Universitat de Girona
**Escola Politècnica Superior**

# Treball final de grau

**Estudi:** Grau en Enginyeria Informàtica

**Títol:** Quadcopter Arduino

**Document:** Memòria del Treball de Fi de Grau

**Alumne:** Joaquim Pascual Solà

**Tutor:** Antonio Bueno Delgado
**Departament:** Arquitectura i Tecnologia de Computadors
**Àrea:** Arquitectura i Tecnologia de Computadors

**Convocatòria (mes/any):** 12/2020

# Índex

# 1. Introducció, motivacions, propòsit i objectius del projecte

Des de fa uns anys podem observar un dels elements que més a cridat l'atenció dins el mercat tecnològic han sigut els drons, tant per un ús recreatiu com professional (per exemple en el sector audiovisual ja que permeten gravar plans que abans era molt difícil i car aconseguir).
Si ens centrem en l'ús recreatiu podem trobar una gran varietat de drons al mercat, des de drons molt senzills amb preus que no superen els 100 euros fins a drons amb molt bones característiques amb preus per sobre dels 1.000 euros. També trobem la opció de comprar les diferents peces que formen un dron per separat per tal de muntar el nostre propi dron amb les característiques desitjades i aprendre més sobre el hardware d'aquests.
La idea d'aquest projecte és anar un pas més enllà i, no només muntar un dron, sinó també programar el seu controlador mitjançant la plataforma Arduino. El dron que muntarem serà de iniciació per aprendre, serà gran i fàcil de manejar, muntar, canviar components a posteriori, etc.

L'objectiu que espero assolir en aquest treball final de grau és realitzar el muntatge d'un dron, més concretament d'un quadcopter, desenvolupar el seu controlador sota una llicència Open Source i documentar-ho amb l'objectiu que qualsevol persona amb un mínim coneixement en Arduino i electrònica pugui aconseguir fer-se el seu propi dron i entendre els conceptes següents:

- Com funciona un dron a nivell de hardware i entendre cada component per separat.
- Com desenvolupar el controlador que permetrà que un dron pugui volar de forma controlada i segura.
- Saber més sobre la legislació vigent sobre drons.

Aquest projecte pot servir tan a nivell educatiu per ensenyar diferents temes relacionats amb l'electrònica i els drons com a persones que estiguin interessades en fer el seu propi dron per tenir una base on llavors poder desenvolupar les funcionalitats específiques que desitgin.

# 2. Estudi de viabilitat

Des de fa uns anys podem observar que l'electrònica no para d'evolucionar, ho podem veure observant coses que fem servir quotidianament com el telèfon mòbil, ordinadors, domòtica, etc. A més a més els components electrònics s'han abaratit molt, això fa que plantejar-se construir un quadcopter o algun aparell semblant a casa no sembli una idea absurda com podia ser fa uns anys.
Sabent els components que necessitem per desenvolupar un quadcopter i que veurem amb detall a l'apartat 6. 1 mirem quin pressupost necessitem pel que fa a material.

| Component | Lloc de compra | Preu / Unitat | Unitats | Total |
|---|---|---|---|---|
| Frame F450 | Banggood | 7,96 € | 1 | 7,96 € |
| Comandament RC + Receptor | Amazon | 58,99 € | 1 | 58,99 € |
| IMU MPU-6050 | Banggood | 2,53 € | 1 | 2,53 € |
| Motor A2212 + ESC 30A + Hèlix | Banggood | 18,50 € | 4 | 74,00 € |
| Bateria 3S 3000mAh 30C | Amazon | 21,57 € | 1 | 21,57 € |
| Arduino Uno | Arduino | 16,00 € | 1 | 16,00 € |
| | | | | 181,05 € |

**Figura 2. 1: Pressupost del quadcopter.**

A aquest pressupost hem de tenir en compte que una gran part del preu és per el comandament RC que hem escollit. Tal i com s'explica s'ha escollit aquest perquè ens permet lligar varis receptors i no haver de tenir un comandament para cada aparell. Agafant un comandament més senzill podríem reduir costos.
També podem reduir costos agafant una rèplica de l'Arduino Uno que en podem trobar per un preu de més o menys 2.75 €, ara bé si comprem una Arduino original estarem donant suport a la seva comunitat.
A part d'aquests costos que hem vist que podríem reduir també hem de tenir en compte que gastarem material electrònic: connectors, resistències, díodes i cables, a més a més si és el primer cop que tenim una bateria LiPo necessitarem un carregador de bateries.

Finalment amb aquestes modificacions ens quedaria el pressupost següent:

| Component | Lloc de compra | Preu / Unitat | Unitats | Total |
|---|---|---|---|---|
| Frame F450 | Banggood | 7,96 € | 1 | 7,96 € |
| Comandament RC + Receptor | Banggood | 37,99 € | 1 | 37,99 € |
| IMU MPU-6050 | Banggood | 2,53 € | 1 | 2,53 € |
| Motor A2212 + ESC 30A + Hèlix | Banggood | 18,50 € | 4 | 74,00 € |
| Bateria 3S 3000mAh 30C | Amazon | 21,57 € | 1 | 21,57 € |
| Arduino Uno o similar | Banggood | 2,73 € | 1 | 2,73 € |
| Carregador bateries LiPo | Banggood | 17,73 € | 1 | 17,73 € |
| | | | | 164,51 € |

**Figura 2. 2: Segon pressupost quadcopter.**

Amb aquests pressupostes podem veure muntar el nostre propi quadcopter ens pot sortir perfectament per un preu inferior a 200€. Cal dir que el quadcopter que muntarem és d'iniciació pel que fa a que té una mida gran i això, juntament amb els motors i hèlixs que hem escollit, faran que sigui un quadcopter fàcil de portar. Això no vol dir que sigui un quadcopter amb limitacions de velocitat, acceleració, duració de la bateria o distància d'abast del control RC sinó tot el contrari, els components que hem escollit ens permeten fer un quadcopter de qualitat i, a la vegada, fàcil de controlar per usuaris aficionats.

# 3. Metodologia i planificació

Al ser un projecte individual i de recerca i aprenentatge fa que sigui difícil establir una metodologia de treball. Tot i això m'he establert la següent planificació per tal d'aconseguir assolir tots els requisits plantejats en aquest projecte:

1. Buscar informació sobre com funcionen el drons, quins tipus hi ha, quin seria més senzill d'implementar, legislació vigent, etc.
2. Estudiar quins són els components necessaris per tal de poder muntar i programar un dron.
3. Realitzar la compra dels components, hem de tenir en compte que si volem aconseguir components bé de preu moltes vegades l'enviament pot tardar 3 o 4 setmanes.
4. Mentre esperem que arribin els components podem començar a buscar informació sobre com funciona cada component per tal d'avançar feina teòrica.
5. És de gran importància conèixer el funcionament de cada part del quadcopter, és per això que fins que no haguem realitzat proves amb els següents components no avançarem a la següent fase.
    a. Comandament RC i el seu receptor.
    b. IMU. Lectura de l'angle d'inclinació.
    c. Motors, ESC i hèlixs.
    d. PID.
    e. Bateria.
6. Una vegada el funcionament d'aquests components estigui assolit podem passar a realitzar el muntatge del dron.
7. Ja muntat haurem de desenvolupar el controlador amb el que haguem aprés anteriorment sobre cada component.
8. Realitzar proves amb el quadcopter funcionant però sense les hèlixs per comprovar que respon correctament a les ordres del comandament RC i a la desestabilització.
9. Un cop comprovat i veient que el funcionament del controlador és correcte serà moment de muntar les hèlixs i buscar els paràmetres del PID que aconsegueixin estabilitzar el dron.
10. Arribats a aquest punt donarem el projecte per finalitzat i pensarem treballs futurs que podem realitzar a partir d'ara.

# 4. Marc de treball i conceptes previs

## 4. 1. Conceptes bàsics sobre drons

Comunament anomenem dron al que oficialment s'anomena **VANT** (per les sigles de Vehicle Aeri No Tripulat) o **UAV** (per les sigles en anglès *Unmanned Aerial Vehicle*), així doncs podem definir dron com a un tipus d'aeronau reutilitzable i sense persones a bord controlat per una persona a distància o amb un cert grau d'autonomia.

Els drons s'utilitzen en molts àmbits i per moltes aplicacions diferents, alguns exemples serien:

- Àmbit militar. Els drons van sorgir originàriament amb motius militars com vigilància o reconeixement.
- Situacions d'emergència. Els drons destaquen per la seva efectivitat en situacions límit, especialment en àrees aïllades o de difícil accés. Són molt útils en zones que van ser afectades per desastres naturals. La seva velocitat de vol permet recórrer grans àrees en poc temps permetent portar l'ajuda necessària o, en una fase prèvia, per avaluar la magnitud del desastre.
- Zones rurals. Per els agricultors, gràcies a les fotos i vídeos en alta definició possibilita la monitorització de grans dimensions i permet la detecció ràpida de plagues. També s'utilitzen per el control del remat.
- Control de incendis forestals. Els primers drons a Espanya es van dissenyar per la prevenció i el control d'incendis forestals. La seva feina és reunir la informació necessària per anticipar-se en tot lo possible pel que fa a la prevenció i expansió d'incendis.
- Investigació. Tal i com hem dit al segon punt, els drons poden accedir a àrees de difícil accés , això sumat a la seva gran velocitat ajuda a diversos àmbits d'investigació com l'arqueologia, la geologia i la biologia.
- Oci. Els drons han entrat de ple al món de l'oci (esdeveniments, cinema, fotografia, etc). La seva mida permet volar molt més baix i més a prop de la gent que un helicòpter real i té moltes més possibilitats de maniobra que una grua.

A més a més d'aquests àmbits podem trobar drons a més llocs com: control d'obres i topografia, transport, manipulació de materials nocius, etc.

Figura 4. 1. 1: Dron en l'àmbit militar

Figura 4. 1. 2: Dron en un incendi forestal.

Figura 4. 1. 3: Dron en l'agricultura.

Figura 4. 1. 4: Dron en una situació d'emergència.

L'evolució de la tecnologia i les xarxes socials han fet que els drons guanyin molta importància en l'àmbit recreatiu i també en publicitat. Gravar un vídeo o captar imatges aèries fins fa uns anys era impensable per molts per el cost elevat que això comportava, des de fa uns anys però podem trobar al mercat drons molt fàcils de controlar que ens permeten captar molt imatges i vídeos d'alta qualitat per preus molt més assequibles.

Figura 4. 1 .5: Dron d'ús recreatiu.


Figura 4. 1. 6: Dron de proves de repartiment propietat d'Amazon

## 4. 2. Estructura

Podem diferenciar els drons pel que fa a estructura en dos grans grups:

1. Ala fixa. Són els que tenen forma d'avió, els quals tenen la capacitat de planejar i ens donen l'avantatge d'un consum menor.


Figura 4. 2. 1: Dron d'ala fixa.

2. Ala rotatòria. Són els que consten d'una estructura en braços i un cert número d'hèlixs per poder volar, ens donen l'avantatge d'una gran manejabilitat, possibilitat d'estabilitzar-se i quedar-se quiets a un punt. Dins d'aquests grup classifiquem els drons segons el número de braços que tenen: tricòpter, quadcopter, etc.


Figura 4. 2. 2: Diferents models de drons d'ala rotatòria.

Hi ha totes mides de drons, des de drons que fan menys de 5cm fins a drons enormes capaços de transportar més de mitja tona de pes. Ara bé no hi ha cap classificació oficial de drons segons mida o pes a no ser que sigui en temes de legislació que veurem a continuació.

Si mirem els drons que trobem al mercat per ús recreatiu veurem que la gran majoria són drons d'ala rotatòria, més en concret quadcopters ja que són els que més estabilització ens permeten amb menor cost. És per això que aquest projecte es basarà en explicar quin és el funcionament d'un quadcopter, quins components utilitza i com realitzar el muntatge i programació del controlador.

## 4. 3. Funcionament bàsic d'un quadcopter

Tal i com hem explicat a l'apartat anterior un quadcopter és un dron format per quatre hèlixs, aquestes hèlixs s'han de trobar equidistants entre elles i rotant tal i com indica la Figura 4. 3. 1: dos en sentit horari i dos en sentit antihorari.


**Figura 4. 3. 1: Estructura i funcionament de les hèlixs d'un quadcopter**

Aquesta rotació oposada de les hèlixs es deu a la necessitat d'equilibrar les forces que realitzen ja que si totes giressin en el mateix sentit les forces es sumarien i el dron agafaria un moviment rotacional. És el mateix que passa quan un helicòpter perd la seva hèlix posterior: aquest comença a girar sense control per culpa d'una descompensació en les forces dels motors.
També hem de tenir en compte la forma de l'hèlix per saber en quin sentit ha de girar, a la Figura 4. 3. 2 podem observar com és una hèlix que gira en sentit horari i com és una que gira sentit antihorari, una és el reflex de l'altre.

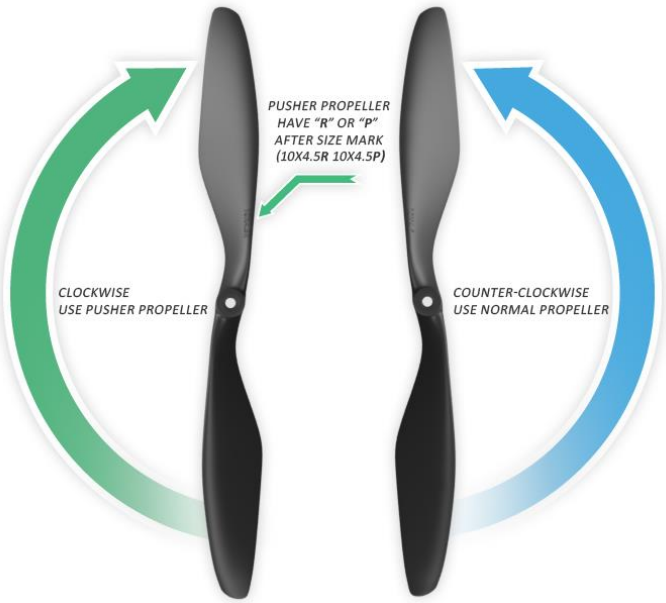



**Figura 4. 3. 2: Diferencia entre una hèlix que gira en sentit horari i una que gira en sentit antihorari.**

En els quadcopters podem trobar dos tipus de configuracions segons la seva orientació: la configuració en forma de "+" on la part davantera del quadcopter coincideix amb un dels braços o la configuració en forma de "x" on la part davantera es troba entre mig de dos braços, a la Figura 4. 3. 3 es pot veure un exemple de cada una de les dos configuracions.

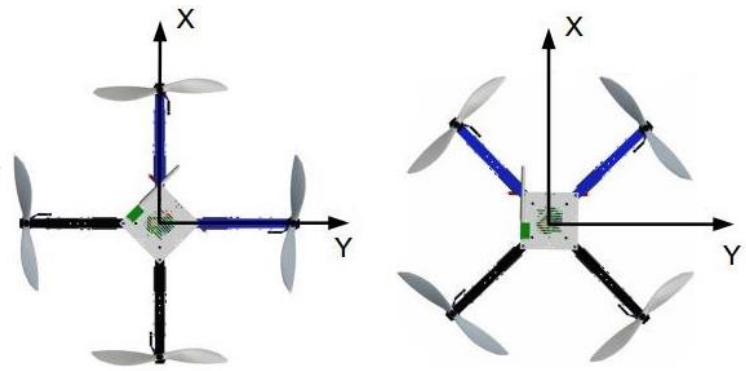



Figura 4. 3. 3: A l'esquerra quadcopter amb configuració "+", a la dreta quadcopter amb configuració "x".

La gran diferencia entre aquestes dos configuracions és que en la configuració "+" cada direcció (X, Y) només es veu afectada per dos hèlixs mentre que en la configuració "x" sempre intervenen les quatre hèlix. Això comporta que el controlador de la configuració "+" sigui més senzill però també farà que el quadcopter sigui menys estable.
La majoria de quadcopters del mercat funcionen amb la configuració "x" ja que com hem dit ofereix més estabilitat, és per això que el quadcopter tractat en aquest projecte també serà amb la configuració "x".

Un dron té quatre tipus de moviments:

1. Altitud. Elevació o descens en vertical.
2. Roll. Rotació respecte l'eix que va de la part davantera a la part posterior. S'utilitza per indicar si volem anar a la dreta o a l'esquerra sense canviar l'orientació del dron.
3. Pitch. Rotació respecte l'eix que va de la part esquerra a la part dreta. S'utilitza per indicar si volem anar endavant o endarrere.
4. Yaw. Rotació respecte l'eix vertical. S'utilitza per canviar l'orientació del dron.

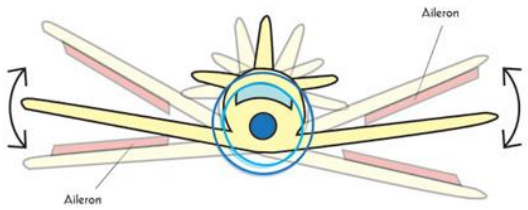



Figura 4. 3. 4: Moviment Roll

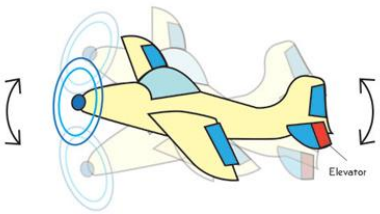



Figura 4. 3. 5: Moviment Pitch

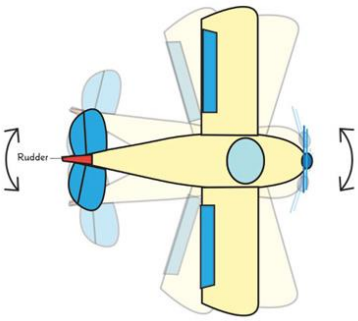



Figura 4. 3. 6: Moviment Yaw

Basant-nos en això a la Figura 4. 3. 7 podem veure quines hèlixs han d'agafar més i menys velocitat segons el moviment que volem fer:

A) Anar endarrere – Pitch backward.
B) Anar endavant – Pitch forward.
C) Moure'ns cap a la dreta (sense canviar la orientació del quadcopter) – Roll right.
D) Moure'ns cap a l'esquerra (sense canviar la orientació del quadcopter) – Roll left.
E) Canviar l'orientació cap a la dreta – Yaw right.
F) Canviar l'orientació cap a l'esquerra – Yaw left.
G) Elevar – Throttle up.
H) Descendir – Throttle down.

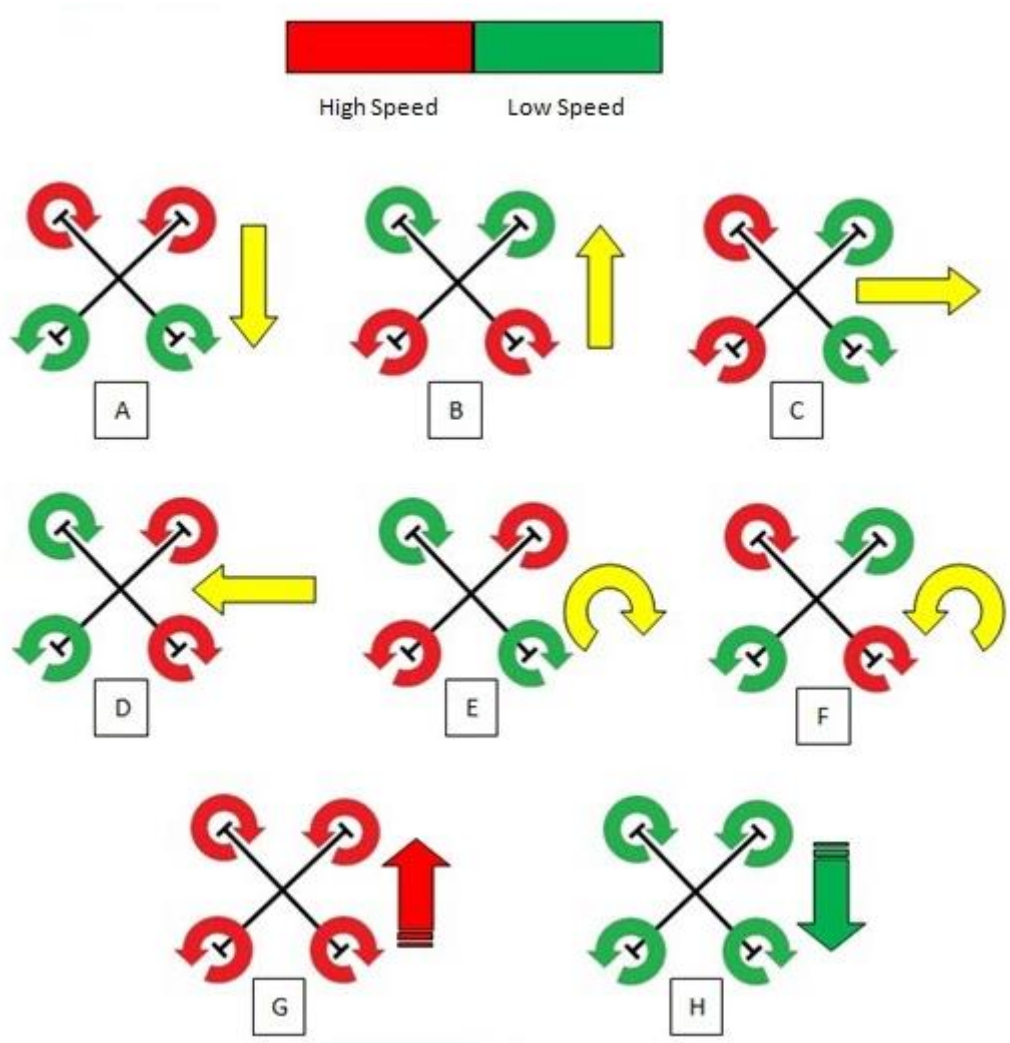



Figura 4. 3. 7: Hèlixs afectades segons el moviment.

Per controlar aquests moviments mitjançant el nostre comandament RC ho farem de la següent forma:

- Mitjançant el joystick esquerra:
  - Amunt significarà accelerar i avall reduir l'acceleració.
  - Cap als costats indicarem la rotació en yaw.
- Mitjançant el joystick dret:

- Amunt i avall serà la rotació en pitch (que el nostre dron vagi endavant o endarrere).
- Cap als costats indicarem la rotació en roll.

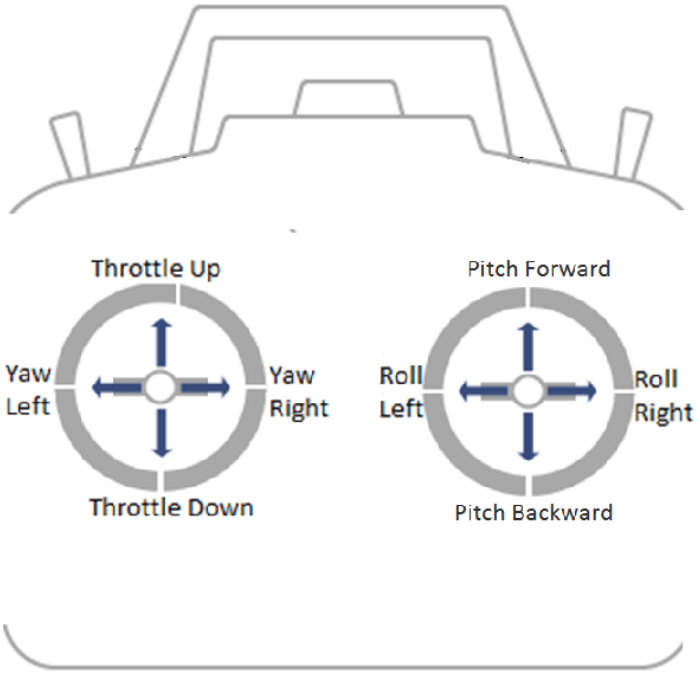



Figura 4. 3. 8: Moviments del comandament RC.

Per últim repassarem dos modes de vols molt utilitzats en el mon dels quadcopters:

- Mode acrobàtic. L'estabilitzador del quadcopter només s'encarrega de compensar la rotació no desitjada, no la inclinació.
- Mode estable. L'estabilitzador del quadcopter s'encarrega de fer tornar al quadcopter la posició inicial, és a dir, angle pitch i roll a 0.

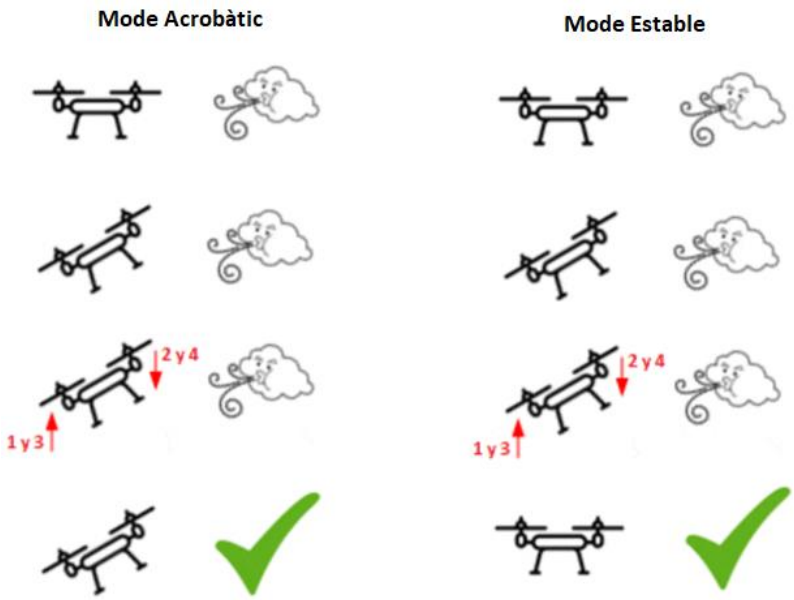



Figura 4. 3. 9: Compració entre el mode acrobàtic i el mode estable.

