Gestió de les comunicacions dins del entorn Joc de drones

Marc Guerrero Molero

Resum — En els últims anys gràcies a un avenç tècnic s'ha produït una gran expansió dels drones en l'oci. Avui en dia, hi ha molta varietat en la utilització de *drones*, des de prendre fotografies fins a participar en carreres d'obstacles. Joc de drones és un joc basat en la clàssica tipologia de captura de bandera, on hi ha dos o més equips intentant capturar bases o eliminant als *drones* contraris per a poder guanyar el joc. L'objectiu principal d'aquest projecte és el disseny, gestió i implementació de les comunicacions entre cada component del joc: bases, drones, controladors i àrbitre. També, s'implementa l'àrbitre, qui serà l'encarregat de comptar puntuacions i aplicar sancions, així com un servei web perquè l'espectador pugui consultar en qualsevol moment l'estat actual del joc.

Paraules clau — *Drones*, Joc de *drones*, IoT, Comunicacions IoT, Web IoT, Connexió WiFi IoT.

Abstract — For the last years, there has been a great expansion of drones at lesure because of a technical advance in tecnology. Nowadays, there is a lot of variety in drone uses, from taking a picture to participating in obstacle races. Game of Drones is a game based on the classic flag capture, where there are two or more teams trying to capture bases or eliminating the drones form other teams in order to win the game. The project main objective is the design, management and implementation of communication between each game component: base, drones, controllers and referee. Also, the referee, who will be in charge of counting scores and applaying sanctions, is designed and implemented, as well as a web service so that the viewer can check the game corrent status at any time.

Index Terms — Drones, Game of Drones, IoT, Comunications IoT, Web IoT, IoT WiFi connection.

1 Introducció

ENCARA que a principis del segle XX ja es van començar a fabricar els primers *drons* no tripulats, no ha sigut fins el segle XXI on han començat a tenir un paper més rellevant en la nostra societat. Actualment en l'àmbit de l'oci, aquests s'estan utilitzant per a fer curses entre ells posant a prova les habilitats i els reflexes dels pilots conjuntament amb la tecnologia dels fabricants.

En aquest projecte, forma part d'una nova alternativa en la seva utilització per l'oci. La proposta s'anomena Joc de *Drons* i consisteix en equips que han de conquerir les bases enemigues sense perdre les seves, i per tant, el que les obtingui serà el guanyador.

Els problemes que s'hauran de superar són, per una banda, el desconeixement en l'àmbit i per l'altra, la dificultat en dissenyar, implementar e integrar el sistema en el *dron*, les *bases*, els *controladors* y les *comunicacions*.
En aquest article s'exposarà el procés seguit per a dur a terme totes les comunicacions i l'administració del joc.

- *E-mail de contacte: marc.guerrerom@e-campus.uab.cat*
- *Menció realitzada: Enginyeria de Computadors i electrònica de comunicacions.*
- *Treball tutoritzat per: Màrius Montón (microelectrònica)*
- *Curs 2017/18*

2 Objectius

Seguidament, es plantegen els objectius generals ha assolir en el treball. Aquests són:

- Dissenyar i implementar les comunicacions sense fils entre els drons, la base i el comandament de control.
- Gestionar les normes i marcadors del joc.
- La latència de les comunicacions no poden excedir-se de 30 mil·lisegons.

Com a objectius opcionals, es plantegen els següents:

- Integrar la part de la gestió del joc situada en un ordinador en un sistema encastat.
- Implementar una pàgina web per consultar l'estat del joc.

3 Estat de l'art

En aquest apartat es fa una revisió a les altres possibles solucions per les comunicacions entre els diferents dispositius.

3.1 Constrained Application Protocol

Constrained Application Protocol (CoAP) és un protocol asíncron estandarditzat en 2014 similar a HTTP, dissenyat especialment per IoT, que et permet comunicar diferents dispositius que capacitats reduïdes de bateria, còmput, memòria, etc. [12]

CoAP va sorgir de la necessitat de fer un protocol diferent dels convencionals, ja que consumeixen bastants recursos, per a les aplicacions IoT. En aquest àmbit, els dispositius que formen una xarxa estan alimentats per bateries, i per tant, han de consumir poc per a poder aguantar una gran quantitat de temps funcionant.

Aquest protocol fa servir UDP per les comunicacions i està dissenyat per enviar missatges en xarxes bastant congestionades. A més a més, té una escalabilitat molt bona i és compatible amb altres protocols com MQTT, HTTP, WebSocket, etc. [11]

3.2 Representational State Transfer

Representational State Transfer (REST) és un protocol senzill de comunicació, basat en HTTP, que permet la transferència, modificació o eliminació de dades en la web a través d'uns mètodes ja definits.

REST és un protocol de client/servidor sense estat, és a dir, cada petició HTTP conté tota la informació necessària per executar-la, per tant, permet que ni el client ni el servidor necessitin recordar cap estat prèvi. Per altra banda, els objectes sempre es manipulan a partir de la URI, ja que aquesta facilita inequívocament l'accessibilitat a la informació.

La intrefície que proporciona REST és uniforme que sistematitza el procés amb la informació. Les operacions més importants relacionades amb les dades són quatre [13]:

- **GET**: Llegir o consultar dades.
- **POST**: Crear dades.
- **PUT**: Editar dades.
- **DELETE**: Eliminar dades.

3.3 Web Sockets

Web Socket és una connexió mitjançant TCP de baixa latència, en temps real i bidireccional. Aquesta és una bona solució per a actualitzar i sincronitzar dades a temps real i control i monitoratge de IoT [14].
L'aspecte dúplex de WebSocket permet a un servidor comunicar-se amb un client sempre que ho desitgi sense que hagi de ser el client específicament qui obri aquesta comunicació. Això fa que es prengui un enfocament diferent a l'establir una connexió persistent i bidireccional que permet al servidor actualitzar l'aplicació del client sense una sol·licitud d'aquest. A causa d'aquest fet, es produeix una millora de latència en les comunicacions, així com, una reducció del tràfic en la xarxa.

Web Socket utilitza un *handshake* compatible amb HTTP per establir una connexió entre el client i el servidor. Aquesta comença quan el client envia un HTTP GET amb el camp "actualitzar". Un cop que aquesta primera connexió és inicialitzada, la comunicació canvia a bidireccional.

4 METODOLOGIA

La metodologia que s'ha dut a terme en aquest projecte és iterativa i incremental. Per tant, s'han implementat petites parts, però utilitzables, fins a arribar a la conclusió del projecte.

4.1 Planificació del treball

Per a la planificació del treball, s'ha tingut en compte tres fases principals:

- **Plantejament i disseny del sistema global**: Aquesta etapa consisteix a dissenyar els blocs i tasques que es necessiten per a poder realitzar el projecte amb èxit. El temps previst per aquesta etapa ha estat de 2 mesos.
- **Implementació**: Aquesta etapa consisteix a implementar els blocs que s'han dissenyat i plantejat en l'etapa anterior. Tal com s'ha comentat anteriorment, per a dur a terme aquesta es farà servir una metodologia iterativa i incremental. El temps estimat per aquesta tasca ha estat de 4 mesos.
- **Integració i test**: En aquesta etapa s'ha de fer una integració del sistema complet i fer els testos adients. Per aquesta tasca s'ha previst un temps estimat de 2 mesos.

En la planificació es contempla al final un temps de 2 mesos per a la redacció de l'informe i possibles imprevistos, així com la introducció dels objectius opcionals. Per tant, la duració total del projecte és de 10 mesos. En l'apèndix 1 es pot trobar el diagrama de Gant amb totes les tasques planificades.

