

T.C.
ERCİYES ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ

VERTICAL TAKING OFF & LANDING (VTOL)

Hazırlayan

**Ömer Can VURAL
1030516774**

Danışman

Dr.Öğr.Üyesi Fehim KÖYLÜ

Bilgisayar Mühendisliği Bölümü

Design Project Final Raporu

**ŞUBAT 2022
KAYSERİ**

1. GİRİŞ.....	3
1.1 Problemin Tanımı.....	3
1.2 Çalışmanın Amacı.....	3
1.3 Projenin Tanımı.....	4
1.4 Çalışma Kapsamı	5
1.5 Çalışmanın Gerçekçi Kısıtlar Açısından Analizi	6
2. MATEMATİKSEL YÖNTEM VE TASARIM	7
2.1 Yöntem Hakkında Genel Bilgi	7
2.2 Tasarım.....	8
2.2.1 Mekanik	8
2.2.2 Elektronik.....	12
2.2.3 Yazılım.....	14
3. UYGULAMA ÇALIŞMALARI.....	15
3.1 Uygulamada Kullanılan Araç ve Gereçler.....	15
3.2 Uygulamada Gelinen Nokta ve Prototipleme	17
3.2.1 Fiziksel Model ve Elektronik	17
3.2.2 Yazılım.....	27
3.2.2.1 STM32CubeIDE üzerinde Konfigürasyonların Yapılması	29
3.2.2.2. MPU6050 Gyro Sensöründen Verilerin Okunması	40
3.2.2.3 PID ile Hata Katsayılarının Programa Dahil Edilmesi	63
3.2.2.4 ADC Okuma ve PWM Pulse Değerlerini Ayarlama.....	67
3.2.2.5 Harici Interrupt ile Modlar Arası Geçiş	70
3.2.2.5 RC Alıcı ile STM Kontrol Denemesi	73
4. KAYNAKLAR.....	81

1. GİRİŞ

1.1. Problem Tanımı

İnsansız hava araçları, ülkeler için savunma sanayinde önemli bir kuvvet göstergesidir ve aktif olarak savaş alanlarında kullanılmaktadır.(TB2, Harob Drone vb.). İnsansız hava araçlarının bu derece önemli olmasının sebebi savaş uçaklarına göre daha ucuz, hafif ve gözden çıkarılabilir olmalarıdır. Ancak buna rağmen yine de kalkış için belli koşullara ihtiyaç duyarlar ve taşınmaları kolay değildir. Bu bağlamda hem ülkemizin en büyük eksikliklerinden olan uçak gemisi fikrininin fizibilitesini daha kolay hale getirmek, hem de hemen her yerden kalkış yapabilir ve kullanılabilir hava aracı çözümlerinden F35B, AV8B Harrier, V-22 Osprey araçlarında bulunana benzer dikey kalkış ve iniş özelliğinin insansız hava araçlarına uygulanması çözümü büyük önem arz eder. Bu çözüm uçak gemilerinde daha fazla taşıma kapasitesi açmayı, normalde uçağın iniş yapamadığı bölgelerde helikopter gibi rahatlıkla iniş kalkış becerisi kazandırmayı ve hem drone gibi kalkış yapabilen hem de sabit kanatlar kadar hızlı araçlar geliştirmeye yönelik bir uçuş kontrol kartı yazılımını prototiplemeye amaçlar.

1.2. Çalışmanın Amacı

Bu çalışmanın yapılması ile hedeflenen, problem tanımında da belirtilen otonom olarak dikey kalkış, iniş işlemlerini yerine getirebilecek konsept niteliğinde sistemler geliştirmek ve bu sistemlerin enine boyuna incelenerek amaca ulaşılmasını sağlayacak başarı oranı en yüksek ve en kısa yolla imal edilebilecek sistemin gerekse mekanik tasarım ve gerekse kullanılacak malzeme bakımından, gerekse kullanılacak elektronik komponentler ve bunları çalıştırılan program bakımından bir araya getirilmesi işlemlerini prototipleyerek ortaya bir uçuş kontrol kartı/yazılımı geliştirmek ve bu işlemler sonucu bir araya getirilen sistemin hedeflenen görevi yüksek doğruluk oranı ile yerine getirebilmesidir.

1.3. Projenin Tanımı

Bu çalışmanın amacına ulaşması için kullanılması planlanan komponentler bir döner kanat tipi iha ve aynı şekilde bir sabit kanat tipi iha'nın birlikte yapılabilmesi ve havalandırılabilmesi için gereken minimum parçaya (Uçuş kontrolcüsü, ESC, Fırçasız motor, Alıcı, Kumanda vb.) sahiptir. Hedeflenen sonuca ulaşmak için gereken, VTOL sırasında insan müdahalesi olmaksızın bir uçuş gerçekleştirmeyi gerektirdiğinden VTOL işlemi uçuş kontrolcüsü aracılığı ile gerçekleştirilir. Uçuş kontrolcüsünün komutları ile ESC modülleri motorların hızını ve dolaylı olarak cihazın dengesini sağlar. Bu komponentlerin bir araya getirilmesi ve aralarındaki haberleşmenin sağlanması ile oluşan sistem insan müdahalesi olmaksızın dikey kalkış, iniş yapabilmelidir.

Bu doğrultuda tasarlanan cihaz uçuşa bir döner kanat ile tamamen aynı mantıkta başlar, kullanıcının gelen komut ile kullanıcının 2. bir komut gelene kadar belirli bir irtifaya kadar yükselir. Bu sırada cihazın havalandırmasını sağlayan kaldırma kuvveti sadece motorlar aracılığı ile sağlanır. Kullanıcının gelen 2. komut ile havada kalmamızı sağlayacak olan lifting motorlarından kanatlara geçene kadar cihaz tek bir eksen üzerine yan yatar. Cihaz yan yattığında ve kaldırma kuvveti artık kanatlardan sağlanmaya başladığında motorlar sadece uçağı itme görevini üstlenir ve bu andan sonra cihaz döner kanat uçuş modundan sabit kanat uçuş moduna geçer.

Cihaz indirilmek istendiğinde ise sabit kanat modunda iken kaldırma kuvveti kanatlar yerine motorlar tarafından üstlenilenilmesi gerektiğinden cihaz motor itiş vektörünün 90 derece olana kadar tek bir eksen üzerinde döner dikey olarak döner. Burada vektör yönlerinin doğrulanması ve modlar arası geçişti uçuş kontrolcüsü aracılığı ile sağlarız.

1.4. Çalışma Kapsamı

Genel olarak projenin gerçeklenimi için gerekli olan komponentlerin piyasa araştırılmasının yapılması, alınması ve bir araya getirilmesi ile ortaya bir döner kanat veya sabit kanat tipi İHA çıkartılabilir. Fakat bu çalışma kapsamında gerçeklenimi hedeflenen cihaz tam olarak ne bir döner kanat ne de bir sabit kanat İHA kategorisine girmeden piyasadan satın alabilecek sıradan uçuş kontrolcüleri ile bu projenin gerçeklenimi mümkün değildir. Bu sebeple kendi uçuş kontrolcümüzü yazmalıyız. Ben burada sektörde benzerlerinin aynı amaçla sıkça kullanımından dolayı STMicroelectronics'in STM32F407G-Disc1 programlanabilir mikroişlemci kartını kullanacağım. Bu kart kendi üzerinde lojik ve analog giriş çıkışlara ve kendi çevresel birimlerine sahip, bu sebeple yazılan programları simüle etmesi daha basit. Güçlü bir işlemciye sahip olduğundan karmaşık ve uzun işlemler kısa sürelerde yüksek kararlılıkla yapılabilir.

Yazılan kodların bu kart üzerinde denenmesi ve çalışabilirliğinin doğrulanmasından sonra daha küçük ve hafif olması sebebi ile STM32F407G-Disc1 kartı üzerinde yazılmış olan yazılım STM32F103C8T6 modeline uyarlanacaktır.

Uçuş kontrolcüsü yazılımı denemeleri esnasında STM32F407G-Disc1 kartı üzerinde zaten entegre olan LIS302DL 3 eksenli ivme ölçer kullanılacaktır. Denemelerin tamamlanması ve yazılımın başarılı bir şekilde çalışması halinde bahsedildiği üzere aynı LIS302DL 3 eksen ivme ölçer sensörü bu sefer harici bir komponent olarak STM32F103C8T6 ya bağlanacaktır.

Motorların kontrolü ESC üzerinden ve ESC ye gelen PWM sinyali de yine STM32 kartı üzerinden kontrol edilecektir.

1.5. Çalışmanın Gerçekçi Kısıtlar Açısından Analizi

Çalışma sonucunda deneme için kullanılmış olan STM32F407G-Disc1, programlayıcı gibi çevre elemanlar haricinde ekonomik maliyete sahip başlıca malzemeler :

- STM32F103C8T6
- Kumanda-Alıcı
- 4x Fırçasız motor
- 4x Pervane
- 4x ESC
- LIS302DL 3 eksen ivmeölçer
- 11.1 V 3S Li-Po Pil.

Burada birim başına en fazla maliyete sahip parçalar Kumandadır. Çalışma/İletişim frekans aralığı, alıcı kalitesi gibi faktörler cihazın kontrol edilme uzaklığını ve uçuş kararlılığını etkiler.

Keza ivmeölçer sensörünün ve kullanılan STM32 kartının birbirleri arasında ki haberleşme çözünürlülüğü ile sensörün kendi çözünürlülüğü ve temiz okuma yapması en kararlı uçuş performansının sağlanması açısından önemlidir. İvmeölçerin MPU6050, MPU9250, Adxl345 gibi alternatifleri çokça mevcuttur.

Motorlar, pervaneler, ESC ler ve Li-Po pil kullanılan modele, ağırlığa ve amaca yönelik olarak değişiklik gösterebilirler.

Bu listenin toplam maliyeti +-4000 Türk Lirasıdır. Çalışmanın fiziksel gerçeklenimi sadece bu malzemelerin birbirine montajından ibaret olduğundan kolaydır. Kullanıcıya ya da etrafa sağlık/çevre bakımından herhangi bir zararı bulunmamaktadır.

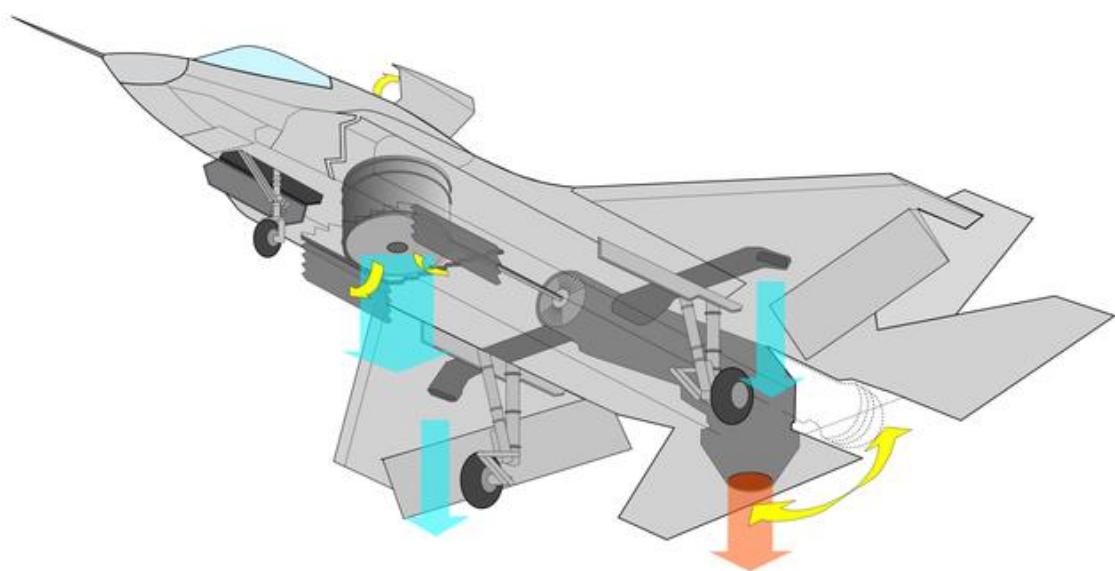
2. MATEMATİKSEL YÖNTEM VE TASARIM

Bu bölümde çalışmada kullanılan yöntem ya da yöntemlere ait bilgilendirme ve yöntemin çalışmada kullanımına ilişkin literatür araştırmalarına yer verilmiştir. Yöntem veya yöntemlerin problemin çözümünde kullanımını ve tasarım aşamaları da bu bölümde verilmektedir.

2.1. Yöntem Hakkında Genel Bilgi

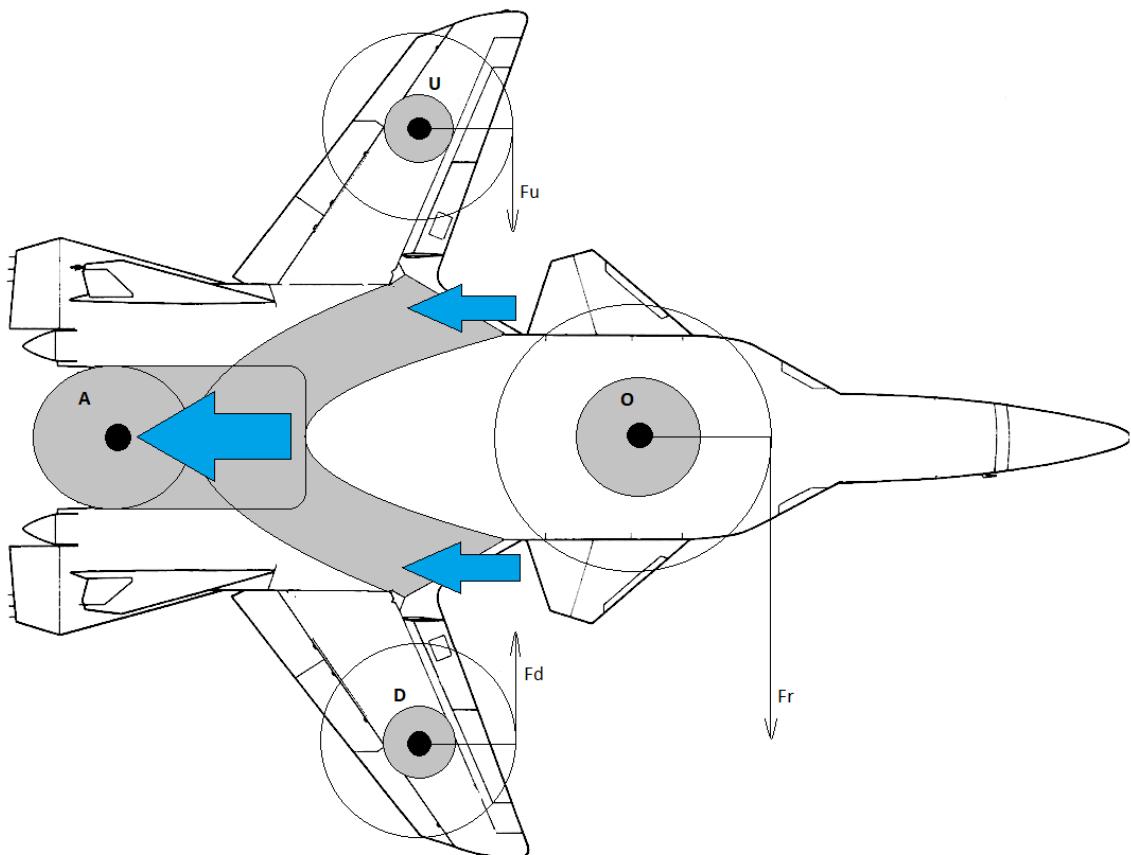
Ünlü VTOL özelliğine sahip savaş uçaklarını, misal AV8B Harrier, SU-47, V-22 Osprey modellerini inceleyeceğimizde genel olarak uluslararası arenada geçen bir problemini görüşürsünüz: "Stabilizasyon". Askeri amaçlarla üretilen teknolojilerde her şey inanılmaz bir hassasiyetle ölçülür/biçilir. Fakat projenin aynı anda hem ucuz, hem hassas olmasını sağlamak zordur. Hali ile F35B projesine kadar geneli çok kullanılmadı veya hep prototip olarak kaldı.

Bu nedenle bu çalışmanın prototiplenmesi sırasında literatür araştırmasında karşılaşılan mühendislik problemlerinden olabildiğince kaçınılmaya çalışıldı ve en mantıklı olduğu düşünülen konsept üzerine çalışmalar başlandı.



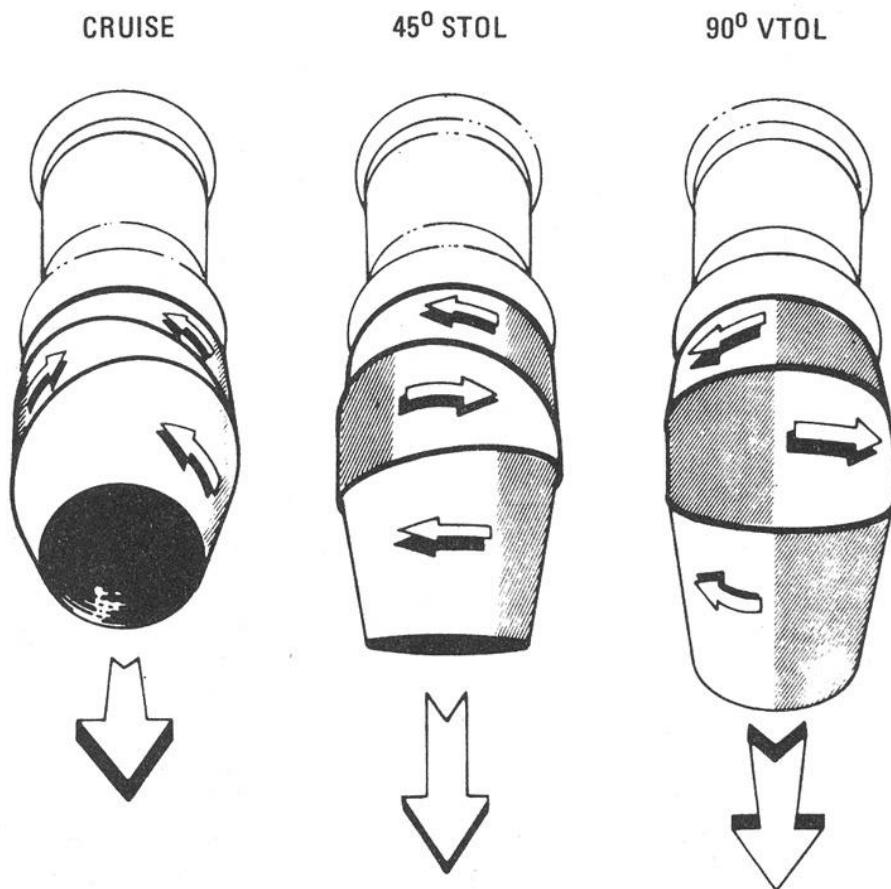
2.2. Tasarım

2.2.1. Mekanik

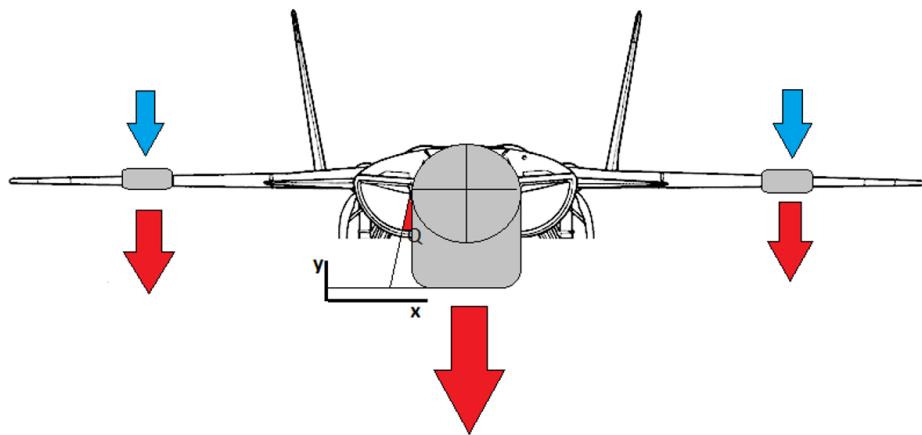


Görülen konsept tasarımda 4 adet simetrik şekilde yerleştirilmesi planlanan motor bulunmaktadır, bunlardan 2 tanesi kanatlarda ve küçük motorlar olup (U ve D motorları) VTOL sırasında uçağın dengede kalabilmesi ve dönebilmesi için gereklidir, diğer 2 motor ise ön (O) ve arkada (A) olmak üzere arkadaki motor hem ön motor ile beraber VTOL için gerekli olan kalkış gücünü hem de tek başına itiş gücünü sağlar.

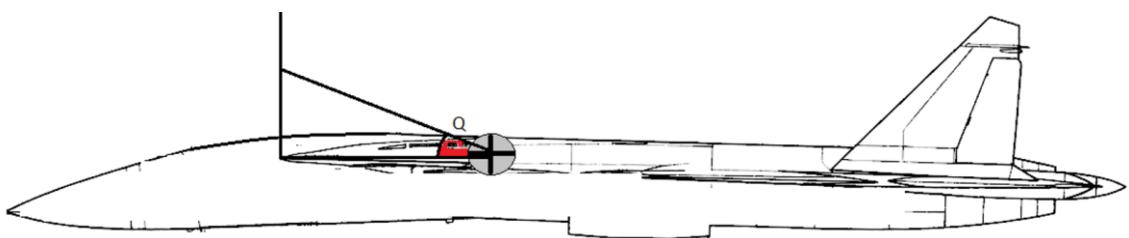
İtiş gücünü sağlayan arka motor yukarıya bakmaz (y eksenine dik değildir) ve itiş vektörü uçağın arkasına doğrudur, itiş vektörünün yönlendirilmesi için bir “swivel nozzle” modeli kullanılmıştır. (İtiş vektörü bu şekilde x-y arasında değiştirilebilir)



Diğer 3 motor ise VTOL ve denge amaçlı kullanılacağından itiş vektörleri aşağı doğrudur. Bu sebepten motorların açısal momentumları arasında bir dengesizlik yaşanır ve uçak VTOL sırasında dengesiz gücün (F_x) etkisiyle sürekli havada tek bir yöne (sağa veya sola) dönmeye başlar. Arka motora bağlı olan nozzle Q açısı ile bir $-F_x$ kuvveti uygular ve dengesizliğin ortadan kaldırılması amaçlanır.

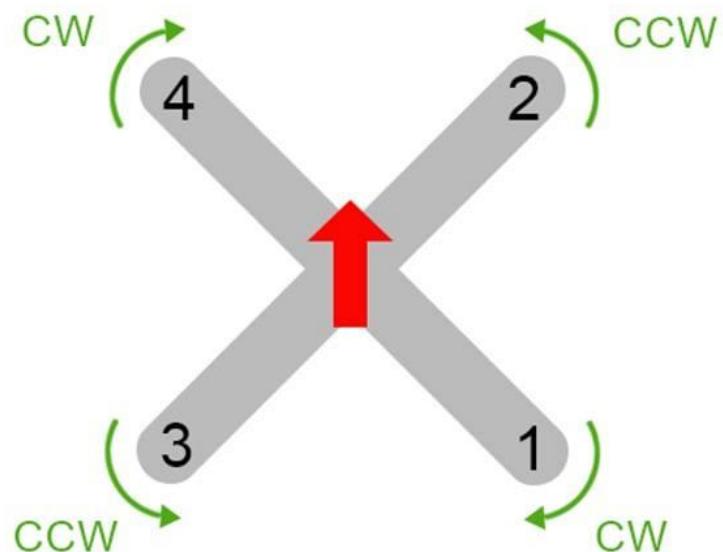


A motoru da kendi açısal momentumu ile uçağa sürekli x ekseninde dönmesi yönünde bir F_a kuvveti uygular, F_a kuvvetinin dengelenmesi için uçağın kanatlarında varsayılan flap derecesi (Q) bulunmalıdır, bu şekilde uçuş sırasında uçak F_a kuvvetinden etkilenmez.



Flaplarda bir varsayılan derece farkı olması F_a kuvvetinin etkilerini uçuş sırasında dengeleyebilir fakat VTOL sırasında da dengesizliğe sebep olabilir, bu dengesizliği en aza indirmek için dengesizliği yaşadığı yöndeki U ya da D motorlarından biri diğerine göre daha fazla lifting uygulamak adına daha fazla döner. Bu şekilde ortaya çıkan yeni F_x dengesizliği tekrar hesaplanır.

Bu şekilde izlenen bir gerçeklenim yönteminde stabilizasyonu sağlamak zordur ve çok hassas ölçümler gerektirir. Peki bu durumda ek hesaplamalarda kurtulmak için motorların açısal momentumlarından kaynaklanan kuvvetlerin eşit olması ve birbirlerini sıfırlamaları en iyisidir.



Fakat bu tasarım bir döner kanat tasarımlı ve itme-vektörleme bu tasarımda yapılsa bile kaldırma kuvveti hep motorlar sayesinde sağlanacaktır. Bu durumda nasıl bir tasarım izlenmeli?



Bu modelin ismi “H-Wing Swan k1”, bir VTOL özelliğine sahip İHA. Peki bunu nasıl yapıyor. Görselde görüldüğü üzere kalkış sırasında ki pozisyonu da bu oluyor. Normal bir quad-copter gibi havalandırıyor ve belli bir irtifaya ulaştığında kullanıcının kumandadan verdiği komut üzerine tek eksendeki motorların hızının artması ile yan yatıyor ve artık uçması için gerekli kaldırma kuvvetini kanatlarından almaya başlıyor. Uygulaması daha basit ve ucuz. Motorlar ve vektörleri tamamen aynı. Sistemi dengesizleştiren bir unsur söz konusu değil.

Bu çalışmada da amaçlanan aynı şekilde VTOL özelliğini çalışma sırasında yapılan bir İHA na kazandırmak.

2.2.2. Elektronik

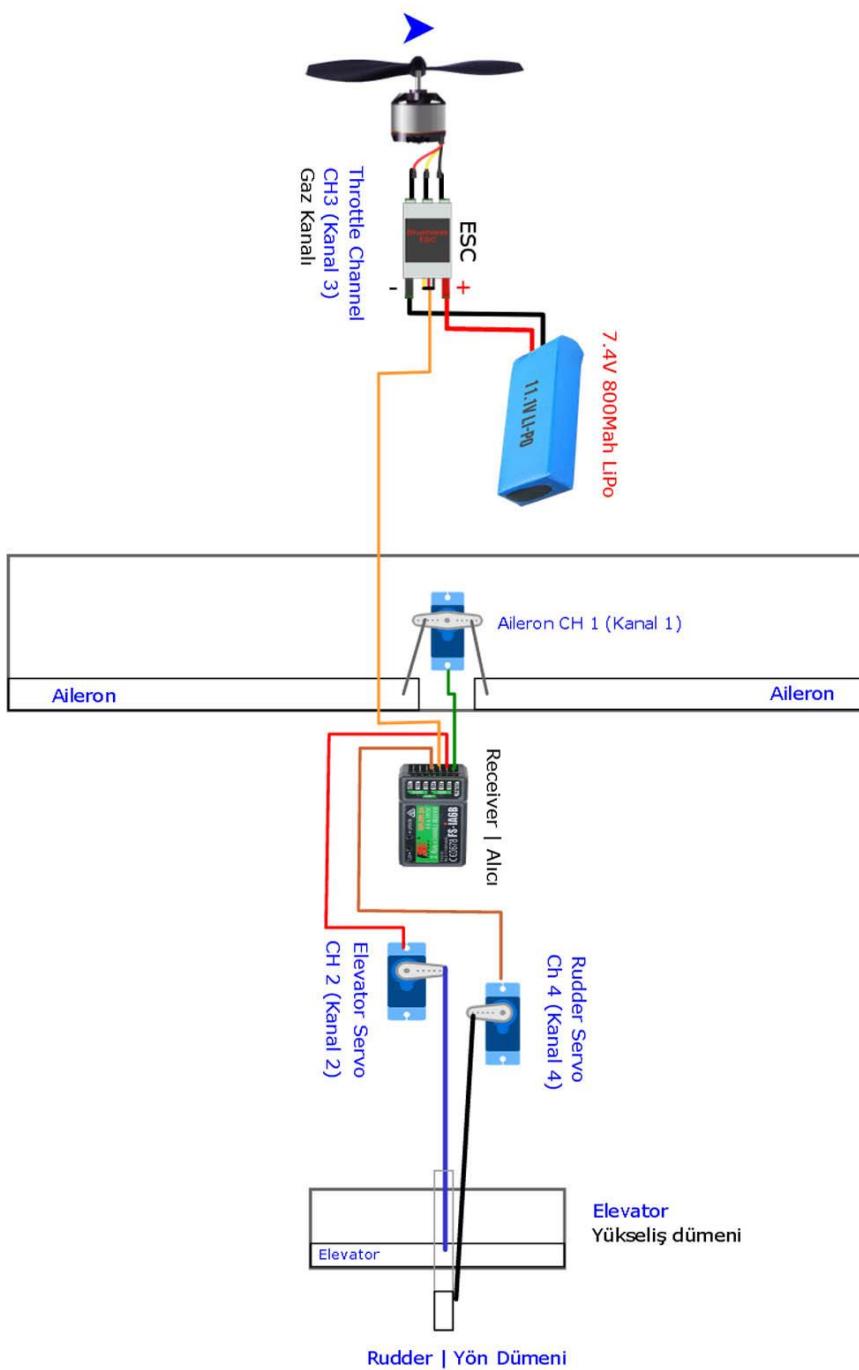
Kumanda ile uçurulan model uçaklarda uçuş kontrolcüsü şart değildir ve direkt olarak kumanda alıcısından alınan komutlar doğrultusunda manuel olarak motorlar sürürlür ve uçak kontrol edilir, fakat biz VTOL özelliğini amaçladığımızdan uçuş kontrolcüsü ile 4 fırçasız motor arasındaki dengeyi sağlamak şart.