Tal i com podem observar, en el mode acrobàtic serà molt més difícil de mantenir "quiet" a l'aire un quadcopter ja que requereix totalment de l'habilitat del pilot.

# 4. 4. Legislació actual a Espanya sobre drons

L'organisme que s'encarrega de regular l'ús dels drons a Espanya és l'Agència Estatal de Seguretat Aérea (AESA) i l'última normativa establerta la regeix el Real Decret 1036/2017 del 15 de desembre que modifica el Real Decret 552/2014 del 27 de juny.

La normativa anterior no tenia en compte l'ús recreatiu dels drons i hi havia moltes restriccions com, per exemple, no estava permès el següent:

- Realitzar vols en plena ciutat i zones rurals.
- Vols nocturns.
- Vols de drons fora de l'abast visual de prototips amb un pes superior a 2Kg.
- Vols en espais aeris controlats i vols dirigits per visió augmentada.

En la nova regulació aquests camps han rebut llum verda en l'àmbit professional, sempre i quan s'ajustin als requisits i normes de seguretat establertes.

## Llicencia

Per volar drons de manera professional és obligatori:

- Tenir llicència de pilot de drons.
- Estar donat d'alta com operador de drons per l'AESA.
- Tenir una assegurança per drons que et cobrirà a tu com a pilot.
- Tenir un certificat mèdic en vigor que serà LAPL en cas d'aeronaus lleugeres o certificat mèdic de classe II per aeronaus amb un més major a 25Kg.

Ara bé, per ús recreatiu no és necessària cap llicencia ni assegurança, simplement complir la normativa. Ara bé si el dron que vols utilitzar supera els 25Kg de pes necessitaràs el descrit anteriorment. Encara que no sigui obligatori tenir una assegurança és molt recomanable per cobrir els danys que puguis causar.

## Condicions per l'ús recreatiu

Per utilitzar drons d'ús recreatiu s'han establert restriccions importants que s'han de complir:

- Distància de vol respecte els aeroports. S'han establert 8 km com a distància mínima.
- Espais aeris controlats. En aquestes àrees està totalment restringit el vol de drons per part d'aficionats.
- Altura i distància de vol. El vol amb dron no pot sobrepassar els 120 metres d'altura a nivell de terra o a nivell d'obstacle més alt que es trobi a un radi de 150 metres de l'aeronau.
- Vols dins l'abast visual. Els vols s'han de realitzar amb el límit d'abast visual de l'usuari.

- Vols en ciutat i sobre aglomeracions de persones. Anteriorment estaven prohibits els vols sobre aglomeracions de persones, igual que en àrees urbanes i rurals. No obstant aquest reglament ha canviat per els drons de menys de 250 grams. Actualment està permès realitzar vols a una altura no major als 20 metres i a una distància prudencial de les edificacions circumdants.
- Mai pots posar en perill la seguretat de persones o béns materials.
- Àrees restringides. A més a més dels espais aeris controlats hi ha altres zones restringides. L'AESA és l'agència encarregada de determinar les àrees de vol permeses o restringides. A la pàgina web https://icarusrpa.info podem trobar tota la informació sobres les restriccions de cada zona d'Espanya.
- En cas de difusió de imatges de persones o espais privats s'ha de complir la regulació de Protecció de Dades, Llei de Dret al Honor, a la intimitat personal i familiar i necessitaràs tenir una autorització sobre les persones.

## Identificació del dron

La nova llei exigeix, tant a pilots professionals com a aficionats, que l'aeronau i el control remot estiguin clarament identificats mitjançant una placa ignífuga identificativa on consti la següent informació:

- Nom del fabricant.
- Tipus de dron.
- Model de dron.
- Número de sèrie.
- Nom de l'operador i dades de contacte.

## Vols en zones urbanes i rurals a nivell professional

Els punts a tenir en compte abans de realitzar vols en aquestes zones són:

- El pes del dron està limitat a 10Kg.
- El vol s'ha de realitzar dins l'abast visual.
- Marge de seguretat. La zona que es vola ha d'estar acordonada per l'autoritat competent o s'ha de mantenir una distància de 50 metres sobre persones alienes.
- El dron ha de disposar d'un sistema d'amortidor de caiguda.
- Autorització d'operativitat aprovada per l'AESA.

## Vols nocturns a nivell professional

- Es requereix una autorització per part de AESA que avaluarà les condicions de la petició i el patró de vol que s'espera realitzar.
- Per la petició del permís s'ha d'incloure un estudi de seguretat específic en el qual s'indica la trajectòria ben planificada i segura.
- El dron ha de tenir colors vistosos i llums que ajudin a identificar la seva posició.
- L'aeronau no pot pesar més de 10 Kg.

## Vols en espais aeris controlats a nivell professional

- En el cas de drons amb un pes major a 25Kg han d'estar equipats amb un microxip en freqüència S.
- Es requereix una autorització per part de AESA sota un previ estudi de seguretat que descrigui la ruta de l'aeronau.
- El pilot ha d'estar acreditat com a radiofonista i parlar l'idioma en el que s'expressa tot l'equip de control.
- Només es permeten vols diürns.

## Vols fora de l'abast visual

- Els drons amb un pes superior als 2Kg han d'estar equipats amb un sistema de rastreig.
- Els vols s'han de fer en una àrea allunyada de persones i edificis.
- Per major seguretat el pilot de l'aeronau no ha d'excedir el límit segur de la freqüència que utilitza.
- En cas de tenir una càmera de visió ha d'estar orientada cap a davant.
- L'operador ha d'haver cursat els estudis de seguretat subjectes al NOTAM.
- Es requereix l'aprovació de AESA que s'encarregarà de certificar el recorregut i verificar l'estudi de seguretat.

## Recomanacions sobre l'ús de drons a nivell mundial

Si tens pensat utilitzar un dron, tan a nivell professional com recreatiu, a fora d'Espanya és molt important de que t'asseguris de conèixer totes les lleis que regulen l'ús d'aquests dispositius al país de destí.

També hem de tenir en compte que si volem viatjar amb el nostre dron ens podem trobar dos dificultats:

- Les bateries que porten els drons són considerades un article perillós dins els avions, és per això que normalment no estan permesos dins la bodega de l'avió i no es pot portar un dron a l'equipatge facturat a no ser que treguis la bateria i la portis com a equipatge de mà. Tot i així val més sempre mirar les normes de l'aerolínia amb la que voles per saber quina normativa tenen respecte els drons.
- Les normes d'entrada al país. Consulta amb el departament d'immigració si tenen alguna prohibició d'entrar drons al país, sinó tel podrien requisar a l'arribada a l'aeroport de destí.

# 5. Requisits del sistema

Els requisits que hem d'assolir per tal d'aconseguir el nostre objectiu en aquest projecte serien els següents:

- Aprendre el funcionament bàsic d'un dron i com es controla mitjançant un comandament RC.
- Explicar la legislació vigent sobre drons a Espanya.
- Aprendre quins són els components bàsics que formen un quadcopter.
- Fer un programa de test que rebi la senyal del comandament RC correctament.
- Llegir els angles sobre els eixos X i Y de la IMU mitjançant només el giroscopi.
- Llegir els angles sobre els eixos X i Y de la IMU mitjançant només l'acceleròmetre.
- Llegir els angles sobre els eixos X i Y de la IMU mitjançant el giroscopi i l'acceleròmetre a la vegada per tal de resoldre els problemes que puguem trobar en els dos punts anteriors.
- Calibrar i poder controlar mitjançant el comandament RC un motor.
- Realitzar un controlador PID senzill de només un eix de rotació per tal d'experimentar les seves diferents constants i veure el seu comportament.
- Llegir el voltatge de la bateria a través d'Arduino.
- Realitzar el muntatge d'un quadcopter.
- Ajuntar totes les proves sobre els components fetes anteriorment per tal de desenvolupar un controlador que ens permeti:
    - Controlar amb el comandament RC els moviments bàsics d'un quadcopter.
    - S'encarregui de que el quadcopter per tal que aquest voli en mode estable, és a dir que corregeixi la seva inclinació.

# 6. Estudis i decisions

Al següent apartat s'estudiarà quins components i quin programari necessitem per aconseguir el nostre objectiu: muntar i programar el nostre propi quadcopter per tal d'entendre com funciona cada component i el software que fa que aquest pugui volar de manera controlada i estable.

## 6. 1. Components bàsics d'un quadcopter

A l'apartat 4 hem pogut veure l'estructura i el funcionament d'un quadcopter. Tot seguit estudiarem quins són els components que necessitem i escollim per poder-lo muntar i programar.

### Frame

Anomenem "frame" a l'estructura del dron, podem trobar-ne de varies mides i formes diferents. Jo he optat per fer servir un frame amb unes dimensions de 450mm a les diagonals, és una mica gran però, al ser el primer dron que faig, prefereixo poder treballar amb una estructura gran i no tenir problemes al col·locar els components, a més a més ens serà més fàcil d'estabilitzar un quadcopter gran que un petit. Tot seguit, a la Figura 6. 1. 1 podem observar el frame que utilitzarem.

Figura 6. 1. 1: Frame F450.

### Comandament RC

Per tal de controlar el nostre quadcopter podem fer-ho de varies formes: fent un aplicació mòbil que es comuniqui amb el nostre controlador a través d'un mòdul bluetooth o Wi-Fi o a través d'un comandament radio control. Jo he optat per aquesta segona opció ja que el radio control ens permet controlar el dron a molta més distància que un mòdul Wi-Fi o bluetooth.

Necessitem un comandament RC i un receptor que ens permetin almenys enviar quatre canals de dades, un per cada tipus de moviment d'un quadcopter: elevació, roll, pitch i yaw.

El comandament i receptor que jo he decidit utilitzar és el que veiem a la Figura 4. 4. 2, disposa de 6 canals i utilitza una freqüència de 2,4 GHz. He escollit aquest ja que et permet configurar varis controladors, d'aquesta manera només comprant altres receptors pots utilitzar-lo per controlar més d'un aparell. Si només volguéssim el comandament per el quadcopter podríem buscar-ne un de més senzill i econòmic.

Figura 6. 1. 2: Comandament Flysky FS-T6 amb receptor Flysky FS-R6B de 6 canals.

## IMU

Una IMU (de les sigles en anglès Inertial Measurement Unit) és un dispositiu electrònic que mesura i informa sobre la velocitat, orientació i forces gravitacionals d'un aparell mitjançant la combinació d'acceleròmetres i giroscopis.

En la implementació entrarem en detall sobre com treballar amb una IMU, per ara podríem resumir que necessitem una IMU amb almenys 3 acceleròmetres i 3 giroscopis disposats de tal manera que formen 3 eixos ortogonals (x, y i z). Això comporta que serà una IMU de almenys 6 DOF (Degrees Of Freedom). Amb aquesta informació podrem veure la inclinació i velocitat angular del nostre quadcopter.

Per realitzar el quadcopter he optat per utilitzar la IMU MPU6050 ja que el seu preu és molt competitiu (recomanable comprar-ne més d'una per poder realitzar tot tipus de proves i tenir-ne de recanvi) i perquè és molt utilitzat en el món de l'electrònica, té molta fiabilitat i podem trobar molta documentació. A la Figura 6. 1. 3 podem observar com és aquest sensor.

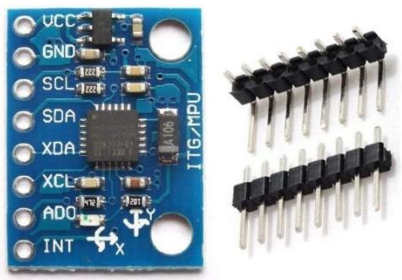



Figura 6. 1. 3: MPU6050

## Motors, ESC i Hèlixs

Els motors que s'utilitzen en el món dels drons són els anomenats "Brushless Motors" o, en català, motors sense escombretes. La principal avantatge d'aquests motors sobre els motors amb escombretes és que el desgast que tenen és menor, tenen major eficiència i més rangs de velocitats.

Per poder utilitzar aquest tipus de motors és necessari que cadascun vagi connectat a un ESC (Electronic Speed Control). Un ESC és un controlador de velocitat electrònic, un sistema capaç de definir la velocitat de gir d'un motor mitjançant la generació de polsos compatibles amb aquest tipus de motors.

Quan estiguem escollint quin motor podem utilitzar per el nostre dron ens fixarem que tots els motors tenen un valor indicat com a KV, aquest valor correspon al número de revolucions per minut (rpm) que serà capaç d'oferir-nos el motor quan se li apliqui 1 volt de tensió. Per exemple, si un motor indica que té 2300KV i rep una tensió de 11 volts significa que el número de revolucions per minut del motor serà de 26.640 mentre que si un motor indica que té 1000KV el número de revolucions per minut quan li apliquem la mateixa tensió serà de 11.000. Aquestes dades són teòriques ja que els fabricants realitzen aquestes proves sense cap tipus de pes sobre el motor, en el moment de col·locar un motor a un dron intervenen molts més factors com la bateria, els ESC utilitzats, el tipus d'hèlix, el pes del dron, etc.
Un cop explicat que significa el valor KV, quin valor hem d'escollir pel nostre dron? A grans trets es recomana el següent:

- KV baix. Recomanat per drons que necessiten més força i menys velocitat. Indicat per moure hèlixs de grans dimensions. Són motors menys vius, més suaus i amb un consum elèctric menor.
- KV alt. Recomanat per drons que necessitin menys força i més velocitat. Indicat per moure hèlixs de petites dimensions. Són motors més vius, amb reaccions violentes i un consum elèctric major.

Una cosa semblant als KV dels motors passa amb les hèlixs, hi ha molts tipus i mides de hèlixs diferents però podríem resumir-ho de la següent forma:

- Hèlix gran. Aporta més força, menys velocitat i més consum elèctric. Això comporta que la velocitat de reacció al accelerar o reduir la velocitat és més lent.
- Hèlix petita. Aporta menys força, més velocitat i menys consum elèctric. Això comporta que la velocitat de reacció al accelerar o reduir la velocitat és més ràpid (ens permet fer moviments més bruscs).

Tenint en compte les recomanacions anteriors i que volem construir un dron de iniciació (més gran i que sigui fàcil de manejar) he optat per escollir un motor amb un KV baix i unes hèlixs grans.
Un cop escollits uns motors i hèlixs hem de triar els ESC que s'adapten millor. A les especificacions d'un ESC s'indica quina capacitat de corrent és capaç de proporcionar de forma continuada i quina en un període màxim de 10 segons. Aquest valor l'hem de comparar amb la informació sobre la corrent que necessita el motor escollit (tenint també en compte l'hèlix utilitzada ja que fa variar molt el consum d'aquest).

Tal i com podem veure escollir motors, hèlixs i ESC no és una feina fàcil, i menys per gent que vol entrar dins el món dels drons per primer cop. És per això que a diferents plataformes de venda ja podem trobar conjunts de motors, hèlix i ESC compatibles entre ells i que ens poden servir per fer el nostre primer dron. Jo finalment m'he decantat per el paquet que es pot veure a la Figura 6. 1. 4 on hi ha inclòs:

- Motor A2212 1000KV.
- ESC 30A.
- 2 hèlixs (una CW i una CCW) de tipus 1045.

He escollit aquest paquet ja que els la mida dels motors s'ajusta sense problemes al frame i perquè tenen un KV baix, això ens aportarà més força (el quadcopter és gros), menys velocitat i que el quadcopter sigui menys brusc.


Figura 6. 1. 4: Motor, ESC i hèlix.

## Bateria

Les bateries que s'utilitzen en els drons són les de polímers de Liti, conegudes com a LiPo. Aquestes bateries tenen una alta densitat d'energia, alta velocitat de descàrrega i un pes lleuger. En el moment d'escollir una bateria ens hem de fixar en les seves tres especificacions: el seu voltatge, la seva capacitat expressada en miliamperis hora (mAh) i la seva capacitat de descàrrega.

- Voltatge. Les bateries LiPo estan composades de cel·les individuals connectades en sèrie. Cada cel·la té un voltatge nominal de 3.7 V, per tant per saber quin voltatge té

una bateria hem de saber quantes cel·les té, normalment indicat amb un número seguit d'una "S". Exemples: 1S = 1 cel·la = 3.7 V, 2S = 2 cel·les = 7.4 V.
Cal mencionar que el voltatge d'una cel·la sempre hauria d'estar entre els 3V i els 4.2V, si una bateria es descarrega a menys de 3V podria causar un dany irreparable, és per això que no es recomana descarregar les LiPo per sota dels 3.5V.

- Capacitat. Indica quanta corrent pots extreure de la bateria en una hora fins que es buida. Exemple: una bateria de 1300 mAh es descarregaria totalment en una hora si s'extreu 1.3 A d'ella. Si s'extragués 2.6 A el temps es reduiria a la meitat.
- Capacitat de descàrrega. També conegut com a "C Rating" ens indica quina és la descàrrega màxima que es pot fer de la bateria en forma continua de manera segura. Exemple: si tenim una bateria de 1300 mAh i un C Rating de 50C ens indicarà que la descàrrega màxima continua és de 65A.

Vist els tres paràmetres que afecten una bateria LiPo podem buscar quina escollim tenint en compte el següent:

- Necessitem que el voltatge sigui l'indicat pel fabricant dels motors, en el nostre cas ens indica que ha de ser una bateria de 2 a 4 cel·les.
- Els nostres motors funcionen a màxim de 10 Ampers, com que tindrem quatre motors necessitem que la descàrrega màxima continua sigui almenys de 40A.

Tenint en compte els anteriors punts he optat per una bateria de 3 cel·les (3S, 11.1V), 3000mAh i 30C. D'aquesta manera tenim una descàrrega màxima continua de 90A, molt per sobre del que necessitem i també tenim una capacitat total alta, per tal de incrementar l'estona de vol.


Figura 6. 1. 5: Bateria LiPo de 3S, 3000mAh i 30C.

## Controlador

Ja per acabar els components bàsics d'un quadcopter ens trobem amb la necessitat de tenir un controlador que agafi la informació de la nostra IMU i comandament RC i transmeti les odres als ESC i motors. Al mercat podem trobar molts controladors de vol ja programats que només s'han de instal·lar i ajustar de manera simple, ara bé, l'objectiu d'aquest projecte també és aprendre com funciona un controlador d'un quadcopter, és per això que necessitarem escollir un microcontrolador per tal de programar el nostre quadcopter.

Hi ha molts microcontroladors i plataformes disponibles però jo he escollit **Arduino** per varies raons:

- Familiaritat. L'he tractat anteriorment a alguna assignatura del Grau en Enginyeria Informàtica.
- Assequible. El seu preu és molt econòmic.
- Multiplataforma.
- Open Source.
- Té una gran comunitat a darrera on trobar informació.

El model Arduino escollit és l'Arduino Uno Rev 3 que podem veure a la Figura 6. 1. 6 ja que té totes les característiques necessàries per poder desenvolupar el nostre quadcopter.

- Comunicació mitjançant un bus en sèrie anomenat $I^2C$ que ens servirà per comunicar-nos amb la IMU.
- Almenys 8 ports digitals d'entrada / sortida per poder comunicar-nos amb els quatre canals del receptor del comandament RC i amb els quatre ESC.
- Almenys 1 port analògic d'entrada per poder mesurar el nivell de bateria disponible.
- Pins d'alimentació per a perifèrics a 5V i 3,3V.


Figura 6. 1. 6: Arduino UNO Rev 3.

# 6. 2. Altres components i eines

Ja tenim els components bàsics que formaran el nostre quadcopter, tot i així hi ha altres components i eines que ens poden ser útils i ens ajudaran en el procés de desenvolupament. Tot seguit en deixo uns quants indicant si són imprescindibles, recomanables o opcionals.

## Balancejador d'Hèlix (Recomanable)

Les dos "ales" d'una hèlix acabada de comprar difícilment pesaran el mateix, això provoca vibracions que poden afectar al correcte funcionament del nostre quadcopter. Amb l'aparell que podem veure a continuació podem veure quina "ala" pesa més per tal d'arreglar aquesta descompensació (més endavant s'explica com utilitzar-lo).

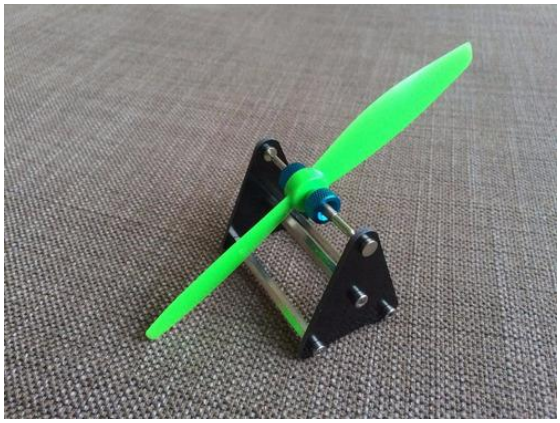


**Figura 7. 2. 8: Balancejador d'hèlixs amb una hèlix descompensada.**

## Carregador de bateries LiPo (Imprescindible)

Les bateries LiPo requereixen una càrrega i un manteniment especial per tal que rendeixin al màxim i ofereixin una bona vida útil. Carregar les bateries LiPo de forma incorrecte podria provocar un incendi, és per això que es imprescindible tenir un carregador específic per bateries LiPo com el que es pot observar a la Figura 6. 2. 2 i llegir atentament les instruccions.


**Figura 6. 2. 2: Carregador bateries LiPo.**

## Placa de distribució de potència (Opcional)

Aquesta placa que podem observar a la Figura 6. 2. 3 la utilitzarem per soldar els quatre ESC a la bateria. Aquesta connexió es podria fer simplement utilitzant cable però és més còmode i segur fer-ho utilitzant una placa com aquesta.

En el meu cas finalment no l'he necessitat ja que el frame adquirit ja incorporava una placa de distribució de potència.

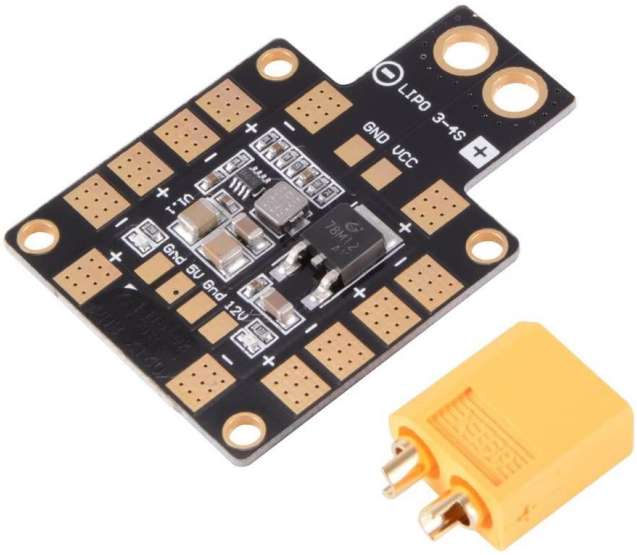



Figura 6. 2. 3: Placa de distribució de potència.

## Placa de muntatge (Opcional)

La podem utilitzar per soldar-hi els diferents components del dron de forma còmode i senzilla.

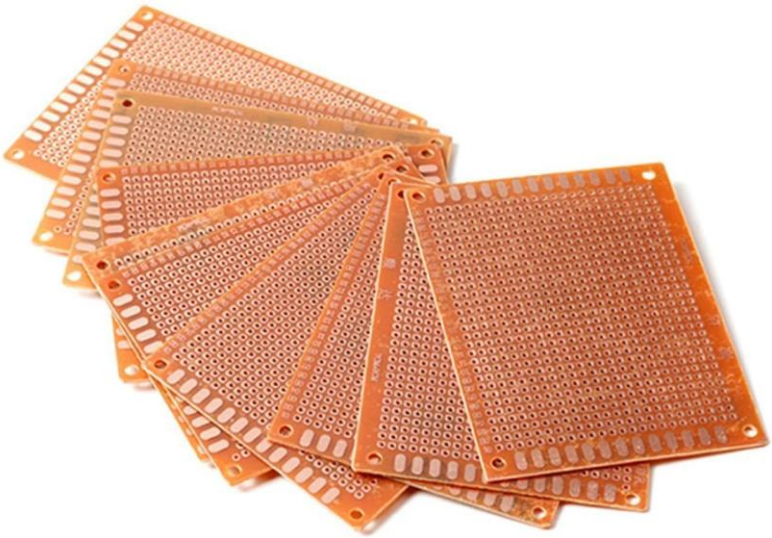


Figura 6. 2. 1: Plaques de muntatge.