Per altra banda, s'utilitzaran eines com Trello per la gestió de tasques i repositoris com GitHub [20] pel codi. A més a més, es faran reunions setmanals per avaluar la resolució o el progrés de les tasques planificades.

5 COMPONENTS

Aquest projecte està pensat per a poder-se integrar en *drons* a control remot. Aquests són el model més estès en l'àmbit de l'oci, ja que et permeten controlar totalment el *drone* fent qualsevol tipus de maniobra desitjada.
En els següents apartats s'explicaran tots els components utilitzats per assolir els objectius del projecte.

5.1 Internet of Things

Internet of Things (IoT) és un concepte basat en la inter-

connexió de qualsevol dispositiu amb qualsevol altre del seu voltant. Aquest poden ser des de una nevera fins a un llibre. L'objectiu és fer que tots els dispositius es comuniquin entre ells i, per tant, siguin més independents i intel·ligents. Per l'IoT és necessari fer servir el protocol IPv4 o IPv6 [1] permetent la comunicació entre els dispositius i serveis a la xarxa, per exemple, amb serveis *cloud*.

5.2 Wi-Fi

Wi-Fi fa referència a una marca comercial impulsada per la Wi-Fi Alliance, on la seva finalitat era crear un mecanisme de connexió sense fils que fóra compatible per a diversos dispositius, d'aquí, va sorgir l'estàndard 802.11.

Aquest és un mecanisme que permet l'accés de diversos dispositius a una xarxa local o a Internet. La connexió sense fils és possible gràcies a la utilització de radiofreqüències e infrarojos per la transmissió de la informació[2].

Per altra banda, la connexió Wi-Fi es realitza mitjançant un punt d'accés, el qual ha d'estar situat a una distància menor aproximada de cent metres, on els dispositius es connecten i poden formar una xarxa. Les avantatges d'aquest són la simplicitat i flexibilitat de la instal·lació i la seva escalabilitat. Per contra, hi ha una pèrdua de velocitat respecte la connexió amb cables i és més dèbil en la seguretat de l'enviament de dades.

5.3 MQTT

Message Queue Telemetry Transport (MQTT) és un protocol ideat per IBM enfocat en la comunicació *Machine to Machine*. Aquest segueix una tipologia d'estrella on hi ha un node central, anomenat broker, que és l'encarregat de gestionar i transmetre els missatges a un gran nombre de clients. Els principis del disseny del protocol són[4]:

- Minimitzar els requisits d'amplada de banda
- Assegurar la confiança dels missatges
- Assegurar la seguretat d'entrega dels missatges

La comunicació es basa en el que s'anomena *topics* on els clients publiquen els missatges i els dispositius que volen rebre'ls se subscriuen. Cal aclarir que un dispositiu pot publicar en un *topic* i ser subscriptor en el mateix o en un altre. La figura 1 resumeix el funcionament de l'intercanvi de missatges mitjançant els *topics*.

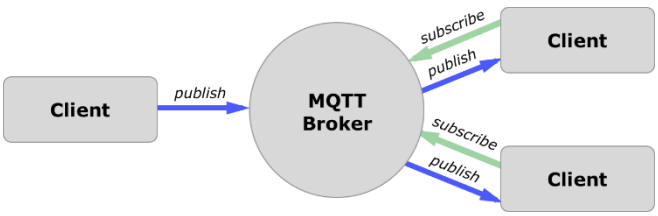


Figura 1: funcionament MQTT

Pel seu funcionament, es defineixen 5 operacions bàsiques:

- **Connect**: Estableix una connexió amb el *broker*.
- **Disconnect**: Finalitza la connexió esperant abans la finalització de qualsevol treball que estigui pendent.
- **Subscribe**: Subscriu al client a un *topic*.
- **Unsubscribe:** Elimina la subscripció del client al a un *topic*.
- **Publish:** Publica un missatge en un determinat *topic*.

Un exemple de la filosofia de MQTT podria ser el següent: es té 5 bombetes a la cuina de casa i 5 més a una habitació. Si la bombeta esta a la cuina estarà subscrita al *topic* casa/cuina, i si esta a la habitació, estrà subscrita al *topic* casa/habitació. En el cas de que es volgues encendre totes les bombetes de la casa, es podria enviar el missatge "ON" al *topic* casa i totes les bombetes s'encendrien perquè estan subscrites a aquest. Encanvi, si es volgues apagar les de l'habitació, s'hauría d'escriure el missatge "OFF" al *topic* casa/habitació, i per tant, només s'apagarien les bombetes que estan subscrites, és a dir, les de l'habitació.

5.3.1 Mosquitto

Mosquitto és un broker de missatges que implementa el protocol MQTT permetent als clients connectar-se a aquest servei per a poder publicar i rebre missatges en un o varis *topics*.

Aquest servidor està implementat en C, això permet que pugui ser executat en màquines amb capacitat molt reduïda.

A més a més, mitjançant la crida a mosquitto_sub o mosquitto_pub es pot subscriure o publicar missatges respectivament en els *topics* desitjats. Aquestes funcions són molt útils per a poder simular o alterar el sistema introduint petites accions a través d'una comanda.

5.4 Python

Python és un llenguatge de programació orientada a objectes de propòsit general. Aquest és un llenguatge interpretat, és a dir, no necessita compilació per a la seva execució, això afavoreix a la rapidesa en el seu desenvolupament però una menor velocitat d'execució[5].

Els grans avantatges d'aquest llenguatge són:

- La gran quantitat de llibreries disponibles
- La velocitat i senzillesa amb què es creen els programes
- Gran quantitat de plataformes on es pot desenvolupar
- És gratuït i de codi obert.

5.4.1 Paho-mqtt

El projecte Paho-mqtt proporciona unes llibreries per a treballar amb MQTT en diferents llenguatges, un d'ells, python.

Paho-mqtt és de codi obert proporcionant totes les utilitats i clients pensades per aplicacions Machine-to-Machine i de IoT. A través d'aquestes llibreries es pot connectar a un broker, com pot ser mosquitto, rebre i publicar missatges en els *topics* desitjats[6].

Per tant, això permet poder crear una aplicació que interactui amb els dispositius IoT que es tenen a la xarxa.

Per gestionar els missatges en els *topics* subscrits, es necessita cridar a la funció loop_start() perquè es gestionin els *buffers* d'entrada i de sortida. Hi ha tres modes diferents de crida a la funció *loop()*: [15]

- **loop()**: S'ha de fer la crida manualment de manera regular per a mantenir la connexió.
- **loop_start()**: Crea un nou thread que crida regularment a la funció *loop()* per a mantenir la connexió.
- **loop_forever()**: Manté la crida a *loop()* regularment bloquejant el programa per només gestionar els buffers.
- **loop_stop()**: Para la crida regular a *loop()*.

Quan un missatge arriba a un topic subscrit, es criden a unes funcions automàticament (callbacks) que han de ser assignades a altres funcions gestionades pel programador. Aquestes funcions són les següents:

- **on_connect():** Quan es rep la confirmació de la connexió acceptada del *broker* aquesta és la funció que es crida.
- **on_disconnect():** Quan s'envia el missatge de desconnexió aquesta és la funció que es crida.
- **on_publish():** Quan es publica un missatge en un *topic* desitjat, aquesta funció és cridada.
- **on_subscribe():** Quan el *broker* confirma la subscripció a un *topic*, aquesta és la funció que es crida.
- **on_unsubscribe():** Quan el *broker* confirma la baixa a un *topic* anteriorment subscrit, aquesta funció es crida.
- **on_message():** Aquesta funció és cridada quan un missatge ha estat publicat en un *topic* subscrit.