Her motorun önünde kendi ESC si bulunur. Uçuş kontrolcüsünde yapılan işlemler sonucu her ESC için çıkış verilir ve motor hızları, yani dolaylı olarak uçağın dengesi sağlanır.

Her bir modül ayrı olarak kendi güç kaynağına bağlıdır. Bu çalışma için ESC ler direkt olarak bataryaya bağlı iken STM32 kartımız ve servo motorlarımız ESC nin BEC özelliğinden sağlanan 5 V a bağlıdır.

VTOL gerçekleştikten sonra tüm yönetim kullanıcı aracılığı ile kumandadan verilen komutlar sonucu gerçekleşir, kanatlardaki flapları ve motorları yöneten artık kullanıcıdır.

Uçuş kontrolcüsünün STM32F4XX modeli ile yapılması planlanmıştır.



2.2.3. Yazılım

Çalışma sırasında yazılım çalışmalarında hem simülasyon hem de araştırma/geliştirme işleri için STM32F407G-Disc1 kartı kullanılmıştır. Bu kart hem 100 adet I/O pinine ve 32 bitlik işlemciye hem de birlikte kullanılabilecek kendi entegre çevre birimlerine sahiptir.

Amacımız bu kartı uygun çevre birimleri ve sensörleri kullanarak hedefin gerçeklenmesine yönelik prototip bir uçuş kontrol kartı yazılımı oluşturmaktır.

Yazılım yazılrken gerekse konfigürasyon kolaylığı sağlanması, gerekse muhtemel problemlere topluluk içerisinde çözüm bulunması kolaylığı sebebi ile STMCUBEIDE yazılımı üzerinde HAL kütüphaneleri kullanılmıştır.

Hiçbir hazır kütüphane veya kendim yazamayacağım bir kodu kullanmadım.

Yazılımın geliştirilmesi sırasında kullanılan modüller ve edinilmesi gereken birikimi sıralayacak olursak;

- Fırçasız motorların kontrolü işlemcinin TIMER 4 modülü üzerindeki PWM özelliği kullanılarak gerçekleştirılmıştır.
- PWM değeri ADC birimleri kullanılarak 10 kOhm luk Potansiyometreler yardımı ile ayarlanmıştır.
- Duruş kontrolünün sağlanması için gerekli açı bilgileri MPU6050 6 eksenli Gyro Sensörü kullanılarak I2C haberleşme protokolü üzerinden elde edilmiştir. Bunun için MPU6050 nin DataSheet inde belirtilen yazmacılardan veriler okunarak belli sabitler ile işleme sokulur.
- Dikey Kalkış > Yatay Uçuş Moduna Geçiş > Yatay Uçuş gibi modları tanımlamak ve uygulamak için Interrupt kullanıldı
- PID (Proportional-Integral-Derivative) hesaplamaları yapıldı
- Bir RC kumanda alıcısı üzerinden değer okunulmaya çalışıldı
- CUBEMX üzerinde kullandığımız işlemcinin/kartın hedefe yönelik
 - RCC Clock Konfigürasyonları
 - GPIO Konfigürasyonları
 - ADC Birimi Konfigürasyonları
 - TIMER Birimi Konfigürasyonları
 - I2C Konfigürasyonu

Herhangi bir çevresel birimi kullanmadan önce bu şekilde gerekli matematiksel işlemleri ayarlayan konfigürasyonları yapmak gereklidir. Bunun sebebi kullanılan çevresel birimlerin bulunduğu sinyal hatlarının farklı frekanslarda çalışıyor olabilmesidir. (Örn. STM32F407 Max Clock Frekansı 168 MHz de çalışırken TIMER modülü 84 MHz de çalışıyor.)

3. UYGULAMA ÇALIŞMALARI

3.1. Uygulamada Kullanılan Araç ve Gereçler

Uygulama bölümü çalışmanın şu ana kadar geldiği nokta itibarı ile prototipleme, deney ve gözlem aşamasındadır.

Uygulamanın fiziksel kısmı için şu ana kadar ortaya konulan prototipte kullanılan başlıca araç ve gereçler şunlardır;

- STM32F407-Disc1
- Arduino Nano x2
- NRF24L01 x2
- Fotoblok
- 12A 2-3S ESC
- 1806 2280KV BLDC
- 5045 Pervane 3 Pal
- 7.2V 2S Li-Po x2

Burada Arduino Nanolar ve NRF24L01 kablosuz iletişim modülleri kumanda-alıcı yapımı amacı ile kullanılmıştır. STM32F407-Disc1 programlanabilir kartı ise program yazmak ve yazılan programların simülasyonu amacıyla kullanılmıştır. Fotoblok şu anda prototip için kullanılan yapı malzemesidir. Geri kalan malzemeler ile birlikte bir model uçak elektronik-fiziksel modeli oluşturulur.

Prototipin ve denemelerin başarı ile tamamlanması sonucu asıl oluşturulmak istenen çalışmada kullanılması planlanan başlıca araç ve gereçler şunlardır;

- STM32F103C8T6
- KDS K-7XII 7 CH Kumanda – KDS K-8X 8 CH Alıcı
- Fotoblok *DSD (Alternatifleri : Komposit, 3D Print)
- 2205 2300 KV BLDC x4
- 30A 3-4S ESC x4
- 5045 Pervane 3 Pal x4
- 11.1V 3S Li-Po

Burada STM32F407G-Disc1 yerine STM32F103C8T6 kullanılmasının sebebi STM32F103C8T6'nın STM32F407G-Disc1'e göre daha küçük, hafif ve bu çalışmada gerçeklenimi planlanan proje için özelleştirilmesi daha uygun olmasıdır.

Motor büyüklüğünün ve sayısının değişmesi ile ESC'ler ve batarya da değişmiştir. Bunun sebebi ise gerçekleştirimi planlanan projenin fizibilitesinin daha kolay hale getirilmesi ve asıl çalışmaya geçiş sürecidir.

Artık prototipte ki gibi ortaya konulan tek bir sabit kanat model uçak değil, hem döner kanat hem de sabit kanat özelliklerine sahip bir cihazdır.

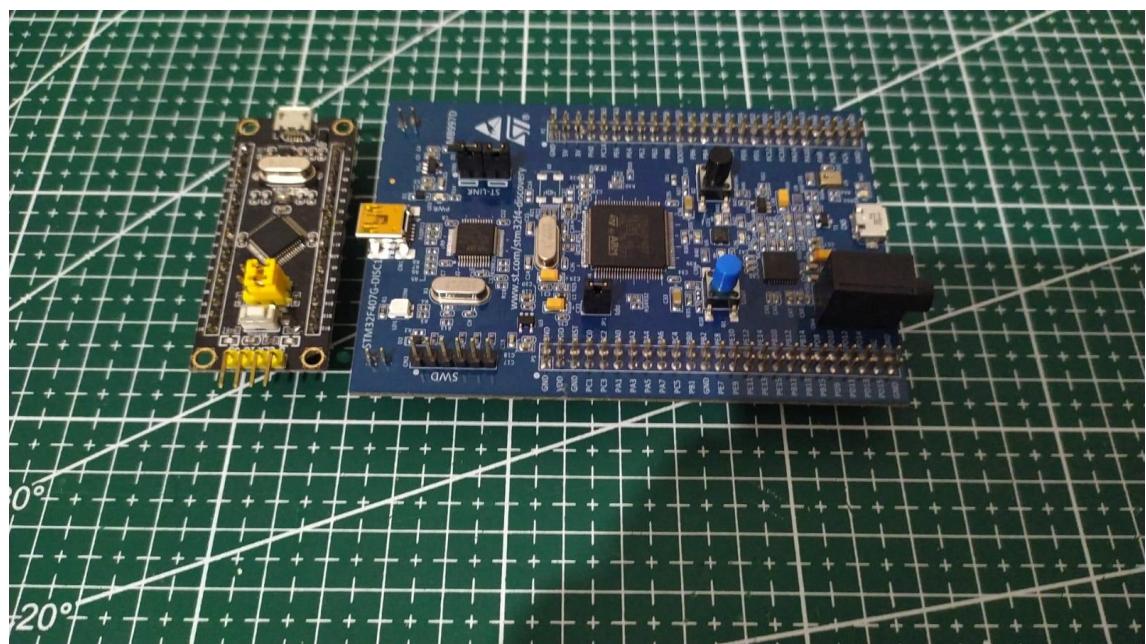
Bu malzemeler ile oluşturulması planlanan fiziksel modelin yönetimi;

Kullanıcı > Kumanda > Alıcı > STM32F103C8T6

şeklindedir. Bu durumda STM32F103C8T6 kartının yazılımı için C dili gömülü sistemlere yönelik olarak kullanılmaktadır (Embedded C). AtollicTRUESTUDIO platformu aracılığı ile 3 farklı kodlama şekli kullanılması uygun olduğunun düşünüldüğü kısımlarda farklı şekillerde kullanılabilir. Bunlar;

- Register Kodlama
- STDPeriph Kütüphanesi (~SPL)
- HAL Kütüphanesi

Şu ana kadar imal edilen prototipte “Register Kodlama” ve “SPL Kütüphanesi” yöntemleri kullanılmıştır.

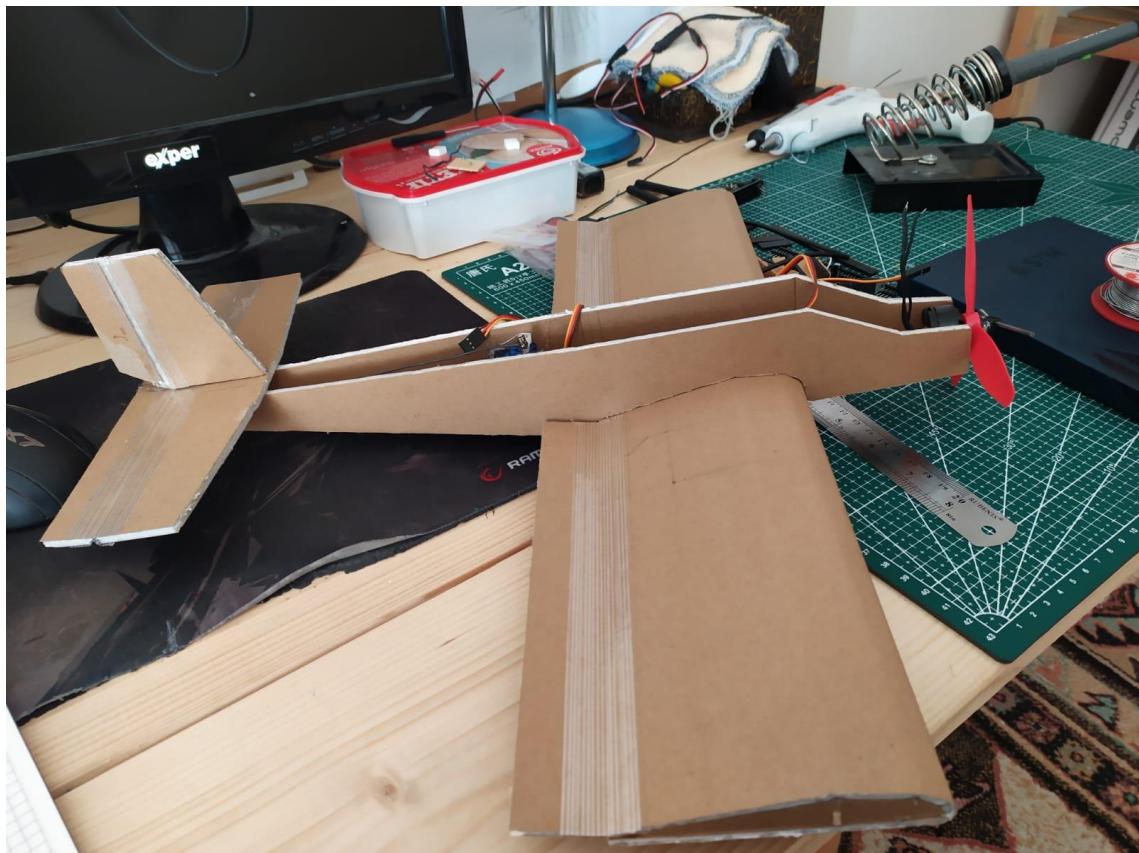


3.2. Uygulamada Gelenen Nokta ve Prototipleme

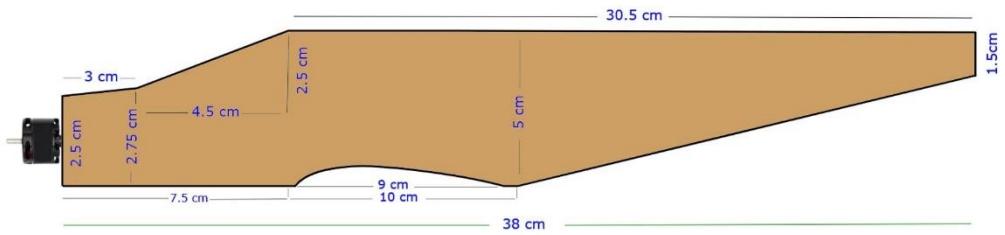
3.2.1. Fiziksel Model ve Elektronik

Gerçeklenimi amaçlanan proje aslında bir VTOL özelliği kazandırılmış sabit kanat iha olduğundan ve tasarımı VTOL sırasında döner kanat gibi stabil kalmayı gerektirdiğinden ortaya çıkarılacak olan modelde bir uçuş kontrolcüsü şartı vardır. Fakat amaçlanan görev sadece belli bir süre için VTOL özelliklerinin sergilenmesi olduğundan ve piyasada ki uçuş kontrolcülerini bu görevi yerine getiremeyeceğinden kendi uçuş kontrolümüzü imal etmeliyiz. Genel olarak piyasada kullanılan uçuş kontrolcülerini ile aynı yapıda olduğundan ve sıkça kullanıldığından bu iş için bir STM32 programlanabilir kartı kullanılması planlandı.

Ancak bu kart üzerinde yazılan programların denenebilmesi için önce bir prototip oluşturuldu. Deney ve gözlemlerin sağlıklı yapılabilmesi için prototipin de sağlıklı oluşturulması gerekiyor ve ortaya bir sistem çıkarma çabası bulunduğuundan aerodinamik, mekanik gibi konulardaki eksikliğini hazır modelleri kopyalayarak ve değiştirerek tamamlamak istedim ve sonuç olarak ortaya bir prototip çıkardım.

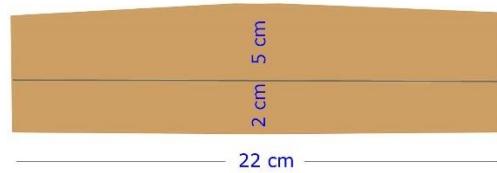
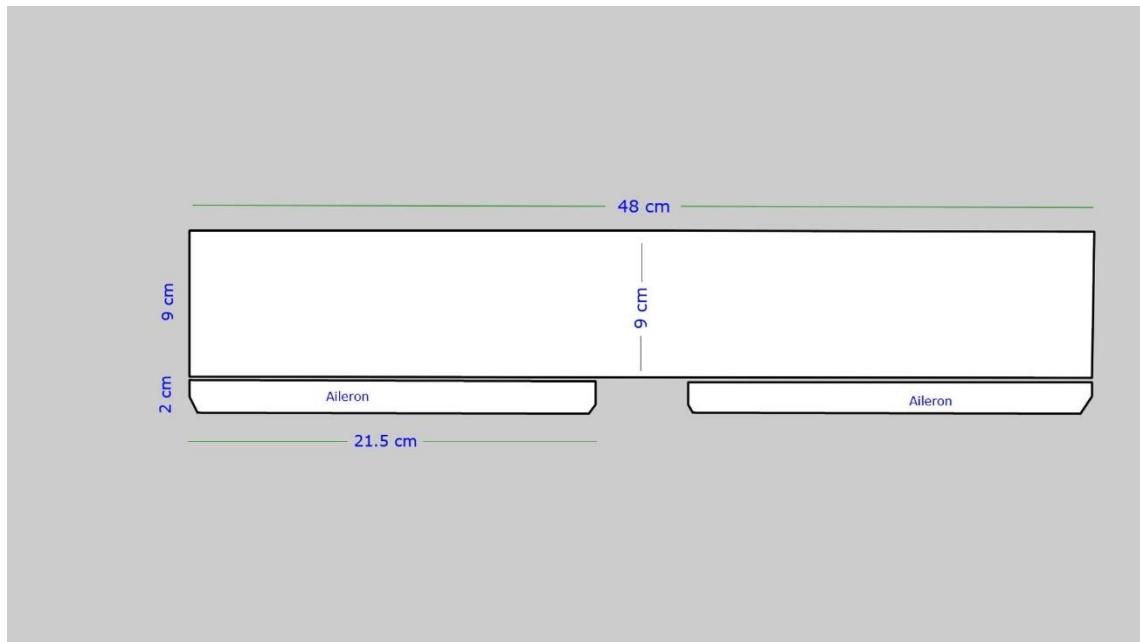


Bu prototipi ortaya çıkarırken yararlandığım kaynak ve planlar:

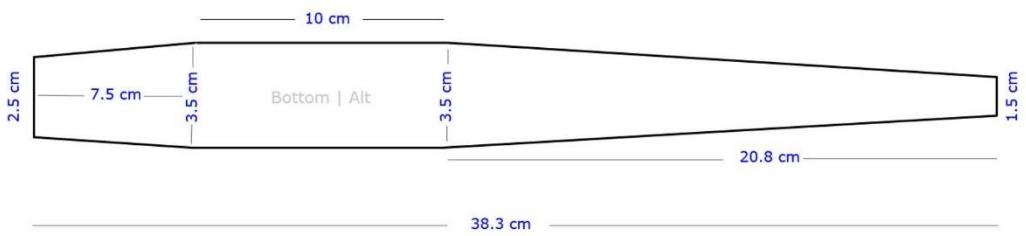
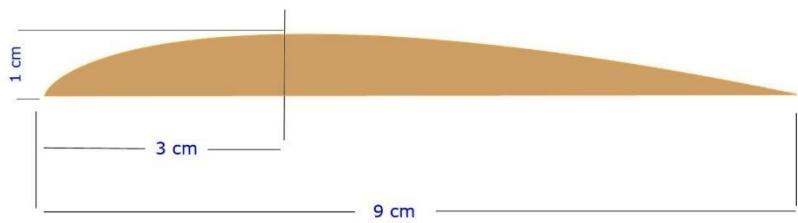
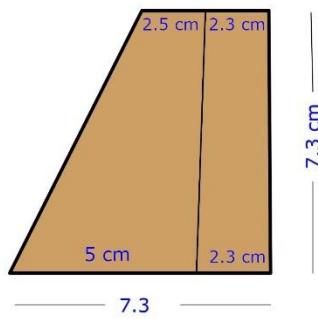


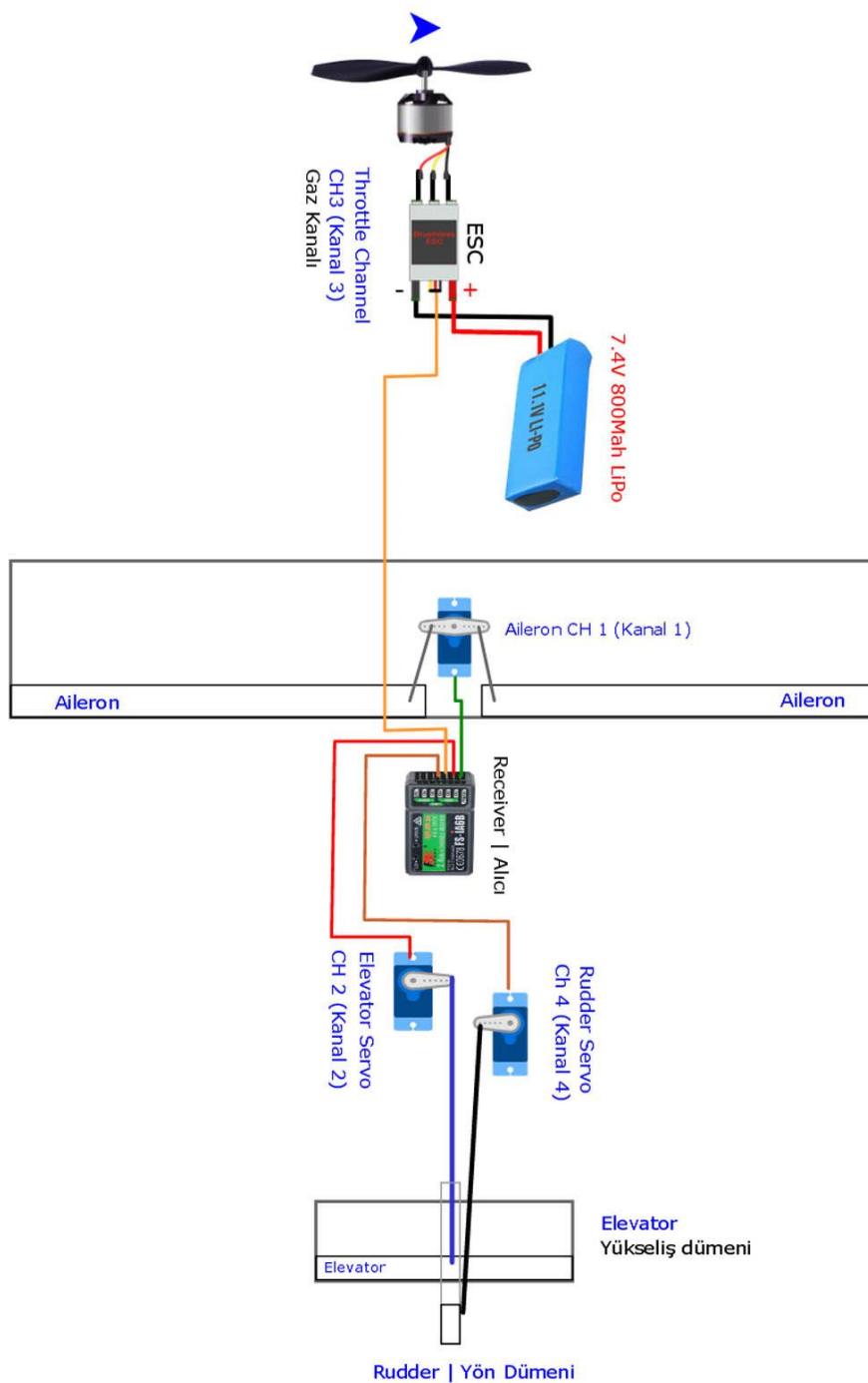
Body material: 3mm Foamboard

Gövde malzemesi: 3mm Fotoblok

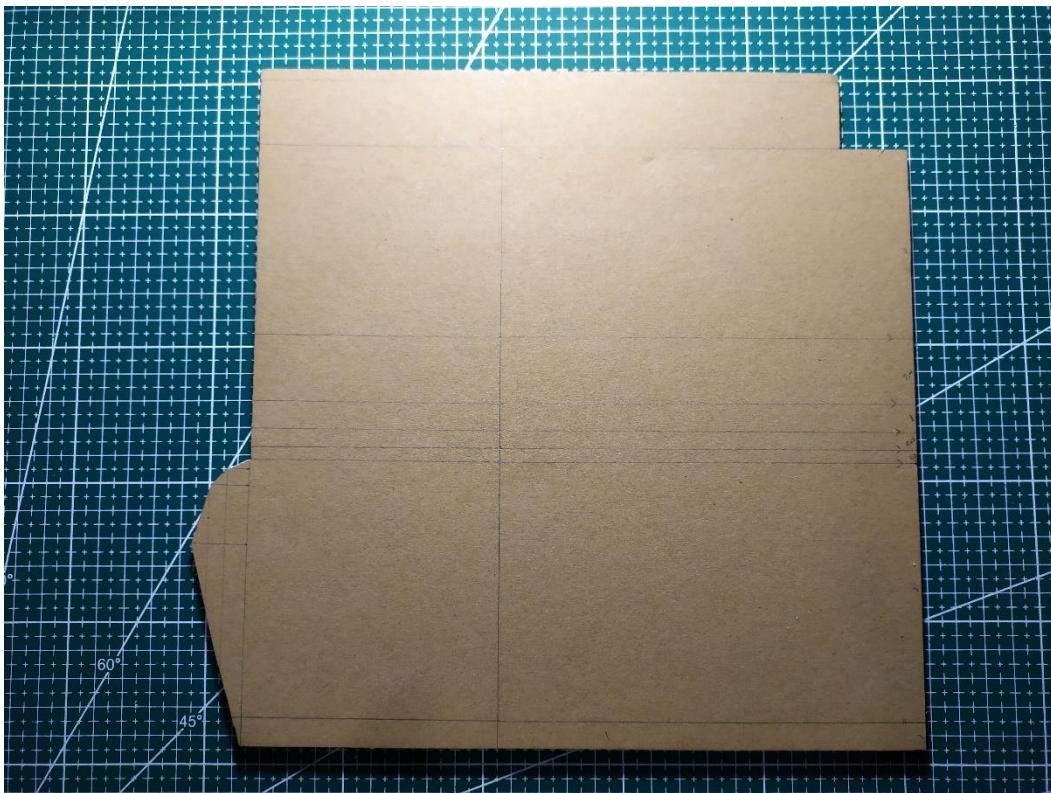


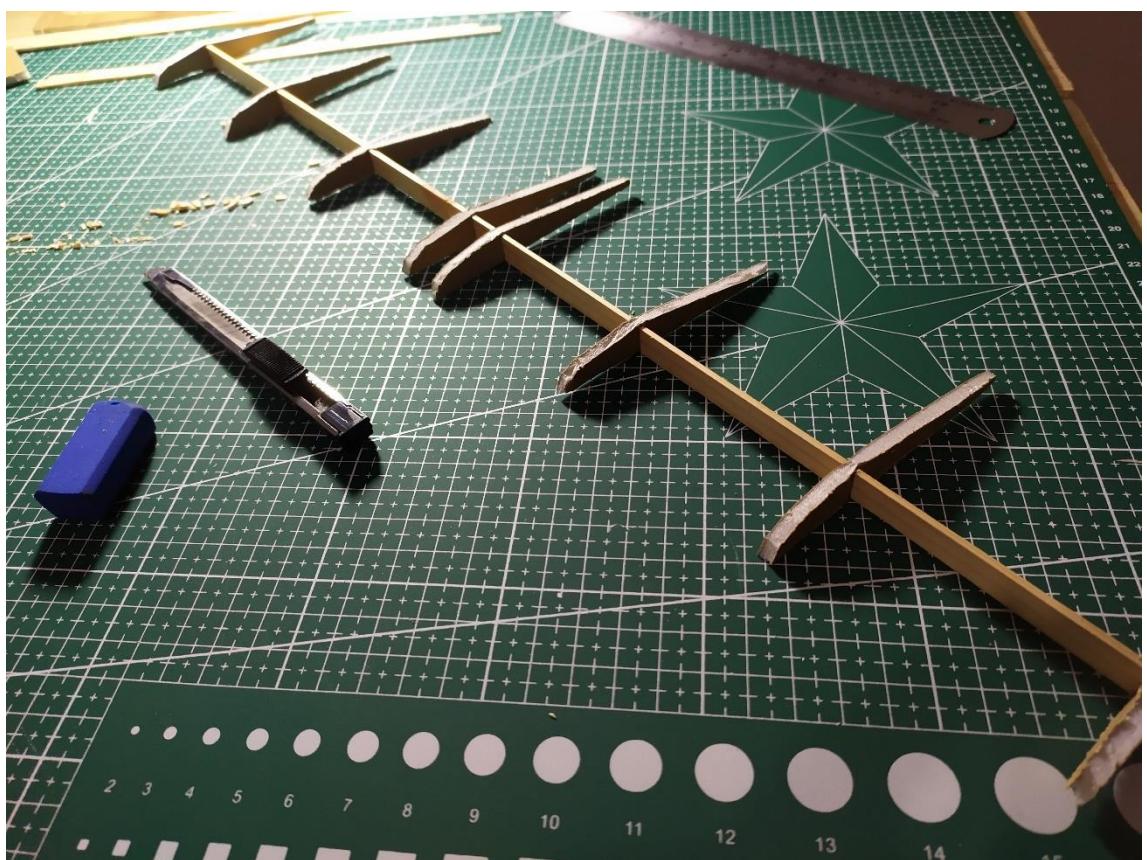
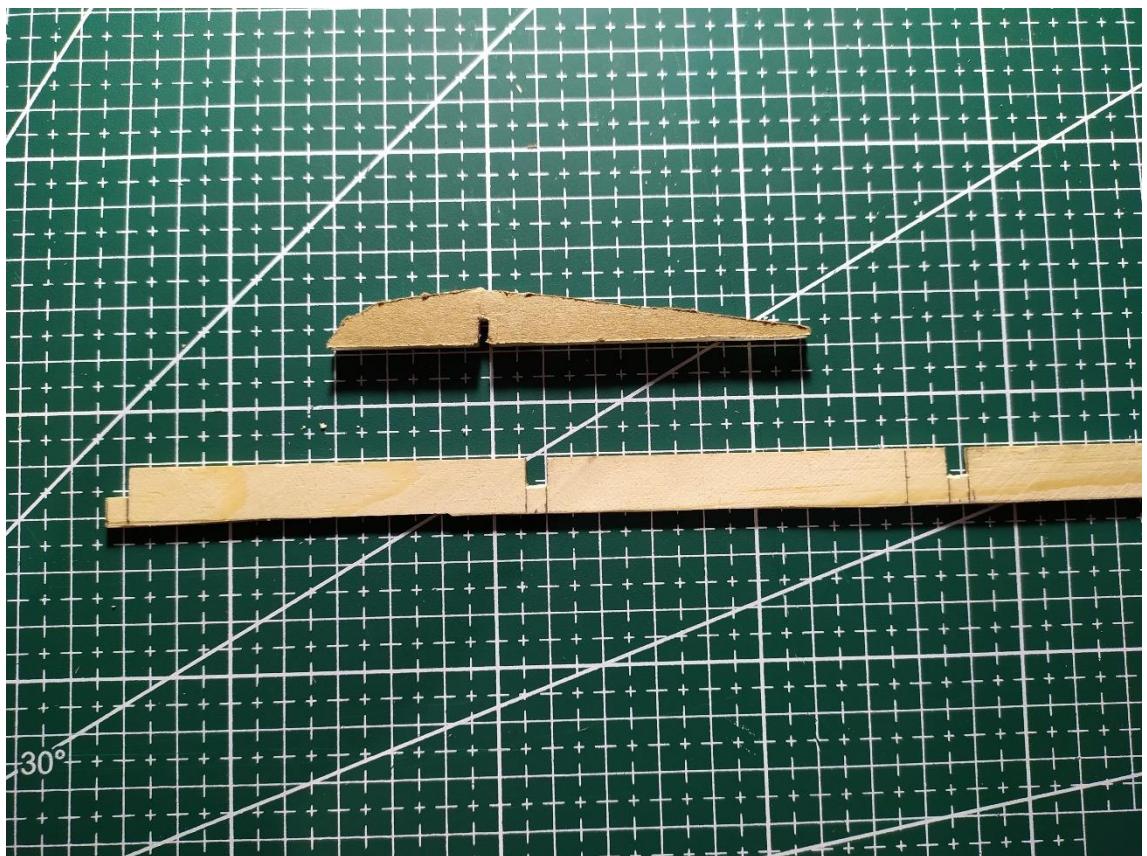
Material: 3mm foamboard
Malzeme: 3mm Fotoblok