## LCD 16x2 (Opcional)

L'utilitzarem per visualitzar variables durant el procés de desenvolupament. En cas d'utilitzar-lo assegurar-se que inclou l'adaptador $I^2C$ per facilitar la connexió.

Figura 6. 2. 4: LCD 16x2 i adaptador $I^2C$.

## Soldador d'estany (Imprescindible)

Per tal de poder soldar els components de forma robusta. També ens serà molt útil disposar de varies puntes, un desoldador i varies mides d'estany.

Figura 6. 2. 5: Kit de soldadura amb estany.

## Multímetre (Imprescindible)

Molt útil per poder mesurar la tensió i impedàncies. Ens ajudarà a comprovar que els components estiguin soldats correctament i que arriba la tensió adequada per el seu funcionament.


Figura 6. 2. 6: Multímetre.

## Oscil·loscopi (Recomanable)

Ens serà de gran ajuda per trobar errors, poder veure les senyals del comandament RC, les senyals enviades als ESC, comprovar quin temps ens tarda cada cicle del nostre programa, etc. No cal tenir un oscil·loscopi professional ja que tenen un preu elevat. Jo utilitzo el que es pot visualitzar a la Figura 6. 2. 7 que té un preu de 30€.


Figura 6. 2. 7: Oscil·loscopi digital ETEPON.

## Varis components electrònics (Recomanable)

Ens serà molt útil disposar de varis components electrònics com:

- Leds. Per tal de visualitzar informació durant el vol.
- Connectors de bala. Per connectar els ESC amb els motors, d'aquesta manera si hem de canviar un motor o girar algun cable no hem de soldar-lo cada vegada.
- Connector bloc terminal amb cargol. Ens seran molt útils per connectar les entrades / sortides de Arduino amb els diferents perifèrics (IMU, ESCs, leds, etc).
- Crimpadora i terminals. Molt útils per utilitzar juntament amb els connectors anteriors.
- Interruptor de 20 A. Per encendre i apagar el dron de forma segura i controlada.
- Kit de resistències. L'utilitzarem per fer un divisor de tensió i mesurar la bateria.

Figura 6. 2. 9: Leds.

Figura 6. 2. 10: Connectors de bala.

Figura 6. 2. 11: Connectors de Bloc Terminal amb cargol.


Figura 6. 2. 12: Crimpadora i terminals.

Figura 6. 2. 13: Interruptor 20 A.


Figura 6. 2. 14: Kit de resistències.

# 7. Anàlisi i disseny del sistema

## 7. 1. Muntatge del quadcopter

Per tal de muntar els diferents components de forma correcte és molt important que mirem les seves especificacions als seus respectius manuals d'usuari o "Datasheets", aquests documents els podem trobar a la web del fabricant o on hem comprat el component.

Tot seguit repassarem els components del nostre dron i analitzarem com els hem de connectar al nostre controlador per tal d'aconseguir dissenyar l'esquema complert de connexions del nostre quadcopter.

### Receptor RC

Si accedim a la web del fabricant del nostre receptor podem veure que ens indica que necessita una connexió de 4.0 a 6.5 V. Disposa de 6 canals, dels quals en farem servir 4 per l'accelerador, roll, pitch i yaw. Tot i que no disposem d'un manual d'usuari és molt senzill trobar per internet com funcionen els receptors RC i quin és el seu esquema de connexió. Tot seguit, a la Figura 7. 1. 1, podrem observar l'esquema d'un receptor similar al nostre.

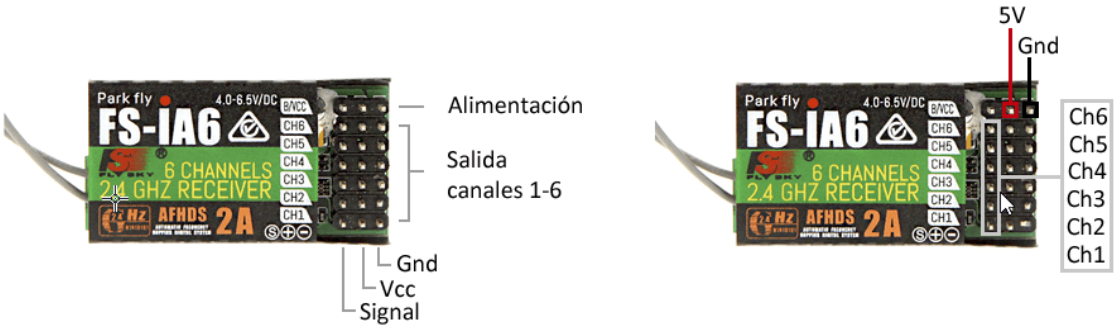



Figura 7. 1. 1: Esquema de connexió d'un receptor RC.

Com hem pogut observar simplement hem de connectar una tensió de 5V i el cable de terra als pins indicats a l'esquema. Llavors connectarem els canals que necessitem a pins de la nostra Arduino, per saber a quins hem de conèixer com és la senyal que ens arribarà.
A la pàgina del fabricant ens indica que cada canal transmet una senyal de tipus PWM (per les seves sigles en anglès Pulse Width Modulation). Aquesta senyal es tracta d'una ona quadrada que no sempre té la mateixa relació entre el temps que està alta i que està baixa, a la Figura 7. 1. 2 podem observar com són aquestes senyals.

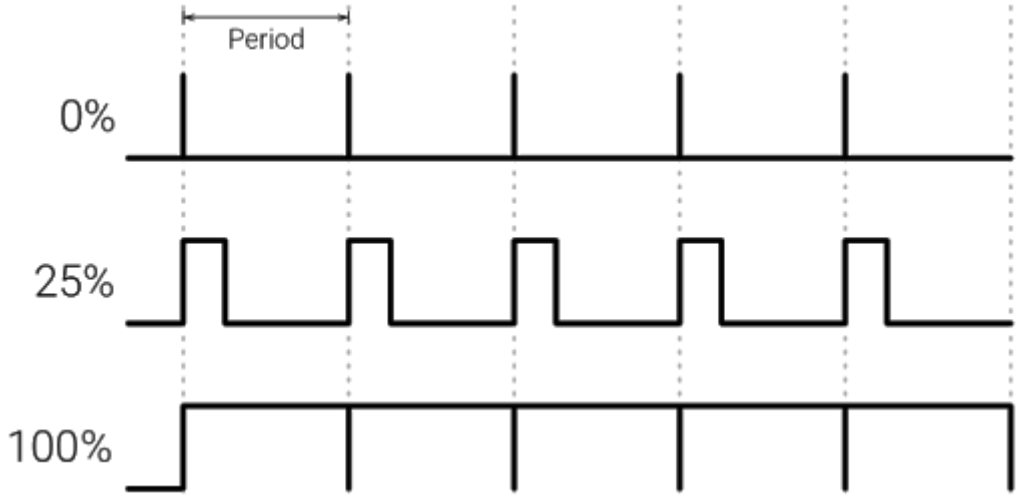



Figura 7. 1. 2: Senyals PWM.

Veient com són les senyals que rebrem del receptor i hem de llegir amb Arduino optarem per connectar el receptor a ports digitals d'entrada ja que el que ens interessa és saber l'amplitud d'aquesta senyal i no a quin valor està. A la següent figura podem veure l'esquema de connexió.

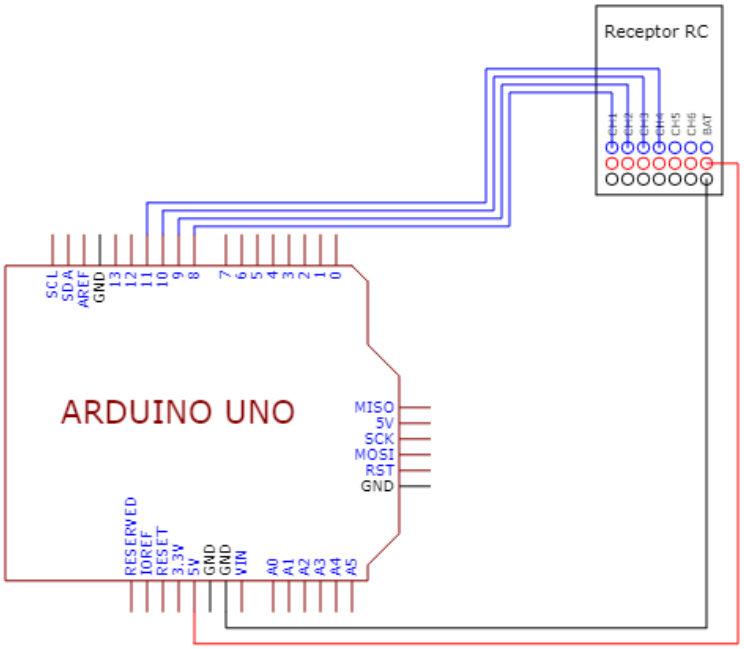



Figura 7. 1. 3: Esquema de connexió entre Arduino i el receptor RC.

## IMU

Comprovant el "datasheet" del nostre sensor MPU6050 i consultant diferents webs trobem fàcilment exemples de com utilitzar aquest sensor amb Arduino, hi ha molta informació sobre aquest sensor ja que és molt utilitzat.

La connexió necessària és la que es pot apreciar a la Figura 7. 1. 4, connectar una tensió d'entrada de 5V al pin VCC del sensor, el cable de terra i, finalment, connectar els pins del bus $I^2C$ dels que disposen tant la nostra Arduino com el sensor per comunicar-se.

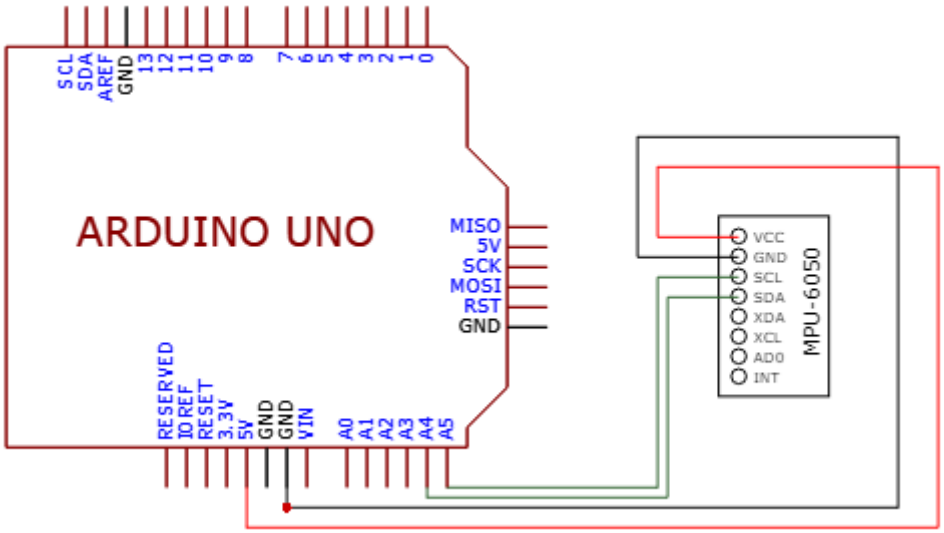



Figura 7. 1. 4: Esquema de connexió entre Arduino i el sensor MPU6050.

## Motors i ESC

Els motors Brushless i els ESC funcionen pràcticament de la mateixa forma tots, encara que siguin diferents models de diferents marques. És per això que buscant informació de seguida trobes exemples de com realitzar la connexió.

Primer de tot és important que sapiguem el funcionament bàsic d'un ESC, un ESC es controla mitjançant una senyal PWM amb una amplitud mínima de 1ms i una amplitud màxima de 2ms. Quan hem explicat la connexió del receptor RC ja em vist que per llegir senyals PWM utilitzarem ports digitals d'entrada, per tant per generar una senyal PWM utilitzarem ports digitals de sortida.

Un cop vist el funcionament anem a realitzar la connexió. Podem apreciar que els ESC que hem escollit tenen per un costat tres cables gruixuts de color blau i per l'altre dos cables gruixuts (un vermell i un negre) i tres cables prims junts. Doncs bé, realitzarem les connexions següents:

- Els tres cables blaus es connecten amb els tres cables que té el motor.
- Els cables vermell i negre de l'altre costat van connectats directament a la bateria, el vermell al positiu i el negre al negatiu (en el nostre cas a la placa de distribució de potència).
- Dels tres cables prims necessitarem connectar el negre amb el terra de la nostra Arduino, el vermell no el connectarem ja que és una sortida de tensió de 5V que no ens fa falta per alimentar cap component i el cable restant anirà connectat a un port digital de la nostra Arduino per poder enviar al ESC la senyal PWM.

Si ens trobem en el cas que un motor gira en el sentit contrari al que volem simplement hem de intercanviar dos dels tres cables que connecten l'ESC amb el motor. Podem veure-ho a la següent figura.

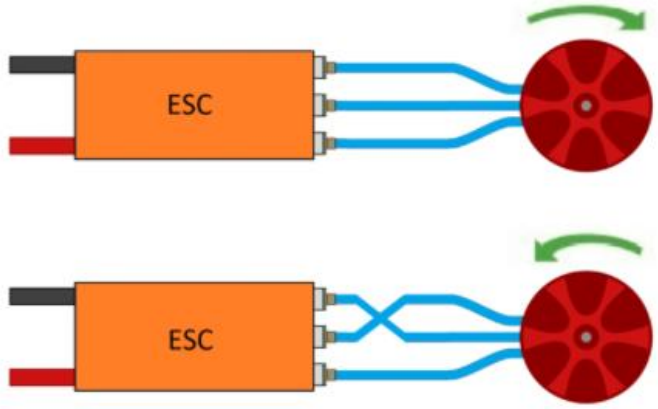



Figura 7. 1. 5: Connexió per canviar el sentit de gir d'un motor.

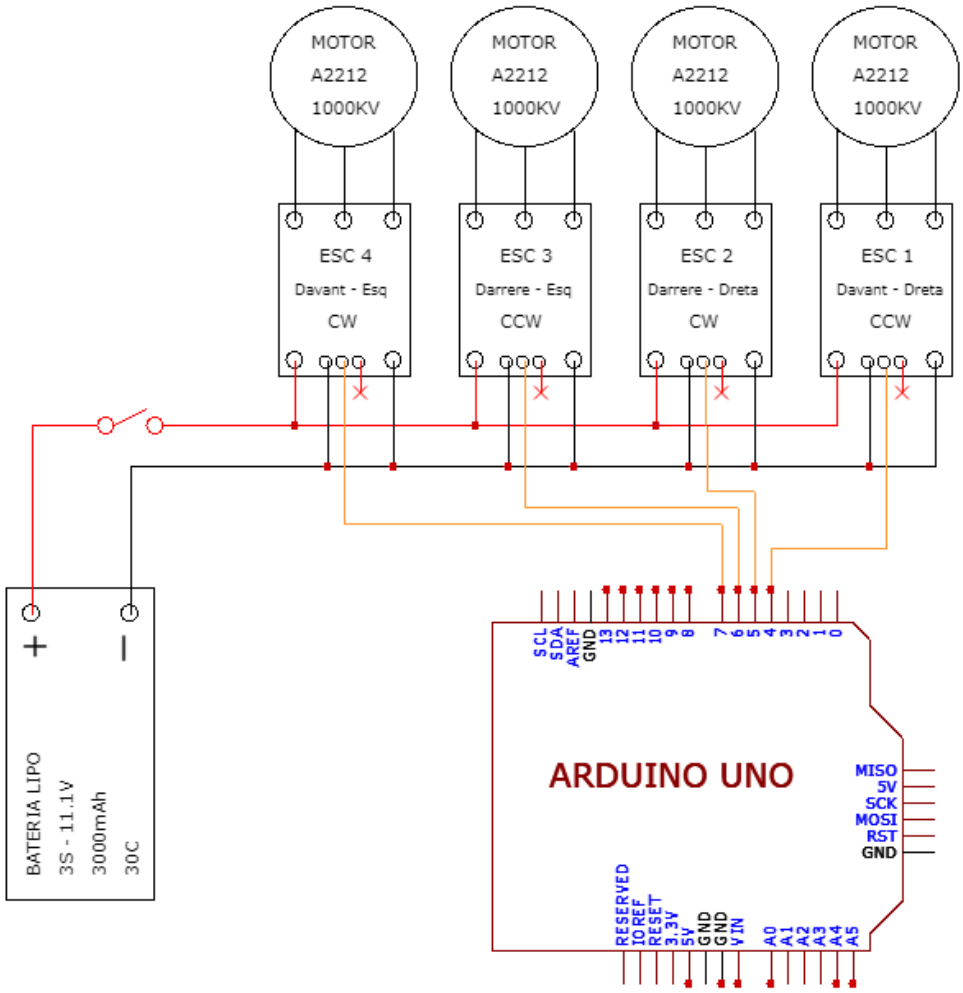



Figura 7. 1. 6: Esquema de connexió dels motors i ESC

Quan muntem l'hèlix a un motor hem de comprovar que estigui ben balancejada, és a dir, que les dos "ales" de l'hèlix pesin el mateix, sinó això provocarà moltes vibracions al nostre quadcopter. Per balancejar les hèlixs hi ha dos formes:

1. Fer un petit programa amb Arduino que llegeixi l'acceleròmetre i amb això comprovem les vibracions que ens arriben.
2. Comprar l'aparell que hem vist a l'apartat 6. 2 per veure quina ala pesa més que l'altre.

En els dos casos agafarem l'hèlix, afegirem cinta de celo a una ala (per augmentar el pes) o llimarem amb paper de vidre (per disminuir el pes), i tornarem a fer la prova fins que les vibracions siguin similars a les que ens dona l'acceleròmetre quan no hi ha l'hèlix muntada o fins que l'hèlix quedi estable al balancejador.
Jo personalment he optat per la segona opció tot i que poder la primera seria millor ja que també comproves que l'hèlix tingui poques vibracions un cop muntada (les vibracions poden venir per com queda encaixada a l'eix del motor).

## Bateria

Tal i com hem dit anteriorment utilitzarem la bateria per alimentar els ESC però també per alimentar l'Arduino. Si mirem la documentació que podem trobar a la pàgina web veiem que podem proporcionar una tensió de 9-12V a la nostra Arduino a través del pin **Vin**, la nostra bateria és de 3S per tant té un voltatge nominal total de 11.1V, per tant podem connectar-la directament a aquest pin.
Buscant més informació hem trobat que el pin Vin també alliberà tensió de sortida, com que no volem que ens afecti necessitarem col·locar un díode (model 1N4001 o similar en farem prou) que no deixi sortir aquesta tensió.

També ens interessarà monitoritzar quanta bateria queda per tal de mostrar un indicador i parar el dron abans que pugui acabar la bateria, la faci malbé i hi hagi perill de caiguda. Per fer-ho podem utilitzar els pins analògics d'Arduino, si observem la documentació veurem que aquests pins funcionen a un rang entre 0-5V i converteixen aquest valor a un número entre 0 i 1023, per tant si arriben 3V d'entrada a un pin analògic aquest donarà un valor de 613.
Tal i com hem vist el voltatge màxim en el que treballen aquests pins són 5V, la nostra bateria té un voltatge superior per el que necessitarem un divisor de tensió .

Per fer un divisor de tensió tindrem en compte de fer servir uns valors de resistència alts, d'aquesta manera evitarem un consum excessiu. L'esquema de connexió entre la bateria i Arduino quedarà de la següent forma.

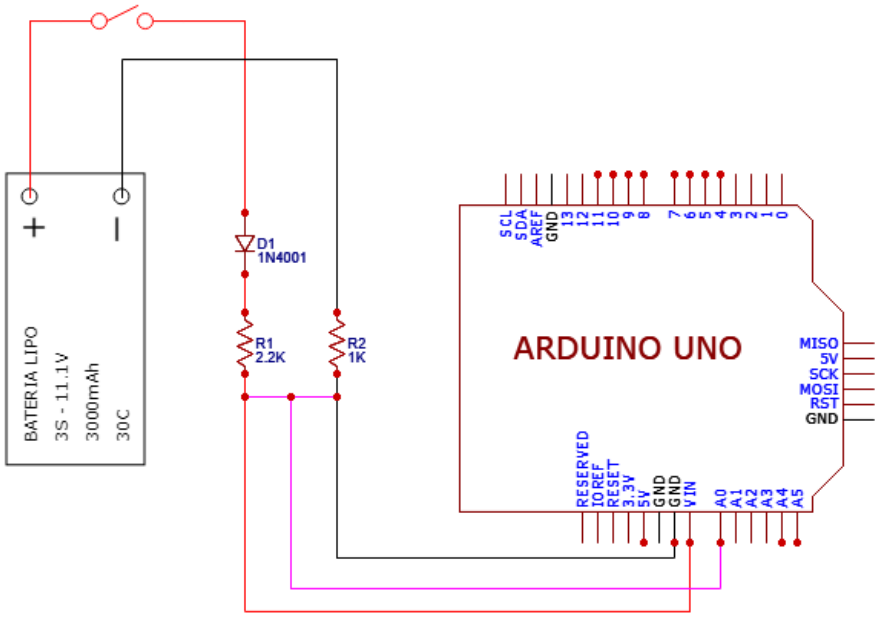



Figura 7. 1. 7: Esquema de connexió entre Arduino i la batería.

Podem calcular que amb les resistències utilitzades són suficients per obtenir un voltatge d'entrada al pin A0 d'Arduino inferior a 5V.

1. Aplicant la llei de Ohm trobem que la intensitat del circuit és 0.00346A (11.1 / 3200).
2. El voltatge de la primera resistència serà: 7.63125V.
3. El voltatge de l'entrada A0 serà 3.47V (voltatge total – voltatge resistència).

D'aquesta manera hem pogut comprovar que el nostre divisor de tensió és suficient i no la tensió que rep el pin A0 no és massa gran.

## Quadcopter

Un cop hem vist els diferents apartats per separat ja estem apunt per dissenyar l'esquema de connexions del nostre quadcopter. L'esquema de connexions generales mostra a la Figura 7. 1. 8 on també es pot veure que hem afegit dos Leds que ens serviran per mostrar si hi ha algun problema en el nostre programa (com per exemple que la bateria s'estigui acabant).

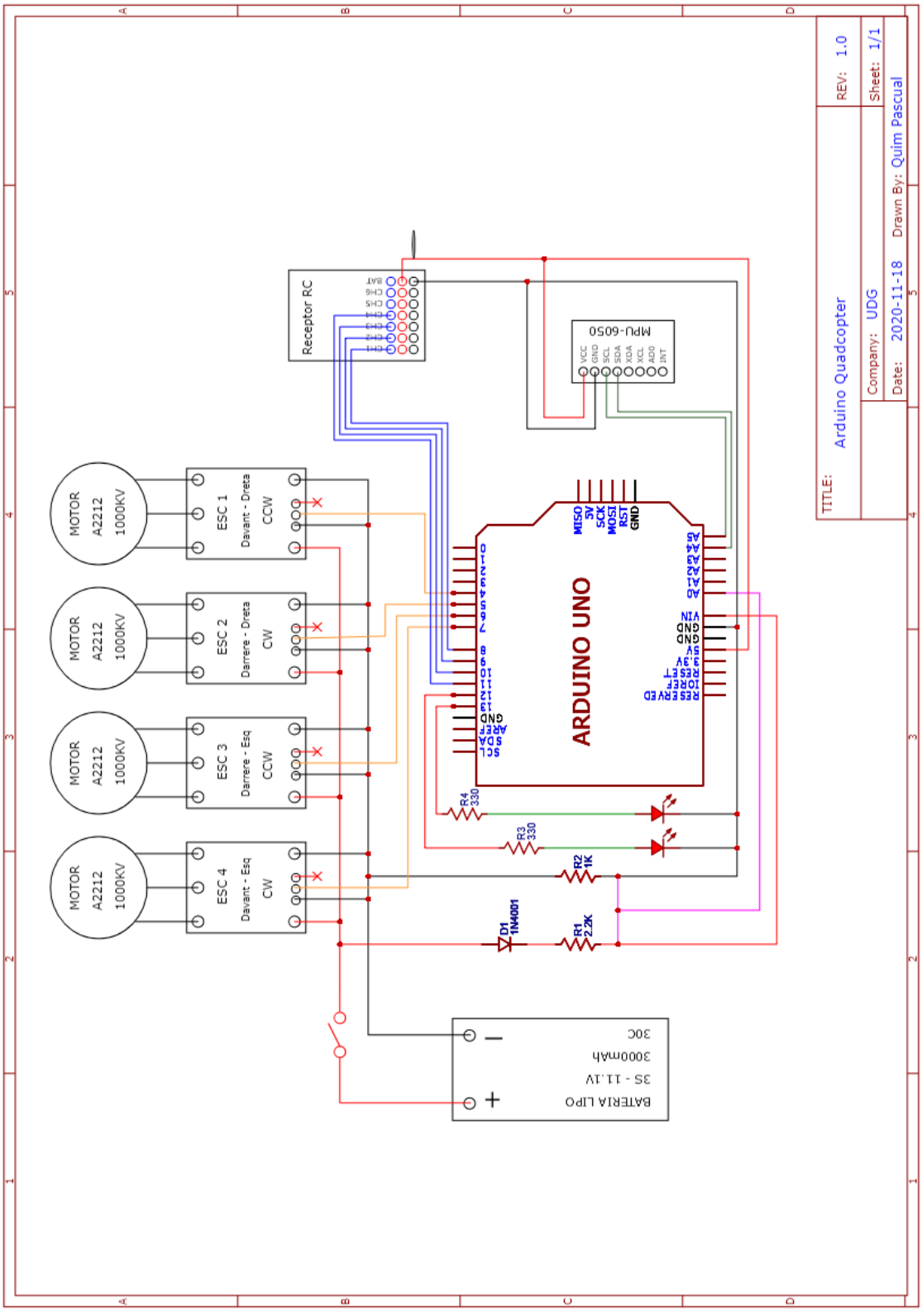



Figura 7. 1. 8: Esquema de connexió del nostre quadcopter.