5.4.2 Flask

Flask és un *microframework* de Python de codi obert que permet portar a terme les tasques més comunes en el desenvolupament web.

Generalment, flask s'enfoca en el mínim necessari per a poder fer un prototipat de projectes relativament ràpid. D'altra banda, inclou un servidor web de desenvolupament per a poder provar les aplicacions sense instal·lar cap tipus de servidor, com podria ser Apache o Nginx.

A més a més, flask conté les següents característiques[7]:

- Unitat de test integrada
- Compatible amb aplicacions RESTful.
- Utilitza el motor de plantilles Jinja2[8].
- Suporta seguretat de *cookies*.
- Compatible amb WSGI 1.0
- Basat en *Unicode*[9] .

5.5 ESP8266

ESP8266 és un SOC (System on Chip) amb connexió WiFi i compatible amb el protocol TCP/IP entre d'altres. L'objectiu principal és donar accés a qualsevol microcontrolador a una xarxa.

La gran avantatja que té aquest mòdul és el seu baix consum, per tant, és un producte ideal per a *weareables* i dispositius IoT. Les seves principals característiques són: [16]

- Voltatge d'operació entre 3V i 3,6V.
- Corrent d'operació 80 mA.
- Temperatura d'operació entre -40ºC i 125 ºC.
- Suporta IPv4 i els protocols TCP/UDP/FTP.
- 17 ports GPIO però només 10 són utilitzables.
- Els ports GPIO poden ser configurats amb resistència Pull-up o Pull-down.
- Soporta els principals busos de comunicació:
  - SPI
  - I2C
  - UART
- Consumeix 0,5 µA en repòs i 170 mA quan es transmet al màxim de senyal.
- Treballa en tres modes:
  - *Active*: al màxim rendiment.
  - *Sleep*: només el Real Time Clock (RTC) està actiu per a mantenir la sincronització. Es queda en mode alerta per detectar els esdeveniments que calgui canviar de mode. Manté a la memòria les dades de connexió, així doncs, no cal tornar a establir connexió. En aquest mode, el consum varia entre 0,6mA i 1 mA
  - *Deep sleep*: el RTC està encès però no operatiu. Per a despertar, ha de passar pel mode *sleep*. Les dades que no s'hagin guardat amb anterioritat a aquest mode es perdran. El consum oscil·la entre els 20 µA.

5.5.1 NodeMCU

Aquest component està compost per un ESP12E, el qual és una variant del ESP8266, afegit d'una memòria *flash*, així com una entrada USB per on es pot alimentar i programar. Les característiques més rellevants d'aquest són les següents:

- Voltatge d'entrada: 5V
- Voltatge de sortida pels pins: 3,3V

- Voltatge de referència de l'ADC: 3,3V
- Corrent nominal per pin: 12mA
- Freqüència del processador: des de 80MHz fins 160MHz
- Memòria flash de 4MB

El NodeMCU és el node que s'ha utilitzat en els casos dels *drones* i bases dins del protocol MQTT, per tant, permet la connexió a internet dels altres dispositius.

Aquest mòdul és un dels més característics, ja que té inclòs tot el necessari per a poder treballar de forma autònoma.

El NodeMCU es pot programar, mitjançant la descàrrega d'un firmware, en diferents llenguatges com LUA, Python, Basic i JavaScript. La versió firmware per defecte suporta comandes AT. A més a més de poder programar amb una sintaxi molt similar a la d'Arduino, també es pot programar a través la IDE d'aquest.

El *pinout* d'aquest component es pot veure en l'apèndix 2.

5.6 Arduino WiFi shield

Arduino WiFi shield és el node que s'ha utilitzat en el cas de controlador. Funcionalment, aquest component és molt semblant al NodeMCU, amb la diferència que en els pins del port UART es fa una conversió de 3,3V a 5V. A més a més, en ser una shield especial per Adruino, el hardware s'integra de millor manera, evitant així possibles errors en les connexions.

5.7 Rapberry Pi 3

Raspberry Pi és un microcomputador de baix consum i preu. Aquesta pot executar diferents tipus de sistemes operatius, però generalment estan basats en Linux. El sistema operatiu que s'ha utilitzat ha sigut Raspbian. Aquest està basat en Debian però optimitzat pel hardware de Raspberry Pi.

A més a més de les funcions generals d'un computador, Raspberry incorpora funcions electròniques com GPIO i comunicacions UART, SPI i I²C. Això comporta que es puguin connectar sensors i actuadors.

El model utilitzat ha estat el 3. Aquest model té com a diferència principal, a més a més de l'augment de rendiment, la capacitat de connectar-se a una xarxa sense fils WiFi. Les característiques més rellevants d'aquest component són les següents: [19]

- **Processador**:
  - Chipset Broadcom BCM2387
  - 1,2 GHz de quatre nuclis ARM Còrtex-A53
- **GPU**
  - Dual Core VideoCore IV Multimedia Co-procesador. Proporciona Open GL ES 2.0, Open VG accelerat per hardware, i 180p30 H.264 d'alt perfil de decodificació
- **RAM**: 1GB LPDDR2
- **Connectivitat**:
  - Ethernet Socket Ethernet 10/100 BaseT
  - 802.11 LAN sense fils y Bluetooth 4.1

La funció que compleix dins del sistema és la d'àrbitre i servidor web. Aquestes dues funcions es porten a terme mitjançant un programa en Python, que l'intèrpret ja està preeinstal·lat en el sistema operatiu.

6 LÒGICA DE LA COMUNICACIÓ

En aquest apartat es fa referència a com s'ha dissenyat l'esquema de comunicacions, incloent-hi el protocol.

Primerament, es defineix cadascun dels elements del joc com un node dins de l'esquema MQTT. Aquests nodes estan subscrits i publiquen en els *topics* corresponents depenent de la seva funció.

La informació a enviar ve donada per la connexió dels nodes als corresponents components mitjançant un port sèrie. Un cop rebuda la informació, aquesta s'envia per Wi-Fi al *broker* MQTT.

Quan un node publica informació en un *topic*, el subscriptor destinatari l'envia mitjançant el port sèrie al component connectat.

Quan l'àrbitre veu a través dels missatges publicats en els *topics* una acció que requereix aplicar la lògica del joc, l'aplica i envia per un altre *topic* l'acció a fer. Per tant, es pot considerar que l'àrbitre és un altre node independent publicant i enviant missatges MQTT.

6.1 Protocol de comunicació MQTT

En aquest apartat es comenten les situacions que es poden donar durant el joc, i en conseqüència els missatges enviats entre els diferents dispositius amb el protocol MQTT.

