ells de forma directa. Si cal comunicació constant, es recomana fer-la mitjançant un ordinador que faci de servidor.

4.2. Protocol de comunicacions IP

L'ús del dispositiu ESP8266 obliga a utilitzar l'estàndard IEEE802.11 b/g/n per a la comunicació sense fils i el protocol IP per a la creació d'una xarxa de dispositius i el redireccionament (*Routing*) de les dades entre tots els clients d'aquesta xarxa. L'ús d'aquests protocols per part de tots els dispositius del sistema de comunicacions (ESP8266, punt d'accés sense fils, enrutador i ordinador) és un avantatge perquè aquests s'encarregaran de fer arribar les dades transmeses al seu destinatari, de forma transparent, en forma de paquets de xarxa. Per a treballar amb l'ESP8266 cal tenir un coneixement bàsic dels conceptes *protocol IP, capa de transport, paquet de xarxa, port, socket* i *adreça IP*.

El protocol IP (Internet Protocol) proporciona una comunicació entre clients en una mateixa xarxa utilitzant datagrames. Els datagrames són paquets de xarxa que contenen una capçalera amb caràcters de control (destinatari, etc.) i les dades a transmetre. Cada paquet de xarxa és tractat de forma individual. Un paquet de xarxa no s'envia directament al destinatari, de fet, no es pot saber a priori la localització del destinatari en la xarxa. Els paquets de xarxa s'envien a uns nodes intermedis (coneguts com a *enrutadors*) que s'encarreguen de redirigir el paquet fins que aquest arriba al destinatari final.

Els dispositius connectats a la xarxa es distingeixen assignant-los un identificador anomenat Adreça IP. Aquest identificador té una mida de 32 bits en la implementació IPv4, (la utilitzada avui en dia). Per a una millor llegibilitat es representa com a quatre números en decimal de vuit bits, separats per punts, per exemple, 255.255.255.255.

La transmissió d'informació entre dos membres de la xarxa es fa a través d'una capa secundària que s'executa sobre el protocol IP, anomenada capa de transport (Transport Layer). Mentre que el protocol IP s'encarrega d'enrutar paquets fins a la màquina destinatària, la capa de transport consisteix en un protocol addicional que defineix el contingut d'aquests paquets per tal d'aconseguir una transmissió de dades fiable. S'encarrega de l'establiment de connexions entre dispositius, transmissió i verificació de les dades, control del flux i capacitat de transmissió i de la seva multiplexació.

Hi ha dos protocols de capa de transport principals: TCP i UDP.

- **TCP** (Transmission Control Protocol) és un protocol orientat a connexions. Cal establir una connexió entre dos membres de la xarxa abans de transmetre informació. S'utilitza per a transmetre fluxos d'informació. Es confirma la correcta recepció de cada paquet de dades i inclou una pesada capa extra de verificació, cosa que no el fa apropiat per a aplicacions on el temps de resposta sigui important.

- **UDP** (User Datagram Protocol) és un protocol molt més simple que TCP, amb la mínima sobrecàrrega possible. UDP no necessita un establiment previ d'una connexió entre dos dispositius. Es limita a enviar paquets de xarxa de mida variable, sense cap procés de verificació ni confirmació. S'utilitza en aplicacions on el temps de transmissió és important.

El destinatari d'un paquet de xarxa és una adreça IP, però un ordinador pot executar alhora múltiples aplicacions que necessitin funcionalitats de xarxa. Per a determinar quina aplicació és la destinatària, a l'adreça IP s'hi afegeix un segon identificador anomenat port de xarxa. El port de xarxa és un identificador de 16 bits (valor numèric de 0 a 65535) que proporciona informació addicional sobre el destinatari del paquet. El sistema operatiu d'un ordinador, per exemple, utilitza aquesta informació per a saber a quina aplicació entregar-lo. Els protocols TCP i UDP suporten ambdós la identificació del port destí dels paquets.

Les aplicacions que s'executen en un ordinador d'escriptori demanen al sistema operatiu que els hi assigni en exclusivitat la comunicació amb una adreça IP destí, utilitzant un port determinat i una capa de transport determinada (TCP o UDP). Aquesta assignació s'anomena socket.

4.3. Microcontrolador STM32F4

Cada vehicle autònom està governat per una placa de desenvolupament STM32F4 Discovery, amb un microcontrolador STM32F407VG. És una placa d'alt rendiment, amb un microprocessador d'arquitectura ARM Cortex-M i nombrosos perifèrics. A la Figura 4-3 es mostra una fotografia d'aquesta placa de desenvolupament. A la taula Taula 4-6 se'n descriuren les característiques clau.

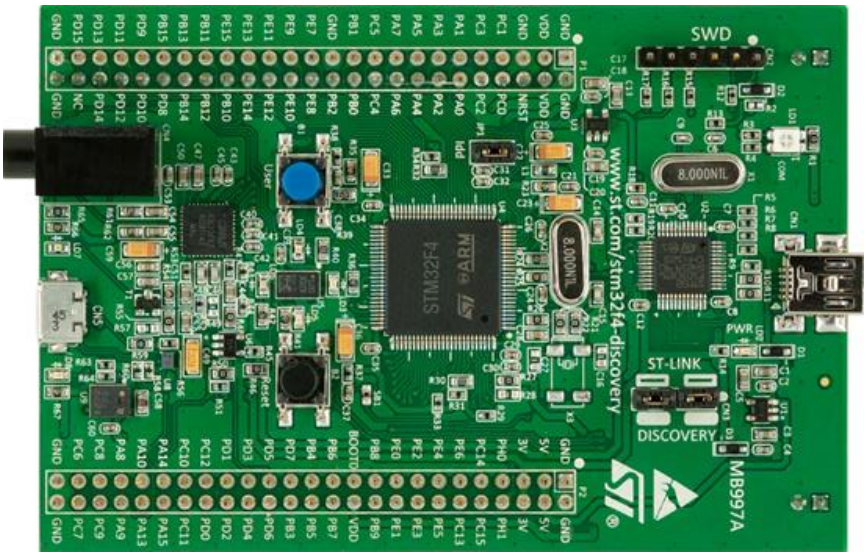

Figura 4-3. Placa de desenvolupament STM32F4 Discovery. (ST Microelectronics, 2016)

Placa de desenvolupament STM32F4 Discovery	
Microcontrolador	STM32F407 CPU ARM Cortex-M4 de 32 bits amb unitat de coma flotant 1 MB memòria flash 192 kB memòria RAM
Perifèrics microcontrolador	FPU, USART, SPI, ADC, DAC, I2C, PWM, DMA, Rellotges
Alimentació	5V
Extres	8 LEDs 2 polsadors Acceleròmetre de 3 eixos Micròfon

Taula 4-6. Característiques clau de la placa de desenvolupament STM32F4 Discovery (ST Microelectronics, 2016)

Aquesta placa s'ha d'encarregar també del govern del dispositiu ESP8266. Tot el software que s'ha dissenyat per a aquesta placa pot ser adaptat fàcilment a altres dispositius integrats. A l'annex A.2 (guia d'usuari) s'indica el procediment a seguir.

4.4. Ordinador d'escriptori

Un ordinador d'escriptori forma part (opcionalment) del sistema de comunicacions. L'ordinador ha de servir com a eina per a monitoritzar en temps quasi-real l'estat tots els vehicles, representar variables gràficament, transmetre ordres, modificar paràmetres de funcionament, etc.

La comunicació entre l'ordinador i la resta de la xarxa no comporta gaires dificultats, perquè els sistemes operatius d'escriptori s'encarreguen de gestionar automàticament la connexió a qualsevol xarxa constituïda pel protocol IP.

Un sistema operatiu d'escriptori permet assignar connexions a cada programa que executa. Aquesta assignació es fa en forma de *socket* (explicat a l'apartat 4.2). El sistema operatiu emmagatzema tots els paquets de xarxa que rep en un buffer. Cada programa pot llegir en qualsevol moment els paquets que han arribat i estaven assignats al seu *socket*.

4.5. Punt d'accés WiFi i enrutador de xarxa

Per a establir una xarxa sense fils WiFi és necessari un dispositiu mestre que s'encarregui de crear la xarxa (punt d'accés). Aquest dispositiu s'encarrega de d'autentificar els clients de la xarxa sense fils i de la comunicació entre ells. És habitual que aquest dispositiu també faci la funció d'enrutador, fent arribant tots els paquets de xarxa emesos pels seus clients al seu destinatari, segons el protocol IP .

El mòdul WiFi ESP8266 inclou la funcionalitat de ser usat com a punt d'accés, però s'ha decidit usar un punt d'accés extern per a preservar la simetria del sistema.

Qualsevol ordinador d'escriptori amb connectivitat WiFi pot funcionar com a punt d'accés i com a enrutador, eliminant la necessitat d'utilitzar un dispositiu dedicat. A l'annex A s'inclouen instruccions per fer-hi amb un ordinador que executa el sistema operatiu Windows.

5. Funcionalitat i estructura del sistema de comunicacions

A partir de l'anàlisi de tots els components, i en especial, de les possibilitats que ofereix el mòdul WiFi ESP8266, es defineixen el sistema de comunicacions a nivell funcional i estructural i les tasques específiques que cal realitzar per a posar-lo en marxa.

5.1. Funcionalitat

En un sentit ampli, la funcionalitat del sistema de comunicacions pot ser definida a nivell de participants, i a nivell de transmissió.

A nivell de participants, cal poder establir les següents connexions:

- **Connexió entre un PC i tots els vehicles**
- **Connexió de vehicle a vehicle**

A nivell de transmissió cal, com a mínim, transmetre dades de les següents formes:

- **Transmissió contínua de dades a temps real,** amb el menor retard i la major taxa de refresc possibles.
- **Transmissió puntual d'informació en forma d'ordres**, que poden contenir algun paràmetre. Per exemple, encendre, apagar, informar d'algun esdeveniment, canviar algun paràmetre numèric de configuració, etc.

5.2. Estructura

Els components a utilitzar, i en especial el mòdul WiFi, obliguen a utilitzar el protocol IP de comunicacions, introduït a l'apartat 4.2. Això significa que la tasca de dirigir les dades des del dispositiu emissor fins al dispositiu receptor es fa de forma automàtica i transparent. Només cal assignar un identificador únic (adreça IP) a cada vehicle.

S'ha decidit utilitzar un punt d'accés dedicat (*router*) per a la creació de la xarxa sense fils. Tot i que també es podria designar un dels vehicles com a vehicle mestre i que aquest actués com a punt d'accés, s'ha determinat a l'apartat 4.1 que les característiques del mòdul ESP8266 no són les adients. A més, la finalitat principal d'aquest sistema de comunicacions és la investigació en un entorn de laboratori, on tenir un element fixe no suposa cap inconvenient.

Els vehicles es connecten tots a la xarxa creada pel *router*. L'ordinador es connecta també al *router* (amb cable o inalàmbricament). A la Figura 5-1 s'han representat els elements que formen part del sistema de comunicacions. La connexió d'un ordinador a la xarxa és opcional. Cal aclarir que amb aquesta configuració es poden enviar dades directament entre vehicles, però aquestes són redirigides pel router (per exigències del protocol). Les dades transmeses entre vehicles no passen per l'ordinador.

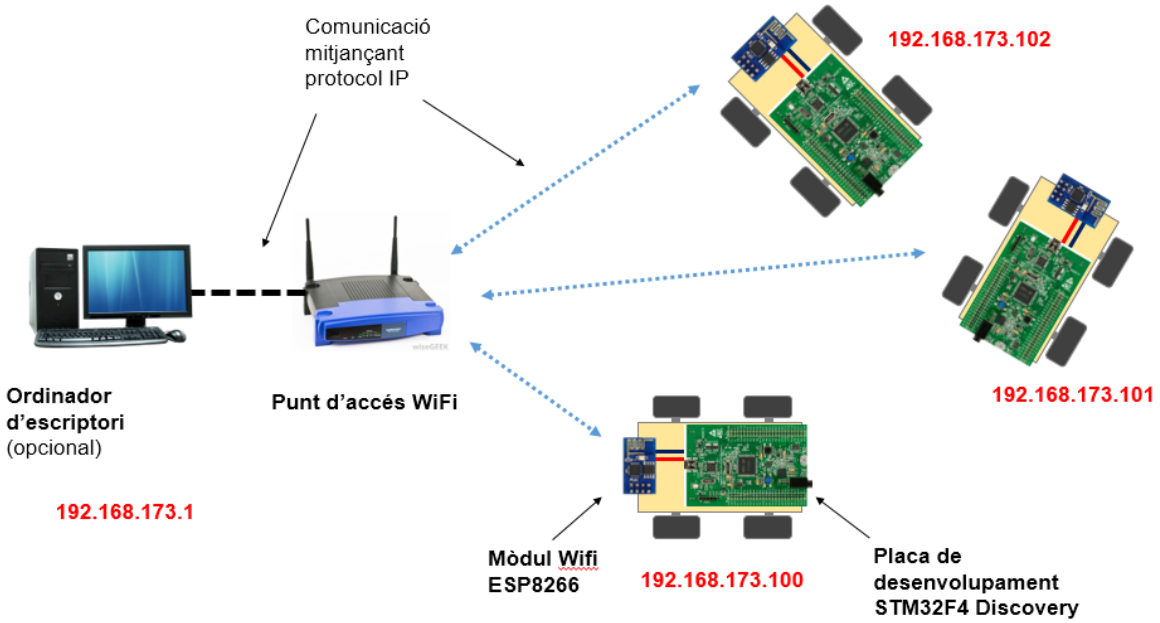

Figura 5-1. Estructura i elements del sistema de comunicacions.

Tenint en consideració les limitacions que imposa el hardware utilitzat, els requeriments funcionals es cobriran amb aquesta implementació:

- **Totes** les transmissions es faran mitjançant **paquets UDP** perquè arriben instantàniament al destinatari, mentre que els paquets TCP inclouen un mecanisme de verificació que allarga molt el procés de transmissió. També es transmetran via UDP els paquets puntuals (del tipus ordre), perquè el mòdul WiFi està limitat en el nombre de connexions que pot obrir (16). Si es decideix usar TCP i UDP alhora, cal obrir dues connexions per dispositiu a la xarxa, limitant-la a un màxim de 7 vehicles.

- La **transmissió contínua de dades a temps real** s'assolirà mitjançant un **feix de paquets UDP** enviats a una freqüència de **25 paquets per segon** (un paquet cada 40ms). A l'apartat 4.1 s'ha determinat que aquesta és la major taxa de transmissió que dóna resultats fiables. El temps de refresc de 40ms és suficient per a la visualització de les dades, però probablement és insuficient per a aplicacions que necessitin actuar sobre les dades rebudes i realimentar el control dels vehicles.

- Els vehicles poden establir connexions entre ells i enviar-se dades directament. Cal tenir en consideració les limitacions que imposa el mòdul WiFi: es poden establir un màxim de 16 connexions (que es tradueix en transmissions entre 15 vehicles i 1 PC). Per altra banda, la limitació de 25 paquets per segon fa que no sigui raonable mantenir 15 fluxos continuats d'informació amb un únic mòdul WiFi. En aquest cas, els vehicles només podrien enviar-se informació puntual (com a màxim un o dos paquets per segon) entre ells de forma directa. Si cal comunicació constant, es recomana fer-la mitjançant un ordinador que faci de servidor.

6. Desenvolupament de la solució

En aquest capítol s'estableix una codificació de les dades que viatgen en forma de paquets de xarxa, es descriu la implementació del mòdul WiFI ESP8266 en un sistema integrat i s'explica el desenvolupament d'una aplicació per a PC que permeti establir comunicació amb la resta de dispositius de la xarxa.

6.1. Codificació de la transmissió

L'establiment d'una estructura de xarxa sota el protocol IP permet l'enviament de paquets individuals de xarxa entre qualsevol dels seus integrants, com s'explica a l'apartat 4.2. Els paquets enviats són de tipus UDP i poden contenir fins a 512 bytes de dades.

Cal establir una sintaxis comuna entre tots els dispositius per tal que aquests puguin intercanviar informació.

No és òptim codificar tots els paquets amb la mateixa sintaxi: s'establirà una sintaxi diferent per als paquets que transmetin dades a temps real, i els paquets puntuals que contenguin una ordre. El primer byte determinarà el tipus de paquet.

6.1.1. Paquets de dades per a una transmissió constant d'informació

Aquests paquets transmeten el valor de diferents variables de forma continuada (25 cops per segon, o un paquet cada 40ms, com s'ha determinat en apartats anteriors) . S'anomenaran **paquets d'estat**.

Els paquets d'estat es codificaran de la següent forma:

- El primer byte és una '**E**' majúscula en ASCII (0x45). (E d'estat).
- La variable que representa cada byte, o grup de bytes, s'identifica per la seva **posició** en el paquet.
- Emissor i receptor coneixen prèviament quines variables s'intercanvien, quines posicions ocupen en el paquet i quin tipus de dades (enter, float, ...) representen.

És imperatiu, o molt recomanable, encabir totes les variables que es volen transmetre en el mateix paquet, perquè si no ja no es pot mantenir una taxa de refresc de 40ms. Identificar les variables per posició fa que el procés d'empaquetat i desempaquetat sigui més ràpid.