# 7. 2. Desenvolupament del controlador

El nostre objectiu és programar un quadcopter que el puguem controlar mitjançant un comandament RC però també hem de tenir en compte que no tots els motors faran la mateixa força quan girin a la mateixa velocitat ni el pes del quadcopter estarà repartit perfectament, això juntament amb la fricció de l'aire provocarà que el dron tingui tendència a inclinar-se, desestabilitzar-se i caure.
També hem vist anteriorment que les revolucions per minut d'un motor es veuen afectades per el voltatge que reben, això implica que si la bateria està totalment carregada els motors giraran molt més ràpid que quan la bateria hagi baixat.
Per tant el nostre controlador no només ha de donar les ordres als motors del dron per realitzar el moviment que li indiquem a través del comandament sinó també per corregir la desestabilització i compensar la caiguda de tensió de la bateria.

Per dissenyar el nostre controlador primer hem de fer un llistat amb les funcionalitats que s'han de realitzar:

1. Configuracions: Inicialització de variables, comprovacions inicials per tal de poder arrancar (nivell de bateria, recepció senyal RC, recepció dades IMU), configuració de components.
2. Llegir la informació que ens envia el comandament RC per els quatre moviments del quadcopter.
3. Llegir la informació de la IMU.
4. Calcular les correccions per el roll, pitch i yaw.
5. Calcular el pols per cada ESC.
6. Llegir el nivell de la bateria i compensar els polsos per la caiguda de tensió.
7. Enviar els polsos calculats als ESC.

Tots aquests passos menys el primer els hem de realitzar en bucle i cada vegada, ara bé, quina freqüència ha de tenir aquest bucle?
La freqüència del nostre programa depèn dels ESC ja que realitzem tots aquests passos amb la finalitat d'enviar un pols a cadascun dels quatre ESC, no servirà de res que el nostre programa s'executi a 1KHz si els ESC no tenen una freqüència de refresc tan ràpida. La gran majoria de ESC poden treballar amb freqüències des de 50Hz fins a 400Hz tot i que per estar-ne segurs hem de mirar la documentació del fabricant. Una freqüència de 50Hz comporta que el nostre programa ha de tenir una duració de 20ms mentre que si la freqüència és de 400Hz la duració seria de 2.5ms. Buscant informació per internet sobre quina és la millor freqüència per un ESC d'un podem trobar que 50Hz no és gens recomanable ja que els motors s'actualitzarien massa lentament i seria molt difícil aconseguir l'estabilització del dron, la de 400Hz seria massa ràpida ja que hem de tenir en compte que el pols màxim que podem enviar a un ESC és de 2ms, això ens deixaria només 0.5ms per realitzar tots els càlculs. Per aquests motius jo he optat per una freqüència de 250Hz, d'aquesta manera el nostre programa ha de durar un total de 4ms, si enviem el pols màxim a un ESC encara tindrem 2ms extres per realitzar tots els càlculs.

Un dels passos que ens pot comportar més temps és el primer, llegir la informació que ens envia el comandament RC per cada un dels quatre moviments d'un dron. Hem vist anteriorment que un comandament RC envia polsos PWM, aquests polsos funcionen semblant als ESC, el seu valor mínim és de 1ms i el valor màxim és de 2ms. Tenint en compte que hem de llegir quatre moviments cada vegada, si tots els moviment estiguessin en el seu valor màxim tindríem que necessitem almenys 8ms en total per llegir els polsos. Tal i com ens podem imaginar això és un temps molt elevat i faria reduir massa la freqüència del nostre programa, hem de buscar una alternativa.
El microcontrolador de Arduino disposa de interrupcions, una interrupció és un mecanisme que ens permet associar una funció a la ocurrència d'un esdeveniment determinat. Quan es produeix l'esdeveniment el processador surt immediatament del flux normal del programa i executa la funció associada. A la pàgina web d'Arduino explica quins tipus de interrupcions hi ha i a quins pins ho podem assignar. A més a més de les interrupcions "normals" el nostre microcontrolador també té un tipus de interrupció interna anomenada PCINT (Pin Change INTerrupt) que ens permet disparar una interrupció cada cop que canviï el valor de qualsevol dels nostres pins (podem configurar quins poden disparar la interrupció i quins no). D'aquesta manera només hem de mesurar els temps entre que el pin agafa valor 1 i que torna a valor 0. Al treballar amb interrupcions no farà falta que incloem aquesta tasca dins el nostre programa principal.

A la figura següent veurem quin és l'esquema del nostre controlador.

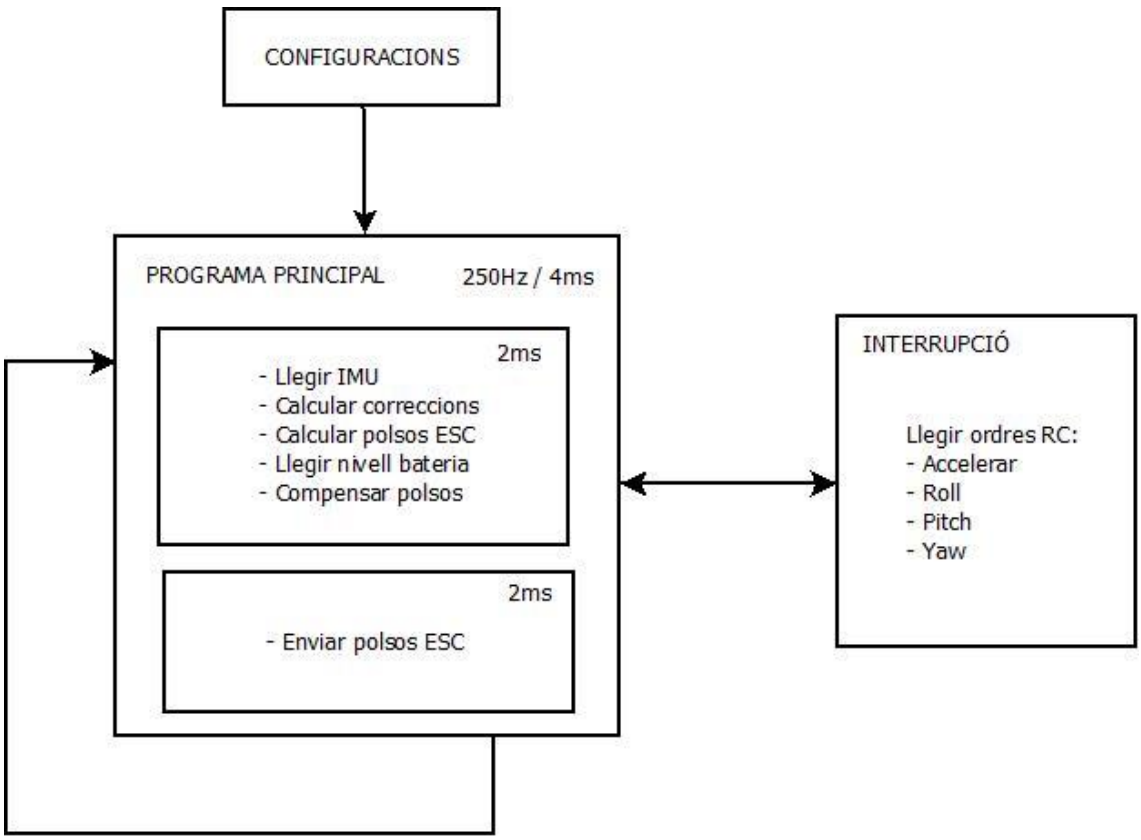



**Figura 7. 2. 1: Esquema del controlador.**

Tal i com hem vist el temps serà un factor molt important a tenir en compte a l'hora de programar el nostre controlador, haurem de optimitzar tan com puguem el nostre codi per tal de ser eficients.

Un aspecte que influeix sobre el temps d'execució és el fet de utilitzar les funcions predefinides d'Arduino com, per exemple si fem un petit programa utilitzant les funcions *pinMode, digitalRead* i *digitalWrite* mirem quina mida ocupa un cop compilat.

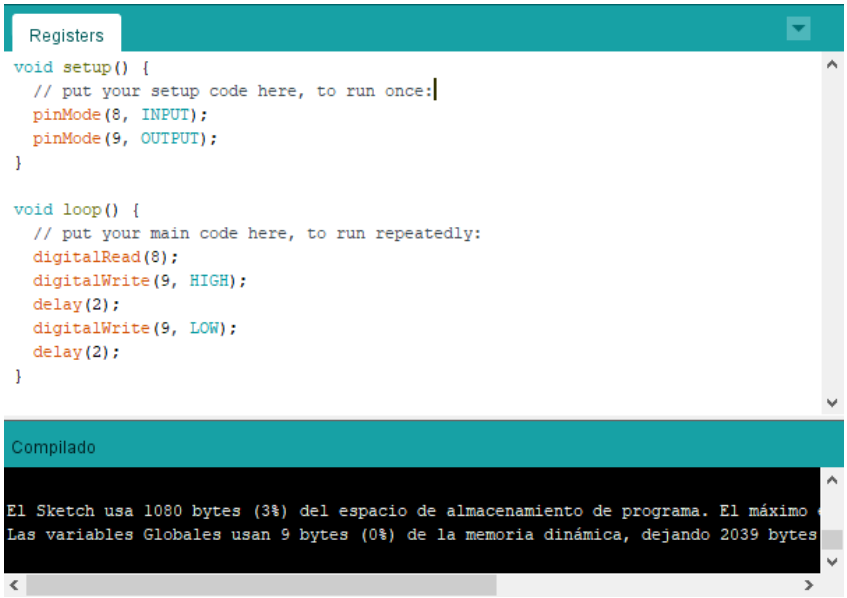



Figura 7. 2. 2: Programa senzill utilitzant funcions d'Arduino.

Si el mateix programa el reescrivim canviant el valor dels registres directament podem observar que l'espai que ocupa disminueix bastant.

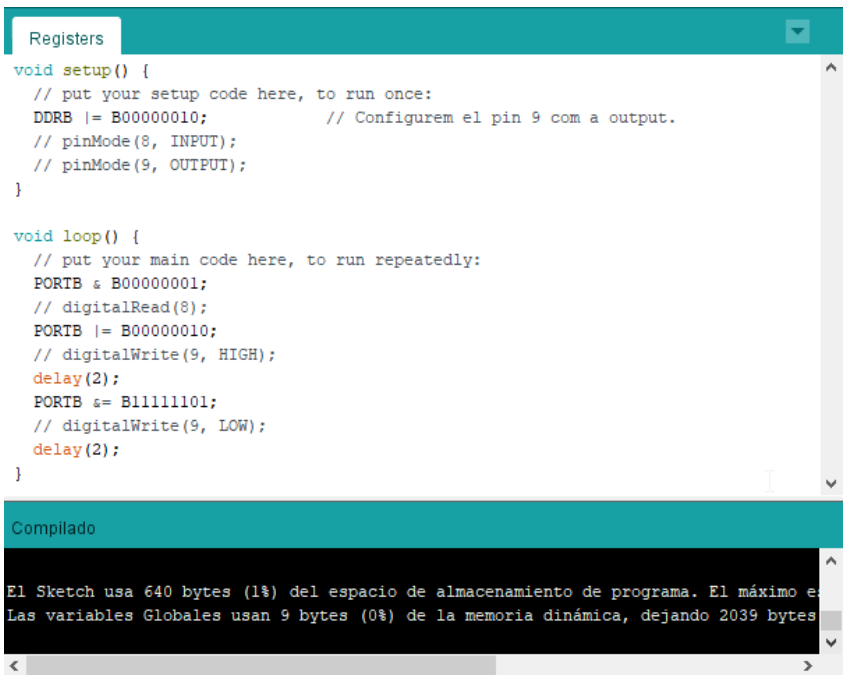



Figura 7. 2. 3: Programa senzill utilitzant registres.

Podem observar que programar d'una forma visualment més entenedora i neta ens perjudica augmentant molt la memòria que ocupa el nostre controlador i, en conseqüència, sembla que hauria d'augmentar el temps d'execució. Tot seguit farem la prova utilitzant l'oscil·loscopi per veure quin temps d'execució tardaríem per començar la senyal PWM als 4 ESC utilitzant funcions d'Arduino o utilitzant registres.

```
void loop() {
  digitalWrite(9, HIGH);
  digitalWrite(10, HIGH);
  digitalWrite(11, HIGH);
  digitalWrite(12, HIGH);
  digitalWrite(13, HIGH);
  digitalWrite(9, LOW);
  delay(4);
}
```


Figura 7. 2. 4: Temps d'execució utiltizant funcions d'Arduino.

```
void loop() {
  digitalWrite(9, HIGH);
  PORTB |= B00111100;
  digitalWrite(9, LOW);
  delay(4);
}
```

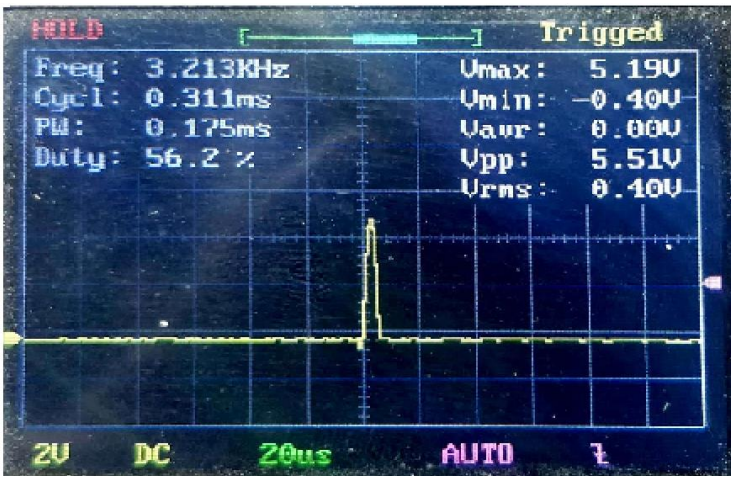



Figura 7. 2. 4: Temps d'execució utiltizant registres.

Tal i com podem veure si que ens perjudica en el temps d'execució, en un punt com donar la senyal PWM als ESC ens hem d'assegurar que la pèrdua de temps sigui mínima, és per això que utilitzarem els registres per accedir i modificar valors d'alguns pins. Ens ajudarà molt tenir el diagrama de pins de la nostra Arduino Uno a mà per poder-lo consultar, aquest diagrama el podem trobar adjuntat a l'annex 13. 1.

## PID

Tal i com hem vist al principi d'aquest apartat el nostre controlador no només ha de donar les ordres als motors del dron per realitzar el moviment que li indiquem a través del comandament sinó també per corregir la desestabilització provocada per varis factors com la distribució del pes del quadcopter, la fricció amb l'aire o la força que genera cada motor.
Així doncs una de les parts més importants del nostre controlador és aconseguir aquesta correcció que ha de enviar a cada un dels motors per tal que el quadcopter es mantingui estable a l'aire.
També hem vist a l'apartat 4. 3 que hi ha dos modes de control d'un quadcopter, l'acrobàtic on el controlador només s'encarrega de contrarestar la rotació no desitjada i l'estable on el controlador s'encarrega de contrarestar la rotació no desitjada i, a més a més, la inclinació.
Com que jo sóc un aficionat en el món dels drons he optat per realitzar el meu estabilitzador per tal que el quadcopter funcioni en mode estable ja que és més senzill de fer volar.

Un cop definit el mode de control que utilitzarem hem de dissenyar com podem fer aquesta estabilització amb Arduino. Per fer-ho desenvoluparem un controlador PID.
Un controlador PID és un mecanisme que calcula la diferencia entre el valor real d'una variable i el valor desitjat d'aquesta i corregeix aquest error perquè es minimitzi mitjançant la suma dels seus tres paràmetres fonamentals:

- Paràmetre Proporcional (P). Mesura la diferència entre el valor actual i el desitjat i actua de forma proporcional sobre aquest multiplicant per la constant *P*. Aquesta part farà que el control d'estabilitat del nostre dron sigui més o menys agressiva.
  Sortida_P = error x P
- Paràmetre Integral (I). Mesura l'error que anem acumulant al llarg del temps i el multiplica per la constant *I*. Gràcies a això aconseguim que l'error al llarg del temps sigui sempre 0.
  Sortida_I = ∑errors x I
- Paràmetre Derivatiu (D). Mesura la diferència d'errors entre cicles i el multiplica per la constant *D*. Serveix per suavitzar la resposta del control.
  Sortida_D = (error – error_previ) x D

A les figures següents podem veure com evoluciona un error al aplicar-hi cada paràmetre d'un controlador PID.

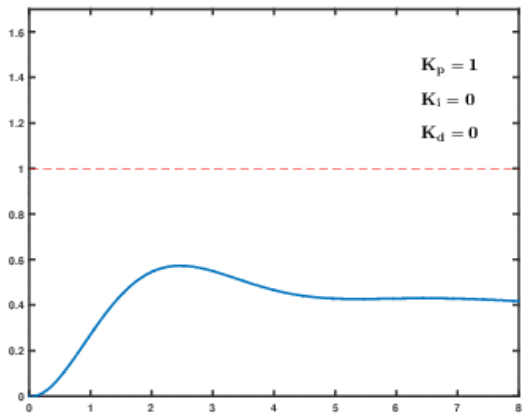



Figura 7. 2. 5: Error inicial.

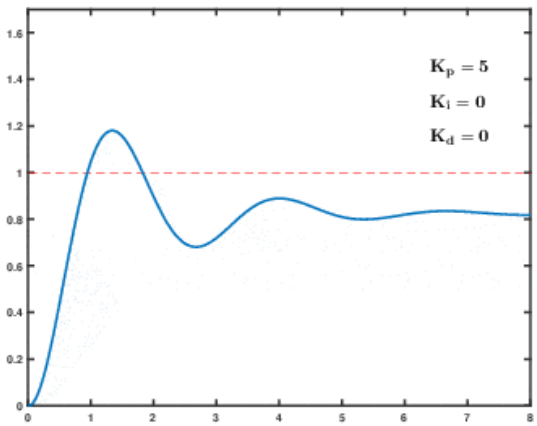



Figura 7. 2. 6: Error al aplicar el paràmetre P.

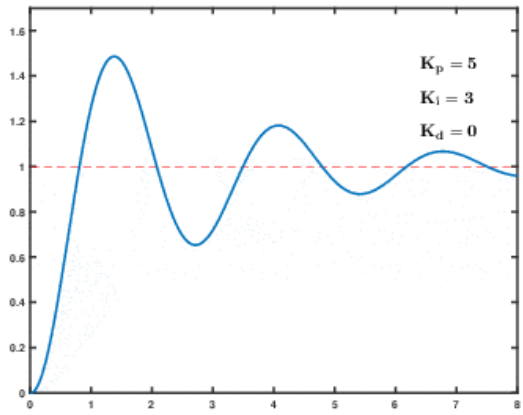
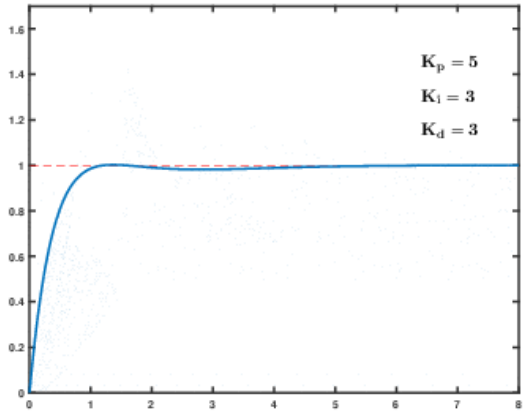



Figura 7. 2. 7: Error al aplicar el paràmetre I.




Error 8. 2 8: Error al aplicar el paràmetre D.

De forma esquemàtica l'algoritme d'un PID seria de la forma següent:

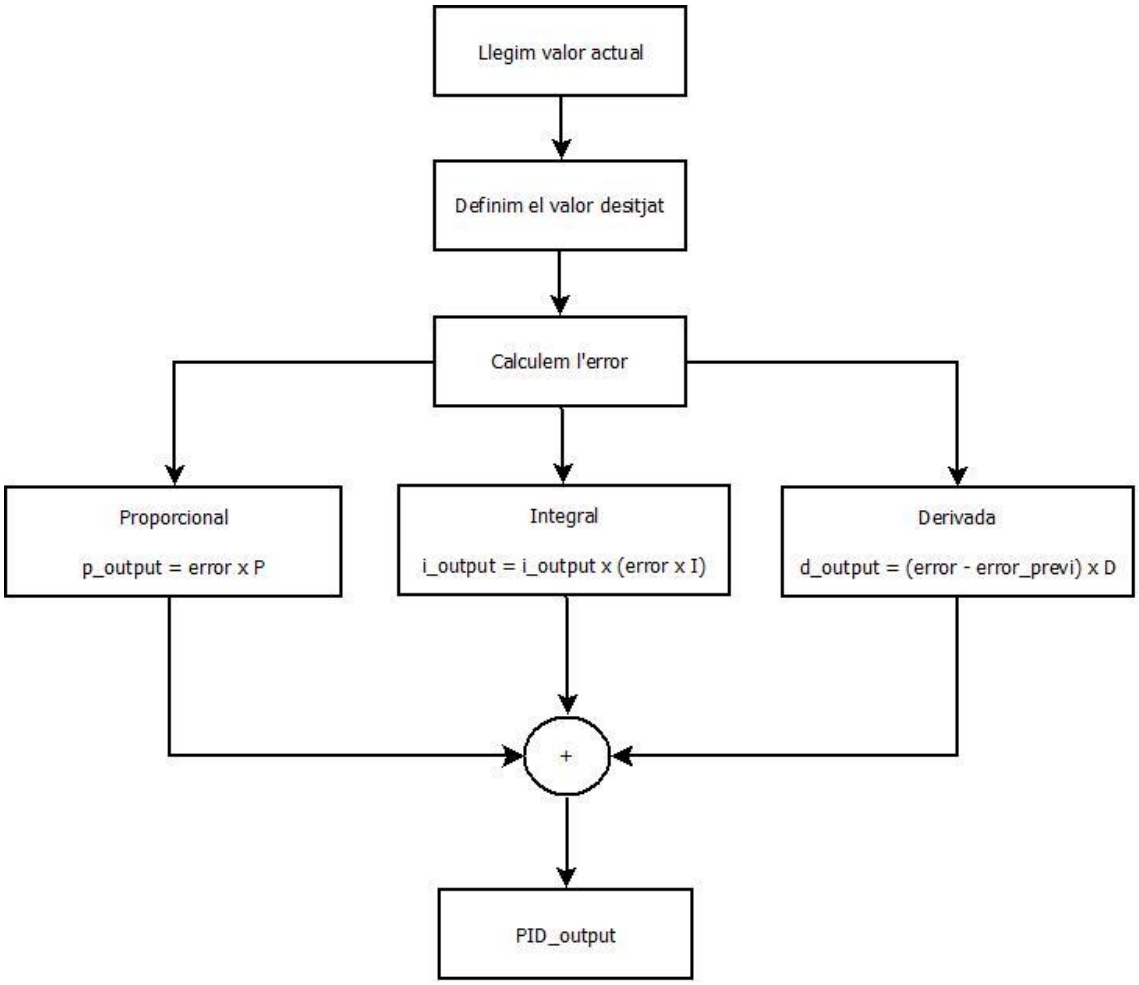



Figura 7. 2. 9: Esquema de l'algoritme d'un PID.

Nosaltres haurem d'aplicar aquest algoritme per corregir la rotació en roll, pitch i yaw i també per corregir la inclinació en roll i pitch.

# 8. Implementació i proves

L'objectiu d'aquest projecte és entendre bé com funciona cada component d'un quadcopter, és per això que la planificació es basa en primer fer proves unitàries de cada component per separat per llavors poder realitzar el controlador final.

Tots els retalls de codi que trobarem a aquest apartat i al següent es troben al repositori públic indicat a continuació sota una llicència OpenSource:
https://bitbucket.org/qpascualsola/quadcopter

A cada prova s'indicarà el nom del projecte d'Arduino que li correspon per fer més senzill el seguiment. També serà de gran ajuda disposar del *datasheet* del microcontrolador ATMega328 i del diagrama de pins que em vist a l'apartat anterior.