6.1.1 Inscripció

Aquest cas és el primer que és dona, ja que per a mantenir un control dels participants del joc, només iniciar-se tots els dispositius han de inscriure's, perquè l'àrbitre els tingui en compte, enviant un missatge al *topic WantToPlay*. El format dels missatges ve determinats per l'element que representa dins del joc, és a dir, si es base, controlador o drone. Aquests es mostren a continuació:

- **Base:**
  - <NomDeLaBase,Equip>
- **Drone:**
  - <NomDelDrone,Equip,Controlador>
- **Controlador:**
  - <NomDelControlador,Equip,Drone>

El diagrama de seqüència d'aquesta acció es pot veure a continuació:

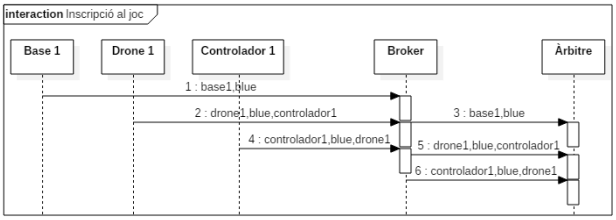


Figura 2: Diagrama de seqüència d'Inscripció al joc

6.1.1 Tret

Aquest cas és dona quan un *drone* ha sigut disparat per un altre. El missatge que envia el *drone* disparat és: <FIRE,DroneTocat,DroneDisparador,arma,costat> on:

- **FIRE**: és el *topic* on s'ha de publicar per aquesta acció.
- **DronTocat**: és el *drone* que ha estat tocat i l'encarregat d'enviar el missatge.
- **DroneDisparador**: és el *drone* que ha disparat.
- **Arma**: és l'arma utilitzada pel disparador.
- **Costat**: és el costat en el qual ha impactat el tret.

Seguidament, l'àrbitre informa, publicant en el *topic* "*movement*", al controlador del dron tocat que ha estat ferit perquè apliqui la sanció corresponent. El missatge que envia és: <MOVEMENT,ControladorDroneTocat, dreta, esquerra, davant, endarrere, arma> on:

- **MOVEMENT:** és el *topic* on s'ha de publicar per aquesta acció.
- **ControladorDroneTocat:** és el controlador del *drone* que ha estat tocat.
- **Dreta:** Si té sanció al costat dret.
- **Esquerra:** Si té sanció al costat esquerre.
- **Davant:** Si té sanció a davant.
- **Endarrere:** Si té sanció enredere.
- **Arma:** Arma utilitzada pel tret.

Quan el *drone* ha complert amb la sanció, per defecte està fixada en 10 segons, es torna a enviar un missatge com l'anterior actualitzant els camps de sanció identificat per la paraula *penalty* en el camp d'arma.

Per últim, l'arbitre informa al *drone* de les vides que té per actualitzar els seus marcadors visuals. El missatge és: <LED, drone, dreta, esquerra, davant, endarrere> on:

- **LED:** és el *topic* on s'ha de publicar per aquesta acció.
- **Drone:** és el *drone* que ha estat tocat.
- **Dreta:** valor del led dret.
- **Esquerra:** valor del led esquerre.
- **Davant:** valor del led frontal.
- **Endarrere:** valor del led posterior.

El diagrama de seqüència d'aquesta acció es pot veure a continuació:

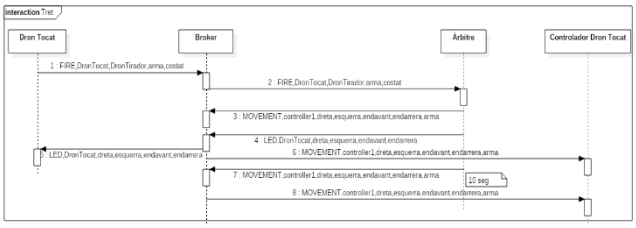


Figura 3: Diagrama de seqüència de Tret

6.1.2 Ordre de disparar

Aquest cas és dona quan el pilot pressiona el botó de disparar en el comandament. El missatge que envia és: <CFIRE,controlador,arma> on:

- **CFIRE:** és el *topic* on s'ha de publicar per aquesta acció.
- **Controlador:** és el controlador del *drone* que envia l'ordre.
- **Arma:** és l'arma seleccionada per a disparar.

Aquest missatge el rep el *drone* pertanyent al comandament perquè executi l'acció. El diagrama de seqüència d'aquesta acció es pot veure a continuació:

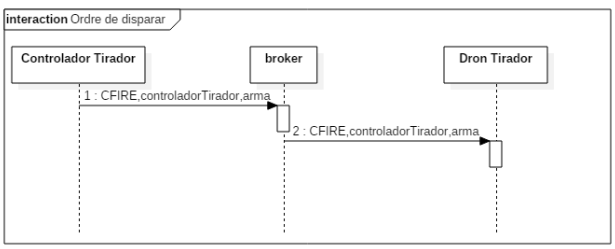


Figura 4: Diagrama de seqüència d'Ordre de disparar

6.1.3 Captura d'una base

Aquest cas és dona quan una base ha sigut conquerida per un drone, i per tant, passa a formar part del mateix equip. Perquè l'àrbitre confirmi que la base ha estat capturada, s'ha d'esperar que la base enviï dos missatges de captura, un amb la identificació "irda" i l'altre amb "rfid". El missatge que envia és: <CATCH,BaseConquerida,DroneConqueridor,Id> on:

- **CATCH:** és el *topic* on s'ha de publicar per aquesta acció.
- **BaseConquerida:** és la base conquerida.
- **DroneConqueridor:** és el *drone* que ha conquerit la base.
- **Id:** és la identificació del mètode que ha fet servir per a conquerir-la.

Un cop rebut els dos missatges, l'àrbitre pren la decisió de si ha estat o no vàlida la conquesta, si és valida, envia el següent missatge a la base perquè canviï de color: <CBA-

SE,BaseConquerida,Color> on:

- **CBASE:** és el *topic* on s'ha de publicar per aquesta acció.
- **BaseConquerida:** és la base conquerida.
- **Color:** és el nou color que ha de mostrar la base.

El diagrama de seqüència d'aquesta acció es pot veure a continuació:

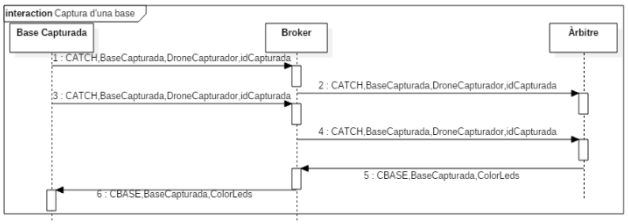


Figura 5: Diagrama de seqüència de captura d'una base

6.1.4 Mort

Aquest cas és dona quan un *drone* ha perdut totes les seves vides, i per tant, està mort. El missatge que s'envia al *drone* i al seu controlador és el següent: <DEAD,ControladorDroneEliminat, DroneEliminat> on:

- **DEAD:** és el *topic* on s'ha de publicar per aquesta acció.
- **ControladorDroneEliminat:** és el controlador del drone eliminat.
- **DroneEliminat:** és el *drone* que ha estat eliminat.

El diagrama de seqüència d'aquesta acció es pot veure a continuació:

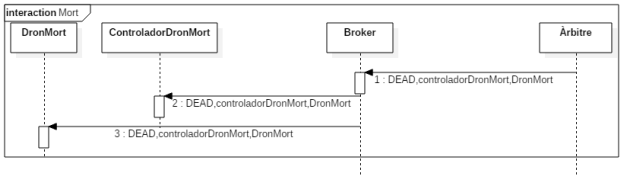


Figura 6: Diagrama de seqüència de mort

6.1.5 Finalització del joc

Aquest cas és dona quan un dels equips ha guanyat. Quan hi ha aquesta situació, s'envia un missatge a tots els components per a finalitzar el joc. Aquest és el següent: <GOVER> on:

- **GOVER:** és el *topic* on es publica per a la finalització del joc.