A la Figura 6-1 es mostra un exemple de paquet d'estat que transmet un enter (de 2 bytes) i un float (de 4 bytes). Tant emissor com receptor han de saber que el paquet conté la variable velocitat, que aquesta està situada a la posició 3 (el primer caràcter), i que és de tipus float. Ídem per a la variable temperatura.

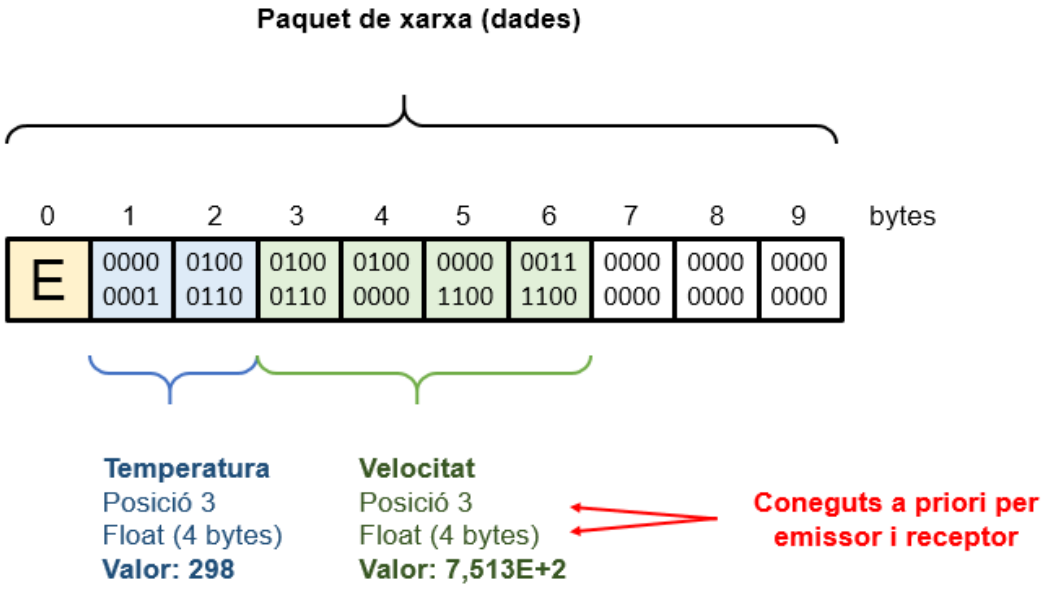

Figura 6-1. Contingut d'un paquet d'estat. Els valors que conté es fixen per posició i s'han de conèixer prèviament.

En el cas d'existir diversos tipus de paquet d'estat (amb contingut o sintaxi diferents), es poden identificar amb diferents lletres (F, G, H, ...)

6.1.2. Paquets de dades per a una transmissió puntual d'informació (ordres)

Aquests tipus de paquets de dades s'envien puntualment per informar d'un esdeveniment, donar una ordre, modificar un paràmetre, etc. (Per exemple, START, STOP, OK, COLISIO, KP=50, VMAX=300). S'anomenaran **paquets d'ordre.**

Els paquets d'ordre es codificaran de la següent forma

- El primer byte és el caràcter **'>'** en ASCII (0x3E)
- La resta del paquet es composarà per una cadena de caràcters en ASCII, que serà el nom de l'ordre i el seu identificador únic
- Opcionalment poden portar un paràmetre. El nom de l'ordre i el paràmetre es separaran amb el caràcter **'='**, per tant, aquest no pot formar part del nom de l'ordre.
- Si hi ha més d'un paràmetre, es separaran amb el caràcter **','.**

A la Figura 6-2 es mostra un exemple de paquet de tipus ordre, on VMAX és el nom de l'ordre i 500 el paràmetre. Cada casella representa un byte (8 bits). Quan en una casella es mostra un caràcter gran, és la representació del byte en ASCII.

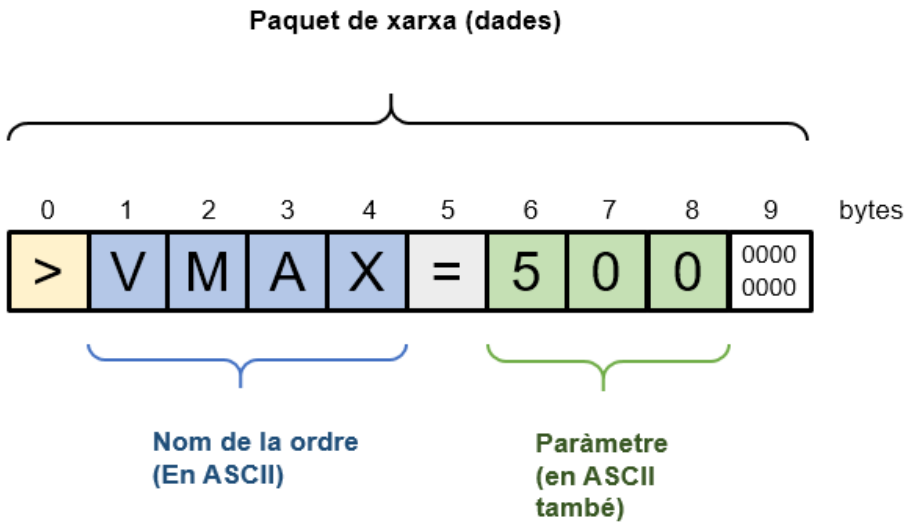

Figura 6-2. Contingut d'un paquet d'ordre amb un paràmetre.

6.2. Implementació del mòdul WiFi ESP8266 en un sistema integrat

La implementació del mòdul WiFi ESP8266 en un sistema integrat és un dels objectius principals. També resulta ser, de llarg, la tasca més complexa i laboriosa d'aquest treball. En aquest cas concret el sistema integrat és un microcontrolador ARM STM32F4, però tot el contingut d'aquest apartat s'aplica a qualsevol altre tipus de sistema.

Tota la comunicació entre el mòdul WiFi i el microcontrolador es fa a través del port sèrie (UART). Per tant, aquesta part del desenvolupament consisteix bàsicament en aconseguir gestionar de forma intel·ligent els canals de comunicació d'entrada i sortida utilitzant el mínim possible de temps de CPU i, sobre tot, sense interferir amb l'execució de tasques prioritàries per part d'aquesta.

Cal tenir en compte que la finalitat del sistema de comunicacions que es vol dissenyar en el present treball és poder ser usat en dispositius integrats que executen tasques crítiques. En concret, el disseny inicial es fa pensant en un vehicle seguidor de línia, el qual ha d'encarregar-se de llegir la informació dels sensors, processar aquestes dades, generar l'acció de control

necessària, governar els perifèrics, etc. El mateix microcontrolador que s'encarrega de totes aquestes tasques s'ha d'encarregar també de gestionar les comunicacions (enviar i rebre ordres al mòdul WiFi, interpretar-les, empaquetar i desempaquetar les dades a transmetre, efectuar canvis en el sistema , etc).

S'estableix que la implementació mòdul WiFi – sistema integrat ha de complir incondicionalment amb els següents objectius:

- **No interferir amb l'execució del programa principal:** El microcontrolador haurà d'executar altres tasques més prioritàries a part d'encarregar-se de gestionar les comunicacions. No s'ha d'interferir de cap manera amb aquestes tasques. Aquestes tasques s'han d'executar exactament de la mateixa manera, amb idèntiques funcionalitats i cronometratge, que si no s'hagués instal·lat el mòdul WiFi. Això no ha de requerir en cap cas fer modificacions en el programa principal.

- **Minimitzar el consum de memòria RAM i permetre regular-lo segons sigui necessari:** Per raons idèntiques a les del primer objectiu, el consum de memòria ha de suposar una fracció del total disponible en el sistema integrat. Per tal de permetre una implementació més flexible, aquest ha de ser regulable segons les necessitats de comunicació i/o els recursos disponibles en cada microcontrolador.

- **Aconseguir un sistema de comunicacions fiable:** El sistema ha de ser robust davant possibles problemes. En primer lloc, un hipotètic mal funcionament del sistema de comunicacions no ha de poder-se contagiar al programa principal en cap cas. En segon lloc, el sistema de comunicacions ha de ser robust davant d'errors interns i ha de ser capaç de recuperar-se després de qualsevol contratemps (entrades externes errònies o inesperades, mal funcionament del port sèrie, error en el mòdul WiFi, temps de CPU insuficient, etc)

- **Aconseguir un sistema de comunicacions flexible i transparent:** S'ha de poder instal·lar de forma fàcil en un sistema integrat que estigui executant altres tasques, i s'ha d'adaptar a elles. El sistema no s'ha de dissenyar inicialment pensant en com seran aquestes tasques, els espais lliures de temps de CPU que deixaran, etc. sinó que ha de fer-se un disseny general. En cap cas han de ser necessàries modificacions ad-hoc en el programa principal ni en el programa de comunicacions.

6.2.1. Control en sistemes integrats

La “tasca prioritària” que executa el sistema integrat serà, en la majoria de casos, algun tipus d’algorisme de control. El sistema de referència usat com a base en aquest treball, el vehicle seguidor de línia fet amb el microcontrolador STM32F4, s’encarrega de processar les dades rebudes pels sensors, de calcular l’acció de control mitjançant un controlador PID i de generar la senyal PWM necessària per a controlar els motors de les rodes. La seva funcionalitat serà ampliada en un futur.

Hi ha diverses estratègies que es poden seguir per a implementar aquest esquema de control en un sistema integrat. Totes elles passen en configurar interrupcions a partir d’un rellotge intern del microcontrolador, de forma que una **rutina de control** s’executi a períodes fixes. En el cas del vehicle seguidor de línia, aquesta rutina de control inclou tant la cadena d’adquisició com la cadena d’actuació, però podria no ser així. A la Figura 6-3 es representa gràficament aquesta estratègia, mostrant el temps de CPU que consumeix la rutina de control i el temps que queda lliure.

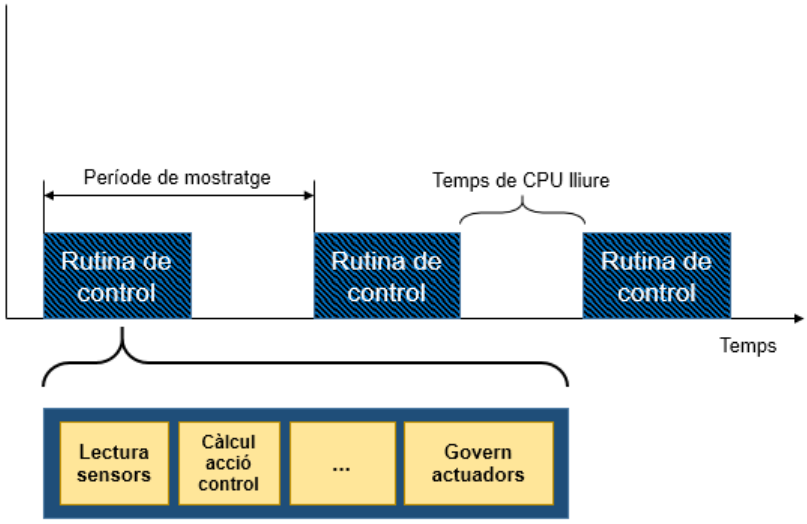

Figura 6-3. Representació de l’ocupació de la CPU en el temps per part d’un dispositiu integrat que efectua una tasca simple de control.

6.2.2. Transmissió de dades a través d’un port sèrie (UART) del microcontrolador.

La majoria de microcontroladors incorporen com a mínim un perifèric UART, que permet establir una connexió sèrie d’entrada/sortida. El propi perifèric s’encarrega de la transmissió de dades, però no les emmagatzema. En el cas del microcontrolador STM32F4, el perifèric UART emmagatzema el darrer byte rebut en un registre. Cal copiar manualment cada byte

que arriba des d'aquest registre a la memòria RAM (per exemple, amb la CPU) abans que arribi el següent, si no, es perd. Existeix la possibilitat de generar una interrupció en la CPU cada cop que hi ha un nou paquet disponible. De forma similar, les dades que es volen transmetre s'han de copiar byte a byte a un registre del perifèric en els instants adequats. A la Figura 6-4 es mostra esquemàticament el procés i tots els elements que hi intervenen, en el cas que la CPU sigui l'encarregada de moure les dades entre la memòria RAM i el perifèric.

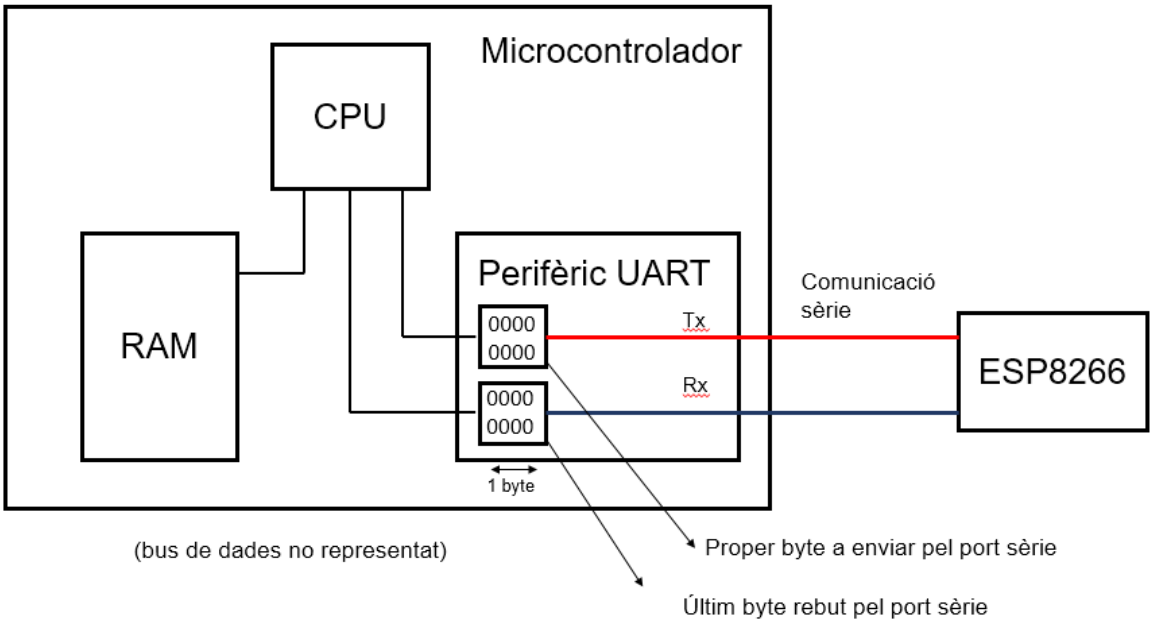

Figura 6-4. Representació esquemàtica dels elements que intervenen en el procés d'enviament i recepció de dades mitjançant una comunicació sèrie en un dispositiu integrat.

La comunicació amb el dispositiu ESP8266 es fa a 112500 bauds (bits per segon), per tant, arriba un byte cada 71 ms (representat a la Figura 6-5). No hi ha garanties de que arribin de forma sincronitzada sense coincidir amb la rutina de control. Cal establir algun mecanisme per a poder tractar aquestes entrades i sortides de forma asíncrona.

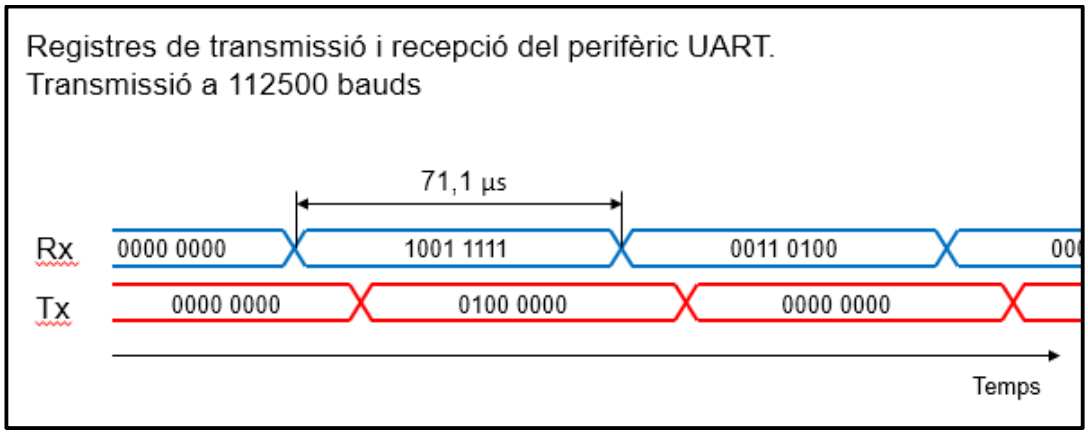

Figura 6-5. Representació del valor dels registres d'entrada i sortida del perifèric UART en el temps. (un byte per registre i instant de temps)

S'ha establert un buffer per a la recepció de dades (detallat a l'apartat 6.2.2.1) , que actuarà com a memòria intermèdia entre l'arribada de cada byte i el seu tractament. El seu ús està motivat per dos punts importants: cal rebre un paquet sencer abans de tractar-lo (no es pot tenir la CPU ocupada durant la recepció sencera), i les dades podrien arribar en un moment en què el programa principal presenti pics de demanda de temps de CPU, per tant els paquets rebuts s'haurien de poder acumular per a poder ser tractats quan la CPU estigui disponible.

