Estudi i implementació de l'incorporació de micropython a un sistema de control de microsatèl·lits

Ferran Campos Llopart

Resum– Amb l'objectiu d'innovar en el sector aeroespacial, el treball de final de grau pretén, per primer cop dins del sector, introduir l'ús de codi Python per al control de microsatèl·lits. Aquest tipus de satèl·lits utilitzen microcontroladors amb característiques i recursos molt reduïts, amb l'objectiu de reduir al màxim el seu cost i mida. Les limitacions que presenten suposen desenvolupar Software encastat, de baix nivell, per a poder esprémer al màxim les seves capacitats, deixant de banda altres llenguatges més pesats, com els interpretats. La intenció del projecte és trencar aquesta regla, i aconseguir que el software controlador de satèl·lits de l'Institut de Ciències de l'Espai, permeti als seus usuaris utilitzar Python per a la creació de seqüències de control dels seus microsatèl·lits.

Paraules clau– Microsatèl·lit, Python, Microcontrolador, Interpretats, Encastat

Abstract– With the goal of innovating in the aerospace sector, this final project aims to introduce the usage of Python code in order to control microsatellites. These types of satellites use microcontrollers, with scarce resources, with the purpose of reducing its volume and cost. Its limitations mean that embedded software must be developed to fully exploit its potential, leaving behind other programming languages, as interpreted ones. The aim of this project is to break this rule and achieve that the Space Science Institute's software for controlling microsatellites, allow its users to use Python for creating microsatellite's control sequences.
Keywords– Microsatellite, Python, Microcontroller, Interpreted, Embedded

1 Introduction

L'any 1957, l'URSS sorprèn el món gràcies al llançament del *Spútnik 1*[1], fita que marca l'inici de l'exploració espacial en ser el primer satèl·lit artificial de la història. Aquell aparell no era més que una esfera d'alumini de la mida d'una pilota de futbol, amb un pes de 86 kg, que tenia l'objectiu de brindar als científics soviètics d'informació valuosa sobre la densitat de l'atmosfera, els efectes de l'ús d'ones de ràdio en aquesta o els mètodes soviètics pel seguiment de la posició del satèl·lit.

Des d'aquell dia, la investigació sobre dels satèl·lits artificials ha anat a l'alça i, conjuntament amb l'evolució de les tecnologies i el disseny de coets més potents, ha propiciat que aquests disposin de components més potents que els permeten executar tasques d'una dificultat major, sense importar que el seu volum i massa augmentessin.

Encara així, els satèl·lits artificials tenen la característica de tenir un cost elevat, a causa de la quantitat i qualitat d'aquests components, a més del cost d'investigació i de llançament, que prové del coet en el qual es llança. Així doncs, els pressupostos per aquests projectes han de ser elevats, reduint considerablement el rang d'investigadors capaços de finançar-ho, sent normalment les agències espacials estatals, les que financen escasses missions d'aquesta índole.

Per a solucionar-ho, van aparèixer els microsatèl·lits, els quals redueixen el seu cost de llançament i construcció en utilitzar microcontroladors i petits components de prestacions més senzilles. En contrapartida, aquests no poden exercir missions complicades, per les seves limitacions, i per tant, són utilitzats en projectes amb objectius senzills i concrets.

L'estàndard més famós dels microsatèl·lits són els *CubeSats*. Aquestes especificacions van ser presentades per primer cop l'any 1999 pel català Jordi-Puig Suari, de la Universitat Politècnica de l'Estat de Califòrnia, i en Bob Twiggs, de la Universitat de Stanford[2], amb l'objectiu permetre als seus alumnes una plataforma per a dissenyar, construir, provar i operar un satèl·lit de prestacions similars a la del *Sputnik 1*.

• E-mail de contacte: ferran.campos@e-campus.uab.cat
• Menció realitzada: Enginyeria del Software
• Treball tutoritzat per: Lluís Gesa Bote (Departament de Ciences de la Computació)
• Curs 2019/20

1.1 Components i especificacions dels CubeSat

Les especificacions presentades es van esdevenir un estàndard amb el temps, gràcies a la necessitat d'una alternativa als satèl·lits convencionals. Aquestes dicten que cada CubeSat pot estar compost d'una o més unitats, les quals han de complir unes mètriques de 10x10x11.35 cm i tenir una massa que no excedirà els 1.33kg.

A més, cada satèl·lit, independentment del seu tipus, disposa els següents parts principals de *hardware*:

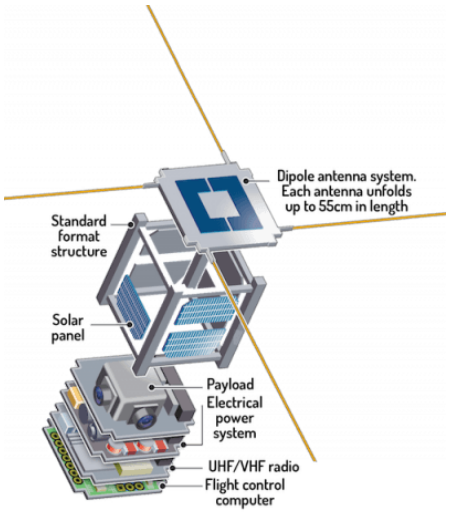

Fig. 1: Diferents parts d'un microsatèl·lit[3]

- **Ordinador d'abord**: Anomenat normalment normalment OBC(On Board Computer), o Flight Control Computer com es veu a la imatge. Aquest compleix la funció de controlar les diferents parts del satèl·lit i proveirà a l'usuari d'un ventall de funcionalitats que permetran dur a terme la seva missió o executar rutines de comprovació de l'estat dels diferents components del *CubeSat*.

- **Sistema de Telecomunicacions**: Compost per les diferents antenes, dota al satèl·lit de la capacitat d'enviar i rebre missatges des de terra.

- **Sistema d'Energia**: Es tracta dels diferents panells solar i bateries que abasten els diferents components del *CubeSat* d'electricitat per al seu funcionament.

- **Sistema de Navegació**: Permet localitzar el satèl·lit dins la seva òrbita.

- **Càrrega Útil**: Es tracta d'aquella maquinària necessària, normalment, per a la realització de l'experiment o de la missió.

1.2 Estat de l'art

Pel que fa al *software* dels satèl·lits, s'encarrega del seu control, i per tant, conté rutines per la utilització i comprovació de l'estat dels components de *hardware* esmentats anteriorment, com poden ser rutines per l'enviament i recepció de missatges.

Per altra banda, normalment utilitzen un sistema operatiu, que els hi permeti l'ús de *threading*[4]. Aquests sistemes operatius depenen dels components del *hardware*, ja que el seu conjunt ocupa un volum elevat de memòria. Així doncs, si es tracta de microsatèl·lits, s'utilitzen els *sistemes operatius per a sistemes encastats*, com per exemple *FreeRTOS*[5]. Aquests sistemes operatius són *Real-Time*[6], i per tant, simplement atorguen la funcionalitat de *threading*. En canvi, en sistemes amb més recursos, poden utilitzar-se sistemes operatius amb més funcionalitats, com pot ser *Linux*.

A causa de les prestacions i al baix cost dels microsatèl·lits, hi ha hagut un increment notori dels seus llançaments en l'última dècada i s'estima que aquest nombre vagi a l'alça, havent un total de 449 llançaments previstos per a aquest any 2020.

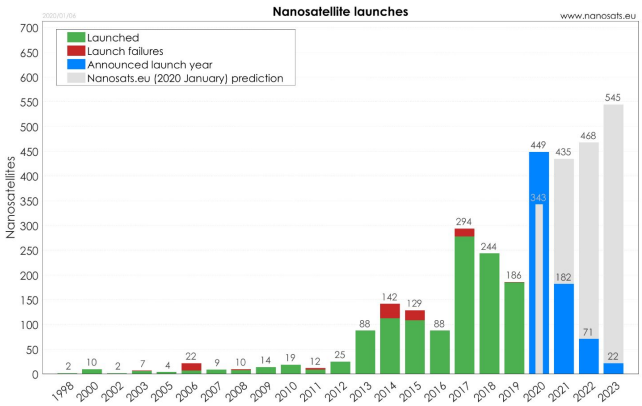

Fig. 2: Nombre de llançaments de microsatèl·lits a cada any[7]

Entre els llançaments podem trobar missions dutes a terme per corporacions i per universitats, com la missió **Lume-1**[8], produïda per l'*Universitat de Vigo*, la qual va llançar el seu propi *CubeSat* l'any 2018 i pretén utilitzar-ho per a ajudar en la lluita contra els incendis forestals.

Les agencies espacials també s'interessen per impulsar aquest mercat. Per exemple, l'*Agència Espacial Europea* realitza el concurs anual *Fly Your Satellite*[9], que consisteix en brindar ajuda professional i financera a aquells grups d'estudiants que hagin presentat les propostes més atractives. Un dels projectes finalistes del passat any 2019 s'anomena **UCAnFly**[10], duta a terme per l'*Universitat de Càdis* amb l'objectiu de mesurar camps magnètics. En ell han ajudat l'*Institut de Ciències de l'Espai*, entitat en la que s'ha realitzat aquest treball, proporcionant el seu projecte **C3SatP**, com a base per al desenvolupament del seu software de control.

1.3 El projecte C3SatP

Catalan Cube Satellite Platform, normalment anomenat **C3SatP**, és el projecte realitzat per l'*Institut de Ciències de l'Espai* juntament amb l'*Institut d'Estudis Espacials de Catalunya*, el qual té l'objectiu de crear Hardware i Software des del qual, qualsevol projecte que involucri la creació d'un *CubeSat*, pugui utilitzar-lo com a punt de partida per a la construcció del seu projecte. D'aquesta forma, els clients eviten el disseny i el desenvolupament de les parts bàsiques

del CubeSat, podent-se centrar en simplement en les funcionalitats específiques de la seva missió.