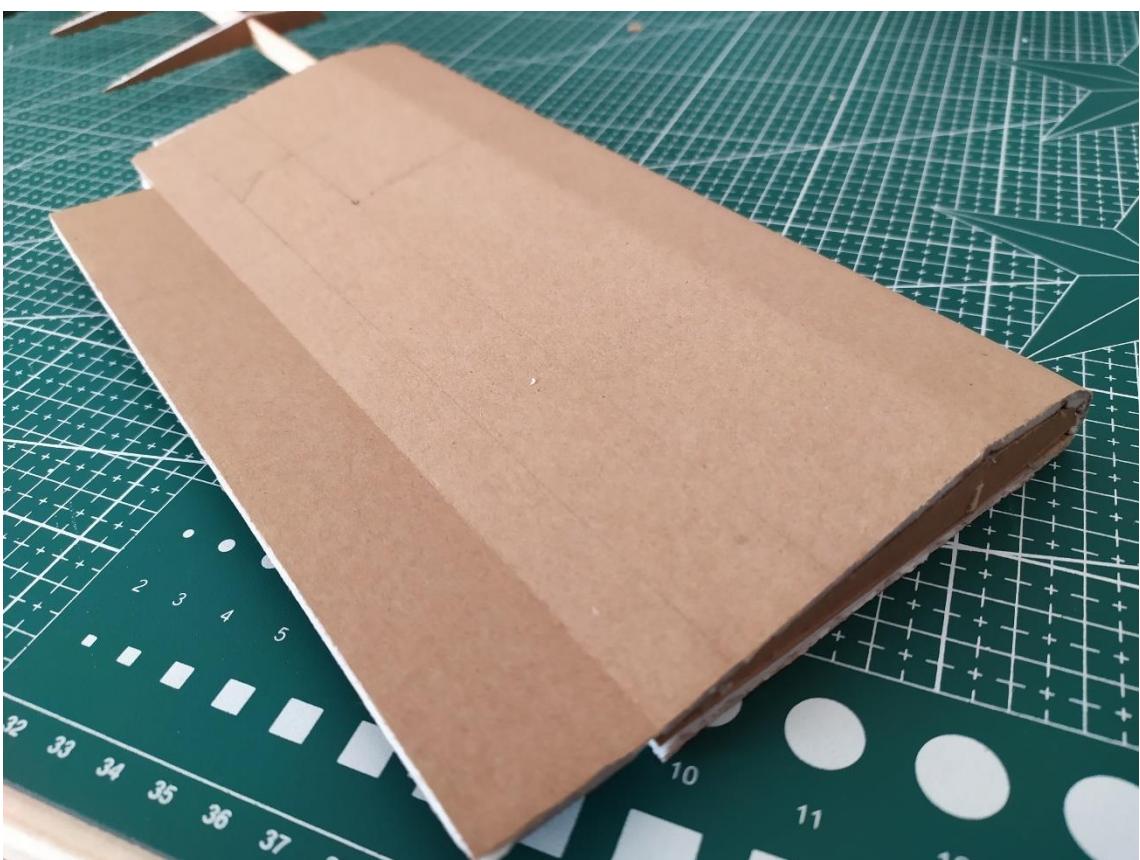
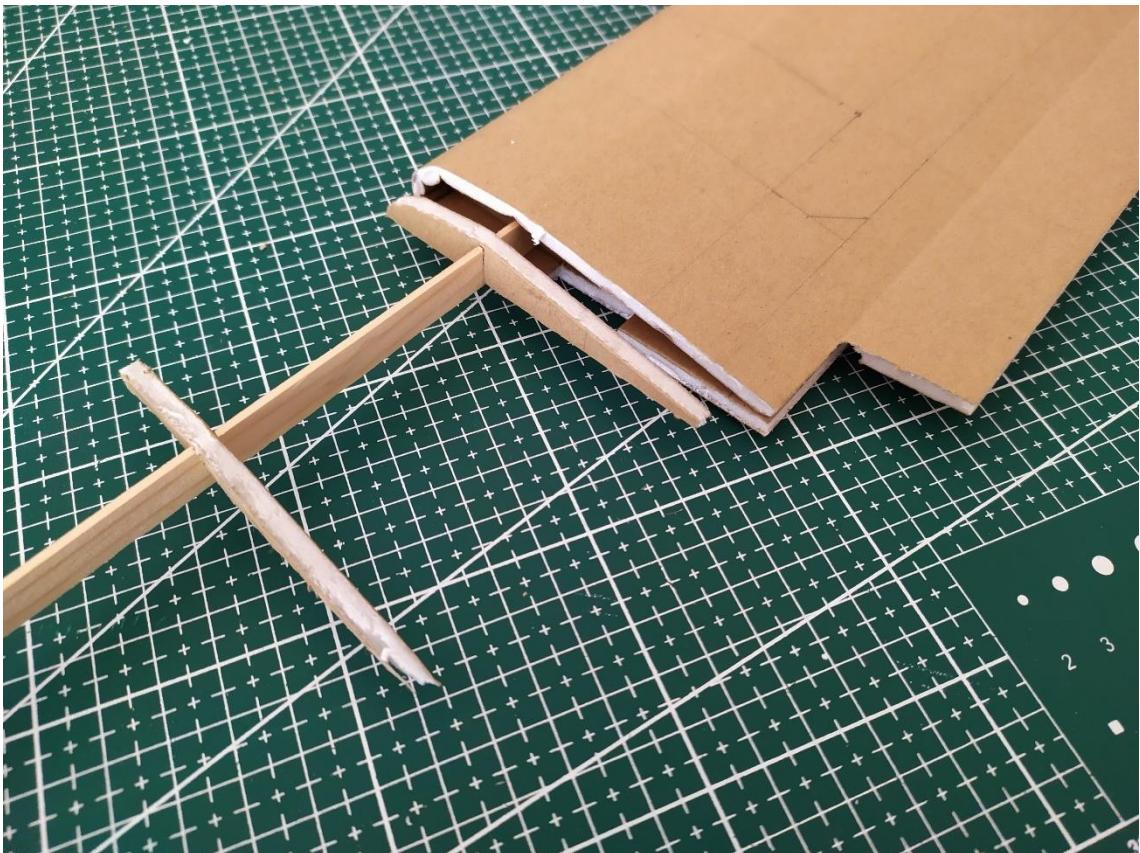


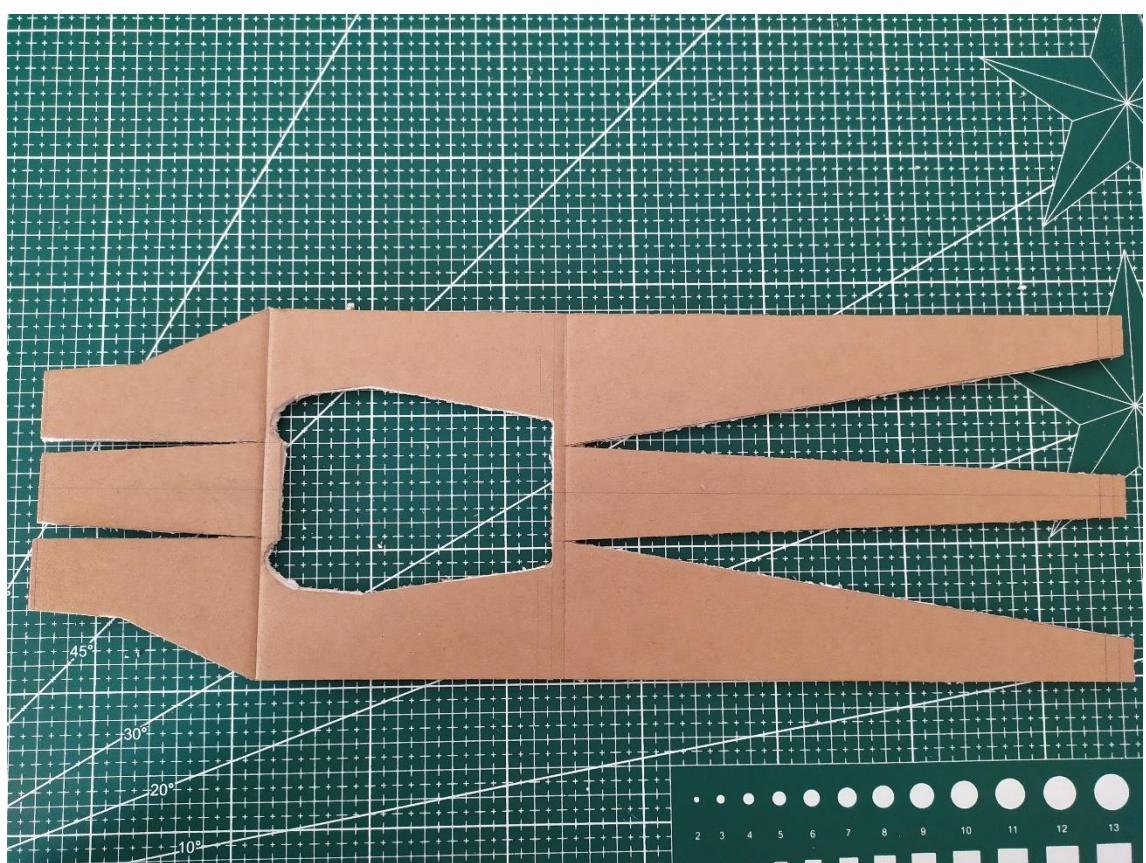
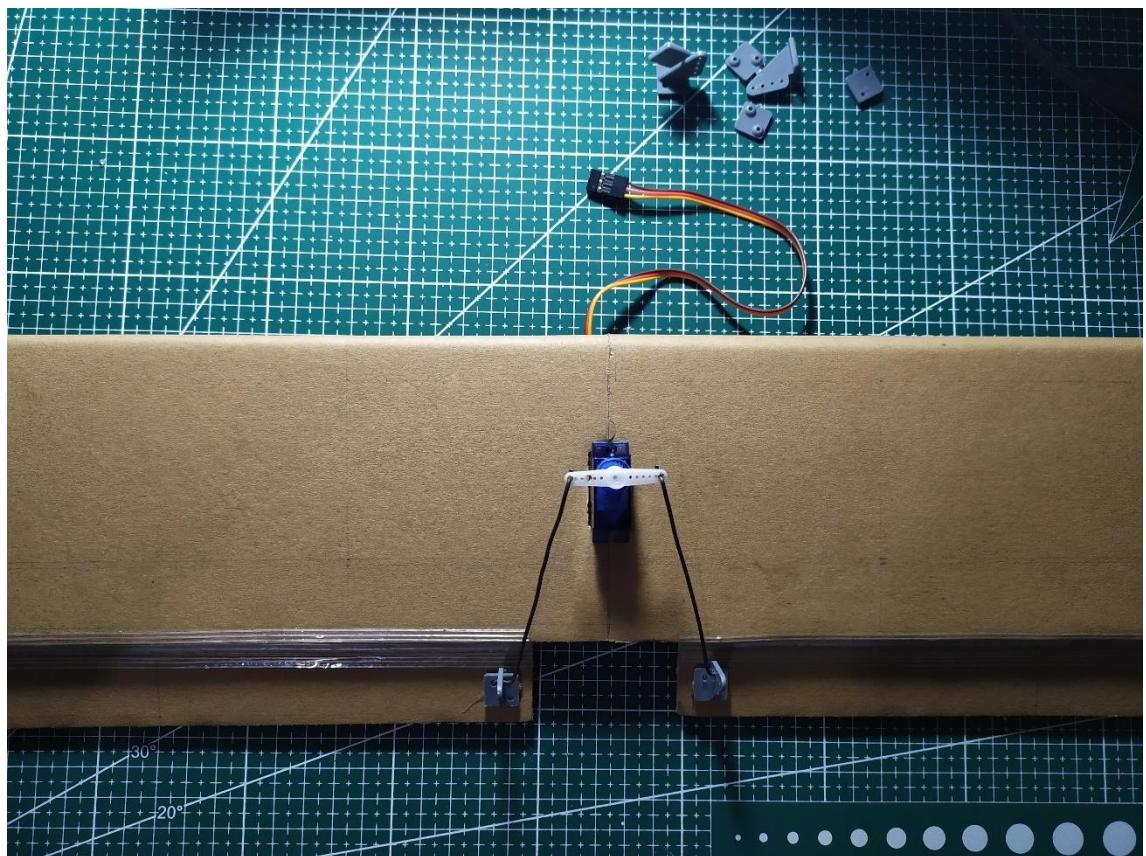


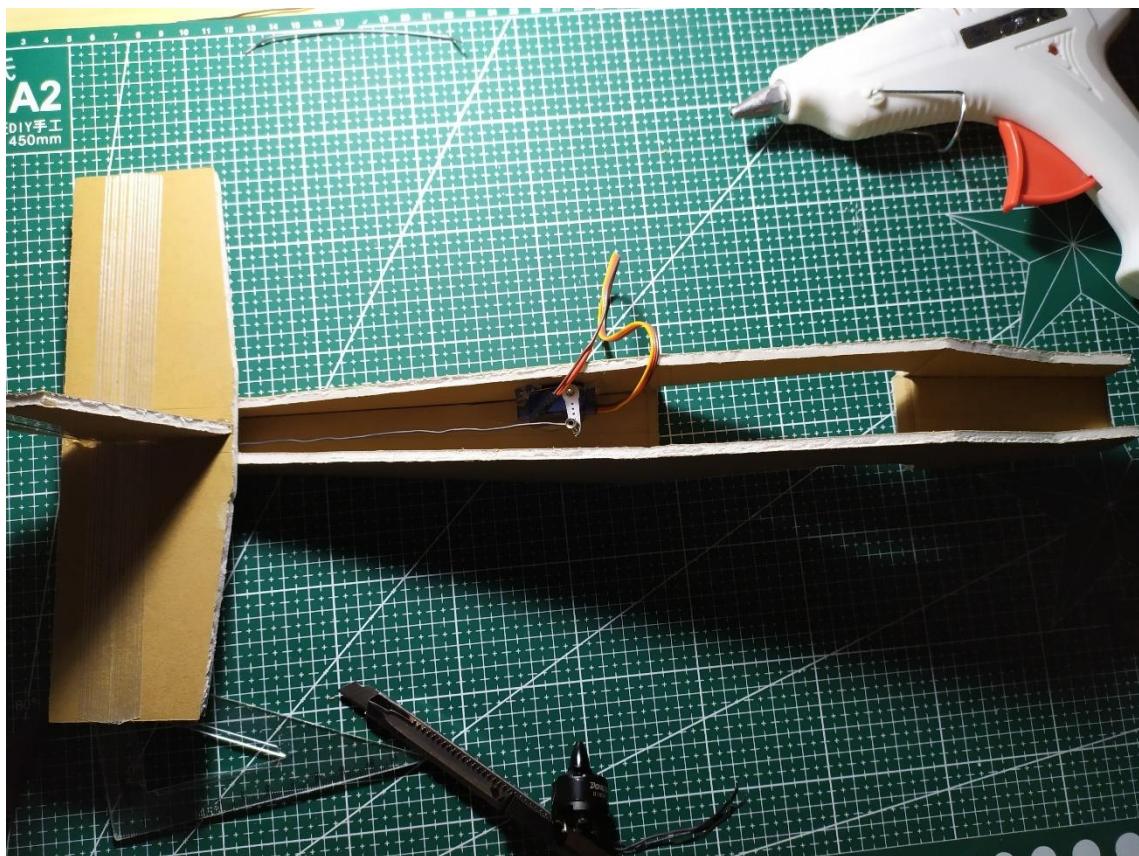
Ve sonuç olarak bu planlar üzerinde elimde bulunan komponentlere göre değişiklik/modifikasyon yaparak elde ettiğim devre ve prototip





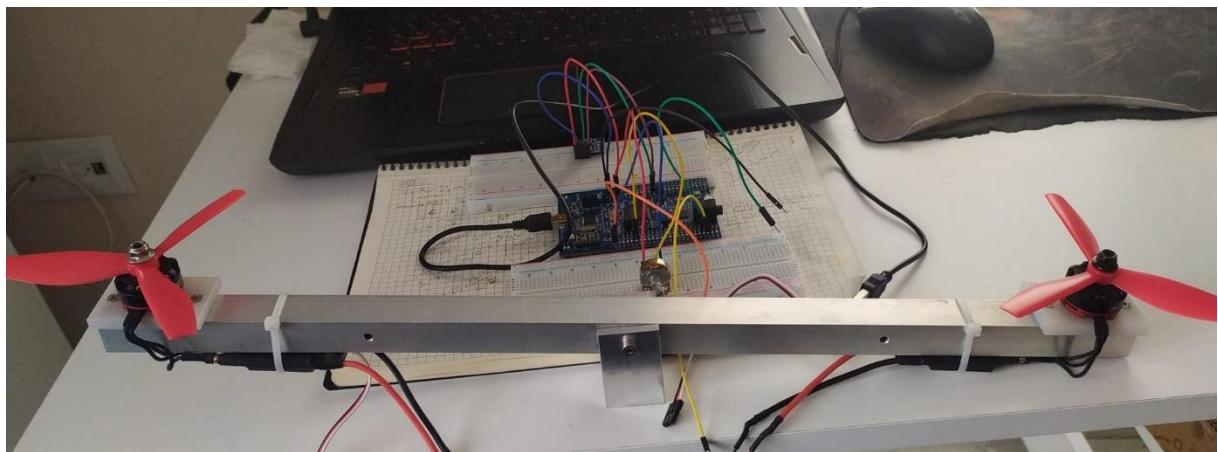






Bu prototip üzerinde flap kontrolleri ve kumanda alıcı-verici çalışmaları yapıldı ancak uçuş kontrol kartı yazılımında kullanılan yöntemlerin deneylerinin yapılması ve gözlemlenen sonuçlara göre yazılım üzerinde değişikliğe gidilmesi bu prototip ile mümkün değildi.

Bu sebeple MPU6050 Gyro sensöründen okunan veriler ve PID hesaplamalarının da gözlemlenebilir olması için yeni bir deney platformu oluşturdum.



Oluşturduğum platform iki ucunda da motorlar yukarı bakacak şekilde konumlanmak üzere serbest salınımı bir terazi mantığındadır.

Amaç ortaya konulan MPU6050 Gyro sensöründen okunan verilere göre teraziyi dengede ya da istenilen açı konumunda konumlandırmaktır.

3.2.2. Yazılım

Daha öncesinde ki çalışmalarımın STDPERIPH kütüphanelerini kullanmıştım fakat konfigürasyon zorlukları, kaynak kısıtlılığı ve karşılaşılması muhtemel problemlere yönelik bulunabilecek çözüm bakımından toplulukta çok fazla veri olmamasından dolayı konfigürasyonlar sırasında kolaylık sağlayan, kaynak ve topluluk açısından da şu anda yaygın olarak kullanılan HAL Kütüphanelerini kullanmaya başladım.

Yazılım çalışmalarını STM32F407G-Disc1 programlanabilir kartı üzerinde yürüttüm.

STM32F407G-Disc1 Genel Özellikleri

- 32 bit ARM CORTEX M4 mimari mikroişlemci
- 1 Mb Flash hafızası
- 192 Kb RAM
- 3 eksen MEMS ivmeölçer sensörü
- MEMS ses sensörü ve dijital mikrofon
- Kullanıcı ve Reset butonları
- 4 ü kullanıcısı olmak üzere 8 LED
- Entegre ST-LINK, USB Vbus veya harici besleme
- 3V ve 5V harici uygulama kaynağı

Başlıca programlanması gereken kısım uçuş kontrolcüsü olduğundan ve bir uçuş kontrol kartı için yazılımı gereklili minimum fonksiyonlar;

- PWM Generation
- PID (Proportional-Integral-Derivative)
- Gyro Sensör Reading

Bunlardan ilki Timer ile PWM kontrolü, basit gözükmesine karşın ESC ler servo motorlar gibi aynı şekilde PWM sinyalini işleyip motorlara ilettiğinden hayatı öneme sahiptir.

İkinci program ise MPU6050 6 eksenli ivme ölçer sensörünü kullanarak okuduğum x-y-z değerlerine göre açı değerlerini işleyerek bu açı değerlerine göre hesaplamalar yapmak. Bu kısımda MPU6050 sensörünü I2C1 birimi üzerinden okudum.

Elde ettiğim açı değerlerine göre cihazı dengede tutacak ve üretilmesi gereken PWM sinyalini hesaplayacak olan son ana fonksiyonumuz ise PID oldu. Burada istenen açı değeri ile anlık açı değeri arasındaki farka göre hangi motorların ne kadar hızlanması yani o motora ne kadar PWM sinyali gönderilmesi gerektiğini bu kısımda hesaplıyorum.

Ayrıca ;

- RC Alıcıdan veri okumak
- Potansiyometri ADC birimi üzerinden okuyarak ana PWM Pulse değerini ayarlamak

Yine bu proje için hayatı önem arz ediyor.

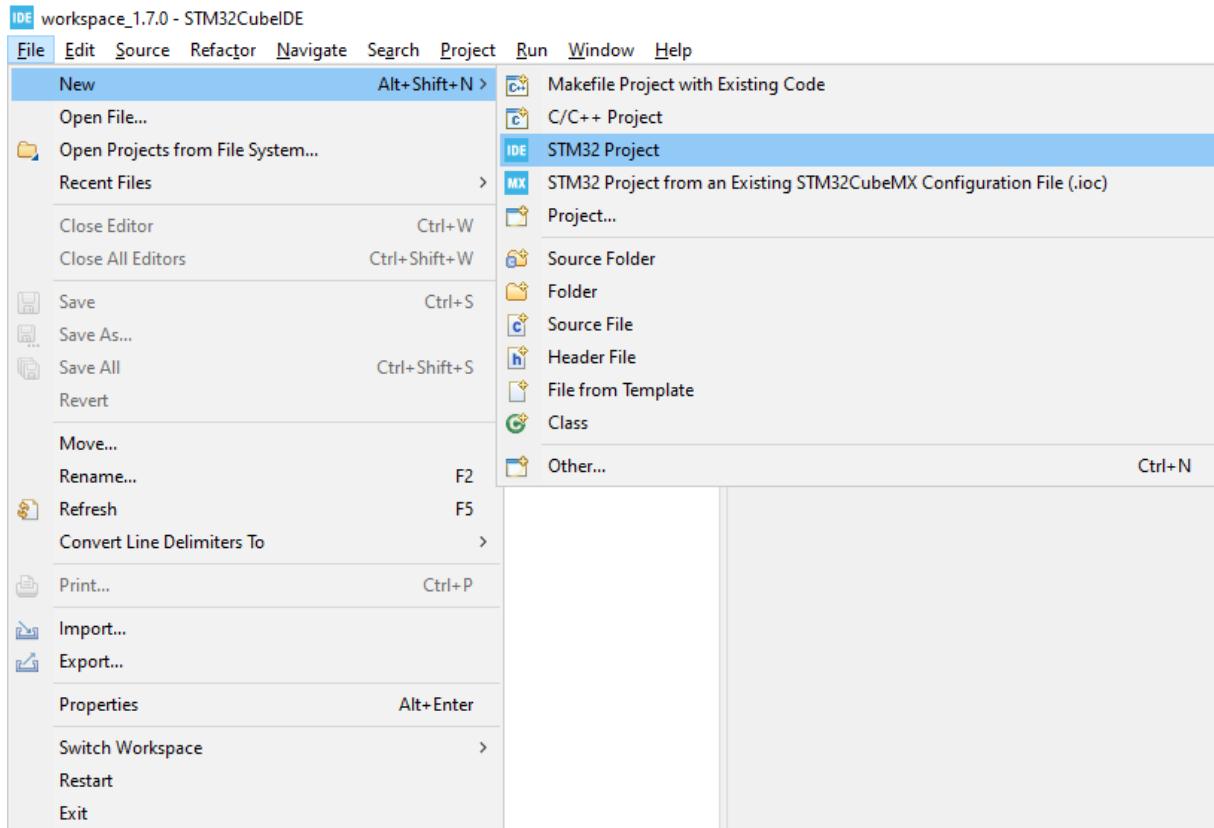
RC Alıcıdan veri okumayı başarsaydım Potansiyometre kullanmama gerek kalmayacaktı.

Yazılımda dahil edilmesi gereken önemli kısımları bildiğimizde göre CUBEMX üzerinden projemizi oluşturarak gerekli konfigürasyonları yapabiliriz.

3.2.2.1 STM32CubeIDE Üzerinde Konfigürasyonların yapılması

İlk olarak STM32CubeIDE yi açtıktan sonra yeni proje oluşturarak başlayalım

File → New → STM32 Project



Aynı şekilde bu işlem STM32CubeMX yazılımı ile de yapılabilir.