De forma similar, s'ha configurat un buffer de sortida. D'aquesta manera és possible posar paquets en cua per a ser enviats en qualsevol moment, fins i tot des del programa principal. La rutina de comunicacions s'encarregarà de donar una separació temporal suficient als paquets i d'enviar-los tan bon punt sigui possible.

6.2.2.1. Descripció de la comunicació pel port sèrie mitjançant buffers d'entrada i sortida i interrupcions

Els dos buffers s'han aconseguit establint una estructura de dades que reserva un fragment de memòria RAM de mida constant. Els buffers reserven un espai de memòria dedicat d'un nombre fixe de bytes. A més, aquesta estructura manté un control sobre dos punters que proporcionen la posició de memòria relativa al proper byte que serà llegit i al proper byte que serà escrit al buffer. En arribar a l'últim byte de l'espai de memòria reservat, els punters tornen al primer byte. A la Figura 6-6 es representa esquemàticament aquesta estructura.

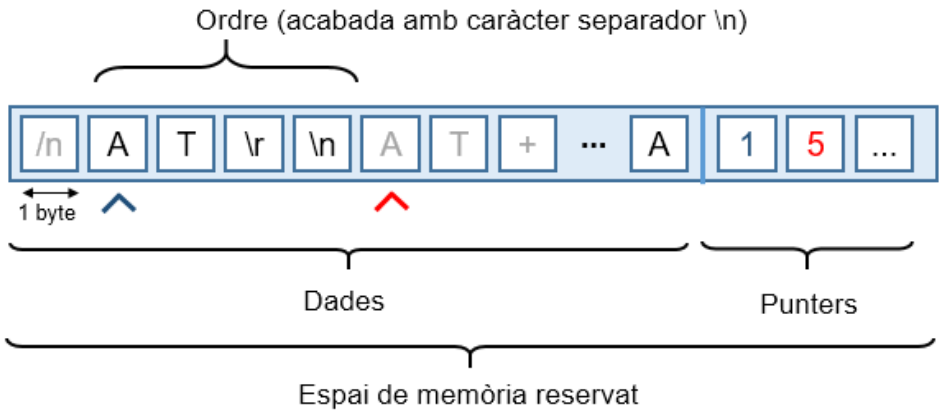

Figura 6-6. Buffer de dades implementat.

Per a llegir un nou byte del buffer, cal copiar el valor que està assenyalant el punter de lectura, i tot seguit avançar el punter. Quan el punter d'escriptura arriba a la mateixa posició de memòria que el punter de lectura, vol dir que no queden dades noves per a llegir. L'adreça de memòria de cada byte és la corresponent a sumar l'adreça de memòria del primer byte del buffer (coneguda) i el valor del punter.

La lectura dels buffers es farà en ordres senceres, no de byte en byte. Les dades rebudes pel perifèric ESP8266 s'intercalen entre elles mitjançant els caràcters de retorn de carro i de nova línia (CR+LF). Aquest mètode és usat habitualment en sistemes DOS/Windows. Altres sistemes utilitzen només el caràcter LF. Cada acció de lectura es farà fins al proper caràcter LF, per a assegurar que totes les ordres es llegeixen senceres. Per tant, cal detectar si hi ha algun LF sense llegir en el buffer abans d'extreure informació. En la codificació ASCII, el caràcter LF és el byte 0x0A. Tot i ser un caràcter no imprimible, la majoria de compiladors permeten representar-lo com a '\n'.

Per a afegir un nou byte al buffer, cal sobreescriure el valor que assenyala el punter d'escriptura. Aquest valor es considera que ja està llegit (el punter de lectura l'ha avançat), i ja no té cap utilitat. Quan el punter d'escriptura arriba a la mateixa posició de memòria que el punter de lectura, significa que hi ha hagut un sobreeiximent del buffer perquè les dades no s'han llegit suficientment ràpid. Si es dóna el cas, cal triar si descartar el primer paquet que va entrar al buffer i encara no ha estat processat, o descartar l'últim que està intentant entrar. En aquest cas s'ha optat per a descartar el primer paquet, perquè és més probable que les seves dades hagin quedat obsoletes. Això es fa avançant el punter de lectura fins al proper caràcter de salt de línia (però sense llegir les dades). En qualsevol cas, en condicions nominals de funcionament els buffers s'han de buidar més ràpid que omplir-se, i un sobreeiximent d'algun d'ells sempre significa una incidència.

La mida d'aquests buffers pot ser ajustada modificant un paràmetre (veure Annex A.3). Com a mínim, hauria de ser 10-20 cops la longitud mitjana d'un paquet de dades, o superior si es preveuen pics de demanda de capacitat de comunicació i/o períodes amb temps de CPU extraordinàriament reduït. Per al vehicle seguidor de línia, s'ha triat un valor conservatiu de 1024KB. En cas de memòria limitada, es pot reduir la seva mida. No es compromet l'estabilitat del sistema, però es podrien perdre alguns paquets.

Amb els buffers establerts, enviar o rebre una ordre es converteix en una tasca fàcil: només cal afegir-la al buffer de sortida, o llegir-la del buffer d'arribada. A més, es pot fer en qualsevol moment de l'execució del programa, de forma asíncrona amb el seu enviament/recepció. Cal establir també un mecanisme que s'encarregui moure cada byte entre aquests buffers i els registres d'entrada i sortida del perifèric UART. Es representa esquemàticament el procés a laFigura 6-7.

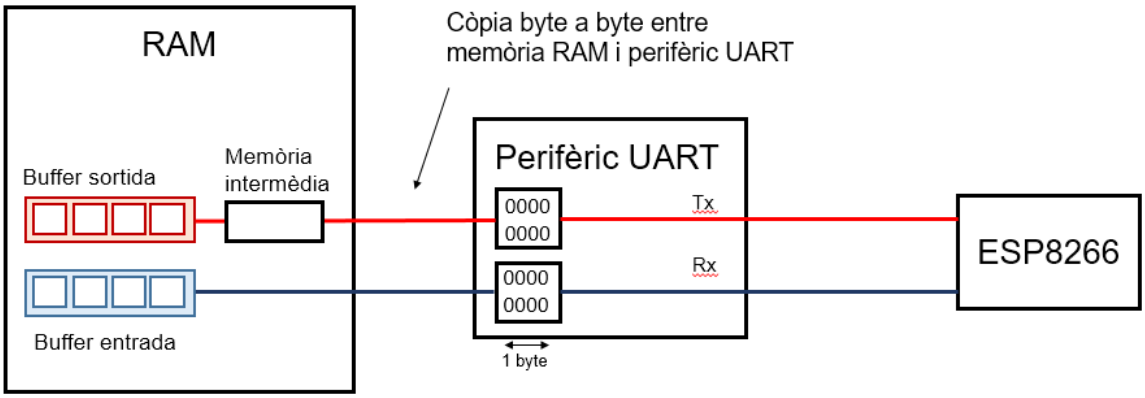

Figura 6-7. Representació esquemàtica del camí que ha de recórrer cada byte que s'envia o es rep pel port sèrie.

Per al correcte funcionament del buffer d'arribada, és necessari copiar-hi cada byte rebut pel port sèrie abans que arribi el següent. S'ha programat el sistema de forma que la CPU és l'encarregada fer totes aquestes còpies. Això es fa mitjançant una interrupció que genera el perifèric UART cada cop que hi ha noves dades disponibles.

Per altra banda, cal moure cada byte des del buffer de sortida fins al registre de sortida del perifèric UART. Aquesta còpia no es fa directament llegint a la localització del buffer en memòria, sinó que les dades es copien transitòriament a una localització intermèdia. Es fa aquesta còpia perquè l'enviament s'ha de fer per ordres senceres (separades per caràcter de fi de línia). Un cop es marca cada ordre com a llegida del buffer, el contingut de la seva adreça de memòria podria variar. La còpia a aquesta localització intermèdia es pot fer quan la CPU estigui lliure.

A partir d'aquesta localització intermèdia s'han de copiar tots els bytes al registre de sortida perifèric UART, la qual cosa s'ha de fer de manera síncrona amb la transmissió. Per tant, aquesta còpia s'ha de poder realitzar durant l'execució del programa principal. S'utilitzarà la CPU per a fer aquest moviment. Es configurarà el perifèric UART per a que generi una interrupció després d'enviar un byte, i aquesta interrupció s'aprofitarà per a enviar el següent.

A la Figura 6-8 es representen les còpies entre el perifèric UART i la memòria RAM superposades amb l'execució del programa principal.

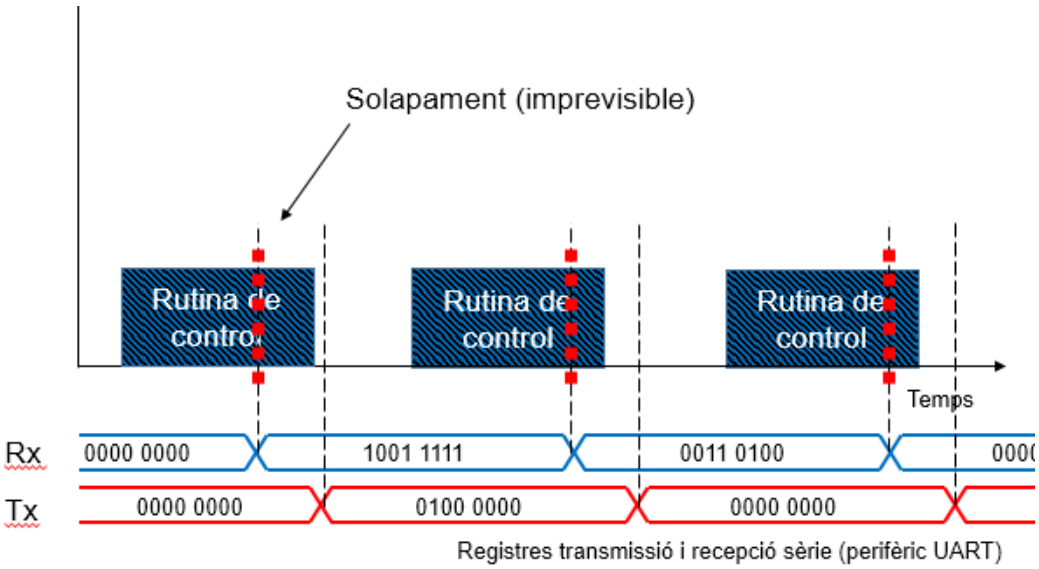

Figura 6-8. Representació temporal dels valors dels registres d'entrada i sortida del perifèric UART i la possible interferència amb alguna altra activitat prioritària per part de la CPU.

Les interrupcions a la CPU que s'han configurat per a moure bytes entre la memòria RAM i els registres del perifèric UART tenen més prioritat que l'execució del programa principal, però només es composen per unes poques instruccions i el seu impacte s'ha determinat menyspreable. Es considera, doncs, que aquest mètode respecta la condició de no interferir amb el programa principal. Molts microcontroladors incorporen una funcionalitat anomenada DMA (Direct Memory Access) que permet la transmissió d'informació entre els perifèrics del microcontrolador i la memòria sense necessitat d'executar cap instrucció. No obstant això, els resultats no han estat satisfactoris amb el controlador DMA del microcontrolador STM32F4 degut a l'alta freqüència d'operació. A l'Annex A.2 (guia d'instal·lació) s'inclouen indicacions per a implementar l'enviament i recepció via DMA.

6.2.3. Enviament i recepció de paquets mitjançant un mòdul WiFi ESP8266

A l'apartat 4.1 s'ha descrit el funcionament del mòdul WiFi ESP8266. En concret, a l'apartat 4.1.2 s'ha exposat el seu mètode de govern, que consisteix en comandes curtes de text, codificades en ASCII, que s'han d'enviar a través d'un canal de comunicacions sèrie (UART).

Si es vol dedicar el microcontrolador exclusivament a les comunicacions, la implementació no és molt complicada. La connexió a una xarxa WiFi, l'establiment d'una connexió amb un altre dispositiu i l'enviament de paquets de xarxa es fa de forma molt senzilla. Per altra banda, en rebre un paquet de dades de l'exterior, el mòdul WiFi envia immediatament un missatge pel port sèrie que conté aquest paquet de dades.

Amb l'establiment dels buffers de comunicacions, s'ha facilitat molt l'enviament i recepció de paquets de xarxa, la codificació dels quals s'ha definit a l'apartat 6.1.

Per a cada paquet rebut s'han d'efectuar les següents accions:

- Llegir una línia sencera (fins el proper salt de línia) del buffer d'entrada i separar el contingut del paquet de la resta de caràcters de control.
- Desempaquetar el paquet segons la codificació definida a l'apartat 6.1
- Assignar els valors desempaquetats a les variables corresponents. En cas que sigui un paquet del tipus "ordre", executar-la en aquest moment.

Per a cada paquet a enviar s'han d'efectuar les següents accions:

- Obtenir la informació que es vol enviar.
- Empaquetar el paquet segons la codificació definida a l'apartat 6.1.
- Afegir al buffer de sortida una ordre al mòdul WiFi, anunciant que es vol transmetre un paquet de xarxa.
- Afegir al buffer de sortida, immediatament després, el paquet que es vol transmetre.

6.2.4. Execució concurrent de la gestió de les comunicacions WiFi i d'altres activitats prioritàries

La implementació en el sistema integrat del govern de les comunicacions WiFi no ha d'interferir en cap cas amb l'execució del programa principal, sinó que s'ha d'adaptar als instants amb temps de CPU lliure.

Es defineix una **rutina de gestió de comunicacions** que efectua les següents accions:

- Llegeix un element del buffer d'arribada, el desempaqueta i el processa, seguint el procediment de l'apartat 6.2.3.
- Posa en cua al buffer de sortida els paquets d'enviament periòdic que s'han programat (paquets d'estat), seguint el procediment de l'apartat 6.2.3, amb el període especificat. Per exemple, en el cas del vehicle seguidor de línia s'envien la velocitat, lectura dels sensors i altres variables cada 40ms.
- Prepara l'enviament del primer element del buffer de sortida, si ha passat suficient temps des de l'enviament del paquet anterior (per limitacions del mòdul WiFi). Això comporta copiar-lo a una localització de memòria intermèdia (representada anteriorment a la Figura 6-7, pàgina 38), i prepara les interrupcions necessàries per tal que aquest element es vagi enviant byte a byte pel port sèrie.

A la Figura 6-9 es representa esquemàticament aquesta rutina.

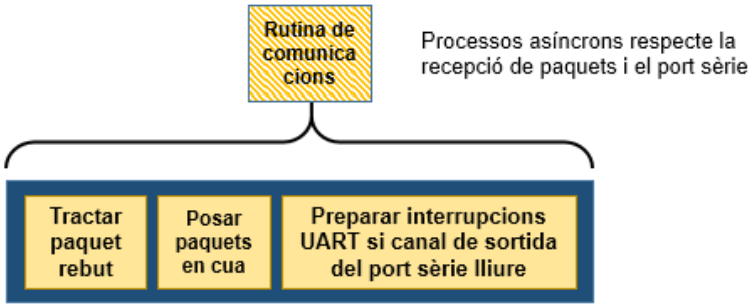

Figura 6-9. Accions que efectua la rutina de gestió de comunicacions.

L'avantatge de definir aquesta rutina és que es pot cridar en qualsevol moment, no ha d'anar sincronitzada ni amb el programa principal, ni amb la comunicació pel port sèrie, ni amb la recepció o enviament dels paquets. Això és un requisit, ja que no es pot saber a priori quan hi haurà temps de CPU lliure.

Finalment es té una rutina que pot combinar-se amb el programa principal. Aquesta combinació pot fer-se de diverses maneres. Una d'elles consisteix en executar la gestió de les comunicacions immediatament després d'executar la part prioritària, tal i com es representa a la Figura 6-10.

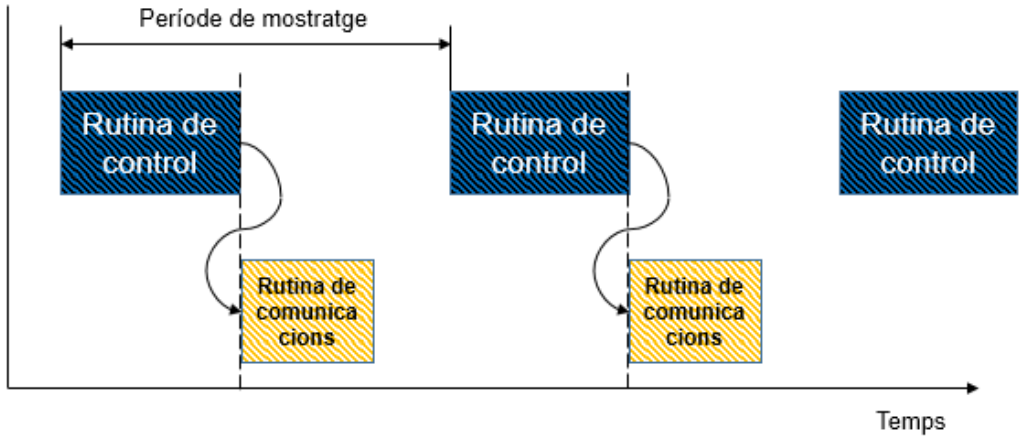

Figura 6-10. Execució consecutiva de la rutina principal i la rutina de gestió de comunicacions.