Un dels objectius del projecte *C3SatP* és permetre la personalització de certs aspectes del *Flight Control Software* mitjançant la incorporació de llenguatge de programació *Python*. Aquesta inclusió als microsatèl·lits solament havia sigut estudiada[13] i realitzada en *OBC*s potents, i per tant amb més recursos[14]. Així doncs, la seva incorporació dins de microsatèl·lits d'amb menor capacitat computacional i menors recursos seria una innovació dins del sector aeroespacial.

1.3.1 Hardware

El desenvolupament del hardware del projecte consisteix en el muntatge d'aquest mitjançant productes del tipus *Commercial Off The Shelf*[11], o *COTS*, els quals poden adaptarse al comprador, abaratint així els costos, ja que un producte dissenyat i realitzat exclusivament pel client té un preu més elevat. Ja s'han comentat les diverses parts d'un microsatèl·lit al punt **1.1 Components i especificacions dels CubeSats**, encara així descriurem l'*On Board Computer* utilitzat dins del projecte *C3SatP*, el qual determina els recursos amb els quals treballarem en el desenvolupament del software. Aquest component es tracta d'una *STM32-F446RE*[12], un microcontrolador amb un processador *ARM Cortex®-M4* que opera a 180 MHz, una *memòria flash* de 512KB i 128KB de *memòria RAM*.

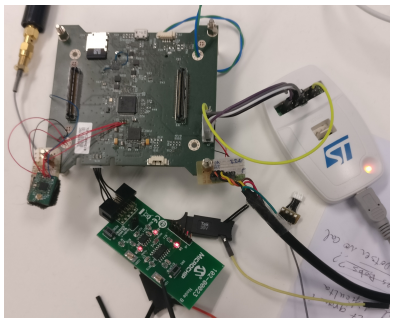

Fig. 3: Imatge de l'*OBC* del *C3SatP*

Encara així, s'utilitza el sistema que en proporciona el kit *NUCLEO-F446ZE*[15], per a les proves de software en tenir les mateixes especificacions i disposar de més pins d'entrada i sortida que faciliten la connexió d'un major nombre de components.

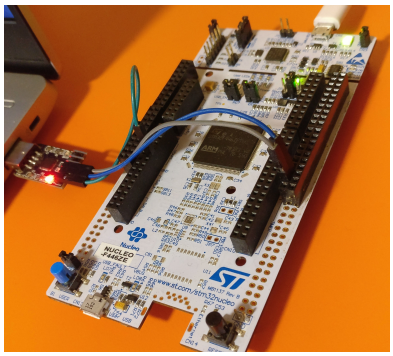

Fig. 4: Imatge de la *STM32-F446ZE* utilitzada com a entorn de proves del software

1.3.2 Software

El software del projecte el podem dividir en quatre components:

- **Sistema Operatiu**: Disposa d'un sistema operatiu especial per a sistemes encastats, anomenat *FreeRTOS*. Aquest va ser escollit per tractar-se d'un sistema *Real-Time* molt lleuger, que permet l'ús de certes funcionalitats específiques d'aquests com és el *threading*.
- **Firmware**: Es tracta de tot aquell codi, proveït pel fabricant del *Hardware*, en aquest cas *STM32*, que facilita el seu control.
- **Basis**: Llibreria creada per l'*Institut d'Estudis Espacials de Catalunya*, o IEEC, que conté totes aquelles rutines bàsiques per a tot projecte, com per exemple, les rutines d'inicialització del codi o transmissió de missatges.
- **Aplicació**: Aquesta és la part que el client haurà de desenvolupar per l'acompliment de la seva missió. Així doncs, podrà utilitzar totes les funcionalitats proveïdes per la resta de components, per a la creació d'un sistema de control per al satèl·lit.

Tot aquest software desenvolupat es tracta de **codi encastat**[16], el qual permet als desenvolupadors expremer al màxim les capacitats dels components del hardware programant el seu funcionament de forma optima per a que el seu rendiment sigui el màxim possible.

1.3.3 Llenguatges Compilats i Interpretats

El software encastat és desenvolupat utilitzant llenguatges de programació que faciliten aquesta programació dels components, i per això, els llenguatges compilats són els escollits. Aquests es caracteritzen per tenir un pas anomenat **compilació**, en el qual, l'ordinador comprova que no conté cap error i procedeix a transformar-lo a *codi màquina* per al hardware especificat. Així doncs, quan el *OBC* és programat, simplement executa aquest codi, el qual està optimitzat per al seu ús, fent més ràpida la seva execució. En aquest cas, el projecte C3SatP és programat utilitzant un d'aquests, anomenat *llenguatge C*.

Els llenguatges interpretats són l'alternativa als compilats, i es caracteritzen per no compilar el codi, utilitzant un **intèrpret** per a transformar a codi màquina a mesura que es va executant. Això vol dir que els errors no són trobats fins que s'intenten executar, a més de tenir un temps d'execució més alts als dels compilats, a causa d'interpretar el codi durant la seva execució. Els avantatges dels llenguatges interpretats consisteixen a facilitar la programació als programadors, en tenir una sintaxi més senzilla, i habilitar l'execució del codi en hardware amb característiques desconegudes, ja que l'intèrpret de cada màquina s'encarrega de transformar a codi màquina. Un exemple, en serien els llenguatges Python o Javascript.

En resum, els llenguatges compilats són utilitzats per al desenvolupament de codi que sigui òptim per al hardware en el qual s'executarà, com és requerit en el món dels microsatèl·lits. En canvi, els llenguatges interpretats, són usats en múltiples entorns, facilitant l'escriptura als seus programadors, per exemple, en la programació de pàgines web.

1.4 Micropython

Com podem veure, la programació mitjançant llenguatges interpretats no té cabuda en l'àmbit dels *CubeSat*, encara així, el projecte Micropython intenta donar-los un lloc dins dels circuits integrats.

Micropython és un projecte de llicència pública, que implementa un motor que permet l'execució de codi Python en sistemes encastats, mitjançant el llenguatge C. Així doncs, es tracta d'un projecte escrit mitjançant un llenguatge compilat, el qual té diferents versions per habilitar el seu ús en diferents tipus de hardware. Aquestes versions són normalment referides com a *ports*, i d'entre ells, destaca la versió *Minimal port*, la qual es caracteritza per contenir simplement les funcionalitats mínimes per al funcionament de Micropython dins de sistemes Linux i Nucleo.

Les funcionalitats que otorga aquesta versió són:

- **Consola de Python:** Crear una consola en la qual podrem escriure comandes Python que s'executaran en temps real i rebrem el seu resultat.
- **Execució d'arxius Python:** L'execució d'arxius dependrà de l'entorn utilitzat, ja que de tenir un sistema d'arxius podrà executar els arxius Python existents en ell, en cas que l'entorn no disposi d'un, com podrien ser alguns sistemes de *STM32*, existeix l'opció d'indicar quin arxiu Python volem incloure a la compilació, afegint aquest fitxer al Micropython i permetent la seva posterior execució.

Encara que aquesta versió abasti el mínim de Python, ens permet decidir sobre quines funcionalitats volem que tingui la nostra versió, afegint-les a la compilació de Micropython. Així doncs, mitjançant un fitxer de configuració podem decidir si Python podrà treballar, o no, amb nombres decimals, o fins i tot activar o desactivar qualsevol de les dues funcionalitats bàsiques de Micropython exposades a la llista anterior.

1.5 Objectius

La programació dels *sistemes encastats* requereix una corba d'aprenentatge important. En canvi, els llenguatges interpretats no ho són, a més cal remarcar que molts investigadors tenen certs coneixements d'alguns d'aquests llenguatges, sent el més famós *Python*. L'objectiu principal del treball és el d'investigar i incloure Python dins del projecte C3SatP, amb la finalitat que els usuaris tinguin la possibilitat de programar, enviar i executar rutines o seqüències de control per al satèl·lit d'una forma més fàcil. A més, en tractar-se d'un llenguatge interpretat, permet que els usuaris personalitzin aquestes seqüències de control, fet impossible amb l'ús exclusiu de codi encastat.

Encara que aquest fos l'objectiu principal, el treball es va dividir en diversos objectius, els quals han de ser assolits per a completar el treball:

- **TFG-OBJ-1 - Adaptació a un projecte existent**: Inclusió de *Micropython* dins del projecte *C3SatP*, permetent l'ús de les funcionalitats dels dos a la vegada.
- **TFG-OBJ-2 - Execució dinàmica d'arxius Python precompilats**: Implementació d'una nova funcionalitat per a Micropython, en el qual permeti l'execució d'arxius precompilats (d'extensió *.mpy*) de forma dinàmica, és a dir, que sigui capaç d' emmagatzemar, executar o eliminar aquest fitxer.
- **TFG-OBJ-3 - Creació de mòduls controladors**: L'objectiu tracta de crear diversos mòduls amb els quals, un usuari futur, pugui usar funcionalitats del CubeSat (implementades al software del projecte C3SatP) mitjançant codi Python. Així doncs, ens referim a crear un seguit de llibreries Python amb les quals l'usuari seria capaç de, per exemple, agafar dades d'un component o d'enviar bits d'informació mitjançant l'antena de comunicació.
- **TFP-OBJ-4 - Enginyeria del Software**: Aquest objectiu pretén enfocar el treball dins del marc de l'Enginyeria del Software, utilitzant les metodologies i tècniques apreses al llarg de la carrera. Així doncs, encara que s'explica més detalladament en el següent punt, s'utilitzarà una metodologia que implicarà una fase d'anàlisis, disseny, desenvolupament i proves per a totes les tasques del treball amb l'objectiu d'aconseguir els objectius i validar els resultats obtinguts.
- **TFG-OBJ-OPC-1 - Inclusió de mòduls de forma dinàmica**: El projecte Micropython, encara que permet la inclusió de nous mòduls creats per l'usuari, són gravats al disc dur, fent-los llavors invariables i provocant que sigui impossible incloure'n un de nou de forma dinàmica. Així doncs, aquest objectiu consistiria a permetre a l'usuari incloure nous mòduls dinàmicament, tal com pretenem fer amb els arxius d'execució. Aquest objectiu és opcional, degut al temps i a la dificultat d'aquest.