## 8. 1. Comandament RC i receptor

*- Projecte Arduino: Test → RCTest*

Tal i com hem dit a apartats anteriors un comandament RC funciona enviant una senyal a un receptor i aquest envia aquesta senyal PWM a la nostra Arduino connectada a per pins digitals.

Per no ocupar temps dins el programa principal utilitzarem la interrupció *"change"* que farà que l'execució del nostre programa s'aturi cada cop que el valor d'un pin digital canviï per executar una funció que definirem a continuació.

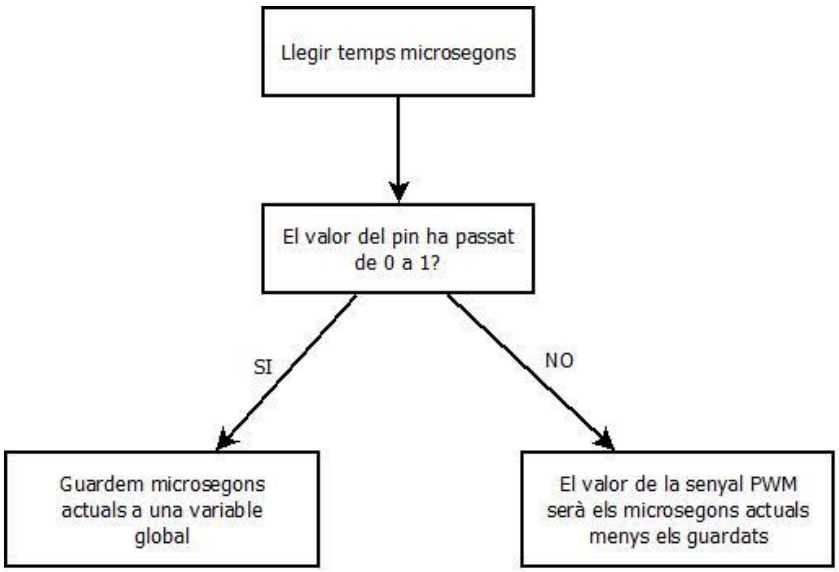



**Figura 8. 1. 1: Disseny de la rutina d'interrupció.**

El funcionament és molt simple, necessitem saber quan de temps un pin està en valor 1 (o HIGH) per tant cada cop que entrem a la funció de la interrupció utilitzarem la funció *micros* d'Arduino que ens torna el número de microsegons des de que ha començat el nostre

programa. Llavors mirarem si la interrupció s'ha produït perquè el pin ha canviat de 0 a 1, en cas afirmatiu guardarem aquests microsegons a una variable global, en cas que el pin hagi passat de valor 1 a 0 el que farem es guardar el valor de la senyal PWM que serà els microsegons actuals menys els que hem guardat anteriorment.

Farem el mateix disseny per els quatre pins digitals que volem llegir que es corresponen als quatre canals del comandament RC que ens interessen: altitud, roll, pitch i yaw. Els comandaments RC permeten configurar quin número de canal té cada un.

Ara que tenim clara com serà la nostra rutina de la interrupció simplement hem d'indicar a la nostra Arduino que volem utilitzar la interrupció PCINT per els quatre ports digitals on hem connectar el receptor RC, en el nostre cas són els ports D8, D9, D10 i D11.
Per activar la PCINT hem de modificar el valor del registre PCICR (Pin Change Interrupt Control Register) indicant quins grups de pins poden disparar la interrupció. Els tres grups de pins són:

- PCMSK0. Es troba al bit 0 (amb nom PCIE0) i conté els pins D8 fins D13.
- PCMSK1. Es troba al bit 1 (amb nom PCIE1) i conté els pins A0 fins A5.
- PCMSK1. Es troba al bit 2 (amb nom PCIE2) i conté els pins D0 fins D7.

A les següents figures podem observar la informació del *datasheet* del microcontrolador de la nostra Arduino sobre els registres que hem de modificar i com podem fer-ho amb Arduino.

**PCICR – Pin Change Interrupt Control Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x68) | – | – | – | – | – | PCIE2 | PCIE1 | PCIE0 | PCICR |
| Read/Write | R | R | R | R | R | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

Figura 8. 1. 2: Registre PCICR per activar / desactivar les PCINT.

**PCMSK0 – Pin Change Mask Register 0**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x6B) | PCINT7 | PCINT6 | PCINT5 | PCINT4 | PCINT3 | PCINT2 | PCINT1 | PCINT0 | PCMSK0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

Figura 8. 1. 3: Registre PCMSK0 per activar / desactivar quins pins del D8 al D13 disparen la PCINT.

```
void setup() {
  PCICR |= (1 << PCIE0);                    // Activem les interrupcions per el registre PCMSK0.
  PCMSK0 |= (1 << PCINT0);                  // Activem el pin digital 8 perque dispari la interrupco al canviar de valor.
  PCMSK0 |= (1 << PCINT1);                  // Activem el pin digital 9 perque dispari la interrupco al canviar de valor.
  PCMSK0 |= (1 << PCINT2);                  // Activem el pin digital 10 perque dispari la interrupco al canviar de valor.
  PCMSK0 |= (1 << PCINT3);                  // Activem el pin digital 11 perque dispari la interrupco al canviar de valor.

  Serial.begin(9600);
}
```

Figura 8. 1. 4: Activem la PCINT per els pins D8 a D11.

Finalment només haurem de definir la funció que s'ha d'executar quan es dispari la nostra interrupció tal i com es pot veure a la figura següent:

```
ISR(PCINT0_vect) {
  current_time = micros();

  // Canal 1
  if(PINB & B00000001){                                    // Si el pin D8 - HIGH
    if(last_channel_1 == 0){                               // Si abans el pin D8 - LOW
      last_channel_1 = 1;                                  // Guardem valor actual del pin D8.
      timer_1 = current_time;                              // Guardem current_time a timer_1.
    }
  }
  else if(last_channel_1 == 1){                            // Altrament si el pin D8 - LOW i abans estava HIGH
    last_channel_1 = 0;                                    // Guardem valor actual del pin D8.
    receiver_input_channel_1 = current_time - timer_1;     // Guardem valor senyal del canal 1.
  }


  // Canal 2
  if(PINB & B00000010){                                    // Si el pin D9 - HIGH
    if(last_channel_2 == 0){                               // Si abans el pin D9 - LOW
      last_channel_2 = 1;                                  // Guardem valor actual del pin D9.
      timer_2 = current_time;                              // Guardem current_time a timer_2.
    }
  }
  else if(last_channel_2 == 1){                            // Altrament si el pin D9 - LOW i abans estava HIGH
    last_channel_2 = 0;                                    // Guardem valor actual del pin D9.
    receiver_input_channel_2 = current_time - timer_2;     // Guardem valor senyal del canal 2.
  }
}
```

**Figura 8. 1. 5: Definició de la funció d'interrupció.**

D'aquesta forma obtenim el valor de la senyal PWM del canal 1 i canal 2 en microsegons, per fer els dos canals restants seria igual que en aquests simplement canviant el pin.
Ara podem afegir uns *prints* al nostre programa principal (funció *loop*) i així veure els valor que rebem del nostre comandament RC per els diferents moviments.

```
Channels Values 1:1540  2:1488  3:1012  4:1496
Channels Values 1:1540  2:1488  3:1016  4:1492
Channels Values 1:1544  2:1488  3:1012  4:1496
Channels Values 1:1544  2:1488  3:1016  4:1492
Channels Values 1:1544  2:1488  3:1016  4:1496
Channels Values 1:1544  2:1484  3:1016  4:1492
Channels Values 1:1544  2:1488  3:1016  4:1492
Channels Values 1:1544  2:1488  3:1012  4:1496
Channels Values 1:1540  2:1488  3:1012  4:1492
Channels Values 1:1544  2:1484  3:1016  4:1496
Channels Values 1:1544  2:1488  3:1016  4:1496
```

```
Channels Values 1:1992  2:1448  3:1992  4:1496
Channels Values 1:1992  2:1440  3:1996  4:1492
Channels Values 1:1992  2:1440  3:1992  4:1492
Channels Values 1:1988  2:1444  3:1992  4:1492
Channels Values 1:1992  2:1440  3:1992  4:1496
Channels Values 1:1992  2:1444  3:1992  4:1492
Channels Values 1:1992  2:1440  3:1996  4:1492
Channels Values 1:1992  2:1440  3:1992  4:1492
Channels Values 1:1992  2:1444  3:1992  4:1492
Channels Values 1:1992  2:1444  3:1992  4:1492
Channels Values 1:1992  2:1440  3:1996  4:1492
```

**Figura 8. 1. 6: Sortida del Serial d'Arduino mostrant els valors rebuts del comandament RC.**

# 8. 2. IMU

Tal i com hem vist anteriorment una IMU és un dispositiu electrònic que mesura i informa sobre la velocitat, orientació i forces gravitacionals d'un aparell mitjançant la combinació d'acceleròmetres i giroscopis. Al tenir tres acceleròmetres i tres giroscopis disposats de manera que formen tres eixos ortogonals (x, y, z) ens permet obtenir la inclinació (acceleròmetres) i velocitat angular del nostre quadcopter (giroscopis).

Per entendre bé el seu funcionament farem varies proves: primer provarem d'obtenir l'angle només utilitzant el giroscopi, després ho farem només utilitzant l'acceleròmetre i, finalment, provarem d'obtenir l'angle amb la combinació dels dos.

## Giroscopi

*- Projecte Arduino: Test → Test_MPU6050_LCD_ONLY_GYRO*

Tal i com hem indicat anteriorment un giroscopi ens permet obtenir la velocitat angular de l'aparell en º/s. Un cop obtinguda la velocitat de rotació calcular l'angle és senzill, si sabem que el nostre quadcopter ha estat rotant a una determinada velocitat durant un cert període de temps podem saber quin angle s'ha desplaçat i, per tant, la seva orientació actual.
Tal i com hem dissenyat el nostre controlador sabem que el programa principal s'executarà amb una freqüència de 250Hz, és a dir cada 4ms, per tant tenim totes les variables que ens fan falta per calcular l'angle de inclinació. Anem a fer-ho.

Primer de tot necessitem consultar el *datasheet* de la nostra IMU MPU6050 per saber quins registres hem de llegir i quins valors ens donarà.

**6.1 Gyroscope Specifications**

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, $T_A$ = 25°C

| PARAMETER | CONDITIONS | MIN | TYP | MAX | UNITS | NOTES |
|---|---|---|---|---|---|---|
| **GYROSCOPE SENSITIVITY** | | | | | | |
| Full-Scale Range | FS_SEL=0 | | ±250 | | º/s | |
| | FS_SEL=1 | | ±500 | | º/s | |
| | FS_SEL=2 | | ±1000 | | º/s | |
| | FS_SEL=3 | | ±2000 | | º/s | |
| Gyroscope ADC Word Length | | | 16 | | bits | |
| Sensitivity Scale Factor | FS_SEL=0 | | 131 | | LSB/(º/s) | |
| | FS_SEL=1 | | 65.5 | | LSB/(º/s) | |
| | FS_SEL=2 | | 32.8 | | LSB/(º/s) | |
| | FS_SEL=3 | | 16.4 | | LSB/(º/s) | |

**Figura 8. 2. 1: Especificacions del giroscopi.**

Podem observar que podem configurar el giroscopi amb diferents sensibilitats, en el nostre cas escollirem la configuració 1 que ens proporciona una sensibilitat de 500 º/s i hauria de ser suficient. Tot i així podríem seleccionar una sensibilitat més elevada si volguéssim.
Les especificacions també ens indiquen que la longitud de paraula és de 16 bits i que cada 65.5LSB (Least Significant Bit) que llegim del sensor equival a 1 º/s de velocitat angular.
Per tant simplement hem de llegir el valor que ens dona el giroscopi i dividir-lo entre 65.5 per obtenir la velocitat de rotació.

Exemples:

- Si la nostra IMU gira a una velocitat de 1 º/s el valor que llegirem del giroscopi serà de 65.5.
- Si la nostra IMU gira a una velocitat de 6 º/s el valor que llegirem del giroscopi serà de 393.
- Si la nostra IMU gira a una velocitat de 500 º/s el valor que llegirem del giroscopi serà de 32750 (valor similar al màxim que podem guardar dins una variable de tipus *int*).

Un cop vist el funcionament anem a programar-lo, el primer que hem de fer és configurar el giroscopi del nostre MPU6050. Com que utilitzem el bus $I^2C$ per comunicar-nos amb la IMU farem servir llibreria *Wire* que ens ajudarà a escriure i llegir registres. Podem realitzar la configuració amb la següent funció:

```
void setup_IMU() {
  uint8_t gyro_address = 0x68;
  Wire.beginTransmission(gyro_address);                    // Comencem la comunicació amb el sensor MPU6050.
  Wire.write(0x6B);                                        // Volem escriure al registre PWR_MGMT_1 (0x6B).
  Wire.write(0x00);                                        // Escrivim 00000000 al registre per activar el sensor.
  Wire.endTransmission();                                  // Fi de la comuniació.

  Wire.beginTransmission(gyro_address);                    // Comencem la comunicació amb el sensor MPU6050.
  Wire.write(0x1B);                                        // Volem escriure al registre GYRO_CONFIG (0x1B).
  Wire.write(0x08);                                        // Escrivim 00001000 al registre per definir la sensibilitat a 500dps.
  Wire.endTransmission();                                  // Fi de la comuniació.

  // Comprovem que les dades s'han guardat correctament.
  Wire.beginTransmission(gyro_address);                    // Comencem la comunicació amb el sensor MPU6050.
  Wire.write(0x1B);                                        // Volem llegir el registre GYRO_CONFIG (0x1B).
  Wire.endTransmission();                                  // Fi de la comuniació.
  Wire.requestFrom(gyro_address, 1);                       // Demanem 1 byte al sensor.
  while(Wire.available() < 1);                             // Esperem a que tinguem les dades per llegir.

  if(Wire.read() != 0x08){                                 // Comprovem que el valor sigui el que hem escrit anteriorment.
    Serial.println("GYRO_CONFIG INCORRECT");
    while(1) delay(10);
  }

  Wire.beginTransmission(gyro_address);                    // Comencem la comunicació amb el sensor MPU6050.
  Wire.write(0x1A);                                        // Volem escriure al registre CONFIG (0x1A).
  Wire.write(0x03);                                        // Escrivim 00000011 al registre per definir el Digital Low Filter.
  Wire.endTransmission();                                  // Fi de la comuniació.
}
```

**Figura 8. 2. 2: Configuració sensor MPU6050 per utilitzar el giroscopi.**

Un cop configurat ja podem utilitzar el giroscopi de la nostra IMU i llegir les dades de la següent manera:

```
void read_gyro_data() {
  Wire.beginTransmission(0x68);                            // Comencem la comunicació amb el sensor MPU6050.
  Wire.write(0x43);                                        // Volem llegir el registre GYRO_XOUT_H (0x4).
  Wire.endTransmission();                                  // Fi de la comuniació.

  Wire.requestFrom(0x68, 6);                               // Demanem 6 bytes al sensor.
  while (Wire.available() < 6);                            // Esperem a que tinguem les dades per llegir.

  gyro_data_x = (Wire.read() << 8) | Wire.read();          // Llegim els 16 bits X.
  gyro_data_y = (Wire.read() << 8) | Wire.read();          // Llegim els 16 bits Y.
  gyro_data_z = (Wire.read() << 8) | Wire.read();          // Llegim els 16 bits X.
}
```

**Figura 8. 2. 3: Llegir dades del giroscopi.**

Si executem un programa que ens mostri per el Serial aquestes dades podrem veure que ens apareixen valors diferents de 0 encara que no l'IMU no es trobi en moviment, això és degut a que necessitem calibrar el giroscopi abans de utilitzar-lo. Per fer-ho simplement agafarem un número elevat de mostres i farem la mitjana del valor del giroscopi per x, y i z, aquest valor mig el restarem del valor obtingut en cada lectura del giroscopi.

```
void calibrate_gyro() {
  int x = 0;
  for (int i = 0; i < CALIB_GYRO_LOOPS; i++) {
    read_gyro_data();
    gyro_cal_x += gyro_data_x;
    gyro_cal_y += gyro_data_y;
    gyro_cal_z += gyro_data_z;
    delay(3);
  }

  gyro_cal_x /= CALIB_GYRO_LOOPS;
  gyro_cal_y /= CALIB_GYRO_LOOPS;
  gyro_cal_z /= CALIB_GYRO_LOOPS;
}
```

**Figura 8. 2. 4: Calibració del giroscopi.**

Un cop calibrat el giroscopi ja podem llegir correctament les seves dades i calcular l'angle. Abans de fer-ho tornem un moment a l'exemple que hem utilitzat abans i fem uns càlculs senzills. Si la nostra IMU gira a una velocitat constant de 6 º/s voldrà dir que dona una volta sencera en un minut. Com podem calcular-ho amb les dades que ens dona el giroscopi?

- El valor que llegirem del giroscopi en qualsevol moment és 393.
- Si fem una lectura del giroscopi cada segon durant un minut tindrem una lectura total de 23580 (393 x 60s).
- Si aquest valor llegit el dividim entre 65.5 obtindrem exactament 360 que són els graus que ha girat la nostra IMU.

Per adaptar aquests càlculs al nostre quadcopter hem de fer-ho de la següent forma:

- El nostre programa s'executa amb una freqüència de 250Hz. Això significa que llegirem les dades del giroscopi 250 vegades en un segon.
- Veiem que la lectura total del giroscopi l'hem de dividir entre 250 ja que ara no llegim una vegada cada segon sinó 250. Finalment el resultat el dividim entre 65.5 i obtindrem els graus que ha girat la nostra IMU.
  Per no haver de fer dos divisions podem multiplicar la lectura total del giroscopi per 0.0000611 que s'obté de 1 dividit entre 250 i dividit entre 65.5.
- Si mirem l'exemple anterior veurem que ara fem 250 lectures per segon, això vol dir que en un minut farem 15000 lectures amb valor 393 cada una. Això suma un total de 5895000.
- Si aquest valor llegit el multipliquem per 0.0000611 obtindrem que la nostra IMU ha girat 360.1845 graus, un aproximació prou acceptable.

Per trobar els angles amb Arduino ho faríem com es mostra a continuació:

```
// Llegim les dades del giroscopi.
read_gyro_data();

// Restem les dades de calibració.
gyro_data_x -= gyro_cal_x;
gyro_data_y -= gyro_cal_y;
gyro_data_z -= gyro_cal_z;

// Càlcul d'angles giroscopi.
// 0.0000611 = 1 / 250Hz / 65.5
angle_pitch_gyr += gyro_data_y * 0.0000611;                    // Calculem l'angle pitch rotat i el sumem a l'angle_pitch per saber l'orientació.
angle_roll_gyr += gyro_data_x * 0.0000611;                     // Calculem l'angle roll rotat i el sumem a l'angle_roll per saber l'orientació.
```

**Figura 8. 2. 5: Càlcul dels angles pitch i roll mitjançant el giroscopi.**

Si ara provem de pintar aquests dos angles pot semblar que el nostre programa ja funciona correctament però anem a veure que passa quan fem el moviment que es pot veure en la Figura 8. 2. 6.

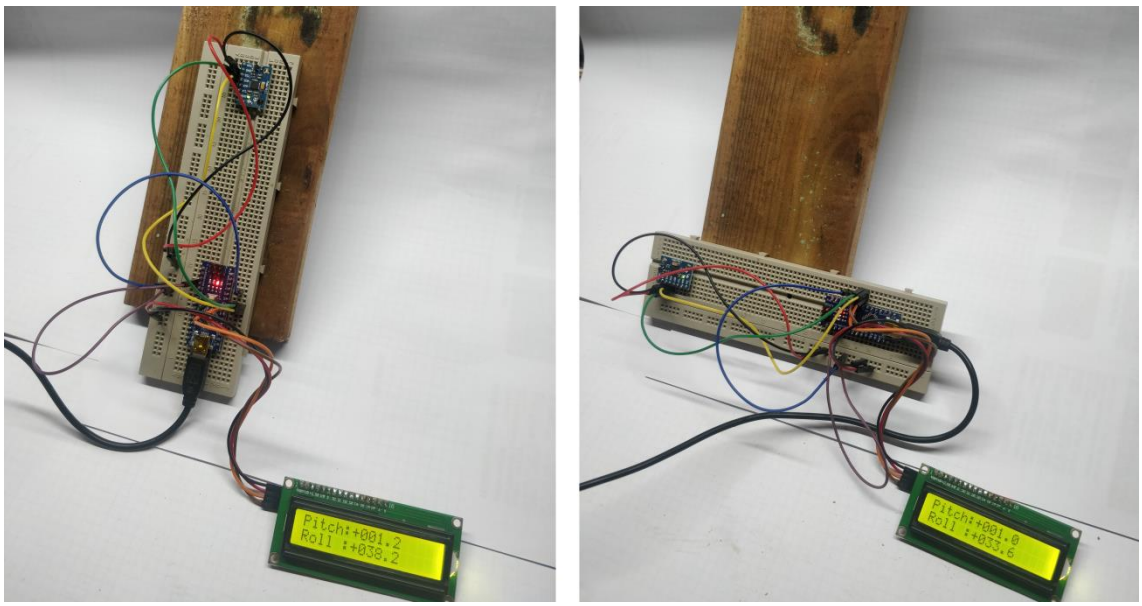



**Figura 8. 2. 6: Problema en el càlcul d'angles al no tenir en compte la rotació en yaw.**

Tal i com podem observar primer hem col·locat la nostra IMU sobre una base inclinada en roll a més o menys 40 º, llavors hem rotat 90 º en yaw la nostra IMU i els angles calculats no han variat pràcticament quan realment podem observar que la IMU ja no té inclinació en roll sinó que la té en pitch. Això ens ha passat perquè en el nostre programa només tenim en compte els moviment en pitch i en roll, al no tenir en compte la rotació en yaw tenim un càlcul incorrecte dels angles pitch i roll.

Per arreglar això si busquem informació trobarem que la funció que es correspon a l'intercanvi d'angles entre pitch i roll quan hi ha un moviment yaw es correspon a la funció sinus de l'angle rotat en yaw, per tant els càlculs de l'angle es farien de la següent forma:

```
// Càlcul d'angles giroscopi.
// 0.0000611 = 1 / 250Hz / 65.5.
angle_pitch_gyr += gyro_data_y * 0.0000611;                    // Calculem l'angle pitch rotat i el sumem a l'angle_pitch per saber l'orientació.
angle_roll_gyr += gyro_data_x * 0.0000611;                     // Calculem l'angle roll rotat i el sumem a l'angle_roll per saber l'orientació.

//0.000001066 = 0.0000611 * (3.142(PI) / 180degr) La funció sin és en radiants.
angle_pitch_gyr -= angle_roll_gyr * sin(gyro_data_z * 0.000001066);          // Si la IMU ha rotat en yaw transferim l'angle roll a l'angle pitch.
angle_roll_gyr += angle_pitch_gyr * sin(gyro_data_z * 0.000001066);          // Si la IMU ha rotat transferim l'angle pitch a l'angle roll.
```

**Figura 8. 2. 7: Càlcul dels angles pitch i roll mitjançant el giroscopi i tenint en compte la rotació en yaw.**

Si ara tornem a realitzar proves veurem que funciona correctament, almenys una estona ja que al cap d'uns segons es començarà a manifestar el que és conegut com a *drifting*. Encara que no moguem la nostra IMU els angles aniran variant mica en mica degut a l'error que es va acumulant el poc error que hi ha en cada càlcul dels angles ja s'utilitzen aproximacions.

Un altre problema que ens trobem és quan la nostra IMU comença en un pla inclinat ja que el giroscopi agafarà com a angles 0 el pla inclinat on es troba i no un pla que realment no tingui inclinació.

Per solucionar aquests dos problemes necessitarem l'ajuda de l'acceleròmetre.

## Acceleròmetre

*- Projecte Arduino: Test → Test_MPU6050_LCD_ONLY_ACC*

Per fer una explicació senzilla l'acceleròmetre ens permet obtenir la força d'acceleració en unitats *g* que està experimentant el nostre sensor en cada un dels tres eixos indicats en el seu manual d'usuari. A la Figura 8. 2. 8 podem observar com estan posicionats aquests eixos.

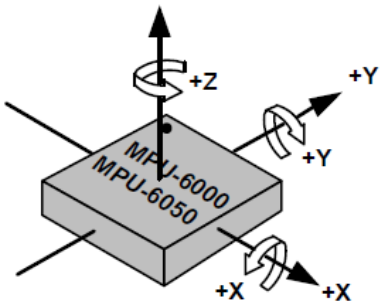



**Figura 8. 2. 8: Eixos del sensor MPU6050.**

Primer de tot, igual que amb el giroscopi, hem de configurar l'acceleròmetre, com que anteriorment ja hem vist com es fa només direm que la configuració que farem servir és la que ens dona una sensibilitat de 8g. Aquesta configuració implica que quan obtenim les dades de l'acceleròmetre llegirem un valor de 4096 per cada 1g.