Aquesta estratègia presenta l'avantatge de què s'independitzen les dues rutines, cosa que suposa un menor pic de consum de memòria (cada rutina assigna memòria dinàmicament i l'allibera abans de passar a la següent rutina). També suposa conèixer de forma determinista els moments en els quals s'executarà la rutina de gestió de comunicacions, tot i que això no és important perquè aquesta s'ha definit de forma que el seu instant d'execució no és important.

No obstant això, executar seqüencialment la rutina de control (prioritària) i la rutina de gestió de comunicacions presenta nombrosos desavantatges. En primer lloc, cal que entre dues execucions successives de la rutina de control hi hagi temps suficient per a executar la rutina de gestió de comunicacions, i aquesta té un temps d'execució variable (segons l'estat dels buffers), per tant es desaprofita una part important del temps de CPU. En segon lloc, cal adaptar de forma "ad-hoc" la primera i la segona rutina, per tal que no es trepitgin. Això pot no ser senzill quan l'acció prioritària no es composa d'una sola rutina sinó de vàries. També implica haver de fer ajustaments cada cop que es vulguin modificar les funcionalitats existents en qualsevol de les rutines o afegir-ne de noves.

Una segona opció és executar contínuament la rutina de gestió de comunicacions amb una prioritat més baixa que la rutina de control (prioritària). Aquesta estratègia es representa a la Figura 6-11.

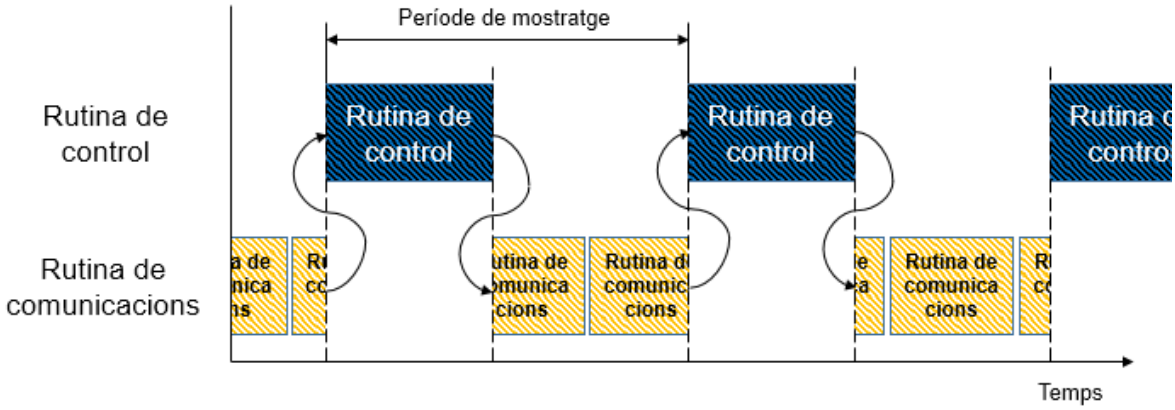

Figura 6-11. Execució de la rutina prioritària interrompent la rutina de comunicacions.

Aquesta estratègia no presenta cap des desavantatges anteriors: no cal planificar espai de temps de CPU suficient per a la rutina de gestió de comunicacions (s'adapta al disponible), no queden espais buits desaprofitats, i a més no cal fer una adaptació ad-hoc entre ambdues rutines. De fet, només cal preveure que la rutina prioritària deixi un percentatge de temps de CPU lliure, sense importar l'instant exacte. La rutina de comunicacions (no prioritària) s'hi adapta automàticament.

S'ha triat aquesta segona opció per a la implementació d'aquest sistema de comunicacions, pel seu millor aprofitament del temps de CPU i perquè un dels objectius marcats era que la seva implementació fos flexible i transparent.

Els desavantatge principal d'aquesta estratègia és que necessita una major quantitat de memòria RAM, perquè en el moment que s'executa una rutina mentre una altra està interrompuda és necessari mantenir assignada la quantitat de memòria dinàmica que les dues necessiten. A més, és necessari adaptar la rutina de gestió de comunicacions per tal que pugui ser interrompuda en qualsevol moment. Aquesta rutina ja ha estat dissenyada de forma en què el seu moment d'execució no sigui rellevant. També cal posar especial cura en el disseny, ja que les dades (variables) que es llegeixen i escriuen poden canviar de valor en qualsevol moment de l'execució.

6.2.5. Resum de la solució implementada en el sistema integrat

El resultat final és una implementació que compleix amb tots els objectius plantejats a l'apartat 6.2. La gestió de les comunicacions utilitza la CPU del microcontrolador amb una prioritat mínima, de forma que no s'interfereix de cap manera amb l'execució del programa principal (sigui quina sigui la seva planificació), adaptant-se contínuament als espais de temps de CPU

disponibles, fins i tot quan aquests espais no estan sincronitzats amb la recepció de dades externes ni amb la freqüència de transmissió del port sèrie. Només s'interromp el procés prioritari per a enviar o rebre dades pel port sèrie. La successió temporal d'aquests esdeveniments s'ha representat a la Figura 6-12. Aquesta rutina, per disseny, no es bloquejarà ni crearà un mal funcionament en el perifèric fins i tot en el cas que no se li deixi temps de CPU suficient. L'única conseqüència de no donar-li temps suficient és la possible pèrdua de paquets, però el sistema es recupera automàticament tan bon punt torna a tenir temps de processament suficient

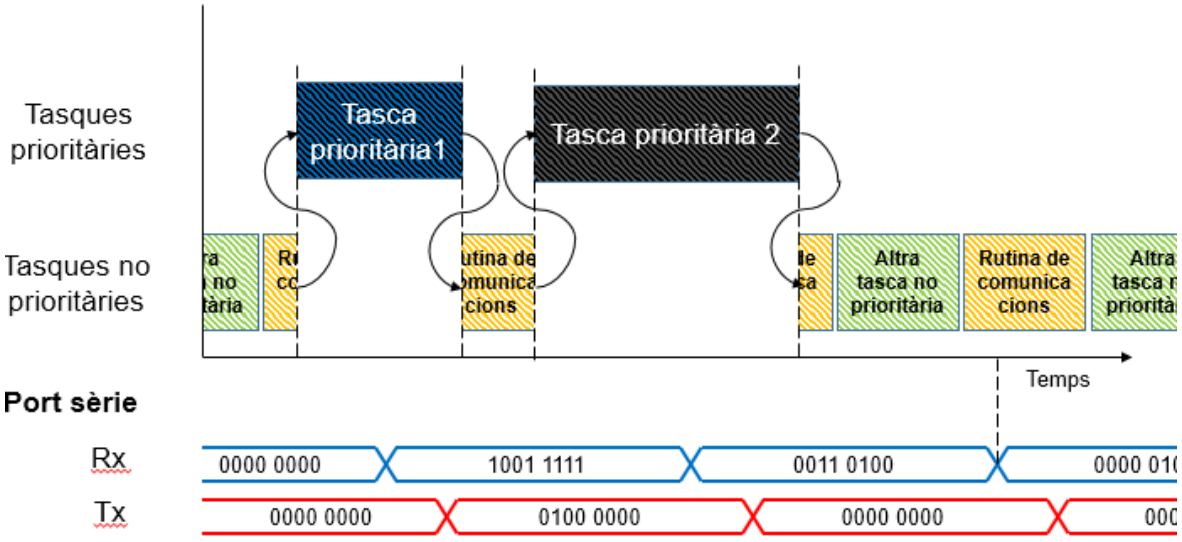

Figura 6-12. Representació temporal de l'execució de la rutina de gestió de comunicacions amb la implementació final. Cada canvi en els registres del port sèrie implica una interrupció per a copiar el seu valor que no s'ha representat per claredat.

El consum de memòria RAM depèn àmpliament de la mida dels paquets de xarxa i de la mida dels buffers. Aquests paràmetres són modificables. A l'annex A.3 s'explica el procediment i els criteris a seguir.

A nivell de programació, per a implementar la gestió de comunicacions WiFi en un sistema integrat cal configurar interrupcions d'enviament i de recepció del perifèric UART i configurar l'execució contínua de la rutina de gestió de comunicacions. A la Figura 6-13 es mostra un fragment de *pseudocodi* amb un exemple d'estratègia per a assolir aquesta implementació, col·locant la rutina de gestió de comunicacions en un bucle infinit del main. A l'annex A.2 (guia d'instal·lació) hi ha instruccions detallades per a implementar el sistema.

```
int interrupcioDadaRebudaUART(uint8_t *dada, int mida) {
      escriuBuffer(&bufferEntradaUART, &dada, mida)
}

int COM_gestionarComunicacions() {
      /* Rutina principal de comunicacions */

      /* Executar aquesta funció contínuament però amb prioritat
      baixa. Pot ser interrompuda en qualsevol moment.       */


      /* Tractar un element del buffer de dades rebudes */
      while(tractarDadesRebudesUART() == 99);

      /* Posar en cua un enviament de l'estat del microcontrolador
      */
      if((HAL_GetTick() - tickEnviamentEstat) >
      PERIODE_ENVIAMENT_ESTAT) {
            posarCuaEnviamentEstat();
            tickEnviamentEstat = HAL_GetTick();
      }

      /* Enviar un element del buffer de comandes pendents per
      enviar. */
      if((HAL_GetTick() - tickEnviamentDadesUART) >
      PERIODE_ENVIAMENT_DADES_UART) {
            enviarBufferUART();
            tickEnviamentDadesUART = HAL_GetTick();
      }
}

int main() {
      inicialitzarPeriferics();
      while(1) {
            COM_GestionarComunicacions();
            // Situar aquí altres tasques no prioritàries
      }

}
```

Figura 6-13. Pseudocodi amb un exemple d'estratègia d'implementació de la rutina de comunicacions en un programa existent. Es mostren les tres accions bàsiques que executa la rutina de gestió de comunicacions. Per a instruccions detallades cal consultar l'annex A.2.

6.3. Desenvolupament d'una aplicació auxiliar per a ordinador

El sistema de comunicacions es complementa amb una aplicació per a ordinador d'escriptori. La seva finalitat és monitoritzar en temps real tots els vehicles de la xarxa i controlar-los. Aquesta aplicació es comporta com un element més de la xarxa. No fa les funcions d'un servidor i no és un element imprescindible del sistema de comunicacions.

S'estableixen les següents funcionalitats específiques:

- Poder rebre paquets d'estat de tots els vehicles de la xarxa alhora.
- Detectar automàticament els vehicles que emeten el seu estat. No s'ha de fer una configuració específica per a cada vehicle.
- Disposar d'una interfície gràfica.
- Permetre representar gràficament una variable emesa per un vehicle en temps real.
- Permetre enviar ordres a un o a diversos vehicles, i rebre les seves respostes.
- Permetre enregistrar les dades rebudes en un fitxer per al seu posterior tractament.
- Poder-se parametritzar mitjançant un fitxer de configuració.

Per al desenvolupament d'aquesta aplicació s'ha decidit usar el llenguatge de programació **Python**, degut a la gran quantitat de llibreries disponibles.

A l'annex A.4 es descriu el software i les llibreries usades. La seva parametrització es descriu a l'annex A.5.1. Una captura de pantalla de la interfície es mostra a la Figura 6-14.

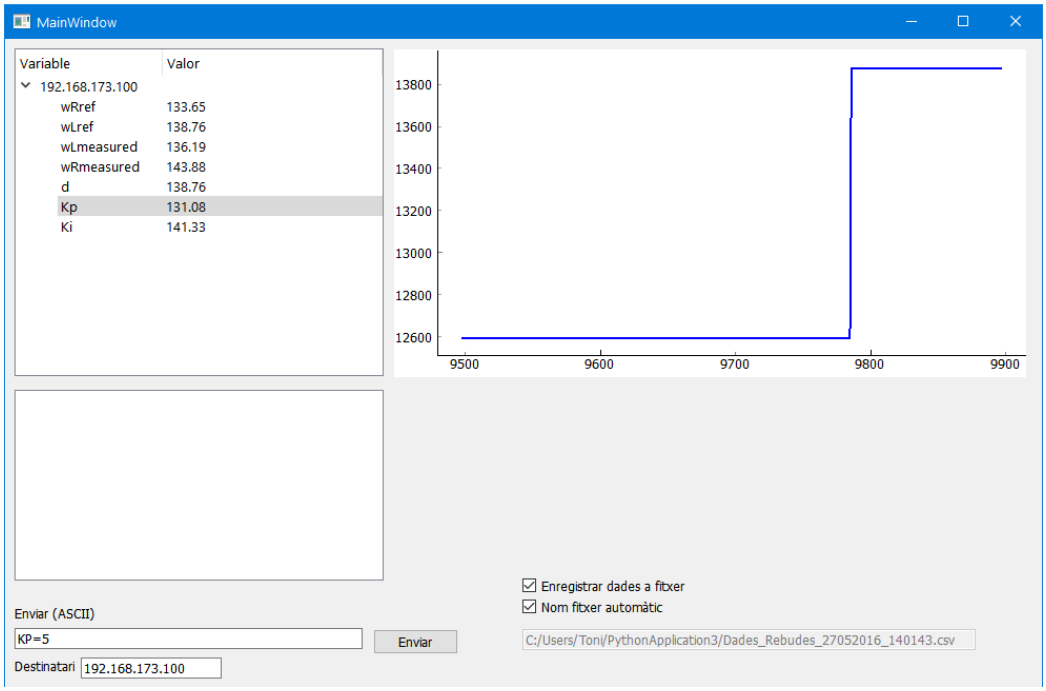

Figura 6-14. Interfície gràfica de l'aplicació auxiliar per a ordinador.

7. Planificació temporal i pressupost

La elaboració d'aquest treball acadèmic ha suposat 300 hores de dedicació, corresponents a 12 crèdits ECTS. Una part important d'aquest temps (60h) ha estat dedicat a la recerca, revisió de literatura, experimentació i determinació de les possibilitats del mòdul ESP8266. 50 hores han estat dedicades a la formació autònoma en coneixements necessaris per al desenvolupament exitós del treball. A la Taula 7-1 es resumeix el temps dedicat a cada tasca.

Tasca	Dedicació (h)
Anàlisi ESP8266	60
Definició especificacions	20
Desenvolupament software	120
Elaboració documentació	50
Formació	50

Taula 7-1. Dedicació temporal per tasques.

La distribució temporal d'aquestes tasques es mostra a la Figura 7-1, excepte la formació, que ha estat repartida durant tota la duració del treball però amb una major incidència en els primers mesos. En general aquesta planificació coincideix amb l'establerta a l'inici del treball, amb l'inici del desenvolupament situat un mes després de començar el projecte.

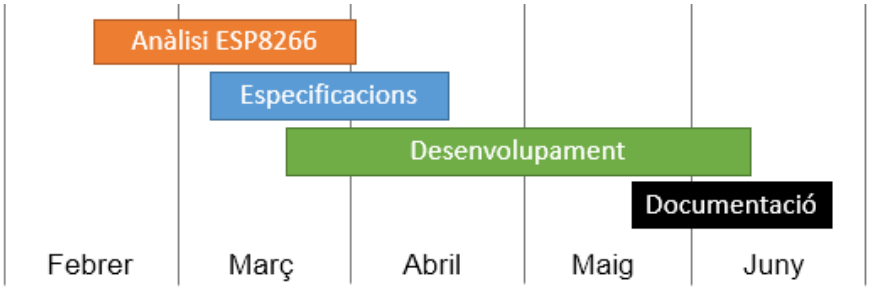

Figura 7-1. Distribució temporal per tasques, sense la formació.

L'elaboració d'aquest projecte no ha suposat despeses addicionals més enllà de l'adquisició dels materials de desenvolupament i l'ús ocasional de material de laboratori. El pressupost es resumeix a la Taula 7-2.

Tasca	Cost unitari (€)	Cost (€)
300 hores de treball	8	2400
50 hores d'ús de laboratori	10	500
1 Placa de desenvolupament STM32F4 Discovery	40	40
1 Mòdul WiFi ESP8266	6	6
	TOTAL	**2946**

Taula 7-2. Pressupost d'elaboració del projecte.

Conclusions

S'ha aconseguit implementar amb èxit el sistema de comunicacions de forma integral, sent la tasca més complicada, sens dubte, el desenvolupament d'un software per a governar el mòdul WiFi ESP8266 des d'un microcontrolador, sense influir en altres tasques que aquest pugui estar executant. El resultat ha estat molt positiu i aquesta implementació pot ser estesa a altres aplicacions i/o microcontroladors.