1.6 Metodologia

La metodologia escollida va ser el **desenvolupament incremental**, el qual divideix les funcionalitats que es volen implementar en un seguit d'iteracions (o increments) en les quals, utilitzant l'última versió del software desenvolupat, es pretén 'incrementar-lo' afegint les funcionalitats establertes a ser implementades en aquell increment[17].

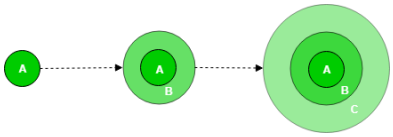

Fig. 5: Concepte del mètode incremental[17]

Pel que fa a aquest treball, es comença amb una fase de captura de requeriments, on s'estableixen i documenten tots els seus requeriments, consultant al client. Aquests requeriments dicten les restriccions del treball, les quals s'hauran de complir per a donar-lo com a finalitzat. Amb la consolidació d'aquest document, es determinen un conjunt de fases, de les quals el treball en va tenir un total de 4, les quals van ser:

- **TFG-FA-01**: Definició dels objectius, tasques, metodologia i definició del document de requeriments

- **TFG-FA-02**: Desenvolupament de l'objectiu *TFG-OBJ-01* i de l'objectiu *TFG-OBJ-02*
- **TFG-FA-03**: Desenvolupament de l'objectiu *TFG-OBJ-03*
- **TFG-FA-04**: Desenvolupament dels documents finals del treball i fase de correcció d'errors

D'aquestes fases, solament les **TFG-FA-02, TFG-FA-03 i TFG-FA-04** són fases d'implementació de codi. Aquestes fases són homologues als *increments*, i cadascuna d'elles té els següents passos per a completar-les:

- **Disseny:** Es dissenya, mitjançant diagrames *UML*, el funcionament i les estructures del codi a desenvolupar durant l'increment, amb la intenció d'assolir l'objectiu de la fase.
- **Codi:** S'escriu el codi d'acord amb els diagrames fets al pas anterior. En cas de trobar que el disseny és erroni, i per tant no es pot implementar per qualsevol raó, es tornarà a la fase de Disseny per a començar de nou el seu disseny.
- **Proves:** Es dissenyen els diferents casos de test i s'implementen (en cas de ser tests automàtics), o s'escriuen els passos a realitzar dins d'una taula de pas-resultat (en cas de ser tests manuals). Després es prova el codi, si es troba un error, es tornarà fins a la fase de Codi o Disseny (depenent de l'error trobat), en cas contrari, es finalitzarà l'increment per a procedir al següent.

Podeu consultar les tasques i la planificació, d'una forma més detallada, **a l'apartat C de l'apèndix.**

El treball manté un control de versions, creant una de nova cada vegada que s'estimi necessari, i sempre que es completi una de les passes anteriorment descrites.

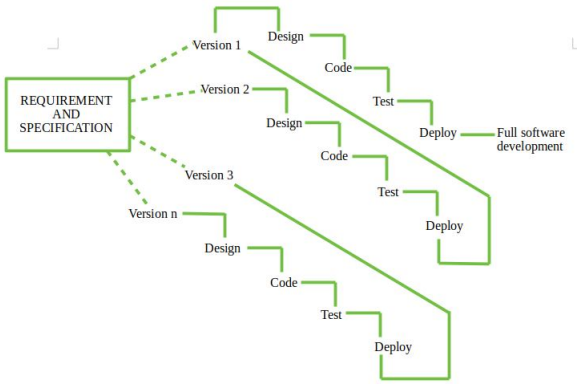

Fig. 6: Mètode de Desenvolupament Incremental[17]

1.7 Eines del Desenvolupament

A continuació es llisten les eines utilitzades per a la configuració del projecte C3SatP a l'entorn de treball, i les usades per al desenvolupament d'aquest treball.

- **Overleaf**: Editor de documents, que permet la creació i escriptura d'aquests en format LaTeX, format que s'utilitza en la documentació del projecte C3SatP.
- **GitLab**: Eina de control de versions que permet interactuar amb el repositori del projecte C3SatP.
- **NUCLEO-F446ZE**: Kit de desenvolupament proporcionat per *STM32*, que ens proporciona un sistema de laboratori amb el mateix microcontrolador que fa servir a l'*OBC* del CubeSat.
- **Ac6**: Eina de desenvolupament de software per als sistemes *Nucleo*. És basada en Eclipse i permet la programació d'aquests sistemes.
- **Ubuntu**: Sistema Operatiu basat en Linux, el qual ens permetrà l'execució de diverses proves al codi mitjançant *Google Test*
- **Google Test**: Eina per l'execució de proves de caixa negra i blanca a codis desenvolupats mitjançant el llenguatge C.

1.8 Organització del Document

Aquest document consisteix en una introducció on s'explica el context, els objectius i la metodologia a seguir. A continuació, es presentarà les diverses fases del treball, així com els seus motius, resultats i problemes. Una vegada finalitzades aquestes explicacions, es procedirà a definir el mètode d'avaluació dels resultats, juntament amb una reflexió sobre ells.

El document finalitza amb una conclusió del treball, on s'esmenten les possibles continuacions i millores d'aquest, els agraïments i la bibliografia utilitzada per recollir la informació exposada al document.

2 Desenvolupament del Treball

2.1 Análisi i Disseny

Aquest va iniciar amb la fase **TFG-FA-01**, que es correspon amb l'investigació i consolidació de l'informe inicial on es van planificar els objectius, tasques, fases, metodología i eines pel seu desenvolupament. Aquest document pot ser trobat dins del dossier del treball, si la seva consulta és requerida. A més, la fase també engloba la documentació dels seus requeriments.

Per a la seva realització, es va procedir a realitzar un conjunt d'entrevistes amb el tutor, el qual exerceix la funció de client dins del treball, en la que van sortir a la llum un seguit de funcionalitats i especificacions indispensables al producte final. Diverses iteracións van ser necessàries per a consolidar la llista de requeriments, per això, aquestes entrevistes han sigut en persona i vía correu electrònic en aquelles ocassions on al client li era impossible reunir-se.

Una vegada realitzada la llista es va procedir a la seva redactació. Aquest és dividit en dues parts, en la secció de requeriments funcionals s'indiquen aquelles funcionalitats o tasques que el software ha de permetre executar, entre ells trobem que el producte final ha d'haver assolit els objectius **TFG-OBJ-01, TFG-OBJ-02 i TFG-OBJ-03**. En canvi, els requeriments no funcionals especifiquen les limitacions del software.

Alguns dels requeriments més destacables indicaven el següent:

- Es seguirà la metodologia incremental per al desenvolupament del treball
- S'utilitzarà el *Minimal Port* de la versió de Micropython 1.11
- Tant el projecte C3SatP com el Micropython han de romandre el màxim d'independents possibles
- El projecte original de micropython ha de romandre intacte, és a dir, tota la part que no té a veure amb el *Minimal port* no es pot modificar, per a així facilitar les futures actualitzacions de la versió de Micropython.
- S'utilitzarà el sistema que ens proporciona el kit *NUCLEO-F446ZE* com a entorn de proves
- Se seguirà la filosofia del C3SatP en tot moment, mantenint l'estructura de fitxers i dels documents establerta
- Es desenvoluparan els objectius anteriorment descrits

Amb la consolidació del document de requeriments, s'inicia la segona fase del treball, amb l'objectiu de dissenyar, implementar i provar l'objectiu **TFG-OBJ-01** i **TFG-OBJ-02**.

2.2 Funcionalitats del Micropython al C3SatP

L'objectiu **TFG-OBJ-01** es caracteritza per modificar el projecte *C3SatP* amb el fi de poder utilitzar les funcionalitats de Micropython, explicades anteriorment al punt **1.4 Micropython**.

Com s'ha dit amb anterioritat al punt *1.3.2 Software*, el *C3SatP* és dividit en **Sistema Operatiu, Firmware, Basis i Aplicació**. L'intenció és afegir un cinqué membre dins del *C3SatP*, Micropython, que permetrà l'ús de les seves funcionalitats dins del *C3SatP*. Així doncs, hi han dues tasques a desenvolupar en aquesta fase, primerament s'afegirà Micropython al projecte i, posteriorment, es desenvoluparà una *aplicació*, per provar que es pot crear un software de control a partir de les eines que ens proporcionen el **Sistema Operatiu, Firmware, Basis i Micropython**.

Per a dur a terme aquestes dues tasques, va haver una certa reestructuració dels directoris del projecte *C3SatP*.