**6.2 Accelerometer Specifications**

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, $T_A$ = 25°C

| PARAMETER | CONDITIONS | MIN | TYP | MAX | UNITS | NOTES |
|---|---|---|---|---|---|---|
| **ACCELEROMETER SENSITIVITY** | | | | | | |
| Full-Scale Range | AFS_SEL=0 | | ±2 | | *g* | |
| | AFS_SEL=1 | | ±4 | | *g* | |
| | AFS_SEL=2 | | ±8 | | *g* | |
| | AFS_SEL=3 | | ±16 | | *g* | |
| ADC Word Length | Output in two's complement format | | 16 | | bits | |
| Sensitivity Scale Factor | AFS_SEL=0 | | 16,384 | | LSB/*g* | |
| | AFS_SEL=1 | | 8,192 | | LSB/*g* | |
| | AFS_SEL=2 | | 4,096 | | LSB/*g* | |
| | AFS_SEL=3 | | 2,048 | | LSB/*g* | |
| Initial Calibration Tolerance | | | ±3 | | % | |

**Figura 8. 2. 9: Especificacions de l'acceleròmetre.**

Un cop realitzada la configuració podem llegir els valors de l'acceleròmetre i, deixant la nostra IMU en repòs en un pla no inclinat veurem que el valor Z de l'acceleròmetre ens dona un valor de més o menys 4096. Això és perquè la nostra IMU està partint la força de la gravetat que és exactament 1g.

Si inclinem la nostra IMU a 45 º veurem que el valor de l'acceleròmetre en Z disminueix i en X o Y augmenta de manera que tenim un valor pràcticament igual en els dos eixos. Això passa perquè la força de la gravetat queda repartida en els dos eixos enlloc de només en l'eix Z.
A la Figura 8. 2. 10 podem veure com queden repartides les forces i en la 9. 2. 11 podem veure que si organitzem les fletxes podem formar un triangle rectangle.

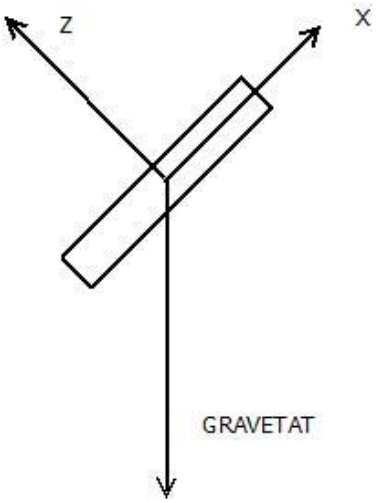



Figura 8. 2. 10: Forces acceleròmetre.

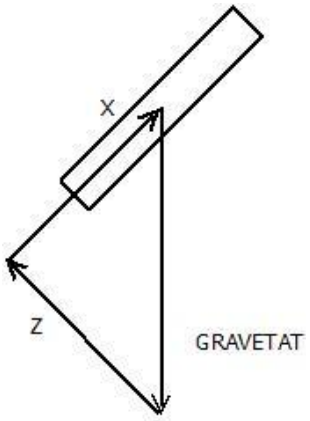



Figura 8. 2. 11: Forces acceleròmetre formant un triangle rectangle.

Sabem que la hipotenusa d'aquest rectangle val 1g, per tant l'acceleròmetre ens torna un valor de més o menys 4096. Aplicant el teorema de Pitàgores podem calcular que el valor de l'acceleròmetre en X i Z és més o menys 2896. Aquest valor podem comprovar-lo llegint les dades de l'acceleròmetre. Un cop tenim tots els costats d'aquest triangle podem buscar l'angle que formen els vectors X i Gravetat:

- Dividim el costat contigu (vector X) amb la hipotenusa. 2896 / 4096 = 0.707.
- Aquest valor és el cosinus de l'angle. Utilitzant la funció *arccos* trobem que l'angle és de 45º tal i com esperàvem.

L'acceleròmetre ens proporciona sempre els dos catets d'aquest triangle rectangle que es forma, per tant podem aplicar la trigonometria per calcular els angles en que està orientada l'IMU.

```
// Llegim les dades de l'accelerometre.
read_acc_data();

// Càlcul d'angles acceleròmetre.
acc_total_vector = sqrt((acc_data_x*acc_data_x)+(acc_data_y*acc_data_y)+(acc_data_z*acc_data_z));   // Calculem el vector total d'acceleració.
// 57.296 = 1 / (3.142 / 180) La funció asin d'Arduino treballa amb radiants.
angle_pitch_acc = asin((float)acc_data_y/acc_total_vector)* 57.296;                                 // Calculem l'angle pitch.
angle_roll_acc = asin((float)acc_data_x/acc_total_vector)* -57.296;                                 // Calculem l'angle roll.
```

Figura 8. 2. 12: Càlcul dels angles pitch i roll mitjançant l'acceleròmetre.

D'aquesta manera, si ara fem un *print* dels nostres angles pitch i roll semblaria que ja ens mostra uns valors correctes, hem solucionat els dos erros del giroscopi: el *drift* i si comencem a un pla inclinat el càlcul també és correcte.
Ara bé, si provem de simular vibracions en les direccions dels eixos de pitch i roll veurem que, tot i no modificar l'angle, els angles que hem calculat si que pateixen modificacions, això és degut a les acceleracions que pateix l'acceleròmetre amb les vibracions.
Quan tinguem la nostra IMU muntada al quadcopter també patirà vibracions i això fa que els angles obtinguts mitjançant l'acceleròmetre no ens siguin útils.

## Global

*- Projecte Arduino: Test → Test_MPU6050_LCD_GYRO_ACC*

Fem un resum del que hem vist fins ara:

- El giroscopi és poc sensible a les vibracions.
- El giroscopi pateix els problemes de *drift* i que no calcula bé l'angle si comença en un pla inclinat.
- L'acceleròmetre és massa sensible a les vibracions.

Amb tot això la conclusió on hem d'arribar és: com podem utilitzar les dades de l'acceleròmetre per solucionar els dos problemes del giroscopi?

És mes senzill del que pot semblar, simplement necessitarem fer un filtre complementari on les dades obtingudes per el giroscopi tindran molt més pes que les dades obtingudes per l'acceleròmetre. D'aquesta manera solucionarem el problema del *drift*.

```
angle_pitch_gyr = angle_pitch_gyr * 0.9996 + angle_pitch_acc * 0.0004;     // Corregim el drift de l'angle pitch del giroscopi amb l'angle pitch de l'accelerometre.
angle_roll_gyr = angle_roll_gyr * 0.9996 + angle_roll_acc * 0.0004;        // Corregim el drift de l'angle roll del giroscopi amb l'angle roll de l'accelerometre.
```

**Figura 8. 2. 13: Filtre complementari per corregir el *drift*.**

Anteriorment hem vist que els angles obtinguts amb l'acceleròmetre quan la nostra IMU es troba estable són molt fiables, és per això que utilitzarem aquests angles per inicialitzar els angles del giroscopi un cop acabada la calibració.

```
if (set_gyro_angles) {
  angle_pitch_gyr = angle_pitch_gyr * 0.9996 + angle_pitch_acc * 0.0004;     // Corregim el drift de l'angle pitch del giroscopi amb l'angle pitch de l'accelerometre.
  angle_roll_gyr = angle_roll_gyr * 0.9996 + angle_roll_acc * 0.0004;        // Corregim el drift de l'angle roll del giroscopi amb l'angle roll de l'accelerometre.
  angle_pitch_output = angle_pitch_gyr;
  angle_roll_output = angle_roll_gyr;
}
else {
  angle_pitch_gyr = angle_pitch_acc;
  angle_roll_gyr = angle_roll_acc;
  angle_pitch_output = angle_pitch_acc;
  angle_roll_output = angle_roll_acc;
  set_gyro_angles = true;
}
```

**Figura 8. 2. 14: Inicialització dels angles del giroscopi amb els angles de l'acceleròmetre.**

D'aquesta manera tan simple hem pogut solucionar els nostres problemes i obtenir unes dades prou bones com per ser utilitzades en el nostre quadcopter.

# 8. 3. Motors i ESC

*- Projecte Arduino: Test → RCTest_ESC_1Motor*

Hem vist anteriorment que la velocitat a la que gira el motor es controla mitjançant un ESC que és un dispositiu electrònic que s'encarrega de convertir la senyal de corrent contínua que li arriba de la bateria en una corrent alterna regulable en funció de la senyal PWM que rep.
Aquesta senyal PWM ha de ser de freqüència constant (en el nostre cas tindrà una freqüència de 250Hz) i d'amplada variable en funció de la velocitat de gir que volem obtenir. En la gran majoria dels ESC aquesta amplada de pols va de mínim 1ms i màxim 2ms, d'aquesta manera amb una amplada de pols de 1ms els motors no giraran i amb una de 2ms giraran a velocitat màxima.

Per generar aquesta PWM Arduino disposa de la llibreria *Servo* que té la funció *WriteMicroseconds* on només hem d'indicar el número de microsegons que ha de tenir d'amplada la nostra senyal PWM. L'inconvenient d'aquesta funció és que els polsos que genera són a una freqüència de 50Hz (20ms) i tal i com hem vist abans això no ens serveix per aconseguir la estabilització d'un quadcopter. És per això que ens haurem de generar els polsos d'una altra manera.

Tot seguit farem un programa bastant-nos amb el codi de l'apartat 8. 1 on enviarem a un ESC un PWM amb el valor que obtenim de l'accelerador del nostre comandament RC. Provarem el codi que podem veure a la Figura 8. 3. 1 però sense connectar encara l'ESC a l'Arduino ja que utilitzarem l'oscil·loscopi per comprovar que generem una senyal correcte. A la Figura 8. 3. 2 podem veure varies senyals llegides amb l'oscil·loscopi i apreciar com podem variar l'amplada del pols mitjançant el comandament RC.

```
void loop() {
  while (zero_timer + 4000 > micros());                      // Freqüència 250Hz.
  zero_timer = micros();

  throttle = receiver_input_channel_3;                       // Llegim el valor d'accelaració RC.
  if (throttle < 1000)
    throttle = 1000;
  else if (throttle > 2000)
    throttle = 2000;

  esc_1 = throttle;                                           // Definim l'amplada del nostre pols.

  PORTD |= B00010000;                                         // Canviem el valor del pin D4 a HIGH.

  timer_channel_1 = esc_1 + zero_timer;                       // Calculem el temps quan haurem de canviar el valor del pin D4 a LOW.

  while(PIND & B00010000){                                    // Mentre el valor de D4 és HIGH.
    esc_loop_timer = micros();                                // Obtenim el temps actual.
    if(timer_channel_1 <= esc_loop_timer) PORTD &= B11101111;  // Quan el temps ha passat canviem el valor del pin D4 a LOW.
  }
}
```

**Figura 8. 3. 1: Generació d'una senyal PWM amb freqüència de 250Hz.**

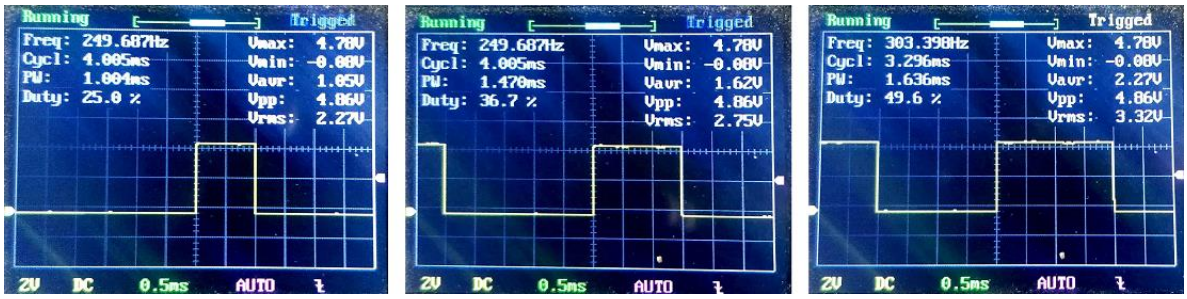



Figura 8. 3. 2: Senyals PWM llegides amb l'oscil·loscopi.

Un cop ens hem assegurat que estem generant i controlant la senyal PWM amb freqüència constant podem provar de utilitzar el nostre programa per controlar un únic motor.
Quan volem utilitzar un ESC per primera vegada hem de fer la seva calibració per tal que sàpiga quin és el seu valor màxim i el seu valor mínim, si no féssim aquesta calibració ens podríem trobar que dos ESC i motors que reben la mateixa senyal PWM no girin igual de ràpid, això provocaria que es desestabilitzés el nostre quadcopter.

**Molt important! Quan estiguem realitzant proves amb ESC i motors sempre ho farem sense les hèlixs ja que la força que fan els motors quan hi ha una hèlix muntada és molt gran i podríem patir un accident.**

Per realitzar la calibració hem de seguir els passos descrits al manual d'usuari dels nostres ESC tot i que moltes vegades per diferents models és fa de la mateixa manera que és la següent:

1. Sense connectar la bateria, alimentem la nostra Arduino mitjançant USB i carreguem el programa que estem treballant en aquest apartat. Obrim el monitor *Serial*.
2. Executem el programa sense connectar la bateria.
3. Quan veiem escrit *"GO"* al monitor *Serial* posem el joystick d'acceleració al màxim.
4. Connectem la bateria.
5. El motor emetrà dos *"pip"*.
6. Posem el joystick d'acceleració al mínim.
7. El motor emetrà el so *"pip pip pip, piiiiip"*.
8. Finalitzat, s'ha calibrat el nostre ESC.

Un cop realitzada la calibració si ara tornem a pujar el joystick d'acceleració veurem que el motor comença a girar i que gira més ràpid segons el valor que li donem amb el comandament.
Els següents cops que vulguem utilitzar aquest ESC ens hem d'assegurar que quan rebi corrent de la bateria el nostre joystick d'acceleració estigui al mínim ja que sinó es pensaria que l'estem tornant a calibrar (podem calibrar un ESC tants cops com vulguem).

# 8. 4. PID

*- Projecte Arduino: Test → Test_PID_ONLY_PITCH*

A l'apartat 7. 2 hem analitzat i dissenyat com ha de ser l'algoritme del nostre controlador PID que ens permetrà poder controlar el nostre quadcopter en mode estable. Si passem l'algoritme del PID dissenyat anteriorment a Arduino quedaria de la manera següent.

```
void calculate_pid() {
  pid_error_temp = gyro_pitch_input - pid_pitch_setpoint;
  pid_i_mem_pitch += pid_i_gain_pitch * pid_error_temp;
  if(pid_i_mem_pitch > pid_max_pitch) pid_i_mem_pitch = pid_max_pitch;
  else if(pid_i_mem_pitch < pid_max_pitch * -1) pid_i_mem_pitch = pid_max_pitch * -1;

  pid_output_pitch = pid_p_gain_pitch * pid_error_temp + pid_i_mem_pitch + pid_d_gain_pitch * (pid_error_temp - pid_last_pitch_d_error);
  if(pid_output_pitch > pid_max_pitch) pid_output_pitch = pid_max_pitch;
  else if(pid_output_pitch < pid_max_pitch * -1) pid_output_pitch = pid_max_pitch * -1;

  pid_last_pitch_d_error = pid_error_temp;
}
```

**Figura 8. 4. 1: Càlculs PID per corregir la rotació i inclinació en pitch.**

Aquesta funció simplement fa les multiplicacions i sumes de les tres parts d'un controlador PID, l'única part afegida és que tant amb la sortida total com la sortida de la part integral validem que no tinguem una sortida massa gran o massa petita que llavors ens pot implicar que la senyal que enviem als motors sobrepassi els 2ms o sigui inferior a 1ms.

La clau d'un controlador PID és definir correctament el *setpoint* (valor desitjat de la nostra variable), primer de tot fem una mica de repàs sobre les variables que utilitzem:

- Hem de tenir en compte que nosaltres llegim la informació del giroscopi en º/s.
- Els motors hem d'enviar una senyal PWM amb una amplitud entre 1ms i 2ms.
- Del comandament llegim una senyal PWM amb una amplitud entre més o menys 1ms i 2ms.

Les ordres que ens provenen del comandament són les que ens marquen quina velocitat de rotació ha d'agafar el nostre quadcopter però les rebem en microsegons enlloc de º/s com el giroscopi. Si tenim en compte que quan rebem una senyal de més o menys 1500 microsegons (deixarem una mica de marge) vol dir que el joystick es troba en posició central, mentre que si la senyal és de 1000 o 2000 microsegons vol dir que el joystick es troba en un extrem podem fer el següent:

```
pid_pitch_setpoint = 0;
if (receiver_pitch > 1508) pid_pitch_setpoint = receiver_pitch - 1508;
else if (receiver_pitch < 1492) pid_pitch_setpoint = receiver_pitch - 1492;
```

**Figura 8. 4. 2: Càlcul setpoint, part 1.**

D'aquesta manera sempre tindríem un *setpoint* amb un valor entre 0 i 492. Podríem dir que aquest valor és els graus per segon de velocitat de gir que ens arriba del joystick. Quan col·loquem el joystick en posició central estem ordenant que el dron es mantingui amb una velocitat de gir de 0 º/s mentre que si tenim el joystick a un extrem estarem dient al nostre

quadcopter que volem que giri a una velocitat de 492 º/s. Com podem veure 492 º/s és una velocitat molt i molt ràpida que no ens ajudarà a aprendre a controlar el quadcopter ja que els joysticks de gir seran molt agressius. Per solucionar-ho podem dividir el valor que ens resulta per una constant, si ho dividim per 2 obtindrem una velocitat màxima de gir de 242 º/s, si ho dividim per 3 de 164 º/s, si ho dividim per 4 de 123 º/s, etc.
Aquesta constant la podem escollir nosaltres, com més baix sigui el valor més agressiu serà el nostre quadcopter, jo he escollit dividir-ho per 3.
Fent aquesta modificació ja tindríem el *setpoint* definit, com que només tenim en compte que la velocitat de gir del quadcopter sigui igual a la velocitat de gir especificada per el comandament el PID actual ens actuaria per tal que el dron volés en mode acrobàtic.

```
gyro_pitch_input = (gyro_pitch_input * 0.7) + ((gyro_data_x / 65.5) * 0.3);       // Pitch input en dps aplicant un filtre complementari per reduir soroll.

// El setpoint del PID estarà en dps ja que volem que dependgui del pitch rebut del comandament RC.
// Dividim per 3.0 per ajustar que la velocitat màxima de rotació quan posem el joystick a un extrem sigui de 164dps ((500-8)/3 = 164dps).
// Si volguessim poder girar més ràpid o més lent hauriem de modficar aquest 3.
pid_pitch_setpoint = 0;
if (receiver_pitch > 1508) pid_pitch_setpoint = receiver_pitch - 1508;
else if (receiver_pitch < 1492) pid_pitch_setpoint = receiver_pitch - 1492;
pid_pitch_setpoint /= 3.0;                                                        // Dividim per 3 per ajustar la velocitat angular màxima.
```

**Figura 8. 4. 3: Càlcul setpoint, mode acrobàtic.**

Al ser un aficionat en portar quadcopters jo prefereixo començar a volar en mode estable que és més senzill. Llavors ara ens falta realitzar un control per tal de que la inclinació del nostre quadcopter es correspongui a la que rebem per el comandament RC.
Per fer això podríem fer un nou controlador PID que només s'encarregués de corregir la inclinació i llavors el seu valor resultant el tinguéssim en compte juntament amb el resultat del controlador PID anterior o podem mirar d'adaptar el controlador PID actual perquè també tingui en compte la inclinació. La primera alternativa segurament seria més senzilla i entenedora però ens perjudicaria en temps de càlcul, per tant optem per la segona.

Imaginem que el nostre quadcopter es troba amb una inclinació de 10 º i el nostre joystick es troba en el punt central (1500 microsegons), això vol dir que la inclinació desitjada és 0º. Si multipliquem aquest angle per una constant i aquest valor diem que és la correcció que hem de fer llavors passarà el següent:

- Multipliquem l'angle 10 º per una constant, per exemple, 15. Obtenim una correcció amb un valor de 150.
- Restem aquesta correcció al valor que rebem del comandament, ens queda un pols de 1350 microsegons. Això implica que l'angle començarà a disminuir.
- Quan arribem a una inclinació de 5 º, la correcció serà de 75. El pols de 1425 microsegons i, per tant, l'angle seguirà disminuint.
- Finalment quan arribem a una inclinació de 0 º, la correcció serà 0. El pols valdrà 1500 microsegons i el quadcopter no rotarà.

Anem a mirar que passa si el joystick no està a una posició central, el trobem a la meitat tirant cap a la dreta amb un valor de 1750 microsegons.

- Si l'angle d'inclinació és 0 º, la nostra correcció serà 0. El pols es mantindrà a 1750 i el quadcopter tornarà a rotar i guanyarà inclinació.

- Si l'angle d'inclinació és 5 º, la nostra correcció serà 75. El pols agafarà un valor de 1675 microsegons i el quadcopter seguirà rotant.
- Si l'angle d'inclinació és 16.7 º, la nostra correcció serà 250. El pols valdrà 1500 microsegons i el quadcopter pararà la seva rotació.

Sembla que aquesta tècnica ens pot funcionar, anem a mirar que passa quan implantem aquesta funcionalitat amb el càlcul del *setpoint* que teníem anteriorment. Imaginem que tenim el joystick en la posició anterior i rebem un valor de 1750 microsegons, el quadcopter per el contrari amb una inclinació de -10 º.
Fem els càlculs:

- La correcció de l'angle té un valor de -150.
- El *setpoint* inicialment té un valor de 242 (1750 – 1508).
- Si al *setpoint* li restem la correcció ens queda un valor de 392.
- Finalment limitem la velocitat dividint per 3.0 i tenim un *setpoint* final de 130. Estem indicant al dron que roti en sentit contrari d'on ho està fent i amb velocitat de 130 º/s.
- Simulem que el dron ha rotat a una velocitat de 130 º/s durant 0.1s, per tant ara es troba amb una inclinació de 3 º rotant a la velocitat de 130 º/s. El joystick es manté.
- Si tornem a calcular el *setpoint* ens surt que la correcció de l'angle val 45 i el *setpoint* final té un valor de 65 º/s. Per tant segueix rotant.
- Tornem a fer la mateixa simulació, ha mantingut aquesta velocitat durant 0.1s, es troba amb una inclinació de 9.5 º. El joystick es manté.
- La correcció val 142.5 mentre que el *setpoint* final té un valor de 33 º/s. Al cap de 0.1s es trobarà amb una inclinació de 12.8 º.

Amb aquest exemple veiem que el dron ha corregit ràpidament la seva inclinació negativa i va alentint la seva rotació per tal d'arribar als 16.7 º que hem calculat anteriorment.
Per acabar aquests exemple suposarem que ens trobem en l'estat anterior però ara posem el joystick en posició central ja que volem que el dron es mantingui amb una inclinació de 0 º.

- La correcció val 192, per tant el *setpoint* val -64 º/s. Al cap de 0.1s es trobarà amb una inclinació de 6.4 º.
- La correcció val 96, per tant el *setpoint* val -32 º/s. Al cap de 0.1s es trobarà amb una inclinació de 3.2 º.

Podem veure que funciona perfectament ja que també s'acosta a la inclinació 0 cada cop més a poc a poc, per tant ja tenim l'algoritme per calcular el nostre *setpoint*.

```
gyro_pitch_input = (gyro_pitch_input * 0.7) + ((gyro_data_x / 65.5) * 0.3);       // Pitch input en dps aplicant un filtre complementari per reduir soroll.
pitch_level_adjust = angle_pitch_output * 15;                                     // Calculem l'ajustament per compensar la inclinació en pitch.

// El setpoint del PID estarà en dps ja que volem que dependgui del pitch rebut del comandament RC.
// Dividim per 3.0 per ajustar que la velocitat màxima de rotació quan posem el joystick a un extrem sigui de 164dps ((500-8)/3 = 164dps).
// Si volguessim poder girar més ràpid o més lent hauriem de modficar aquest 3.
pid_pitch_setpoint = 0;
if (receiver_pitch > 1508) pid_pitch_setpoint = receiver_pitch - 1508;
else if (receiver_pitch < 1492) pid_pitch_setpoint = receiver_pitch - 1492;
pid_pitch_setpoint -= pitch_level_adjust;                                         // Restem l'ajustament de l'angle al setpoint.
pid_pitch_setpoint /= 3.0;                                                        // Dividim per 3 per ajustar la velocitat angular màxima.

calculate_pid();
```

**Figura 8. 4. 4: Càlcul setpoint, mode estable.**

Tot i que és poder una mica més complicat d'entendre, almenys sense mirar varis exemples com hem fet, podem veure que a nivell de programació només implica dos línies: multiplicar l'angle per la constant 15 i restar aquest ajustament del *setpoint*.
La constant amb valor 15 la podem modificar per canviar l'angle màxim que volem que s'inclini el nostre quadcopter al girar, amb el valor actual tenim un angle màxim de 33.33 º, si l'augmentem a 20 l'angle màxim serà de 25 º i si el reduïm a 10 l'angle màxim serà de 50 º. D'aquesta manera podem fer que sigui més o menys agressiu.