El diagrama de seqüència d'aquesta acció es pot veure a continuació:

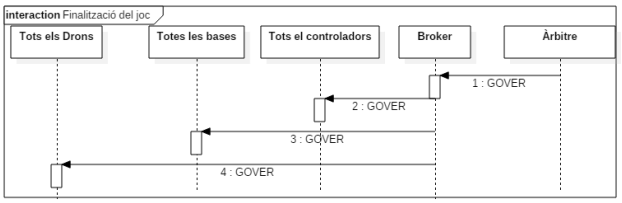


Figura 7: Diagrama de seqüència de finalització del joc

6.2 Protocol de comunicació sèrie

En aquest apartat es comenta el protocol de comunicació pel port sèrie. Aquest està definit per a comunicar el dispositiu que participa en el joc, és a dir, la base, el *drone* o el comandament amb el NodeMCU-ESP8266 o la *Shield* d'Arduinio.

Per simplicitat, la definició s'ha fet quasi idèntica a la del protocol MQTT. La diferència entre aquests és que el dispositiu que participa no guarda en cap moment el seu identificador, per tant, els camps on s'indica o necessita saber-ho són afegits pel codi funcionant a la *Shield* d'Arduinio o el NodeMCU-ESP8266.

Per veure les diferències entre els protocols, seguidament es mostren dos exemples en forma de diagrama de seqüència: la comunicació de tret i l'ordre de disparar.

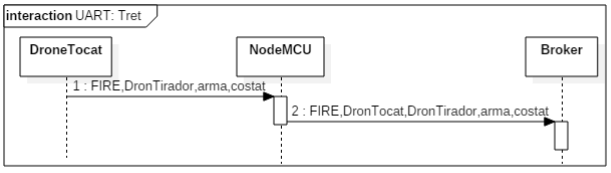


Figura 8: Diagrama de seqüència de comunicació sèrie per acció Tret

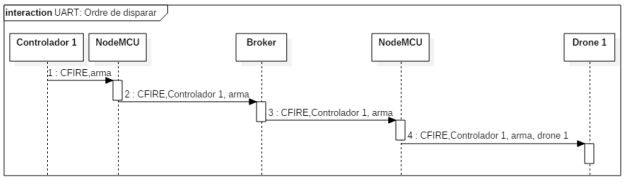


Figura 9: Diagrama de seqüència de comunicació sèrie per acció Ordre de disparar

En el cas de tret, el node MQTT ha d'afegir el camp de DronTocat, ja que el dispositiu no guarda qui és.

En el cas de l'ordre de disparar, el node MQTT ha d'afegir a l'orde qui és perquè ho pugui enviar al dron que dispara.

7 IMPLEMENTACIÓ

En aquest apartat es comenten la lògica d'implementació pels tres mòduls del projecte: Node MQTT, àrbitre i interfície web.

7.1 Node MQTT

Com ja s'ha comentat en apartats anteriors, aquest mòdul es programa en el NodeMCU. Aquest és l'encarregat de l'interfície de comunicació del dispositiu amb la resta del joc, per tant, ha de transmetre pel port sèrie tot el que es publiqui en els *topics* que estigui dirigit cap a ell i tot el que el dispositiu vulgui publicar en aquests.

A més a més, és l'encarregat de subscriure's al joc. Aquesta acció és bloquejant i esta implementada perquè s'executi només encendres.

A continuació, es pot veure un diagrama de flux de la lògica del programa.

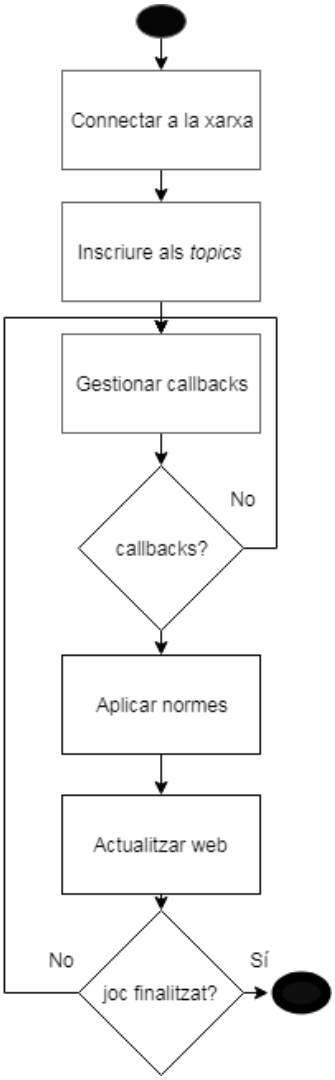


Figura 10: Diagrama de flux del programa en NodeMCU

7.2 Àrbitre

L'àrbitre és l'encarregat d'administrar el joc, és a dir, tenir el recompte dels punts, aplicar penalitzacions, determinar si una acció és vàlida, etc.

Per donar flexibilitat al joc, l'àrbitre guarda diccionaris amb tots els participants que s'han inscrit. En cada clau del diccionari, que ve donada pel nom del participant, està guardat un objecte de la classe drone, base o controlador on cada objecte té els camps essencials per a poder saber l'estat d'aquest dins del joc.

Aquest mòdul està implementat en la Raspberry pi en un programa escrit en Python. A més a més, l'àrbitre és l'encarregat de proporcionar totes les dades a la interfície web mitjançant sockets.

A continuació, es pot veure el diagrama de flux de la lògica del programa.

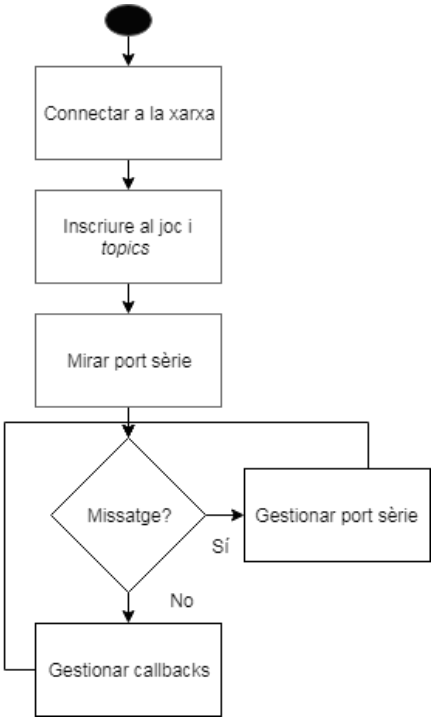


Figura 11: Diagrama de flux del àrbitre

7.3 Interfície web

Per a facilitar el seguiment del joc a l'usuari, s'ha creat aquest mòdul que està implementat en la Raspberry pi mitjançant Flask.

La web rep les dades mitjançant un socket que està sempre obert al port 9999. Aquesta informació s'emmagatzema de la mateixa forma que en l'àrbitre, és a dir, en un diccionari amb referència a objectes.

Com l'execució de l'aplicació web és bloquejant, s'ha utilitzat *multiprocessing* amb comunicació per memòria compartida entre el procés que executa l'aplicació i el procés que executa el socket.

Per a utilitzar memòria compartida entre processos en Python, cal indicar en la creació del nou procés quina part de codi executarà i quines variables són les que es vol compartir. Això es fa mitjançant els paràmetres en la crida a la funció. Un exemple sería el següent:
Process(target=refreshData(),args=(redDronesAlive, blueDronesAlive))

El mètode anterior s'utilitza per a variables de tipus *ctype*. Si es vol compartir un diccionari, com és el cas de la web, s'ha de cridar a l'objecte *manager* de la llibreria multiprocessing i crear un diccionari dins d'aquest. D'aquesta manera, el diccionari queda guardat en un *Proxy* local i es pot accedir des de qualsevol dels dos processos.