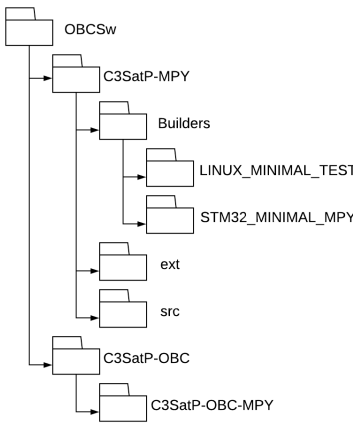

Fig. 7: Estructura de directoris importants, per aquest treball, del projecte C3SatP

El directori que engloba tots els fitxers de codi del projecte *C3SatP* s'anomena *OBCSw*, dins d'ell podem trobar les seves queatre parts *Sistema Operatiu, Firmware, Basis i Aplicació*. El directori encarregat d'emmagatzemar les *aplicacións* del projecte s'anomena *C3SatP-OBC*. Per a provar si es pot crear un software de control utilitzant Micropython es necessita crear una nova aplicació, aquesta va tenir ser anomenada *C3SatP-OBC-MPY*. Encara així, manca la presència de Micropython dins del *C3SatP*, el qual abastirà el *C3SatP-OBC-MPY* amb les seves capacitats. Per a incloure'l, es va acordar la creació d'un nou directori, al mateix nivell que la resta d'eines del *C3SatP*, anomenat *C3SatP-MPY*, que contindrà:

- **ext**: Directori que conté tots els arxius del projecte Micropython v1.11 i els quals no poden ser modificats
- **src**: Directori que contindrà els arxius de codi implementats al llarg d'aquest projecte, els quals el proveiran de noves funcionalitats.
- **Builders**: Directori que conté diferents configuracions de Micropython, especialitzades per a compilar una solució preparada per a funcionar en un determinat entorn o per a una determinada finalitat. Per exemple, el projecte *LINUX_MINIMAL_TEST*, permet compilar una versió per fer test dins d'un entorn Linux, mentre que el projecte *STM32_MINIMAL_MPY*, permet compilar una versió executable dins d'arquitectures *STM32*, com pot ser el nostre entorn de proves o l'*OBC* del *C3SatP*.

Aquestes configuracions que trobem al *Builders*, abastiran el *C3SatP-OBC-MPY* amb les funcionalitats de Micropython mitjançant una llibreria estàtica[18], la qual podrà ser compilada i inclosa per la utilització de Python dins el *C3SatP*. D'aquesta forma, s'aconsegueix que ambdós projectes siguin el més independent possible, respectant un dels requeriments no funcionals del treball, tal com podeu veure al punt *2.1 fase Inicial*.

Per a crear aquesta llibreria s'ha hagut de modificar el fitxer de *Makefile* del *Minimal port*, degut a que l'arxiu original genera un executable, mentres que ens interessa que generi una llibreria estàtica amb totes les seves funcionalitats. La seva modificació va comportar la creació d'un nou arxiu, anomenat *initMPY*, que conté per una part, funcions del codi original de Micropython modificades per a ser una llibreria estàtica, i noves funcions que adapten les ja existents a l'entorn del *C3SatP*.

Amb la seva creació i una vegada superada la fase de proves, es va finalitzar l'objectiu **TFG-OBJ-01**, obtenint un nou directori de treball dins el projecte, en el qual existeix el Micropython i un projecte *Aplicació* que permet la inicialització, l'execució i la finalització de Micropython dins d'un *thread* del projecte C3SatP i, dins d'un altre, executar alguna funcionalitat del C3SatP, permetent així l'ús de les funcionalitats dels dos a la vegada.

2.3 Execució d'arxius precompilats

L'objectiu **TFG-OBJ-02**, consisteix en la implementació de la funcionalitat de rebre, esborrar i executar arxius precompilats. Aconseguir aquesta fita suposaria a més, restar treball al CubeSat, ja que tal com s'ha mencionat al punt

1.3.3 Llenguatges Compilats i Interpretats, els llenguatges Interpretats, com Python, interpreten el codi escrit pel programador i el transformen a codi màquina en temps d'execució. Permetre l'execució d'arxius precompilats suposa llavors, permetre l'execució d'arxius que contenen el codi màquina, és a dir, no fa falta el pas d'interpretació, ja que s'ha fet prèviament al seu enviament al CubeSat. Això permet reduir la càrrega de treball, estalviant energia i temps que poden ser invertits en l'execució d'altres tasques.

A més, els fitxers enviats des de terra han sigut prèviament avaluats en cerca d'errors, i per tant, al moment de ser enviats, s'ha comprovat exhaustivament que són funcionals, fet que comporta que la interpretació del satèl·lit és prescindible, ja que aquest es limita a executar codi.

Per a fer-ho, s'implementaria un gestor d'arxius per aquesta funcionalitat, permetent l'emmagatzematge, cerca, execució i esborrament d'aquests arxius.

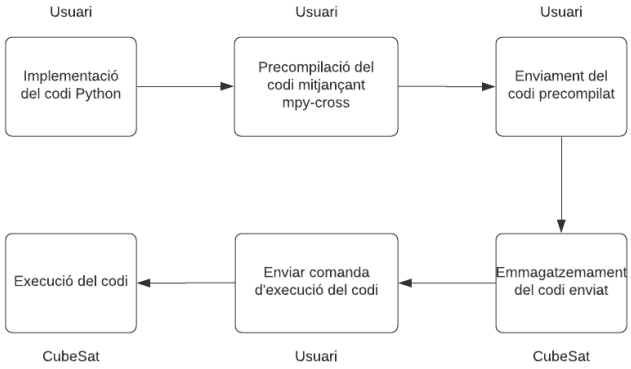

Fig. 8: Passos per a l'execució d'un arxiu al CubeSat

El gestor d'arxius disposa de dues formes per a cercar un fitxer, pot fer-ho mitjançant el seu nom o mitjançant la seva posició dins de l'estructura, el mateix fa per l'esborrat o l'execució. Si s'escull l'esborrat de l'arxiu, el codi alliberarà l'espai ocupat per ell i permetrà desar-ne un de nou en el seu lloc. D'altra banda, si es decideix executar el codi, es procedeix a l'execució d'un codi precompilat seguint les funcions de Micropython.

Pel que fa a la interpretació dels arxius Python en terra, s'utilitza l'eina *mpy-cross*, la qual els transforma a codi màquina[19]. Aquesta eina pot ser trobada dins del mateix projecte de micropython, o pot ser descarregada *online*.

Cal destacar, que per implementar aquesta funcionalitat, va ser necessari l'ús de l'algoritme *Garbage Collector*[20] de Micropython, ja que permet esborrar totes l'espai de memòria ocupat per l'execució d'un fitxer una vegada hagi finalitzat, evitant que la memòria RAM del microsatèl·lit sigui col·lapsada per dades que no es tronaran a utilitzar.

Així doncs, una vegada acabada la fase **TFG-FA-02** i assolit l'objectiu **TFG-OBJ-02** el projecte és capaç de, dins d'un *thread* executar alguna funcionalitat del C3SatP i, en un altre, inicialitzar, emmagatzemar, esborrar i executar indefinidament un fitxer precompilat de Python.

2.4 Creació de llibreries Python

L'inici de la fase **TFG-FA-03** consisteix en assolir l'objectiu **TFG-OBJ-03**, que preten demostrar que es poden crear llibreria a Micropython que no solament emmagatzemin funcions, sinó que permetin lligar aquestes funcions a certes funcionalitats del *C3SatP*. D'aquesta forma, els usuaris podrien escriure codi Python, enviar-lo al CubeSat i utilitzar funcionalitats externes al propi Micropython, com per exemple, poder paralitzar l'execució d'aquest un nombre determinat de segons, funcionalitat que pertany al sistema operatiu *FreeRTOS*.

La part Basis del *C3SatP* conté una gran quantitat de funcionalitats pel control del microsatèl·lit. Així doncs, per a demostrar que les llibreries de Micropython poden utilitzar funcionalitats externes a ell, es va escollir desenvolupar-ne una, la qual contindrà una de les funcionalitats més senzilles que aporta *Basis*, anomenada *Sleep*, que és capaç de bloquejar l'execució del *thread* que la crida el nombre de milisegons que hi indiqui. Així doncs, la llibreria permetria al usuari pausar l'execució de *thread* de Micropython un nombre determinat de milisegons.

Micropython permet, de sèrie, la inclusió de llibreries creades per l'usuari, així doncs no s'ha hagut de modificar cap fitxer de Micropython, sinó que la feina d'aquest objectiu ha consistit en la investigació i la creació de la llibreria. Mitjançant els escassos manuals de Micropython existents, i certes entrades al fòrum del projecte, es va entendre com s'inclouen les llibreries i es va dissenyar la seva estructura.

Primerament, les funcions de la llibreria són implementades a un arxiu de codi. Aquestes funcions són desades dins un diccionari, el qual utilitza Micropython per a associar l'espai de memòria on localitzar la funció amb una sentència de codi. Per exemple, es troba la línia *time.sleep* dins del codi Python, Micropython sap que es tracta d'una funció de la llibreria *time* i la buscarà dins del diccionari de la llibreria per a executar-la.

Aquest diccionari és desat a la vegada dins d'un objecte *mòdul*, que s'inclou a la compilació mitjançant una llista de llibreries, encarregada d'emmagatzemar una referència de cada objecte mòdul per a afegir els diccionaris que contenen a la compilació. Podeu veure una vista més detallada a la *Figura 8* d'aquest document.

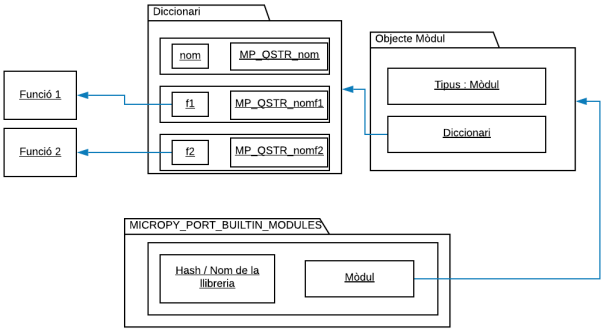

Fig. 9: Estructura de les llibreries a Micropython

Així doncs, es va crear una llibreria de nom *C3task*, la qual engloba la funció *sleep*, que rep un nombre enter com a paràmetre, on l'usuari indicarà quants mil·lisegons vol aturar l'execució, i crida la funció del *C3SatP* que executa aquesta pausa durant el temps indicat. A més, és capaç de detectar errors en els paràmetres d'entrada, per exemple si el nombre és menor o igual a zero o si es tracta d'un nombre flotant, entre d'altres.