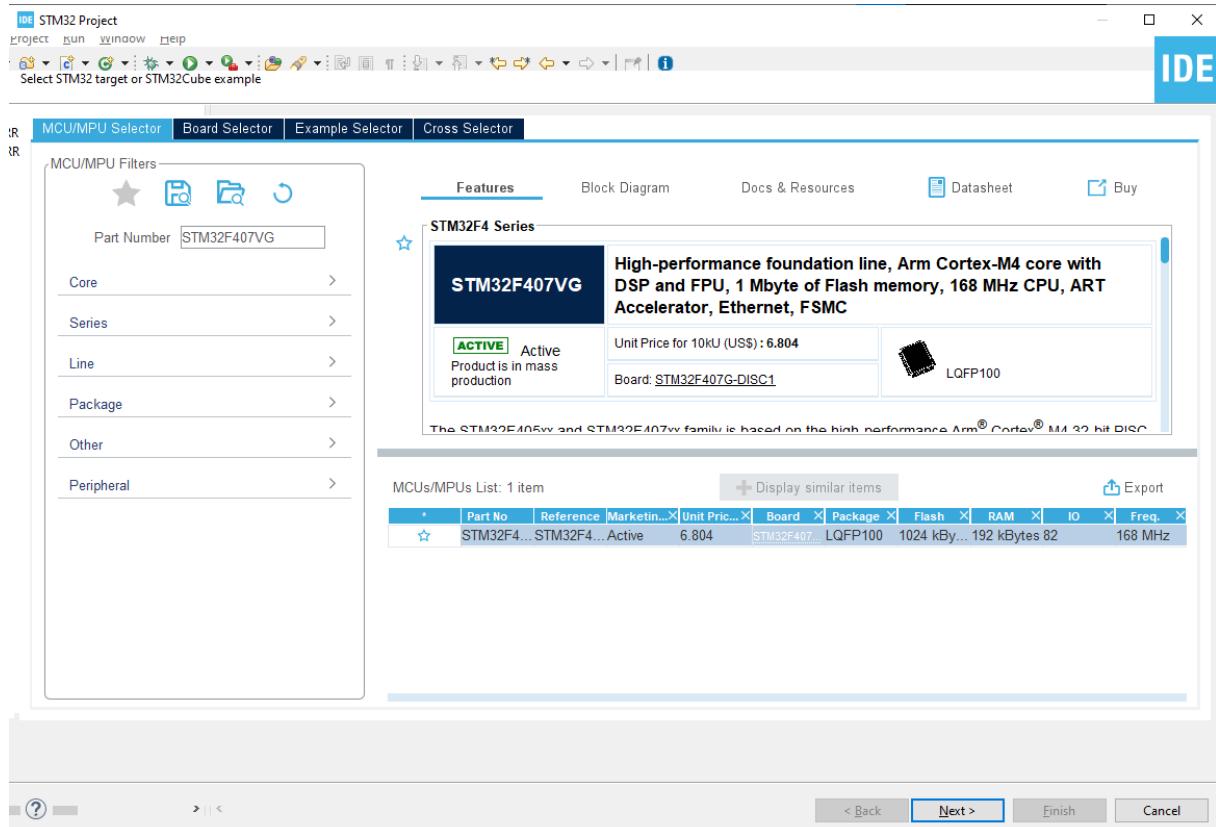
Fakat CubeMX de proje dosyası oluşturulduktan sonra oluşturulan proje dosyasını kullandığımız IDE ye import etmemiz gereklidir.

Hangi IDE yi kullandığımıza bağlı olarak CubeMX de konfigürasyonu GENERATE ederken seçebiliriz.

Örneğin Atollic TrueSTUDIO seçerek CubeMX de oluşturduğumuz projeyi buraya dahil edebiliriz.

Sonrasında karşımıza “STM32CubeIDE Target Selector” ekranı gelecek. Bu ekranda konfigürasyonlarını yapıp programlayacağımız kartın özelliklerini seçerek uygun kartı bulabilir ya da “Part Number” kısmına kartın ismini yazarak programlamak istediğimiz kartı bulabiliriz.

Burada ben “Part Number” Kısmına STM32F407VG Disc. kullandığımından F407VG yazıyorum ve aradığım kartı listede gördüğümde üzerine tıklayıp “Next” diyerek bir sonraki adıma geçiyorum.



Ve sonrasında projeme isim vererek “Finish” diyorum.

Karşımıza konfigürasyonları yapacağımız .ioc penceresi gelecektir.

Burada kullandığımız kartdaki mikroişlemci tüm pinleri ile beraber görüntülenmekte. İster pinlerin üzerine tıklayarak sahip oldukları fonksiyonları görüntüleyebilir ve tek tek konfigüre edebilirsiniz. Ya da sol taraftaki “Categories” sekmesinden daha detaylı konfigüre edebilirsınız.

Biz öncelikle sistemimizin CLOCK konfigürasyonunu yapacağız. Bunun için öncelikle menüden harici osilatörü aktif ediyorum.

```
/ System Core  
> RCC  
    High Speed Clock (HSE) = Crystal/Ceramic Resonator
```

Bu konfigürasyonu yaptığımda işlemci üzerindeki **PH0** ve **PH1** pinlerinin **RCC_OSC_IN** ve **RCC_OSC_OUT** olarak değiştigini görüyoruz. Yani işlemcimizin harici osilatör pinleri aktif oldu.

Bu konfigürasyonu yapmamızın sebebi mikro işlemcinin içinde dahili olarak bulunan kristal osilatör **Internal Clock** yerine kart üzerinde harici olarak mikroişlemcimize bağlı olan 8 MHz lik harici kristal **External Clock** kullanmak istememiz. External Clock kullanmanın avantajı sistemin çalışma esnasındaki hassasiyetini daha iyi sağlamasıdır.

Pinout & Configuration

Clock Configuration

Project Manager

Tools

Pinout view

System view

RCC Mode and Configuration

Categories A-Z

System Core

High Speed Clock (HSE) Crystal/Ceramic Resonator

Low Speed Clock (LSE) Disable

DMA

GPIO

IWDG

NVIC

RCC

SYS

WWDG

Analog

Timers

Connectivity

Multimedia

Security

Computing

Middleware

RCC Parameters

Reset Configuration

User Constants

NVIC Settings

GPIO Settings

Configure the below parameters :

Search [Ctrl+F]

System Parameters

VDD voltage (V) 3.3 V

Instruction Cache Enabled

Prefetch Buffer Enabled

Data Cache Enabled

Flash Latency(WS) 0 WS (1 CPU cycle)

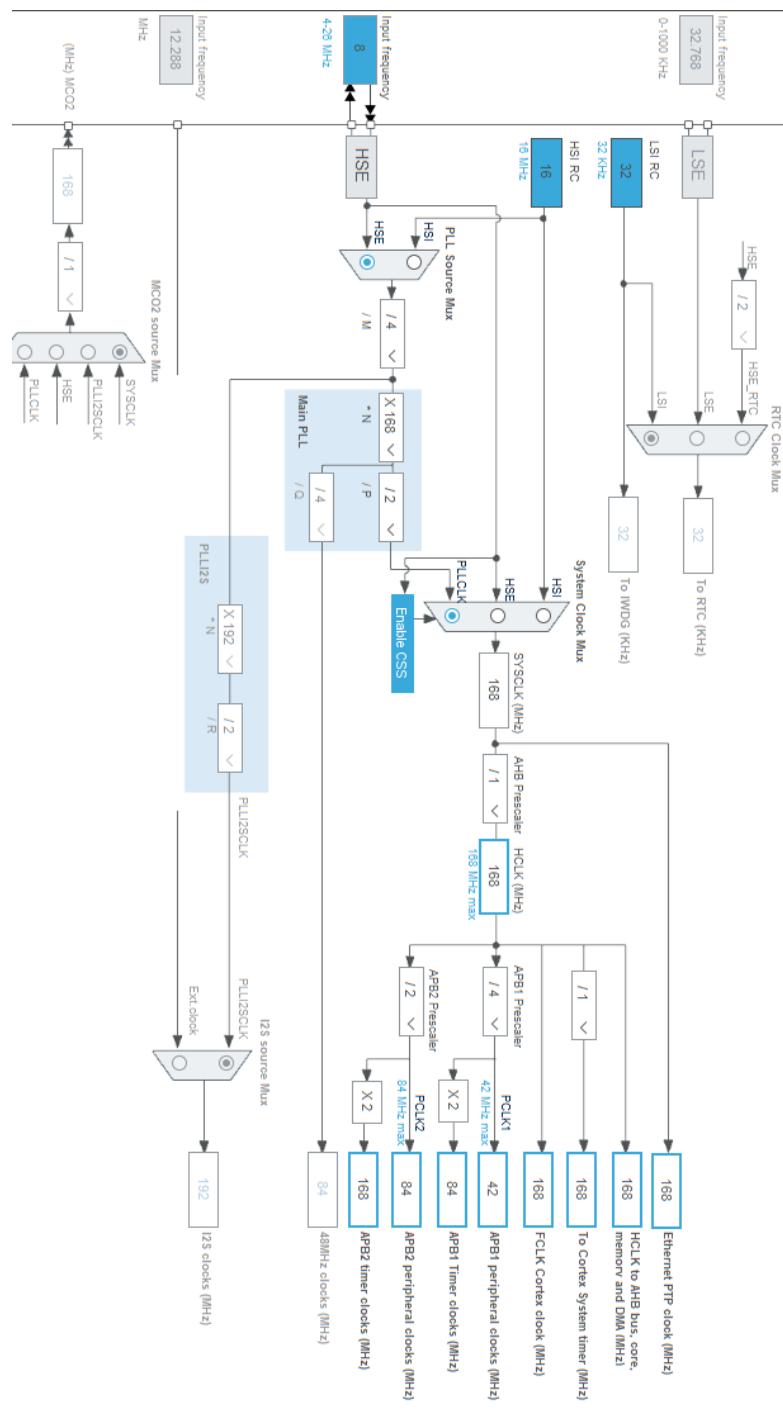
STM32F407VGTx
LQFP100

Pinout

P83	VSS	VDD
P84	P85	P86
P85	P86	P87
P86	P87	P88
P87	P88	BOOT1
P88	P89	P85
P89	P90	P84
P90	P91	P83
P91	P92	P82
P92	P93	P81
P93	P94	P80
P94	P95	P79
P95	P96	P78
P96	P97	P77
P97	P98	P76
P98	P99	P75
P99	P100	P74
P100	P101	P73
P101	P102	P72
P102	P103	P71
P103	P104	P70
P104	P105	P69
P105	P106	P68
P106	P107	P67
P107	P108	P66
P108	P109	P65
P109	P110	P64
P110	P111	P63
P111	P112	P62
P112	VCAP	VDDA
VDDA	VDD	P80
P80	P81	P82
P81	P82	P83
P82	P83	P84
P83	P84	P85
P84	P85	P86
P85	P86	P87
P86	P87	P88
P87	P88	P89
P88	P89	P90
P89	P90	P91
P90	P91	P92
P91	P92	P93
P92	P93	P94
P93	P94	P95
P94	P95	P96
P95	P96	P97
P96	P97	P98
P97	P98	P99
P98	P99	P100
P99	P100	P101
P100	P101	P102
P101	P102	P103
P102	P103	P104
P103	P104	P105
P104	P105	P106
P105	P106	P107
P106	P107	P108
P107	P108	P109
P108	P109	P110
P109	P110	P111
P110	P111	P112
P111	P112	VDD

Harici osilatörü aktif ettikten sonra “Clock & Configuration” sekmesine gelip saat frekans hızımızı ayarlamamız gereklidir. Ben sistemi maksimum hızda kullanacağım. Bunun için “HCLK (MHz)” değerini 168 MHz olarak ayarlamalıyım.

Öncesinde harici osilatörümüzü aktif ettiğimizden “Input Frequency” kısmını 8 MHz olarak değiştiriyorum çünkü kartımızın üzerindeki harici osilatör 8 MHz. Sonrasında “PLL Source Mux”’ı “HSE” ve “System Clock Mux”’ı “PLL” olarak ayarlıyorum. Ve HCLK değerimi 168 olarak değiştirmek ayarlarının yapılmasını bekliyorum.



Bu şekilde sistem clock konfigürasyonumu bitirmiş oldum. Burada dikkat edilmesi gereken sağ taraftaki modüllerin Clock Divisorları, bu şekilde bazı birimlerin farklı frekanslarda çalıştığını görebiliyoruz.

Şimdi sırası ile yapacağım diğer bir konfigürasyon, modlar arası geçiş sağlayacak olan Interrupt pini.

Kullanıcıdan gelen komut doğrultusunda sistemimiz Interrupt a girecek ve diğer moda geçecektir. Bu fonksiyonu kesme ile gerçekleştirmemın amacı modlar arası geçişin anında gerçekleşmesini sağlamak. Aynı fonksiyon bir pinin sadece button giriş'i olarak ayarlanması ile de gerçekleştirilebilir.

Kolaylık sağlama açısından zaten kartımın üzerinde bulunmakta olan USER butonunu kullanacağım. Bu buton PA0 pinine bağlı.

Bu durumda işlemcimin üzerine gelip bu pine tıklayarak dışardan bir kesme alacağım için **GPIO_EXTI0** olarak ayarlıyorum.

Pin N.	Signal o.	GPIO ou.	GPIO m.	GPIO Pu.	Maximu...	User Label	Modified
PA0-WKUP	n/a	n/a	External...	Pull-down	n/a		<input checked="" type="checkbox"/>

USER buton kullanıldığında ARC oluşumunu önlemek için GPIO ayarlarından Pull-Down seçiyorum. Bu buton zaten kart üzerinde default olarak Pull-Down bağlı.

Pini kesme olarak ayarladığım için NVIC yani kesme yöneticisini de GPIO için aktif hale getirmem gerekiyor.

Sıradaki işlem ADC okuma fonksiyonu. Cihazın hızını ayarlamak için motorlara vereceği PWM pulse değerini ADC1 birimini, ileri, geri, sağ, sol yön kontrol işlemleri için ADC2 ve kendi ekseni etrafında dönmesi için de ADC3 birimini kullanacağım.

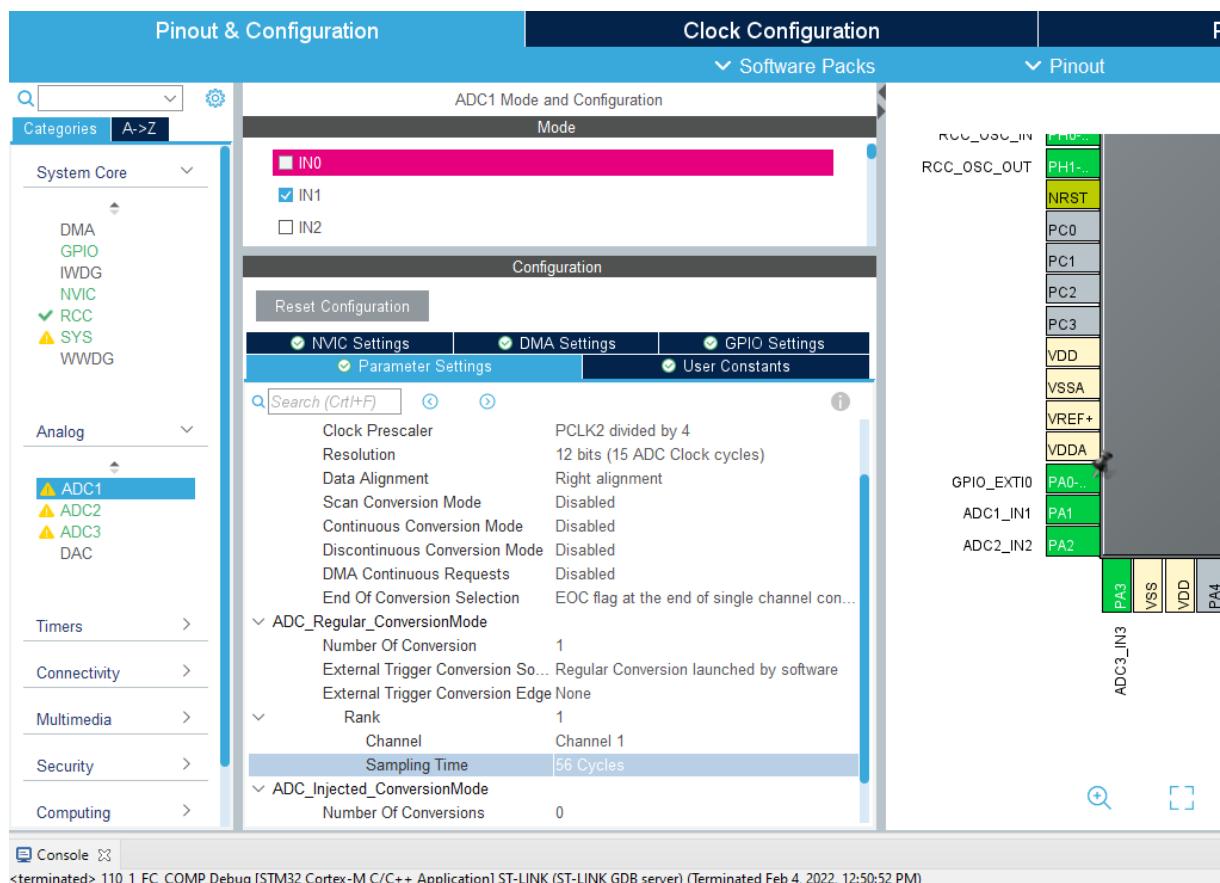
ADC birimlerinin ilk pini PA0 pini idi fakat biz bu pini USER butonu ile External Interrupt işlemi için kullandık.

Bu durumda ADC1 için Channel 1 olarak IN1 yani PA1

ADC2 için Channel 2 olarak IN2 yani PA2

ve ADC3 için Channel 3 olarak IN3 yani PA3 pinlerini kullanacağım.

12 Bitlik çözünürlükte okuma yapacağım ve bir okuma bittiğinde diğer okumayı yapması için “Continuous Conversion Mode” u ENABLE ediyorum ve EOC bayrağını tek bir okuma sonunda resetlenmesi için ayarlıyorum. 56 Cycle da bir okuma yapacağım.

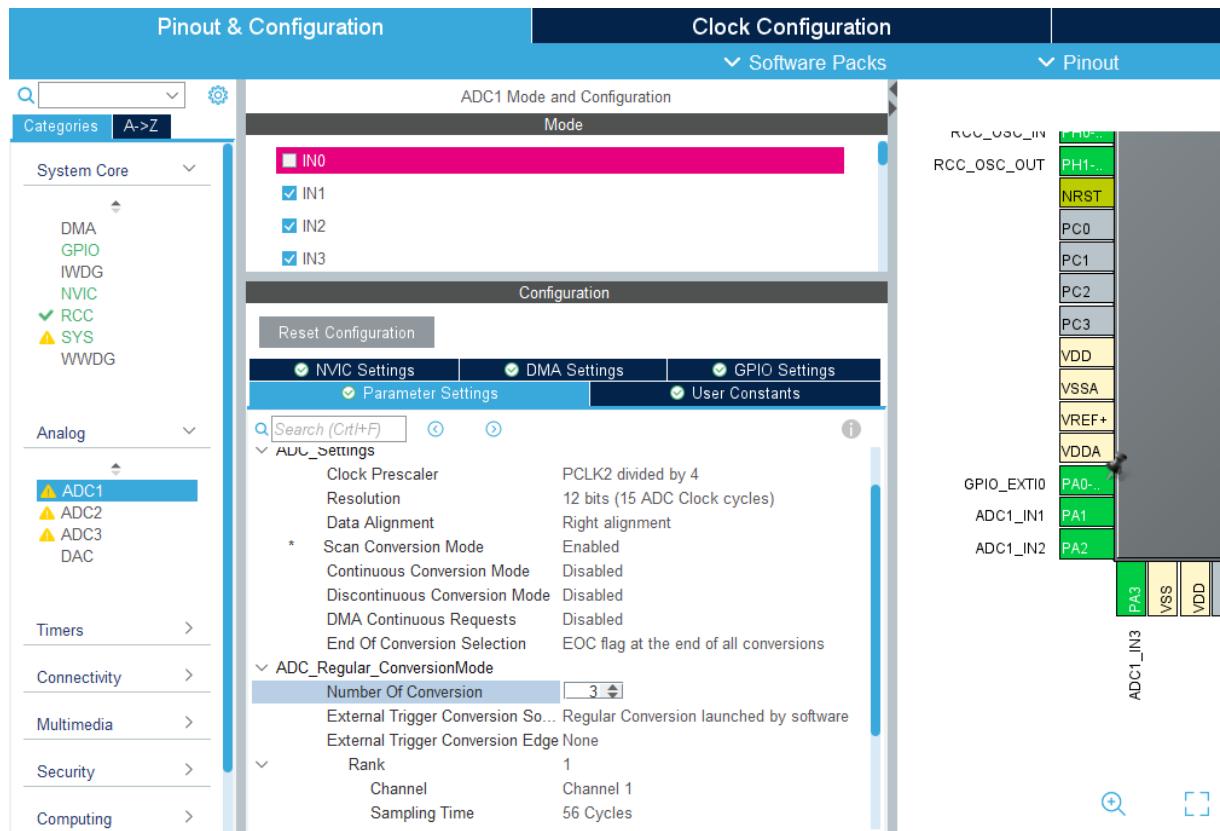


Bütün ADC birimlerim için aynı konfigürasyonu uyguluyorum.

ADC birimlerinin okunması işlemi tek bir ADC birimi üzerinden ve kesmeler ile yapılabiliirdi.

Bu durumda sadece ADC1 birimini kullanacağımı farzedersek ADC1 için IN0 IN1 ve IN2 kanallarını Channel1, Channel2 ve Channel3 olarak PA1, PA2 ve PA3 pinleri için aktif edecektim.

Burada önceki konfigürasyondan farklı olarak ayriyeten “Scan Conversion Mode” u kanallar arasında gezinerek okuması için aktif edicektim ve EOC bayrağını bütün çevrimler bittiğinde resetlenmesi için ayarlayacaktım.



Şimdi sıra TIMER biriminde. 4 motora 4 farklı PWM sinyali göndereceğim. Bu işlem için TIM4 birimini kullanacağım. Timer biriminin clock frekansı Div2 parametresinden dolayı 84 Mhzdi, bu durumda benim PWM sinyalimi ayarlarken kullanacağım method:

$$Period = \left(\frac{\text{Timer_Tick_Freq}}{\text{PWM_Freq}} \right) - 1$$

$$PWM_Freq = \frac{\text{Timer_Tick_Freq}}{(\text{Period} + 1)}$$

$$Timer_Tick_Freq = \frac{\text{Timer_CLK}}{(\text{Prescaler} + 1)}$$

Ben PWM sinyalimi bir servo motor için ayarlayacağım. Fırçasız motorların hassasiyetini artırmak için parametreler ile oynanabilir.

Misal SG90s Micro Servo için gerekli olan PWM frekansı 50 Hz ve periyodu 20 ms. Ve benim Timer Clock frekansım 84 Mhz.

Bu durumda;

$$Timer_Tick_Freq = \frac{84.000.000}{84}$$

$$Timer_Tick_Freq = 1.000.000 \text{ Hz}$$

“Timer_Tick_Freq” değerini artık bildiğimize göre şimdi PWM frekansımızı bilmek istiyoruz. “Period” değerimiz zaten bilindiğinden “PWM_Freq” değerimiz :

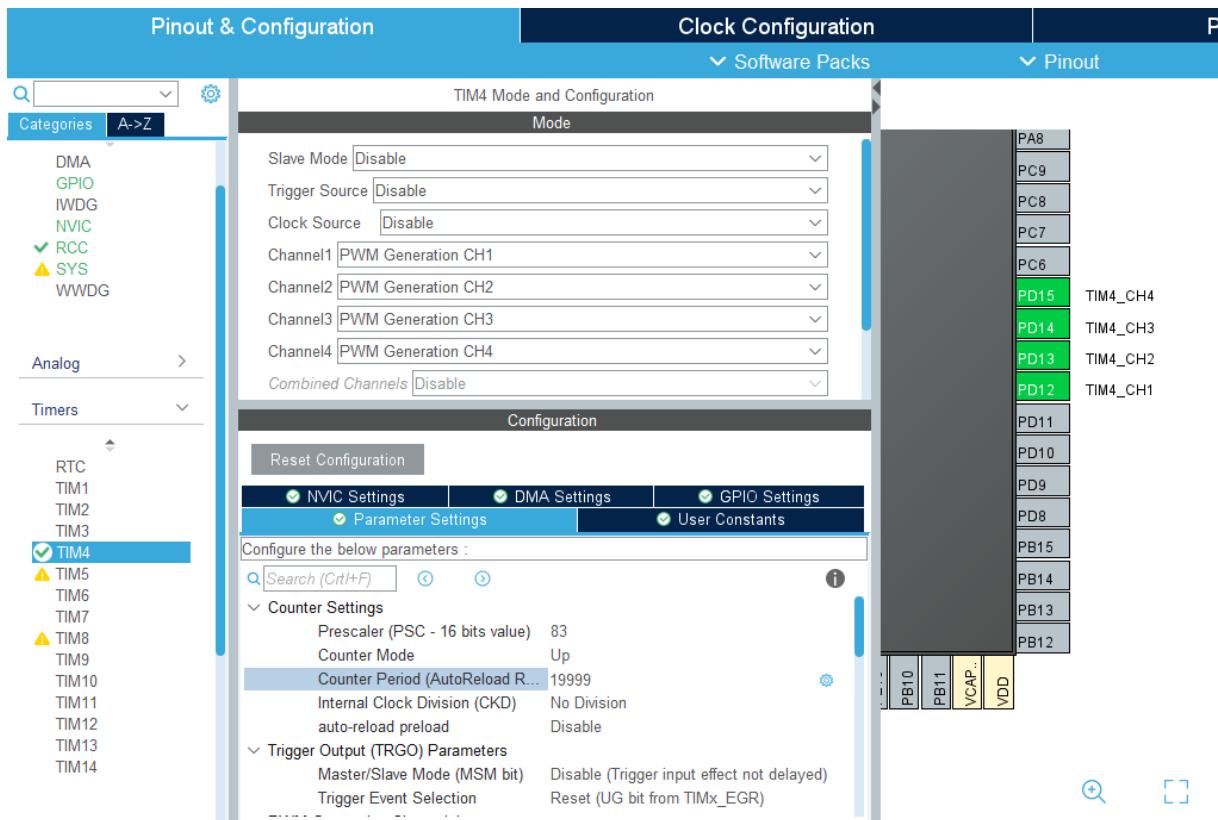
$$PWM_Freq = \frac{1.000.000}{20.000}$$

$$PWM_{Freq} = 50 \text{ Hz}$$

Şimdi TIM4 birimini bu şekilde konfigüre edecek olursak;

$$\text{Prescaler} = 83$$

$$\text{Period} = 19999$$

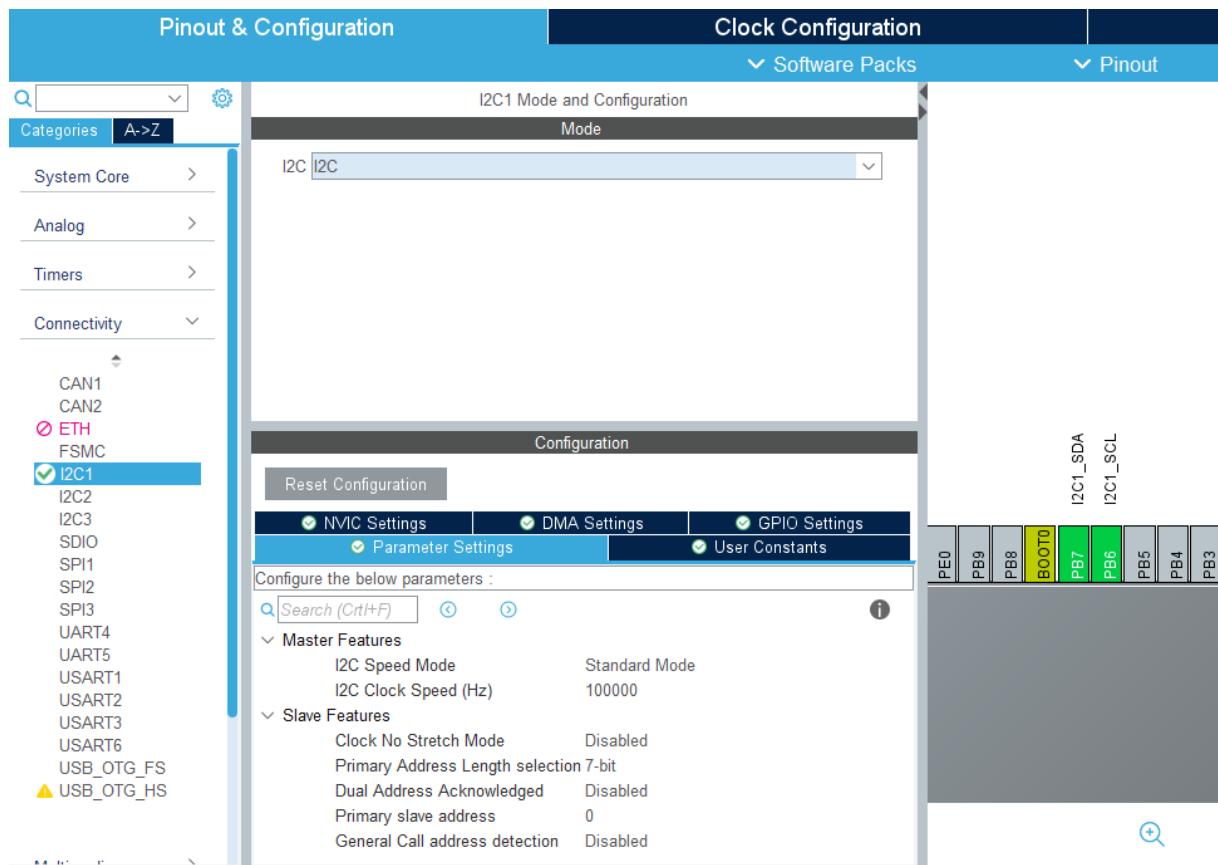


4 Farklı motor için 4 farklı PWM sinyali üreteceğimizi söylemiştim.

Bu durumda TIM4 ün Channel1, Channel2, Channel3 ve Channel4 kanallarının hepsini PWM Generation CHx olarak ayarlıyorum.