Per altra banda, la interfície web té els apartats següents:

- **General Follow-up**: En aquest apartat es pot veure la informació més general del joc, com per exemple, el número de drons que juguen, els que estan vius, quantes bases tenen cada equip, etc.
- **Red team**: En aquest apartat es veuen les estadístiques de l'equip vermell, així com els seus integrants. Mitjançant un desplegable, es pot veure tota la informació de cada jugador, com per exemple en el cas dels *drones*, les bases capturades, a quina hora l'ha capturat, el registre de baixes que ha produït, etc. Per part de les bases, es pot veure quin *drone* ha sigut l'últim en capturarla, el registre de captures, a qui pertany, etc.
- **Blue team**: En aquest apartat es veuen les estadístiques de l'equip blau. Els camps d'informació són exactament els mateixos que en l'apartat anterior.
- **MVP**: En aquest apartat es mostren els jugadors més destacats en els camps següents:
    - El que més ha disparat encertadament.
    - El que més bases ha conquerit.
    - La base més conquerida.

A continuació, es pot veure el diagrama de flux de la lògica del programa.

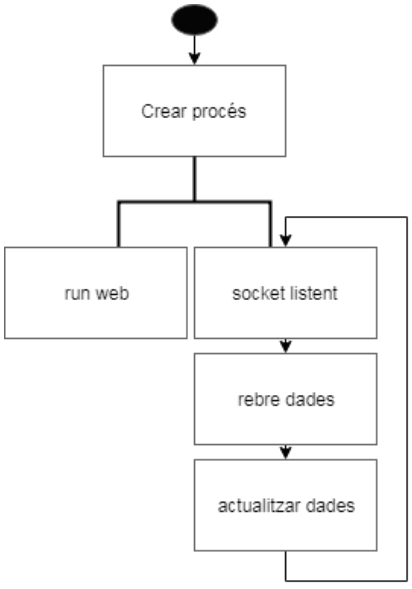


Figura 12: Diagrama de flux de la web

8 TEST

Tal com s'ha comentat en l'apartat de metodologia, s'han fet test funcionals i d'integració. En aquest apartat es comenten els tests més significatius.

8.1 Test funcionals

Per a validar que totes les funcionalitats, s'han realitzat test de caixa negre, és a dir, s'han injectat totes les entrades possibles observant la resposta del sistema. Una simplificació d'aquestes proves es poden veure en els diagrames següents:

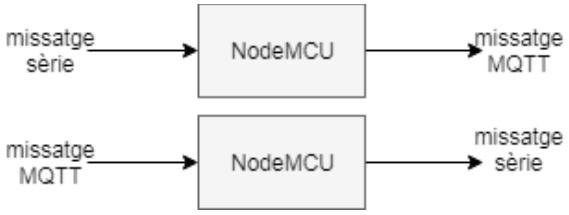


Figura 13: Diagrama de blocs del test funcional

A més a més, per a poder validar els mòduls d'àrbitre i servei web, s'ha utilitzat les comandes de Mosquitto per a publicar en els *topics* necessaris. Les simulacions s'han automatitzat mitjançant scripts bash per a agilitzar totes les proves necessàries per a la seva validació.

8.2 Test de latència

Tal com es mostren en l'apartat d'objectius, un requisit és la latència. Aquesta no pot excedir-se de 30 ms, i per tant, s'han realitzat test per comprovar que aquest objectiu es compleix.

8.2.1 Test de latència de comunicacions

La primera prova s'ha fet per validar el sistema de comunicació, és a dir, des que es rep un missatge d'un disposi-

tiu, fins que aquest és entregat al destinatari. El procediment per portar-lo a terme és enviar un missatge pel port sèrie, i en el moment que aquest és rebut, es configura un pin del NodeMCU en mode sortida. Quan aquest missatge és rebut i enviat per la UART en l'altre extrem del esquema MQTT, és a dir, en el que està subscrit al *topic*, aquest configura un altre pin en mode sortida.

El diagrama de seqüència de la comunicació es pot veure a continuació:

D'aquesta manera, connectant un oscil·loscopi o un analitzador lògic, es pot mesurar el temps que hi ha entre la configuració d'un pin i l'altre.

Per a obtenir una millor aproximació en el càlcul, la prova s'ha repetit 7 vegades. Els resultats es poden veure en la següent taula:

Intent	Resultat en ms
1	0.8 ms
2	0.8 ms
3	2,28 ms
4	2,20 ms
5	1,56 ms
6	2,64 ms
7	0,74 ms

Com a resultat final del test s'ha calculat la mitjana aritmètica, per tant, el temps és de 1,57 ms.

Les imatges de l'oscil·loscopi es poden veure en l'apèndix 3.

8.2.1 Test de latència del sistema

En aquest test el que es busca validar és la latència del sistema i si influeix en l'experiència del jugador. Així doncs, es mesura des que el jugador polsa el botó de disparar fins que finalment el *drone* dispara. El temps total entre una acció i l'altra, no pot ser més alt de 30 ms.

Per a realitzar aquest càlcul, s'utilitza el mètode anterior on es configura un pin en mode sortida quan el jugador polsa el botó i un altre quan el *drone* realitza l'acció. El diagrama de seqüència d'aquest test es pot veure en l'apartat 6.1.2.

Finalment el temps total obtingut en aquest test és de 5,49 ms.

9 RESULTATS

Finalment, tal com es comenta en l'apartat de test, en els mòduls s'han realitzat test per a comprovar que la seva funcionalitat i latència. Per tant, totes les funcionalitats han estat comprovades i les latències calculades, obtenint la validació esperada.

10 CONCLUSIONS

Els objectius d'aquest projecte són:

- Dissenyar i implementar les comunicacions sense fils entre els drons, la base i el comandament de control.
- Gestionar les normes i marcadors del joc.
- La latència de les comunicacions no poden excedir-se de 30 mil·lisegons.

Per tant, tal com s'ha demostrat en els tests funcionals i de latència, així com en els d'integració, s'ha arribat a complir tots els objectius marcats.

En el temps marcat de latència per les comunicacions, s'ha obtingut un resultat de 5,49 ms, mentre que l'objectiu era 30 ms. Per tant, el temps obtingut ha estat de 24,51 ms menys. Tanmateix, s'ha de remarcar que per falta de recursos, no s'ha pogut provar aquest amb una gran quantitat de dispositius, fet que hagués incrementat el temps de latència.

A més a més, també s'havien marcat objectius opcionals. Aquest són:

- Integrar la part de la gestió del joc situada en un ordinador en un sistema encastat.
- Implementar una pàgina web per consultar l'estat del joc.

Aquests objectius s'han realitzat dins del temps del projecte, això és degut a l'aprenentatge durant el projecte amb la conseqüència d'implementar els mòduls i resoldre els errors sorgits de forma més rapida a mesura que el projecte avançava. Per tant, s'ha integrat el sistema de gestió, és a dir, l'àrbitre dins d'una Raspberry Pi, així com la realització d'un servei web dins del mateix hardware.

Com a millores futures, s'hauria d'implementar més seguretat per a inscriure's als *topics*, fent així que només els participants reals del joc puguin fer-ho i evitar intrusions que poden modificar el transcurs del joc. MQTT es pot configurar per a tenir un usuari i una clau d'accés, per tant, podria ser una possible solució.

AGRAÏMENTS

Primerament, m'agradaria agrair al meu tutor Màrius Montón pel temps, recursos i saviesa que ha aportat a aquest projecte.