Aquest projecte començava amb l'anàlisi de les possibilitats que pot oferir el dispositiu ESP8266, i a partir dels resultats inicials es definien les especificacions generals. Tot i que el dispositiu no ha resultat estar a l'altura de les expectatives inicials (es contemplava usar-lo per a localització i comunicació descentralitzada), degut al seu baix cost és un dispositiu ideal per al seu ús com a eina d'experimentació, diagnòstic i desenvolupament en entorns de laboratori, si es poden assumir les limitacions de rendiment que presenta. Per altra banda, es desaconsella totalment el seu ús en entorns crítics o de producció.

El sistema de comunicacions està concebut com una capa base sobre la qual s'hi han d'implementar aplicacions. Amb el treball fet fins ara, l'usuari pot transmetre informació de forma totalment transparent entre dispositius integrats equipats amb el mòdul WiFi ESP8266 i ordinadors d'escriptori, per a qualsevol finalitat que sigui requerida en el seu projecte, sense haver de preocupar-se dels maldecaps que comporta definir com s'ho faran les dades per arribar a la seva destinació.

Agraïments

Dono les gràcies als tutors Víctor Repecho del Corral i Arnau Dòria-Cerezo per tota l'ajuda que m'han proporcionat durant la realització d'aquest TFG i per totes les hores que m'han dedicat. No esperava que en cinc mesos pogués aprendre tantes coses. Espero que els resultats hagin estat de la seva satisfacció.

També m'agradaria agrair a la meva família i amics la seva comprensió en els moments que no he pogut estar amb ells i tot el suport que m'han donat.

Aquest treball és la culminació de quatre anys d'aprenentatge. Tot i que no han fet una contribució directa en aquest treball, no puc oblidar-me de donar les gràcies a tots els professors que he tingut durant els meus estudis de grau per l'esforç que fan amb l'objectiu d'oferir assignatures amb continguts d'immensa qualitat.

Bibliografia

Referències bibliogràfiques

BENCHOFF, B. *The current state of ESP8266 Development.* 2015. [http://hackaday.com/2014/09/06/the-current-state-of-esp8266-development/; 25 de febrer de 2016]

EXPRESSIF SYSTEMS. *ESP8266 Datasheet.* Juny 2015.

EXPRESSIF SYSTEMS. *ESP8266 AT Instruction Set. Version 1.4.* Octubre 2015.

EXPRESSIF SYSTEMS. *ESP8266 Hardware Overview.* Febrer de 2016.

PRATS MARTIHO, IVAN. *Control design and implementation for a line tracker vehicle*. Juny de 2016. Pàgines 23-29.

PYTHON SOFTWARE FOUNDATION. *Python standard library documentation. Struct library*. 2016. [https://docs.python.org/2/library/struct.html; 20 de juny de 2016]

ST MICROELECTRONICS. *STM32F4 Discovery Datasheet*. Febrer de 2016.

TACKS, V.. *First experiences with the ESP8266.* Desembre de 2014. [http://diy.viktak.com/2014/12/first-experiences-with-esp8266.html; 25 de febrer de 2016]

Bibliografia complementària

EXPRESSIF SYSTEMS. *ESP8266 AT Command Examples*. 2015. [http://www.microtechnica.tv/support/manual/; 25 de febrer de 2016]

EXPRESSIF SYSTEMS. *ESP8266 Flash Tool Manual.* 2015. [https://www.espressif.com/en/content/esp8266-flash-tool-user-guide; 15 de març de 2016]

FAIRHURST, G. *The Internetwork Protocol (IP).* Novembre de 2008 [http://www.erg.abdn.ac.uk/users/gorry/eg3567/inet-pages/ip.html; 1 de juny de 2016]

NOVIELLO, C. *Mastering STM32.* February 2016.

PYQTGRAPH.ORG. *pyQtGraph documentation.* 2016
[http://www.pyqtgraph.org/documentation/; 10 d'abril de 2016]

PYTHON SOFTWARE FOUNDATION. *Python 2.7 documentation.* 2016.
[https://docs.python.org/2/; 10 d'abril de 2016]

SKAALI, T.B. *Concurrency and concurrent systems.* Febrer de 2011.

SNMPTOOLS.NET. *Network basics: Internet Protocol (IP).* 2007.
[http://www.snmptools.net/netbasics/ip/, 1 de juny de 2016]

ST MICROELECTRONICS. *UART Implementations*. Juny de 2007.

ST MICROELECTRONICS. *Using the STM32F0xx DMA controller.* Maig de 2012.

ST MICROELECTRONICS. *STM32F407 Datasheet*. Març de 2015.

ST MICROELECTRONICS. *RM0090 Reference manual*. Octubre de 2015.

W3SCHOOLS. *The TCP/IP protocol.* Febrer 2016
[http://www.w3schools.com/website/web_tcpip.asp, 3 de març de 2016]

Annex A. Manual d'usuari

El software desenvolupat durant la realització d'aquest treball pot resultar útil en entorns de producció, però ha estat creat amb l'objectiu principal de ser utilitzat com una eina d'experimentació, diagnòstic i desenvolupament. Aquest aspecte és important a l'hora de definir qui és l'usuari final.

Es considerarà que l'usuari és l'enginyer que està dissenyant una aplicació per al sistema integrat. Aquesta guia, per tant, s'enfoca a la implementació del software de comunicacions: instal·lació, compilació, configuració, parametrització i possibles ampliacions de funcionalitat. Conseqüentment, els continguts aquí exposats pressuposen un mínim de coneixement sobre la plataforma en la qual es vol implementar aquest software, i s'han de complementar amb la resta de la memòria. Les instruccions a seguir es detallen per a una placa de desenvolupament STM32F4 Discovery, però s'indiquen les accions necessàries per a poder-lo implementar en sistemes diferents.

També s'explica el funcionament del software client per a ordinador i es donen instruccions per a configurar-lo. El seu ús, en principi, no necessita coneixements avançats, sempre que s'hagi implementat correctament. Amb aquesta memòria s'adjunta també com a exemple la implementació del software en un vehicle seguidor de línia.

A.1 Descripció dels fitxers

- Carpeta ***client***: Conté el codi font en C per a compilar el software de comunicacions per a qualsevol dispositiu integrat. Aquest manual conté indicacions per a fer-ho.
    - ***comunicacions.c***: Fitxer que conté totes les funcions necessàries per a gestionar les comunicacions del dispositiu. Es compilen en un únic binari.
    - ***comunicacions.h*** Fitxer de capçalera on es defineixen les estructures de dades, i lloc únic on ajustar paràmetres relatius a les comunicacions.
    - ***main.c*** Exemple de fitxer principal que només conté la inicialització del programa de comunicacions i un exemple de la seva implementació.

- Carpeta ***SeguidorLinia***: Conté tots els fitxers necessaris per a compilar el software del vehicle seguidor de línia presentat al prefaci d'aquesta memòria, utilitzant com a base un DSP STM32F4 i les llibreries HAL. En el cas de voler començar un projecte nou amb aquests elements, també és més fàcil partir d'aquesta plantilla que començar de zero.

- Carpeta ***servidor***: Conté un script *Python* per a executar un servidor de xarxa en un ordinador (Windows, Mac, Linux). Inclou una interfície gràfica que permet mostrar en pantalla totes les variables d'estat dels dispositius connectats i enviar comandes a qualsevol dispositiu. La seva funcionalitat és fàcilment ampliable. El seu ús es descriu a l'apartat A.4 d'aquest manual.

    - ***main.py*** Programa de mostra que executa una interfície gràfica i permet portar a terme diverses tals com enviar i rebre dades, representar-les gràficament o enregistrar-les en un fitxer. Utilitza les capacitats del programa ***servidor.py***.
    - ***servidor.py*** Software que fa les funcions de servidor de xarxa, i empaqueta i desempaqueta les dades per a poder ser tractades amb facilitat per qualsevol script Python.
    - ***parametres.txt*** Fitxer que conté la configuració del servidor, inclosa la composició dels paquets. Cal ajustar-lo en modificar les variables d'estat que envia el client, procediment que s'explica a l'apartat A6.
    - ***baseInterficieGracia.py*** Fitxer base per a dibuixar la interfície gràfica (localització dels botons, mida de finestra, etc). Es pot generar automàticament amb el programa QtDesigner, inclòs amb la majoria d'instal·lacions científiques de Python.

A.2 Instal·lació

Amb el paquet s'inclou el software integrat en una aplicació per al control d'un vehicle seguidor de línia, per al dispositiu STM32F4.

En el cas que el lector necessiti incorporar el software de comunicacions en una aplicació distinta o en un dispositiu diferent, aquestes són les passes generals a seguir. Es detallen per a un dispositiu que utilitzi llibreries HAL, però s'indiquen els aspectes que s'han de cuidar si es vol instal·lar en un altre tipus de dispositiu.

1. Afegir els fitxers ***comunicacions.c*** i ***comunicacions.h*** al projecte.
2. Incloure el fitxer ***comunicacions.h*** al programa principal. (normalment, al fitxer ***main.c***).

   Si no es fa servir un dispositiu STM32F4, modificar ***comunicacions.h*** per tal que no inclogui els fitxers ***stm32f4xx_hal.h*** ni ***stm32f4_discovery.h***, afegint en el seu lloc llibreries que continguin els controladors del port sèrie per al dispositiu corresponent.

3. Configurar un port UART (sèrie) que pugui enviar i rebre sense bloquejar completament la CPU durant el procés, ja sigui per DMA o mitjançant interrupcions.

   Amb la majoria de sistemes integrats aquest codi es pot generar automàticament. En el cas d'un dispositiu amb llibreria HAL, cal utilitzar el software STM32CUBE.

   A l'hora de triar entre DMA o interrupcions s'han de tenir en compte algunes consideracions:

   - En cas de rebre dades per interrupcions, la CPU ha de copiar cada byte del registre d'arribada del port UART (Rx) un a un, a cada instant que arriben (o, com a molt tard, abans de què arribi el següent). Aquesta operació triga fraccions de microsegon (depèn del dispositiu), però s'ha d'executar amb prioritat alta. En concret, no es pot deixar cap byte sense copiar. Això implica garantir-ne l'execució cada 80 µs per a una transmissió a 112500 bauds.

   - En el cas d'utilitzar DMA, no es necessita la CPU. Cal configurar un doble buffer, ja que amb un buffer senzill generalment no es pot llegir i escriure alhora. Això pot donar problemes amb transmissions de mida petita a altes freqüències, però depèn del dispositiu i de la configuració concreta. Per aquest motiu, es recomana utilitzar interrupcions.

4. Configurar la còpia de les dades rebudes per aquest port UART al buffer d'entrada.

   Les dades rebudes es copien a un buffer d'entrada per a poder ser tractades de manera asíncrona, independentment de l'instant d'arribada i de si la CPU està ocupada amb tasques de més prioritat.
   Els bytes rebuts per UART s'han d'anar copiant, un per un o en blocs, a aquest buffer. No cal preocupar-se del cronometratge però l'ordre ha de ser estrictament el d'arribada.

   Per a escriure, cal cridar la funció ***escriuBufferFIFO***. El nom del buffer d'entrada és ***bufferEntradaUART***. Normalment es cridarà dins de la interrupció generada per dades rebudes. En el cas d'utilitzar la llibreria HAL, aquesta interrupció crida la funció ***HAL_UART_RxCpltCallback()***.

   Al Fragment de codi 0-1 es mostra un exemple d'implementació en aquesta mateixa crida d'interrupció. El segon paràmetre és l'adreça de memòria de les dades rebudes i el tercer la mida (es pot veure la definició completa d'aquesta funció a l'annex 2).

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle)
{
  //...
  escriuBufferFIFO(&bufferEntradaUART, &caracterRx, 1);
  //...

}
```

Fragment de codi 0-1. Línia de codi que efectua la còpia de dades al buffer d'entrada. En aquest cas s'ha afegit a la interrupció de dades rebudes pel port sèrie. El fitxer normalment és stm32f4xx_it.c o main.c.

5. Configurar l'enviament de dades mitjançant aquest port UART.

   Cal modificar la funció ***enviaUart()*** del fitxer ***comunicacions.c.*** S'ha d'afegir dins del ***return()*** una crida a la funció d'enviament de dades. S'han d'enviar les dades contingudes dins de ***&paquetSortida***, amb el número de bytes contingut a ***mida*** (entre 1 i 255). Com s'explica a l'inici d'aquest annex, és imperatiu triar un mètode d'enviament que no bloquegi la CPU (DMA o UART). El programa garanteix que el contingut de l'adreça de memòria ***&paquetSortida*** no canviarà

fins que no hagi acabat la transmissió. En el Fragment de codi 0-2 s'ha subratllat la línia que cal canviar.

```
/**
  * Enviar dades per UART
  * Paràmetres:
  *   pDadesTx:   Punter a dades a transmetre
  *   mida:       Nombre de bytes a transmetre
  */
int enviaUart(uint8_t *pDadesTx, int mida) {
      /* Cada byte s'enviarà per interrupció. Cal copiar les
dades a una variable global. */
      memcpy(&paquetSortida, pDadesTx, mida);
      return(HAL_UART_Transmit_IT(&UartHandle,
&paquetSortida, mida));

}
```

Fragment de codi 0-2. Funció enviaUart() amb el fragment de codi subratllat que defineix el mètode d'enviament de dades pel port sèrie. Fitxer comunicacions.c.

6. Afegir al cos del programa principal les ordres d'arrancada de la connexió i crides contínues a la rutina ***COM_gestionarComunicacions***.

   La rutina ***COM_gestionarComunicacions***, en cridar-la, processa els buffers d'enviament i recepció i s'encarrega de controlar tota la part de comunicacions. Es pot executar en qualsevol moment, i està dissenyada perquè pugui ser interrompuda. Evidentment, però, si no es deixa suficient temps lliure de CPU es perdran paquets. (això sí, sense interferir amb el programa principal)

   Hi ha diverses estratègies a seguir. Normalment les accions prioritàries que ha d'executar el sistema integrat (per exemple, el control) s'executen a períodes fixes mitjançant interrupcions de rellotge, i el cos del ***main*** té menor prioritat.

   La forma més fàcil de fer-ho és incloure-la dins el ***while(1)*** del ***main***, a la mateixa alçada d'altres rutines no prioritàries, com es mostra al Fragment de codi 0-3.

```
inicialitzarUart();
configurarConnexioWifi();
establirConnexio();
inicialitzaBufferFIFO(&bufferEntradaUART);
while (1)
{
      COM_gestionarComunicacions();
      //Altres rutines de prioritat baixa