Bu konfigürasyonu yaptığımda PD12, PD13, PD14 ve PD15 pinleri TIM4 ün kanalları olmuş oluyor.

Son olarak MPU6050 sensörü ile haberleşebilmek için Connectivity kısmından I2C1 birimini aktif ediyorum.



Konfigürasyonlarımızı bu şekilde bitirmiş oluyorum. Artık kodlamaya geçebilirim. Bunun için .ioc da yaptığım konfigürasyonları GENERATE ile koda dönüştürmem lazım.



Ve bundan sonrasında proje dosyamız tam anlamı ile oluşturulmuş olacak ve main.c dosyamızda kodlamaya başlayabileceğiz.

3.2.2.2 MPU6050 Gyro Sensöründen Verilerin Okunması

Bu işlem için önce MPU6050 sensörünü tanıyalım.

MPU6050 içerisinde 3 eksen gyroscope, 3 eksen ivmeölçer ve bir sıcaklık sensörü bulunduruyor. Tabii biz burada gyro ve ivme ölçeri niçin kullanıyoruz?

İvme ölçer, hareket ettiği zaman üzerine ivme etki edecek eksenlerinde, bizim drone eksenlerine baktığımızda YAW, PITCH, ROLL eksenleri ile ilgilenildiğinde hava araçlarında bu temel eksenler vardır.

Burada temel amacımız ortaya bir uçuş kontrol kartı yazılımı çıkarmak, tabii bu amaçla bizim bir aim bilgisine ihtiyacımız var. Biz bu ihtiyacımızı MPU6050 ile gidereceğiz.

İvmeölçer bir ham veri çıktısı verir her eksen için, biz bunları basit işlemlere sokarak açı değerine ulaşırabiliyoruz. Fakat gyro da bu şekilde işlemiyor, gyro bize ham veriyi veriyor fakat veri gönderilirken derece/sn ye çevriliyor,

Biz açısal hızda saniye istemediğimiz için ve direk dereceye odaklandığımız yani derece verisini elde etmek istediğimiz için saniyeyi iki örnekleme süresi arasında yani gyrodan her veri aldığımızda o iki verinin arasında geçen süreye biz örnekleme zamanı diyoruz.

Bu süre ile çarparak yani aslında birnevi integralini alarak biz aslında bir açı değeri elde ediyoruz.

```

50/* Private define -----
51 /* USER CODE BEGIN PD */
52 #define MPU6050_ADDR 0x68<<1
53 #define PWR_MGMT_1_REG 0x6B
54 #define SMPLRT_DIV_REG 0x19
55 #define WHO_AM_I_REG 0x75
56 #define ACCEL_CONFIG_REG 0x1C
57 #define GYRO_CONFIG_REG 0x1B
58 #define LPF_REG 0x1A
59 #define RAD_TO_DEG 57.295779513082320876798154814105
60 #define MPU6050_GYRO_SENS_250 ((float) 131)
61 #define MPU6050_GYRO_SENS_500 ((float) 65.5)
62 #define MPU6050_GYRO_SENS_1000 ((float) 32.8)
63 #define MPU6050_GYRO_SENS_2000 ((float) 16.4)
64 #define MPU6050_ACCE_SENS_2 ((float) 16384)
65 #define MPU6050_ACCE_SENS_4 ((float) 8192)
66 #define MPU6050_ACCE_SENS_8 ((float) 4096)
67 #define MPU6050_ACCE_SENS_16 ((float) 2048)
68 /* USER CODE END PD */

```

Define kısmında sayısal olarak bir değeri bir kaç yerde girmek yerine daha anlaşılır bir kod elde etmek için misal 0x68 in 1 bit sola kaydırılmış halini "MPU6050_ADDR" olarak çağrıdığımda aynı görevi yapıyorlar.

I2C haberleşmesinde kullanacağımız MPU6050-Slave adresi 0x68 dir. I2C de adreslemeyi 7 bit olarak ayarladığımız için bu veriyi 0x68<<1 olarak bir bit sola kaydırıyoruz. Değerlikli kısmın kırılmaması için.

Bizim normalde haberleşme protokollerleri ile alıp gönderdiğimiz veriler 8 bit olarak evrenseldir. Biz bu 8 bit veriyi alıp gönderirken bu haberleşme adres gönderme kısmında farklı bir durum ortaya çıkıyor.

Bu durumda 0. bitin durum biti olması (protokol kendine ayırmış ~ R/W). Diyor ki ben bu slave adresi ile iletişim kurucam, ben bu slave adresindeki cihaza veri mi yazıcam yoksa bu cihazdan bir veri mi okuyacağım, işte bu durumu belirtmek için R/W durum bitini taşıyan bir biti var adresin, bu yüzden bize adres girmek için 8 bitlik pakette 7 tane bit kalıyor.

Tabii biz bunu yine böyle kullanamıyoruz, görüldüğü üzere bizim 0. bit durum biti olduğu için 1. den 7. bite kadar olan kısmı biz kullanabiliyoruz. Ve bu da bize kaydırma işlemini anımsatmalı, 1 bit kaydırıldıktan sonra bu 0. bit (en sağdaki bit) hariç soldaki 7 biti kullanabileceğimiz anlamına geliyor. Bu nedenle biz bu adresi 1 bit sola kaydırık.

Görüldüğü üzere ben burada #define edilmiş değerleri bir çok yerde kullanacağım için bunları #define lar ile birnevi sözel karakterlere atamış oldum.

Aynı şekilde kodu başka bir şekilde başka insanlar okuduğu zaman da kolaylık sağlayabilir. Burada gördüğümüz sayısal değerler aslında adresler ve sabitler.

```

84 /* USER CODE BEGIN PV */
85 uint8_t Data, Check = 104;
86 uint8_t Acc_Set[6], Gyro_Set[6];
87 int16_t Gyro_Raw[3], Acc_Raw[3];
88 float Gyro_Cal[3];
89 int16_t Acc_Total_Vector;
90 float Gyro_Pitch_Angle, Gyro_Roll_Angle;
91 float Acc_Pitch_Angle, Acc_Roll_Angle;
92 float Pitch_Angle, Roll_Angle;
93 int i;
94 float PrevTime;
95 float PrevTime_1, Time_1, ElapsedTime_1;
96 float PrevTime_Cal, Time_Cal, ElapsedTime_Cal;
97 uint8_t Address;
98 float Gyro_Mult;
99 float Acce_Mult;
100 HAL_StatusTypeDef Set_Gyro;

```

Burada yazılım için tanımladığım değişkenler mevcut. Kısaca özetleyecek olursak sonrasında işlemek üzere kaydettiğimiz/okduğumuz değerleri bu şekilde listelerde tutuyoruz. Bu şekilde listeler kullanarak çok fazla değişken tanımlamaktan (çok fazla değişkenden kasıt mesela "Gyro_Raw[3]" te 0.,1. ve 2. indexlerde olmak üzere 3 tane eleman var. 0. index X ekseninin, 1. index Y ekseninin ve 2. indexte Z ekseninin verilerini tutuyor.) yani tek tek bunları tanımlamaktansa listeye atmak daha mantıklı. Ve burada değişken tipleri de çok önemli.

Bu değişken tipleri dökümantasyonlarda belirtilir zaten. Bu değişken tiplerini iyi belirlemek lazım, mesela int16_t işaretli 16 bit yani -32k dan +32k ya kadar 65k lık bir değer aralığına sahip bir değişken tipidir bu. uint8_t kısmı ise data yani adres kısmı, bunun pozitif olması lazım adres negatif olmuyor. Bu yüzden unsigned integer 8 _t şeklinde tanımlıyoruz. Tabii bizim iletişim protokollerimizde 8 bit kullanır, 8 bitlik paketlerle haberleşir, o yüzden 8 bitlik uint bir değişken tanımladık.

```

131/* Private user code -----*/
132 /* USER CODE BEGIN 0 */
133 void MPU6050_Initialize(I2C_HandleTypeDef *hi2c, MPU6050_Accelerometer AccSens, MPU6050_Gyroscope GyroSens)
134 {
135     HAL_I2C_IsDeviceReady(hi2c, MPU6050_ADDR, 3, HAL_MAX_DELAY);
136
137     if(HAL_I2C_Mem_Read(hi2c, MPU6050_ADDR, WHO_AM_I_REG, 1, &Check, 1, HAL_MAX_DELAY))
138     {
139
140         Data = 0x00;
141         HAL_I2C_Mem_Write(hi2c, MPU6050_ADDR, PWR_MGMT_1_REG, 1, &Data, 1, HAL_MAX_DELAY);
142
143         Data = (uint8_t)AccSens;
144         HAL_I2C_Mem_Write(hi2c, MPU6050_ADDR, ACCEL_CONFIG_REG, 1, &Data, 1, HAL_MAX_DELAY);
145
146         Data = (uint8_t)GyroSens;
147         HAL_I2C_Mem_Write(hi2c, MPU6050_ADDR, GYRO_CONFIG_REG, 1, &Data, 1, HAL_MAX_DELAY);
148     }
149     switch (AccSens) {
150         case MPU6050_Accelerometer_2G:
151             Acce_Mult = (float)1 / MPU6050_ACCE_SENS_2;
152             break;
153         case MPU6050_Accelerometer_4G:
154             Acce_Mult = (float)1 / MPU6050_ACCE_SENS_4;
155             break;
156         case MPU6050_Accelerometer_8G:
157             Acce_Mult = (float)1 / MPU6050_ACCE_SENS_8;
158             break;
159         case MPU6050_Accelerometer_16G:
160             Acce_Mult = (float)1 / MPU6050_ACCE_SENS_16;
161         default:
162             break;
163     }
164
165     switch (GyroSens) {
166         case MPU6050_Gyroscope_250s:
167             Gyro_Mult = (float)1 / MPU6050_GYRO_SENS_250;
168             break;
169         case MPU6050_Gyroscope_500s:
170             Gyro_Mult = (float)1 / MPU6050_GYRO_SENS_500;
171             break;
172         case MPU6050_Gyroscope_1000s:
173             Gyro_Mult = (float)1 / MPU6050_GYRO_SENS_1000;
174             break;
175         case MPU6050_Gyroscope_2000s:
176             Gyro_Mult = (float)1 / MPU6050_GYRO_SENS_2000;
177         default:
178             break;
179     }
180 }
181 void MPU6050_Calibration(I2C_HandleTypeDef *hi2c)
182 {
183     for(i=0; i<2000; i++)
184     {
185         PrevTime_Cal = Time_Cal;
186         Time_Cal = HAL_GetTick();
187         ElapsedTime_Cal = (Time_Cal-PrevTime_Cal)*1000;
188
189         Gyro_Set[0]=0x43;
190         HAL_I2C_Master_Transmit(hi2c, MPU6050_ADDR, Gyro_Set, 1, HAL_MAX_DELAY);
191         HAL_I2C_Master_Receive(hi2c, MPU6050_ADDR, Gyro_Set, 6, HAL_MAX_DELAY);
192
193         Gyro_Raw[0] = (Gyro_Set[0] << 8 | Gyro_Set[1]);
194         Gyro_Raw[1] = (Gyro_Set[2] << 8 | Gyro_Set[3]);
195         Gyro_Raw[2] = (Gyro_Set[4] << 8 | Gyro_Set[5]);
196
197         Gyro_Cal[0] += Gyro_Raw[0];
198         Gyro_Cal[1] += Gyro_Raw[1];
199         Gyro_Cal[2] += Gyro_Raw[2];
200
201         HAL_Delay(3);
202     }
203
204     Gyro_Cal[0] /= 2000;
205     Gyro_Cal[1] /= 2000;
206     Gyro_Cal[2] /= 2000;
207 }

```

```

212 void MPU6050_ComputeAngle(I2C_HandleTypeDef *hi2c)
213 {
214     PrevTime_1 = Time_1;
215     Time_1 = HAL_GetTick();
216     ElapsedTime_1 = (Time_1 - PrevTime_1) * 1000;
217
218     Acc_Set[0] = 0x38;
219     HAL_I2C_Master_Transmit(&hi2c1, MPU6050_ADDR, Acc_Set, 1, HAL_MAX_DELAY);
220     HAL_I2C_Master_Receive(&hi2c1, MPU6050_ADDR, Acc_Set, 6, HAL_MAX_DELAY);
221
222     Acc_Raw[0] = (Acc_Set[0] << 8 | Acc_Set[1]);
223     Acc_Raw[1] = (Acc_Set[2] << 8 | Acc_Set[3]);
224     Acc_Raw[2] = (Acc_Set[4] << 8 | Acc_Set[5]);
225
226     Gyro_Set[0] = 0x43;
227     HAL_I2C_Master_Transmit(&hi2c1, MPU6050_ADDR, Gyro_Set, 1, HAL_MAX_DELAY);
228     HAL_I2C_Master_Receive(&hi2c1, MPU6050_ADDR, Gyro_Set, 6, HAL_MAX_DELAY);
229
230     Gyro_Raw[0] = (Gyro_Set[0] << 8 | Gyro_Set[1]);
231     Gyro_Raw[1] = (Gyro_Set[2] << 8 | Gyro_Set[3]);
232     Gyro_Raw[2] = (Gyro_Set[4] << 8 | Gyro_Set[5]);
233
234     Gyro_Raw[0] -= Gyro_Cal[0];
235     Gyro_Raw[1] -= Gyro_Cal[1];
236     Gyro_Raw[2] -= Gyro_Cal[2];
237
238     Gyro_Pitch_Angle += Gyro_Raw[0] * Gyro_Mult * 0.004;
239     Gyro_Roll_Angle += Gyro_Raw[1] * Gyro_Mult * 0.004;
240
241     Gyro_Pitch_Angle += Gyro_Roll_Angle * sin(Gyro_Raw[2] * Gyro_Mult * 0.004 / RAD_TO_DEG);
242     Gyro_Roll_Angle -= Gyro_Pitch_Angle * sin(Gyro_Raw[2] * Gyro_Mult * 0.004 / RAD_TO_DEG);
243
244     Acc_Total_Vector = sqrt((Acc_Raw[0] * Acc_Raw[0]) + (Acc_Raw[1] * Acc_Raw[1]) + (Acc_Raw[2] * Acc_Raw[2]));
245
246     Acc_Pitch_Angle = asin((float)Acc_Raw[1]/Acc_Total_Vector)* RAD_TO_DEG;
247     Acc_Roll_Angle = asin((float)Acc_Raw[0]/Acc_Total_Vector)* - RAD_TO_DEG;
248
249     Acc_Pitch_Angle -= 0.00;
250     Acc_Roll_Angle -= 0.00;
251
252     if(Set_Gyro)
253     {
254         Pitch_Angle = Gyro_Pitch_Angle * 0.9996 + Acc_Pitch_Angle * 0.0004;
255         Roll_Angle = Gyro_Roll_Angle * 0.9996 + Acc_Roll_Angle * 0.0004;
256     }
257     else
258     {
259         Pitch_Angle = Acc_Pitch_Angle;
260         Set_Gyro = true;
261     }
262
263     while((HAL_GetTick() - PrevTime)*1000 < 4000);
264     PrevTime = HAL_GetTick();
265 }

```

main kısmının dışındaki değişkenler global değişkenlerdir, aynı dosya içerisinde her yerden ulaşabilirsiniz, geliyoruz fonksiyonumuzun ilk kısımlarını yazmaya.

Arduino dan bilirsiniz, değişkenleri tanımladınız, void setup ve void loop kısmı vardır. Void Setup kısmında her devre çalıştığında/program çalıştığında 1 kez çalıştırılması gereken/1 sefer ayarlanması gereken kodlar vardır, bunları Void Setup() kısmına yazazız. Sürekli, devre/kart çalıştıkça sürekli işlenmesi gereken verileri de Void Loop (döngü) içerisinde yazarız. Burada da aynı mantık. int main kısmının içerisinde USER CODE BEGIN kısımlarında tek seferlik çalışacak kodlarımızı yazıyoruz. Bunlar her çalıştırıldığında veya her resetlendiğinde tek sefer çalıştırılıyor.

Bunlardan sonra while döngüsüne giriyor, while döngüsü bizim sürekli çalışan kısmımız (void loop gibi).

Burada yani setup/main kısmında belirli konfigürasyonlar yapmamız lazım, stm32 kartımızın konfigürasyonlarını yaptık, main kısmında kod yazılımına geçtik.

Kodlamanın başlarında kodlamayı sabit değerler üzerinden yaptım, bu sebepten burada da anlaşılmasının için bu şekilde anlatacağım. Daha sonra fonksiyonları bir bütün şeklinde açıklayarak son halleri üzerinde duracağım.

Bundan sonra MPU6050 nin konfigürasyonlarını yapmamız lazım. Nedir bu konfigürasyonlar, belirli registerleri ayarlamamız lazım, örneğin ivme ölçerin hassasiyeti, gyronun hassasiyeti veya güç ayarlamaları MPU6050 sensörü kartının. İşte burada ki 3 fonksiyonla bunları temel düzeyde ayarlayabiliyoruz.

4.28 Register 107 – Power Management 1 PWR_MGMT_1

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE_RESET	SLEEP	CYCLE	.	TEMP_DIS	CLKSEL[2:0]		

Görüldüğü üzere her bitin veya kombinasyonlarının tuttuğu belli ayarlar var, misal 3. bitin tuttuğu ayar 1 olursa Temperature yani sıcaklık sensörü DISABLE olur. Bit 4 zaten rezerved. Bit 6 da eğer cihaz uyku modunda kullanılmak isteniyorsa sleep kısmı aktif edilebilir ben 0 yapıyorum. Diğer ayarlarda aynı şekilde 0, yani bu 0 olarak ayarlanması gereken bir register. Tabii hepsi 0 olduğundan hex olarak ta 0x00 değeri vereceğiz.

```
140     Data = 0x00;
141     HAL_I2C_Mem_Write(hi2c, MPU6050_ADDR, PWR_MGMT_1_REG, 1, &Data, 1, HAL_MAX_DELAY);
```

kodda da görüldüğü üzere "data" değişkeni bir veriyi tutuyor. Fonksiyon ile bizim bunu register a yazmamız lazım. Bu fonksiyon da tabii HAL kütüphanesinin I2C kısmının "Memory Write" yani orada ki hafıza birimine yazılması işleminden oluşuyor.

Fonksiyona gelecek olursak almakta olduğu parametreleri görebiliyoruz :

1. parametre : ilgilendiğiniz, üzerinde işlem yaptığınız birimin point olarak adresinin dönderilmesi, biz 3 I2C modülünden I2C1 modülünü kullanıyoruz.
2. parametre : bizim haberleşmek istediğimiz cihazın slave adresi,
3. parametre : power management 1 registerının adresi yani haberleştiğimiz cihazda ki yazım yapacağımız yazmaç
4. parametre : yazacağımız bellekteki/ belleğin içeriisndeki hücreler ne kadar, hücrelerin boyutu ne kadar, 1 BYTE
5. parametre : göndereceğimiz değer i bana yazma pointer olarak gönder diyor, göndereceğimiz değer "data",
6. parametre : yine bu sefer gönderdiğimiz verinin boyutu, 1 byte yine.
7. parametre : timeout süresi, bu fonksiyonun çalışması için belirli bir süre belirler, ben burada misal 100 dersem 100 ms içerisinde bu fonksiyon gerçekleştmezse eğer bir daha bu fonksiyonu gerçekleştirmek için uğraşma ve bu fonksiyondan çık. Bizim yazdığımız HAL_MAX_DELAY kısmı da işte gereken miktarı kendisi derleyici otomatik olarak veriyor.

Bu şekilde biz PWR_MGMT_1_REG yani Power Management Register 1'i ayarladık.

Devamında aynı şekilde GYRO'nun ve ivmeölçerin de ayarlamaları var. Bunlara gelecek olursak misal niye burada "data" kısmında öncesinde 0x00 iken burada

0x08 veya ACC için 0x10.

Bunlar teker teker işlendiği için datanın sürekli değerinin değişmesi bir şey ifade etmiyor çünkü değiştirdikten hemen sonra fonksiyonlar ile biz gereken yerlere öncesinde zaten göndereceğimizi göndermişiz.

Hemen GYRO config ayarlarına gelelim.

4.4 Register 27 – Gyroscope Configuration GYRO_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]	-	-	-	-

Burada ki 0 dan 2 ye kadar olan 3 bitlik kısım reserved.

Bize 3. ve 4. biti FS_SEL şeklinde ayarlamak için vermiş. Burada ölçüm skalamız var. Açısal hızımızı derece olarak saniye başına ölçebileceğimiz farklı aralıkların verildiği bir ölçüm skalamız var. Biz burada +500 /s değerini kullanacağız (FS_SEL = 01). Benim için yeterli bir değer.

7. 6. 5. Bitleri de şu an 0 gireceğiz.

Bu durumda bizim bu register için değerimiz 0000 1000 b = 0x08 H

Bu durumda GYRO konfigürasyonu için kullanacağım fonksiyon şekli ;

```
146     Data = (uint8_t)GyroSens;
147     HAL_I2C_Mem_Write(hi2c, MPU6050_ADDR, GYRO_CONFIG_REG, 1, &Data, 1, HAL_MAX_DELAY);
```

Ve gelelim ivmeölçer konfigürasyonuna.

4.5 Register 28 – Accelerometer Configuration ACCEL_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]	-	-	-	-

Gyroya benzer ayarlar mevcut. Yine 0:3 bitleri sistem tarafından rezerve edilmiş. Bu nedenle 0 0 0 olarak kabul ettik.

7. 6. ve 5. bitlerde yine aynı şekilde 0 0 0 olarak belirledik.

Burada yine 2 tane bit ayarlamamız gerekiyor. AFS_SEL ölçüm skalasına bakacak olursak +-2g den +-16g gibi farklı aralıklarda ölçüm aralıkları mevcut. Benim burada seçeceğim. +-8g olacak (10);

```
143     Data = (uint8_t)AccSens;
144     HAL_I2C_Mem_Write(hi2c, MPU6050_ADDR, ACCEL_CONFIG_REG, 1, &Data, 1, HAL_MAX_DELAY);
```

Yani 0001 0000 b = 0x10 H yani benim data = 0x10 dememim sebebi register a göndermem gereken hex kodunun bu olması.

Yani 0001 0000 b = 0x10 H yani benim data = 0x10 dememim sebebi register a göndermem gereken hex kodunun bu olması.

Bu ayarlamalar ile bile aslında biz MPU6050 sensörümüzü hangi parametrelere bağlı kalara ölçüm yapmasını belirttik. Bundan sonra yapacağımız iş, gyro kısmını ile ilgili.

Burada gyrolar ile ilgili bazı sıkıntılar var, durduğun yerde hiç dönmediği/herhangi bir hızda dönmediği zaman bile belirli değerler dönderiyor. Tabii bu değerler aşağı yukarı birbirine yakın/sabit değerler. Bizim bunları hesap edip ölçümlerimizden bu sabit değerleri çıkartmamız lazım ki ölçümlerimiz temiz, net, güvenli olsun.

Buna OFFSET işlemi diyebiliriz (başlangıç hatası da diyebiliriz.). Yani herhangi bir ölçüm gerektirecek bir durum yokken bir değer döndürüyor bana. Bu değer ölçüm yaparken de dönderilecek, bu değerin üzerine benim ölçümlerim kurulacağı için benim o toplam değişkenden offset/hata sabitlerini çıkarmam lazım.

Peki bunu nasıl yapacağız? Ortalama metodu ile; Nedir? Bu çalışmada ki ne göre örneğin 2000 tane döngüden verilerimi alıcam 2000 farklı verimi, bunları teker teker toplayıp 2000 e böleceğim ki elimde 2000 ölçümün ortalamasına ait bir değer oluşacak. Ve bu şekilde öz kalibrasyon değerleri elde etmiş olacağız

```

183 void MPU6050_Calibration(I2C_HandleTypeDef *hi2c)
184 {
185     for(i=0; i<2000; i++)
186     {
187         PrevTime_Cal = Time_Cal;
188         Time_Cal = HAL_GetTick();
189         ElapsedTime_Cal = (Time_Cal-PrevTime_Cal)*1000;
190
191         Gyro_Set[0]=0x43;
192         HAL_I2C_Master_Transmit(hi2c, MPU6050_ADDR, Gyro_Set, 1, HAL_MAX_DELAY);
193         HAL_I2C_Master_Receive(hi2c, MPU6050_ADDR, Gyro_Set, 6, HAL_MAX_DELAY);
194
195         Gyro_Raw[0] = (Gyro_Set[0] << 8 | Gyro_Set[1]);
196         Gyro_Raw[1] = (Gyro_Set[2] << 8 | Gyro_Set[3]);
197         Gyro_Raw[2] = (Gyro_Set[4] << 8 | Gyro_Set[5]);
198
199         Gyro_Cal[0] += Gyro_Raw[0];
200         Gyro_Cal[1] += Gyro_Raw[1];
201         Gyro_Cal[2] += Gyro_Raw[2];
202
203         HAL_Delay(3);
204     }
205
206     Gyro_Cal[0] /= 2000;
207     Gyro_Cal[1] /= 2000;
208     Gyro_Cal[2] /= 2000;
209 }
```

ilk satırında "Gyro_Set[0]=0x43;" demişiz, yani "cuffer" listesinin 0. index ine 0x43 değerini atmışız. Peki neden?

Daha öncesinde "HAL_I2C_MemWrite" yani memory-write fonksiyonları idi, bunlar ise Master_Transmit ve Master_Receive yani ben STM32 mi Master cihaz olarak kullanıyorum. Slave olarak ta MPU6050 sensörümü kullanıyorum. Ben Master cihaz olarak diyorum ki slave den veri alacağım, önce bunun haberini gönderiyorum.

Yine I2C protokolü ile bir pointer kullanıyorum hangi birimi kullandığımı belirtmek için. Cihazımın slave adresine giriyorum (0x68). Ve sonrasında listeye giriyorum (Gyro_Set), aslında bu "data".

önceki anlattığımız kısımda da "uint8_t *pData" vardı, burada da aynı şekilde, yani diyor ki benim haber vereceğim, içerisindeinden veri alacağım register in adresini bana söyle. E peki biz bunu yukarıda pointer olarak gönderdik? Listelerde 0. index olarak tutulurken ve 0. index bir adres tutuyor aslında (0x43), yani direk verinin kendisini göndermiyor, 0. indexte tutulan verinin adresini gönderiyor, bu yüzden ben buraya "&" koymuyorum, direkt pointer olarak dönderiyorum zaten listelerde.

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT_H	R								ACCEL_XOUT[15:8]
3C	60	ACCEL_XOUT_L	R								ACCEL_XOUT[7:0]
3D	61	ACCEL_YOUT_H	R								ACCEL_YOUT[15:8]
3E	62	ACCEL_YOUT_L	R								ACCEL_YOUT[7:0]
3F	63	ACCEL_ZOUT_H	R								ACCEL_ZOUT[15:8]
40	64	ACCEL_ZOUT_L	R								ACCEL_ZOUT[7:0]
41	65	TEMP_OUT_H	R								TEMP_OUT[15:8]
42	66	TEMP_OUT_L	R								TEMP_OUT[7:0]
43	67	GYRO_XOUT_H	R								GYRO_XOUT[15:8]
44	68	GYRO_XOUT_L	R								GYRO_XOUT[7:0]
45	69	GYRO_YOUT_H	R								GYRO_YOUT[15:8]
46	70	GYRO_YOUT_L	R								GYRO_YOUT[7:0]
47	71	GYRO_ZOUT_H	R								GYRO_ZOUT[15:8]
48	72	GYRO_ZOUT_L	R								GYRO_ZOUT[7:0]

Burada 43 adresi ile başlayıp 48 ile biten 6 tane register görüyoruz. Sistem 8 bitlik olduğu için ve veriler de 16 bitlik olduğu için tek seferde ifade edemiyoruz verileri, 2 tane 8 bitlik paketlere bölüyor, bizim bunu 16 bite dönüştürmemiz lazım sonrasında. Misal görüldüğü gibi H ve L olmak üzere 0 dan 15. bite kadar 16 bitlik veriler XOUT_H ve XOUT_L gibi olmak üzere 2 şerli 8 bitten oluşan registerlarda tutuluyor. Ve başlangıç adresi 43, yani 0x43 adresine sahip. O nedenle biz burada "cuffer[0]" için 0x43 yazdık ki biz bu adrese veri gönderiyoruz senden veri alacağım diy.

Yalnız ben burada 6 tane adresten/registerden veri alıcam, fakat burada ;

```
226 Gyro_Set[0] = 0x43;
227 HAL_I2C_Master_Transmit(&hi2c1, MPU6050_ADDR, Gyro_Set, 1, HAL_MAX_DELAY);
228 HAL_I2C_Master_Receive(&hi2c1, MPU6050_ADDR, Gyro_Set, 6, HAL_MAX_DELAY);
```

şeklinde koda diyoruz ki biz bu cuffer da ki adresse 1 byte lık veri gönderiyorum, diyorum ki bu register dan başla (cuffer[0] = 0x43)

Receive fonksiyonunda ise 6 olarak tanımlıyorum yani 6 tane register i oku. Yani şu başlangıç değerine sahip registerden başla, 6 tane ileri git ki zaten burada sıralı 43,44,45.. diye.