Una vegada finalitzada, es va procedir a redactar un manual d'usuari perquè aquests puguin desenvolupar noves funcions o noves llibreries dins d'aquest. Per a fer-ho, es va utilitzar la documentació que proveeix Micropython respecte aquest tema [21].

Amb la finalització de la llibreria i el manual, es va donar com a finalitzada la fase **TFG-FA-03** i l'objectiu **TFG-OBJ-03**, assolint així tots els objectius no opcionals del treball.

2.5 Llibreries dinàmiques

Durant l'última fase del projecte es va procedir a la investigació de l'objectiu **TFG-OBJ-OPC-01**, que pretenia intentar fer que les llibreries foren incloses dinàmicament, tal com es fa amb els arxius d'execució. Aquest objectiu ja va ser declarat com a opcional a causa del curt temps del qual es disposa i a la dificultat d'aquesta. Durant la fase d'investigació, es va entendre el funcionament de la inclusió de les llibreries al Micropython. Com s'ha dit en l'anterior punt, les llibreries són definides dins un array que emmagatzema apuntadors a les estructures de tipus mòduls de les llibreries (aquelles estructures amb el diccionari que conté un apuntador a cada funció de la llibreria). Aquest array es defineix dins de la part modificable de Micropython, i és utilitzat dins d'una funció que s'encarrega d'afegir certs objectes a la compilació de Micropython, entre ells les llibreries. Aquí surgeix un problema, ja que aquesta funció es troba a la part no modificable del Micropython i, per tant, és intocable. La funció requereix de l'array, que és definit externament, i a més, és definit mitjançant un *define*, és a dir, es tracta d'un array del precompilador que ha de ser definit manualment i, per tant, tampoc és modificable.

```
90 #define MICROPY_PORT_BUILTIN_MODULES \
91         { MP_OBJ_NEW_QSTR(MP_QSTR_c3task), (mp_obj_t)&mp_module_c3task }, \
92
```

Fig. 10: Definició de l'array de llibreries

Així doncs, aquest últim objectiu s'ha desestimat a causa de l'impossibilitat de permetre la inclusió dinàmica de llibreries respectant els requeriments inicials.

2.6 Proves a l'entorn del client

La fase **TFG-FA-04**, finalitza amb la comprovació del funcionament del codi a l'entorn de treball del client i a l'*On Board Computer* del projecte *C3SatP*. Per a fer-ho, es va citar un dia a l'*Institut de Ciències de l'Espai*, on vam comprovar que el projecte no era capaç de compilar dins de l'entorn de treball del client. Així doncs, es va iniciar una fase d'investigació de l'error i finalment va ser solucionat, permetent la seva compilació en l'entorn de treball del client.

Per últim, es va comprobar si era possible la programació del Hardware del CubeSat i la seva capacitat d'executar Micropython, sent el seu resultat positiu i donant per finalitzat el desenvolupament del projecte.

2.7 Fases de Proves

En aquesta secció es descriurà com s'ha procedit a la realització de les fases de test, però abans, cal destacar que solament s'ha implementat i executat proves per aquelles funcionalitats desenvolupades al llarg del treball i, per tant, les funcionalitats originals de Micropython i del C3SatP no han sigut provades.

Aquesta fase s'ha produït sempre després de la finalització d'un objectiu, amb la finalitat d'esbrinar si aquesta ha sigut idònia, i tots els requeriments relacionats amb ell es compleixen. S'han executat tant tests de caixa negra com de caixa blanca.

Per una banda, els tests de caixa negra validen si el codi compleix els requisits funcionals, i per tant, tots els tests de caixa negra produïts al llarg del projecte, volen comprovar-ho o, pel seu defecte, comprovar si és capaç de suportar errors, com *inputs* inesperats. Dins d'aquest grup podem trobar *casos de prova d'acord amb els casos d'ús*, els quals són la gran majoria, i *particions equivalents*[22].

Pel que fa als test de caixa negra, s'han implementat de dos tipus, automàtics i manuals.

Els tests automàtics, tal com el seu nom indica, són codi que revisa els resultats del codi al qual volem fer test i els compara amb els resultats esperats, per a determinar si són iguals i donar el veredicte. L'avantatge dels tests automàtics és el fet de poder executar-los de forma ràpida en el moment que s'estimi, estalviant una quantitat notable de temps que testing manual no pot, així doncs la majoria de proves s'han realitzat seguint aquest tipus, per la facilitat de l'usuari de comprovar el correcte funcionament del codi.

Per altra banda, el testing manual és una guia amb un seguit de passos i els seus resultats esperats, perquè una persona els executi i comprovi manualment si els resultats obtinguts són iguals als esperats. Com s'ha dit amb anterioritat, els tests manuals són més lents que els automàtics, però aquests permeten el testing en àmbits on el test automàtic no arriba, com pot ser comprovar si s'ha generat un arxiu en el moment de compilar.

L'eina utilitzada per a la realització de tests al projecte *C3SatP* s'anomena *Google Test*, i ens permet executar tests automàtics. Encara així, la utilització d'aquesta eina dins d'un sistema encastat és impossible, ja que està dissenyada per a ser usada en entorns basats en *Linux*.

El projecte C3SatP permet crear executables per a diferents entorns d'execució, entre ells podem trobar la *STM32-F446RE* o *Linux*. A més, cal recordar que el *Minimal port* de Micropython també permet la generació de llibreries estàtiques per a ser usades en aquests mateixos entorns, i per tant, es pot crear un executable per *Linux* al qual, mitjançant l'eina de *Google Test*, podem realitzar aquests tests automàtics. Els tests manuals són rellevats a comprovar allò que no pot realitzar els tests automàtics, com pot ser comprovar el funcionament del codi a la *STM32-F446ZE*.

Pel que fa al test de caixa blanca, s'utilitza per comprovar si totes les línies de codi són utilitzades, amb la finalitat d'optimitzar la seva escriptura el màxim possible. Per a ells, també s'ha utilitzat l'eina GoogleTest, la qual mitjançant *LCOV* produeix un arxiu *html* capaç de mostrar quines línies de codi s'han executat durant l'execució dels tests automàtics, permetent a l'usuari saber si hi ha línies de codi que no s'han testejat o bé no es fan utilitzar.

En aquest cas, podem trobar *Statement Coverage, Decision coverage, Path coverage* i algun cas de *Condition Coverage*[23]. Pel que fa als bucles, tots ells són d'un nombre fixat de repeticions, i per tant, no s'ha pogut dur a terme un *loop testing*[24], pel simple fet que el codi no permet jugar

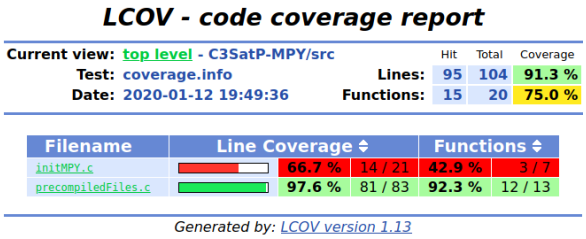

Fig. 11: Exemple d'un arxiu html generat per LCOV, on mostra el percentatge de linies de codi executades dels arxius d'una directori. En aquest cas, són els fitxers que conté la carpeta **src** de Micropython.

amb els seus paràmetres, ja que té un nombre de repeticions fixat.

3 Documentació creada

A continuació es llisten els documents creats al llarg del projecte. Aquests documents són escrits mitjançant la llengua anglesa i la seva estructura és basada en la documentació ja existent del projecte C3SatP, amb el fi de mantenir l'homogeneïtat de les estructures d'aquests. Tota la documentació referent al treball realitzat, és desat dins del directori de documentació del projecte, el qual disposa d'un directori especial per a guardar la documentació realitzada durant el treball.

3.1 Documentació de Requeriments

El document de requeriments va ser realitzat durant la fase **TFG-FA-01**, i segueix el mateix estil que el document de requeriments original del C3SatP, reutilitzant els *templates* utilitzats per a la redacció d'aquests. El document es divideix en **requisits funcionals** i **requeriments no funcionals** i són basats en les especificacions suggerides pel client, o pel seu defecte, el tutor del treball.

Cada requeriment requereix d'un **identificador, un títol, una descripció, la prioritat, la forma de verificació i els requeriments dels quals deriva**. Es pot observar un exemple d'aquesta taula a l'apartat **A.1 Taules de Requeriments del Projecte** de l'apéndix del document.

3.2 Documentació de Disseny

El document de disseny té la funció de documentar tots els diagrames *UML* realitzats al llarg de les fases que han implementat un objectiu. Podeu consultar-ne alguns exemples a l'apartat **B.4 Diagrames UML** de l'apèndix. Aquest conté tres parts, la primera amb una explicació de l'estat del projecte, una altra que conté els diagrames estructurals i l'última amb els diagrames de comportament.

Durant cada fase que requereix la implementació d'una certa funcionalitat s'actualitza el contingut, afegint o actualitzant els diagrames existents. Cada funcionalitat requereix crear nous diagrames de seqüència, d'activitat i d'objectes, i actualitzar els diagrames existents de components, classes, casos d'ús, i d'estats.

Així doncs, el document de disseny reflecteix els resultats de les diferents fases de disseny realitzades al llarg del projecte i és guardat dins del repositori de treball, juntament amb la resta de documentació creada durant el treball.

3.3 Document de Procediments de Test

El document conté un seguit de taules, on cadascuna conté la informació d'un test a implementar. Així doncs, compleix la funció de contenir tots els tests que s'han o s'hauran d'implementar al llarg del projecte, permetent a l'usuari consultar quines proves ha superat el codi i, per tant, quines garanties té sobre aquest. Es divideix a la vegada en dues parts, la primera consisteix en un seguit de taules que contenen informació bàsica del test, per a la consulta ràpida de quins tests existeixen, fent la funció d'índex de totes les proves realitzades.