}
```

Fragment de codi 0-3. Funcions d'arrancada que cal cridar i exeple d'implementació situant COM_gestionarComunicacions() al while(1). Fitxer main.c.

Les quatre rutines que es mostren a sobre del ***while(1)*** s'han de cridar per a inicialitzar el software de comunicacions. Es pot veure la seva definició a l'annex 2. Si no s'utilitzen llibreries HAL, s'ha de modificar la funció ***inicialitzaUart()*** i incloure les ordres d'arrancada del port UART.

7. Parametritzar el software de comunicacions segons les necessitats específiques de l'usuari. A l'apartat 3 d'aquest manual s'indica quines variables es poden ajustar.

8. Afegir les funcionalitats requerides per l'usuari.

- A l'apartat A.5.1 s'explica com triar quina informació es transmet de forma contínua ("variables d'estat") i com configurar el software servidor per a poder-la llegir correctament.

- A l'apartat A.5.2 s'explica com enviar un paquet de xarxa amb qualsevol contingut, a qualsevol destinatari, sense interferir amb l'execució del programa principal.

- A l'apartat A.5.3 d'aquest manual s'explica com modificar el software per a que pugui admetre noves comandes enviades des del servidor (o un altre vehicle/sistema) i hagin d'executar accions en el receptor.

A.3 Paràmetres de configuració

Es poden ajustar diversos paràmetres per a afinar el software de comunicacions a les necessitats de l'usuari i a l'entorn. Es classifiquen en paràmetres funcionals, de xarxa i de perifèric. Tots es defineixen al fitxer ***comunicacions.h***.

En una nova instal·lació usant els elements de referència (sistema integrat STM32F4 i mòdul WiFi ESP8266 amb firmware 1.4.1 oficial), només és imprescindible configurar les dades de connexió a la xarxa.

A.3.1 Paràmetres funcionals

Relatiu al funcionament del sistema, és important calcular i indicar el número de bytes necessaris del paquet d'estat i la mida màxima dels paquets que s'enviaran i es rebran. Els paràmetres funcionals es descriuen a la Taula 0-1.

Si hi ha restriccions relatives a la memòria RAM del dispositiu, es pot afinar la mida dels buffers d'entrada i sortida de dades. Per defecte es reserven 2Kb, que podrien ser reduïts notablement si fos necessari. De mitja, els buffers es buiden més ràpid del que s'omplen, però una mida de buffer que pugui suportar 10-20 paquets és important per a absorbir pics de demanda de capacitat de comunicació i/o períodes d'activitat extraordinària per part de la CPU. En cas de sobreeiximent d'algun buffer, es descartaran puntualment alguns paquets, però no afectarà a l'estabilitat del sistema.

A.3.2 Paràmetres de xarxa

S'han de definir les dades de la connexió WiFi que s'usarà per a les comunicacions (nom, contrasenya, ...) i la direcció IP que tindrà el dispositiu. Aquesta ha de ser obligatòriament única perquè s'usa per identificar el dispositiu, tant a nivell de xarxa com per part del software servidor. Per tant, a l'hora de desplegar el mateix software en més d'un dispositiu s'ha d'anar canviant aquest valor en cada còpia. Els paràmetres de xarxa es descriuen a la Taula 0-2.

A.3.3 Paràmetres de perifèric

El perifèric WiFi ESP8266 presenta un comportament que varia molt àmpliament segons la revisió exacta i firmware del dispositiu. Alguns valors (per exemple, el temps de processament de comanda) s'han de determinar experimentalment. Els paràmetres que venen per defecte s'han determinat i optimitzat per a la versió de firmware 1.4.1 oficial del fabricant, i no es recomana tocar-los. En el cas que s'utilitzi una versió distinta o es vulguin afinar, es descriuen a la Taula 0-3.

Paràmetres funcionals		
Paràmetre	**Valor per defecte**	**Descripció i unitats**
MIDA_BUFFER	1024	Longitud, en bytes, dels buffers d'entrada i sortida del port sèrie. Contenen els paquets de xarxa en una sintaxi especial i també les ordres que s'envien i es reben dispositiu WiFi. Com a mínim, ha de ser 10-20 cops la longitud mitjana d'un paquet de dades. Augmentar si es preveuen pics de demanda de capacitat de comunicació i/o períodes amb temps de CPU extraordinàriament reduït. El sobreeiximent d'un buffer només implica la pèrdua puntual de paquets.
MIDA_MAXIMA_ENTRADA	128	Mida màxima en bytes que pot tenir una línia d'entrada del port sèrie. Màxim 255 bytes, ajustar al màxim per estalviar memòria. Ha de ser, com a mínim, 8 bytes superior a *MIDA_MAXIMA_PAQUET.*
MIDA_MAXIMA_PAQUET	120	Mida màxima en bytes que pot portar un paquet de xarxa (TCP/UDP).
MIDA_PAQUET_ESTAT	20	Mida en bytes del paquet que conté les variables d'estat. El seu valor està acotat entre 10 i 255.

Taula 0-1. Descripció dels paràmetres funcionals definits al fitxer comunicacions.h.

Paràmetres de xarxa		
Paràmetre de xarxa	**Valor per defecte**	**Descripció i unitats**
IP	192.168.173.100	Adreça IP del dispositiu. Cada dispositiu s'identificarà mitjançant aquesta adreça.
IP_SERVIDOR	192.168.173.1	Adreça IP del servidor (ordinador que executa el software d'escriptori), si es fa servir.
MASCARA_XARXA	255.255.255.0	Defineix el rang de direccions IP possibles.
SSID_WIFI	wireless	Nom identificador de la xarxa WiFi a la qual s'ha de connectar el dispositiu.
CONTRASENYA_WIFI	xarxaesp8266	Contrasenya de la xarxa WiFi
PORT_TCP	6780	Port TCP que s'usarà per a connexions amb el servidor.
PORT_UDP	6789	Port UDP que s'usarà per a totes les connexions.

Taula 0-2. Descripció dels paràmetres de xarxa definits al fitxer comunicacions.h.

Paràmetres del perifèric		
Paràmetre	**Valor per defecte**	**Descripció i unitats**
PERIODE_ENVIAMENT_DADES_UART	15	Separació mínima en milisegons entre l'enviament de dues comandes successives al perifèric WiFi esp8266. Aquest valor varia àmpliament segons el firmware i s'ha de determinar experimentalment. El valor de 15ms correspon a la versió 1.4.1 oficial del fabricant.
PERIODE_ENVIAMENT_ESTAT	40	Període en milisegons d'enviament del paquet que conté les variables d'estat. Per a enviar un paquet de dades s'han d'enviar dues comandes al perifèric, per tant, aquest valor ha de ser obligatòriament superior al doble de *PERIODE_ENVIAMENT_DADES_UART*.
USART_PORT	USART2	Nom del port sèrie que s'ha configurat (només per a llibreria HAL).
BAUDRATE	115200	Bauds per segon de la comunicació sèrie. Depèn de la configuració del perifèric WiFi.

Taula 0-3. Descripció dels paràmetres del perifèric definits al fitxer comunicacions.h.

A.4 Instal·lació i ús del Software per a ordinador

A.4.1 Python i paquets necessaris

Per a executar programari en Python, cal tenir instal·lat un intèrpret. En el cas d'usar Windows, es recomana instal·lar la distribució Python(x,y) perquè inclou nombroses llibreries d'ús científic.

https://python-xy.github.io/

En cas de instal·lar una distribució alternativa, cal que aquesta inclogui la llibreria pyQt.

Addicionalment, cal instal·lar la llibreria pyqtgraph. La instal·lació és automàtica amb el fitxer.exe.

http://www.pyqtgraph.org/

A.4.2 Ús del software

Per a executar el programari, s'ha de fer doble clic a main.py.

El software mostrarà automàticament tots els vehicles que estiguin connectats a la mateixa xarxa que l'ordinador, i estiguin emetent alguna informació cap a l'adreça IP de l'ordinador. Com s'ha explicat a l'apartat A.3, aquesta configuració es fa en el paràmetre IP_SERVIDOR del fitxer comunicacions.h.

A la Figura 0-1 es mostra la finestra principal del programa.

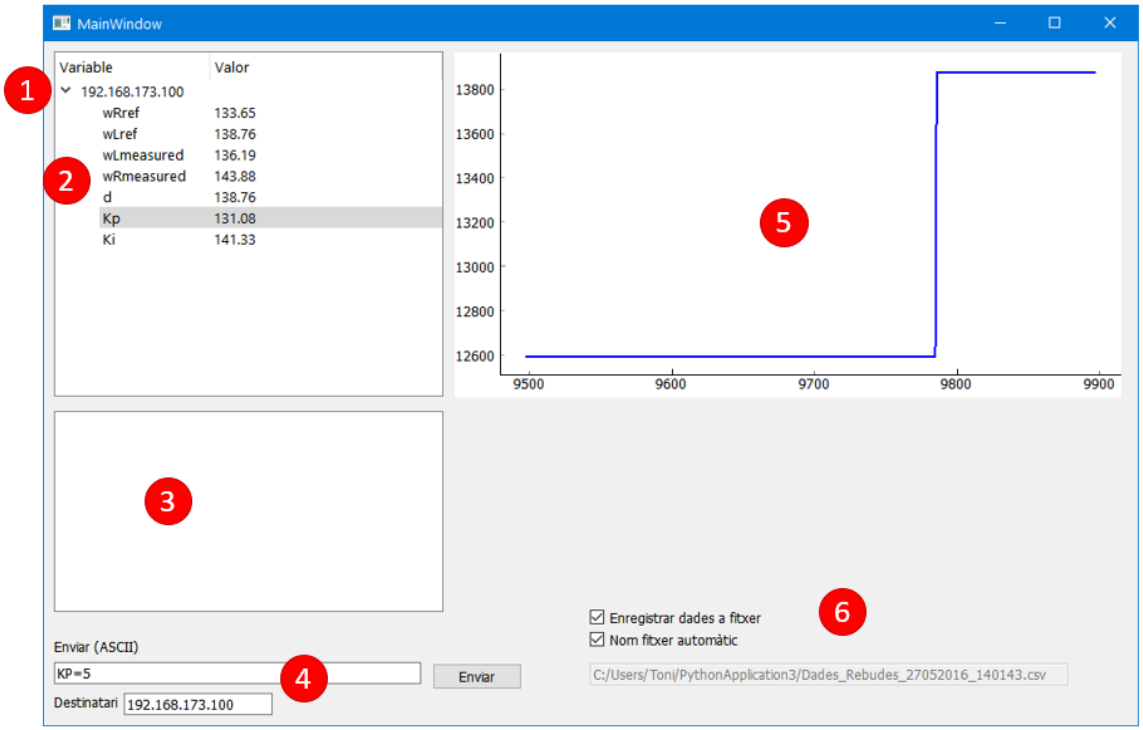

Figura 0-1.Finestra principal del software per a ordinador d'escriptori.

A continuació, s'explica cada element enumerat en la Figura 0-1**.**

1. En aquest panell apareix cada vehicle que estigui transmetent informació cap al servidor. L'adreça IP de cada vehicle s'utilitza com a identificador. Cada vehicle es presenta com una llista desplegable.

2. Fent doble clic sobre una adreça IP es mostren totes les variables que el vehicle corresponent emet a temps real. Fent dobe clic sobre una variable fa que aquesta es representi gràficament.

3. Historial d'ordres enviades i rebudes.

4. Quadre de text que serveix per a enviar comandes (ordres) a un o varis vehicles.

5. Gràfic que mostra en temps real el valor d'una variable.

6. Tauler de control que serveix per a gestionar l'enregistrament de totes les variables rebudes en un fitxer.

A.4.2 Crear una xarxa WiFi amb Windows (opcional)

Si s'utilitza el sistema operatiu Windows i un ordinador amb connectivitat WiFi, es pot aprofitar aquest com a enrutador i punt d'accés WiFi, així no s'ha d'usar un dispositiu dedicat. Es pot estar connectat alhora a una segona xarxa que proporcioni internet.

Cal obrir una línia de comandes com a administrador (amb el botó esquerre sobre el botó inici) i executar les següents comandes:

- **`netsh wlan set hostednetwork mode=allow ssid="NOM_SSID" key=PASSWD keyUsage=persistent`**
- **`netsh wlan start hostednetwork`**

On NOM_SSID és el nom de la xarxa que es crearà i PASSWD la contrasenya.

Ara, anant a Tauler de control -> Xarxa i internet -> Centre de xarxes i recursos compartits apareix aquesta xarxa. L'adreça IP de l'ordinador és diferent per a cada xarxa i es pot consultar en aquesta finestra.

La xarxa es desconnecta automàticament en apagar l'ordinador o al cap d'un temps d'inactivitat, però la configuració no s'esborra. En qualsevol moment es pot connectar, desconnectar i veure el número de clients amb aquestes comandes:

- **`netsh wlan start hostednetwork`**
- **`netsh wlan show hostednetwork`**
- **`netsh wlan stop hostednetwork`**

A.5 Afegir i modificar funcionalitat

A.5.1 Modificar enviament de variables d'estat

Per afegir o treure variables d'estat (les variables que es transmeten en temps real a l'ordinador) cal fer una modificació en el software del microcontrolador, i modificar un fitxer de configuració en el software d'ordinador.

Les variables d'estat s'envien en un paquet d'estat, amb la codificació definida a l'apartat 6.1. Cada variable es distingeix per la seva posició dins del paquet, i cada variable pot disposar d'un número de bytes diferent i representar un tipus d'estructura de dades diferent (enter, float, etc).

Degut a la naturalesa del microcontrolador, les modificacions s'han de fer a baix nivell, tocant el codi del fitxer ***comunicacions.c***. S'ha de modificar la funció ***enviarEstatUART()***, la qual es mostra al Fragment de codi 0-4.

```
int enviarEstatUart() {
	/* ***** Envia l'estat directament per UART. ****** */

	// Definir MIDA_PAQUET_ESTAT al fitxer comunicacions.h


	uint8_t dadesPaquet[MIDA_PAQUET_ESTAT] = {'E'}; // 'E' caràcter
	de control (estat).


	int int1 = arrodonirFloat(wRref);

	// No tocar el byte 0. Conté un caràcter de control.
	memcpy(&dadesPaquet[1], &int1, sizeof(int));
	memcpy(&dadesPaquet[3], &float1, sizeof(int));


	/* Aquí ja s'envia directament el paquet. NO es posa en cua.
(Ja estava en cua!) */

	enviaUart(&dadesPaquet, MIDA_PAQUET_ESTAT);

	return(0);
}
```

Fragment de codi 0-4. Funció enviarEstatUart(). Fitxer comunicacions.c.

Aquesta funció crea un array de bytes anomenat **dadesPaquet**, que és el contingut del paquet d'estat. El primer caràcter és una 'E', com s'ha definit a l'apartat 4.2. Les dades que es vulguin enviar s'han de situar en aquest array.<

Per a afegir una variable cal copiar-la directament a aquest array mitjançant la funció **memcopy**. Es pot prendre com a exemple un dels casos ja existents.

Per exemple, per a afegir l'enviament d'un enter (int2) a la situació exposada en el Fragment de codi 0-4 s'han de seguir les següents passes:

1. Triar una posició en l'array, normalment s'agafarà el primer byte lliure després de l'últim valor. A l'exemple (Fragment de codi 0-4), el darrer valor que es copia a l'array (**float1**) comença a la posició 3. Com que aquest valor és de tipus ***float***, ocupa 4 bytes de l'array **dadesPaquet**. Llavors, la posició que ocuparà aquest enter serà la 3+4=7.

2. Determinar la mida del paquet d'estat complet. Com que l'últim valor ocupa la posició 7, i és del tipus ***int*** (2 bytes), el paquet d'estat sencer necessita com a mínim 9 bytes. Cal comprovar que el paràmetre *MIDA_PAQUET_ESTAT* del fitxer ***comunicacions.h*** és igual o superior a 9. (Veure apartat A.3).

3. Afegir una línia de codi per a copiar el nou valor a l'array, a la posició que s'ha determinat (7) (sota les dues línies subratllades a l'exemple)
```
memcpy(&dadesPaquet[7], &int2, sizeof(int));
```

A continuació cal configurar el programari per a ordinador per tal que entengui que a la posició 7 arriba un enter. En aquest cas no cal modificar cap línia de codi, sinó que s'ha de modificar el fitxer ***paràmetres.txt***. El contingut d'aquest fitxer es mostra al Fragment de codi 0-5, amb la nova línia ja afegida, la sintaxi i les instruccions.

```
# Fitxer de definicio de la codificacio dels paquets d'estat
# No poseu accents

# Sintaxi:
#
# NOM_PAQUET POSICIO_INICIAL POSICIO_FINAL TIPUS_DADES DIVIDIR
#
# Separar els valors amb espais o tabuladors.
#
# TIPUS DADES: 'h' enter de 2 bytes, 'f' float estandard (4 bytes)
# Altres tipus: https://docs.python.org/2/library/struct.html
# DIVIDIR: El valor es dividira per aquest numero (posar 1 si no s'ha de
dividir)


int1                1       3       h       100
float1              3       7       f       1
int2                7       9       h       1
```

*Fragment de codi 0-5. Fitxer **parametres.txt** amb l'enter de l'exemple int2 afegit (destacat en negreta)*

El quart valor de cada fila (*TIPUS_DADES*) és un caràcter clau que identifica el tipus de dada per tal que pugui ser parsejat correctament, com s'explica a la figura. ('h' enter de 2 bytes, 'f' float estandard (4 bytes), etc). A la Taula 0-4es mostren més caràcters clau per al valor *TIPUS_DADES*.

Caràcters clau per a identificar tipus de dades en el fitxer paràmetres.txt			
Caràcter clau	**Tipus de dada en C**	**Tipus de dada en Python**	**Bytes**
c	char	string	1
b	signed char	integer	1
B	unsigned char	integer	1
?	_Bool	bool	1
h	short	integer	2
H	unsigned short	integer	2
i	int	integer	4
I	unsigned int	integer	4
l	long	integer	4
L	unsigned long	integer	4
q	long long	integer	8
Q	unsigned long long	integer	8
f	float	float	4
d	double	float	8
s	char[]	string	

*Taula 0-4. Caràcters clau per a identificar tipus de dades en el fitxer **parametres.txt**, amb la seva equivalència en C i en Python, en sintonia amb els de la llibreria Struct (Python Software Foundation, 2016)*

A.5.2 Enviar un paquet de xarxa arbitrari codificat en ASCII

L'enviament d'un paquet de xarxa suposa l'enviament de dues comandes consecutives al mòdul WiFi ESP8266, en el moment precís i separades entre elles un temps determinat. Tota aquesta gestió la fa el programa automàticament. Només s'ha d'executar una única funció ***enviarASCII()***, la definició i instruccions d'ús de la qual es mostra al Fragment de codi 0-6. L'enviament del paquet es posa en cua i serà enviat tan bon punt sigui possible. Aquesta funció pot ser cridada en qualsevol punt, tant en el programa principal com a la rutina de comunicacions.