Un aspecte que ens falta comentar del PID és que en tots els exemples i càlculs que hem fet donem per suposat que els valors que rebem del comandament RC van dels 1000ms als 2000ms exactes, si comprovem la prova que em realitzat a l'apartat 8. 1 veurem que això no és així. Això ens pot perjudicar molt ja que poder ens trobarem que quan volem indicar que no hi hagi moviment en roll el valor rebut és 1490 i quan volem indicar que no hi hagi moviment en pitch el valor rebut és 1530. Al subapartat següent veurem com podem arreglar aquest problema.

## Calibració senyal RC

*- Projecte Arduino: Prod → Setup / Test → RCTest2*

Per tal de solucionar el problema anterior sobre les senyals RC rebudes haurem d'aplicar algun tipus de conversió perquè quan llegim el valor d'una senyal siguem capaços de convertir-lo a una escala que vagi de 1000 a 2000.
Si nosaltres sabem que una senyal de RC rebuda el seu valor mínim és 960 i el seu valor màxim 1935 i el seu valor mig és 1510 podem fàcilment projectar aquests valors dins una escala que vagi de 1000 a 2000 fent les operacions següents.

```
if (actual < center){                                                  // El valor llegit és més petit que el centre.
  if (actual < low) actual = low;                                      // El valor llegit no pot ser més petit que el valor mínim.
  difference = ((long)(center - actual) * (long)500) / (center - low); // Projectem el valor dins una escala de 1000 - 2000us.
  return 1500 - difference;                                            // Retornem el valor.
}
else if(actual > center){                                              // El valor llegit és més gran que el centre.
  if(actual > high)actual = high;                                      // El valor llegit no pot ser més gran que el valor màxim.
  difference = ((long)(actual - center) * (long)500) / (high - center);// Projectem el valor dins una escala de 1000 - 2000us.
  return 1500 + difference;                                            // Retornem el valor.
}
else return 1500;                                                      // Retornem el valor cèntric.
```

**Figura 8. 4. 5: Calibració senyal RC.**

Com veiem l'algoritme és senzill, l'única pega és que hem de conèixer el valor mínim, central i màxim de la senyal. Podem buscar aquests valors demanant a l'usuari que faci tots els moviments dels joysticks durant la configuració del controlador però si cada cop que volem utilitzar el dron hem de fer-ho és una mica inútil.
La solució es basa en utilitzar la memòria **EEPROM** d'Arduino que és una memòria no volàtil que podem utilitzar per guardar informació entre els reinicis de la nostra controladora.
Guardarem els valors mínims, centrals i màxims de les quatre senyals amb les que estem treballant per llavors recuperar aquests valors i aplicar l'algoritme de projecció a l'escala 1000 – 2000 microsegons.

Als codis adjuntats com annexos podem veure com guardar aquesta configuració i com em modificat el codi del test de l'apartat 8. 1 per veure que la projecció és fa correctament.

### Test

*- Projecte Arduino: Test → Test_PID_ONLY_PITCH*

Ja hem vist com realitzar el codi d'estabilització d'un PID i com calibrar les senyals rebudes del comandament RC, ara bé, hem de provar que el nostre PID funcioni correctament directament sobre el quadcopter?

Jo el que proposo és muntar una estructura com la que es pot veure a la Figura 8. 4. 6 on podem situar un motor a cada banda (un que giri CW i l'altre CCW amb les seves respectives hèlixs) i la IMU al centre per tal de provar un controlador PID amb només un eix. D'aquesta manera podem jugar amb les constants P, I i D per separat i veure les reaccions.

**Important: és el primer cop que muntem les hèlixs, ens hem d'assegurar que estiguin ben collades i que la nostra estructura també sigui robusta per tal de no patir cap accident.**

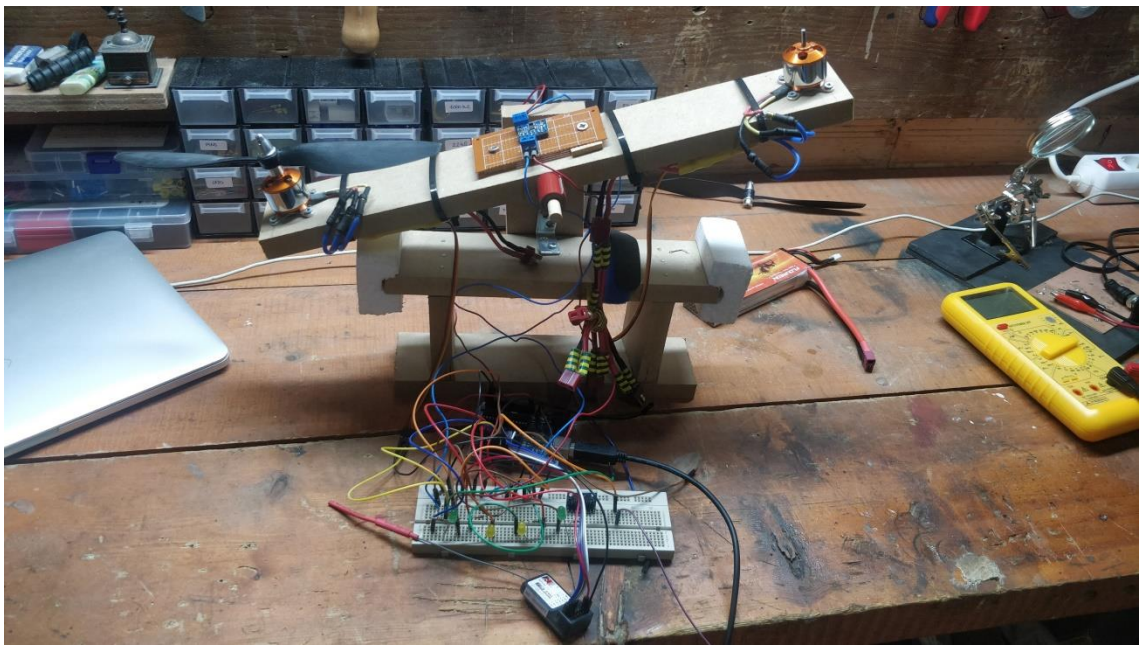


Figura 8. 4. 6: Estructura per provar el PID en un sol angle de rotació.

## 8. 5. Bateria

*- Projecte Arduino: Test → check_battery*

Els nostres motors tenen un valor de 1000KV, això implica que quan rebin el voltatge de 12.6v que és el màxim de la nostra bateria haurien de girar a 12600rpm mentre que quan anem perdent tensió la velocitat de gir també es veurà disminuïda. Com és evident si un motor gira

més lent farà menys força que el mateix motor girant més ràpid, és per això que necessitarem mesurar el nivell de la bateria des de el nostre codi per tal de compensar la caiguda de tensió que anirem patint. La funció que utilitzarem per mesurar el nivell de la bateria és el següent:

```
void check_battery() {
  float ar = analogRead(a0);

  if (ar > 0) {
    volt = (volt * 0.7) + ((ar * (5.00 / 1023.00) * 3.20) * 0.3);
  }
  else volt = 0;
}
```

**Figura 8. 5. 1: Llegir nivell de la bateria.**

Sabem que els pins d'entrada analògica divideixen el voltatge rebut entre 1023, és per això que agafem el valor llegit i el multipliquem per la divisió de 5V entre 1023, així obtenim el voltatge d'entrada. Aquest voltatge d'entrada al venir d'un divisor de tensió l'hem de multiplicar per el "multiplicador" del nostre circuit.
Aquest multiplicador serà igual a la divisió de la resistència total del circuit per la segona resistència del divisor, per tant serà 3.2 (3200 / 1000). Finalment a la figura anterior també podem veure que apliquem un filtre complementari per reduir el soroll.

# 9. Implantació i resultats

Ja hem vist com funciona i com podem programar cada un dels components, és el moment de realitzar el muntatge del quadcopter i desenvolupar el nostre controlador.

## 9. 1. Muntatge del quadcopter

Pel que fa al muntatge del quadcopter la forma com volem fer-ho, les connexions que volem utilitzar és una mica lliure, cadascú ho ha de fer com es senti més còmode i vegi més senzill. Jo deixo un seguit de recomanacions:

- Podem col·locar la IMU a qualsevol lloc del nostre quadcopter però per anar bé hauríem de mirar de fer-ho a un lloc on pugui rebre poques vibracions, com més al centre del quadcopter millor.
- Hem de tenir en compte que se'ns pot espatllar algun component com un motor o un ESC, és per això que hauríem d'utilitzar connexions que siguin robustes (que no es deixin anar amb facilitat) però que ens permetin canviar els components amb facilitat.
- Hem de col·locar la nostra IMU de forma que l'eix X miri cap a la part davantera del quadcopter i l'eix Y miri cap a la part dreta (en cas que no sigui així haurem de comprovar a quin eix hem assignat la rotació roll i la rotació pitch).

A les següents figures es pot observar el muntatge que he realitzat pas a pas:

**Figura 10. 1. 1: Inici muntatge del frame.**

Figura 10. 1. 2: Connexions dels ESC a la placa de distribució de potencia i motors muntats.

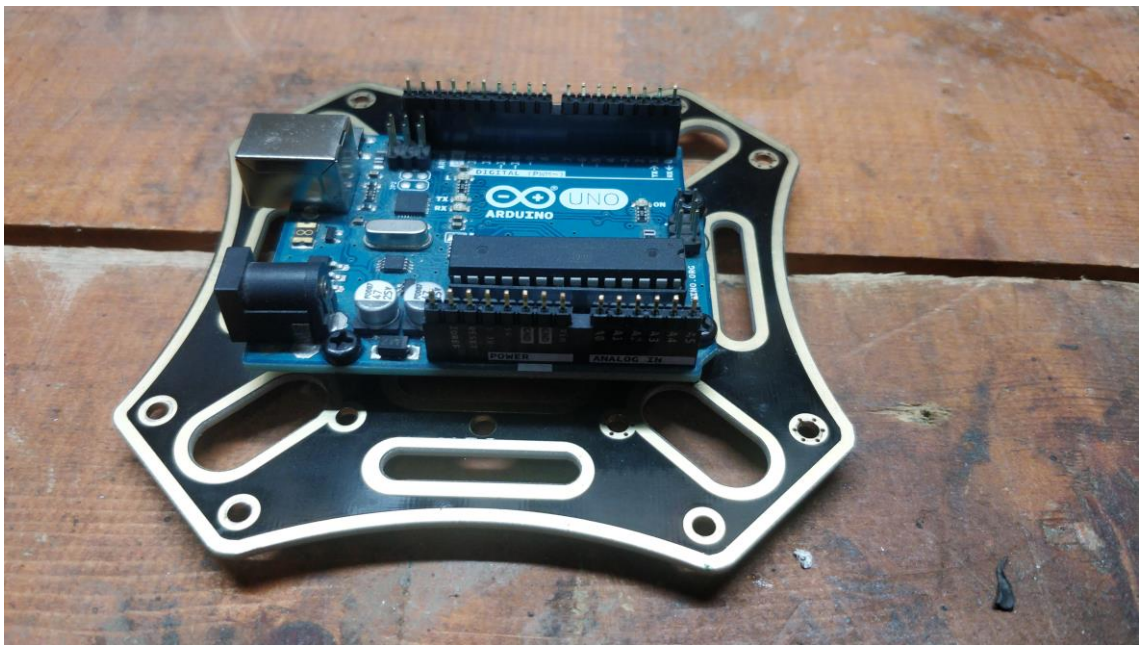



Figura 10. 1. 3: Muntatge de la placa Arduino al frame.

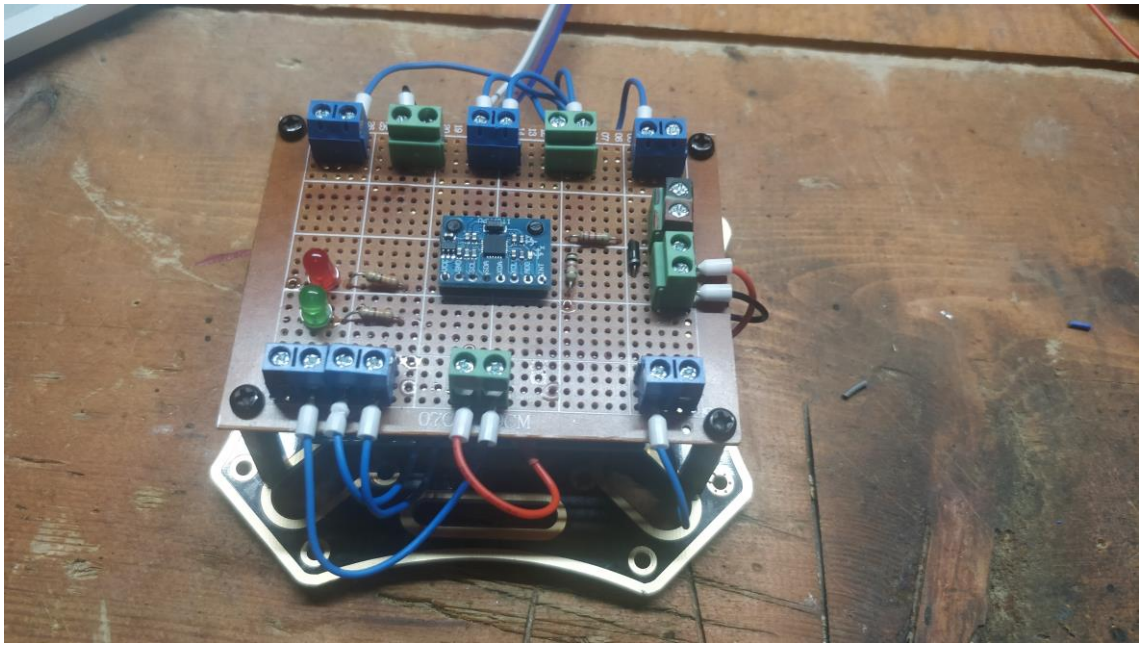


Figura 10. 1. 4: IMU, divisor de tensió, leds i connectors soldats a un PCB i muntats damunt l'Arduino.

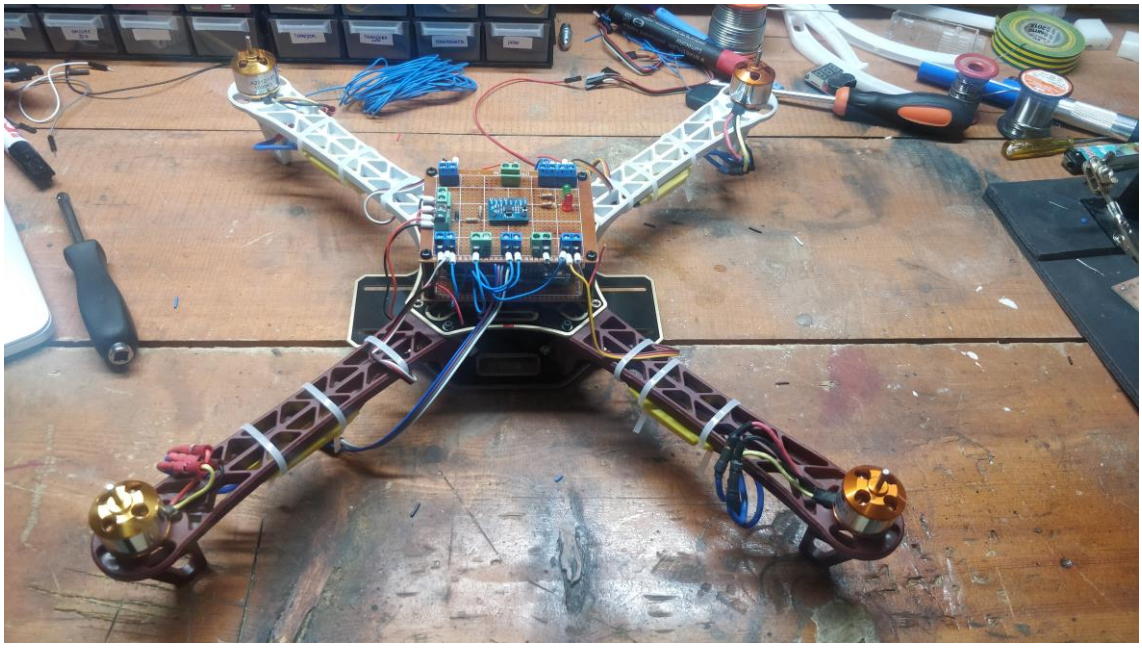


Figura 10. 1. 5: Ajuntar la part superior del frame a les potes.

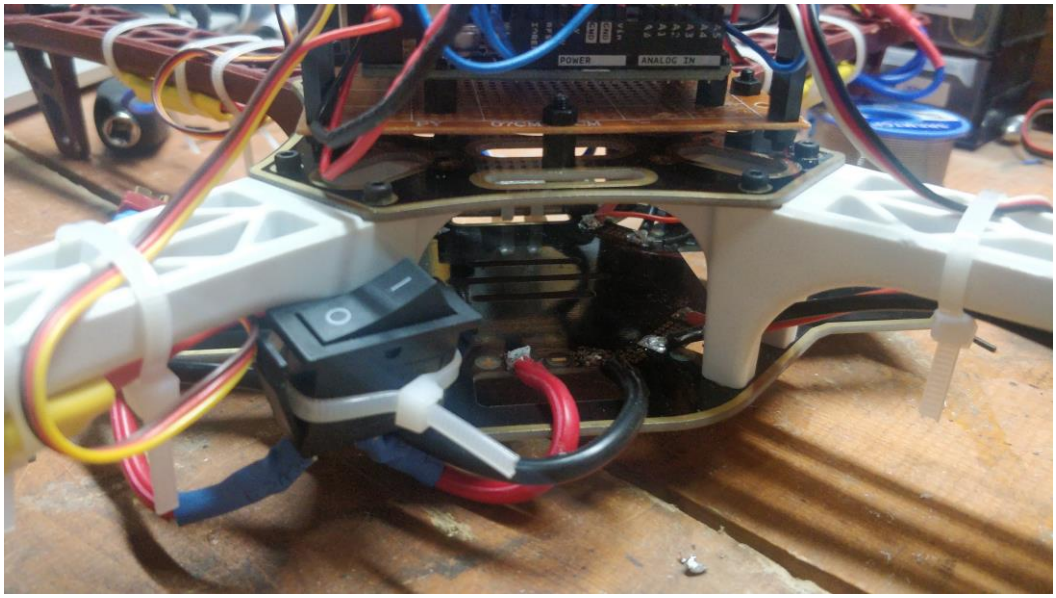



Figura 10. 1. 6: Botó que ens permet engegar / parar el dron.

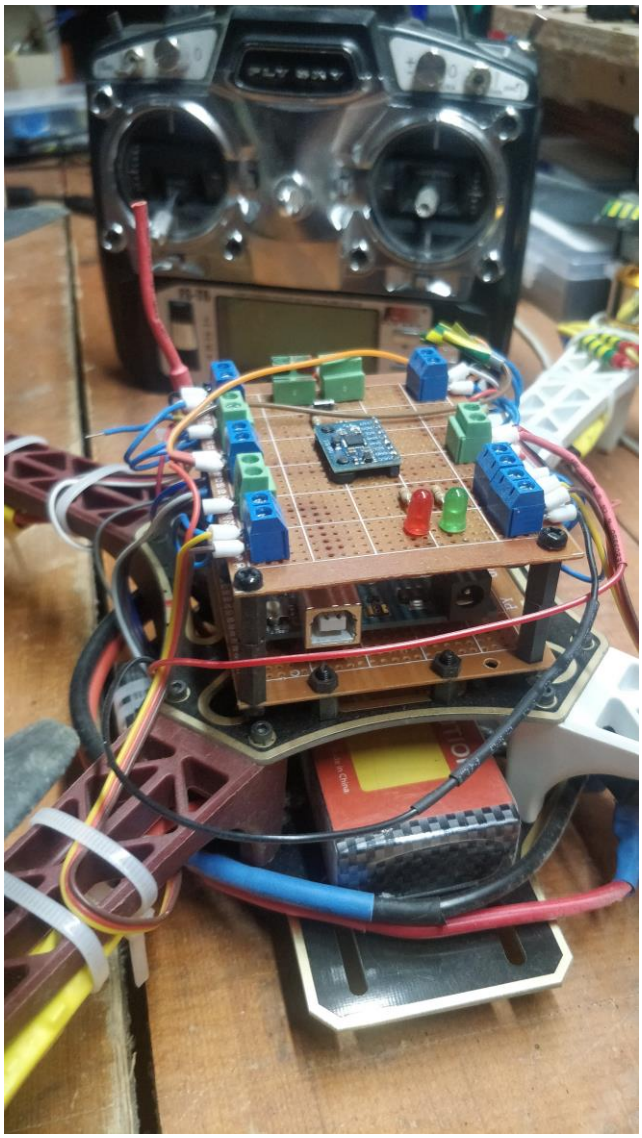



Figura 10. 1. 7: Muntatge de la bateria a la part inferior.

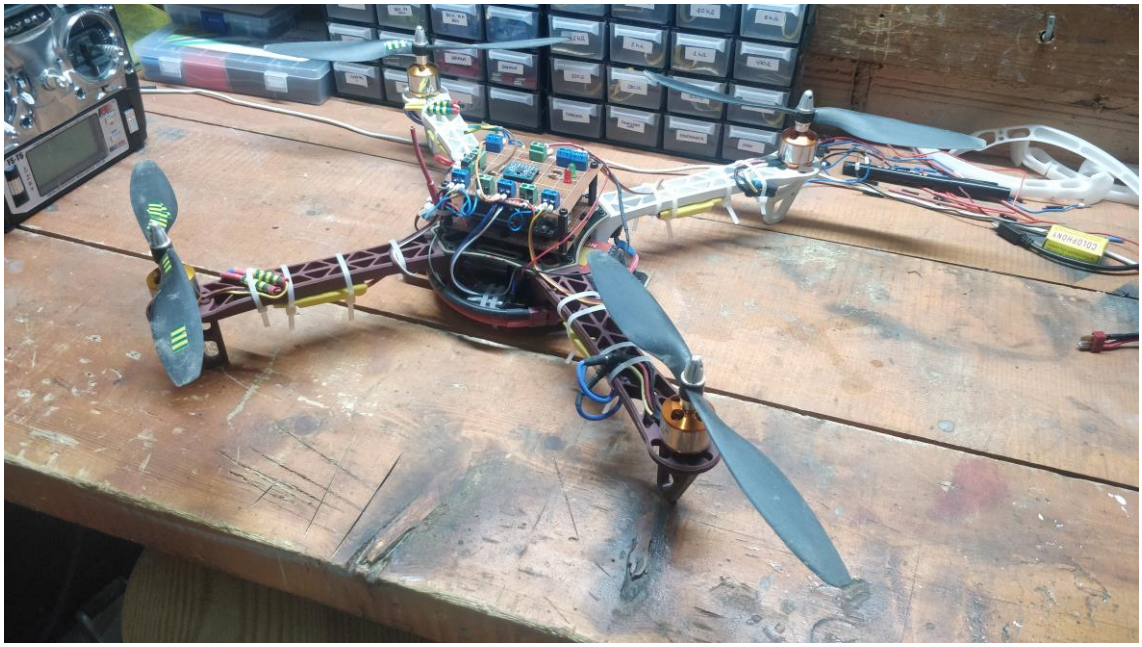


Figura 10. 1. 8: Muntatge del quadcopter finalitzat.

### Calibració dels quatre motors

*- Projecte Arduino: RCTest_ESC*

Un cop realitzat el muntatge del quadcopter hauríem de calibrar els quatre ESC a la vegada i comprovar que comencen a girar al mateix moment un cop calibrats. Això podem fer-ho amb un projecte semblant al que hem utilitzat a l'apartat 8. 3 però enviant la senyal als quatre ESC enlloc de només un.

**Important: En aquest punt desmuntarem les hèlixs, no les tornarem a posar fins que no haguem provat el controlador i veiem que ens dona els resultats esperats.**

## 9. 2. Controlador

*- Projecte Arduino: Prod → Controller*

Per començar a ajuntar les proves que hem fet a l'apartat anterior amb els diferents components i desenvolupar el controlador que ens permetrà fer volar i controlar el nostre quadcopter en mode estable hem de tenir present totes les funcionalitat que ha de fer, són les que hem dissenyat a l'apartat 7. 2.

Tot programa d'Arduino té dos parts, el *setup* on aniran les nostres configuracions i el *loop* on anirà el nostre programa principal (aquesta funció ha de tenir una freqüència de 250Hz). A més a més al utilitzar la PCINT també tindrem la funció d'interrupció.

Comencem definint quines funcionalitats hem de fer dins el *setup* per realitzar les configuracions necessàries perquè els components funcionin correctament:

- Definirem la interrupció PCINT als pins D8, D9, D10 i D11 per tal de llegir les senyals que ens arriben del comandament RC.
- Configurarem els pins D3, D4, D5, D6, D7, D12 i D13 com a *outputs*. El primer l'utilitzarem per comprovar amb l'oscil·loscopi la freqüència del nostre *loop*. Els altres són per els ESC i dos leds.
- Llegirem la informació guardada a la EEPROM per tal de poder calibrar la senyal RC.
- Farem la configuració de la IMU.
- Calibrarem el giroscopi.