Aquestes taules especifiquen informació bàsica del test, tal com l' **identificador del test, el títol, la descripció i els requeriments que verifica.** Podeu trobar un exemple d'aquesta taula a l'apartat **A.2.1 Taula d'Informació d'un Test Manual** de l'apéndix del document.

L'altra part consisteix en l'aprofundiment de cada test exposat a la primera part, contenint informació més especifica com el tipus de test (**test manual** o **test automàtic**), les precondicions, les postcondicions i la ruta fins a l'arxiu de test. En trobareu un exemple a l'apartat **A.2.2 Taula d'Informació d'un Test Automàtic** del l'apéndix del document.

3.4 Document de Resultats de Test

Conté tots els resultats dels tests especificats a l'arxiu **de procediments** i es divideix en dues parts. Una part correspon als resultats dels tests manuals i l'altre als tests automàtics. Els tests manuals disposen d'una taula extensa, que conté informació referent a ell, com l'identificador, precondicions, postcondicions i la guia de passos a executar.Podeu trobar un exemple d'aquesta taula a l'apartat **A.3.2 Taula de Resultats d'un Test Manual** de l'apéndix del document.

Aquesta taula permet que un usuari planifiqui el test, escrivint els passos a seguir i els resultats esperats, mentre que un d'altre pot executar-lo i verificar si el codi compleix els resultats esperats. Totes aquestes taules són disponibles per a un possible ús en un futur i es pot trobar juntament amb aquest document dins del projecte C3SatP.

Per altra banda, els tests automàtics disposen d'una taula on es mostra informació com l'Identificador, el nombre de l'execució (referint-se a quantes vegades s'ha executat aquell test fins ara), el resultat, errors trobats, i canvis al codi per a intentar solucionar-ho.

Podeu trobar un exemple d'aquesta taula a l'apartat **A.3.1 Taula de Resultats d'un Test Automàtic** de l'apéndix del document.

3.5 Manual de Micropython

El document es pot trobar dins del projecte, i exposa la funció de cadascun dels directoris inclosos al C3SatP per al funcionament de Micropython, una guía sobre com compilar Micropython i sobre com actualitzar Micropython a una nova versió, les normes sobre el desenvolupament de Micropython i, per últim, una presentació i descripció de totes les noves funcions disponibles i el seu ús.

3.6 Manual per la creació de llibreries a Micropython

Ensenya als futurs usuaris quines són les estructures necessàries per a la implementació i/o expansió de llibreries al Micropython. Així doncs, es detalla informació sobre l'array de llibreries, el seu lloc d'implementació, com crear les noves funcions i lligar-les al Micropython, a més de com nomenar aquestes llibreries i funcions.

4 Resultats

Com s'ha pogut observar, l'objectiu *TFG-OBJ-01* ha sigut completat satisfactòriament, permetent la creació d'una llibreria estàtica que atorga al *C3SatP* les seves funcionalitats. Encara així, aquesta fase va tenir una durada més llarga de la platejada, endarrerint per tant la resta de fases. L'objectiu *TFG-OBJ-02* no solament va patir aquest problema, sinó que es va haver de solucionar diversos problemes referents a l'ús de memòria RAM dels fitxers en execució, utilitzant l'algoritme del *Garbage Collector*. Així doncs, aquest objectiu va aconseguir, fóra de termini, un gestor d'arxius precompilats que permeti el seu emmagatzemament, esborrament i execució. L'objectiu *TFG-OBJ-03* va resoldre l'endarreriment generat durant la implementació del primer objectiu, aconseguint generar la llibreria C3task i permetent utilitzar una funcionalitat externa al mateix Micropython.

```
import c3task

while True:
    c3satp.sleep(500)
    #  Temperature = c3satp.readSensor(1)
    # c3satp.sendmessage(Temperature)
```

Fig. 12: Exemple de codi que usa la llibreria C3Task. Les línies comentades són possibles funcions que es poden desenvolupar en un futur, i per tant, també poden ser considerades una línia de continuació.

Una vegada es va descartar l'objectiu *TFG-OBJ-OPC-01* i es va comprovar que el projecte *C3SatP*, amb les modificacions fetes, era funcional a l'entorn de treball de client, es va donar com a finalitzada la fase de desenvolupament d'aquest treball, donant com assolit l'objectiu *TFG-OBJ-04*.

5 Conclusions

El treball ha aconseguit assolir tots els seus objectius, modificant el projecte *C3SatP* perquè pugui utilitzar Micropython per a la construcció d'un software de control de microsatèl·lits, permetent així que l'usuari pugui enviar i executar codi Python per a controlar-lo. A més, el projecte s'ha dut a terme dins dels marcs de l'enginyeria del software, seguint una metodologia definida des de el principi i redactant un conjunt de documents que recullen tots els passos seguits durant el desenvolupament d'aquest treball.

Pel que fa a les línies de continuació, el treball pot millorar implementat un conjunt de llibreries de Micropython que permetin l'ús de totes les funcionalitats del *C3SatP*.

També pot millorar intentant reduir el seu volum, eliminant certes funcionalitats que no siguin utilitzades. Per exemple:

- **Tractament d'errors**: Com s'ha dit amb anterioritat, els codis són revisats abans de ser enviats al satèl·lit, per tant, podem donar per suposat que mai s'enviarà un codi erroni i podem eliminar la funcionalitat evitant el temps que inverteix Micropython a determinar si existeix un error o no.

- **Funcionalitats originals**: Totes aquelles funcionalitats que no s'utilitzin, tals com la *Consola Python* o l'*Execució d'arxius no precompilats*, poden ser eliminades.

L'última línia de continuació té a veure amb el projecte **UCAnFly**. Aquest, tot i quedar finalistes en l'última edició del concurs *Fly Your Satellite!*, tornarà a presentar-se en la següent i llavors, és possible que es provi per primer cop l'ús de Micropython en un microsatèl·lit.

Agraïments

Agraeixo al meu tutor, Lluís Gesa, per donar-me l'oportunitat de realitzar un treball de final de grau d'una temàtica que sempre m'ha apassionat, i per facilitar-me la seva ajuda en tot moment.

A la meva família i els meus amics, per donar-me suport en moments clau del treball.

Referències

[1] **Wikipedia, 2019.** https://ca.wikipedia.org/wiki/Sputnik_1

[2] **IsisSpace, 2019.** https://www.isispace.nl/cubesats/

[3] **Alen Space, 2019.** https://alen.space/basic-guide-nanosatellites/

[4] **BackBlaze, 2017.** https://www.backblaze.com/blog/whats-the-diff-programs-processes-and-threads/

[5] **ST, 2017.** https://www.st.com/en/embedded-software/freertos-kernel.html#overview

[6] **Wikipedia, 2019.** https://en.wikipedia.org/wiki/Real-time_operating_system

[7] **Nanosats, 2019.** https://www.nanosats.eu/

[8] **Skyrocker, 2019.** https://space.skyrocket.de/doc_sdat/lume-1.htm

[9] **ESA, 2019.** https://www.esa.int/Education/CubeSats_-_Fly_Your_Satellite/Fly_Your_Satellite!_programme

[10] **UCA, 2019.** https://ucanfly.uca.es/

[11] **Wikipedia, 2019.** https://en.wikipedia.org/wiki/Commercial_off-the-shelf

[12] **ST, 2020.** https://www.st.com/en/evaluation-tools/nucleo-f446re.html

[13] S.Plamauer, M.Langer . Evaluation of MicroPython as Application Layer Programming Language onCube-Sats. ARCS 2017, April, 3 – 6, 2017.

[14] Maximillian Holliday et al. PyCubed: An Open-Source, Radiation-Tested CubeSat Platform Programmable Entirely in Python. SSC19-WKIII-04

[15] **ST, 2020.** https://www.st.com/en/evaluation-tools/nucleo-f446ze.html

[16] **Wikipedia, 2020.** https://en.wikipedia.org/wiki/Embedded_software

[17] **Geeks For Geeks, 2018.** https://www.geeksforgeeks.org/software-engineering-incremental-process-model/

[18] **Tlpd, 2001.** http://tldp.org/HOWTO/Program-Library-HOWTO/static-libraries.html

[19] **Pypi, 2018.** https://pypi.org/project/mpy-cross/

[20] **Python, 2001.** https://docs.python.org/3/library/gc.html

[21] **Micropython, 2019.** https://docs.micropython.org/en/latest/develop/cmodules.html

[22] **Guru99, 2019.** https://www.guru99.com/equivalence-partitioning-boundary-value-analysis.html

[23] **Guru99, 2020.** https://www.guru99.com/code-coverage.html

[24] **Guru99, 2019.** https://www.guru99.com/loop-testing.html

APÈNDIX

A.1 Taules de Requeriments del Projecte

ID	REQ-MPY-FR-G-01
Title	MPY Library
Description	The Micropython project must be able to create a Static Library that will provide all the functionalities coded in it to any other project.
Priority	H
Verification	R
Parents	-

Fig. 13: Exemple de taula d'informació del test bàsica d'un test manual

A.2 Taules d'Informació del Test

A.2.1 Taula d'Informació d'un Test Manual

Test ID	MPY_TEST_INC1_BBT_INT_TWOTHREADS
Test Title	Create a MPY thread and a C4SatP thread
Test Description	Manual test which pretends to see through console if two different threads, one running C3SatP functionalities and another one running Micropython, can coexist and work perfectly at the same time.
Requirements to Verify	REQ-MPY-FR-SW-03-03

Fig. 14: Exemple de taula d'informació del test bàsica d'un test manual

A.2.2 Taula d'Informació d'un Test Automàtic

Test ID	MPY_TEST_INC1_BBT_UNT_ALPHABET
Test Name	Print Alphabet
Test Type	AUTOMATED TEST
Path to Test	test/increment_1
Pre-Conditions	Have the the test environment set up as needed. LINUX_MINIMAL_TEST has to be built with the Build-Configuration "Debug". The same goes with "Debug-Test"
Post-Conditions	The alphabet (in lower and in uppercase) has been printed and MPY has been deinitialized without a problem.