```
/*
 * Posar en cua l'enviament d'un paquet de xarxa (contingut en
      ASCII), per la connexio especificada.
 *
 * pDadesTx: Punter a les dades a transmetre. (adreça de memòria de
      l'element 0 de l'array). Les dades es copiaran immediatament,
      no cal conservar el valor en memòria fins que s'hagin enviat.
 * Mida:      Numero de bytes que s'han d'enviar.
 * Connexio:  Identificador de la connexio (0 = ordinador).
 *
 */
int enviarASCII(uint8_t *pDadesTx, int mida, int connexio);
```

*Fragment de codi 0-6. Definició de la funció **enviaAscii()**. Fitxer comunicacions.c.*

A.5.3 Afegir ordres

Una ordre és un paquet especial que fa que el vehicle executi una acció preprogramada. Opcionalment, cada ordre pot portar un paràmetre. La codificació exacta del paquet de xarxa corresponent a una ordre s'ha establert a l'apartat 6.1.

No cal cap modificació en programa d'escriptori. Aquest incorpora un quadre on s'escriuen les ordres a enviar.

S'ha de modificar el software del microcontrolador per a canviar les ordres que aquest admet. Degut a la seva naturalesa, les modificacions es fan directament en el codi font, al fitxer ***comunicacions.c***, i s'han d'adaptar a cada escenari en concret.

Cal modificar la funció **tractarPaquet(),** dins de l'expressió condicional "`else if (paquet[0] == '>')`", tal i com es mostra a la FIGURA X. Les ordres es transmeten per la xarxa pel caràcter ">" seguit del seu nom, tal i com es defineix a l'apartat 6.1.

Cal afegir una sentència condicional (if) que avaluï el contingut del paquet, incloent el primer caràcter '>' i els dos últims caràcters de salt de línia "/r/n", i el compari amb el nom de cada ordre. Com a exemple s'inclou la funció LED6, que encén i apaga un led, per tal que es pugui usar de referència. Es pot copiar a sota i canviar el nom de LED6 per un altre i les mides dels memcpy. A la FIGURA X es mostra aquest exemple.

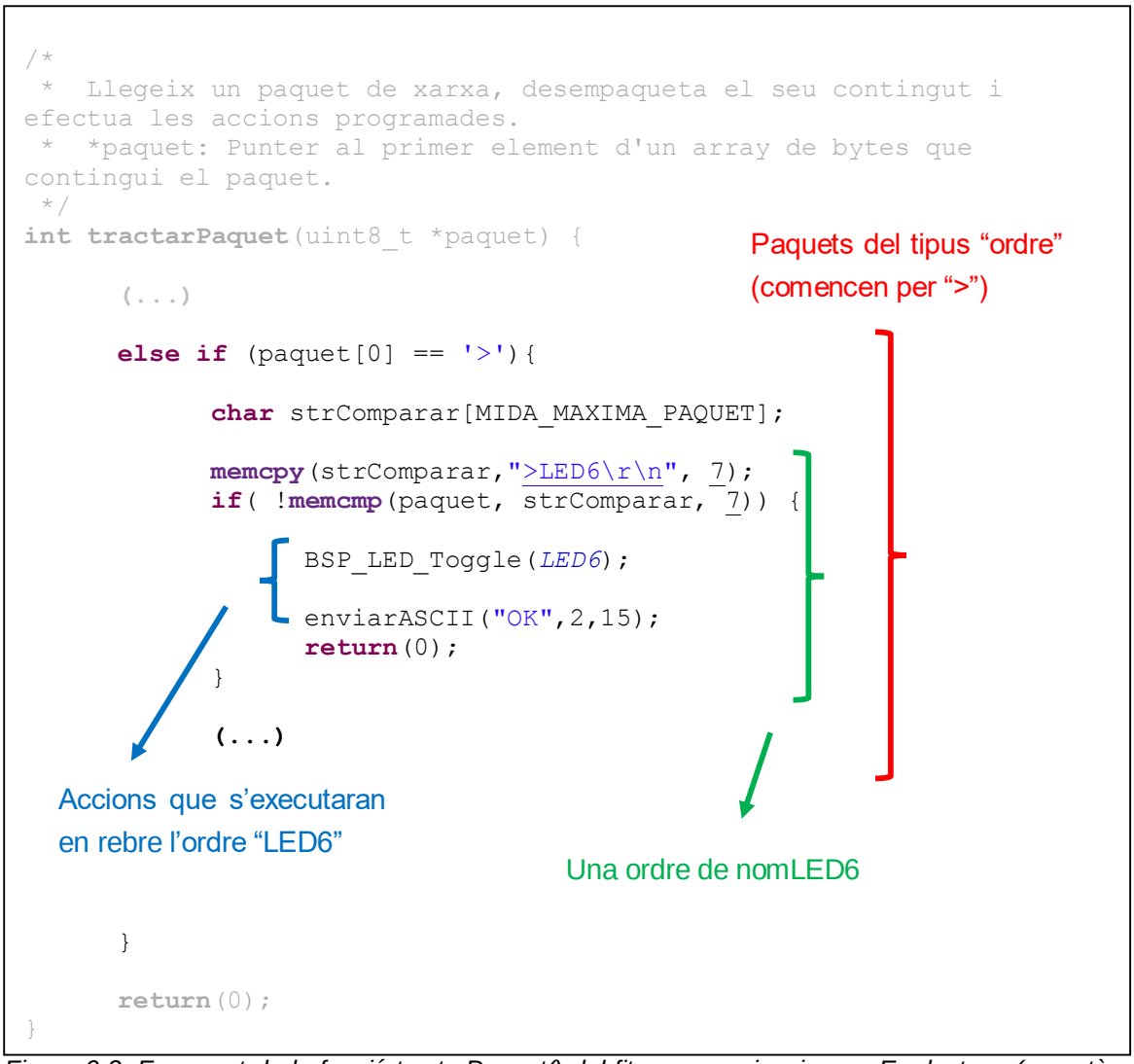

Figura 0-2. Fragment de la funció tractarPaquet() del fitxer comunicacions.c. Es destaca (no està en gris) la part encarregada de llegir els paquets tipus "ordre" (comencen per ">"). Es mostra com a exemple una ordre que encén i apaga 'un LED quan rep l'ordre "LED6" perquè es pugui agafar de base (copiant-la i enganxant-la). S'han subratllat els tres elements que cal canviar

Les accions s'executaran immediatament quan es tracti el paquet. Com que no hi ha manera de saber l'instant exacte en què això passarà, es recomana que en aquest punt només es modifiqui el valor de variables de bandera (p. ex. encès=1), i aquestes variables siguin llegides en el programa principal per a executar allà les accions necessàries.

Normalment resulta útil incloure aquí l'enviament d'una resposta amb la funció **enviaASCII()** introduïda a l'apartat A.5.2, segons el resultat de l'ordre (OK, ERROR, etc.)

Existeix una petita possibilitat de què el programa sigui interromput mentre executa totes les accions. En el cas que sigui imprescindible executar-les totes seguides, cal desactivar les interrupcions momentàniament.

És possible crear ordres amb paràmetres. Cal detectar només l'inici de l'ordre, sense el retorn de línia ('/r/n') i incloure el pàrsing d'aquest paràmetre entre les accions que s'executen en rebre l'ordre. Aquest procediment depèn dels objectius concrets de l'usuari.

Annex B. Codi font

B.1 Fitxer comunicacions.c

```
#include "comunicacions.h"
#include "string.h"

UART_HandleTypeDef UartHandle;


uint8_t paquetSortida[MIDA_MAXIMA_PAQUET] = {0};


int entradaA, entradaB;

int temporitzador = 0;



/* Variables globals a transmetre o modificar */
extern float wRref;
extern float wLref;
extern float wLmeasured;
extern float wRmeasured;
extern float d[2];
extern float ki;
extern float kp;

extern uint8_t start;

/* BUFFERS */

BufferFIFO bufferEntradaUART;
BufferFIFO bufferSortidaUART;

/* Definit al fitxer de capcelera */
/*
typedef struct _BufferFIFO {
      uint8_t dades[MIDA_BUFFER];
      uint16_t longitud;                  // Longitud del buffer
      uint16_t pEntrada;                  // Punter escriptura
      uint16_t pSortida;                  // Punter lectura
} BufferFIFO;
*/

/* TODO Treure això d'aquí */



int COM_gestionarComunicacions() {
      /* Executar aquesta funció contínuament però amb prioritat
baixa. Pot ser interrompuda en qualsevol moment.
```

```
       * HAL_GetTick() retorna el valor del rellotge intern en
milisegons (contant a partir del HAL_Init() a l'engegada)   */


      static uint32_t tickEnviamentDadesUART = 0;
      static uint32_t tickEnviamentEstat = 0;


      /* Tractar un element del buffer de dades rebudes */
      HAL_GPIO_WritePin(GPIOB,GPIO_PIN_11,GPIO_PIN_SET);
      while(tractarDadesRebudesUART() == 99);         /* Tractar
dades. Retorna 99 quan el buffer conté paquets que no hem de tractar.
*/
      HAL_GPIO_WritePin(GPIOB,GPIO_PIN_11,GPIO_PIN_RESET);

      /* Enviar un element del buffer de comandes pendents per
enviar. */
      if((HAL_GetTick() - tickEnviamentDadesUART) >
PERIODE_ENVIAMENT_DADES_UART) {
            enviarBufferUART();
            tickEnviamentDadesUART = HAL_GetTick();
      }

      /* Posar en cua (al buffer de comandes pendents per enviar) un
enviament de l'estat del microcontrolador */
      if((HAL_GetTick() - tickEnviamentEstat) >
PERIODE_ENVIAMENT_ESTAT) {
            posarCuaEnviamentEstat();
            tickEnviamentEstat = HAL_GetTick();
      }


      /* Ampliacions o estratègies alternatives */
      /*
      if( (temporitzador % PERIODE_TRACTAMENT_DADES_REBUDES) == 0) {
            tractarDadesRebudesUART();
      }
      if(temporitzador % PERIODE_ENVIAMENT_DADES_UART == 0) { // 400

            enviarBufferUART();
      }

      if(temporitzador % PERIODE_ENVIAMENT_ESTAT == 0) {    // 2000
(100ms)
            // Posem l'enviament en cua directament. Temporal
            temporitzador = 0;
            uint8_t arrayEstat[] = "+++ESTAT\r\n";
            enviarASCII(&arrayEstat,10,15);
      }
      */
      /* Ampliar aquí per a afegir nous períodes (moure
<temporitzador = 0> si escau) */
      return(0);
}

int posarCuaEnviamentEstat() {
      /* TODO nom funció provisional */
```

```
	// Per a enviar l'estat, posem al buffer de sortida "+++ESTAT
\n" amb la mida del paquet i acabat en \n.
	// Fem això perquè al buffer només podem emmagatzemar
caràcters.
	// Si emmagatzemem altres coses al buffer, un byte que fós 0x0A
(salt de línia) (p. ex. un enter amb el número 10) tallaria el
paquet.

	uint8_t arrayEstat[MIDA_PAQUET_ESTAT] = "+++ESTAT";
		/* Se substituirà per les variables d'estat en el moment
precís de l'enviament */
	arrayEstat[MIDA_PAQUET_ESTAT - 1] = '\n';
			//Acabem el paquet amb un salt de línia
	enviarASCII(&arrayEstat,MIDA_PAQUET_ESTAT,15);
	return(0);
}

int enviarEstatUart() {
	/* ***** Envia l'estat directament per UART. No posa el paquet
en cua. ***** */



	// Definir MIDA_PAQUET_ESTAT al fitxer comunicacions.h

	uint8_t dadesPaquet[MIDA_PAQUET_ESTAT] = {'E'}; // 'E' caràcter
de control (estat)

	int int1 = arrodonirFloat(wRref);
	int int2 = arrodonirFloat(wLref);
	int int3 = arrodonirFloat(wLmeasured);
	int int4 = arrodonirFloat(wRmeasured);
	int int5 = arrodonirFloat(d[1]);
	int int6 = arrodonirFloat(kp);
	int int7 = arrodonirFloat(ki);


	// No tocar el byte 0. Conté un caràcter de control.
	memcpy(&dadesPaquet[1], &int1, sizeof(int));
	memcpy(&dadesPaquet[3], &int2, sizeof(int));
	memcpy(&dadesPaquet[5], &int3, sizeof(int));
	memcpy(&dadesPaquet[7], &int4, sizeof(int));
	memcpy(&dadesPaquet[9], &int5, sizeof(int));
	memcpy(&dadesPaquet[11], &int6, sizeof(int));
	memcpy(&dadesPaquet[13], &int7, sizeof(int));


	/* Aquí ja s'envia directament el paquet. NO es posa en cua.
(Ja estava en cua!) */
	enviaUart(&dadesPaquet, MIDA_PAQUET_ESTAT);

	return(0);
}
```

```
/* Retorna:
 * 0: Paquet tractat OK
 * 1: No hi ha dades per tractar
 * 2: Les dades al port UART no són un paquet de xarxa (són, p. ex.,
informació de l'estat de la connexió, errors, etc.)
 * 99: Totes les dades que no s'han de tractar (actualment inclòs el
punt 2)
 */
int tractarDadesRebudesUART() {

	/* Obtenim una nova dada del buffer UART (fins el següent salt
de línia) */

	uint8_t comanda[MIDA_MAXIMA_ENTRADA] = {0};
	int mida = 0;
	if (llegirBufferFIFO(&bufferEntradaUART, &comanda, &mida) !=
0){
		/* No hi ha noves comandes per llegir */
		return(1);
	}

	/* Parsing de les dades rebudes pel port UART.
	 * Actualment es diferència entre un paquet de dades (comença
per "+") i informació de l'estat del mòdul WIFI. */

	if (comanda[0] == '+') {
		// Síntaxi d'un paquet de dades:
		// "+IPD,a,b:xxxxx" on a = id connexió, b = número de
caràcters, xxxxx bytes del paquet.
		// Cal utilitzar les comes i els dos punts per saber a
partir de quin caràcter llegir. (a i b tenen llargaria variable)
		int i = 1;
		while (comanda[i] != ':') {
			i++;
			if (i == 20) {
				/* No són les dades que estàvem esperant. */
				return(99);
			}
		}

		i++;
		int longitud = 0;

		/* Paquet contindrà els bytes que s'han enviat per
TCP/UDP */
		uint8_t paquet[MIDA_MAXIMA_PAQUET] = {0};

		while( (longitud < MIDA_MAXIMA_PAQUET - 1) && (i <
MIDA_MAXIMA_ENTRADA)) { // -1 perque acabi en \0 (null)
			paquet[longitud] = comanda[i];
			longitud++;                                    /* TODO
canviar nom variable */
		    i++;
		}

		/* Funcio de parsing, desempaquetament i tractament de
les dades, al fitxer de funcions d'usuari */
```

```
			tractarPaquet(&paquet);
		}
		else {

			/* TODO Tractar dades connexió */
			return(99);
		}
		return(0);
}

/*
 *  Llegeix un paquet de xarxa, desempaqueta el seu contingut i
efectua les accions ne
 *  *paquet: Punter al primer element d'un array de bytes que
contingui el paquet.
 */
int tractarPaquet(uint8_t *paquet) {

		if(paquet[0] == 'E') {
			/* Paquet d'estat. Desempaquetar el contingut i assignar-
ho com a variable local. */


		}
		else if (paquet[0] == '>'){
			char strComparar[MIDA_MAXIMA_PAQUET];

			memcpy(strComparar,">LED6\r\n", 7);
			if( !memcmp(paquet, strComparar, 7)) {

				BSP_LED_Toggle(LED6);

				enviarASCII("OK",2,15);
				return(0);
			}

			memcpy(strComparar,">START\r\n", 8);
			if( !memcmp(paquet, strComparar, 8)) {

				start = 1;

				enviarASCII("OK",2,15);
				return(0);
			}

			memcpy(strComparar,">START\r\n", 8);
			if( !memcmp(paquet, strComparar, 8)) {

				start = 1;

				enviarASCII("OK",2,15);
				return(0);
			}

			memcpy(strComparar,">STOP\r\n", 7);
			if( !memcmp(paquet, strComparar, 7)) {
```

```
                    start = 0;

                    enviarASCII("OK",2,15);
                    return(0);
              }

              memcpy(strComparar,">KP=", 4);
              uint8_t primerCaracterNumeric = 4; //Començant de 0
              if( !memcmp(paquet, strComparar, 4)) {

                    /* Obtenir el valor numeric d'un string. Podeu
reaprofitar aquest codi, per això he creat la variable
primerCaracterNumeric */
                    /* Cal detectar el 0 per separat. Si no no podem
distingir errors de parsing */
                    if(paquet[primerCaracterNumeric] == '0') {
                          //kp = 0;
                          //enviarASCII("OK",2,15);
                    } else {
                          long xifra =
atof(paquet[primerCaracterNumeric]);
                          if (xifra != 0) {
                                kp = (float) xifra;
                                enviarASCII("OK",2,15);
                          } else {
                                enviarASCII("ERROR",2,15);
                          }
                    }

                    return(0);
              }

        }
        return(0);
}

int inicialitzaBufferFIFO(BufferFIFO *buffer) {
        //Inicialitza un struct BufferFIFO.
        buffer->longitud = MIDA_BUFFER;
        buffer->pEntrada = 0;
        buffer->pSortida = 0;

        for(int i = 0; i < MIDA_BUFFER; i++) {
              buffer->dades[i] = 0;
        }

        return(0);
}

void incrementaPunterEntrada(BufferFIFO *buffer) {
        buffer->pEntrada++;
        if (buffer->pEntrada == buffer->longitud) {
              buffer->pEntrada = 0;
        }

        /*
        if(buffer1.pEntrada - buffer1.pSortida >= 0) {
```

```
			debug = buffer1.pEntrada - buffer1.pSortida;
		}
		else {
			debug = buffer1.pEntrada + MIDA_BUFFER -
buffer1.pSortida;
		}
		if(debug == MIDA_BUFFER - 1) {
			HAL_Delay(0);
		}
		if(debug>debug_max) {
			debug_max = debug;