Podem veure que en algunes parts d'aquest codi enviem una senyal de 1ms als ESC, això ho fem perquè si no reben cap senyal van emetent un *"pip"* i és una mica molest. Durant el procés de calibració del giroscopi he fet que s'encengui el led vermell de forma intermitent per avisar que no hem de moure el quadcopter.

Un cop tenim la part de configuració ens podem posar amb el programa principal, el més important és que la lectura dels valors de la IMU sigui cada 4ms ja que sinó els càlculs que realitzem per calcular l'angle d'inclinació no serien correctes (depèn de on fem la lectura de la IMU podria passar que algun procés anterior durés més o menys en alguns casos i per tant la lectura no es fes cada 4ms), és per això que aquesta funció serà la primera, a més a més necessitem les dades de la IMU per fer càlculs posteriors. Les funcions que hauria de fer el nostre controlador serien les següents:

1. Esperem que el *loop* porti 4m per començar un nou cicle.
2. Llegir dades IMU.
3. Calibrar senyals del comandament RC (només calibrar ja que de llegir-les se'n encarrega la interrupció).
4. Calcular l'angle del quadcopter.
5. Calcular *setpoint* del PID.
6. Calcular PID.
7. Llegir tensió bateria i engegar leds segons el nivell de càrrega.
8. Definir els valors dels quatres ESC.
9. Enviar polsos ESC.

D'aquesta manera ja tindríem un controlador que ens funcionaria, tot i així podem aplicar algunes millores que ens seran de gran ajuda com:

- Poder "apagar" i "engegar" el quadcopter a través del comandament RC, és a dir, que fent un moviment amb el joystick quan tenim el nostre quadcopter a terra puguem indicar que el nostre programa comença, i també que amb un altre moviment acabi. Pot semblar una tonteria però és de gran utilitat ja que d'aquesta manera pots parar i engegar els motors quan vols i, en cas que al provar el controlador vegis que el quadcopter fa moviments estranys quan s'està començant a enlairar pots parar els motors abans de que passi algun accident.

  **Hem de tenir en compte que el controlador PID començarà a funcionar només d'engegar el programa, com que la potència dels motors quan estan al mínim no és**

**suficient per enlairar el quadcopter aquest no varia la seva inclinació i la part integral del PID anirà sumant l'error i fent que els motors pugin de velocitat. Abans que passi això nosaltres hem de provar d'enlairar** i, si falla, és de gran utilitat poder parar els motors i tornar-ho a provar (també evitarem trencar hèlixs).

Quan donem la ordre d'engegar els motors assignarem als angles pitch i roll els valors que provenen de l'acceleròmetre i reiniciarem els valors de memòria del PID (per la integral i la derivada). Mentre els motors no estan engegats enviarem un pols de 1ms als ESC perquè no facin soroll.

El procés que farem per engegar els motors és moure el joystick esquerra a la part inferior esquerra i tornar a la part inferior central. Llavors per parar-lo farem el moviment cap a la part inferior dreta.

```
void check_start() {
  // Per engegar els motors: accelerador al mínim i yaw a l'esquerra (pas 1).
  if (throttle < 1050 && receiver_yaw < 1050) start = 1;

  // Quan el yaw torna a estar a la part central engeguem (pas 2).
  if (start == 1 && throttle < 1050 && receiver_yaw > 1450) {
    start = 2;
    digitalWrite(GREEN, HIGH);

    // Definim els angles amb els valors de l'acceleròmetre.
    angle_roll_gyr = angle_roll_acc;
    angle_pitch_gyr = angle_pitch_acc;
    angle_roll_output = angle_roll_acc;
    angle_pitch_output = angle_pitch_acc;

    // Reiniciem les variables del PID.
    pid_i_mem_roll = 0;
    pid_last_roll_d_error = 0;
    pid_i_mem_pitch = 0;
    pid_last_pitch_d_error = 0;
    pid_i_mem_yaw = 0;
    pid_last_yaw_d_error = 0;
  }

  // Si els motors estan engegats i posem l'accelerador al mínim i el yaw a la dreta els parem.
  if (start == 2 && throttle < 1050 && receiver_yaw > 1950) start = 0;
}
```

**Figura 10. 2. 1: Procés d'engegar / parar els motors.**

- El nostre programa es divideix en dos blocs de 2ms cada un: per una part tots els càlculs a realitzar i per l'altre l'enviament dels polsos als ESC. Ara bé, el pols mínim que enviarem sempre serà de 1ms, això és molt de temps per esperar sense fer res i podríem aprofitar-ho per fer algun càlcul que ens ocupés força temps, d'aquesta manera estaríem més segurs que el nostre *loop* no excedeix els 4ms.

  Si comprovem amb l'oscil·loscopi quins càlculs consumien més temps veurem que la lectura de l'IMU ens ocupa uns 600 microsegons, per tant podríem fer-ho mentre enviem el pols ESC.

  Al fer aquest canvi hem de modificar la línia on ens esperem a que el loop porti 4ms per començar un altre cicle ja que, com hem dit abans, la lectura de dades de la IMU és el que s'ha de fer primer.

```
// Loop de 250Hz (4ms).
while (zero_timer + 4000 > micros());
zero_timer = micros();

PORTD |= B00001000;                                      // Posem el pin D3 a HIGH per l'oscil·loscopi.
PORTD |= B11110000;                                      // Posem els pins D4, D5, D6, D7 a HIGH per començar a enviar la senyal.
timer_esc_1 = esc_1 + zero_timer;                        // Calculem el temps final per l'ESC 1.
timer_esc_2 = esc_2 + zero_timer;                        // Calculem el temps final per l'ESC 2.
timer_esc_3 = esc_3 + zero_timer;                        // Calculem el temps final per l'ESC 3.
timer_esc_4 = esc_4 + zero_timer;                        // Calculem el temps final per l'ESC 4.

// Tenim sempre almenys 1ms on no fem res. Aprofitem-ho.
read_IMU_data(true);                                     // Llegim les dades de la IMU.

while (PORTD >= 16) {                                    // Mentre els ports D4, D5, D6 o D7 estiguin a HIGH.
  esc_loop_timer = micros();                             // Comprovem el temps actual.
  if(timer_esc_1 <= esc_loop_timer)PORTD &= B11101111;   // Quan el temps del ESC 1 ha finalitzat: D4 - LOW.
  if(timer_esc_2 <= esc_loop_timer)PORTD &= B11011111;   // Quan el temps del ESC 2 ha finalitzat: D5 - LOW.
  if(timer_esc_3 <= esc_loop_timer)PORTD &= B10111111;   // Quan el temps del ESC 3 ha finalitzat: D6 - LOW.
  if(timer_esc_4 <= esc_loop_timer)PORTD &= B01111111;   // Quan el temps del ESC 4 ha finalitzat: D7 - LOW.
}
```

**Figura 10. 2. 2: Lectura de la IMU mentre enviem el pols PWM als ESC.**

Amb aquestes dos modificacions el flux del nostre controlador quedaria de la següent forma:

1. Calibrar senyals del comandament RC (només calibrar ja que de llegir-les se'n encarrega la interrupció).
2. Calcular l'angle del quadcopter.
3. Comprovar si s'ha donat l'ordre de parar / engegar els motors.
4. Calcular *setpoint* del PID.
5. Calcular PID.
6. Llegir tensió bateria i engegar leds segons el nivell de càrrega.
7. Definir els valors dels quatres ESC.
8. Esperem que el *loop* porti 4m per començar un nou cicle.
9. Enviar polsos ESC i llegir dades IMU.

```
convert_receiver_inputs();
calculate_angles();
check_start();
calculate_pid_setpoints();
calculate_pid();
read_battery_voltage(true);
show_battery_state();
set_esc_values();

PORTD &= B11110111;                                      // Posem el pin D3 a LOW per l'oscil·loscopi.

// Encenem el led vermell si el nostre loop excedeix els 4050 microsegons.
if(micros() - zero_timer > 4050) digitalWrite(RED, HIGH);

// Loop de 250Hz (4ms).
while (zero_timer + 4000 > micros());
zero_timer = micros();

PORTD |= B00001000;                                      // Posem el pin D3 a HIGH per l'oscil·loscopi.
PORTD |= B11110000;                                      // Posem els pins D4, D5, D6, D7 a HIGH per començar a enviar la senyal.
timer_esc_1 = esc_1 + zero_timer;                        // Calculem el temps final per l'ESC 1.
timer_esc_2 = esc_2 + zero_timer;                        // Calculem el temps final per l'ESC 2.
timer_esc_3 = esc_3 + zero_timer;                        // Calculem el temps final per l'ESC 3.
timer_esc_4 = esc_4 + zero_timer;                        // Calculem el temps final per l'ESC 4.

// Tenim sempre almenys 1ms on no fem res. Aprofitem-ho.
read_IMU_data(true);                                     // Llegim les dades de la IMU.

while (PORTD >= 16) {                                    // Mentre els ports D4, D5, D6 o D7 estiguin a HIGH.
  esc_loop_timer = micros();                             // Comprovem el temps actual.
  if(timer_esc_1 <= esc_loop_timer)PORTD &= B11101111;   // Quan el temps del ESC 1 ha finalitzat: D4 - LOW.
  if(timer_esc_2 <= esc_loop_timer)PORTD &= B11011111;   // Quan el temps del ESC 2 ha finalitzat: D5 - LOW.
  if(timer_esc_3 <= esc_loop_timer)PORTD &= B10111111;   // Quan el temps del ESC 3 ha finalitzat: D6 - LOW.
  if(timer_esc_4 <= esc_loop_timer)PORTD &= B01111111;   // Quan el temps del ESC 4 ha finalitzat: D7 - LOW.
}
```

**Figura 10. 2. 3: *Loop* del nostre controlador.**

## Test del controlador

Un cop desenvolupat el controlador els valors de les constants del PID encara no les sabem, les buscarem més endavant, de moment podem posar valor 1 a totes les constants P i 0 a les D i I ja que abans hem de fer varies comprovacions per assegurar-nos que el quadcopter respon correctament, per fer-ho utilitzarem el monitor *Serial* d'Arduino assegurant-nos que en cap moment el nostre programa excedeix els 250Hz de freqüència (podem fer un print diferent a cada cicle per no excedir-nos en el temps).

Les comprovacions a fer són les següents:

1. Comprovarem que rebem les senyals correctes del nostre RC i que aquestes estan ben calibrades (els valors estan entre 1000 i 2000 microsegons).
2. Els motors es poden engegar i parar mitjançant el comandament RC.
3. Els angles roll i pitch s'han de calcular correctament.
4. El valor dels quatre ESC s'ha de correspondre amb els moviments que simulem al quadcopter, farem dos tipus de proves:
    a. Comprovar que els ESC agafin el valor correcte quan donem ordres amb el comandament RC.
    b. Comprovar que, deixant el comandament RC amb tots els controls al valor mig (1500 microsegons), si inclinem el quadcopter aquest augmenta o disminueix el valor d'alguns ESC per compensar correctament la inclinació.
5. Comprovar que es llegeix correctament el nivell de la bateria i es fa bé la compensació dels motors.

Un cop hem comprovat tot això és el moment de muntar les hèlixs als motors, tenint en compte de muntar la hèlix CW als que giren de forma horària i la CCW als que giren de forma antihoraria, i fer una última prova. ´

Un cop el quadcopter hagi calibrat el giroscopi l'agafarem amb una ma per la zona central vigilant molt no fer-nos mal amb les hèlixs (recomanable portar guants i ulleres de protecció) i engegarem els motors, incrementarem l'acceleració fins que notem que el quadcopter no pesa tan i ja començaria a volar per si sol i fem les mateixes comprovacions que hem fet en el punt 4 anterior:

- Comprovem que el quadcopter respon als diferents moviments que li indiquem amb el nostre comandament RC (pitch, roll i yaw en totes direccions).
- Comprovem que, sense indicar-li cap moviment, quan l'inclinem en diferents direccions els motor intenten compensar aquesta inclinació.

Si aquests dos passos són correctes podem passar a l'últim pas del nostre projecte, en cas contrari tornem a treure les hèlixs i repassem un altre cop les comprovacions que hem dit anteriorment.

## Constants PID

Buscar els valors de les constants del nostre PID pot ser llarg, hem de tenir paciència i fixar-nos molt en el que fem, d'aquests valors dependrà que el nostre quadcopter voli correctament o no. Si busquem per internet trobarem vaires maneres que buscar aquests paràmetres ja que no hi ha una fórmula matemàtica per buscar-los sinó que cadascú ho fa com a ell li sembla millor. A continuació explicaré la manera que he utilitzat jo i que m'ha servit:

1. Reiniciarem tots els valors a 0 i posarem la P-yaw a 3 i la I-yaw a 0.02. D'aquesta manera evitarem que el quadcopter roti en yaw quan estiguem buscant les constants per el roll i pitch (les constants per aquests dos eixos seran les mateixes).
2. Totes les proves que farem a continuació seran aguantant el quadcopter amb la mà i amb l'accelerador al punt que notem que el quadcopter començaria a volar. Llavors farem moviments en roll i pitch per tal de notar la seva reacció.
    a. Incrementarem la constant D de tres en tres fins el punt que notem que el quadcopter és molt agressiu i comença a vibrar.
    b. Disminuïm la constant D d'un a un fins que torni a anar suau. Ens quedarem el 75% d'aquest valor.
    c. Aquest seria el valor D que ens quedaríem.
3. Les següents proves ja les farem amb el quadcopter a terra, si pot ser sobre un lloc tou millor per si ens falla alguna cosa evitar que es trenqui res.
    a. Incrementarem la constant P en intervals de 0.2, de seguida veurem que el quadcopter ja comença a volar de forma correcte, anirem incrementant fins que veiem que es sobrecompensa ( no està mai estable ja que les seves reaccions són massa brusques).
    b. Quan trobem aquest límit on es comença a sobrecompensar reduirem la constant P un 50%.
    c. Incrementarem la constant I en intervals de 0.01, hauríem d'observar que el quadcopter cada cop vola d'una manera més estable, seguirem incrementant fins que veiem que es torna a sobrecompensar (no serà tan exagerat com en la constant P sinó que serà un moviment més de oscil·lació).
    d. Quan trobem aquest punt reduirem la constant un 50%.
    e. Aquest serà el valor I que ens quedarem.
    f. Tornarem a incrementar la constant P en intervals de 0.2 fins que ens trobem que es torna a sobrecompensar.
    g. Disminuïm la constant P una mica fins que deixi de oscil·lar i es mantingui estable.
    h. Aquest serà el valor P que ens quedarem.

**Finalment ja tenim el nostre quadcopter apunt per volar**!

A partir d'aquí farem proves amb el quadcopter per mirar les seves reaccions i fer les variacions en les constants del PID que considerem oportunes. Les constants del yaw és possible que no les haguem de tocar ja que la d'estabilització en yaw només es forma les per les hèlixs, és per això que no ens cal la constant derivativa.

**Figura 10. 2. 4: Quadcopter finalitzat.**

# 10. Conclusions

Com a conclusions d'aquest projecte podem estar satisfets ja que s'han assolit tots els requisits que s'estableixen a l'apartat 5. Hem après quin és el funcionament bàsic d'un quadcopter i com funciona cada component per separat. Finalment hem estat capaços de muntar el dron correctament, desenvolupar el seu controlador i trobar els paràmetres del controlador PID que ens permeten que voli de forma estable i controlada.

Finalment seguint aquesta memòria podem aconseguir un dron per un preu inferior a 200€ que està realment bé i val la pena si llavors volem realitzar-hi millores i personalitzacions com afegir una càmera, fer algun canvi de hardware, afegir funcionalitats de software, etc. En cas que no sigui així podem trobar altres drons al mercat per un preu similar que tenen funcionalitats semblants al que acabem de desenvolupar i que realment poden ser una bona opció segons quina sigui la nostra finalitat.
Cal remarcar que en tots els casos (tan si fem el nostre propi dron com si en comprem un) és molt important que ens informem i prenem consciència de la legislació vigent sobre drons ja que molta gent no té constància dels llocs on es pot i no es pot fer volar un dron i posen en perill a persones i béns materials.

Aquest projecte també venia en part motivat a que a l'assignatura de Sistemes Empotrats que es realitza a quart d'Enginyeria Informàtica un company i jo vam provar de fer el muntatge i desenvolupament d'un quadcopter ja que no ens semblava tan complicat com esperàvem, finalment per la falta de temps i per haver subestimat la feina que realment comportava no vam aconseguir-ho. Això em va fer pensar que seria un bon projecte per fer com a Treball de Final de Grau.

La planificació inicial explicada en l'apartat 3 es va veure afectada per l'alertà sanitària viscuda des de el març ja que les dates d'entrega de components es van endarrerir i les botigues d'electrònica estaven tancades. Això sumat a altres motius personals també deguts a l'alerta sanitària han provocat l'endarreriment en l'entrega d'aquest projecte.

# 11. Treball futur

Aquest projecte es basa en aprendre cada component d'un dron com funciona i com podem desenvolupar el controlador. És per això que el treball futur que ens dona és moltíssim, podem canviar alguns components com provar altres motors, provar un frame més petit... També podem modificar el codi del nostre controlador per tal que la conducció del nostre quadcopter sigui més agressiva o menys.

A més a més de realitzar canvis sobre el ja desenvolupat en aquest treball podem afegir moltes funcionalitats a un quadcopter, jo tinc pensades les següents:

- Afegir el que és coneix com una càmera FPV (First Person View). Són unes càmeres que funcionen amb radiofreqüència i, un cop instal·lades al nostre dron, ens permeten visualitzar a un receptor de vídeo la imatge de la càmera. D'aquesta forma podem controlar el dron en primera persona. Com que funcionen per radiofreqüència la latència que tenen és molt baixa. Aquesta millora és molt fàcil de fer ja que només hem de donar tensió a la càmera perquè funcioni.


**Figura 12. 1: Càmera FPV amb receptor que ens permet visualitzar la imatge al mòbil.**

- Programació d'un OSD (On-Screen Display). Si fem la millora anterior podem també afegir un OSD que ens permetrà enviar informació i mostrar-la superposada a la imatge de la càmera. D'aquesta manera podem informar sobre el nivell exacte de bateria, la inclinació del nostre quadcopter, etc.
- Afegir un sensor de pressió atmosfèrica que ens permeti afegir una funcionalitat de control d'altitud. Ara mateix el nostre dron es mostra estable pel que fa a inclinació però no en altitud, quan volem anar endavant ens trobem que hem d'augmentar l'acceleració també perquè sinó, com que el quadcopter s'inclina, comença a baixar mentre tira endavant, això fa que el seu control sigui una mica més complicat.
  Afegit un control de pressió atmosfèrica podem saber la variació d'alçada del quadcopter i ajustar l'acceleració automàticament perquè es mantingui a una alçada concreta quan, per exemple, posem el joystick de l'acceleració en posició mitja.

- Afegir un control GPS al nostre quadcopter per tal de poder mira la ruta que hem fet, que segueixi el nostre telèfon mòbil a través d'una aplicació o poder indicar-li quina ruta a de realitzar i que la faci de forma autònoma. Aquesta modificació, evidentment, portarà força feina de desenvolupament.

També tinc pensat realitzar un quadcopter amb una mida més petita ja que el que hem desenvolupat en aquest projecte és bastant gros i difícil de transportar a llocs.

Totes les millores que vagi fent sobre el meu controlador les podreu trobar al repositori públic: https://bitbucket.org/qpascualsola/quadcopter

A més a més de les millores a nivell de hardware i software un projecte futur que es podria realitzar és fer un kit educatiu on s'incloguin els diferents components d'aquest projecte, aquesta memòria i el software desenvolupat per tal de posar-lo a disposició de instituts, universitats, entitats que realitzen cursos o, fins i tot, per tal de comercialitzar-lo i extreure'n un benefici.

# 12. Bibliografia

Tota la informació explicada en aquest projecte està extreta dels següents treballs, tesis i pàgines web. És per això que vull agrair a totes les persones que han realitzat aquesta documentació.


1. LEGASA MARTÍN-GIL, X. (2012). *Cuadrícoptero Arduino por control remoto Android.* Projecte Final de Carrera. Barcelona: Facultat de Informàtica de Barcelona, Universitat Politècnica de Catalunya.

2. JIINEC T. (2011). *Stabilization and control of unmanned quadcopter*. Tesis de Màster. Praga: Facultat d'Enginyeria Elèctrica, Universitat Técnica de Praga.

3. ETXEBERRIA MENDEZ J. y GOICOECHEA FERNANDEZ J. (2015). *Implementació de un dron cuadricóptero con Arduino.* Treball de Final de Grau. Pamplona: Escola Técnica Superior d'Enginyeria Industrial, Informàtica i Telecomunicacions.

4. GASCÓN IGLESIAS, C. (2016). *Ús de vehicles aeris no tripulats (UAV) en àmbits forestals.* <http://institutmollerussa.com/wp-content/uploads/2016/04/%C3%9As-de-vehicles-aeris-no-tripulats-en-%C3%A0mbits-forestals-2015.pdf>

5. ARDUPROJECT. *Como hacer un drone con Arduino, paso a paso*. < https://arduproject.es/>

6. JOOP BROOKING. *Project YMFC-AL – The Arduino auto-level quadcopter.* <http://www.brokking.net/ymfc-al_main.html>

7. ARDUINO. <https://www.arduino.cc/>

8. ATMEL. *ATmega328p Datasheet*. <http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf>

9. IVENSENSE. *MPU-6050 Product Specification.* <https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf >

10. IVENSENSE. *MPU-6050 Register Mapa and Descriptions.* <https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf>

11. CIRCUITO.IO BLOG. *An introduction to Arduino Uno pinout*.
<https://www.circuito.io/blog/arduino-uno-pinout/>

12. PYME AL DIA. *15 Aplicacions dels drons a la nostra societat*.
<https://www.pimealdia.org/es/15-aplicacions-dels-drons-a-la-nostra-societat/>

13. NOVODRONE. *Tipos de drones.*
<https://novodrone.com/tipos-de-drones/>

14. TIERRA DE DRONES. *Nueva ley de drones en España 2020.*
<https://www.tierradedrones.com/nueva-ley-de-drones-en-espana/>

15. ONEAIR. *Normativa de drones en España 2020.*
<https://www.oneair.es/normativa-drones-espana-aesa/>

16. MOBUS. *Motors brushles para drones, todo lo que necesitas saber.*
<https://mobus.es/blog/motores-brushless-para-drones-todo-lo-que-necesitas-saber/>

17. MOBUS. *ESC para drones, ¿para qué sirven?*
<http://mobus.es/blog/esc-para-drones-para-que-sirven/>

18. PROMOTEC. *Lo que hay que saber para elegir una bateria LiPo.*
<https://www.prometec.net/elegir-bateria-lipo/>

19. PROMOTEC. *Lo que hay que saber para elegir las hèlices para un cuadracóptero.*
<https://www.prometec.net/elegir-helices-dron/>

20. COCHES RC ONLINE. *¿Como cargar una bateria LiPo?*
<https://cochesrc.online/como-cargar-una-bateria-lipo/>

21. LUIS LLAMAS. *Conectar una emisora radio control con Arduino.*
<https://www.luisllamas.es/conectar-emisora-radio-control-con-arduino/>

22. LUIS LLAMAS. *Determinal la orientación con Arduino y el IMU MPU-6050.*
<https://www.luisllamas.es/arduino-orientacion-imu-mpu-6050/>

23. LUIS LLAMAS. *Qué son y cómo usar interrupcions en Arduino*.
<https://www.luisllamas.es/que-son-y-como-usar-interrupciones-en-arduino/>

24. LUIS LLAMAS. *Interrupciones en todos los pines de Arduino con la PCINT.*
<https://www.luisllamas.es/interrupciones-en-todos-los-pines-de-arduino-con-pcint/>

25. LUIS LLAMAS. *Guardar variables entre reinicios con Arduino y la memòria no volatil EEPROM.*
<https://www.luisllamas.es/guardar-variables-entre-reinicios-con-arduino-y-la-memoria-no-volatil-eeprom/>

26. LUIS LLAMAS. *Calculadora divisor de tensión*.
<https://www.luisllamas.es/calculadora-divisor-de-tension/>

27. AREATECNOLOGIA*. Divisor de tensión.*
<https://www.areatecnologia.com/electronica/divisor-de-tension.html>

28. RINCON IGENIERIL. *Que es PWM y para que sirve.*
<https://www.rinconingenieril.es/que-es-pwm-y-para-que-sirve/>

29. ROBOLOGS. *Programación de un ESC con Arduino.*
<https://robologs.net/2016/02/01/programacion-de-un-esc-con-arduino/>

30. INSTRUMENTACIÓN Y CONTROL. *Resumen P, I, D: lo justo y necesario que debes saber (y que nunca entendiste).*
<https://instrumentacionycontrol.net/resumen-p-i-d-lo-justo-y-necesario-que-debes-saber-y-que-nunca-entendiste/>

31. LISERGIO-Z. *Añade un control de las baterias a tus proyectos.*
<https://lisergio.wordpress.com/2014/05/20/anade-un-control-de-las-baterias-a-tus-proyectos/>

# 13. Annexos

## 13. 1. Diagrama de pins d'Arduino Uno.