Yani burada ben bildirim yapıyorum, senden veri alıcam diyorum.

Receive fonksiyonunda ise diyorum ki ben bildirim yaptım, belirttiğim başlangıç register ından itibaren başlayarak 6 tane arka arkaya gelen (başlangıç dahil) register dan bana veri gönder. Sonra "Gyro_Set[]" listesine index olarak at diyorum. Yani 43 tekini 0. index e, 44 dekini 1. indexe ... 48 dekini 5. index e gibi.

Bu değerler Gyro_Set listesine atıldıktan sonra görüldüğü üzere yaptığımız bir takım işlemler mevcut. Misal ;

```
230 Gyro_Raw[0] = (Gyro_Set[0] << 8 | Gyro_Set[1]);
231 Gyro_Raw[1] = (Gyro_Set[2] << 8 | Gyro_Set[3]);
232 Gyro_Raw[2] = (Gyro_Set[4] << 8 | Gyro_Set[5]);
```

İşlemlerini ele alalım. Burada ne demisiz, cuffer[0] yani 0x43 register ından ve cuffer[1] yani 0x44 register ından gelen değerler yani X ekseninin değerleri bunlar. Değerleri 8 er bitlik 2 paket halinde gönderdiğimizi söylemişistik. İşte burada 0-7. bite kadar olan veri 0x44 register ından okunan ve 8-15. bite kadar olan veriler de 0x43 register ından okunmakta.

Ben de burada diyorum ki, Gyro_Set[0] indexini 8 bit sola kaydırır, niye? Çünkü gelen değer 0x43 deki 15:8 olacak ve bu geldiği gibi 16 bitlik bir değişkende ilk 8 biti yani 0:7 alanını kaplayacak. Bu yüksek değerlikli veri olduğundan benim bunu 15:8 alanına kaydılmam lazımdır, bu yüzden 8 bit sola kaydırarak ben bunu olması gereken yüksek değerlikli kısma yazdırıyorum. Sonra diyorum ki 0x44 registerinden gelen 7:0 verisini şimdilik 15:8 alanına kaydırduğımız 16 bitlik veri ile OR işlemine tabi tut.

Yani şu şekilde;

```
gyro_raw[0] = 0000 0000 0000 0000
```

yani 16 bitlik bir veri söz konusu

ben 0x43 ten okuduğum yüksek değerlikli 15:8 verisini bu değere atıyorum

```
gyro_raw[0] = 0000 0000 1010 1101
```

yüksek değerlikli veri ilk yazıldığından 7:0 bitlerine yazıldı.

bu benim yüksek değerlikli verim olduğundan yüksek değerlikli kısma kaydırıyorum

```
gyro_raw[0] = 1010 1101 0000 0000
```

daha sonra kalan düşük değerlikli okuduğum 0x44 registerındaki 7:0 değerini de bu değere yazıyorum

```
gyro_raw[0] = 1010 1101 1010 1011
```

ve bu şekilde ayrık olan 0x43 ve 0x44 registerlarındaki 8 bitlik 2 paketimi doğru sıra ile birleştirip tek bir 16 bitlik veri elde etmiş oluyorum.

Yani biz burada aslında X ekseninin 8 bitlik 2 paketi 16 bitlik bir dğeişkene atadık, o da gyro_raw[0] yani gyro raw X. Zaten en başta da gyro_raw değişkenini işaretli 16 bit bir dizi olarak tanımladık yani :

```
int16_t gyro_raw[3]..
```

Aynı işlemleri Y (Gyro_Set[2] 0x45 : yüksek değerlikli ve Gyro_Set[3] 0x46 düşük değerlikli 8 bitlik Y eksenleri paketleri) ve Z eksenleri için (Gyro_Set[4] 0x47: yüksek değerlikli ve cuffer[5] 0x48 düşük değerlikli 8 bitlik Z eksenleri paketleri) de tekrarlıyoruz.

ve bu şekilde Gyrodan verilerimizi aldık derece/sn olarak. Biz bunları sonra KALİBRASYON verisi oluşturmak amacıyla ile şu kısımlara atıyoruz;

```

195     Gyro_Raw[0] = (Gyro_Set[0] << 8 | Gyro_Set[1]);
196     Gyro_Raw[1] = (Gyro_Set[2] << 8 | Gyro_Set[3]);
197     Gyro_Raw[2] = (Gyro_Set[4] << 8 | Gyro_Set[5]);
198
199     Gyro_Cal[0] += Gyro_Raw[0];
200     Gyro_Cal[1] += Gyro_Raw[1];
201     Gyro_Cal[2] += Gyro_Raw[2];
```

Şimdi, 2000 değeri üst üste toplayacağım demiştim, değerlerim gyro_raw[3] listesinde. Ben bu değerleri sürekli bir tane değişkenin içerisine atıp toplamam lazım 2000 döngü boyunca. "gyro_cal" değişken ismi de Gyro Calibration listesinden geliyor. 0. index e X olmak üzere 1. ve 2. indexlerde de Y ve Z ekseni verilerini tutuyoruz. Gelen değerler teker teker toplanıp eşitleniyor, ve 2000 değer sonra burada toplanmış büyük bir veri oluşturuyor.

Bu kısımda 2000 döngü boyunca elde ettiğimiz verileri topladık. Ve şimdi bir ortalamasını almamız lazım. Yani 2000 e bölüm bu değeri eşitememiz lazım.

```
206 Gyro_Cal[0] /= 2000;
207 Gyro_Cal[1] /= 2000;
208 Gyro_Cal[2] /= 2000;
```

Yani benim burada değerim 2000 ise, 2000 değer toplandı ise gyro_cal[0] index inde, 2000 e bölüyor,.

Burada Gyro dan verileri alırken belirli bir süre/zaman aralığı olması lazım, ben bunu sabit bir şekilde gerçekleştiriyorum, burada anlatılan komutlar benim için sistemimde 1000 us kadar zaman almakta, bende gyro ölçümülerinde aralığını 4000 us olarak belirledim yani bu da 4000 ms yeye denk geliyor.

Her ölçüm aralığı 4 ms olsun dedim, burada for döngüsü içerisinde 1 ms sürdüğü için ben de HAL_Delay(3) fonksiyonu ile 3 ms daha ekliyorum ki 4 ms lik bir ölçüm aralığı elde edeyim, eğer ölçüm aralığı farklı olur ve bu sistemde kod yazarsanız değişkenleriniz/kalibrasyon offsetlemeniz doğru olmayacağından emin olun.

Burada bu şekilde ortalama alarak ta kalibrasyon offset lerimizi ayarladık.

Artık temel Loop kısmına yani Döngü/While kısmına geçebiliriz.

```
214 PrevTime_1 = Time_1;
215 Time_1 = HAL_GetTick();
216 ElapsedTime_1 = (Time_1 -PrevTime_1) * 1000;
```

Bu kısmı ne için kullanıyorum? Bu kod bütünü/ bu fonksiyon benim her while döngüsüne girdiğimde, yani bir while döngüsüne girdim, while döngüsü içerisindeki işlemlerin hepsi bitti, tekrar 2. kez girdim ve bu kod bloğuna geldi, bu 2 kez tekrarlanan işlemde geçen süreyi bu kod bloğu ile ben görebiliyorum. Ki gerçekten 4000 us geçmiş mi geçmemiş mi bu kod bloğu ile ben bunu anlayabiliyorum.

Devamında Acc_Set, Gyro_Set şeklinde ki listeler görülmekte. Aslında öncesinde anlattığım kullandığımız I2C modülü ile haberleşme ve sensörden okunan değerlerin 16 bitlik yeni değişkenlere atanması işinin aynısı.

Öncesinde bir kalibrasyon verisi oluşturmuştum, bu kalibrasyon verisi tabii for döngüsü ile sürekli I2C protokolü ile GYRO dan veri okumaya yönelikti.

Burada da aynı şekilde GYRO da hiç bir değişiklik yok yine aynı şekilde yapılan bir işlem.

İvme ölçer ve sıcaklık sensöründen okunan veriler de aynı şekilde.

```

218 Acc_Set[0] = 0x3B;
219 HAL_I2C_Master_Transmit(&hi2c1, MPU6050_ADDR, Acc_Set, 1, HAL_MAX_DELAY);
220 HAL_I2C_Master_Receive(&hi2c1, MPU6050_ADDR, Acc_Set, 6, HAL_MAX_DELAY);
221
222 Acc_Raw[0] = (Acc_Set[0] << 8 | Acc_Set[1]);
223 Acc_Raw[1] = (Acc_Set[2] << 8 | Acc_Set[3]);
224 Acc_Raw[2] = (Acc_Set[4] << 8 | Acc_Set[5]);

```

Çünkü ivme ölçer registerının başlangıç register i 3B ile başlıyor ve 40 adresine kadar gidiyor. Aynı şekilde burada da 6 adet 8 er bitlik eksenlere göre H ve L olmak üzere paketlerden oluşuyor.

Bu verileri yine kaydırarak ve bitsel şekilde OR işlemine tabi tutarak 16 bitlik veri elde edeceğiz.

3B adresine sahip registerin başlangıç değeri olarak kabul edildiği 6 tane registeri okumak için ben bir okuma isteği gönderiyorum STM32 den MPU6050 ye ve gönderdiğim verinin boyutu da 1 BYTE, ve diyorum ki 2. fonksiyonumda ben 6 BYTE lik veri alacağım, yani her register 1 BYTE tutuyordu zaten, bu 6 BYTE da sırası ile 0x3B den başlamak üzere teker teker 0x40 a gidiyor. Verileri tuffer listesine atıyor. Yine 0 ve 1. indexler X eksenine ait olduğu için bunları yüksek değerlikli olan 8 biti sola 8 bit ötelemeye düşük değerlikli 8 bitlik paketle bitsel olarak OR layarak 16 bitlik acc_raw ham verisini elde etmiş oluyoruz. Aynı işlemleri Y ve Z eksenleri için tekrarlıyorum.

Bu kısmın hemen ardından yine GYRO kısmına geliyoruz, GYRO kısmında yine aynı şekilde veri okuma isteği gönderdik, sonra 6 farklı registeri yine okuyoruz FOR döngüsünde yaptığımızla aynı şekilde. İşte burada o hesapladığımız değerleri görebiliriz. Kalibrasyon Offset değerlerinden bahsetmişistik hatırlarsanız, Şimdi ben başta elde ettiğim kalibrasyon değerlerini ölçüyüm değerlerden çıkartıacam.

```

234 Gyro_Raw[0] -= Gyro_Cal[0];
235 Gyro_Raw[1] -= Gyro_Cal[1];
236 Gyro_Raw[2] -= Gyro_Cal[2];

```

Görüldüğü üzere ölçtüğümüz değerler ve hata sabiti var. Bunu ölçtüğüm değerden çıkartıp tekrar bu değişkene eşitliyorum ki benim artık bu gyro_raw HAM verilerim daha düzgün bir veri olsun.

Misal gyro_raw[0] değerim 500 çıktı diyelim anlık olarak x ekseninde gyronun ölçüduğu değer. Kalibrasyon değerim zaten o 2000 ölçümden dolayı sabit artık 100 diyelim. 500-100 den 400 geliyor ve artık gyro_raw[0] değerim buna eşitleniyor. İleride kullanacağım komutlarda da bu değişken 400 olarak geçecektir.

Kısa bir açıklama; gyro dan işlenmemiş ham verileri elde ettik, şimdije kadar anlattığımız burada bir tek temperature değeri işlendi!. O da basit olduğundan direkt veriler elde edildiği gibi formül uygulandı. Tabii gyro ve ivme ölçer verilerini bir çok işleme sokacağım için hemen yapamadık. O kısımlar aşağıda.

Dökümantasyon da devam ettiğimizde bir filtre karşılıyor bizi, "Complementary Filter", yani tamamlayıcı filtre, bu bize şu şekilde bir hizmet sağlıyor;

İvmeölçer uzun vadede doğru sonuçlar gösterir fakat kısa vadede doğru sonuçlar vermez bize, Gyro ise kısa vadede doğru sonuçlar verir, kısa vadede doğru sonuçlar vermemeye başlar. "Drifting" adlı bir hatası var, biraz fazla döndürdüğünüzde, hızlı döndürdüğünüzde ve zaman geçtikçe derecelerde kayma meydana gelebiliyor, bu nedenle biz bu hatayı ivmeölçer ile tolere edebiliyoruz. Tabii bu bir filtre şeması dahilinde ivmeölçer verileri durduğun halde, durup dururken herhangi bir müdafalede bulunmuyor tabii ki.

Ne demişti? Gyro verilerini okuyup bizim bir derece değeri elde etmemiz lazım demişti, çünkü derece/sn olarak geliyor demişti.

Bu şemada ;

Gyroscope Nx oku ile gittiğimiz kutucukta integral alma ifadesini görüyoruz. Yani integralini alıyoruz aslında. İvmeölçerde ise herhangi bir şey yok, ivmeölçerde direk aldığımız veriyi (oklarda tabii işlemler var ama belirtmemize gerek yok, küçük işlemler.). İvmeölçer i "Alçak Geçiren" (LOW PASS FILTER - LPF) filtreye sokuyoruz. Gyroscope verisini de integralini aldıktan sonra "Yüksek Geçiren" (HIGH PASS FILTER) filtresine soktuktan sonra bu değerleri toplayarak bir açı değeri elde ediyoruz.

Bunu açıklamak gerekirse ;

```
#1 gyro_raw[0] /= 65.5; // işlenebilecek verileri elde etme - hassasiyet ayarlaması

#2 gyro_raw[0] *= 0.004; // örneklemme süresi ile çarparak derece/saniye birimini dereceye çeviriyoruz.

#3 angle_pitch_gyro += gyro_raw[0];

#4 gyro_raw[0] / 65.5 * 0.004 = gyro_raw[0] * 0.0000611,
= gyro_raw[0] / 65.5 / 250 = gyro_raw[0] * 0.0000611;

#5 angle_pitch_gyro += gyro_raw[0] * 0.0000611; //X ekseni

#6 angle_roll_gyro += gyro_raw[1] * 0.0000611; //Y ekseni
```

Ben ivme ölçerde +-8 g lik bir ölçüm aralığı belirlemiştim, burada bana veriler +-8 g ölçüm aralığını referans alarak gelicek, bu gelen veriler için diyor ki anlamsızdır/ham verilerdir, senin bunu 4096 ya bölmen lazım ki anlamlı olsun. (LSB Sensivity : 4096 LSB/g)

Gyroya gelecek olursam, gyroda da aynı şekilde -+500 derece/sn lik bir seçim belirlemiştim, burada da diyor ki gelen verileri 65.5 e bölmeyenin eğer, bu sonuçlar seni yanlıtır diyor, 65.5 e bölmen lazım diyor (LSB Sensivity : 65.5 LSB derece/s) değerlerini.

Şimdi görüldüğü üzere bizim aldığımız değerleri de vermemiz gerekiyormuş, hemen kodumuz da inceleyelim; Koda baktığımızda hiç 65 li bir şey göremiyoruz, neden göremiyoruz, çünkü şöyle ki, şu şekilde bir sabit veriyoruz : "0.0000611;" ve 65.5 değerimiz de aslında bu sabitin içerisinde.

Buna da değinecek olursak misal biz "gyro_raw" verisi elde ettik, bunu 65.5 e böldüğümüzde işlenebilecek veriyi elde ediyoruz yani hassasiyet ayarlaması yapıyoruz. Bu while döngüsünün her işlendiğinde süresinin 4000 us olabilmesi gerekiyor demiştiç çünkü ölçümleri aynı şekilde gyro ölçümlerini burada da yapıyoruz.

Her 4000 us de yani 4 ms de bir bu fonksiyona gelmesi lazım bu döngünün, bu yüzden bunu da hesaba katıyoruz. Ne demiştiç, bizim bir değerimiz var, GYRO değerimiz, bunun derece/sn biriminde geldiğini ifade etmiştiç. Biz bunu 65.5 e bölücez ama birimimiz değişimeyecek yine derece/sn olarak

gelicek. Biz burada aldığımız verilerin doğruluğunu ispatladık aslında seçimimizden kaynaklı olarak. Sonrasında da bizim integral almamız gerekiyor:

Integral nedir aslında, daha doğrusu belirli bir süre aralığında ölçümlerimizi yapmamız gerekiyor çünkü derece/sn olarak ölçüm yapıyoruz, bizim bu ölçüm süremizi eğer bu değer ile çarparsa yani;

$$\frac{\text{Derece}}{\text{sn}} \text{sn}$$

sn ler birbirini götürür ve sadece derece kısmı kalır yani istediğimiz birime ulaşmış oluruz. Burada da $4 \text{ ms} = 0.004 \text{ sn}$ ye denk gelmekte. Yani görüldüğü üzere ben bunu çarptığım zaman ben bir derece biriminde değer elde ederim. Yani ilk etapta 65.5 ile bölüyoruz. Sonra gelip bir daha 0.004 ile çarpiyoruz. Biz de bu işlemleri kod içerisinde sıra sıra ya da uzun uzadıya yapmaktansa tek bir sabit haline getirerek (yani $65.5 * 0.004 \text{ ten} = 0.0000611$) 0.0000611 değerini elde ettim.

Aynı şekilde $65.5 / 250$ diyerek te frekans cinsinden işlem yapmış olduk. $1/T = f$ dır, burada $T = 0.004$, yani $f = 1 * 250 / 1 = 250$. Burada saniye ile çarpiyorduk, şimdi de frekans ile çarparak ta yine aynı değeri elde ederiz. Sadece bu sefer zaman değil de frekans olarak işlem yapmış oluruz.

Bundan sonrasında ise şöyle bir durum sözkonusu, GYRO değerleri arttığı zaman hep önceki değeri referans alarak artar bu gyronun çalışma mantığı ile alakalı bir durum. Bu nedenle direkt olarak gelip bir hesaplama yaparken ham değerleri ben bir işleme sokara derece verisi elde ettim evet ama önceki değerden bağımsız olarak hesap ettiğinizde sürekli hatalı bir değer elde ettiğini görebilirsin. (burada dikkat edersen işlem +- şeklinde yani : angle_pitch_gyro += gyro_raw[0] * 0.0000611; kodunda ki gibi), bunu yapmazsa işte hatalı değer alırız. Eğer yaparsak ta bir önceki değeri de hesaba kattığı için değerlerimiz olması gerektiği gibi olur. Burada Pitch ekseni (X) ve Roll ekseni (Y) için işlemler görüldüğü üzere aynı, sadece baz aldıkları okunan değerler farklı.

Kodumuza döndüğümüzde biz burada kalibrasyon verilerini çıkardık, değerimizi oluşturduk tamam, sonra 65.5 e böldük, 0.0004 ile de çarptık, ve GYRO nun bir derecesini elde ettik. (angle_pitch_gyro += gyro_raw[0] * 0.0000611; //X ekseni ve Y ekseni için olan kısıma kadar ki kısım)

fakat Gyrolarda şu şekilde bir durum söz konusu, X ekseninin Y ekseninin kayması gibi ya da döndürüdüğümüz zaman aslında birbirini etkileyen bir durum var. Bu olayı ortadan kaldırmak için bir denklem ile hesaplamalar

yapıyoruz, yani GYRO nun değerini "sin(gyro_raw[2] * 0.000001066);" (Z ekseni işte burada devreye giriyor! Niye kullanmadık diye soracak olursan). Bizim bu Z eksenine göre Z eksenini referans olarak kabul ederek bu ikisini ayarlamamıza, daha doğru sonuç vermesini sağlayabiliyoruz.

```
241 Gyro_Pitch_Angle += Gyro_Roll_Angle * sin(Gyro_Raw[2] * Gyro_Mult * 0.004 / RAD_TO_DEG);
242 Gyro_Roll_Angle -= Gyro_Pitch_Angle * sin(Gyro_Raw[2] * Gyro_Mult * 0.004 / RAD_TO_DEG);
```

Buradan geliyoruz. Gyronun Roll kısmının yani Y ekseninin, Z ekseninin 0.000001066 ile çarpının (bu nerden geldi, biz yukarıda bir sabit belirlemiştik (0.0000611) bu sabit ile derece olarak veri elde ediyorduk. $1 / (\pi / 180)$, bu şekilde radyanı dereceye çeviren bir blok var, biz gelip bu sabitimizi bu dereceyi radyana çeviren sabite böldüğümüzde (derecemiz / 57.296) burada ki 0.000001066 değerini elde ediyoruz. Yani $0.0000611 / 57.296 = 0.000001066$.) Biz bunu Z ekseninin değeri ile çarpıp sinüsünü aldığımız zaman aslında bu eksenin diğer eksen üzerindeki etkisini bulmuş oluyoruz. Bu Roll ekseni Pitch ekseninde azaltıcı etki yaptığı için biz bu bulduğumuz değeri buna ekliyoruz. Tam tersi olarak ta Pitch ekseni Roll ekseni üzerinde artırmacı bir etki yaptığı için bu etkiyi de Pitch den çıkartarak asıl olması gereken değerlere getirmiş oluyoruz bu denklemler ile..

```
244 Acc_Total_Vector = sqrt((Acc_Raw[0] * Acc_Raw[0]) + (Acc_Raw[1] * Acc_Raw[1]) + (Acc_Raw[2] * Acc_Raw[2]));
```

Burada da evet ivmenin, tüm ivmelerin net bir ivmede ifade edilmesini denklem haline getirmiştir. İşte kök içerisinde $x^2 + y^2 + z^2$ ki hali ile tümleşik vektörü tanımlamışız.

Göründüğü üzere ivme ölçerin Y eksenini biz bu değere böldüğümüz zaman aslında biz X eksenini buluyoruz yani Pitch eksenini buluyoruz. Tabi burada ArcSin alıyoruz, Tabii bizim bu sin içerisinde kullandığımız şeyler radyan olduğu için bunları sonradan dereceye dönüştürmemiz lazım. Dereceye dönüştürme sabiti de aslında demin gördüğümüz 57.296 (RAD_TO_DEG) değeri idi aslında.

```
246 Acc_Pitch_Angle = asin((float)Acc_Raw[1]/Acc_Total_Vector)* RAD_TO_DEG;
247 Acc_Roll_Angle = asin((float)Acc_Raw[0]/Acc_Total_Vector)* - RAD_TO_DEG;
```

Buradan derece olarak değerimizi gönderdik, "angle_pitch_acc" yani ivme ölçerden elde ettiğimiz açı değeri Pitch ekseninde, diğerinde de Roll ekseninde.

```
249 Acc_Pitch_Angle -= 0.00;
250 Acc_Roll_Angle -= 0.00;
```

Burada ise görüldüğü üzere 2 tane daha denklem tanımlamışım. Şöyled ki misal telefonunu aldın mesela telefondan ivmeölçer uygulamasını açtıñ, sonra aldın ve sensörün üzerine koy, eğer telefonunuz 1 derece gösterirken sizin ivmeölçeriniz bu yazılımdan sonra 2 derece gösteriyorsa işte 1 derecelik bir sapma payı var. Bunu çıkarıp toplayarak değerinizi bulmanız lazım. Mesela bende 0.05 müş Pitch ekseninde, bunu çıkardığım zaman 0 dereye ulaşabiliyorum. Veya daha doğru bir dereceye ulaşabiliyorum diyelim. Roll ekseninde ise görüldüğü üzere "-1.32" yi çıkarmam lazım müş.

Sonrasında;

```

252 if(Set_Gyro)
253 {
254     Pitch_Angle = Gyro_Pitch_Angle * 0.9996 + Acc_Pitch_Angle * 0.0004;
255     Roll_Angle = Gyro_Roll_Angle * 0.9996 + Acc_Pitch_Angle * 0.0004;
256 }
257 else
258 {
259     Pitch_Angle = Acc_Pitch_Angle;
260     Set_Gyro = true;
261 }
262
263 while((HAL_GetTick() - PrevTime)*1000 < 4000);
264 PrevTime = HAL_GetTick();
265 }
```

Eğer GYRO set edildi ise, ki bu ifade bir BOOL değişkenidir yani 0 veya 1 dönderir. Eğer GYRO değeri set edildiyse yani 1 dönderiyorsa IF içindekileri yap yani.

Eğer GYRO değeri set edilmedi ise yani 0 dönderiyorsa eğer. Else içindeki işlemleri gerçekleştir. 0 durumundan başlayalım. Eğer 0 sa ivmeölçer çalışır çalışmaz direk değer hesaplamaya başladı Gyro ise değer hesaplamaya başlamayabilir bu nedenle ivme ölçer değerini biz çıktı olarak direk değişkenimize atıyoruz. "angle_pitch" ve "angle_roll" bizim sistemimizin çıktısıdır filtre olarak çıktısıdır. Yani "Complementary Filter" daki "Tilt Angle". "angle_pitch" ve "angle_roll" olarak biz ayarladık/yazdık. Ben burada ki "angle_pitch" değerini ilk döngüde ivmeölçerin açısına eşitliyorum, ondan sonra "set_gyro" değişkeninin değerini bool değişkeninde TRUE olarak dönderiyorum. 2. Döngüye girdiği zaman bu artık "if(set_gyro)" nun içine girecek, 3.,4.,5. döngüde de hep "set_gyro" true olduğu için bir daha "else" kısmına ilk döngü dışında yüksek ihtimalle girmeyecek.

"if(set_gyro)" içerisinde de filtremizin Yüksek Geçiren (HPF), Alçak Geçiren (LPF) ayarlarını yaptığım kısım. Görüldüğü üzere Gyrodan gelen verileri

0.9996, ivme ölçerden gelen değerin de küçük bir oranını alıyor($0.9996 + 0.0004 = 1$). Complementary Filter a da bakacak olursan zaten GYRO dan gelen verilerin integrali alındıktan sonra Yüksek Geçiren Filtreye, ivmeölçerin de Alçak Geçiren Filtre ye sokulması ile elde edilen derecelin toplanması ile biz bir OUTPUT/Çıkış verisi elde ediyoruz. Biz burada Pitch eksenini hesaplarken de zaten çıktı olarak hesaplarken Gyrodan gelen değerin büyük bir oranda çarpılmış hali ile ivme ölçerden gelen küçük bir oranda çarpılmış halini çıktı olarak veriyoruz.

Daha öncesinde bahsettiğimiz gibi kısa vadede güvenilir değil ivmeölçer, uzun vadede güvenilir. Bu güvenilirliği, bu düzeltme olayını 0.0004 katsayısı mı sağlayacaktır yani ? Evet bu kadar küçük bir oranda bir ivme ölçerin bu kadar etkisi var.

Açıklanması gereken en son kısmada gelecek olursak

HAL_GetTick() fonksiyonları başlatıldığı/fonksiyonun başladığı andan itibaren süre saymaya başlarlar (ms cisinden), bu da bizim hani 4000 us yani 4 ms de ki ölçüm arası zamanımız var demistik, eğer 4 ms yi geçerse ya da 3000, 3500 us de olursa GYRO dan doğru ölçümler elde edemeyeceğimizi belirtmiştik. Bu yüzden her bölümün 4 ms sürmesi için böyle bir kod yazıyoruz. while döngüsünde görüldüğü üzere ";" var. Bu ne demek, içerisindeki koşul sağlanmadığı sürece while döngüsünün içerisinde kal, içerisindeki koşul sağlanmadığı zaman çıkışın alttan devam etsin demektir. Biz "HAL_GetTick()" fonksiyonu ile başlatıyoruz zamanı ölçmeyi, bu fonksiyon başladığı zaman zaten devam ediyor yani ilk defa çalıştırığınız zaman devreyi STM kartındaki kodu ve sürekli saymaya devam ediyor, ondan sonra başladığını diyalim, "prevtime" yani önceki zaman var mıdır ilk döngüde, yoktur yani 0 olacaktır bu. Bu sayacak sayacak zaten geldiği andan itibaren, 1000 ile çarpıyoruz çünkü eğer ms yi 1000 ile çarparsanız mikro saniye (us) türünden elde edersiniz veriyi, diyorum ki eğer 4000 us altındaysa while döngüsünün içerisinde kalsın, 4000 us olana kadar süre beklesin, 4000 i geçtiği zaman çıkışın diyorum, ve anlık olarak o değeri "prevtime" değişkenine atayalım diyorum, çünkü bir sonraki değişkende, bir sonraki hesaplamada prevtime ı kullanıcaz. İlk çalıştığımızda prevtime yani önceki zaman olmadığı için "while" döngüsündeki halinde 0 almıştık (burda : `while((HAL_GetTick() - prevtime)*1000 < 4000);`), geliyoruz ondan sonra prevtime kısmına, bunu yine HAL_GetTick(); i buraya eşitledik dedik. Sonraki döngüye geldik, HAL_GetTick() sürekli saymaya devam ediyor zaten, prevtime da işte bir önceki while döngüsü içerisinde eşitlediğimiz değerdi, HAL_GetTick() den prevtime ı çıkarıyoruz tekrar, yine 1000 ile çarpıp us olarak elde ettiğimiz zaman değişkeni yine kontrol ediyor 4000 den küçük mü, küçükse bekle diyor 4000 olana kadar, sonra tekrar alttaki komut işliyor, böyle böyle her while döngüsü 4000 us yani 4 ms sürüyor açıkçası, eğer 4000

ms den uzun sürecek bir kod bloğunu varsa eğer, bunu biraz kendiniz ölçmeniz lazım. Ben aslında onu şu kısım ile ölçüyorum ;

```
214 PrevTime_1 = Time_1;
215 Time_1 = HAL_GetTick();
216 ElapsedTime_1 = (Time_1 -PrevTime_1) * 1000;
```

Bu fonksiyon bloğu ile ölçüyorum ne kadar zaman geçtiğini, yani burada 4000 us olsun dedim ama çalışıp çalışmadığını kontrol etmem lazım, Bu kontrolü de bu kısım ile sağlıyorum aslında, eğer aşağıdaki while kısımını kaldırırdığım zaman tüm kodun işlenmesi ortalama olarak 3000+- us sürüyor. İşte eğer 4000 den fazla ise bu muhtemelen sıkıntı verir. Bu olmadan bu kodu çalıştırıldığınız zaman ne kadar sürede gerçekleştiğini görmeniz lazım. Ben 4000 i seçtim ki etkiliyor da buranın 4000 olması. Görüldüğü üzere biz aslında 4000 us olduğuna göre derece sabitlerimizi elde etmiştik. Yani bu zamanlama bizim için çok önemli. Aynı şekilde for döngüsündeki kod bloğunun da 1000 us sürdüğünden bahsetmiştık. HAL_Delay ile 3 ms ekleyerek 3000 us ile yani toplamda 4000 us lik bir ölçüm aralığı elde etmiştik.

