LECTURA, CONTROL I SIMULACIÓ D'UN BICOPTER:
Informe del projecte

Enric Puigvert Coromina

Laboratori de Sistemes Empotrats 1

Juny de 2020

Enginyeria Electrònica de Telecomunicacions

Universitat de Barcelona

Contents

INTRODUCCIÓ

Aquest projecte forma part de l'assignatura, de la carrera Enginyeria Electrònica de Telecomunicacions de la Universitat de Barcelona, Laboratori de Sistemes Electrònics 1. És un projecte diferent als que hem fet anteriorment a altres assignatures, ja que té la intenció de fer que els alumnes treballin de forma autònoma i en equip.

Aquest any, però, a causa de la COVID-19, ens hem hagut de readaptar (tant professors com alumnes), i fer el projecte a distància, individualment i adaptat al material que tenia cadascú a casa.

OBJECTIU

L'objectiu inicial del projecte era dissenyar des de zero un bicopter auto-estabilitzat. Donada la situació hem adaptat el projecte als medis de cadascú, i ens hem centrat més en el software.

En el meu cas, he tingut la sort de disposar dels recursos suficients per poder fer lectures i comunicacions amb sensors reals.

Així doncs, l'objectiu general del projecte ha sigut fer una simulació el més pròxima possible del que hauríem tingut en situació normal. Així doncs, els objectius específics han sigut els seguents:

1. Disseny d'una PCB, en forma de Shield per a una placa de desenvolupament de Texas Instruments MSP432-P401R, per a controlar un bicopter auto-estabilitzat. A part, també ha de contenir un Electronic Speed Control (ESC) dissenyat des de zero per al control de un motor brushless (això no ha canviat ja que ho vam fer abans del confinament).
2. Lectura de dades mitjançat protocol I2C del sensor MPU6050, tipus IMU, que conté un acceleròmetre, un giroscopi i un termòmetre.
3. Fer una simulació (o "mocking") de les dades que no podem llegir ja que no en tenim els sensors, com poden ser, l'angle de posició dels motors i la fase activa o el magnetòmetre de la IMU que hauríem tingut en situació normal (la que disposo a casa, no conté aquest sensor).
4. Comunicació mitjançant protocol UART amb el PC i el programa de programació visual LabView.
5. Presentació de les dades llegides de la IMU al PC mitjançant LabView, fent un programa estèticament correcte, amb gestió de configuracions i errors i, en el meu cas, un sistema de "mocking" de la MSP432.
6. Extreure'n una aplicació executable.

Per acabar aquesta secció comentar, que s'ha anat fent pràctiques intermitjes per anar assolint conceptes i poder acabar amb el programa final.

DESCRIPCIÓ DE L'APLICACIÓ

L'aplicació que he dissenyat fa d'interfície gràfica entre el que es llegeix al sensor i l'humà. D'aquesta manera, presenta les dades mitjançant gràfics del 3 eixos, la temperatura del sistema (en aquest cas és la ambient) i la fase i l'angle del motor que havíem de controlar amb el ESC dissenyat per nosaltres. Tot això va dins d'una aplicació creada amb LabView que explicaré en l'apartat corresponent. També es pot controlar la configuració de la UART (o VISA) quan tenim l'aplicació en mode IDLE (es a dir, START desactivat). A més, he creat un mode de "mocking" per a "debug/testeig" de l'aplicació de manera que substitueixi la MSP432 (entren les dades que es vulguin pel VISA).

Tots aquest mode van controlats per un seguit de "tabs" o pestanyes, que es navega per elles mitjançant uns botons que canvien segons la pestanya que estiguem en aquell moment. En canvi en tenir el programa aturat, al no funcionar els botons de navegació, s'habiliten els controls de les pestanyes propis de LabView.

Per últim, hi ha un mode "debug" on hi està també ubicat el sistema de "mocking", on s'hi poden veure les dades en "raw" (en cru) i també els missatges d'error.

DESCRIPCIÓ DEL SIMULADOR

Per a fer aquest projecte he fet servir generalment el llenguatge C, amb la llibreria DriverLib, que facilita el codi fent-lo més llegible, per a la programació de la placa MSP432, i llenguatge Python per a la creació de les dades que no podem llegir i que simularem.

El més fàcil és començar a mirar com he extret dades. Com ja he dit, aquestes dades les he extret de dues formes: lectura de la IMU i "mocking" de la resta.

Per al "mocking", he creat un programa en Python que genera dades en dues funcions:

- ***imuData***: Aleatòriament dintre d'uns límits màxim i mínim, genera 3600 dades per simular les de la imu. En aquest cas, només simulem el giroscopi fent dades aleatòries dins dels límits que li posem (li he posat els límits que hauria de tenir la IMU configurats, és a dir, ±4900). De fet generem les dades dels tres sensors, tot i que només en llegirem el del giroscopi.
- ***motorData:*** Les dades es generen en tipus dent de serra, per part de l'angle, i en el cas de la fase, canvia segons si estem en uns rangs concrets, és a dir, de 0 a 120º es genera fase 1, de 121º a 240º, fase 2 i la resta fins a 360º, fase 3. Aquesta funció permet també genera 3600 dades.