Fig. 15: Exemple de taula d'informació del test detallada d'un test automàtic

A.3 Taules de Resultats del Test

A.3.1 Taula de Resultats d'un Test Automàtic

Test Case ID	MPY_TEST_INC1_BBT_UNT_ALPHABET
Execution Number	1
Test Result	Passed
Errors Found	Any
Test Result Description	The test passed without any problem, printing the alphabet in lower case and upper case, and doesn't throw any error, as expected
Code Changes Description	Any

Fig. 16: Exemple de taula de resultats d'un test automàtic

A.3.2 Taula de Resultat d'un Test Manual

Test Case ID	MPY_TEST_INC2_BBT_INT_EXECUTESTOP		
Description	Manual test which pretends to see through console if a thread running a precompiled Python code can be stopped.		
Prepared by	Ferran Campos	**Date of preparation**	12/11/2019
Revised / Updated by		**Date of revision**	
Executed by	Ferran Campos	**Date of execution**	12/11/2019
Preconditions	The enviroment is setted up. All Build Variables (specified on the manuals) have been set. The active build configuration is 'Debug' and the C3SatP-OBC-MPY is clean. On obc-v1.0-config, TWOHREADS equals 1. On STM32_MINIMAL_MPY/Porting/build, micropython.a exists.		
Postconditions	The thread has stopped its execution.		

Test activities

Sl. No.	Step Description	Expected results	Results
1	Right click to C3SatP-OBC-MPY	A curtain of options appears	Expected
2	Select 'Build Project'	The project begins to be build, showing a progress bar	Expected
3	Wait for project to end the Building process	The project ends the building process without any error	Expected
4	Open Putty with the baudrate and port that the board uses	A black terminal appears	Expected
5	On Ac6, Click on Debug C3SatP-OBC-MPY Debug	The code is flashed to the board	Expected
6	Click on continuing Debug	The black terminal shows the phrases that were supposed to be printed executing the alphabet.py and a timestamp. But in a certain time, the alphabet.py stops executing, showing only the time stamps.	Unexpected

Conjunto de datos de test

Data type	Data Set 1	Data Set 2	Data Set 3

Test Case Result	Failure. The C3SatP task is never executed and the Micropython one never stops executing.

Fig. 17: Exemple de taula de resultats d'un test manual

B.4 Diagrames UML

B.4.1 Diagrama Cas d'Ús

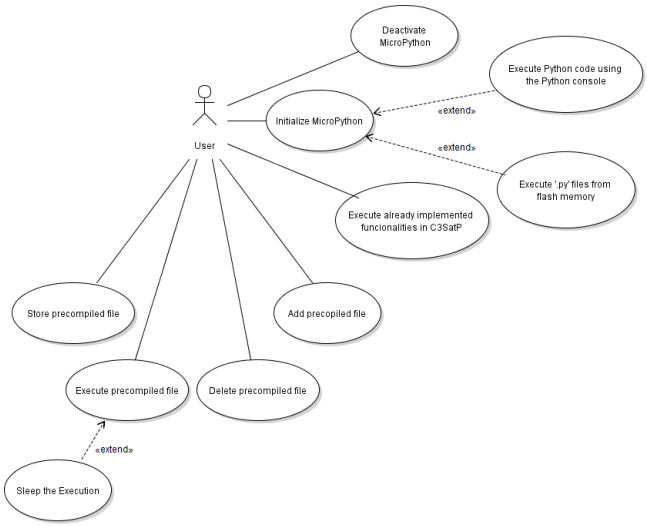

Fig. 18: Diagrama de Cas d'Ús del treball

B.4.2 Diagrama d'Estat

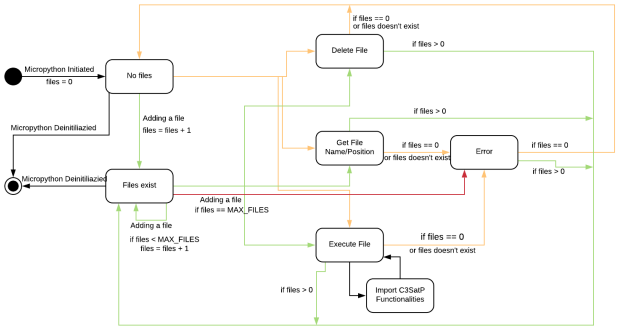

Fig. 19: Diagrama d'Estats del treball

B.4.3 Diagrama d'Activitat

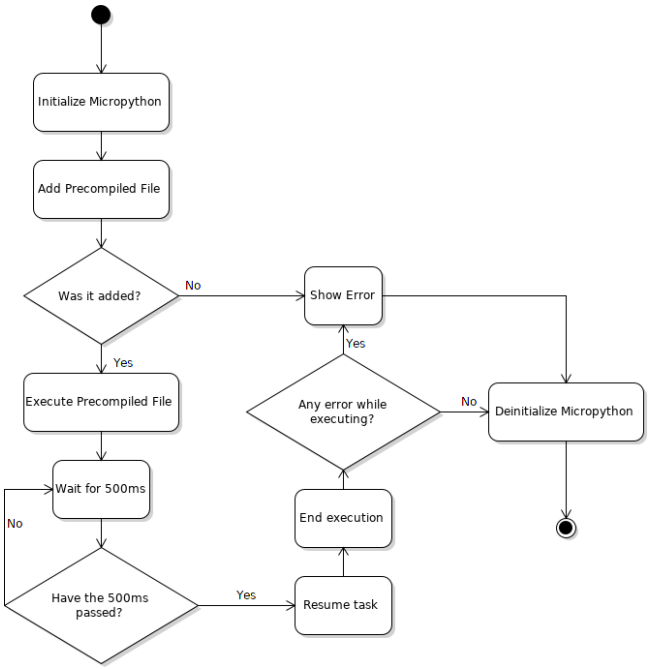

Fig. 20: Diagrama d'Activitats per la funció sleep de la llibreria llibreria c3task