També, agrair a en David de l'empresa Airk drones per donar-me suport, idees i ajuda en aquest projecte.

A més a més, als meus companys i amics de Joc de Drones, l'Ander, Iván i Omar, per haver fet que aquesta experiència compartida hagi sigut tan satisfactòria.

BIBLIOGRAFIA

[1] Hipertextual.
<https://hipertextual.com/2015/06/internet-of-things>
[20 Juny 2015]

[2] ValorTop
<http://www.valortop.com/blog/que-es-wifi-que-significa-y-para-que-sirve>
[5 Desembre 2017]

[3] International Organitzation for Standardization. Message Queuing Telemetry Transport (MQTT) v3.1.1. Technical Report 20922, ISO, 2016.
[Octubre 2017]

[4] Pàgina oficial del protocol MQTT.
http://www.mqtt.org
[Febrer 2018]

[5] Pàgina oficial Python.
http://www.python.org
[Març 2018]

[6] Pàgina oficial del projecte Paho-mqtt.
http://www.pypi.org/project/paho-mqtt
[Març 2018]

[7] Pàgina oficial de Flask.
http://www.flask.pocoo.org
[Abril 2018]

[8] Pàgina oficial de Jinja.
http://www.jinja.pocoo.org
[Abril 2018]

[9] Pàgina oficial de Python unicode.
http://www. docs.python.org/2/howto/unicode.html
[Abril 2018]

[10] Documentació del programa StarUML.
http://www.docs.staruml.io
[Maig 2018]

[11] Internet of things agenda.
http://www.internetofthingsagenda.techtarget.com/feature/Constrained-Application-Protocol-CoAP-is-IoTs-modern-protocol
[Agost 2017]

[12] Castro, Jorge, 2014. Uso del protocolo CoAP para la implementación de una aplicación domótica con redes de sensores inalámbricas (Capítol 3).
http://www.repositorio.upct.es/bitstream/handle/10317/4163/pfc5908.pdf;sequence=1
[Agost 2017]

[13] REST API Tutorial
http://www.restfulapi.net/http-status-codes
[Agost 2017]

[14] NetBurner
http://www.netburner.com/learn/websockets-for-real-time-web-and-iot-applications
[Agost 2017]

[15] API paho-mqtt
http://www. pypi.org/project/paho-mqtt
[Octubre 2017]

[16] Document Tècnic de ESP8266
http://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf
[Octubre 2017]

[17] Documentació de hardware de ESP8266
http://www.espressif.com/sites/default/files/documentation/esp8266-hardware_matching_guide_en.pdf
[Octubre 2017]

[18] Documentació Arduino WiFi shield
http://www. store.arduino.cc/arduino-wifi-shield
[Maig 2018]

[19] Pàgina oficial de Raspberry pi
http://www. raspberrypi.org/products/raspberry-pi-3-model-b
[Octubre 2017]

[20] Repositòri de codi
http://www.github.com/dronsuab/Communications

APÈNDIX

A1. DIAGRAMA DE GANT DE LA PLANIFICACIÓ

El diagrama de Gant de la planificació es troba a continuació:

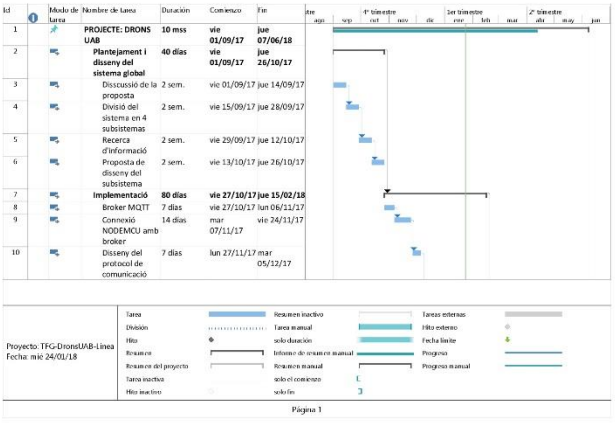


Diagrama de Gant 1: Planificació del projecte (part 1)

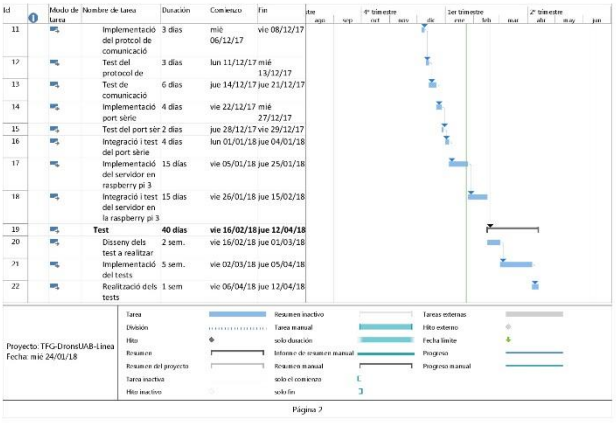


Diagrama de Gant 2: Planificació del projecte (part 2)

A2. NODEMCU: PINOUT

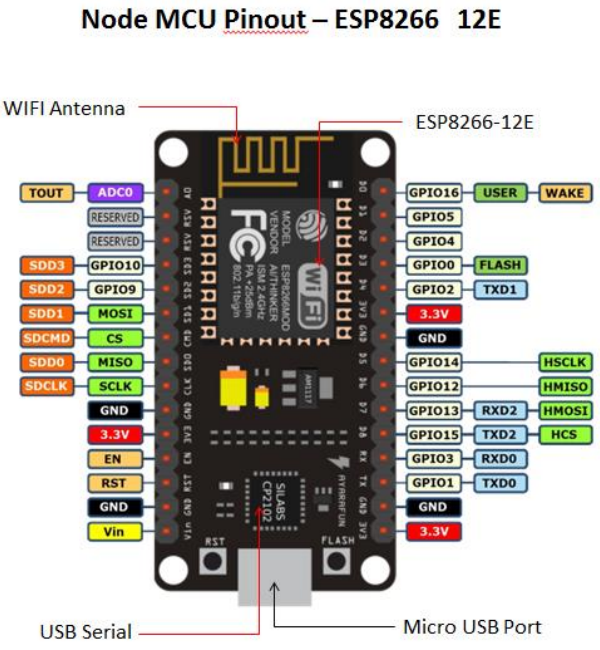
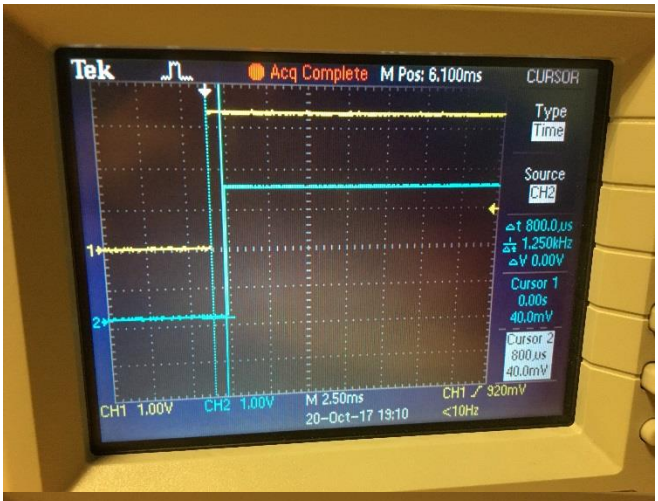
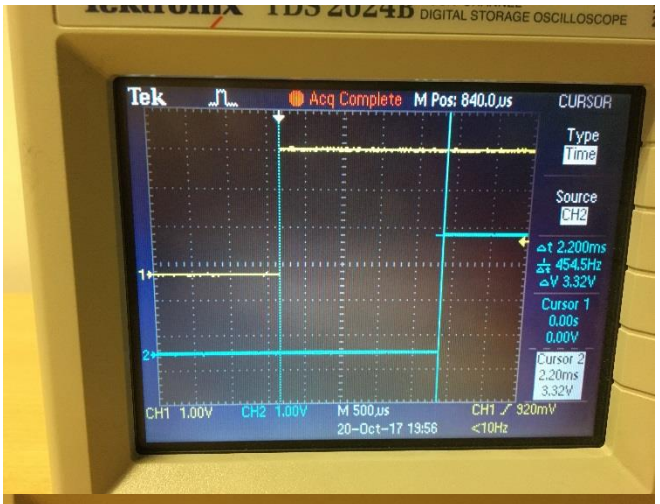
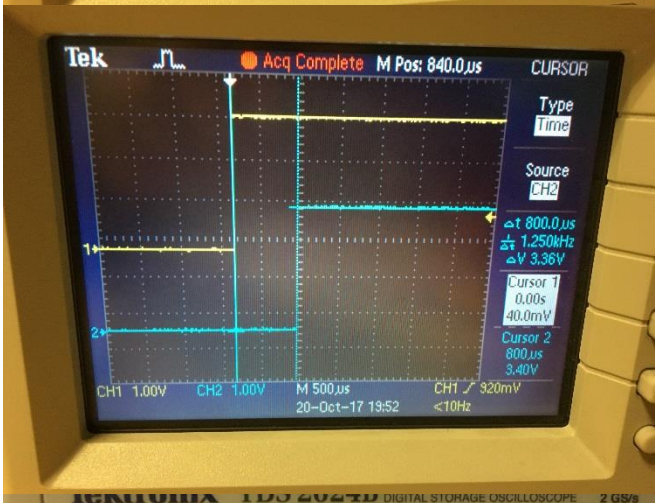