Les dues funcions esmentades, a part de generar les dades, les organitzen i col·loquen expressament per a que pugui entendre-les el Code Composers, creant un arxiu de tipus ".h", escrivint els valors en "*arrays"s* del tipus "*const float*" i "*const uint16_t*" escrivint automàticament tot el fitxer, que queda guardat a la carpeta de llibreries del projecte.

Així doncs, acabem tenint 3600 dades de cada sensor preparades per ser llegides per la MSP432 en un fitxer ".h".

Per la lectura de dades de la IMU faig servir el protocol I2C que em facilita molt la comunicació. Mitjançant una interrupció que genera la IMU automàticament cada vegada que té dades noves, avisa a la MSP432 que comença a llegir en mode ràfega les dades de l'acceleròmetre, el giroscopi i el termòmetre. Cal dir

que aquestes dades venen dividides en dos bytes, i que per tant, en llegim 6 per sensor de moviment, ja que tenim 3 eixos, i 2 més per al termòmetre, i en format *float*.

Aquestes dades les converteixo a una *string* de caràcters ASCII (ja que m'és més fàcil de tractar les dades, tant en el codi en C com en LabView, encara que perdi més temps, que en aquest cas no ens afecta). Llavors les envio per protocol UART al LabView, caràcter a caràcter (cada caràcter ocupa un byte), per a que aquest les tracti com explicaré en el següent apartat.

Per a la gestió d'aquest programa he utilitzat una màquina d'estats lineal, és a dir, que sempre passa al següent estat (per això no estan definits els estats), aconseguint un bucle infinit que permet llegir contínuament les dades reals de la IMU i anar repetint el "mock" de les dades generades amb Python. Tots els protocol funcionen amb interrupcions permetent que el programa funcioni de forma reactiva als inputs exteriors. A més, he posat èmfasi en no sobrecarregar les interrupcions, posant al *main()* les operacions de pes per al processador per no aturar el programa durant el temps d'execució d'aquestes. Ho aconsegueixo posant *flags* a totes les interrupcions.

A més, si ens parem a mirar el *main()* podríem pensar que el programa és curt, però cal tenir en compte que la majoria de les funcions queden ubicades i definides a la carpeta de llibreries *lib*, ubicada al path *IMULabs_proj/lib*. En aquesta hi trobem 3 llibreries amb els seus fitxers *".c"* i *".h"* a part de la que hem creat amb Python pel "mocking". Aquestes llibreries són dedicades a:

- ***i2cLib:*** Conté totes aquelles funcions relacionades amb el protocol *I2C*, des de la inicialització, configuració i interrupcions d'aquest, fins la configuració i lectura de la IMU (ja que funciona amb aquest protocol). També contindrà les variables i *structs* relacionades amb aquestes funcions.
- ***misc:*** Conté totes aquelles funcions i interrupcions d'allò que no podem relacionar amb cap protocol o dispositiu, com poden ser les inicialitzacions del *clock*, la unitat *FPU*, els *timers*, *delays*, etc.
- ***uartLib:*** Conté totes aquelles funcions relacionades amb el protocol UART, tant la inicialització, com la configuració o les interrupcions d'aquest. També contindrà les variables i *structs* relacionades amb aquestes funcions.
- ***mock:*** Conté dades generades en Python per simular que rebem des de sensors que no tenim.

Amb aquest codi, aconseguim llegir dades tant reals com simulades i enviar-les tractades a l'ordinador on podrem fer-ne ús i presentar-les amb el programa LabView, com seguidament explicaré.

DESCRIPCIÓ DEL LABVIEW

El programa de LabView es pot consultar al path *LabView/main.vi*. Per a la construcció del codi, he fet servir dos mòduls:

- **Control i tractament de dades: Conté dos "*whiles*":**
  - "*While"* extern: Configura el VISA i gestiona els errors. Hi tenim lògica de visibilitat d'errors en la pestanya de configuració.
  - *"While*" intern: Primer de tot ens trobem un "*subVI*" que conté un *"case"*. Aquest és el que en cas de estar en l'estat *FALSE*, estem en mode lectura, i per tant, ens trobem el "*VISA Read*". En cas d'estar en l'estat *TRUE*, ens trobem amb un mòdul que fa un "*clear"* dels buffers d'entrada i sortida del "*VISA"*. A part, una entrada, resetejable mitjançant un botó (ho he fet amb una "*Local Variable"*), que permet fer el *"mocking"* del *"VISA Read"*. D'aquest *"subVI*" en surt una *"string"*, que entra a un *"case"* que defineix l'estat del programa. Tenim dos estats: WAIT i READ. El primer esdevé quan tenim el botó START deshabilitat i el segon quan el tenim actiu.
    En l'estat WAIT, bàsicament tenim un buffer de lectura no-visible que llegeix i buida el buffer del *"VISA"*.
    En canvi, en l'estat READ, tenim dos "*subVI"* que destrien les dades i les posen al visualitzador que toca (IMU i motors en gràfics, i la temperatura en una barra. Les fases del motor en leds a part del gràfic corresponent). A més propaguen les dades mitjançant clústers de clústers d'*"strings"* de manera que el gràfic mantingui la dada fins que canviï aquesta. Per presentar les dades es fa mitjançant 3 gràfics de la IMU (acceleròmetre, giroscopi i magnetòmetre, un 4rt gràfic que presenta dades del motor amb 3 LEDs auxiliars que marquen la fase del motor i una barra que mostra la temperatura. Tot té indicadors amb el seu valor numèric.
    Per últim ens trobem, a la part baixa del *"while"* una lògica de control amb un últim *"subVI"* que conté més lògica. Aquest sistema permet primer de tot, que tots els botons START i P_OFF (al codi STOP) es controlin i comportin alhora respectivament, per treure'n una sola senyal conjunta. També permet controlar el següent estat. Veiem que iniciaríem amb l'estat WAIT i en cas de pitjar algun botó START, l'estat canviaria a estat READ. En cas de desactivar algun dels botons de START (que ara es diran STOP), passaríem a l'estat WAIT, i sortiríem del bucle intern, per a refrescar la configuració del *"VISA".*
    En tot cas, si pitgéssim algun botó de P_OFF, primer sortiríem del bucle intern, és a dir, tornaríem a l'estat WAIT, i en cas de tornar a pitjar-lo tancaríem l'aplicació (en cas d'estar en WAIT directament tancaríem l'aplicació). Això ho fem mitjançant la *"flat secuence"* del final de tot on tanquem el programa després de sortir del *"while"* exterior.
- **Control de botons per les pestanyes:**
  Per a la navegació entre pestanyes, com he explicat anteriorment, l'usuari ho fa a partir de botons de navegació. Això ho he programat dins d'un bucle *"while"* extern als anteriorment comentats, per aconseguir una funcionalitat asíncrona. Així doncs, tenim un *"property node"* que ens diu en quina pestanya estem, que entra en un case. Segons la pestanya que estem posem botons de les altres per poder-nos-hi referenciar i en cas de pitjar un botó, s'escriu al *"property node"* de escriptura de pestanya la pestanya demanada per l'usuari. Aquest bucle no s'atura mai fins a tancar el programa.

Per últim podem veure que tenim a l'inici uns *"property node"* per a desactivar la visibilitat dels controls de les pestanyes (ja que les controlem amb els botons de navegació), la pestanya per la qual volem iniciar el programa i també per "resetejar" els valor del *"VI"* a l'inici. Al final tenim el mateix *"property node"* que activa la visibilitat dels controls de les pestanyes per a que el programador pugui moure's pel *"front panel"* (ja que els botons no funcionen amb el programa aturat).

Cal destacar que tenim els cables d'error connectats entre *"property nodes"* per establir un ordre d'execució de dreta a esquerra. D'aquesta manera aconseguim que el codi s'executi seqüencialment en cada mòdul.

RESULTATS

El resultat que n'obtenim és una aplicació totalment funcional, que permet llegir les dades d'una IMU a gran velocitat i amb bona precisió i amb una interfície intuïtiva.

En cas d'haver tingut la possibilitat de fer el curs amb normalitat i tenir el bicòpter, es podria partir del treball fet i posar-hi capes per sobre per poder-ne treure la aplicació final que en un principi havia estat planificada.

CONCLUSIONS

Puc començar amb les conclusions amb allò que he aconseguit, que es arribar a tots els objectius que s'havien proposat pel projecte. Per altra banda, fent la vista enrere, crec que hi ha punts en LabView que podria millorar com per exemple no fer manualment el triatge de dades, sinó fer servir la funció de LabView *"SpreadSheet String to Array"*, que vaig descobrir tard, i que per no perdre la feina feta, i com a repte personal no vaig fer servir. Pel mateix motiu, no tinc un sobre-dimensionament en la lògica de control dels botons d'START/STOP i P_OFF. El repte en aquest cas ha sigut fer-los canviar de posició segons la pestanya i que tots funcionessin alhora. Ara potser ho posaria fora del clúster i per tant, només tindria un botó de cada. Tot i així, crec que això m'ha permès experimentar amb LabView i fer-me més a la idea de com programar-hi. De fet, dels errors se n'aprèn, no?

Com a conclusió general, puc dir que aquest projecte ha servit per assolir un nivell de programació en C de protocols de comunicació, lectura i tractament de dades i programació en general de la MSP432 i LabView bastant alt per part de l'alumnat. Per altra banda ha ajudat a consolidar conceptes de les assignatures que s'estan cursant alhora com MISE o Potència i Control.

Tot i així podria dir, i ara ja com a opinió personal, que aquesta assignatura s'hauria de fer després d'haver acabat les assignatures abans esmentades, ja que els coneixements s'adquireixen durant el curs i crec que és impossible (o molt difícil) seguir-lo en condicions normals en cas de cursar totes les assignatures i poder-ne assolir els conceptes a temps també és un repte molt gran.