		}
		*/
}

void incrementaPunterSortida(BufferFIFO *buffer) {
		buffer->pSortida++;
		if (buffer->pSortida == buffer->longitud) {
			buffer->pSortida = 0;
		}
		/*
		if(buffer1.pEntrada - buffer1.pSortida >= 0) {
			debug = buffer1.pEntrada - buffer1.pSortida;
		}
		else {
			debug = buffer1.pEntrada + MIDA_BUFFER -
buffer1.pSortida;
		}
		if(debug == MIDA_BUFFER - 1) {
			HAL_Delay(0);
		}
		if(debug>debug_max) {
			debug_max = debug;

		}
		*/
}

/** Afageix bytes a la cua d'un objecte BufferFIFO
 *	Parametres:
 *		*buffer: Adreça de memòria de l'objecte BufferFIFO
 *		*entrada: Adreça de memòria de l'array de bytes que es
vol afegir
 *		mida: Número de bytes que es volen copiar (fins a 255)
 *	Retorna (int):
 *		0:  OK
 *		99: Overflow
 */
int escriuBufferFIFO(BufferFIFO *buffer, uint8_t *entrada, uint8_t
mida) {
		/* Primer escriu el valor, i despres incrementa el punter. */
		for (int i=0; i < mida; i++) {

			/* Aquest if només és per quan hi ha overflow. Si no,
mirar més avall */
```

```
            if( (buffer->pEntrada + 1 == buffer->pSortida) || (
(buffer->pEntrada + 1 == buffer->longitud) && (buffer->pSortida == 0)
) ) {
                        /* Overflow. Sobreescriurem la propera
comanda sencera, la qual es perd. */
                        /* Per tant, cal avancar el punter de lectura
fins caracter posterior al seguent \n. */

                        /* Actualment es descarten els elements més
antics. Per a impedir l'entrada de nous elements
                         * fins que s'hagin tractat els actuals,
descomentar el return(1): */
                        //return(1);

                        int pSaltLinia = buffer->pSortida;

                        while (buffer->dades[pSaltLinia] != '\n') {
                              pSaltLinia++;
                              if (pSaltLinia == buffer->longitud)
{pSaltLinia = 0;}   //Recórrer circularment
                              if (pSaltLinia == buffer->pEntrada) {
                                    /* Error. El text que estem
intentant escriure ocupa més que tot el buffer sencer.  */
                                    /* La comanda es tallarà i no es
podrà parsejar bé. Després el sistema es recupera. */
                                    inicialitzaBufferFIFO(buffer);
                                    return(99);
                              }
                              /* Si s'aconsegueix sortir del while,
pSaltLinia apunta al seguent '\n' */
                        }
                        /* Avancem el punter de lectura, descartant
la primera comanda que s'havia de processar */
                        buffer->pSortida = pSaltLinia + 1;
                        if (buffer->pSortida == buffer->longitud)
{buffer->pSortida = 0;}
                  }
            buffer->dades[buffer->pEntrada] = entrada[i];
            incrementaPunterEntrada(buffer);
      }
      return(0);
}



/**
 * Llegeix una "comanda" (linia) del buffer, es a dir, fins al proper
salt de linia (\n).
 * Incrementa el punter de lectura. Aquest no pot avancar mai el
punter d'escriptura.
 * Parametres: Punter al buffer i punter a un array de sortida de
mida MIDA_MAXIMA_ENTRADA, inicialitzat amb zeros.
 * Si hi ha comandes pendents per llegir, escriu la primera a l'array
comanda i retorna 0.
 * Si no hi ha noves comandes per llegir, retorna 1.
 *
 * Primer es llegeix el byte i despres s'incrementa el punter.
```

```
 */
int llegirBufferFIFO(BufferFIFO *buffer, uint8_t *comanda, int *mida)
{
	/*Cal recorrer l'array desde pSortida fins a pEntrada per a
buscar un salt de linia. Si no n'hi ha, llavors no hi ha
	noves comandes per llegir.*/
	if(buffer->pSortida == buffer->pEntrada) {
		/* No hi ha cap caràcter sense tractar */
		return(1);
	}


	int pSaltLinia = buffer->pSortida ;

	while (buffer->dades[pSaltLinia] != '\n') {
		pSaltLinia++;
		if (pSaltLinia == buffer->longitud) {pSaltLinia = 0;}
//Recorrer circularment
		if (pSaltLinia == buffer->pEntrada) {
			/* No hi ha cap comanda sencera que no hagi estat
tractada */
			return(1);
		}

	/* Si s'aconsegueix sortir del while, pSaltLinia apunta al
seguent '\n' */
	}

	/* Llegir fins al proper salt de linia */
	int i = 0;
	/* Per entrendre millor immagineu-vos que només posa
while(buffer->pSortida != pSaltLinia + 1)
	 * La part dreta de l'OR (||) és per quan \n es troba a la
última posició.  */
	while ( ( (pSaltLinia + 1 != buffer->longitud) && (buffer-
>pSortida != pSaltLinia + 1)) || ( (pSaltLinia + 1 == buffer-
>longitud) && (buffer->pSortida != 0))) {
		comanda[i] = buffer->dades[buffer->pSortida];
		i++;
		incrementaPunterSortida(buffer);
	}
	*mida = i;

	/* Es deixa el punter apuntant al seguent caracter despres del
\n */
	return(0);
}

/*
int empaquetaSortida() {
	//Actualment s'envien 10 bytes

	//Exemple per a enviar un Float complet
	union {		// Fent això, la mateixa posició de memòria
pot ser accedida com a un float o com un array de bytes.
		float valorFloat;
		uint8_t bytes[4];
```

```
	} magnitudFloat1;
	magnitudFloat1.valorFloat = 2.75;


	uint8_t dadesPaquet[10] = "\0\0\0\0\0\0\0\0\0\0";
	//enviar paquet

	return(0);
}
*/



int desempaquetaEntrada(uint8_t *entrada) {
	//Exemple amb tres bytes que representen tres enters.

	entradaA = entrada[0];
	entradaB = entrada[1];

	//...
	return(99);
}

int inicialitzarUart() {

		inicialitzaBufferFIFO(&bufferEntradaUART);
		inicialitzaBufferFIFO(&bufferSortidaUART);


	  UartHandle.Instance          = USART_PORT;

	  UartHandle.Init.BaudRate     = BAUDRATE;
	  UartHandle.Init.WordLength   = UART_WORDLENGTH_8B;
	  UartHandle.Init.StopBits     = UART_STOPBITS_1;
	  UartHandle.Init.Parity       = UART_PARITY_NONE;
	  UartHandle.Init.HwFlowCtl    = UART_HWCONTROL_NONE;
	  UartHandle.Init.Mode         = UART_MODE_TX_RX;
	  UartHandle.Init.OverSampling = UART_OVERSAMPLING_16;


	  inicialitzaBufferFIFO(&bufferEntradaUART);
	  inicialitzaBufferFIFO(&bufferSortidaUART);

	  return(HAL_UART_Init(&UartHandle));
}

int configurarConnexioWifi(){

	HAL_Delay(500);

	uint8_t comandaATE[] = "ATE0\r\n";
	//Desactivar eco
	enviaUart(&comandaATE, COUNTOF(comandaATE) -1);

	HAL_Delay(500);

	inicialitzaBufferFIFO(&bufferEntradaUART);
```

```
	uint8_t comandaCWMODE[] = "AT+CWMODE=1\r\n";
	//Mode multi connexio
	enviaUart(&comandaCWMODE, COUNTOF(comandaCWMODE) -1);

	HAL_Delay(500);   //TODO Delay

	uint8_t comandaCIPSTA[] =
"AT+CIPSTA=\"192.168.173.100\",\"192.168.173.1\",\"255.255.255.0\"\r\
n"; //TODO
	enviaUart(&comandaCIPSTA, COUNTOF(comandaCIPSTA) -1);

	HAL_Delay(500);    //TODO Delay

	uint8_t comandaCWJAP[] =
"AT+CWJAP=\"wireless\",\"xarxaesp8266\"\r\n"; //TODO
	enviaUart(&comandaCWJAP, COUNTOF(comandaCWJAP) -1);

	HAL_Delay(7000);    //TODO Delay

	return(0); //TODO temporal
}



int establirConnexio() {
	//ANTIC
	//Establir la connexio bloqueja durant uns segons el vehicle.
	//Cal activar les interrupcions per UART despres de fer-ho.

	HAL_Delay(1000);

	uint8_t comandaCipmux[] = "AT+CIPMUX=1\r\n";
	//TODO definir les comandes a part
	enviaUart(&comandaCipmux, COUNTOF(comandaCipmux) -1);

	HAL_Delay(1000);  //TODO Delay

	uint8_t comandaNetejarConnexioUDP[] = "AT+CIPCLOSE=0\r\n";
//TODO
	enviaUart(&comandaNetejarConnexioUDP,
COUNTOF(comandaNetejarConnexioUDP) -1);

	HAL_Delay(1000);

	uint8_t comandaConnexioUDP[] =
"AT+CIPSTART=0,\"UDP\",\"192.168.173.1\",6789,6789,2\r\n"; //TODO
	enviaUart(&comandaConnexioUDP, COUNTOF(comandaConnexioUDP) -1);

	HAL_Delay(1000);    //TODO Delay

	return(0); //TODO temporal
}


/**
  * Enviar dades per UART
```

```
  * Paràmetres: pDadesTx Punter a dades a transmetre, mida
  *
  */
int enviaUart(uint8_t *pDadesTx, int mida) {
      /* Cada byte s'enviarà per interrupció. Cal copiar les dades a
una variable global. */
      memcpy(&paquetSortida, pDadesTx, mida);
      return(HAL_UART_Transmit_IT(&UartHandle, &paquetSortida,
mida));
}

int rebreUartIT(uint8_t *pDadesRx, int mida) {
      //Nota: Retorna 0 (HAL_OK) o el codi d'error.
      return(HAL_UART_Receive_IT(&UartHandle, pDadesRx, mida));
}

int enviarBufferUART() {
      /* Enviar una comanda a l'esp8266. Hi ha d'haver una separació
de xx ms entre comandes successives*/

      uint8_t paquet[MIDA_MAXIMA_PAQUET] = {0};
      int mida;

      /* Obtenir la primera comanda de la cua. Si no retorna 0, no
n'hi ha. */
      if( llegirBufferFIFO(&bufferSortidaUART, &paquet, &mida) != 0)
{
            /* No hi ha noves comandes per llegir */
            return(1);
      }

      /* Enviem la comanda a l'esp8266 pel port UART */

      if(paquet[0] == '+'){
            enviarEstatUart();
      }
      else {
            enviaUart(&paquet, mida);
      }

      return(0);
};

/*
 * Posar en cua l'enviament d'un paquet de xarxa (contingut en
ASCII), per la connexio especificada.
 *
 * *pDadesTx: Punter a les dades a transmetre. (adreça de memòria de
l'element 0 de l'array)
 *                  Les dades es copiaran immediatament, no cal
conservar el valor en memòria fins que s'hagi enviat
 * Mida:      Numero de bytes que s'han d'enviar.
 * Connexio:  Identificador de la connexio (0 = ordinador).
 * */
int enviarASCII(uint8_t *pDadesTx, int mida, int connexio) {
```

```
	/* Posem en cua dues comandes consecutives: CIPSEND (per a
preparar l'esp8266 per a rebre dades),
	 * i les dades d'un paquet. */
	if(connexio == 15) {
		int midaComandaCipsend;
		if(mida<10) {
			midaComandaCipsend = 17;
		} else if(mida<100) {
			midaComandaCipsend = 18;
		} else {
			midaComandaCipsend = 19;
		}

		uint8_t comandaCipsend[midaComandaCipsend];
		sprintf(&comandaCipsend, "AT+CIPSEND=0,%d\r\n", mida);
		/* Posa en cua l'ordre CIPSEND i el paquet en sí
consecutivamenet */
		escriuBufferFIFO(&bufferSortidaUART, &comandaCipsend,
COUNTOF(comandaCipsend) -1 );
		escriuBufferFIFO(&bufferSortidaUART, pDadesTx, mida);
	}
	return(0);
}


// Els callbacks es troben al Main
/*
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *UartHandle)
{
  BSP_LED_Toggle(LED3);
}


void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle)
{
  BSP_LED_Toggle(LED4);

  escriuBufferFIFO(&buffer1, &caracterRx, 1);
  rebreUartIT(&caracterRx, 1);

}

*/
```

B.2 Fitxer comunicacions.h

```
#ifndef __COMUNICACIONS_H
#define __COMUNICACIONS_H

#include "stm32f4xx_hal.h"
#include "stm32f4_discovery.h"

#define MIDA_BUFFER                         1024   //Mida Buffer dades
rebudes en bytes
#define MIDA_MAXIMA_ENTRADA         128  //Mida maxima en bytes que
pot tenir una linia d'entrada pel port UART
#define MIDA_MAXIMA_PAQUET          120    //Mida maxima en bytes que
pot portar un paquet de xarxa (TCP/UDP)

#define USART_PORT            USART2
#define BAUDRATE        115200

#define PERIODE_TRACTAMENT_DADES_REBUDES  1
#define PERIODE_ENVIAMENT_DADES_UART            15          //
Milisegons entre l'enviament de dues comandes per UART
#define PERIODE_ENVIAMENT_ESTAT                 40          //
Milisegons entre l'enviament de paquets amb les variables d'estat

#define MIDA_PAQUET_ESTAT     20                                    //
Mida de les variables d'estat que es volen enviar. De 10 a 255 bytes.

#define UART_ENVIA_TIMEOUT 1000
#define COUNTOF(__BUFFER__)   (sizeof(__BUFFER__) /
sizeof(*(__BUFFER__)))


/* Parametres connexio */

#define IP                          192.168.173.1
#define IP_SERVIDOR           192.168.173.100
#define MASCARA_XARXA         255.0.0.0
#define SSID_WIFI             wireless
#define CONTRASENYA_WIFI      xarxaesp8266
#define PORT_TCP              6780
#define PORT_UDP              6789


/* Definicions */


#define arrodonirFloat(x) ((x)>=0?(int)((x*100)+0.5):(int)((x*100)-
0.5))      //Float a int amb punt fixe. De -327,68 a 327,67

typedef struct BufferFIFO {
      uint8_t dades[MIDA_BUFFER]; // El buffer emmagatzema bytes
      uint16_t longitud;                  // Longitud del buffer
      uint16_t pEntrada;                  // Punter escriptura
      uint16_t pSortida;                  // Punter lectura
} BufferFIFO;
```

```
//Pendent funcio que defineixi l'estat al començament, per estalviar
passes
typedef enum
{
  CONNECTAT             = 0x00U,
  NO_CONNECTAT          = 0x01U,
  ERROR_CONNEXIO  = 0x02U,
} EstatConnexioTypeDef;

typedef enum
{
  UDP_SERVIDOR          = 0x00U,
  TCP_SERVIDOR          = 0x01U,
  UDP_PROVES            = 0x0FU,
} Connexions;

/* Capceleres de funcions */
int temporitzadorComunicacionsTick();
int posarCuaEnviamentEstat();
int enviarEstat();
int tractarDadesRebudesUART();
int tractarPaquet(uint8_t *paquet);
int inicialitzaBufferFIFO(BufferFIFO *buffer);
void incrementaPunterEntrada(BufferFIFO *buffer);
void incrementaPunterSortida(BufferFIFO *buffer);
int escriuBufferFIFO(BufferFIFO *buffer, uint8_t *entrada, uint8_t
mida);
int llegirBufferFIFO(BufferFIFO *buffer, uint8_t *comanda, int
*mida);

int empaquetaSortida();
int desempaquetaEntrada();
int inicialitzarUart();
int configurarConnexioWifi();
int establirConnexio();
int enviaUart(uint8_t *pDadesTx, int mida);
int rebreUartIT(uint8_t *pDadesRx, int mida);
int enviarBufferUART();
int enviarASCII(uint8_t *pDadesTx, int mida, int connexio);









// Configuracio especifica dispositiu

/* Definition for USARTx clock resources */
#define USARTx                           USART2
```

```
#define USARTx_CLK_ENABLE()
__HAL_RCC_USART2_CLK_ENABLE();
#define USARTx_RX_GPIO_CLK_ENABLE()       __HAL_RCC_GPIOA_CLK_ENABLE()
#define USARTx_TX_GPIO_CLK_ENABLE()       __HAL_RCC_GPIOA_CLK_ENABLE()

#define USARTx_FORCE_RESET()
__HAL_RCC_USART2_FORCE_RESET()
#define USARTx_RELEASE_RESET()
__HAL_RCC_USART2_RELEASE_RESET()

/* Definition for USARTx Pins */
#define USARTx_TX_PIN                     GPIO_PIN_2
#define USARTx_TX_GPIO_PORT               GPIOA
#define USARTx_TX_AF                      GPIO_AF7_USART2
#define USARTx_RX_PIN                     GPIO_PIN_3
#define USARTx_RX_GPIO_PORT               GPIOA
#define USARTx_RX_AF                      GPIO_AF7_USART2

/* Definition for USARTx's NVIC */
#define USARTx_IRQn                       USART2_IRQn
#define USARTx_IRQHandler                 USART2_IRQHandler



/* variables globals */
//extern BufferFIFO buffer1;

#endif
```