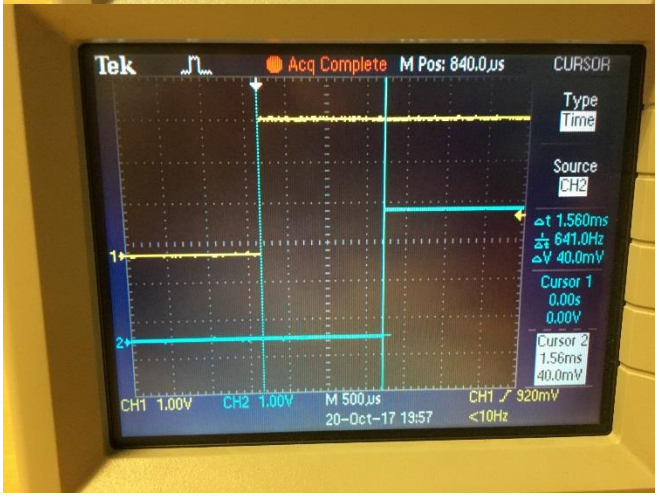
Figura 14: NodeMCU pinout

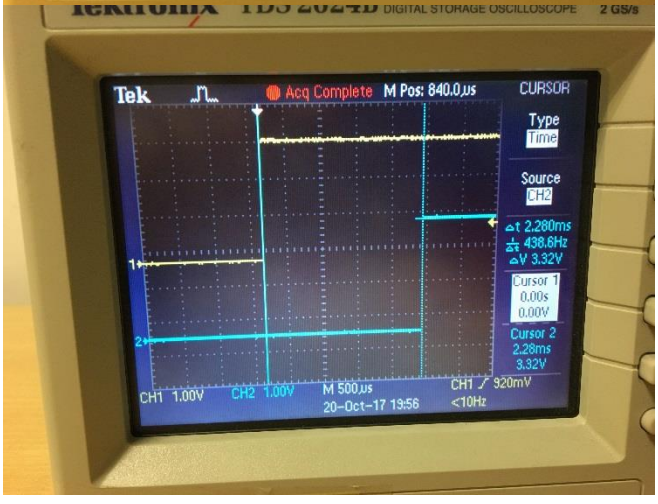
A3. IMATGES OSCIL·LOSCOPI PEL TEST DE LÀTENCIA

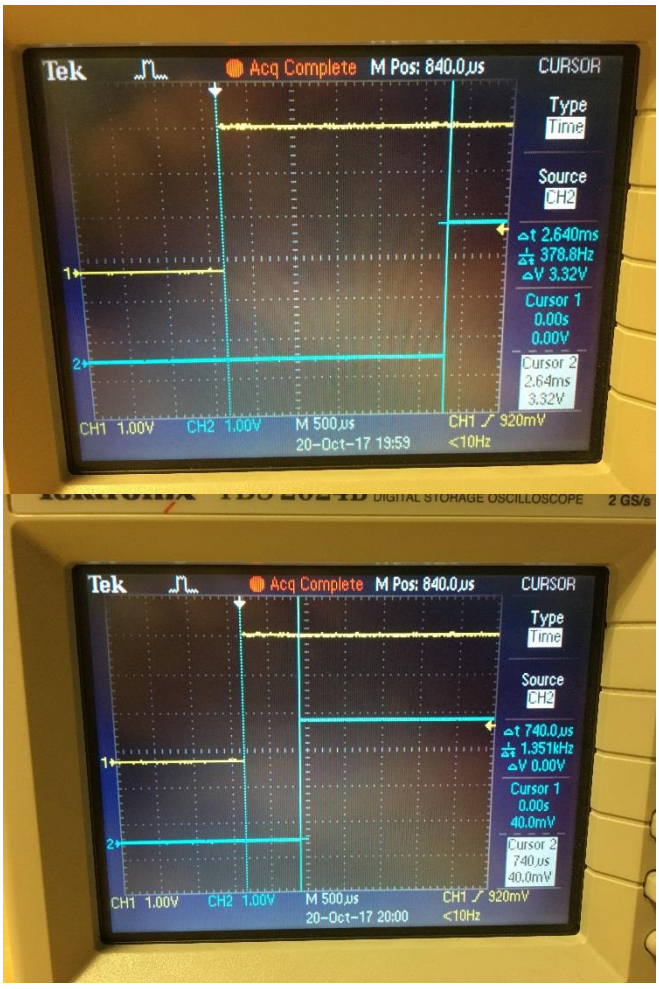











Tek
Acq Complete M Pos: 840.0µs
CURSOR
Type
Time
Source
CH2
Δt 2.640ms
1/Δt 378.8Hz
ΔV 3.32V
Cursor 1
0.00s
0.00V
Cursor 2
2.64ms
3.32V
CH1 1.00V
CH2 1.00V
M 500µs
CH1 ⁄ 920mV
20-Oct-17 19:59
<10Hz
DIGITAL STORAGE OSCILLOSCOPE
2 GS/s
Tek
Acq Complete M Pos: 840.0µs
CURSOR
Type
Time
Source
CH2
Δt 740.0µs
1/Δt 1.351kHz
ΔV 0.00V
Cursor 1
0.00s
40.0mV
Cursor 2
740µs
40.0mV
CH1 1.00V
CH2 1.00V
M 500µs
CH1 ⁄ 920mV
20-Oct-17 20:00
<10Hz