Bu kodumuz bitti, bizim değişkenleri görmemiz lazım.

STMStudio dan baktığımızda mesela "acc_raw[2]" değerine bakalım, biz dökümantasyondan nasıl ayarlamıştık, +-8g şeklinde ayarlamıştık, şöyle bir doğru okuyor mu diye kontrol edelim. Diyor ki dökümanda +-8 g hassasiyetle okumalığında 4096 değeri gelir diyor (yani 12 bit her gravity için). Yani ortalama olarak ta biz yer çekimine maruz kalırız, değişir bu dünyanın farklı yerlerinde.

Ayriyeten Pass Filter yerine Kalman Filtresi Çalışmalarında da bulundum fakat başarılı olamadım.

Onlarıda kısaca fonksiyon olarak paylaşayım.

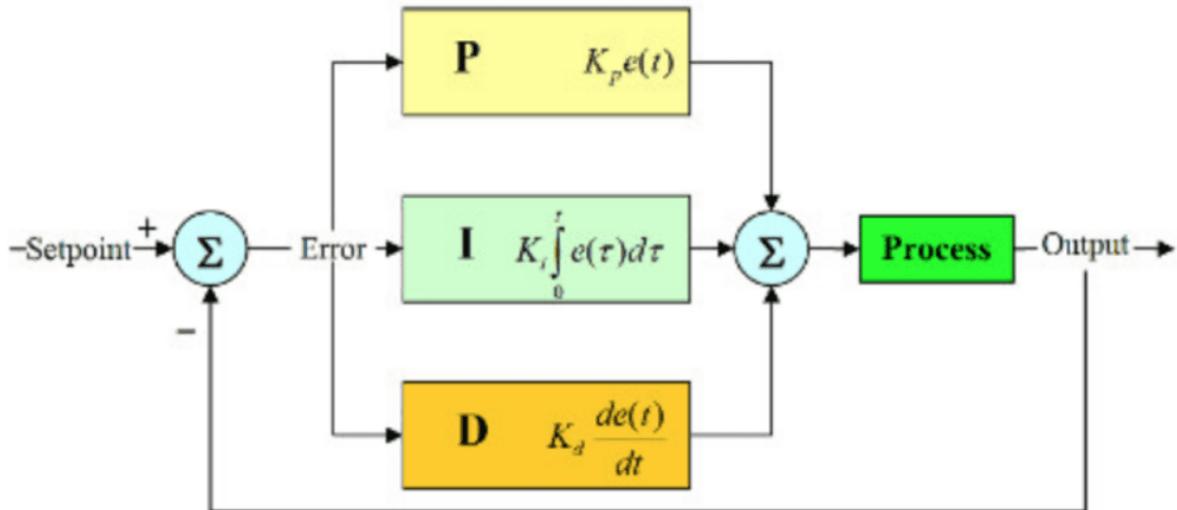
```

71 //Kalman structure
72 typedef struct {
73     double Q_angle;
74     double Q_bias;
75     double R_measure;
76     double angle;
77     double bias;
78     double P[2][2];
79 }Kalman;           96 int16_t gyro_rawK[3], acc_rawK[3];
10    .Q_angle = 0.001f,
11    .Q_bias = 0.003f,
12    .R_measure = 0.03f,
13 };
14 };
15
16 Kalman KalmanY = {
17     .Q_angle = 0.001f,
18     .Q_bias = 0.003f,
19     .R_measure = 0.03f,
20 };

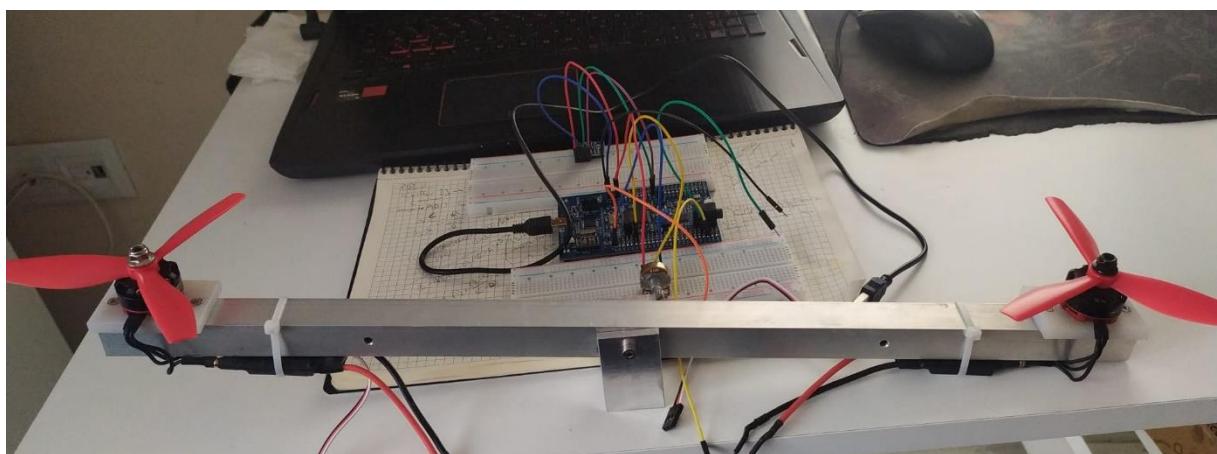
179 void MPU6050_GetAngleKalman(I2C_HandleTypeDef *hi2c, MPU6050_Struct* DataStruct)
180 {
181     uint8_t Rec_Data[14];
182     int16_t temp;
183     HAL_I2C_Mem_Read(hi2c, MPU6050_ADDR, ACCEL_XOUT_H_REG, 1, Rec_Data, 14, HAL_MAX_DELAY);
184     acc_rawK[0] = (int16_t)(Rec_Data[0] << 8 | Rec_Data[1]); //x
185     acc_rawK[1] = (int16_t)(Rec_Data[2] << 8 | Rec_Data[3]); //y
186     acc_rawK[2] = (int16_t)(Rec_Data[4] << 8 | Rec_Data[5]); //z
187
188     temp = (int16_t)(Rec_Data[6] << 8 | Rec_Data[7]);
189     gyro_rawK[0] = (int16_t)(Rec_Data[8] << 8 | Rec_Data[9]);
190     gyro_rawK[1] = (int16_t)(Rec_Data[10] << 8 | Rec_Data[11]);
191     gyro_rawK[2] = (int16_t)(Rec_Data[12] << 8 | Rec_Data[13]);
192
193     Ax = acc_rawK[0] / DataStruct->Acce_Mult;
194     Ay = acc_rawK[0] / DataStruct->Acce_Mult;
195     Az = acc_rawK[0] / Acc_Z_Corrector;
196     tempK = (float)((int16_t)temp / (float)340.0 + (float)36.53);
197     Gx = gyro_rawK[0] / DataStruct->Gyro_Mult;
198     Gy = gyro_rawK[1] / DataStruct->Gyro_Mult;
199     Gz = gyro_rawK[2] / DataStruct->Gyro_Mult;
200
201     //Kalman angle solve
202     double dt = (double)(HAL_GetTick() - timer) / 1000;
203     timer = HAL_GetTick();
204     double roll;
205     double roll_sqrt = sqrt(acc_rawK[0] * acc_rawK[0] + acc_rawK[2] * acc_rawK[2]);
206     if (roll_sqrt != 0.0)
207     {
208         roll = atan(acc_rawK[1] / roll_sqrt) * RAD_TO_DEG;
209     }
210     else
211     {
212         roll = 0.0;
213     }
214     double pitch = atan2(-acc_rawK[0], acc_rawK[2]) * RAD_TO_DEG;
215     if ((pitch < -90 && angle_kalmanY > 90) || (pitch > 90 && angle_kalmanY < -90))
216     {
217         KalmanY.angle = pitch;
218         angle_kalmanY = pitch;
219     }
220     else
221     {
222         angle_kalmanY = Kalman_getAngle(&KalmanY, pitch, Gy, dt);
223     }
224     if (fabs(angle_kalmanY) > 90)
225         Gx = -Gx;
226     angle_kalmanX = Kalman_getAngle(&KalmanX, roll, Gx, dt);
227 }
```

```
229 double Kalman_getAngle(Kalman *Kalman, double newAngle, double newRate, double dt)
230 {
231     double rate = newRate - Kalman->bias;
232     Kalman->angle += dt * rate;
233
234     Kalman->P[0][0] += dt * (dt * Kalman->P[1][1] - Kalman->P[0][1] - Kalman->P[1][0] + Kalman->Q_angle);
235     Kalman->P[0][1] -= dt * Kalman->P[1][1];
236     Kalman->P[1][0] -= dt * Kalman->P[1][1];
237     Kalman->P[1][1] += Kalman->Q_bias * dt;
238
239     double S = Kalman->P[0][0] + Kalman->R_measure;
240     double K[2];
241     K[0] = Kalman->P[0][0] / S;
242     K[1] = Kalman->P[1][0] / S;
243
244     double y = newAngle - Kalman->angle;
245     Kalman->angle += K[0] * y;
246     Kalman->bias += K[1] * y;
247
248     double P00_temp = Kalman->P[0][0];
249     double P01_temp = Kalman->P[0][1];
250
251     Kalman->P[0][0] -= K[0] * P00_temp;
252     Kalman->P[0][1] -= K[0] * P01_temp;
253     Kalman->P[1][0] -= K[1] * P00_temp;
254     Kalman->P[1][1] -= K[1] * P01_temp;
255
256     return Kalman->angle;
257 }
```

3.2.2.3 PID ile Hata Katsayılarının Programa Dahil Edilmesi



Motorun (proses) pwm pulse değeri. Bu değer TIM4 birimi üzerinden üretilerek Fırçasız motorları süren ESC lere yönlendiriliyor. Motorların hızına göre terazimin hızlı dönen motorun bağlı olduğu kısmı havalanıyor. Bu hızlanmayı bizim istediğimiz hedef açı (Target_Angle) değer ile o an okuduğu açı değerinin arasındaki farka bakarak ayarlıyor. Bu fark **Error** yani hata olarak adlandırılıyor. İstediğimiz hızı STM32 ye potansiyometremiz ile ADC üzerinden bildiriyoruz.



Şimdi ölçduğumuz hata değerine bağlı olarak motor akımını PWM değerini arttırıp azaltarak ayarlayacağız. Bu ayarlamayı yaparken 3 faktöre baktamızı söylüyor yukarıdaki diyagram.

Birincisi : Hata ne kadar büyükse o kadar büyük bir düzeltme yap. Yani PWM e hata ile orantılı bir ekleme ya da çıkartma yapacağız. Çok büyük ekleme çıkartmalar yaparsak bu sefer de hedefi aşabiliriz, küçük ekleme çıkartmalar da hedefe ulaşmamızı geciktirir. O halde K_p dediğimiz bir çarpanla düzeltme miktarını

ayarlayalım. Bu PID nin **Proportional**, yani oransal düzeltme faktörü oluyor. Şu anki durumumuzu değerlendiriyoruz.

İkincisi : Hata ne süredir devam ediyor ? Buna göre de bir düzeltme gerekebilir. Hata küçük olduğu için çok küçük düzeltmeler yapıyor, hatta hiç düzeltmeye gerek görmüyor olabiliriz. Örneğin motor sürtünmeleri ya da üzerindeki yükten dolayı sürekli olarak hedef hızın biraz altında çalışıyor olabilir. Hatta duruyor, ama sectığımız düşük çalışma hızından dolayı uyguladığımız küçük PWM değerleri - hata küçük ya- başlangıç sürtünmelerini ve ataletini aşmaya yetmiyor olabilir. O halde, başlangıçtan itibaren biriken hataların toplamına da bakmak lazım. Hata küçük ama uzun süredir devam etmekte ise düzeltme faktörünü büyütelim. İşte bu da PID nin **Integral** faktörü oluyor. zaman içinde biriken hatayı da bir katsayı ile düzeltme faktörüne eklemek gerekiyor. Bu faktör ile geçmişteki durumu değerlendirmeye katıyoruz.

Üçüncüsü : uygulamakta olduğumuz düzeltmelere motorumuz ne kadar hızlı tepki gösteriyor. Örneğin ; motorun üzerindeki yük hızlanması yavaşılatıyor olabilir, o zaman hedef hızı çok uzun sürede ulaşacaktır. Bu durumda motorun hızının artış hızına – yani ivmeye bakarak da bir düzeltme faktörü eklemek gerektir. Bu faktör de PID nin “D” si, yani türev bileşenini oluşturuyor. Bu faktör ile de geleceğe bakıyoruz.

Bir başka deyişle, düzeltme faktörümüzün içinde şu an (**P: Proportional – oransal**), geçmiş zaman (**I: integral**) ve gelecek zaman (**D : derivatives – türevsel**) üç bileşen var.

```

102 double Proportional_Factor=30,Integral_Factor=1,Derivative_Factor=5;
103 double Target_Angle = 0;
104 float Proportional=0, Integral=0, Derivative=0;
105 float PID;
106 float Angle_Error;
107 float Previous_Error = 0;
108 unsigned long Time_Instant, Time_Last, ElapsedTime_PID;

266@ void ComputePID()
267 {
268     Time_Last = Time_Instant;
269     Time_Instant = HAL_GetTick();
270     ElapsedTime_PID = (Time_Instant - Time_Last) / 1000;
271     Angle_Error = Roll_Angle - Target_Angle;
272     Proportional = Proportional_Factor * Angle_Error;
273     if( -10 < Angle_Error && Angle_Error < 10)
274     {
275         Integral = Integral + (Integral_Factor * Angle_Error);
276     }
277     Derivative = Derivative_Factor * ((Angle_Error - Previous_Error) / ElapsedTime_PID);
278     PID = Proportional + Integral; //+ Derivative;
279     if(PID < -2500)
280     {
281         PID = -2500;
282     }
283     if(PID > 2500)
284     {
285         PID = 2500;
286     }
287     Previous_Error = Angle_Error;
288 }
```

İlk olarak Proportional Faktörü ele alalım. Basitçe istediğimiz hedef açı değeri ile o anda olduğumuz açı değeri arasındaki farkı bir Proportional_Faktor katsayısı ile çarparak Proportional değerini elde ediyor.

Bu Öncesinde bahsettiğimiz gibi hata ne kadar büyükse o kadar düzeltme uygulanacağı anlamına geliyor.

```
272     Proportional = Proportional_Factor * Angle_Error;
```

Integral faktörüne gelecek olursak, integral faktörü çok daha küçük düzeltmeler için kullanılıyor. Bu sebeple hata payı belli bir düzeyin altına indiğinde bu faktörü uygulamaya koymuyor.

```
273     if( -10 < Angle_Error && Angle_Error < 10)
274     {
275         Integral = Integral + (Integral_Factor * Angle_Error);
276     }
```

Yani bu kısıma göre Integral faktörü istenen hedef açı ile o anki açı değeri arasındaki fark sadece 10 derecenin altına düşüğünde uygulamaya konulacak.

Türev Faktörüne gelecek olursak, bu faktör geçen zamana göre oluşan hata ve uygulanan pwm değeri parametrelerine bağlı.

Örneklemek gerekirse Oransal hata düzeltme çok fazla olursa bu faktör geçen süreye göre uygulanan orantısız düzeltmeyi görecektir ve ona göre bir değer uygulayacaktır.

```
268     Time_Last = Time_Instant;
269     Time_Instant = HAL_GetTick();
270     ElapsedTime_PID = (Time_Instant - Time_Last) / 1000;

277     Derivative = Derivative_Factor * ((Angle_Error - Previous_Error) / ElapsedTime_PID);

287     Previous_Error = Angle_Error;
```

Son olarak toplam uygulanması gereken PID değeri de bu 3 faktörün toplamına eşittir.

```
278     PID = Proportional + Integral; //+ Derivative;
```

Burada bizim yapacağımız, hesaplama sonucu elde edilen PID değerini, istenen açı değerine ulaşmak için Motorlara uygulanan PWM değerinden eklemek ya da çıkartmak.

Bu sebeple istenen değerden fazla düzeltmeler yaşanmaması için / yeterli olduğu için PID değerini -2500 ile 2500 arasında kısıtladım.

Motorları terazimiz üzerinde tam güçte kullanmayacağız fakat bunu simüle etmek istiyorsak misal motorlara uygulayacağımız maximum PWM değeri 10000 olsun, bu durumda PID dışındaki faktörlerin 7500 den fazla PWM değeri ile sonuçlanması bizim periyodumuzu aşar, sonuç olarak 7500 PWM periyod değerinden yüksek değerlerde ki bir değere PID de eklediğimizde cihazımızın havada denge kurabilmesi öncesi kadar kolay olmayacağındır.

3.2.2.4 ADC Okuma ve PWM Pulse Değeri Ayarlama

ADC1, ADC2, ADC3 birimlerinden sırası ile pulse değeri, yön ve dönme fonksiyonları için okumalar yapacağımızı söylemiştık. Ve okumaları iki şekilde yapabileceğimizden bahsetmiştik.

```
110 uint16_t ADC_Value[3];
111 uint16_t Pulse_Value[3];
```

İlki farklı ADC birimleri üzerinden sırası ile PoolForConversion yöntemi ile okuma

```
289 void Read_ADC(ADC_HandleTypeDef *hadc)
290 {
291     HAL_ADC_Start(&hadc1);
292     if(__HAL_ADC_GET_FLAG(&hadc1, ADC_FLAG_EOC) != RESET)
293     {
294         ADC_Value[0] = HAL_ADC_GetValue(&hadc1);
295     }
296     HAL_ADC_Stop(&hadc1);
297 }
```

Burada ADC1 biriminden okuduğumuz değeri ADC_Value setinin 0. indexine eşitliyoruz.

ilk önce ADC1 birimini başlatıyoruz.

Sonrasında EOC bayrak durumu okunabilir gösteriyorsa okunan değeri dizimizdeki indexe eşitliyoruz. İşlem bittikten sonra ADC1 birimini sonlandırıyoruz.

Aynı işlemleri sırası ile ADC2 ve ADC3 için de uyguluyoruz.

İkinci yöntem ise kesmeler ile tek bir ADC birimi üzerinden okuma idi.

```
376 HAL_ADC_Start_IT(&hadc1);
377 HAL_ADC_Start_IT(&hadc2);
378 HAL_ADC_Start_IT(&hadc3);

298 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
299 {
300     if(__HAL_ADC_GET_FLAG(&hadc1, ADC_FLAG_EOC) != RESET)
301     {
302         ADC_Value[0] = HAL_ADC_GetValue(&hadc1);
303     }
304     if(__HAL_ADC_GET_FLAG(&hadc2, ADC_FLAG_EOC) != RESET)
305     {
306         ADC_Value[1] = HAL_ADC_GetValue(&hadc2);
307     }
308     if(__HAL_ADC_GET_FLAG(&hadc3, ADC_FLAG_EOC) != RESET)
309     {
310         ADC_Value[2] = HAL_ADC_GetValue(&hadc3);
311     }
312 }*/
```

Önce ADC1 i START ediyorum, yine adreslerini verdim (&*), şu anda çevrim işlemi başladı ve "Continuous Conv" seçtiğim için "while(1)" içerisinde yazmama bile gerek yok, 1 kez yazdığında sürekli çevrime giricek.

"ADC_IRQHandler()" fonksiyonunu kullanmamızı istiyor ki zaten "stm32f4xx_it.c" dosyasında otomatik olarak tanımlanmış.

Interrupt olduğu zaman "HAL_ADC_ConvCpltCallback()" fonksiyonunda dallanır. O zaman ben bu fonksiyonun konumuna gidecek olursam yani burada denildiği üzere işlemleri bu fonksiyonun içerisinde yaparsam olur:

O zaman ben bu fonksiyonu alıyorum ve "main.c" dosyam içerisinde yapıştıryorum; Ve bu arkadaşı artık kullanabilirim ben, yani bu fonksiyonun içerisinde yazacağım şeyler aslında ben Interrupt a girdiğimde olacak şeyler. Yani zaten "IRQhandler" ı kullandık otomatik olarak. Kesme olduğunda "IRQhandler" a gidicek, o da buraya gelicek. "Callback" fonksiyonunda şimdi istediğim işlemleri yapabilirim. Burada 2 tane farklı kanal var, o yüzden bu fonksiyona benim hangisi için Interrupt a girdiğini yani ADC1 e girdiğini söylemem lazım.

O yüzden;

```
if(__HAL_ADC_GET_FLAG())
```

Flag bilgisini bir alalım, konumuna gidersem hangi parametreleri istediğimi ve içerisinde neler yazmam gerektiğini rahatça görebilirim. bu fonksiyon bizden "HANDLE" ve "FLAG" istiyor. "HANDLE" zaten bizim ADC Handler ımız. FLAG olarak da End of Conv (EOC) Flag i .Bu şekilde ne olacak, eğer bu arkadaş RESET değilse, yani RESET olmuşsa o zaman demektir ki "ADC1" in Interrupt ı aktif olmuş.

Özetle diyorum ki; Eğer ADC1 için interrupt a girdiysen "&hadc1" ifadesi olanı okuyacağız

"adc1_value" değeri vardı hatırlarsan, ve bu 3 kanal barındırdığı için buna "count" değeri ile hangi kanalı okuyacağını seçiyoruz. Yani "count" kaç ise o kanalı okuyacağım. Neyle okuyacağım, yine aynı;

```
adc1_value[count] = HAL_ADC_GetValue(&hadc1);
```

okuduğum ADC değerlerine göre PWM pulse değeri ayarlamalarına gelecek olursak;

Burada 12 bitlik okuduğumuz ADC değerlerini 16 bitlik ADC_Value dizisinde saklıyorduk. 12 bitlik okuma yaptığımız için okuduğumuz değer de 0-4095 arasında. O zaman benim bu aralıktaki okuduğum değeri istediğim PWM pulse değeri aralığına ayarlamam lazım.

Bunun için bir map fonksiyonu yazıyorum;

```
313 uint16_t map(int In, int Inmin, int Inmax, int Outmin, int Outmax)
314 {
315     return (In - Inmin) * (Outmax - Outmin) / (Inmax - Inmin) + Outmin;
316 }
```

Bu şekilde

```
388     MPU6050_ComputeAngle(&hi2c1);
389     ComputePID();
390     //Read_ADC();
391     Pulse_Value[0] = map(ADC_Value[0], 0, 4095, 0, 5000);
392     Pulse_Value[1] = map(ADC_Value[1], 0, 4095, 1000, 2000);
393     Pulse_Value[2] = map(ADC_Value[2], 0, 4095, 1000, 2000);
```

Pulse_Value dizisindeki indexleri ADC_Value dizisindeki indexleri kendi istediğim aralığa getirerek set etmiş oldum.

Pulse_Value[0] motorlarımıza gidecek ana PWM değerini belirtiyor

Pulse_Value[1] bizim yönlere doğru ilerleyebilmemiz için iki motorun PWM değeri arasındaki farklı belirtecek

Pulse_Value[2] ise kendi eksenimiz etrafında dönmemizi sağlayacak.

3.2.2.5 Harici Interrupt ile Modlar Arası Geçiş

Modlar arası geçiş için PA0 a bağlı USER butonu kullanacağımızı söylemiştim

Bu pini external interrupt olarak belirlemiştik, doğal olarak fonksiyonu .it dosyasında.

Biz bu fonksiyonu direkt olarak main.c dosyamıza alıp kullanabiliriz.

```

115 int Exti = 0;
116 char Mode;

317 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
318 {
319     Exti++;
320     if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0))
321     {
322         if(Exti == 1)
323         {
324             Mode = 'Drone';
325         }
326         else if(i == 2)
327         {
328             Mode = 'Yatay';
329         }
330         else if(i == 3)
331         {
332             Mode = 'Sabit Kanat';
333             Exti = 0;
334         }
335     }
336 }
```

İki farklı şekilde modlar arası geçiş uygulaması yaptım. İki farklı uygulama yapmamın sebebi elimdeki minib usb dönüştürücünün bozulması sebebi ile yazdığım kodları derleyemiyor olmam.

Öncelikle her kesme geldiğinde birer birer artacak bir Exti değişkeni tanımladım. Bu değişkenin değeri bizim hangi modda çalıştığımızı gösterecek.

Fonksiyon 1

```

401     switch (Mode) {
402         case 'Drone':
403             Target_Angle = 0;
404             PWM_M1 = Pulse_Value[0] + Pulse_Value[1] + Pulse_Value[2] + PID;
405             //PWM_M2 = Pulse_Value[0] - Pulse_Value[1] + Pulse_Value[1] + PID;
406             PWM_M3 = Pulse_Value[0] + Pulse_Value[1] - Pulse_Value[1] - PID;
407             //PWM_M4 = Pulse_Value[0] - Pulse_Value[1] - Pulse_Value[1] - PID;
408             break;
409         case 'Yatay':
410             Target_Angle = 90;
411             if( -10 < Angle_Error && Angle_Error < 10){
412                 Exti++;
413             }
414             break;
415         case 'Sabit Kanat':
416             PWM_M1 = Pulse_Value[0] + Pulse_Value[1] + Pulse_Value[2];
417             //PWM_M2 = Pulse_Value[0] - Pulse_Value[1] + Pulse_Value[1];
418             PWM_M3 = Pulse_Value[0] + Pulse_Value[1] - Pulse_Value[1];
419             //PWM_M4 = Pulse_Value[0] - Pulse_Value[1] - Pulse_Value[1];
420             break;
421         default:
422             break;
423     }

```

Fonksiyon 2

```

425     if(Exti = 1)
426     {
427         PWM_M1 = Pulse_Value[0] + Pulse_Value[1] + Pulse_Value[2] + PID;
428         //PWM_M2 = Pulse_Value[0] - Pulse_Value[1] + Pulse_Value[1] + PID;
429         PWM_M3 = Pulse_Value[0] + Pulse_Value[1] - Pulse_Value[1] - PID;
430         //PWM_M4 = Pulse_Value[0] - Pulse_Value[1] - Pulse_Value[1] - PID;
431     }
432     else if(Exti = 2)
433     {
434         Target_Angle = 90;
435         if( -10 < Angle_Error && Angle_Error < 10)
436             Exti++;
437     }
438     else if(Exti = 3)
439     {
440         PWM_M1 = Pulse_Value[0] + Pulse_Value[1] + Pulse_Value[2];
441         //PWM_M2 = Pulse_Value[0] - Pulse_Value[1] + Pulse_Value[1];
442         PWM_M3 = Pulse_Value[0] + Pulse_Value[1] - Pulse_Value[1];
443         //PWM_M4 = Pulse_Value[0] - Pulse_Value[1] - Pulse_Value[1];
444     }
445     else
446     {
447         Exti = 1;
448     }

```

Terazimizde yalnızca karşılıklı 2 motor kullandığımızdan şu anda sadece motor 1 ve bir drone a göre tam karşısında olması gereken motor 3 ü kullanıyorum.

Exti = 0 ken (ya da 1 ken, interrupt ile ilgili çözemediğim bir problem), Drone modunda çalışıyoruz, bu modda PID hesapları motorlara gönderilen PWM sinyali değeri hesabına katılıyor. Dengede tutmaya çalıştığımız açı değeri ise 0.