B.4.4 Diagrama Seqüència

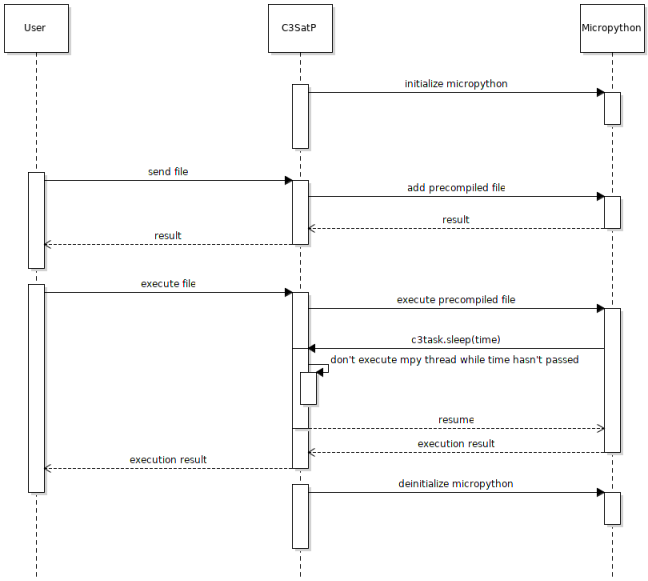

Fig. 21: Diagrama de Seqüència per la funció sleep de la llibreria c3task

B.4.5 Diagrama d'Objectes

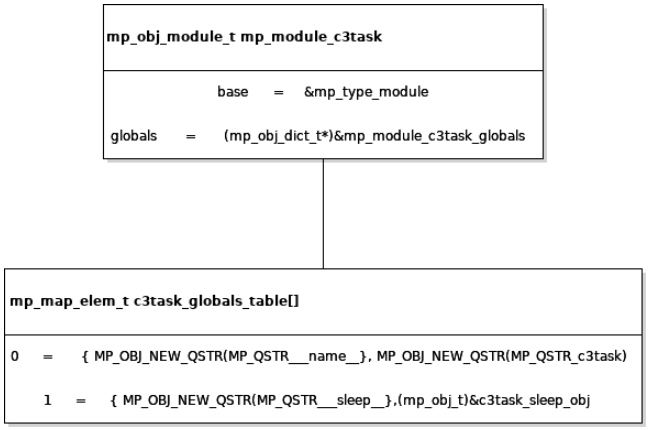

Fig. 22: Diagrama d'Objectes per la funció sleep de la llibreria c3task

B.4.6 Diagrama de Components

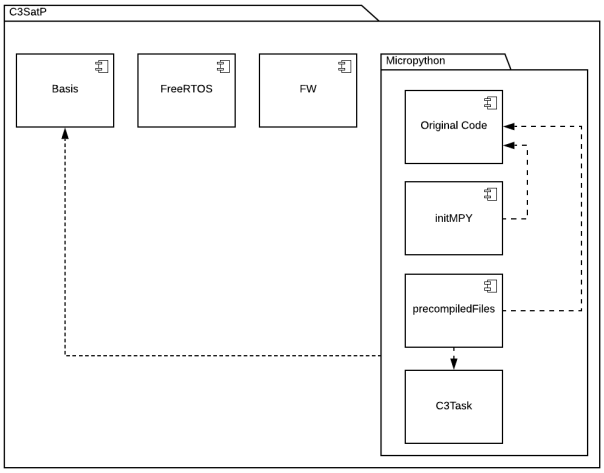

Fig. 23: Diagrama de Components del treball

B.4.7 Diagrama de Classes

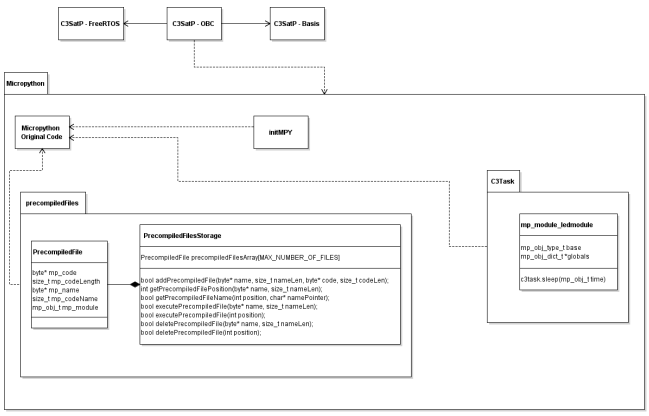

Fig. 24: Diagrama de Classes del treball

C.5 Tasques

- **TFG-TK-01 Planificació de les tasques**: La primera tasca a realitzar es tracta d'un període d'anàlisis d'objectius on es fixen les tasques que formen el projecte, així com les dates d'entrega basades en el document *03-Avaluació.pdf* que es pot trobar al portal web del treball de final de grau.
- **TFG-TK-02 Realització de l'informe inicial**: La tasca, com el seu nom indica, consistirà a dedicar un parell de setmanes a la realització de l'informe inicial on s'explicarà la motivació, s'exposaran els objectius, la planificació, la metodologia del treball i les eines que s'usaran per al desenvolupament d'aquest.
- **TFG-TK-03 Captura de requeriments**: La tasca, que finalitza la fase, implica la captura i formalització dels requeriments del projecte dins d'un document dedicat a ells i el qual tindrà un format similar al document de requeriments del projecte *UCAnfly*.
- **TFG-TK-04 Disseny de l'adaptació**: La fase començaria amb aquesta tasca, la qual pretén iniciar una fase d'anàlisi i disseny del software a implementar per aconseguir l'objectiu *TFG-OBJ-1*. Durant aquesta es realitzarà un document que contindrà tots els diagrames UML generats durant la fase.
- **TFG-TK-05 Implementació de l'adaptació**: Durant aquesta, mantenint el document creat a la tasca *TFG-TK-04* com a guia, s'implementarà el codi per complir l'objectiu *TFG-OBJ-2*. La duració d'aquesta serà curta, llevat que durant l'estada de pràctiques es va aconseguir aquest objectiu, encara que no seguint l'Enginyeria del Software, com es vol fer en aquest cas. Així doncs, la tasca consisteix a crear una llibreria estàtica del codi Micropython, en comptes d'implementar codi, i enllaçar-la al projecte UCAnfly, el qual podrà usar les funcions definides a l'altra.
- **TFG-TK-06 Test de l'adaptació**: Una vegada s'hagi completat l'adaptació dels dos projectes, es procedirà a l'escriptura dels casos de test i a la seva pròpia execució, amb l'objectiu de validar que, el projecte UCAnfly, pot invocar les funcions estàndard de Micropython, com per exemple executar fitxers o crear una consola Python. Tots els casos de prova quedaran recollits dins d'un document que serà desat dins del repositori del projecte.
- **TFG-TK-07 Resolució d'errors de l'adaptació**: Durant aquesta tasca, la qual es durà a terme juntament amb la *TFG-TK-06*, pretén la resolució dels errors trobats durant la fase de proves. Així doncs, aquesta tasca s'executarà fins que la *TFG-TK-06* sigui completada exitosament.
- **TFG-TK-08 Disseny per l'execució de codi precompilat**: Aquesta tasca posa en marxa una fase d'anàlisis i disseny de Software per a la implementació de l'objectiu *TFG-OBJ-2*. Així doncs, es crearan un seguit de diagrames UML, els quals es recolliran en un document de disseny, amb la intenció de deixar clar sobre el paper com s'implementarà el codi.

- **TFG-TK-09 Implementació per l'execució de codi precompilat**: La tasca consisteix en l'escriptura del codi segons la base feta a la tasca *TFG-TK-08*. En completar la fase, obtindrem un codi que assoleixi l'objectiu *TFG-OBJ-2*.

- **TFG-TK-10 Proves del codi capaç d'executar d'arxius precompilats**: Es tracta de l'equivalència de la tasca *TFG-TK-06* però pel codi implementat a la *TFG-TK-09*. Es crearan casos de test i s'executaran per a trobar-hi errors. Tots ells seran recollits dins d'un document que es desarà al repositori.

- **TFG-TK-11 Resolució d'errors del codi capaç d'executar d'arxius precompilats**: Consistirà en la solució dels errors trobats a la tasca *TFG-TK-10*. Si no existeixen errors, llavors aquesta tasca no s'executarà, en cas contrari sempre s'executarà després d'ella, és a dir, si existeixen errors, se solucionaran i es tornarà a executar tots els casos de prova existents del projecte.

- **TFG-TK-12 Realització de l'informe de progrés 1**: La duració d'aquesta és igual a la de la fase, ja que durant tota ella es recollirà, dins l'*Informe de progrés 1*, els avenços aconseguits. L'entrega de l'informe, dia **28 d'Octubre de 2019**, coincideix amb la finalització de la tasca i de la fase.

- **TFG-TK-13 Disseny dels mòduls**: Durant aquesta fase es realitzaran els diagrames UML corresponents a tots els mòduls que s'implementaran en la següent tasca. Aquests diagrames, com tots els altres, seran recollits en un document que es desarà dins el repositori del projecte.

- **TFG-TK-14 Implementació dels mòduls**: Es realitzarà el codi corresponent a tots els mòduls Micropython basant-se en els diagrames de la tasca anterior.

- **TFG-TK-15 Proves dels mòduls**: Durant aquesta fase s'implementaran i s'executaran els casos de test corresponents als mòduls, amb l'objectiu de validar si els requeriments es compleixen. Aquests casos de prova, com els altres, seran recollits en un document que es desarà al repositori del projecte.

- **TFG-TK-16 Resolució d'errors dels mòduls**: Durant aquesta tasca es pretén resoldre tots els errors que apareguin durant l'execució dels casos de prova de la fase anterior. Aquests errors també seran recollits al document de casos de proves. Després de resoldre els errors, es tornarà a realitzar la tasca *TFG-TK-15*. Aquest bucle durarà fins que no apareguin errors en l'execució dels casos de prova.

- **TFG-TK-17 Realització de l'informe de progrés 2**: Es realitzarà al llarg de tota la fase l'escriptura de l'informe de progrés 2, en el qual es presentaran els avenços i problemes ocorreguts durant la fase. Aquest informe ha de ser entregat la data del *17 de Decembre de 2019* en la qual finalitzarà la fase.

- **TFG-TK-18 Realització de l'informe final**: L'última fase comença amb la realització de l'informe final, el qual té una entrega prèvia la data del *2 de Gener de 2020* i la data final d'entrega el dia *20 de Gener de 2020*.

- **TFG-TK-19 Realització de la presentació**: Es realitzarà la presentació la qual s'utilitzarà d'eina visual per la defensa del treball. Aquesta té una entrega prèvia la data del *10 de Gener de 2020* i la data final d'entrega el dia *20 de Gener de 2020*, coincidint amb l'entrega final de l'informe final.

- **TFG-TK-20 Realització del pòster**: Es realitzarà el pòster on es resumirà el treball realitzat. Aquesta tasca començarà després de l'entrega final i tindrà la data d'entrega el dia *24 de Gener de 2020*.

- **TFG-TK-21 Configuració i aprenentatge de l'entorn de treball**: Consisteix a aprendre, en base de la lectura del manual d'usuari proporcionat pel tutor, a configurar l'entorn de treball, així com l'estructura del projecte i la filosofia d'aquest.

- **TFG-TK-22 Configuració i aprenentatge de l'entorn de test**: Consisteix a configurar Google Test en el sistema per a, posteriorment, realitzar un aprenentatge sobre les funcionalitats que aquest proporciona.

C.6 Diagrama de Gantt

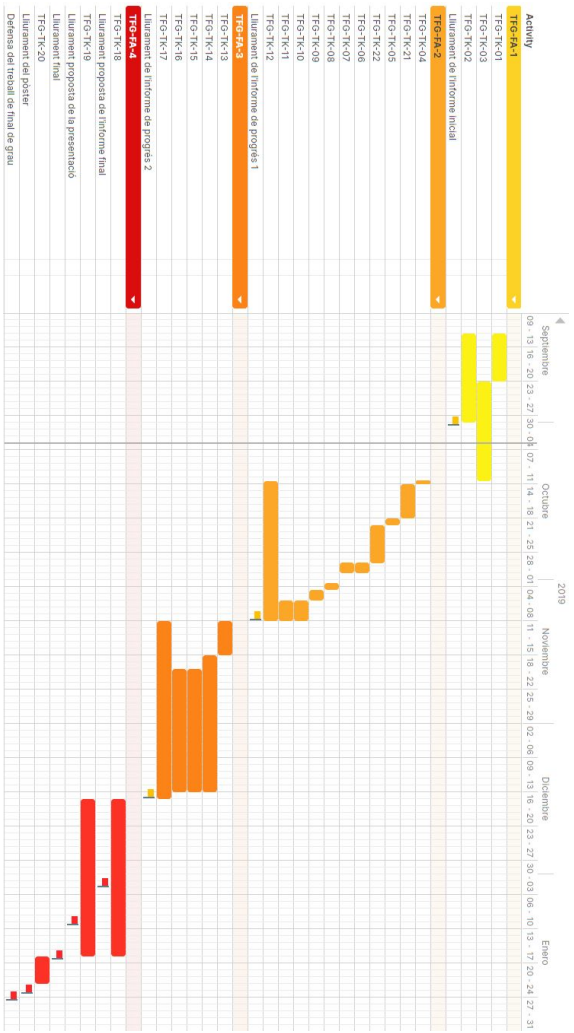

Fig. 25: Planificació del treball