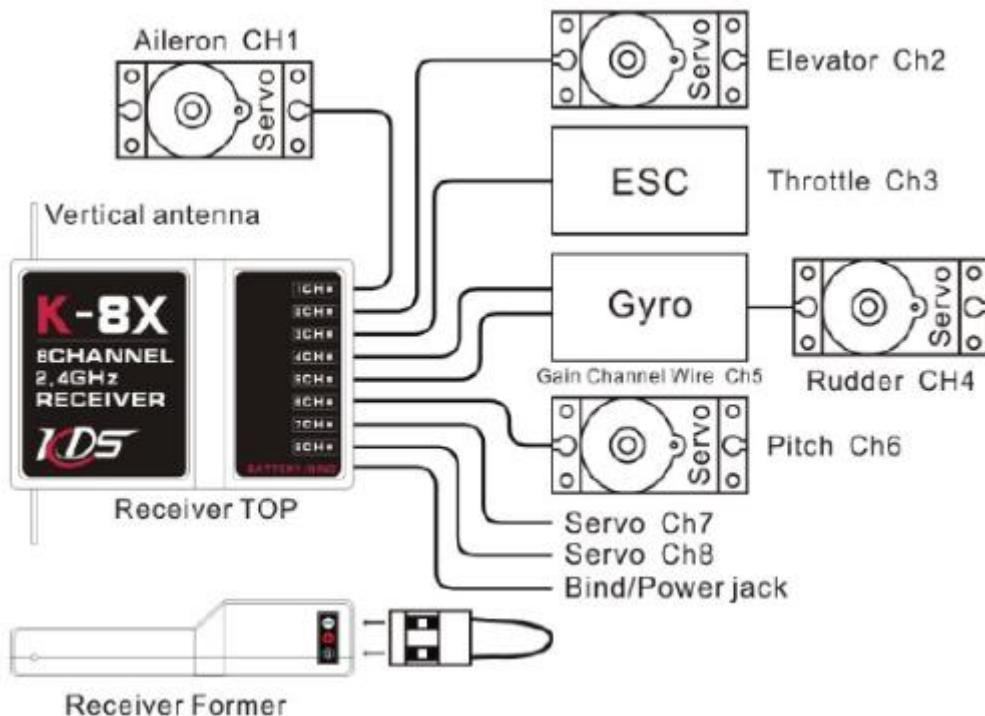
Exti = 2 iken Yatay moda geçiyoruz. Bu modda PID hesaplamaları yine işin içine katılıyor fakat bu sefer hedeflediğimiz açı 90 derece. Tam 90 dereceyi tutturması zor olacağından belli bir hata payının altına düştüğünde direkt olarak Exti değerini arttırıp diğer moda geçmesini söyledim.

Exti = 3 iken artık sabit kanat modundayız ve tüm motorlar herhangi bir dönme değeri gelmediği sürece aynı PWM sinyali ile çalışıyor.

Eğer Exti değeri 3 ü geçerse tekrar Drone moduna dönüyoruz yani Exti sıfırlanıyor.

3.2.2.6 RC Alıcı ile STM Kontrol Denemesi

Elimizde KDS 7XII RC kumanda ve alıcısı bulunmaktadır. Son etap olan bu kumanda ile STM32F407VG kartımız üzerindeki motorları nasıl süreceğimize bir bakalım



Burada ESC nin alının CH3 kanalına bağlı olduğunu göstermiş (Sabit kanat modu) Yani motorların hızı bu kanaldan gelen PWM sinyalinin değerine göre ayarlanıyor.

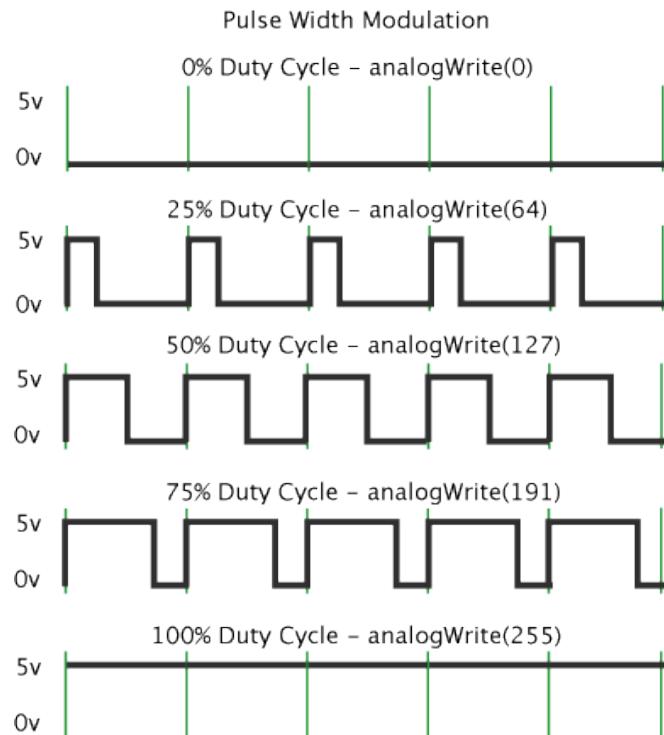
Bu durumda PWM sinyalini TIMER birimimizin Input Capture özelliği ile okumaya ve Potansiyometre ile ADC okuyarak ayarladığımız Pulse değerimizi artık alıcı ile ayarlamaya çalışacağız.

İşte öncelikle PWM sinyalini anlamakla başlayalım.

Pulse width Modulation (Darbe genişlik modülasyonu); bir kare dalga sinyalin, yüksek seviyede kalma süresine müdahale ederek, bu sinyalin geriliminin ortalama değerinin değiştirilmesi olarak tanımlanabilir.

PWM endüstride iletişim, motor kontrol, ısıtma, aydınlatma gibi önemli bir çok alanda kullanılmaktadır.

Aşağıdaki şekilde sinyalin görev yapan kısmın (duty cycle) süresinin değiştirilmesi, yani PWM olayını göstermektedir.



Bu kısımda hesaplama şöyle oluyor;

- Duty Cycle'in %25 olduğu noktada sinyalin ortalama değeri 5V'un $1/4$ 'ü olacaktır yani; $5V/4 = 1.25$ V
- Görev çevrimi %50 olan adımda sinyalin $1/2$ 'si alınacak ve değeri $5V/2 = 2.5$ V olur.

Burada önemli olan nokta sinyalin periyodu hiç bir zaman değişmez. Yani frekans değeri sabit kalır. PWM'in endüstriyel kullanımına genellikle frekans sabit kalır ve sinyalin görev çevrimi değeri değiştirilir.

STM32 serisinde PWM Timer'ının özelliğidir. STM32F4 Discovery kartında 14 adet PWM verilen Timer Birimi bulunur.

3.2.2.6.1 PWM Modları

Mod 1: Yukarı doğru sayarken $CNT < CCRx$ (Capture Compare Register) dan düşükse kanal aktif, diğer durumda pasif olur. Aşağı doğru sayarken $CNT > CCRx$ ise kanal pasif, değilse aktif olur.

Mod 2: Yukarı doğru sayarken $CNT < CCRx$ (Capture Compare Register) dan düşükse kanal pasif, diğer durumda aktif olur. Aşağı doğru sayarken $CNT > CCRx$ ise kanal aktif, değilse pasif olur.

Yani misal Mod 1 imiz %25 Duty Cycle ile çalışıyor olsun. Bu durumda High demek Mod 1 de yani %25 i HIGH, %75 i LOW demek, Mod 2 de ise tersi, bu örnek için Mod 2 olsaydı bu durumda %75 HIGH, %25 LOW verecekti.

3.2.2.6.2 Pulse Ölçümü

Sinyalin lojik -1 olduğu süreyi ölçmek için mikrodenetleyici içerisinde bulunan Timer(Zamanlayıcı/Sayıcısı) birimlerini kullanırız.

Olay örgüsünü özet bir şekilde anlatacak olursak;

- # Zamanlayıcımız/Sayıcımız çalışmaya başladı
- # Sinyalin Yükselen kenarında bir kesme oluştu ve o anda sayıcının sahip olduğu değer kaydedildi
- # Sinyalin düşen kenarında bir kesme oluştu ve o anda sayıcının sahip olduğu değer kaydedildi
- # Düşen kenarda kaydedilen değerden, yükselen kenarda kaydedilen değeri çıkarttığımızda genişlik değerimizi elde ederiz.

Zamanlayıcı biriminin STM32 tabanlı kartlarda, Input Capture(Sinyal Yakalama) modu vardır. Belirli ayarlar ve HAL kütüphanesinin basit kullanımı ile sinyallerimizin genişliklerini ölçüceğiz.

Burada geliştirilen uygulama sizler için iskelet yapısı niteliğindedir. Fikir verme amacı gütmektedir. Kendi uygulamanıza göre kodları uyarlayabilirsiniz.

Bu uygulamada TIM 1 birimi kullanılacaktır.

Ben sistemi genel olarak 168 MHz üzerinden çalıştırıldım ; (STM32L476RGTx LQFP64)

* Görüldüğü üzere TIM1 biriminin saat frekansı olarak beslendiği hat PCLK2 Hattıdır.

* Genel saat frekansı 84 MHz olarak tercih ettim.

* TIM1 birimini besleyen hatta bu 84 MHz değerine sahiptir.

Fakat ben zamanlayıcı biriminin 1 MHz de çalışmasını istiyorum. 1 MHz de çalışan bir zamanlayıcı, her periyodu/çevrimi/sayımını 1 us lik bir değer üzerinde

gerçekleştirir. Bu da demek oluyor ki ben 1 saydığım zaman 1000 us yi 2000 us yi farklı değerlerle, farklı frekans değerleri ile çarpmadan bölmeden doğrudan elde ettiğim değerle uygulamada kullanabilirim.

Benim TIM1 i kullanıyorum, Timers dan TIM1 4 kanalı "Input Capture Direct Mode" olarak ayarladım. Ve "parameter settings" de Timer imizin genel ayarları var. Biz Timer imizi 1 MHz hızda çalıştırmak istediğimizi söylemiştim. Bunun için bu TIMER 1 besleyen yolun frekansını bölmemiz lazım.

Timer 1 besleyen yolun frekansı 84 MHz di, Ben 1 MHz i elde etmek istiyorsam tabii ki de 84 ye bölmem gerekiyor.

Burada baktığımız zaman 84-1 yazmışız, bunun nedeni bazı Registerlar değerin 1 fazlasını alıyor, yani ben buraya 84 yazdığında sistem bunu 85 olarak değerlendirecekti.

Ve burada TIM1 imizin çalışma frekansını 1 MHz olarak ayarlamış olduk.

Counter Mod : Up yani yukarı sayıcı bir sistem.

diğer ayarlar zaten default geliyor

farklı bir ayar yapacaksak eğer Reference Manual lerden DataSheetlerden vs bakman lazım. Bu dökümanları açıp üzerinde çalışacağımız timerları incelememiz gerek, çünkü belki de üzerinde çalışmak istediğimiz TIM biriminde yakalama modu veya başka bir şey olmayabilir.

Sonrasında sayıcının sayma değeri var (Counter Period), yani burada 16 bitlik bir sayıcıdan bahsediyoruz yani 65.536 değerine kadar yani 1 us de her sayımda da 65.536 us ye kadar sayıyor, ondan sonra bir update ile tekrar sıfırlanıyor ve tekrar saymaya başlıyor.

Polarity Selection : Rising edge yani yükselen kenar tetiklemesine sahip 4 kanalda.

NVIC settings e kesmeleri aktifleştirdiğimiz ayarların olduğu kısım. Burada "TIM1 capture compare interrupt" yani her kanaldan bir kesme geldiğinde programımız kesme fonksiyonuna dallancak, bunu aktifleştirmemiz/tiklememiz lazım.

SYS kısmından da "serial wire" seçeneğini bilgisayar ile bağlantı kurmak için seçtik. Bundan sonrasında kodumuzu generate edebiliriz (Alt+K)

Misal : Burada "void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)" isimli bir fonksiyonumuz var. Buna

Drivers>STM32F4xx_HAL_Driver>Inc & Src dosyası içerisinde ilgili konumuzun .h ve .c dosyalarından fonksiyonların kendisine ve tanımlamalarına ulaşabiliriz. Direkt programdan da CTRL+RC ile fonksiyonun kendisine de bakabiliriz.

```
49 /* USER CODE BEGIN PV */
50 volatile uint32_t ch1_rising = 0, ch2_rising = 0, ch3_rising = 0, ch4_rising = 0;
51 volatile uint32_t ch1_falling = 0, ch2_falling = 0, ch3_falling = 0, ch4_falling = 0;
52 volatile uint32_t pre_ch1 = 0, ch1 = 0, pre_ch2 = 0, ch2 = 0, pre_ch3 = 0, ch3 = 0, pre_ch4 = 0, ch4 = 0;
```

```
66 void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
67 {
68     if(htim->Instance == TIM1)
69     {
70         switch(htim->Channel)
71         {
72             case HAL_TIM_ACTIVE_CHANNEL_1:
73                 if((TIM1->CCER & TIM_CCER_CC1P)==0)
74                 {
75                     ch1_rising = TIM1->CCR1;
76                     TIM1->CCER |= TIM_CCER_CC1P;
77                 }
78             else
79             {
80                 ch1_falling = TIM1->CCR1;
81                 pre_ch1 = ch1_falling - ch1_rising;
82                 if(pre_ch1 < 0)pre_ch1 += 0xFFFF;
83                 if(pre_ch1 < 2010 && pre_ch1 > 990)ch1=pre_ch1;
84                 TIM1->CCER &= ~TIM_CCER_CC1P;
85             }
86             break;
87         case HAL_TIM_ACTIVE_CHANNEL_2:
88             if((TIM1->CCER & TIM_CCER_CC2P)==0)
89             {
90                 ch2_rising = TIM1->CCR2;
91                 TIM1->CCER |= TIM_CCER_CC2P;
92             }
93             else
94             {
95                 ch2_falling = TIM1->CCR2;
96                 pre_ch2 = ch2_falling - ch2_rising;
97                 if(pre_ch2 < 0)pre_ch2 += 0xFFFF;
98                 if(pre_ch2 < 2010 && pre_ch2 > 990)ch2=pre_ch2;
99                 TIM1->CCER &= ~TIM_CCER_CC2P;
100            }
101        break;
102    }
103 }
```

```

102     case HAL_TIM_ACTIVE_CHANNEL_3:
103         if((TIM1->CCER & TIM_CCER_CC3P)==0)
104         {
105             ch3_rising = TIM1->CCR3;
106             TIM1->CCER |= TIM_CCER_CC3P;
107         }
108     else
109     {
110         ch3_falling = TIM1->CCR3;
111         pre_ch3 = ch3_falling - ch3_rising;
112         if(pre_ch3 < 0)pre_ch3 += 0xFFFF;
113         if(pre_ch3 < 2010 && pre_ch3 > 990)ch3=pre_ch3;
114         TIM1->CCER &= ~TIM_CCER_CC3P;
115     }
116     break;
117     case HAL_TIM_ACTIVE_CHANNEL_4:
118         if((TIM1->CCER & TIM_CCER_CC4P)==0)
119         {
120             ch4_rising = TIM1->CCR4;
121             TIM1->CCER |= TIM_CCER_CC4P;
122         }
123     else
124     {
125         ch4_falling = TIM1->CCR4;
126         pre_ch4 = ch4_falling - ch4_rising;
127         if(pre_ch4 < 0)pre_ch4 += 0xFFFF;
128         if(pre_ch4 < 2010 && pre_ch4 > 990)ch4=pre_ch4;
129         TIM1->CCER &= ~TIM_CCER_CC4P;
130     }
131     break;
132     default:
133         break;
134     }
135 }
136 }
```

if(htim->Instance == TIM1)

Kesme TIM1 den mi geliyor

switch(htim->Channel)

Hangi kanaldan okuma geliyorsa Switch ile Channelx e git

```

if((TIM1->CCER & TIM_CCER_CC1P)==0)
{
    ch1_rising = TIM1->CCR1;
    TIM1->CCER |= TIM_CCER_CC1P;
}
else
{
    ch1_falling = TIM1->CCR1;
    pre_ch1 = ch1_falling - ch1_rising;
    if(pre_ch1 < 0)pre_ch1 += 0xFFFF;
    if(pre_ch1 < 2010 && pre_ch1 > 990)ch1=pre_ch1;
    TIM1->CCER &= ~TIM_CCER_CC1P;
}
break;

```

Gelen sinyalin yükselen veya düşen kenar olmasına göre sistemimiz tetikleniyor. Eğer yükselen kenar kesmesi ise bu register da 2 tane önemli bitimiz var CC1NP ve CC1P, eğer yükselen kenar seçiceksen / buradan kesme gelicekse bu iki biti de 0 laman lazım, eğer düşen kenarda bir kesme bekliyorsan CC1NP = 0, CC1P = 1 yazman lazım. Burada CC1NP her iki durumda da 0 bu nedenle CC1P yi kontrol ediyoruz.

Yani bu register ile bu pinini/bitini AND ledigim zaman 0 geliyorsa kod bloğuna gir, 0 gelirse bu yükselen kenar kesmesi ile gelmiş bu kesme demektir.

ch1_rising = TIM1->CCR1; // yükselen kenar degerini kaydet

TIM1->CCER |= TIM_CCER_CC1P; // polariteyi düşen kenar olarak degistir

Şimdi biz zaten yükselen kenarı elde ettik, şimdi PULSE ölçebilmek için düşen kenarın değerini de elde etmemiz lazım yani düşen kenarda bir kesme elde etmemiz ve o kesme geldiğinde ki değerini almamız lazım bu yüzden bir sonraki döngü için polariteyi değiştiriyoruz. Yani düşen kenar tetiklemesine odaklı bir ayar yapmamız lazım bunun için de buna göre |= ile buna göre bir ayar yapıyoruz. Biz bu biti bu kod yardımcı ile =1 yapmış oluyoruz, CC1P = 1 olduğu zaman CC1NP ile 0 1 olacağından artık bu register düşen kenardaki değeri barındıracaktır.

Buradan sonra tabii ELSE bloğuna da girmeyecek çünkü zaten IF bloğunu işlemiş olacak, "break" ile çıkışacak, bundan sonrasında 2.kesmeyi beklicez.

İkinci kesme geldi diyelim, ikinci kesmede düşen kenarla bir kesme geldi, IF şartı artık ==1 olacağı için IF bloğuna zaten girmeyecektir. Direkt ELSE bloğunu işlemeye başlayacaktır. Burada da bahsettiğimiz gibi o anki timer değerini bu register a attığı için donanım bu da düşen kenarın değeri olacak CH1 için. Biz bu iki değer arasındaki farklı elde ettiğimizde PULSE u yani Darbe Modülasyonunun değerini elde edicez. Burada degenilmesi gereken bir

kaç önemli detay var. Bunlardan biri "pre_chx" yani bir önceki değer olarak işlemlerin isimlendirilmesi

Eğer yükselen kenar değilse, düşen kenar ile oluşan bir kesme ise

```
pre_ch1 = ch1_falling - ch1_rising; // dusen kenar degerini kaydet ve  
yukselen kenar degerinden cikar
```

```
if(pre_ch1 < 0)pre_ch1 += 0xFFFF;// eger sonuc negatifse taban tumleme yap
```

Eğer sonuç negatifse gitsin 65.536 değeri ile toplayıp eşitlensin demişim. Niye böyle bir şey yapmışız ; Şimdi bizim Timerımız 65.535 e kadar sayıyor demişti. Timer tam 65.000 değerindeyken sinyalin yükselen kenarı geldi, kaydedildi diyelim. Sonra düşen kenarda bir tetikleme geldi, değeri de 570, çünkü 65.535 değerinden sonra timer kendini sıfırlayıp tekrar saymaya başlıyordu. Şimdi Falling değerinden Rising değerini çıkarıyoruz, e bu sefer bir sayı geldi ve biz bunu istemiyoruz.Bu durumda taban tımlıme yapacağız. Biz burada 65.535 e göre işlem yaptığımız için biz burada bu sayıyı 65.535 ile topladığımızda pozitif doğru değeri/asıl elde etmek istediğimiz değeri elde etmiş olacağız.

2. kısımda ise değerler belirli periyodlar ile max değerleri alıyor ve saçma sapan değerler almaya başlayabiliyor. Bu nedenle bu şekilde bir kontrol yapısı oluşturduk. 2010 un altında ise ve 990 in yukarısında ise, zaten 1000 ile 2000 us arasında olacak demistik değerlerimiz için. Bu değerler arasında ise pre_ch1 değeri ch1 e aktarılın. Eğer bunların dışında ise bir önceki değer bu döngüde tekrardan işlenicek. Böylece düşen kenar tetiklemesi ile yapılacak işlemleri yaptım, bundan sonrasında bizim tekrar yükselen kenar ile gelecek olan tetiklemeyi beklememiz lazım, bu yüzden yine registerımızın ayarını değiştirerek CC1NP ile CC1P = 0 0 kombinasyonunu sağlayarak polariteyi değiştiriyoruz.

Aynı şekilde bu kod bloğu işlendikten sonra sırada "break" geliyor ve döngüden çıkıyor, ve 3. kesmeyi bekliyor. Bu şekilde teker teker bu fonksiyonlara dallanacak, oluşturduğumuz kontrol mekanizmalarının içine girecek. Burada yaptığımız işlemler sadece CH1 içindi, CH2 den gelen kesmeler için de aynı şeyler gerçekleşecek yani bir farklılık yok aslında, aynı kodların sadece CH2, CH3 vs için revize edilmiş hali.

TIMERımızın kesme geldiğinde dallanacağı, anafonksiyonumuzun altfonksiyonu bir kesme fonksiyonuydu. Bunu başlatmamız lazım, yani bizim bu kanallardan gelebilecek kesmelerin CPU ya iletilmesini sağlayabileceğimiz bir fonksiyonu çalıştırmanız lazım. Burada ki fonksiyonları çalıştırduğumızda bu kanallardan gelen kesmeler dikkate alınacak ve CPU tarafından işlenecek demek.

KAYNAKÇA

- <https://mjwhite8119.github.io/Robots/mpu6050>
- <https://ayazfurkan.blogspot.com/2019/04/stm32f4-i2c-mpu6050.html>
- <http://sercanerat.blogspot.com/2015/09/merhaba-arkadaslar-bu-yazda-mpu6050.html>
- <https://github.com/jrowberg/i2cdevlib>
- <https://github.com/MYaqoobEmbedded/STM32-Tutorials/tree/master/Tutorial%2035%20-%20MPU6050%20IMU%20Module>
- https://github.com/Ahmed0Ayman/IMU_STM32_Example/tree/master/C_ore
- <https://github.com/MYaqoobEmbedded/STM32-Tutorials/blob/master/Tutorial%2035%20-%20MPU6050%20IMU%20Module/main.c>
- <https://dronebotworkshop.com/mpu-6050-level/>
- <http://www.geekmomprojects.com/gyroscopes-and-accelerometers-on-a-chip/>
- <https://letanphuc.net/2014/06/stm32-mpu6050-dma-i2c/>
- <https://github.com/metinakkin/STM32F4-KEIL-ARM-9-I2C-Communication-MPU6050>
- https://github.com/srcnert/STM32F4DISCOVERY_MPU6050_DMP
- <https://github.com/metinakkin>
- <https://maviled.wordpress.com/2016/02/18/cubemx-ile-i2c-projesi/>
- https://bestofcpp.com/repo/ibrahimcahit-STM32_MPU6050_KalmanFilter
- https://github.com/ibrahimcahit/STM32_MPU6050_KalmanFilter/blob/main/Core/Src/main.c
- https://github.com/Harinadha/STM32_MPU6050lib
- <https://www.programmersought.com/article/19197777753/>
- <https://github.com/leech001/MPU6050>
- <https://github.com/TKJElectronics/KalmanFilter>
- <https://www.programmerall.com/article/81991134406/>
- <https://www.programmerall.com/article/33681241390/>
- <https://www.programmerall.com/article/90511189443/>
- <https://www.programmerall.com/article/62581905681/>
- https://eggelectricunicycle.bitbucket.io/MicroWorks_30B4_board--Datasheets_30B4--MPU6050--DMA.html
- <https://milisaniye.home.blog/2020/09/11/stm32-mpu6050-sensor-uygulamasi/>
- <https://github.com/EnesTayfun>
- <https://selcukozbayraktar.com/2021/08/25/ivme-ve-gyro-sensorleri-ile-egim-acilarinin-olculmesi/>
- <https://github.com/ibrahimcahit>

- <https://selcukozbayraktar.com/2021/08/31/bir-cubugu-pid-kontrolu-ile-istenen-acida-tutmak-bolum-2-2-motor/>
 - <https://selcukozbayraktar.com/2021/08/08/bir-cubugu-pid-kontrolu-ile-istenen-acida-tutmak/>
 - <https://selcukozbayraktar.com/2020/08/21/stm32-ile-pid-kontrol-altinda-dc-motor-surmek/>
 - <https://elektronikatolyem.home.blog/2019/12/18/stm32f103c8-ile-pid-motor-kontrol-uygulamasi/>
 - <https://electrooobs.io/tutorial/11#>
 - <https://www.instructables.com/Self-Balancing-Robot-Using-PID-Algorithm-STM-MC/>
 - <https://electrosome.com/controlling-motors-mpu-6050/>
 - <https://github.com/avem-labs/Avem>
 - <https://opensourcelibs.com/lib/avem>
 - <http://repository.ust.hk/ir/Record/1783.1-61072>
 - <https://www.oalib.com/articles/5287164#.Yf08BupByUm>
 - <https://blog.katastros.com/a?ID=01600-5b9761d0-ac9b-49c1-bac0-56fe0d63004e>
 - <https://www.programmersought.com/article/78648383738/>
 - <https://blog.katastros.com/a?ID=01600-90a3013c-2025-42d5-958e-e96bbe26838c>
 - <https://elektronikatolyem.com/2019/12/28/stm32-pid-motor-kontrol-uygulamasi/>
 - <https://thecodeprogram.com/position-control-of-dc-motor-with-stm32f4-pwm-and-pid>
 - <https://www.instructables.com/Speed-Control-of-DC-Motor-Using-PID-Algorithm-STM3/>
 - <https://tr.oxfordcityvisitorsguide.com/8604132-speed-control-of-dc-motor-using-pid-algorithm-stm32f4>
 - <https://programmersought.com/article/79948234616/>
 - https://programmersought.com/article/79948234616/#2_NRF24L01_82
 - <https://usermanual.wiki/KDS-Model-Technologies/KDS7XII/html>
 - <https://www.manualslib.com/manual/862008/Kds-Kds-7xii.html?page=24#manual>
 - <https://docs.arduino.cc/tutorials/generic/secrets-of-arduino-pwm>
 - <https://stackoverflow.com/questions/43914651/reading-pwm-signals-in-stm32f407>
 - <https://controllerstech.com/input-capture-in-stm32/>
 - <http://elohak.blogspot.com/2016/09/5-stm32f4-pwm-kullanm.html>
 - <https://medium.com/@mehmetgkk/stm32-ile-i%C4%91CC%87%C4%95Flem-s%C4%93BCresi-%C4%93%C4%93B61%C4%93A7%C4%93BCm%C4%93BC-d394df844049>
 - <https://www.mcu-turkey.com/stm32f3-pwm-kullanimi/>
 - https://topic.alibabacloud.com/a/stm32-timer-pwm-mode-input-capture_8_8_10261357.html

- <https://stackoverflow.com/questions/66042701/stm32-timer-cant-capture-pwm-input>
- <https://www.adoclib.com/blog/stm32f4-pwm-input-mode-with-dma-cannot-measure-duty-cycle.html>
- <https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=ysahn2k&logNo=221260639215>
- <https://blog.naver.com/wararat/221082394588>
- <https://elektrokod.wordpress.com/2020/11/01/stm32f103c8-ile-periyot-ve-pwm-duty-cycle-olcumu-pwm-input-mode/>
- <https://medium.com/vicara-hardware-university/stm32-guide-pwm-7763516e62f2>
- <https://selcukozbayraktar.com/2019/08/20/stm32-timer-ile-darbe-dizisi-uretimi-hal-kutuphaneleri-interrupt-ve-cube-mx-kullanarak/>
- <https://medium.com/@mehmetgkk/stm32-ve-pwm-kullan%C4%B1m%C4%B1-af9bbaad7002>
- <https://320volt.com/pwm-period-pwm-duty-cycle-hesaplama/>
- <https://muhittinkaplan.wordpress.com/2017/02/13/stm32f4-cubemx-capture/>
- <http://aybakana.blogspot.com/2016/12/stm32-pwm-uretimi-ve-duty-cycle-hesab.html>
- http://www.brokking.net/Video_83.html
- <https://www.instructables.com/Falling-in-Stm32-Remote-Control-for-Home-Media-Cen/>
- <https://www.programmerall.com/article/43731872594/>
- https://www.reddit.com/r/microcontrollers/comments/bdits2/help_with_pwminput_capture_on_stm32_nucleo/
- <https://www.codetd.com/en/article/12842375>
- https://github.com/EnesTayfun/signal_capture_stm32_nucleol476rg/blob/main/signal_capture_multi_channel
- <http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/>
- <http://www.lojikprob.com/embedded/stm32/stm32-hal-kutuphanesi-ve-cubemx-ile-kesme-kullanimi/>